

Computational Thinking through Modular Sounds Synthesis

Andrew M. Olney

2022-09-21

Contents

Welcome	5
1 Introduction	7
1.1 Why this book?	7
1.2 Computational thinking	8
1.3 Modular synthesis	10
1.4 Moving forward	15
I Sound	17
2 Physics and Perception of Sound	19
2.1 Waves	19
2.2 Frequency and pitch	26
2.3 Amplitude and loudness	28
2.4 Waveshape and timbre	30
2.5 Phase and interference	33
3 Harmonic and Inharmonic Sounds	37
3.1 Phase reflections, standing waves, and harmonics	37
3.2 Resonators, formants, and frequency spectrum	41
3.3 Inharmonic standing waves and noise	44
3.4 Dynamics and Envelopes	48

II	Fundamental Modules	51
4	Basic Modeling Concepts	53
4.1	Modules are the model elements	54
4.2	Signals are how the model elements interact	54
4.3	Signals are interpreted by modules	57
4.4	Pulling it all together	59
4.5	Moving forward	63
5	Controllers	65
5.1	Clocks	65
5.2	Sequencers	68
5.3	Summing up	71
6	Create	73
7	Modify	75
III	Sound Design 1	77
8	Kick & Cymbal	79
9	Lead & Bass	81
IV	Complex Modules	83
10	Trigger	85
11	Create	87
12	Modify	89
V	Sound Design 2	91
13	Minimoog & 303	93

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!



This website is free to use and is licensed under the [Creative Commons Attribution-NonCommercial-NoDerivs 4.0 License](#).

¹I’ve also created [PDF](#) and [EPUB](#) versions; these are best used when you don’t have internet because they are **not** interactive.

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 1980s¹
- 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:

¹Old 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!

²There 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/>

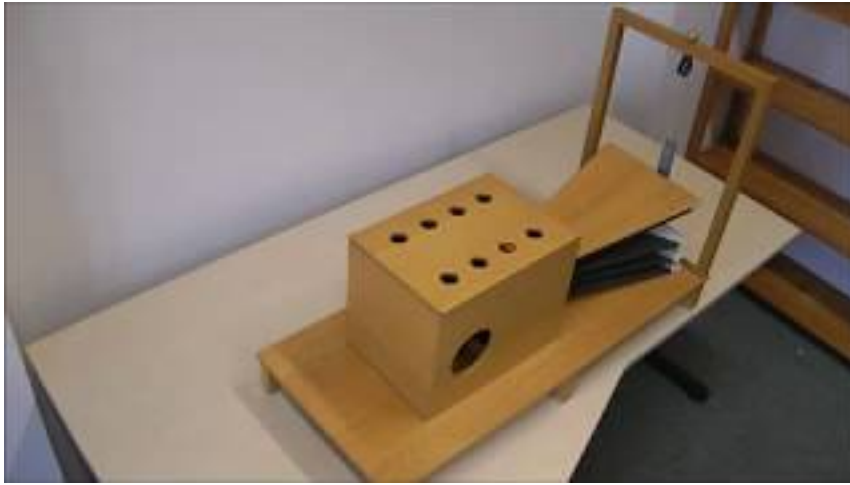


Figure 1.2: [YouTube video](#) of Wolfgang von Kempelen's speaking machine circa 1780. Image © [Fabian Brackhane](#).

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, these 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 © [mikaël 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 molecules¹ bombarding the

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

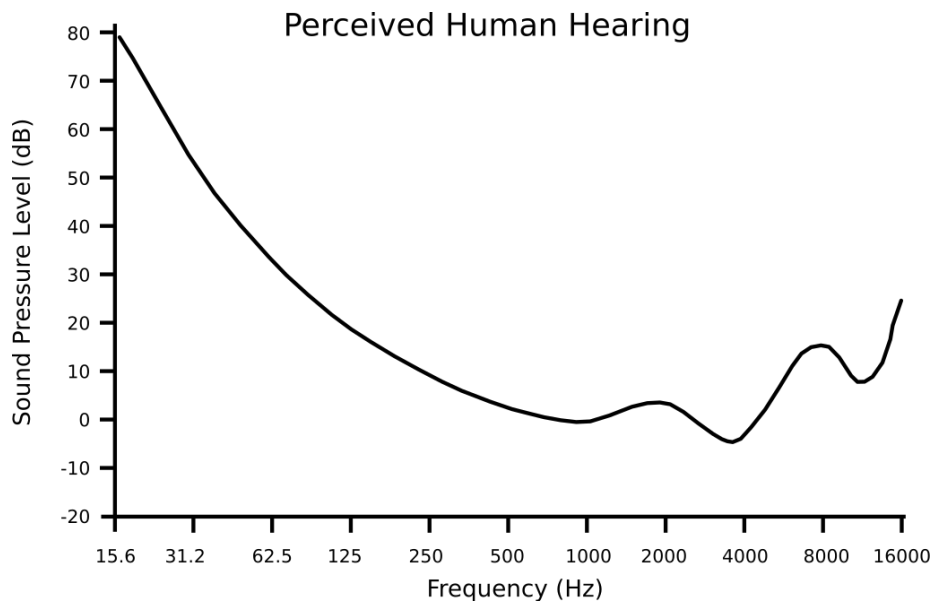


Figure 2.1: An equal loudness contour showing improved sensitivity to frequencies between 500Hz and 4KHz, which approximately matches the range of human speech frequencies. Image [public domain](#).

