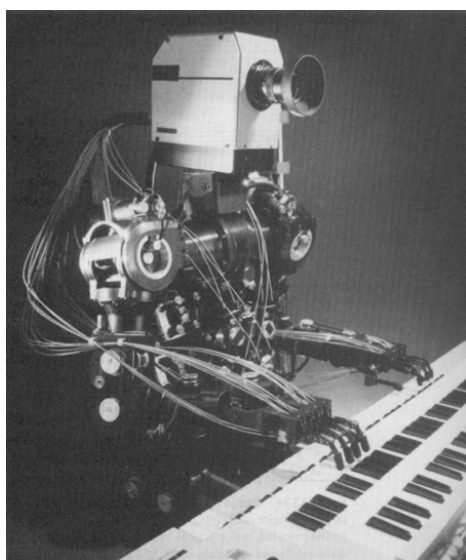


CHAPTER 7

Synthesis and digital music

7.1. Introduction



WABOT-2
(Waseda University and
Sumitomo Corp., Japan 1985)

In this chapter, we investigate synthesis of musical sounds. We pay special attention to Frequency Modulation (or FM) synthesis, not because it is a particularly important method of synthesis, but rather because it is easy to use FM synthesis as a vehicle for conveying general principles. We also discuss other aspects of digital music, such as aliasing and Nyquist's theorem, MIDI, and internet resources.

Interesting musical sounds do not in general have a static frequency spectrum. The development with time of the spectrum of a note can be understood to some extent by trying to mimic the sound of a conventional musical instrument synthetically. This exercise focuses our attention on what

are usually referred to as the attack, decay, sustain and release parts of a note (ADSR). Not only does the amplitude change during these intervals, but also the frequency spectrum. Synthesizing sounds which do not sound mechanical and boring turns out to be harder than one might guess. The ear is very good at picking out the regular features produced by simple minded algorithms and identifying them as synthetic. This way, we are led to an appreciation of the complexity of even the simplest of sounds produced by conventional instruments.

Of course, the real strength of synthesis is the ability to produce sounds not previously attainable, and to manipulate sounds in ways not previously possible. Most music, even in today's era of the availability of cheap and powerful digital synthesizers, seems to occupy only a very small corner of the available sonic palette. The majority of musicians who use synthesizers just punch the presets until they find the ones they like, and then use them without modification. Exceptions to this rule stand out from the crowd; listening

to a recording by the Japanese synthesist Tomita, for example, one is struck immediately by the skill expressed in the shaping of the sound.

Further listening: (See Appendix R)

Isao Tomita, *Pictures at an Exhibition* (Mussorgsky).

7.2. Digital signals



The commonest method of digital representation of sound is about as simple minded as you can get. To digitize an analog signal, the signal is sampled a large number of times a second, and a binary number represents the height of the signal at each sample time. Both of these processes are sometimes referred to as *quantization* (don't worry, there's no quantum mechanics involved here), but it is important to realize that the processes are separate, and need to be understood separately.

For example, the Compact Disc is based on a sample rate of 44.1 KHz, or 44,100 sample points per second.¹ At each sample point, a sixteen digit

binary number represents the height of the waveform at that point.

One way to represent the process of sampling a signal is as multiplication by a stream of Dirac delta functions (see §2.15). Let N denote the sample rate, measured in samples per second, and let $\Delta t = 1/N$ denote the interval between sample times. So for example for compact disc recording we

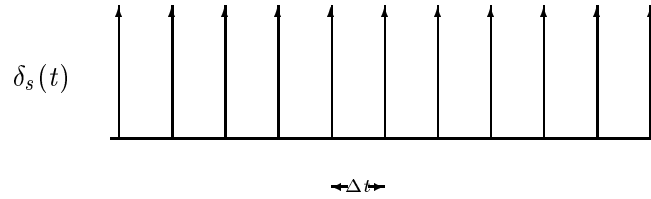
¹It is annoying that the default sample rate for DAT (Digital Audio Tape) is 48 KHz, thereby making it difficult to make a digital copy on CD directly from DAT. This seems to be the result of industry paranoia at the idea that anyone might make a digital copy of music from a CD (DAT was originally designed as a consumer format, but never took off except among the music business professionals). The excuse that the higher sample rate for DAT gives a higher cutoff frequency and therefore better audio fidelity is easily seen through in light of the fact that the improvement is about three quarters of a tone, which is essentially insignificant.

Fortunately, the ratio 48,000/44,100 can be written as a product of small fractions, $4/3 \times 8/7 \times 5/7$, which suggests an easy method of digital conversion. To multiply the sample rate by 4/3, for example, we use linear interpolation to quadruple the sample rate and then omit two out of every three sample points. This gives much better fidelity than converting to an analog signal and then back to digital.

want $N = 44,100$ samples per second, and $\Delta t = 1/44,100$ seconds. We define the *sampling function* with spacing Δt to be

$$\delta_s(t) = \Delta t \sum_{n=-\infty}^{\infty} \delta(t - n\Delta t).$$

The reason for the factor of Δt in front of the summation is so that the integral of this function over an interval of time approximates the length of the interval.



If $f(t)$ represents an analog signal, then

$$f(t)\delta_s(t) = \Delta t \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta t) = \Delta t \sum_{n=-\infty}^{\infty} f(n\Delta t)\delta(t - n\Delta t)$$

represents the sampled signal. This has been digitized with respect to time, but not with respect to signal amplitude. The integral of the digitized signal $f(t)\delta_s(t)$ over any period of time approximates the integral of the analog signal $f(t)$ over the same time interval.

One of the keys to understanding the digitized signal is Poisson's summation formula from Fourier analysis.

THEOREM 7.2.1.

$$\Delta t \sum_{n=-\infty}^{\infty} f(n\Delta t) = \sum_{n=-\infty}^{\infty} \hat{f}\left(\frac{n}{\Delta t}\right). \quad (7.2.1)$$

PROOF. This follows from the Poisson summation formula (2.14.1), using Exercise 3 of §2.13. \square

COROLLARY 7.2.2. *The Fourier transform of the sampling function $\delta_s(t)$ is another sampling function in the frequency domain,*

$$\hat{\delta}_s(\nu) = \sum_{n=-\infty}^{\infty} \delta\left(\nu - \frac{n}{\Delta t}\right).$$

PROOF. If $f(t)$ is a test function, then the definition of $\delta_s(t)$ gives

$$\int_{-\infty}^{\infty} f(t)\delta_s(t) dt = \Delta t \sum_{n=-\infty}^{\infty} f(n\Delta t).$$

Applying Parseval's formula (2.13.4) to the left hand side (and noting that the sampling function is real, so that $\delta_s(t) = \overline{\delta_s(t)}$) and applying formula

(7.2.1) to the right hand side, we obtain

$$\int_{-\infty}^{\infty} \hat{f}(\nu) \widehat{\delta_s}(\nu) d\nu = \sum_{n=-\infty}^{\infty} \hat{f}\left(\frac{n}{\Delta t}\right).$$

The required formula for $\widehat{\delta_s}(\nu)$ follows. \square

COROLLARY 7.2.3. *The Fourier transform of a digital signal $f(t)\delta_s(t)$ is*

$$\widehat{f\delta_s}(\nu) = \sum_{n=-\infty}^{\infty} \hat{f}\left(\nu - \frac{n}{\Delta t}\right)$$

which is periodic in the frequency domain, with period equal to the sampling frequency $1/\Delta t$.

PROOF. By Theorem 2.16.1(ii), we have

$$\widehat{f\delta_s}(\nu) = (\hat{f} * \hat{\delta_s})(\nu),$$

and by Corollary 7.2.2, this is equal to

$$\int_{-\infty}^{\infty} \hat{f}(u) \sum_{n=-\infty}^{\infty} \delta\left(\nu - \frac{n}{\Delta t} - u\right) du = \sum_{n=-\infty}^{\infty} \hat{f}\left(\nu - \frac{n}{\Delta t}\right). \quad \square$$

7.3. Nyquist's theorem

Nyquist's theorem² states that the maximum frequency that can be represented when digitizing an analog signal is exactly half the sampling rate. Frequencies above this limit will give rise to unwanted frequencies below the *Nyquist frequency* of half the sampling rate.

To explain the reason for this, consider a pure sinusoidal wave with frequency ν , for example

$$f(t) = A \cos(2\pi\nu t).$$

Given a sample rate of $N = 1/\Delta t$ samples per second, the height of the function at the M th sample is given by

$$f(M/N) = A \cos(2\pi\nu M/N).$$

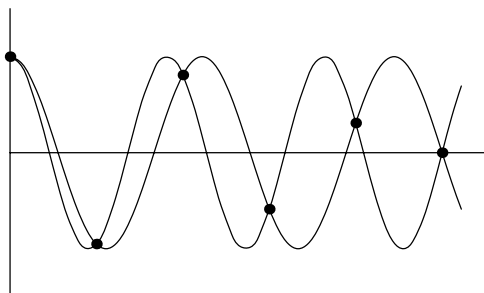
If ν is greater than $N/2$, say $\nu = N/2 + \alpha$, then

$$\begin{aligned} f(M/N) &= A \cos(2\pi(N/2 + \alpha)M/N) \\ &= A \cos(\pi M + 2\pi\alpha M/N) \\ &= (-1)^M A \cos(2\pi\alpha M/N). \end{aligned}$$

Changing the sign of α makes no difference to the outcome of this calculation, so this gives exactly the same answer as the waveform with $\nu = N/2 - \alpha$ instead of $\nu = N/2 + \alpha$. To put it another way, the sample points in

²Harold Nyquist, *Certain topics in telegraph transmission theory*, Transactions of the American Institute of Electrical Engineers, April 1928. Nyquist retired from Bell Labs in 1954 with about 150 patents to his name. He was renowned for his ability to take a complex problem and produce a simple minded solution that was far superior to other approaches.

this calculation are exactly the points where the graphs of the functions $A \cos(2\pi(N/2 + \alpha)t)$ and $A \cos(2\pi(N/2 - \alpha)t)$ cross.



The result of this is that a frequency which is greater than half the sample frequency gets reflected through half the sample frequency, so that it sounds like a frequency of the corresponding amount less than half. This phenomenon is called *aliasing*. In the above diagram, the sample points are represented by black dots. The two waves have frequency slightly more and slightly less than half the sample frequency. It is easy to see from the diagram why the sample values are equal. Namely, the sample points are simply the points where the two graphs cross.

For waves at exactly half the sampling frequency, something interesting occurs. Cosine waves survive intact, but sine waves disappear altogether. This means that phase information is lost, and amplitude information is skewed.

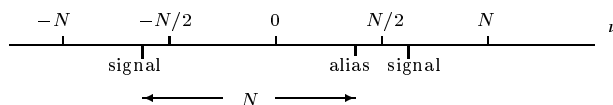
The upshot of Nyquist's theorem is that before digitizing an analog signal, it is essential to pass it through a low pass filter to cut off frequencies above half the sample frequency. Otherwise, each frequency will come paired with its reflection.

In the case of digital compact discs, the cutoff frequency is half of 44.1 KHz, or 22.05 KHz. Since the limit of human perception is usually below 20 KHz, this may be considered satisfactory, but only by a small margin.

We can also explain Nyquist's theorem in terms of Corollary 7.2.3. Namely, the Fourier transform

$$\widehat{f\delta_s}(\nu) = \sum_{n=-\infty}^{\infty} \hat{f}\left(\nu - \frac{n}{\Delta t}\right)$$

is periodic with period equal to the sampling frequency $N = 1/\Delta t$. The term with $n = 0$ in this sum is the Fourier transform of $f(t)$. The remaining terms with $n \neq 0$ appear as aliased artifacts, consisting of frequency components shifted in frequency by multiples of the sampling frequency $N = 1/\Delta t$. If $f(t)$ has a nonzero part of its spectrum at frequency greater than $N/2$, then its Fourier transform will be nonzero at plus and minus this quantity. Then adding or subtracting N will result in an artifact at the corresponding amount less than $N/2$, the other side of the origin.



Another remarkable fact comes out of Corollary 7.2.3. Namely, provided the original signal $f(t)$ satisfies $\hat{f}(\nu) = 0$ for $\nu \geq N/2$, in other words, provided that the entire spectrum lies below the Nyquist frequency, the original signal can be reconstructed exactly from the sampled signal, without any loss of information. Namely, $\hat{f}(\nu)$ is reconstructed by truncating $\widehat{f\delta_s}(\nu)$, and then $f(t)$ is reconstructed using the inverse Fourier transform. Carrying this out in practise is a different matter, and requires very accurate analog filters.

7.4. The z -transform

For digital signals, it is often more convenient to use the z -transform instead of the Fourier transform. The point is that by Corollary 7.2.3, the Fourier transform of a digital signal is periodic, with period equal to the sampling frequency. So it contains a lot of redundant information. The idea of the z -transform is to wrap the Fourier transform round the unit circle in the complex plane. This is achieved by setting

$$z = e^{2\pi i\nu\Delta t}$$

so that as ν changes in value by $1/\Delta t$, z goes exactly once round the unit circle in the complex plane. Any periodic function of ν with period $1/\Delta t$ can then be written as a function of z . The Fourier transform of the sampled signal $f(t)\delta_s(t)$ is then

$$\begin{aligned} \int_{-\infty}^{\infty} f(t)\delta_s(t)e^{-2\pi i\nu t} dt &= \int_{-\infty}^{\infty} \left(\sum_{n=-\infty}^{\infty} \Delta t f(t)\delta(t - n\Delta t) \right) z^{-t/\Delta t} dt \\ &= \sum_{n=-\infty}^{\infty} \Delta t f(n\Delta t) z^{-n}. \end{aligned}$$

The factor of Δt is just an annoying constant, and so the z -transform of the digitized signal is simply defined as

$$F(z) = \sum_{n=-\infty}^{\infty} f(n\Delta t) z^{-n}. \quad (7.4.1)$$

The Fourier transform may be recovered as

$$\widehat{f\delta_s}(\nu) = \Delta t F(e^{2\pi i\nu\Delta t}).$$

Warning. It is necessary to exercise caution when manipulating expres-

sions like equation (7.4.1), because of *Euler's joke*. Here's the joke. Consider a signal which is constant over all time,

$$\begin{aligned} F(z) &= \cdots + z^2 + z + 1 + z^{-1} + z^{-2} + \cdots \\ &= \sum_{n=-\infty}^{\infty} z^n. \end{aligned}$$

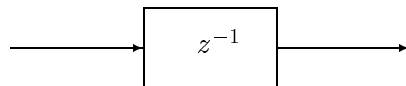
Divide this infinite sum up into two parts, and sum them separately.

$$\begin{aligned} F(z) &= (\cdots + z^2 + z + 1) + (z^{-1} + z^{-2} + \cdots) \\ &= \frac{1}{1-z} + \frac{z^{-1}}{1-z^{-1}} \\ &= \frac{1}{1-z} + \frac{1}{z-1} \\ &= 0. \end{aligned}$$

This is clearly nonsense. The problem is that the first parenthesized sum only converges for $|z| > 1$, while the second sum only converges for $|z| < 1$. So there is no value of z for which both sums make sense simultaneously.

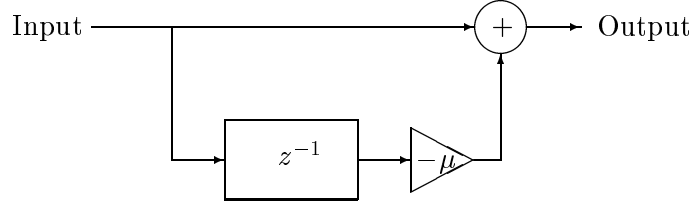
The resolution of this problem is only to allow signals with some finite starting point. So we assume that $f(n\Delta t) = 0$ for all large enough negative values of n . Then the sum converges inside the unit circle in the complex plane.

In terms of the z -transform, delaying the signal by one sample corresponds to multiplication by z^{-1} . So in the literature, you will see the block diagram for such a digital delay drawn as follows. We shall use the same convention.



7.5. Digital filters

The subject of digital filters has a vast literature. We shall only touch the surface, in order to illustrate how the z -transform enters the picture. Let us begin with an example. Consider the following diagram.



If $f(n\Delta t)$ is the input and $g(n\Delta t)$ is the output, then the relation represented by the above diagram is

$$g(n\Delta t) = f(n\Delta t) - \mu f((n-1)\Delta t). \quad (7.5.1)$$

So the relation between the z -transforms is

$$G(z) = F(z) - \mu z^{-1}F(z) = (1 - \mu z^{-1})F(z).$$

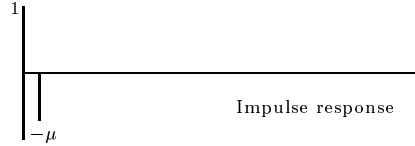
This tells us about the frequency response of the filter. A given frequency ν corresponds to the points $z = e^{\pm 2\pi i \nu \Delta t}$ on the unit circle in the complex plane, with half the sampling frequency corresponding to $e^{\pi i} = -1$.

