# 2

# *Delay Line Effects*

Delay lines are the fundamental building blocks of many of the most important effects. They are rather easy to implement, and only small changes in how they are used allow many different audio effects to easily be constructed. In this chapter, we look at some common effects that are built using delay lines.

## Delay

Delay is a simple effect with powerful applications. In the simplest case, adding a single delayed copy of a sound to itself can enliven an instrument's sound in a mix or, at longer delay times, allow a performer to play a duet with himself or herself. Many familiar effects, including chorus, flanging, vibrato, and reverb, are also built on delays.

### Theory

#### *Basic Delay*

The basic delay plays back an audio signal after a specified *delay time*. Depending on the application, the delay time might range from a few milliseconds to several seconds or longer. Figure 2.1 shows a block diagram of the basic delay. It is common to mix the delayed output with the original input, thereby producing two copies of the sound. For this reason the basic delay is also sometimes known as an *echo* effect (though as we will see, the perception of echo also depends on the delay time).

We can express the output audio samples $y[n]$ as a function of the input samples $x[n]$, the delay time $N$ (expressed in samples), and the gain $g$ of the delayed signal:

$$y[n] = x[n] + gx[n - N] \tag{2.1}$$

Delay is a *linear, time-invariant* effect. To show linearity, consider two signals $x_1[n]$ and $x_2[n]$. Let $y_1[n]$ and $y_2[n]$ be the delayed output of each signal

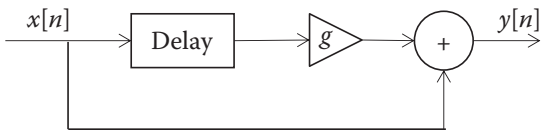


**FIGURE 2.1**
Diagram of the basic delay unit, or an echo device. The box marked "delay" is commonly known as a delay line.

individually. If we delay the sum of the signals, we find that the output is the sum of the individual delayed signals:

$$\begin{aligned} y[n] &= \big(x_1[n] + x_2[n]\big) + g\big(x_1[n-N] + x_2[n-N]\big) \\ &= \big(x_1[n] + gx_1[n-N]\big) + \big(x_2[n] + gx_2[n-N]\big) = y_1[n] + y_2[n] \end{aligned} \tag{2.2}$$

Time invariance implies that shifting the input in time produces an identical shift in the output. Consider taking the delay of $x_d[n] = x[n - M]$ for some number of samples $M$:

$$y_d[n] = x[n-M] + gx[n-M-N] = y[n-M] \tag{2.3}$$

Using the $Z$ transform, we can also find the frequency response of the basic delay:

$$Y(z) = X(z) + gz^{-N}X(z) \qquad H(z) = \frac{Y(z)}{X(z)} = 1 + gz^{-N} = \frac{z^N + g}{z^N} \tag{2.4}$$

Since the transfer function $H(z)$ has no poles outside the unit circle, the basic delay must be *stable* in all cases; i.e., a bounded input will always produce a bounded output.

### *Delay with Feedback*

The simple "feedforward" delay is limited in application, producing only a single echo. Most audio delay units also have a feedback control (sometimes called regeneration), which sends a scaled copy of the delay output back to the input, as shown in Figure 2.2. Feedback causes the sound to repeat continuously, and assuming a feedback gain less than 1, the echoes will become quieter each time. Though the echoes are theoretically repeated forever, they will eventually become so quiet as to be below the ambient noise in the system and thus be inaudible.

To find the time domain difference equation for delay with feedback, it helps to consider the signal $d[n]$ at the output of the delay line:

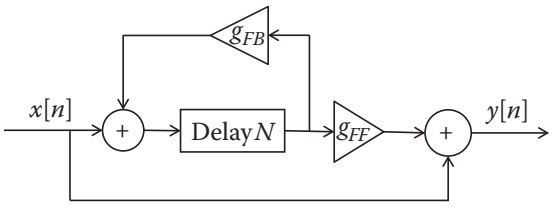


**FIGURE 2.2**
Diagram of the basic delay unit with feedback.

$$y[n] = x[n] + g_{FF}d[n] \quad \text{where} \quad d[n] = x[n-N] + g_{FB}d[n-N] \tag{2.5}$$

The form of $d[n]$ is similar to the delayed output signal $y[n-N]$, which lets us substitute and write $y[n]$ directly in terms of $x[n]$:

$$\begin{aligned} y[n-N] &= x[n-N] + g_{FF}d[n-N] \\ d[n] &= \frac{g_{FB}}{g_{FF}}y[n-N] + \left(1 - \frac{g_{FB}}{g_{FF}}\right)x[n-N] \\ y[n] &= g_{FB}y[n-N] + x[n] + (g_{FF} - g_{FB})x[n-N] \end{aligned} \tag{2.6}$$

Taking the $Z$ transform, we can find the frequency domain transfer function:

$$\begin{aligned} Y(z) - g_{FB}z^{-N}Y(z) &= X(Z) + (g_{FF} - g_{FB})z^{-N}X(z) \\ H(z) = \frac{Y(z)}{X(z)} &= \frac{1 + z^{-N}(g_{FF} - g_{FB})}{1 - z^{-N}g_{FB}} = \frac{z^N + g_{FF} - g_{FB}}{z^N - g_{FB}} \end{aligned} \tag{2.7}$$

Thus, the system has poles at the $N$ complex roots of $g_{FB}$. As with the basic delay, delay with feedback is linear and time invariant. The above transfer function implies the effect will be stable whenever the poles are inside the unit circle, that is, when $|g_{FB}| < 1$. This result aligns with intuition, in that only when the feedback gain is less than 1 will the echoes grow softer over time.

## Other Delay Types

### *Slapback Delay*

A *slapback delay* is identical to a basic delay without feedback and with a relatively short delay time, typically between 60 and 150 ms. The delayed copy is perceived as a separate sound that appears immediately after the original sound. A longer delay, where there is a noticeable gap between the original and delayed sounds, is often referred to as an *echo*.

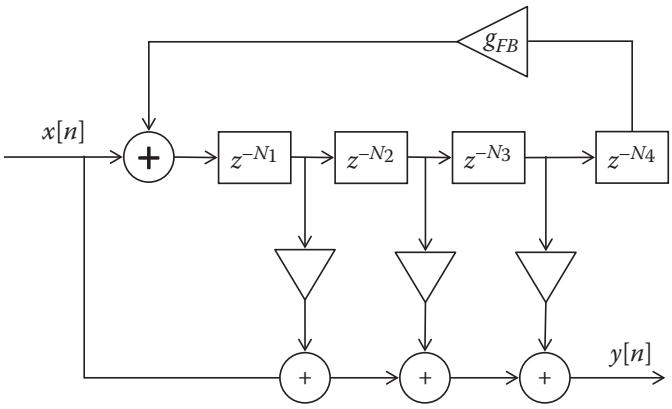


**FIGURE 2.3**
Flow diagram of a three-tap delay. If the last delay value is zero, and only the third tap is used, the system is equivalent to the basic delay.