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,

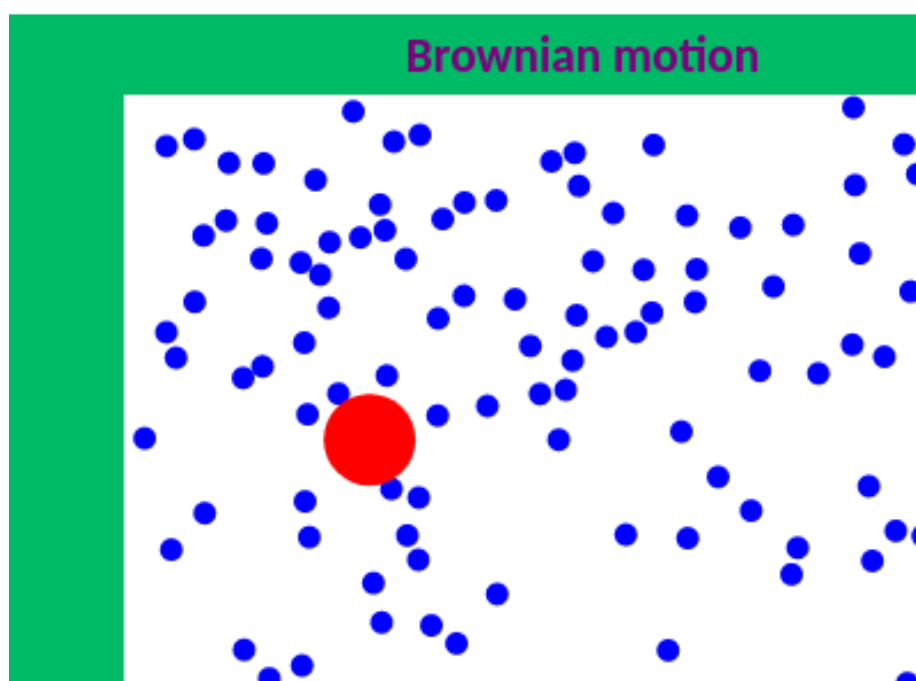


Figure 2.2: [Simulation](#) of Brownian motion. Press **Pause** to stop the simulation.
© Andrew Duffy/[CC-BY-NC-SA-4.0](#).



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).

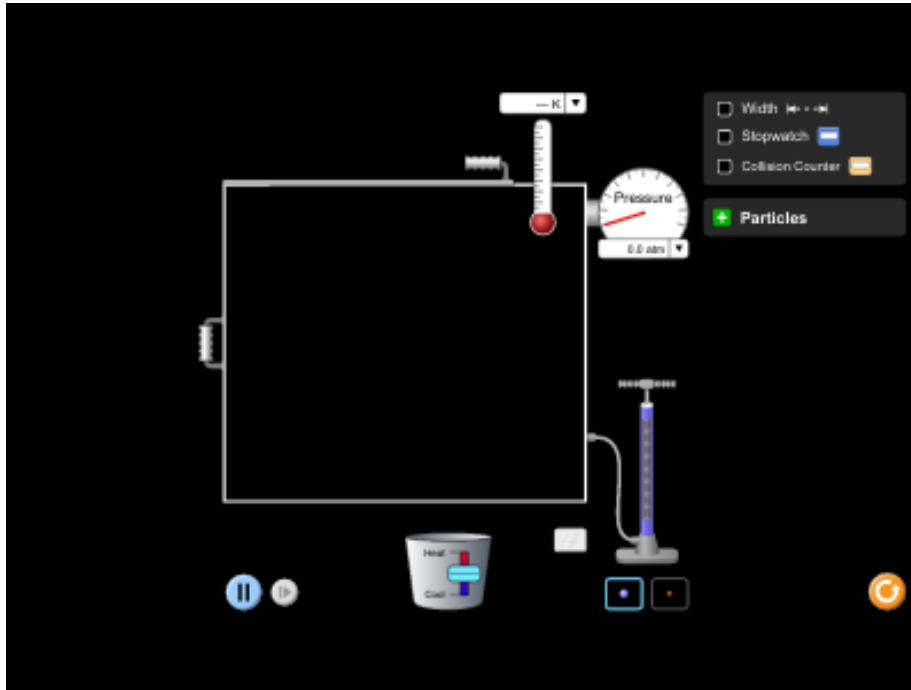


Figure 2.5: [Simulation](#) of gas in a chamber. Simulation by [PhET Interactive Simulations](#), University of Colorado Boulder, licensed under [CC-BY-4.0](#).

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 correspond 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.

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

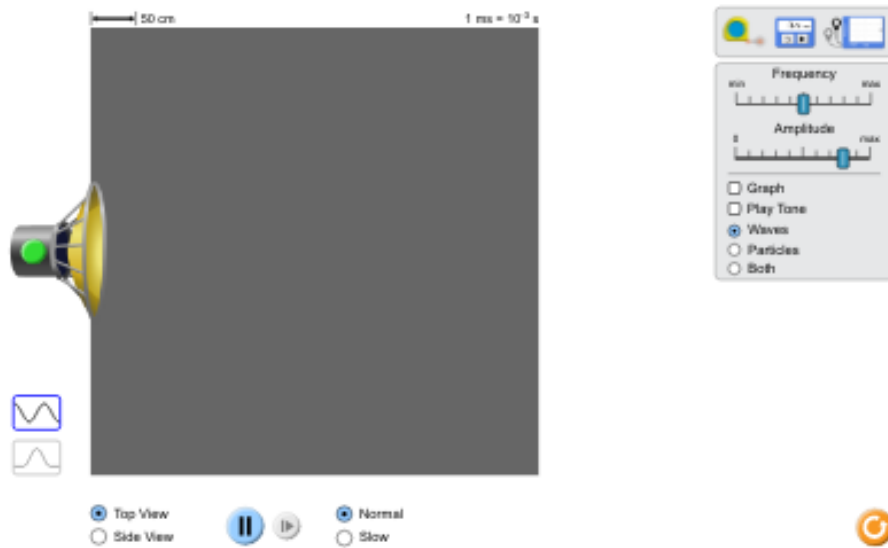


Figure 2.7: [Simulation](#) of sound waves. Simulation by [PhET Interactive Simulations](#), University of Colorado Boulder, licensed under [CC-BY-4.0](#).

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 zero³), 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.

