Computational Thinking through Modular Sounds Synthesis

Andrew M. Olney

2022-11-01

Contents

Welcome

This is the official website for "Computational Thinking through Modular Sound Synthesis." This book will teach you computational thinking through modular sound synthesis (hereafter *modular*). You'll learn how to trigger sounds, create sounds, and modify sounds to solve specific sound design problems and create compositions. Along the way, you'll learn computational thinking practices that transcend modular and can be applied to a variety of problem-solving domains, but which are particularly relevant to information processing domains like computing.

If you're wondering whether this is a book about computational thinking, or a book about modular, the answer is both: on the surface, most content is about modular, but computational thinking is a style of thinking reflected in the presentation of the material and gives it additional coherence. As you work through the book, you'll become more proficient in computational thinking practices like decomposition, algorithmic design, evaluation of solutions, pattern recognition, and abstraction.

This book is *interactive*, which is why it is an e-book rather than a paper book.[1] Throughout you will encounter examples, simulations, and exercises that run in your browser to demonstrate and reinforce key concepts. Don't skip the interactive activities!

The recommended browser is Firefox on a non-iOS system. The modular software embedded in the book has a large download (100+ MB), and Firefox will cache the files so you only download them once. Other browsers will make you download them every time, which slows things down. And iOS browsers won't work at all.[2]

[1] I've also created PDF and EPUB versions; these are best used when you don't have internet because they are **not** interactive. Detailed instructions, solutions, and starter templates only exist for the web version.

[2] As long as Safari (WebKit) doesn't support fixed-width SIMD on web assembly and non-WebKit browsers are disallowed, the embedded modular software won't work on iOS!

Chapter 1

Introduction

1.1 Why this book?

Let's start with why I'm writing it. I got into electronic music in the 1990s when I lived in London but never transitioned from DJing to making music, though several of my friends did. A few years ago, they started talking about modular, and in talking to them and trying to find out more about it, I realized a few things:

- The best books (to me) were from the 1970s and 1980s[1]
- Modular synthesis is really well aligned with *computational thinking*

If you've never heard of computational thinking and/or modular, that last point won't make a lot of sense, so let's break it down.

Modular sound synthesis (modular) creates sound by connecting modules that each perform some function on sound. Different sounds are created by combining modules in different ways.

Computational thinking creates runnable models to solve a problem or answer a question. Models can be scientific models (e.g. meteorology), statistical models (e.g. statistics/data science), computation models (computer science), and perhaps other kinds of models.

How are they connected? Modular involves computational thinking when we:

[1] Old books that I like are Crombie (1982) and Strange (1983). Newer books of note are Bjørn and Meyer (2018), which gives a great overview of module hardware and history, Eliraz (2022), which gives a broader overview of issues related to musical equipment and production, and Dusha et al. (2020), which gives a modern but briefer introduction to modular than the older books. There are also some online courses (paid), but since I haven't taken them, I'm not listing them here.

- Simulate an instrument by reverse engineering its sound
- Create new sounds based on models of signal processing

Why should you read this book? This book is about modular, but it approaches modular in a way that highlights computational thinking. I believe this deeper approach to modular will help you do more with modular, other synthesizers, and studio production tools. Additionally, the computational thinking approach should help accelerate your learning of computational-thinking domains in the future. Since computational thinking involves problem solving, this book is full of interactive activities that will let you hone your modular skills - something you won't find in most books!

The next sections give some background on computational thinking and modular to better explain where this book is coming from.

1.2 Computational thinking

Tedre and Denning (2016) present a nice overview of the history of computational thinking. Here's a brief summary.

When the field of computing was taking off in the 1950s, there was interest and discussion about how it was different from other fields (e.g. math). One argument was that computing involved *algorithmic thinking*, which is designing algorithms to solve problems (cf. programming), and this kind of thinking was unique to computing. Some even thought that this kind of thinking could improve thinking generally.

Computational thinking appears to have been coined in the 1980s by Seymour Papert and popularized in his book *Mindstorms* (Papert, 1980). Papert was a mathematician by training, and his approach was much broader than the algorithmic thinking approach that came before. Papert's approach was empirical and embraced model building, which he implemented using simulated microworlds containing robots (LEGO Mindstorms takes its name from this work). It was revolutionary in its time and received a lot of attention from educators and policy makers of widely different backgrounds.

Unfortunately, today it's very hard to get agreement on what computational thinking is, so definitions tend to be squishy. This is likely due to the widespread use of computers and the tendency for everyone to frame computational thinking in terms of what *they* do with computers. Some want to reduce it to computer literacy, others to basic programming, and yet others to discovery learning with computers, etc.

I take a more unified view of computational thinking based on model building and problem solving. I define computational thinking as building a *runnable* model to solve a problem:

- For an algorithmic problem, this is a model of computation (the original computer science view)
- For data science/statistics, this is a statistical model
- For general scientific fields, this is a scientific model of a phenomenon or process

The model doesn't need to run on a computer, but to be a runnable model, it needs to be mechanistic. One of my favorite examples of a non-computer model is MENACE (Michie, 1963), which plays tic-tac-toe (AKA noughts and crosses). MENACE plays tic-tac-toe using matchboxes full of colored beads as shown in Figure 1.1. Each possible board position (starting with a blank board) is represented by a matchbox, and each move is represented by one of nine colored beads. To make a move, a human assistant selects the correct box for the current board position and randomly samples a colored bead, which determines where MENACE makes its move. If MENACE wins the game, the chosen bead from each box is replaced along with extra beads of the same color, and if MENACE loses, the chosen bead is removed. Over time, these bead adjustments make winning moves more likely and losing moves less likely.

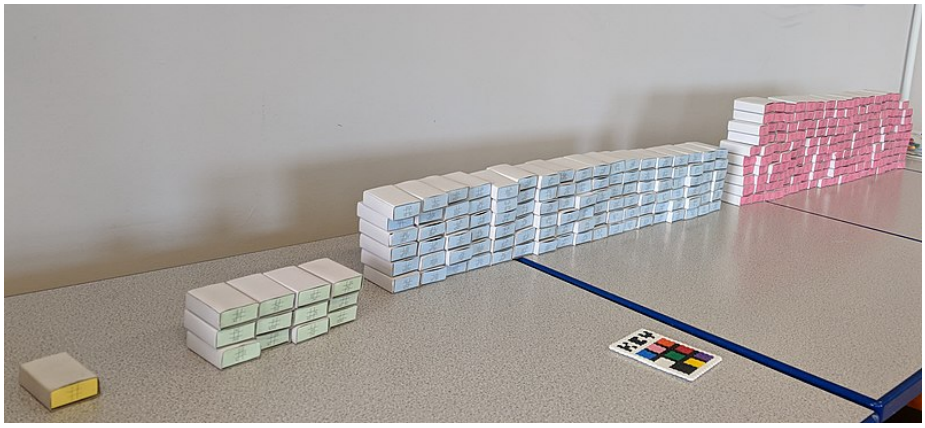

Figure 1.1: Machine Educable Noughts and Crosses Engine (MENACE). Each matchbox corresponds to a possible board position and is full of colored beads corresponding to moves. The color key in the foreground shows the board location indicated by each colored bead. Image © Matthew Scroggs/CC-BY-SA-4.0.

MENACE is a nice example of computational thinking without computers because algorithmic game playing has a long history in computer science and AI. However MENACE is not "an exception to the rule" - teaching computer science without computers has been part of the model curriculum for almost 20 years (Tucker, 2003; Bell, 2021). We really don't need computers for computational thinking!

So how do we *learn* to build runnable models to solve problems (i.e., how do we learn computational thinking)? Well, models are made of interacting elements, so we need to learn those elements, and we need to learn how the elements interact.

Once we know those things, we can customize general problem solving, which has the same basic steps (Polya, 2004):

- Understand the problem
- Make a plan
- Implement the plan
- Evaluate the solution

For any new domain, the big things to learn are the "understand the problem" and the "make a plan" steps of problem solving. That's the approach of this book - for the domain of modular synthesis.

1.3 Modular synthesis

While we can pinpoint the invention of modular synthesis with some precision, it is useful to consider it in a broader context. This section briefly overviews the history of synthesis and how modular fits into it.

Humans have long been interested in musical instruments that incorporate automation or in reproducing sounds by mechanical means. Wind chimes, which play a series of notes when disturbed by wind, appeared in the historical record thousands of years ago. Even before the complete electrification of instruments (synthesizers are electric by definition), there were numerous attempts to partially automate or model sounds, such as barrel organs, player pianos, or speech synthesis using bellows (Dudley and Tarnoczy, 1950) as shown in Figure 1.2.

Consider the difference between wind chimes or a player piano and this speaking machine. Neither of the former is a model of the sound but rather uses mechanical means to trigger the sound (later we will refer to this as sequencing). In contrast, the speaking machine is a well-considered model of the human speech mechanism.

Synthesizers using electricity appeared in the late 19th century.[2] Patents were awarded just a few years apart to Elisha Gray, whose synthesizer comprised simple single note oscillators and transmitted over the telegraph, and Thaddeus Cahill, whose larger Telharmonium could sound like an organ or various wood instruments but weighed 210 tons!

[2]There is some difference of opinion on what qualifies as usage of electricity in this context. For a fuller history of synthesizers, see https://120years.net/wordpress/

Figure 1.2: YouTube video of Wolfgang von Kempelen's speaking machine circa 1780. Image © Fabian Brackhane.

The modular synthesizer was developed by Harald Bode from 1959-1960 (Bode, 1984), and this innovation quickly spread to other electronic music pioneers like Moog and Buchla. The key idea of modular is flexibility. This is achieved by refactoring aspects of synthesis (i.e. functions on sound) into a collection of modules. These modules may then be combined to create a certain sound by patching them together and adjusting module parameters (e.g. by turning knobs or adjusting sliders). An example modular synthesizer is shown in Figure 1.3.

In the 1970s, *semi-modular* synthesizers were developed that did not require patching to make a sound. Instead, semi-modulars were pre-set with an invisible default patch, meaning that the default patch wiring was internal and not visible to the user. Users could then override this default patch by plugging in patch cables. Most semi-modulars from this period also included an integrated keyboard. Arguably, theses changes made semi-modulars more approachable to typical musicians. An example semi-modular synthesizer is shown in Figure 1.4.

Digital technology began replacing the analog technology of synthesizers in the 1980s. As a result, synthesizers got smaller and cheaper. Digital synthesizers made increasing use of preset sounds so that most users never needed to create custom sounds. In comparison to digital synthesizers, modular synthesizers were more expensive and harder to use. An example digital synthesizer is shown in Figure 1.5.

By the 1990s the digital transformation was complete, such that computers could be used to create and produce music in software. Although computers were still relatively expensive at this time, they provided an all-in-one solution that included editing, mixing, and other production aspects. Over the next few

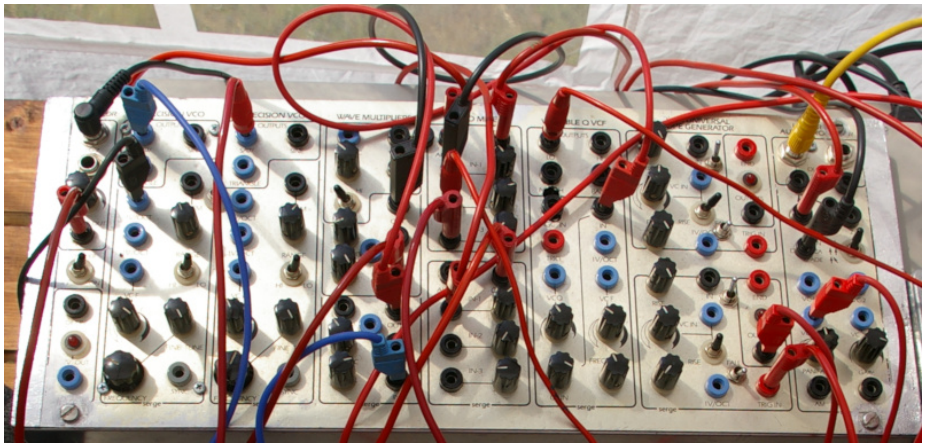

Figure 1.3: A Serge modular system based on a 1970s design. Each module is labeled at the top edge, e.g. `Wave Multiplier`, and extends down to the bottom edge in a column. Note that although the modules have the same height, they have different widths. Image © mikael altemark/CC-BY-2.0.

Figure 1.4: A Minimoog semi-modular system from the 1970s. Patch points are primarily on the top edge and hidden from view. Image public domain.

Figure 1.5: A Yamaha DX7 from the 1980s. Note the menu-based interface and relative lack of controls compared to modular and semi-modular synthesizers. Image public domain.

decades as personal computers and portable computing devices became common household items, the costs associated with computer-based music making became dominated by the cost of software and associated audio and MIDI interfaces. Figure 1.6 shows digital audio workstation (DAW) software commonly used in music production.

Figure 1.6: Logic Pro digital audio workstation software. Additional functionality is provided by 3rd-party plugins showing as additional windows on the screen. In the foreground are an audio interface and a MIDI keyboard used for recording/playing audio and entering note information respectively. Image © Musicianonamission/CC-BY-SA-4.0.

The computer-centric approach dominated synthesis for a decade or more, but by the 2010s, improved electronics manufacturing, smartphone technology, and the open-source movement led to lower cost modular synthesizers. Additionally, the Eurorack standard (Doepfer Musikelektronik, 2022a,b) was widely adopted,

leading to +10,000 interoperable modules.[3] As a result, modular synthesis saw a resurgence in popularity. Figure 1.7 shows a Eurorack modular synthesizer.

Figure 1.7: A Eurorack modular synthesizer. The different modules designs and logos reflect the adoption of the Eurorack standard which makes modules from different manufacturers interoperable. Image © Paul Anthony/CC-BY-SA-4.0.

It is perhaps surprising that some 60 years after its creation, modular synthesis is more popular than ever. One possible reason is the reduction in price over time, shown in Table 1.1. However, other trends seem to be at work. While the modular synthesizer was simplified for wider adoption early in its history, first with semi-modular and later with digital synthesizers, the culmination of this trend led to large preset and sample banks that transformed the task of creating a specific sound to searching for a pre-made sound. It's plausible that as the search for sounds became more intensive, the time savings of presets diminished, making the modular approach more attractive. An intersecting trend is a commonly-expressed dissatisfaction with using computers for every aspect of music making and a corresponding return to hardware instruments, including modular.

Table 1.1: The cost of modular, semi-modular, and computer synthesizers over time. Prices are in 2022 dollars.

Decade	Synthesizer	Cost
1960s	Moog modular synthesiser	$96,000
1970s	Minimoog semi-modular	$10,000
1980s	Yamaha DX7	$6,000
1990s	Gateway computer with Cubase	$8,000
2010s	ALM System Coupe modular	$2,400

[3] https://www.modulargrid.net/

Decade	Synthesizer	Cost
...	VCVRack virtual modular	Free

Earlier in this chapter, I argued that a computational thinking approach to modular could help with other synthesizers and studio production tools. Hopefully this brief history helps explain why: modular represents the building blocks of synthesis that later approaches have appropriated and presented in their own way. A square wave oscillator in modular is fundamentally the same as that in another hardware synth or DAW software. If you understand these building blocks in modular, you should understand them everywhere.

1.4 Moving forward

The next two chapters will focus on "understanding the problem." Chapter 2 reviews both the physics of sound and our perception of it, which perhaps surprisingly, are not the same. From there we move into sounds commonly found in music and their properties, ranging from harmonic sounds to inharmonic sounds like percussion in Chapter 3.

Modular synthesis is properly introduced in Chapter 4 with an overview of modules and how they connect together. This is the foundation of the "make a plan" stage of problem solving. The remainder of the book alternates between learning model elements (modules), how they interact (patches), problem solving (sound design). The progressive *Modules* and *Sound Design* sections build up from basic approaches to the more complex. By the time we're done, you should have a good foundation to create patches to solve new sound design problems.

Part I

Sound

Chapter 2

Physics and Perception of Sound

From the outset, it's important to understand that the physics of sound and how we perceive it are not the same. This is a simple fact of biology. Birds can see ultraviolet, and bats can hear ultrasound; humans can't do either. Dogs have up to 40 times more olfactory receptors than humans and correspondingly have a much keener sense of smell. We can only perceive what our bodies are equipped to perceive.

In addition to the limits of our perception, our bodies also *structure* sensations in ways that don't always align with physics. A good example of this is equal loudness contours. As shown in Figure 2.1, sounds can appear equally loud to humans across frequencies even though the actual sound pressure level (a measure of sound energy) is not constant. In other words, our hearing becomes more sensitive depending on the frequency of the sound.

Why do we need to understand the physics of sound *and* perception of sound? Ultimately we hear the sounds we're going to make, but the process of making those sounds is based in physics. So we need to know how both the physics and perception of sound work, at least a bit.

2.1 Waves

Have you ever noticed a dust particle floating in the air, just randomly wandering around? That random movement is known as Brownian motion, and it was shown by Einstein to be evidence for the existence of atoms - that you can see with your own eyes! The movement is caused by air molecules[1] bombarding the

[1] In what follows, we will ignore that air is a mixture of gases because it is irrelevant to the present discussion.

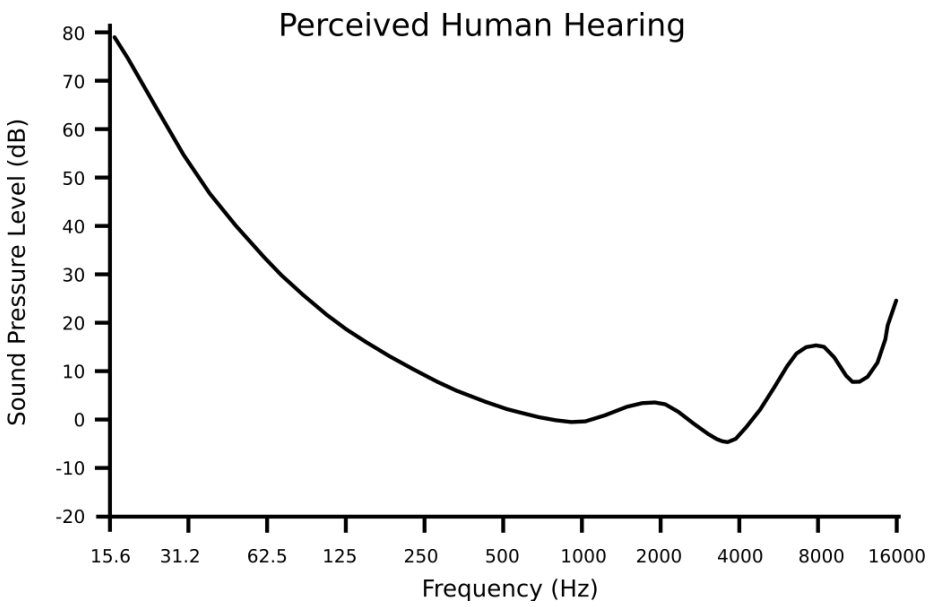

Figure 2.1: An equal loudness contour showing improved sensitivity to frequencies between 500Hz and 4KHz, which approximately matches the range of human speech frequencies. Image public domain.

much larger dust particle from random directions, as shown in Figure 2.2.

Amazingly, it is also possible to see sound waves moving through the air, using a technique called Schlieren photography. Schlieren photography captures differences in air pressure, and sound is just a difference in air pressure that travels as a wave. The animation in Figure 2.3 shows a primary wave of sound corresponding to the explosion of the firecracker in slow motion, and we can see that wave radiate outwards from the explosion.

Let's look at a more musical example, the slow motion drum hit shown in Figure 2.4. After the stick hits the drum head, the head first moves inward and then outward, before repeating the inward/outward cycle. When the drum head moves inward, it creates more room for the surrounding air molecules, so the density of the air next to the drum head decreases (i.e., it becomes less dense, because there is more space for the same amount of air molecules). The decrease in density is called rarefaction. When the drum head moves outward, it creates less room for the surrounding air molecules, so the density of the air next to the drum head increases (i.e., it becomes more dense, because there is less space for the same amount of air molecules). The increase in density is called compression.

You can see an analogous simulation of to the drum hit in Figure 2.5. If you add say 50 particles, grab the handle on the left, and move it to the right,

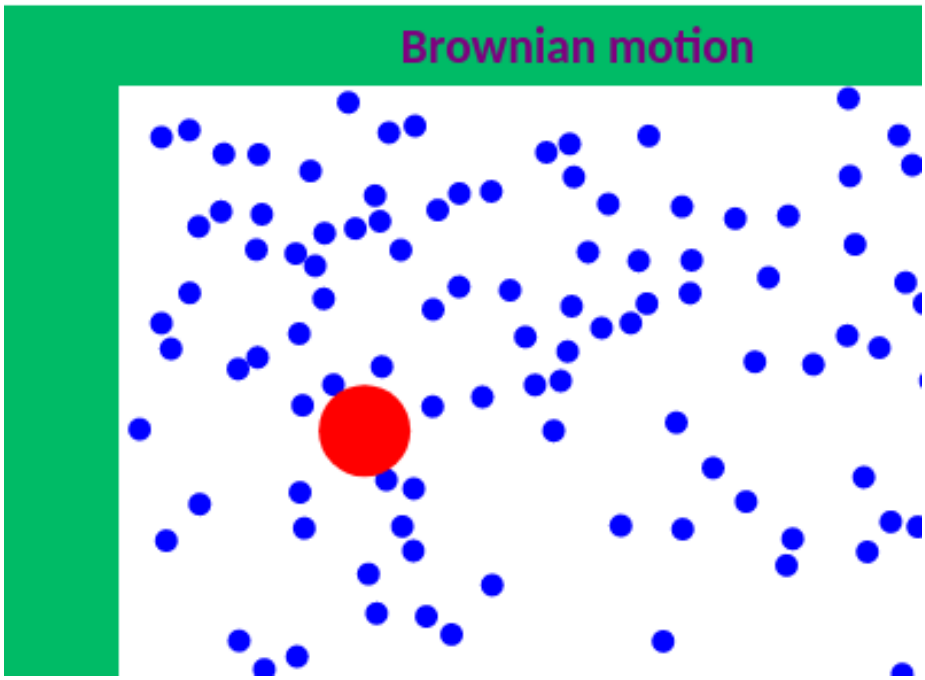

Figure 2.2: Simulation of Brownian motion. Press `Pause` to stop the simulation.

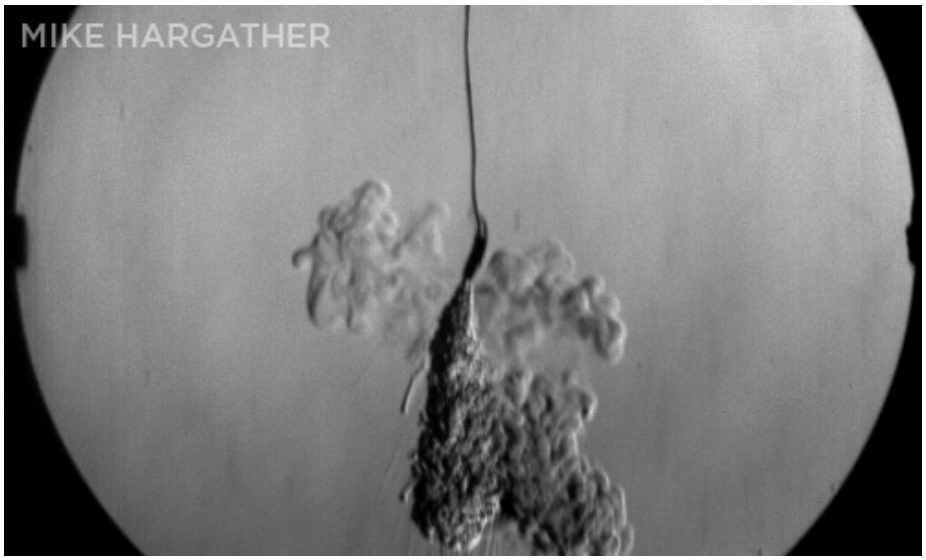

Figure 2.3: Animation of a firecracker exploding in slow motion, captured by Schlieren photography. Note the pressure wave that radiates outward. Image © Mike Hargather. Linked with permission from NPR.

Figure 2.4: Youtube video of a slow motion drum hit. Watch how the drum head continues to move inward and outward after the hit. Image © Boulder Drum Studio.

the volume of the chamber decreases, and the pressure in the chamber goes up (compression). Likewise, if you move the handle to the left, the volume of the chamber increases, and the pressure goes down (rarefaction). In the drum example, when the stick hits the head and causes it to move inward, the volume of air above the head will rush in to fill that space (rarefaction), and when the head moves outward, the volume of air above the head will shrink (compression).

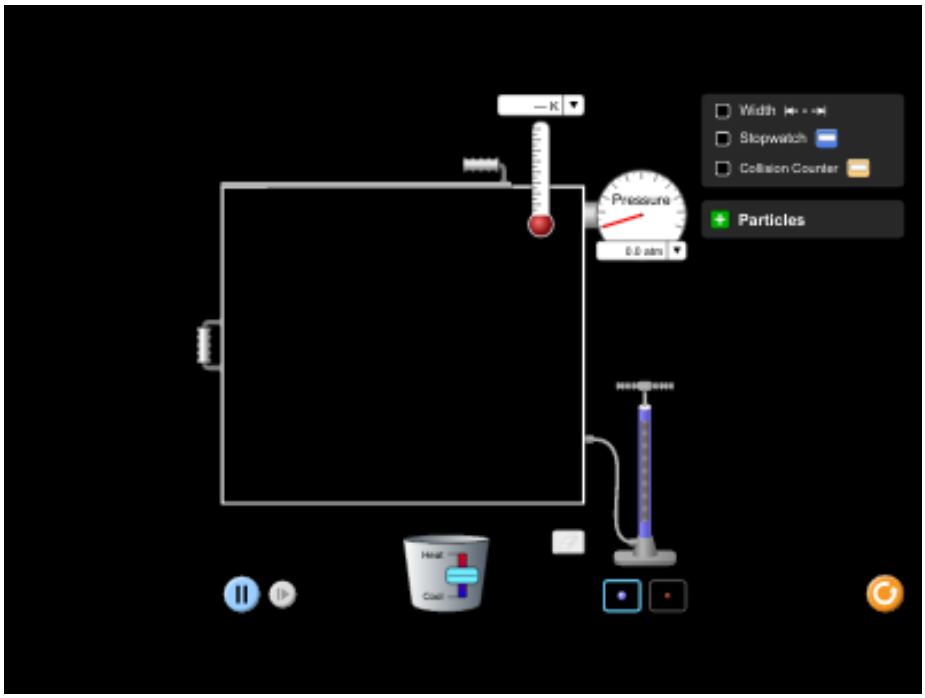

Figure 2.5: Simulation of gas in a chamber. Simulation by PhET Interactive Simulations, University of Colorado Boulder, licensed under CC-BY-4.0.

Sound is a difference in air pressure that travels as a wave through compression and rarefaction. We could see this with the firecracker example because the explosion rapidly heated and expanded the air, creating a pressure wave on the boundary between the surrounding air and the hot air. However, as we've seen with the drum and will discuss in more detail later, musical instruments are designed to create more than a single wave. The Schlieren photography animation in Figure 2.6 is more typical of a musical instrument.

The rings in Figure 2.6 represent compression (light) and rarefaction (dark) stages of the wave. It is important to understand that air molecules aren't moving from the speaker to the left side of the image. Instead, the wave is moving the entire distance, and the air molecules are only moving a little bit as

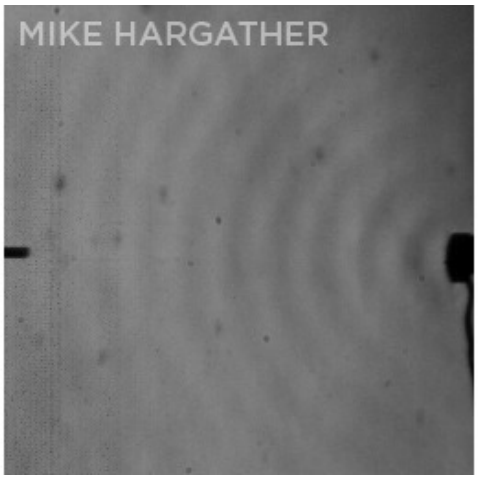

Figure 2.6: Animation of a continuous tone from a speaker in slow motion, captured by Schlieren photography. The resulting sound wave shows as lighter compression and darker rarefaction bands that radiate outward. Image © Mike Hargather. Linked with permission from NPR.

a result of the wave.[2]

To see how this works, take a look at the simulation in Figure 2.7. Hit the green button to start the sound waves and then select the `Particles` radio button. The red dots are markers to help you see how much the air is moving as a result of the wave. As you can see, every red dot is staying in their neighborhood by **moving in opposite directions** as a result of compression and rarefaction cycles. If you select the `Both` radio button, you can see the outlines of waves on top of the air molecules. Note how each red dot is moving back and forth between a white band and a dark band. If you further select the `Graph` checkbox, you will see that the white bands in this simulation correpond to increases in pressure and the black bands correspond to decreases in pressure. This type of graph is commonly used to describe waves, so make sure you feel comfortable with it before moving on.

Now that we've established what sound waves are, let's talk about some important physical properties of sound waves and how we perceive those properties. Most of these properties directly align with the shape of a sound wave.

[2]The air molecules are moving randomly in general, so the simulation shows only the movement attributable to the effect of the wave.

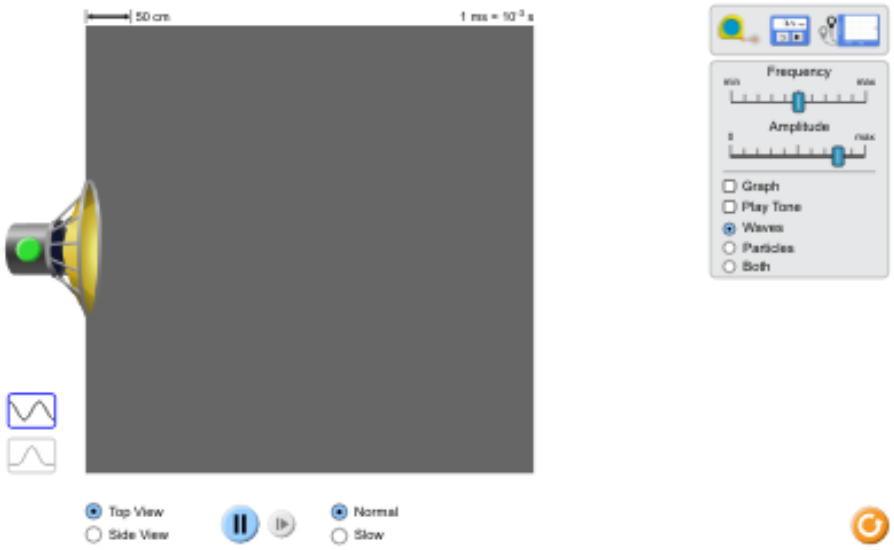

Figure 2.7: Simulation of sound waves. Simulation by PhET Interactive Simulations, University of Colorado Boulder, licensed under CC-BY-4.0.

2.2 Frequency and pitch

Almost all waves we'll talk about are periodic, meaning they repeat themselves over time. If you look at the blue wave in Figure 2.8, you'll notice that it starts at equilibrium pressure (marked as zero[3]), goes positive, hits zero again, and then goes negative before hitting zero at 2 seconds. So at 2 seconds, the blue wave has completed 1 full cycle. Now look at the yellow wave. The end of its first cycle is indicated by the circle marker at .5 seconds. By the 2 second mark, the yellow wave has repeated its cycle 4 times. Because the yellow wave has more cycles than the blue wave in the same amount of time, we say that the yellow wave has a higher frequency, i.e. it repeats its cycle more frequently than the blue wave. The standard unit of frequency is Hertz (Hz), which is the number of cycles per second. So the blue wave is .5 Hz and the yellow wave is 2 Hz.

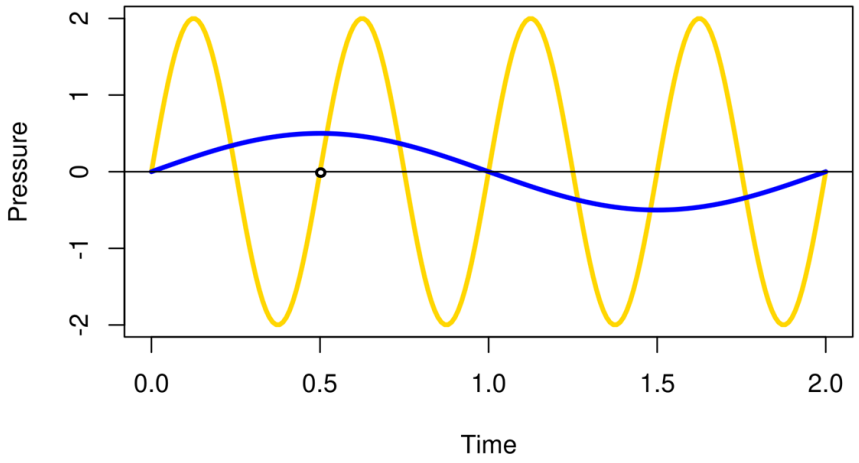

Figure 2.8: Two waves overlaid on the same graph. The yellow wave completes its cycle 4 times in 2 seconds and the blue wave completes its cycle 1 time in 2 seconds, so the frequencies of the waves are 2 Hz and .5 Hz, respectively.

