UNIVERSITY OF CALIFORNIA, SAN DIEGO

Unsampled Digital Synthesis and the Context of Timelab

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Music

by

David Medine

Committee in charge:

Professor Miller Puckette, Chair
Professor Anthony Burr
Professor David Kirsh
Professor Scott Makeig
Professor Tamara Smyth
Professor Rand Steiger

2016

The dissertation of David Medine is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2016

DEDICATION

To my mother and father.

EPIGRAPH

For those that seek the knowledge, it's there. But you have to seek it out, do the knowledge, and understand it on your own.

—The RZA

Always make the audience suffer as much as possible.

—Aflred Hitchcock

TABLE OF CONTENTS

List of Supplemental Files

Sound Recording 1: medine_FM_a.wav

Sound Recording 2: medine_FM_b.wav

Sound Recording 3: medine_FM_c.wav

Sound Recording 4: medine_FM_d.wav

Sound Recording 5: medine_harmonic_FBFM.wav

Sound Recording 6: medine_inharmonic_FBFM.wav

Sound Recording 7: medine_sync_FM.wav

Sound Recording 8: medine_sync_FM_sketch_a.wav

Sound Recording 9: medine_sync_FM_sketch_b.wav

Sound Recording 10: medine_sync_FM_sketch_c.wav

Sound Recording 11: medine_moog.wav

Sound Recording 12: medine_moog_timevarying.wav

Sound Recording 13: medine_moog_extensions_a.wav

Sound Recording 14: medine_moog_extensions_b.wav

Sound Recording 15: medine_noisebox.wav

Sound Recording 16: medine_chua.wav

Sound Recording 17: medine_lorenz.wav

LIST OF FIGURES

ACKNOWLEDGEMENTS

First and foremost, I acknowledge the tremendous faith, support, and help that my adviser Miller Puckette has shown me throughout this process. He took a chance on me when he didn't have to and it changed my life for the better. For this I owe him more than words can say. Thank you, Miller!

I would like to extend my personal thanks to *all* my fellow graduate students, present and past, at the UCSD department of music. They comprise an amazing collection of talent and I can't stress enough how fortunate I have been to be in contact with this remarkable set of individuals. In particular I would like to acknowledge a handful of my colleagues. Thanks to William Brent for helping me learn C, Kevin Larke for everything he has ever said to me, to Joachim Gossman for helping me with that performance of *Kontakte* (and showing me a glimpse of true *Tonmeister* at work), to Rick Snow for being the straw that stirred the cup, and to Cooper Baker for putting on all those computer music concerts. I would also like to thank Drew Allen, Joe Mariglio and Brendan Gaffney for being my friends and for all the knowledge that I've gotten from them a result. I also want to extend my thanks to Steve Solook, Scott Worthington, Kurt Miller, Michael Matsuno, Jon Hepfer, Paul Hembree, Rachel Beetz, Judith Hamman, and Matthew Kline for helping me out with the UCSD Chamber Orchestra over the years. Also, thanks to Batya, Steve (again) and Will for playing solos with us.

Thanks to my committee for taking my work seriously and given me extremely valuable feedback at my defense. Although he had to bow out of the committee in the end, I would like to acknowledge Shlomo Dubnov with whom I have had the good fortune to work with (albeit on a project doomed by university politics). I'd also like to thank Rand Steiger for stepping in at the last minute and saving the day.

My gratitude is extended to Scott Makeig and all my colleagues at the Swartz Center for Computational Neuroscience. This has been my home at the

university since reaching my term limit and it has been a joy. A special shoutout goes to my office mate and *compañero* Joaquin Rapela whose lively discussions I look forward to on my way to work in the morning.

Thank you to Batya for putting up with me these 9 years. I certainly could not have done this without you.

Finally, propriety obliges me to point out that there are several figures that appear in previous publications (both authored by myself). Figures 2.6-2.8 are in 'David Medine. Timelab: Yet, yet another audio programming environment. *In Proceedings of the International Computer Music Conference*, Perth, 2013'. Figures 5.6-5.7 originally appear in 'David Medine. Dynamical systems for audio synthesis: Embracing delay free loops. *Applied Sciences*, 2016'.

VITA

2005	B. A. in Music, Manhattan School of Music
2008-2009	Graduate Teaching Assistant, University of California, San Diego
2009	M. A. in Music, University of California, San Diego
2009-2014	Associate in Department, University of California, San Diego
2014-present	Multi-Modal EEG Programmer and Analyst, Swartz Center for Computational Neuroscience, University of California, San Diego.
2016	Ph. D. in Music, University of California, San Diego.

PUBLICATIONS

Medine, David. 'Timelab: Yet, Yet Another Real-Time Audio Programming System', *Proceedings of the International Computer Music Conference*, 441, 2013.

Medine, David. 'Unsampled Digital Synthesis: Computing the Output of Implicit and Nonlinear Systems', *Proceedings of the International Computer Music Conference*, 2015.

Medine, David. 'Dynamical Systems for Audio Synthesis: Embracing Delay Free Loops', *Applied Sciences*, 2016.

ABSTRACT OF THE DISSERTATION

Unsampled Digital Synthesis and the Context of Timelab

by

David Medine

Doctor of Philosophy in Music

University of California, San Diego, 2016

Professor Miller Puckette, Chair

A thesis about how to construct digital audio synthesis unit generators using dynamical models and what they sound like.

Chapter 1

Introduction

1.1 Preamble

This thesis is about two things that are related. The first is an approach to digital sound synthesis that is called 'unsampled digital synthesis' (UDS). The other thing is software called 'timelab' that demonstrates this approach. The claims being made are the following. UDS is a method for creating digital synthesis routines wherein the temporal width of one audio sample is not the basic unit of time. This is accomplished through the use of dynamical models and numerical solvers to differential equations. As far as timelab is concerned, the claim is that it is a good idea to have a standardized software package that can ease the development of UDS routines. It is also true that the design of real-time audio synthesis engines is not a well documented subject and the following thesis at least provides some introduction to that world as well. In particular, issues of control timing and scheduling are discussed.

To put it succinctly, UDS is a means of computing the output of (certain) systems that are implicit without resorting to implicit methods. Only a subset of all possible implicit systems is suitable for this kind of implementation. Namely, the system must be able to be designated as a set of inter-related first order ordinary differential equations (ODEs). The equations themselves must be explicit, but this does not mean that the system itself need be an explicit one. For example, consider

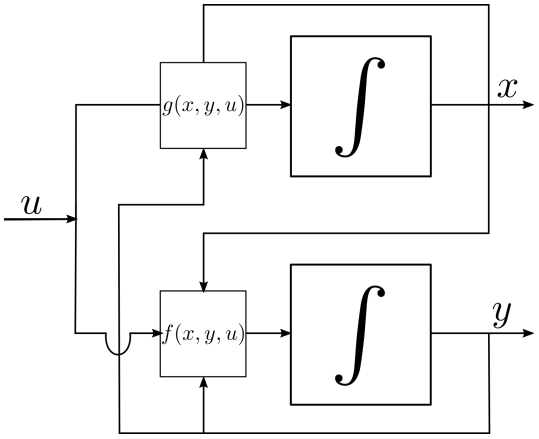


Figure 1.1: Signal flow diagram for the system given in Equation 1.1.

the following system:

$$\begin{aligned} \dot{x} &= g(x, y, u) \\ \dot{y} &= f(x, y, u). \end{aligned} \tag{1.1}$$

Both equations are explicit ODEs, and even if g and f are nonlinear, there are many, many means available for computing the time varying values of x and y (we take u to be some input to the system) even though the system itself is implicit. The system is implicit in the sense that both x and y are time varying functions of each other. That is to say, there is no delay in the feedback loop between the computation of x and y. However, since each node in the network (here expressed as an equation) is explicit, we can apply a numerical solver and it will return a more or less accurate value give the states and parameter values at the time of computing the solution.

Figure 1.1 shows a signal flow diagram representing the system given in Equation 1.1. Linear, time invariant (LTI) digital filters are inadequate for implementing this system as it is both nonlinear and time varying (in the sense that one can think of x and y as parameters governing the behavior of f and g respectively). It is also an implicit network with multiple feedback loops, none of which contain any delay.

To put it another way, UDS is a technique of audio synthesis that poses implicit systems as sets of inter-related explicit ODEs and unleashes explicit numerical methods to compute the output (here the time varying values x and y).

This allows us to experiment quickly and easily with systems that have delay-free loops, and these may be of some interest.

In the game of electrical engineering research in academia, the theory of such a technique is as far as we need to conern ourselves. This thesis, however is a musical one, so we will proceed beyond the theory of dynamical systems and their solutions and proceed towards a musical discussion as well. There is a 'theory' of music, of course, but when it comes to the act of music-making, that theory is far less important than the actual *practice* of music, which is the act of actually making sound. Thus we must present material related to the actual manifestation of the theory of UDS, which is software by which such systems as the one shown in Equation 1.1 can be specified and computed. This software is the library and API known as 'timelab' which this author continues to develop.

Of course it is somewhat problematic to describe continually changing software in an unchanging document, such as a PhD thesis. Obviously, the software will undergo any number of changes (including, perhaps, total death) as time goes on; but this text, once submitted, will not. However, said software description is included here both because it reflects a great deal of work on the part of the author, and without that work the theoretical part of the thesis (UDS itself) could never have been formed by him. Furthermore, music (and this is a computer *music* thesis) is a form of artistic expression and music is, more than perhaps any other artistic medium, concerned with detailed minutiae in the procession of time. Since UDS is concerned with an approach to the treatment of time that is somewhat alternative to 'classical' digital signal processing in computer music, the details of its implementation at both the theoretical *and* practical level are pertinent.

The name 'timelab' was chosen because it reflects the fact that this software is overly concerned with issues of timing; and, because everything needs a name. Timelab is an application programming interface (API) and a kind of plugin host written in C. It takes a compiled C object file (a dynamic library) that has a small number of required and specially named functions for creating, computing, controlling, and destroying a UDS routine. The runtime engine itself has methods for loading the supplied UDS 'library', sending and receiving samples from a host

application, running the library functions that compute the new samples, memory management, and control messaging.

In real-time computer music software, one must not only develop a way to compute audio samples, but also a way to interact with the system. This interaction may occur on two levels. The first is at the parameter level. That is to say, it is desirable that one be able to adjust the parameters of the system while listening to its output in real-time–control messaging. The second level of interaction in computer music systems is at the programmatic level. The developer of a computer music system not only interacts with virtual instruments, but also constructs them. In some cases (such as the one presented here) the developer also develops the tools with which the virtual instruments are constructed. So we pack it all in the following document.

Because timelab is an API, and because it is written in C, without any reliance on non-standard C libraries, it is highly extensible and can be compiled for any operating system or even (with some work) an embedded device. Timelab has been used in the creation of a Pure Data[1] extern as well as a standalone application, using Qt[2] as a graphical user interface (GUI) and PortAudio[3] as an audio interface backend. Though this hasn't yet been attempted, there is no reason to believe that it cannot be extended to interface with other plugin hosts or audio APIs. As mentioned above, it is written in plain C without any non-standard libraries.

It is usually the case that a computer music system consists of a analog to digital converter which takes a number of real world sound signals and renders them into time-quantized streams of audio samples, a digital computer to process those samples, any number of controllers to message the system (these can be hardware or software), and a digital to analog converter to change the newly rendered digital data streams into voltages that will drive loudspeakers. Here (as is the case with most literature about computer music) we are concerned with the middle of this diagram–everything but the analog/digital converters.

[1] http://msp.ucsd.edu/
[2] http://www.qt.io/
[3] http://www.portaudio.com/

Nevertheless, at some point a continuous time signal is converted to a discrete time signal and vice versa. Dealing with discrete rather than continuous time audio streams is advantageous for a number of reasons[4], but it also comes with certain unwanted baggage. On the one hand, digital computers are cheap, ubiquitous, accurate, and programs can be represented by lightweight and transportable media. On the other hand, in order to process a digital signals, digital processing is needed. Quantization and discretization, while a convenient means to a desired end, distorts the signal. Furthermore, chopping up a signal into atomic units of time considerably limits the lower limit with which one can apply feedback. In a digital network we are constantly working with the unit delay as a fundamental building block. In an 'analog' network, feedback is considered to be instantaneous and delay is zero.

It may seem at first blush that this problem is moot. After all, we have the many tools of digital signal processing theory at our disposal, and clearly they work very well (or else we wouldn't have things like communications satellites and high definition television broadcasts). But there are situations for us musicians in which the fact of unit delay is enough distortion to prevent us from achieving the sound or behavior that we are looking for. Thus we have, for example, a field of audio research known as Virtual Analog (VA). There, the name of the game is using digital signal processes to emulate the timbres of analog sound producing hardware.

1.2 'Analog'

It is sometimes bizarre the name that sticks to new technology. The term 'analog' came about because in the early days of transistor based technology, the circuit itself was an analogy for some non-electrical behavior.[5] Since the early days

[4]The most obvious reason is that software can be written to deal with discrete time systems. To deal with signals purely in the continuous time realm requires the development of specialized hardware at every phase. This makes development cycles exceedingly long.

[5]'Digital' computers–programmable devices built around discrete circuit technology–are likewise oddly named. Why a system of discrete transistor switches should be 'of the fingers' remains somewhat obscure.

of transistor based circuit technology, the term 'analog', when used to modify the word 'circuit, has all but completely shed this original (and literal) denotation. Yet the original definition persists. For example, we still sometimes see the circuit-as-analogy in pedagogic literature. A mass-spring systems is explained as a simple circuit wherein an inductor 'is' the mass and a capaticitor 'is' the spring (see for example [Smi10, 5.10]).

In the context of musical devices, the etymology of this term is particularly interesting. Beginning with the Theremin (which produces a nearly sinusoidal waveform) and continuing with the modular sound producing analog devices of Robert Moog and Donald Buchla, analog circuits that make sound (which are now referred to as 'analog synthesizers') introduced tones and timbres that are very difficult if not impossible to create with real world mechanical systems. These, more than most devices comprised of analog circuit technology, are therefor mis-named. The material they produce is not actually an analog to anything, but rather something new in and of itself.

Beyond that, the term 'analog', when applied to the realm of sound, has come to take on a deeper meaning – one that suggests fidelity, authenticity, and (to some extent) purity. Partly, this is exasperated by the rapid proliferation of digital media (sound files), which is most often subjected to data compression. Particularly in the early days of ubiquitous and liquid digital music files (the 'Napster era' of the late 1990s [Ald08]) sonic distortion from data compression was painfully evident. Again we see digitization as a trade off. The process makes the material itself much more transportable, but at a cost of decreased sound quality. Since the time when digital media files first began to spread prolifically on the internet, data compression techniques have become much better (indeed, we now have 'lossless compression'). This theme of producing a rough estimate of the desired result through digitization and then incrementally improving it through increasingly sophisticated processing techniques is one that is seen again and again in the realm of VA.

The association of 'analog' with purity might also be encouraged by the disembodied-ness of the digital audio file (its lack of album art, abnegation of

human contact at a record-store, the fact that digital soundfiles go against the ethos of 'the album', etc.). So the digital (virtual) medium 'distorts' music not only sonically, but culturally as well. The natural consequence of this is to increase the status of analog forms of electronic synthesis and recording to a privileged status in (at least certain sectors of) the popular imagination. In 2014, 14 million vinyl records were sold in the United State, but in the first half of 2015, this number was already at 9 million [Bri15] indicating roughly a %30 increase. We can point to the advent of smart phones and streaming technology as contributing to the diminishing sales of recorded music generally. Meanwhile (at least in part) the increasing fetish for 'analog sound' accounts for vinyl's renaissance.[6]

The Whole Foods grocery store in my neighborhood recently started selling newly pressed vinyl records. Having been born in 1982, I never remember seeing records in grocery stores before this, although they used to be commonly sold there. In *Steal This Book* Abbie Hoffman mentions that records can be stolen by concealing them in frozen pizza boxes [Hof02].

1.3 Digitization

Even though digital signal processing may not provide tools with which to replicate the material tokens that contribute to the entirety of the analog mystique, many musicians strive to capture may wish to reproduce the 'analog sound' using digital computers. One trouble with this endeavor, and the one we focus on in this text, is that the sounds that we wish to emulate are primarily modeled through dynamical systems. Dynamical systems by definition do not contain unit delay. Thus, when we discretize such a model, the model itself is distorted. Because a Turing machine, by definition, performs discrete operations on discrete samples (time samples, in the case of audio), it can only deal with changes over one variable at a time. Dynamical systems such as the harmonic oscillator, $m\ddot{x} = -kx$, have

[6]Tape, which some audiophiles may claim sounds better than vinyl – it is certainly less susceptible to pops and scratches – is inferioir for practical reasons. Cassette tapes and their players are very complicated machines. The rubber bands on the driver reels wear out and are extremely difficult to find replacements for. Both magnetic tape and vinyl degrade in quality on every listen, another feature of their mystique.

plural variables ($\ddot{x}$ and x) and changes over any one will instantaneously affect all the others.

Even in the case of parallel computing, it is impossible to affect plural, mutually dependent variables simultaneously. This is true because in order to compute digital samples in such a scenario, not only are simultaneous processes necessary, but each process must also have access to each others' states at any given time. This is very difficult if not impossible to enact determistically. This is in contrast to the analog 'computer' (circuit, really) that describes a harmonic oscillator. There, the state variables that affect one another, thereby creating the oscillation, and their rate of change can be read off simultaneously at any two or more points within the circuit.

Needless to say, digital computers (which we may simply refer to as 'computers' from here on) can compute a very near approximation of a harmonic oscillator; and, much, much more complicated systems as well. It is of course the theory of digital signal processing with which a computer's power for computing audio is harnessed. Digital computers outstrip their analog counterparts significantly in terms of accuracy, program-ability, repeatability, durability (a computer program never needs to be repaired), weight, cost, etc. Beyond that, digital signal processing theory allows us the ability to create sounds and control methodologies that are unique to the digital realm and extremely impressive in their own right. *Bon chance* to he that wishes to build a 1024 band phase vocoder with op-amps.

1.4 Structure

The following is purely concerned with software to be run on digital computers. It is also about implementing digital signal processing techniques. But, at least we aim for developing tools for using models in such a way that we needn't fight against the constraints that unit delay and linear time invariance place on signal processing theory. We shall develop a number of digital solutions for computing the output of nonlinear and implicit systems.

The thesis is structured in the following way. The Chapter 2 is concerned

with the history of electronic music systems in a way the builds towards the current state of the art in real-time computer music systems. It goes on to discuss the challenges associated with building such a system at the procedural level. Chapter 3 is a description of how timelab works. It is in no way documentation of the API. This is done on purpose since it will undoubtedly change. The fundamental structure (which may also change) is discussed.

Chapter 4 is a summary of notable computer music techniques. All of these feature the audio sample as the basic unit of time and its procession as a fundamental building block. Chapter 5 is about UDS, its theory and applications, which (hopefully) will support the claim that its treatment of time is fundamentally different than what is shown in Chapter 4 and that that this is a good.

Chapter 6 (which is added by request of the committee supervising the creation of this document) contains some thoughts on designing systems that are suitable for UDS and the related subject of what such systems actually sound like and why they sound that way. Richard Boulanger's 1985 thesis about the musical applications of convolving recorded speech with other audio signals [Bou86] is used as kind of model for structuring this discussion.

Chapter 2

A Brief History of Computer Music Techniques

A subject of this dissertation is the audio programming environment called timelab and its usefulness. A question worth asking here is why create a whole computer music system, when a simple plugin or Pd extern would do? One motivation for having a complete system rather than a specialized plugin to some other, pre-existing complete system, is that UDS demands that audio samples be calculated at a higher rate than the sampling rate of the audio driver. This could, however, be dealt with in a plugin in a more or less elegant way. Another motivation is that there are many possible UDS routines of interest, so they must be programmed. Again, we could simply write a Pd extern that takes as input the specification for any such routine and has the machinery to carry it out and interface with input and output from Pd.

A final motivation is simple general-ness. Again, since time is dealt with differently, and since connections between nodes in UDS are different than those in any existing host application, pretty much the whole UDS algorithm is contained within itself. This implies that it has a kind of stand alone nature.

Yet another motivation is that UDS is computationally expensive. This immediately suggests 'bare metal' (embedded) deployment. All of these things considered (and because the author became interested in the process) timelab was written in order to load and solve UDS networks.

In order to provide context for both UDS as a programming paradigm and timelab as a programming environment for that paradigm, we proceed with a brief history of electronic music systems. We will then turn to issues of scheduling and the timing of control.

2.1 The Telharmonium

Also known as the Dynamophone, the Telharmonium was perhaps the first fully electronic instrument.[1] Its inventor, Thadeus Cahill was granted a patent for its design on April 26 of 1897. The instrument was massive and controlled by a keyboard mechanism. When a key was depressed, a current flowed through a circuit that was coupled to a belt driven cylinder (one for each pitch). Each cylinder had a series of rheotomes mounted on it that would periodically come into contact with statically mounted wire brushes. Each rheotome had an integer number of conducting strips that would contact the brushes throughout the rotation, starting with one, then two and so on. Thus each rheotome would produce an overtone of the fundamental frequency of the rotating cylinder. Since these signals operated on a 'make and break' principle, the signal coming off of each rheotome must have resembled a square wave. These were then smoothed into a sinusoidal shape by being fed through a series of LC filters before being fed into loudspeakers that were constructed from telephone receivers.[Wei95]

There are several notable aspects to this design. The first, is that the instrument itself was clearly inspired by the church organ. Instead of bellows, the sound was produced electrically, but the control/synthesis relationship was exactly that of an organ.

Second (and of more importance to the present discussion) is that the signal flow of the system resembles in two ways modern synthesis techniques. The fact that each rheotome produced a spectrally rich signal (a square wave has all the odd

[1] Obviously mechanics have been an integral part of musical instrument design for quite sometime. But, to avoid the risk of getting bogged down in discussions of the meaning of harpsichords and church organs, we limit our discussion to one of purely electronic instruments. For a thorough discussion of mechanical as well as electronic instruments and associated social and music implications, see [LR11]

overtones) which was then filtered resembles very much the so-called 'source filter' model (also known as 'subtractive synthesis') which became an important one in vocal synthesis[AH71] [RS11]. The second foreshadowing feature was that tones were ultimately built up out of harmonically related sinusoids. This approach to synthesis is now known as 'modal' or 'additive' synthesis (see, for example, the foundation laying work of Risset [Ris05] and [RW+99]). These techniques are discussed in a number of important treatments of digital audio techniques as they are both in frequent use. For example [Moo90] [Puc07] and [Roa96] all discuss these techniques.[2]

Indeed these models, source-filter/subtractive and modal/additive synthesis, may be generalized as configurations of oscillators and filters; or, just filters, if we view an oscillator as merely a special kind of filter. This paradigm for audio modeling and synthesis is one that has dominated electronic music since the time of the Telharmonium.

2.2 The Theremin

The Theremin (named after the Anglicized name of its Russian inventor, Lev Segeryivitch Termen) is arguably the most successful electronic musical instrument of all time (the radio and the personal computer – if indeed they may be called 'instruments' – excepted). It works by treating the performer's hand as a grounding plate in an LC circuit. By moving the hand nearer and farther from an antenna, the capacitance of the circuit is varied, thus changing the frequency of a radio frequency (RF) oscillator. Another RF oscillator is held at a fixed frequency and the difference tone between the two is in the audible range.

The Theremin (unlike the Telharmonium which weighed over 200 tons) could, even in the days before silicon transistors, fit inside of a suitcase. It could

[2]One important note is that there is a slight distinction in the physical modeling community between the meaning of modal synthesis and additive synthesis as well as between source-filter model and subtractive synthesis. But, the distinction seems to be largely one of intention rather than technique. For example, if one wishes to synthesize the sound of a trumpet by modeling the buzz as a source and the tube as a filter, one is using a source filter model. But if one simply makes a synthesizer patch that uses a time varying filter to modify a triangle wave, one would be doing subtractive synthesis.

also be cheaply and easily constructed by anyone who possessed any skill at radio construction/repair (a common hobby in pre-war America). It is thus that the Theremin's true power shows itself. As La Rosa argues, the Theremin's most important innovation is its represent-ability as a circuit diagram.[LR12]

2.3 Analog Synths

Donald Buchla and Robert Moog independently and simultaneously brought analog synthesizers into maturity by instituting the idea of Voltage Control (VC). Since synthesizer modules not only output, but also received as input a voltage signal, VC meant that the output of any synthesizer module could be the input to any other. This greatly expanded the oscillator/filter configuration model simply by greatly increasing the configurability.

One important note is that control parameters (such as the frequency of an oscillator) are now assignable to an automatic process rather then human input. The level of complexity that such modularity affords is quite large and, thus, a great number of sounds and approaches to composition become available. Buchla in particular saw this feature as an important innovation, and thus he eschewed Moog's painstaking development of a piano-sized key board that could throw normalized control voltages at his modules:

> 'I saw no reason to borrow from a keyboard, which is a device invented to throw hammers at strings, later on operating switches for electronic organs and so on.'[PTP09]

This implies an attitude towards pitch that is quite a departure from most musical traditions. Throwing away predictable, governable control over frequency displaces pitch from its relatively lofty position over other musical parameters, such as timbre or loudness. Using similar processes to control both the frequency of an oscillator (pitch) and the cutoff frequency of a filter (timbre) democratizes these elements. Donald Buchla wasn't the first musician to have such attitudes, of course (we may go on a lengthy tangent about Arnold Shönberg and the dodechaphonic system here), but the voltage controlled modular synthesizer was perhaps the first

instance of this attitude actually being inherent to a pitched musical instrument itself.

We note that the modularity of analog synthesizers is also very powerful. One can 'plug and play', thereby modifying sounds in real-time, as they are being generated. This means that *playing* the instrument includes active *construction* of the instrument.

Modular synthesizers are still being manufactured and are still prized for their sound and the compositional psychology they afford. However, to a large extent, analog gear has been digitized, that is to say that software that emulates the 'analog sound' often replaces the genuine article for reasons discussed above (convenience, durability, cost, etc.). Nevertheless, these devices survive and are an example of one of very few analog computer systems still in production. This fact is a testament to the power of sound and music.

2.4 Digital Synthesis

Indeed the 'analog sound' is a real thing; and, moreover, the ability to control such sounds with the kind of plug and play operation afforded through patchable modules is a powerful and meaningful paradigm for music making. Part of what is good about that paradigm is that it is easy to do and the results can be both astonishing and entirely unpredictable. By the same token, however, music as an art is one that is often conerned with precision and predictability. Analog computers are imprecise, but digital computers are just the opposite. The exactitude of specification that *software* affords is also a powerful position for a musician to hold, and it is to that position that we now turn.

2.4.1 The Early Days

In 1962, Bell Labs released the album *Music From Mathematics* whose most famous track is undoubtedly the computer rendition of Harry Dacre's 'Daisy Bell (Bicycle Built for Two)'. This song was featured in *2001: A Space Odyssey* when HAL sings this song as Dave is shutting him down towards the end of the picture.

The choice of song was a direct allusion to the Bell Labs recording. Max Mathews programmed the accompaniment and John Kelly and Carol Lochbaum synthesized the 'vocals'.

The release of *Music From Mathematics* was meant in part as a demonstration of Bell Labs' sophistocated vocal synthesis technique which modeled the vocal tract as a series of acoustic tubes of varying sizes [KL62]. This technique was later abandoned by Bell Labs in favor of more efficient speech coding techniques such as the vocoder and the (already mentioned) source-filter model. However, this methodology was later taken up by Cook in his PhD thesis, which improved upon the model by adding a branch for the nasal cavity as well as losses due to radiation from the neck[Coo90]. Furthermore, the use of d'Alembert's solution to the wave equation as a way to calculate the traveling wave through the cross sections of the vocal tract was an early example of what later became known as Digital Waveguides (DWGs), a term coined by Smith[Smi85], and are still the state of the art in one-dimensional physical models of tubes and strings.