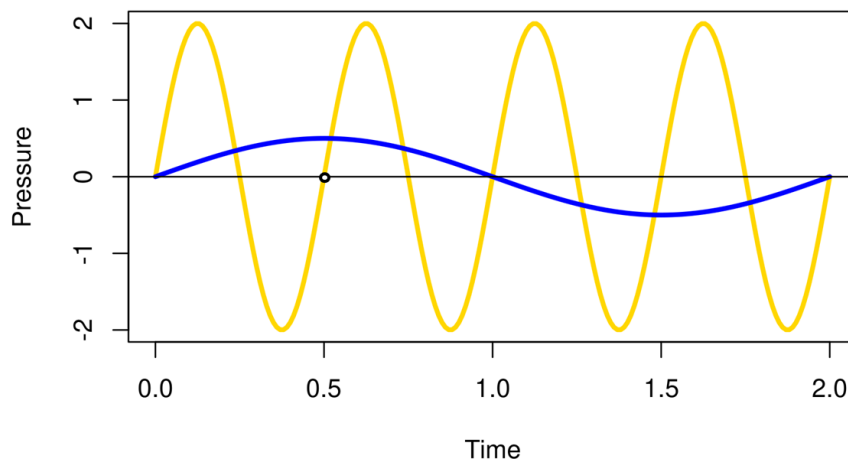


Figure 2.8: Two waves overlaid on the same graph. The yellow wave completes its cycle 4 times in 2 seconds and the blue wave completes its cycle 1 time in 2 seconds, so the frequencies of the waves are 2 Hz and .5 Hz, respectively.

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 Hz⁴,

³Recall 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.

⁴Also 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.

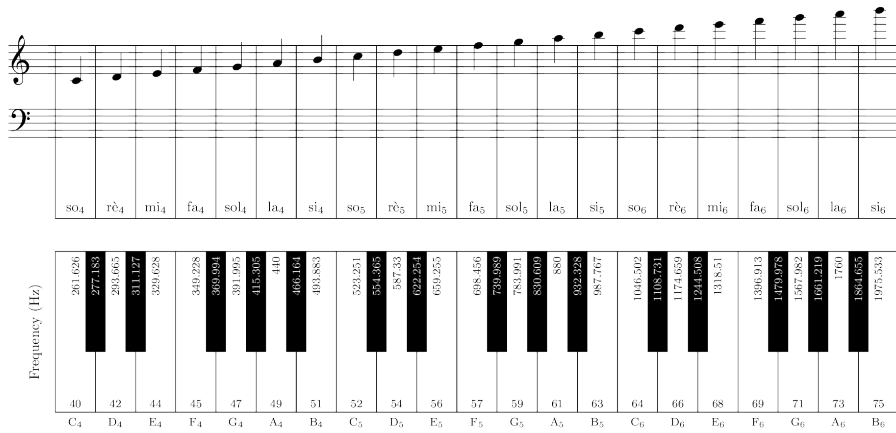


Figure 2.9: Part of an 88 key piano keyboard with frequency of keys in Hz on each key. The corresponding note in musical notation and solfège is arranged above the keys. Zoom in on the image for more detail.

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 apart,

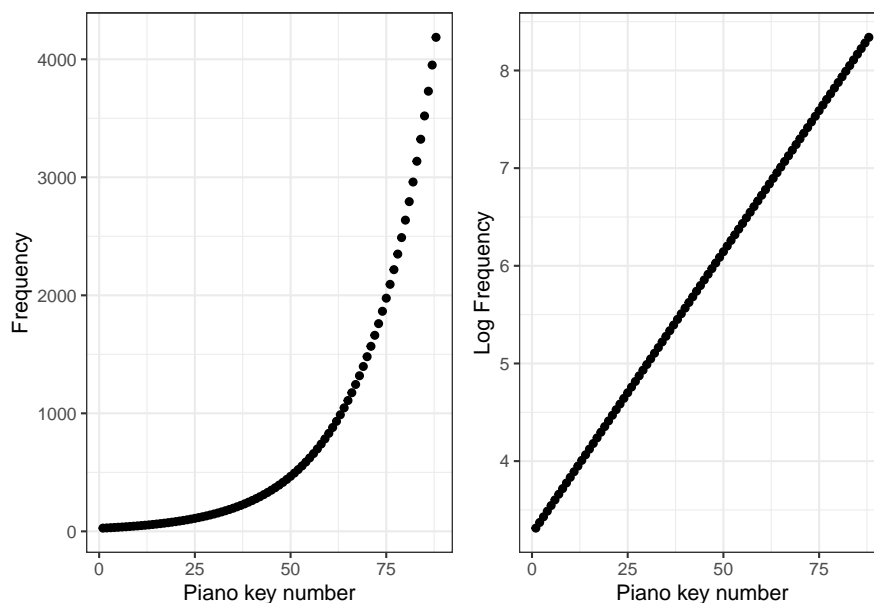


Figure 2.10: (ref:log-freq)

but the lowest key⁵ 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 temperament](#). 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

⁵Also 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.

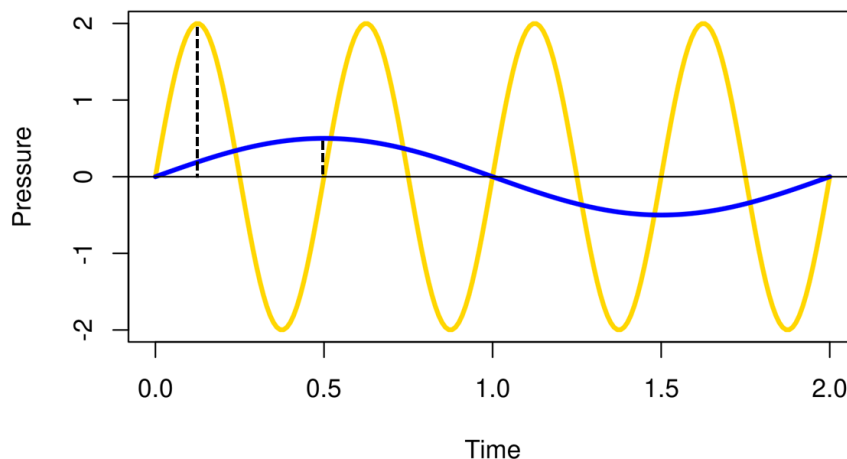


Figure 2.11: Two waves overlaid on the same graph, with a dashed line marking the amplitude of each wave as the deviation from equilibrium.