Humans perceive sound wave frequency as pitch. As a sound wave cycles faster, we hear the sound's pitch increase. However, the relationship between frequency and pitch is nonlinear. For example, the pitch A above middle C is 440 Hz[4],

[3] Recall sound is a pressure wave, and it is the change in air pressure we care about. Subtracting out the equilibrium pressure to get zero here makes the positive/negative changes in air pressure easier to see.

[4] Also called A4.

but the A one octave higher is 880 Hz, and the A two octaves higher is 1760 Hz. If you wanted to write an equation for this progression, it would look something like $A_n = 440 * 2^n$, which means the relationship between frequency and pitch is exponential. Figure 2.9 shows the relationship between sound wave frequency and pitch for part of a piano keyboard, together with corresponding white keys and solfège. Notice that the difference in frequencies between the two keys on the far left is about 16 Hz but the difference in frequencies between the two keys on the far right is about 111 Hz. So for low frequencies, the pitches we perceive are closer together in frequency, and for high frequencies, the pitches we perceive are more spread out in frequency.

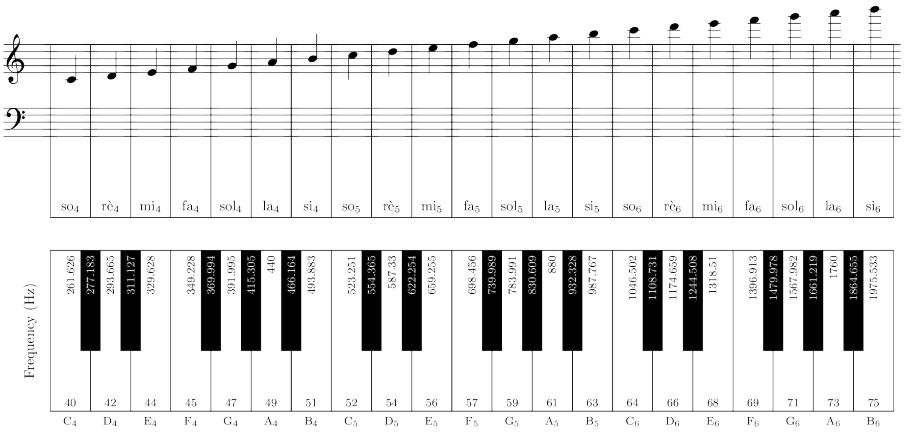

Figure 2.9: Part of an 88 key piano keyboard with frequency of keys in Hz on each key. The corresponding note in musical notation and solfège is arranged above the keys. Zoom in on the image for more detail.

Of course we experience pitch linearly, so the difference in pitch between the two keys in the far left is the same as the difference between the keys on the far right. We can make the relationship linear by taking the logarithm of the frequencies. On the left side of Figure 2.10, we see the exponential relationship between frequency and pitch: as we go higher on the piano keys and pitch increases, the frequencies increase faster, such that the differences in frequencies between keys gets wider. On the right side of Figure 2.10, we see the same piano keys, but we've taken the logarithm of the frequencies, and now the relationship is linear. It turns out that, in general, our perception is logarithmic in nature (this is sometimes called the Weber-Fechner law). Our logarithmic perception of pitch is just one example.

You might be wondering if there's point at which pitches are low enough that the notes run together! It seems the answer to this is that our ability to hear sound at all gives out before this happens. Going back to the 88-key piano keyboard, the two lowest keys (keys 1 & 2; not shown) are about 1.5 Hz appart,

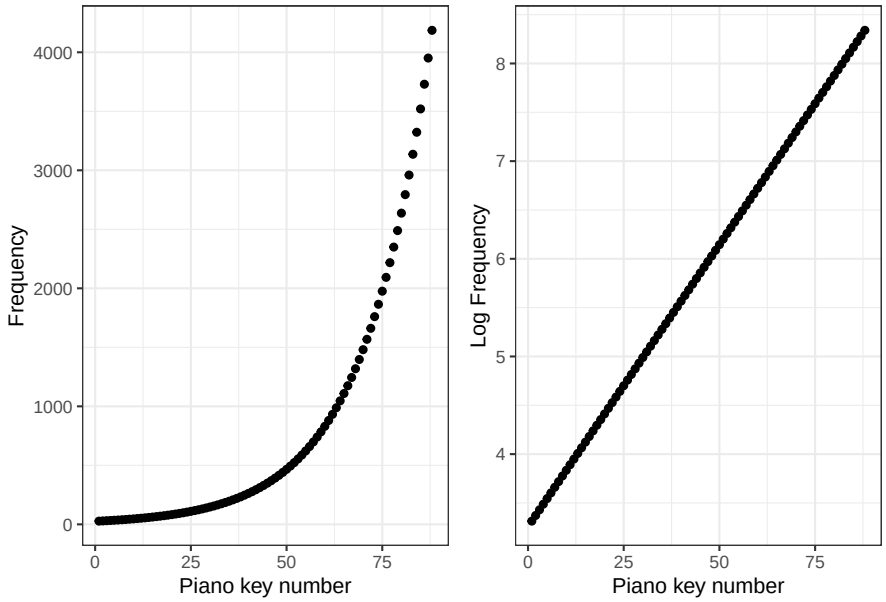

Figure 2.10: (ref:log-freq)

but the lowest key[5] is 27.5 Hz. Humans generally can only hear frequencies between 20 Hz and 20,000 Hz (20 kHz). Below 20 Hz, sounds are felt more than heard (especially if they are loud), and above 20 kHz generally can't be heard at all, though intense sounds at these frequencies can still cause hearing damage.

You may also be curious about the fractional frequencies for pitches besides A. This appears to be largely based on several historical conventions. In brief, western music divides the octave into 12 pitches called semitones based on a system called twelve-tone equal temperment. This is why on a piano, there are 12 white and black keys in an octave - each key represents a semitone. It's possible to divide an octave into more or less than 12 pitches, and some cultures do this. In fact, research suggests that our perception of octaves isn't universal either (Jacoby et al., 2019). We'll discuss why notes an octave apart feel somehow the same in a later section.

2.3 Amplitude and loudness

As discussed, sound is a pressure wave with phases of increasing and decreasing pressure. Take a look at the yellow and blue waves in Figure 2.11. The peak compression of each wave cycle has been marked with a dashed line. For the

[5] Also called A0

yellow wave, the peak positive pressure is 2, and for the blue wave, the peak positive pressure is .5. This peak deviation of a sound wave from equilibrium pressure is called amplitude.[6] It is perhaps not surprising that we perceive larger deviations (with corresponding large positive and negative pressures) as louder sounds.

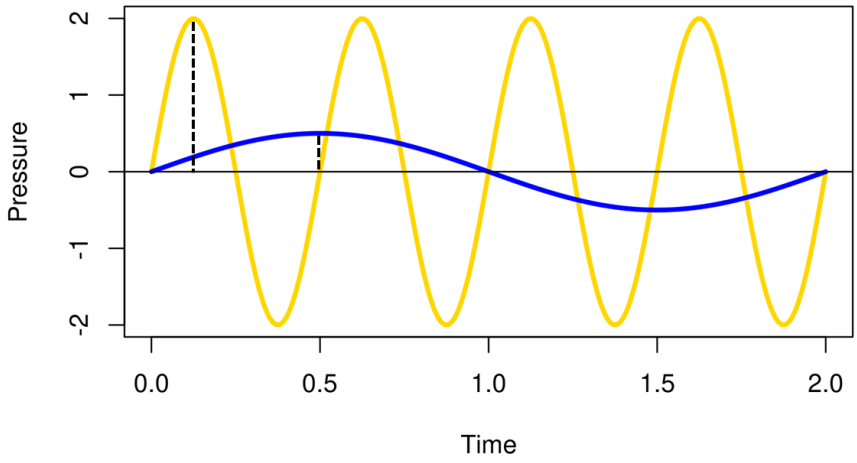

Figure 2.11: Two waves overlaid on the same graph, with a dashed line marking the amplitude of each wave as the deviation from equilibrium.

The relationship between amplitude and loudness is also nonlinear: we hear quiet sounds very well, and sounds must get a lot louder before we perceive them as being louder. In fact, the nonlinear relationship between amplitude and our perception of loudness is *even more extreme* than the relationship for frequency and pitch.

You might have heard of the unit of loudness before, the decibel (dB). Unfortunately, the decibel is a bit harder to understand than Hz, and it use used as a unit of measurement for different ways of expressing the strength of a sound, like sound pressure, sound power, sound intensity, etc. The most important thing you need to understand about decibels is that they are not an absolute measurement, but rather a relative measurement. Therefore, decibels are always based on a reference value. For hearing, that reference value is the quietest sound people can detect, which is defined as 0 dB. Some examples of 0 to 10dB sounds

[6]Note that since the positive and negative pressures are equivalent, amplitude could be measured down from equilibrium to peak negative pressure as well

are a mosquito, breathing, a pin drop, or a leaf hitting the ground.[7]

If we call the reference sound pressure S_0 and the sound pressure we are measuring S, then we calculate the sound pressure level dB of S as $20 * log_{10}(S/S_0)$ dB. Under this definition, a 6 dB increase in sound pressure level means amplitude has doubled: $20 * log_{10}(2) = 6.02$ dB.[8] Since our hearing is quickly damaged at 120 dB, you can see that our range of hearing goes from the quietest sound we can hear (0 dB) to a sound that is 1,000,000,000,000 times more intense (120 dB).

Remember from Figure 2.1 that frequency affects our perception of loudness. As a result, we can't say how loud a person will perceive a random 40 dB sound - not in general. One way of approaching this problem is to choose a standard frequency and define loudness for that frequency. The phon/sone system uses a standard frequency of 1 kHz so that a 10 dB increase in sound pressure level is perceived as twice as loud. This relationship is commonly described as needing 10 violins to sound twice as loud as a single violin. There are alternative ways of weighting dB across a range of frequencies rather than just 1 kHz, so the 10 dB figure should be viewed as an oversimplification, though a useful one. Table 2.1 summarizes the above discussion with useful dB values to remember.

Table 2.1: Useful values for working with dB. All values reflect sound pressure or sound pressure level.

Value	Meaning
0 db	Reference level, e.g. quietest audible sound.
6 db increase	Twice the amplitude
10 db increase	Twice as loud
20 db increase	Ten times the amplitude

2.4 Waveshape and timbre

When we talked about frequency and amplitude, we used the same waveshape in Figures 2.8 and 2.11. This wave shape is called the sine wave. The sine wave is defined by the trigonometric sine function and found throughout physics. There are an unlimited number of waveshapes in principle, but in electronic music you will encounter the sine wave and other waveshapes in Figure 2.12 often because they are relatively easy to produce using analogue circuitry.

[7] These are commonly given examples, but see below for how they are misleading when you take frequency into account.

[8] Note that if we'd used sound intensity level, the corresponding value would be 3 dB. Sound pressure level is more useful in our context: it connects directly to sound wave amplitude, it is measured by a microphone and reflected in microphone output voltage, and it has the same formula as dBV, a decibel measurement of voltage common in audio electronics. Sound intensity level is the square of the sound pressure level.

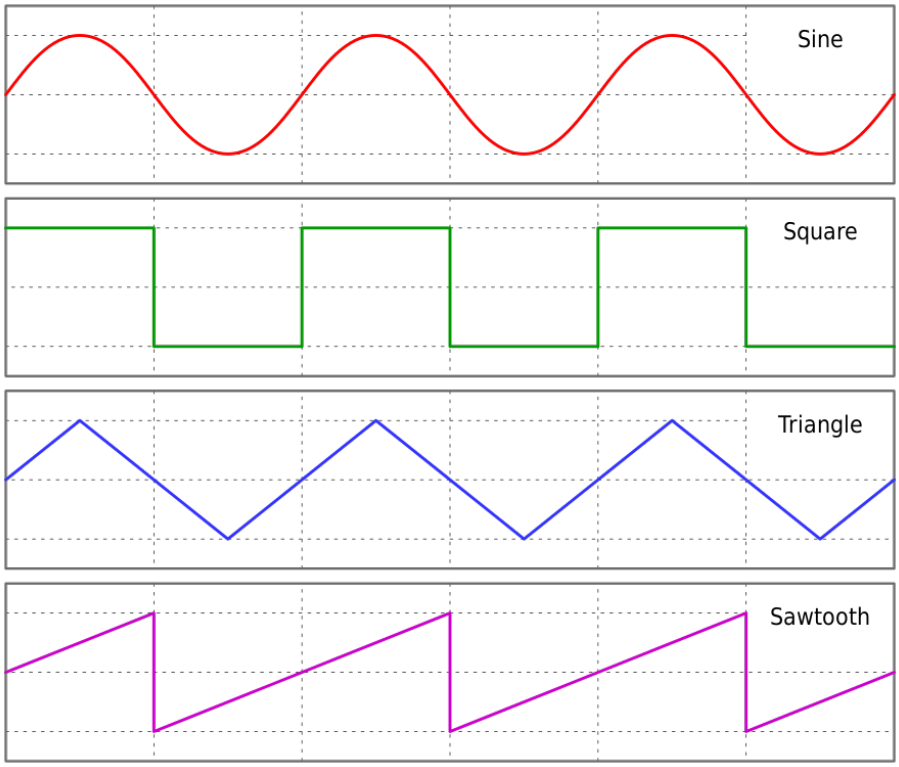

Figure 2.12: The four classic waveshapes in analogue electronic music. Image © Omegatron/CC-BY-SA-3.0.

Perhaps another reason for the widespread use of these waveshapes is that their sounds are rough approximations to real instrument sounds. As we discussed in Section 1.3, the development of electronic music has been strongly influenced by existing instruments. Figure 2.13 presents sounds for each of these waveshapes, together with real instruments that have similar sounds.

As you listen to the waveshape sounds, take a moment to consider their qualities with respect to each other, e.g. how noisy are they and how bright? The differences you're hearing are referred to as timbre or tone color. Each of the waveshape sounds is the same frequency (220 Hz; A3), and the different timbres illustrate how waveforms can have the same frequency but still have their own characteristic sound.

As you listen to the real instrument sounds, consider what about them matches the timbre of the waveshapes and what does not. You may need to listen to some real instruments longer to notice the similarities and differences. For example, when an instrument is played quietly, it may have a different timbre than when it is played loudly. We'll explore these dynamic differences in an upcoming section.

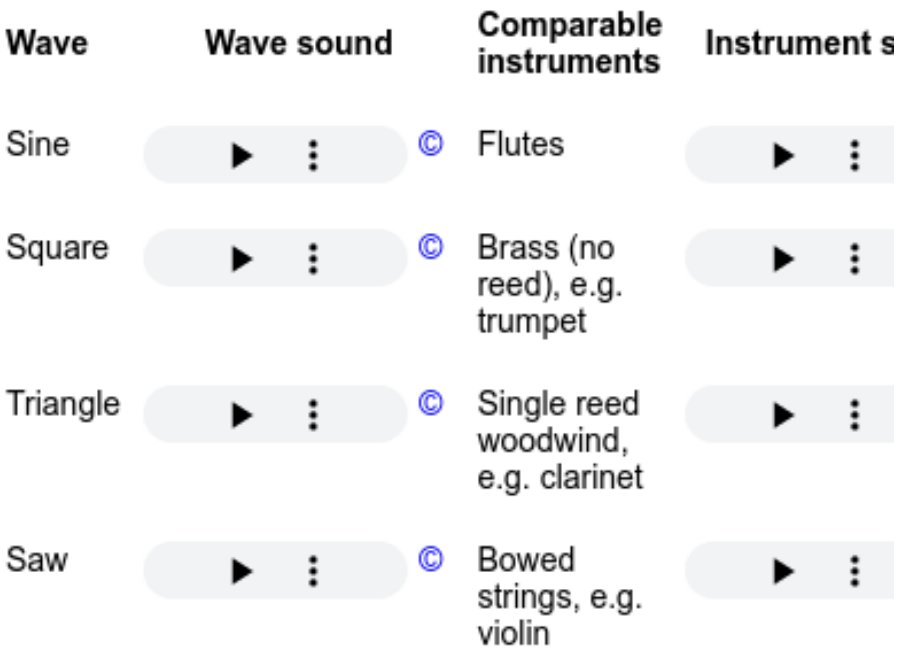

Figure 2.13: Sounds of the four classic waveshapes in electronic music, together with example instruments that make similar sounds.

There is a variation of the square waveform that you will frequently encounter called the pulse wave.[9] In a square wave, the wave is high and low 50% of the time. In a pulse wave, the amount of time the wave is high is variable: this is called the duty cycle. A pulse wave with a duty cycle of 75% is high for 75% of its cycle and low the rest of the time. By changing the duty cycle, pulse waves can morph between different real instrument sounds. At 50% they sound brassy, but at 90%, they sound more like an oboe. Figure 2.14 shows a pulse wave across a range of duty cycles, including 100% and 0%, where it is no longer a wave.

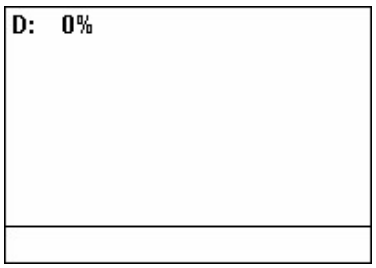

Figure 2.14: Animation of a pulse wave across a range of duty cycles. Note that 100% and 0% it ceases to be a wave. Image public domain.

There are multiple ways of creating waveshapes beyond the four discussed so far. One way is to combine waveshapes together. This is somewhat analogous to mixing paint, e.g. you can mix primary colors red and blue to make purple, and we'll get more precise about how it's done shortly. Alternatively, we can focus on what a wave looks like rather than how we can make it with analogue circuitry. Since waves are many repetitions of a single cycle, we could draw an arbitrary shape for a cycle and just keep repeating that to make a wave - this is the essence of wavetable synthesis and is only practical with digital circuitry.

2.5 Phase and interference

The last important property of sound waves that we'll discuss is not about the shape of the wave but rather the *timing* of the wave. Suppose for a moment that you played the same sine wave out of two speakers. It would be louder, right? Now suppose that you started the sine wave out of one speaker a half cycle later than the other, so that when the first sine wave was in its negative phase, the second sine wave was in its positive phase. What would happen then? These two scenarios are illustrated in Figure 2.15 and are called constructive and destructive interference, respectively. In both cases, the waves combine to create a

[9] One could say the square wave is a special case of the pulse wave. Starting from the circuit or the mathematical definition will give you a different perspective on this; both are true from a certain point of view.

new wave whose amplitude is the sum of the individual wave amplitudes. When the phase-aligned amplitudes are positive, the amplitude of the resulting wave is greater than the individual waves, and when the phase-aligned amplitudes are a mixture of positive and negative, the amplitude of the resulting wave is less than the individual waves.

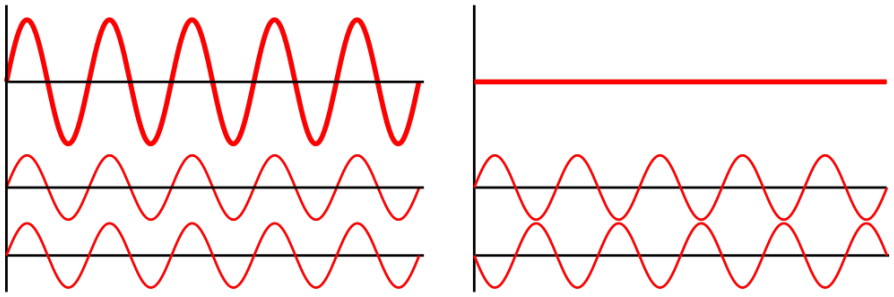

Figure 2.15: Constructive (left) and destructive interference (right). For these matched sine waves, being perfectly in phase or out of phase causes the resulting wave amplitude to be either double or zero, respectively. Image © Haade; Wjh31; Quibik/CC-BY-SA-3.0.

Destructive interference is the principle behind noise cancelling headphones, which produce a sound within the headphones to cancel out the background noise outside the headphones. Figure 2.15 shows how this can be done with with a sine wave using an identical, but perfectly out of phase sine wave. However, being perfectly out of phase is not enough to guarantee cancellation in general. Take a look again at the waveshapes in Figure 2.12. You should find it relatively easy to imagine how the first three would be cancelled by an out of phase copy of themselves. However, the sawtooth wave just doesn't match up the same way, as shown in Figure 2.16.

Figure 2.16 shows how cancellation will only happen if the perfectly out of phase wave is identical to *inverting* the original wave. Inverting a wave means reflecting it on the x-axis so we get a mirror image of the wave.[10] When we add a wave to its mirror image, we achieve perfect cancellation, even for waves like the sawtooth. It just so happens that the first three waves in Figure 2.12 are symmetric, which makes their perfect out of phase and and inverted versions identical.

So what is phase exactly? If we consider the cycle of a wave to go from 0 to 1, then the phase of a wave is the position of the wave on that interval. You've likely seen this concept before in geometry with the sine function, where you can describe one cycle either in degrees (360°) or in radians (2π). The unit is somewhat arbitrary: the important thing to remember is that if two waves are out of sync in their cycles, they will interfere with each other, and we describe

[10] This can be accomplished by multiplying the y coordinates by -1

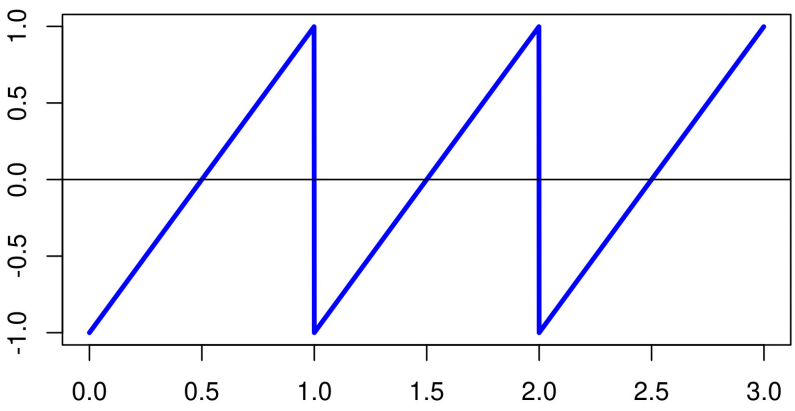

Figure 2.16: Animation of interference from a perfectly out of phase saw wave (yellow) combined with the original wave (blue). The resulting wave (green) has non-zero amplitude because the out of phase wave does not have opposite amplitude to the original wave at all locations. Dotted lines indicate the positions of the original waves.

this difference in cycle position as a difference in phase. The interference caused by phase differences can result in a wide range of effects from cancellation, to the creation of a new wave, to an exact copy of the original wave with double amplitude.

There are a few very practical contexts for phase to keep in mind. The first is that sounds waves reflect, so you don't need two speakers to get phase effects as in the examples above. We'll discuss how phase and reflection are intrinsic to the sounds of many instruments in the next chapter, but any reflective surface can affect phase, which has implications for acoustics in rooms. The second practical context is that you will often want to play multiple notes at the same time, and when you have an electronic instrument, those notes can be triggered very precisely - precisely enough that you have some control over the phase relationships and can use them to shape the sound to your liking.[11]

[11] Any wave property that lends itself to interference can be used similarly, including frequency and amplitude.

Chapter 3

Harmonic and Inharmonic Sounds

We reviewed four common waveshapes in Section 2.4, but we did not explain *why* the waveshapes have their own distinctive timbre. The short answer is that the waveshapes have different harmonics. Understanding the relationship between waveshape and harmonics will be extremely useful to you as you design your own sounds. In order to explain the harmonics behind the different waveshape timbres, we need to understand what harmonics are and how they are created. Not all sounds are harmonic, however. Most percussion sounds are inharmonic, with drums and cymbals as prominent examples.[1] These sounds have a high degree of "noise" associated with them. Finally, traditional instruments producing harmonic and inharmonic sounds have distinctly different dynamics that contribute to their timbre. Harmonic, inharmonic, and dynamic elements all play an important role in shaping sounds.

3.1 Phase reflections, standing waves, and harmonics

When we discussed wave phase and interference in Section 2.5, we mentioned the importance of phase and reflection in musical instruments. Before going any further, ask yourself what makes something a musical instrument? Is a stick hitting a sheet of paper a musical instrument? What about a stick hitting an empty glass?

[1] I prefer to think of these sounds as pitched but inharmonic, but this is not a common view.

A common property of musical instruments is that they make reliable pitches rather than noise. These pitches are created by waves reflecting in the instrument to create standing waves and, in harmonic instruments, get rid of noise. Standing waves are fairly similar and straightforward in strings and pipes, so we'll begin our discussion with strings, where the waves are easily visible.

Standing waves are created when two waves moving in opposite directions interfere with each other to create a new wave that appears to remain in place (i.e. it "stands" rather than moves). In a string, the two waves are created by an initial wave that reflects back from the ends of the string, which creates an out of phase wave. Figure 3.1 shows a simulation of a reflected wave. Select the checkbox for `Pulse` and move `Damping` to `None`, then press the green button to initiate a pulse. As you can see, when the pulse reaches a fixed end, it reflects both in direction and in phase, i.e. if it is above the line going right, it will flip below the line and go left when it hits a fixed end.

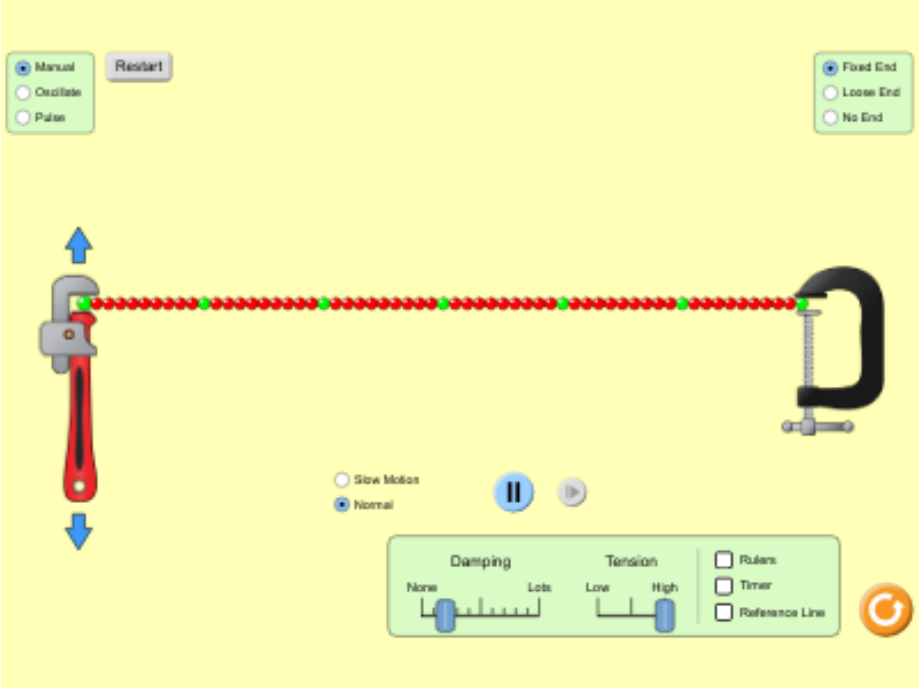

Figure 3.1: Simulation of waves on a string. Simulation by PhET Interactive Simulations, University of Colorado Boulder, licensed under CC-BY-4.0.

You can use the same simulation to make a standing wave. Set `Amplitude` to .20, `Frequency` to .17, `Damping` to `None`, `Tension` to `Low`, and select `Oscillate`. These settings will send a wave from the left to interfere with the reflected wave coming back from the right. You will very quickly see a standing wave with

three nodes, indicated by the green beads. Two of the nodes are at either end of the string, and the third node is exactly in the middle. This standing wave is the 2nd harmonic. You can make the first harmonic (commonly called the fundamental frequency) by reducing `Frequency` to half, $.17/2 = .085 \approx .09$ and hitting restart. Observe that the only nodes are the two ends of the string. Finally, you can create additional harmonics by multiplying the frequency of the fundamental by integers greater than 1. Try $.085 * 3 = .255 \approx .26$, $.085 * 4 = 34$, $.085 * 5 = .425 \approx .43$, etc. Note the green beads don't always mark nodes, which by definition don't move; instead the green beads sometimes mark antinodes, or places of greatest motion.

For the odd harmonics, the simulation requires us to do some rounding that results in an imperfect standing wave. Eventually these imperfect standing waves will cancel out because they are not true standing waves. True standing waves on our string model have frequencies that are perfectly aligned with the length of our string. When this happens, the two waves that create the standing wave constructively interfere with each other, i.e. they are self-reinforcing. We can see this in the simulation when we don't round the frequencies because the amplitude of the waves keeps increasing over time as we add more energy into the system through the oscillator on the left hand side. Any wave whose frequency does not create a standing wave will eventually cancel itself out even if damping is zero.

Standing waves on a string explain why harmonics are integer multiples of the fundamental frequency. In the simulation for the first harmonic, the oscillator started its second pulse at the moment the first reflected pulse returned to the oscillator, so the first pulse had to travel twice the length of the string. For the second harmonic, the oscillator started its second pulse at the moment the first pulse reached the fixed end, or the full length of the string. In each case, the combined speed of the waves must evenly divide the length of the string (giving an integer) in order for the out of phase passing waves to align and create a standing wave. The integer relationship of harmonics is very important to how we perceive musical instruments and may even be the reason for our perception of fundamental musical relationships like octaves and fifths. For example, recall from Section 2.2 than one octave above a pitch is double the frequency of that pitch. Since the second harmonic is double the frequency of the fundamental, the second harmonic is one octave above the fundamental frequency. Similarly, the third harmonic is one fifth above the second harmonic, and the fifth harmonic is a major third over the fourth harmonic.[2]

Before moving on, it's important to note that the oscillator simulation in Figure 3.1 does not accurately represent what happens in a plucked string because the simulation doesn't fix both sides of the string. Instead, the simulation uses an oscillator on one side to generate sine waves. We can, however, observe a very

[2]One might speculate that we created instruments in order to consistently create pitches based on the fundamental, but the instruments instilled in us a music theory as a side effect of their harmonics.

similar behavior to the simulation when a string is plucked, as shown the slow motion video in Figure 3.2. In the video, the tap on the string (a reverse of a pluck) creates two pulses that move in opposite directions, reflect off their respective ends of the string and switch phase, and then constructively interfere in the center to create an apparent standing wave.

Figure 3.2: Youtube video of a slow motion tap on a long string. Watch how the tap creates two pulses that reflect off their respective ends of the string, switching phase, and then constructively interfere to create an apparent standing wave. Image © Kemp Strings.

Although Figure 3.2 looks straightforward and might lead us to believe that the differences between the simulation and plucking a string are superficial, the true story is more complicated. A real string pluck does not create a single standing wave but rather a stack of standing waves happening all at once. This is because, unlike the simulation in 3.1, the wave created by plucking a string is not a sine wave but has an initial shape more like a triangle. In addition, a string pluck is highly likely not to occur in the middle of the string. These differences mean that when a string is plucked, waves of all different frequencies are created and begin racing back and forth on the string. Those that correspond to harmonic frequencies are sustained longer and create a tone. The remaining frequencies are known as transients and quickly cancel out.[3]

[3]Transients are commonly modeled using noise in electronic music in order to make a simulated sound more realistic.

3.2 Resonators, formants, and frequency spectrum

In a real string instrument, the vibrations of the strings are transmitted to the rest of the instrument, which includes a resonator that is typically shaped like a box with an opening. The resonator vibrates at the same frequencies as the string (primarily the harmonic frequencies as discussed) but pushes against a larger volume of air than the string, which has a small surface area by comparison. Therefore the resonator creates a larger change in air pressure for a higher amplitude (and louder) sound wave. The resonator does not amplify the string frequencies perfectly, however. Some frequencies are better amplified than others, and this means that the resonator affects the timbre of the instrument. This is why two guitars with different resonators will sound different even if they have identical strings. The effect a resonator has on a frequency's amplitude is called Q, and the relative strengths of frequencies emitted by a resonator are called formants.[4]

You can probably imagine how differences in the construction and operation of other instruments might lead to the differences in their characteristic sounds. Their mode of operation (string, wind, etc.) and their resonators (wood, metal, etc.) affect both what harmonics are produced and the relative strengths of these frequencies (the formants). For example, closed end pipes produce only odd harmonics! In each case, the instrument is producing multiple harmonic frequencies at once, with some instruments producing more harmonics than others. These differences in characteristic sounds are reflected in the four waveshapes presented in Section 2.4.