At a particular point on the unit circle, the value of $1 - \mu z^{-1}$ gives the frequency response. Namely, the amplification is $|1 - \mu z^{-1}|$, and the phase shift is given by the argument of $1 - \mu z^{-1}$.

More generally, if the relationship between the z -transforms of the input and output signal, $F(z)$ and $G(z)$, is given by

$$G(z) = H(z)F(z)$$

then the function $H(z)$ is called the *transfer function* of the filter. The interpretation of the transfer function, for example $1 - \mu z^{-1}$ in the above filter, is that it is the z -transform of the *impulse response* of the filter.



The impulse response is defined to be the output resulting from an input which is zero except at the one sample point $t = 0$, where its value is one, namely

$$f(n\Delta t) = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0. \end{cases}$$

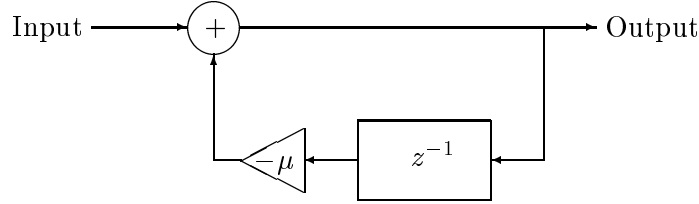
The sampled function $f\delta_s$ is then a Dirac delta function.

For digital signals, the convolution of f_1 and f_2 is defined to be

$$(f_1 * f_2)(n\Delta t) = \sum_{m=-\infty}^{\infty} f_1((n-m)\Delta t)f_2(m\Delta t).$$

Multiplication of z -transforms corresponds to convolution of the original signals. This is easy to see in terms of how power series in z^{-1} multiply. So in the above example, the impulse response is: 1 at $n = 0$, $-\mu$ at $n = 1$, and zero for $n \neq 0, 1$. Convolution of the input signal $f(n\Delta t)$ with the impulse response gives the output signal $g(n\Delta t)$ according to equation (7.5.1).

As a second example, consider a filter with feedback.



The relation between the input $f(n\Delta t)$ and the output $g(n\Delta t)$ is now given by

$$g(n\Delta t) = f(n\Delta t) - \mu g((n-1)\Delta t).$$

This time, the relation between the z -transforms is

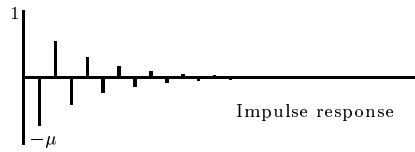
$$G(z) = F(z) - \mu z^{-1} G(z),$$

or

$$G(z) = \frac{1}{1 + \mu z^{-1}} F(z).$$

Notice that this is unstable when $|\mu| > 1$, in the sense that the signal grows without bound. Even when $|\mu| = 1$, the signal never dies away, so we say that this filter is *stable* provided $|\mu| < 1$. This is easiest to see in terms of the impulse response of this filter, which is

$$\frac{1}{1 + \mu z^{-1}} = 1 - \mu z^{-1} + \mu^2 z^{-2} - \mu^3 z^{-3} + \dots$$



Filters are usually designed in such a way that the output $g(n\Delta t)$ depends linearly on $f((n-m)\Delta t)$ for a finite set of values of $m \geq 0$ and on $g((n-m)\Delta t)$ for a finite set of values of $m > 0$. For such a filter, the z -transform of the impulse response is a rational function of z , which means that it is a ratio of two polynomials

$$\frac{p(z)}{q(z)} = a_0 + a_1 z^{-1} + a_2 z^{-2} + \dots$$

The coefficients a_0, a_1, a_2, \dots are the values of the impulse response at $t = 0$, $t = \Delta t$, $t = 2\Delta t$, \dots

The coefficients a_n tend to zero as n tends to infinity, if and only if the poles μ of $p(z)/q(z)$ satisfy $|\mu| < 1$. This can be seen in terms of the complex partial fraction expansion of the function $p(z)/q(z)$.

The location of the poles inside the unit circle has a great deal of effect on the frequency response of the filter. If there is a pole near the boundary, it will cause a local maximum in the frequency response, which is called a *resonance*. The frequency is given in terms of the argument of the position of the pole by

$$\nu = (\text{sample rate}) \times (\text{argument})/2\pi.$$

Decay time. The decay time of a filter for a particular frequency is defined to be the time it takes for the amplitude of that frequency component to reach $1/e$ of its initial value. To understand the effect of the location of a pole on the decay time, we examine the transfer function

$$H(z) = \frac{1}{z-a} = \frac{z^{-1}}{1-az^{-1}} = z^{-1} + az^{-2} + a^2z^{-3} + \dots$$

So in a period of n sample times, the amplitude is multiplied by a factor of a^n . So we want $|a|^n = 1/e$, or $n = -1/\ln|a|$. So the formula for decay time is

$$\boxed{\text{Decay time} = \frac{-\Delta t}{\ln|a|} = \frac{-1}{N \ln|a|}} \quad (7.5.2)$$

where $N = 1/\Delta t$ is the sample rate. So the decay time is inversely proportional to the logarithm of the absolute value of the location of the pole. The further the pole is inside the unit circle, the smaller the decay time, and the faster the decay. A pole near the unit circle gives rise to a slow decay.

Exercises

1. (a) Design a digital filter whose transfer function is $z^2/(z^2 + z + \frac{1}{2})$, using the symbol z^{-1} in a box to denote a delay of one sample time, as above.
- (b) Compute the frequency response of this filter. Let N denote the number of sample points per second, so that the answer should be a function of ν for $-N/2 < \nu < N/2$.
- (c) Is this filter stable?

Further reading:

R. W. Hamming, *Digital filters* [40].

Bernard Mulgrew, Peter Grant and John Thompson, *Digital signal processing* [75].

7.6. The discrete Fourier transform

The discrete Fourier transform is

$$F(k) = \frac{1}{N} \sum_{n=0}^{N-1} f(n) e^{-2\pi i n k / N}$$

$$f(n) = \sum_{k=0}^{N-1} F(k) e^{2\pi i k n / N}.$$

The fast Fourier transform is a way to compute the discrete Fourier transform using $2N \log_2 N$ operations rather than N^2 . The number of sample points N has to be a power of two for it to be this efficient, but the algorithm works for any highly composite value of N .

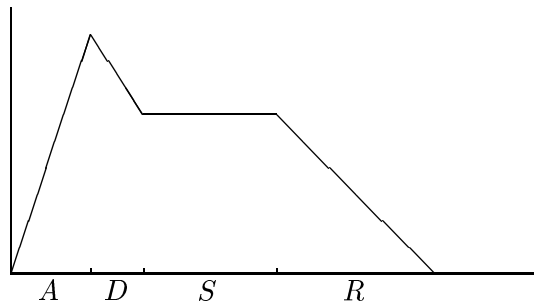
Further reading:

G. D. Bergland, *A guided tour of the fast Fourier transform*, IEEE Spectrum 6 (1969), 41–52.

James W. Cooley and John W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. of Computation 19 (1965), 297–301. This is usually regarded as the original article announcing the fast Fourier transform as a practical algorithm.

7.7. Envelopes and LFOs

Whatever method is used to synthesize sounds, attention has to be paid to envelopes, so we discuss these first. Very few sounds just consist of a spectrum, static in time. If we hear a note on almost any instrument, there is a clearly defined attack at the beginning of the sound, followed by a decay, then a sustained part in the middle, and finally a release. In any particular instrument, some of these may be missing, but the basic structure is there. Synthesis follows the same pattern. The commonly used abbreviation is ADSR envelope, for attack/decay/sustain/release envelope.



It was not really understood properly until the middle of the twentieth century, when electronic synthesis was taking its first tentative steps, that the attack portion of a note is the most vital to the human ear in identifying the

instrument. The transients at the beginning are much more different from one instrument to another than the steady part of the note.

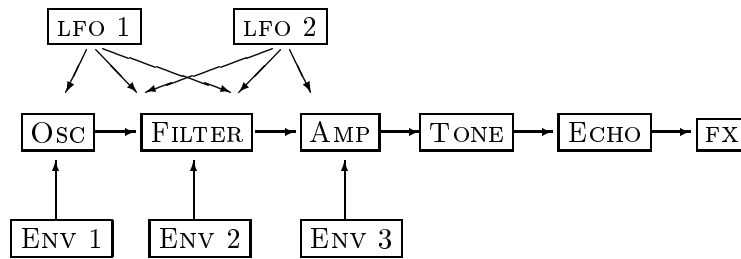
On a typical synthesizer, there are a number of envelope generators. Each one determines how the amplitude of the output of some component of the system varies with time. It is important to understand that amplitude of the final signal is not the only attribute which is assigned an envelope. For example, when a bell sounds, initially the frequency spectrum is very rich, but many of the partials die away very quickly leaving a purer sound. Mimicking this sort of behavior using FM synthesis turns out to be relatively easy, by assigning an envelope to a modulating signal, which controls timbre. We shall discuss this further when we discuss FM synthesis, but for the moment we note that aspects of timbre are often controlled with an envelope generator. When the synthesizer is controlled by a keyboard, as is often the case, it is usual to arrange that depressing a key initiates the attack, and releasing the key initiates the release portion of the envelope.

An envelope generator produces an envelope whose shape is determined by a number of programmable parameters. These parameters are usually given in terms of levels and rates. Here is an example of how an envelope might work in a typical keyboard synthesizer or other MIDI controlled environment. Level 0 is the level of the envelope at the “key on” event. Rate 1 then determines how fast the level changes, until it reaches level 1. Then it switches to rate 2 until level 2 is reached, and then rate 3 until level 3 is reached. Level 3 is then in effect until the “key off” event, when rate 4 takes effect until level 4 is reached. Finally, level 4 is the same as level 0, so that we are ready for the next “key on” event. In this example, there are two separate components to the decay phase of the envelope. Some synthesizers make do with only one, and some have even more.

Similar in concept to the envelope is the *low frequency oscillator* or LFO. This produces an output which is usually in the range 0.1–20 Hz, and whose waveform is usually something like triangle, sawtooth (up or down), sine, square or random. The LFO is used to produce repeating changes in some controllable parameter. Examples include pitch control for vibrato, and amplitude or timbre control for tremolo. The LFO can also be used to control less obvious parameters such as the cutoff and resonance of a filter, or the pulse width of a square wave (pulse width modulation, or PWM), see Exercise 6 in §2.4.

The parameters associated with an LFO are rate (or frequency), depth (or amplitude), waveform, and attack time. Attack time is used when the effect is to be introduced gradually at the beginning of the note.

Here is a block diagram for a typical analog synthesizer.

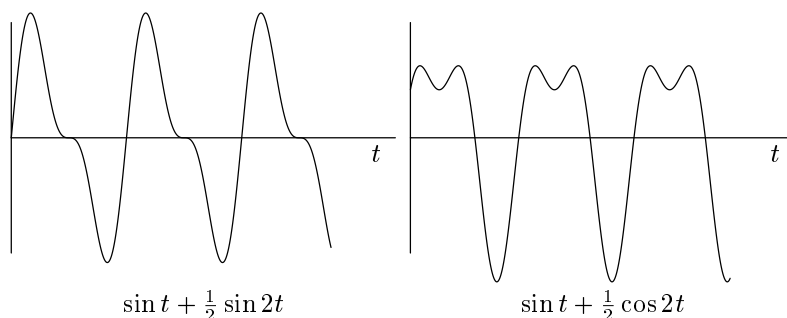


The oscillator (OSC) generates the basic waveform, which can be chosen from sine wave, square wave, triangular wave, sawtooth, noise, etc. The envelope (ENV 1) specifies how the pitch changes with time. The filter specifies the “brightness” of the sound. It can be chosen from high pass, low pass and band pass. The envelope (ENV 2) specifies how the brightness varies with time. Also, a *resonance* is specified, which determines the emphasis applied to the region at the cutoff frequency. The amplifier (AMP) specifies the volume, and the envelope (ENV 3) specifies how the volume changes with time. The tone control (TONE) adjusts the overall tone, the delay unit (ECHO) adds an echo effect, and the effects unit (FX) can be used to add reverberation, chorus, and so on. Low frequency oscillators (LFO 1 and LFO 2) are provided, which can be used to modulate the oscillator, filter or amplifier.

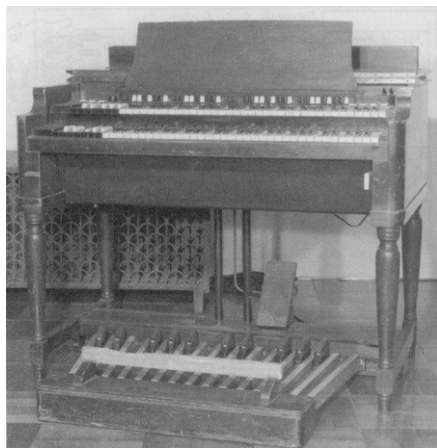
7.8. Additive Synthesis

The easiest form of synthesis to understand is additive synthesis, which is in effect the opposite of Fourier analysis of a signal. To synthesize a periodic wave, we generate its Fourier components at the correct amplitudes and mix them. This is a comparatively inefficient method of synthesis, because in order to produce a note with a large number of harmonics, a large number of sine waves will need to be mixed together. Each will be assigned a separate envelope in order to create the development of the note with time. This way, it is possible to control the development of timbre with time, as well as the amplitude. So for example, if it is desired to create a waveform whose attack phase is rich in harmonics and which then decays to a purer tone, then the components of higher frequency will have a more rapidly decaying envelope than the lower frequency components.

Phase is unimportant to the perception of steady sounds, but more important in the perception of transients. So for steady sounds, the graph representing the waveform is not very informative. For example, here are the graphs of the functions $\sin(\omega t) + \frac{1}{2} \sin(2\omega t)$ and $\sin(\omega t) + \frac{1}{2} \cos(2\omega t)$.



The only difference between these functions is that the second partial has had its phase changed by an angle of $\pi/2$, so as steady sounds, these will sound identical. With more partials, it becomes extremely hard to tell whether two waveforms represent the same steady sound. It is for this reason that the waveform is not a very useful way to represent the sound, whereas the spectrum, and its development with time, are much more useful.



Hammond B3 organ

In some ways, additive synthesis is a very old idea. A typical cathedral or church organ has a number of register stops, determining which sets of pipes are used for the production of the note. The effect of this is that depressing a single key can be made to activate a number of harmonically related pipes, typically a mixture of octaves and fifths. Early electronic instruments such as the Hammond organ operated on exactly the same principle.

More generally, additive synthesis may be used to construct sounds whose partials are not multiples of a given fundamental. This will give non-periodic waveforms which nevertheless sound like steady tones.

Exercises

1. Explain how to use additive synthesis to construct a square wave.
[Hint: Look at §2.2]
2. Explain in terms of the human ear (§1.2) why the phases of the harmonic components of a steady waveform should not have a great effect on the way the sound is perceived.

Further reading:

F. de Bernardinis, R. Roncella, R. Saletti, P. Terreni and G. Bertini, *A new VLSI implementation of additive synthesis*, Computer Music Journal 22 (3) (1998), 49–61.

7.9. Physical modeling

The idea of physical modeling is to take a physical system such as a musical instrument, and to mimic it digitally. We give one simple example to illustrate the point. We examined the wave equation for the vibrating string in §3.1, and found d'Alembert's general solution

$$y = f(x + ct) + g(x - ct).$$

Given that time is quantized with sample points at spacing Δt , it makes sense to quantize the position along the string at intervals of $\Delta x = c\Delta t$. Then at time $n\Delta t$ and position $m\Delta x$, the value of y is

$$\begin{aligned} y &= f(m\Delta x + nc\Delta t) + g(m\Delta x - nc\Delta t) \\ &= f((m+n)c\Delta t) + g((m-n)c\Delta t). \end{aligned}$$

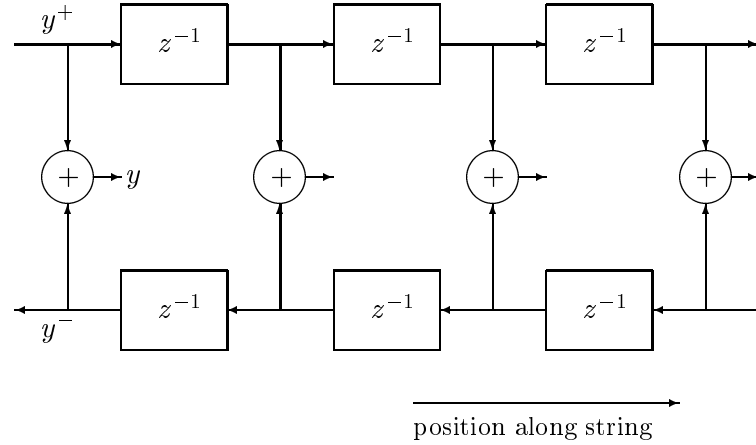
To simplify the notation, we write

$$y^-(n) = f(nc\Delta t), \quad y^+(n) = g(nc\Delta t)$$

so that y^- and y^+ represent the parts of the wave traveling left, respectively right along the string. Then at time $n\Delta t$ and position $m\Delta x$ we have

$$y = y^-(m+n) + y^+(m-n).$$

This can be represented by two delay lines moving left and right:



It is a good idea to make the string an integer number of sample points long, let us say $l = L\Delta x$. Then the boundary conditions at $x = 0$ and $x = l$ (see equations (3.1.3) and (3.1.4)) say that

$$y^-(n) = -y^+(-n)$$

and that

$$y^+(n+2L) = y^+(n).$$

This means that at the ends of the string, the signal gets negated and passed round to the other set of delays. Then the initial pluck or strike is represented by setting the values of $y^-(n)$ and $y^+(n)$ suitably at $t = 0$, for $0 \leq n < 2L$.

