Kevin Karplus
Computer Science Department
Cornell University
Ithaca, New York 14853

Alex Strong
Computer Science Department
Stanford University
Stanford, California 94305

Digital Synthesis of Plucked-String and Drum Timbres

Introduction

There are many techniques currently used for digital music synthesis, including frequency modulation (FM) synthesis, waveshaping, additive synthesis, and subtractive synthesis. To achieve rich, natural sounds, all of them require fast arithmetic capability, such as is found on expensive computers or digital synthesizers. For musicians and experimenters without access to these machines, musically interesting digital synthesis has been almost impossible.

The techniques described in this paper can be implemented quite cheaply on almost any computer. Real-time synthesis implementations have been done for Intel 8080A (by Alex Strong), Texas Instruments TMS9900 (by Kevin Karplus), and SC/MP (by Mike Plass) microprocessors. David Jaffe and Julius Smith have programmed the Systems Concept Digital Synthesizer at the Center for Computer Research in Music and Acoustics (CCRMA) to perform several variants of the algorithms (Jaffe and Smith 1983).

Not only are the algorithms simple to implement in software, but hardware realizations are easily done. The authors have designed and tested a custom n-channel metal-oxide semiconductor (nMOS) chip (the Digitar chip), which computes 16 independent notes, each with a sampling rate of 20 KHz.

Despite the simplicity of the techniques, the sound is surprisingly rich and natural. When the plucked-string algorithm was compared with additive synthesis at Bell Laboratories, it was found that as many as 30 sine wave oscillators were needed to produce a similarly realistic timbre (Sleator 1981). The entire plucked-string algorithm requires only as much computation as one or two sine wave oscillators.

The parameters available for control are pitch, amplitude, and decay time. The pitch is specified by an integer that is approximately the period of the sound, in samples (periodicity parameter p). Amplitude is specified as the initial peak amplitude A. Decay time is determined by the pitch and by a decay stretch factor S.

The algorithms in this paper lack the versatility of FM synthesis, additive synthesis, or subtractive synthesis. They are, however, cheap to implement, easy to control, and pleasant to hear. For musicians interested primarily in performing and composing music, rather than designing instruments, these algorithms provide a welcome new technique. For those interested in instrument design, they open a new field of effective techniques to explore.

Wavetable Synthesis

One standard synthesis technique is the *wavetable synthesis* algorithm. It consists of repeating a number of samples over and over, thus producing a purely periodic signal. If we let Y_t be the value of the t^{th} sample, the algorithm can be written mathematically as

$$Y_t = Y_{t-p}.$$

The parameter p is called the *wavetable length* or *periodicity parameter*. It represents the amount of memory needed and the period of the tone (in sam-

This research was supported in part by the Fannie and John Hertz Foundation.

Computer Music Journal, Vol. 7, No. 2,
Summer 1983, 0148-9267/83/020043-13 $04.00/0,

ples). The initial conditions of the recurrence relation completely determine the resulting timbre. Normally, a sine wave, triangle wave, square wave, or other simple waveform is calculated and loaded into the wavetable before the note is played. With a sampling frequency of f_s, the frequency of the tone is f_s/p.

The wavetable-synthesis technique is very simple but rather dull musically, since it produces purely periodic tones. Traditional musical instruments produce sounds that vary with time. This variation can be achieved in many ways on computers. The approach in FM synthesis, additive synthesis, subtractive synthesis, and waveshaping is to do further processing of the samples after taking them from the wavetable. All the algorithms described in this paper produce the variation in sound by modifying the wavetable itself.

What sort of modifications to the wavetable are useful? If no modification is done, the harmonic content is fixed, and the sound is purely periodic. To get an almost periodic output, the changes from period to period must be small. To keep the amount of computation and the number of memory accesses small, only one entry in the wavetable is changed for each sample output. Furthermore, since we have to look at a value in the wavetable each sample time, it makes sense to attempt to change that value. With the most recently read sample of the wavetable being the only one changed, the wavetable can be viewed as a delay line of length p. Figure 1 illustrates the general form of the algorithms from a delay-line standpoint.

Plucked-String Algorithm

The simplest modification, invented by Alex Strong in December 1978, is to average two successive samples. This can be written mathematically as

$$Y_t = \frac{1}{2}\,(Y_{t-p} + Y_{t-p-1}).$$

It turns out that this averaging process produces a slow decay of the waveform. The resulting tone of this algorithm has a pitch that corresponds to a period of $p + \frac{1}{2}$ samples (frequency $f_s/(p + \frac{1}{2})$), and sounds remarkably like the decay of a plucked string. Since no multiplication is required (only adding and shifting), the algorithm is fast and easy to implement on microprocessors.

This recurrence can be viewed as a digital filter without inputs, as in Fig. 2. The naturalness of the sound derives largely from differing decay rates for the different harmonics. No matter what initial spectrum a tone has, it decays to an almost pure sine wave, eventually decaying to a constant value (silence). Later in the paper we will use digital filtering techniques to show that the decay time for the n^{th} harmonic is roughly proportional to p^3/n^2.

As with any recurrence relation, initial conditions must be specified. In concrete terms, this amounts to preloading the wavetable with appropriate values. The initial values can form a sine wave, triangle wave, or any other desired waveform, just as for wavetable synthesis. To produce a realistic string sound, it is desirable to start the note with a lot of high harmonics. To accomplish this, the wavetable is filled with random values at the beginning of each new note. Since the samples in the wavetable are repeated, the randomness does not produce hiss or noise.