Unfortunately, when we look at a waveshape, we see the sum of all the harmonic frequencies - we can't see the individual harmonics just by looking at a waveshape. Wouldn't it be nice if we could somehow see all the harmonics that make up a waveshape? It turns out we can decompose any complex waveshape into components using a technique called Fourier analysis. Each component extracted by Fourier analysis is a sine wave called a partial, and we can reconstruct a complex waveshape by adding the sine wave partials together (potentially an infinite number of them). When the waveshape is from a harmonic instrument, the partials are harmonics, so we can use Fourier analysis to see all the harmonics in a waveshape.

Figure 3.3 is a simulation showing how Fourier analysis can use sine waves to approximate more complex waveshapes. The first waveshape is a sine, which is quite trivially approximated by a single sine wave. Take a moment to look at the amplitude of the red line (the first harmonic) and how it corresponds to the red bar in the upper bar plot. You can grab the top of that bar and move it up and down to change the amplitude of the first harmonic. As you move the

[4] You will often see the word "formants" applied to human speech, where the resonator is the vocal tract, but it also applies to instruments with resonators.

bar, note how the bottom graph showing the sum of harmonics changes exactly the same way - this is because it is the sum of harmonics, and we only have one harmonic, so the sum is equal to that harmonic.

Use the dropdown on the right to select a triangle waveshape. Note that all the harmonics are odd for the triangle, and that the sum is clearly different from the sine wave of the first harmonic. It turns out you can't get any closer to a triangle waveshape with only 11 harmonics, but feel free to try by moving the sliders around. What you will find is that as you try to change the shape at one part of the waveshape, you end up making changes everywhere. This is because the sine waves that are being summed up keep going up and down everywhere. The only way to get closer to a triangle shape is to use more and smaller sine waves. You can see what this looks like by using the `Infinite Harmonics` checkbox on the bottom. This illustrates a general principle of Fourier analysis: sharp edges in a waveshape mean high frequency partials.

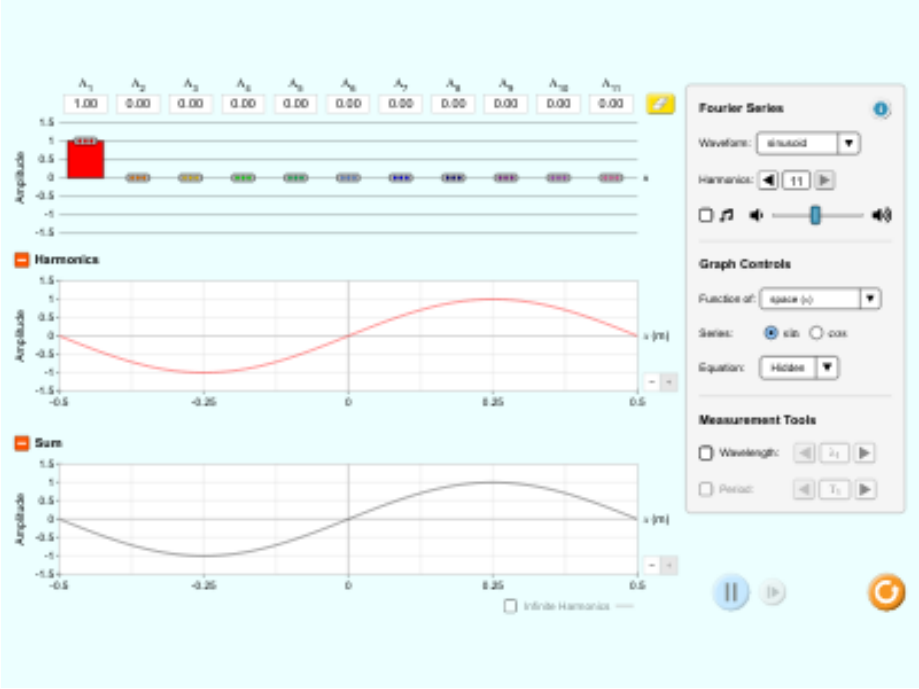

Figure 3.3: Simulation showing how Fourier analysis can approximate a complex wave using sine waves. Simulation by PhET Interactive Simulations, University of Colorado Boulder, licensed under CC-BY-4.0.

The square and sawtooth waveshapes give more impressive examples of this. If you select square, you'll see that again only odd harmonics are used, but you'll also see that the sum of harmonics is pretty far from the square waveshape

we looked at before. The sawtooth waveshape, which uses both even and odd harmonics (with opposite signs), also looks pretty different from the sawtooth waveshape we saw before. Both of these waveshapes have sharp edges that need more harmonics to approximate. They also have straight regions that don't line up well with the straight-ish part of the first harmonic, and these straight regions also need more high frequency components to straighten out. In both cases, you can check infinite harmonics to see how additional harmonics would help.

It may have already occurred to you that you could create any sound by adding together the sine waves in the right combinations. This is exactly what additive synthesis does! Running a Fourier analysis to get sine wave partials of a sound and then recombine them to reproduce the sound is very appealing. However, even though the idea of additive synthesis has been around a long time, it was not practical with analogue technology because of the many oscillators and precise timings involved. Conceptually, the alternative to additive synthesis is subtractive synthesis, which has been a very popular approach in analogue synthesis to the present day. Subtractive synthesis starts with complex waveforms and then removes harmonics to create the desired sound. Harmonics can be removed with relatively simple analogue electronics as we'll discuss in a later chapter.

While the simulation in Figure 3.3 is useful for understanding how Fourier analysis works, it's difficult to see all of the frequenecy components because they are stacked on top of each other. An alternative way of visualizing a Fourier analysis is a frequency spectrogram. A frequency spectrogram shows each sine wave based on its frequency and amplitude. Figure 3.4 shows a frequency spectrogram of the same four waveshapes at 1 Hz with harmonics side by side and amplitudes normalized so all harmonics sum to 1. The order of harmonics at 1 Hz shows how much energy each waveshape has in the first harmonic, the order of harmonics at 2 Hz shows the energy at the second harmonic, etc.[5]

Figure 3.4 clearly shows what we noted before: only sawtooth has even harmonics, and only sawtooth and square have easily visible harmonics above the third harmonic. It's quite amazing that these waves have such small visible differences in their frequency spectrums and yet have such distinctive sounds. For all waveshapes, the fundamental has the most energy (highest amplitude). This is crucial to our perception of the overall pitch of the sound. If you go back to Figure 3.3 and increase the amplitude of a harmonic above the amplitude of the harmonic (and click the checkbox to hear the sound), your perception of the pitch will shift to the louder harmonic.

Are all sounds actually made out of sine waves, or is Fourier analysis only an approximation of sounds? When we talk about sounds produced using standing waves, harmonic motion tells us the sound waves are sine waves. Therefore

[5] 1 Hz is a convenient fundamental frequency to get 10 harmonics on a compact plot, but we'd see the same pattern regardless of the fundamental frequency.

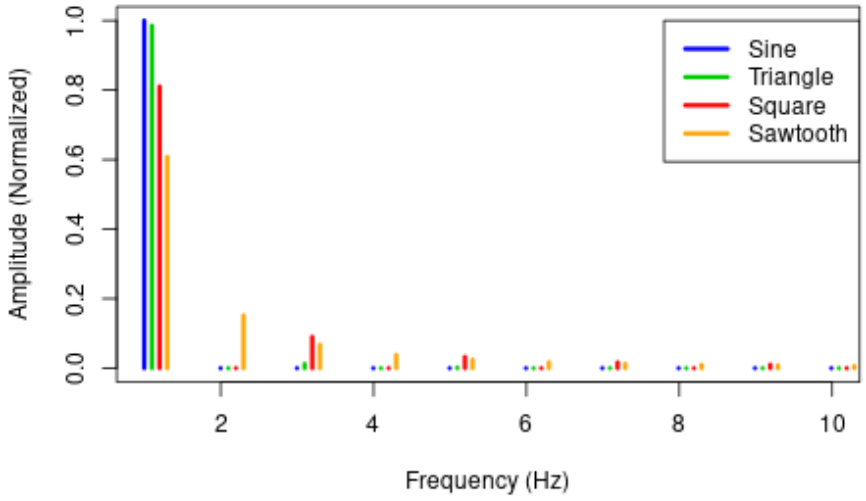

Figure 3.4: Frequency spectrum of four basic waveshapes at 1 Hz. Amplitude is normalized so all harmonics sum to 1. Harmonics are offset for comparison.

when Fourier analysis is applied to these types of sounds, its solution closely corresponds to how the sounds were generated. However, when the sounds are produced by other means, Fourier analysis no longer corresponds to how the sounds were generated, even though it may approximate them arbitrarily. These distinctions perhaps don't matter when we are concerned with listening to sound, since our auditory system decomposes complex sounds into frequency bands (Oxenham, 2018) analogous to Fourier analysis, regardless of how the sound was produced.

3.3 Inharmonic standing waves and noise

While standing waves in one dimension, like a string, are necessarily harmonic, standing waves in two dimensions are typically not harmonic. Inharmonic means that the frequencies of the standing waves are not integer multiples of the fundamental. Two-dimensional standing waves are common in drums, cymbals, and related percussion instruments, where they are called modes, and the physics behind the acoustics of these instruments can get very complex. Recall that standing waves in a string had nodes which were points of no movement. In a circular membrane (a drum) or a circular plate (a cymbal) the nodes are lines and circles, because the waves can now travel in two dimensions and reflect off

the edges of the vibrating surface. Some modes of an ideal clamped vibrating membrane are shown in Figure 3.5. Each of these modes has 0, 1, or 2 nodal lines across the diameter of the membrane and a nodal circle around the perimeter of the membrane. Modes are usually denoted in pairs (d, c) where d is the number of nodal lines across the diameter and c is the number of nodal circles.

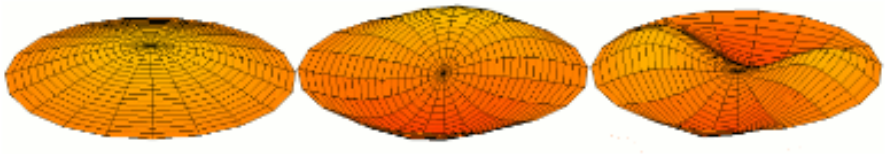

Figure 3.5: Animations of membrane vibration modes with a single nodal circle and 0, 1, or 2 nodal lines. Images public domain.

As with standing waves on a string, when a drum or cymbal is struck, infinitely many modes are excited all at once. Some of these modes are very efficient in transferring energy to the air while others are less efficient, which causes the frequencies of the modes to be relatively short, or longer lived, respectively. The fundamental mode, in particular, is so efficient at transferring energy to the air that it quickly disappates energy and dies out, which means that in these instruments, the fundamental frequency is very weak compared to harmonic instruments.

In general, the modes have inharmonic frequency relationships to each other as shown in Table 3.1. The timpani is a notable counterexample where the kettle and the style of playing enhance the $(d, 1)$ modes, a few of which in the timpani have harmonic relationships to each other. Cymbals similarly have inharmonic frequency relationships but differ in modes and spread of ratios across modes

(Leissa, 1969).

Table 3.1: Modes of an ideal clamped vibrating membrane like a drum head and their relative frequencies with respect to (0,1). Note the ratio as not integers and so the series is inharmonic.

Mode	Frequency ratio
(0,1)	1
(1,1)	1.594
(2,1)	2.136
(0,2)	2.296
(3,1)	2.653
(1,2)	2.918
(4,1)	3.156
(2,2)	3.501
(0,3)	3.600
(5,1)	3.652

Harmonics are defined as integer multiples of the fundamental frequency, and the frequency relationships in Table 3.1 are clearly not integers. Note also that they are much more closely packed together than integers: there are 10 modes with a frequency ratio below 4, whereas in a harmonic series we'd only expect four standing waves in that space. Because the frequency relationships are inharmonic, we can only call the standing wave frequencies of such instruments partials - they are not harmonics.[6]

Because the partials of percussion instruments are dense, their relationships inharmonic, and the fundamental weak, percussion instruments are commonly approximated in electronic music using some form of noise. Noise is a random mixture of frequencies, so by definition it does not have harmonic relationships or a fundamental, and the respective frequencies are very close together. For percussion instruments that have more pitch to them, one can add a simple wave, like a sine wave, to convey the sense of a fundamental.

Various types of noise have been defined with different acoustic properties. The noise types have color names that may be helpful for remembering which is which, though in some cases the color names seem a bit arbitrary. The first and most commonly though of noise is white noise. You can see this kind of noise on an old television set (the "snow" on channel with no broadcast). White noise has equal energy across its frequency spectrum. Since we perceive each doubling of frequency as an octave increase in pitch, this means that high frequency sounds are more prominent in white noise. All other colors of noise are based on white noise but change the distribution of energy in the frequency spectrum in some way, i.e. increase the amplitude of some frequencies and decrease the amplitude of others, as shown in Figure 3.6.

[6] They are not overtones either. An overtone is a harmonic above the fundamental.

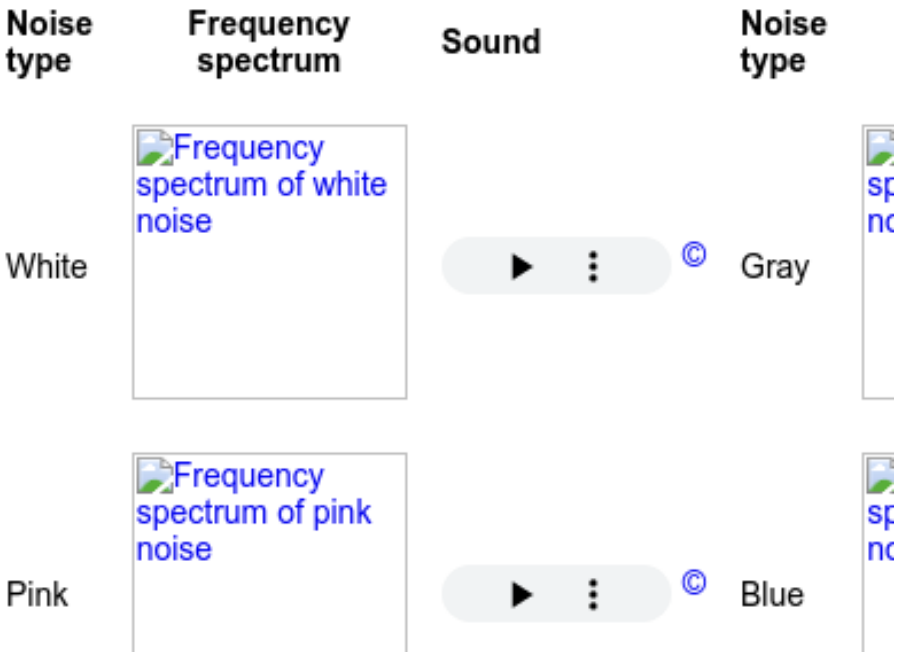

Figure 3.6: Frequency spectrum and sound of various "colors" of noise.

Pink noise balances energy across each octave to *roughly* approximate human perception by reducing energy (i.e., amplitude) as frequency increases. Brown noise (also called red noise) is like pink noise but reduces energy more quickly as frequency increases, which puts more energy in lower octaves. Blue noise is the opposite of pink noise: it increases energy as frequency increases by the same amount that pink noise reduced energy. Similarly, violet noise (also called purple noise) is the opposite of brown noise, and increases energy with frequency by the same amount. Finally, grey noise is based on psychoacoustics and places energy on a curve that might seem familiar - compare it to the equal loudness contour in Figure 2.1. Grey noise thus takes into account that humans hear some frequencies better than others, and allocates energy so that all frequencies sound equally loud. One might consider grey noise the the evolution of pink noise. Both take human perception into account, but grey noise does so in a more nuanced way than pink noise.

3.4 Dynamics and Envelopes

Up to this point, the discussion has focused on frequencies of sound and their relative strengths, with only occasional reference to how sound unfolds over time. However, the way a sound unfolds over time plays a critical role in its timbre. For example, think back to a time when you heard a sound played backwards. That reversed sound had the exact same frequencies and distribution of energy as the original, but the reversed version probably sounded pretty bizarre. How sounds unfold over time is sometimes referred to as dynamics in music and envelopes in physics. Our focus is on the sound of a single instrument over time, so is aligned with timbral dynamics and envelopes.

The basic concept of an envelope is that the amplitude of a sound changes over time. Ideally we'd model this change in sound with a complex curve that goes from zero, up for a time, and then back down to zero for each instrument. Such curves would be fairly complicated for different instruments and would clearly vary with how hard the instrument was played, e.g. how hard a string was plucked or a drum hit. To simplify matters, early developers of electronic music settled on envelopes with discrete stages: attack, decay, sustain, release (ADSR). All of the stages except sustain are based on time, as shown in Figure 3.7. Attack is the time it takes for an instrument's sound to reach peak amplitude, decay is the time it takes to decrease from peak amplitude to the next stage or zero, and release is the time it takes to decrease from sustain to zero. None of these set amplitude levels - they only determine how fast amplitude levels are reached (max, sustain, and zero, respectively). Sustain, on the other hand, is an amplitude level rather than a time. The duration of sustain is as long as the stage is held, e.g. a finger on a key. In this way, and ADSR envelope is a simple, yet fairly flexible model of a physical envelope.

The full ADSR envelope makes the most sense on a keyboard instrument where

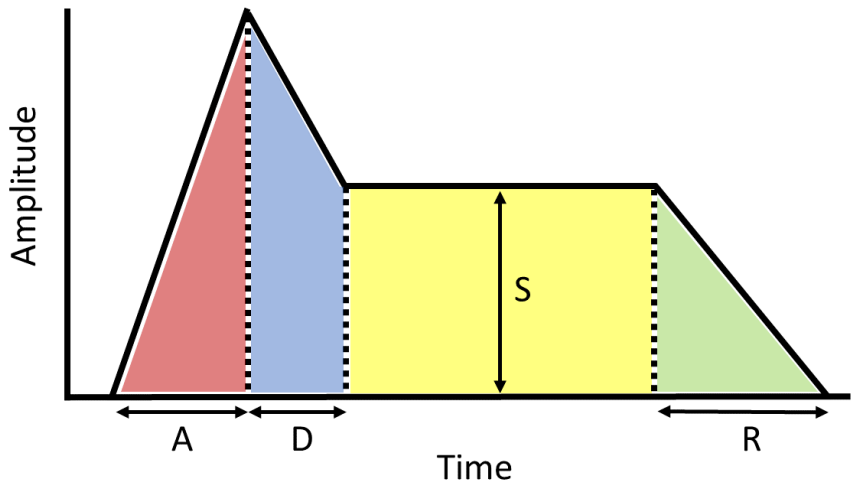

Figure 3.7: An example Attack-Decay-Sustain-Release (ADSR) envelope. Sustain ends with manual control and is the only parameter that sets amplitude level. All other stage lengths are controlled by time parameters as indicated.

pressing a key begins the attack for its duration, which then gives way to decay for its duration. Sustain is then held as long as the key is held, and release begins as soon as the finger leaves the key and lasts for its duration before the amplitude returns to zero. Clearly the full ADSR envelope does not make sense for other instruments. For example, a drum only needs AD, as does a plucked string. Examples of sounds shaped by envelopes are given in Figure 3.8. While the examples in Figure 3.8 only use envelopes for amplitude, envelopes are commonly used to control other properties that change over time. One example is the brightness of a string when it is first plucked, followed by a mellowing of the tone as the higher frequencies disappear. This effect can be created by using an envelope on a filter, a technique we will cover in a future chapter.

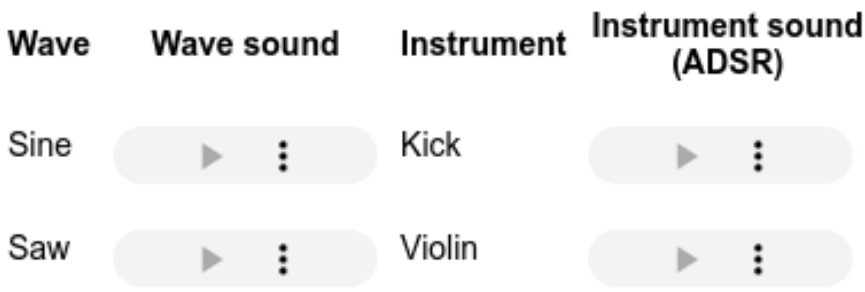

Figure 3.8: Sounds of basic sound waves shaped by envelopes. The kick has fast attack and decay, and the violin has relatively slow attack, decay, and release.

Part II

Fundamental Modules

Chapter 4

Basic Modeling Concepts

Chapters 2 and 3 focused on the "understand the problem" stage of problem solving, and they introduced both the basic concepts and terminology of modeling sound. The present chapter pivots to the make/implement a plan stages of problem solving by introducing the model elements and how they interact. Since we are building models for modular synthesis, the model elements are the modules, and their interactions are driven by how they are connected together in a patch. Figure 4.1 shows an example patch on a real modular synthesizer from Chapter 1.

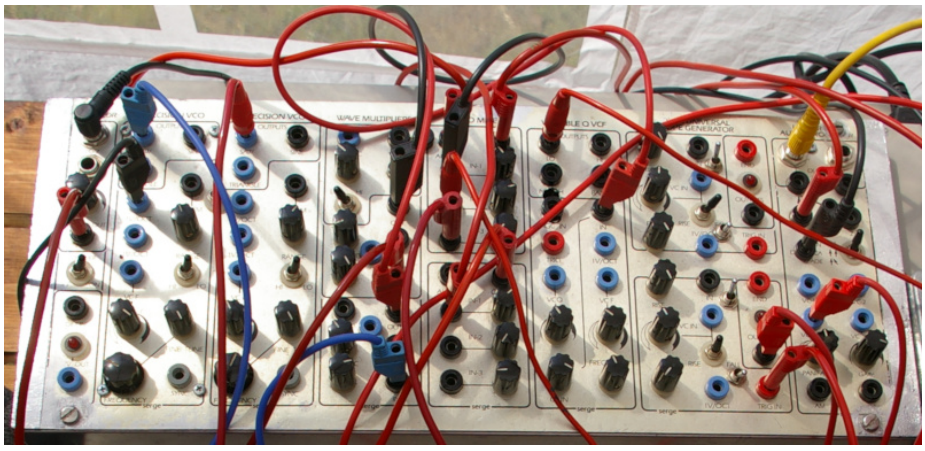

Figure 4.1: A Serge modular system based on a 1970s design. Each module is labeled at the top edge, e.g. `Wave Multiplier`, and extends down to the bottom edge in a column. Note that although the modules have the same height, they have different widths. Image © mikael altemark/CC-BY-2.0.

Here and throughout the book, we will use open source modular software called

VCVRack (VCV, 2022) that has been ported to the web (Car, 2022). This version is integrated with the the web version of the book so you can read about modular and solve sound design problems within the same environment. VCVRack is widely used in practice and has emulations of many hardware modules, so any details you learn about specific modules here could be useful down the road.

4.1 Modules are the model elements

The universe of modules is rather vast, which can make learning about modules overwhelming. For example, ModularGrid lists over 10,000 modules and classifies them according to 56 categories based on their function. While those 56 categories are certainly real and useful, you can get the main idea with only three categories: controllers, generators, and modifiers. You might think of these as "categories of categories."

Controllers initiate musical events. An example controller is a module that waits for key presses and, on receiving them, sends a signal to initiate a musical event. Another example is a sequencer, which you can think of as way of emulating timed key presses to initiate musical events.

Generators create audible sound. An example generator is an oscillator that generates one or more of the four basic waveshapes discussed in Section 2.4. Another example is a noise generator that generates one or more colors of noise described in Section 3.3.

Modifiers modify incoming signals that may be either audio or some kind of control signal. An example audio modifier is a reverb, which adds diffuse echoes to sound, giving it the feel of being played in a room. An example control signal modifier is an envelope, which we can use to control the amplitude of a sound over time, as discussed in Section 3.4.

4.2 Signals are how the model elements interact

Modules interact with each other through signals sent via patch cables. Starting with the signals, we can broadly separate them into two types: audio and control signals. Audio signals are a voltage representation of sound. Recall that sound is a pressure wave with high and low phases of pressure. It's perhaps not surprising that audio signals use corresponding positive and negative voltage to represent a sound wave. This kind of voltage is AC voltage. Anytime a signal is bipoloar, i.e. it crosses zero, you can assume the voltage is AC.

Control signals represent everything besides audio. Unlike audio signals, control signals are unipolar voltage representations, i.e. they don't cross zero, and

thus use DC voltage.[1] There are two main types of control signals in modular synthesis. Both of these can be understood in terms of light switches as shown in Figure 4.2. Regular light switches are either on or off. In terms of voltage, they are either at minimum voltage or maximum voltage. Dimmer switches in contrast can smoothly adjust voltage between their minimum and maximum.

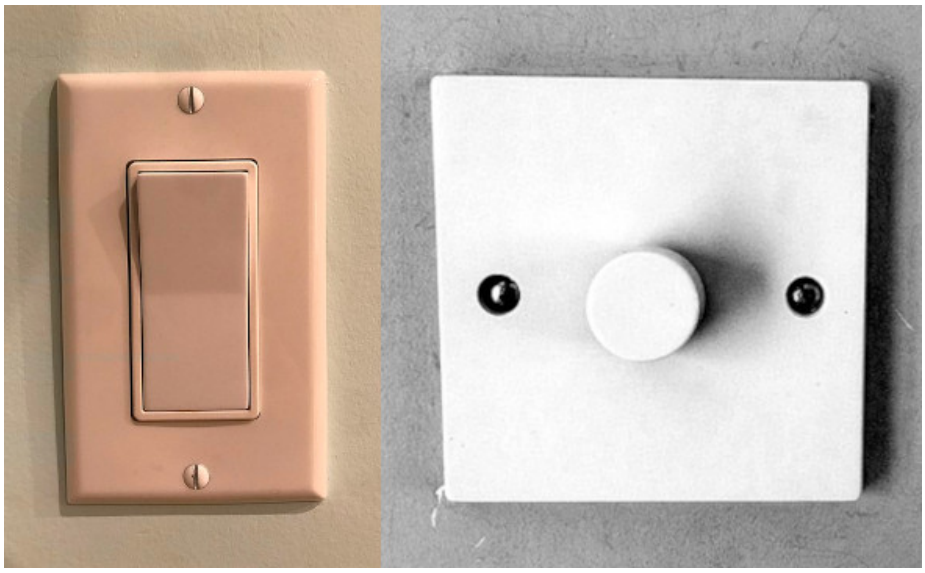

Figure 4.2: On/off light switch (left) and dimmer light switch (right). While on/off switches can only be at minimum or maximum voltage, dimmer switches can be at all voltages in between. Images © DemonDays64/CC-BY-4.0 and © Paolomarco/CC-BY-4.0.

On/off signals are either gates or triggers, as shown in Figure 4.3. Both triggers and gates are rectangular pulses that control a musical event. The difference is that triggers are used to start musical events, but gates are used to both start and end musical events. That's why triggers have the same length, but the length of gates varies by the end time of the musical event. As a result, we can use a trigger to model a drum hit, but we need a gate to model holding down a key on keyboard.

Envelopes are a great example of a continuous control voltage. As discussed in Section 3.4, envelopes can be used to control the amplitude of a sound wave and thus its loudness. We can represent an envelope in control voltage (y-axis) as shown in Figure 4.4. This envelope example illustrates how flexible control voltage can be - any voltage level or shape over time is possible.

In the Eurorack modular format, signal voltages typically span 10 volts. This is much move voltage than common audio devices, so connecting modular gear

[1] Under this definition, low frequency oscillators are audio signals, not control signals.

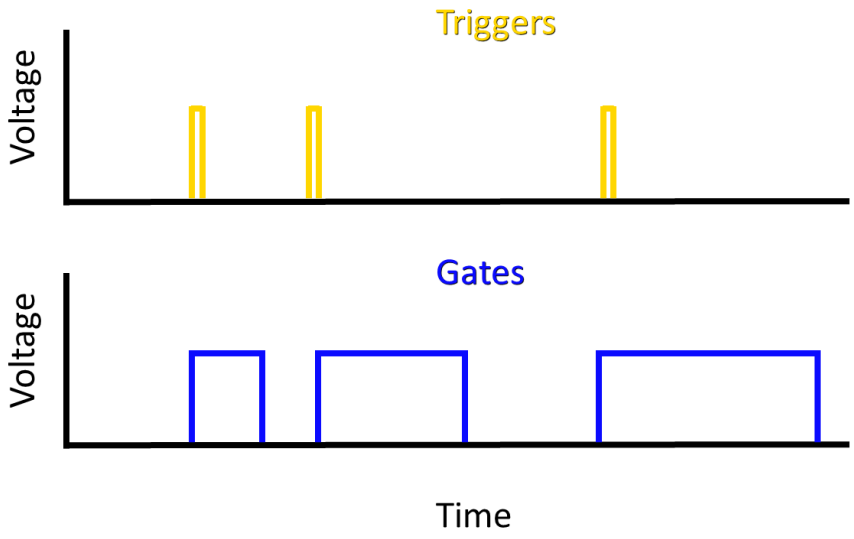

Figure 4.3: Triggers and gates shown over time. Both are on/off unipolar voltages that show as rectangular pulses. Only gates have variable duration.

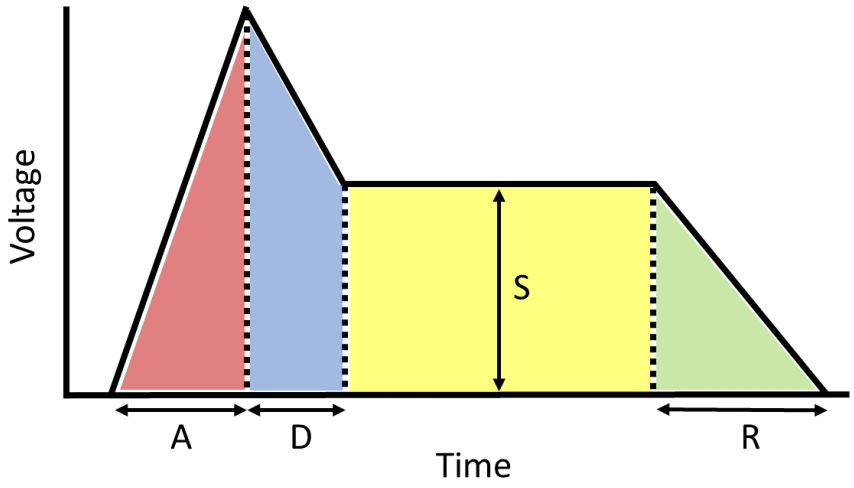

Figure 4.4: An example Attack-Decay-Sustain-Release (ADSR) envelope represented as control voltage.

to standard audio equipment requires some care. Table 4.1 presents the voltage levels for modular signals in Eurorack, together with common audio signals for reference.

Table 4.1: The voltage ranges for Eurorack modular signals compared with standard audio equipment.

Modular signals	Voltage range	Common audio signals	Voltage range
Audio signal	-5 to +5	Pro audio level	-1.7 to +1.7
Continuous control signal[2]	0 to 10	Line audio level	-.45 to +.45
On/off control signal	0 to 5	Instrument level	-.02 to +.02
		Microphone level	-.005 to +.005

Modular signals are transmitted by patch cables.[3] Patch cables are primarily mono but sometimes stereo if a jack supports it. These two patch cable types look identical, except at their connectors, as shown in Figure 4.5. Stereo patch cables are TRS (tip-ring-sleeve), whereas mono patch cables are TS (tip-sleeve).

4.3 Signals are interpreted by modules

One of the most powerful aspects of modular synthesis is the variety of ways that modules can be connected together. You can often send an unusual signal to a module, and that module will still respond. However, its also possible to send a signal to a module and for nothing to happen. The reason for this is that modules expect a certain type of signal at each input jack, and they will interpret signals they receive according to these expectations.

Take gates and triggers for example. What makes a gate a gate and a trigger a trigger? As we said above, gates have a variable duration, but that doesn't prevent them from being used as triggers. Modules that receive triggers listen for the leading edge of the rectangular pulse and then stop listening. So a module expecting a trigger will be satisfied with a gate. If the trigger is wide enough, it might also satisfy a module expecting a gate, because that module will listen for the trailing edge of the rectangular pulse to mark the end of a gate. We probably wouldn't want to do this in practice because we wouldn't be able to change the length of the musical event, but you get the idea.

[2] Can be inverted

[3] The Eurorack standard does allow for signals to be transmitted using a bus on the in-case ribbon cable, but few modules support this. Likewise there are other bus conventions that are not widely followed like i2c. Often if two modules are connected behind the panel, one is an expansion module that adds additional connections or capabilities to the other.