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 is 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

⁶Note 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](#).

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

⁸Note 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.

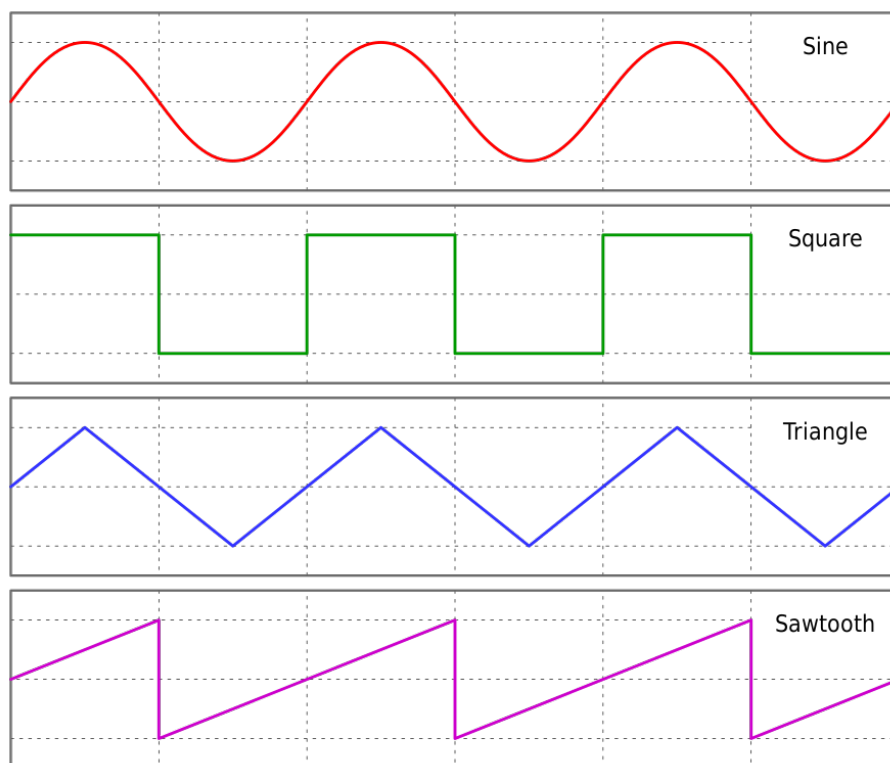


Figure 2.12: The four classic waveshapes in analogue electronic music. Image © Omegatron/CC-BY-SA-3.0.

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.





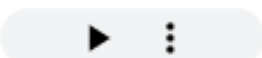



Wave	Wave sound	Comparable instruments	Instrument s
Sine		© Flutes	
Square		© Brass (no reed), e.g. trumpet	
Triangle		© Single reed woodwind, e.g. clarinet	
Saw		© Bowed strings, e.g. violin	

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

⁹One 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.

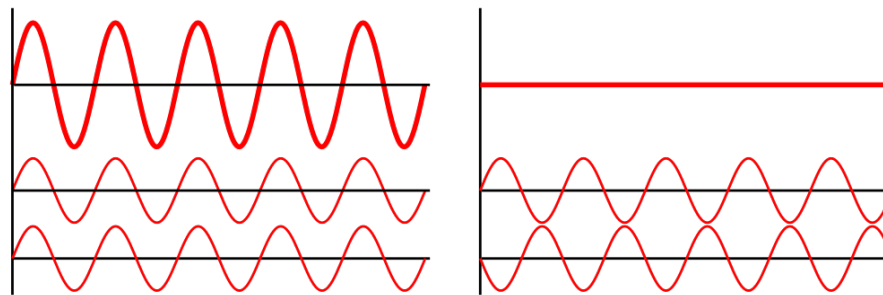


Figure 2.15: Constructive (left) and destructive interference (right). For these matched sine waves, being perfectly in phase or out of phase causes the resulting wave amplitude to be either double or zero, respectively. Image © Haade; Wjh31; Quibik/CC-BY-SA-3.0.

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 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 all cases. 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 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

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

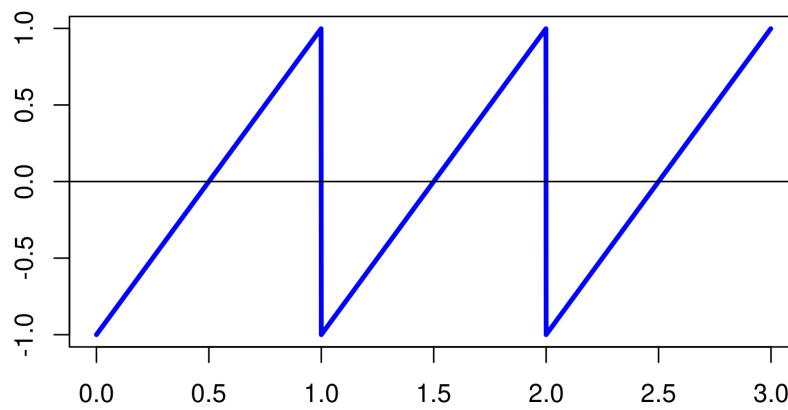


Figure 2.16: [Animation](#) of interference from a perfectly out of phase saw wave (yellow) combined with the original wave (blue). The resulting wave (green) has non-zero amplitude because the out of phase wave does not have opposite amplitude to the original wave at all locations. Dotted lines indicate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

¹¹Any 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?

¹I 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.