Without the decay algorithm, a random wavetable has essentially equal harmonics up to the Nyquist frequency, sounding like a reed organ. With decay, the higher harmonics decay rapidly, producing a plucked-string sound very similar to that of a guitar. The use of random initial load also has the advantage of giving each repetition of the same pitch a slightly different harmonic structure. This variation is small enough that the notes sound as if they come from the same instrument, but large enough that the notes don't sound like mechanical repetition. Of course, the wavetable can be copied to get truly identical notes.

One fast way to provide the initial randomness in the wavetable is to use *two-level randomness*. Mathematically, the initial conditions are

$$Y_t = \begin{cases} +A & \text{probability } \frac{1}{2} \\ -A & \text{probability } \frac{1}{2} \end{cases} \quad \text{for } -p \leq t \leq 0.$$

Fig. 1. Generic design for wavetable-modification techniques.

Fig. 2. Basic plucked-string (guitar) technique.

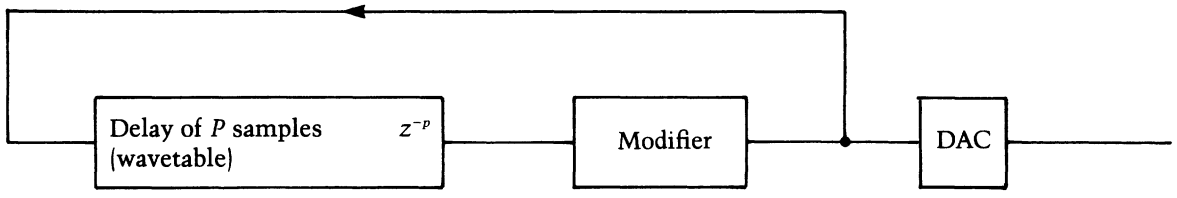
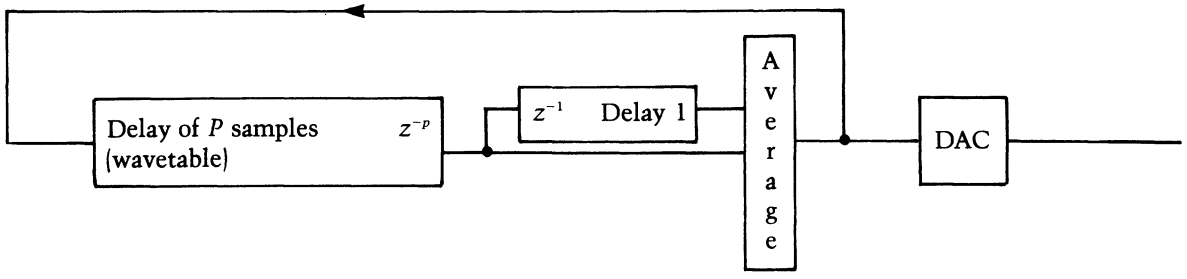


Fig. 2



The root-mean-square (rms) amplitude of the output is A, which is half the peak-to-peak amplitude. Using two-level randomness provides a signal about 5 db louder than using uniform random numbers between $-A$ and $+A$. Only a single bit of randomness is needed for each sample, so a feedback shift-register random-bit generator can be used (Knuth 1981, p. 29). A random-bit generator is simpler than a full random-word generator in either software or hardware.

There are some limitations on the values of the periodicity parameter p. If p is small, the variation between different initial conditions will be relatively large, resulting in poor control of amplitude. Also, since the pitch of a note is determined by p, and p must be an integer, not all frequencies are available. When p is large, the available frequencies are close together, but when p is small, they are fairly far apart. These effects, combined with the short decay times for small p, make p values less than about 32 undesirable. If the full range of a guitar is desired (up to about 880 Hz), this restriction requires sampling rates of at least 28.6 KHz. With some care in the choice of values for p, adequate performance can be achieved for sampling rates down to about 20 KHz. To get finer frequency resolution than is available by just changing p, it is often possible to vary the sampling rate, as well as the value of p.

After a note has been played, the wavetable is normally reloaded with random values before the next note is played. If the wavetable is not changed, the effect is that of a slur or tie. If p is unchanged, the note continues; if p is changed, the result is a slur between the two pitches. Frequent changes to p can be used to produce glissando and vibrato effects.

In digital filtering terms, preloading of the wavetable can be viewed as switching between an input burst and the feedback (see Fig. 3). By switching rather than adding, we avoid arithmetic overflow and eliminate the need for large word sizes. David James independently published a synthesis technique that has many similarities to ours (James 1978, p. 38). It also uses a digital filter excited by a noise burst. However, it provides only decaying amplitude, with no change in harmonic structure, and was intended for use as an excitation waveform for subtractive synthesis. His technique suffers from the usual digital filtering problem; it requires high-speed, high-precision arithmetic (particularly multiplication).

Fig. 3. Noise-burst input to wavetable.

Fig. 4. Basic drum technique.

