Spring 5-20-2016

Soft Sound: A Guide to the Tools and Techniques of Software Synthesis

Quinn Powell
California State University, Monterey Bay

California State University, Monterey Bay

Soft Sound:

A Guide to the Tools and Techniques of Software Synthesis

Quinn Powell

Professor Sammons

Senior Capstone Project

23 May 2016

Introduction

Edgard Varèse once said: “To stubbornly conditioned ears, anything new in music has always been called noise. But after all, what is music but organized noises?” (18) In today’s musical climate there is one realm that overwhelmingly leads the revolution in new forms of organized noise: the realm of software synthesis. Although Varèse did not live long enough to see the digital revolution, it is likely he would have embraced computers as so many millions have and contributed to these new forms of noise.

Computers provide a powerful, convenient, and affordable means of creating music. No longer do musicians have to lug around heavy electronics or spend hundreds of dollars on instruments each time they seek a new noise. Instead, software synthesizers provide the endless exploration of sound at the tips of the fingers.

This paper explores the various techniques and tools of software synthesis. Part I deals with necessary background information for a thorough understanding of synthesizers. Part II delves into the specifics of software synthesis, with an emphasis on forms that are more commonly realized in software rather than analog technology. Part III briefly introduces some of the environments for realizing such techniques. Ultimately, all of the tools and techniques of software synthesis have one thing in common: they provide an affordable and convenient means for realizing a virtually infinite number of sounds.

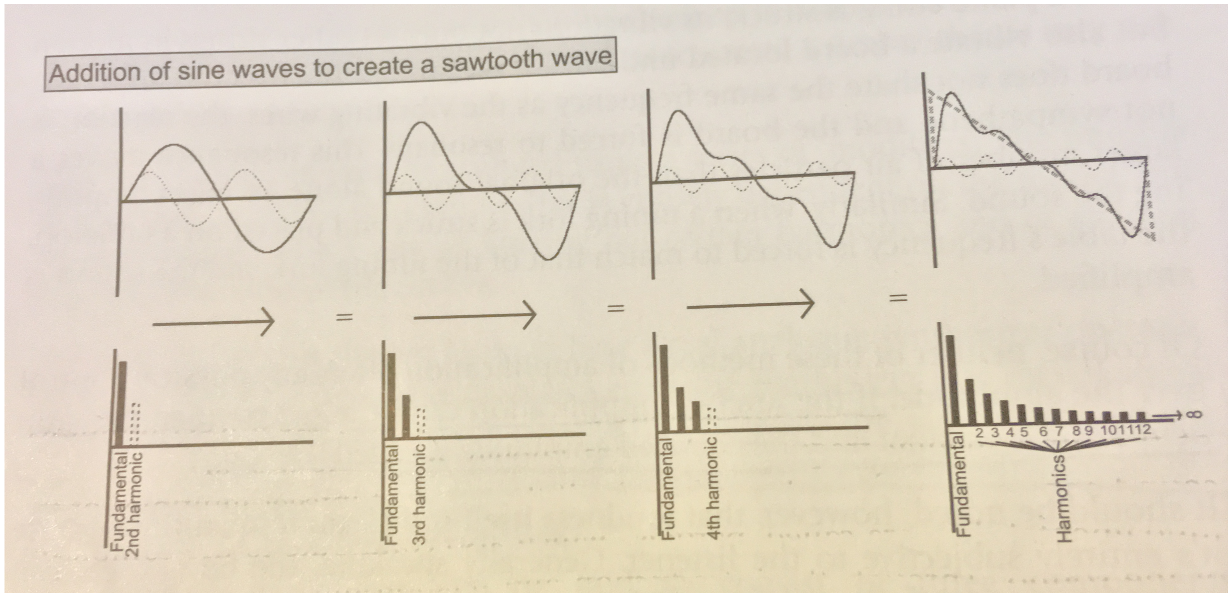
Part I: Sound, Synthesis, and Digital Audio

Sound

Any discussion of synthesis generally begins with the principles of sound. It is important to realize that there are two lenses through which we can view sound. Acoustics studies the physical nature of sound waves, while hearing is the perception of these sound waves. At the most basic level, sound waves occur when energy disrupts a physical medium (Pohlmann 1), such as water, air, or even concrete. Most of the time, this takes the form of fluctuations in air pressure that eventually reach our ears. For example, when a guitar string is plucked, it moves upward and compresses the air molecules above it into a region of higher air pressure. When the string moves downward, it leaves a region of lower air pressure, called a rarefaction. These regions of high and low air pressure propagate, or spread outward, much like the waves from a stone dropped into a body of water. Eventually, this acoustic energy reaches our ears, causing the eardrum to vibrate, triggering the sensation of sound.

Sound is graphically represented as a two dimensional wave, showing fluctuations in sound pressure level (the Y axis) over time (the X axis). The resulting wave may take many shapes. The simplest possible waveform is a sine wave, which has a natural, smooth shape to its motion. In practice, most musical instruments produce complex waveforms. A violin, for example, produces a wave that looks similar to a sawtooth, because as the bow is pulled the string sticks to the resin on the bow and is pulled too. It periodically snaps back to its equilibrium, producing the sudden, downward portion of the waveform (Reid, “Synthesizing Bowed Strings: the Violin family”). The number of cycles that a wave completes per second is the frequency, which is measured in Hertz (Hz). The frequency determines the pitch of a musical sound. The height of the wave is the amplitude, and it corresponds to the loudness of a sound. A final important consideration is phase, which is a measurement of how far along one cycle the wave is, measured in degrees. One full wave cycle is 360 degrees, so a wave that is 180 degrees out of phase with another wave is halfway through its cycle while the other is beginning its cycle.