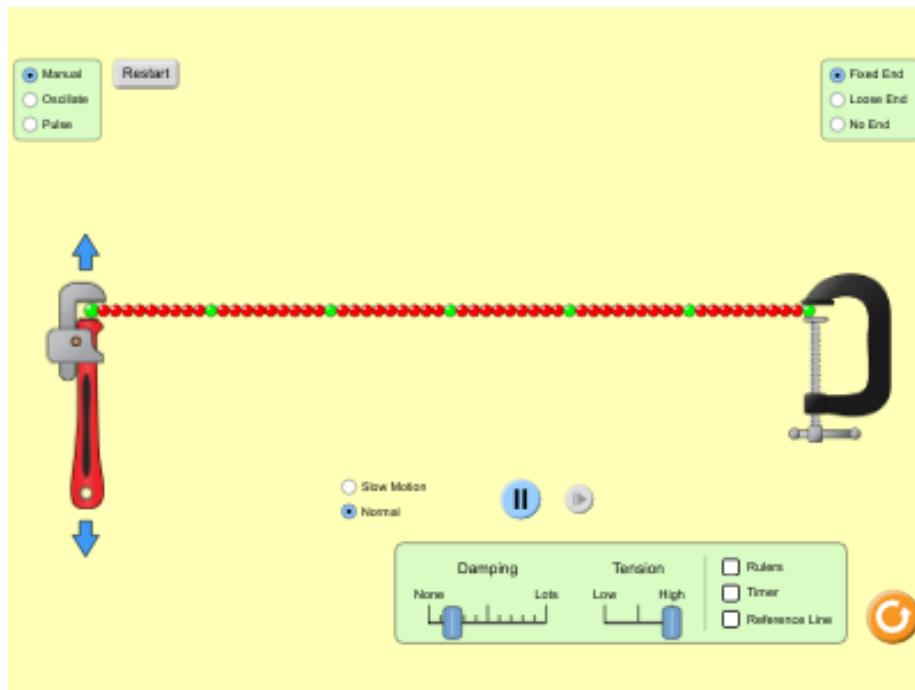


Figure 3.1: [Simulation](#) of waves on a string. Simulation by [PhET Interactive Simulations](#), University of Colorado Boulder, licensed under [CC-BY-4.0](#).

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](#) that 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.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 similar behavior to the simulation when a string is plucked, as shown the slow

²One 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.

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

³Transients 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

⁴You 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.

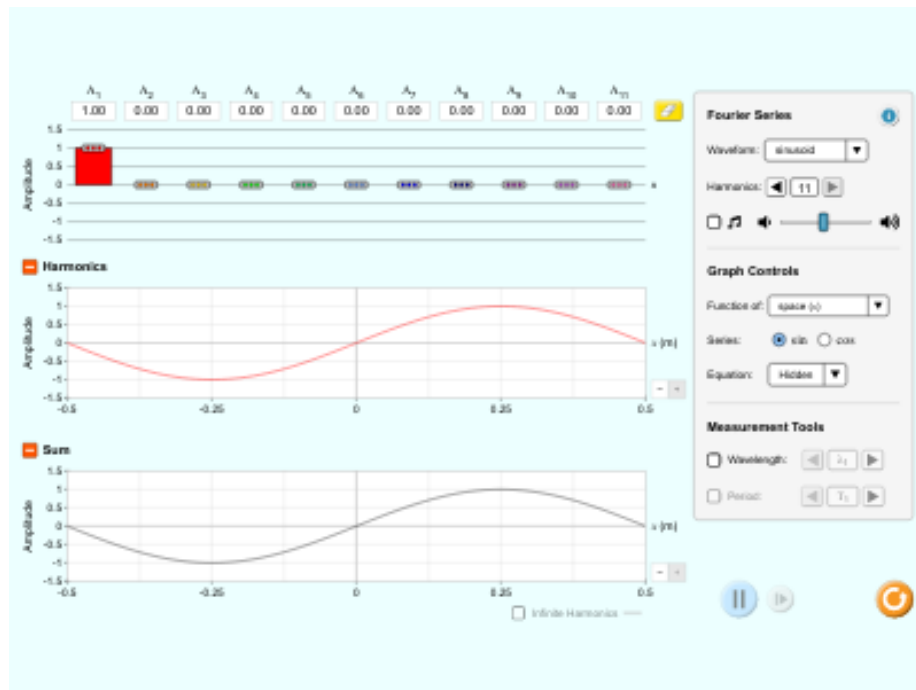


Figure 3.3: [Simulation](#) showing how Fourier analysis can approximate a complex wave using sine waves. Simulation by [PhET Interactive Simulations](#), University of Colorado Boulder, licensed under [CC-BY-4.0](#).

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 frequency 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

⁵1 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.

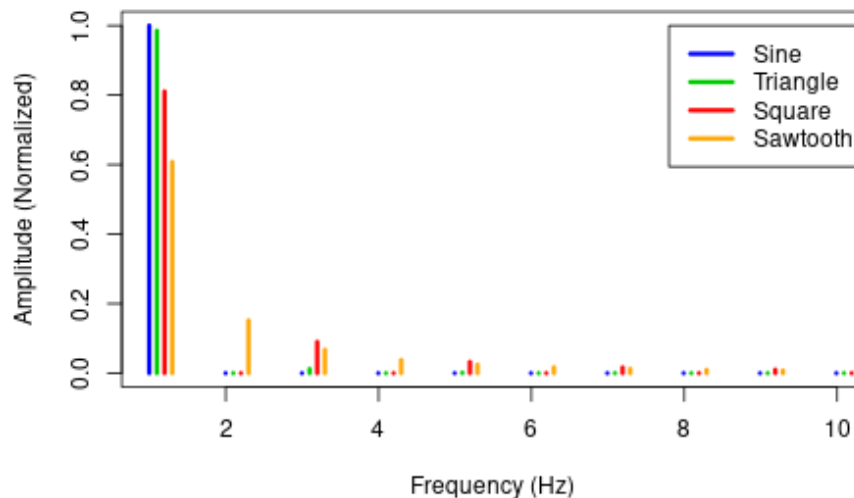


Figure 3.4: Frequency spectrum of four basic waveshapes at 1 Hz. Amplitude is normalized so all harmonics sum to 1. Harmonics are offset for comparison.