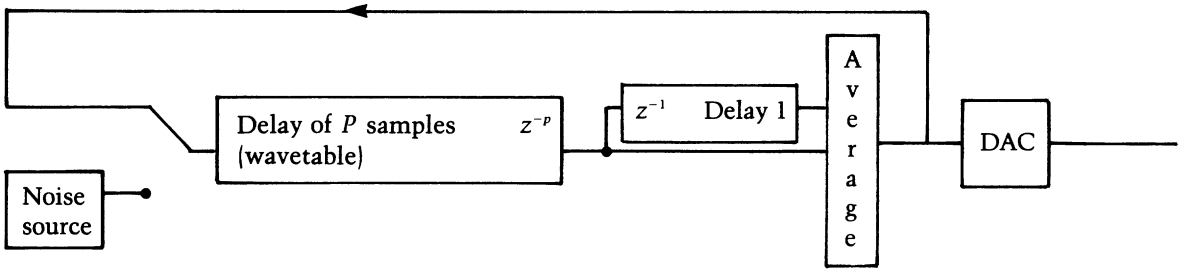
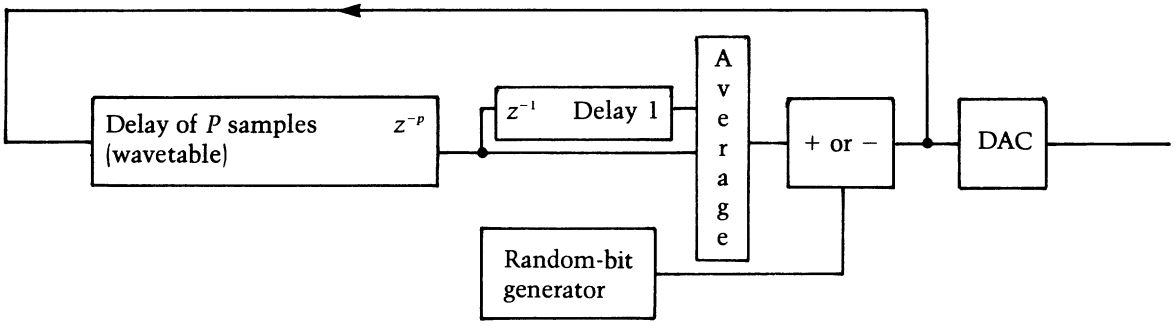


Fig. 4



Drum Algorithm

A simple variation of Strong's basic algorithm yields drum timbres. This was discovered by Kevin Karplus in December 1979. The simplest description of the drum variant is a probabilistic recurrence relation:

$$Y_t = \begin{cases} +\frac{1}{2}(Y_{t-p} + Y_{t-p-1}) & \text{probability } b \\ -\frac{1}{2}(Y_{t-p} + Y_{t-p-1}) & \text{probability } 1 - b. \end{cases}$$

Figure 4 shows the block diagram of the corresponding digital filter.

The parameter b is called the *blend factor*. With a blend factor of 1, the algorithm reduces to the basic plucked-string algorithm, with p controlling the pitch. With a blend factor of ½, the sound is drumlike. Intermediate values produce sounds intermediate between plucked string and drum, some of which are quite interesting musically. A blend factor of 0 negates the entire signal every $p + ½$ samples. This drops the frequency an octave and leaves only odd harmonics of the new fundamental. For fairly high pitches, this is a rather odd timbre that we call a "plucked bottle." For lower pitches, the sound is harplike.

For $b \approx ½$, the wavetable length does not control the pitch of the tone, as the sound is aperiodic. Instead, it controls the decay time of the noise burst. The decay time is roughly proportional to p. For fairly large p (200 or more) and a sampling frequency of 20 KHz, the effect is that of a snare drum. For small p (around 20), the effect is that of a brushed tom-tom. Intermediate values provide intermediate timbres, allowing smooth transition from one drum sound to another.

The initial wavetable can be filled with a constant (A), since the drum algorithm will create the randomness itself. For blends other than $b = ½$, starting with a constant gives some buildup before the decay, while starting with randomness gives maximum amplitude initially. Blends near 1 require

nonconstant initial loading of the wavetable, as little or no randomness is introduced. If b is restricted to 0, 1, or ½ (the most interesting values), then only a single random bit is needed for each sample. If arbitrary values are allowed for b, a more sophisticated random-number generator is required.

Modifications in the Basic Algorithm

Since the overall decay of a note is very roughly proportional to p^3, notes with a short wavetable (high pitch) decay very rapidly. There are several ways to make these notes last longer. The first method is to use a longer wavetable, but fill it with several copies of the same waveform. For example, if the wavetable is doubled in length, the frequency of the fundamental is dropped about an octave. However, if the first half of the wavetable is identical to the second half, only even harmonics will be present, so the note will sound an octave higher than the value of p would indicate. The n^{th} harmonic of the sound we hear is the $2n^{\text{th}}$ harmonic of the fundamental pitch for the lengthened wavetable.

Since the decay time for the n^{th} harmonic is roughly proportional to p^3/n^2, the decay time for the n^{th} harmonic of such a doubled wavetable is proportional to $p^3/(2n)^2 = p^3/4n^2$, instead of $(p/2)^3/n^2 = p^3/8n^2$, as it would be if we used a wavetable of length $p/2$ directly. Note that if p is odd, it isn't possible to fill the buffer with two identical copies, so some of the "fundamental" pitch will remain. However, this small amount will not be noticeable until the note has decayed a long way. Odd values of p used this way can improve the tuning of high notes. The wavetable could also be filled with three, four, or more copies of the same waveform to get still higher notes. Making n copies of a waveform in a table of length p produces a note lasting about n times as long as using a table of length p/n. We call this technique the *harmonic trick*, since it allows us to get any of the first few harmonics of our fundamental pitch.