In the early 1800s, a mathematician named Jean Baptiste Joseph Fourier showed that any complex periodic waveform can be broken down into a series of sine waves at different frequencies and amplitudes (Smith Ch. 8). As the image below shows, enough sine waves at different frequencies can be added together to eventually produce a sawtooth (Snoman 7):



This can be witnessed on the vibrating strings of a guitar. When viewed at a slow enough rate, there are actually multiple vibrations that are occurring at once across different regions of the guitar string. These different frequencies add up to produce the complex tone, or timbre, that a guitar generates.

The relationship amongst the different sine wave frequencies and their relative amplitudes is what gives a sound its particular timbre. The sensation of pitch results from a specific series of sine waves known as the harmonic series. The lowest frequency of the harmonic series is the fundamental frequency, and it determines the perceived pitch of a sound. Frequencies that are integer multiples of the fundamental are called harmonics. For example, a piano playing concert pitch A4 has a fundamental frequency of 440 Hz, but it also contains harmonics at 880 Hz, 1320 Hz, 1760 Hz, and other multiples of 440. Higher ordered harmonics tend to be weaker than lower ones. A timbre may also include frequencies that are not integer multiples of the fundamental and these are called partials.

Certain timbres may be thin and contain only a fundamental frequency and two harmonics. Some timbres may be rich and contain the entire harmonic series. Other sounds may contain frequencies that do not form a harmonic series at all but are distributed randomly across the frequency spectrum. These timbres would not result a sense of pitch but rather noise, such as the crash of a cymbal. The human ear can hear frequencies in a range from 20 to 20,000 Hz, providing a wide range of possible timbres, both noisy and pitched.

Synthesis

Although forms of electronic instruments have existed since the late 1800s, it was not until the 1950s that people began taking full advantage of Fourier's discovery and synthesizing complex signals from sine waves, a process known as additive synthesis. Additive synthesis was pioneered by a group of scholars and musicians, most notably Karlheinz Stockhausen, who began creating compositions of the style *electronische Musik* at a radio studio in Cologne. For Stockhausen, "the desire to exercise total control over the processes involved in specifying timbres led, after some deliberation, to the selection of the sine wave oscillator as the most suitable source of electronic material" (Manning 43). A sine wave oscillator generates an electrical signal of varying voltage over time, with control over the frequency and amplitude of the signal. This electrical signal was then converted into acoustic sound waves via a loudspeaker. Using only one sine wave oscillator and multi-track tape machines, Stockhausen recorded different sine waves over many different takes. Although tedious, the process offered precise control over the resulting timbres, and culminated in two of the defining works of *electronische Musik*: *Studie I* and *Studie II*.

Eventually, musicians in Cologne and elsewhere began using oscillators that generated complex waves, such as a square or noise waves (Manning 44). These complex sounds were generally followed by a filter, a device that boosts or attenuates specific frequencies in a signal. This process of filtering complex waveforms is known as subtractive synthesis. One of the earliest synthesizers with subtractive capabilities was the Mark II, by RCA. The Mark II consisted of a series of modules, including oscillators and filters, that could be flexibly routed (96). Other modules included envelope generators, which vary the amplitude of a wave over time, and low frequency oscillators (LFOs), which could be routed to create subtle movement in pitch or volume. The synthesizer filled an entire room at Columbia University and required complex programming to create sounds. Other notable analog synthesizers were created by Buchla and Moog and featured keyboards as means of controlling synthesis, thus offering a more musical control method (102).

Ultimately, synthesizers did not permeate popular music until the digital revolution. The first experiments using computers to synthesize audio signals began in the 1950s at Bell Telephone Laboratories, where research was being conducted into digitizing phone conversations. One of the engineers, Max Mathews, began experimenting with software that could produce audio waveforms from scratch. The first program, Music I, was capable of producing a single triangle wave. Music II offered 16 possible waveforms, and subsequent version of the software, such as Music III and Music IV offered multiple oscillators, filters, and envelopes, among other tools (Manning 187).

Although a significant historical advancement, only a few experimental and academic pieces were composed using the Music N programs. Computing technology of the time was not fast enough to generate complex signals in real time, instead requiring time for the waveform calculation results to be stored in a separate digital audio file, which was then played back. Complicated processes required longer waiting times to hear the results. Thus, there was no physical means of interacting with the sounds, as with a musical instrument, and no means of improvisation or live performance (Manning 187-190).

The next generation of digital synthesizers overcame these limitations by utilizing the silicon transistor and microprocessor, innovations that brought significant advancement in terms of the size, cost, and speed of computers. The first of these synthesizers, the Synclavier, was developed at Dartmouth College, and was a self-contained instrument complete with a dedicated audio microprocessor and keyboard controller. It was capable of producing 24 sine waves to form complex timbres and included foot pedals to control the envelope and overall volume of the sounds. The next important digital hardware synthesizer, the Fairlight, created timbres from previously recorded waveform samples, rather than computing them in real time (224-225). This synthesizer found its way into the hands of some mainstream users, such as John Paul Jones and Stevie Wonder (Manning 222-223).