### *Multitap Delay*

In a standard delay with or without feedback, the time between copies is always the same, since the output signal is taken after the signal reaches the end of the delay line. Multitap delay provides more flexibility by taking several additional outputs in the middle of the delay line, where the signal has been delayed only part of the total time. This process is known as "tapping" the delay line, following the analogy of adding taps along a water pipe to get water at various locations. Multitap delay is commonly labeled according to the number of taps; for example, a four-tap delay would have four total outputs at various points on the delay line. The delay between each tap is typically not the same, so multitap delays allow more complex patterns to be created that can add interesting rhythmic qualities to an instrument. A diagram of multitap delay is shown in Figure 2.3.

The multitap delay is a more general case of the basic delay design. The multitap delay can be further generalized by allowing feedback from the tap outputs to the beginning of the delay line as well. In this case, care must be taken with the feedback gains to avoid creating an unstable system.

### *Ping-Pong Delay*

Ping-pong delay is a multichannel delay-based effect that produces a bouncing sound from one channel to the other, hence the name. It is implemented as a delay with feedback with at least two distinct delay lines (Figure 2.4). Each delay line may be driven by a separate input, or only one input can be used. The output of each delay line, rather than feeding back to itself, attaches to the input of the opposite delay line. In its two-channel configuration, ping-pong delay produces a sound that bounces between left and right channels in a stereo track.

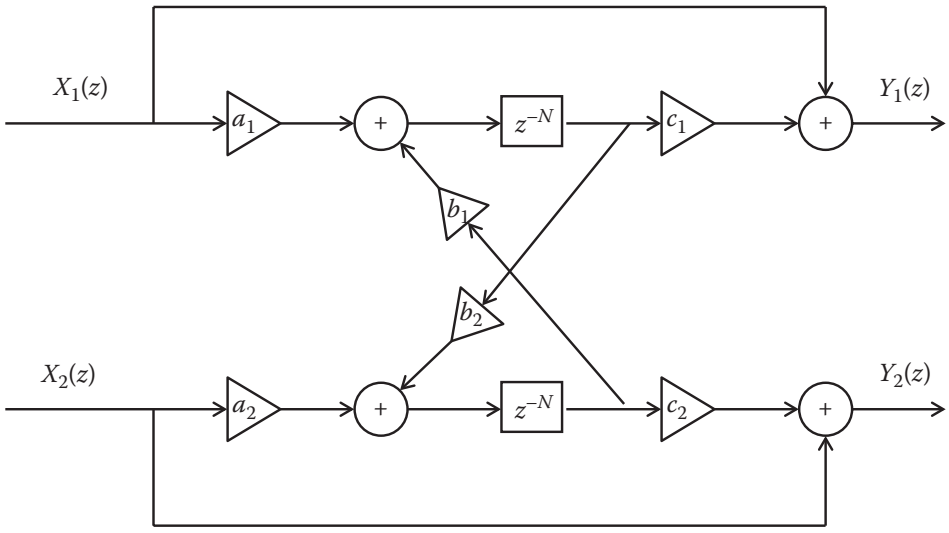


**FIGURE 2.4**
Flow diagram of a ping-pong delay unit.

## Implementation

### *Basic Delay*

Before the days of digital audio, delays were one of the more cumbersome effects to implement, relying on physical audiotape or analog "bucket brigade device" integrated circuits to store and retrieve audio signals. In the digital realm, delay is one of the simplest effects. Audio samples are stored in a preallocated memory *buffer* as they arrive, while previously stored samples are read from the buffer once the delay time has elapsed. In other words, in a simple delay, each sampling period includes one read operation (retrieving the delayed signal) and one write operation (storing the current signal). When the end of the memory buffer is reached, the system should loop around to the beginning of the buffer. In signal processing, this process is known as a *circular buffer*, and it is quite efficient. Programming considerations for circular buffers are discussed in Chapter 13.

### *Variations*

Delay with feedback is implemented identically to a simple delay, but rather than storing the raw input signal in the memory buffer, the sum of the input signal and the delayed, scaled output is stored ($x[n] + g_{FB}x[n - N]$ in Figure 2.2). Ping-pong delay uses two independent memory buffers and, hence, two read operations and two write operations for each sample.

Multitap delay uses a single memory buffer with multiple *read pointers*. At each sample, the current input is written into a slot in the buffer (the *write pointer*), but samples are read back from multiple locations in the buffer

(the read pointers). The spacing in memory between the pointers determines the difference in their delay times. This process is further discussed in Chapters 10 and 13.

### *Delay Line Interpolation*

Digital delays are implemented using a memory buffer of discrete audio samples. To change the delay time, we change the distance in the buffer between where samples are written and where they are read back. However, many effects, including the flanger and chorus, require a delay that changes over time. To achieve a smooth variation in delay, rounding to the nearest-integer number of samples is usually not good enough.

Delay becomes more complex when a noninteger number of samples is required. Consider a simple delay with a length of 0.5 samples. From Equation (2.1) we have

$$y[n] = x[n] + gx[n - 0.5] \Rightarrow y[1] = x[1] + gx[0.5] \tag{2.8}$$

However, $x[n]$ is defined only for integer $n$, so $x[0.5]$ does not exist. Strictly speaking, it is not correct to think of $x[0.5]$ as "halfway between" $x[0]$ and $x[1]$. However we might ask what the result would be if $x[n]$ were converted from discrete to continuous time, shifted half a sampling period, then reconverted to discrete time. An excellent discussion of the underlying mathematics of this process, which is based on sinc functions $(sin(n)/n)$, can be found in [3]. Interestingly, exact reconstruction of fractional sample values requires knowledge of the entire signal (i.e., $x[n]$ from $n = -\infty$ to $\infty$), which is clearly impossible in a real-time audio context.

In practice, fractional delays in audio are implemented using interpolation or allpass filters [7]. Interpolation involves using a weighted combination of surrounding samples to approximate the fractional sample value. Interpolation involves estimating a value of a continuous function, given discrete points, and can be used to estimate values between points on a delay line. Polynomial interpolation is where the function is estimated to be an $N$th-order polynomial, $x(t) = c_N t^N + c_{N-1} t^{N-1} + \ldots c_1 t^1 + c_0$.

If values out of the delay line closest to the required point are read, this is zeroth-order, or nearest-neighbor, interpolation. It is probably the least computationally expensive approach. However, the output now has abrupt jumps between values. The quality of this approach is quite poor. Clicking in the output may be heard as the delay length changes, also known as *zipper noise*.

Linear interpolation, or first-order interpolation, is implemented by connecting two known samples by a straight line and then reading the desired value from that line. This is given in the following equation:

$$x(t) = (n + 1 - t)x[n] + (t - n)x[n + 1],\ n \le t < n + 1 \tag{2.9}$$

Linear interpolation is simple to calculate and produces much better results than nearest-neighbor interpolation. However, it is still only a rough approximation to the ideal continuous time case, and it can introduce noise and aliasing into the signal. In many cases, audibly better quality will be obtained with a more computationally complex interpolation method.