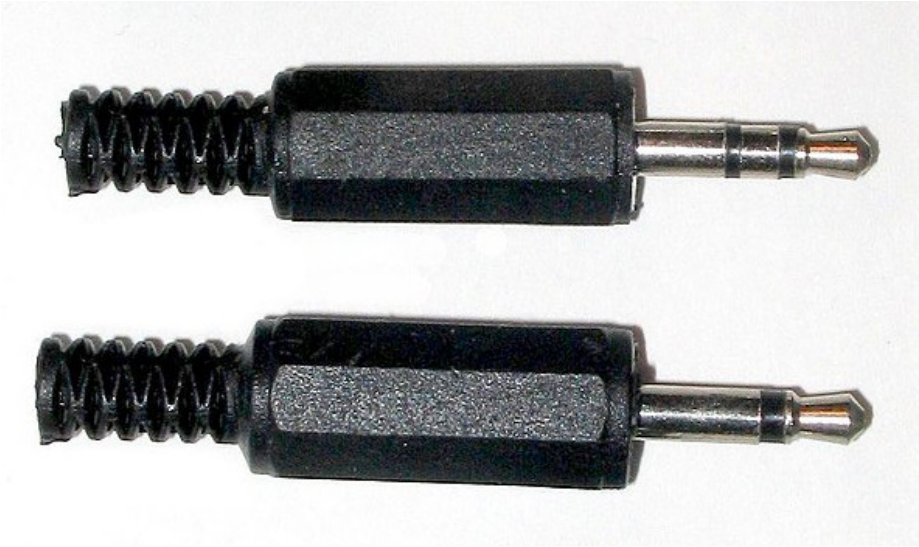

Figure 4.5: Stereo (upper) and mono (lower) patch cable connectors. Both have a tip and sleeve conductor, but stereo has an additional "ring" conductor to carry a second audio channel. Image public domain.

Modular signals are also somewhat interchangeable between audio signals and control signals. Recall that human hearing is sensitive to frequencies between 20 Hz and 20 kHz. So if we generated a train of rectangular gate pulses in this frequency range, we are generating the same shape as audio, and we can listen to our gates as a pulse wave. Similarly, if we generate envelopes repeatedly at an audible frequency, our envelope becomes a waveshape, and we will hear a sound based on the shape of the envelope. However, in other cases you might hear nothing! This is because jacks that expect audio are AC-coupled, which removes low frequency signals, noise, and offset bias that would interfere with sound quality.

One signal that deserves special mention is volt per octave (V/Oct). This signal used to tell oscillators and other generators what pitch they should play. The V/Oct standard was pioneered by Moog and is widely used in the Eurorack format. Note that V/Oct goes straight from the voltage representation to pitch perception, with no mention of frequency. This makes it easy for musicians: since each volt represents an octave, it is easy to play a note one octave above the current note by adding one volt to the current signal. Similarly, one semitone above a note corresponds to 1/12 of a volt above the current voltage.[4] While just about any signal could be used,[5] continuous control signals are commonly

[4] Western music divides the octave into 12 pitches called semitones based on a system called twelve-tone equal temperment.

[5] Typically unipolar signals are used for V/Oct.

used in order to access a range of pitches.

When signals don't work as expected, it can be hard to figure out why. One of the best tools you can use to diagnose problems is an oscilloscope. Oscilloscopes display voltages over time, so they are great for displaying modular signals, including rapidly changing signals like audio. A bench oscilloscope is shown in Figure 4.6, but several modular manufacturers have created compact oscilloscopes that fit into a case, and you can also use a software oscilloscope if you have a DC coupled audio interface. Another great tool for diagnosing problems is the manual for the module in question. If the signal in the oscilloscope looks correct, then it's likely you have a misconception about the type of signal the module is expecting.

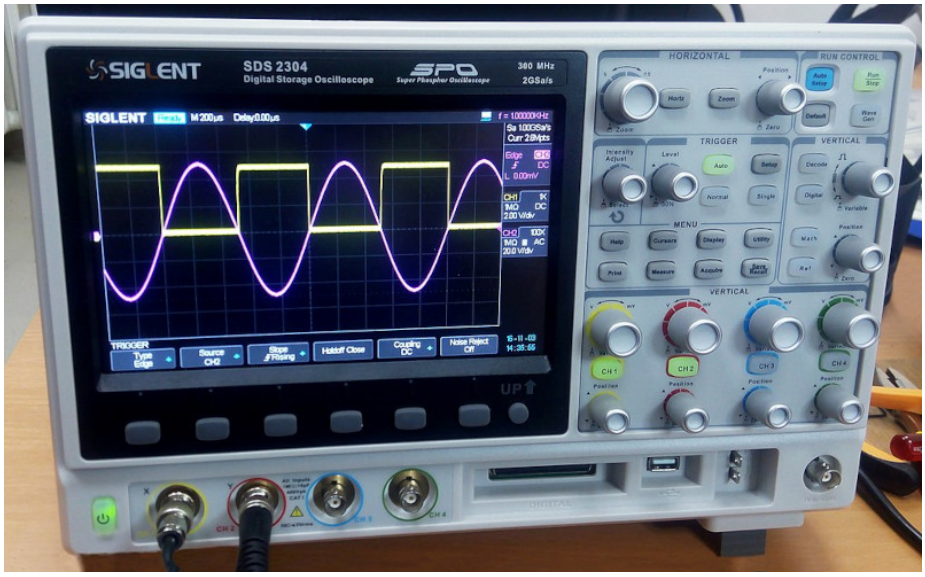

Figure 4.6: A bench oscilloscope showing a sine wave and offset square wave simultaneously. Image © Wild Pancake/CC-BY-4.0.

4.4 Pulling it all together

Let's put these ideas into practice with some basic patches! As stated, we'll be using a web-ported version of VCVRack, which you can launch using buttons below. The port currently has some quirks:

- It will not run on Safari because Safari does not support fixed-width SIMD.[6] That means it will not run on iOS at all per app store rules.[7]

[6] https://webassembly.org/roadmap/
[7] https://developer.apple.com/app-store/review/guidelines/#2.5.6

[8]

- It has a large download because it bundles modules. For best performance, you should use Firefox, which currently caches the files better than Chrome.[9]

In the following sections, we'll start with a patch and incrementally expand it to add more functionality.

4.4.1 Drone

The most basic patch that makes a sound is a *drone* patch. It's the most basic because it only has one real module, which is a *generator* to make the sound. We'll use an oscillator for the generator and one extra module, an audio interface module, to connect the oscillator output to our speakers. For this patch only, I'm going to demonstrate using the video in Figure 4.7 to explain the user interface of the modular software.

Figure 4.7: Youtube video describing the VCVRack/Cardinal interface and building a drone patch.

After you watch the demonstration, try to make the patch yourself using the button in Figure 4.8. When you press the button, you'll see an interface that also includes `Instructions`, `Solution`, and `Close` buttons. These are self-explanatory, but in particular the `Close` button will return you to the book.

[8] https://en.wikipedia.org/wiki/WebKit
[9] https://github.com/DISTRHO/Cardinal/issues/287#issuecomment-1245929349

Figure 4.8: Virtual modular for making a drone patch.

4.4.2 Using an oscilloscope

You can use an oscilloscope to see the wave produced by the oscillator. The most important controls on an oscilloscope are the time and the gain. You can think of the time as slowing the wave down enough so it just about stays still. If you don't do this, most waves will just look like a blur because their frequencies are so high. Gain, on the other hand, is useful if the wave has a low amplitude. If you're expecting a wave and only see a line, you should try increasing the gain. You can think of gain as zooming in on the wave. Finally, oscilloscopes (scopes) don't change the signal at all. So whatever you patch into them is copied exactly on their output jacks. This makes it easy to put a scope in between other modules without changing the resulting sound. Try adding a scope between the modules in the previous patch using the button in Figure 4.9.

Figure 4.9: Virtual modular for making a drone patch with a scope.

4.4.3 Controlling pitch

You're probably realized why the drone patch has its name - it just produces a constant pitch at a constant volume. To make things more interesting, let's control the pitch that goes into the oscillator. You'll need a *controller* for this. Let's use the "Twelve-Key" (12 key), which has a miniture keyboard built into the module. The output of the 12 key that we are interested in is the control voltage (CV) that outputs V/Oct. If you connect that output to the V/Oct input of the VCO, the VCO will change its frequency every time a key is pressed. This is analogous to precisely and instantly moving the frequency knob on the VCO. Try adding the 12 key to the left of the VCO in the previous patch using the button in Figure 4.10.

Figure 4.10: Virtual modular for making a single voice patch with a scope and keyboard control of pitch.

4.4.4 Controlling note duration (on/off volume)

In the last patch, pressing a key changed the pitch, but the note continued until the next key was pressed. You can control note duration independently of pitch using a voltage controlled amplifier (VCA). Just like you can control the frequency of the VCO by sending it control voltage, you can change the amplitude of a VCA by sending it control voltage. The control voltage we'll use is the gate output of the 12 key. That way, each time a key is pressed, two control voltages will be sent out at the same time: a V/Oct to control pitch and a gate to control note duration. The VCA will "open" and let the full amplitude of the wave through when the gate voltage is high, and the VCA will "close" and let nothing through when the gate voltage is zero. Try adding a VCA between the VCO and Scope modules using the button in Figure 4.11.

Figure 4.11: Virtual modular for making a single voice patch with a scope and keyboard control of pitch and note duration.

4.4.5 Controlling note dynamics (volume during note)

We can make the sound even more interesting by making the volume of the note change while the gate is active. To do this, we'll use an envelope generator, specifically the ADSR module. The ASDR module will detect the leading edge of the gate and proceed through its (A)ttack and (D)ecay stages. As long as the gate is held, the ASDR module will hold at the (S)ustain level. Finally, when the trailing edge of the gate is detected (meaning the key has been released), the ASDR module will proceed through the (R)elease stage. Remember that an envelope generator doesn't do anything by itself; if we want to control the volume of the signal with the ADSR, we must connect the ADSR to the VCA. Try adding an ADSR module between the VCO and VCA modules using the button in Figure 4.12.

Figure 4.12: Virtual modular for making a single voice patch with a scope and keyboard control of pitch and note duration, and an envelope to control dynamics during the note.

4.5 Moving forward

The basic modeling concepts and patches in this chapter provide a high level overview of what's to come. The following chapters will go into the three module categories (controllers, generators, and modifiers) in more detail and similarly provide concrete examples along the way. From there we will continue to spiral outward into increasingly complex modules and applications.

Chapter 5

Controllers

Controllers initiate musical events, and in some cases, also define when musical events end. Chapter 4 introduced the keyboard controller, which is perhaps the most intuitive controller because it mirrors a traditional instrument. Not all controllers are as obvious or as intuitive, however. This chapter focuses on clocks and sequencers, which are canonical examples of modular controllers. Unlike keyboards, clocks and sequencers are partially automated. This means that, once started, they continue to produce control signals without additional manual control.

5.1 Clocks

Clocks in modular synthesis typically perform two functions. The first function is synchronization. Just like group of musicians may use a common reference to stay on the beat, e.g. drums, different modules can use a common clock to stay in time with each other. A modular clock is similar in several respects to a metronome, which is a musical aid for keeping time. Metronomes like the one shown in Figure 5.1 create a regular click or pulse that is adjustable to a given beats per minute (BPM). For reference, most current music ranges from 80-160 BPM.[1] Likewise, a modular clock creates a trigger pulse[2] at a given BPM and makes this signal available on an output jack.

Many clock modules also support ways of marking time off the main clock signal. Most often these are clock divisions. A clock division divides the clock frequency by an integer to get a new, lower frequency. For example, if our clock is 120 BPM, then it is producing triggers at 2 Hz. A clock division of 2 would

[1] https://blog.musiio.com/2021/08/19/which-musical-tempos-are-people-streaming-the-most/

[2] Some clocks will produce gates.

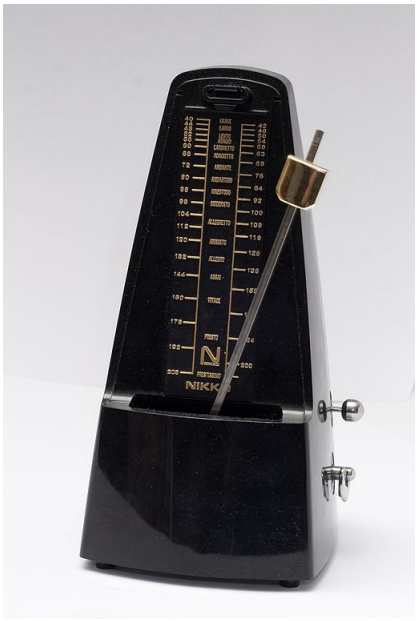

Figure 5.1: A metronome produces a periodic sound to help musicians keep rhythm. The mechanism is an inverted pendulum with a weight on the end; moving the weight up/down changes the speed of the metronome accordingly. Image © Vincent Quach/CC-BY-3.0.

therefore produce triggers at 1 Hz, or 60 BPM. Clock divisions of 2, 4, and 8 are relatively common. Each of these specialized signals requires its own jack, so typically only a few such divisions are available on a clock module.

Clock divisions can be used to implement time signatures that track how many beats occur in a measure (also known as a bar). A common time signature in popular music is $\frac{4}{4}$, which indicates there are four quarter notes per measure, in contrast to $\frac{3}{4}$, which indicates there are three quarter notes per measure. Assuming each pulse indicates a quarter note, an end of measure signal can be generated by using a clock division matching the top of the time signature. Because a clock pulse represents the lowest resolution control signal and thus the shortest note, using more pulses per quarter note allows more resolution and shorter notes.[3]

The second function of a modular clock is transport control, which is recording terminology for controls like play, pause, stop, rewind, etc. Since clocks are not involved in recording per se, they typically have start, stop, and reset controls. A reset signal is analogous to a rewind in the sense that all receiving modules will start at their initial states rather than where they left off. Resets are a useful tool in both composition and recording. Each of the transport controls is likely to have its own button for manual control, plus an output jack to send the control signal to downstream modules.

As with all things modular, it's important to realize that although there are specialized clock modules, anything that produces a trigger/gate pulse can be used as a clock.[4] Since clocks are sources of triggers, clocks can also be used for non-clock purposes.

5.1.1 Clock under a scope

To get a better sense of what's going on, let's take a look at clock output on a scope. Try connecting a scope to a clock's main output and bar output using the button in Figure 5.2. You should see regular pulses off the main output and a different color bar output overlaid every 4th beat using the default time signature.

Figure 5.2: Virtual modular for making a clock patch with a scope.

[3] Not all modules receiving clock will interpret a pulse as a quarter note. Some have higher resolutions and so require multiple pulses to advance. Twenty-four pulses per quarter note is one such standard.

[4] Some modules will accept non-rectangular signals as a clock signal as long as the voltage exceeds some internal threshold.

5.1.2 Clock as a generator

Since clock modules produce a regular stream of pulses, we can hear them if they pulse at audio rates. Recall the bottom threshold of human hearing is approximately 20 Hz (20 cycles per second). Since 1200 BPM corresponds to 20 Hz, we should be able to hear clock signals above this BPM. The clock module from the last patch only goes up to 300 BPM, but if we use the 16ths output[5], we can achieve four times that, or 1200 BPM. Try adding an audio out module after the scope using the button in Figure 5.3 to extend the last patch. The sixteenth note pulses should show as approximately a square wave, and by moving the BPM up and down, you should hear a change in pitch.

Figure 5.3: Virtual modular for making a clock-based drone patch with a scope.

5.2 Sequencers

The basic idea of a sequencer is very simple: a sequencer replays a control signal at a particular moment in time. To represent time, sequencers use the idea of a *step*. Each pulse of the clock will advance the sequencer by one step. Any given step will have one or more control signals pre-stored. On each step, these control signals will be sent to the corresponding output jacks. When a sequencer reaches the last available step, it will loop back to the first step. Sequencers typically have a switch to lower the number of available steps from the hardware maximum, e.g. from eight steps to seven. The steps for a given control signal are respectively referred to as a *channel*.

In analogue hardware, sequencers supporting only one control signal tend to be very compact and require external clock inputs. Sequencers supporting multiple control signals tend to have their own clock and transport controls. When a sequencer has only one control signal, it can be anything, but is most frequently either a trigger or a control voltage. When a sequencer has multiple control signals, they are frequently a combination of triggers and control voltages arranged in different channels. As previously discussed, these control signals naturally correspond to initiating musical events and pitches for notes, but any musical parameter that can be controlled by voltage can be stored in an analogue sequencer.[6]

[5] This particular clock does not use division notation.

[6] Some digital sequencers can replay voltage-based control signals, but not all.

5.2.1 Clocks as sequencers

You could consider a clock as a kind of sequencer that produces triggers on a single channel. Let's look at this by making a bass drum patch driven by a clock, which will form the basis of other patches in this section. You should refer back to Section 4.4.5 if you don't recall some of the modules involved. This *very* basic kick patch uses a sine wave from the VCO and controls the amplitude of the sine wave with a fast attack, medium decay envelope. Try making as much of the patch as possible using the button in Figure 5.4 before referring to the instructions.

Figure 5.4: Virtual modular for making a clock-driven kick patch.

We could expand this example by using clock divisions to add multiple percussion parts. For example, a closed hi-hat on the clock, a kick drum on /2, and an open hi-hat on /4. Keep the clock sequencer example in mind as your continue through this chapter. Ultimately any module that produces the signals we want can be used as a sequencer, regardless of whether the module is called a sequencer.

5.2.2 Trigger sequencers

The main difference between a clock and a trigger sequencer is the precise control over the timing of the triggers. Let's look at a simple trigger-based sequencer to extend the previous kick patch. Try adding the TRG module between the Clock and VCO using the button in Figure 5.5. Once you have it set up, activate a pattern of steps that isn't strictly regular, e.g. some beats immediately after each other and some spaced apart. Each activated step will issue a gate. Control of irregular timings is where trigger sequencers really shine compared to sequencing off a clock.

Figure 5.5: Virtual modular for a trigger-sequenced kick patch.

5.2.3 Control voltage sequencers

Control voltage sequencers replay voltage for each step. Typically the voltage is constant for the duration of the step, which is useful for sending V/Oct signals to play notes. To keep things simple and to build on the previous patch, let's create a saw-based synth voice whose pitch is controlled by a separate control voltage sequencer. Try duplicating the VCO, ADSR, and VCA modules from the last patch and controlling the VCO with the ADDR-SEQ module by using the button in Figure 5.6. Each sequencer step has a voltage controlled by a knob, so turning these knobs sets the note pitch for each step. Because the volumes of these two instruments, or voices, is so different, you'll need to run each into a mixer module rather than directly into the host audio.

Figure 5.6: Virtual modular for a trigger-sequenced kick mixed with a control-voltage sequenced saw wave.

5.2.3.1 Sequencing rests

A limitation with the ADDR-SEQ module is that it always plays a note and never pauses, or rests. One option would be to adjust the pitch on a step so it is outside the range of human hearing. While this option is simple and effective, it's possible that the sound will interfere with other sound waves in ways that become audible. Another option is to replace the clock signal into the ADDR-SEQ module with the output of the trigger sequencer. Since the trigger sequencer has steps with no trigger, this means that no clock would be sent to ADDR-SEQ on these steps. The effect of using the trigger sequencer as a clock depends, however, on how the VCO envelope is gated, either with the clock or with trigger sequencer. Try this using the button in Figure 5.7 to explore the different combinations.

Figure 5.7: Virtual modular for a trigger-sequenced kick mixed with a control-voltage sequenced saw wave, using the trigger sequence as a clock on the voltage-controlled sequencer.

The patch in Figure 5.7 illustrates that the way the sequencers are connected to each other and the clock plays a big difference on the resulting sound. If

the gate and the clock are from the same source, and therefore match, every sequenced note will be heard. If the gate source is the trigger sequencer, rests will be produced. These effects are summarized in Table 5.1. The second row is comparable to a multichannel sequencer that allows steps to be silenced, thus reducing the overall pitched steps available by replacing them with silence. In contrast, the third row *inserts* silence between pitched steps, creating a longer sequence where no pitched steps are discarded.

Table 5.1: Effects of gate and clock mismatching when combining trigger and control voltage sequencers to produce rests.

Clock source	Gate source	Effect
Clock	Clock	Every note played, no rests
Clock	Trigger sequencer	Notes skipped, with rests
Trigger sequencer	Trigger sequencer	Every note played, with rests
Trigger sequencer	Clock	Notes repeated, no rests

5.2.3.2 Sequencing note duration

A second limitation of the ADDR-SEQ is that all the notes are the same duration. This limitation can be somewhat addressed by using variable length release on respective envelopes. A longer release would lengthen the duration of the sound, but for any instrument that has a sustain, the volume would not be correct using this method. The fundamental problem is that the gate into the ADSR is a fixed width which results in a fixed length note. What is needed is to change the clock signal into a gate where the length of the gate matches the length of the desired note. Ideally we'd use a separate control voltage sequencer to set the gate lengths for us, but in the interest of keeping things relatively simple, let's control gate length manually. Try to add a trigger to gate module that takes the clock and sends out a gate to the saw envelope using the button in Figure 5.8. As you move the gate length knob, you'll hear that the note lengths change correspondingly.

Figure 5.8: Virtual modular for a trigger-sequenced kick mixed with a control-voltage sequenced saw wave, using the trigger sequence as a clock on the voltage-controlled sequencer and a trigger to gate module to control note length.

5.3 Summing up

Clocks and sequencers are canonical modules for creating control signals in a modular system. Unlike manual controller like a keyboard, both clocks and sequencers are semi-automated: once they are set up, they will continue to generate control signals without additional human intervention. The patches in this chapter explored more traditional analogue sequencing using separate modules. Larger analogue sequencer modules with multiple channels and features can often be viewed as composites of these smaller modules, as shown in Figure 5.9. Perhaps the most powerful aspect of modular controllers is that anything generating a control signal can be used as a controller. Thus one can build a custom sequencer out of basic components to solve a particular musical problem. Future chapters will explore additional controllers and control techniques.

Figure 5.9: The SEQ3 sequencer module from VCVRack. The trigger sequence channel is highlighted in red and the control voltage channels are highlighted in blue.

Chapter 6

Generators

Generators create audible sound. Chapter 4 introduced oscillators, which are the prototypical generator. This chapter expands upon oscillators and how they can be combined together to produce fuller and more interesting sounds. Additionally, two new generators are introduced, noise and samplers. Noise can also be combined with oscillators and other generators to create richer and more realistic sound. Samplers represent the extreme of realism in generators, at least as far as their digital representation allows.

6.1 Chords

Section 3.1 briefly mentioned the relationship between harmonics and fundamental relationships like octaves and fifths. To recap, the second harmonic has double the frequency of the first and so is one octave above the first harmonic. We can express this relationship as 2:1, i.e. an octave is twice the frequency of the the fundamental. Similarly the third harmonic is a fifth above the second harmonic, so we can express a fifth as 3:2.[1]

Now consider what happens if two notes an octave apart or a fifth apart are played at the same time. Clearly some of their harmonics will be the same. Thus playing two or more notes this way will add and enhance harmonics beyond that of a single note, creating a richer and fuller sound. Of course, notes may be played that are not harmonically related, to a different effect.[2] When two notes are played simultaneously, it is called an interval, with three notes, a trichord, with four notes, a tetrachord, etc. Many harmonically related notes can be played simultaneously to good effect. For example, the THX Deep Note

[1] Whole number ratios require just intonation. The twelve tone equal temperment used in Western music is a close approximation, e.g. 3:2 is 1.498 in twelve tone equal temperment.

[2] The reader may be interested in consonance/dissonance as discussed in music theory.

in Figure 6.1 resolves to 11 notes, mostly harmonically related by 3, 4, and 7 semitones (minor/major thirds and fifths).

Figure 6.1: Youtube video of the THX Deep Note, which resolves to 11 harmonically related notes. Image © THX Ltd.

In modular synthesis, there are at least two ways to create chords, which correspond to stages of the process to create a single note. At the first stage, we have a V/Oct control signal, so we can create chords by making new control signals harmonically related to the original signal. At the final stage, we have an audio signal, so we can create chords by making new audio signals harmonically related to the original signal. Let's look at both.

To create harmonically related control voltage signals, we can use a voltage offset module to create new signals offset from the original signal, i.e. the original voltage plus/minus a harmonically-related voltage. It can take a little math to do this if the offset module is labeled in volts[3], but if the offset module is labeled in semitones, it is fairly straightforward, e.g. a fifth is seven semitones. Additionally we will need multiple oscillators to play a note for each offset voltage. We could reasonably do this with separate oscillators, but some chord-oriented modules contain multiple oscillators and take separate V/Oct for each. Try making a trichord using a fifth and an octave using the button in Figure 6.2.

To create harmonically related audio signals, we can use a clock divider as long as the original signal can be interpreted as clock and the clock divider in question can accept audio rates. This approach is related to a classic technique for creating a fatter sound, the sub-octave square wave. A sub-octave square wave is exactly that: a square wave one octave below the fundamental. Since

[3]Recall 1 V/Oct means a semitone is 1/12 of a volt.

Figure 6.2: Virtual modular for making a chord using signal offsets.

a clock division of /2 cuts the source clock frequency in half, we can easily generate a sub-octave square wave using a clock divider, albeit with the same assumptions. Perhaps more interestingly, if our clock divider supports both even and odd divisions we can create subharmonic chords, or perhaps more appropriately, an inverted chord. All that is required is knowing the harmonic ratios mentioned above, e.g. 3:2 means that the divisions /3 and /2 are a fifth apart. Try making a trichord using a fifth and an octave using the button in Figure 6.3.

Figure 6.3: Virtual modular for making a chord using clock divisions.

6.2 Chorus

We can also combine different oscillators using the same, or almost the same pitch. As discussed in Section 3.1, two identical waves will constructively interfere with each other and produce a new wave with twice the amplitude. However, if the waves are slightly out of phase or have slightly different frequencies, a subtle interference pattern will be produced that slowly changes the amplitude and timbre of the sound over time. This effect is used in pianos, which have more than one string for most notes, as well as 12-string guitars that have six pairs of identical strings. In fact, this effect is so general is used as an audio processing effect called chorus.

To achieve this effect in modular, one only needs to slightly detune one of the VCOs with respect to the other or change the phase relationship between oscillators. Typically two independent oscillators will already be out of phase, but in modules that contain multiple oscillators there is sometimes a control to adjust the phase of one oscillator with respect to the others. Both detuning and phase changes must be subtle in order to increase the richness of the sound. If the detuning is too great, two different pitches will be perceived as out of tune rather than a richer single sound. Similarly, as phase is increased there is greater destructive interference and potentially cancellation when completely out of phase. Remember that most interference will create spans of sound with

greater loudness and quietness, as shown as shown in Figure 6.4.

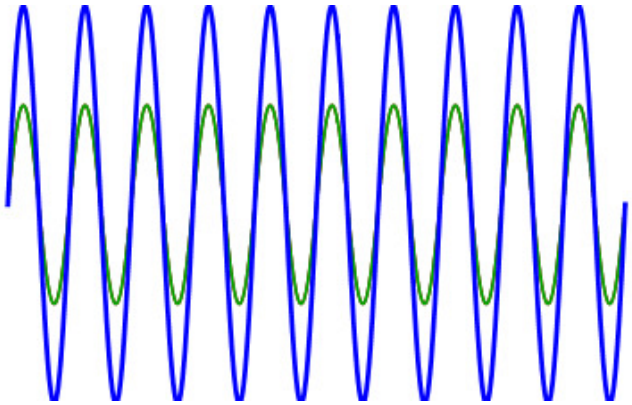

Figure 6.4: Animation of interfering sine waves as one increases in frequency. Note how multiple beats appear in this short span as the frequency increases. Image © Adjwilley/CC-BY-SA-3.0.

These changes in volume can be perceived as beats if they are strong enough and occur within a certain range of frequencies. Typically such beats are unwanted, so the interference between the sound waves should either be too slow/fast to be perceive as beats or the beats themselves are so soft that they are not perceived as beats. Returning to the first patch, try to detuning the oscillator a fifth above the root while silencing the oscillator an octave above, using the button in Figure 6.3. You should hear a beat whose beat increases as you increase detuning.

Launch Virtual Modular

Figure 6.5: Virtual modular for detuning a second oscillator.

6.3 Low frequency oscillators & uses

Low frequency oscillators (LFOs) are modules that produce waves at frequencies below audio rates. It may seem strange to introduce LFOs in a chapter on generators, which by our definition create audible sound. The reason is that LFOs are essentially identical to regular oscillators (VCOs) except that LFOs operate at lower frequencies. In fact, it's common to find LFOs that can operate at audio rates and VCOs that operate below audio rates. One other common difference is that LFOs often have a unipolar switch. The switch changes the

output of the LFO, which are by default bipolar and range from -5 to +5 V, to unipolar output of 0 to 10 V. We've already met a unipolar LFO by another name - a clock is a unipolar LFO that uses a pulse wave!

LFOs have many potential uses, but perhaps the most prominent is to add movement to a sound. For example, it's generally thought that chorus adds to sound through movement, specifically movement caused by interference and changing harmonics. LFOs can create similar movement in other contexts as described below. What all these contexts share is two properties: the speed of the movement (frequency) and the depth of the movement (amplitude).

6.3.1 Pulse width modulation

LFOs can create the effect of two interfering oscillators with only one VCO. This is an interesting effect in its own right, but is particularly valuable when only one VCO is present. The prototypical case for this effect is with pulse waves, where it is called pulse width modulation (PWM). As a motivating example, consider Figure 6.6, which shows the interference of two pulse waves at 25% and 50% duty cycles, respectively. The resulting wave looks like the 25% wave where the wave is positive and like the 50% wave where the wave is negative. This occurs because the two component waves, which have the same amplitude, combine to either exactly double the amplitude or sum to zero at all points of the wave.

Because this is such a useful and interesting effect, many oscillators that produce pulse waves have a PWM input that allows the duty cycle of the wave to be controlled by another module. Try to implement PWM with an LFO using the button in Figure 6.7. If you hook up the VCO square wave to a scope, you can see the resulting waveshape. The LFO frequency and the PWM depth knobs give control over the speed of the PWM and the depth (i.e. the range of duty cycles covered), respectively. As you listen to the result of PWM, take note of the changes in the harmonics. Square waves have only odd harmonics, but pulse waves more generally have every nth harmonic removed, where n is the denominator of the duty cycle of the wave. For example, a square wave with a 50% duty cycle (1/2) has every 2nd harmonic removed, a pulse wave with a 33% duty cycle (1/3) has every 3rd harmonic removed, and so on. Therefore PWM, by modulating the duty cycle, is continuously adding and removing harmonics as the duty cycle changes.

6.3.2 Vibrato

Vibrato is movement around pitch that can be defined in terms of speed of movement around a center pitch and depth of movement, or distance from, that same center pitch. Vibrato is widely used in music, but is perhaps most strongly associated with opera as shown in Figure 6.8.

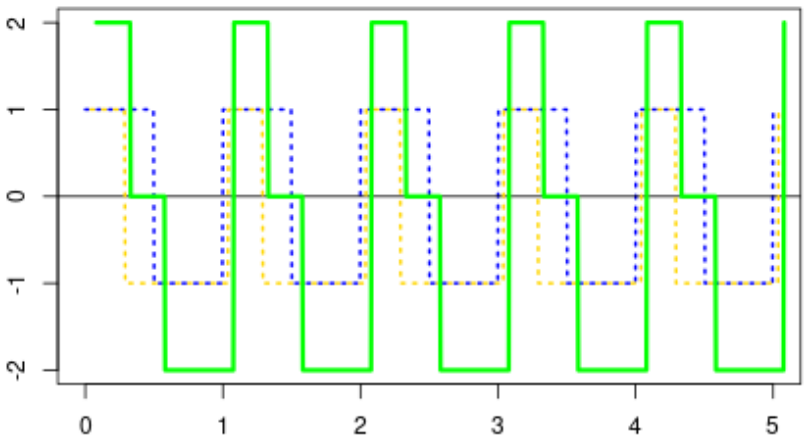

Figure 6.6: Interference of a pulse wave with 25% duty cycle (gold) with a pulse wave with a 50% duty cycle (blue). Note the resulting wave (green) has a positive signal matching the 25% wave and a negative signal matching the 50% wave. Waves are offset for comparison.

Launch Virtual Modular

Figure 6.7: Virtual modular for implementing pulse width modulation (PWM) using a low frequency oscillator (LFO).

Figure 6.8: Youtube video of an opera singer's vibrato, or frequency variation around a central note. Image © jiggle throat.

LFOs can create vibrato on a VCO by controlling V/Oct. Unlike PWM, an additional VCA is needed to control the depth of the vibrato.[4] Try to create vibrato with an LFO using the button in Figure 6.9. With a little adjustment of the parameters, it's possible to match the vibrato of the opera singer just discussed. Since no vibrato occurs when the LFO frequency is zero, one can turn the vibrato effect on and off by setting LFO frequency, e.g. through a sequencer.

Launch Virtual Modular

Figure 6.9: Virtual modular for implementing vibrato using a low frequency oscillator (LFO).