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 matter when we are concerned with listening to sound, since our auditory system [decomposes complex sounds into frequency bands](#) 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 a vibrating drum are shown

in Figure 3.5. Each of these modes has 0, 1, or 2 nodal lines across the diameter of the drum head and a nodal circle around the perimeter of the drum head. 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.

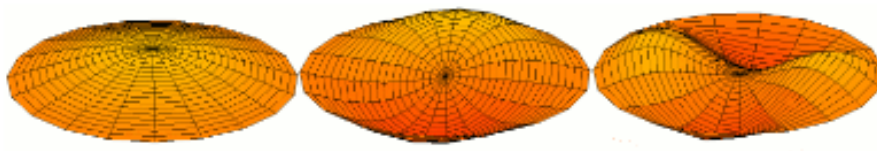


Figure 3.5: Animations of drum head vibration modes with a single nodal circle and 0, 1, or 2 nodal lines. Images public domain.

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 dissipates 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.

Table 3.1: Modes of a 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 thought 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.

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

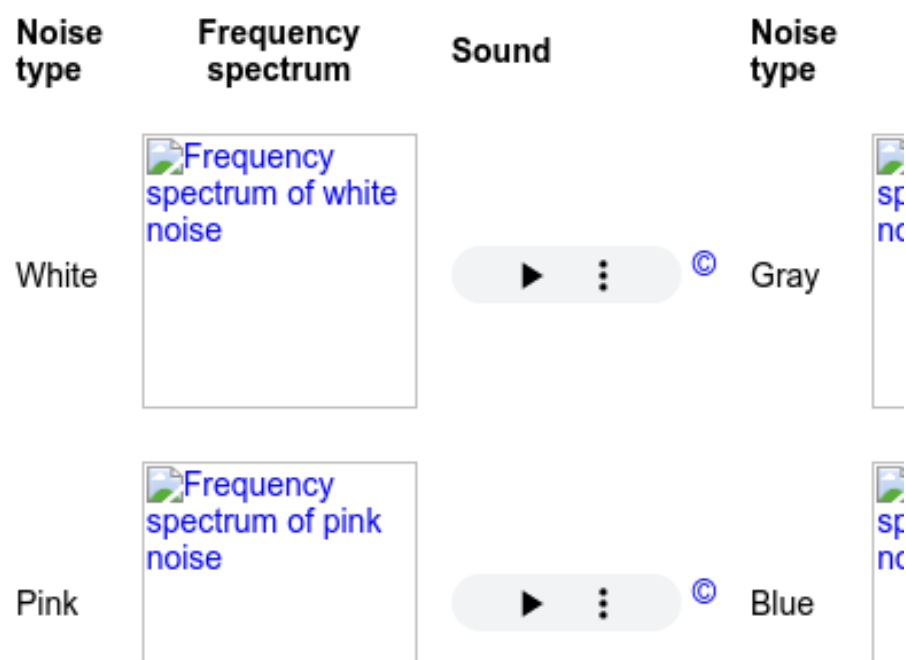


Figure 3.6: Frequency spectrum and sound of various “colors” of noise.

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 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, an 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

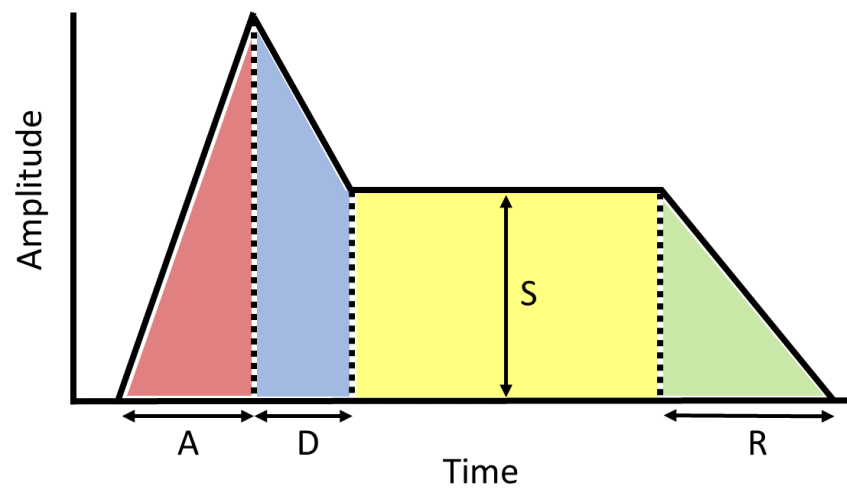


Figure 3.7: An example Attack-Decay-Sustain-Release (ADSR) envelope. Sustain ends with manual control and is the only parameter that sets amplitude level. All other stage lengths are controlled by time parameters as indicated.

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.

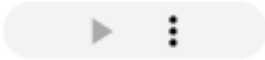
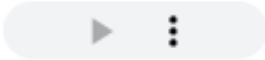


Wave	Wave sound	Instrument	Instrument sound (ADSR)
Sine		Kick	
Saw		Violin	

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.

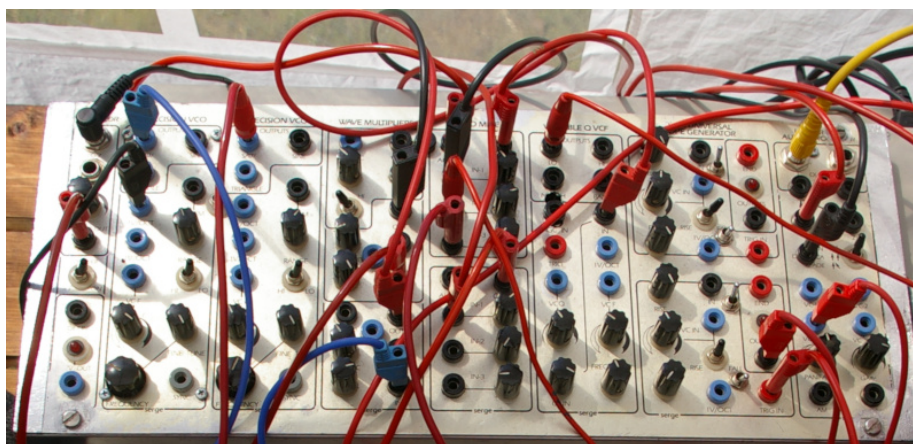


Figure 4.1: A Serge modular system based on a 1970s design. Each module is labeled at the top edge, e.g. **Wave Multiplier**, and extends down to the bottom edge in a column. Note that although the modules have the same height, they have different widths. Image © [mikaël altemark/CC-BY-2.0](#).