One such method is second-order polynomial interpolation. Consider three successive samples, $x[n-1]$, $x[n]$, and $x[n+1]$. We would typically use these samples if we are trying to interpolate a value of $x$ near $x[n]$. Let's define a new function, $y(\tau) = x(t)$, where $\tau = t - n$. Thus, $x[n-1]$, $x[n]$, and $x[n+1]$ become $y[-1]$, $y[0]$, and $y[1]$. Assuming that these points are on a second-order polynomial, we have the following three equations:

$$\begin{aligned} y[-1] &= c_2(0-1)^2 + c_1(0-1) + c_0 \\ y[0] &= c_2 0^2 + c_1 0 + c_0 \\ y[1] &= c_2(0+1)^2 + c_1(0+1) + c_0 \end{aligned} \tag{2.10}$$

where $c_0$, $c_1$, and $c_2$ are the coefficients of some second-order polynomial. This can be solved to give

$$\begin{aligned} c_0 &= y[0] \\ c_1 &= \big(y[1] - y[-1]\big)/2 \\ c_2 &= \big(y[1] - 2y[0] + y[-1]\big)/2 \end{aligned} \tag{2.11}$$

So our interpolated values are

$$\begin{aligned} x(t) = y(\tau) &= c_2\tau^2 + c_1\tau + c_0 \\ &= \big(x[n+1] - 2x[n] + x[n-1]\big)(t-n)^2/2 + \big(x[n+1] - x[n-1]\big)(t-n)/2 + x[n] \\ &= \frac{(t-n-1)(t-n)x[n-1] - 2(t-n-1)(t-n+1)x[n] + (t-n)(t-n+1)x[n+1]}{2} \end{aligned} \tag{2.12}$$

Cubic interpolation requires more computation, but it can give better results with lower added noise resulting from inaccuracies than the ideal sinc-function case. There are many forms of cubic interpolation; a detailed discussion is available in [8]. The simplest case uses the four samples surrounding the interpolated location:

$$x(t) = c_3(t-n)^3 + c_2(t-n)^2 + c_1(t-n) + c_0 \quad \text{for} \quad n \le t < n+1 \tag{2.13}$$

where the coefficients are given by

$$\begin{aligned} c_3 &= -x[n-1] + x[n] - x[n+1] + x[n+2] \\ c_2 &= x[n-1] - x[n] - a_0 \\ c_1 &= x[n+1] - x[n-1] \\ c_0 &= x[n] \end{aligned} \tag{2.14}$$

### *Code Example*

The following C++ code fragment, adapted from the code that accompanies this book, shows the implementation of a basic delay with feedback, and without interpolation.

```
// Variables whose values are set externally:
int numSamples;     // How many audio samples to process
float *channelData; // Array of samples, length numSamples
float *delayData;   // Our own circular buffer of samples
int delayBufLength; // Length of our delay buffer in samples
int dpr, dpw;       // Read/write pointers into delay buffer

// User-adjustable effect parameters:
float dryMix_;      // Level of the dry (undelayed) signal
float wetMix_;      // Level of the wet (delayed) signal
float feedback_;    // Feedback level (0 if no feedback)

for (int i = 0; i < numSamples; ++i)
{
    const float in = channelData[i];
    float out = 0.0;

    // The output is the input plus the contents of the
    // delay buffer (weighted by the mix levels).

    out = (dryMix_ * in + wetMix_ * delayData[dpr]);

    // Store the current information in the delay buffer.
    // delayData[dpr] is the delay sample we just read, i.e.
    // what came out of the buffer. delayData[dpw] is what
    // we write to the buffer, i.e. what goes in

    delayData[dpw] = in + (delayData[dpr] * feedback_);

    if (++dpr >= delayBufLength)
        dpr = 0;
```

```
    if (++dpw >= delayBufLength)
        dpw = 0;

    // Store output sample in buffer, replacing the input
    channelData[i] = out;
}
```

This example assumes that a single channel of input audio data is present in the `channelData` array. In a real-time effect, `channelData` will hold only the most recent block of audio samples, rather than the entire input signal. To implement the delay, we write each input sample into `delayData` at a position indicated by the write pointer `dpw`, while reading a delayed sample previously written into the buffer using read pointer `dpr`.

`delayData` is a circular buffer: after each sample is processed, `dpr` and `dpw` are incremented, and when either of them reaches the end of the buffer, it is reset back to position 0. Notice that there is no parameter for the delay length in this code example. It is the difference between the read and write pointers `dpr` and `dpw` that determines the delay length. These values are initialized elsewhere in the effect. More details and a more extensive code example can be found in Chapter 13.

### Applications

Delays are a very common effect in music production. Even a single basic delay, without feedback, has many uses. A common use is to combine an instrument's sound with a short echo (for example, around 50–100 ms) to create a doubling effect. This can create a wider, more lively sound than the original single version. If the original and delayed copies are panned differently in a stereo mix, short delays can also help make the mix sound "bigger." More complex arrangements including feedback and multiple taps can start to simulate the sound of a reverb unit, though reverb (covered in Chapter 11) typically creates more complex patterns than can be simulated with simple delays.

Longer delays become less subtle once the original and delayed copies are easily perceived as two separate acoustic events. One common trick is to synchronize the delay time with the tempo of the music, such that the delayed copies appear in rhythm with the track [9]. If the delay time is especially long, for example, equal to one or more bars of music, a musician can play over himself or herself and develop elaborate harmonies and textures.

Sampler and looper pedals are fundamentally based on the delay, with additional features to be able to define the start and end of an audio segment, which then can loop continuously. Other features are possible, including the ability to mix additional sounds onto a loop that was previously recorded, to play a loop backwards, or to have multiple simultaneous loops. Some

**FRIPPERTRONICS AND CRAFTY GUITARISTS**

Robert Fripp (King Crimson, The League of Crafty Guitarists, League of Gentlemen, solo artist…) is known as one of the greatest and most influential guitarists of all time. Evolving out of his work with Brian Eno in 1973, he devised a tape looping technique to layer his guitar sounds in real-time. It used two reel-to-reel tape recorders. The tape traveled from the supply reel of one recorder to the take-up reel of the second one. Then the tape from the second machine is fed back to the first one, and the delay can be changed by adjusting the distance between the two machines. Furthermore, it also provided a recording of the complete overlayed recording, and could be used in live performance. Fripp's girlfriend later named this technique 'Frippertronics,' though we would describe it as a time varying delay with feedback. Also among Robert Fripp's more unusual contributions are many of the sounds for the Windows Vista operating system.

Many other famous guitarists are also known for music technology innovations. For instance, Tom Scholz of the band Boston designed a wide range of novel guitar effects devices, including the Rockman amplifier. But one of the most famous guitarists is also one of the people who most influenced music technology, Les Paul. His solid body electric guitar designs were some of the first and most popular, and he is credited with many innovations in multitrack recording.

standard delay pedals will include basic looper capability, though often limited to a single loop of a few seconds at most.

## Vibrato Simulation

*Vibrato* is defined as a small, quasi-periodic variation in the pitch of a tone. Traditionally, vibrato is not an audio effect but rather a technique used by singers and instrumentalists. On the violin, for example, vibrato is produced by rhythmically rocking the finger back and forth on the fingerboard, slightly changing the length of the string. However, vibrato can be added to any audio signal through the use of *modulated delay lines*.

Vibrato is characterized by its *frequency* (how often the pitch changes) and *width* (total amount of pitch variation). On acoustic instruments, especially wind instruments, vibrato is often accompanied by some degree of amplitude modulation or tremolo (Chapter 5), with the pitch and amplitude of the signal changing in synchrony.