One of the most important events in terms of expanding digital synthesizers into the commercial market was the development of the Musical Instrument Digital Interface, or MIDI. Announced in 1982, MIDI was developed after much deliberation amongst several manufacturers, including Yamaha, Korg, and Moog (Manning 267). Despite what its name suggests, MIDI is not so much an interface as it is a standard protocol for communicating musical information in a digital format. MIDI reduces musical information such as pitch, note length, and velocity (how hard a note is hit) into a single stream of data, capable of controlling a variety of audio-producing software that speaks the MIDI language. It also allows digital devices to communicate via a single cable, meaning that one MIDI keyboard is capable of controlling a variety of synthesis software in a computer.

Digital Audio

Before moving on to software synthesis, it is important to have a basic understanding of computers and digital audio. Computers take in data about the physical world in the form of numbers. They store that information, process it, and spit the results back out. For this reason, most computers are thought to have these main parts: input devices, such as keyboards, memory for storing data, a central processing unit which performs calculations, an output device such as a screen, and perhaps some form of bulk storage, such as a hard drive (Manning 183-185).

In the case of audio, the real-world data that is being stored and processed is a series of numbers that represent the amplitudes of a sound wave. However, whereas analog electronics store a continuous audio signal in the form of varying electrical voltage, digital electronics are capable of storing only two voltages: a low or a high state, nothing in between. This means they can only store discrete numbers as opposed to the infinite resolution of analog gear. The two states of low and high voltage are used to represent the numbers 0 and 1, respectively. It is possible to store any value in a binary or base-2 number system using just 0s and 1s. While a decimal or base-10 number system has its advantages for humans (we can count with 10 fingers and represent large numbers with less digits), binary math is ruthlessly efficient. Binary number systems have existed for hundreds of years, although it was not until technological developments of the 20th century that people were able to take full advantage of binary math and binary logic (Pohlmann 5-7).

In short, digital audio is a series of binary numbers representing the amplitudes of a sound wave. In the case of digital recording, these numbers are “sampled” from analog signals at regular intervals in time. The number of samples taken per second is known as the sampling rate, and it determines the highest frequency capable of being represented by a digital signal. This highest frequency is called the Nyquist frequency and is roughly half the sampling rate. So, a sampling rate of 44.1 kHz is capable of representing frequencies up to roughly 22 kHz. If frequencies above the Nyquist frequency are sampled, they will be inaccurately stored as lower frequencies, a type of distortion known as aliasing (Pohlmann 20-25).

Another important consideration in digital audio is the bit depth. The bit depth is the amount of memory dedicated to storing one sample, measured in binary digits or bits for short. Common bit depths include 16-bit, 24-bit, and 32-bit. The bit depth determines the number of possible values that can be stored as a sample and thus determines the dynamic range and noise floor of a digital audio signal (Pohlmann 28-37).

Analog signals are not the only source of digital samples. It is possible to create digital audio signals from scratch, using mathematical formulas to generate each sample. This, in a nutshell, is digital audio synthesis.

Part II: Software Synthesis Techniques

Software synthesis has come a long way since its inception by Max Mathews. Contrary to early forms, contemporary software synthesis can be performed in real time on the tiniest of computers. Whether realized on a desktop, tablet, or smart phone, the fundamental techniques remain the same.

The basis of any software synthesizer is the unit generator, an individual module capable of performing a simple task (Roads, *Computer Music Tutorial* 89). Unit generators are essentially the software equivalent of analog modules, each performing a distinct task such as waveform generation or filtering. Like modules, unit generators provide flexibility, so that a limited number of generators can be routed to create a variety of synthesis instruments.

Perhaps the most important type of unit generator is the wavetable oscillator. A wavetable is a short array of samples stored in memory, representing one cycle of a waveform. The alternative to wavetables is to calculate each sample in real time according to mathematical formulas for individual waveforms, such as sine waves. While intuitive, this method requires quick and intense processing to calculate each approaching sample, which is not an efficient use of computing power. The values stored in wavetables have been pre-computed and simply need to be read through at the appropriate sample rate to produce repeating waveforms. This technique of cycling through wavetable values is called table-lookup synthesis, and it is at the heart of most forms of software synthesis. The only downside is that a small amount of memory must be taken up, although this is negligible by today's memory standards (Roads, *Computer Music Tutorial* 90).

The envelope is another unit generator that is used to control a certain parameter over time, generally the amplitude. Envelopes are usually triggered when a new note begins. It is common to speak of ADSR envelopes, which have four distinct stages: attack, decay, sustain, and release. The attack portion refers to the time it takes for a sound to reach its highest amplitude once a note begins. The decay refers to how long it takes for the signal to drop from its peak amplitude to its sustain level. The sustain portion of an envelope is the amplitude at which it will stay for as long as a note is held. Finally, the release refers to the time it takes for the amplitude to fall to zero once the note is released. The envelope of a sound has a dramatic effect on its perceived timbre. Some sounds, such as the pluck of a guitar string, have a fast attack and therefore sound up-front and percussive. Sounds with slower attacks include the swell of a violin, and they often appear to be quieter, although they may reach the same peak amplitude. It is also possible to assign envelopes to modulate other parameters, such as pitch and filter cutoff frequency, to create interesting effects (Roads, *Computer Music Tutorial* 97).

A final notable unit generator is the low frequency oscillator, or LFO. Like a standard oscillator, an LFO creates a signal that varies over time, albeit at a rate below the range of human hearing. Therefore, rather than create sound of their own, LFOs can be used to create subtle or extreme rhythmic variations in certain parameters. LFOs are important because they create movement in volume, pitch, or tone, breathing life into otherwise static sounds. Like the analog module, a filter boosts or reduces specific frequencies of a complex tone. Other unit generators include amplifiers, which simply boost or reduce the amplitude of a signal, and filters, which boost or reduce certain frequencies within a signal.