2.4.2 Music N and Csound

Out of the early computer music experiments at Bell Labs in the 1950s came a series of iterations of a computer music programming language called Music N. As early as 1963, Mathews wisely foresaw that digital computers could be used as both control devices and sound producers.[Mat63] With this duality in mind, Music N was designed to be both a language for specifying algorithms that produced series of numbers that can represent time-varying wave forms, and for specifying the inputs to the parameters that govern the behavior of said algorithms. The same is true of csound, originally by Barry Vercoe, which grew out of Music N and after more than 40 years, is still under active development (http://www.csounds.com/).

These languages are notable in that they maintain a separate notion of a 'score', and an 'instrument'. This is in stark contrast to the analog synthesizer approach which blurs the lines between sound-making (instrumental) and sound-controlling (compositional) objects.

Also developed by Mathews and others at Bell Labs, the Groove system

[MMR74] was a hybrid analog/digital system in which the computer played the role of piano reel, outputting pre-composed control voltages that controlled an array of analog synthesizer modules. The motivation for this setup was to allow for real-time control over sound synthesis. Computing power in the 1970s was not yet sufficient for robust real-time sonic output. It was sufficient, however for manipulating parameters and processes that controlled parameters.

2.4.3 Patching Languages

We skip the vast and intriguing, but now mostly forgotten, world of programmable commercial digital synthesizers. We proceed directly on to personal computer audio programming systems. For a comprehensive treatment of this subject see [Thé97].

The 1980s saw the initial development of Max. Max and its kin (Max/MSP and Pure Data) are graphical programming languages and to a great extent represent a return to the analog synth philosophy of systems in which anything can be plugged into anything else. In part this came about because the Sogitec X4 computer (for which Max was written) was capable of outputting sound in real-time [Puc96]. Thus, in contrast to the Groove system, there was no need to offload real-time sound generation to analog machines. Since everything could be done in the computer and (within limits) in real-time, this notion of arbitrary interconnectivity was possible to realize.

In the patching paradigm we see the playing out of the ever-so-blurry analog/digital dichotomy. What this means is that although Max, Max/MSP, Pure Data (Pd), and the text-based patching languages SuperCollider, ChucK, etc. are all completely 'in the computer', they represent an approach to programming that is inherent to analog synthesizers. That is to say, patching languages are comprised of atomic elements (which are more or less 'objects' in the computer science sense of the word) that have inputs and outputs and can be connected to one another (see Figure 2.1). This is exactly how systems of analog circuits are designed [HHH89, 65]. But, this is not in keeping with the canonical Von Nuemann (Fig-

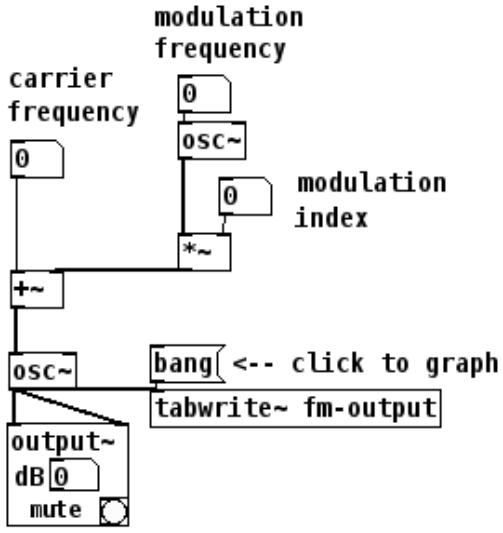
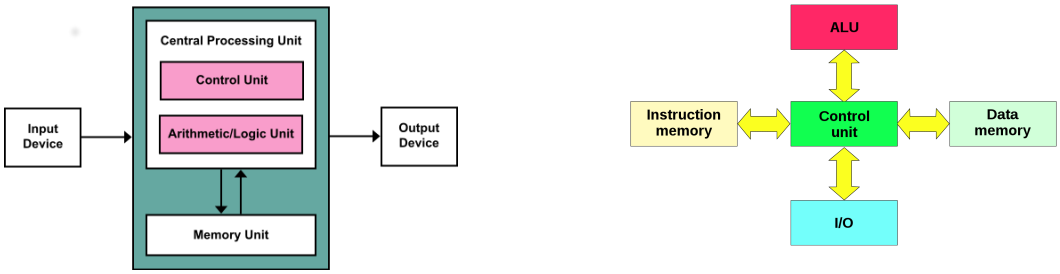


Figure 2.1: A Pd patch for FM synthesis, from Pd's help menu.



(a) The Von Neumann architecture.

(b) The Harvard architecture.

Figure 2.2: The CPU-centric digital computer architectures inside all of our desktops, laptops and mobile devices.

ure 2.2a[3]) or Harvard digital computer architectures (Figure 2.2b[4]) in which the central processing unit plays such a monolithic and critical role.

So the patching paradigm, in its way, encapsulates an 'analog' notion of how to organize things, but (of course) performs that organizational scheme according to the (obviously very useful and flexible) digital computer schematic. It is a digital analogy for an analog system.

[3]This image is available through wikicommons: by Kapooht - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=25789639.

[4]Also from wikicommons: by Nessa los - Own work, CC BY-SA 3.0, .https://commons.wikimedia.org/w/index.php?curid=10303637.

However, and this is important to note as a practical issue and will be discussed in greater detail later, the patching languages listed above all (with the exception of ChucK [WC+03]) maintain a separate sample rate for control messaging and audio processing. This is to save time. Digital audio, due to the Nyquist theorem, can only represent frequencies up to one half of the sampling rate. Since humans can hear up to about 20 kHz, the sampling rate should be at least twice that in order to be musically meaningful.[5] However, control input need not be faster than either a person can move, or as fast as the just-noticeable-difference (JND) between two sound events. Both of these periods are much less than 1/40000 seconds.

A final remark about patching languages is that they come in the visual (Max/MSP and Pd) and textual (SuperCollider and ChucK) variety. Some things are easier to represent visually, others textually. For example, the connections between objects in a Pd patch is easier to grasp by simply examining the lines that connect them than are connections between objects in various lines of code. On the other hand, writing a simple `for` loop in Pd in a flow diagram is rather inelegant.

2.4.4 Plugins

Plugins are small computer music programs written in adherence to some software development kit (SDK) so that they may be suitably loaded and understood by some other pre-existing software. Csound, and all the patching languages cited above are extensible through developing new audio and control objects (also known as Unit Generators or UGens) that can be loaded by their native engine. Commercial systems such as Live, Pro Tools, etc. also benefit from the plugin paradigm. Open or semi-open source SDKs for composing plugins allow commercial (very much closed source) digital audio workstations (DAWs) to benefit from third-party development without having to open their own source code to the public.

The plugin is a handy tool for the developer as well. It allows him to

[5] Curiously enough, speech can be clearly understood at much, much lower sampling rates.

lean on the already completed and presumably very well made established audio systems. He need not write low level code to interface with audio drivers and human computer interface (HCI) control streams for every operating system on the market every time he gets an idea about some cool spectral filtering effect.

2.4.5 Moving Forward

Despite a crowded marketplace (dominated by the several examples listed in the previous section), people keep inventing new computer music programming environments. To take a notable example, Vesa Norilo continues to develop Kronos [Nor16]. That language is very interesting since patches can be specified by both a visual environment, and a text-based language interpreter. The software will seamlessly translate between those representations on the fly.

This is a very good idea. But, it remains unclear as to what is to be done next. There is always a trade off between low-level control (what Joel Chadebe would call 'control') and high-level effectiveness (what he would call 'power'). This is reflected in the trade-off between ease of use (moving patch chords around) and specificity (precisely specified software). Patching languages are a wonderful compromise between the extremes. They offer a high level abstraction over low level processes all the while providing powerful access to those processes.

The art of designing a computer music programming environment lies in finding a sweet spot wherein users are afforded the ability to have enough low-level control over the details they need whilst simultaneously providing enough automated infrastructure so that they need not have to deal with things like compiling, interfacing with computer audio devices, garbage collection, etc.

2.4.6 Conclusions

One final thing to note about digital synthesizers, before turning to a more technical discussion of timing in software systems, is that just as there is an 'analog sound' there is also a 'digital sound'. In part, this author would argue, this has to

do with the precision of digital hardware.[6] It is also due to the fact that sampled audio streams make it impossible to represent certain waveforms and filters. It is impossible to make a good triangle wave with a computer.

2.5 Latency, Priority Scheduling, and Control

The following discussion is, to a large degree, a summary of lectures given at IRCAM by Miller Puckette [Puc12], James McCartney [McC12], and Roger Dannenberg [Dan13]. The subject of that lecture series (the MuTant Multimedia Seminars in Real-Time Computing) is timing in real-time audio systems.

One of the challenges in creating a real-time music programming environment is scheduling. For obvious reasons, in order for audio to be coherently output, buffers full of digital samples need to be delivered to the digital to analog converter in regular and precisely timed intervals. This is tricky business, but one that operating systems' audio drivers may handle for us.

In general, the real-time scheduling game is played like this. Step 1: wait for a clocked[7] timing mechanism to indicate that it is time to shove new samples into the DAC. This can be done either through a callback routine (a function that calls the clocking mechanism and asks to be 'called back' later when it is time to execute), or through blocking. The idea behind blocking is that while the DAC is busy (i.e. it is not ready to receive new audio samples to convert to analog voltages), the functions associated with the processing of control input and audio samples goes to sleep (they are blocked). Step 2: query changes to control inputs and perform necessary processing associated with that. Step 3: and query changes to the synthesis program itself and compute the necessary number of samples of output that the DAC requires. If there is also input to the system, this would be gathered during step 3. These steps can be done in any order.

[6] Anecdottally, the author may support this assertion with a short story about the first time he and his family, he was a young boy at the time, heard a compact disc being played. A neighbor had bought a CD player and the families gathered to listen. Everyone kept thinking that they needed to turn the volume up. The absence of surface noise actually made the music less audible at first blush.

[7] The most reasonable choice for a clock to refer to is the actual DAC hardware clock. CPU clocks can also be used, but are prone to drift and other inconsistencies.

However, there are further complications. Obviously, the computer can not know in the future how parameters to an algorithm (real-time control input) or the algorithm itself (if the system is dynamically patchable) may change. By the same token, it is not clear how much time these changes will need to execute (actually get processed by the computer) or how long the act of processing audio samples will take.

Finally, as CPU processing speeds are reaching their maximum, throughput is increased not by making faster CPUs, but rather, making multi-core CPUs that break up processing tasks into smaller bits that can be handled by multiple cores in parallel. This paradigm creates a whole other layer of scheduling problems. Since real-time audio synthesis systems are highly modular and are comprised of many atomic objects, it is not clear how to appropriately delegate the execution of these various and critically timed processes in a coherent parallel structure. So, before turning to a description of timelab and how it works, we shall present a brief discussion of scheduling problems and some modern solutions.

2.5.1 Logical and Real Time

In general, schedulers have to deal with dilemmas associated with the incongruity between 'logical time' and 'real time'[8]. Here we take logical time to mean the times that things are finally executed by the computer. Real time means time in the physical universe (or at least as humans are able to perceived its passage on planet Earth).

Computers and their processors, as we all know, are not infinitely fast. There are delays between when instructions are received, when processes are carried out, and when processes return. The first delay we may consider is the time it takes to actually find a process and cue it for execution. We shall call this 'scheduling latency'. The next is the amount of time that process actually takes to compute – 'execution latency'. There is a third kind of time delay called quantization error. This is a very small amount of error that results from the fact that we

[8]Here, 'real time', which is a type of time, should not be confused with 'real-time' which is a kind of computer music program that can be dynamically altered while producing audible output.

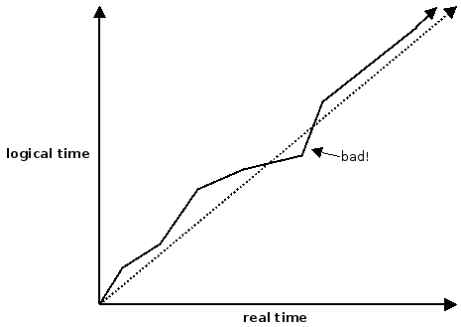


Figure 2.3: Computing audio in terms of logical and real time.

are limited by numerical precision as to how accurately a scheduled process may actually by instructed to begin. On modern computers, this error is very, very small, but if not checked, it can accumulate into noticeable distortion.

2.5.2 Time Overhead

Again, as we all know, computers need a finite amount of time in order to compute things. Logically then it is clear that in order to hear the effect of some execution of audio or music, we must wait.[9] In general, though, waiting is better than not being able to spit out the right samples in time, which would cause audible clicks and distortion in the output of the computer's music. Figure 2.3 illustrates the situation. The dashed line shows an ideal (impossible) situation and the jagged slope shows some probable actuality.

In an ideal world, the output of the computer would coincide exactly with the passage of time and audio would march up the 45 degree line; e.g. logical time would be the same as real time. But this is impossible due to execution and scheduling latencies. It is also the case that in a computer with an operating system, the CPU may intermittently be given other tasks to compute in concurrence with an audio processing algorithm. Thus, we face not only the fact of latency, but also latency of random duration. So, in reality, the best we can do is keep the actual output of samples (meaning their output from the CPU, not from the

[9]Of course, this is true for acoustic instruments as well. The brass section of an orchestra is accustomed to playing ahead of the beat in order to compensate for the speed of sound.

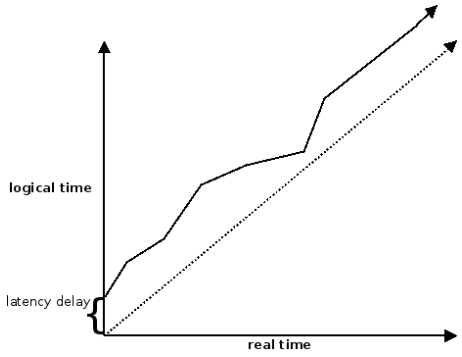


Figure 2.4: The effect of introducing delay on the real time axis.

DAC) above the dashed line. This would ensure that when the DAC is ready for the next sample or group of samples, they will have been computed and are sitting in a buffer somewhere, ready to go.

If, however, the output goes below the dashed line then we are in trouble because the DAC will be calling for output that does not yet exist. The possibility for this occurring can be reduced by artificially advancing logical time; or, to put it another way, delaying real time in the eyes of the computer program. This has the effect of providing time overhead so that the CPU has more time to compute the audio samples. This adjustment can be seen in Figure 2.4.

This gives us a very good way to make a rule for scheduling computational tasks. In a complete music system, there are a number of things that need to get done apart from preparing samples and sending them to the DAC. These include processing changes to the GUI, allocating/freeing memory, polling other processes, etc. Some of these may be more or less necessary than getting the samples out. For example, making changes to the GUI may be less important than polling the control layer to see if parameters in the algorithm need to be affected. Thus it behooves us to set a limit above which we may execute tasks in the scheduler and below which we need to catch up. This is illustrated in Figure 2.5. Here, we have two diagonal lines, one being the coincidence of real and logical time, the other, some slightly delayed copy of this line which determines whether or not processes

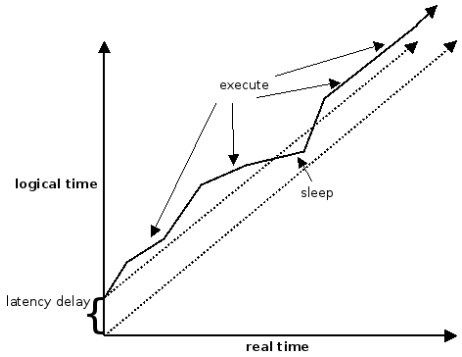


Figure 2.5: A rule to justify real and logical time.

can execute or sleep.

One important note is that if we want our scheduler to be interactive, we cannot set the latency delay too high. If the latency delay (which we want to be higher than the worst case total latency that results from execution and scheduling) is on the order of minutes, then obviously we can not interact with our system in a meaningful way. In general a 5-10ms latency is considered quite good in a real-time music system.

2.5.3 Priority Scheduling

We presume that our music scheduler needs to accomplish a number of tasks before samples are due to be delivered at the DAC with appropriate timing. This may include polling control input, drawing a GUI, accessing memory, reading from a disk, and (of course) computing audio.

Dannenberg points out that the minimum total latency delay, α ought to be greater than the worse case total sum of latencies associated with each task, $\bar{\lambda_i}$ [Dan13]. We may write this down as:

$$\alpha > \Sigma_i \bar{\lambda_i}$$

We may, perhaps, gain significant speedup if we prioritize tasks that are more critical. The idea is that things that are super essential (like computing audio) get priority control of the processor above other tasks – even if they have a higher

maximum latency ($\bar{\lambda}$). Then, we have a minimum value for α that is no longer the sum of all the worst case latencies, but rather a sum of some subset of all the tasks. Lower priority tasks must wait until the higher priority tasks complete before getting time to execute.

SuperCollider does this in a seemingly straight forward way: by maintaining two threads, a real-time and 'non-real-time' (meaning audio processing and all other processing) thread. Unfortunately, this arrangement necessitates a data transfer routine to communicate between real-time and non-real-time threads. Pd handles the problem by not having multiple threads at all – it simply contents itself with having hard edges for the user to be constrained by. Don't write a Pd patch that allocates memory while the DSP engine is on, it will probably result in dropped samples.

2.5.4 Blocking

Unfortunately, this term has two different and unrelated meanings. Above we used 'blocking' to mean when one process prevents another one from executing (as in 'block' a shot in a basketball game). Here, and throughout the rest of this work, the word 'blocking' will refer to the practice of computing audio samples in groups (as in a 'block' of houses on a street), rather than one at a time.

A common technique used to cut down on scheduling latency is to compute audio samples in blocks. Music processing algorithms are usually created by sending signals in and out of a series of processes that are networked in some kind of order to achieve a desired end result. By definition this implies that a potentially large number of function calls need to be cued onto the processor's execution stack. It takes time for the computer's operating system to track down the functions and put them on the correct place on the stack—even if the order is not changing.

One solution for dealing with this is to render the entire DSP chain down into a single function. This can be a useful solution when rendering a DSP chain down to some static, unchanging algorithm.[10] However, the instruction set for this

[10]This could be very useful, for example, in an embedded programming paradigm. A higher level interpreter can specify a synthesis routine and this routine may be adjustable in real-time on a computer. Then, once the desired results are achieved, the conglomeration of modular

single (possibly huge) function may exceed the cache space on the processor, and we are back to the execution bottleneck problem. The other way to deal with this is to specify that each stage of the DSP chain compute not one but many audio samples per function call. This way, if each function in the DSP chain computes, for example, 64 samples per call, rather than 1, the scheduling latency will be significantly reduced. This technique is often referred to as 'blocking'.

The only ways to cut down on execution latency are either to optimize or get a faster CPU.

2.5.5 Blocking and Control

It is usual in computer music software to speak of control rate. This is the frequency with which control input is read and is often an order of magnitude slower than the sampling rate. The reason for this is both practically and perceptually motivated. Practically, it is convenient to clock control at the block level and perceptually this is justified by the fact that (given that the blocks are 64 samples wide and that the audio rate is 44.1kHz) this period (about 1.5ms) will be shorter than a mere mortal can move. There is a general problem that arises in a real-time music processing system when the blocking strategy is employed, and this has to do with the application of control input to the algorithm. Figure 2.6 illustrates this problem. If control arrives at points a, b, and c in real time, then the earliest we can apply these control changes is in the next block of samples. We may then apply changes at the beginning of the next block, or attempt to apply them to the sample relative to the point within the next block that corresponds to their arrival in the first.

This second attitude is more complicated than the first. Applying control with sample accuracy implies altering the control rate of the system for special cases. This is indeed what Pd does with the [vline~]. This object maintains its own sense of time and can be instructed to hit target values with sub-sample accuracy. It is used by feeding it a new control value and a time (usually very small) at which to linearly interpolate to that value. Presuming that the time given is

functions can be boiled down into a single, larger function.

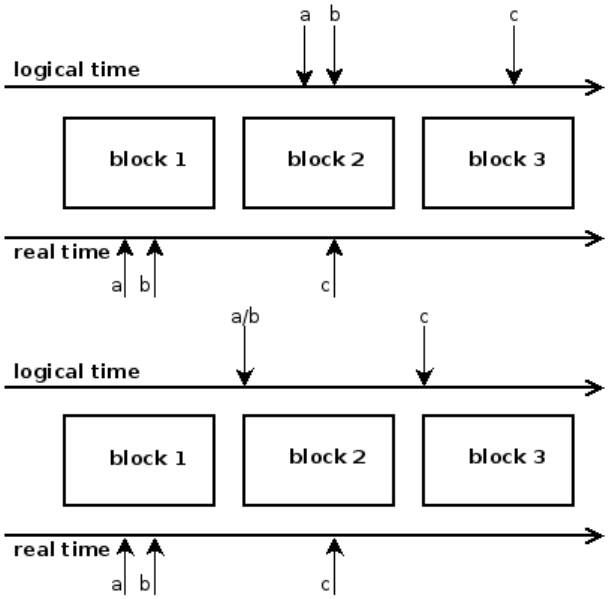


Figure 2.6: Control input to an audio engine that utilizes blocking.

equivalent to the time of one audio block, the action of [vline~] would look like the bottom figure of Figure 2.7. The top of Figure 2.7 shows what would happen if the same arguments were given to the non-sample accurate linear control interpolation [line~]. This illustrates the difference between [line~] and [vline~] objects in Pd. In this example the time to interpolate across is given as 5/3s of one block. Notice how [line~] distorts this period to quantizing not only its starting point, but also its endpoint to block boundaries.

SuperCollider has an alternative solution for enacting sample accurate control. It has a unit generator called OffsetOut that can shift a block to the right by some partial block delay. However, as McCartney notes, this could result in the possibility of two synths reading the same control envelope with different partial block delays. This situation, which would result in an audible flam, is depicted in Figure 2.8. Here is depicted two SuperCollider synths A and B that are delayed to the right in time by the UGen OffsetOut. They are reading from the same global signal Z (perhaps an attack envelope) but since they have different partial block delays, Z is applied at staggered points in time – creating a flam. Z is aligned with A and B, but not with itself. This figure is drawn by the author, but copied from McCartney's slide from his 2012 IRCAM talk.[McC12]

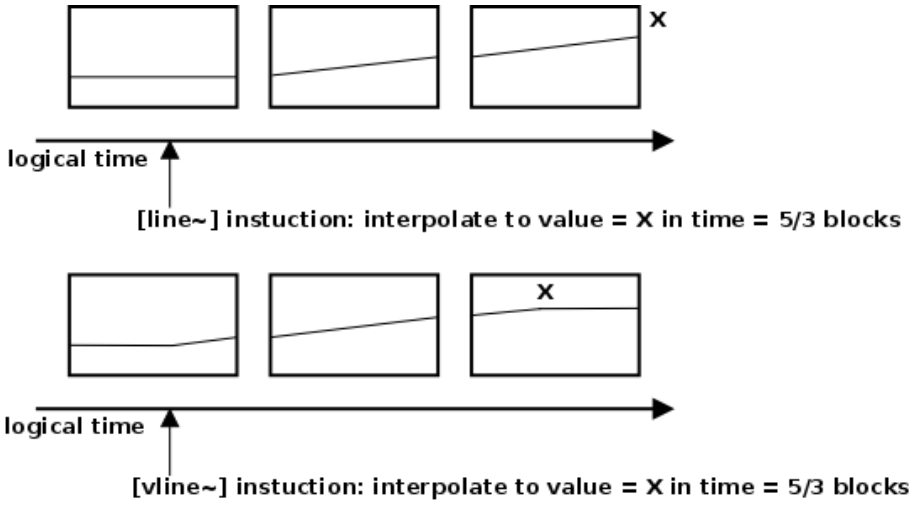


Figure 2.7: The scheme of the [vline˜] object.

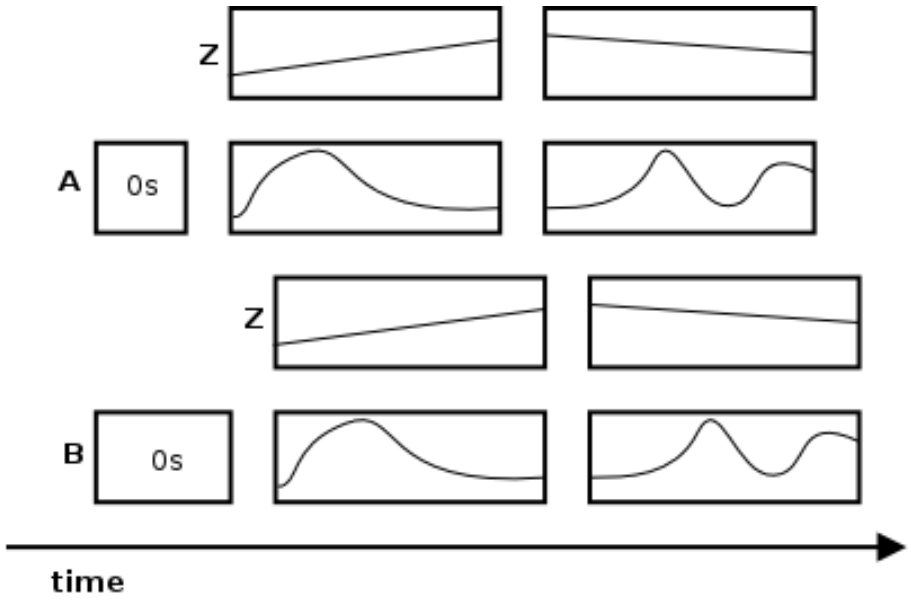


Figure 2.8: Illustration of SupperCollider's OffsetOut.

A fuller description of timelab follows in the next chapter. However, as a sneak preview, timelab may compute audio in blocks of any length (depending on how it is implemented). When it does this, control input is linearly interpolated between block boundaries. So, control values arrive once per block, but at each sample within the block, the corresponding interpolated values are given as parametric input. This control scheme is not as nuanced as the solutions providing sample accurate control timing in Pd and SC, but it is simple. This is a good choice given the fact that timelab is an API that aims at being plugin-able to other host systems.

This chapter contains material that appears in 'David Medine. Timelab: Yet, yet another audio programming environment. *In Proceedings of the International Computer Music Conference*, Perth, 2013'.

Chapter 3

Timelab Specifications

In the previous chapter, we saw a brief overview of the development of computer music technology in terms of programming paradigms. In the present chapter, we present timelab, which was written by the author in order to realize the unsampled digital approach to audio synthesis. The following is a description of timelab and how it works.

Timelab is a C language API and and audio plugin host for programs of a very special kind. It provides methods for loading plugins (which we may call 'modules'), methods for giving/taking audio samples to/from a higher level host application, as well as methods for gathering incoming control data and using these to parameterize modules. There are audio and control processing routines that handle the scheduling and allocation/freeing of memory for these objects. All of the above methods operate behind the scenes. What one wants to do with timelab, is write modules that represent a UDS routine (a dynamical system). There are also methods for computing the output of dynamical systems specified by modules themselves.

3.1 Structure

The core of timelab is very simple. It consists of a loader function, a DSP loop, and cleanup routines. There are a number of ancillary helper functions to

glue everything together and a small list of DSP 'objects'[1] (data structures and associated functions) that grind out audio samples. The most important of these are an 'ADC' (which doesn't actually do analog to digital conversion, but simply grabs samples passed in from a host application), a 'DAC' (same comment, but in reverse) and a solver (for dealing with UDS routines).