Thinking in terms of digital filters, the z -transform of the y^+ signal

$$Y^+(z) = y^+(0) + y^+(1)z^{-1} + y^+(2)z^{-2} + \dots$$

satisfies

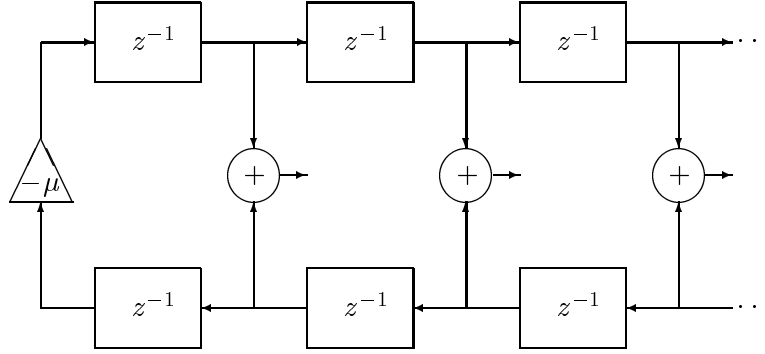
$$Y^+(z) = z^{-2L}Y^+(z) + (y^+(0) + y^+(1)z^{-1} + \dots + y^+(2L-1)z^{-(2L-1)})$$

or

$$Y^+(z) = \frac{y^+(0)z^{2L} + y^+(1)z^{2L-1} + \dots + y^+(2L-1)z}{z^{2L} - 1}.$$

The poles are equally spaced on the unit circle, so the resonant frequencies are multiples of $N/2L$, where N is the sample frequency. Since the poles are actually *on* the unit circle, the resonant frequencies never decay.

To make the string more realistic, we can put in energy loss at one end, represented by multiplication by a fixed constant factor $-\mu$ with $0 < \mu \leq 1$, instead of just negating.



The effect of this on the filter analysis is to move the poles slightly inside the unit circle:

$$Y^+(z) = \frac{y^+(0)z^{2L} + y^+(1)z^{2L-1} + \dots + y^+(2L-1)z}{z^{2L} - \mu}.$$

The absolute values of the location of the poles are all equal to $|\mu|^{\frac{1}{2L}}$. The decay time is given by equation (7.5.2) as

$$\text{Decay time} = \frac{-2L}{N \ln |\mu|}.$$

The above model is still not very sophisticated, because decay time is independent of frequency. But it is easy to modify by replacing the multiplication by μ by a more complicated digital filter. We shall see a particular example of this idea in the next section. Another easy modification is to have two or more strings cross-coupled, by adding a small multiple of the signal at the end of each into the end of the others. Adding a model of a sounding board is not so easy, but it can be done.

Further reading:

M. Laurson, C. Erkut, V. Välimäki and M. Kuuskankare, *Methods for modeling realistic playing in acoustic guitar synthesis*, Computer Music Journal 25 (3) (2001), 38–49.

Julius O. Smith III, *Physical modeling using digital waveguides*, Computer Music Journal 16 (4) (1992), 74–87.

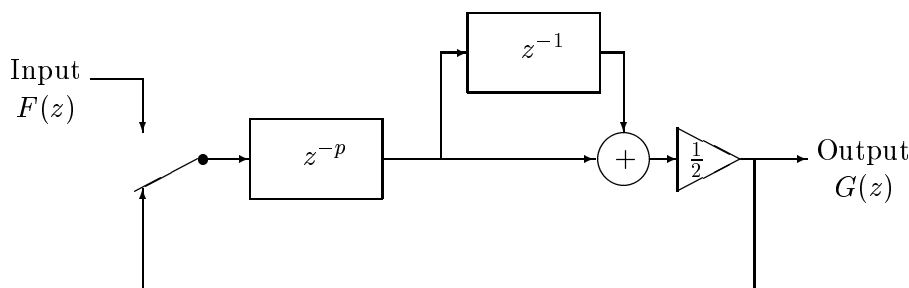
Julius O. Smith III, *Acoustic modeling*, appears as article 7 in Roads et al [94], pages 221–263.

7.10. The Karplus–Strong algorithm

The Karplus–Strong algorithm gives very good plucked strings and percussive instruments. The basic technique is a modification of the technique described in the last section, and consists of a digital delay followed by an averaging process. Denote by $g(n\Delta t)$ the value of the n th sample point in the digital output signal for the algorithm. A positive integer p is chosen to represent the delay, and the recurrence relation

$$g(n\Delta t) = \frac{1}{2}(g((n-p)\Delta t) + g((n-p-1)\Delta t))$$

is used to define the signal after the first $p+1$ sample points. The first $p+1$ values to feed into the recurrence relation are usually chosen by some random algorithm, and then the feedback loop is switched in. This is represented by an input signal $f(n\Delta t)$ which is zero outside the range $0 \leq n \leq p$.



Computationally, this algorithm is very efficient. Each sample point requires one addition operation. Halving does not need a multiplication, only a shift of the binary digits.

Let us analyse the algorithm by regarding it as a digital filter, and using the z -transform, as described in §7.5. Let $G(z)$ be the z -transform of the signal $g(n\Delta t)$, and $F(z)$ be the z -transform of the signal given for the first $p+1$ sample points, $f(n\Delta t)$. We have

$$G(z) = \frac{1}{2}(1 + z^{-1})z^{-p}(F(z) + G(z)).$$

This gives

$$G(z) = \frac{z+1}{2z^{p+1} - z - 1} F(z),$$

and so the z -transform of the impulse response is $(z+1)/(2z^{p+1} - z - 1)$. The poles are the solutions of the equation

$$2z^{p+1} - z - 1 = 0.$$

These are roughly equally spaced around the unit circle, at amplitude just less than one. The solution with smallest argument corresponds to the fundamental of the vibration, with argument roughly $2\pi/(p + \frac{1}{2})$. A more precise analysis is given in §7.11.

The effect of this is a plucked string sound with pitch determined by the formula

$$\text{pitch} = (\text{sample rate}) / (p + \tfrac{1}{2}).$$

Since p is constrained to be an integer, this restricts the possible frequencies of the resulting sound in terms of the sample rate. Changing the value of p without introducing a new initial values results in a slur, or tie between notes.

A simple modification of the algorithm gives drumlike sounds. Namely, a number b is chosen with $0 \leq b \leq 1$, and

$$g(n\Delta t) = \begin{cases} +\frac{1}{2}(g((n-p)\Delta t) + g((n-p-1)\Delta t)) & \text{with probability } b \\ -\frac{1}{2}(g((n-p)\Delta t) + g((n-p-1)\Delta t)) & \text{with probability } 1-b. \end{cases}$$

The parameter b is called the *blend factor*. Taking $b = 1$ gives the original plucked string sound. The value $b = \frac{1}{2}$ gives a drumlike sound. With $b = 0$, the period is doubled and only odd harmonics result. This gives some interesting sounds, and at high pitches this gives what Karplus and Strong describe as a *plucked bottle* sound.

Another variation described by Karplus and Strong is what they call *decay stretching*. In this version, the recurrence relation

$$g(n\Delta t) = \begin{cases} g((n-p)\Delta t) & \text{with probability } 1-\alpha \\ \frac{1}{2}(g((n-p)\Delta t) + g((n-p-1)\Delta t)) & \text{with probability } \alpha. \end{cases}$$

The *stretch factor* for this version is $1/\alpha$, and the pitch is given by

$$\text{pitch} = (\text{sample rate}) / (p + \tfrac{\alpha}{2}).$$

Setting $\alpha = 0$ gives a non-decaying periodic signal, while setting $\alpha = 1$ gives the original algorithm described above.

There are obviously a lot of variations on these algorithms, and many of them give interesting sounds.

7.11. Filter analysis for the Karplus–Strong algorithm

We saw in the last section that in order to understand the Karplus–Strong algorithm in its simplest form, we need to locate the zeros of the polynomial $2z^{p+1} - z - 1$, where p is a positive integer. In order to do this, we

begin by rewriting the equation as

$$2z^{p+\frac{1}{2}} = z^{\frac{1}{2}} + z^{-\frac{1}{2}}.$$

Since we expect z to have absolute value close to one, the imaginary part of $z^{\frac{1}{2}} + z^{-\frac{1}{2}}$ will be very small. If we ignore this imaginary part, then the n th zero of the polynomial around the unit circle will have argument equal to $2n\pi/(p + \frac{1}{2})$. So we write

$$z = (1 - \varepsilon)e^{2n\pi i/(p+\frac{1}{2})}$$

and calculate ε , ignoring terms in ε^2 and higher powers. Already from the form of this approximation, we see that the resonant frequency corresponding to the n th pole is equal to $nN/(p + \frac{1}{2})$, where N is the sample frequency. This means that the different resonant frequencies are at multiples of a fundamental frequency of $N/(p + \frac{1}{2})$.

We have

$$2z^{p+\frac{1}{2}} = 2(1 - \varepsilon)^{p+\frac{1}{2}} \approx 2 - 2(p + \frac{1}{2})\varepsilon,$$

and

$$\begin{aligned} z^{\frac{1}{2}} + z^{-\frac{1}{2}} &= (1 - \varepsilon)^{\frac{1}{2}} e^{n\pi i/(p+\frac{1}{2})} + (1 - \varepsilon)^{-\frac{1}{2}} e^{-n\pi i/(p+\frac{1}{2})} \\ &\approx (1 - \frac{1}{2}\varepsilon)(1 + \frac{1}{2}i(\frac{2n\pi}{p+\frac{1}{2}}) - \frac{1}{8}(\frac{2n\pi}{p+\frac{1}{2}})^2) \\ &\quad + (1 + \frac{1}{2}\varepsilon)(1 - \frac{1}{2}i(\frac{2n\pi}{p+\frac{1}{2}}) - \frac{1}{8}(\frac{2n\pi}{p+\frac{1}{2}})^2) \\ &\approx 2 - (\frac{n\pi}{p+\frac{1}{2}})^2 + \frac{1}{2}i\varepsilon(\frac{2n\pi}{p+\frac{1}{2}}). \end{aligned}$$

So equating the real parts, we find that the approximate value of ε is

$$\varepsilon \approx \frac{n^2\pi^2}{2(p + \frac{1}{2})^3}.$$

Using the approximation $\ln(1 - \varepsilon) \approx -\varepsilon$, equation (7.5.2) gives

$$\text{Decay time} \approx \frac{2(p + \frac{1}{2})^3}{Nn^2\pi^2}$$

where N is the sample rate. This means that the lower harmonics are decaying more slowly than the higher harmonics, in accordance with the behavior of a plucked string.

Further reading:

D. A. Jaffe and J. O. Smith III, *Extensions of the Karplus-Strong plucked string algorithm*, Computer Music Journal 7 (2) (1983), 56–69. Reprinted in Roads [92], 481–494.

M. Karjalainen, V. Välimäki and T. Tolonen, *Plucked-string models: From the Karplus-Strong algorithm to digital waveguides and beyond*, Computer Music Journal 22 (3) (1998), 17–32.

K. Karplus and A. Strong, *Digital synthesis of plucked string and drum timbres*, Computer Music Journal 7 (2) (1983), 43–55. Reprinted in Roads [92], 467–479.

F. R. Moore, *Elements of computer music* [72], page 279.

C. Roads, *The computer music tutorial* [93], page 293.

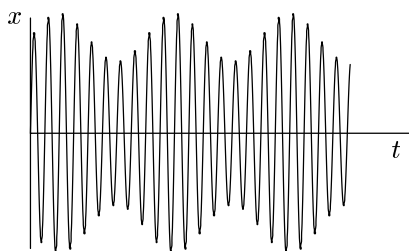
C. Sullivan, *Extending the Karplus–Strong plucked-string algorithm to synthesize electric guitar timbres with distortion and feedback*, *Computer Music Journal* 14 (3), 26–37.

7.12. Amplitude and frequency modulation

The familiar context for amplitude and frequency modulation is as a way of carrying audio signals on a radio frequency carrier (AM and FM radio). In the case of AM radio, the carrier frequency is usually in the range 500–2000 KHz, which is much greater than the frequency of the carried signal. The latter is encoded in the amplitude of the carrier. So for example a 700 KHz carrier signal modulated by a 440 Hz sine wave would be represented by the function

$$x = (A + B \sin(880\pi t)) \sin(1400000\pi t),$$

where A is an offset to allow both positive and negative values of the waveform to be decoded.



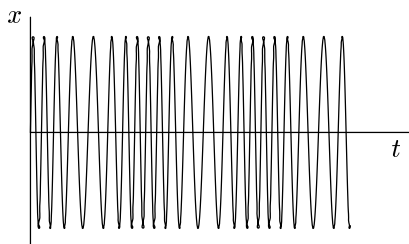
Decoding the received signal is easy. A diode is used to allow only the positive part of the wave through, and then a capacitor is used to smooth it out and remove the high frequency carrier wave. The resulting audio signal may then be amplified and put through a loudspeaker.

In the case of frequency modulation, the carrier frequency is normally around 90–120 MHz, which is even greater in comparison to the frequency of the carried signal. The latter is encoded in variations in the frequency of the carrier. So for example a 100 MHz carrier signal modulated by a 440 Hz sine wave would be represented by the function

$$x = A \sin(10^8 \cdot 2\pi t + B \sin(880\pi t)).$$

The amplitude A is associated with the carrier wave, while the amplitude B is associated with the audio wave. More generally, an audio wave represented by $x = f(t)$, carried on a carrier of frequency ν and amplitude A , is represented by

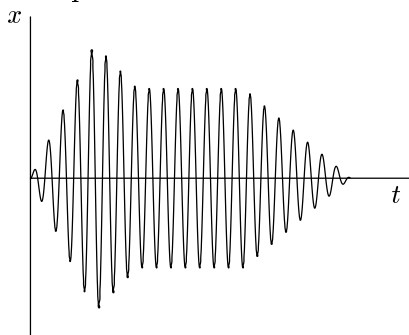
$$x = A \sin(2\pi\nu t + Bf(t)).$$



Decoding frequency modulated signals is harder than amplitude modulated signals, and will not be discussed here. But the big advantage is that it is less susceptible to noise, and so it gives cleaner radio reception.

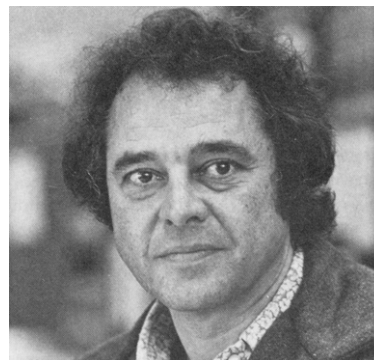
An example of the use of amplitude modulation in the theory of synthesis is ring modulation. A ring modulator takes two inputs, and the output contains only the sum and difference frequencies of the partials of the inputs. This is generally used to construct waveforms with inharmonic partials, so as to impart a metallic or bell-like timbre. The method for constructing the sum and difference frequencies is to multiply the incoming amplitudes. Equations (1.7.7), (1.7.10) and (1.7.11) explain how this has the desired result. The origin of the term “ring modulation” is that in order to deal with both positive and negative amplitudes on the inputs and get the right sign for the outputs, four diodes were connected head to tail in a ring.

Another example of amplitude modulation is the application of envelopes, as discussed in §7.7. The waveform is multiplied by the function used to describe the envelope.



A great breakthrough in synthesis was achieved in the late nineteen sixties when John Chowning developed the idea of using frequency modulation instead of additive synthesis.