These unit generators provide the basis for any form of digital synthesis, including both hardware and software synthesis. Digital hardware synthesizers, such as those discussed in the history of digital synthesis, feature dedicated signal processing chips that perform necessary calculations (Roads, *Computer Music Tutorial* 100). These chips are specifically designed to deal with audio signals, and so they provide reliability and real time results. However, they are often not flexible, and can perform only the synthesis tasks that they are designed to perform by the manufacturer. Software, on the other hand, is a set of instructions in memory that can be executed by a general computer. Software can be reprogrammed to perform a variety of synthesis tasks, and thus it provides more flexibility and convenience than hardware. As renowned computer musician and scholar Curtis Roads explains:

> [A]ll of the pioneering work in computer music was carried out via software synthesis. Today a variety of synthesis programs run on inexpensive personal computers. …A great advantage of software synthesis is that a small computer can realize any synthesis method—even the most computationally intensive—

> provided that the musician has the patience to wait for results. Thus, with little else needed but a musical will, computers are primed and ready for high-quality music synthesis. (*Computer Music Tutorial* 100)

Even the act of waiting for results is becoming a thing of the past, as personal computers become more and more powerful. Today, it is not uncommon to synthesize audio signals in real time even on the smallest of computers, such as tablets and smart phones. Thus, software synthesis provides the means to create virtually infinite sounds, limited only by time and imagination.

Using just the concepts provided so far, it is possible to mimic the two most common forms of analog synthesis: additive and subtractive synthesis. There are many available software synthesizers, or soft synths, capable of doing just that. Modern software, however, is capable of performing new forms of synthesis that are not possible or practical in the analog domain. These forms include wavetable modulation, frequency modulation, and granular synthesis, among others.

Wavetable Modulation Synthesis

In order to synthesize specific waveforms such as triangle or sawtooth waves, analog synthesizers require specialized circuits. These circuits generally produce the same waveform cycles over and over again, providing a consistent tone, which may be a building block for complex timbres. Wavetables, on the other hand, require only a small amount of memory to store one cycle of a waveform (Roads, *Computer Music Tutorial* 90). Furthermore, digital oscillators can easily switch between reading through different wavetables. This means it is fairly easy to create evolving timbres by cycling through multiple waveforms in a short period of time - a process known as wavetable modulation synthesis.

Wavetable modulation synthesis goes by many names, including vector synthesis, multiple wavetable synthesis, transitional synthesis, and compound synthesis, though the process is always the same. Whereas subtractive and additive synthesis generate complex timbres and the harmonics fade away leaving only the fundamental, wavetable modulation allows a gradual change in harmonic content (Wiffen). Through the use of envelopes and LFOs, the sound appears to morph from one timbre to another. This is generally achieved by crossfading between waveforms. That is, one waveform gradually fades away while another one gradually fades in. It is important that the different waveforms being used are closely related in harmonic content. Otherwise, the transition will sound unnatural and harsh, possibly introducing clicks and pops. It is possible to morph between very different waveforms, although a larger number of wavetables should be used to make the transition seem natural.

Groups of similar wavetables are referred to as “wave terrains” or “three-dimensional wavetables” (Roads, *Computer Music Tutorial* 164). Often times, wavetables may be drawn in by the user to create wave terrains, as was the case with the Fairlight, arguably the earliest version of a wavetable modulation synth (Wiffen). In other cases, wave terrains can be generated mathematically or simply provided by the manufacturer. Native Instruments’ Massive, for example, is one of the most powerful wavetable modulation synths available and provides over 80 wave terrains to choose from (“Massive Feature Details”).

Wavetable modulation is uniquely powerful in its ability to create morphing timbres and thus create interest. One of the great challenges of synthesis is to avoid creating static, unchanging sounds that often bore or even annoy the ear. While traditional methods can create movement by modulating pitch, filters, or amplitude, wavetable modulation provides the ability to truly change the timbre of a sound in ways that filters cannot.

Frequency Modulation Synthesis

Frequency modulation is a technique that had been around long before it was a synthesis technique. It was and is the technique used in FM radio, whereby the frequency of a signal is modulated at a rate of nearly 100 MHz for the purpose of traveling long distances. However, for many years no one had thought of modulating a signal's frequency at a rate in the range of human hearing. In 1967, this all changed. In that year, musician John Chowning was experimenting with vibrato at a Stanford studio when he brought the vibrato rate into the range of human hearing. When this happened, rather than hearing a distinct movement of pitch, he heard an increasingly complex tone take the place of the original one. This was the first example of frequency modulation synthesis, or FM synthesis. Due to the instability of pitch in analogue synthesizers, FM is almost exclusively implemented in the digital realm (Reid, "Yamaha GS1 & DX1").

FM synthesis requires at least two oscillators which are called operators in this technique. One operator is the carrier, which is the signal that gets modulated. The other is the modulator, the signal that gets routed to the frequency of the carrier. As the amplitude of the modulator increases, so does the energy of the additional partials added to the signal. These partials are known as "sidebands" in FM synthesis, because they occur both above and below the fundamental frequency. This fact is one of main reasons FM synthesis can create such unique timbres.