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. VCV Rack 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 bipolar, 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.

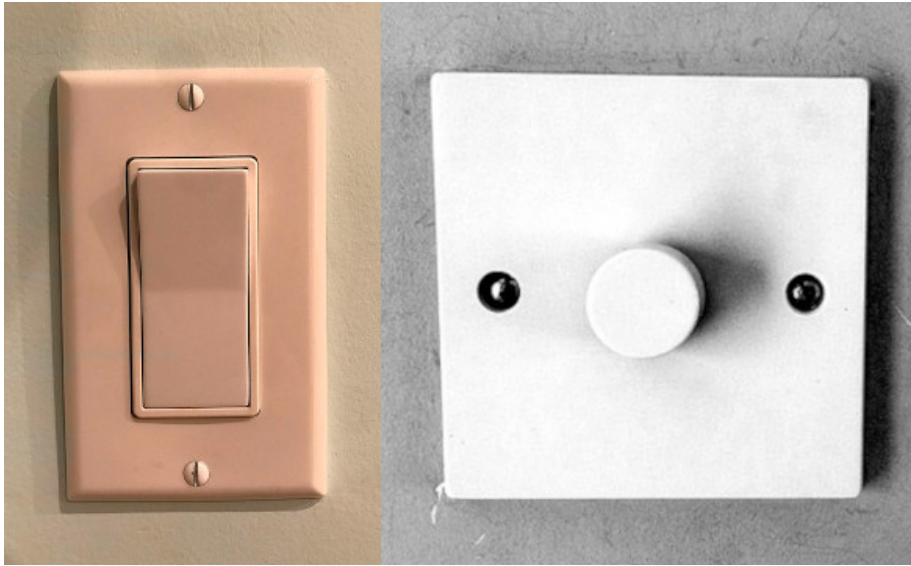


Figure 4.2: On/off light switch (left) and dimmer light switch (right). While on/off switches can only be at minimum or maximum voltage, dimmer switches can be at all voltages in between. Images © [DemonDays64/CC-BY-4.0](#) and © [Paolomarco/CC-BY-4.0](#).

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 more voltage than common audio devices, so connecting modular gear

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

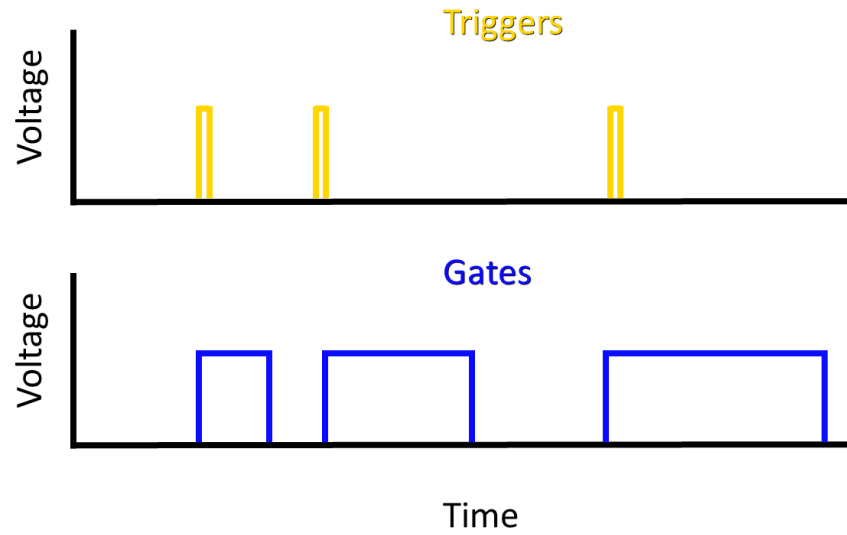


Figure 4.3: Triggers and gates shown over time. Both are on/off unipolar voltages that show as rectangular pulses. Only gates have variable duration.

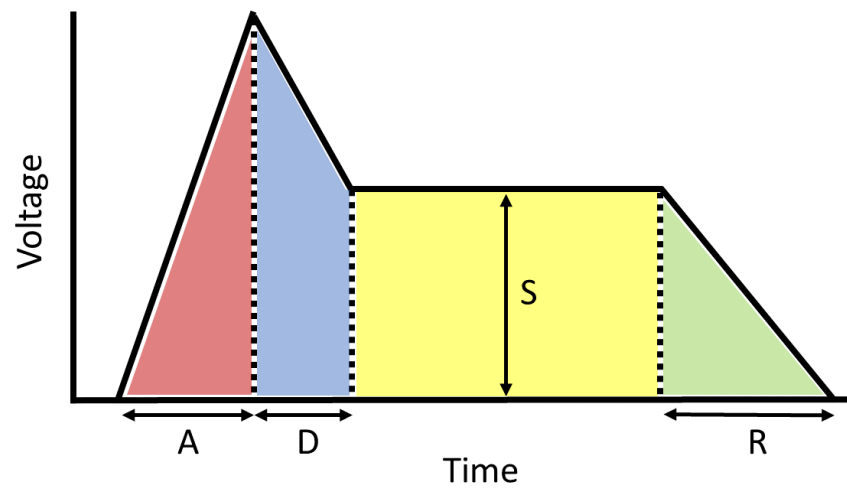


Figure 4.4: An example Attack-Decay-Sustain-Release (ADSR) envelope represented as control voltage.