Decay stretching is a more general, more powerful, and more computationally expensive method for lengthening decay times. (It is still very cheap, since only random numbers are needed, not multiplication or additional table lookup.) The recurrence relation for stretching the basic plucked-string algorithm is

$$Y_t = \begin{cases} Y_{t-p} & \text{probability } 1 - \dfrac{1}{S} \\ + \dfrac{1}{2}\left(Y_{t-p} + Y_{t-p-1}\right) & \text{probability } \dfrac{1}{S}. \end{cases}$$

The new parameter S is called the *stretch factor*, and is always at least 1. The decay time of each overtone is approximately multiplied by S, when compared with the decay time for the same overtone in the basic algorithm. The pitch of the sound is also affected by S, as the period is now about $p + 1/2S$ instead of $p + ½$. The optimum choice for S depends on the sampling rate, p, and the effect desired. By choosing S proportional to p^{-1} or p^{-2}, the decay times for the n^{th} harmonic can be made proportional to p^2/n^2 or p/n^2, instead of p^3/n^2, as they would be for a constant stretch factor. Note that for $S = 1$ the recurrence relation simplifies to the unstretched algorithm. For $S = \infty$ the sound does not decay; this is simple wavetable synthesis. Decay stretching can be used to solve the tuning problem caused by having $p + ½$ instead of p as the period of the basic algorithm. By making S proportional to p^{-1}, exact (just intonation) intervals can easily be tuned. Even simple approximations (such as doubling S for each higher octave) help with the tuning of intervals.

If nonrandom wavetable loads are used with large values of S (long decays), woodwindlike sounds can be produced. Strong has produced sounds that he refers to as a "plucked bassoon." More research is being done to determine appropriate initial wavetable loads and parameter settings. The "pluck" can be eliminated in a variety of ways; for example, by starting two strings exactly out of phase, then having them drift apart, or by using an external amplitude envelope. Whether realistic woodwind attacks are obtainable is currently unknown.

The recurrence relation for stretched drums is

$$Y_t = \begin{cases} +Y_{t-p} & \text{probability } b\left(1 - \frac{1}{S}\right) \\ -Y_{t-p} & \text{probability } (1-b)\left(1 - \frac{1}{S}\right) \\ +\frac{1}{2}(Y_{t-p} + Y_{t-p-1}) & \text{probability } b\frac{1}{S} \\ -\frac{1}{2}(Y_{t-p} + Y_{t-p-1}) & \text{probability } (1-b)\frac{1}{S}. \end{cases}$$

Note that the stretch factor and blend factor are independent, so the algorithm can be implemented with two separate tests, and no multiplies are needed. For drums (b near ½), increasing S increases the "snare" sound, allowing smaller values of p to be used for the same duration. If $b = 1$, the recurrence simplifies to the stretched algorithm for string sounds. If $b = ½$ and $S = \infty$, single-bit white noise is produced.

In many digital synthesizers, fast multiplies are available, but probabilistic algorithms like decay stretching are difficult to implement. On these machines, *multiplicative decay stretching* can be done:

$$Y_t = cY_{t-p} + dY_{t-p-1}.$$

To get an effect similar to the probabilistic decay-stretching algorithm, set $d = 1/2S$ and $c = 1 - d$. If $c + d < 1$, then there is an overall loss in the feedback loop, so decay times are reduced, and the signal eventually decays to zero, rather than just to a constant. David Jaffe has been experimenting with this algorithm at CCRMA (Jaffe and Smith 1983).

Shortening the decay times is more difficult than lengthening them. Use of multiplicative decay stretching, with $c + d < 1$, is one way to shorten decay times. Another possibility is to change the recurrence to one that smooths out the waveform faster. For example, we have experimented with the *1-2-1 weighting algorithm*:

$$Y_t = \frac{Y_{t-p-1} + 2Y_{t-p} + Y_{t-p+1}}{4}.$$

Unfortunately, the extra computation time this algorithm takes may increase the time per sample enough to offset the reduced number of samples needed for the signal to decay. Decay stretching and the harmonic trick work just as well with this algorithm as with the basic plucked-string algorithm. It can also be modified to a drum algorithm, but there seems to be no advantage to this, since decay time for drums can be adequately controlled by varying p.

One advantage to the 1-2-1 weighting algorithm is that the frequency is f_s/p, rather than $2f_s/(2p+1)$. This allows better tuning, since consonant intervals are integer ratios of frequency.

Alan Siegel has suggested the following variation for reducing decay times:

$$Y_t = \frac{Y_{t-p} + Y_{t-1}}{2}.$$

This variant reduces decay times enormously for the higher harmonics but does not change the lower harmonics nearly as much. The resulting sound is still a plucked-string sound, but it is softer, more like a nylon string than a steel one. All the modifications to the basic algorithm can also be applied to this variation.

Generalizations of the Algorithms

Siegel's variant, multiplicative decay stretching, and the basic algorithm can all be viewed as variants of the *two-point recurrence*,

$$Y_t = cY_{t-g} + dY_{t-h}.$$

For stability, we need to have $c + d \leq 1$. So far, we've only investigated algorithms where g and h are near p or near 1. Many other possibilities (such as $g = 2h$) need exploring.

The 1-2-1 weighting algorithm is one of many possible three-point recurrences. Jaffe has been experimenting with others to get independent control of decay time and pitch. There is no reason (other than increased computational cost) not to explore recurrence relations with even more terms.

Probabilistic decay stretching can be generalized to probabilistic choice from any set of recurrence relations. In some cases, this can be used to get the effect of weighted averaging without multiplication

(as in decay stretching). It can also produce wholly new sounds (as in the drum algorithms).