The idea behind FM synthesis or *frequency modulation synthesis* is similar to FM radio, but the carrier and the signal are both in the audio range, and usually related by a small rational frequency ratio. So for

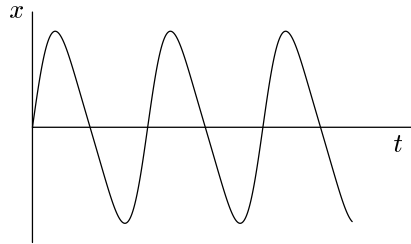


John Chowning

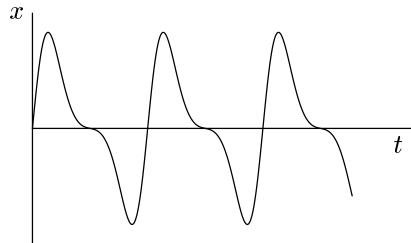
example, a 440 Hz carrier and 440 Hz modulator would be represented by the function

$$x = A \sin(880\pi t + B \sin(880\pi t)).$$

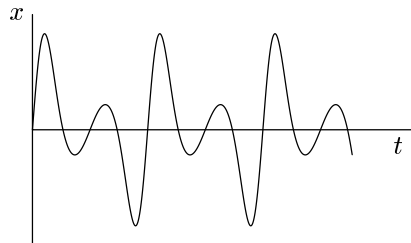
The resulting wave is still periodic with frequency 440 Hz, but has a richer harmonic spectrum than a pure sine wave. For small values of B , the wave is nearly a sine wave



whereas for larger values of B the harmonic content grows richer

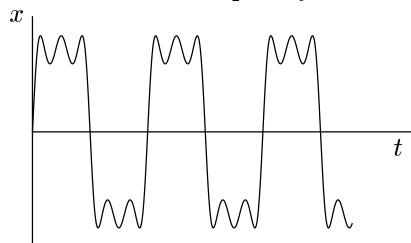


and richer.

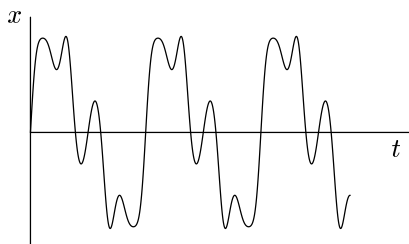


This gives a way of making an audio signal with a rich harmonic content relatively simply. If we wanted to synthesize the above wave using additive synthesis, it would be much harder.

Here are examples of frequency modulated waves in which the modulating frequency is twice the carrier frequency

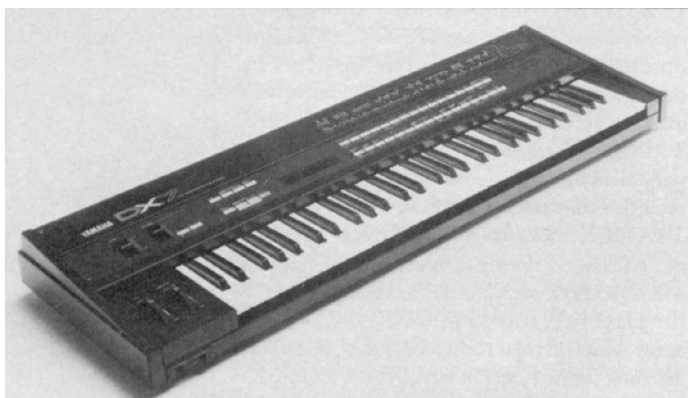


and three times the carrier frequency.



In the next section, we discuss the Fourier series for a frequency modulated signal. The Fourier coefficients are called Bessel functions, for which the groundwork was laid in §2.8. We shall see that the Bessel functions may be interpreted as giving the amplitudes of side bands in a frequency modulated signal.

7.13. The Yamaha DX7 and FM synthesis



Yamaha DX7

The Yamaha DX7, which came out in the fall of 1983,³ was the first affordable commercially available digital synthesizer. This instrument was the result of a long collaboration between John Chowning and Yamaha Corporation through the nineteen seventies. It works by FM synthesis, with six configurable “operators”. An operator produces as output a frequency modulated sine wave, whose frequency is determined by the level of a modulating input, and whose envelope is determined by another input. The power of the method comes from hooking up the output of one such operator to the modulating input of another. In this section, we shall investigate FM synthesis in detail, using the Yamaha DX7 for the details of the examples. Most of the discussion translates easily to any other FM synthesizer. In Appendix B, there are tables which apply to various models of FM synthesizers. Later on, in §§7.16–7.17, we shall also investigate FM synthesis using the CSound computer music language.

The DX7 calculates the sine function in the simplest possible way. It has a digital lookup table of values of the function. This is much faster than

³Original price US \$2000; no longer manufactured but easy to obtain second hand for around US \$250–\$450.

any conceivable formula for calculating the function, but this is at the expense of having to commit a block of memory to this task.

Let us begin by examining a frequency modulated signal of the form

$$\sin(\omega_c t + I \sin \omega_m t). \quad (7.13.1)$$

Here, $\omega_c = 2\pi f_c$ where f_c denotes the carrier frequency, $\omega_m = 2\pi f_m$ where f_m denotes the modulating frequency, and I the index of modulation.

We first discuss the relationship between the index of modulation I , the maximal frequency deviation d of the signal, and the frequency f_m of the modulating wave. For this purpose, we make a linear approximation to the modulating signal at any particular time, and use this to determine the instantaneous frequency, to the extent that this makes sense. When $\sin \omega_m t$ is at a peak or a trough, namely when its derivative with respect to t vanishes, the linear approximation is a constant function, which then acts as a phase shift in the modulated signal. So at these points, the frequency is f_c . The maximal frequency deviation occurs when $\sin \omega_m t$ is varying most rapidly. This function *increases* most rapidly when $\omega_m t = 2n\pi$ for some integer n . Since the derivative of $\sin \omega_m t$ with respect to t is $\omega_m \cos \omega_m t$, which takes the value ω_m at these values of t , the linear approximation around these values of t is $\sin \omega_m t \simeq \omega_m t - 2n\pi$. So the function (7.13.1) approximates to

$$\sin(\omega_c t + I\omega_m(t - 2\pi)) = \sin((\omega_c + I\omega_m)t - 2\pi I\omega_m).$$

So the instantaneous frequency is $f_c + If_m$. Similarly, $\sin \omega_m t$ *decreases* most rapidly when $\omega_m t = (2n + 1)\pi$ for some integer n , and a similar calculation shows that the instantaneous frequency is $f_c - If_m$. It follows that the maximal deviation in the frequency is given by

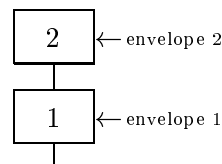
$$d = If_m. \quad (7.13.2)$$

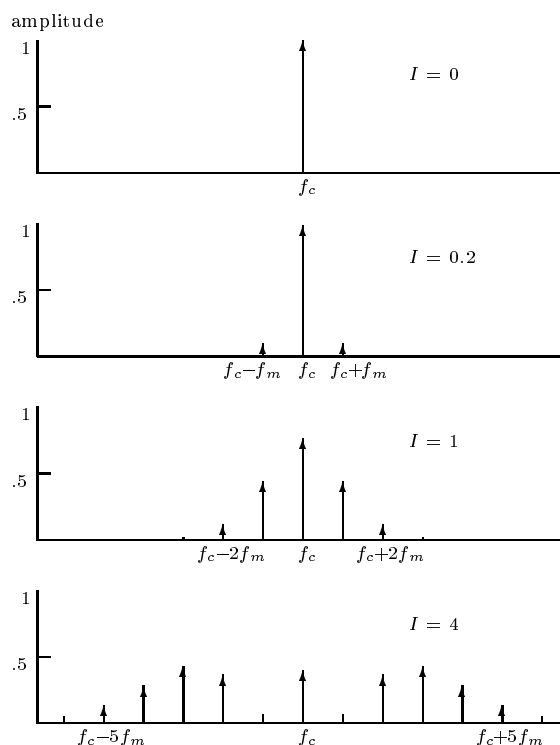
The Fourier series for functions of the form (7.13.1) were analysed in §2.8 in terms of the Bessel functions.

Putting $\phi = \omega_c t$, $z = I$ and $\theta = \omega_m t$ in equation (2.8.9), we obtain the fundamental equation for frequency modulation:

$$\sin(\omega_c t + I \sin \omega_m t) = \sum_{n=-\infty}^{\infty} J_n(I) \sin(\omega_c + n\omega_m)t. \quad (7.13.3)$$

The interpretation of this equation is that for a frequency modulated signal with carrier frequency f_c and modulating frequency f_m , the frequencies present in the modulated signal are $f_c + nf_m$. Notice that positive and negative values of n are allowed here. The component with frequency $f_c + nf_m$ is called the n th *side band* of the signal. Thus the Bessel function $J_n(I)$ is giving the amplitude of the n th side band in terms of the index of modulation. The block diagram on the DX7 for frequency modulating a sine wave in this fashion is as shown above. The box marked “1” represents the operator producing the carrier signal and the box marked “2” represents the operator producing the modulating signal.





Each operator has its own envelope, which determines how its amplitude develops with time. So envelope 1 determines how the amplitude of the final signal varies with time, but it is less obvious what envelope 2 is determining. Since the output of operator 2 is frequency modulating operator 1, the amplitude of the output can be interpreted as the index of modulation I . For small values of I , $J_0(I)$ is much larger than any other $J_n(I)$ (see the graphs in Section 2.8), and so operator 1 is producing an output which is nearly a pure sine wave, but with other frequencies present with small amplitudes. However, for larger values of I , the spectrum of the output of operator 1 grows richer in harmonics. For any particular value of I , as n gets larger, the amplitudes $J_n(I)$ eventually tend to zero. But the point is that for small values of I , this happens more quickly than for larger values of I , so the harmonic spectrum gives a purer note for small values of I and a richer sound for larger values of I . So envelope 2 is controlling the *timbre* of the output of operator 1.

Example. Suppose that we have a carrier frequency of 3ν and a modulating frequency of 2ν . Then the zeroth side band has frequency 3ν , the first 5ν , the second 7ν , and so on. But there are also side bands corresponding to negative values of n . The minus first side band has frequency ν . But there's no reason to stop there, just because the next side band has negative frequency $-\nu$. The point is that a sine wave with frequency $-\nu$ is just the same as a sine wave with frequency ν but with the amplitude negated. So really the way to think of it is that the side bands with negative frequency undergo reflection to make the corresponding positive frequency.

Notice also in this example that $3 + 2n$ is always an odd number, so only odd multiples of ν appear in the resulting frequency spectrum. In general, the frequency spectrum will depend in an interesting way on the ratio of f_m to f_c . If the ratio is a ratio of small integers, the resulting frequency spectrum will consist of multiples of a fundamental frequency. Otherwise, the spectrum is said to be *inharmonic*.

Let us calculate the spectrum in this example for various values of I . First we use a small value such as $I = 0.2$. Consulting Appendix B, we see that $J_0(I) \approx 0.9900$, $J_1(I) \approx 0.0995$, $J_2(I) \approx 0.0050$ and $J_n(I)$ is negligibly small for $n \geq 3$. Using equation (2.8.4) ($J_{-n}(I) = (-1)^n J_n(I)$), we see that $J_{-1}(I) \approx -0.0995$, $J_{-2}(I) \approx 0.0050$ and $J_{-n}(I)$ is negligibly small for $n \geq 3$. So the frequency modulated signal is approximately

$$0.0050 \sin(2\pi(-\nu)t) - 0.0995 \sin(2\pi\nu t) + 0.9900 \sin(2\pi(3\nu)t) \\ + 0.0995 \sin(2\pi(5\nu)t) + 0.0050 \sin(2\pi(7\nu)t).$$

Since $\sin(-x) = -\sin(x)$, this is

$$-0.1045 \sin(2\pi\nu t) + 0.9900 \sin(6\pi\nu t) + 0.0995 \sin(10\pi\nu t) + 0.0050 \sin(14\pi\nu t).$$

This will be perceived as a note with fundamental frequency ν , but with very strong third harmonic.

Now let us carry out the same calculation with a larger value of I , say $I = 3$. Again consulting Appendix B, we see that $J_0(I) \approx -0.2601$, $J_1(I) \approx 0.3391$, $J_2(I) \approx 0.4861$, $J_3(I) \approx 0.3091$, $J_4(I) \approx 0.1320$, $J_5(I) \approx 0.0430$, $J_6(I) \approx 0.0114$, $J_7(I) \approx 0.0025$, $J_8(I) \approx 0.0005$, and only around $n \geq 8$ is $J_n(I)$ negligibly small. So the harmonic spectrum of the resulting frequency modulated signal is much richer, and the first few terms are given by

$$-0.0430 \sin(2\pi(-7\nu)t) + 0.1320 \sin(2\pi(-5\nu)t) - 0.3091 \sin(2\pi(-3\nu)t) \\ + 0.4861 \sin(2\pi(-\nu)t) - 0.3991 \sin(2\pi\nu t) - 0.2601 \sin(2\pi(3\nu)t) \\ + 0.3391 \sin(2\pi(5\nu)t) + 0.4861 \sin(2\pi(7\nu)t)$$

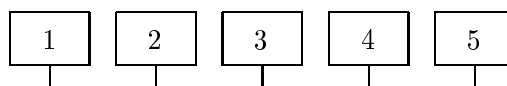
which makes

$$-0.8852 \sin(2\pi\nu t) + 0.0490 \sin(6\pi\nu t) + 0.2071 \sin(10\pi\nu t) + 0.5291 \sin(14\pi\nu t),$$

but it is clear that even higher harmonics than this are present with fairly large magnitude, up to about the seventeenth harmonic ($3 + 2 \times 7 = 17$), and then it starts tailing off. So the resulting note is very rich in harmonics. Notice also how we have conspired to choose I so that the amplitude of the third harmonic is now very small.

Suppose, for example, that operator 2 is assigned an envelope which starts at zero, peaks near the beginning, and then tails off to zero. Then the resulting frequency modulated signal will start off as a pure sine wave, fairly quickly attain a rich harmonic spectrum, and then tail off again into a fairly pure sine wave. It is easy to see that the possibilities opened up with even two operators is fairly wide.

In terms of block diagrams, additive synthesis for a waveform with five sinusoidal components is represented as follows.

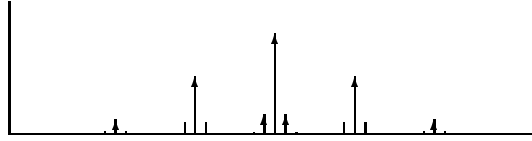


So in the above example, to synthesize the corresponding sound additively would require a large number of oscillators. The exact number would depend on where the cutoff for audibility occurs.

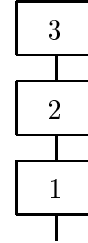
The DX7 allows a large number of different configurations or “algorithms” which mix additive and FM components. So for example if two sinusoidal waveforms of different frequencies are added together and the result used to modulate another sine wave, then the block diagram is as shown to the left. Oscillators labeled “2” and “3” are added together and used to modulate oscillator “1”. The corresponding waveform is given by

$$\begin{aligned} & \sin(\omega_1 t + I_2 \sin \omega_2 t + I_3 \sin \omega_3 t) \\ &= \sum_{n_2=-\infty}^{\infty} \sum_{n_3=-\infty}^{\infty} J_{n_2}(I_2) J_{n_3}(I_3) \sin(\omega_1 + n_2 \omega_2 + n_3 \omega_3) t. \end{aligned}$$

So the side bands have frequencies given by adding positive and negative multiples of the two modulating frequencies to the carrier frequency in all possible ways. The amplitudes of these side bands are given by multiplying the corresponding values of the Bessel functions.



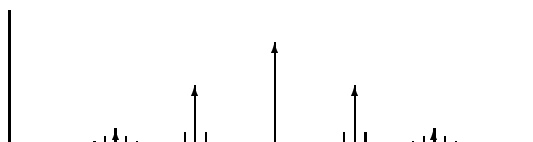
Another possible configuration is a cascade in which the modulating signal is also modulated. This should be thought of as equivalent to a larger number of added sine waves modulating a single sine wave, in an extension of the previous discussion. The block diagram for this configuration is shown to the right. The corresponding formula is obtained by feeding formula (7.13.3) into itself, giving



$$\begin{aligned} & \sin(\omega_1 t + I_2 \sin(\omega_2 t + I_3 \sin \omega_3 t)) \\ &= \sum_{n_2=-\infty}^{\infty} J_{n_2}(I_2) \sin(\omega_1 t + n_2 \omega_2 t + n_2 I_3 \sin \omega_3 t) \\ &= \sum_{n_2=-\infty}^{\infty} \sum_{n_3=-\infty}^{\infty} J_{n_2}(I_2) J_{n_3}(n_2 I_3) \sin(\omega_1 + n_2 \omega_2 + n_3 \omega_3) t. \end{aligned}$$

Here, the subscripts 2 and 3 correspond to the numbering on the oscillators in the diagram. Again, the frequencies of the side bands are given by adding positive and negative multiples of the two modulating frequencies to the carrier frequency in all possible ways. But this time, the amplitudes of

the side bands are given by the more complicated formula $J_{n_2}(I_2)J_{n_3}(n_2I_3)$. The effect of this is that the number of the side band on the second operator is used to scale the size of the index of modulation of the third operator. In particular, the original frequency has no side bands corresponding to the third operator, while the more remote side bands of the second are more heavily modulated.