Timelab runs C files that have been compiled as shared objects.[2] The compilation uses a standard makefile evoking gcc and are suffixed '`.tl`' to distinguish them. These binaries are referred to as 'modules' and are passed to timelab's loader routine.

3.2 Tl Modules

A proper `.tl` file must contain four special functions in order for the timelab loader and audio engine to be able to deal with it. At load time, the timelab loader receives the full path to the tl file (along with optional arguments), and excerpts the name of the object (sans the .tl suffix) itself. This name (expressed as a C string) is then appended to the following four strings:

- `tl_init_`
- `tl_kill_`
- `tl_dsp_`
- `tl_reveal_ctls_`.

The `.tl` file itself must contain functions with these names suffixed by the name of the module itself or else the loader will abort. For example, suppose there is a `.tl` file called 'filter.tl'. When its path is handed to the loader, the loader will look for functions in the object called `tl_init_filter`, etc. If the loader cannot find the appropriate functions, it will return without having loaded the module.

[1] Timelab is written in an object-oriented style in only the loosest sense. It does associate data structures with certain functions, however it does this somewhat informally. There is no notion of inheritance or polymorphism.

[2] This is Linux parlance. 'Dynamically loaded library' is synonymous with 'shared object'.

The `tl_init_`, `tl_kill_`, and `tl_dsp_` functions are stored in singly linked lists (there is infrastructure for expressing multiple modules) and are used by the timelab engine to initialize, destroy, and run the dsp routine of the module. The call to `tl_reveal_ctls` simply returns a pointer to the stack of timelab control objects that are associated with the module so that external control can be passed as parameters in real-time during run time.

In order for all of this to work, the loader also needs to have an instance of the timelab engine itself. This is a special data structure and associated functions known as a `tl_procession`. A variable of type `tl_procession` contains pointers to the control, DSP, initialization and kill stacks. It is the host's way of interfacing with the timelab audio engine, control processing engine, and cleanup routines. The creation and destruction of a `tl_procession` variable must be handled by the timelab host.

This is a fairly flexible way of doing things. For example, timelab as a Pd extern creates a `procession` and DSP processing loop and control processing loop (which are associated with the `procession` are called during the DSP loop in the Pd extern.

If, on the other hand, timelab is instantiated as a stand alone program, a `procession` object can be setup to maintain two threads, one for the DSP and one for control, both of which sleep until the operating system requests a new set of audio samples. The way to do this depends on what audio driver the operating system is using.

3.3 Control Messaging

The constraints on the interface are that blocks of audio signals be handed in a buffer of pre-determined and unchanging size and that control messages be of the form of a duple containing a reference number (an integer indicating which control parameter is being addressed) and a control value (a floating point value the parameter should ramp to). So-called 'bang' messages – ones that cause some user defined function to be called – are also admissible.

Note that the input to a control object is a single value. The output, however, is a vector of values, the length of an audio block. Every time a value changes, a line must be drawn between the old value and the new one. For each time point in the computation of a block of samples, the corresponding value from the applied control vector is applied (in whatever manner the `.tl` file calls for.

At each tick of the DSP loop, the list of control objects is inspected and if a value has changed, it is pushed onto a special control stack which will draw a line between the old value and the new value. Once the value is not changing anymore, the control output vector needs to be 'leveled off' (write the steady control value to every time point in the output vector) so that it stays at the last requested value (otherwise it will effectively repeat the last control value ramp).

As mentioned above, timelab maintains a singly-linked list of control objects. They are referenced simply by an integer number which corresponds to each control object's place on the control stack. It is wise, but not required, to provide each control object with a unique name. There are functions for printing out the name and number of all the objects within the control stack. The control stack grows in size dynamically at load time and each object is automatically destroyed.

3.4 Audio Signals

Like any audio programming environment and scheduler, timelab passes vectors of audio signals between DSP routines. In timelab, DSP objects have inlets and outlets. Each inlet and outlet has an associated 'signal' class associated with it. In the case of outlets, memory is allocated behind the signal pointers when they are initialized and this memory is free when the module is exiting.

Inlets, on the other hand, are simply pointers to signal classes. Since they either point to an outlet or nothing at all, there is no need to allocate memory to inlets. Connection between an inlet and an outlet is made by simply equating an inlet to the desired outlet. This allows for the possibility of dynamic patching because the equating of these pointers can be done at any time. Control objects could also be instantiated to connect or disconnect inlets and outlets. Since no

memory needs to be allocated or freed to make a connection change, this can be done more or less immediately and cheaply (i.e. without causing hiccups in the output of audio samples).

3.5 DSP Routines – ADC and DAC

As noted above, the only DSP routines in timelab that are germane to the present study are the sample grabber ('ADC'), sample sender ('DAC'), and the UDS solver itself. The classes `tl_adc` and `tl_dac` need to know about the host application's block size. Given this, they maintain a pre-allocated buffer of audio samples which are written to and read from in the course of the host application's DSP tick. These buffers serve as interfaces to timelab's own notion of samples, blocks, signals, etc. Helper functions exist to grow or shrink buffer sizes after they've been initialized.

3.6 DSP Routines – `tl_UDS_solver`

The class `tl_UDS_solver` is where the magic happens, so to speak. It is a scheduler within a scheduler. It consists of a constructor, a destructor, a sample computing routine, and any number of `tl_UDS_node` objects. These objects are instances of classes that have a function (which the solver literally solves), any number of pointers to `data_in` pointers of type 'audio sample' [3] and a single pointer to a `data_out` sample pointer. That is to say, any variable of type `tl_UDS_node` has only one output, and any number of inputs. These are different from signals because there are single values behind the pointers (as opposed to vectors of values).

Typically (but not always) UDS functions take the form of two or more ordinary differential equations that are implicitly related. Details of this theory will be discussed below. But, as a practical illustration of how this all works, consider the pair of equations:

[3] Currently, timelab samples are single precision floating point values.

$$\begin{aligned} \dot{x} &= ay \\ \dot{y} &= bx + cy \end{aligned} \tag{3.1}$$

To instantiate this system, we would create a node for each equation. We may call them `x_node` and `y_node`. In this case `x_node` would have one `data_in` field, pointing to the output of `y_node`. The node texttty_node, on the other hand, would have two inlets, one pointing to `x_node->data_out` and the other pointing to its own `data_out` pointer.

As in the case of signals, the outlets are instantiated in memory, but the inlets need not be. They can simply be equated to the desired outlet in order to share data.

We would then define a C function for each node that would contain (in the `x_node` case): `return a * *x_node->data_in[0];`.[4] This value (which is changing) is returned to the `tl_UDS_solver` object and corresponds to the derivative $\dot{x}$. The state of the integrated value x gets updated by the solver for every sample in a block of samples.

Our `tl_UDS_solver` maintains a list of all the nodes and simply cranks through them one at a time according to its numerical solving algorithm. The connections between the nodes must be established according to whatever system of equations we are solving. The nice thing about this is that given a system of ordinary differential equations of arbitrary complexity, it is very easy to develop code that will solve the whole system.

As a final note, timelab control objects are easily thrown into the mix so that the parameters `a`, `b`, and `c` in our example can be changed in real-time. As a convenience, the data type `tl_UDS_node` has a `void *extra_data` field. Groups of control objects associated with function parameters can be assigned to this general purpose pointer for ease of use.

[4]The translation of this C code is 'return the value stored in the variable `b` multiplied by the value pointed to by the variable `data_in[0]` which is a field in the data structure pointed to by the variable `x_node`.

3.7 Summary

The above is a brief and general description of how timelab can be setup to compute a UDS routine and how it actual goes about doing the computation. The details of how this is implemented will doubtless change over time, but the architecture itself has proved to be a stable and effective one.

One may notice that if timelab is merely computing the output of a `tl_UDS_solver` object and passing samples to/from it and a host application, it is more or less superfluous to provide an entire, general purpose DSP engine. Indeed, this extra layer of function calls does nothing but slow things down (i.e. increase CPU usage). However, this infrastructure exists in the hopes that timelab will continue to develop as a stand-alone solution for any deployment, including bare-metal. In order to keep timelab from relying on external infrastructure provided by operating systems, audio drivers, and host applications, this extra layer of organization is provided.

Chapter 4

Current Approaches to Real-Time Synthesis

As we all know, Digital Signal Processing (DSP) is an area of mathematics theory that is concerned with manipulating discrete time series. In audio processing, this is typically a digitally sampled analog input (like the voltage signal coming off a microphone) which is then altered in some way by software before being converted again into an analog signal (likely, another voltage that drives a loudspeaker). Alternatively, digital signals may be originally constructed by software itself (a digital oscillator, for example).

A large subset of the mathematical operations in the realm of music DSP are digital filters. Even processes that aren't commonly described in terms of filter nomenclature and symbols, are easily expressible in those terms (e.g. interpolation, oscillators, and discrete Fourier transforms can all be formulated in terms of filter theory).

In the time domain, linear digital filters are expressed in terms of a difference equation. A most general formula is:

$$a_0y(n)+a_1y(n-1)+\ldots+a_My(n-M) = b_0x(n)+b_1x(n-1)+\ldots+b_Nx(n-N) \quad (4.1)$$

or (neglecting the output gain factor a_0):

$$y(n) = \sum_{i=0}^{M} b_i x(n-i) - \sum_{j=1}^{N} a_j y(n-j) \quad (4.2)$$

in which form the term 'difference equation' is obvious. Basically this formula defines an output sequence, $y(n)$, in terms of a linear combination of some N of its previous values and an input sequence $x(n) \ldots x(n-M)$. This is a most convenient method for treating digital signals because it takes natural advantage of unit delay in its formulation. The transfer function of the delay operation ($y(n) = x(n-1)$) is $H(z) = z^{-1}$.

In a continuous time representation, an output signal $y(t)$ may be stated in terms of a linearly filtered input signal $x(t)$ in the following form:

$$a_0 y(t) + a_1 \frac{\mathrm{d}y}{\mathrm{d}t} + \ldots + a_n \frac{\mathrm{d}^n y}{\mathrm{d}t^n} = b_0 x(t) + b_1 \frac{\mathrm{d}x}{\mathrm{d}t} + \ldots + b_m \frac{\mathrm{d}^m x}{\mathrm{d}t^m} \tag{4.3}$$

or (again, ignoring the gain factor a_0)

$$y(t) = \sum_{i=0}^{M} \frac{\mathrm{d}x^m}{\mathrm{d}t^m} - \sum_{j=1}^{N} \frac{\mathrm{d}y^n}{\mathrm{d}t^n} \tag{4.4}$$

This appears very similar to the discrete time representation. However, due to the sampling theorem and the fact that the s plane does not map directly onto the z plane, there is not a straight forward relation between the coefficients of continuous time and digital filters. It is possible to approximately map a continuous time transfer function (a polynomial in s) to a discrete time transfer function (a polynomial in z) and a great deal of signal processing literature is concerned with this problem [RG75].

Digital computers may be more precise than analog computers, but the time resolution in the continuous time representation is literally infinite. In the digital representation, we are limited by the sampling rate and its inverse, the temporal width of the unit sample.

For most industrial applications, digital filters are quite obviously the correct choice. They are cheap, well understood and can be designed for guaranteed stability. However, when a musician walks in the room, she may have a very different constellation of needs as does an RFIC engineer (to take one example). The musician's thought processes runs like this: 'Hey, this digital filter sounds really great! What if I put two of them together?'. This is fine. There is no problem.

But then she thinks to herself 'well, in my analog synthesizer, I can control the cutoff frequency of my state-variable filter with one of my oscillators.' Now the filter is time-varying, and the straight-forward discrete version of the filter is no longer satisfactory.

4.1 Digital Filters for Real-Time Audio

One reason to eschew a continuous time representation is that digital filters may be very efficiently implemented by way of their impulse responses. Recall that the discrete Fourier transform (DFT) of a N samples of a signal is:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-2\pi i k n/N} \quad k = 0, 1, \ldots, N-1 \tag{4.5}$$

and that the impulse response of a filter $h(n)$ has a DFT of $H(k)$ that is its frequency response. We may also note that the the output of a digital filter can be computed as a convolution of the filter's impulse response, and the input signal:

$$y(n) = h(n) * x(n) \tag{4.6}$$

Convolution in the time domain is equivalent to multiplication in the frequency domain. Thus, we may write down the application of a digital filter as the inverse Fourier transform of the product of the short time Fourier transform of the input signal and the frequency response of the filter:

$$\begin{aligned} Y(k) &= H(k)X(k) \\ y(n) &= \tfrac{1}{N} \sum_{k=0}^{N-1} Y(n) e^{2\pi i k n/N} \quad n = 0, 1, \ldots, N-1 \end{aligned} \tag{4.7}$$

Given the highly optimized Fast Fourier Transform, and its inverse, this is an extremely efficient way to implement digital filters, especially long filters (i.e. ones with many coefficients). This is especially useful in the case of Finite Impulse Response (FIR) filters (these are ones that have zero valued feedback coefficients – a_j) as they tend to be of very high order. FIR filters are nice because they can easily be designed to have a linear phase response, which is often desirable.

This method doesn't lend itself well to real-time music systems, however, for two reasons. The first is that making use of short time Fourier series to sidestep

direct implementation in the time domain incurs a latency penalty that is equal to the length of the Fourier transform (N samples). The second is that filters expressed in this method are not easily time-varied, and in music, things that change over time are usually the more meaningful sounds.

Thus it is that there are a handful of small, elementary filters that get used over and over again in real-time digital systems. For example, Pd's [bp~], Csound's `reson` unit generator and cmusic's `nres` are all nearly identical two-pole resonant filters. Indeed the poles to these filters are identical, but they feature different zeros which affect the normalization of the filter. Moore describes this filter in detail [Moo90] and the similarity between this and the other two filters mentioned may be verified by inspecting the source code to Pure Data and Csound.

The transfer function of this filter can be written as:

$$H(z) = \frac{a_0}{1 - 2Rcos\theta z^{-1} + R^2 z^{-2}} \tag{4.8}$$

This is very nice, because given a sampling rate of SR, we may specify a center frequency of $\theta = 2\pi f_c/SR$ and approximate a bandwidth given by $R \approx e^{-\pi B/SR}$.

However, the above filter is linear and will remain stable if varied slowly. If we want to represent either a nonlinear filter or one whose response varies quickly with time (which is the case of the popular analog Moog ladder filter and its kin) we need to develop a different theory of representation.

4.1.1 'Traditional' Digital Oscillators

One can think of a sinusoidal oscillator as an extremely tight bandpass filter. Indeed, it can be formulated this way. All we need to do to create such a filter is place a pair of complex conjugate poles on the unit circle in the z-domain that correspond to the desired frequency.

For example, given a sampling rate of 48kHz, and a desired frequency of 1kHz, the pole-zero plot and impulse response of such a filter would look like Figure 4.1. The filter is designed essentially the same way as the filter shown in Equation 4.8. Noting that we wish to have maximum resonance at $f_c = 1$kHz at a sample rate of $SR = 48$kHz, and that in order to have real output from a complex filter,

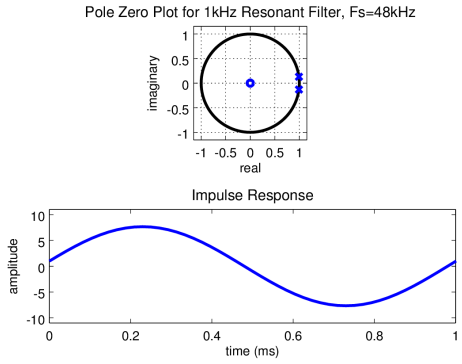


Figure 4.1: Pole-Zero plot and impulse response for a digital filter as oscillator.

we need to include a complex conjugate pair for each pole, the transfer function is simply:

$$H(z) = \frac{1}{(1 - e^{i2pif_c/SR}z^{-1})(1 - e^{-i2pif_c/SR}z^{-1})}. \tag{4.9}$$

This is identical to Equation 4.8 with a bandwidth of 1Hz.

To determine the difference equation, we simply combine the polynomial coefficients in $z^{-}1$ to obtain:

$$y(n) = x(n) - 1.9829y(n-1) + y(n-2) \tag{4.10}$$

A simpler method (and much more versatile method) for realizing a digital sinusoidal oscillator, is to precompute 1 period of a sinusoid, store the values in a large table, then reference each value given a time varying index value. Mathematically, this can be expressed as

$$\begin{aligned} x(0) &= \phi \\ y(n) &= \sin(2\pi x(n)) \\ \delta &= \tfrac{\omega}{SR} \\ x(n+1) &= x(n) + \delta, \end{aligned} \tag{4.11}$$

where ϕ is the initial phase, ω is the desired frequency and SR is the sampling rate as before.

In practice, however, instead of computing the sine function at every step (as noted above) this is pre-computed and stored in a table. $x(n)$ is then wrapped so that it goes from 0 to 1 once every ω Hz and this output is multiplied by the

length of the table to determine which value to reference.[1] Hence, this kind of oscillator is known as a 'lookup' oscillator. It is very handy and we will refer to it again.

4.2 Nonlinear Real-Time Digital Networks

For a digital system to be linear it must satisfy the condition that for an input sequence $x(n) = Ax_1(n) + Bx_2(n)$ that produces an output sequence $y(n)$ it must be the case that $y(n) = Ay_1(n) + By_2(n)$. Indeed, in order to be linear, any linear combination of inputs must be proportional to the same linear combination of outputs. A consequence of linearity is that the the (linear) transfer function $h(n)$ that relates $x(n)$ to $y(n)$ can alter the amplitude and phase of $x(n)$, but not its frequency. This rule applies to continuous time systems as well.

As Moore points out, the term *nonlinear* is not particularly meaningful:

> ... talking about the class of nonlinear systems is like talking about the class of all animals that are not elephants.[Moo90, 315]

And so, while the basic building blocks of LTI systems and clever implementations of such systems can buy a lot of sounds in computer music, it is only a very small subset of total possiblities.

Many nonlinear effects are quite common. Compression, to take an example, is one of the most frequently used audio effects in studio production. Limiters are also commonplace as is guitar distortion (which is really just clipping). For a discussion of these techniques as well as the effect of triode and pentode tubes see [DDHZ11]. One may also consult Chapter 3.5.4 of [Moo90] for a discussion of nonlinear waveshaping.

Other typical nonlinear synthesis techniques include frequency and amplitude modulation. Both these forms of modulation add harmonics to the output of the system, and thus do not satisfy the conditions of linearity.

[1] Obviously in order to provide good smoothness, some interpolation is needed. Incidentally, this is exactly how `stdlib.h` in the C programming language computes the output trigonometric functions.

4.2.1 Clipping

To create nonlinear harmonic distortion, a simple clipping rule such as:

$$y(n) = \begin{cases} .8 & \text{for } x(n) >= .8 \\ -.8 & \text{for } x(n) <= -.8 \\ x(n) & \text{for } -.8 < x(n) < .8 \end{cases}$$

can be applied. This 'hard' clip will create harmonic distortion so that the ouput signal has frequency content that the input lacks. This can easily be verified by inspecting the magnitude spectra of a sinusoid and the same signal after the above clipping rule is applied.

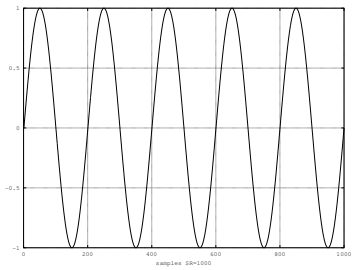
Incidentally this symmetrical hard clipper can be used in conjunction with the two-pole filter we discussed above (or any filter for that matter) to create a Virtual Analog filter. The scheme shown in Figure 4.3 is described by Rossum in [Ros92] and was also used in the Emu EMAX II synthesizer which came out in 1989[VBS+11].

The clipping technique can construct a familiar set of 'analog-sounding' effects (clipping is the basis of guitar distortion after all) and it is very easy to implement. Utilizing so-called 'soft' clipping functions, such as hyperbolic tangents or cubic functions, provide a way to parametrically tailor sounds. Furthermore, by studying the clipping characteristics of analog circuit componenents, one may inform her choice of clipping function to yield a particular set of nonlinear harmonic distortion to match the character of some analog device.

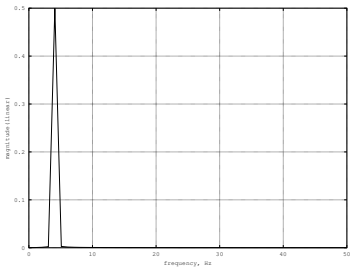
Waveshaping Clipping is a specific kind of a general nonlinear technique known as waveshaping. Waveshaping is accomplished by passing a signal $x(n)$ through some function (which need not be a clipping function) to produce a harmonically distorted $y(n)$. Mathematically, $y(n)$ can be seen as a composition of the waveshaping function $g(n)$ and the input signal $x(n)$:

$$y(n) = g(x(n)) \tag{4.12}$$

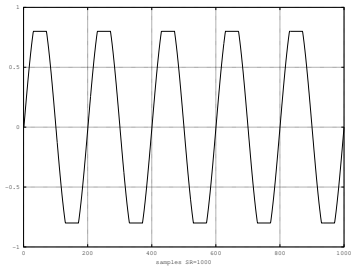
Incidentally, this formulation suggests a very simple digital implementation. The function of $g(n)$ can be precomputed and stored in a lookup table. Then the



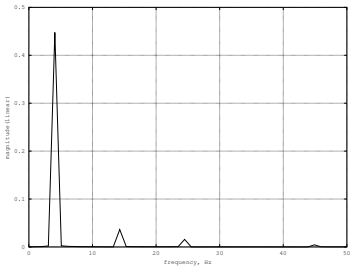
(a) 5Hz sinusoid



(b) Magnitude spectrum of sinusoid



(c) The same sinusoid clipped



(d) Magnitude spectrum of clipped

Figure 4.2: A 10Hz sinusoid, its magnitude spectrum, and the clipped version. Note that there is new harmonic energy in the clipped sinusoid's spectrum.

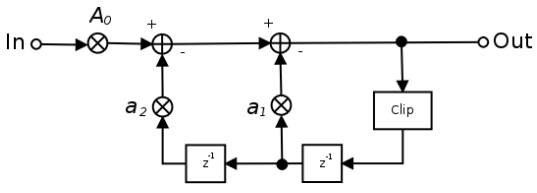


Figure 4.3: Filter diagram for a two-pole filter with clipping function.

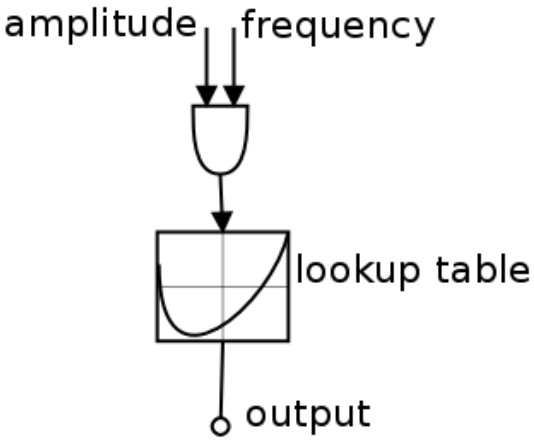


Figure 4.4: Signal flow diagram for a digital waveshaping algorithm.

values of $x(n)$ can simply be used as lookups to this stored function. Clearly, unless $g(n)$ is a scalar function, this system will be nonlinear. For detailed description of waveshaping and its applications, please refer to [Moo90], [DJ97] and [Puc07].

FM Synthesis A frequently used waveshaping algorithm is Frequency Modulation (FM). In FM the both the waveshaping function and the input signal are sinusoidal. The harmonic distortion that results from FM synthesis well studied and the system itself is highly stable in reasonable realm of operation. Thus, FM has been a staple of computer music ever since its serendipitous discovery as a musical tool by John Chowning in the 1970s [Cho77]. Most famously, the Yamha DX-7 was designed to utilize the rich variety of sounds that could be generated with FM [Thé97].

FM works by modulating the prescribed frequency of a sinusoidal oscillator with some other sinusoid. The time domain form of the function may be written as:

$$x(n) = \cos(A\cos(\omega_m n) + \omega_c n) \tag{4.13}$$

where ω_m is called the *modulation* frequency and ω_c is the *carrier* frequency. A is a scalar that we may take to be the *depth* of modulation. The spectrum of an FM signal is predictable given these three parameters. There are numerous texts that formulate the Bessel functions that describe this spectrum (again, see for example [Moo90], [DJ97] and [Puc07]).

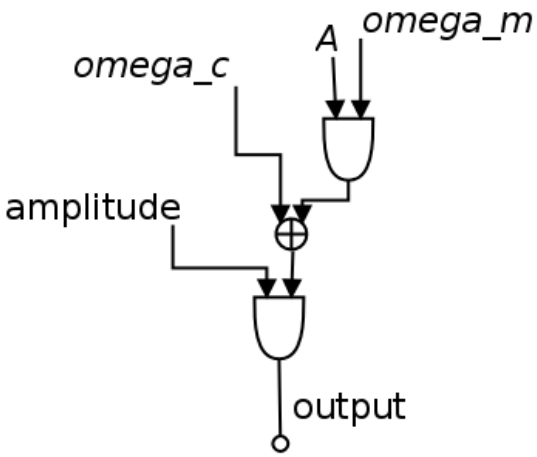


Figure 4.5: Signal flow diagram for a digital Frequency Modulation scheme.

Since it is waveshaping, FM is easily implemented as a digital algorithm. Provided we have an oscillator whose frequency may be dynamically varied, we simply use the output of some other oscillator whose frequency is ω_m and whose amplitude is scaled by the factor A and sum this with whatever carrier frequency ω_c that we want. This time-varying sum is then used to control the frequency the other oscillator.

Depending on the values of ω_m and A (both of which may be time-varying), FM synthesis produces new harmonic content in evenly spaced intervals about the center (carrier) frequency ω_c. If ω_c is sufficiently low and the spread of the new harmonics is sufficiently wide, this may produce undertones that are less than 0Hz. These negative frequencies are then 'mirrored across the DC frequency which may produce a rich, inharmonic spectrum. Similar foldover effects can be created with other waveshaping functions.

The structure of FM (one oscillator modulating another of exactly the same type) immediately suggests enlarging the synthesis network. One way would be to add extra oscillators and create modulation signals out of an FM signal. As Puckette points out, this is well-trodden ground[Puc07]. Another possibility is to use non-sinusoidal oscillators in the network.

Yet another possibility is to provide a feedback loop between the modulat*ee* and the modulat*or*. However, this creates an implicit relation and there will be

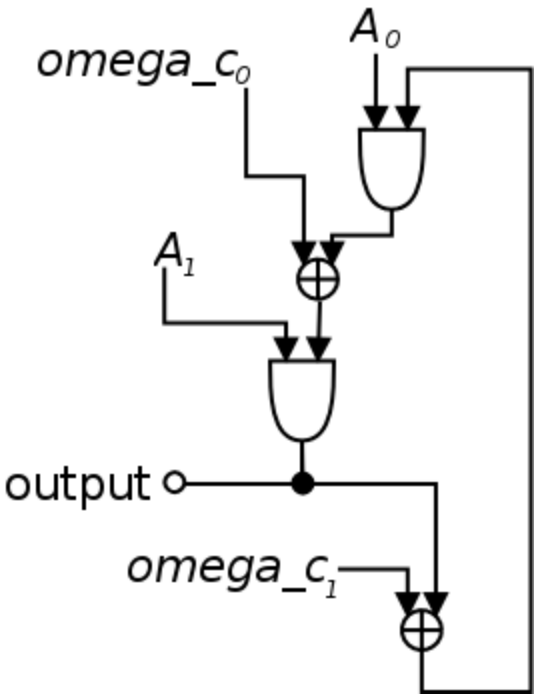


Figure 4.6: Frequency Modulation scheme with feedback.

a delay greater than or equal to one sample if the network is implemented with lookup oscillators. We shall return to this point and study a UDS implementation of just such a network

4.3 Physical Modeling and Virtual Analog

The techniques outlined above may be used to emulate the sounds of real world instruments or analog sound circuits. However, they are not physics based. This distinguishes them from what we call physical modeling and virtual analog where digital filters are formulated by physically informed ideal models of wave behavior. In bot the case of Digital Wageguides and Wave Digital Filters, we see a formalism for defining LTI systems that maps to a physical structure according to the rules of that formalism. In both cases, nonlinearities may be coupled to they system, expanding their sound pallettes. The difference between this way of doing things and what we here call UDS is that here, nonlinearities are special cases and implicit structures are very cumbersome. UDS, on the other hand is

much more general in that nonlinear systems are exactly as viable as linear ones. Implicit structures are also perfectly reasonable to represent using UDS, but there is a limit to the generality of this claim, which will be discussed in more detail in the next chapter.