Hints for Implementation

On the Intel 8080A, four voices at a sampling rate of at least 10 KHz have been obtained (also, two voices at 20 KHz). By using a processor with a faster clock rate (Z80A or 8085), the sampling rate can be increased by a factor of about 2 with no change in the programs. On the TMS9900, two voices at 10 KHz or one voice at 20 KHz have been achieved. These times are for the basic string algorithms (except for the four-voice implementation, which involved a number of tricks). Drums, decay stretching, and the 1-2-1 weighting algorithm all are slower. These implementations and the Digitar chip (16 voices at 20 KHz) are described in a separate paper (Karplus and Strong 1983). Jaffe has programmed the Systems Concept Digital Synthesizer at CCRMA to implement 29 voices at 17 KHz.

There are many different ways to implement the recurrence relations for wavetable synthesis and the decay algorithms. The two most interesting algorithms are the *decreasing-counter* and *circular-buffer* techniques. For the decreasing-counter method, a wavetable is stored backward in sequential memory locations, with a pointer to the current value. As each sample is output, the pointer is decremented to get the next value. When the pointer is decreased below the bottom of the table, it is reset to the top.

Normally, the decreasing-counter technique is good for single-voice implementations only. Multiple voices cause difficulty with "balancing the loop" (keeping the sampling rate constant), since the pointer resettings occur at different times. Using a faster processor doesn't help, unless there is enough spare time in each sample to do both pointer resettings. A separate piece of hardware to resynchronize samples (a first-in-first-out [FIFO] buffer before the digital-to-analog converter [DAC]) would eliminate this problem. However, a two-voice implementation is possible using the decreasing-counter technique without extra hardware, if one voice is a plucked string and the other is a drum. The same p value is used for both voices, sacrificing control of the drum timbre but allowing both pointers to be reset together.

One trivial variant of the basic algorithm replaces Y_{t-p-1} with Y_{t-p+1}, changing the nominal period to $p - \frac{1}{2}$. With a one-voice, decreasing-counter algorithm, this variant permits compensation to period p by using the extra time needed for restoring the pointer. If this extra time can be set to half the normal sample time, then the average sampling period is $1 + 1/2p$ times as long as the inner-loop time. This means that the frequency of the tone is

$$\frac{f_s}{\left(p - \frac{1}{2}\right)\left(1 + \frac{1}{2p}\right)} = \frac{f_s}{p - \frac{1}{4p}},$$

very near the desired frequency f_s/p. This trick, like the 1-2-1 weighting algorithm, allows easier tuning of consonant intervals.

The circular-buffer technique uses two pointers into an area of memory at least as large as p. The pointers are separated by p. The value is read from the position pointed to by the trailing pointer, output, then copied to the position pointed to by the leading pointer. Both pointers are then incremented around the buffer (with the first position coming immediately after the last one). Clever choice of the position and size of the buffer often allows the pointer wraparound to be done with no extra instructions. Multiple voices can be done by having several buffers with pointer pairs. If indexed addressing is used, the voices can share a common leading pointer, with different base addresses for the different voices. Alternatively, for two voices, three pointers can be used in a single larger buffer, with the middle pointer used as a trailing pointer for the first voice, and a leading pointer for the second voice. For a given amount of memory this allows a larger value of p for one voice, as long as the other voice has a small value of p.

Slurring works better with the circular-buffer technique than with the decreasing-counter technique. Increases in p merely tap more of the previous samples, instead of tapping undefined (though probably usable) values past the end of the table. With the circular buffer, slurring to a subharmonic

(new p a multiple of its orginal value) is essentially the same as the harmonic trick, since no energy is introduced at the new fundamental. To slur a subharmonic, the algorithm first slurs to an intermediate note, waits about a period, and then slurs to the final pitch.

Average sampling rate can be increased by taking as much code as possible out of the innermost loop. For example, the harmonic trick is faster than decay stretching, because it only takes extra time during wavetable loading (between notes), not during notes.

Most software implementations require a timing counter to determine when to stop a note and read in new parameters. This counter could be decremented and tested on every sample to get very precise timing control. An alternative method is to subtract the buffer size every time the pointer wraps around. In the decreasing-counter technique, this is particularly attractive because there is plenty of spare time during the wraparound.

Using small word sizes (like 8 bits) makes round-off error a serious problem. In the algorithms described in this paper, round-off error is not random but rather a consistent rounding down of the samples. This effect significantly reduces the decay time of the fundamental frequency (compared to the theoretical decay time or to the decay time when the algorithm is computed with much larger word sizes). The effect can almost be eliminated by randomly adding 0 or 1 to $Y_{t-p} + Y_{t-p-1}$ before dividing by 2. This *dither* technique lengthens the final decay of the fundamental roughly back to its theoretical decay time, without appreciably lengthening the initial attack of the tone.

Since the round-off error is consistently in the same direction, it introduces a dc drift to the decay. This is not serious, because the algorithm is guaranteed not to cause arithmetic overflow (the usual danger with dc drift). Dithering reduces the drift considerably by converting it to a random walk, but does not entirely eliminate it. The dc component can cause clicks if a voice is silenced by being set to some constant value (such as 0). In a one-voice implementation, the simplest way to silence a voice without clicks is to stop sending new values to the DAC, letting it remain at the last value it received. Another way to silence a note is to change p to a very small value (such as 2), producing almost instantaneous decay.

Analysis of the Plucked-String Algorithm

What makes the simple plucked-string algorithm sound so realistic? How can we predict how long notes will last? To answer these questions, we have to look at the decay of the overtones. From listening to the output of the algorithm, it is clear that the higher harmonics die very quickly, while the fundamental and the lower harmonics last a long time. Low notes last much longer than high notes.