## Theory

The vibrato effect works by changing the playback speed of the sampled audio. The effect of playback speed is familiar to many listeners: playing a sound faster raises its pitch, and playing it slower lowers the pitch. To add a vibrato, then, the playback speed needs to be periodically varied to be faster or slower than normal.

Implementation of the vibrato effect is based on a modulated delay line, a delay line whose delay length changes over time under the control of a *low-frequency oscillator* (LFO), as shown in Figure 2.5. Unlike other delay effects, the input signal $x[n]$ is not mixed into the output.

Delay alone does not introduce a pitch shift. Suppose the length $M[n]$ of the delay line does not change. Then at every sampling period $n$, exactly one sample $x[n]$ goes in and one sample $y[n - M]$ comes out. The sound will be delayed but otherwise identical to the original. But now suppose that the length $M[n]$ decreases each sample by an amount $\Delta m$:

$$M[n] = M_{\max} - (\ \ m)n \tag{2.15}$$

Examining the output of the delay line, we can see that each new input sample $n$ moves the output by $(1 + \Delta m)$ samples:

$$y[n] = x\left[n - M_{\max} + (\ \ m)n\right] = x\left[(1 + \ \ m)n - M_{\max}\right] \tag{2.16}$$

In other words, the playback rate from the buffer is $(1 + \Delta m)$ times the input rate. Correspondingly, the frequencies in the input signal $x[n]$ will all be scaled upwards by a factor of $(1 + \Delta m)$. For similar reasons, if the length of the delay line *increased* each sample ($\Delta m < 0$), the frequencies of the output signal would all be scaled down compared to the original.

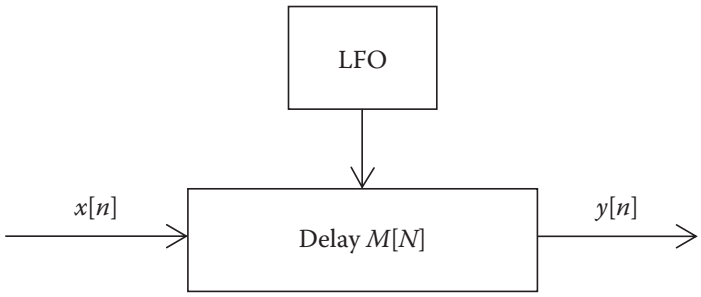


**FIGURE 2.5**
Modulated delay and pitch shift.

Notice that the pitch shift is sensitive only to the change in delay $\Delta m$, not its initial value $M_{max}$. We can generalize to write that pitch shift is dependent on the *derivative* (or, in discrete time, *first difference*) of delay length, with increasing length producing lower pitch:

$$f_{ratio}[n] = \frac{f_{out}}{f_{in}}[n] = 1 - \left(M[n] - M[n-1]\right) \tag{2.17}$$

To create a vibrato effect, then, the delay is periodically lengthened and shortened.

### *Interpolation*

Strictly speaking, Equation (2.16) is defined only for integer samples of $x$, i.e., when $(1 + \Delta m)n - M_{max}$ is an integer. In general, this is not always the case for a modulated delay line where delay lengths change gradually from sample to sample, so *interpolation* must be used to approximate noninteger values of $x$. See the earlier section on fractional delay for a more detailed discussion of interpolation and its implementation.

### Implementation

Though Equation (2.17) suggests that arbitrary pitch shifts are possible by choosing the rate $\Delta m$ at which the delay length varies, real-time audio effects are limited by two practical considerations. First, real-time effects must be *causal*, which implies that the delay length $M[n]$ must be nonnegative. If we want to increase the pitch of a sound, this means that for any finite initial delay $M_{max}$, $M[n]$ will eventually reach 0, at which point it will no longer be possible to maintain the pitch shift. The second consideration is that computer systems have finite amounts of memory. To decrease the pitch of a sound, $M[n]$ must steadily increase, which requires an ever greater memory buffer to hold the delayed samples. Because the output is being played more slowly than the input, eventually it will be impossible to hold all the intervening audio in memory.

To satisfy these two considerations, the *average change* in delay length over time must be 0. In a vibrato effect, the delay length varies periodically around a fixed central value, producing periodic pitch modulation, but a sustained increase or decrease in pitch is not possible. In Chapter 8, we will examine a technique for real-time pitch shifting using the phase vocoder.

### *Low-Frequency Oscillator*

The delay effect described earlier can be characterized as a linear, time-invariant filter. But many of the audio effects that we will encounter are not time invariant. That is, the effect acts like a filter, but now the output

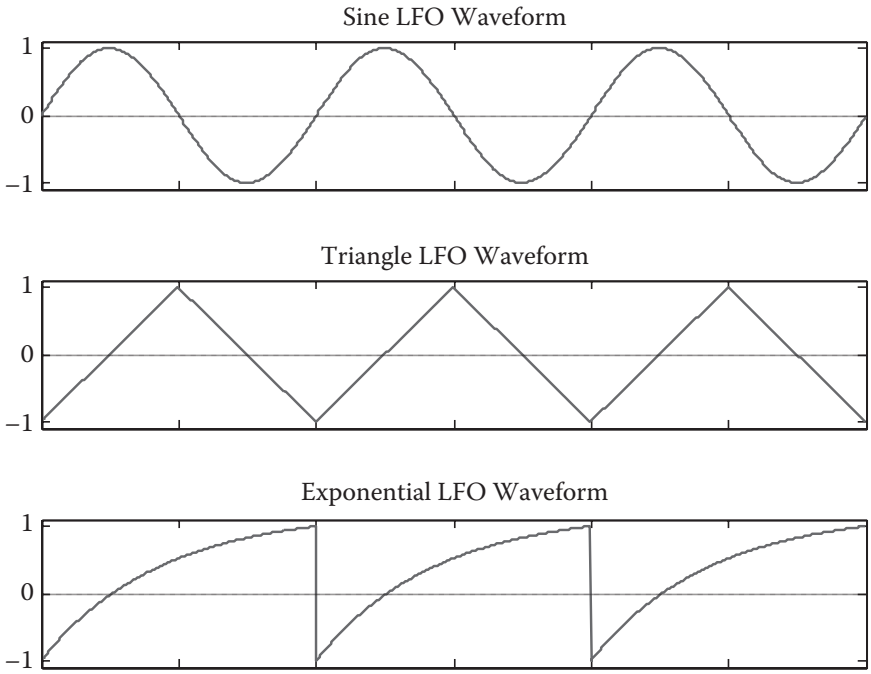


**FIGURE 2.6**
Three commonly used low-frequency oscillator (LFO) waveforms.

as a function of input depends on the time or, in discrete form, the sample number. This is most often accomplished by driving the effect (making some parameters explicitly a function of time) with a low-frequency oscillator (LFO). This is the case for vibrato and the other delay line-based effects described in the following sections.

LFOs do not have a formal definition, but they may be considered to be any periodic signals with a frequency below 20 Hz. They are used to vary delay lines or as modulating signals in many synthesizers, and they will be used in many of the effects featured later in the book. Like their audio frequency counterparts, LFOs typically use periodic waveforms such as sine, triangle, square, and sawtooth waves. However, any type of waveform is possible, including user-defined waveforms read from a wave table.