6.3.3 Tremolo

In electronic music, tremolo is movement around loudness that can be defined in terms of speed of movement around a center volume and depth of movement, or distance from, that same center volume.[5] Note that in other forms of music, there are different definitions of tremolo. Tremolo has been widely used in

[4]Technically you can use the FM input and attenuator, but there are pedagogical reasons for not using the FM input at the moment, namely that we have not discused FM yet.

[5]The mnemonic "tremolo**ud**" might help prevent confusion with vibrato.

popular guitar music since the 1960's, and guitar pedals producing tremolo effects as shown in Figure 6.10 are quite common today.

Figure 6.10: Youtube video of a tremolo guitar pedal. The flashing light corresponds to the speed of loudness changes around a center volume. Image © CheaperPedals.com.

LFOs can create tremolo by controlling the amplitude of VCO output. A VCA is is used to the depth of the tremelo. However, there are two differences with respect to previous patches. First, a VCA accepts unipolar control CV, so the LFO must be set to unipolar. Second, when the LFO goes to zero, it will completely close the VCA just like an envelope does. In order to keep some base line of loudness, you must mix a copy of the VCO output with the tremolo-modified VCO output. Try to create tremolo with an LFO using the button in Figure 6.11 and adjust the parameters to match the tremolo of the guitar pedal in Figure 6.10. As before, no tremolo occurs when the LFO frequency is zero, so on can turn the tremolo effect on and off by setting LFO frequency using a sequencer or other means.

Launch Virtual Modular

Figure 6.11: Virtual modular for implementing tremolo using a low frequency oscillator (LFO).

6.4 Synchronization

Techniques that allow you to remove/reduce phase relationships between oscillators are called synchronization (sync). Despite seeming contradictory to the preceding discussion, sync can create interesting effects exactly because phase relationships have been removed/reduced. The basic idea of sync is quite simple. Every oscillator has an internal reset trigger that tells it when to start drawing its waveshape again. A sync signal overrides this internal trigger, so we can control when the reset happens.

The most common form of sync is known as hard sync. In hard sync, the sync signal removes phase by forcing an immediate reset. Typically the sync signal is the wave output of one oscillator (the leader), which is connecte to the sync input of another oscillator (the follower). If the two waves were identical in frequency, hard sync would simply align them in phase; however this is rarely the case. When the two waves are different in frequency, one wave is reset and therefore has a sharp edges in its waveshape. As discussed in Section 3.2, sharp edges contribute higher partials Sync therefore changes the harmonic content of a wave in a different way than chorus type effects. Notably, the closer the follower's frequency is to an integer multiple of the leader's frequency, the more the sync will emphasize the harmonics of the leader, but in general the new partials will not be harmonically related to the leader. An example of hard sync is shown in Figure 6.12, where the two sine waves on the left are hard synchronized on the right.

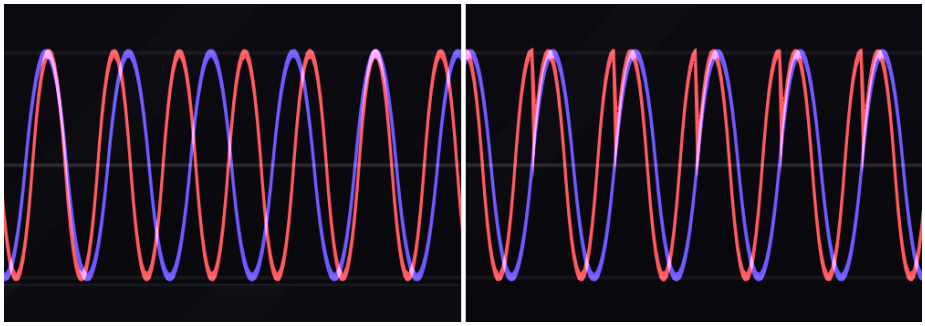

Figure 6.12: An example of hard sync using two sine waves. On the left, the sine waves are not synchronized. On the right, the leader's sine output is connected to the follower's sync input. As soon as the leader's sine wave (blue) increases above zero, the follower's sine wave (red) resets and begins its cycle again, creating a sharp edge in its waveshape.

There are several variations of sync known as soft sync, but they all attempt to align the two oscillators in phase without creating the sharp edge found in hard sync. One notable form of soft sync is reverse soft sync (or flip soft sync). Reverse soft sync reverses the direction of the wave at the moment of reset.

This causes the follower to match the direction of the leader without exactly matching the leader's phase. An example of reverse soft sync is shown in Figure 6.13, where the two sine waves on the left are soft synchronized on the right.

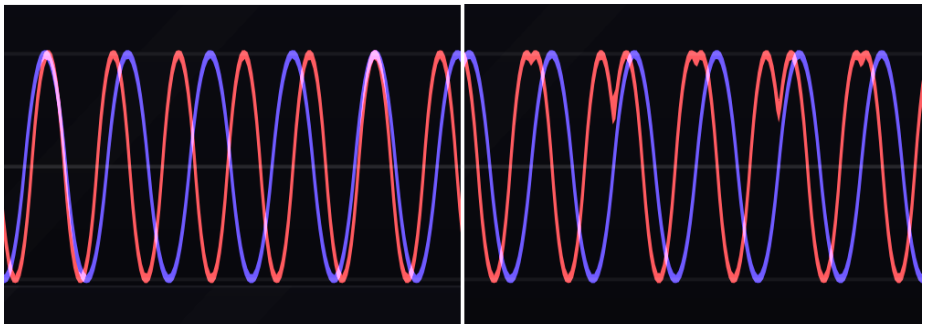

Figure 6.13: An example of reverse soft sync using two sine waves. On the left, the sine waves are not synchronized. On the right, the leader's sine output is connected to the follower's sync input. As soon as the leader's sine wave (blue) increases above zero, the follower's sine wave (red) reverses, i.e. runs in the opposite direction, and begins its cycle again, creating a less sharp edge in its waveshape than hard sync.

Try to create hard and soft sync using the button in Figure 6.14 and that matches Figures 6.12 and 6.13, respectively. Each type of sync produces a characteristic sound that depends on the waveshapes involved and their relative frequencies.

Launch Virtual Modular

Figure 6.14: Virtual modular for implementing hard and soft sync.

6.5 Noise

Noise is commonly used in synthesis to complement other sounds. As discussed in Section 3.3, there are many different kinds of noise that can be distinguished by the frequencies they emphasize. In modular, noise is typically provided by specialized modules, with separate jacks for different colors of noise.

Noise blended with other generators can create transients and complementary sounds that produce more realistic percussion. For example, a kick drum has an initial noise transient that quickly dies out, and a snare has a long-lasting noise component that comes from the metal snares below the bottom membrane.

However, the kick drum noise is tilted towards lower frequencies (e.g. red noise) and the snare is tilted towards higher frequencies (e.g. blue noise). Try to create kick and snare drums with noise using the button in Figure 6.15. The patches are identical except for different noise sources and knob settings.

Figure 6.15: Virtual modular for adding noise to kick and snare drums.

6.6 Samplers

Samplers are popular type of generator that can be used to produce very realistic sounds but simply playing back prerecorded sounds. Samplers are thus very useful for performing with sounds that are difficult to synthesize (e.g. speech) or when it is impractical to use modules to synthesize multiple instruments due to cost/space constraints (as is often the case with percussion).

Samplers do not perform synthesis per se, though there are crossovers like wavetable synthesis, as discussed in Section 2.4. Instead, samplers represent audio data, typically digital audio data, without representing the process that generated the data. Digital audio data is represented according to two parameters that each can be considered a kind of resolution, as shown in Figure 6.16.

Sampling rate measures how close together each sample is taken. If the distance between the samples is small, a straight line is a good approximation of the curve of the wave, and the digital representation has good fidelity to the original sound. However, if frequency of the wave is high or the wave otherwise changes suddenly, those changes may be *between* samples and not show up in the digital representation at all. Common sample rates are 8 kHz (phone quality), 16 kHz (speech recognition quality), and 44.1 kHz (CD quality).

Why sample at 44.1 kHz when the upper limit of human hearing is around 20 kHz? The 44.1 kHz rate is the Nyquist rate for 22,050 Hz, i.e. the Nyquist rate is **double** that frequency, which is about 2 kHz above the standard human limit.[6] The reason to sample at twice the highest frequency is pretty simple. If we assume all sounds are made of sine waves, as covered in Section 3.2, then we need to be able to define the highest frequency component in our audio as a sine wave. If we evenly sample at least two points for a cycle of that sine wave, and we know the maximum possible frequency, then there is only one sine wave that can pass through those points.[7] Without that limit, there are an infinite

[6]The 2 kHz padding allows use of an anti-aliasing filter that prevents higher frequencies from being sampled. We'll cover how filters work in the next chapter.

[7]This is analogous to two data points to solve for two unknowns using linear algebra.

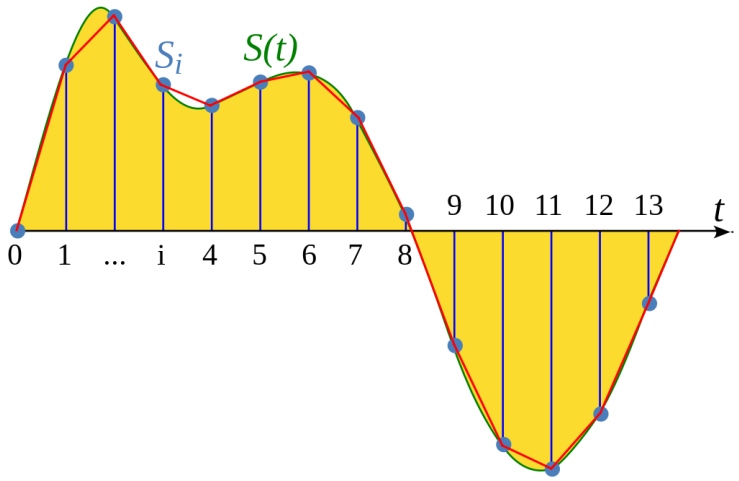

Figure 6.16: A wave digitized by sampling. The sampling rate corresponds to the distance between sample times on the horizontal t axis. The bit depth corresponds to the accuracy of distance between the axis and sampled points on the wave. The straight red segments are the digital reconstruction of the original wave based on the samples. Image public domain.

number of alternative sine waves that will pass between the two points, leading to an ambiguity problem called aliasing where we don't know what sine wave was sampled. A familiar example of aliasing occurs when speed of wheels exceed the sample rate (frame rate) in video/film.

Bit depth measures the accuracy at which the sample points are measured from 0. Imagine you had a ruler with only inches marked - you could only measure inches, right? Digital representations are similar in that we can only divide a space into as many pieces as we have bits. A byte, which is 8 bits, has $2^8 = 256$ possible partitions, so if we represent a 10 V peak-to-peak signal with 8 bits, we have a resolution of $10/256 \approx .04$. This means is that if two different points of the wave are within this resolution, they will be digitized to the same value. This is why higher bit depths have better fidelity to the original signal. Common bit depths are 16 and 24 bits per sample.

While samplers offer less control over their representations than other forms of synthesis, they typically have many options for control, including playback within a sample, playing a sample at a faster/slower speed than the original, playing a sample in reverse, etc. These performance parameters are typically under voltage control and so can be controlled by a keyboard, sequencer, or other controller. Let's take a look at keyboard control of a sampler that changes the playback position of a sample based on the keyboard's output voltage. Try this patch using the button in Figure 6.17. You will need to download the sample file to load it into the sampler.

Figure 6.17: Virtual modular for keyboard control of a sampler's playback position.

We can play, reverse play, and change the playback speed the sample by using an LFO and a gate-style controller. Try to construct this patch using the button in Figure 6.18. Because the LFO produces continuous changes in voltage, the playback is continuous, unlike the keyboard-controlled playback where each key stepped the voltage by 1/12 of a volt.

Figure 6.18: Virtual modular for LFO control of a sampler's playback: forward, reverse, and speed.

Chapter 7

Modifiers

Modifiers modify incoming signals that may be either audio or some kind of control signal. Generators create audible sound. Chapter 4 introduced voltage controlled amplifiers (VCAs) that modify the amplitude of incoming signals, which can be either audio or control voltage. That chapter also introduced envelopes that modify control voltage, and we have routinely used envelopes to control VCAs in order to model the dynamics of various instruments.

This chapter expands upon audio modifiers specifically and focuses on two foundational categories of modifiers, effects and voltage controlled filters. Effects can substantially enrich the sounds you create both in terms of thickness of sound and sense of acoustic space, and voltage controlled filters create some of the most defining sounds in electronic music.

7.1 Effects

There are perhaps an infinite number of possible audio effects. This section focuses on time-based effects, as the ideas behind these are complementary to the following discussion of voltage controlled filters. Time and phase are closely related when we're talking about a repeating waveshape. Imagine you put a red dot on the outside of a car tire and start driving. At any moment, you can stop the car and the red dot will be in a certain position, which you can describe as an angle relative to the ground. If the car has been moving at the same speed the entire time, you can exactly predict where the red dot will be. Figure 7.1 shows an abstract rendering of this example, highlighting the relationship between the rotation of the wheel (in radians) and the shape of a sine wave.

The discussion of simultaneously-sounding oscillators in Chapter 6 emphasized phase relationships between the oscillators as well as slight tuning differences between the oscillators, both of which can give a fuller sound. Phase captures

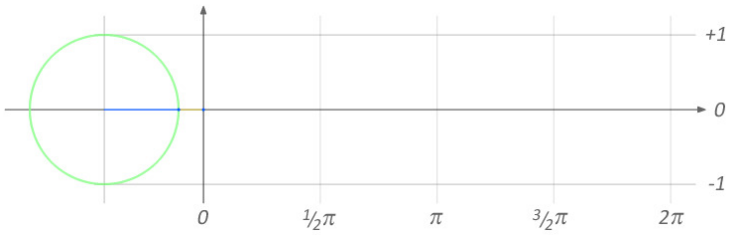

Figure 7.1: Relation between a unit circle and a sine wave. Image © Brews ohare/CC-BY-4.0.

everything about the relationship between continuous tones because the amount one signal has been offset relative to another can always be reduced to between 0 and 360 degrees (or $2pi$).[1] However, music does not consist of continuous tones.

When we begin to consider music more broadly, it makes sense to think of phase and offset distance being decoupled depending on how much time has passed. In real music, simultaneous sounds don't repeat forever but rather change over fairly short timescales, e.g. the envelope of a kick drum. For brief moments of time, phase relationships capture most everything, but for longer offset distances, the amount of time that has passed is more relevant than phase relationships. For this reason, time-based effects for short periods of time are often described in terms of phase, and longer-period effects are described in terms to time. Figure 7.2 illustrates this idea with 6 seconds of natural sound (upper) compared to .03 seconds of that same sound (lower). While the longer stretch is clearly not repeating, the shorter stretch is approximately repeating and could be usefully characterized in terms of phase.

Unlike the topics we've previously covered, time-based effects tend to be implemented by single modules rather than by collections of modules working together. Therefore our discussion will focus on the concepts behind these effects and modules that implement them. We'll begin with effects operating on the longest time scales and move progressively downward until the effects are based in phase.

7.1.1 Delays

A delay effect copies some portion of of the signal and then replays it, typically mixed in with the original signal. The ratio of original signal to processed signal is commonly referred to as dry/wet, where dry is the original signal and wet is the processed signal. Common parameters of delays include the length of the

[1] If the offset is more than 360 degrees, it simply wraps around to a value equivalent to a value between 0 and 360 degrees, e.g. 370 degrees is the same as 10 degrees.

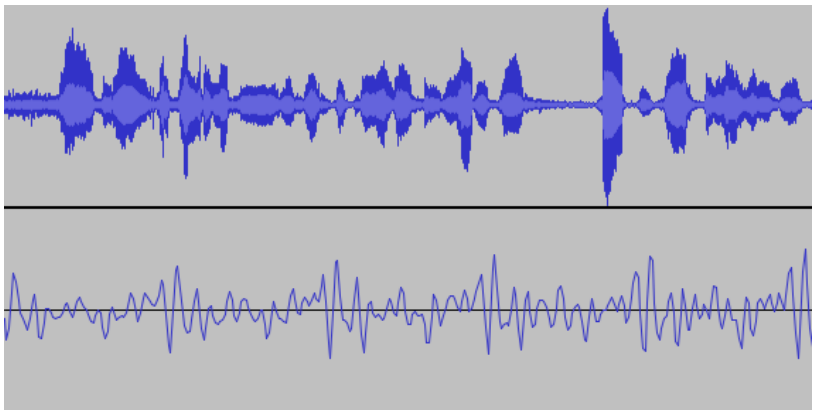

Figure 7.2: Six seconds of natural sound (upper) with no clear repeating structure and .03 seconds of that same sound (lower) with approximate repeating structure.

delayed and the number of times it repeats. Often a decay parameter is applied so that the delayed copy gets attenuated with each repetition. Some delays can play back multiple copies at once, which blurs the boundary between a delay effect and reverb. An example delay effect is shown in Figure 7.3. The original signal (JFK's famous moon speech), is copied and then a delay effect applied to the copy, with decay. Note that the delay effect extends well beyond the length of the original audio. Delay is the longest time-based effect we will discuss, typically ranging from a hundred milliseconds to multiple seconds.

Let's take a look at a delay module. The main controls are the time (length of the delay offset), the feedback (amount of decay), and the wet/dry mix. These parameters can be set using knobs for a consistent sound but also modulated to create very unusual sounds. Try patching up a reverb into a single voice keyboard patch using the button in Figure 7.4.

7.1.2 Reverb

A reverb effect is a delay-time effect with multiple overlapping copies and relatively fast decays. Reverb conveys a sense of space, or room that the sound is occuring in, because it emulates the reflections of sound off various surfaces at different distances. Sound travels at about 1100 feet per second, or a 1.1 feet per millisecond, so each foot of different distance in the reflection should cause an offset of about 1 millisecond relative to other reflections and the original sound. One of the earliest ways of creating reverb was to take a recording and play it back in a particular space while simultaneously re-recording it with a microphone in that space. The approach effectively "bakes in" the natural reverb of the room onto the recording. Various electronic methods have been

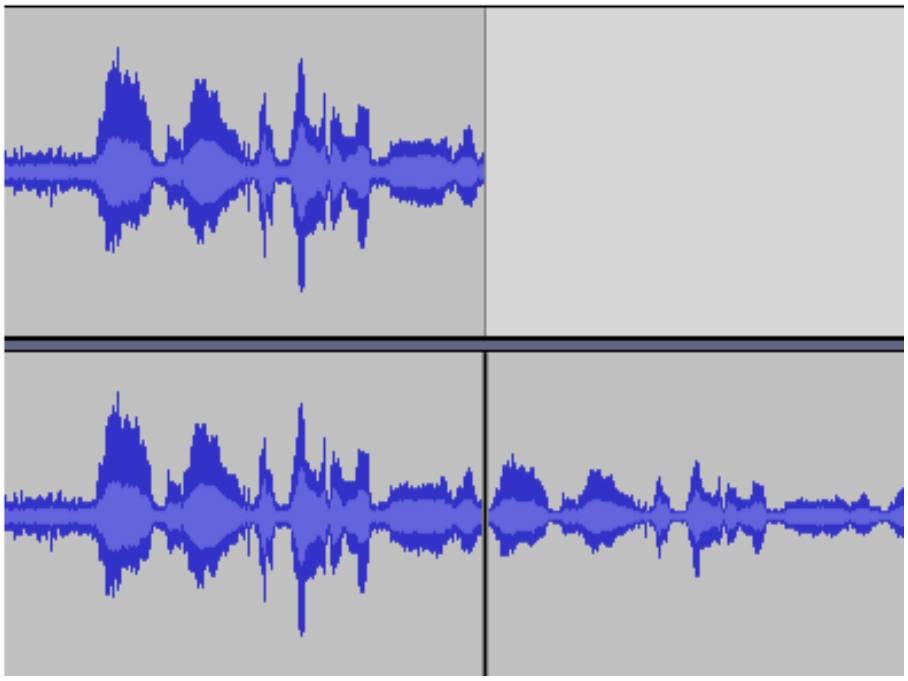

Figure 7.3: Delay applied to an audio clip. The original sound (upper) is duplicated (lower) and the effect applied. The decaying echos of the delay make the lower track significantly longer than the original.

Launch Virtual Modular

Figure 7.4: Virtual modular for a delay effect.

used to create a reverb effect using physical media. Notably among these is the spring reverb, which passes the signal through a spring and uses the reflections created in the spring to model the reflections of sound.

Reverb has been added to the speech example in in Figure 7.5, and like delay extends beyond the length of the original audio. However, the duration it extends is much shorter (about 300 milliseconds), reflecting a faster decay time. Note that this recording was taken in a stadium that already had significant natural reverb.

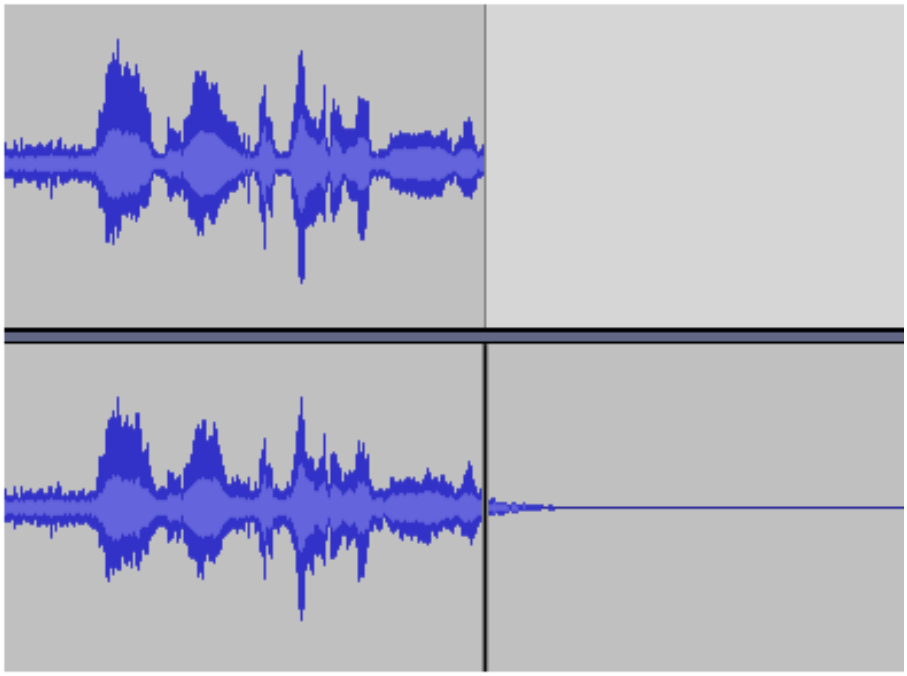

Figure 7.5: Reverb applied to an audio clip. The original sound (upper) is duplicated (lower) and the effect applied. The added reverb slightly extends the length of the original track and is most noticeable at that point due to the natural reverb in the original recording.

Let's take a look at a reverb module. The main controls are the input level, high-pass filtering[2], and the wet/dry mix. This reverb module uses a combinations of sliders and knobs for parameter settings, but these can again be both manually set and modulated to create interesting sounds. Try patching up a reverb into a single voice keyboard patch using the button in Figure 7.6.

[2]High-pass filtering will be explained in detail later in the chapter.

Figure 7.6: Virtual modular for a reverb effect.

7.1.3 Chorus

A chorus effect creates copies of the original signal with slight frequency and phase changes relative to the original. We previously discussed chorus in Section 6.2 in the context of multiple oscillators playing together. The chorus effect is similar except that it applies to the overall audio signal, and it typically has an internal LFO that controls the phase offsets between copies of the signal.

Chorus has been added to the speech example in in Figure 7.7, and unlike the previous effects, does not noticeably extend beyond the length of the original audio. Instead you can see an effect on the shape of the wave as the copies interfere with each other. In this example, the chorus delay is 20 milliseconds, which is in the middle of the typical chorus range of 10-30 milliseconds.

Let's take a look at a chorus module. The main controls are the rate and depth of the internal LFO and the wet/dry mix. As before, the controls can be manually set or voltage-controlled. Try patching up a chorus into a single voice keyboard patch using the button in Figure 7.8.

7.1.4 Flanger

A flanger, or flanging effect is created by creating one copy of the signal and delaying it very briefly, typically 1-5 milliseconds. This slight shift causes harmonically-related frequencies in the signal to be amplified. For example, 1 millisecond shift means that every 1000 Hz, the original signal will be doubled in amplitude through constructive interference. This doubling will occur for all integer multiples of 1000 Hz, and so produces a harmonic series with 1000 Hz as the fundamental. Thus a stable offset can create a harmonic sound even out of noise, as shown in Figure 7.9. Flangers typically use an LFO to vary the offset to create a sweep across such sounds.

A flanger has been added to the speech example in in Figure 7.10, and like chorus does not noticeably extend beyond the length of the original audio. Additionally, and like chorus, you can see an effect on the shape of the wave as the delayed copy of the signal interferes with the original. In this example, the flanger delay is 1 millisecond and so produces a sweeping tone around 1000 Hz.

Let's take a look at a flanger module. The main controls are the rate and depth of the internal LFO and the flanging offset. As before, the controls can be

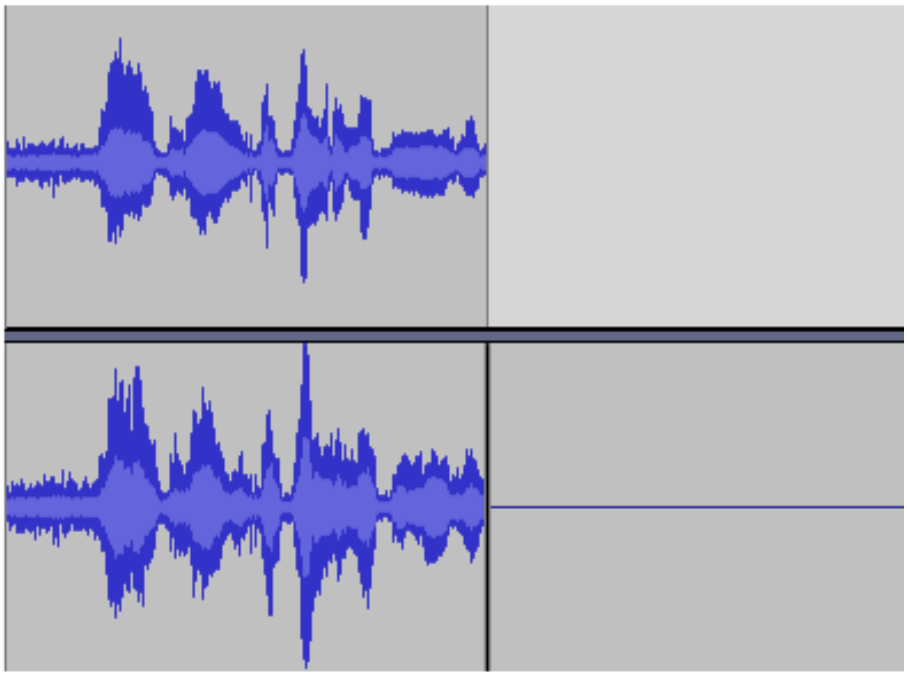

Figure 7.7: Chorus applied to an audio clip. The original sound (upper) is duplicated (lower) and the effect applied. The shape of the wave is affected by the interference of copies of the original signal delayed by 20 milliseconds.

Launch Virtual Modular

Figure 7.8: Virtual modular for a chorus effect.

Figure 7.9: Youtube video of pink noise with increasing 1 millisecond offsets. Note the harmonic noise that emerges as well as the comb shape of the frequency spectrum. Image © Sweetwater.

manually set or voltage-controlled. Try patching up a flanger into a single voice keyboard patch using the button in Figure 7.11.

7.1.5 Phaser

A phaser effect does not apply a delay. Instead, a phaser shifts the phase of frequencies based on the values of those frequencies. As a result, some frequencies are phase shifted behind the original signal while others are phase shifted ahead of the original signal. Typically phasers have multiple stages of phase shifting to magnify the effect. The variation in phase shifting means that while a phaser produces peaks in frequency spectrum like a flanger, those peaks are not harmonically related. A rough analogy would be that a flanger is to a phaser like a single delay is to a reverb: the former cases use a single copy whereas the latter cases use multiple diffuse copies.

A phaser has been added to the speech example in in Figure 7.12, and like chorus and flanger does not noticeably extend beyond the length of the original audio. The effect on the shape of the wave is quite subtle compared to the flanger, reflecting the constructive and destructive interference resulting from the phaser.

Let's take a look at a phaser module. The main controls are the rate and depth of the internal LFO and feedback. As before, the controls can be manually set or voltage-controlled. Try patching up a phaser into a single voice keyboard patch using the button in Figure 7.13.

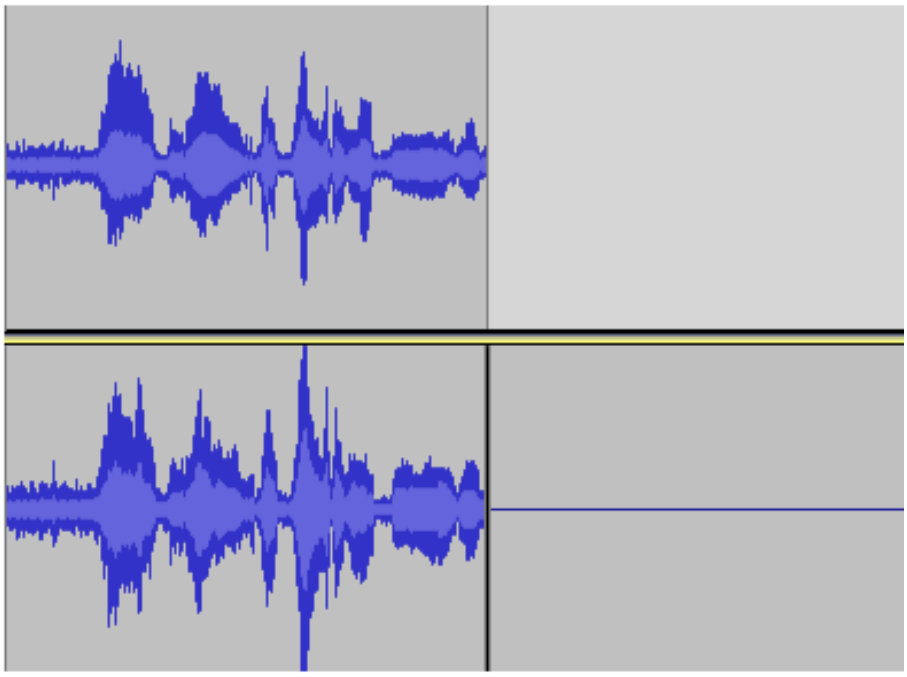

Figure 7.10: Flanger applied to an audio clip. The original sound (upper) is duplicated (lower) and the effect applied. The shape of the wave is affected by the interference of a copy of the original signal delayed by 1 millisecond.

Launch Virtual Modular

Figure 7.11: Virtual modular for a flanger effect.

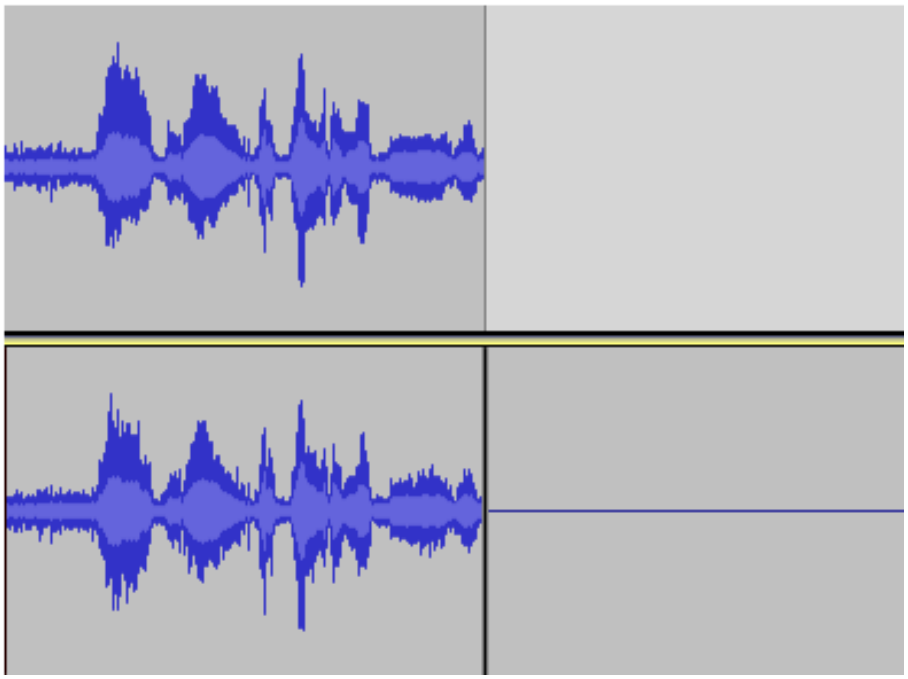

Figure 7.12: Phaser applied to an audio clip. The original sound (upper) is duplicated (lower) and the effect applied. The effect on the shape of the wave is subtle and reflects both constructive and destructive interference.