It would be interesting to compare the theoretical analysis of our synthesis technique with an existing analysis of a guitar, lute, mandolin, or other string instrument. Unfortunately, we could not find a published analysis and did not have the tools to perform our own analysis. Some previous work has been done using physical models for synthesizing string sounds (Hiller and Ruiz 1971), but these models do not help to explain the high quality of the sound produced with our technique.

Before plunging into the mathematics, it's worth taking an informal look at what is happening to the harmonics. Essentially, one pass (p samples) takes what is in the wavetable and averages it with another copy delayed by one sample time. For sinusoids with long periods, one sample time is a very small phase difference, while for short periods it is a large difference. Averaging two sinusoids with a small phase difference decreases the amplitude slightly, while averaging two with a large phase difference (up to half a period) causes much more cancellation. Since the phase difference results from a time difference of one sample, it is always less than half a period for frequencies up to the Nyquist frequency.

The informal argument can be made more explicit if we give estimates of the decay rates of the harmonics (Jaffe and Smith 1983). However, by borrowing some techniques from digital-filter design, we can compute both the decay rates and the frequencies of the overtones accurately and see how good the simpler approximations are. If we view the

algorithm as a digital filter with no input, the overtones and their decay rates correspond to poles in the z-transform of the impulse response of the filter. More information on z-transform techniques can be found in the literature (Moore 1978; Antoniou 1979). The decay time of a partial is inversely proportional to the log of the magnitude of the corresponding pole, and the frequency is proportional to the argument of the pole. Writing the location of pole in polar form as $ae^{i\omega}$ allows us to write the frequency and decay-time constant of the corresponding overtone:

$$f = \frac{f_s \omega}{2\pi}$$

$$D = \frac{-1}{f_s \ln a}.$$

The decay time is the time it takes for the partial to decay to $1/e$ of its initial amplitude. In audio work, the time it takes to decay 60 db is often used instead. To get the 60-db decay time, we multiply D by $\ln 1000 \approx 6.908$.

We can obtain the z-transform of the impulse response from the recurrence relations by using the standard techniques for digital filter analysis:

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

The poles of the z-transform are the roots of $2z^{p+1} - z - 1 = 0$. There is only one zero at $z = -1$ (corresponding to the Nyquist frequency $\frac{1}{2}f_s$). The roots are easy to approximate if we rearrange the equation to be $2z^{p+1/2} = z^{1/2} + z^{-1/2}$. Replacing z by $ae^{i\omega}$ gives us

$$\begin{aligned} 2a^{p+1/2}e^{i\omega(p+1/2)} &= a^{1/2}e^{i(\omega/2)} + a^{-1/2}e^{-i(\omega/2)} \\ &= \sqrt{a + a^{-1} + 2\cos\omega}\cdot e^{i\theta} \end{aligned}$$

for some angle θ. Looking at just the imaginary parts of the right-hand sides, we get

$$\sqrt{a + a^{-1} + 2\cos\omega}\,\sin\theta = (a^{1/2} - a^{-1/2})\sin\frac{\omega}{2}.$$

Approximations are easily found if we assume that a is only slightly less than 1. This assumption is easily verified numerically. If $a = 1 - \varepsilon$, then

$$a + a^{-1} = 2 + \varepsilon^2 + \varepsilon^3 + \ldots \approx 2$$

$$a^{1/2} - a^{-1/2} = -\varepsilon^2/8 - \ldots \approx 0.$$

This gives us that $\sin\theta \approx 0$, so $e^{i\omega(p+1/2)} = e^{i\theta} \approx 1$, which we can solve for ω to get a first approximation of the frequency of the n^{th} partial:

$$\omega = \frac{\theta + 2\pi n}{p + \frac{1}{2}} \approx \frac{2\pi n}{p + \frac{1}{2}}$$

Since $\sqrt{2 + 2\cos\omega} = 2\cos\omega/2$, the magnitude of the pole is $2a^{p+1/2} \approx 2\cos\omega/2$. Solving for a, we get an approximation for the decay time of the n^{th} harmonic:

$$a \approx \left(\cos\frac{\omega}{2}\right)^{1/(p+1/2)} = \left(\cos\frac{2\pi n}{2p+1}\right)^{1/(p+1/2)}$$

$$D \approx \frac{-1}{f_s \ln a} = \frac{p + \frac{1}{2}}{-f_s \ln\cos\frac{2\pi n}{2p+1}}.$$

These estimates are the same as the ones obtained by less formal analysis. For large p and small n, they are quite accurate (for $p = 240$ and $n = 1$ the magnitude and frequency estimates are both correct to seven significant figures).

More accuracy can be obtained by using these estimates as starting values for iterative improvement. The simplest technique is to use a few iterations of Newton's method to improve the approximations for the poles:

$$z \leftarrow \frac{2pz^{p+1} + 1}{2(p+1)z^p - 1}.$$

For a more intuitive grasp of the relationships between p, n, and decay time, it is worthwhile to expand the decay-time estimate in powers of n:

$$D \approx \frac{1}{f_s}\left(\frac{(2p+1)^3}{4\pi^2 n^2} - \frac{2p+1}{6} - \frac{\pi^2 n^2}{15(2p+1)} - \ldots\right).$$

Roughly speaking, decay time increases as p^3, and

decreases as n^{-2}. Appendix 1 provides a tabulation of decay-time constant, estimated-time constant, and the first term of the power series for various values of p and n. For most purposes, the first term of the power series is an accurate enough estimate of the decay time.