Exercises

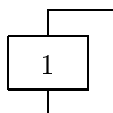
1. Find the amplitudes of the first few frequency components of the frequency modulated wave

$$y = \sin(440(2\pi t) + \frac{1}{10} \sin 660(2\pi t)).$$

Stop when the frequency components are attenuated by at least 100dB from the strongest one.

You will need to use the tables of Bessel functions in Appendix B. Also remember that power is proportional to square of amplitude, so that dividing the amplitude by 10 attenuates the signal by 20dB.

7.14. Feedback, or self-modulation



One final twist in FM synthesis is feedback, or self-modulation. This involves the output of an oscillator being wrapped back round and used to modulate the input of the same oscillator. This corresponds to the block diagram on the left, and the corresponding equation is

given by

$$f(t) = \sin(\omega_c t + I f(t)). \quad (7.14.1)$$

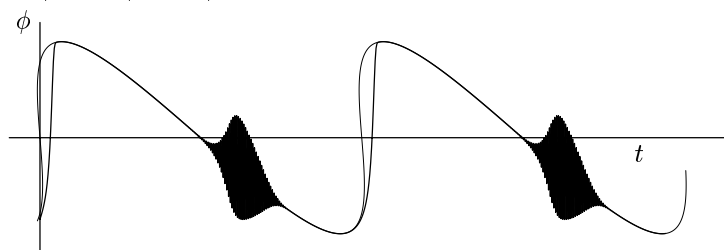
We saw in §2.11 that this equation only has a unique solution provided $|I| \leq 1$, and that then it defines a periodic function of t . The Fourier series is given in equation (2.11.4) as

$$f(t) = \sum_{n=1}^{\infty} \frac{2J_n(nI)}{nI} \sin(n\omega_c t).$$

For values of I satisfying $|I| > 1$, equation (7.14.1) no longer has a single valued continuous solution (see §2.11), but it still makes sense in the form of a recursion defining the next value of $f(t)$ in terms of the previous one,

$$f(t_n) = \sin(\omega_c t_n + I f(t_{n-1})). \quad (7.14.2)$$

Here, t_n is the n th sample time, and the sample times are usually taken to be equally spaced. The effect of this equation is not quite intuitively obvious. As might be expected, the graph of this function stays close to the solution to equation (7.14.1) when this is unique. When it is no longer unique, it continues going along the same branch of the function as long as it can, and then jumps suddenly to the one remaining branch when it no longer can. But the feature which it is easy to overlook is that there is a slightly delayed instability for small values of $f(t)$. Here is a graph of the solutions to equations (7.14.1) and (7.14.2) superimposed.

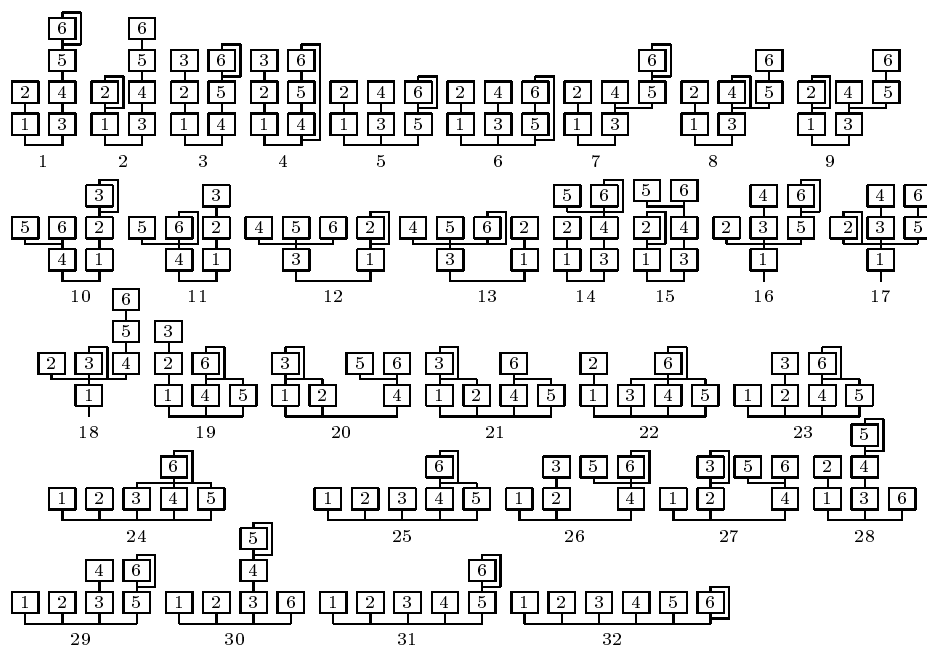


The effect of the instability is to introduce a wave packet whose frequency is roughly half the sampling frequency. Usually the sampling frequency is high enough that the effect is inaudible.

Feedback for a stack of two or more oscillators is also used. It seems hard to analyse this mathematically, and often the result is perceived as “noise”. According to Slater (reference given on page 210), as the index of modulation increases, the behavior of a stack of two FM oscillators with different frequencies, each modulating the other, exhibits the kind of bifurcation that is characteristic of chaotic dynamical systems. This subject needs to be investigated further.

In the DX7, there are a total of six oscillators. The process of designing a patch⁴ begins with a choice of one of 32 given configurations, or “algorithms” for these oscillators. Each oscillator is given an envelope whose parameters are determined by the patch, so that the amplitude of the output of each oscillator varies with time in a chosen manner. Here is a table of the 32 available algorithms.

⁴Yamaha uses the nonstandard terminology “voice” instead of the more usual “patch”.



Not all the operators have to be used in a given patch. The operators which are not used can just be switched off. Output level is an integer in the range 0–99; index of modulation is not a linear function of output level, but rather there is a complicated recipe for causing an approximately exponential relationship. A table showing this relationship for various different FM synthesizers can be found in Appendix B.

We now start discussing how to use FM synthesis to produce various recognisable kinds of sounds. In order to sound like a brass instrument such as a trumpet, it is necessary for the very beginning of the note to be an almost pure sine wave. Then the harmonic spectrum grows rapidly richer, overshooting the steady spectrum by some way, and then returning to a reasonably rich spectrum. When the note stops, the spectrum decays rapidly to a pure note and then disappears altogether. This effect may be achieved with FM synthesis by using two operators, one modulating the other. The modulating operator is given an envelope looking like the one on page 189. The carrier operator uses a very similar envelope to control the amplitude.

Next, we discuss woodwind instruments such as the flute, as well as organ pipes. At the beginning of the note, in the attack phase, higher harmonics dominate. They then decrease in amplitude until in the steady state, the fundamental dominates and the higher harmonics are not very strong. This can be achieved either by making the modulating operator have an envelope looking like the one on page 189 only upside down, or by making the carrier frequency a small integer multiple of the modulating frequency so that for small values of the index of modulation, this higher frequency dominates. In any case, the decay phase for the modulating operator should be omitted for

a more realistic sound. For some woodwind instruments such as the clarinet, it is necessary to make sure that predominantly odd harmonics are present. This can be achieved, as in the example on page 203, by setting $f_c = 3f$ and $f_m = 2f$, or some variation on this idea.

Percussive sounds have a very sharp attack and a roughly exponential decay. So an envelope looking like the graph of $x = e^{-t}$ is appropriate for the amplitude. Usually a percussive instrument will have an inharmonic spectrum, so that it is appropriate to make sure that f_c and f_m are not in a ratio which can be expressed as a ratio of small integers. We saw in Exercise 1 of §6.2 that the golden ratio is in some sense the number furthest from being able to be approximated well by ratios of small integers, so this is a good choice for producing spectra which will be perceived as inharmonic. Alternatively, the analysis carried out in §3.5 can be used to try to emulate the frequency spectrum of an actual drum.

Section §7.15 and the ones following it consist of an introduction to the public domain computer music language CSound. One of our goals will be to describe explicit implementations of two operator FM synthesis realizing the above descriptions.

Further reading on FM synthesis:

J. Bate, *The effect of modulator phase on timbres in FM synthesis*, Computer Music Journal 14 (3) (1990), 38–45.

John Chowning, *The synthesis of complex audio spectra by means of frequency modulation*, J. Audio Engineering Society 21 (7) (1973), 526–534. Reprinted as chapter 1 of Roads and Strawn [95], pages 6–29.

John Chowning, *Frequency modulation synthesis of the singing voice*, appeared in Mathews and Pierce [66], chapter 6, pages 57–63.

Chowning and Bristow [13].

A. Horner, *Double-modulator FM matching of instrument tones*, Computer Music Journal 20 (2) (1996), 57–71.

A. Horner, *A comparison of wavetable and FM parameter spaces*, Computer Music Journal 21 (4) (1997), 55–85.

A. Horner, J. Beauchamp and L. Haken, *FM matching synthesis with genetic algorithms*, Computer Music Journal 17 (4) (1993), 17–29.

M. LeBrun, *A derivation of the spectrum of FM with a complex modulating wave*, Computer Music Journal 1 (4) (1977), 51–52. Reprinted as chapter 5 of Roads and Strawn [95], pages 65–67.

F. R. Moore, *Elements of computer music* [72], pages 316–332.

D. Morrill, *Trumpet algorithms for computer composition*, Computer Music Journal 1 (1) (1977), 46–52. Reprinted as chapter 2 of Roads and Strawn [95], pages 30–44.

C. Roads, *The computer music tutorial* [93], pages 224–250.

S. Saunders, *Improved FM audio synthesis methods for real-time digital music generation*, Computer Music Journal 1 (1) (1977), 53–55. Reprinted as chapter 3 of Roads and Strawn [95], pages 45–53.

W. G. Schottstaedt, *The simulation of natural instrument tones using frequency modulation with a complex modulating wave*, Computer Music Journal 1 (4) (1977), 46–50. Reprinted as chapter 4 of Roads and Strawn [95], pages 54–64.

D. Slater, *Chaotic sound synthesis*, Computer Music Journal 22 (2) (1998), 12–19.

B. Truax, *Organizational techniques for c : m ratios in frequency modulation*, Computer Music Journal 1 (4) (1977), 39–45. Reprinted as chapter 6 of Roads and Strawn [95], pages 68–82.

7.15. CSound

CSound is a public domain synthesis program written by Barry Vercoe at the Media Lab in MIT in the C programming language. It has been compiled for various platform, and both source code and executables are freely available.

The program takes as input two files, called the *orchestra* file and the *score* file. The orchestra file contains the instrument definitions, or how to synthesize the desired sounds. It makes use of almost every known method of synthesis, including FM synthesis, the Karplus–Strong algorithm, phase vocoder, pitch envelopes, granular synthesis and so on, to define the instruments. The score file uses a language similar in conception to MIDI but different in execution, in order to describe the information for playing the instruments, such as amplitude, frequency, note durations and start times. The utility MIDI2CS mentioned in §7.24 provides a flexible way of turning MIDI files into CSound score files. The final output of the CSound program is a file in some chosen sound format, for example a WAV file or an AIFF file, which can be played through a computer sound card, downloaded into a synthesizer with sampling features, or written onto a CD.

We limit ourselves to a brief description of some of the main features of CSound, with the objective of getting as far as describing how to realise FM synthesis. The examples are adapted from the CSound manual.

Getting it. The source code and executables for a number of platforms can be obtained from

`ftp://ftp.maths.bath.ac.uk/pub/dream/`

(mirrored at `ftp://ftp.musique.umontreal.ca/pub/mirrors/dream/`)

as can the manual and some example files. For a minimalist installation on an MS-DOS (or Windows) based machine, get the file `csound_new412.zip` from the subdirectory `newest/` at the above site (or a later version if available; the above version was released on March 6, 2001). Unzip it⁵ into a directory

⁵To unzip a file under *Windows*, get hold of *Winzip* from `winzip.com`. This is shareware, but can be used indefinitely without registration. If you prefer to use a free utility, get hold of Info-ZIP's free MS-DOS based program `unzip.exe` (138 kB) from

of your choice, and make sure the directory is in your path by editing the `autoexec.bat` file if necessary. If you are *really* short of space, delete everything except the files `csound.txt`, `csound.exe` and `dos4gw.exe` (total around 1 megabyte), and you will still be able to run all the examples described here. Make a new subdirectory for your orchestra and score files, and run CSound from that subdirectory. Instructions for running CSound can be found on page 213.

If you are running in an MS-DOS window under Windows 95/98/ME or NT/2000, the above still works, but the file `csound_con412.zip` contains a smaller and more efficient version of just the `csound.exe` file and the `csound.txt` file; you won't need `dos4gw.exe`. The disadvantage is that the displays are in ascii instead of full screen graphics. There is also a Windows front end in `csound_win412.zip`. This is capable of realtime sound output and realtime MIDI handling, which the MS-DOS version is not, but apart from that, it is quite primitive. For example, the program needs to restart every time it is run, and cannot just replay the output.

The most up to date version of the manual is version 4.10, which can be found at

<http://lakewoodsound.com/csound/download.htm>

This version does not seem to have made it onto the Bath ftp site mentioned above.

The orchestra file. This file has two main parts, namely the *header* section, which defines the sample rate, control rate, and number of output channels, and the *instrument* section which gives the instrument definitions. Each instrument is given its own number, which behaves like a patch number on a synthesizer.

The header section has the following format (everything after a semi-colon is a comment):

```
sr = 44100 ; sample rate in samples per second
kr = 4410 ; control rate in control signals per second
ksmps = 10 ; ksmps = sr/kr must be an integer, samples per control period
nchnls = 1 ; number of channels
```

(7.15.1)

An instrument definition consists of a collection of statements which generate or modify a digital signal. For example the statements

```
instr 1
  asig oscil 10000, 440, 1
  out asig
endin
```

(7.15.2)

generate a 440 Hz wave with amplitude 10000, and send it to an output. The two lines of code representing the waveform generator are encased in a pair

<http://www.cdrom.com/pub/infozip/UnZip.html>

—there is also a copy on my machine:

<ftp://byrd.math.uga.edu/pub/win/compression/unzip.exe>

of statements which define this to be an instrument. For WAV file output, the possible range of amplitudes before clipping takes effect is from -32768 to $+32767$, for a total of 2^{15} possible values. The final argument 1 is a waveform number. This determines which waveform is taken from an *f* statement in the score file (see below). In our first example below, it will be a sine wave. The label *asig* is allowed to be any string beginning with *a* (for “audio signal”). So for example *a1* would have worked just as well. The *oscil* statement is one of CSound’s many signal generators, and its effect is to output periodic signals made by repeating the values passed to it, appropriately scaled in amplitude and frequency. There is also another version called *oscili*, with the same syntax, which performs linear interpolation rather than truncation to find values at points between the sample points. This is slower by approximately a factor of two, but in some situations it can lead to better sounding output. In general, it seems to be better to use *oscil* for sound waves and *oscili* for envelopes (see page 215).

As it stands, the instrument (7.15.2) isn’t very useful, because it can only play one pitch. To pass a pitch, or other attributes, as parameters from the score file to the orchestra file, an instrument uses variables named *p1*, *p2*, *p3*, and so on. The first three have fixed meanings, and then *p4*, *p5*, ... can be given other meanings. If we replace 440 by *p5*,

```
asig oscil 10000, p5, 1
```

then the parameter *p5* will determine pitch.

The score file. Each line begins with a letter called an *opcode*, which determines how the line is to be interpreted. The rest of the line consists of numerical parameter fields *p1*, *p2*, *p3*, and so on. The possible opcodes are:

- f* (function table generator),
- i* (instrument statement; i.e., play a note),
- t* (tempo),
- a* (advance score time; i.e., skip parts),
- b* (offset score time),
- v* (local textual time variation),
- s* (section statement),
- r* (repeat sections),
- m* and *n* (repeat named sections),
- e* (end of score),
- c* (comment; semicolon is preferred).

If a line of the score file does not begin with an opcode, it is treated as a continuation line.

Each parameter field consists of a floating point number with optional sign and optional decimal point. Expressions are not permitted.

An `f` statement calls a subroutine to generate a set of numerical values describing a function. The set of values is intended for passing to the orchestra file for use by an instrument definition. The available subroutines are called `GEN01`, `GEN02`, Each takes some number of numerical arguments. The parameter fields of an `f` statement are as follows.

p1 Waveform number

p2 When to begin the table, in beats

p3 Size of table; a power of 2, or one more, maximum 2^{24}

p4 Number of `GEN` subroutine

p5, p6, ... Parameters for `GEN` subroutine

Beats are measured in seconds, unless there is an explicit `t` (tempo) statement; in our examples, `t` statements are omitted for simplicity.