Launch Virtual Modular

Figure 7.13: Virtual modular for a phaser effect.

7.2 Voltage controlled filters

Voltage controlled filters (VCFs) are an essential component of subtractive synthesis. Subtractive synthesis, as you recall, is characterized by taking harmonically complex waveshapes and then removing harmonic content to create the desired sounds. The opposite approach, additive synthesis, takes harmonically simple waveshapes and adds them together to create desired sounds; however, this becomes complex in analogue circuitry and is better suited to digital computers, which is why subtractive synthesis has historically been the dominant approach to synthesis. Filters, if you haven't already guessed, are a primary method for removing harmonic content, which is why VCFs are an essential component of subtractive synthesis.

7.2.1 Filters are imperfect

As discussed in Chapter 3, timbre is intimately connected to waveshape, and removing harmonic content affects both the shape of the wave and its timbre. You may assume from our discussion of Fourier analysis that a filter will remove selected harmonics completely. If that were the case, we might expect a square wave with all but the first two harmonics filtered to look like Figure 7.14.[3]

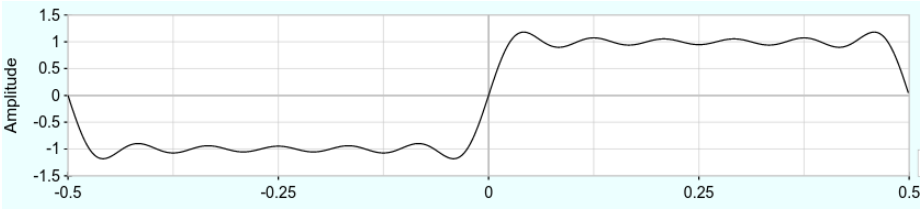

Figure 7.14: Fourier approximation of the first 11 harmonics of a square wave.

Using Fourier decomposition to understand VCFs is misleading in many respects, as will become clear by the end of this section. It is also not practical to build a filter based on Fourier decomposition for two at least two reasons. First, we want our filter to happen in real time, i.e. before a wave has even completed its cycle, which Fourier can't do. Second, Fourier introduces Gibbs error that you can see in Figure 7.14 as peaks where there should be 90 degree angles in the square wave. If we were to use Fourier as the basis of a filter, those peaks would introduce ringing artifacts.

The output of a real VCF filtering out all but the first 11 harmonics is shown in Figure 7.15. The VCF output has two striking differences with the Fourier example. First, the transition from maximum positive to maximum negative

[3]This figure was created using the simulator in Section 3.2 which you can use to further explore these concepts.

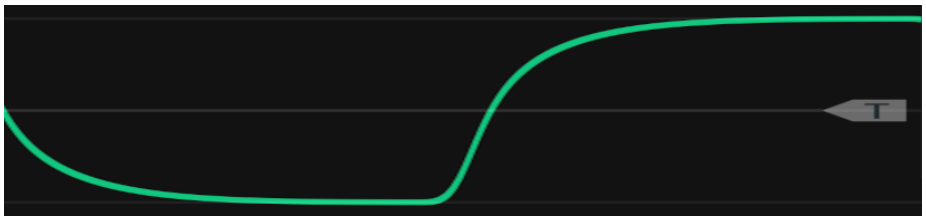

Figure 7.15: VCF output on a square wave that has been aligned and scaled for comparison to Figure 7.14. Note the relatively flat low and high regions of the wave, which imply the presence of higher harmonics.

(and vice versa) is substantially smoother for the VCF, creating rounder edges. Second, the VCF has relatively flat maximum regions of the wave, which imply higher harmonics. By comparing the Fourier and VCF outputs, we can see that the VCF is removing more low frequency content than we would expect (rounder edges) but that the VCF is removing less high frequency content than we would expect (flatter maximum regions of the wave). So a VCF is not like a net that either lets harmonics through or not. Instead, a VCF affects a range of harmonics, but within that range, it affects some harmonics more than others.

Let's take a look at this in some patches that filter white noise. The advantage of filtering white noise is that white noise contains all frequencies, so it is easy to see the shape of the filter's effect. The patches will use two common filters: low-pass filters and high-pass filters. Low-pass filters (LPF) let low frequencies pass through relatively unaffected, and high-pass filters (HPF) let high frequencies pass through relatively unaffected. The effect of each filter is controlled using the cutoff frequency, which defines a range of affected frequencies. Try patching up the two filters using the button in Figure 7.16.

Figure 7.16: Virtual modular for applying low- and high-pass filtering on white noise.

Let's take another look, this time with a square wave and an oscilloscope, which will let us look at the effect on the wave rather than the frequency spectrum. Of course the disadvantage of white noise is that you can't usefully look at the effect Try modifying your patch to use an audio rate LFO and an oscilloscope using the button in Figure 7.17. As you can see, LPF creates curious shapes that occasionally approximate shapes we recognize, like triangle and sine waves (though with reduced amplitude). HPF creates perhaps even more curious shapes that quickly turn into increasingly sharp pulses.

Launch Virtual Modular

Figure 7.17: Virtual modular for applying low- and high-pass filtering on a square wave.

7.2.2 Filters change frequency and phase

To understand why VCFs are behaving this way, we need to understand how they work. The best way to understand VCFs, which are complex, is in terms of the passive RC circuit. When a signal runs through an RC circuit, the circuit leeches out energy from certain parts of the waveshape and returns that energy at other parts, as shown in Figure 7.18. While a full explanation of how the RC circuit works is out of scope,[4] we can nevertheless describe what it does to explain what a VCF does. The overarching idea is that when an RC circuit leeches energy out of a signal, it slows down or speeds up frequency components of the signal and so phase shifts those components. Again, VCFs change more than just frequency spectrum - they also change phase across the frequency spectrum.

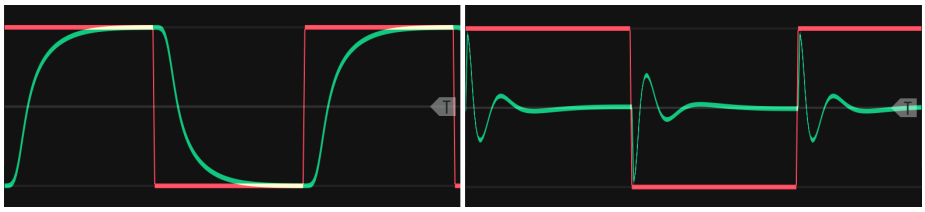

Figure 7.18: Square wave through a low-pass (left) and high-pass (right) filter. Note the symmetry of the difference between the square wave and the filtered signal: the shape of the filtered signal below the square wave in the positive region is the same as the filtered signal above the square wave in the negative region, but flipped.

Let's look first at the LPF case, using the left side of Figure 7.18 as a reference. As the signal increases, the RC circuit leeches out energy until it fills up, at which point it has no effect on the signal. Once the signal starts decreasing, the RC circuit releases energy until it runs out. Sharp signal increases are slowed down, meaning they are phase shifted negatively. Slowing them means removing high frequencies. Here's an analogy that might help. Imagine you're driving on the freeway and a wind blows from ahead whenever you try to accelerate and blows from behind whenever you decelerate. The wind is opposing your ability to change speed. The RC circuit likewise is opposing the signal's voltage changes.

[4] For a good basic explanation of an RC circuit, see this video.

The HPF case on the right side of Figure 7.18 is different. As the signal increases and stops changing, the RC circuit leeches out energy until the capacitor fills up, at which point it has no effect on the signal. Once the signal starts decreasing and stops changing, the capacitor releases energy until it runs out. Flat signals are speeded up, meaning they are phase shifted positively. Speeding them up means removing lower frequencies. Using the wind and car analogy, imagine you're driving on the freeway and a wind blows whenever you try to keep a constant speed.[5] The wind is opposing your ability to maintain constant speed. The RC circuit likewise is opposing the signal's constant voltage.

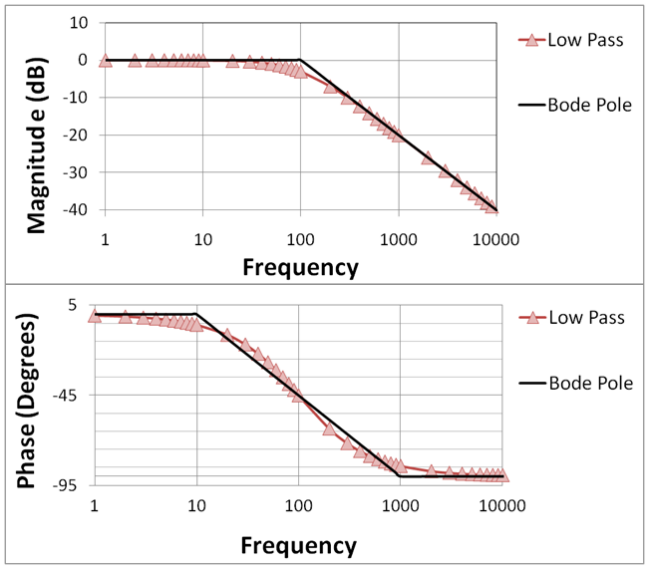

Figure 7.19: A Bode plot for a low-pass filter with a cutoff frequency of 100 Hz. Note the lines marked Bode pole represent idealized behavior of the filter and the Low Pass markers indicate actual behavior. The cutoff point is exactly aligned with a -45 degree phase shift. Image public domain.

The effect of the RC circuit on frequency and phase spectrum can be summarized by a Bode plot, as shown for an LPF in Figure 7.19. The lines labeled "Bode pole" are idealized and the actual behavior of the LPF is indicated by the markers. Let's start with the upper subplot, which shows how the filter reduces the amplitude of the signal across frequencies. The cutoff frequency occurs where the line changes from 0 dB to a downward angle. As you can see, the actual behavior of the LPF is to reduce frequencies even before this point as shown by the markers below the line to the left of the cutoff point. The decrease in amplitude at the cutoff frequency is 3 dB.[6] The decrease to the right of the

[5] More precisely, the wind blows from ahead when you start cruising after increasing speed, and the wind blows from behind when you start cruising after reducing speed.

[6] Recall a 6 dB decrease means amplitude is cut in half.

cutoff point is 6 dB for each doubling of frequency, or 6 dB/Oct. That means the amplitude of the signal will decrease by half for each additional octave above the cutoff frequency. Let's look at the lower subplot, which shows phase shift how the filter changes phase across frequencies. As previously explained, the LPF slows down higher frequency components of the wave, and creates a maximum phase shift of -90 degrees for those frequencies. At the cutoff, the phase shift is already -45 degrees.

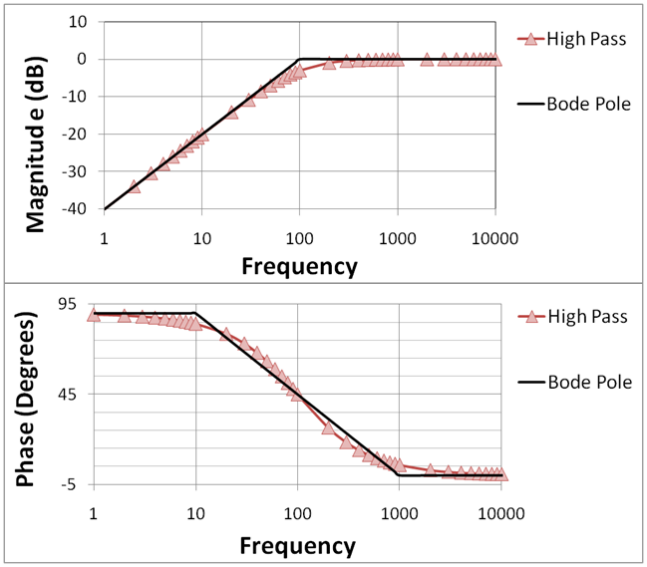

Figure 7.20: A Bode plot for a high-pass filter with a cutoff frequency of 100 Hz. Note the lines marked Bode pole represent idealized behavior of the filter and the High Pass markers indicate actual behavior. The cutoff point is exactly aligned with a 45 degree phase shift. Image © Brews ohare/CC-BY-4.0.

The Bode plot for an HPF looks very similar to that of an LPF, but reversed as shown in Figure 7.20. As before the range of frequencies affected by the filter extends on both sides of the cutoff point, the signal is already attenuated by 3 dB at the cutoff point, and the decrease in amplitude (this time to the left of the cutoff) is 6 dB/Oct. The main difference is in the sign of the phase shift in the lower subplot. Instead of being negative because the frequencies have been slowed down, the phase shift is now positive because the affected frequencies have been sped up.

7.2.3 Combining filters

In the previous Bode plots, the cutoff frequency marked the location past which the amplitude reduced by 6 dB/Oct. This bend in the frequency response of

the filter is called a pole, and the RC circuit is therefore an example of a 1-pole filter element. To get greater reductions than 6 dB/Oct, multiple 1-pole filter elements can be combined.[7] Each filter element has a distinct pole, so the overall shape of the drop off has bends corresponding to these poles. This means that a 4-pole filter would have four distinct regions of drop off rather than the single region we looked at before. Each region would contribute both to amplitude attenuation and phase shift at a given frequency. Common VCFs range from 1-4 poles and therefore implement 6, 12, 18, or 24 dB/Oct reductions in amplitude. It should now be clear why filters have such characteristic sounds: the many different design choices in a filter, ranging from the selection of components to the arrangement of filter elements, each contributes to a specific effect on both amplitude attenuation and phase for each frequency filtered.

In addition to combining filters of the same type, we can also combine filters of different types. As you might expect, multiple filters working in different frequency directions create complex phase relationships as each independently contributes to amplitude attenuation and phase for each frequency filtered. A band-pass filter can be created by combining an LPF and HPF in series. A band-pass filter mostly lets through frequencies within a range but more strongly attenuates frequencies outside that range. Try making a band-pass filter using two VCFs using the button in Figure 7.21. Since there are now two cutoffs to control to hear the combined filter sweep, hook up an LFO to each VCF cutoff frequency.

Figure 7.21: Virtual modular for combining LPF and HPF filters to make a band-pass filter and animating it with an LFO.

A notch filter[8] can be created by combining an LPF and HPF in parallel. A notch filter mostly lets through every thing but frequencies within a range. Try making a notch filter using two VCFs and a mixer using the button in Figure 7.22. As before, there are two cutoffs to control to hear the combined filter sweep, so an LFO can be used to control each VCF cutoff frequency.

7.2.4 Resonance

Resonant filters have been an important sound in electronic music even before modular synthesizers[9]. We previously discussed resonance in Section 3.2, but in

[7] RC circuits cannot be simply combined, but they remain useful for understanding the additive effect of each element.

[8] A notch filter is sometimes called a band-reject filter.

[9] https://120years.net/wordpress/the-fonosynth-paul-ketoff-paolo-ketoff-julian-strinigino-marinuzzi-jr-italy-1958/

Figure 7.22: Virtual modular for combining LPF and HPF filters to make a notch filter and animating it with an LFO.

terms of real instruments. Electrical resonance is similar and can be understood analogously to waves reflecting off the fixed end of a string to create a standing wave. In the case of the filter, feedback from the frequency at the filter cutoff point "reflects" back into the circuit and interferes with the original signal. As long as the two signals are aligned in phase, they will constructively interfere and amplify the frequencies at the cutoff point. If the feedback increases beyond a certain point, the filter will begin to oscillate and effectively turn into a sine wave VCO. Try patching up a resonant filter using white noise using the button in Figure 7.23. If you crank up the resonance on the filter, you will get a sine wave even though the input is just noise. This illustrates that resonance is a property of the filter itself, not of the input. Try also extending the patch using square wave input to see what resonant filter sweeps sound and look like on a wave. You should see that the cutoff frequency and resonance interact in interesting ways and that resonance can, in some cases, add frequency content back into the sound that the filter had removed.

Figure 7.23: Virtual modular for applying resonant filtering on white noise and a square wave.

Although filter resonance has clear regions where the filter is self-oscillating or not, there is also an intermediate region where the filter begins to oscillate but the oscillation quickly dies out. This is analogous to plucking a string or pushing on a pendulum. Because energy is not continuously being introduced into the system, the dampening in the system brings it to rest. The filter resonance region that acts as a dampened oscillator is used for an effect called pinging. Pinging sends a gate or trigger to a filter in the dampened oscillator region of resonance. That signal excites the filter enough to create a short oscillation that quickly decays. So instead of using a VCO, VCA, and envelop to get a brief oscillation, one can simply ping a resonant filter. This is particularly useful for percussive sounds.

Try to ping a filter using the button in Figure 7.24. Once the resonance is adjusted to be on the edge of oscillation, you can ping it by sending a pulse from a square LFO. The pitch of the pinged response can be adjusted by the

filter's cutoff. Because the shape of the pulse used to ping a filter represents energy over time, the shape of the pulse will affect how the filter rings in response to the ping. Any kind of pulse can be used to ping a filter, including envelopes.

Launch Virtual Modular

Figure 7.24: Virtual modular for applying resonant filtering on white noise and a square wave.

7.3 Waveguides

Time-based effects and filters can be combined to make a synthesis technique called a waveguide. The simplest waveguide is known as the Karplus-Strong algorithm. The basic idea of a waveguide is somewhat similar to a resonant filter. A resonant filter has a feedback path that cycles energy back into the circuit and sets up oscillation. A waveguide creates a delayed copy of the signal (often noise), filters it, and the feeds back to mixes it in with the incoming signal as shown in Figure 7.25. Unlike a resonant filter, however, waveguides are highly influenced by the nature of the delay. The time-based effects discussed in this chapter apply to waveguides. For example, the flanger effect of creating harmonics in a short delay will affect the harmonics of a waveguide. While waveguides can be used to create a wide range of sounds, Karplus-Strong is typically used for string pluck sounds.

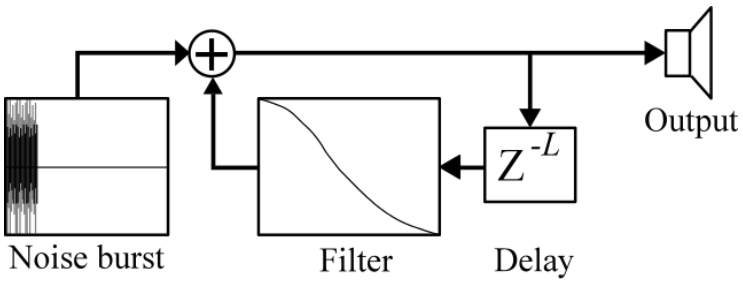

Figure 7.25: A diagram of the Karplus-Strong algorithm. The plus symbol mixes the feedback path with the incoming signal. Image © PoroCYon/CC-BY-SA-3.0.

Try to create a guitar string pluck by waveguide using the button in Figure 7.26. One of the tricky things about waveguides is that they have many parameters that depend on each other. The initial sound source will bleed through more to the final sound if you use a wider envelope that lets more of it through.

The delay time will set the fundamental and have some effect on the amount of vibration. However, much of the vibration, i.e. how long it takes the vibration to decay, is controlled by the filter cutoff, the mixer controlling feedback from the delay loop, and the amount of initial sound. It's best to go back and forth between these settings to see what effect they have to get the sound you want.

Figure 7.26: Virtual modular for applying resonant filtering on white noise and a square wave.

Part III

Sound Design 1

Chapter 8

Designing a Kick Drum

This chapter focuses on sound design for synthesizing a specific sound, a kick drum, as well as the problem solving that accompanies that process. The goal is to outline and emphasize the thinking process rather than presenting a recipe for a kick drum sound. By understanding the thinking process, you should be able to apply it to new sounds that you want to synthesize in the future.

Before proceeding, a quick disclaimer on the term "sound design" and the narrow usage of that term here. Sound design is a widely-used term that has deep roots in the film and theatre industries. It encompasses all aspects of creating sound, including performance and editing. Even in the music production industry, sound design includes editing, mixing, and processing recorded sound. Herein sound design refers to only the creation of sounds and effects using modular synthesis.

8.1 Problem solving for sound synthesis

Let's revisit and expand upon the problem solving approach introduced in Chapters 1 and 4. The general problem solving stages are (Polya, 2004):

- Understand the problem
- Make a plan
- Implement the plan
- Evaluate the solution

The make/implement a plan stages of problem solving involves building a model by connecting modules together with patch cables, as we've covered in the last few chapters. While reasonable at first glance, the above problem solving approach is much more useful if it is expanded and elaborated in the context of

sound design. Below is my interpretation of general problem solving (Polya, 2004) and computational thinking principles (Papert, 1980) (Anderson, 2016) as they fit into these stages.

8.1.1 Understand the problem

The problem is to create a specific sound. It may be a natural, physically produced sound, a sound of unknown origin from a recording, or a sound that exists only in your mind. The key questions are then:

- What is the sound? What are its defining characteristics? How does it differ from other sounds?
- How is the sound made in real life? Do you understand the principles and mechanism for how the sound is created? If it doesn't exist in real life, how could it be made if it did?
- Is all the information needed to make the sound available? If it is not available, how could you get the information you need?

8.1.2 Devise a plan

This stage is the most complex because you must think about your thinking[1]. It's best to try to stay flexible and evaluate multiple alternatives so you can pick the most promising one. This will prevent you from wasting effort on suboptimal strategies.

There are many ways to solve a problem, and often a combination of approaches is required. Approaches in the list below require different levels of information and expertise.

- **Guess and check** is the simplest strategy, but if the guesses are uninformed, it is the same as blind search. Given the number of ways modules can be combined, there are too many possibilities for a blind search to be efficient. However, informed guesses can be efficient, and thoughtful exploration can lead to brand new ideas.
- **Looking at related problems** is sometimes called reasoning by analogy. The idea is to find a known solution to a similar problem and then adapt the solution to the current problem. To do that, you need to identify what parts of the solution should carry over and what parts shouldn't (also known as abstraction and generalization).
- **Decomposing the problem** is a very powerful strategy that is easily combined with other strategies. The key idea is to divide a complex problem into smaller problems that are easier to solve. Because there's often

[1] Thinking about your thinking is often called metacognition.

more than one way to decompose a problem, it can help to think deeply about different decompositions. For example, related problems might suggest decompositions, and a particular decomposition might create smaller problems you already know how to solve.
- **Using a model of how sound is made in real life** can help identify modules that correspond to that model. It can also help with deciding how to decompose a problem and identifying related problems.
- **Working backwards** is a data-driven strategy that easily combines with other strategies. Instead of using your perception of the sound or preconceived notions about the sound, this strategy analyzes the properties of the sound itself to identify key features like frequency spectrum and dynamics.

8.1.3 Carry out the plan (and replanning)

This stage involves both model building and keeping track of what's working and what isn't. Model building has been well elaborated in the last few chapters, but it's worth reminding that sometimes the model idea is correct but parameters, i.e. knob positions, aren't quite right. Additionally, sometimes the module type may be correct, but the specific choice in module isn't quite right for the effect to be achieved. For example, in a previous patch we looked at a flanger module that has an internal LFO that can't be completely disabled, so if your patch idea required flanging without an LFO controling the offset, this would be a suboptimal choice. So it's important as you carry out the plan that you consider how issues like these might trick you into thinking that your solution isn't good when it actually is but just needs a little tweaking.

Keeping track of what working and what isn't can get rather involved for any amount of extended problem solving. Here are some strategies to help with managing alternative solutions to a sound design problem.

- **Having evaluation criteria** is absolutely essential because otherwise, you won't know if you succeeded. You also won't be able to say whether one solution is better than another. The simplest criteria might be a single scale, e.g. 1-10, whereas better criteria can have scales for the defining characteristics of the sound.
- **Keeping notes** like a list can help keep track of what has been done and prevent repeating solutions. If each solution is documented with its evaluation, it makes it easier to identify common solution elements that seem to be beneficial or not.
- **Eliminating possibilities** can help you avoid unpromising directions. If every approach with a certain element hasn't worked well, then you might consider avoiding those elements in future attempts.
- **Using symmetry** can similarly help you understand what elements are exchangeable in a solution and might help generate new approaches. For

example, you might have an approximate solution but realize that a module or two could be substituted with something that has a similar function for a better effect.

8.1.4 Evaluate the solution

Because this is a design problem, there's never a single absolute solution. It's best to consider solutions relative to each other in terms of evaluation criteria. Additionally, if resources are limited, it makes sense to consider solutions in terms of efficient use of resources, i.e. does one solution use fewer modules than another. Regardless of whether the solution is better than one you already generated, its worthwhile pausing to consider what you can learn from this example both in terms of the current problem as well as other potential problems.

It may be obvious that the stages of problem solving blend into each other and that the process of solving problems is iterative. Especially in design problems, there is never an absolute end to the process because there is never a single absolute solution. Thus the process iterates until you are either satisfied with your best solution or run out of time.

8.2 Reviewing previous kick drum patches

Let's review the kick drum patches from previous chapters to outline the thinking behind them and consider how they could be improved. The evaluation criteria for the kick drum sound is that it should be realistic and clean (not muffled). This review in some sense reflects the notes you might keep while carrying out your sound design plan.

8.2.1 Sine with envelope

The first patch was presented in Section 5.2.1, and it used a sine wave with an envelope to control its amplitude.

Understanding the problem. The only identifying characteristic considered was that kick is a low frequency sound. We know an idealized membrane (like a drum head) is inharmonic from Section 3.3 so it doesn't make sense to use a harmonically rich waveform.

Devising a plan. Looking at related problems of generating sound, we used the fundamental patch of an oscillator through an envelope to control its amplitude to create a single pitched note. Our model of the sound in real life considered the vibration of the drum after the head was hit by the mallet.

Carrying out the plan/Evaluating the solution

No particular care was given to these stages because the goal at the time was to demostrate clocks as a sequencer. However, we can do this now by reviewing the patch itself and the associated sound, both of which are presented in Figure 8.1. The sound gets a 5 for realism but a 10 for cleanliness. Altogether this was a pretty naive attempt.

Figure 8.1: Kick made using a sine wave with an envelope to control its amplitude.

8.2.2 Sine with an envelope plus noise burst

The second patch was in Section 6.5, and it extended the first by mixing in red noise with an envelope to control its amplitude.

Understanding the problem. The new identifying characteristic was the sound of a mallet contact on the membrane in addition to the resonant sound of the drum. Intuitively, this would be a bunch of short lived frequencies, i.e. noise.

Devising a plan. This patched used decomposition by breaking down the kick sound into two components and using the existing patch as a solution for one of the components. Our model of the sound in real life added the sound when the head was hit by the mallet.

Carrying out the plan/Evaluating the solution

As before, no particular care was given to these stages because the goal then was to demonstrate different types of noise. Reviewing the patch itself and the associated sound presented in Figure 8.2, the realism in the sound has improved a bit (6) but also dropped in cleanliness (8). Of course it might be possible to improve on it by tweaking the noise envelope or the mix levels a bit more.

Figure 8.2: Kick made using a sine wave with an envelope to control its amplitude and mixed with red noise through an envelope to control amplitude.

8.3 Alternative approaches

We need strategies to develop alternative approaches if we're going to improve on these kick drum patches. The most relevant problem solving stages to consider are understanding the problem and devising a plan.

8.3.1 Improving our understanding of the problem

Let's return to the understanding the problem stage: can we come up with more defining characteristics of the sound or elaborate on how the sound is made in real life? Personally, I can't come up with anything beyond what's already been said. However, we can turn to the Internet to see what research has been done on the acoustics of kick drums for more ideas.

One acoustic study found that the frequency of the kick drum isn't constant as we assumed but rather changes very quickly from a higher pitch to a lower pitch (a drop of about 5 Hz)(Fletcher and Bassett, 1978). The presumed mechanism for this behavior is that the drum head's tension is very high just after it has been struck, and so it vibrates at a higher frequency until is resumes its initial position. We can operationalize this in modular using an envelope to control the pitch of the sine wave so that it goes high and then decays to the fundamental pitch. Similarly we can envelope the pitch of the noise. Since our noise module doesn't have frequency control, we can control the pitch of the noise using a filter on the noise module, and controlling the cutoff of the filter with an envelope. This raises another idea: we probably don't need low frequency noise below the fundamental, so let's use another filter, a high pass filter, to remove that noise to get a cleaner kick. Try patching these improvements into the last kick patch using the button in Figure 8.3.

Figure 8.3: Virtual modular for a kick drum using a sine mixed with noise where both have frequency controlled, either by envelopes or by filters.

A version of this patch with my specific settings is shown in Figure 8.4 along with a recording of the sound. It seems much improved compared to the last version in realism (8) and cleanliness (9).

8.3.2 Devising new plans

What more can be done with this patch? The problem has been decomposed into generation of a fundamental and a noise burst, each with their own envelopes to control dynamics. If we consider related problems, we recently learned two ways of generating a fundamental that doesn't use a VCO. First, we can ping a resonant filter. We saw in Section 7.2.4 that pinging can give a somewhat natural plucky or percusive sound because the ping acts like a dampened oscillator. Second, we can use a waveguide as shown in Section 7.3. The waveguide also created a plucky sound, but that was much more string like. If we consider the symmetry between these two approaches and a VCO sine wave with an envelope,

Figure 8.4: Kick made using a sine wave with an envelope to control its amplitude and mixed with red noise through an envelope to control amplitude, with additional envelope control of initial frequency of the sine and band-pass filtering of the noise.

we realize that we can substitute these into our last patch to see if the sound is improved. Try patching a resonant filter in place of the VCO sine wave with an envelope using the button in Figure 8.5.

Figure 8.5: Virtual modular for a kick drum using a resonant filter mixed with noise where both have frequency controlled, either by envelopes or by filters.

A version of this patch with my specific settings is shown in Figure 8.6 along with a recording of the sound. It sounds much more realistic, perhaps because of the way the resonant filter handles the dampened oscillation (9) However it is also a bit muffled and looses out on cleanliness (6). One of the problems we seem to have with this approach is a lack of control of the frequency of the fundamental. The cutoff and resonance parameters affect both the initial transient as well as the body of the kick, whereas with a sine wave there was better separation in control of these two phases.

Let's see if the waveguide would be any better than the resonant filter, both in terms of sound and our ability to control the sound. Try patching a waveguide in place of the resonant filter using the button in Figure 8.7.

A version of this patch with my specific settings is shown in Figure 8.8 along with a recording of the sound. The sound seems worse and is more like a bass string pluck than a kick drum. So it sounds realistic but the wrong instrument (4), and the sound is a bit muffled too (6). As with the resonant filter, it seems the problem in making this patch work towards our goal is the difficulty in controlling specific aspects of the sound, since any one parameter changes seems to affect multiple aspects of the sound at once.

Personally my favorite of these patches so far is the resonant filter for the overall realism and quality of sound, but I suspect that in some contexts it would muddy up a mix, i.e. it's not clean enough for some uses.

8.3.3 Working backwards

So far the approach has been to create "a" kick sound, one that scores highly on realism and cleanliness. But what if we wanted to synthesize a kick sound based on a recording of a real kick? We'd need more information to do this, specifically:

- A recording of the sound
- A diagram of the wave shape of that sound

Figure 8.6: Kick made using a resonant filter mixed with red noise, with additional envelopes to control of initial frequency and resonance of the filter and band-pass filtering of the noise.

Launch Virtual Modular

Figure 8.7: Virtual modular for a kick drum using a waveguide mixed with noise where the noise frequency is controlled by filters.

Figure 8.8: Kick made using a waveguide mixed with red noise, with additional envelopes to control of initial frequency and resonance of the filter and band-pass filtering of the noise.

- A diagram of how the frequency spectrum of the sound changes over time (a spectrogram)
- Diagrams of frequency spectrum at key time points in the sound

Unfortunately, the modules currently available in VCVRack/Cardinal are insufficient for most of the diagram needs. There are no modules that produce quality spectograms, and the tools for frequency spectra (like Sassy scope) are too coarse/noisy to give good information. We can use some additional software to help analyze the sound instead, and since this book is written using the R programming language, the analysis below uses that language. You could reasonably install R along with the libraries used below and run the commands yourself with your own sound files, or you may already have access to other software that would create spectrograms and frequency spectra, like Audacity[2].

Figure 8.9: Kick recording used as the reference sound. Sound © Mattgirling/CC-BY-SA-3.0.

The real kick recording we'll use is presented in Figure 8.9. A spectrogram of this recording is presented in Figure 8.10. A spectrogram is merely the frequency spectrum of a sound over time. Because frequency spectra already have two dimensions and this adds a third dimension (time), the amplitude of each frequency is shown using color instead of an axis. The R code below loads the kick sound file and then creates a spectrogram with a maximum frequency of .7 KHz and uses 20 μPa as 0 dB.

```
s <- tuneR::readWave("images/kick2.wav")
seewave::spectro(s,flim=c(0,.7),osc=T,dBref=2*10e-5,heights=c(1,1))
```

The spectrogram shows an initial burst of frequencies, with the most energy at about 200 Hz, followed by a lower range of frequencies with the most energy around 20-30 Hz. The corresponding diagram of the wave shape aligns with these regions and shows the initial burst has higher frequency until around 25 milliseconds, followed by a more regular decaying frequency. These two phases of the sound broadly align with our earlier patches, though our patches had both noise and fundamental beginning at the same time rather than as separate phases.