The frequency estimates for the overtones are purely harmonic. It is interesting to examine how much inharmonicity is actually present. This requires estimating θ somewhat more accurately. If we replace $a + a^{-1}$ by 2 as before, replace $a^{1/2}$ by $(\cos \omega/2)^{1/(2p+1)}$, and rearrange the equation involving $\sin \theta$, we get

$$\theta \approx \sin^{-1}\left(\frac{1}{2}\tan\frac{\omega}{2}\left(\left(\cos\frac{\omega}{2}\right)^{1/(2p+1)} - \left(\cos\frac{\omega}{2}\right)^{-1/(2p+1)}\right)\right).$$

Expanding by powers of ω gives

$$\theta \approx \frac{\omega^3}{16(2p+1)} - \frac{\omega^5}{128(2p+1)} + \cdots .$$

Plugging in our previous approximation for ω, computing θ, then recomputing ω gives us

$$\omega \approx \frac{4\pi n}{2p+1} - \frac{8\pi^3 n^3}{(2p+1)^5} - \cdots .$$

Appendix 2 tabulates the frequency of the pure harmonic, the improved estimate of ω, the frequency obtained by using Newton's method to refine the estimate of the pole, and the inharmonicity of the overtone. It can be seen that (except for small values of p) the overtones are almost pure harmonics.

The frequencies and decay times for the 1-2-1 weighting algorithm can be analyzed in a similar fashion. The partials are pure harmonics, and the decay times are about half the decay times for corresponding partials obtained using the basic algorithm.

Analysis of the Drum Algorithm

Since the drum algorithm produces aperiodic signals, we need to use different tools to analyze it. Instead of amplitudes of overtones, let's look at the rms amplitude. *Root-mean-square amplitude* is the square root of the expected value of the square of the amplitude $= \sqrt{E(Y_t^2)}$. Squaring the recurrence relation for the drum algorithm yields

$$E(Y_t^2) = \frac{1}{4}E(Y_{t-p}^2) + \frac{1}{4}E(Y_{t-p-1}^2) + \frac{1}{2}E(Y_{t-p}Y_{t-p-1}).$$

Since Y_{t-p} and Y_{t-p-1} have independent signs, $E(Y_{t-p}Y_{t-p-1}) = 0$. This reduces the recurrence relation to

$$E(Y_t^2) = \frac{1}{4}(E(Y_{t-p}^2) + E(Y_{t-p-1}^2)).$$

Note that for $-p < t \leq 0$, $E(Y_t^2) = A^2$. If we make the simplifying assumption that $E(Y_{t-p}^2)$ is approximately $E(Y_{t-p-1}^2)$, we get

$$E(Y_t^2) \approx \frac{1}{2}E(Y_{t-p}^2) = 2^{-\lceil t/p \rceil}A^2.$$

We can improve this estimate somewhat by changing our assumption that $E(Y_{t-p}^2)$ is nearly the same as $E(Y_{t-p-1}^2)$. If we instead assume that $E(Y_t^2)$ decays exponentially, we can express $E(Y_t^2)$ in the form $2^{\alpha t}A^2$. To compute α, we plug into the previous recurrence relation, getting

$$2^{\alpha t}A^2 = \frac{1}{4}(2^{\alpha(t-p)}A^2 + 2^{\alpha(t-p-1)}A^2).$$

This can be simplified to

$$2^{\alpha p} = \frac{1}{4}(1 + 2^{-\alpha}).$$

Since $1 + 2^{-\alpha} \approx 2(2)^{-\alpha/2}$, we can conclude that

$$E(Y_t^2) \approx 2^{-2t/(2p+1)}A^2$$

and

$$\text{rms } Y_t \approx 2^{-t/(2p+1)}A.$$

This is not a complete analysis of the drum algorithm, since the absolute values of successive values are correlated, as are the absolute values of samples p apart. Experiments still need to be done to determine whether there is a perceptual difference between the drum algorithm and a Gaussian noise source with an exponentially decaying envelope.

Conclusions and Future Research

We have developed simple but powerful algorithms that can be implemented on a variety of different processors and synthesizers. They allow programmers and musicians to experiment with computer music using inexpensive equipment. The algorithms do not have the versatility of FM synthesis or additive synthesis, but provide surprisingly rich timbres. Readers interested in commercial application of these algorithms should contact the Office of Technology Licensing at Stanford University about licensing agreements.

In addition to the problems mentioned in the main body of the paper, there are still a lot of questions that need answering. For example, what sounds can be achieved by using the guitar decay algorithm as a digital filter to produce a tuned reverberator? Or by cross-coupling two strings at slightly different pitches? What about more complicated modifiers in the feedback loop (two delay-and-mix units instead of one, allpass filters, and so on)? What about locking together two voices (same initial load) with opposite amplitudes so that they cancel each other, then letting them drift apart by using independent probabilistic decay stretching? Some of these problems are examined by Jaffe and Smith (1983); others are being investigated with the Digitar chip.

Those interested in mathematical analyses could probably improve the current analyses a bit, and other variants have yet to be analyzed. A good approximation for the poles of the general two-point recurrence would be particularly welcome. It would also be interesting to convert the recurrence relations to differential equations, and to assign a physical interpretation. Techniques for taking the z-transform of a probabilistic algorithm are needed to perform a proper analysis of decay stretching.

References

Antoniou, A. 1979. *Digital Filters: Analysis and Design.* New York: McGraw-Hill.