Figure 2.6 depicts three of the most commonly used waveforms. Different LFO waveforms are preferred for different effects. In the vibrato effect, the delay length is typically controlled by a sinusoidal LFO:

$$M[n] = M_{avg} + W \sin\left(2\pi nf / f_s\right) \tag{2.18}$$

where $M_{avg}$ is the average delay (for real-time effects, it is chosen so that $M[n]$ is always nonnegative), $W$ is the width of the delay modulation, $f$ is the LFO frequency in Hz, and $f_s$ is the sample rate. The rate of change from one sample to the next can be approximated by the continuous time derivative:

$$\begin{aligned} M[n] - M[n-1] &= W \ \sin\left(2\pi nf / f_s\right) - \sin\left(2\pi(n-1)f / f_s\right) \\ &\approx 2\pi fW \cos\left(2\pi nf / f_s\right) \end{aligned} \tag{2.19}$$

We can then find the frequency shift for the vibrato effect:

$$f_{ratio}[n] = 1 - \left(M[n] - M[n-1]\right) \approx 1 - 2\pi fW \cos\left(2\pi nf / f_s\right) \tag{2.20}$$

Notice that in the implementation of the LFO, $W$ indicates the amount that the delay length changes, not the amount of pitch shift. Equation (2.20) shows that pitch shift depends on both $W$ and $f$, such that for the same amount of delay modulation, a faster LFO will produce more pitch shift. As Figure 2.7 demonstrates, this is an expected result, since increasing the frequency of a sine wave while maintaining its amplitude will result in a greater derivative.

Given the desired amount of maximum pitch shift and an LFO frequency, we can calculate the approximate required amount of delay variation:

$$W = \left(f_{ratio}[n] - 1\right) / 2\pi f \tag{2.21}$$

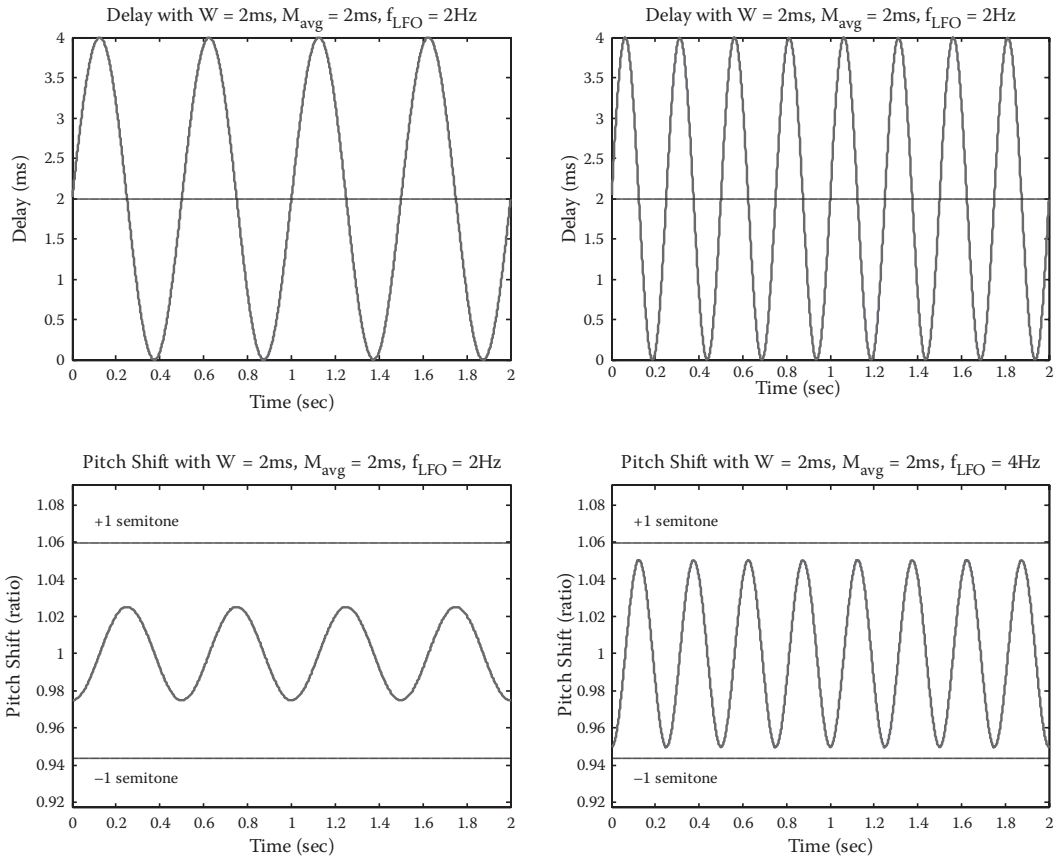


**FIGURE 2.7**
Vibrato in operation. The LFO waveform on top and the pitch shift on bottom, for LFO frequency 2 Hz (left) and 4 Hz (right).

From this, we can also find the average delay $M_{avg} \geq W$ needed to keep the effect causal. In all but the most extreme cases, $M_{avg}$ will be small enough that no delay will be perceptible at the output of the vibrato effect.

### *Parameters*

The vibrato effect is completely characterized by *LFO frequency*, *LFO waveform*, and *vibrato (pitch shift) width*. The pitch shift parameter is used to calculate the amount of delay modulation, which is what ultimately produces the vibrato effect. A typical violin vibrato has a frequency on the order of 6 Hz, with frequency variation of around 1% (i.e., approximately 0.99 to 1.01 in frequency ratio) [1]. With a sinusoidal LFO, these settings would produce a delay variation $W$ of 0.265 ms in either direction. By way of comparison, two notes a semitone apart differ in frequency by $\sqrt[12]{2} \oplus 1.059$, or 5.9%.

Sinusoidal waveforms are best for emulating normal instrumental vibrato, but other waveforms can be used for special effects. For example, a triangle waveform (Figure 2.8a) has only two slopes (rising and falling), and accordingly, the pitch will jump back and forth between two fixed