So for example, the statement

```
f1 0 8192 10 1
```

uses `GEN10` to produce a sine wave, starting “now”, of size 8192, and assigns it to waveform 1. The subroutine `GEN10` produces waveforms made up of weighted sums of sine waves, whose frequencies are integer multiples of the fundamental. So for example

```
f2 0 8192 10 1 0 0.5 0 0.333
```

produces the sum of the first five terms in the Fourier series for a square wave, and assigns it to waveform 2.

An `i` statement activates an instrument. This is the kind of statement used to “play a note”. Its parameter fields are as follows.

p1 Instrument number

p2 Starting time in beats

p3 Duration in beats

p4, p5, ... Parameters used by the instrument

An `e` statement denotes the end of a score. It consists of an `e` on a line on its own. Every score file must end in this way.

For example, if instrument 1 is given by (7.15.2) then the score file

```
f1 0 8192 10 1 ; use GEN10 to create a sine wave
```

```
i1 0 4 ; play instr 1 from time 0 for 4 secs
```

```
e
```

(7.15.3)

will play a 440Hz tone for 4 seconds.

Running CSound. The program `CSound` was designed as a command line program, and although various front ends have been designed for it, the command line remains the most convenient method. Having installed `CSound` according to the instructions that accompany the program, the procedure is to create an orchestra file called `<filename>.orc` and a score file called

<filename>.sco using your favorite (ascii) text processor.⁶ The basic syntax for running CSound is

```
csound <flags> <filename>.orc <filename>.sco
```

For example, if your files are called ditty.orc and ditty.sco, and you want a WAV file output, then use the -W flag (this is case sensitive).

```
csound -W ditty.orc ditty.sco
```

This will produce as output a file called test.wav. If you want some other name, it must be specified with the -o flag.

```
csound -W -o ditty.wav ditty.orc ditty.sco (7.15.4)
```

If you want to suppress the graphical displays of the waveforms, which csound gives by default, this is achieved with the -d flag.

We are now ready to run our first example. Make two text files, one called ditty.orc containing the statements (7.15.1) followed by (7.15.2), and one called ditty.sco containing the statements (7.15.3). If the program is properly installed, then typing the command (7.15.4) at the command line should produce a file ditty.wav. Playing this file through a sound card or other audio device should then sound a pure sine wave at 440Hz for 4 seconds.

Warning. Both the orchestra and the score file are case sensitive. If you are having problems running CSound on the above orchestra and score files, check that you have typed everything in lower case.

There is also an annoying feature, which is that if the last line of text in the input file does not have a carriage return, then a wave file will be generated, but it will be unreadable. So it is best to leave a blank line at the end of each file.

Our “ditty” wasn’t really very interesting, so let’s modify it a bit. In order to be able to vary the amplitude and pitch, let us modify the instrument (7.15.2) to read

```
instr 1
  asig oscil p4, p5, 1 ; p4 = amplitude, p5 = frequency
  out asig
endin (7.15.5)
```

Now we can play the first ten notes of the harmonic series (see page 99) using the following score file.

⁶Word processors such as Word Perfect or Word by default save files with special formatting characters embedded in them. CSound will choke on these characters. In MS-DOS, the command

```
edit <filename>
```

will invoke a simple ascii text processor whose output will not choke CSound in this way. If you are running in an MS-DOS box inside Windows, the command

```
notepad <filename>
```

will start up the ascii text processor called notepad in a separate window, which is more convenient for switching between the editor and running CSound.

```

f1 0 8192 10 1 ; sine wave
i1 0.0 0.4 32000 261.6 ; fundamental (C, to nearest tenth of a Hz)
i1 0.5 0.4 24000 523.2 ; second harmonic, octave
i1 1.0 0.4 16000 784.8 ; third harmonic, perfect fifth
i1 1.5 0.4 12000 1046.4 ; fourth harmonic, octave
i1 2.0 0.4 8000 1308.0 ; fifth harmonic, just major third
i1 2.5 0.4 6000 1569.6 ; sixth harmonic, perfect fifth
i1 3.0 0.4 4000 1831.2 ; seventh harmonic, listen carefully to this one
i1 3.5 0.4 3000 2092.8 ; eighth harmonic, octave
i1 4.0 0.4 2000 2354.4 ; ninth harmonic, just major second
i1 4.5 0.4 1500 2616.0 ; tenth harmonic, just major third
e

```

(7.15.6)

This file plays a series of notes at half second intervals, each lasting 0.4 seconds, at successive integer multiples of 220Hz, and at steadily decreasing amplitudes. Make an orchestra file from (7.15.1) and (7.15.5), and a score file from (7.15.6), run CSound as before, and listen to the results.

Data rates. Recall from (7.15.1) that the header of the orchestra file defines two rates, namely the *sample rate* and the *control rate*. There are three different kinds of variables in CSound, which are distinguished by how often they get updated. a-rate variables, or audio rate variables, are updated at the sample rate, while the k-rate variables, or control rate variables, are updated at the control rate. Audio signals should be taken to be a-rate, while an envelope, for example, is usually assigned to a k-rate variable. It is possible to make use of audio rate signals for control, but this will increase the computational load. A third kind of variable, the i-rate variable, is updated just once when a note is played. These variables are used primarily for setting values to be used by the instrument. The first letter of the variable name (a, k or i) determines which kind of variable it is.

The variables discussed so far are all *local* variables. This means that they only have meaning within the given instrument. The same variable can be reused with a different meaning in a different instrument. There are also global versions of variables of each of these rates. These have names beginning with ga, gk and gi. Assignment of a global variable is done in the *header* section of the orchestra file.

Envelopes. One way to apply an envelope is to make an oscillator whose frequency is $1/p3$, the reciprocal of the duration, so that exactly one copy of the waveform is used each time the note is played. It is better to use oscili rather than oscil for envelopes, because many sample points of the envelope will be used in the course of the one period. So for example

```
kenv oscili p4, 1/p3, 2
```

uses waveform 2 to make an envelope. The first letter k of the variable name kenv means that this is a control rate variable. It would work just as well to

make it an audio rate variable by using a name like `aenv`, but it would demand greater computation time, and result in no audible improvement.

The subroutine `GEN07`, which performs linear interpolation, is ideal for an envelope made from straight lines. The arguments `p4`, `p5`, ... of this subroutine alternate between numbers of points and values. So for example, the statement

```
f2 0 513 7 0 80 1 50 0.7 213 0.7 170 0 ; ADSR envelope
```

in the score file produces an envelope resembling the one on page 189 with ADSR sections of length 80, 50, 213, 170 samples, with heights varying linearly

$$0 \rightarrow 1 \rightarrow 0.7 \rightarrow 0.7 \rightarrow 0,$$

and assigns it to waveform 2. The numbers of sample points in the sections should always add up to the total length `p3`.

Recall that the total number of sample points must be either a power of two, or one more than a power of two. It is usual to use a power of two for repeating waveforms. For waveforms that will be used only once, such as an envelope, we use one more than a power of two so that the number of intervals between sample points is a power of two.

To apply the envelope to the instrument (7.15.5), we replace `p4` with `kenv` to make

```
instr 1
  kenv oscili p4, 1/p3, 2 ; envelope from waveform 2
                        ; p4 = amplitude
  asig oscil kenv, p5, 1 ; p5 = frequency
  out asig
endin
```

It would also be possible to replace the waveform number 2 in the definition of `kenv` with another variable, say `p6`, to give a more general purpose shaped sine wave.

Exercises

1. Make orchestra and score files to play a major scale using a sine wave with an ADSR envelope. Check that your files work by running `CSound` on them and listening to the result.

7.16. FM synthesis using `CSound`

Here is the most basic two operator FM instrument:

```
instr 1
  amod oscil p6 * p7, p6, 1 ; modulating wave, p6 = modulating frequency
                        ; p7 = index of modulation
  kenv oscili p4, 1/p3, 2 ; envelope, p4 = amplitude
  asig oscil kenv, p5 + amod, 1 ; p5 = carrier frequency
```

```

    out asig
endin

```

(7.16.1)

The parameter `p7` here represents the index of modulation; the reason why it is multiplied by `p6` in the definition of the modulating wave `amod` is that the modulation is taking place directly on the frequency rather than on the phase. According to equation (7.13.2), this means that the index of modulation must be multiplied by the frequency of the modulating wave before being applied. The argument `p5 + amod` in the definition of `asig` is the carrier frequency `p5` plus the modulating wave `amod`. The wave has been given an envelope `kenv`.

For a score file to illustrate this simple instrument, we introduce some useful abbreviations available for repetitive scores. First, note that the `i` statements in a score do not have to be in order of time of execution. The score is sorted with respect to time before it is played. The *carry* feature works as follows. Within a group of consecutive `i` statements in the score file (not necessarily consecutive in time) whose `p1` parameters are equal, empty parameter fields take their value from the previous statement. An empty parameter field is denoted by a dot, with spaces between consecutive fields. Intervening comments or blank lines do not affect the carry feature, but other non-`i` statements turn it off.

For the second parameter field `p2` only, the symbol `+` gives the value of `p2 + p3` from the previous `i` statement. This begins a note at the time the last one ended. The symbol `+` may also be carried using the carry feature described above. Liberal use of the carry and `+` features greatly simplify typing in and subsequent alteration of a score. Here, then, is a score illustrating simple FM synthesis with $f_m = f_c$, with gradually increasing index of modulation.

```

f1 0 8192 10 1 ; sine wave
f2 0 513 7 0 80 1 50 0.7 213 0.7 170 0 ; ADSR
i1 1 1 10000 200 200 0 ; index = 0 (pure sine wave)
i1 + . . . . 1 ; index = 1
i1 + . . . . 2 ; index = 2
i1 + . . . . 3 ; index = 3
i1 + . . . . 4 ; index = 4
i1 + . . . . 5 ; index = 5
e

```

Sections. An `s` statement consisting of a single `s` on a line by itself ends a section and starts a new one. Sorting of `i` and `f` statements (as well as `a`, which we haven't discussed) is done by section, and the timing starts again at the beginning for each section. Inactive instruments and data spaces are purged at the end of a section, and this frees up computer memory.

The following score, using the same instrument (7.16.1), has three sections with different ratios $f_m : f_c$ and with gradually increasing index of modulation.

```

f1 0 8192 10 1 ; sine wave
i1 1 1 10000 200 200 0 ; index = 0, fm:fc = 1:1
i1 + . . . . 1 ; index = 1
i1 + . . . . 2 ; index = 2
i1 + . . . . 3 ; index = 3
i1 + . . . . 4 ; index = 4
i1 + . . . . 5 ; index = 5
s
i1 1 1 10000 200 400 0 ; index = 0, fm:fc = 1:2
i1 + . . . . 1 ; index = 1
i1 + . . . . 2 ; index = 2
i1 + . . . . 3 ; index = 3
i1 + . . . . 4 ; index = 4
i1 + . . . . 5 ; index = 5
s
i1 1 1 10000 400 200 0 ; index = 0, fm:fc = 2:1
i1 + . . . . 1 ; index = 1
i1 + . . . . 2 ; index = 2
i1 + . . . . 3 ; index = 3
i1 + . . . . 4 ; index = 4
i1 + . . . . 5 ; index = 5
e

```

Pitch classes. CSound has a function `cspch` for converting octave and pitch class notation in twelve tone equal temperament into frequencies in Hertz. This function may be used in an instrument definition, so that the instrument can be fed notes from the score file in this notation.

The octave and pitch class notation consists of a whole number, representing octave, followed by a decimal point and then two digits representing pitch class. The pitch classes are taken to begin with .00 for C and end with .11 for B, although higher values will just overlap into the next octave. The octave numbering is such that 8.00 represents middle C, 9.00 represents the octave above middle C, and so on. So for example the A above middle C can be represented as 8.09, or as 7.21, so that

$$\text{cspch}(8.09) = \text{cspch}(7.21) = 440.$$

Notes between two pitches on the twelve tone equal tempered scale can be represented by using further digits. So if four digits are used after the decimal point then the value is interpreted in cents. For example, if 8.00 represents middle C, then a just major third above this would be 8.0386, taken to the nearest cent.

7.17. Simple FM instruments

The bell. In this section, we use CSound and FM synthesis to imitate some instruments. We begin with the sound of a bell.⁷ For a typical bell sound, we need an inharmonic spectrum. We can obtain this by using simple two operator FM synthesis where f_c and f_m have a ratio which cannot be expressed as a simple ratio of two integers. The golden ratio is particularly good in this regard, for reasons explained in Exercise 1 of §6.2, so we take f_m to be 1.618 times f_c .

The bell sound is most easily made using envelopes representing exponential decay for both amplitude and timbre. The subroutine GEN05 is designed for this. It performs *exponential interpolation*, which is based on the fact that between any two points (x_1, y_1) and (x_2, y_2) in the plane, with y_1 and y_2 positive, there is a unique exponential curve. It is given by

$$y = y_1^{\frac{x-x_2}{x_1-x_2}} y_2^{\frac{x-x_1}{x_2-x_1}}.$$

If y_1 and y_2 are both negative, replace them by the corresponding positive number in the above formula and then negate the final answer.

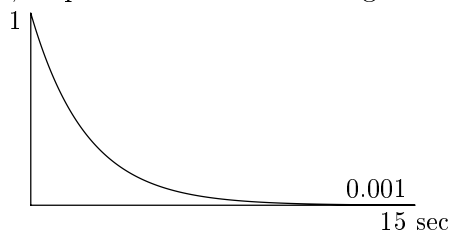
The fields for the GEN05 subroutine are the same as for GEN07 (see page 216), except that the values p5, p7, . . . must all have the same sign. Referring back to the discussion of envelopes on page 215, we see that if we put

```
f2 0 513 5 1 513 .0001
```

in the score file and

```
kenv oscili p4, 1/p3, 2
```

in the instrument definition, we will create an envelope with name kenv which decays exponentially from 1 to 0.0001. For a bell sound, we use an envelope like this for amplitude⁸ and an envelope decaying exponentially from 1 to 0.001 scaled up by a factor of 10 for index of modulation. We also use a very long decay time, to permit the sound to linger.



This explains the following instrument definition. Pitches have been converted from octave and pitch class notation as explained above. In spite of the fact that lower frequency components are present, the perceived pitch of the note produced is equal to the carrier frequency.

⁷The examples in this section are adapted from an article of Chowning, reprinted as chapter 1 of [95].

⁸Don't forget that amplitude is perceived logarithmically, so this sounds like a linear decrease, and indeed is a linear decrease when measured in decibels.

```

instr 1 ; FM bell
ifc = cspch(p5) ; carrier frequency
ifm = cspch(p5) * 1.618 ; modulating frequency
    kenv oscili p4, 1/p3, 2 ; envelope, p3 = duration, exp decay f2
                        ; p4 = amplitude
    ktmb oscili ifm * 10, 1/p3, 3 ; timbre envelope, max = 10, exp decay f3
    amod oscil ktmb, ifm, 1 ; modulator
    asig oscil kenv, ifc + amod, 1 ; carrier
    out asig
endin

```

Here is the score file to play notes E, C, D, G for a chime, using this instrument.

```

f1 0 8192 10 1
f2 0 513 5 1 513 .0001
f3 0 513 5 1 513 .001
i1 1 15 8000 8.04 ; 15 seconds at amplitude 8000 at middle C
i1 2.5 . . 8.00
i1 4 . . 8.02
i1 5.5 . . 7.07
e

```

A general purpose instrument. It is not hard to modify the instrument described above to make a general purpose two operator FM synthesis instrument.

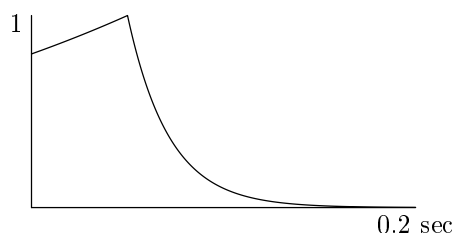
```

instr 1 ; Two operator FM instrument
ifc = cspch(p5) * p6 ; p6 = carrier frequency multiplier
ifm = cspch(p5) * p7 ; p7 = modulator frequency multiplier
    kenv oscili p4, 1/p3, p8 ; p3 = duration
    ; p4 = amplitude, p8 = carrier envelope
    ktmb oscili ifm * p10, 1/p3, p9 ; p9 = modulator envelope
    ; p10 = maximum index of modulation
    amod oscil ktmb, ifm, 1 ; modulator
    asig oscil kenv, ifc + amod, 1 ; carrier
    out asig
endin

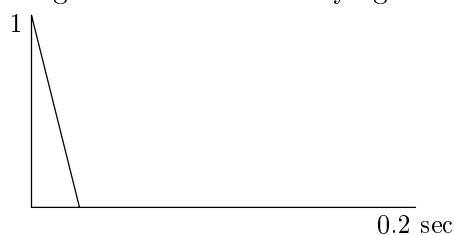
```

The rest of the examples in this section are described in terms of this setup.