Physical modeling of acoustical spaces, instruments, and analog hardware is a rich sub-field of computer music and electrical engineering. For a concise introduction to the state of the art, consult [VPEK06]; and, (of course) J. O. Smith's definitive text on the subject [Smi10]. As mentioned above, in physical modeling, the idea is to analyze the physical laws that control the behavior of the system itself and then compute a virtual replica given that analysis.

In order for all of this to work, a great deal of simplification and idealization must be done. This may tarnish the result by stripping the system being modeled of some inherent, characteristic richness. The trade-off here is simplicity, control, stability, and precision.

It is often the case that physical models operate on so-called Kirchoff variables (voltage/current, force/velocity, pressure/volume velocity, etc.), or wave variables (bi-directional sections of a traveling wave)[KES03]. Thus, when we switch between K-variables and W-variables, a transform is involved. For example, if we are modeling the behavior of an electrical circuit, we are concerned with the K-variables voltage and current (u and i respectively). But, if we wish to model the circuit in terms of W-variables (the incoming and reflected waves at a particular circuit element) we would be speaking about W-variables a and b. The nature of this relation is particular to the thing being modeled. This will be discussed further in regards to Digital Waveguides (DWGs) and their kin, Wave Digital Filters (WDFs).

4.3.1 Digital Waveguides

Digital Waveguides (DWGs) are digital models of analog systems based on the d'Alembert solution to the one dimensional wave equation:

$$\frac{\partial^2 y}{\partial x^2} = c^2 \frac{\partial^2 y}{\partial t^2} \tag{4.14}$$

As can be seen this is a partial differential equation in time and space. It says that in any point in a uni-dimensional medium (e.g. an ideal string or a column of air) the acceleration, $\partial^2 y/\partial t^2$ is proportional to the curvature of the medium at that point,$\partial^2 y/\partial x^2$. In acoustic models, the proportion is the square of the speed of sound c^2. In a more compact form, the wave equation can be written:

$$\epsilon \ddot{y} = K y'' \tag{4.15}$$

In which form the propagation speed c can be expressed:

$$c = \sqrt{K/\epsilon}. \tag{4.16}$$

It was shown by d'Alembert[d'A73] that this equation can be solved by treating the wave over the one dimensional surface as a sum of two other waves moving in opposite directions:

$$y(x,t) = y_r(x - ct) + y_l(x + ct) \tag{4.17}$$

or

$$y(x,t) = y_r(t - x/c) + y_l(t + x/t) \tag{4.18}$$

This solution is valid for any arbitrary wave that can be expressed as a weighted sum of even and odd functions (for example, any continuous function).

It is then easy to imagine how this can be digitized, providing the wave we model is band-limited to the Nyquist frequency:

$$y(x_k, t_n) = y_r(n - k) + y_l(n + k) \tag{4.19}$$

where the indices n and k access times and places quantized by the time step (inverse of the sampling rate $1/f_s$) and the spatial step (cT) respectively. This can then be implemented as a bi-directional delay line. This idealized model can be enhanced by implementing digital filters that categorize wave behavior at the boundaries of the delay lines. This is depicted in Figure 4.7.

At one end of Figure 4.7 is a perfect inverting reflection ($H(z) = -1$—an idealized string termination at one end of the medium) and a lowpass filter that

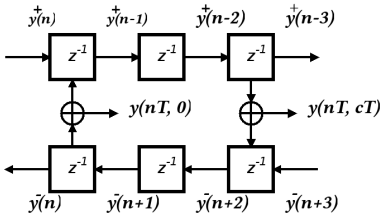


Figure 4.7: A depiction of the digital waveguide as a network of filters and delay lines.

characterizes the transfer function at the other end ($H(z) = G(z)$).[2] Furthermore physical details such as losses, bending stiffness, and frequency dispersion can all be characterized by digital filters which can be lumped together at an arbitrary point in the system.

The DWG formulation is the basis of a very large number of physical modeling schemes. Although the term was coined by Smith in 1985[Smi85], its roots lie in the Kelly-Lochbaum algorithm for vocal synthesis[KL62], the algorithm used to synthesize the vocal part to the famed 'Daisy Bell'[3].

In that implementation, the vocal tract was modeled as a series of tubes of different lengths and widths. The wave flow through the tubes is modeled as a bi-directional delay and the junction between the tubes is modeled with N-port scattering junctions. Thus each tube can be characterized by its physical dimensions (its width and length) as well as its admittance/impedance in relation to the tubes at either end (this is the scattering junction). Indeed this scattering junction is often referred to as the 'Kelly-Lochbaum implementation' in the literature.

The junction is characterized by the impedances of either delay line at the junction (which is a function of the physical dimension of the tube). If this impedance is denoted R_n the scattering regime at the junction between junction n and $n+1$ is based on the value $k_1 = R_2 - R_1/R_1 + R_2$. Note that this is the case for a 2-port scattering junction, but the theory is generalizable to N-ports. For a thorough treatment see [Smi10] and section 5 of [VPEK06].

[2]In the case of a violin model, this transfer function can be ascertained by measuring the impulse response of a violin body at the bridge.

[3]https://www.youtube.com/watch?v=41U78QP8nBk

DWGs are also valid in two and three dimensions. DWGs have been used to model percussion membranes in 2D and 3D resonators such as rooms[MNH01] and instrument bodies[HSS00].

One cost of DWG synthesis is the warping that occurs from the approximations associated with digitization schemes. The most obvious approximation is that of time/space quantization. For example, in a simple DWG of a string, the pitch of the string is determined by its length, unit mass and tension. Unit mass and tension determine the rate at which waves travel along the string (c). However, c being held constant, pitch corresponds to string lengths that are integer multiples of the spatial unit. So, if we desire a pitch that is not exactly one of these values (e.g. if we wish our model to change pitch continuously from one frequency to another) we have a problem. Fortunately, all-pass filters can rescue us from this dilemma since their phase delay is a convenient, accurate and controllable digital method for fractional delay [JS83].

In addition to digital filters, (presumably linear, time-invariant ones) nonlinear functions may also be coupled to DWGs. Indeed, this is necessary if one wishes to model a piano-hammer (which deforms as it strikes a string) or a nonlinear driving force such as a violin bow or clarinet reed. We may not attach nonlinear functions to a digital waveguide (or any network of unit delays) without impunity. As has been stressed often is the fact that nonlinearities coupled to a waveguide may not converge, or may converge very quickly to 0.

4.3.2 Finite Difference Time Domain

Finite Difference Time Domain (FDTD) is a direct solution to the wave equation using finite differences to approximate the limit as h approaches zero in the derivatives. This is another implicit instance of a digital filter in digital synthesis. But, unlike DWGs (which use W-variables) FDTD uses K-variables to create difference equations. This approach to solving the wave equation for the purpose of modeling vibrating bodies is not new. Hiller and Ruiz introduce theses methods in 1971 [HR71], but in recent years there has been more interest in the method, particularly in the simulation of complex, multi-dimensional structures

[Bil07] [Bil09], but also as an alternative or as an enhancement of 1D waveguide structures [KES03] [KE04]. It is also one of the most useful techniques for simulating fluid dynamics in the realm of computer graphics (a very similar problem to multi-dimensional acoustic modeling) [Bri08].

There are a number of difference schemes that can be applied to solving the wave equation (or any differential equation for that matter). A generic, and frequently used method, is the so-called leapfrog method. This is a numerical method for solving second order partial differential equations based on the central difference scheme.

$$\begin{aligned} \ddot{y}(n,k) &\approx \tfrac{(y(n+1,k)-2y(n,k)+y(n-1,k)}{h^2} \\ y''(n,k) &\approx \tfrac{(y(n,k+1)-2y(n,k)+y(n,k-1)}{\delta x^2} \end{aligned} \tag{4.20}$$

Here h is the finite time step and δx is the spatial difference. If we choose $h = 1/f_s$ (the width of one sample at the sampling rate, and $\delta x = ch$, where c is the speed of sound in the medium we are modeling (which can be determined by physical observation), we are led to the solution for finding the next value in time at each non-zero width 'point' along the medium we model:

$$y(n+1,k) = y(n,k+1) + y(n,k-1) - y(n-1,k). \tag{4.21}$$

This equation is known as leapfrog recursion. It says that the next point in time at point in space k is determined by summing the values immediately adjacent to that point right now, and the value this point in space had one sample in the past. We then shift from time to space, and do it again with time and space variables reversed. This is done for every point in the model in both space and time.

One advantage of this method is the fact that we can choose arbitrarily small sampling intervals so that very precise approximations are possible. Since this structure may be coupled to DWGs, FDTD models describing very nuanced behaviors can be inserted into simpler models, thereby significantly expanding the power of the DWG. This has, for example, been done to model very fine detail in bow-string interaction [PW98].

The main difference between FTDT (and DWGs, which turn out to be isomorphic [Smi10] and what we refer to here as UDS is that in FTDT, differential equations are transformed into difference equations. For example, in the

case of leapfrog recursion the limit as h approaches 0 in the time derivative, is approximated by the difference in time between successive audio samples:

$$\frac{\mathrm{d}x}{\mathrm{d}t} \triangleq \lim_{h \to 0} \frac{x(t) - x(t-h)}{h} \approx \frac{x(nT) - x[(n-1)T]}{T}, \tag{4.22}$$

where $T = 1/F_s$ – the inverse of the sampling rate (which, as mentioned, can locally be set arbitrarily small). Note that the approximation at the end of Equation 4.22 is implicit to Equation 4.20. This is ultimately a matter of psychology. In UDS we approach problems with the belief that we are actually operating in the continuous time realm, with differential rather than difference equations. What this buys is the freedom to explore models that contain zero delay feedback loops without having to go through the heavy lifting of accounting for this in the digitization process. We can simply write a system of nonlinear differential equations that are coupled together and listen to the output.

This approach also affords the user the pain of having to discretize the differential equaitons himself. This is relatively trivial for simple, well defined systems such as Equation 4.9, but when dealing with massive and aribitrarily defined systems (like a 2D FTDT representation of a drum head, for example) the work involved in discretize the system may be cumbersome (particularly if the system is nonlinear). Automating such procedures is an active topic of research in physical modeling and virtual analog. We will return to this point in our discussion of recent advances in WDFs and the state-space representation.

4.4 Virtual Analog

Virtual analog (VA) is a field of computer music research concerned with physics-based emulation of analog electronic circuits. Much VA research is a subset of physical modeling as it is based on digitizing the continuous time behavior of circuit components. However, there are also VA algorithms that are not physics-based, some of which are discussed below.

Recently, VA has begun to occupy a substantial market share of music and computing research [PV11]. Since the release of the Nord Lead synthesizer in 1995

[Smi96], the pursuit of analog 'sounding' digital synthesis has been an active and marketable research field.

It is a very similar problem to physical modeling in that electrical behavior can be modeled as either K-variables (in this case voltage and current) or W-variables. In general, though, W-variables are inconvenient in VA due the extremely high speeds at which the 'waves' in electrical circuits flow. This leads to a great challenge in the implementation of VA synthesis, because the time delay between interactions at the nodes of a circuit is much nearer to instantaneous than it is to the length of a reasonably chosen time for one sample (approximately 1.5ms at 44.1kHz sampling rate). This problem also gives rise to scheduling problems because of the structural reality of delay-free loops in analog circuits.

In general the problem is side-stepped by treating analog feedback as an instantaneous delay. Such delay-free loops are said to be non-computable, but there are techniques for coercing output of such systems. The first such method is to use iterative solvers to approximate the output of implicit equations. This is inconvenient in the case of real-time synthesis because it may take a long time for such a solver to converge on an accurate enough solution. The so called K-method (cite) is a popular technique for rendering implicit nonlinear systems (which we often encounter in analog networks) computable in real-time. The draw back is that this method requirese the pre-computation of large lookup tables. Also, in certain cases, a system may be such that a matrix inversion is required every time a parameter(e.g. the cutoff frequency of a filter) is adjusted. We will turn to some such examples at the end of this section.

4.4.1 Differentiated Parabolic Waves

Differentiated Parabolic Waves (DPWs) form a class of synthesis techniques that are designed to reduce aliasing in digital waveforms [Val05] [VH06] [VNSA10]. Aliasing (or fold-over) occurs due to the discontinuities in certain waveforms such as the triangle, square, and sawtooth waves.

In Figure 4.8 the spectra of a digital sawtooth before and after DPW are shown. The digital sawtooth is given by a phase accumulator and modulo function.

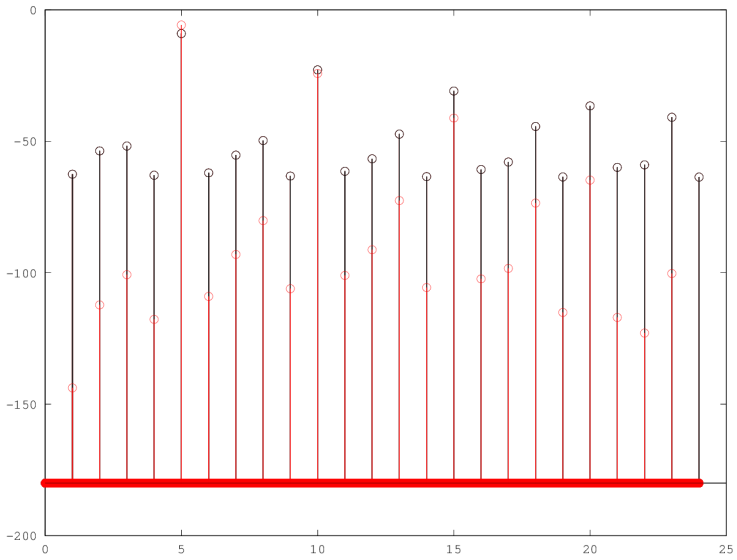


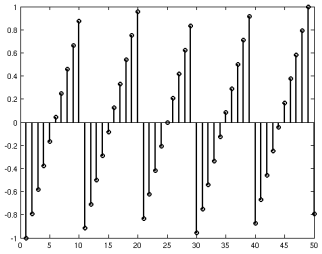
Figure 4.8: The magnitude spectrum (unwindowed) of a 5kHz digital sawtooth at a 48kHz sampling rate (black) and the same sawtooth after DPW (red). The aliasing effects are significantly attenuated.

It is then scaled so that it goes between ± 1 rather than 0 and 1. This signal is then squared, creating a parabolic wave. This is then differentiated, yielding a sometimes smoother sawtooth waveform (which needs to be again scaled in order to keep it normalized to ± 1 [VH06]). A number of other classical waveforms can be generated using similar techniques.

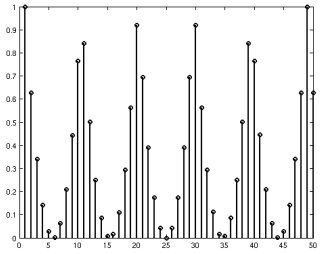
4.4.2 Non Zero Time Delay

An improvement to the DPW methods which is outlined in [VH06] is to use a more sophisticated differentiator than the one proposed therein. In that formulation, the digital differentiation is given by the transfer function $H(z) = (1 - z^{-1})(1 + z^{-1})/2 = (1 - z^{-2})/2$, a very simple FIR filter. A second order filter improves upon the first order solution ($H(z) = 1 - z^{-1}$) by attenuating the upper harmonics more steeply, thus limiting the possibility of some aliased partial beating with a desired one. It follows that a longer filter would improve this situation, but it would also complicate the algorithm and increase the latency.

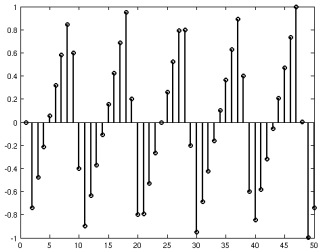
There are other methods. One is the direct method of computing the digital



(a) Digital sawtooth.



(b) Parabolic wave.



(c) Differentiated parabolic wave.

Figure 4.9: The stages of creating a DPW sawtooth wave.

wave at some integer multiple of the sampling rate, attenuating the frequencies that will alias, then under-sampling at the last stage. Another is to introduce ramps and rounded corners into the wave (see for example Chapter 10 in [Puc07]). Yet another is simply to create the desired wave form using modal synthesis (i.e. representing it as a sum of sinusoids appropriately attenuated and phase shifted at integer multiples of the desired fundamental frequency). This is sometimes called a 'band-limited' wave form because we may easily control for frequencies that reach above the nyquist frequency.

DPW is an attractive technique because it provides a way to synthesize high-frequency digital waveforms without resorting to oversampling. However, the factor by which the DPW must be normalized is frequency dependent. Using a FIR filter as a differentiator introduces a zero into the system. The effect of the filter on amplitude is frequency dependent (by definition) and this must be corrected for in order to have a normalized waveform.

4.4.3 Wave Digital Filters

Wave digital filters (WDFs) were developed by Fettweis in the late 1960s [Fet71]. Fettweis also provides a detailed description of the theory and some applications in [Fet86]. These were created with analog circuit emulation in mind.[4]

As the name implies, WDFs operate on W-variables. There are two kinds of building blocks in WDFs: *elements* and *adapters*. Elements are digital filters that emulate the behavior of circuit components (resistors, capacitors, voltage sources, etc.) and adapters are N-port scattering junctions that define the nature of the connection between elements. There are two types of adapters: series and parallel.

Both adapters and elements have associated impedance (also called port resistances and normally denoted Z_p) depending on their type. In order to determine the correct behavior, we must first convert from K-variables to W-variables. This is given by the relation:

$$\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} 1 + Z_0 \\ 1 - -Z_0 \end{bmatrix} \begin{bmatrix} u \\ i. \end{bmatrix} \tag{4.23}$$

We will limit our discussion to the electrical realm and will discuss WDFs in terms of current(i) and voltage (u), but WDFs are applicable to mechanical systems as well.[5] Also, the theory will be confined to one-port adapters – capacitors, resistors, etc. – but two-port adapters such as gyrators and QUARLS are discussed in Fettweis [Fet86].

To go back to K-variables, we do the reverse of the operation to obtain the W-variables:

$$\begin{bmatrix} u \\ i \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1/Z_0 & -1/Z_0 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} \tag{4.24}$$

Now, we define the *port reflectance* as the Laplace transform of the reflected wave divided by the Laplace transform of the incoming wave.

$$S_z(s) = \frac{\mathcal{L}b}{\mathcal{L}a} = \frac{\mathcal{L}u(t) - Z_0 i(t)}{\mathcal{L}u(t) + Z_0 i(t)} = \frac{Z(s) - Z_0}{Z(s) - Z_0} \tag{4.25}$$

[4]The derivation of WDFs that proceeds from here is essentially summary to that given in [VPEK06] and [VBS+11]

[5]For a development of WDF theory in terms of mechanical, rather than electrical, elements, see [Smi10].

In order to determine the appropriate values for Z_0 for any given component, we must apply the bilinear transform, and substitute the following values for Z_0:

$$Z_0 = \begin{cases} R & \text{for Resitance } R \\ 1/2CF_s & \text{for capacitance } C \\ 2LF_s & \text{for inductance } L \\ \infty & \text{for an open circuit} \\ 0 & \text{for a short circuit} \\ 0 & \text{for a voltage source} \\ \infty & \text{for a current source} \end{cases}$$

where F_s is the sampling rate. The corresponding z-transforms of the associated components' reflectances are given by:

$$S(z) = \begin{cases} 0 & \text{for Resitance } R \\ z^{-1} & \text{for capacitance } C \\ -z^{-1} & \text{for inductance } L \\ 1 & \text{for an open circuit} \\ -1 & \text{for a short circuit} \\ 2u - 1 & \text{for a voltage source} \\ 1 - 2u & \text{for a current source} \end{cases}$$

where u is the voltage generated by the source.

Now we have the building blocks of circuitry and their associated, digital transfer functions. These components are then connected by the adapters. The adapters are simply N-port scattering junctions whose refelctances are determined by the port impedance values of the elements that are connected to them. This structure suggests a tree wherein the adapters are nodes and the elements are leaves.

Figure 4.10a shows the waveflow for a serial adapter, Figure 4.10b shows a parallel adapter, and Figure 4.10c shows an imaginary WDF schematic with a nonlinear 'root' element. The root element is the center of the computation schedule and is often of a special nature. The order of operations is first to compute the waves traveling up towards the root, reflect off the root, and then calculate

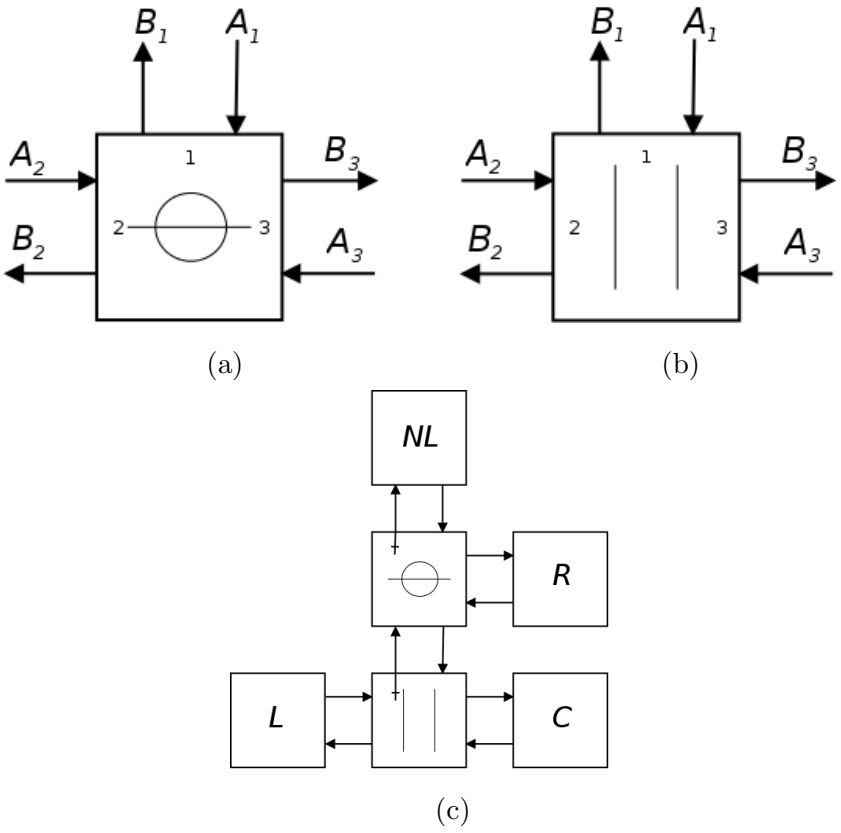


Figure 4.10: A serial adapter, parallel adapter, and simple WDF schematic after [VBS+11].

the waves going down towards the leaves and repeat. The crosses on the arrows pointing towards the root indicate that these ports are *adapted* ports (more on this in a moment).

The behavior at the adapters is nothing unusual and is derived from the same theory of scattering junctions that is used in DWG synthesis. For our three port examples in Figure 4.10c we have:

$$b_n = \begin{cases} a_n - Z_n(a_1 + a_2 + a_3)/(Z_1 + Z_2 + Z_3) & \text{for a serial port} \\ 2(Y_1 a_1 + Y_2 a_2 + Y_3 a_3)/(Y_1 + Y_2 + Y_3) & \text{for a parallel} \end{cases}$$

where b_n is the outgoing wave at point n, a is the incoming wave, Z_n is the port impedance and Y is admittance (the inverse of impedance).

If the root is nonlinear, it poses a problem. The adapter needs to know both its own port impedance and the incoming waves in order to compute the outgoing wave. Since we haven't yet determined the wave coming from the root (it is the last scattering junction we compute) we are faced with an implicit problem. Oddly enough, choosing the correct value for the port impedance at the output of an adapter can eliminate this dilemma. Note that by choosing $Z_1 = Z_2 + Z_3$, where Z_1 is the impedance facing the root, Equation 4.4.3 simplifies to:

$$b_1 = \begin{cases} -a_2 - a_3 & \text{for a serial port} \\ Y_2/(Y_2 + Y_3)a_2 + Y_3/Y_2 + Y_3)a_3 & \text{for a parallel port} \end{cases}$$

This altered port is what is meant by the term *adapted* port.

In order to determine the reflections at the elements, we may simply refer to the transfer functions, which we already know. For example, in the case of the resistor, the outgoing wave is 0 (the entire wave is passed), and for a capacitor it is the previous state of the capacitor (again, we see unit delay in effect).

Knowing the reflections at the elements and the nature of the scattering at the adapters we can completely characterize the network, except for the behavior at the root. Recall that we chose a special value of Z_1 at the junction between the root and the adapter immediately below it in order to adapt the port. Now, by definition of a scattering junction, the impedance looking into the root should be the same at both the adapter and the root. But, we also have a signal dependent

impedance at the root Z_r by its definition. The way to justify this paradox is to set the reflected wave from the root as:

$$b_r = \frac{Z_r - Z_1}{Z_r + Z_1} a_r \tag{4.26}$$

where a_r and b_r are the outgoing and incoming waves at the root, and Z_1 is the port impedance that was adapted for the adapter. It is now possible to compute the wave behavior of the network without resorting to iterative methods at any point.

Augmenting Wave Digital Filter Theory WDFs are an attractive choice for virtual analog and physical modeling. They are computationally cheap and highly modular. Furthermore they have been used to model a variety of nonlinear behaviors: a piano hammer (with a felt tip whose stiffness varies at the moment of impact) [BBMS03] [VDPS94]; tone-hole modeling in woodwind models [vWC03]; and in various virtual analog models [VBS+11] [PK10] [PTK09].

In this explanation, WDFs are presented in a uni-dimensional context so that time, but not distance is considered. This is fitting for the simulation of circuits and (relatively) minute physical objects like the felt on a piano hammer or a tone-hole. But, multidimensional structures are clearly realizable, though exceedingly complex. Fortunately, WDFs can be coupled to other models, such as DWGs, as shown in [VPEK06].

There are several drawbacks to WDFs. The most obvious is that the root element is given special precedence over the other elements and adapters. This greatly limits the modularity. Indeed, the only nonlinearity that is representable is the root element itself (although there is a class of special elements called 'mutators' that are not discussed but can emulate certain kinds of nonlinearities without being the root [SDP99]).

The other drawback of WDFs is that they cannot represent delay-free structures. Recently, the K-method [BDPR00] has been harnessed to overcome this difficulty in the realm of WDFs [WNSIA15] [WSIA15].

Essentially the K-method operates on a dynamical state-space system of

the form:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{C}\mathbf{i}(\mathbf{v}) \tag{4.27}$$

$$\mathbf{y} = \mathbf{D}\mathbf{x} + \mathbf{E}\mathbf{u} + \mathbf{F}\mathbf{i}(\mathbf{v}) \tag{4.28}$$

$$\mathbf{v} = \mathbf{G}\mathbf{x} + \mathbf{H}\mathbf{u} + \mathbf{K}\mathbf{i}(\mathbf{v}) \tag{4.29}$$

where $\mathbf{x}$ is the 'state', $\mathbf{u}$ is an input vector, $\mathbf{y}$ is the output and $\mathbf{i}$ is a nonlinear vector function of $\mathbf{v}$.