There are a few main parameters in FM synthesis that affect the resulting timbre. The frequency of the carrier determines the fundamental frequency of the resulting signal. The frequency of the modulator is the rate at which the modulation occurs. This parameter determines how far apart the sidebands will be. Finally, the modulation index is the peak deviation of the carrier's frequency divided by the amplitude of the modulator ("FM Index"). Peak deviation refers to the maximum change in frequency undergone by the carrier. For example, if the carrier's frequency is 1000 Hz and it is being modulated up to 1100 Hz and down to 900 Hz, the peak deviation is 100 Hz. With an amplitude of 1, the index would be .1. The index is directly proportional to the amplitude of the modulating wave. In other words, increasing amplitude of the modulator results in a larger total amount of modulation, thus increasing the strength of sidebands (Reid, "An Introduction to Frequency Modulation"). It is largely the index and the relationship between the frequencies of the operators that determine the resulting timbre. For this reason, it is common to express the modulating frequency as a ratio to the carrier's frequency rather than in absolute terms, so that the timbre remains constant over a wide range of notes.

With an index of 0, the resulting signal is simply the output of the carrier. As the index increases, energy is gradually "stolen" from the fundamental and given to an increasing number of sidebands (Chowning 527). At relatively low indexes of say .1 or .2, only those sidebands nearest to the fundamental will be heard. At higher indexes, more extreme sidebands begin to gain energy. If the modulating frequency is 100 Hz, then these sidebands will occur at regular intervals of 100 Hz away from the fundamental.

Given this information, the relationship between the three main parameters and the resulting spectra can be summarized by the following formula: F(c) ±K*F(m), where F(c) is the carrier frequency, F(m) is the modulating frequency, and K is all whole numbers beginning with 0 ("Bessel Functions"). To give an example, take a carrier with a frequency of 440 Hz, and a modulator with a frequency of 44 Hz. The location of first-order sidebands will be 396 Hz and 484 Hz, or 440 ± 44 Hz. The second-order sidebands will be located at 352 Hz and 528 Hz, or 440 ± 88 Hz, and the third-order sidebands will be located at 308 Hz and 572 Hz, or 440 ± 132 Hz, and so on. At low index levels, only the first order sidebands will be heard. It takes high index levels for the higher order sidebands to start emerging.

One important concern that is relevant at high indexes is phase. In our previous example, the 11th order sidebands would consist of frequencies at 440Hz ± 484 Hz, or 924 Hz and what would appear to be -44 Hz. A negative frequency simply indicates that same frequency but phase inverted (Reid, "More on Frequency Modulation"). In simple FM patches, this is not relevant and would just be perceived as a partial at 44 Hz, however in patches involving many operators it is something that must be considered.

A final important consideration is the relationship between the modulator and carrier frequencies. When ratios of integer numbers are used, such as 2:1, the resulting spectra will be of the harmonic series (Chowning 529). Ratios of odd numbers, such as 3:1, can create only odd-ordered harmonics, while ratios of say 4:1 would produce only every 4th harmonic. Ratios including irrational numbers such as π:1 will create inharmonic spectra and noisy timbres. Again, the ratio is usually something that is set as a synthesis parameter and thus remains constant across a wide range of notes or fundamental frequencies.

Due to the complex spectra that result from the act of frequency modulation, it is not necessary to use waveforms other than sine waves to create interesting timbres. The use solely of sine waves is known as "simple FM" and can yield many results. However, contemporary FM synthesizers offer the ability to use a variety of waveforms as both the carrier and modulator. Using these waveforms will yield incredibly complex timbres that are often difficult to predict. In fact, even simple FM, while mathematically predictable, is best explored through experimentation rather than math. As Chowning notes:

> Certainly the complexity in the evolution of each of the components of the spectrum makes an important contribution to the lively quality of FM sounds. Because this complexity is a function of the laws of the equation, it is surprising that while the evolution of the components is rigidly determined, they can still produce such rich and

varied subjective impressions. (530)

Indeed, it takes only a matter of minutes to begin exploring the possibilities and surprises of FM synthesis. These surprises will increase as more waveforms and parameters are added to synthesis instruments. The most famous FM synth of all time, the DX7, featured six operators and used only sine waves (Reid, “Yamaha GS1 and DX1”). Today, instruments offer even more substantial capabilities. Native Instrument’s FM8 offer 8 operators and a variety of waves to choose from including such as noise waves (“FM8 Feature Details”). Furthermore, multiple modulators may be used, as well as routing one modulator into another. Newer instruments include many LFOs and envelopes for complex modulation.

Whereas wavetable modulation synthesis creates morphing timbres unlike those heard in the natural world, FM synthesis is known for its unique ability to synthesize realistic instrument timbres. In particular it is great for producing bell-like tones, which naturally have sideband frequencies above and below the fundamental. Percussive sounds are easy to synthesize by featuring a high index during the attack portion, which quickly decays to a low index. Finally, woodwinds and brass sounds may be synthesized by paying careful attention to the carrier to modulator frequency ratio and thus the resulting harmonic series (Chowning 532).

Wavetable and frequency modulation both provide means of exploring continuous tones and timbres. On the other hand, granular synthesis, another technique performed exclusively in the software realm, may be the form of synthesis best suited for creating interesting soundscapes and textures.

Granular Synthesis

Iannis Xenakis was arguably the first composer to implement and describe the technique of granular synthesis (Roads, *Microsound* 65). He describes the central theory of sound grains as follows: “All sound, even continuous musical variation, is conceived as an assemblage of a large number of elementary sounds adequately disposed in time. In the attack, body, and decline of a complex sound, thousands of pure sounds appear in a more or less short interval of time” (Xenakis 43).