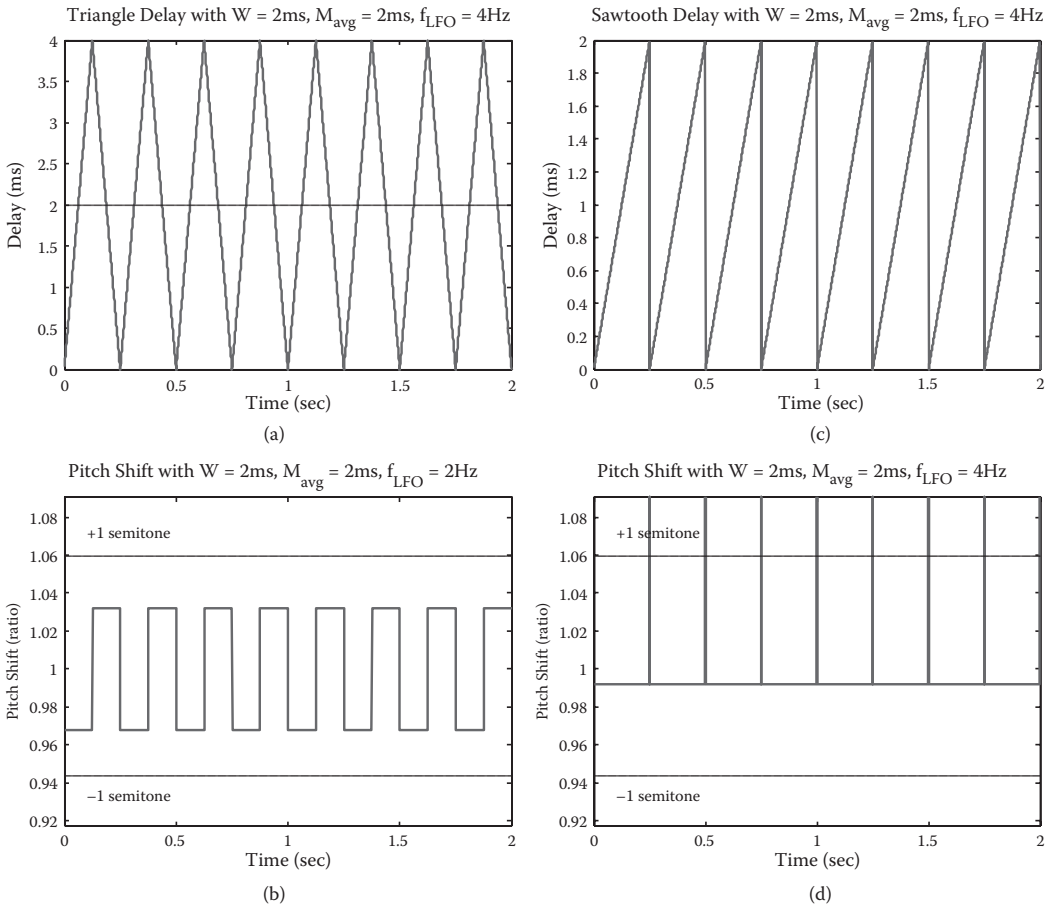


**FIGURE 2.8**
Triangular (a) and sawtooth LFOs (b), with corresponding pitch shift (c and d).

values (Figure 2.8c). A rising sawtooth wave (Figure 2.8b) approximates a pitch-lowering effect since the derivative of delay length is usually positive. However, the periodic discontinuities (Figure 2.8d) in the sawtooth waveform produce artifacts that degrade the quality of the result, so this technique is not normally used for pitch shifting.

### *Code Example*

The following C++ code fragment, adapted from the code that accompanies this book, implements a vibrato with sinusoidal LFO and linear interpolation.

```
// Variables whose values are set externally:
int numSamples;     // How many audio samples to process
float *channelData; // Array of samples, length numSamples
float *delayData;   // Our own circular buffer of samples
int delayBufLength; // Length of our delay buffer in samples
int dpw;            // Write pointer into the delay buffer
float ph;           // Current LFO phase, always between 0-1
float inverseSampleRate; // 1/f_s, where f_s = sample rate

// User-adjustable effect parameters:
float frequency_;   // Frequency of the LFO
float sweepWidth_;  // Width of the LFO in samples

for (int i = 0; i < numSamples; ++i)
{
    const float in = channelData[i];
    float interpolatedSample = 0.0;

    // Recalculate the read pointer position with respect to
    // the write pointer. A more efficient implementation
    // might increment the read pointer based on the
    // derivative of the LFO without running the whole
    // equation again, but this makes the operation clearer.
    float currentDelay = sweepWidth_ * (0.5f +
                              0.5f * sinf(2.0 * M_PI * ph));

    // Subtract 3 samples to the delay pointer to make sure
    // we have enough previous samples to interpolate with
    float dpr = fmodf((float)dpw
                  - (float)(currentDelay * getSampleRate())
                  + (float)delayBufLength - 3.0,
                  (float)delayBufLength);

    // Use linear interpolation to read a fractional index
    // into the buffer. Find the fraction by which the read
    // pointer sits between two samples and use this to
    // adjust weights of the samples
```

```
    float fraction = dpr - floorf(dpr);
    int previousSample = (int)floorf(dpr);
    int nextSample = (previousSample + 1) % delayBufLength;
    interpolatedSample = fraction*delayData[nextSample]
        + (1.0f-fraction)*delayData[previousSample];

    // Store the current information in the delay buffer.
    delayData[dpw] = in;

    // Increment the write pointer at a constant rate. The
    // read pointer will move at different rates depending
    // on the settings of the LFO, the delay and the
    // sweep width.
    if (++dpw >= delayBufLength)
        dpw = 0;

    // Store the output sample in the buffer, replacing the
    // input. In the vibrato effect, the delayed sample is
    // the only component of the output (no mixing with the
    // dry signal)
    channelData[i] = interpolatedSample;

    // Update the LFO phase, keeping it in the range 0-1
    ph += frequency_*inverseSampleRate;
    if(ph >= 1.0)
        ph -= 1.0;
}
```

The code exhibits many similarities to the basic delay in the previous section. One notable difference is that the read pointer `dpr` is now fractional, taking noninteger values. Accordingly, `dpr` cannot be directly used to read the circular buffer `delayData`, since arrays in C++ can be accessed only at integer indices. In this example, the variable `fraction` holds the noninteger component of the read pointer `dpr`; it is used to calculate a weighted average between the two nearest samples in the circular buffer (`previousSample` and `nextSample`). Notice how the index of the sample following `dpr` is calculated:

```
int nextSample = (previousSample + 1)% delayBufLength;
```

The % sign is a *modulo operator*. This means that if the expression `(previousSample + 1)` exceeds `delayBufLength`, it will be wrapped around to the beginning of the buffer. The use of modulo arithmetic is needed to implement a circular buffer.

For the vibrato effect, no feedback or mixing with the original signal is used, so many of the parameters in the basic delay example are not found here. In the complete code example that accompanies this book, a choice of LFO waveforms and interpolation types is offered.

### Applications

Vibrato, when used by vocalists or instrumentalists, can add a sense of warmth and life to a musical line. The width and frequency of vibrato and their evolution over time are important expressive decisions for many performers. Vibrato can also help an instrument or voice stand out from an ensemble. A single musical note will contain energy at discrete, harmonically related frequencies, but by varying the pitch back and forth, a single note can use more of the frequency spectrum.

Vibrato is sometimes used to cover slight errors in pitch, as it is easier to perceive a steady pitched sound as being out of tune than one containing vibrato. However, the use of vibrato to cover pitch errors is generally considered poor musical practice.

The vibrato audio effect is not as flexible as a performer's natural vibrato, since the LFO operates at a constant rate and width regardless of the musical material. Also, a simple vibrato implementation does not synchronize with the beginnings and endings of individual notes as a performer would. More advanced implementations do, though, such as can be found in some synthesizers.

Though it is possible to imagine a vibrato effect with a pedal or other control to give the user more flexibility over the LFO, this is rarely seen in practice. Nonetheless, even a fixed-frequency vibrato can add warmth and body to the sound of an instrument, especially when used with reverberation.

## Flanging

*Flanging* is a delay-based effect originally developed using analog tape machines 50 years ago (see [1] and references therein). It refers to the flange or outer rim of the open-reel tape recorders in common use in studios at the time. To create the flanging effect, two tape machines are set up to play the same tape at the same time. Their outputs are mixed together equally, as shown in Figure 2.9.