The nonlinearity in Equation 4.29 can be solved with an iterative solver such as Newton-Raphson method. Furthermore multidimensional lookup tables can be pre-computed to avoid convergence issues, rendering the solution more suitable to real-time computation. The method also requires discretizing Equations 4.27 - 4.28 (e.g. with Backward Euler or the trapezoidal rule). A consequence of this is that if an adjustable parameter presents itself in the discretized version of matrix $\mathbf{A}$ it may be the case that multiple matrix inversions need to be performed in order to find the correct discretized coefficients that correspond to $\mathbf{B}$, $\mathbf{C}$, etc. There are times when analysis can resolve this issue (as in [DHZ10]), but this difficulty can potentially lead to significant slowdown if controllable parameters are desired.

4.5 Summary

Much of the above is not news to anyone working in digital audio. However, we present commonly used linear and nonlinear synthesis techniques such as digital filters and waveshaping, because a different, UDS implementation of these techniques is presented in the next chapter. The concept of Virtual Analog is also presented as is physical modeling since we will see that that UDS offers some interesting solutions to problems in these realms as well.

We dwell on the practice of representing systems as LTI filters for digital synthesis to emphasize that this is only a small subset of the possibilities that exist. Indeed, the main purpose of UDS is to access more of these possibilities. The reason for explicating DWGs and WDFs is to make the point that there can be a tendency to approach digital audio problems as ones that can building structures

that reduce to the language of digital filter theory; and, that nonlinear elements are treated as perturbations of something that is more inherently linear.

The extension of WDFs with the K-method is an example of this kind of psychology. This is not to say that WDF theory extended with the K-method isn't a powerful tool, because it is. However, our goals here motivate us to seek alternative approaches. Namely, we want to hear the output of a dynamical system immediately, without having to formulate large lookup tables and perform deep analysis in advance. It is *fun* to audition dynamical systems immediately. Furthermore, as mentioned above, it is often the case that a circuit (real or theoretical) represented as a state-space system such as the one given in Equations 4.27 - 4.29 may result in a system wherein a parameter that we wish to control in real-time (e.g. a potentiometer that determines the cutoff frequency of a nonlinear filter) finds its way into a matrix that needs to be inverted in order to solve the system.

With these concerns in mind, UDS is a technique for creating and solving dynamical systes that are restricted to the form given in Equation 1.1. This differs from Equations 4.27 - 4.29 in that each equation is an ordinary differential one. Equations 1.1 show only a two-dimensional system, but any dimensions (within the bounds of computational limitations) may be used.

What we buy by restricting ourselves thus is that we can represent implicit systems with explicit equations. This means that we can solve the system numerically in real-time without having to build any lookup tables, invert any matrices; and, we are free to parameterize the system in any way we wish without incurring extra computational cost beyond that of passing the parameters to the solver. What we lose, is the implicit relation that exists between y and v in Equations 4.28 and 4.29. The following chapter presents a summary of the theory behind solving equations such as 1.1 and number of examples of UDS that illustrate that this is a powerful tool in its own right and that the cost of the restrictions may be sonically rewarding.

Chapter 5

Unsampled Digital Synthesis and Its Applications

5.1 The Problem as it Stands

As with most applications of DSP, digital audio is to a large degree concerned with linear, time-invariant (LTI) systems. Nonlinearities are often dealt with by treating them as special perturbations to a system that is, for the most part, inherently linear. Examples of this abound. In physical modeling we see this in models of air-driven reeds and bow-string interactions. Nonlinear functions (bow/reed) are coupled to standard linear models of (for example) simplified clarinet bores or violin strings. In the realm of WDFs, we see the nonlinear root element as a special case among a conglomeration of linear elements.

In general, nonlinearities are difficult to deal with in digital signal processing. This is because most of the theory used to design and analyzed digital systems is based on assumptions that hold only in the case of linear and time invariant (LTI) systems. Digitizing a nonlinear or time-varying system can result in instability and/or unwanted aliasing.

An exemplar of this practice is the various digital implementations of the Moog ladder filter, which we shall address in some detail later. The history of the theory behind the digital Moog filter is one that starts with taking the bilinear

transform of the continuous-time transfer function of a filter stage (in its linear region) and deriving a discrete time model. Then, the nonlinear part of the filter is tacked on and the delay-free loop in the network is approximated with unit delay. Deep analysis is then applied to rectify the shortcomings in the behavior of the systems, shortcomings that result directly from using LTI tools to model a time-varying, nonlinear system [SS96].

In computer music we are usually in search of easily manipulable tools. When digitizing a system using the standard techniques (impulse-invariant, bi-linear transform, etc.) we often see a situation in which every time we adjust a system parameter, the digital system returned by the digitization process needs to be re-computed. In this practice, every case becomes a special case. Thus, it may be wise to develop an approach to computer synthesis in which we may interact with nonlinear and time-varying systems without having to worry about having to constantly redevelop the patchwork of tweaks that are necessary for keeping a nonlinear or time-varying system stable and predictable.

All of which is to say: we here claim that UDS is a useful approach to digital synthesis. Nonlinear systems are still difficult or perhaps impossible to analyze for the entirety of their parameter space, but at least with UDS, we can synthesize such systems in real-time and listen to them without having to write an engineering thesis. This is fitting with the experimental nature of computer music in general. Furthermore, if it is easier/faster to throw together a functioning dynamical network than it is to study it analytically; and, if that systems turns out not to make noise (i.e. it converges or diverges very rapidly) it doesn't matter that we didn't carry out lengthy mathematical analysis to come to this conclusion. If it doesn't work it doesn't work.

The inverse of this argument also applies. If some system 'sounds good', it doesn't matter (in the context of music making) that we may not be able to completely characterize it by analytic solutions.

Furthermore, sometimes sonification is the best way to quickly examine a system anyway. Recently (at the time of writing) physicists believe they detected gravitational waves [AAA$^+$16]. This result (if vindicated by repeated experiment)

is confirmation of Einstein's theory of gravity stretching and contracting the shape of space time. The physical evidence of this result is a expressed as a sound.

The aim here, of course, is to concoct raw material with which to make music, however, the possiblity of sonifying these things in order to study them (if only in a cursory manner) is a real bonus.

That being said, the following examples show some applications of UDS. The systems are presented as equations, and their character is analyzed via experimentation (i.e. running the simulation and various features are examined). An analytic solution to these systems may be impossible, and, furthermore, we must take the effects of the numerical solution method itself into account in order to completely understand the sonic properties of this stuff anyway. For these reasons (and because this is not a thesis in mathematics) mathematical analyses are eschewed.

5.2 Basic Example

We begin at a good place to start for any computer music system: a sinusoidal oscillator. In physics, simple harmonic motion is often modeled as the motion of a mass connected to a string.

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t} = -kx(t) \tag{5.1}$$

Here, m is the mass, and k is the spring constant. Since we will consistently be relating rates of change over time, to functions of time throughout, we will drop the time variable t when speaking of these dynamical models. We will use the conventional 'dot' notation to denote a derivative over time.

Equation 5.1 is a second order differential equation, but we may render it a first order system in the usual way:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -\tfrac{k}{m}x. \end{aligned} \tag{5.2}$$

Departing from the realm of physical proportions, we can control the frequency of the system by noting that the term $-k/m$ is essentially a gain on the rate

of change for each node in the system. When $-k/m = 1$, the system oscillates at a frequency of $1/2\pi$Hz. By eliminating this term and incorporating a general purpose time-scale gain ω, we may represent the system thus:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -\omega^2 x, \end{aligned} \tag{5.3}$$

where the frequency of the system is $f = \omega/2\pi = \sqrt{k/m}$. One caveat of synthesizing this system is that the values of y (position of the mass) are very, very small compared with the value of x (the velocity of the mass). We may remedy this by placing both x and y in units of position. This can be done by scaling the first equation by ω and the second by $1/\omega$:

$$\begin{aligned} \dot{x} &= \omega y \\ \dot{y} &= -\omega x. \end{aligned} \tag{5.4}$$

We can verify that Systems 5.3 and 5.4 are equivalent by noting that differentiating $\dot{x} = \omega y$ yields $\ddot{x} = \omega \dot{y} = -\omega^2 x$. Another way to say this is that we put the intermediate variabl y in terms of positional units rather than velocity since we are scaling it by the inverse of frequency (which also has time in the demoninator). To make up for the operation, we need to scale y back up when integrating to find the time varying values of x.

The outputs of this system in Equation 5.4 (which we are interested in hearing) are the changing values x and y, which we get by integrating $\dot{x}$ and $\dot{y}$.

Note that the frequency has nothing to do with the sampling rate we choose for our digital simulation. This is in stark contrast to our 'classic' digital oscillator where we determine the phase increment by dividing the intended frequency by the sampling rate. We also note that although the time variable t is now absent, these are still functions of time. The rate of solution (i.e. how long a period between solution points) is clocked by the audio rate, and (indeed) the sample width is a factor that effects the stability of the numerical solution (in the case of fourth order Runge-Kutta, which we choose here[1]). However, if we accept that our solver

[1]For an examination of various numerical solvers in the context of audio synthesis, the reader may consult [Yeh09].

is sufficiently accurate, we may discuss the models independently of the sample rate.

We are now in a position to solve our unsampled digital oscillator given initial conditions and a numerical solver. By understanding the initial values of x and y as the real and imaginary parts of a complex number Z, we can control the amplitude ($|Z|$) and initial phase ($\angle Z$) of the system. For example, by setting $x(0) = 0$ and $y(0) = -1$, x will take the form of a sine function with an amplitude of 1. The output of such a system is a pair of sinusoids in quadrature.

This implementation of simple harmonic motion is not new in digital audio. Mathews and Smith used oscillators in a similar form to design high-q bandpass filters [MS03]; and, this oscillator is analyzed quite thoroughly in Chapter 3 of [Bil09]. In that work it is presented as a basic building block for lumped networks.

In genereal then, this technique of declaring a system of inter-connected ODEs to a software application which will then compute the solution to each node on the network is what we mean when we say UDS. It should be noted, however, that the application of numerical solvers to ODEs is not new in digital audio. The technique has been used, for example, to model diodes in guitar distortion circuitry [YAS07] [MS09]. Furthermore, such a scheme is essentially a subset of the more general family of techniques known as finite difference methods, whose use in digital audio applications is well studied [Bil09].

5.2.1 Eliminating Unit Delay From the Representation

A very important detail here is that each equation in a UDS network must be approximated in an interleaved fashion. Many numerical integrators, such as Runge-Kutta, have multiple stages, and to ensure that the solutions to implicit equations 5.4 are not tainted by unit delay, these stages must be computed in lockstep with one another.

This is a very important point, and the claim that UDS eliminates the impossibility of zero delay in feedback loops hinges on the validity of the previous statement. Here the author may, for a moment, admit to a certain liberty being taken when he says ‘there is zero time delay in such and such a feedback loop’. In

reality there is only an approximation of what the solution to the equations *would* be if, in fact, there were no time delay. This is different than saying there is *some* delay. What is computed is not a tight feedback loop wherein *the delay is very nearly zero* – this being the premise of digital systems theory in the time domain. Rather, we approximate a tight feedback loop where the delay is *actually* zero.

To clarify this most important point in detail, let us investigate the Runge-Kutta solution to the quadrature oscillator. According to the Runge-Kutta method we may solve for y by choosing a temporal step size of $h > 0$ (the sampling period is an obvious choice for this value) and then setting

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \tag{5.5}$$

for $n = 0, 1, 2, ...,$ and

$$\begin{aligned} k_1 &= f(y_n) \\ k_2 &= f(y_n + \frac{h}{2}k_1) \\ k_3 &= f(y_n + \frac{h}{2}k_2) \\ k_4 &= f(y_n + hk_3). \end{aligned}$$

But, since values for x_{n+1} and y_{n+1} depend on y_n and x_n respectively, in order to avoid unit delay, we must interleave the stages of Runge-Kutta in which we compute k_n for x and for y. The written out solution for x_{n+1} and y_{n+1} would then be:

$$\begin{aligned} k_{y1} &= -\omega x_n \\ k_{x1} &= \omega y_n \\ k_{y2} &= -\omega(x_n + \frac{h}{2}k_{y1}) \\ k_{x2} &= \omega(y_n + \frac{h}{2}k_{x1}) \\ k_{y3} &= -\omega(x_n + \frac{h}{2}k_{y2}) \\ k_{x3} &= \omega(y_n + \frac{h}{2}k_{x2}) \\ k_{y4} &= -\omega(x_n + hk_{y3}) \end{aligned}$$

$$k_{x4} = \omega(y_n + hk_{x3})$$
$$y_{n+1} = y_n + \frac{1}{6}h(k_{y1} + 2k_{y2} + 2k_{y3} + k_{y4})$$
$$x_{n+1} = x_n + \frac{1}{6}h(k_{x1} + 2k_{x2} + 2k_{x3} + k_{x4})$$

This interleaving must be done, otherwise there is unit delay between the solution of either x_{n+1} and y_{n+1} or y_{n+1} and x_{n+1}, depending on which order we evaluate for x and y.

Now, it may appear that unit delay has not been eliminated, but merely reduced – that Runge-Kutta is simply a technique for increasing the sampling rate. But at each step of Runge-Kutta, we are taking into account the changes on the local approximations of k_{xn} and k_{yn} at every point. Thus we are not merely upping the sampling rate, but also doing a 4-point, dynamically adjusted interpolation. This interpolation *approximates the exact solution for this pair of implicit continuous-time equations.*

Then, to generalize this solution scheme to any number of nodes we may write:

$$x_i(n+1) = x_i(n) + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{5.6}$$
$$k_{i1} = v_i(n) \tag{5.7}$$
$$k_{i2} = v_i(n) + \frac{h}{2}k_{i1} \tag{5.8}$$
$$k_{i3} = v_i(n) + \frac{h}{2}k_{i2} \tag{5.9}$$
$$k_{i4} = v_i(n) + hk_{i3}. \tag{5.10}$$

In the simulations shown throughout, this scheme, Runge-Kutta (RK4), is used. It has been remarked that any robust Runge-Kutta solver should have a variable sampling rate in order to achieve better accuracy in regions of rapid change, [PTVF96]. However, in the case of real-time sound synthesis, it is unwise to rely on an algorithm that has a compute time that will vary according to unforeseeable conditions.

However, any number of numeric solvers may be used to solve the systems outlined below and to varying degrees of success (given the frequency and sampling

rate, of course). This is well trodden ground and the curious reader may be referred to [Gea71]. For issues of stability and accuracy (which are beyond the scope of the present work) the reader may be referred to [MQ08].

5.3 A Host of Examples

The example above is a trivial one. The lookup oscillator shown in Equation 4.11 is a much more computationally efficient means of synthesizing a sinusoid and the analytic solution to the system given in Equation 5.1 is very well known.[2] However, by treating the sinusoidal oscillator as an implicit network of ordinary differential equations, we open the door to sound synthesis routines that are unrealizable when feedback is delayed by the temporal width of at least one digital sample (z^{-1}).

5.3.1 Frequency Modulation

Frequency modulation (FM) is a tried and true computer audio technique that has been used to simulate the human voice and many other real musical instruments [Cho73]. Keeping with the digital lookup oscillator that we describe in Equation 4.11 we can instantiate a digital FM algorithm using phase modulation as is often done in practice [Puc07]. In this implementation, the output of the modulating oscillator is multiplied by the modulation depth parameter μ and is added to the phase of the accumulator. Ignoring initial phase, this changes our equation for the digital oscillator phase accumulator (Equation 4.11) to:

$$\begin{aligned} \delta &= \tfrac{\omega+\mu m(n))}{F_s} \\ x(n) &= x(nt-1)+\delta \\ y(n) &= \sin\left(2\pi x(n)\right), \end{aligned} \tag{5.11}$$

where $m(n)$ is the output of the modulating oscillator. If we take the output of $y(n)$ and use it to modulate the oscillator produces the output $m(n)$, the two oscillators are modulating each other. If such a network is formed with lookup oscillators

[2]Lookup oscillators are also very handy because they can be generalized for any wave shape, thus provide a robust palette for creating arbitrary periodic wave-forms.

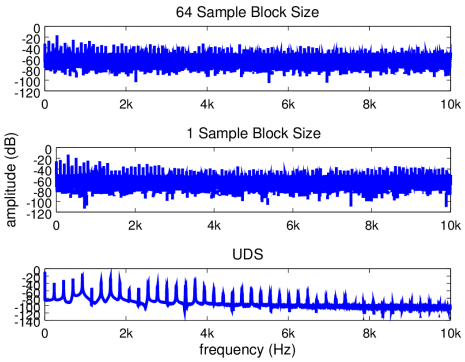


Figure 5.1: Magnitude spectra for various implementations of a reciprocal FM network.

such as the one given in Equation 5.11, any feedback in the network will entail at least one audio sample worth of time delay. This severely distorts the output of the system, essentially adding broad band noise to the desired signal as is evident in Figure 5.1.

In order to make a more accurate digital simulation of a reciprocal FM network (i.e. one with zero delay in the feedback loop), we may modify our UDS oscillator given in Equation 5.4

$$\begin{aligned} \dot{y}_i &= (\omega_i + \lambda_i)x_i \\ \dot{x}_i &= -(\omega_i + \lambda_i)y_i. \end{aligned} \tag{5.12}$$

Here, there are k oscillators that modulate one another (given by x_i and y_i), and λ_i is some linear combination of each other's outputs:

$$\lambda_i = \alpha_{i1}x_1 + \alpha_{i2}x_2 \ldots \alpha_{ij}x_k. \tag{5.13}$$

The coefficients α_{ij} define the depth of modulation on oscillator i by the current state of oscillator j. Note that these oscillators modulate themselves when α_{ij} is non-zero for $i = j$.

In Figure 5.1 spectra of the output of 2 lookup oscillators in a reciprocal FM network and the same network with a UDS implementation are shown. Reciprocal FM with the lookup oscillator implementation creates unwanted broadband noise. The situation is somewhat helped by reducing the block size, but not entirely. In

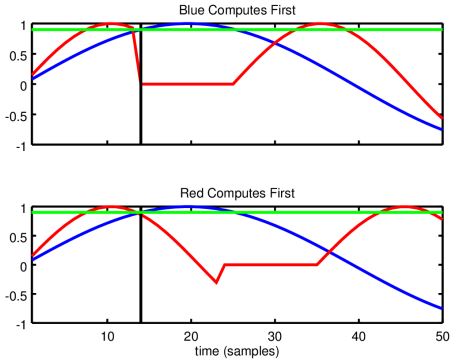


Figure 5.2: Two plots demonstrating the order of operations problem for digital lookup oscillator sync.

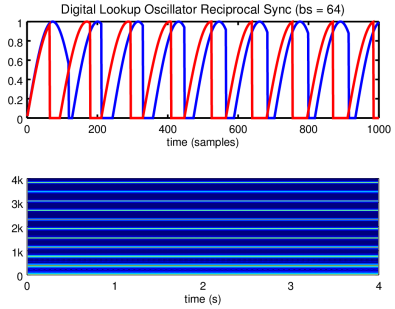
the UDS implementation, on the other hand, broad band noise is absent. In these 3 examples, the frequencies were identical and the modulation depth was analogous. It is also interesting to note that the spectral envelope is quite different in all 3 examples.

5.3.2 Reciprocal Sync

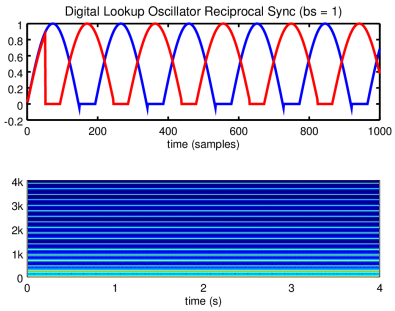
Oscillator 'sync' is a classic technique in analog synthesizer patches. The output of a 'master' oscillator is compared to a threshold value. If the threshold (θ) is crossed, the phase of a 'slave' oscillator is reset. This nonlinear technique is a simple way of creating complex timbres out of two purely sinusoidal wave-forms.[3]

If the routine is computed using lookup oscillators, the order of operations is important. Depending on whether or not the output of the master or slave oscillator is computed first, the output wave-forms will be quite different. This distortion is amplified by the magnitude of the block size (which we denote b_s). By way of illustration, consider Figure 5.2. Here the blue waveform shows the master oscillator, the red the slave ($f_m = 8$ Hz, $f_s = 15$ Hz, $F_c = 100$ Hz, $b_s = 10$,). In the top plot, the master oscillator is computed first, and goes to 0 at the correct time. However, if we calculate the output of the slave first, it will not go to

[3]Oscillator sync is often done with other, harmonically rich wave-forms in analog synthesizers; see for example http://www.keithmcmillen.com/blog/simple-synthesis-part-7-oscillator-sync/.



(a) Time series and spectrogram for digital lookup oscillator in reciprocal sync regime.



(b) The same simulation parameters, but with block size equal to 1.

Figure 5.3: Both time series and corresponding spectrograms are from two oscillators (950 Hz and 900 Hz) in a sync regime. Not only the spectral shape, but also the fundamental frequency is affected by block size. Here $F_s = 48$ kHz.

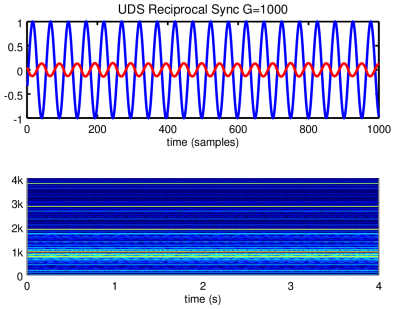
0 until an entire block of samples in the future: it won't 'know' until then whether or not its master crossed θ.

We may couple two unsampled digital oscillators together through a sync rule and achieve very different results. As with the digital lookup oscillators, we define a threshold θ that, when crossed by the master (x_m), the slave (given by x_s and y_s) must reset its phase. We make a rule that if $x_m > \theta$, then substitute

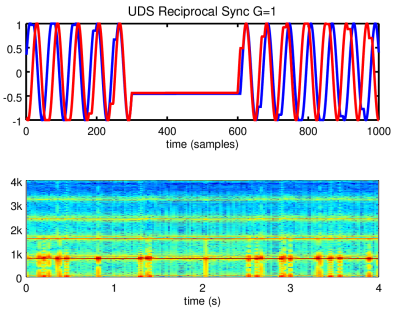
$$\begin{aligned} \dot{x_s} &= (0 - x_s)g \\ \dot{y_s} &= (-1 - y_s)g \end{aligned} \tag{5.14}$$

for Equation 5.4 in the slave oscillator. Here, g is a gain factor that speeds this return to the 'zero' phase ($x = 0$, $y = -1$). We are now free to couple two or more oscillators using this sync rule in a tight feedback loop so that they are both masters and slaves to each other in a time-accurate manner.

Figure 5.4 shows the results of the unsampled digital simulation of a reciprocal sync regime using the same parameters for θ and the same frequencies as those shown in Figure 5.3. By contrast, if a large value is given for g, we see that the effect is essentially to reduce the amplitude and shift the phase of one of the oscillators. Some harmonic distortion is also present due to the wave-shaping effect



(a) A pair of UDS oscillators (sine portion only) in a reciprocal sync regime, $g = 1000$.



(b) The same but with a much smaller gain parameter, $g = 1$.

Figure 5.4: Unsampled digital synthesis for 950 and 900Hz oscillators in a reciprocal sync regime.

of the sync rule. When a small value is chosen for g, chaotic behavior emerges. By adjusting the values of g and θ dynamically and in real time, a vast array of unique and unpredictable sounds are possible.

Another evident difference between the UDS method and the lookup oscillator method, is that in the case of lookup oscillator reciprocal sync has the effect of reducing the fundamental frequency of the oscillators' output. This is due to the nature of the sync rule, which forces the slave to effective maintain an output of 0 when it is in the sync regime.[4] This literally lengthens the period of the wave-form. It is interesting the dynamics of the UDS implementation do not have this effect.

5.3.3 The Moog Ladder Filter

As an example of a musically interesting UDS model that is not based on harmonic motion, we consider the 'ladder filter' invented by Robert Moog in 1965 [Moo65]. This filter has been studied extensively and many digital simulations have been demonstrated (e.g. [SS96][VH06]).

Most digitizations that have been done use the bilinear transform or some

[4]Oscillator sync rules come in a wide variety of flavors, so this feature is somewhat unique to the particular algorithm outline here.

similar procedure to map the linear portion of the analog transfer function to the unit circle. Since there is feedback in the circuit, a standard digitization of the filter includes unit sample feedback – the famous z^{-1}. This is problematic because the unit sample delay in the feedback couples the gain parameter g (which controls the cutoff frequency of the filter) to the resonance parameter r (which controls the bandwidth of the filter). This causes additional phase shifting so that effect of the resonance parameter varies with frequency and vice versa [Huo04]. Many adjustments have been suggested to abnegate this artifact of digitization [Dal12]. Here, we propose an unsampled digital which greatly reduces this effect.

The Moog filter is a four-pole network, wherein the voltage across each stage is defined:

$$\dot{V_i} = \omega(\tanh(V_{i-1}) - \tanh(V_i)), \tag{5.15}$$

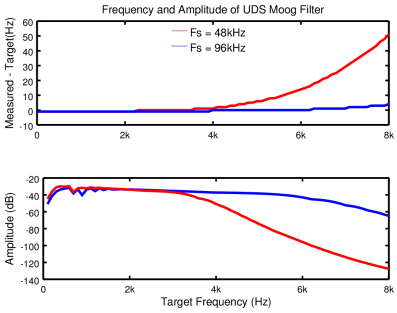
where ω is the gain that governs the cutoff frequency of the filter, $\dot{V_i}$ is the change in voltage across stage i, V_i is the present voltage across stage i, and V_{i-1} is the present voltage across the previous stage for $i = 1, 2, 3$. Ideally, the actual cutoff frequency has a ratio of $f_c = \omega/2\pi$ for all frequencies.

Since each stage of the filter shifts the input signal phase by $pi/4$ radians, the output of the filter is inverted and fed back into the input to create resonance. So, for $i = 0$ we have:

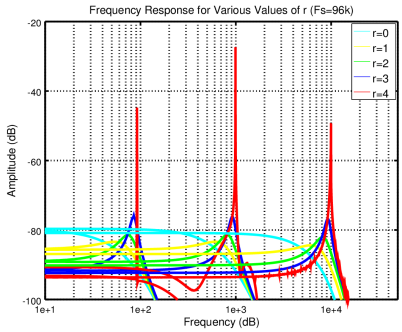
$$\dot{V_0} = \omega(\tanh(V_{in} - rV3) - \tanh(V_0)). \tag{5.16}$$

When $r = 4$ (1 for each stage), the filter saturates will oscillate on its own.

Some results for the performance of the UDS implementation of this filter are shown in Figure 5.5. We see in Figure 5.5a that doubling the sample rate improves the quality of the filter (ideally the difference between target frequency and measured frequency should be equal, as should the peak amplitude at every frequency). In Figure 5.5b we shows that the effect of the resonance parameter r on the cutoff frequency is negligible.



(a) Shown is the difference between target frequency and highest peak in the spectrum as well as amplitude for the UDS Moog filter.



(b) To illustrate the independence of r and f_c, we show the frequency response for various values over a wide frequency range.

Figure 5.5: Plots illustrating the character of the UDS implementation of the Moog filter. Results compare favorable to those shown in [Huo04] and [Dal12].

5.3.4 A Bowed Oscillator

In addition to arranging simple harmonic oscillators in delay-free feedback networks, we may use UDS as a means of introducing physically motivated nonlinearities. As an example, consider a harmonic oscillator driven by friction – such as a point on a violin string.