[2]See Audacity documentation for creating spectrograms and plotting frequency spectrum

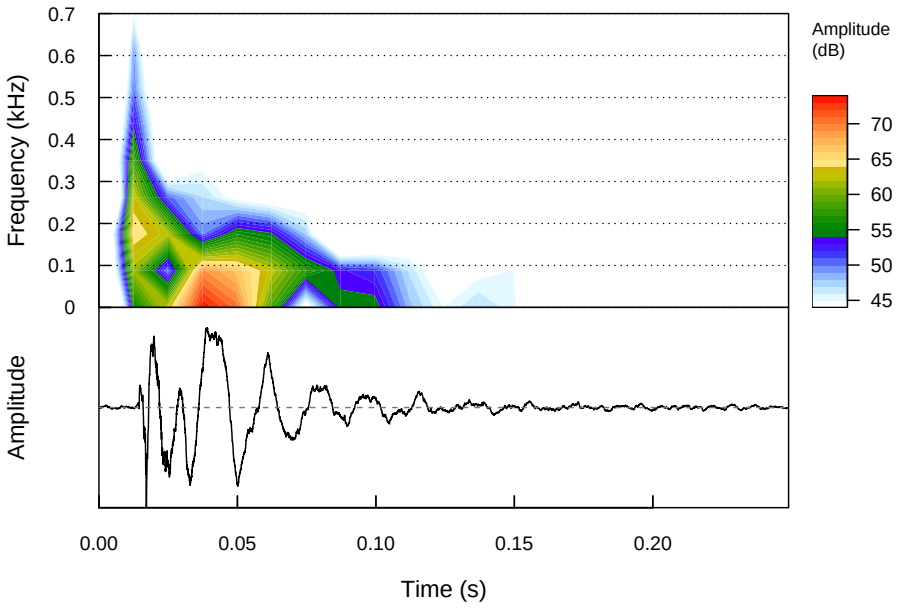

Figure 8.10: Spectrogram of the reference kick sound (upper) and corresponding wave (lower).

A closer examination of the frequency spectrum during the first phase shown in Figure 8.11 indicates a peak around 200 Hz but a fairly wide range of frequencies between 50 Hz and 350 Hz. The shape of the noise burst seems reasonably well aligned with the band pass filter used in the above patches, though some additional tuning of the shape of that filter may be needed to match this spectrum. The code below was used to generate the figure and look for frequency peaks.

```
s.spec <- seewave::spec(s,flim=c(0,.7),from=0,to=0.025,norm=F)
```

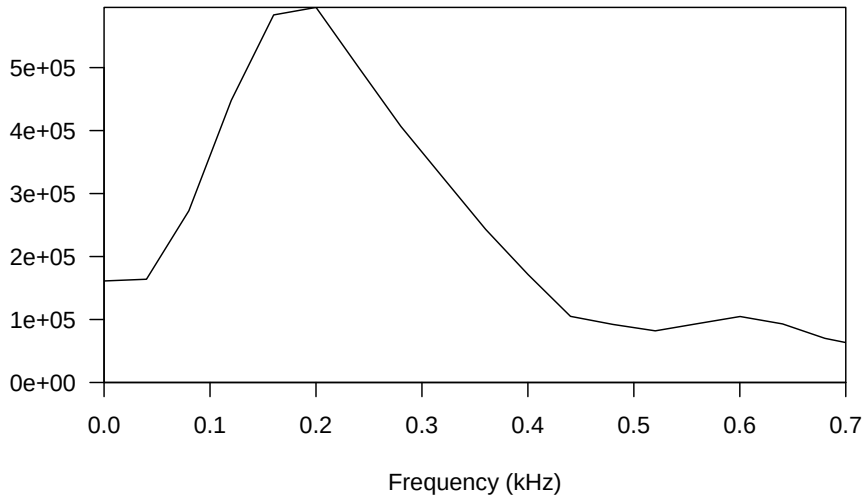

Figure 8.11: Frequency spectrum of the reference kick sound from 0 to 25 ms.

```
seewave::fpeaks(s.spec,f=44100,nmax=4,plot=F)
```

Table 8.1: Frequency peaks in the spectrum of the reference kick sound from 0 to 25 ms. Harmonics of the fundamental are indicated in bold.

Frequency (Hz)	Amplitude (μPa)	Harmonic ratio
200	595,588.06	**1.00**
600	104,763.07	**3.00**
1201	78,152.55	**6.00**

Frequency (Hz)	Amplitude (μPa)	Harmonic ratio
1801	35,439.71	**9.00**

Table 8.1 reveals that the smoothed frequency spectrum in Figure 8.11 is hiding a number of frequency peaks, a few of which are harmonics of the fundamental 200 Hz. This suggests that while there is an initial noise burst, presumably from the mallet hitting the drum head, there is additionally an early harmonic resonant sound happening at the same time. Presumably this is the frequency-shifted sound discussed earlier, caused by the increased tension on the drum head shortly after impact, but a much higher frequency shift than previously patched.

A similar examination of the frequency spectrum during the second phase of the kick sound is shown in Figure 8.12. Unlike the first phase, there are multiple clear peaks in the frequency spectrum, with an apparent fundamental around 50 Hz. The code below was used to generate the figure and look for frequency peaks.

```
s.spec <- seewave::spec(s,flim=c(0,.7),from=0.025,norm = F)
```

```
seewave::fpeaks(s.spec,f=44100,nmax=4,plot=F)
```

Table 8.2: Frequency peaks in the spectrum of the reference kick sound from 25 to the end. Harmonics of the fundamental are indicated in bold.

Frequency (Hz)	Amplitude (μPa)	Harmonic ratio
22	452,508.80	**1.00**
54	2,630,931.40	2.40
94	438,928.40	4.20
125	715,718.10	5.60

Table 8.2 shows the frequency peaks in Figure 8.12. There are no harmonics of the lowest frequency, and interestingly the loudest frequency, 54 Hz, is not the lowest This aligns with our general assumptions about percussion being largely inharmonic.

Let's summarize the findings. The spectrogram reveals two distinct phases of the sound. The first phase, up to 25 ms, blends noise (mostly 0 to 400 Hz) with harmonics. The second phase, from 25 ms on, is inharmonic but with clear peaks.

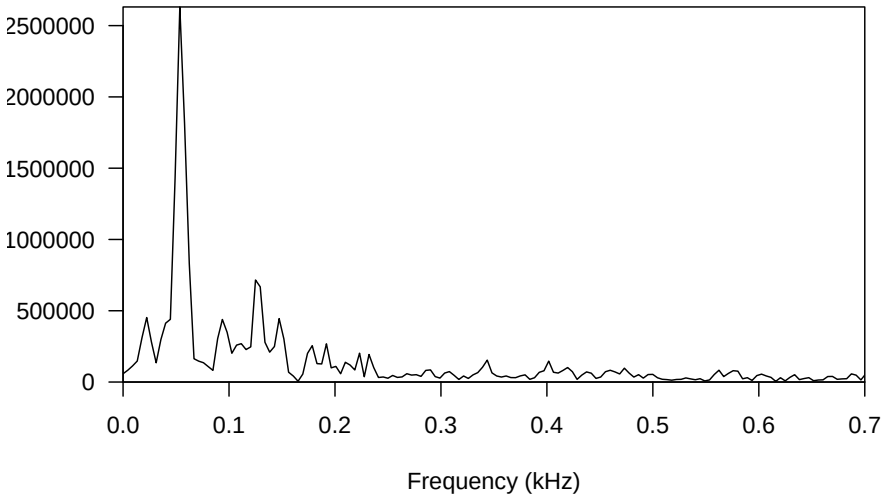

Figure 8.12: (ref:kick-spectrum-late)

There are many ways to model the above findings in modular, and we can get as complex as we want. However, to keep things simple, let's stay relatively close to previous patches and techniques that have already been introduced. Let's use three envelopes for three different sound sources: a band-passed noise source, an early harmonic sound, and a late inharmonic sound. For the non-noise sources, we can use separate oscillators for the partials (additive synthesis). All partials for a source will run through a mixer to produce the composite sound for that source, then all sources will run through a final stage mixer. Because there's clearly some room sound in the recording, run the final mixer output throug a reverb to create a sense of space. Try patching this drum kick using the button in Figure 8.13.

Launch Virtual Modular

Figure 8.13: Virtual modular for a kick drum using two oscillator banks mixed with noise where the noise frequency is controlled by filters.

A version of this patch with my specific settings is shown in Figure 8.14 along with a recording of the sound. It's not as close to the reference sound as I would have hoped, as it both sounds pitched higher in that initial harmonic sound

and thinner in terms of noise frequencies. I suspect that there are extensive inharmonic partials that are characterizing the body of the sound. Nevertheless, the sound is quite realistic (10) and quite clean (8) especially since the reverb can be controlled independently. It's perhaps not fair to compare this sound to the previous kick sounds because it is so different and would likely be used in different contexts. Matching it to the previous kick sounds (rather than the reference recording) would likely involve a tighter decay on the harmonic sound and a drop in frequencies across the sine banks, which are currently pegged around 55 Hz vs. the 40 Hz of the previous kicks.

Figure 8.14: Kick made using two sine oscillator banks mixed with white noise with additional envelopes to control the decay of the sine banks and frequency of the band-pass filtering of the noise.

Chapter 9

Eighties Lead & Chiptune

This chapter explores additional sound design problems using the problem solving approach elaborated in Chapter 8. The first problem is to recreate an 80's-style lead from the show *Stranger Things*. The major strategy used to this problem is working backwards, since the goal is to emulate something that already exists. The second problem is to create a chiptune-style groove reminiscent of arcade games and early computers. For the chiptune problem, the main strategies are decomposition and looking at related problems.

9.1 Eighties Lead

The music of the show *Stranger Things* uses vintage synthesizers in order to match the 1980's setting of the show. The theme song uses an arpeggio as a lead. An arpeggio is a type of broken chord, or chord that is played one note at a time. Arpeggios are a common element in electronic music likely because many synthesizers (and VCOs) have historically been monophonic and so can only play one note at a time. Recreating the *Stranger Things* arpeggio in modular is an interesting sound design problem because it requires working backward from the recording and doing a little detective work to infer how the sound was originally created.

9.1.1 Waveshape

The first step is to determine the waveshape used in the arpeggio. Because the theme is composed using vintage synthesizers, we can expect it to use basic waveshapes like sine, triangle, saw, or square rather than a complex wavetable. Listen to the theme in Figure 9.1 to see if you can identify what kind of waveshape is used, paying particular attention to the brightness of the sound, which indicates higher harmonics.

Figure 9.1: YouTube video of the *Stranger Things* theme song. Image © Netflix.

To me, it starts off a bit dull, like a sine or triangle, then gets brighter around 15 seconds, duller again around 30 seconds, and has a quick run up to brightness around 50 seconds. The brightness suggests that the waveshape is either saw or square. We'll return to the change in brightness in the next section.

Saws and square waves can be distinguished by their frequency spectrum as discussed in Section 3.2. Specifically, saws have both even and odd harmonics, whereas squares have only odd harmonics. Thus a frequency spectrum could help identify which waveshape is being used. Using a frequency spectrum this way allows us to use the working backwards strategy and also use the frequency spectrum as an evaluation criteria.

It is much more convenient to use Audacity than R here because Audacity allows a portion of a track to be auditioned and then frequency spectrum to be computed for that portion. This makes it relatively easy to isolate a portion of the song with just a single note of the arpeggio while it is bright.[1] Figure 9.2 shows the frequency spectrum of a single note of the arpeggio around the 50 second mark.

Characteristics of the frequency peaks are shown in Table 9.1, including the harmonic ratio of each peak with the fundamental. Assume for a moment that these are harmonics even though the harmonic ratios are not strictly integers. Since both even and odd harmonics are present, this spectrum is consistent with a saw wave. However, this pattern is also close to that of two square waves, one with a fundamental at 40 Hz and one with a fundamental at 82 Hz.[2]

[1] To calculate frequency spectrum in Audacity, drag select the region of the audio, audition it using the play button, and then use Analyze -> Plot Spectrum. A high window size is needed for a detailed spectrum, e.g. 4096.

[2] Because 80 Hz is the second harmonic of 40 Hz, the odd harmonics of the 80 Hz square

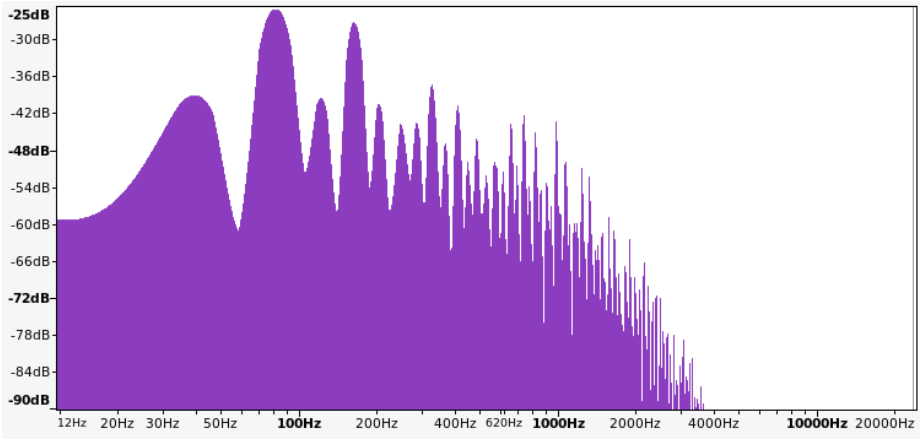

Figure 9.2: Frequency spectrum of a single note from the *Stranger Things* theme arpeggio.

The ambiguity between a single saw and two squares can interrogated by examining the amplitudes in Table 9.1. The amplitude peaks at 82 Hz and 164 Hz could be from constructive interference, specifically a saw at 40 Hz and a square at 82 Hz where the square reinforces the existing harmonics of the saw. Alternatively, these amplitude peaks could potentially be created by two square waves where the square wave at 82 Hz is mixed in more strongly.

Table 9.1: Characterisitcs of the frequency peaks in Figure 9.2.

Frequency	Amplitude	Harmonic Ratio
40	-39.1	1.00
82	-25.2	2.05
122	-39.6	3.05
164	-27.4	4.10
204	-40.5	5.10
247	-43.7	6.18
285	-43.4	7.13
326	-37.1	8.15
367	-46.9	9.18
410	-40.6	10.25

These possibilities can be examined by modeling each in modular. The problem solving approach then is to create a model of each possibility and then use them to reason about how the true frequency spectrum was created. This is a subtle,

wave would match half the even harmonics of 40 Hz.

yet significant shift in using modular: we are now using it to create alternative models in order to reason about the correct model.

Try patching two square waves at 40 and 82 Hz and a saw and square wave at the same frequencies using the button in Figure 9.3. Because the current spectrum analyzers available in Rack are too noisy to closely compare with the target spectrum, you'll need to record output with Audacity and plot spectrum there.[3]

Figure 9.3: Virtual modular for assessing whether two square waves or a saw and a square wave are a closer match for the target spectrum.

Figure 9.4 shows the frequency spectrum of the two square waves and the saw plus square wave model. In both cases, the first three harmonics approximately match the target spectrum in Figure 9.2, but they diverge on the 4th harmonic. Our inability to get the 4th harmonic with either of these models suggests another element is needed. One possibility is a third wave, but we may also be able to further adjust the spectrum in our existing models by applying PWM to the square waves. Try adding PWM to the 82 Hz square wave in both models to match the target spectrum using the button in Figure 9.5.

Figure 9.6 shows the effect of PWM on the 82 Hz square wave. In both cases, the 4th harmonic got boosted but not enough to match the target spectrum in Figure 9.2. However there is a nice flattening of harmonics 5-7 that matches the target spectrum, which suggests PWM should be kept. Thus it seems necessary to add another wave at the 4th harmonic (164 Hz). A square wave is appropriate because it would more precisely target the 4th harmonic, whereas a saw would affect the 4th harmonic relatively less and increase successive harmonics more evenly. Try adding an additional square VCO at 164 Hz square wave in both models to match the target spectrum using the button in Figure 9.7.

Figure 9.8 shows the effect of the new 164 Hz square wave. In both cases, the 4th harmonic looks good and the spectrum is a good match to the target spectrum for the first 8 harmonics. Choosing between them is therefore somewhat subjective, but the all square wave spectrum seems to match the target spectrum a bit more closely to me in the higher harmonics, so let's use that as we move to the next step.

[3]The Bogaudio spectrum analyzers are sufficient for *approximately* matching spectrum at higher quality settings, but these are not currently working in Cardinal and so must be used in VCVRack. High quality spectrum comparisons currently require Audacity regardless.

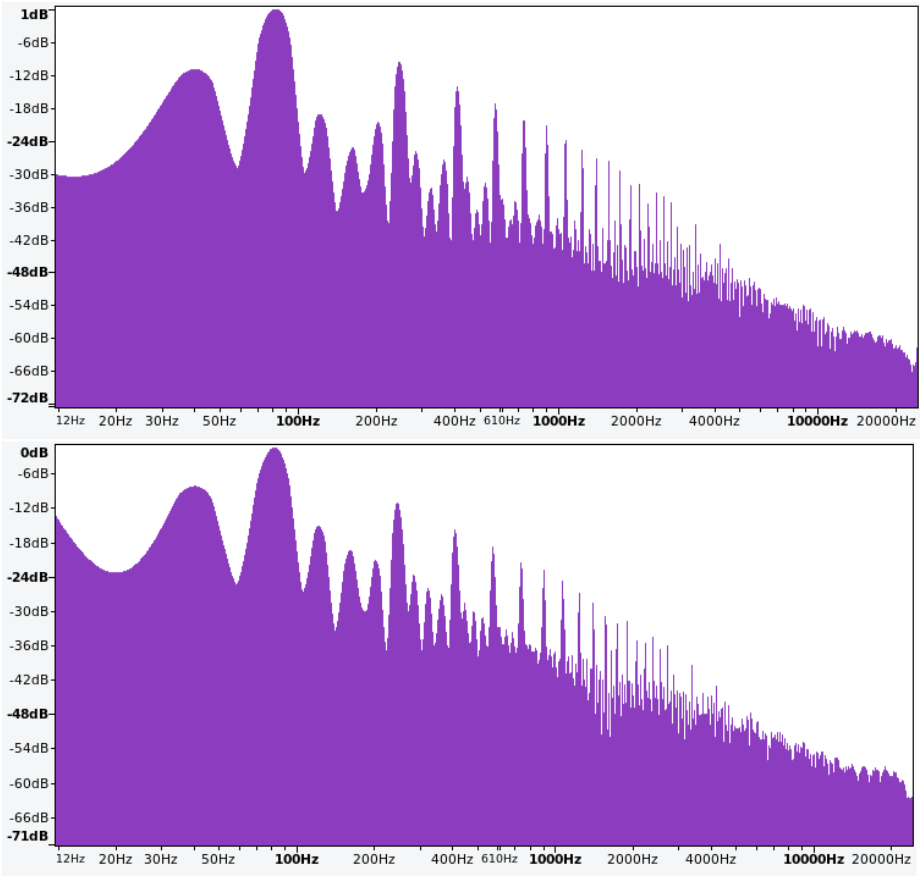

Figure 9.4: Frequency spectrum of two square waves (upper) and a saw and square wave (lower) at 40 and 82 Hz, respectively.

Launch Virtual Modular

Figure 9.5: Virtual modular for assessing whether two square waves or a saw and a square wave are a closer match for the target spectrum, when PWM is applied to the higher frequency square wave.

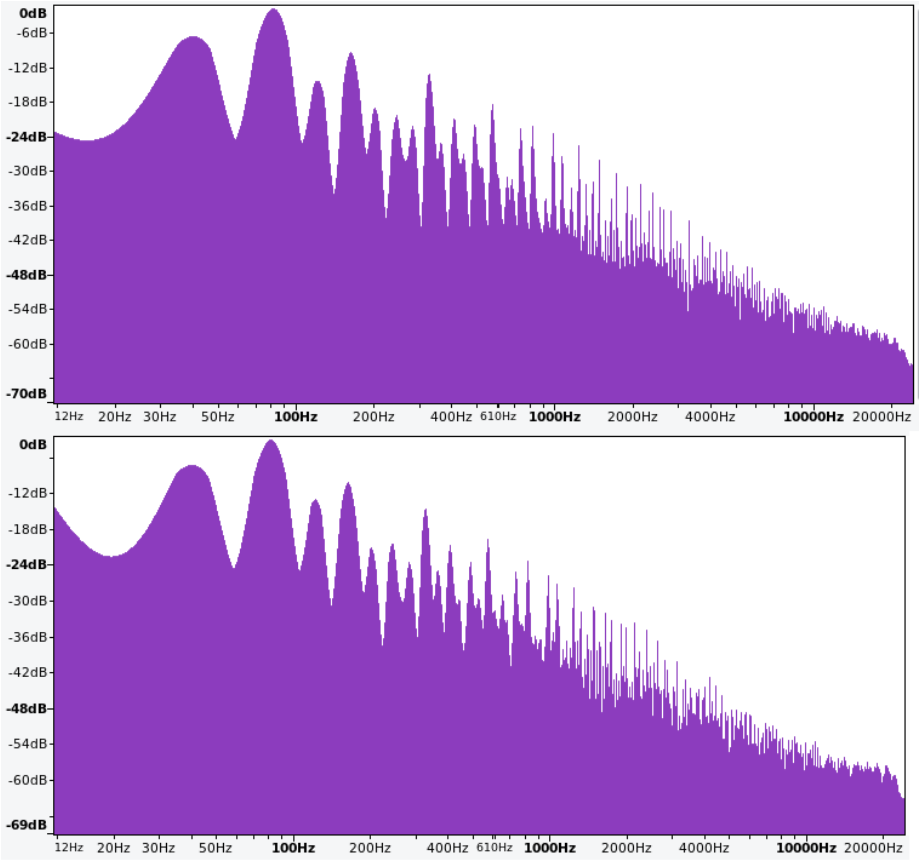

Figure 9.6: Frequency spectrum of two square waves (upper) and a saw and square wave (lower) at 40 and 82 Hz, respectively, when the 82 Hz wave is pulse width modulated with a 36% duty cycle.

Launch Virtual Modular

Figure 9.7: Virtual modular for assessing whether two square waves or a saw and a square wave, both with PWM applied to the higher frequency square wave, are a closer match for the target spectrum when an additional square wave is added at the 4th harmonic.

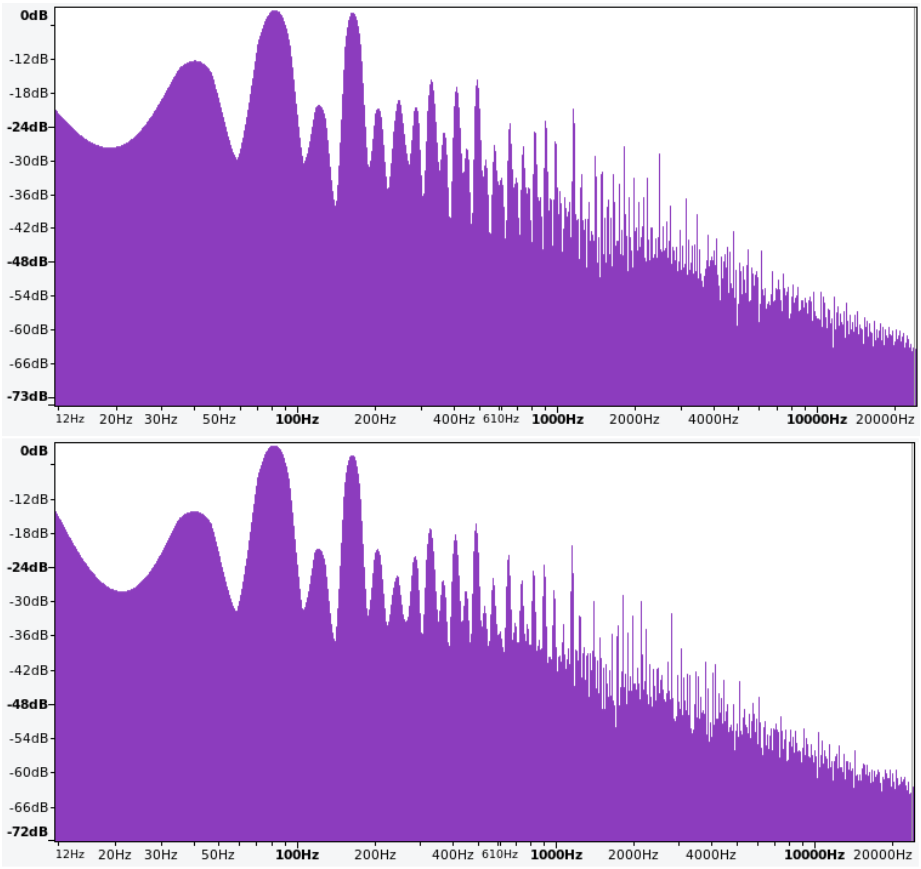

Figure 9.8: Frequency spectrum of three square waves (upper) and a saw with two square waves (lower) at 40, 82, and 164 Hz, respectively, when the 82 Hz wave is pulse width modulated with a 36% duty cycle.

9.1.2 Dynamics

It was previously noted that the arpeggio changes in brightness over time. We know that filter sweeps are a common effect that would change brightness. Since the brightness comes and goes every 15 seconds (1/15 = .07 Hz), we could set up an LFO to change the cutoff on an LPF that would, presumably, create the change in brightness we're looking for. However, there seems to be another more subtle change to the brightness, which is the shape of the envelope. Take another listen to the target recording and see if you hear more pronounced notes when the arpeggio is bright and more slurred notes when the the arpeggio is dull. We can try to use the same LFO to affect the ASDR envelope to create this effect as well.

Before moving on with dynamics, it makes sense to update the patch with sequenced notes rather than a constant pitch. This will facilitate going back and forth between the patch and the target recording when making small parameter changes. Try setting up the arpeggio using the button in Figure 9.9. This will require adding a clock and sequencer, tuning the sequencer steps, and running the VCOs through an envelope. The arpeggio plays up and down over a broken C major 7th chord: low C, E, G, B, C, B, G, and E.

Figure 9.9: Virtual modular for setting up an arpeggio using the final patch from Section 9.1.1.

Now we can add dynamics to the patch as previously discussed. Try adding a VCF and an LFO to control both the VCF and the ADSR using the button in Figure 9.10. Deciding the VCF and ADSR's setpoints and just how much the LFO will affect them can require a lot of back and forth as the changes are fairly subtle.

Figure 9.10: Virtual modular for adjusting dynamics using the arpeggio patch from Figure 9.9.

Altogether the solution seems fairly reasonable, though since the waveshapes were based on a sample of the the spectrum in a particular part of the overall cycle, it could be that the lowest square wave should be a saw in order to sound correct as the filter cutoff cycles up and down. Other possibilities in our current design space include a slight detuning between square waves that changes over time or PWM that changes over time.

9.2 Chiptune

Chiptune is a genre of music so-called because it originated in sound chips used to synthesize music in early electronics like arcade games and computers. These early sound chips had low-fidelity digital implementations of various functions we've encountered in modular, like basic waveshapes, noise generators, and envelopes. Because the sound chips were relatively limited in capability, designers developed various techniques to create bigger and more interesting sounds. One of these tricks, the arpeggio, allowed chips with limited polyphony to emulate it, and thus arpeggio, particularly fast arpeggio, is common in chiptune. If you are not familiar with chiptune, check out the video in Figure 9.11.

Figure 9.11: YouTube video of a chiptune groove. Image © Noise Engineering.

Our design problem is to create a rapid arpeggio reminiscent of chiptune that transposes over time, like Figure 9.11, along with simple percussion in the form of hats and kick drum. Let's use a decomposition strategy and look at related problems along the way, using the following sequence:

- Triad arpeggio
- LFO-controlled PWM
- Secondary sequencer for transposition
- Hats and kick

Building this out incrementally nicely illustrates the power of decomposition into related problems we already know how to solve.

9.2.1 Triad arpeggio

We made an arpeggio in the previous patch, so this that is a related solution we can adapt. A few changes are necessary for chiptune however. First, arpeggios in chiptune are very fast. Second, we need a new chord to work from. Let's go with A minor, so A, B, E. Try patching up this arpgeggio from scratch using the button in Figure 9.12.

Figure 9.12: Virtual modular for setting up a chiptune arpeggio.

9.2.2 LFO PWM

We previously controlled PWM using an LFO in Section 6.3.1, so it is easy to adapt that solution to control the square wave chiptune VCO. Additionally, let's add an ADSR to control the VCA to get more note-like dynamics. Try patching up the LFO into a PWM and ADSR using the button in Figure 9.13.

Figure 9.13: Virtual modular for setting up an enveloped chiptune arpeggio with pulse width modulated by an LFO.

9.2.3 Secondary sequencer for transposition

The transposition element is the most novel technique in this patch, relative to what's been covered so far. Recall that the V/Oct standard means that doubling any voltage will transpose a note up one octave. Therefore, adding any voltage to an existing V/Oct signal will raise the pitch, and subtracting any voltage from an existing V/Oct signal will lower the pitch. We can accomplish adding and subtracting from a signal by using a mixer, since a mixer simply adds two signals together and voltage signals can be negative.

A very simple idea is to use a longer sequence on the transposing sequencer where each voltage either does nothing (0 V) or is the negative of one of the voltages in the arpeggio sequence. Negating the voltage means that the resulting

note will be C[4] To keep things interesting, it's nice to have one or two notes that don't follow this pattern. In the patch below, I used G#, D#, C#, C, G#, D#, E, and C as the transposing sequence, changing the step at each bar. A slowed down video of the two sequencer outputs and their mixed output is shown in Figure 9.14. Try patching up the transposing sequencer and mixer using the button in Figure 9.15.

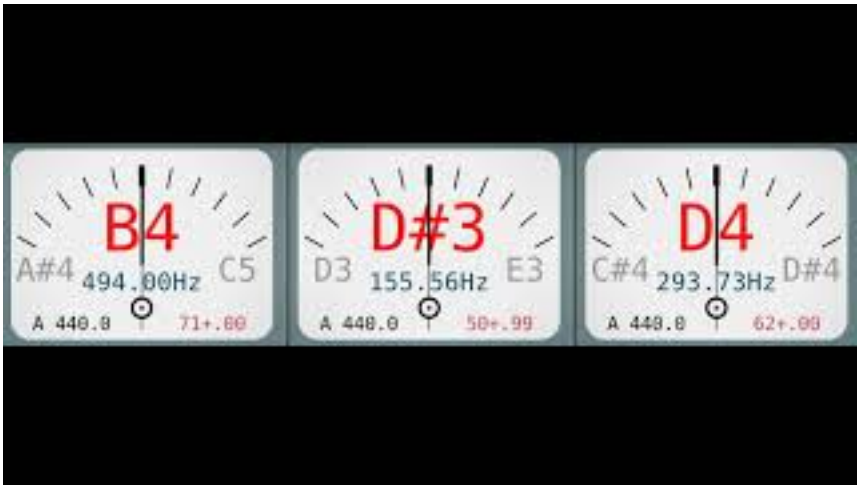

Figure 9.14: YouTube video of a slow-motion arpeggio (left) voltage mixed with the output of another sequencer (middle) to produce a transposed arpgeggio that changes with each bar (right).

Launch Virtual Modular

Figure 9.15: Virtual modular for setting up an enveloped chiptune arpeggio with pulse width modulated by an LFO, transposed by a second sequencer every bar.

9.2.4 Hats and kick

We made hats in Section 6.5 and simple kicks in Section 5.2.1, so these are both problems we know how to solve. A simple hat is just high frequency noise through an envelope/VCA, and a simple kick is just a sine through an envelope/VCA. Both voices are ideally controlled with trigger sequencers, and by using two different sequencers we can use different patterns for each. Try

[4] 0 V is C4 in VCVRack, apparently, though there is no such standard in modular in general. V/Oct is a *relative* standard, not a standard that specifies what frequencies are associated with specific voltages.

patching up hats and kick with their own sequencers using the button in Figure 9.16.

Figure 9.16: Virtual modular for setting up an enveloped chiptune arpeggio with pulse width modulated by an LFO, transposed by a second sequencer every bar, with hats and kick as additional voices.

Part IV

Complex Modules

Chapter 10

Controllers

This chapter extends Chapter 5 by introducing more advanced approaches to sequencing. Sequencing can be considered a kind of memory for musical events where the events can be stored and played back later. Obviously there are gaps between events because not all notes (or beats) play at once. This leads to the idea that sequencing involves both positive space and negative space: positive space where musical events occur and negative space elsewhere.