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, it's 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.

²Can be inverted

³The 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.

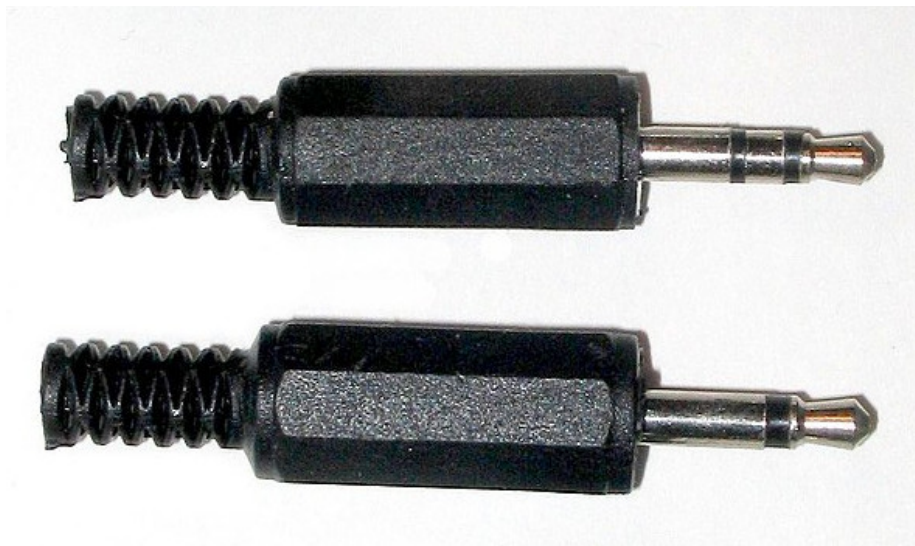


Figure 4.5: Stereo (top) and mono (bottom) patch cable connectors. Both have a tip and sleeve conductor, but stereo has an additional “ring” conductor to carry a second audio channel. Image [public domain](#).

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

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

⁵Typically 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.

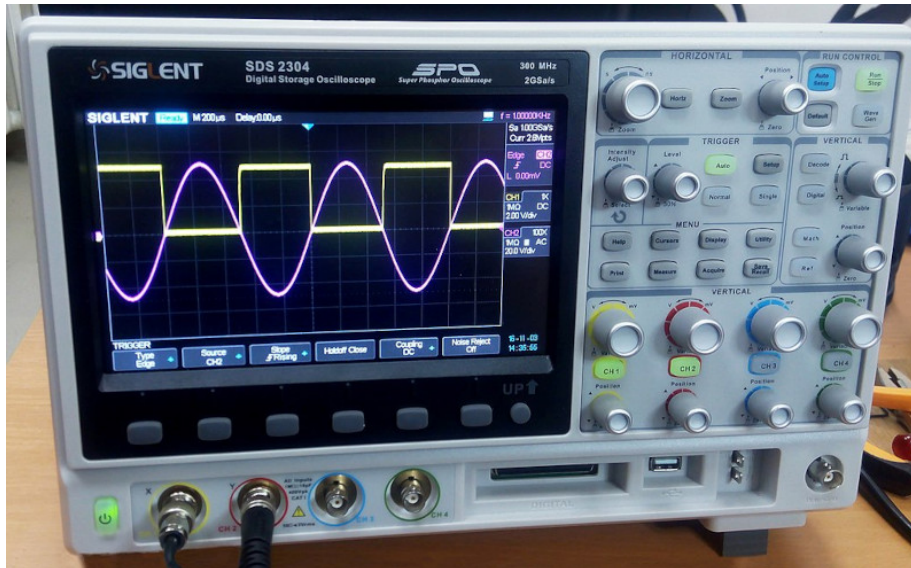


Figure 4.6: A bench oscilloscope showing a sine wave and offset square wave simultaneously. Image © Wild Pancake/CC-BY-4.0.

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-port 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 VCV Rack/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>

Launch Virtual Modular

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](#).

Launch Virtual Modular

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](#).

A dark teal rectangular button with rounded corners and the text "Launch Virtual Modular" in white.

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.

A dark teal rectangular button with rounded corners and the text "Launch Virtual Modular" in white.

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 ADSR 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 ADSR 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 ADSR 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.

Launch Virtual Modular

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 pulse² 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.musio.com/2021/08/19/which-musical-tempos-are-people-streaming-the-most/>

²Some 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.

³Not 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.

⁴Some 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 output⁵, 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.



Launch Virtual Modular

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

⁵This particular clock does not use division notation.

⁶Some 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 like to 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 there.

Launch Virtual Modular

Figure 5.4: [Virtual modular](#) for making a clock-driven kick patch.

Keep this example in mind as you 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.

Launch Virtual Modular

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.

Launch Virtual Modular

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 saw envelope is gated. Try this using the button in Figure 5.7 to explore the different combinations.

Launch Virtual Modular

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 the envelope. 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.

Launch Virtual Modular

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 VCV Rack. The trigger sequence channel is highlighted in red and the control voltage channels are highlighted in blue.

Chapter 6

Create

Chapter 7

Modify

Part III

Sound Design 1

Chapter 8

Kick & Cymbal

Chapter 9

Lead & Bass

Part IV

Complex Modules

Chapter 10

Trigger

Chapter 11

Create

Chapter 12

Modify

Part V

Sound Design 2

Chapter 13

Minimoog & 303

Bibliography

- (2022). DISTRHO Cardinal. original-date: 2021-10-07T10:01:22Z.
- (2022). VCV Rack - The Eurorack simulator for Windows/Mac/Linux.
- 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*. Bbooks. 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: lIA4zgEACAAJ.
- Eliraz, Z. (2022). Loopop’s In-Complete Book of Electronic Music.
- 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.
- Michie, D. (1963). Experiments on the Mechanization of Game-Learning Part I. Characterization of the Model and its parameters. 6(3):232–236.
- 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.
- 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.