Hiller, L., and P. Ruiz. 1971. "Synthesizing Musical Sounds by Solving the Wave Equation for Vibrating Objects." *Journal of the Audio Engineering Society* Part 1: 19(6):462–470; Part 2: 19(7):542–551.

Jaffe, D., and J. Smith. 1983. "Extensions of the Karplus-Strong Plucked-String Algorithm." *Computer Music Journal* 7(2):56–69.

James, D. 1978. "Real Time Synthesis Using High Speed Computer Networks." Ph.D. thesis, Massachusetts Institute of Technology.

Karplus, K., and A. Strong. 1983. "Implementations of the Digitar Algorithms." Unpublished manuscript.

Knuth, D. 1981. *The Art of Computer Programming: Volume 2, Seminumerical Algorithms.* 2nd ed. Reading, Massachusetts: Addison-Wesley.

Moore, F. R. 1978. "An Introduction to the Mathematics of Digital Signal Processing, Part II: Sampling, Transforms, and Digital Filtering." *Computer Music Journal* 2(2):38–60.

Sleator, D. 1981. Private communication.

Appendix 1: Decay-Time Constants

The decay time is the time (in seconds, sampling rate = 20 KHz) it takes for harmonic n to decay to $1/e$ of its initial amplitude, using the plucked-string algorithm (with periodicity parameter p). The first column in the table is a crude estimate of decay time, calculated from the first term of the power-series expansion (see the text):

$$\frac{(2p+1)^3}{4\pi^2 n^2 f_s}.$$

The second column is the estimate from which the power series was derived:

$$\frac{p+\frac{1}{2}}{-f_s \ln \cos \frac{2\pi n}{2p+1}}.$$

The third column is computed from the poles of the z-transform for the plucked-string algorithm (the roots of $2z^{p+1} - z - 1$). To compute the table, the poles were estimated by

$$\left(\cos \frac{2\pi n}{2p+1}\right)^{2/(2p+1)} e^{4\pi i n/(2p+1)}.$$

The estimates were then refined by iterating Newton's method until successive approximations differed by less than 10^{-6}. The log magnitude of the pole was divided by f_s to get the table entry.

p	n	Crude estimate	Better estimate	Refined estimate
2	1	.0002	.0001	.0001
3	1	.0004	.0004	.0004
4	1	.0009	.0008	.0009
4	2	.0002	.0001	.0002
30	1	.2875	.2870	.2869
30	2	.0719	.0714	.0714
30	3	.0319	.0314	.0314
60	1	2.2437	2.2427	2.2416
60	2	.5609	.5599	.5599
60	3	.2493	.2483	.2483
60	4	.1402	.1392	.1392
120	1	17.7281	17.7261	17.8847
120	2	4.4320	4.4300	4.4326
120	3	1.9698	1.9678	1.9682
240	1	140.9436	140.9396	146.4811
240	2	35.2359	35.2319	34.8357
240	3	15.6604	15.6564	15.6683
240	4	8.8090	8.8050	8.7606
240	8	2.2022	2.1982	2.2016
240	15	.6264	.6224	.6225
240	30	.1566	.1525	.1525
240	60	.0392	.0349	.0349
240	120	.0098	.0021	.0027
480	1	1124.0365	1124.0285	1187.5162
480	2	281.0091	281.0011	298.7312
480	3	124.8929	124.8849	134.1789
480	4	70.2523	70.2443	67.3967
480	8	17.5631	17.5551	17.5233

Appendix 2: Frequencies of Overtones

The table in this appendix gives frequencies (in hertz, with a sampling rate of 20 KHz) for various overtones of the plucked-string algorithm. The first column is the simple harmonics of $1/(p + \frac{1}{2})$; the second is the improved estimate, including the n^3 correction term; the third is from Newton's method, using the second estimate as a starting point; the fourth is the deviation of the actual frequency from the pure harmonic (in cents). (Note: n is the number of the harmonic, and p is the periodicity parameter.)

p	n	Harmonic	Corrected	Actual	Inharmonicity
2	1	8000.000	7747.338	7500.000	−111.7313
3	1	5714.286	5667.307	5642.389	−21.9205
4	1	4444.444	4431.073	4427.170	−6.7421
4	2	8888.889	8781.918	8507.279	−75.9665
30	1	655.738	655.737	655.737	−.0025
30	2	1311.475	1311.468	1311.468	−.0101
30	3	1967.213	1967.188	1967.187	−.0233
60	1	330.579	330.578	330.578	−.0002
60	2	661.157	661.157	661.157	−.0006
60	3	991.736	991.735	991.735	−.0015
60	4	1322.314	1322.312	1322.312	−.0026
120	1	165.975	165.975	165.975	.0000
120	2	331.950	331.950	331.950	.0000
120	3	497.925	497.925	497.925	−.0001
240	1	83.160	83.160	83.160	.0000
240	2	166.320	166.320	166.320	.0000
240	3	249.480	249.480	249.480	.0000
240	4	332.640	332.640	332.640	.0000
240	8	665.281	665.281	665.281	.0000
240	15	1247.401	1247.401	1247.401	−.0001
240	30	2494.802	2494.802	2494.802	−.0006
240	60	4989.605	4989.598	4989.596	−.0033
240	120	9979.210	9979.157	9965.363	−2.4038
480	1	41.623	41.623	41.623	.0000
480	2	83.247	83.247	83.247	.0000
480	3	124.870	124.870	124.870	.0000
480	4	166.493	166.493	166.493	.0000
480	8	332.986	332.986	332.986	.0000