The wood drum. To make a reasonably convincing wood drum, the amplitude envelope is made up of two exponential curves using GEN05,



while the envelope for the index of modulation is made up of two straight line segments, decreasing to zero and then staying there, using GEN07.



It turns out to be better to use a modulating frequency lower than the carrier frequency. So we use the reciprocal of the golden ratio, which is 0.618. We also use a large index of modulation, with a peak of 25, and a note duration of 0.2 seconds. This instrument works best in the octave going down from middle C. So the function table generators take the form

```
f1 0 8192 10 1 ; sine wave
f2 0 513 5 .8 128 1 385 .0001 ; amplitude envelope for wood drum
f3 0 513 7 1 64 0 449 0 ; modulating index envelope for wood drum
```

and the instrument statements take the form

```
i1 <time> 0.2 <amplitude> <pitch> 1.0 0.618 2 3 25
```

Brass. For a brass instrument, we use a harmonic spectrum containing all multiples of the fundamental. This is easily achieved by taking $f_c = f_m$. The relative amplitude of higher harmonics is greater when the overall amplitude is greater, so the timbre and amplitude are given the same envelope. This is chosen to look like the ADSR curve on page 189, to represent an overshoot in intensity during the attack. The index of modulation does not want to be as great as in the above examples. A maximum index of 5 gives a reasonable sound. The envelope given below is suitable for a note of duration around 0.6 seconds. It would need to be modified slightly for other durations.

```
f1 0 8192 10 1 ; sine wave
f2 0 513 7 0 85 1 86 0.75 256 0.7 86 0 ; envelope for brass
```

A typical note would then be represented by a statement of the form



```
i1 <time> 0.6 <amplitude> <pitch> 1.0 1.0 2 2 5
```

To improve the sound slightly on the brass tone presented here, we may wish to add a small deviation to the modulating frequency, so that there is a slight tremolo effect in the sound. If we replace the definition of the modulating

frequency by the statement

```
ifm = cpspch(p5) * p7 + 0.5
```

then this will have the required effect.

Woodwind. For woodwind instruments, higher harmonics are present during the attack, and then the low frequencies enter. So we want the carrier frequency to be a multiple of the modulating frequency, and use an envelope of the form  for the carrier and  for the modulator. So the function table generators take the form

```
f1 0 8192 10 1 ; sine wave
```

```
f2 0 513 7 0 50 1 443 1 20 0 ; amplitude envelope for woodwind
```

```
f3 0 513 7 0 50 1 463 1 ; modulating index envelope for woodwind
```

For a clarinet, where odd harmonics dominate, we take $f_c = 3f_m$ and a maximum index of 2. A bassoon sound is produced by giving the odd harmonics a more irregular distribution. This can be achieved by taking $f_c = 5f_m$ and a maximum index of 1.5.

7.18. Further techniques in CSound

The CSound language is vast. In this section, we cover just a few of the features which we have not touched on in the previous sections. For more information, see the CSound manual.

Tempo. The default tempo is 60 beats per minute, or one beat per second. To change this, a tempo statement is put in the score file. An example of the simplest form of tempo statement is

```
t 0 80
```

which sets the tempo to 80 beats per minute. The first argument (p1) of the tempo statement must always be zero. A tempo statement with more arguments causes accelerandos and ritardandos. The arguments are alternately times in beats (p1 = 0, p3, p5 ...) and tempi in beats per minute (p2, p4, p6, ...). The tempi between the specified times are calculate by making the durations of beats vary linearly. So for example the tempo statement

```
t 0 100 20 120 40 120
```

causes the initial tempo to be 100 beats per minute. By the twentieth beat, the tempo is 120 beats per minute. But the number of beats per minute is not linear between these values. Rather, the durations decrease linearly from 0.6 seconds to 0.5 seconds over the first twenty beats. The tempo is then constant from beat 20 until beat 40. By default, the tempo remains constant after the last beat where it is specified, so in this example the last two parameters are superfluous.

The tempo statement is only valid within the score section (cf. page 217) in which it is placed, and only one tempo statement may be used in each section. Its location within the section is irrelevant.

Stereo and Panning. For stereo output, we want to set `nchnls = 2` in the header of the orchestra file (7.15.1). In the instrument definition, instead of using `out`, we use `outs` with two arguments. So for example to do a simple pan from left to right, we might want the following lines in the instrument definition.

```
kpanleft lineseg 0, p3, 1
kpanright = 1 - kpanleft
outs asig * kpanleft, asig * kpanright
```

The problem with this method of panning is that the total sound energy is proportional to the square of amplitude, summed over the two channels. So in the middle of the pan, the total energy is only $1/\sqrt{2}$ times the total energy on the left or right. So it sounds like there's a hole in the middle. The easiest way to correct this is to take the square root of the straight line produced by the signal generator `lineseg`. So for example we could have the following lines.

```
kpan lineseg 0, p3, 1
kpanleft = sqrt(kpan)
kpanright = sqrt(1-kpan)
```

Since $\sin^2 \theta + \cos^2 \theta = 1$, another way to keep uniform total sound energy is as follows.

```
kpan lineseg 0, p3, 1
ipibytwo = 1.5708
kpanleft = sin(kpan * ipibytwo)
kpanright = sqrt((1 - kpan) * ipibytwo)
```

A good trick for obtaining what sounds like a wider sweep for the pan, especially when using headphones to listen to the output, is to make the angle go from $-\pi/4$ to $3\pi/4$ instead of 0 to $\pi/2$. This can be achieved by replacing the definition of `kpan` above with the following line.

```
kpan lineseg -0.5, p3, 1.5
```

A more realistic pan takes account of the fact that at the far reaches of the sweep, the sound should not be entirely concentrated in one channel. A slightly delayed version can be fed into the other channel, with delay varying up to about 0.7 seconds at the extreme end of the sweep.

Display and spectral display. There is a facility for displaying either a waveform or its spectrum, in an instrument file. So for example the instrument

```
instr 1
  asig oscil 10000 440 1
  out asig
```

```

    display asig p3
endin

```

is the same as (7.15.2), except that the extra line causes the graph of `asig` (of length `p3`) to be displayed. If the flag `-d` (see page 214) is set, this line makes no difference at all. Replacing the display line with

```
    dispfft asig p3, 1024
```

causes a fast Fourier transform of `asig` to be displayed, using an input window size of 1024 points. The number of points must be a power of two between 16 and 4096.

Arithmetic. In the orchestra file, variables represent signed floating point real numbers. The standard arithmetic operations `+`, `-`, `*` (times) and `/` (divide) can be used, as well as parentheses to any depth. Powers are denoted `a ^ b`, but `b` is not allowed to be audio rate. The expression `a % b` returns `a` reduced modulo `b`. Among the available functions are

```

int (integer part)
frac (fractional part)
abs (absolute value)
exp (exponential function, raises e to the given power)
log and log10 (natural and base ten logarithm; argument must be positive)
sqrt (square root)
sin, cos and tan (sine, cosine and tangent, argument in radians)
sininv, cosinv, taninv (arcsine, arccos and arctan, answer in radians)
sinh, cosh and tanh (hyperbolic sine, cosine and tangent)
rnd (random number between zero and the argument)
birnd (random number bewteen plus and minus the argument)

```

Conditional values can also be used. For example,

```
(ka > kb ? 3 : 4)
```

has value 3 if `ka` is greater than `kb`, and 4 otherwise. Comparisons may be made using

```

> (greater than)
< (less than)
>= (greater than or equal to)
<= (less than or equal to)
== (equal to)
!= (not equal to).

```

Expressions, as well as variables, may be compared in this way, but audio rate variables and expressions are not permitted.

Automatic score generation. There are a number of methods of avoiding the tedious process of writing a score file for CSound. One method is to use the *score translator* program `scot`. This takes a text file `<filename>.sc` written in a compressed score notation and writes out a score file `<filename>.sco`.

Another is to use Cscore, which is a program for making and manipulating score files. The user writes a control program in the C language, which makes use of a set of function definitions contained in a header file cscore.h. Finally, there is MIDI2CS, a program which takes a MIDI file as input, and outputs a score file. There is also a considerable amount of support for MIDI within the CSound language.

DirectCSound is a realtime version of CSound for the PC, and can be obtained from Gabriel Maldonado's home page at

<http://web.tiscalinet.it/G-Maldonado/home2.htm>

I have not tried it out, so I cannot comment on how well it works, but it looks promising.

Further reading on CSound:

Richard Boulanger, *The CSound book* [9].

Electronic Musician, Feb 1998 issue.

Keyboard, Jan 1997 issue.

7.19. Other methods of synthesis

Sampling is not really a form of synthesis at all, but is often used in digital synthesizers. It is usual to sample sounds at only a small collection of pitches, and then to pitch shift by stretching or compressing the waveform, in order to fill in the gaps. Pitch shifting a digital signal introduces high frequency noise, related to the fact that the sample rate is not being shifted at the same time. This is removed using a low pass filter.

Wavetable synthesis is a method related to sampling, in which digitally recorded wave files are used as raw material to produce sounds which are a sort of hybrid between synthesis and sampling. It is usual to use one wave file for the attack portion of the sound, and another for the sustain portion. In the case of the sustain portion, a whole number of periods of the sound are used to form a loop which is repeated. An envelope is then applied to shape the sound, and then finally the result is pitch shifted and put through a low pass filter. An exception to this general procedure is "one shot" sounds such as short percussive sounds. These are usually just recorded as a single wave-file without looping.

Granular synthesis is a method where the sound comes in small packets called *grains*, whose duration is usually of the order of ten milliseconds. Thousands of these grains are used in each second, to create a sound texture. Usually, some algorithm is used for describing large quantities of grains at a time, so that each grain does not have to be described separately.

Further reading on granular synthesis:

S. Cavaliere and A. Piccialli, *Granular synthesis of musical signals*, appears as article 5 in Roads et al [94], pages 155–186.

John Duesenberry, *Square one: a world in a grain of sound*, Electronic Musician, November 1999.

Curtis Roads, *Granular synthesis of sound*, Computer Music Journal 2 (2) (1978), 61–62. A revised and updated version of this article appears as chapter 10 of Roads and Strawn [95], pages 145–159.

Curtis Roads, *Granular synthesis*, Keyboard, June 1997.

7.20. The phase vocoder

The phase vocoder is a method of sound analysis and manipulation. It is based on the technique of applying a discrete Fourier transform to small windows of the original sound. The transform may then be manipulated, and finally the sound may reconstructed from the manipulated transform. For example, it is not hard to stretch a sound without altering the pitch using this technique.

Further reading:

Mark Dolson, *The phase vocoder: a tutorial*, Computer Music Journal 10 (4) (1986), 14–27.

M.-H. Serra, *Introducing the phase vocoder*, appears as article 2 in Roads et al [94], pages 31–90.

7.21. Chebychev polynomials

Composition of functions in general is a good way of obtaining synthetic tones. For example, if we take a basic cosine wave $\cos \nu t$ and compose it with the function $f(x) = 2x^2 - 1$ then we obtain

$$2 \cos^2 \nu t - 1 = \cos 2\nu t.$$

So composing with this function has the effect of doubling frequency. The corresponding functions for arbitrary integer multiples of frequency are called the *Chebychev⁹ polynomials of the first kind*, which we now investigate.

Let $T_n(x)$ be the polynomial defined inductively by $T_0(x) = 1$, $T_1(x) = x$, and for $n > 1$,

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x).$$

Thus for example we have

$$T_2(x) = 2x^2 - 1,$$

$$T_3(x) = 4x^3 - 3x,$$

$$T_4(x) = 8x^4 - 8x^2 + 1$$

$$T_5(x) = 16x^5 - 20x^3 + 5x$$

$$T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1$$

$$T_7(x) = 64x^7 - 112x^5 + 56x^3 - 7x.$$

⁹Other spellings for this name include Tchebycheff and Chebichev.

LEMMA 7.21.1. For $n \geq 0$ we have $T_n(\cos \nu t) = \cos n\nu t$.

PROOF. The proof is by induction on n . We begin by observing that

$$\cos \nu t \cos(n-1)\nu t - \sin \nu t \sin(n-1)\nu t = \cos n\nu t$$

$$\cos \nu t \cos(n-1)\nu t + \sin \nu t \sin(n-1)\nu t = \cos(n-2)\nu t$$

(see §1.7), so that adding and rearranging, we have

$$\cos n\nu t = 2 \cos \nu t \cos(n-1)\nu t - \cos(n-2)\nu t.$$

Now for $n = 0$ and $n = 1$, the statement of the lemma is obvious from the definition. For $n \geq 2$, assuming the statement to be true for smaller values of n , we have

$$\begin{aligned} T_n(\cos \nu t) &= 2 \cos \nu t T_{n-1}(\cos \nu t) - T_{n-2}(\cos \nu t) \\ &= 2 \cos \nu t \cos(n-1)\nu t - \cos(n-2)\nu t \\ &= \cos n\nu t. \end{aligned}$$

So by induction, the lemma is true for all $n \geq 0$. \square

Using a weighted sum of Chebychev polynomials and composing, we can obtain a waveform with the corresponding weights for the harmonics. Changing the weighting with time will change the timbre of the resulting tone. So for example, if we apply the operation

$$T_1 + \frac{1}{3}T_3 + \frac{1}{5}T_5 + \frac{1}{7}T_7 + \frac{1}{9}T_9 + \frac{1}{11}T_{11}$$

to a cosine wave, we obtain an approximation to a square wave (see equation (2.2.8)). This operation will turn any mixture of cosine waves into the same mixture of square waves.

Exercises

1. Show that $y = T_n(x)$ satisfies *Chebychev's differential equation*

$$(1-x^2)\frac{d^2y}{dx^2} - x\frac{dy}{dx} + n^2y = 0.$$

2. Show that

$$T_n(x) = x^n - \binom{n}{2}x^{n-2}(1-x^2) + \binom{n}{4}x^{n-4}(1-x^2)^2 - \dots$$

3. Draw a graph of $y = T_n(x)$ for $-1 \leq x \leq 1$ and $0 \leq n \leq 5$.

7.22. Digital formats for music

Error correcting codes, cross interleaved Reed–Solomon codes

7.23. MIDI

Most synthesizers these days talk to each other and to computers via MIDI cables. MIDI stands for “Musical Instrument Digital Interface”. It is an internationally agreed data transmission protocol, introduced in 1982, for the transmission of musical information between different digital devices. It is important to realise that in general there is no waveform information present in MIDI, unless the message is a “sample dump”. Instead, most MIDI messages give a short list of abstract parameters for an event.

There are three basic types of MIDI message: note messages, controller messages, and system exclusive messages. Note messages carry information about the starting time and stopping time of notes, which patch (or voice) should be used, the volume level, and so on. Controller messages change parameters like chorus, reverb, panning, master volume, etc. System exclusive messages are for transmitting information specific to a given instrument or device. They start with an identifier for the device, and the body can contain any kind of information in a format proprietary to that device. The commonest kind of system exclusive messages are for transmitting the data for setting up a patch on a synthesizer.

The MIDI standard also includes some hardware specifications. It specifies a baud rate of 31.25 KBaud. For modern machines this is very slow, but the for the moment we are stuck with this standard. One of the results of this is that systems often suffer from MIDI “bottlenecks,” which can cause audibly bad timing. The problem is especially bad with MIDI data involving continually changing values of a control variable such as volume or pitch.

Further reading:

S. de Furia and J. Scacciaferro, *MIDI programmer’s handbook* [32].

F. R. Moore, *The dysfunctions of MIDI*, Computer Music Journal 12 (1) (1988), 19–28.

J. Rothstein, *MIDI, A comprehensive introduction* [100].

Eleanor Selfridge-Field (Editor), Donald Byrd (Contributor), David Bainbridge (Contributor), *Beyond MIDI: The Handbook of Musical Codes*, M. I. T. Press (1997).

7.24. Software and internet resources

The information in this section is, of course, very volatile. So it is likely that by the time you are reading this, a lot of the information will already be out of date.

Scales and Temperaments: The best internet resource on the subject of scales, temperaments and tunings is

<http://www.xs4all.nl/~huygensf/doc/bib.html>

This is part of the Huygens-Fokker Foundation website, maintained by Manuel Op de Coul, and consists of a giant bibliography together with links

to other internet resources on the subject. The front page of the website is at
<http://www.xs4all.nl/~huygensf/english/>

Also on the same website, a discography of microtonal music can be found at
<http://www.xs4all.nl/~huygensf/doc/discs.html>

A large collection of scales and temperaments can be found at
<http://www.xs4all.nl/~huygensf/doc/scales.zip>

and the Scala scales and temperaments software can be found at
<http://www.xs4all.nl/~huygensf/scala/>

To subscribe to the alternate tunings email discussion group, send an empty email message to tuning-subscribe@onelist.com.