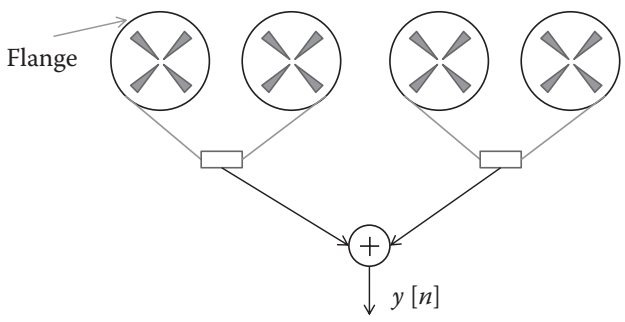


**FIGURE 2.9**
Two tape machines configured to produce a flanging effect.

**KEN'S FLANGER**

Flanging is an unusual name for an audio effect, and it is certainly not a common word in music or signal processing. The flange refers to a rim or edge, especially on a tape reel. Producers were known to manipulate the flange of a tape reel to achieve nice effects on many early tape recordings. One of the earliest known examples of producing a sound similar to the modern flanger is "The Big Hurt" by Tony Fisher, recorded in 1959.

But the origin of the name of the audio effect is an unusual one, and has been well documented by Beatles' historians Bill Biersach and Mark Lewisohn [10].

In 1966, the Beatles recorded *Revolver* at Abbey Road. The studio technician Ken Townsend later said that "they would relate what sounds they wanted and we then had to go away and come back with a solution … they often liked to double-track their vocals, but it's quite a laborious process and they soon got fed up with it. So, after one particularly trying night-time session doing just that, I was driving home and suddenly had an idea."

What Townsend devised was not the modern flanging, but the closely related chorus effect, or artificial double tracking (ADT). But it is implemented using the same approach, slowing down and speeding up a tape machine. The seemingly random variations in speed (and hence also pitch) mimic the effect of a singer trying to harmonize with the original.

John Lennon loved the effect, and asked George Martin, the Beatles' producer, to explain it. As Martin recalled, "I knew he'd never understand it, so I said, 'Now listen, it's very simple. We take the original image and split it through a double vibrocated sploshing flange with double negative feedback … .' He said, 'You're pulling my leg, aren't you?' I replied, 'Well, let's flange it again and see.' From that moment on, whenever he wanted it he'd ask for his voice to be 'flanged,' or call out for 'Ken's flanger'" [11].

If the two machines played perfectly in unison, the result would simply be a stronger version of the same signal. Instead, the operator lightly touches the flange of one of the tape machines, slowing it down and thereby lowering the pitch. This action also causes the tape machine to fall slightly behind its counterpart, creating a delay between them. The operator then releases the flange and repeats the process on the other machine, which causes the delay to gradually disappear and then grow in the opposite direction. The process is repeated periodically, alternately pressing each flange.

If too much delay accumulates between the machines, the mixed output will no longer be heard as a single signal but as two distinct copies. For this reason, the delay must be kept well below the threshold of echo perception (see Chapter 9), i.e., only a few milliseconds in each direction, so the result is heard as a single sound rather than two separate sounds.

The flanging effect has been described as a kind of "whoosh" that passes through the sound. The effect has also been compared to the sound of a jet passing overhead, in that the direct signal and the reflection from the ground arrive at a varying relative delay. And when the delay is modulated rapidly, an audible Doppler shift may be heard [1] (see Chapter 10).

## Theory

### *Principle of Operation*

The flanger is based on the principle of constructive and destructive interference. If a sine wave signal is delayed and then added to the original, the sum of the two signals will look quite different depending on the length of the delay. At one extreme, when the two signals perfectly align in phase, the output signal will be double the magnitude of the input. This is *constructive interference*. At the other extreme, when the delay causes the two signals to be perfectly out of phase, they cancel each other out: an increase in one signal is precisely balanced by a decrease in the other, so they will sum to zero. This is *destructive interference*.

Typical audio signals contain energy at a large number of frequencies. For any given delay value, some frequencies will add destructively and cancel out (*notches* in the frequency response) and others will add constructively (*peaks*). Peaks and notches by themselves do not make a flanger: it is the *motion* of these notches in the frequency spectrum that produces the characteristic flanging sound. As the following sections show, the motion of the peaks and notches is achieved by continuously changing the amount of delay (Figure 2.10).

### *Basic Flanger*

A block diagram of a basic flanger is shown in Figure 2.11. Its operation is closely related to the basic delay discussed previously, with the key difference that the amount of delay varies over time. It is also closely related to the vibrato that was depicted in Figure 2.5. The input/output relation for the flanger can be expressed in the time domain as

$$y[n] = x[n] + gx\ \ n - M[n] \tag{2.22}$$

The delay length $M[n]$ varies under the control of a separate low-frequency oscillator, discussed in the following sections. This structure is known as a

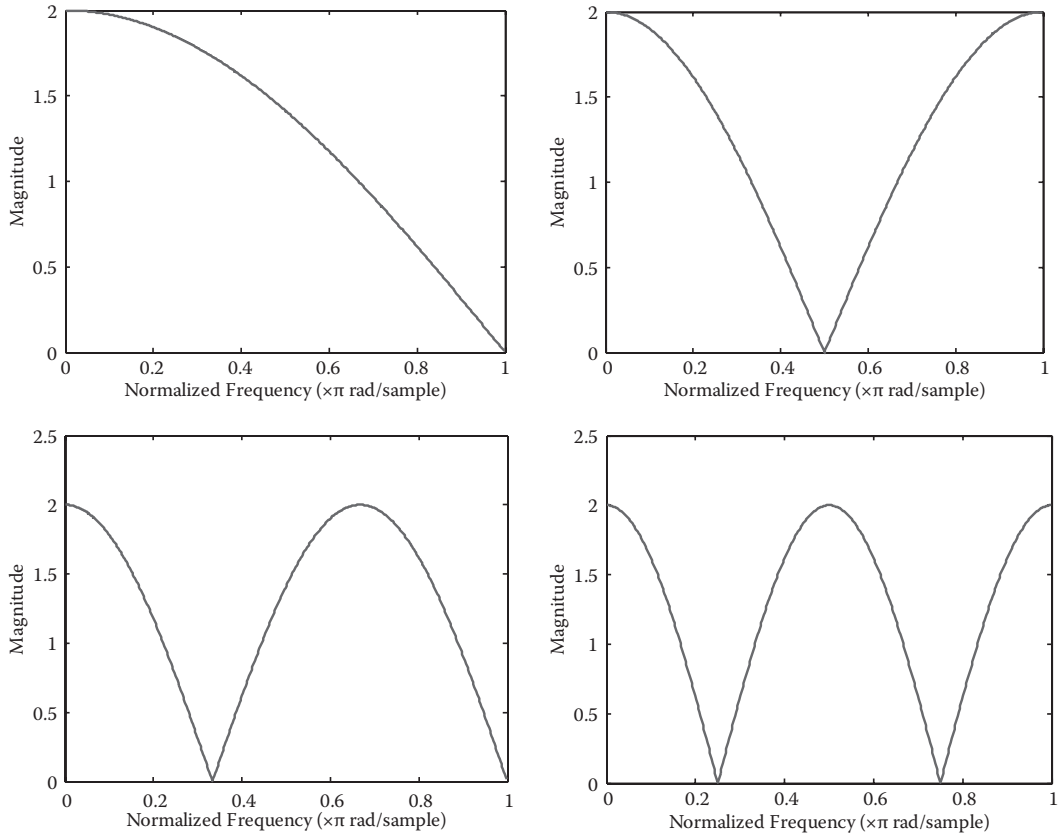


**FIGURE 2.10**
The magnitude response of a flanger with depth set to 1 and delay times set to one, two, three, and four samples.