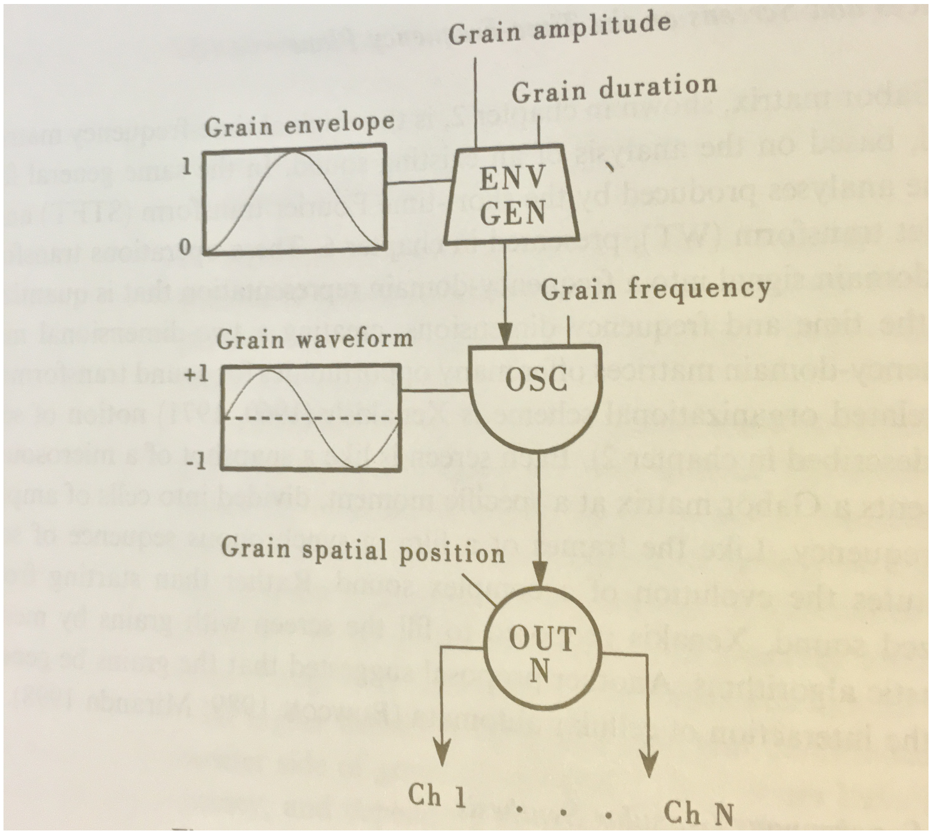
Indeed, there are cases where such “elementary sounds” are quite evident, as in the babbling of a brook, the rolling of the tongue, or the crackling of fire, which was the basis of Xenakis’ famous composition, *Concret PH*. Still, even continuous tones may be said to be the result of individual physical events too fast for the ear to distinguish. In granular synthesis, a short sound event in the range of 1 to 100 ms is used as the basis for creating interesting textures and tones (Roads, *Microsound* 87). These sound events are known as “grains.”

A single or several grains may be the basis for creating complex sounds and textures. According to Roads:

> The grain is an apt representation of musical sound because it captures two perceptual dimensions: time-domain information (starting time, duration, envelope shape) and frequency-domain information (the pitch of the waveform within the grain and the spectrum of the grain). This stands in opposition to sample-based representations that do not capture frequency-domain information, and abstract Fourier methods, which account only for the frequency domain. (*Microsound* 87)

The previously discussed forms of software synthesis rely mostly on manipulating the frequency domain, whereas sampling relies on manipulating the time domain. In this light, granular synthesis can be viewed as a hybrid of synthesis and sampling.

The most important component of a granular synthesizer is the grain generator. A grain generator includes a wavetable oscillator, which may read through traditional waveform tables or wavetables extracted from samples of pre-existing audio (Roads, *Microsound* 90). In either case, there are several different parameters of the oscillator and related unit generators. One important parameter is the envelope shape. Often times, a bell curved Gaussian envelope is used to smooth out harsh edges of a grain and prevent audible clicks and pops, although other envelopes are possible. Other parameters include the frequency, amplitude, and duration of the individual grain, as well as per-grain effects and per-grain panning (91). The routing of a typical grain generator is shown below:



Considering the variety of per-grain parameters as well as the large number of grains that must be used to create several seconds of sound, it would be incredibly tedious to specify parameters and location of each individual grain. Therefore, granular synthesizers generally feature high-level grain organization structures. These structures offer certain controls that organize the density and parameters of grains according to random or specific patterns. There are many ways of organizing grains, but two of the most common forms are synchronous and asynchronous granular synthesis.

Synchronous granular synthesis refers to streams of grains at regular intervals apart (Roads, *Microsound* 93). In this and other forms of granular synthesis, a “density” parameter refers to the number of grains per second, which also may be thought of as the frequency of grains, measured in Hertz. At a density of less than 20 grains per second, rhythmic effects are heard. As the density of grains increases into the range of human hearing and grains begin to overlap, a continuous tone emerges. Usually, the density of grains forms the fundamental frequency of the timbre, while the waveform and envelope shape contribute to possible sidebands and overtone series. Quasi-synchronous granular synthesis also refers to a linear stream of grains but at irregular intervals apart. These intervals may be random or the result of specific modulation.