Just Intonation Network: <http://www.dnai.com/~jinetwk/>

Bohlen–Pierce scale: <http://members.aol.com/bpsite/index.html>

Music Theory: Sites offering free music theory tuition online include

Easy Music Theory (Gary Ewer): <http://www.musictheory.halifax.ns.ca/>

Java Music Theory: <http://academics.hamilton.edu/music/spellman/JavaMusic/>

Online Music Instruction Page (Ken Fansler):

<http://orathost.cfa.ilstu.edu/~kwfansle/onlinemusicpage.htm>

Practical Music Theory: <http://www.teoria.com/java/eng/java.htm>

Sound editors: There are some good shareware sound editors. Among the best are:

Cool Edit: <http://www.syntrillium.com/cooledit/index.html>

Goldwave: <http://www.goldwave.com/>

Acid Wav: <http://www.polyhedric.com/software/acid/>

There are two freeware audio frequency analysers for the PC called

Spectrogram: <http://www.monumental.com/rshorne/gram.html>

Frequency analyzer: <http://www.hitsquad.com/smm/programs/Frequency/>

CSound: This free software is described in §7.15. Versions for various platforms (PC, Mac, Unix, Atari, NeXT) are available from

<ftp://ftp.maths.bath.ac.uk/pub/dream/>

To subscribe to the email discussion group for CSound, send an empty message to csound-subscribe@lists.bath.ac.uk. Further information about CSound can be found at the following www pages:

<http://www.mitpress.com/e-books/csound/frontpage.html>
 (the CSound front page, MIT Press)

http://www.bright.net/~dlphilp/dp_csound.html

(Dave Phillips' PC CSound page)

http://www.bright.net/~dlphilp/linux_csound.html

(Dave Phillips' Linux CSound page)

<http://music.dartmouth.edu/~dupras/wCsound/csoundpage.html>

(Martin Dupras' CSound page)

A utility for PC and Unix called MIDI2CS, written by Rudiger Borrmann, converts MIDI files to Csound scores. It is available from

<http://www.snafu.de/~rubo/songlab/midi2cs/csound.html>

A utility for emulating the Yamaha DX7 with CSound can be found at Jeff Harrington's site

<http://www.parnasse.com/dx72csnd.shtml>

Other synthesis software: This is a rapidly expanding field, and new products turn up almost every week. The ones I know of are as follows.

Audio Architect (PC): <http://www.audiarchitect.com/>

Bitheadz Retro AS-1 (Mac): <http://www.bitheadz.com> (free demo)

CLM (Common Lisp Music, freeware):

<http://www-ccrma.stanford.edu/CCRMA/Software/clm/clm.html>

CMix (Next, Linux, Sparc, SGI, PowerMac; freeware):

<http://www.music.princeton.edu/winham/cmix.html>

Cybersound Studio (Mac, Win 95/98/ME): <http://www.cybersound.com>

Cycling '74 (Mac + Opcode Max): <http://www.cycling74.com> (free demo)

Grain Wave (Mac shareware): <http://www.nmol.com/users/mikeb/>

Ik Multimedia's Groovemaker and Axé (Mac, Win 95/98/ME):

<http://www.ikmultimedia.com> (free demo)

Lemur (Mac): <http://datura.cerl.uiuc.edu/Lemur/AboutLemur.html>

Native Instruments Reaktor/Generator/Transformer (Win 95/98/ME):

<http://www.native-instruments.com/> (free demo)

Nemesis GigaSampler (Win 95/98/ME): <http://www.nemesismusic.com>

Nyquist (freeware):

<http://www.cs.cmu.edu/afs/cs.cmu.edu/project/music/web/music.software.html>

Seer Systems Reality: <http://www.seersystems.com>

Steinberg Rebirth RB-338 (Mac, Win 95/98/ME/NT):

<http://www.us.steinberg.net> (free demo)

Synthesis Toolkit (C++ code):

<http://www-ccrma.stanford.edu/CCRMA/Software/STK/>

Virtual Sampler (Win 95/98/ME/NT):

This can be found at Sonic Spot, <http://www.sonicspot.com/>, or at MAZ, <http://www.maz-sound.com/>. It is shareware, and the unregistered version does everything but save sounds. It includes a complete Yamaha DX7 emulation.

The most impressive site for information on the processes and control of synthesis is Electronic Music Interactive, at

<http://nmc.uoregon.edu/emi/emi.html>

Synthesizers and patches: The best general websites for synthesizers and patches are

Synthesizer and Midi Links Page:

<http://www.interlog.com/~spinner/lbquirke/synthesis/links/>

Synth Site: <http://www.sonicstate.com/bbsonic/synth/index.cfm>

At the anonymous ftp site <ftp://ftp.ucsd.edu>, in the subdirectory `/midi/patches`, there are patches for Casio CZ-1, CZ-2, Ensoniq ESQ1, SQ1, Kawai K1, K4, K5, XD-5, Korg M1, T3, WS (Wavestation), Roland D10, D5, D50, D70, SC55, U20, and Yamaha DX7, FB01, TX81Z, SY22, SY55, SY77, SY85.

For the Yamaha DX7, there is a web page which I maintain at

<http://www.math.uga.edu/~djb/dx7.html>

which contains, among other things, a patch archive and instructions for joining the email discussion group.

Typesetting software:

CMN (Common Music Notation, freeware for NeXT and SGI machines):

<http://ccrma-www.stanford.edu/CCRMA/Software/cmn/cmn.html>

Finale is a commercial music notation package for the Mac and PC Windows (current version Finale 98), and is available from Coda Music Software. Their web site

<http://www.codamusic.com/>

has more information. A free demonstration version of the program is available on this web site. Without academic discount, Finale is very expensive, but with academic discount it costs about \$200–\$250. To subscribe to the email discussion group for Finale, send an email message to listserv@shsu.edu with the phrase “subscribe Finale” or “subscribe Finale-Digest” in the body of the message. To be removed from the list, send “signoff Finale” or “signoff Finale-Digest” to the same address.

Finale forum (not sanctioned by Coda Music): <http://www.cmp.net/finale/>

Finale Resource Page: http://www.peabody.jhu.edu/~skot/finale/fin_home.html

Ftp site for Finale users: <ftp://ftp.shsu.edu/pub/finale/>

Keynote is a public domain textual, graphical and algorithmic music editor for the Unix X Window system, the Mac or the Amiga, available from

<ftp://xcf.berkeley.edu>

LilyPond is a GNU project (and hence free) music typesetter for Unix systems. It is available from

<http://www.cs.uu.nl/~hanwen/lilypond/index.html>

Lime (Mac, Win 95): <http://datura.cerl.uiuc.edu/>

Mozart: <http://www.mozart.co.uk/>

Muzika 3 is a public domain (freeware) music notation package for PC Windows, available from

<ftp://garbo.uwasa.fi/windows/sound/muzika3.zip> or from

<ftp://ftp.cica.indiana.edu/ftp/pub/win3/sounds/muzika3.zip>

Nutation (NeXT, freeware): <ftp://ccrma-ftp.stanford.edu/pub/Nu.pkg.tar>

Overture is a Mac based commercial music notation package.

Score: <http://ace.acadiau.ca/score/links3.htm>

Sibelius is a notation package for the PC: <http://www.sibelius.com/>

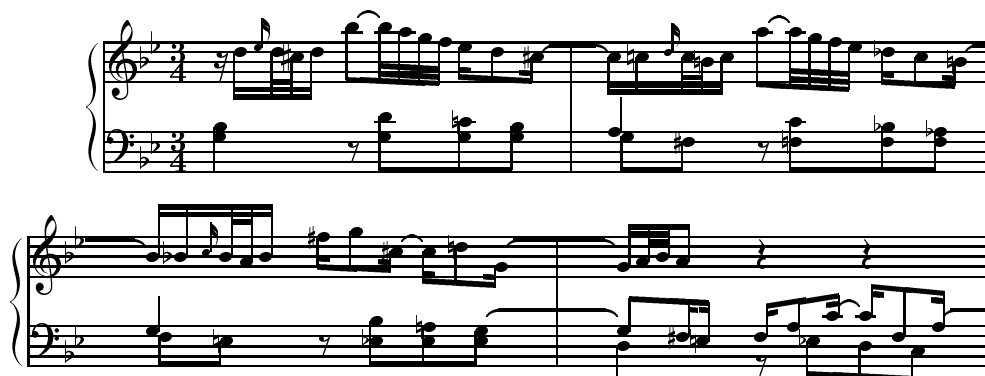
MusicT_EX: MusicT_EX, written by the french organist Daniel Taupin, and its successor MusixT_EX are public domain music typesetting packages to run under Donald Knuth's T_EX program. The necessary files may be found on

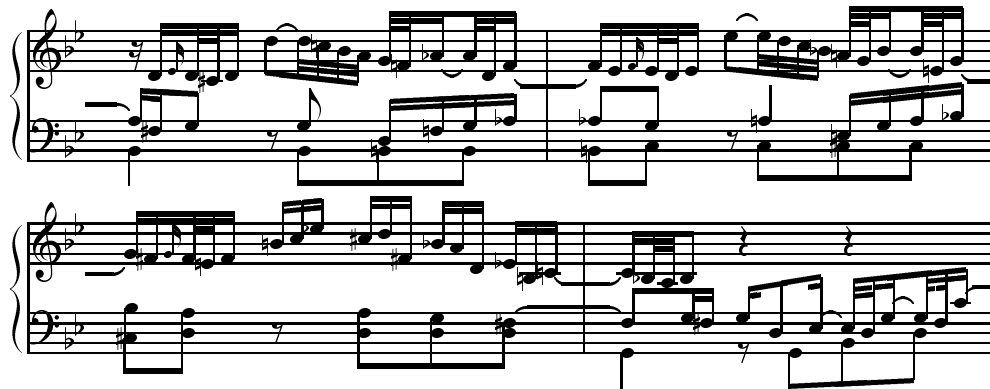
<ftp://rsovax.ups.circe.fr/TeX/musictex/>

See also: <http://www.gmd.de/Misc/Music/>

A public domain version of T_EX for Windows 95/98/ME, called MikT_EX, and can be found at <http://www.miktex.de>. Versions for all platforms are available from CTAN at <ftp.tex.ac.uk>, <ftp.dante.de> or ctan.tug.org. See also TUG (the T_EX user's group) at <http://tug.org>.

Goldberg Variation 25, J. S. Bach



Example of Output from Music \TeX

Mu \TeX is the precursor of Music \TeX , written by Andrea Steinbach and Angelica Schofer. It is in the public domain, and is available by anonymous ftp from ymir.claremont.edu in [anonymous.tex.music.mtex] (VMS).

MIDI2 \TeX is a program written by Hans Kuykens for converting MIDI files into Music \TeX files. The latest version can be found on CTAN (see page 232).

ABC2M \TeX is a program for converting tunes from its own text-based format into Music \TeX files. It is designed primarily for folk and traditional music of Western European origin written on one staff in standard classical notation. It can be obtained directly from its author, Chris Walshaw, via email: C.Walshaw@gre.ac.uk, or from

<ftp://celtic.stanford.edu/pub/tunes/abc2mtex/>

Sequencers: Cakewalk and Cubase are competing commercial Windows based sequencers, neither of which is cheap, but both of which are packed with features. To subscribe to the Cakewalk users' group, send a message to listserv@lists.colorado.edu with the phrase "subscribe cakewalk" in the body of the message. To subscribe to the Cubase users' group, send a message to cubase-users-request@nessie.mcc.ac.uk. Messages for the group should be sent to cubase-users@mcc.ac.uk.

Power Tracks Pro Audio is a very cheap, but fully functional commercial Windows based sequencer, available from PG Music for \$29. More information can be found at

<http://www.pgmusic.com/>

Rosegarden is an integrated MIDI sequencer and musical notation editor. It is free software for Unix and X, and it may be found at

<http://www.bath.ac.uk/~masjpf/rose.html>

WinJammer is a shareware Windows based sequencer, which may be found at

<ftp://ftp.cnr.it/pub/msdos/win3/sounds/wjmr23.zip>

WinJammer Pro (I'm not sure what the difference is) is in the same directory, as wjpro.zip.

Random music: There are a number of freeware/shareware probabilistic music programs designed to run under Windows.

Aleatoric composer (shareware):

<ftp://oak.oakland.edu/msdos/music/alcomp11.zip>

Art Song 2.3 (shareware): <http://members.aol.com/strohbeen/fmlsw.html>

FMusic 1.9 (freeware): <http://members.aol.com/dsinger594/caman/fmusic19.zip>

FractMus 2.3 (freeware): <ftp://ftp.cdrom.com/pub/win95/music/frctmu25.zip>

Fractal Tune Smithy (freeware/shareware):

<http://matrix.crosswinds.net/~fractalmelody/index.htm>

Improvise 1.2 (shareware):

<ftp://ftp.cnr.it/pub/msdos/win3/sounds/impvz120.zip>

Make-Prime-Music (freeware):

<http://members.tripod.de/Latroductus98/index.html>

Mandelbrot Music (freeware): <http://www.fin.ne.jp/~yokubota/mandele.shtml>

MusiNum 2.08 (freeware):

<http://www.forwiss.uni-erlangen.de/~kinderma/musinum/musinum.html>

QuasiFractalComposer 2.01 (freeware):

<http://members.tripod.com/~paulwhalley/>

Tangent (free/shareware): <http://www.randomtunes.com/>

The Well Tempered Fractal 3.0 (freeware):

<http://www-ks.rus.uni-stuttgart.de/people/schulz/fmusic/wtf/wtf30.zip>

MIDI: The MIDI specification can be obtained via email by sending a message with the phrase GET MIDISPEC PACKAGE in the message body, to listserv@auvm.american.edu. There are archives of MIDI files available at

<ftp://ftp.cs.ruu.nl/MIDI/DOC/archives/>

<ftp://ftp.waldorf-gmbh.de/pub/midi/>

There are two programs called mf2t and t2mf which convert standard MIDI files into human readable ASCII text and back again. The MIDI home page on the WWW is

<http://www.eeb.ele.tue.nl/midi/index.html>

A good starting point for information about MIDI is the Northwestern University site

<http://nuinfo.nwu.edu/musicschool/links/projects/midi/expmidiindex.html>

Academic Computer Music: The following departments in American universities have programs in computer music. CalArts (David Rosenboom,

Morton Subotnick), Carnegie Mellon (Roger Dannenberg), MIT (Tod Machover, Barry Vercoe), Princeton (Paul Lansky), Stanford (John Chowning, Chris Chaffe, Perry Cook, etc.), SUNY Buffalo (David Felder, Cort Lippe), UC Berkeley (David Wessel), UCSD (Miller Puckett, F. Richard Moore, George Lewis, Peter Otto).

IRCAM is an institution in Paris for computer music, which has an anonymous ftp site at <ftp.ircam.fr>. In particular, the music/programming environment MAX can be found there.

Music Theory Online (the Online Journal of the Society for Music Theory) can be found at

<http://boethius.music.ucsb.edu/mto/mtohome.html>

FAQs: There are several FAQs (“Frequently Asked Questions” and their answers) available on the internet. Two that I know of are available from the site xcf.berkeley.edu, either by anonymous ftp or by email. They are the electronic and computer music FAQ, in </pub/misc/netjam/doc/ECMFAQ> and the composition FAQ, in </pub/misc/netjam/doc/FAQ/composition/compositionFAQ.entire>. Or send an email message to netjam-request@xcf.berkeley.edu with the subject line “request for ECM FAQ”, respectively “request for composition FAQ”.

Other resources: The following are some interesting WWW pages:

Everyone seems to want to know more about the infamous “Mozart effect”. Volume VII, Issue 1 (Winter 2000) of *MuSICA Research Notes* is devoted to this much overpublicized and misunderstood topic, and can be found at

<http://www.musica.uci.edu/mm/V7I1W00.html>

<http://www.oulu.fi/music.html> is a directory of music sites.

<http://www.music.indiana.edu/misc/music-resources.html> is a catalog of music resources.

<http://sunsite.unc.edu/pub/ianc/index.html> is the Internet Underground Music Archive.

To subscribe to the electronic music email discussion group, send a message to listserv@auvm.bitnet with the line “SUB EMUSIC-L” in the body. Messages for the group should be sent to emusic-l@auvm.bitnet. For the digests only, replace EMUSIC-L with EMUSIC-D.

Online papers: See Appendix O for a selection of relevant papers which can be downloaded from academic journals.