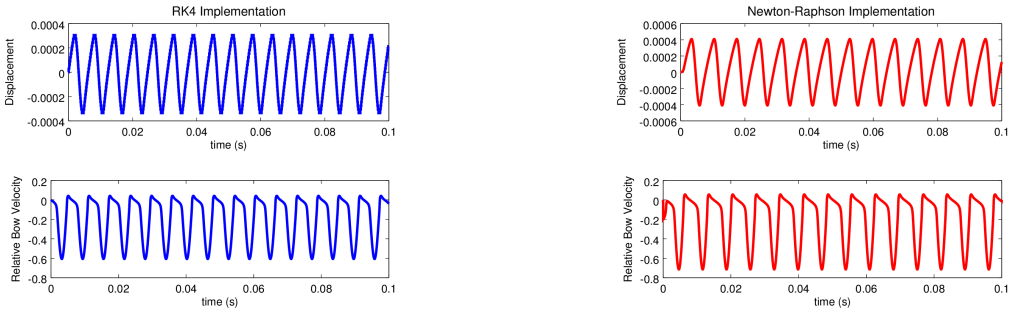
The physics of bow-string interaction is well understood [MW79] [CA84] [Ser04]. The effect of the bow on the point of bowing is usually modeled through a nonlinear function which is coupled to the point of bowing (here given as a single oscillator). The motion of this oscillator is

$$\ddot{u} = -\omega^2 u - F\phi(v - v_b). \tag{5.17}$$

In this model u is displacement of the oscillator, $f = 2\pi\omega$ is the fundamental frequency, F is the force of the bow, v is the oscillator's velocity, v_b is the velocity of the bow, and ϕ is a nonlinear friction model such as:

$$\phi(v_{rel}) = \sqrt{2a} v_{rel} e^{-2av_{rel}^2 + 1/2}, \tag{5.18}$$

where a is a coefficient of friction and $v_{rel} = v - v_b$.



(a) $F/m = 500$ (b) $F = .5$

Figure 5.6: The bowed oscillator model given here (solved with RK4) and that given in [Bil09]. In both implementations, oscillator frequency $f = 200$, $v_b = .2$ and $a = 100$.

As is the case with the second order spring-mass system given in Equation 5.1, we can re-write Equation 5.17 as a pair of ODEs:

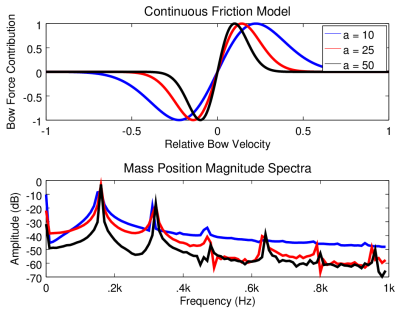
$$\begin{aligned} \dot{y} &= -\omega^2 x - F_b \Phi(y - v_b) \\ \dot{x} &= \omega y. \end{aligned} \tag{5.19}$$

A slightly different form of this model is discussed in Chapter 3 of [Bil09]. In that model, bow force over mass, rather than just bow force, is a parameter. Another difference is that in [Bil09] Newton-Raphson is employed to solve for the nonlinear contribution ϕ and the linear part is solved using Backward Euler. This requires, of course, that ϕ be differentiable and that the derivative be determined. Furthermore, Newton-Raphson is an iterative solver so it may not be appropriate for real-time implementation due to convergence issues. These two implementations are compared in Figure 5.6. It comes as no surprise that they are very similar.

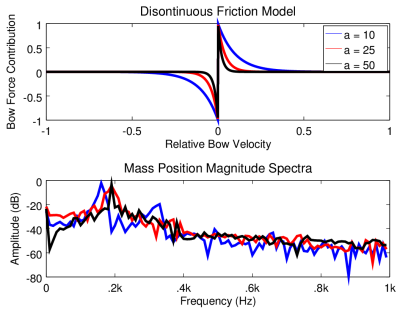
Since RK4 puts us in a position to solve the system without having to differentiate ϕ, we are free to experiment with friction models that are discontinuous. For example, we may study the discontinuous friction model given by

$$\phi(v_{rel}) = \text{sign}(v_{rel}) e^{-a|v_{rel}|}, \tag{5.20}$$

Being able to swap nonlinearities is very good. However, RK4 may not be up to

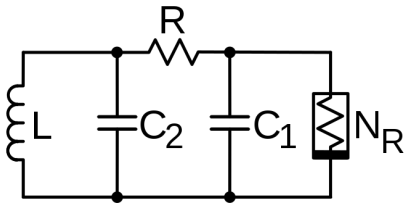


(a) Differentiable friction model.



(b) Discontinuous friction model.

Figure 5.7: RK4 solving Equations 5.19 using (a) the continuous friction model in Equation 5.18 and (b) Equation 5.20.



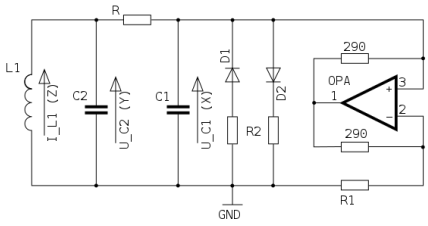


Figure 5.8: A theoretical (a) and real circuit diagram (b) showing Chua's circuit.

the job of accurately integrating the sharp discontinuity—as Figure 5.7 indicates.

In order to get a more symetrical output of both x and y values, we may depart from the realm of physics and employ the same scaling trick that we introduced in Equation 5.4. This leaves the folowing system:

$$\begin{aligned} \dot{y} &= -\omega x - F\phi(v - v_b)/\omega \\ \dot{x} &= \omega y. \end{aligned} \tag{5.21}$$

Not that the contribution from the bow model also needs to be scaled by $1/\omega$.

5.3.5 Developing a Nonlinear Noisebox

Chua's circuit is a very well studied chaotic oscillator that can be realized through analog circuitry [JLK+84] [Mad93]. One such realization is shown

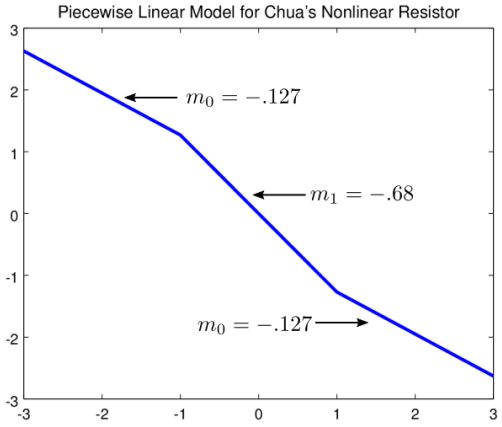


Figure 5.9: Piecewise linear $V - I$ curve in the canonical Chua model.

in Figure 5.8 (b).[5] The abstracted circuit (with a theoretical nonlinear resistive component) is illustrated in Figure 5.8 (a).[6]

The nonlinearity in the circuit is often model as a piecewise linear set of negatively sloping lines. The canonical equation for this (shown in Figure 5.9 is:

$$f(x) = m_1 x + .5(m_0 - m_1)(|x+1| - |x-1|). \tag{5.22}$$

The circuit itself is often modeled:

$$\begin{aligned} \dot{x} &= \alpha(y - x - f(x)) \\ \dot{y} &= x - y - z \\ \dot{z} &= -\beta z, \end{aligned} \tag{5.23}$$

Where x, y, and z are the voltage across C1, voltage across C2, and current across the inductor L (respectively) in Figure 5.8. The parameters α and β are determined by the values of the circuit components and are obtained through analyzing the circuit. When $\beta = 14.286$, $\alpha < 10$, $m_0 = -1.27$, and $m_1 = -.68$, the circuit will exhibit chaotic behavior and show a double scroll pattern [CKEI92].

Rodet [Rod93] proposes a clarinet-like model based on a modified Chua's circuit. In this model, the inductor and capacitors are removed and a voltage

[5]This is a public domain image: Public Domain, https://commons.wikimedia.org/w/index.php?curid=4320592.

[6]This is a wikicommons image: By Chetvorno - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=31049859.

source is added. From this circuit, a time-delayed mapping is developed so that the solution looks like a DWG:

$$\Phi(n) = \gamma(\Phi((n - 2T))), \tag{5.24}$$

where Φ is a sum of left and right going waves, γ is the Chua nonlinearity, and T is a time delay (in seconds).[7]

The resulting waveform from this model is periodic and has a rich, harmonic spectrum, presumably similar to a clarinet reed. By varying the values for $m0$ and $m1$ (which presumably correspond to the canonical piecewise linear $V - I$ curve given in Equation 5.22) the peaks in the spectrum exhibit more or less frequency correlated noise resulting from the chaotic behavior of the model.

We may borrow from this model, but modify it. We can not implement this model using UDS as we don't have the machinery to compute the time delay.[8]

As a side note, if Equation 5.24 were a second order system, we could very well do this. Since the first order Taylor approximation of

$$v(t) = z(t - \tau) \tag{5.25}$$

is

$$v(t) = z(t) - \tau\dot{z}(t), \tag{5.26}$$

if we are in a position to compute the solution to $\dot{z}$ and $\ddot{z}$, we would have the machinery to compute the time delay τ.[9]

However, we may indeed couple a nonlinearity to a UDS oscillator in a similar manner to how we coupled our bow model:

$$\begin{aligned} \dot{x} &= \omega(y + \phi(x)) \\ \dot{y} &= -\omega(x), \end{aligned} \tag{5.27}$$

[7]Although Rodet is here able to boil down a dynamical system into a nonlinear difference equation, he does include the oft repeated comment when dealing with dynamical systems '[t]he study of this dynamical system is difficult to analyze . . . '.

[8]Of course, we note that this model is very much a DWG with a nonlinearity coupled to it. Thus, in this formulation it wouldn't make sense to 'translate' the model directly into a UDS formulation anyway.

[9]This is done in practice. See for example [RA] in which two nonlinear pendulums driven by Van Der Pol torque are synchronized by a time-delayed, second order coupling mechanism in exactly this way.

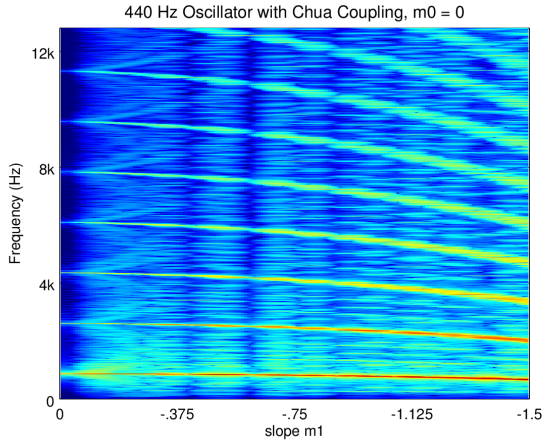


Figure 5.10: Spectrogram of model given in Equation 5.27.

where $\phi(x)$ is given by the piecewise linear map given in Equation 5.22. By setting $m0 = 0$ (hard clip) and adjusting the value for $m1$ we see a flattening effect and nonlinear harmonic distortion (Figure 5.10).

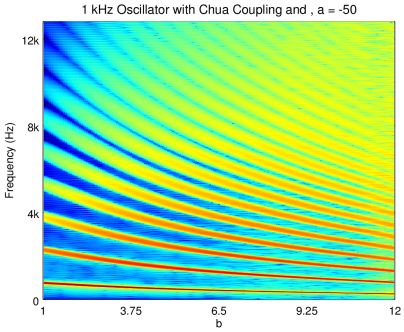
We do not see, however, the chaotic structures reported in [Rod93]. Switching to a cubic $V - I$ curve:

$$g(x) = ax^3 + bx, \tag{5.28}$$

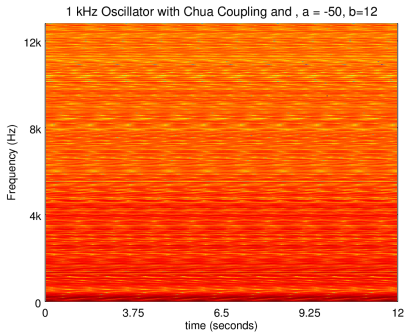
as has been detailed in the vast literature on Chua's circuits (see for example [Zho94] and is hinted at in [Rod93], we can push this nonlinear oscillator into a heavily noisy regions (Figure 5.11).

Again, we see the flattening effect and strengthening of upper partials as the parameter b increases. But as it reaches a certain value, chaotic noise sets in. Apart from the flattening, this begins to resemble Rodet's model in that there is a harmonic spectrum colored by a noise component. It is difficult to tell from Figure 5.11b, but in this regime, there are slow periodic rhythms to the sound. The nature of these rhythms that emerge in the noisy regimes is unpredictable and depend on all the parameters, including the frequency $\omega/2\pi$ which we take to be the 'natural' frequency of the oscillator. Unfortunately, this model will diverge when pushed to far into these regions.

In order to provide more nuanced control, we can add more nonlinearitiesinto the equations. For example, if we compute the output of the following



(a) Spectrogram of model given in Equation 5.27, but with the $V - I$ curve Equation 5.28.



(b) The same, but in the noisy regime.

Figure 5.11: Outputs of our nonlinear oscillator with Chua-inspired $V - I$ curve.

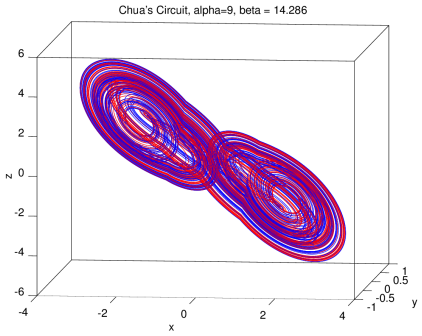
system:

$$\begin{aligned} \dot{x} &= \omega(y + \alpha(g(y) - \beta g(x))) \\ \dot{y} &= -\omega x \end{aligned} \tag{5.29}$$

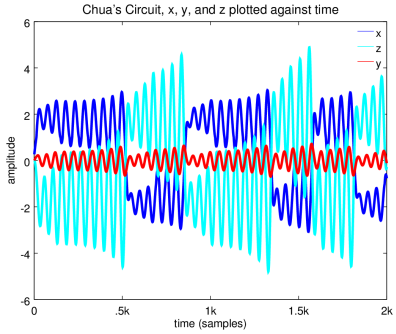
we can produce lovely chaotic noises by adjusting α, β and a and b from Equation 5.28.

The reason for marching through this particular, and rather meandering, process is to illustrate an important point. The dynamical system shown in 5.29 is now somewhat devoid of any physical interpretation. It is a nonlinear oscillator, and contains elements of the Chua circuit, but it is now something entirely different. However, it can be used to make extraordinary sounds.

This suggests that given the programming paradigm shown here, one can construct audible output out of a small subset of atomic elements. Namely, integrators, gains, and nonlinear functions. Of course, there are at least as many nonlinear functions as there are non-elephant animals. But, by drawing ideas from previous work on known and well studied systems, good things may happen.



(a) A Chua circuit given two different initial values.



(b) The Chua circuit in the time domain.

Figure 5.12: Two views of the Chua circuit.

5.3.6 Direct Audition of Chaos

As noted above, the Chua circuit (Equations 5.23 and 5.22) is chaotic in certain regions of its parameter space. In its canonical form, it maintains a 'double-scroll' orbiting pattern. Figure 5.12a shows the output for each variable given two different initial values. Clearly the system exhibits chaos.

We may listen to the output of the circuit at any point. To accomplish this, we first note the locations of the attractors. To do this analytically is difficult, but from inspecting the waveform we see that y is centered about 0, and that x and z jump from oscillating about somewhere in the vicinity of ± 1.8. For our purposes, we wish to have an amplitude that is within ± 1. Thus we can simply divide x and z by 5 and leave y alone.

To control the frequency of the system, we simply use our time gain trick and multiply each of the equations in 5.23 by a parameter ω. In order to determine the character of effect of the time gain factor ω on the system, we can simply measure the peak frequency for various request frequencies. The results for x are shown in Figure 5.13.[10]

We may take α to be a throttle controlling how 'chaotic' the model is. The effects of this is shown in Figure 5.14. There is definitely some flattening of the

[10]The results for z are almost identical, but slightly larger. The y direction corresponds to the movement between the 2 orbits in the system, but is roughly the same frequency.

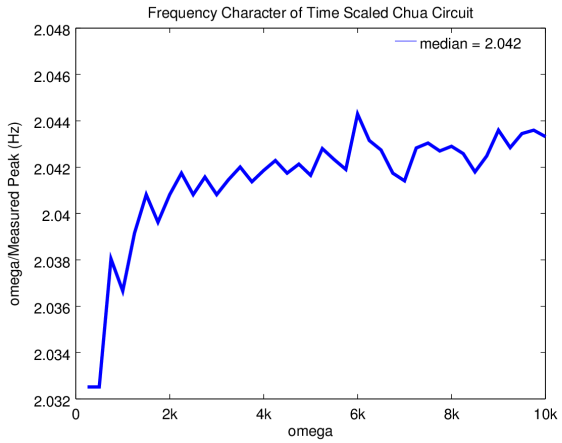


Figure 5.13: Squelching chaos in a Chua circuit to measure frequency: $\alpha = 6.99$, $\beta = 14.2896$, $m0 = -1.27$, and $m1 = -.68$.

desired frequency (approximately one half of ω), but we do have a nice control over the harmonic content of the spectrum. In some examples from the literature, $\alpha = 10$ is given as a reasonable choice for setting the oscillator into a chaotic regime [CKEI92]. In the experiments presented here, this value varies somewhat with ω, but in general values between 6.8 and 8.5 are good. Below the minimum, the system dies, and beyond the maximum it will explode.

Some concluding remarks about the Chua circuit follow. First of all, this is a good example of control afforded by UDS that is not available in WDFs. This circuit, with its simple structure and lone nonlinearity is a good candidate for simulation with WDFs (a WDF tree is only slightly more complicated that the dynamical model given in Equation 5.23). However, that paradigm does not provide the machinery for time scaling up (controlling frequency with ω). Also, since each element (apart from nonlinear root elements) in WDFs are LTI filters, rapid variation of the parameters may not be possible. In this implementation, we can throttle α (which is the product of the inverse of the capacitance of C1 and the conductance resistor R in Figure 5.8) at audio rates with impunity (e.g. with another oscillator).

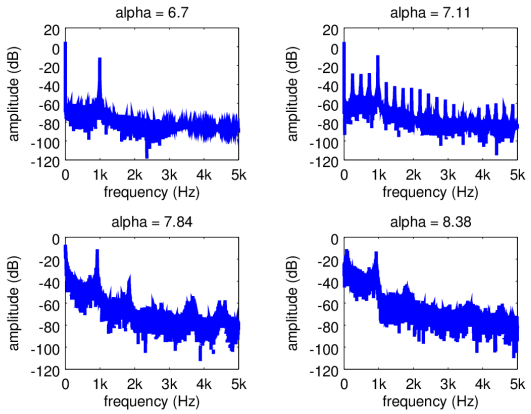


Figure 5.14: The effect of increasing chaos with α: $\omega = 2000$, $\beta = 14.2896$, $m0 = -1.27$, and $m1 = -.68$.

5.3.7 Listening to Lorenz

The Lorenz attractor [Lor63] is another example of an extremely well studied chaotic dynamical system [GW79] [Spa12]. The Lorenz equations are:

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z. \end{aligned} \tag{5.30}$$

The usual initial values used to through this system into a region of chaotic oscillation is $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. This produces the classic butterfly wing shapes that is now so emblematic of chaos theory.

We note that there are circle-ish orbits around the stationary points:

$$(\pm\sqrt{\beta(\rho-1)}, \pm\sqrt{\beta(\rho-1)}, \rho - 1).$$

The origin is also a stationary point when $\rho < 1$.

As was the case with the Chua circuit, the evolution of the Lorenz system can be controlled by multiplying each stage by a time scalar ω. By also changing the parameter β, we can control the degree to which the system tends to move between the two attractors. So, again, we have an oscillator that we can listen to as we throttle it in and out of chaotic regions. Also, again, we are obliged to scale the output to a reasonable range. Here I chose $s_{atten} = 1/(3\sqrt{\beta(\rho-1)})$.

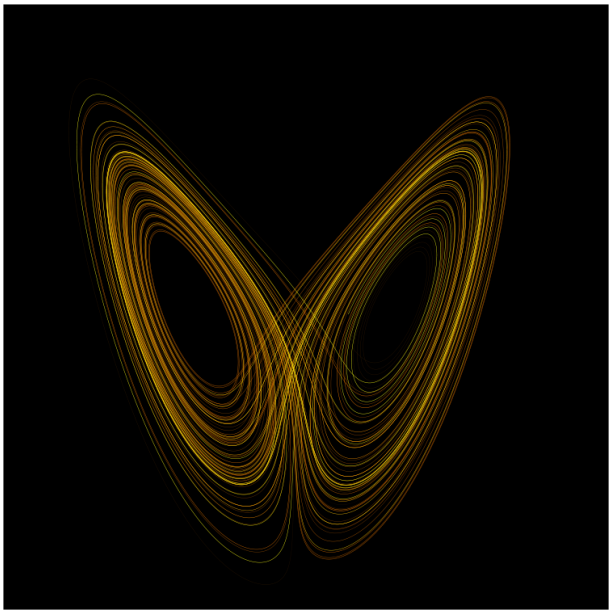


Figure 5.15: Ye olde Lorenz system.

The 'chaos-ness' of the system (in a UDS implementation, at any rate) depends not only on β, but also on ω (all other parameters held the same). But, if we choose $\beta = .5$, we can swing ω from 50-1000 without crossing over into the chaotic regime.

In Figure 5.16 we see 5 scenarios for $\beta = .1$, $\beta = .2, \ldots, \beta = .5$. This is for the x value in the system.

The first thing to note is that when $\beta = .1$, sometimes the second harmonic is of greater magnitude than is the fundamental frequency. This is evinced by the outliers in Figure 5.16a. Obviously there is a strong DC component (the orbit is around one of the stationary points or the other, not around the origin). It is also the case that small changes in β will alter the pitch drastically, although in this range in parameter space, the spectral envelope is quite similar.

We can also look at the effect of β itself in the audible range. As noted, pushing β from very near to 0 to the usual value of 8/3 increases the rapidity with which the x and y values in the system move between the two orbits. The erratic nature of this motion is literally noisiness. A spectrogram illustrates this in Figure 5.17.

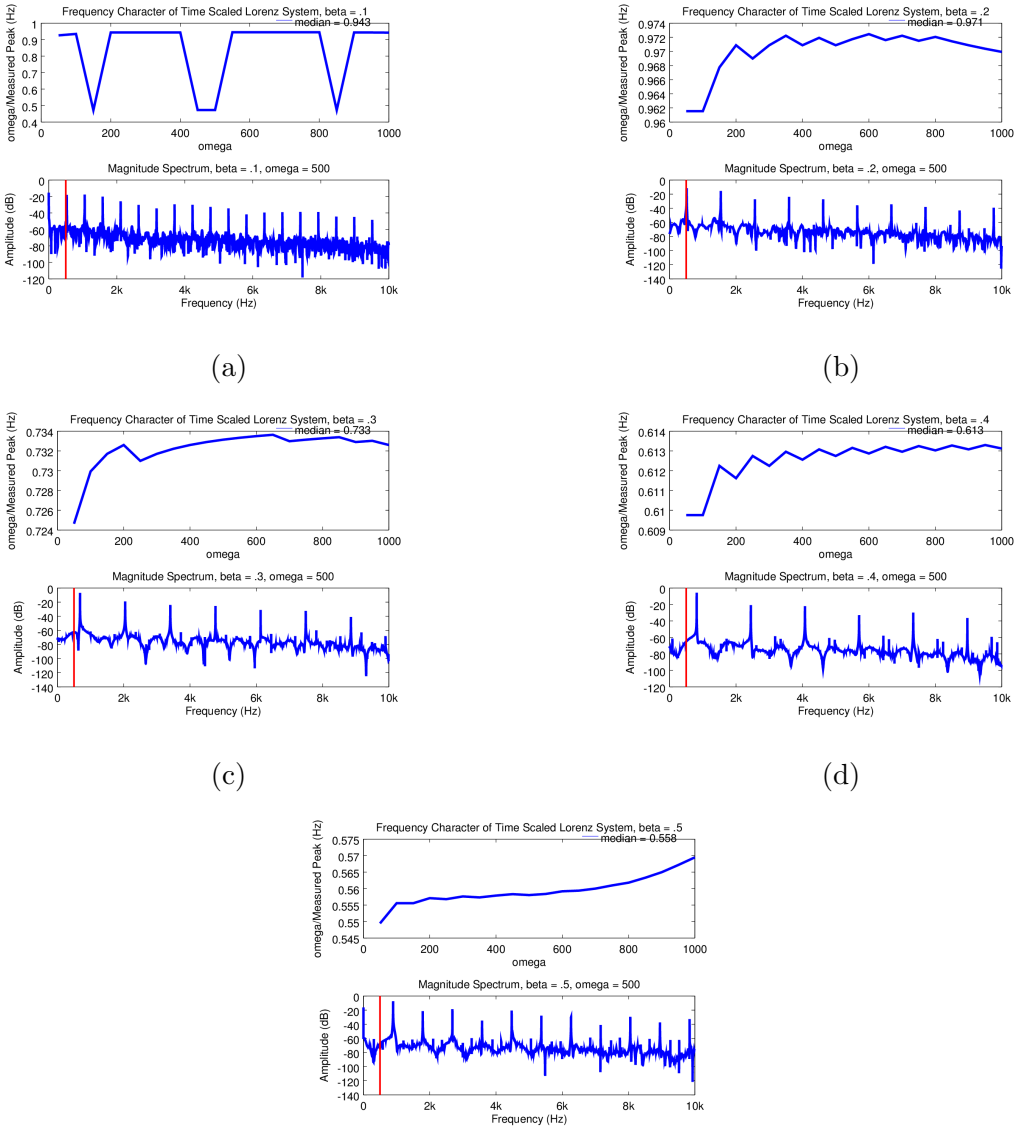


Figure 5.16: The frequency characteristics of the Lorenz system for various values of β and ω. Spectra for $\omega = 500$ are shown for each value of β. The red line in the spectra indicates 500Hz.

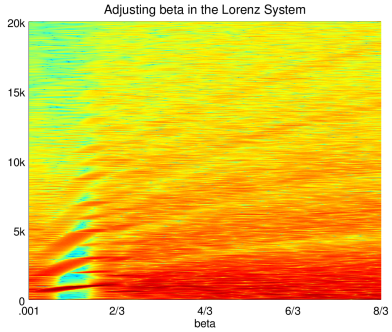


Figure 5.17: The spectral evolution of x in the Lorenz system as β is increased over time. $\omega = 500$, $\rho = 28$, $\sigma = 10$.

5.3.8 Note: Chaotic Amplitude Envelopes and LFOs

Chaotic systems for the use of automatic parameter control in computer music has been and continues to be an active research topic. Often this takes the form of organizing compositional parameters [Spa15]. But the use of chaotic oscillators in the analog synth-style context of LFOs and amplitude envelope generators is not well studied. Again, because we have nuanced, real-time control over frequency and the 'chaos-ness' of the output, we may say this is a special application of dynamical systems in music that this programming paradigm affords and which others do not.

To use either the Lorenz system or the Chua circuit as an LFO is trivial, simply set the desired, low frequency, and scale appropriately. In the case of an envelope generator, there is more work to do. The primary challenge we face is that both oscillators wiggle between orbits that are on either side of 0. This poses a problem because amplitude envelopes are only positive. Thus, if we wish to throttle the chaos lever during an envelopes lifetime, we would either have to start a non-zero initial value (for the envelope) or else add the output to a traditional envelope generator such as an ADSR (attack, sustain, decay, release) envelope – a kind of carrier, modulator situation.