This idea of positive and negative space can be seen clearly in step based sequencers that allow steps without events, such as the trigger sequencers discussed in Chapter 5. However, a step-based conceptualization of sequencing leads to the limitation that a step should represent the smallest duration event, and such fine granularity naturally leads to more steps being empty. An event-based conceptualization that flexibly represents the start and end of each event, in contrast, has less need to explicitly define negative space. For example, consider a sequence with two notes, the first lasting 2 units, followed by 4 units of silence, followed by the second note lasting 1 unit. A step-based representation would use 7 steps and four of them would be empty. An event-based representation would use two events, one starting at time 1 and lasting 2 units and one starting at time 6 and lasting 1 unit.

In practice, modular sequencing often combines both step-based and event-based elements. The reason for combining both is that each has different strengths. To better appreciate these strengths, let's define three ideal properties of a sequencing approach and use it to evaluate alternatives. An ideal sequencing approach should allow:

- Compact representation of variations, i.e. in a minimal amount of space
- Ease of changes between variations
- Precise control of variations

It turns out that sometimes these properties are at odds with each other, such that different combinations of modules can outperform on some and underperform on others. As we examine different options in this chapter, we'll keep this properties in mind.

10.1 Modifying gates

One of the limitations of pure clock-based sequencing is that every gate starts on the beat. Unfortunately, syncopation, which emphasizes off beats, is widespread in modern music. The on-beat property of clock divisions is illustrated in Figure 10.1. If we consider the clock to represent quarter notes, then the gaps between clock gates are eighth note off beats, i.e. if we count 1-and 2-and 3-and 4-and, the off beats are 'and'.

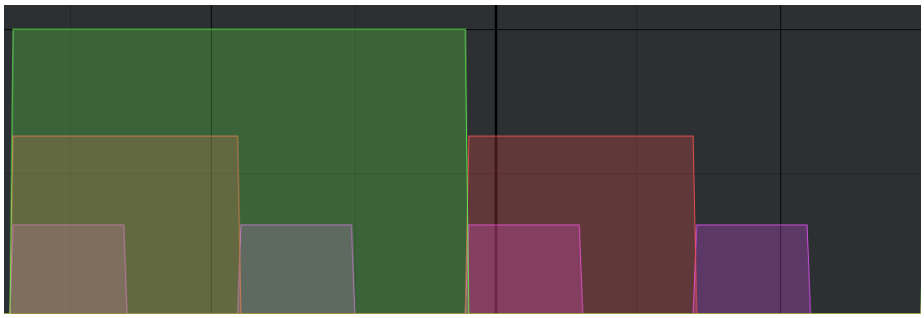

Figure 10.1: Four gates from a clock (small) overlaid with two gates from a /2 clock division (medium) and further overlaid by one gate from a /4 clock division (large). If we consider each clock gate as defining a quarter note, then all three gates occur on the first quarter note and the first two occur on the third quarter note, so the three gates are aligning only on beats.

Let's build a basic patch with clock divisions that illustrates this on beat property both in sound and visually on the scope. To keep things simple throughout this chapter, we will modify this percussion-oriented patch and note differences with sequencing pitches. All the modules and concepts in this patch were introduced in previous chapters, except for a new three-voice percussion module that can produce various sounds. Try patching up this basic percussion patch with clock divisions using the button in Figure 10.2.

Launch Virtual Modular

Figure 10.2: Virtual modular for a basic percussion patch using clock divisions.

In light of the ideal properties for sequencing, the basic clock division approach allows both compact representation in terms of divisions and ease of changing between variations by changing divisions. However it lacks precise control because off beats are inaccessible. Other related complications include increasing the number of hits on a particular beat, sometimes called a roll and skipping a beat. All three of these variations can be approached by modifying existing gates, a concept that was first introduced in Section 5.2.3.2 for sequencing note duration. Since sequencing note duration is identical to sequencing gate duration, we'll look at only the other two techniques here.

Hitting an off beat can be accomplished using a gate delay module. A gate delay module receives an incoming gate (sometimes a trigger) and then create another gate at some time delay. If the time delay corresponds to the length of a beat, then the delayed gate will appear on the off beat. Try extending the last patch with a gate delay to move the open hat to the off beat using the button in Figure 10.3.

Figure 10.3: Virtual modular for a percussion patch using clock divisions and a gate delay to hit an off beat.

With respect to the three ideal properties, the delayed gate approach (combined with clock divisions) is still compact, a bit less easy (because setting the delay is a bit fiddly), and more flexible, though the offset is the same for each clock or clock division. Thus if we want variations of off beat or rolls, we need additional tools.

Drum rolls can be accomplished using a gate multiplier. Gate multipliers will generally need a control voltage telling them how many gates to make, which correspond to the number of hits in the roll. Thus an additional sequencer is needed to supply this control voltage. Try extending the last patch with a gate multiplier and associate sequencer to give the kick a double hit on the first beat using the button in Figure 10.4. Because the kick time is 4:4, the sequencer only needs four steps.

Figure 10.4: Virtual modular for a percussion patch using clock divisions, a delayed gate for an off beat hit, and a sequenced gate multiplier for multiple hits on the beat.

Returning to the three ideal properties, the gate multiplier approach (again

combined with clock divisions) is a bit compact because it relies on another sequencer, a bit less easy (because setting the multiplier voltage is a bit fiddly), and more flexible because it allows different drum rolls on each step. Just as a sequencer was used to control the gate multiplier, a sequencer could be used to control the gate delay offset of the previous patch or drop a step by controlling the length parameter, as shown here.

It's clear that adding more sequencers to control these parameters increases flexibility but also makes the overall control less compact and changing between variations less easy. For example, if one wanted to replace a skipped step with a triple beat, one would have to adjust the length sequencer for a non-zero gate length, then adjust the multiplier sequencer to create multiple hits on that step, i.e. one would have to adjust multiple parameters for that step, and these parameters are spread across different sequencers. Is this suboptimal? Let's compare against other solutions before making judgment. One obvious alternative is to create the same pattern using standard step sequencers without any clock divisions. Try updating the last patch with a step sequencer for the off beat hat and the double beat kick using the button in Figure 10.5. Because the kick time is 4:4, the sequencer only needs four steps.

Figure 10.5: Virtual modular for a percussion patch using step sequencers for an off beat hit and multiple hits on the beat.

Contrast this approach with what we've done so far. The step sequencers are somewhat compact if the pattern can be decomposed into the smallest repeating loops, though not as compact as a clock divider. They are easy to change in some ways but not others. For example, if we decide we want a triple hit on a beat rather than a double, we need to increase the step resolution from two steps per quarter note to three steps per quarter note, which requires changing the entire kick pattern. Conversely, step sequencers have a high level of precision *if* one accepts increasing the number of steps arbitrarily to account for off beats, swing, or dilla, but this then sacrifices compactness. Note that the previous approaches with gate delays and multiplications achieved these results to an arbitrary degree without additional loss of compactness, i.e. we could have a 10 hits instead of 2 with the same gate multiplier patch.

10.2 Making gates with logic

Suppose we wanted to select the offbeats in our current pattern. Looking at Figure 10.1, the ofbeats are simply where the the beats are *not*. So if we had a way of specifying "not beat," then we'd be able make a gate for the off beats.

Boolean logic provides a way of specifying "not beat" and can further be used for more complex beat specifications.

Although it may seem intimidating at first, Boolean logic has just three basic operators, AND, OR, and NOT. AND means two things happen together, OR means at least one thing happens, and NOT means something didn't happen. If we represent something happening (like a gate) as 1 and not happening as 0, then these basic operators are summarized in Table 10.1.

Table 10.1: Basic boolean logic operators on signals S1 and S2. Note that AND and OR consider both signals but NOT considers only one or the other.

S1	S2	AND(S1,S2)	OR(S1,S2)	NOT(S1)	NOT(S2)
1	1	1	1	0	0
1	0	0	1	0	1
0	1	0	1	1	0
0	0	0	0	1	1

Try updating the basic percussion patch with a NOT module to put the open hat on every off beat using the button in Figure 10.5. We'll build on this logic in the following patches.

Figure 10.6: Virtual modular for a percussion patch using clock divisions and the NOT operator to create gates for all off beat hits.

We can use logic to create other gates using combinations of the basic operators. For example, consider every other offbeat in Figure 10.1. These offbeats occur when the clock is not present and the /2 division is present. Try updating the last patch with an AND module to implement this logic using the button in Figure 10.7.

Figure 10.7: Virtual modular for a percussion patch using clock divisions and combining the NOT and AND operators to create gates for every other off beat hits.

Let's get even more specific with logic to match what we did previously with the gate delay, which was a single off beat. Since the last patch used every other off

beat, we can use an additional operator to select just one of those beats. Again looking at Figure 10.1, we see that we can achieve this by using AND with our existing logic and next larger clock division. Try updating the last patch to implement this logic using the button in Figure 10.8.

Figure 10.8: Virtual modular for a percussion patch using clock divisions and combining the NOT and AND operators to create a single gate per bar, matching the previous delayed gate behavior.

Let's consider the three ideal sequencer properties, in terms of logic. The first application of NOT was quite compact, ease to change, and precise. However, as the logic became more complex, the sequencing became less easy. Contrast the last logic patch that needed three or four logical operators to what was previously accomplished by using a single gate delay. It appears that clock divisions with logic are most suited to regular repeating patterns, and that when patterns become less regular, other options may be easier.

10.3 Adding/removing gates with probability

Generally speaking, the more complex sequencing we want (i.e. irreducible to a simple pattern), the less compact our control will be. We can get around this limitation if we give up precise control of the sequencing by turning to probability. A probability specifies of the likelihood of an event, like a gate. In our current approaches, a gate either happens or it doesn't. However, if we assign some probability to a gate happening, more complex patterns emerge because sometimes the gate will happen and sometimes it won't.

Perhaps the easiest way to think about probabilistic gates is to think of them like coin flips. If we flip a coin and get a head, then the gate will be produced, otherwise no gate will be produced. How can we simulate a coin flip in modular? One way is to generate a random voltage and then compare it to a reference value. If the voltage is above the reference value, we count it as a head and produce a gate. We've already seen how to create random voltages with noise generators. The new idea here is a comparator, which is a module that accepts an incoming voltage V, compares it to a reference voltage R, and produces a high signal if $V > R$.[1] Because different noise spectra have different distributions of frequencies (and thus voltages), different noise spectra will give us different probabilities for the same reference voltage.

[1] This approach is chosen for pedagogical reasons. Various modules offer built-in random number generation that is more flexible than combining noise sources with comparators.

We can use the probabilistic gate idea to sequence all the percussion voices in our ongoing example. Since probabilistic gates would be completely random by definition, we'll structure them a bit using logic: a NOT to make two different voices alternate and an AND to combine two different noise signals for a lower probability. Try updating the last patch to drive all gates with probability and logic using the button in Figure 10.9.

Figure 10.9: Virtual modular for a percussion patch using probabilistic gates slightly structured with NOT and AND operators.

The output of this patch is much more complex than we could reasonably achieve using other sequencing methods, but it is also uncontrolled and not very musical, even when shaped a bit by logic. An alternative to randomly creating gates is to take an existing pattern but randomly drop gates. This can be implemented as simply as using noise source, a comparator, and an AND module that receives the output as well as the output for the normal gate. However, to get the random gates to align with the standard clock, we'll use a new module, a sample and hold module. A sample and hold module, when triggered, stores whatever voltage is present at its input and outputs that voltage until a new trigger is received, ignoring whatever input voltages occur in the meantime. By connecting a random gate to a sample and hold and triggering it with a clock, we can synchronize the random gate with our clock and have a clean signal for deciding to drop a gate or not. Note that if we wanted to produce V/Oct control voltage for random notes, we could similarly use sample and hold module directly connected to our random source, i.e. without a comparator. Try updating the previous logic patch that selected every other off beat to further consider a random gate using the button in Figure 10.10.

Figure 10.10: Virtual modular for a percussion patch using a probabilistic gate and logic to decide when to drop a gate from an otherwise non-probabilistic sequence.

As shown by these patches, probability can be used to create interesting variations in several ways. These methods are fairly compact and can switch between variations easily, however, there is a loss of control. Modules specifically designed for randomness typically have additional parameters that allow the user to define how random events should be, which can help mitigate this loss of control.

10.4 Speed variable clocks using LFOs

Recall one of the earlier challenges raised was to increasing the number of hits on a particular beat, skipping a beat, and accessing all possible beats (including off beats). We've tried to solve this problem using gate delays, logic, or an increased step resolution to access off beats and gate multipliers and increased step resolution to increase the number of hits on a particular beat. In each case, one technique would solve some problems but not others, with the exception of step resolution which has the problems previously discussed.

It turns out that we can achieve all three of these goals by running speed variable clocks, one for each percussion voice where we want to have multiple beats, skipping, or offbeats at different times. The idea is that when we want multiple beats, we speed the clock up; when we want to skip a beat, we slow the clock down almost zero, and that when we want to access an offbeat, we use the same sequencer we're using the speed up and slow down the clock. Because a clock is just unipolar rectangular waves, an LFO can be used as a speed variable clock. One of the additional benefits of this approach is that skipping a beat and doubling a beat are controlled by the same parameter, frequency. Contrast this with the previous approach using gate multiplication, where skipping a beat required an additional sequencer to shrink gate length to zero. We still need a master clock however to keep the LFOs in sync, which can be done by sending clock to their reset inputs, which resets them to the beginning of their waveforms.

Let's recreate the off beat open hat with a double kick on the first beat that we've used as a running example. Both of these voices will require their own LFO to serve as a clock, where each LFO has its frequency controlled by a sequencer. Try updating the previous patch with gate delay and gate multiplier to use sequenced LFOs instead using the button in Figure 10.11.

Figure 10.11: Virtual modular for a percussion patch using LFOs as speed variable clocks to achieve gate delays, skipped steps, and off beats using a single parameter.

As you can see, this approach is fairly compact but not as compact as using a gate delay and gate multiplier because we've explicitly represented a number of skipped steps. However it is now easier to change between variations, e.g. adding rolls on different beats or skipping different beats would not require changing the patch, and these can be controlled by a single parameter.

A variation of this approach, though perhaps a somewhat distant one, can be used to generate gate patterns without a clock. Simply mix LFOs at different

frequencies to produce an irregular interference pattern that can then be run through a comparator to generate gates. If these need to be synced to a clock for uniform gate lengths, then the sample and hold method discussed in the previous section can be used. A clock can also be used to periodically trigger the reset of the LFOs, forcing the pattern to repeat.

10.5 Euclidean rhythm

We've mostly discussed sequencing where the patterns have been designed by hand, with the exception of probabilistic gate sequences and the recently mentioned sequenced generated by frequency-mismatched LFOs. What if there was a more predictable, more rule-like way of generating patterns? Euclidean rhythm is a recently discovered method that generates patterns in a rule-like way (Toussaint, 2005). This approach gets its name from the Euclidean algorithm for finding the greatest common divisor, which is the largest natural number that divides two numbers, a and b without a remainder. Here's how it works. Suppose $a > b$. Then the algorithm divides a by b to get a remainder c, e.g. 8/5 has remainder 3. If $c = 0$, then b is the greatest common divisor. Otherwise repeat by substituting b and c for a and b, e.g. 5/3 has remainder 2.[2]

To map the Euclidean algorithm to beats, suppose that we want to evenly distribute b beats over p positions, which we will call $E(b, p)$. Positions with no beat are silent (s). Then we parallel the structure of the Euclidean algorithm and define division as pairing up the b and s, with the unpaired s forming the remainder. The pairs substitute for b, the remainder substitutes for s, and the process repeats until the remainder is a single element or there are no s. An example of this process is shown in Table 10.2 for $E(5, 13)$.

Table 10.2: Steps of the Euclidean algorithm for the greatest common divisor of 8 and 5 (left) and corresponding pattern of beats (x) and silences (.) in the Euclidean rhythm for $E(5, 13)$ (right). Note that the number of groups in the pattern, marked by vertical bars, match the numbers a and b. These bars are removed in final result, x..x.x..x.x..

a	b	c	Pattern
8	5	3	|x|x|x|x|x|.|.|.|.|.|.|.|.|
5	3	2	|x.|x.|x.|x.|x.|.|.|.|
3	2	1	|x..|x..|x..|x.|x.|

[2]This is the division variant of Euclid's algorithm. The original used subtraction and so is slower and slightly more complicated because in subtraction, the difference isn't guaranteed to be smaller than the subtrahend.

a	b	c	Pattern
2	1	0	|x..x.|x..x.|x..|

As shown in Table 10.2, Euclidean rhythms divide the number of beats across positions as evenly as possible. If b divides p without remainder, then there will be an equal number of silences between beats. It turns out that there is an even easier way to calculate Euclidean rhythms by using Bresenham's line algorithm: using the equation for a line, at each natural number x, take the floor[3] of y. In this approach, x is equivalent to p, each increase in y is equivalent to b, and each lack of increase in y is equivalent to s. The result of the algorithm is shown in shown in Figure 10.12. Note that the colored regions mark the closest fitting line with slope 5/13 where the line is constrained to move in square units - this is equivalent to distributing beats as evenly as possible across positions.

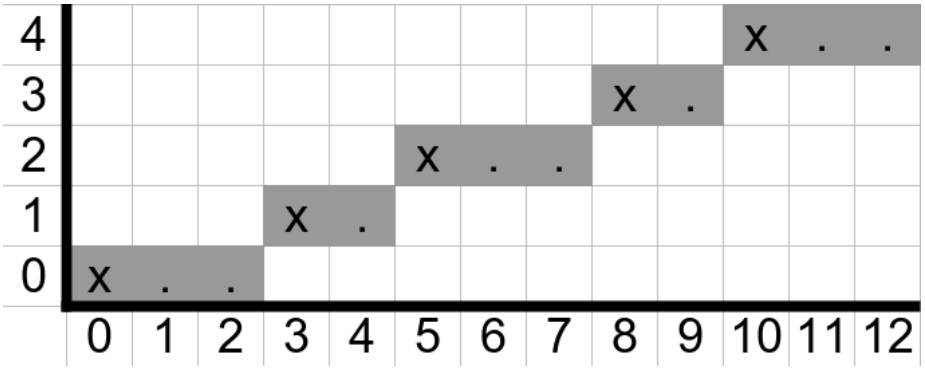

Figure 10.12: Example of Bresenham's line algorithm for the example $E(5, 13)$.

Euclidean rhythms can start at an offset into the sequence such that the skipped early portion is then added to the end. This is sometimes called a rotation, and different rotations are sometimes described as belonging to the same necklace. Let's apply Euclidean rhythms to the basic percussion patch from the beginning of the chapter. We'll use a special model that generates a Euclidean rhythm for each percussion voice, along with a trigger to gate module. Try updating that patch using the button in Figure 10.13.

Figure 10.13: Virtual modular for a percussion patch using Euclidean rhythms for each voice.

[3] A floor operation truncates the decimal portion of a number, e.g. 3.28 becomes 3.

As you can see from the patch, Euclidean rhythms is very compact and flexible, since we can easily change the length of patterns without affecting the proportion of hits and vice versa, just by turning a knob. However, we also lose some precision. For example, it isn't possible to make a double hit on the first beat because Euclidean rhythms by definition evenly distribute beats. So again, we trade precision for compactness and flexibility. It's worth noting that Euclidean rhythms can also define scales. For example, $E(5, 12)$ selects notes in the major pentatonic scale, and $E(7, 12)$ selects notes in the diatonic scale C major.

10.6 Sequential switches

The focus up to this point has been on creating patterns, but music typically involves chains of patterns. While it would be possible to have repetitions of a pattern in a sequencer, this is less compact because any change to a step in the pattern would require changing that step everywhere the pattern was repeated. Instead it makes sense to consider each pattern as a chunk sequence these chunks. Sequential switches are type of module that makes it easy to switch between different patterns in a sequence.

Sequential switches are also known as multiplexers and demultiplexers. A multiplexer is used to send N different signals to 1 place (N:1), and a demultiplexer is used to send 1 thing to N different places (1:N). The signal can be anything, including audio, and the destination is selected either by a clock or control voltage. When the selection is done by clock, each clock advances the switch just like a clock advances a sequencer. When the selection is done by control voltage, different voltage ranges select different destinations so that the selection can go out of order.

As a concrete example, supposed you have a kick drum and you want to send it a pattern, and when that pattern finishes, send it another pattern. That is multiplexing because you have N patterns you are sending to 1 place, the kick drum. On the other hand, suppose you have a pattern that you'd first like to send to flute voice, and when that pattern finishes, send the same pattern to a violin voice, and so on. That is demultiplexing because you have 1 pattern that you are sending N different places. If this is confusing, it's best to assume multiplexing (N:1) as this seems to be the more common operation.

Let's look at a simple example of using a sequential switch. In the previous patch with a gate delay and gate multiplier, the kick had a double hit on the first beat, and this was repeated every four beats. Suppose we wanted to repeat every eight beats instead. One option is to fill up the sequencer with empty steps, but another option is to divide the eight steps into two sequences of four steps each. The first four step sequence is our original sequence, and the second four step sequence is just the kick clock division. A clock division matching the length of the patterns is sent to the switch to advance to the next pattern. Try implementing this patch using the button in Figure 10.14.

Figure 10.14: Virtual modular for a percussion patch using a sequential switch to alternate between patterns.

Using a switch in this way suggests a general strategy for sequencing. Since each of the techniques discussed in this chapter has its strengths and weaknesses with respect to compactness, flexibility, and precision, it makes sense to alternate between them depending on the needs of the sequencing problem at hand and use sequential switches to combine them into larger patterns.

Chapter 11

Generators

This chapter extends Chapter 6 and focuses entirely on audio-rate modulation. Audio-rate modulation uses an audio-rate signal, typically an oscillator, to modulate a parameter, most commonly a parameter on another oscillator.[1] Although the basic idea is simple to explain, sounds produced can be complex and require a new idea to understand: the sideband.

Before proceeding, let's introduce some terminology. The audio rate signal source will be called the *modulator*. The oscillator receiving the modulation is the *carrier*. The depth, or strength, of modulation the carrier receives from the modulator will be called the *modulation index*. The modulation index defines how much the parameter of the carrier changes around its default value. For example, if a carrier has a frequency of 800 Hz, the modulation index may cause it to go +/- 1 Hz (from 799 to 801 Hz) or +/- 100 Hz (from 700 Hz to 900 Hz). The signal resulting from audio-rate modulation is the *output*.

Recall that we can describe any wave by its shape, amplitude, frequency, and phase. If we assume we can describe any waveshape by a collection of sine waves via Fourier transform, then we can focus on the amplitude, frequency, and phase parameters of these waves. Audio-rate modulation can be performed on each of these parameters. When the modulator acts on the amplitude parameter of the carrier, the result is amplitude modulation (AM). When the modulator acts on the frequency parameter of the carrier, the result is frequency modulation (FM). And when the modulator acts on the phase parameter of the carrier, the result is phase modulation (PM).

Each of these types of audio-rate modulation creates sidebands, which are new partials in the output. Audio-rate modulation therefore is related to additive synthesis in that both involve adding partials to change the timbre of the output sound. The main difference is that audio-rate modulation is often not strictly

[1] Audio-rate modulation of a filter's cutoff is another common option but is beyond the scope of this chapter. Try it!

additive but also removes or changes some partials in the carrier. However, with sufficient control of audio-rate modulation, one can create a wide variety of sounds more efficiently than additive synthesis, as shown by the explosion of inexpensive FM keyboards in the 1980's, like the Yamaha DX7.

The following sections outline the use of audio-rate modulation in modular sound synthesis according to the parameters of amplitude, frequency, and phase. However, the story is more complicated than this due to the history of synthesis, limitations of analogue hardware, and mathematical relationships between methods. There are actually two families of methods that cover all three parameters, as will be explained.

11.1 Modulating amplitude

The amplitude family includes amplitude modulation and ring modulation. The difference between the two is that amplitude modulation stops producing sound when the modulation signal crosses below zero (unipolar) and ring modulation continues to produce sound (bipolar). Although this difference may seem like a small one, it leads to several differences in the resulting sidebands and therefore overall timbre.

11.1.1 Amplitude modulation

You might have heard of amplitude modulation (AM) before, as it was one of the earliest technologies used in radio and is still in use today. We actually already covered the basic idea of AM in Section 6.3.3 to produce tremelo. As you recall, we used an LFO at relatively low rates to control a VCA, and the VCA was controlling the output of our main oscillator. AM works exactly the same way as this but at audio rates (>20 Hz). At audio rates, we can hear sidebands and the nature of the sound changes from an oscillation in loudness to a new timbre.

The VCA is key to understanding AM. Recall that the VCA is a level control that lets us let through a certain amount of signal, typically 0-100%.[2]. If you stop to think about this mathematically, it means that the VCA is multiplying the signal by a control value. For example, if the VCA control value is 50%, then it is multiplying the incoming signal by .5. AM stops producing sound when the control value crosses below zero because VCAs are defined to operate in the 0-100% range, i.e., they are unipolar. So when the control value crosses below zero, the VCA output simply stays at zero until the control value becomes positive again. This behavior is illustrated in Figure 11.1.

[2]Most VCAs are attenuators and so only reduce signal. Some VCAs include amplifiers that allow the gain on the signal to go above 100%

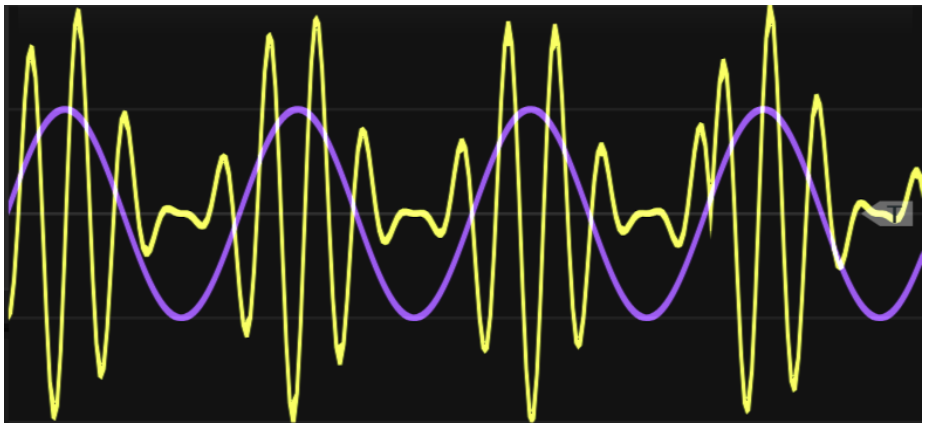

Figure 11.1: An example of amplitude modulation using sine waves for modulator and carrier. Note that where the modulator signal (purple) is at its peak, the output (yellow) is at greatest strength, but where the modulator signal is negative, the output reduces to zero.

Side bands for AM are easiest to understand for sine waves. If both modulator and carrier are sine waves, the spectrum of the output will include a partial a the carrier frequency C_f and two partials offset by the modulator frequency M_f, specifically $C_f - M_f$ and $C_f + M_f$. The sidebands have less strength than the carrier, and the strength of the sidebands is determined by the modulation index, which for AM is defined as the peak change in output amplitude divided by the amplitude of the carrier, $\Delta A/C_a$. When the modulator and/or carrier are not sine waves, the sidebands contain the partials from all pairs of partials between the modulator and the carrier. For example, if M_f has 3 partials and C_f has 5 partials, then there are $3 * 5$ partials in the lower sideband and the same number in the upper sideband. Even though the carrier frequency is not the fundamental, it is perceived as the pitch center of the sound, even when when the sidebands contain many partials. Note that in the above formulas, any partial that crosses below zero disappears. Also, if the modulator frequency goes above the carrier, $M_f > C_f$, the roles of carrier and modulator are reversed.

Implementing AM synthesis as described above is relatively simple in our virtual modular set up, but it has some surprises. First, if we want pure sine waves, i.e. sine waves with only one harmonic, then we need to use wavetable oscillators, which we've briefly mentioned a few times up to this point but never used. A wavetable oscillator is a digital oscillator that defines a wave based on one stored cycle of that wave. We need wavetable oscillators here to get a perfect sine wave, because the analogue implementations faithfully recreate the imperfections of the original hardware. This is also true for real VCAs, which tend to be a bit noisy, so we'll need to use a specific VCA that we've never used before. All this effort is only necessary to get a result that matches the previous equations.

Otherwise you may not care about these small imperfections that give additional character to the sound through the extra sidebands they create. Try creating a simple AM patch from scratch using the button in Figure 11.2.

Figure 11.2: Virtual modular for amplitude modulation using pure sine waves.

AM synthesis will generally produce inharmonic sidebands unless the modulator and carrier frequencies are chosen to create harmonic relationships. For example, if modulator and carrier have the same frequency, then the lower sideband will be zero and the upper sideband will be one octave above the carrier, e.g. 100 and 200 Hz. We can describe such frequency relationships between modulator and carrier in terms of ratios, so if both have the same frequency, the ratio of C:M is 1:1. You can easily check that any N:1 ratio is harmonic, for example using 200 Hz as the carrier would result in 100, 200, and 300 Hz harmonics. Likewise, N:2 where N is odd will give odd harmonics (100, 300, 500 Hz), and N:2 where N in even will give even harmonics (200, 400, 600 Hz).

Typically when a harmonic relationship like this has been established, one wishes to keep it intact while playing different notes. This is as easy to accomplish as keeping two (or more) oscillators in tune as a chord while playing across a keyboard. Try extending the last patch with a keyboard and use V/Oct connections to keep the harmonic relationship between modulator and carrier intact using the button in Figure 11.3. This patch is also a good opportunity to explore the sounds of more complex modulator and carrier waves, as well as inharmonic sounds.

Figure 11.3: Virtual modular for amplitude modulation with keyboard tracking.

11.1.2 Ring modulation

11.2 Modulating frequency

11.2.1 Digital frequency modulation

11.2.2 Analogue frequency modulation

Exponential frequency modulation

Linear frequency modulation

Through-zero frequency modulation

11.2.3 Phase modulation

Chapter 12

Modifiers

Part V

Sound Design 2

Chapter 13

TBD & TBD

Part VI

Sound Design 2

Chapter 14

TBD & TBD

Bibliography

(2022). DISTRHO Cardinal. original-date: 2021-10-07T10:01:22Z.

(2022). VCV Rack - The Eurorack simulator for Windows/Mac/Linux.

Anderson, N. D. (2016). A call for computational thinking in undergraduate psychology. *Psychology Learning & Teaching*, 15(3):226–234.

Bell, T. (2021). CS unplugged or coding classes? *Communications of the ACM*, 64(5):25–27.

Bjørn, K. and Meyer, C. (2018). *Patch & Tweak: Exploring Modular Synthesis*. Bjooks. Google-Books-ID: xmrVvQEACAAJ.

Bode, H. (1984). History of electronic sound modification. 32(10):730–739.

Crombie, D. (1982). *The Complete Synthesizer: A Comprehensive Guide*. Omnipress Books.

Doepfer Musikelektronik (2022a). Construction Details A-100.

Doepfer Musikelektronik (2022b). Technical details A-100.

Dudley, H. and Tarnoczy, T. H. (1950). The speaking machine of Wolfgang von Kempelen. 22(2):151–166. Publisher: Acoustical Society of America.

Dusha, T., Martonova, A., and Johnston, P. (2020). *Modular Sound Synthesis On the Moon*. Modular Moon. Google-Books-ID: llA4zgEACAAJ.

Eliraz, Z. (2022). Loopop's In-Complete Book of Electronic Music.

Fletcher, H. and Bassett, I. G. (1978). Some experiments with the bass drum. *The Journal of the Acoustical Society of America*, 64(6):1570–1576. Publisher: Acoustical Society of America.

Jacoby, N., Undurraga, E. A., McPherson, M. J., Valdés, J., Ossandón, T., and McDermott, J. H. (2019). Universal and Non-universal Features of Musical Pitch Perception Revealed by Singing. 29(19):3229–3243.e12. Publisher: Elsevier.

Leissa, A. W. (1969). Vibration of plates. Technical report. Number: NASA-SP-160.

Michie, D. (1963). Experiments on the Mechanization of Game-Learning Part I. Characterization of the Model and its parameters. 6(3):232–236.

Oxenham, A. J. (2018). How we hear: The perception and neural coding of sound. *Annual review of psychology*, 69:27–50.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, New York.

Polya, G. (2004). *How to solve it: A new aspect of mathematical method*. Princeton University Press.

Strange, A. (1983). *Electronic Music: Systems, Techniques, and Controls*. William C Brown Publishers, 2 edition.

Tedre, M. and Denning, P. J. (2016). The long quest for computational thinking. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, Koli Calling '16, page 120–129, New York, NY, USA. Association for Computing Machinery.

Toussaint, G. (2005). The euclidean algorithm generates traditional musical rhythms. In Sarhangi, R. and Moody, R. V., editors, *Renaissance Banff: Mathematics, Music, Art, Culture*, pages 47–56, Southwestern College, Winfield, Kansas. Bridges Conference.

Tucker, A. (2003). A model curriculum for K–12 computer science: Final report of the ACM K–12 task force curriculum committee. Technical report, New York, NY, USA.