The perceived pitch of the resulting signal is not entirely dependent upon the density of grains, however. There is a relationship between the density of grains, the waveform frequency, and the grain envelope (Roads, *Microsound* 94). For example, if the frequency of the grain’s waveform is 200 Hz and therefore its period is 5 ms, a density of 100 grains per second will not necessarily create a perceived pitch of 100 Hz. In this case, the grains will begin every 10 ms while the grain duration is only 5 ms, leaving a 5 ms gap between grains. On the other hand, if the grain duration is significantly longer than the period of the grain density, the waveform will barely have a chance to unfold before it begins again, and the resulting signal will have low amplitude and little sense of pitch. In general, if the expected pitch is not being obtained, the relationship between these parameters must be explored and adjustments must be made. Still, pitch-ambiguous sounds may be desirable in some cases.

Asynchronous granular synthesis, perhaps the most common form of granular synthesis, does not feature linear streams of grains but rather grains randomly (or semi-randomly) dispersed over a duration, sometimes overlapping and other times not (Roads, *Microsound* 98). These disbursements of grains are known as "clouds," and parameters may be varied within certain defined limits within the duration of a cloud. Some of the parameters that may change within a cloud are grain duration, grain envelope, grain density, waveform, and spatial positioning. These parameters may be varied randomly within defined limits, or perhaps linked to other parameters, such as note or frequency of the sampled grain.

Envelope variation is one of the simplest ways to develop texture within a cloud. Rather than a Gaussian curve, an expodec (exponential decay) envelope may be used to give a percussive quality to grains. A rexpodec (reverse exponential decay) envelope, which features a slow attack and quick decay, gives the illusion of reversed grains. Grain waveform can also be varied within a cloud, leading to cloud "color" type. Monochrome clouds feature only a single grain waveform, polychrome clouds feature two or more waveforms simultaneously, and transchrome clouds feature one consistently evolving waveform over the course of the cloud (Roads, *Microsound* 100).

The overall texture of the cloud is determined by the relationship between grain density and grain duration. The grain density may be low, but if long grain durations are used the texture may appear full and continuous, whereas short grain durations may appear sparse and transparent. Ultimately, the product of density and duration contribute to the cloud's "fill factor." In a sparse fill factor of less than .5, more than half of the cloud is silent. A covered fill factor of roughly 1 means that the cloud is largely filled with grains but not much overlap. A packed fill factor of greater than 1 means the cloud will be dense and the grains overlapping. In any case, even individual clouds may be used as building blocks for larger textures that feature overlapping clouds (Roads, *Microsound* 105).

Several other grain organizations include Gabor matrices, pitch-synchronous granular synthesis, physical modeling, and granulation. Gabor matrices are a way of combining time and frequency domains by analyzing spectra of sounds in short windows of time, rather than purely examining sound in the time or frequency domains, both of which have limitations. Pitch synchronous granular synthesis involves analyzing a sound for its spectral content and then resynthesizing it with a specific pitch and formant region. Physical modeling involves analysis of acoustic events, and granulation involves splitting a sound into individual grains and then reassembling those grains in a new fashion. Granulation is one of the techniques used for time-stretching a sound while keeping its pitch constant (Roads, *Microsound* 92-98).

Granular synthesis is probably one of the least understood forms of software synthesis due to its complexity and many varieties. Whereas wavetable modulation and frequency modulation are both adept at producing interesting timbres, granular synthesis is perhaps best at producing varied textures and soundscapes. As with any form of synthesis, the best way to obtain effective results is through experimentation and practice.

Part III. The Tools of Software Synthesis

Frequency modulation, amplitude modulation, vector synthesis, pulsar synthesis, FOF synthesis; the list goes on and on. For the many daunting varieties and sub-varieties of software synthesis, luckily, there are only a handful of environments for realizing such techniques. These environments can be grouped into three main categories: user-friendly virtual instruments, graphical programming environments, and low-level coding environments.

User-friendly Virtual Instruments

User-friendly virtual synthesizers are just what they sound like; they are ready-to-use software synthesizers that can run as stand-alone applications or as plugins hosted by a digital audio workstation, or DAW (Walker). Support for plugin formats varies according to the DAW, but some of the most common types include VST, AU, and AAX.

Native Instruments, as already mentioned, is one of the leaders in virtual synthesizer plugins, but other manufacturers include Arturia, Korg, and Moog, to name just a few. Synthesis type and features vary widely by manufacturer, but most include multiple oscillators and support for polyphony, as well as plenty of envelopes and LFOs for modulation. Many are emulations of classic analog synths, while others are completely unique to the software realm. Advanced features include built-in sequencers, customizable waveforms, effects, and modular routing.

The main benefit of these synthesizers is that they are ready to use without any programming, thus offering the quickest means for getting musical ideas into the computer. They also include many presets for quickly exploring timbres and are easy to learn. For beginning musicians, these synths are definitely a great starting point for exploring synthesis techniques. The drawbacks of these synths are that they are not flexible in what they can do and they provide only a limited understanding of what is going on beneath the hood, so to speak. Often times, manufacturers use their own terminology for certain parameters, so it is not always clear what a virtual knob or slider does. Finally, sound quality can vary widely according to the manufacturer.

Graphical Programming Environments

Graphical or visual programming environments are a level below user-friendly soft synths. They are applications that allow users to create their own synthesis instruments, generally by patching together different visual objects that each perform a specific task (Storr). Often times, they provide user-created presets or “patches” for quick use, while still offering the ability to deconstruct and reroute the objects. Examples of these environments include Max/MSP, Pure Data, and Native Instruments Reaktor.