In the case of the Lorenz system (5.30, the parameter ρ gives us a convenient means for accomplishing this. Recall that when $\rho < 1$ a Hopf bifurcation occurs and the state of all the variables go to 0 [Spa12]. Thus, if we use the conventional

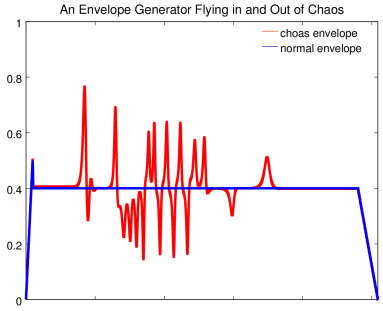


Figure 5.18: A conventional ADSR envelope super imposed on one that is added to the scaled output of a Lorenz system. The end ramps move the system states to 0 through the parameter ρ.

envelope generator's attack ramp to move ρ from 1 to 28 and the release ramp to do the opposite, we can safely move in and out of chaotic oscillations during the sustain period whilst ensuring a start from and return to 0 at the ends of the envelope. This is illustrated in Figure 5.18.

5.4 Concluding Remarks

Dynamical systems are tricky business. They are unpredictable, often unstable, and can be infuriatingly counter-intuitive. However, there are a wide variety of applications for using dynamical systems in computer music. It is good and right to experiment with using these tools in computer music, even if they are seldom within the total control of the musician that wields them.

The technique of Unsampled Digital Synthesis is an approach that makes this possible. Because it removes the audio sample as the basic unit of time, it also makes possible enacting synthesis networks that have zero delay feedback loops.

In order to experiment these synthesis routines and to adjust their parameters in real-time, a generally 'plug-in-able' but specifically 'UDS-ish' audio scheduler called timelab was written in C. It allows for the creation of UDS networks and the computation of their output. It is also written with an eye to maximum compatibility and diversity of employment.

Future work on timelab includes the continuing development of code itself, ensuring that it is as robust as it hopes to be, and developing higher level tools to speed up the development of UDS routines.

Furthermore, all the systems shown here are based on differential equations. There are many systems, however, that we may wish to model that are implicit and nonlinear, but are not expressible through ODEs alone. A family of such examples are the Koren models for vacuum tubes [Kor96]. These models have been digitized using techniques that involve unit delay such as WDFs [PK10] and FDTD schemes [MS10]. This is a prime candidate for implementation through UDS, but since the model contains implicit relations that are not expressed through differential equations, and since timelab does not currently have the machinery to compute implicit equations that are not differential, we can not currently implement it.

There are many models left to explore. Hopefully this work can continue and will be taken up by others. What is most desirable is an editable, real-time patching environment for designing topologies of gains, integrators, sums, and nonlinear functions. This would greatly speed the development cycle for experimenting with dynamical models and provide a means of doing so for musicians who are not C programmers.

This chapter contains material that originally appears in 'David Medine. Dynamical systems for audio synthesis: Embracing delay free loops. *Applied Sciences*, 2016'.

Chapter 6

Sound Examples and Discussion

In this chapter we present a number of soundfiles illustrating some of the existing sounds that have been generated using UDS via timelab. A description of each sound example and why it sounds the way it does accompanies the audio examples. The manner of this discussion is very much modeled on the PhD thesis of Richard Boulanger [Bou86]. A notable difference between the present study and the previously mentioned work of Boulanger is that there, the author contextualizes the meaning of his sounds in the language of 20th Century Western compositional theory and works. Here we choose a context for musical discussion that draws less from such traditions.

It would be possible (although exceedingly difficult) to 'explain' the behavior discussed below with mathematical analysis of the systems. That is to say, one could systematically explore the entire space of parameter combinations for each system and compute flow diagrams and points where Hopf bifurcations occur. However, in addition to being cumbersome to present, such an analysis would not be of much practical use as a means of describing a set of tools intended for musical applications. For one thing, many of the examples below (being the output of dynamical systems) demonstrate unpredictable and chaotic behavior. This can be considered a good feature or as a hurdle.

Sometimes unpredictability is not a good thing. This is particularly true in the case of physical modeling and VA applications. In those cases the user wants stability and full control over the behavior of the system. However, this author is

far more interested in the unstable and the unpredictable (which is exciting) than the development of well behaved computer emulations of mechanical and analog systems (which is commercially viable).

Furthermore, many mechanical instruments are in fact highly unstable, nonlinear dynamical systems. It is the expertise of the player of a violin that makes the violin sound beautiful and clean. The system itself is anything but—a fact to which any parent of a child learning the violin can attest. However, the very richness and potential beauty of such instruments is related to the instability of the system. The following statement is certainly tangential from most of this thesis, but it is the opinion of some, the present author included, that the very act of controlling an unstable musical instrument (which is primarily an act of listening) is in and of itself a musically interesting process–indeed a process that underlies (to some extent) any musical endeavor.

Thus the aim of this chapter is to give a whiff of what it is like to deal with these systems and what kinds of sounds and behaviors they tend to produce.

6.1 Reciprocal FM Examples

In order to represent the nature of the UDS algorithms (or 'patches') that the sound examples demonstrate, we shall resort to signal flow diagrams such as the one in Figure 1.1. Such diagrams are useful for explaining the nature of systems. Furthermore, this a convenient visual form of representation for the creation and defining of UDS routines. Indeed it is a natural extension of the work currently presented here to develop a software layer for organizing the basic atomic elements of a UDS routine (gains, nonlinear functions, and integrators) in such a representations that can then be used to automatically generate the code which represents the same routine in the parlance of timelab. This is left to the general heading of 'future work'.

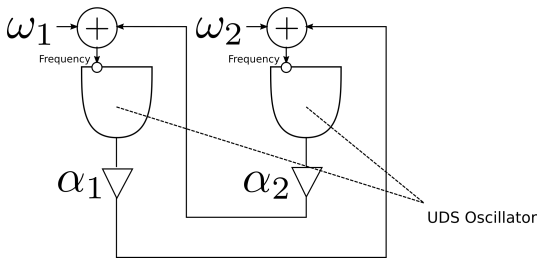


Figure 6.1: Signal flow diagram for a two oscillator reciprocal FM network.

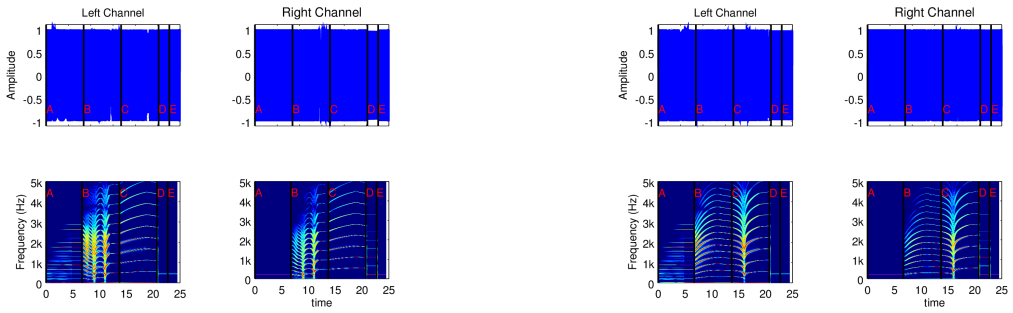
6.1.1 Harmonic Reciprocal FM

As a first example, we examine a reciprocal FM network involving two oscillators. The signal flow diagram for this routine is given in Figure 6.1. The mathematical expression of this system for the general case of n oscillators was given in Equations 5.12 and 5.13. In the case of 2 oscillators, this is:

$$\begin{aligned} \dot{y}_1 &= (\omega_1 + \alpha_{11}x_1 + \alpha_{12}x_2)x_1 \\ \dot{x}_1 &= -(\omega_1 + \alpha_{11}x_1 + \alpha_{12}x_2)y_1 \\ \dot{y}_2 &= (\omega_2 + \alpha_{11}x_1 + \alpha_{12}x_2)x_2 \\ \dot{x}_2 &= -(\omega_2 + \alpha_{11}x_1 + \alpha_{12}x_2)y_2. \end{aligned} \tag{6.1}$$

This is a nonlinear system as it is quadratic in x and y. If we were to couple more oscillators in this manner (there can be N of them up until the CPU limit is reached), the order of the system will be N.

Shown in Figure 6.2 are the time series and spectrograms for the two oscillators in Sound Recordings 1-4. The timeline of the parameter specification is as follows. To start the left channel oscillator is at $f1 = 440$ Hz and the right is at $f2 = 220$ Hz. Here $f = 2\pi\omega$—the conversion from angular frequency to cycles per second. During the first 5 seconds, the modulation index from oscillator 2 to oscillator 1, α_{21} is raised from 0 to 10,000 (event A). Over the next 2 seconds, nothing changes. For 5 seconds after this, the modulation index from 1 to 2, α_{12}, increases from 0 to 5000 (event B). Then there is another 2 second hiatus of parameter adjustment. Following this, α_{12} climbs from 5000 to 7000 over another 5 second period (event C). After 2 seconds, α_{21} is returned to 0 and α_{12} is returned to 0 2 seconds after this (events D and E).



(a) $\phi = 0$

(b) $\phi = \pi/2$

(c) $\phi = \pi$

(d) $\phi = 3\pi/2$

Figure 6.2: Time and spectral series for Sound Recordings 1-4.

The only parametric difference in these examples is the initial phase of the oscillator 1. In this case, oscillator 1 begins at $\phi = 0$ for Sound Recording 1, $\phi = \pi/2$ for Sound Recording 2, $\phi = \pi$ for Sound Recording 3, and $\phi = 3\pi 2$ in Sound Recording 4.

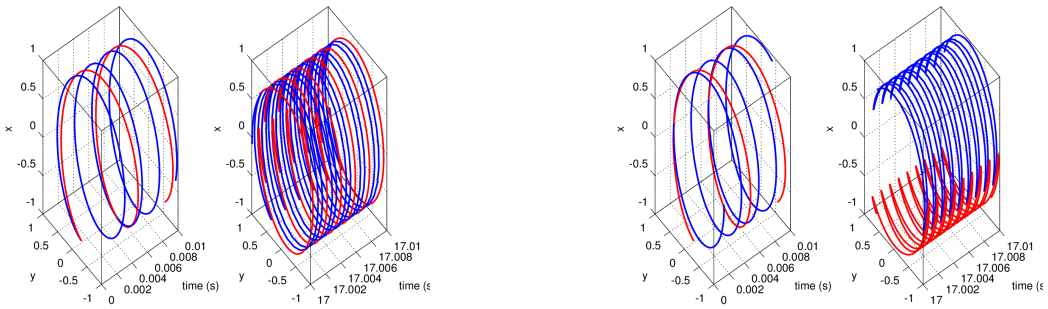
These sounds in and of themselves are not particularly rich or even very interesting. However, they serve to illustrate some important features of such a network. One thing to notice is that in the case of harmonic reciprocal FM, once the feedback loop is completed (event B), the oscillators unite in frequency almost immediately. The harmonic content is different, but the fundamental frequency becomes synchronized.

Also, the initial phase is very important to the behavior of the system. It is evident that there are points in the examples that the oscillators suddenly jump to a higher frequency immediately after a sudden dip down in frequency. These are clearly points of bifurcation. The point in time at which this bifurcation appears is different in each example due the initial phases (except for Sound Recording 3 in which this bifurcation does not occur). For example in Sound Recording 1 and 2, the bifurcation appears twice near the middle of event B, whereas in Sound Recording 2, the system bifurcates approximately one third of the way through event C.

There are other differences as well. The fundamental frequency of the system is different in each case, as is the region on the unit circle that each oscillator occupies in the high frequency (post-bifurcation) regime.

To clarify this last point, it is important to note that the views of the time series in Figure 6.2 is 1 dimensional across time, but each oscillator shown in Equation 6.1 is 2d. Since the system retains its energy throughout, the oscillation, the complex amplitude is always 1.

To illustrate this, we may examine both the x and y values given in equation 6.1. In Figure 6.3 we see that at the very beginning of the sequence of events both oscillators phase plots are on the edge of the unit circle. Midway through the examples, the oscillators still cling to the edge of the circle, but due to the dynamics of the system, they occupy only a portion of the circle, retracing their



(a) Sound Recording 3.

(b) Sound Recording 4.

Figure 6.3: A closer look at the time series for oscillators 1 and 2 at the very beginning of event A and after the bifurcation has occurred.

phases in a pattern prescribed by the initial phases and the progression of the parameters throughout the examples.

An interesting feature of Sound Recordings 1-4 is that the reciprocal feedback synchronizes the fundamental frequency of the oscillators. This is the so-called high frequency regime.

6.1.2 Harmonic FM With FBFM

In Sound Recording 5 the left channel oscillator is again given a frequency of $\omega_1/2\pi = f_1 = 440$ Hz, the right $\omega_2/2\pi = f_2 = 220$. Throughout the first 5 seconds (event A), the modulation index α_{21} (the index of oscillator 2 to oscillator 1) ramps from 0 to 10,000. The classical FM sound is apparent. After 2 seconds, the modulation index α_{22} (from oscillator 2 to itself) is increased from 0 to 1000 over the course of 2.5 seconds (event B). This sound is sustained for 2 seconds at which the same parameter (α_{22}) is ramped from 1000 to 1500 over the next 2.5 seconds (event C). When this value reaches about 1300, we reach a bifurcating point. At this point, the output of oscillator 2 becomes a constant value, that is its oscillation is completely squelched by its own output's contribution to its frequency via its modulation index. Again, we can see that this is correct behavior and not an artifact due to the fact that the oscillator will return to its original frequency and amplitude when the modulation parameters return to 0. At event

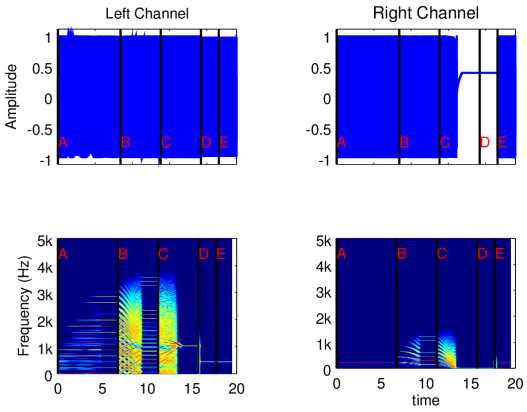


Figure 6.4: Time and spectral series in Sound Recording 5.

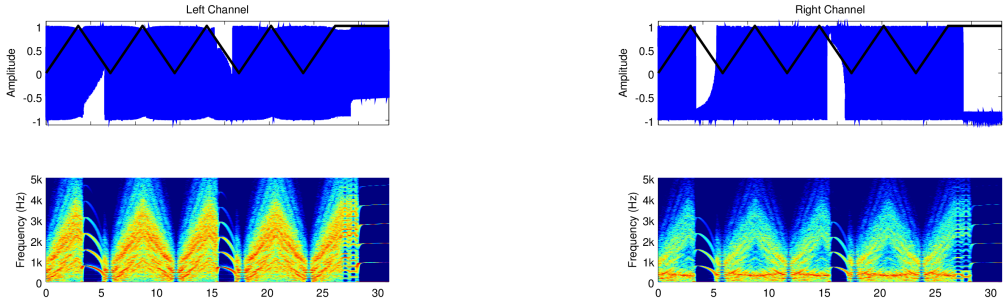
D (16 seconds in) α_{21} returns to 0 and 2 seconds later (event E), α_{22} returns to 0.

The time series and spectrogram of this example is shown in Figure 6.4. One notable feature is that the left channel sound demonstrates an inharmonic spectrum and as well as low frequency pulsing as a result of the feedback FM (FBFM) in oscillator 2. Another is that oscillator 2 is eventually held still as a result of the 0 delay feedback FM. At this point, of course, oscillator 1 loses all of its harmonics since it is effectively unmodulated. Again, once the FM coefficients revert to 0, the oscillators return to their original unmodulated regimes indicating a perfect preservation of energy throughout the course of the parameter changes.

6.1.3 Inharmonic FM

We here present an example of inharmonic FM in a reciprocal network. In Figure 6.5 the left and right channels show y_1 and y_2 for a network given by Equation 6.1. This corresponds to Sound Recording 6. The frequency of oscillator 1 is 440 Hz and 273 for oscillator 2. The value of α_{21} is held constant at 5k, and the value of α_{21} between 0 and 20k. This is shown as a black line in the upper plots of Figure 6.5, although the value is scaled so that it can be superimposed on the image of the waveform. The FBFM indices (α_{11} and α_{22}) are both 0.

The value of α_{21} is given as a triangle wave with a frequency of 1/6 Hz and amplitude range of 0 to 2k. After 4.5 oscillations, it remains at its upper peak for 2



(a) (b)

Figure 6.5: Time and spectral series for Sound Recording 6.

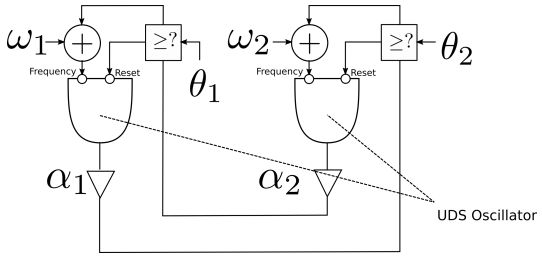


Figure 6.6: Signal flow diagram for a two oscillator reciprocal FM network.

seconds. It is evident that in the 3rd and 4th periods of the parameter adjustment, the oscillators fall into the high frequency regime (the oscillators become united in frequency at these points). It is notable that this happens for different values of α_{21}. It is also notable that this regime is reached on the downward slope of α_{21} in its 1st and 3rd periods; and, that the regime is reached again during the steady state of α_{21} at the end of the example.

6.2 Reciprocal Sync and FM

We now consider a two oscillator network such as the one given in Figure 6.6. This is the same network as the one shown in Figure 6.1,but with the addition of the nonlinear sync rule. Mathematically, the dynamics of the network is defined

thus:

$$
\begin{aligned}
\dot{y}_1 &= \begin{cases} (\omega_1 + \alpha_{11}x_1 + \alpha_{12}x_2)x_1 & \text{if } x_2 < \theta_1 \\ g_1(0 - y_1) & \text{if } x_2 \geq \theta_1 \end{cases} \\
\dot{x}_1 &= \begin{cases} -(\omega_1 + \alpha_{11}x_1 + \alpha_{12}x_2)y_1 & \text{if } x_2 < \theta_1 \\ g_1(-1 - x_1) & \text{if } x_2 \geq \theta_1 \end{cases} \\
\dot{y}_2 &= \begin{cases} (\omega_2 + \alpha_{11}x_1 + \alpha_{12}x_2)x_2 & \text{if } x_1 < \theta_2 \\ g_2(0 - y_2) & \text{if } x_1 \geq \theta_2 \end{cases} \\
\dot{x}_2 &= \begin{cases} -(\omega_2 + \alpha_{11}x_1 + \alpha_{12}x_2)y_2 & \text{if } x_1 < \theta_2 \\ g_2(-1 - x_2) & \text{if } x_1 \geq \theta_2. \end{cases}
\end{aligned}
\tag{6.2}
$$

Sound example Sound Recording 7 illustrates the kind of behavior that can be generated by varying the parameters in Equation 6.2. A spectrogram of y_1 and y_2 are shown in Figure 6.7. In this example the frequency of oscillators 1 and 2 are 440 and 550 Hz, respectively. In event A, α_{12} increases from 0 to 5000 in 2.5 seconds. 2.5 seconds later (event B), α_{21} is increased from 0 to 3000 over 2.5 seconds (reciprocal FM). Event C initiates a 2.5 second period over which θ_1 decreases from 1.2 (well above the amplitude of the oscillator) to .5, initiating a sync regime. 2.5 seconds later, at event D, θ_2 likewise decreased from 1.2 to .5. Event E reduces the modulation from oscillator 1 to 2 (α_{12}) from 5000 to 0 over 2.5 seconds. Another 2.5 seconds of no parameter changes go by. Then, at event F, the FM coefficient α_{21} is brought back down to 0 (again, over the course of 2.5 seconds). Event G initiates a further reduction in θ_1 from 50 to 4. 2.5 more seconds pass with no changes, then (event H) θ_2 is also reduced to 4. 12.5 seconds pass with these parameters held fixed.

At event E, both reciprocal FM and reciprocal sync are in full effect. The result is a highly chaotic pair of oscillators that intermittently lock in and out of oscillating regimes. Both the pitch and the timbre jumps about wildly. Again, a notable effect is the fusion and fission of the hard panned oscillators. As in the earlier examples, reciprocal FM locks the frequencies of the oscillators together. Although the harmonics have different strengths, they are harmonic (and thus also equal in pitch) for both cases. The introduction of a sync threshold destroys this

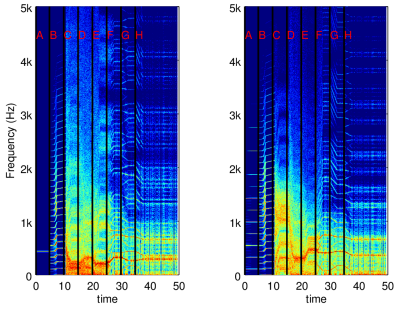


Figure 6.7: The left and right channels for the y in Sound Recording 7.

fusion. But when the reciprocal sync circuit is completed (both θ_1 and θ_2 are below 1) the oscillators are mutually dependent in frequency and behavior and perceptually somewhat united.

6.2.1 Reciprocal FM/Sync Sketches

Sound Recordings 8-10 illustrate a variety of sounds that can be made simply with two oscillators in a reciprocal FM/sync structure as shown in Figure 6.6. In all three examples only the FM indices, sync thresholds, and frequencies of the oscillators were adjusted. There was no post processing apart from fades in and out.

Sound Recording 8 is a free form improvisation and serves to show the range of sounds that such a structure can create. The sudden leaps between of highly chaotic noises to clear high frequency hits that occur in the section beginning around 3:30 was made by placing the oscillators in tight reciprocating loops and quickly ramping the oscillator of one of the oscillators from the low hundreds of Hertz to several thousands of Herz.

Sound Recording 9 is a low frequency process. There are some other minor parameter adjustments, but essentially the oscillators are very near each other in frequency, α_{21} is kept low, and α_{12} is slowly ramped from 0, to 60,000 and back. On the way back down, oscillator 1 is slave to oscillator 2 in the sync regime.

Sound Recording 10 is about high frequencies. Here, both α_{12} and α_{21} are non-zero and $\omega_1 * 2\pi$ is 2500 Hz. Oscillator 2 alternates between slaving to both

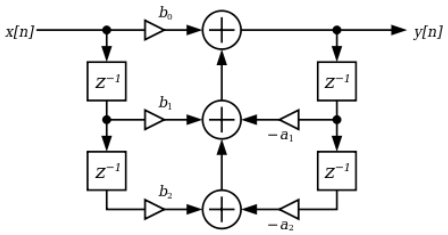


Figure 6.8: Digital biquad filter in direct form II.

oscillator 1 and itself and slaving only to oscillator 1. When the oscillator is slaving to itself, its movement is highly squelch. It is only able to move in short impulse-like bursts. When this sync rule is removed, it is still highly restricted, but has more of a voice. During these periods, α_{12} and ω_2 are adjusted.

The adjustment of parameters in all 3 examples were improvised (with the exception of the slow automatic ramping of α_{12} in Sound Recording 9). These examples are studies in the sonic nature of this highly chaotic network. From the space of all sounds that such a network can create, nearly everything is represented in these three examples. However, the fine grained behavior of the sounds is extremely unpredictable. Through listening and familiarity with the network, one can only somewhat control the output.

6.3 Moog Filter Examples

We begin with an example that illustrates the 'sound' of the UDS implementation of the Moog ladder filter, Sound Recording 11. The input to the filter here is a 100 Hz sawtooth wave. The cutoff frequency is swept between 0 Hz, 2.5 kHz and back in 10 second intervals. There are 4 passes, at the first the parameter r in Equation 5.16, is 1, then 2, then 3, and finally 3.9. This is the feedback parameter that controls the bandwidth of the filter. For lower values, this parameter causes the filter to have a low-pass characteristic. As it increases, the filter becomes a band-pass filter. Once r reaches 4, the filter self oscillates.

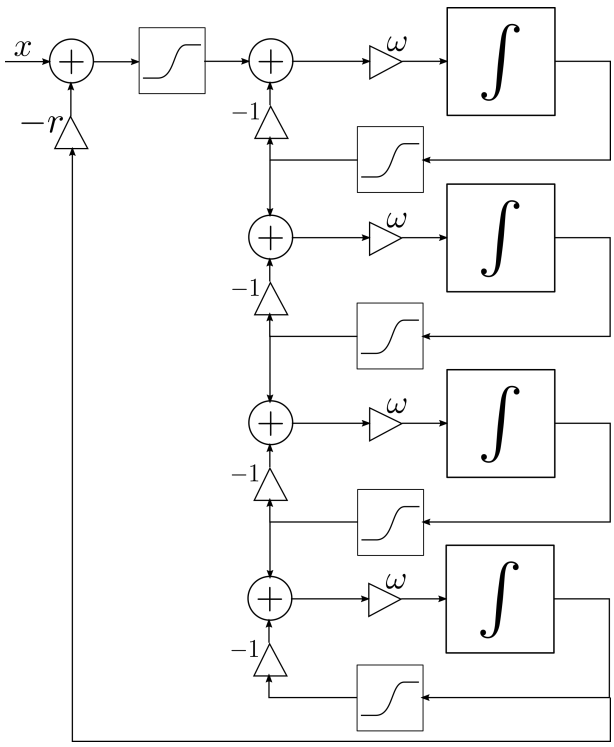


Figure 6.9: Signal flow diagram for the UDS Moog filter; $f_{nl}(x) = \tanh(x)$.

6.3.1 UDS Block Diagram Representation

It is often convenient to represent digital audio pipelines in terms of block diagrams. This is commonly done, especially in describing digital filter topologies. Consider, for example, the oft seen depiction of a digital biquad filter in direct from II (Figure 6.8[1]). This is a more specific language of signal flow than the ones we've seen thus far in that one can infer the actual difference equation from inspecting the diagram itself. Borrowing from this symbology, we may draw UDS algorithm topologies. To accomplish this we introduce a symbol to show integration ($\int$) and a black box to indicate a nonlinear function (f_{nl}). With this repertoire of symbols, the Moog filter depicting the system given by the Equations 5.15 and 5.16 is easily constructed (Figure 6.9). In this case, the nonlinear function is a smooth clipping function ($f_{nl} = \tanh$) and is depicted graphically as such in the diagram.

[1]Taken from Wikipedia, https://en.wikipedia.org/wiki/Digital_filter#/media/File:Biquad_filter_DF-II.svg

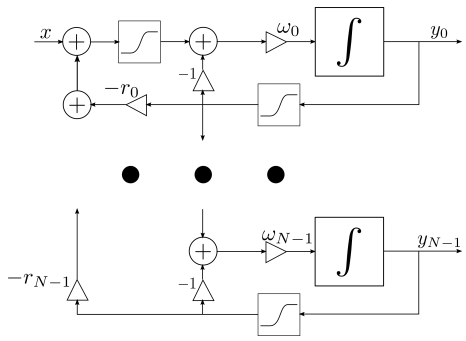


Figure 6.10: Signal flow diagram for an extended UDS Moog filter; the dots stand in for $N-2$ identical stages.