One of the main benefits is a more intimate understanding of digital signal routing and the various forms of synthesis. The user is forced to look more closely at the relationship between different signals such as oscillators, envelopes, and LFOs, while also thinking about certain "control" elements such as shutting off processing when a unit generator is not in use. The other major advantage of these tools is flexibility, as a patch may always be added to or modified. This allows experimentation and combining different forms of synthesis into one instrument. Drawbacks include a steeper learning curve and a certain amount of time spent programming rather than making music, which can take a musician out of the creative mindset.

Low-level Coding Environments

The final tool for realizing synthesis is the low-level coding environment. Rather than using visual objects to perform tasks, low-level tools require writing many lines of code and knowledge of programming languages such as C++ or Python. Lines of code are "compiled" in integrated development environments or IDEs, such as Xcode or Visual Studio. However, the entirety of the code need not be written in the IDE. Standard development kits, or SDKs, are often provided by plugin manufacturers and contain many files of code that aid in audio processing, audio synthesis, or file organization. An SDK might provide code that takes care of the plugin-format and routing audio through the audio driver, allowing the programmer to focus on the actual signal processing or synthesis code. An example of a common SDK is the VST SDK provided by Steinberg.

Although the final lines of code are compiled in IDEs, programmers may use an application programming interface, or API, along the way. An API is an environment dedicated to programming for a specific task, such as audio, which often includes a visual interface as well as pre-written code for common tasks ("Projucer and Juce 4"). The most common API used in the audio world is Juce, a C++ programming environment. Juce provides a visual interface for creating the graphical user interfaces of plugins, meaning that the audio programmer does not need to write lines of code focusing on graphics. It also helps organize SDK files appropriately for the desired plugin format, and includes files of code containing algorithms for common tasks such as filtering or amplification. Some of the manufacturers that develop software in Juce include Cycling ‘74, Universal Audio, Arturia and Korg.

The main advantage of low-level coding is that it provides ultimate flexibility. Any form of synthesis can be realized as well as new forms yet to be explored by the audio community. This method provides an enhanced understanding of audio synthesis. The obvious drawback of low-level coding is the steep learning curve, requiring knowledge of a programming language and perhaps some advanced math. This is also the furthest removed from actual creation of music, requiring hours of coding and debugging before a useable product is created. These environments are best suited for those who want to get into professional audio programming for profit.

Conclusion

Since its creation by Max Mathews in the 1950s, software synthesis has led a revolution in terms of providing a convenient means of sound creation for musicians. However, many of these forms, such as wavetable modulation, frequency modulation, and granular synthesis, are still being explored and expanded upon. Surely, there are new forms of synthesis that will soon be discovered, and the tools for realizing these forms will continue to grow in convenience, flexibility, and affordability. When they do, the general public may first deem these new forms to be nothing but noise, but there will be those few individuals like Varèse at the forefront of the revolution, listening to the music.

Works Cited

"Bessie." *Skidmoremusic.org*. Skidmore College, n.d. Web. 23 May 2016.

Chowning, John. "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation." *Journal of the Audio Engineering Society* 21.7 (1973): 526- 34. Web. 23 May. 2016.

"FM8 Feature Details." *Native-instruments.com*. Native Instruments, n.d. Web. 23 May 2016.

"FM Index." *Skidmoremusic.org*. Skidmore College, n.d. Web. 23 May 2016.

Manning, Peter. *Electronic and Computer Music*. 4th ed. New York: Oxford UP, 2013. Print.

"Massive Feature Details." *Native-instruments.com*. Native Instruments, n.d. Web. 23 May 2016.

Pohlmann, Ken C. *Principles of Digital Audio*. 6th ed. New York: McGraw-Hill, 2011. Print.

"Projucer and JUCE 4." *Juce.com*. Juce, 2016. Web. 23 May 2016.

Reid, Gordon. "An Introduction To Frequency Modulation." *Soundonsound.com*. Sound on Sound, Apr. 2000. Web. 23 May 2016.

Reid, Gordon. "More On Frequency Modulation." *Soundonsound.com*. Sound on Sound, May 2000. Web. 23 May 2016.

Reid, Gordon. "Synthesizing Bowed Strings: The Violin Family." *Soundonsound.com*. Sound on Sound, Apr. 2003. Web. 23 May 2016.

Roads, Curtis. *The Computer Music Tutorial*. Cambridge, Mass: MIT, 1996. Print.

Roads, Curtis. *Microsound*. Cambridge, Mass.: MIT, 2004. Print.

Smith, Steven W. *The Scientist and Engineer's Guide to Digital Signal Processing*. N.p.: California Technical, 1998. Web. 23 May 2016.

Snoman, Rick. *Dance Music Manual: Tools, Toys and Techniques*. 2nd ed. Amsterdam: Focal, 2009. Print.

Storr, Nick. "VST DIY." *Soundonsound.com*. Sound on Sound, Sept. 2009. Web. 23 May 2016.

Varèse, Edgard. "The Liberation of Sound." *Perspectives of New Music* 5.1 (1966): n. pag. *Music.arts.uci.edu*. Web. 23 May 2016.

Walker, Martin. "Using VST Instruments." *Soundonsound.com*. Sound on Sound, Dec. 2000. Web. 23 May 2016.

Wiffen, Paul. "Synth School, Part 7: Transitional Synthesis." *Soundonsound.com*. Sound on Sound, Apr. 1998. Web. 23 May 2016.

Xenakis, Iannis. *Formalized Music; Thought and Mathematics in Composition*. Bloomington: Indiana UP, 1971. Print.