6.3.2 A Time Varying Filter

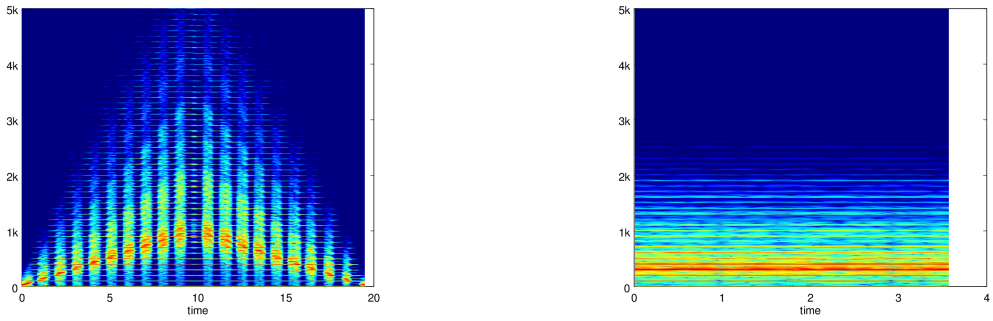
One of the nice things about the Moog ladder filter given in Equations 5.16 and 5.15, as opposed to any LTI filter is that it is time varying. As any analog synthesizer owner knows, filters sound best when they are modulated with other audio signals. Implementing the Moog filter using UDS puts us in a position to use it this way.

Sound Recording 12 gives an impression of some of the games that one can play with a time varying filter such as this. The input is again a 100 Hz sawtooth wave. Various other oscillators are used to control the frequency and amplitude of the cutoff frequency. Throughout this example, the value of r was adjusted by hand between .01 and 10 (which is well beyond the saturation point).

6.3.3 Extensions

This representation immediately suggests modifying the filter simply by adding stages. In Figure 6.10, we have a more general form of that shown in Figure 6.9. Not only are there N stages, each stage can be fed back into the input with a unique r_n resonance parameter. Furthermore, the gain factor ω_n is also generalized so that it is unique and controllable at each stage of the filter.

Sound Recordings 13 and 14 illustrate the sounds of a structure such as the one in Figure 6.10 with 8 stages. In example 8a, the input is a 100 Hz sawtooth and the cutoff frequency for each stage (which is now an independent control in Figure



(a) (b)

Figure 6.11: Spectrograms of Sound Recordings 13 and 14.

6.10) is swept between 0 and 2.5 kHz as in example 6. However, the resonance parameters (r_0 through r_7) are as follows: $r_1 = -1$, $r_2 = -1$, $r_3 = 2$, $r_4 = .5$, and all the rest are 0. In this configuration, the filter takes on multiple resonances and the output (which we read off of the 4th stage) will chaotically jump between them.

This behavior is clarified in Sound Recording 14 which has the same resonance parameters, but the cutoff frequency at each stage is held fixed at 800 Hz. When the cutoff frequency sweeps, the resonance jumping comes off as a kind of roughness to the sound. It is also notable that there is a significant flattening to the sound. Spectrograms for these examples are given in Figure 6.11.

6.4 Chaotic Oscillators

In Section 5.3.5 we saw a nonlinear noisebox that was partially inspired by previous use of the Chua circuit in physical modeling. We also saw some analysis of the unadulterated Chua circuit model with the addition of a frequency throttle. The next section showed a similarly frequency controllable Lorenz attractor. The following sound/signal flow diagrams refer to these examples.

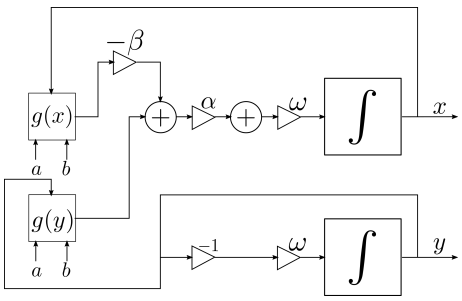


Figure 6.12: Signal flow diagram for Sound Recording 15.

6.4.1 Chua-inspired Nonlinear Noisebox

Sound Recording 15 is the realization of the system given by Equation 5.29 with $g(x)$ given by Equation 5.28. Essentially this is a cubic nonlinear oscillator with a controllable frequency parameter, ω. The signal flow diagram for such a circuit is shown in Figure 6.12.

In the sound example all the parameters are held fixed except for the value of a in Equation 5.28: $\omega = 75$, $\alpha = 2.3$, $\beta = .03$, and $b = -.65$. The value of a is swept from 1 to 60 over the course of 100 seconds.

This gives a taste of the kinds of horrible noise that such a noisebox can generate. There is a great deal of high frequency content in the sound, and aliasing is a strong feature. There are times, though, that the oscillator will settle into a somewhat harmonic regime.

It should also be noted that this system is highly unstable. Furthermore the realm of stability in the system is dependent on the rather complex feature space. Being a chaotic system, the direction from which vectors in the parameter space are approached, and the state of the oscillator at the time the parameters are attained will effect the output significantly.

Such a system is a good example of a viable application for UDS. Indeed this is a highly chaotic and unstable system, but there are some real sonic gems in there if one is willing to hunt for them. 'Practice' controlling such an instrument is not a fruitless endeavor. Of course, the method of controlling such a noisemaker is an enormous and open ended question.

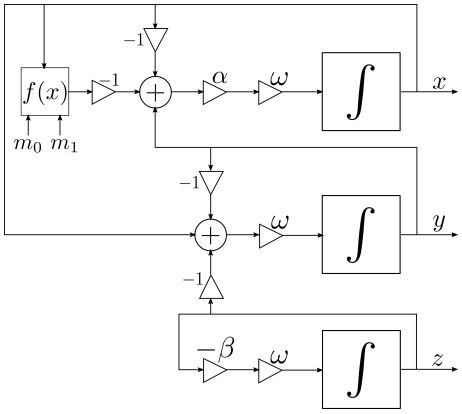


Figure 6.13: Signal flow diagram for example 10.

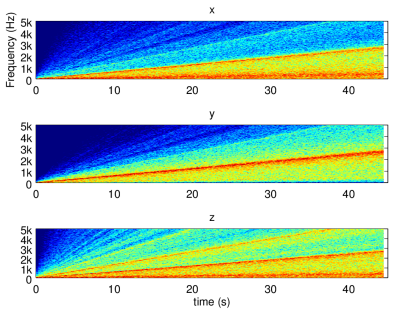


Figure 6.14: Spectrograms for Sound Recording 16.

6.4.2 Audition of the Chua Circuit

Sound Recording 16 is a 3 channel soundfile that illustrates the plain old Chua circuit in action. Recorded in the file is the x, y, and z values as they change over time. The amplitude was attenuated by a factor of 0.025 in order to keep the amplitude safely between -1 and 1 (which is necessary for digital to analog conversion). The circuit itself (in UDS signal flow parlance) is shown in Figure 6.13. the system itself is given in Equation 5.23 with the exception that each node is multiplied by the frequency throttle parameter ω. In the example, the parameters in this example are those given in Section 5.3.5. The value of ω is ramped between 0 and 6000 over the course of 45 seconds. Spectrograms for each variable in the system (x, y and z) are shown in Figure 6.14.

This sound is fairly interesting, but the system itself maintains limit cycles

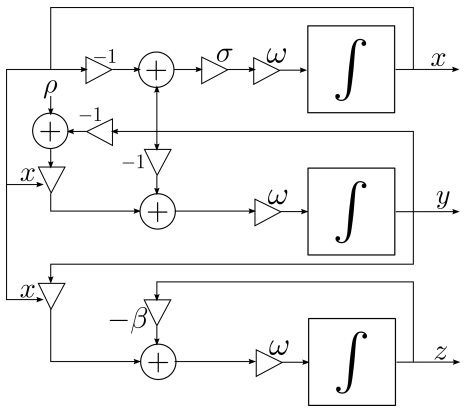


Figure 6.15: Signal flow diagram for Sound Recording 17.

only in a fairly narrow band of parameter space. This means that the texture, while rich, can not be greatly varied. Of course, the chaoticness of the oscillator (and the fact that it has 3 mutually dependent, chaotic outputs suggest that this is a good candidate for use as an LFO.

6.4.3 Lorenz Systems

Figure 6.15 demonstrates the system given by Equation 5.30 with the addition of the frequency throttle parameter ω which multiplies each node of the system. This is the system that was used to generate the spectrogram in Figure 5.17.

Sound Recording 17 is a 3 channel soundfile that uses the same parameters as the one used to generate Figure 5.17. In that example, $\omega = 500$, $\rho = 28$, $\sigma = 10$, and β is swept from .001 to 8/3. That picture also shows only the x dimension of that system. Sound example 11 is the same only its three channels are each of the three dimension in the Lorenz system (x, y and z) and the soundfile lasts 50 seconds (the one used to generate Figure 5.17 only lasted 5 seconds).

The other difference is that in order to place the time-varying values of the state variables in the system in an appropriate range for a DAC, they have had their DC component removed (through a simple high-pass filter with a cutoff frequency of 5 Hz) and attenuated by a factor of .025.

6.4.4 Remarks Regarding Chaotic Systems

The last three examples show highly nonlinear systems. In the case of the last two, these are also three dimensional systems. All of them are chaotic. One issue that arises when working with such 'extreme' systems for audio synthesis is that the parameter space in which they will maintain limit cycles (oscillations) is rather narrow. The Chua circuit is particularly rigid in this regard. This poses a challenge of designing a control mechanism for an instrument incorporating such a system; namely, how does one restrict the user's input so that it does not stray from the 'sweet spots' in the parameter space? Another issue is that the variety of sounds available from such restrictions is also narrow. That doesn't mean the systems aren't rich and useful pieces of larger structures, but these problems are posed. For these very reasons, musical applications of these models has traditionally been in generating very slowly changing structures (i.e. formal structures such as section durations etc.).

6.5 Conclusions

The use of dynamical models in sound synthesis is not well studied. By and large it is still virgin territory and the above is but a small foray into that realm.

Chapter 7

Final Remarks and Conlcusion

To sum up: we present a technique for audio synthesis that we call unsampled digital synthesis, or UDS for short. The methods comprising the technique itself (numerical solutions to dynamical systems given by first order ordinary difference equations) are not new. But, attempting to provide a general framework for experimenting with such methods in a cohesive manner is new. Furthermore, (cohesive framework or no) there is not a lot of work being done with such methods in the realm of digital audio synthesis anyway.

This is odd. After all, many tried and true nonlinear models that are familiar in the computer music literature incorporate dynamical systems. However, usually such models try to linearize and discretize the system prior to the solution stage: that is to say, such systems are often forced into the template of an LTI system with a special case nonlinear modification at some point. This is the case with nonlinear modifications to digital waveguides and linear wave digital filters.

While UDS is not general enough to cover all nonlinear systems (specifically it does solve for implicit *equations*) it does provide a cheap, straight forward, and meaningful avenue to experiment with a large subset of all nonlinear and implicit systems, namely those that are of the form given in Equation 1.1 or those like it of higher dimensions.

Obviously, incorporating tools for handling implicit equations is a desirable extension of what we have here. On the other hand, the light-weight-ness of UDS it is part of its virtue and implicit solvers would obliterate this feature–at least for

the time being.

The question remains: what is the subset of such systems that makes sound? We provide a short list and demonstrate that these are powerful tools. However, more work needs to be done.

With the aim of accomplishing this, there are several techniques. One is experimentation through trial and error, naively getting a feel for what works and what doesn't all the while using one's ears as a guide. To this end, expressing dynamical systems as signal flow diagrams such as the ones shown in the previous chapter is a good way to proceed. An application that could allow a user to 'draw' systems and translate them into the corresponding timelab code could be a powerful tool.

Another technique for fleshing this out is to use mathematical tools to determine the space of systems and the space of parameters within those systems that produce limit cycles. This is nontrivial to say the least.

Yet another way to develop systems that sound good, is to draw inspiration from nature. At no point in this thesis is analysis addressed. However, it is not unreasonable to develop tools that can take sound waves as input and will in turn output parameterized systems that will produce such oscillations, much the same way linear predictive coding does.

All of which is to say, there is plenty of work left to do in this area of research.

And so we come to the end of yet another PhD thesis. Another nugget of knowledge has been added to the project of human understanding. Its value, if any, to the world beyond the walls of the mind of the author, and the cloisters of the University is now free to be proved. As for the greater question (and really the ultimate question) of from whence the world comes and how it came to be, we must acknowledge the insight of Witgenstein and be content to know that the meaning of the world is not contained within the world. That meaning is thus unknowable to us that reside within the world. Where this all came from and where it is going is beyond the scope of our understanding; but, at least we now know how to make a digital noisebox out of interconnected gains, integrators, and

nonlinear functions.

Bibliography

[AAA+16] BP Abbott, R Abbott, TD Abbott, MR Abernathy, F Acernese, K Ackley, C Adams, T Adams, P Addesso, RX Adhikari, et al. Observation of gravitational waves from a binary black hole merger. *Physical Review Letters*, 116(6):061102, 2016.

[AH71] Bishnu S Atal and Suzanne L Hanauer. Speech analysis and synthesis by linear prediction of the speech wave. *The journal of the acoustical society of America*, 50(2B):637–655, 1971.

[Ald08] John Alderman. *Sonic boom: Napster, MP3, and the new pioneers of music*. Basic Books, 2008.

[BBMS03] J Bensa, S Bilbao, Kronland R Martinet, and JO Smith. A power normalized non-linear lossy piano hammer. In *Proceedings of the Stokholm Muisc Acoustics Conference, Stokholm*, pages 365–368, 2003.

[BDPR00] Gianpaolo Borin, Giovanni De Poli, and Davide Rocchesso. Elimination of delay-free loops in discrete-time models of nonlinear acoustic systems. *Speech and Audio Processing, IEEE Transactions on*, 8(5):597–605, 2000.

[Bil07] Stefan Bilbao. A digital plate reverberation algorithm. *Journal of the Audio Engineering Society*, 55(3):135–144, 2007.

[Bil09] Stefan Bilbao. *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*. Wiley Online Library, 2009.

[Bou86] Richard Charles Boulanger. *The Transformation of Speech Into Music: A Musical Exploration and Interpretation of the Two Recent Digital Filtering Techniques*. University Microfilm International, 1986.

[Bri08] Robert Bridson. *Fluid Simulation for Computer Graphics*. A K Peters, Ltd., 2008.

[Bri15] Luke Morgan Britton. Millenials push 2015 vinyl sales to 26-year high in us. *New Music Express*, November 2015.

[CA84] Lothar Cremer and John S Allen. *The physics of the violin*. MIT press Cambridge, MA, USA:, 1984.

[Cho73] John M Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Society*, 21(7):526–534, 1973.

[Cho77] John M Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Computer Music Journal*, pages 46–54, 1977.

[CKEI92] Leon O Chua, Ljupco Kocarev, Kevin Eckert, and Makoto Itoh. Experimental chaos synchronization in chua's circuit. *International Journal of Bifurcation and Chaos*, 2(03):705–708, 1992.

[Coo90] Perry R. Cook. *Identification of Control Parameters in an Articulatory Vocal Tract Model, with Applications to the Synthesis of Singing*. PhD thesis, Stanford, 1990.

[d'A73] J. l. R. d'Alembert. Investigation of the curve formed by a vibrating string, 1747. In R. B. Lindsay, editor, *Acoustics: Historical and Philosophical Development*, pages 119–123. Stroudsburg: Dowden, Hutchinson & Ross, 1973.

[Dal12] Paul Daly. A comparison of virtual analogue moog vcf models. Master's thesis, University of Edinburgh, 2012.

[Dan13] Roger Dannenberg. Principles for effective real-time music processing systems, May 2013.

[DDHZ11] P. Dutilleux, K. Dempwolf, M. Holters, and U. Zölzer. Nonlinear processing. In Udo Zölzer, editor, *DAFX: Digital Audio Effects, Second Edition*. John Wiley & Sons, 2011.

[DHZ10] Kristjan Dempwolf, Martin Holters, and Udo Zölzer. Discretization of parametric analog circuits for real-time simulations. In *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx10)*, 2010.

[DJ97] Charles Dodge and Thomas A Jerse. *Computer music: synthesis, composition and performance*. Macmillan Library Reference, 1997.

[Fet71] Alfred Fettweis. Digital filters related to classical structures. *AEU: Archive f\" ur Elektronik und\" Ubertragungstechnik*, 25:78–89, 1971.

[Fet86] Alfred Fettweis. Wave digital filters: Theory and practice. *Proceedings of the IEEE*, 74(2):270–327, 1986.

[Gea71] C William Gear. *Numerical initial value problems in ordinary differential equations*. Prentice Hall PTR, 1971.

[GW79] John Guckenheimer and Robert F Williams. Structural stability of lorenz attractors. *Publications Mathématiques de l'IHÉS*, 50:59–72, 1979.

[HHH89] Paul Horowitz, Winfield Hill, and Thomas C Hayes. *The art of electronics*, volume 2. Cambridge university press Cambridge, 1989.

[Hof02] Abbie Hoffman. *Steal this book*. instinct. org, 2002.

[HR71] Lejaren Hiller and Pierre Ruiz. Synthesizing musical sounds by solving the wave equation for vibrating objects: Part 1, part 2. *Journal of the Audio Engineering Society*, 19(6):462–470, 542–551, 1971.

[HSS00] Patty Huang, Stefania Serafin, and JO Smith. A waveguide mesh model of high-frequency violin body resonances. In *Proc. 2000 Int. Computer Music Conf., Berlin*, 2000.

[Huo04] Antti Huovilainen. Nonlinear digital implementation of the moog ladder filter. In *Proc. Int. Conf. on Digital Audio Effects (Naples, Italy, October 2004)*, pages 61–4, 2004.

[JLK+84] L Jackson, A Lindgren, Y Kim, et al. A chaotic attractor from chuas circuit. *IEEE Trans. Circuits Syst*, 31(12):1055–1058, 1984.

[JS83] David A Jaffe and Julius O Smith. Extensions of the karplus-strong plucked-string algorithm. *Computer Music Journal*, 7(2):56–69, 1983.

[KE04] Matti Karjalainen and Cumhur Erkut. Digital waveguides versus finite difference structures: Equivalence and mixed modeling. *EURASIP Journal on Applied Signal Processing*, 2004:978–989, 2004.

[KES03] Matti Karjalainen, Cumhur Erkut, and Lauri Savioja. Compilation of unified physical models for efficient sound synthesis. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 5, pages V–433. IEEE, 2003.

[KL62] John L Kelly and Carol C Lochbaum. Speech synthesis. *Proceedings of the Fourth International Congress on Acoustics, Copenhagen*, pages 1–4, 1962.

[Kor96] Norman Koren. Improved vacuum tube models for spice simulations. *Glass Audio*, 8(5):18–27, 1996.

[Lor63] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2):130–141, 1963.

[LR11] Jaime E Oliver La Rosa. *A Computer Music Instrmentarium*. PhD thesis, University of California, San Diego, 2011.

[LR12] Jaime E Oliver La Rosa. Theremin in the press: Construing electrical music. 2012.

[Mad93] Rabinder N Madan. *Chua's circuit: a paradigm for chaos*, volume 1. World Scientific, 1993.

[Mat63] M. V. Mathews. The digital computer as a musical instrument. *Science*, 142(3592):pp. 553–557, 1963.

[McC12] James McCartney. SuperCollider and time, September 2012.

[Med13] David Medine. Timelab: Yet, yet another audio programming environment. In *Proceedings of the International Computer Music Conference, Perth*, 2013.

[Med16] David Medine. Dynamical systems for audio synthesis: Embracing delay free loops. *Applied Sciences*, 2016.

[MMR74] Max V Mathews, FR Moore, and JC Risset. Computers and future music. *Science*, 183:263–268, 1974.

[MNH01] Damian T Murphy, Chris JC Newton, and David M Howard. Digital waveguide mesh modelling of room acoustics: Surround-sound, boundaries and plugin implementation. In *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFx),(Limerick, Ireland)*, 2001.

[Moo65] Robert A Moog. A voltage-controlled low-pass high-pass filter for audio signal processing. In *Audio Engineering Society Convention 17*. Audio Engineering Society, 1965.

[Moo90] F. Richard Moore. *Elements of Computer Music*. Prentice Hall, 1990.

[MQ08] Don Morgan and Sanzheng Qiao. Accuracy and stability in mass-spring systems for sound synthesis. In *Proceedings of the 2008 C 3 S 2 E conference*, pages 69–80. ACM, 2008.

[MS03] Max Mathews and Julius O Smith. Methods for synthesizing very high q parametrically well behaved two pole filters. In *Proceedings of the Stockholm Musical Acoustics Conference (SMAC 2003)(Stockholm), Royal Swedish Academy of Music (August 2003)*, 2003.

[MS09] Jaromir Macak and Jiri Schimmel. Nonlinear circuit simulation using time-variant filter. *Proc. of the International Conference on Digital Audio Effects (DAFx)*, 2009.

[MS10] Jaromir Macak and Jiri Schimmel. Real-time guitar tube amplifier simulation using an approximation of differential equations. In *Proceedings of the 13th International Conference on Digital Audio Effects (DAFx10)*, 2010.

[MW79] ME McIntyre and J Woodhouse. On the fundamentals of bowed-string dynamics. *Acta Acustica united with Acustica*, 43(2):93–108, 1979.

[Nor16] Vesa Norilo. Kronos: A declarative metaprogramming language for digital signal processing. *Computer Music Journal*, 2016.

[PK10] Jyri Pakarinen and Matti Karjalainen. Enhanced wave digital triode model for real-time tube amplifier emulation. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(4):738–746, 2010.

[PTK09] Jyri Pakarinen, Miikka Tikander, and Matti Karjalainen. Wave digital modeling of the output chain of a vacuum-tube amplifier. In *Proceedings of the International Conference on Digital Audio Effects (DAFx09)*, pages 1–4, 2009.

[PTP09] Trevor J Pinch, Frank Trocco, and TJ Pinch. *Analog days: The invention and impact of the Moog synthesizer*. Harvard University Press, 2009.

[PTVF96] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes in C*, volume 2. Citeseer, 1996.

[Puc96] Miller Puckette. Pure Data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, 1996.

[Puc07] Miller Puckette. *The Theory and Technique of Electronic Music*. World Scientific Publishing, 2007.

[Puc12] Miller Puckette. Timeless problems in real-time audio software design, March 2012.

[PV11] Jussi Pekonen and Vesa Välimäki. The brief history of virtual analog synthesis. In *Proceedings of the 6th Forum Acusticum, Aalborg, Denmark*, pages 461–466, 2011.

[PW98] R Pitteroff and J Woodhouse. Mechanics of the contact area between a violin bow and a string. part ii: Simulating the bowed string. *Acta Acustica united with Acustica*, 84(4):744–757, 1998.

[RA] Jonatan Pena Ramırez and Joaqun Alvarez. Conditions to synchronize nonlinear oscillators interacting via time-delayed dynamic coupling.

[RG75] Lawrence R Rabiner and Bernard Gold. Theory and application of digital signal processing. *Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975. 777 p.*, 1, 1975.

[Ris05] Jean-Claude Risset. Computer study of trumpet tones. *The Journal of the Acoustical Society of America*, 38(5):912–912, 2005.

[Roa96] Curtis Roads. *The computer music tutorial*. MIT press, 1996.

[Rod93] Xavier Rodet. Models of musical instruments from chua's circuit with time delay. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 40(10):696–701, 1993.

[Ros92] D Rossum. Making digital filters sound analog. In *Porceedings of the International Computer Music Conference, San Jose*, pages 30–33, 1992.

[RS11] Lawrence Rabier and Ronald Schafer. *Theory and Applications of Digital Speech Processing*. Upsaddle River: Pearson, 2011.

[RW+99] Jean-Claude Risset, David L Wessel, et al. Exploration of timbre by analysis and synthesis. *The psychology of music*, 28, 1999.

[SDP99] Augusto Sarti and Giovanni De Poli. Toward nonlinear wave digital filters. *Signal Processing, IEEE Transactions on*, 47(6):1654–1668, 1999.

[Ser04] Stefania Serafin. *The sound of friction: real-time models, playability and musical applications*. PhD thesis, stanford university, 2004.

[Smi85] Julius Orion Smith. A new approach to digital reverberation using closed waveguide networks. In *Proceedings of the International Computer Music Conference, Vancouver*, pages 47–53, 1985.

[Smi96] Julius O Smith. Physical modeling synthesis update. *Computer Music Journal*, pages 44–56, 1996.

[Smi10] J. O. Smith. *Physical Audio Signal Processing*. W3K Publishing, 2010.

[Spa12] Colin Sparrow. *The Lorenz equations: bifurcations, chaos, and strange attractors*, volume 41. Springer Science & Business Media, 2012.

[Spa15] Miroslav Spasov. Using strange attractors to control sound processing in live electroacoustic composition. *Computer Music Journal*, 2015.

[SS96] Tim Stilson and Julius Smith. Analyzing the moog vcf with considerations for digital implementation. In *Proceedings of the 1996 International Computer Music Conference, Hong Kong, Computer Music Association*, 1996.

[Thé97] Paul Théberge. *Any sound you can imagine: Making music/consuming technology*. Wesleyan University Press, 1997.

[Val05] V Valimaki. Discrete-time synthesis of the sawtooth waveform with reduced aliasing. *Signal Processing Letters, IEEE*, 12(3):214–217, 2005.

[VBS+11] V. Välimäki, S. Bilbao, J. O. Smith, J. S. Abel, J. Pakarinen, and D.Berners. Virtual analog effects. In Udo Zölzer, editor, *DAFX: Digital Audio Effects, Second Edition*. John Wiley & Sons, 2011.

[VDPS94] Scott A Van Duyne, John R Pierce, and Julius O Smith. Traveling wave implementation of a lossless mode-coupling filter and the wave digital hammer. In *Proceedings of the International Computer Music Conference, Aarhus*, pages 411–418, 1994.

[VH06] Vesa Välimäki and Antti Huovilainen. Oscillator and filter algorithms for virtual analog synthesis. *Computer Music Journal*, 30(2):19–31, 2006.

[VNSA10] Vesa Valimaki, Juhan Nam, Julius O Smith, and Jonathan S Abel. Alias-suppressed oscillators based on differentiated polynomial waveforms. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(4):786–798, 2010.

[VPEK06] Vesa Välimäki, Jyri Pakarinen, Cumhur Erkut, and Matti Karjalainen. Discrete-time modelling of musical instruments. *Reports on progress in physics*, 69(1):1, 2006.

[vWC03] Maarten van Walstijn and Murray Campbell. Discrete-time modeling of woodwind instrument bores using wave variables. *The Journal of the Acoustical Society of America*, 113(1):575–585, 2003.

[WC+03] Ge Wang, Perry R Cook, et al. Chuck: A concurrent, on-the-fly audio programming language. In *Proceedings of International Computer Music Conference*, pages 219–226, 2003.

[Wei95] Reynold Weidenaar. *Magic music from the telharmonium*. Reynold Weidenaar, 1995.

[WNSIA15] KJ Werner, V Nangia, JO Smith III, and JS Abel. Resolving wave digital filters with multiple/multiport nonlinearities,. *submitted to DAFx*, 2015.

[WSIA15] KJ Werner, JO Smith III, and JS Abel. Wave digital filter adaptors for arbitrary topologies and multiport linear elements. *submitted to DAFx*, 2015.

[YAS07] David T Yeh, Jonathan Abel, and Julius O Smith. Simulation of the diode limiter in guitar distortion circuits by numerical solution of ordinary differential equations. *Proceedings of the Digital Audio Effects (DAFx07)*, pages 197–204, 2007.

[Yeh09] David Te-Mao Yeh. *Digital implementation of musical distortion circuits by analysis and simulation*. PhD thesis, Stanford University, 2009.

[Zho94] Guo-Qun Zhong. Implementation of chua's circuit with a cubic nonlinearity. *IEEE Transactions on Circuits and Systems-Part I-Fundamental Theory and Applications*, 41(12):934–940, 1994.