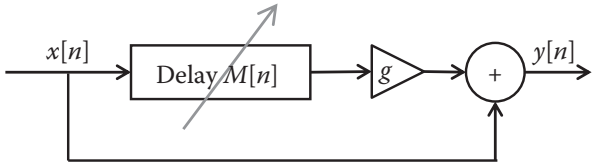


**FIGURE 2.11**
Block diagram of a basic flanger without feedback. The delay length $M[n]$ changes over time.

*feedforward comb filter*, since the delayed signals feed forward from the input to the output (with no feedback). To see why this difference equation results in a comb filter, we should consider its $Z$ transform and transfer function:

$$Y(z) = X(z) + gz^{-M[n]}X(z) \qquad H(z) = \frac{Y(z)}{X(z)} = 1 + gz^{-M[n]} \tag{2.23}$$

To find the frequency response of the flanger for each frequency ω, we substitute $e^{j\omega}$ for $z$ and find the magnitude of the transfer function:

$$H(e^{j\omega}) = 1 + ge^{-j\omega M[n]} = 1 + g\cos(\omega M[n]) - jg\sin(\omega M[n])$$
$$\left|H(e^{j\omega})\right| = \sqrt{\left(1 + ge^{-j\omega M[n]}\right)\left(1 + ge^{j\omega M[n]}\right)} = \sqrt{1 + 2g\cos(\omega M[n]) + g^2} \tag{2.24}$$

Notice that the frequency response is periodic: for $g > 0$, we have $M$ peaks in the frequency response, located at the frequencies when the cosine term reaches its maximum value:

$$\omega_p = 2\pi p/M \quad \text{where} \quad p = 0, 1, 2, \ldots, M-1 \tag{2.25}$$

Likewise, for $g > 0$, there are $M$ notches (minima) in the frequency response. These are located where the cosine term reaches its minimum value:

$$\omega_n = (2n+1)\pi/M \quad \text{where} \quad n = 0, 1, 2, \ldots, M-1 \tag{2.26}$$

The pattern of peaks and notches is shown in Figure 2.12. The equations show that a larger delay $M[n]$ produces more notches and a lower frequency for the first notch. As $M[n]$ varies, the notches sweep up and down through

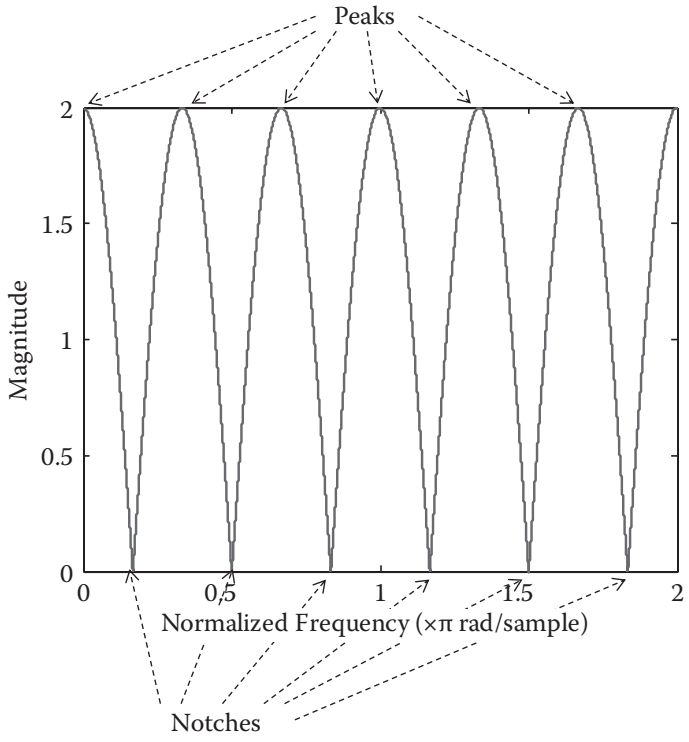


**FIGURE 2.12**
The frequency response of a simple flanger with a six-sample delay and depth set to 1. The locations of the six peaks and six notches over the whole frequency range from 0 to 2π are shown.

the frequency range. The notches are regularly spaced at intervals of $f_s/M$ Hz where $f_s$ is the sampling rate. Their pictorial resemblance in Figure 2.12 to the teeth of a comb is what gives this structure the term *comb filter*. Chapter 4 will examine *phasing*, an effect that produces similar moving notches in the frequency response that are not regularly spaced.

The depth of the notches depends strongly on the gain $g$ of the delayed signal. When $g = 0$, the frequency response is perfectly flat, as we would expect since $g = 0$ corresponds to no delayed signal. When $g$ is between 0 and 1 (or greater than 1), notches appear in the spectrum, but their depth is finite and depends on the value of $g$. When $g = 1$, the notches are infinitely deep, as the frequency response exactly equals 0 at the notch frequencies $w_n$. For this reason, $g = 1$ produces the most pronounced flanging effect.

### *Low-Frequency Oscillator*

The characteristic sound of the flanger comes from the *motion* of regularly spaced notches in the frequency response. For this reason it is critical that the length of the delay $M[n]$ changes over time. Typically, $M[n]$ is varied using a low-frequency oscillator (LFO). The LFO can be one of several waveforms, including sine, triangle, or sawtooth, with sine being the most common choice. Typical delay lengths for the flanger range from 1 to 10 ms, corresponding to notch intervals ranging from 1000 Hz down to 100 Hz. Further details on LFO parameters are discussed in the "Parameters" section below.

### *Flanger with Feedback*

Just as feedback could be added to the basic delay, some flangers incorporate a feedback path that routes the scaled output of the delay line back to its input, as shown in Figure 2.13. Feedback on the flanger is also sometimes referred to as *regeneration*. As with the delay effect with feedback, using feedback in the flanger will result in many successive copies of the input signal spaced several milliseconds apart and gradually decaying over time (Figure 2.14). However, since the delay times in the flanger (typically less than 20 ms) are below the threshold of echo perception (roughly 50–70 ms), these copies are not heard as independent sounds but as coloration or filtering of the input sound.

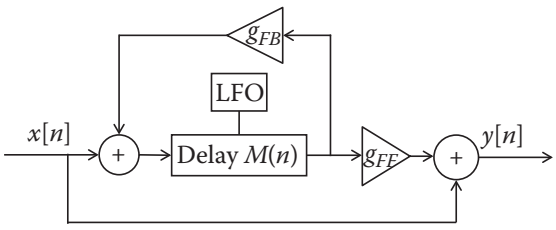


**FIGURE 2.13**
Flanger with feedback. Delay in all flangers is controlled by a low-frequency oscillator (LFO).

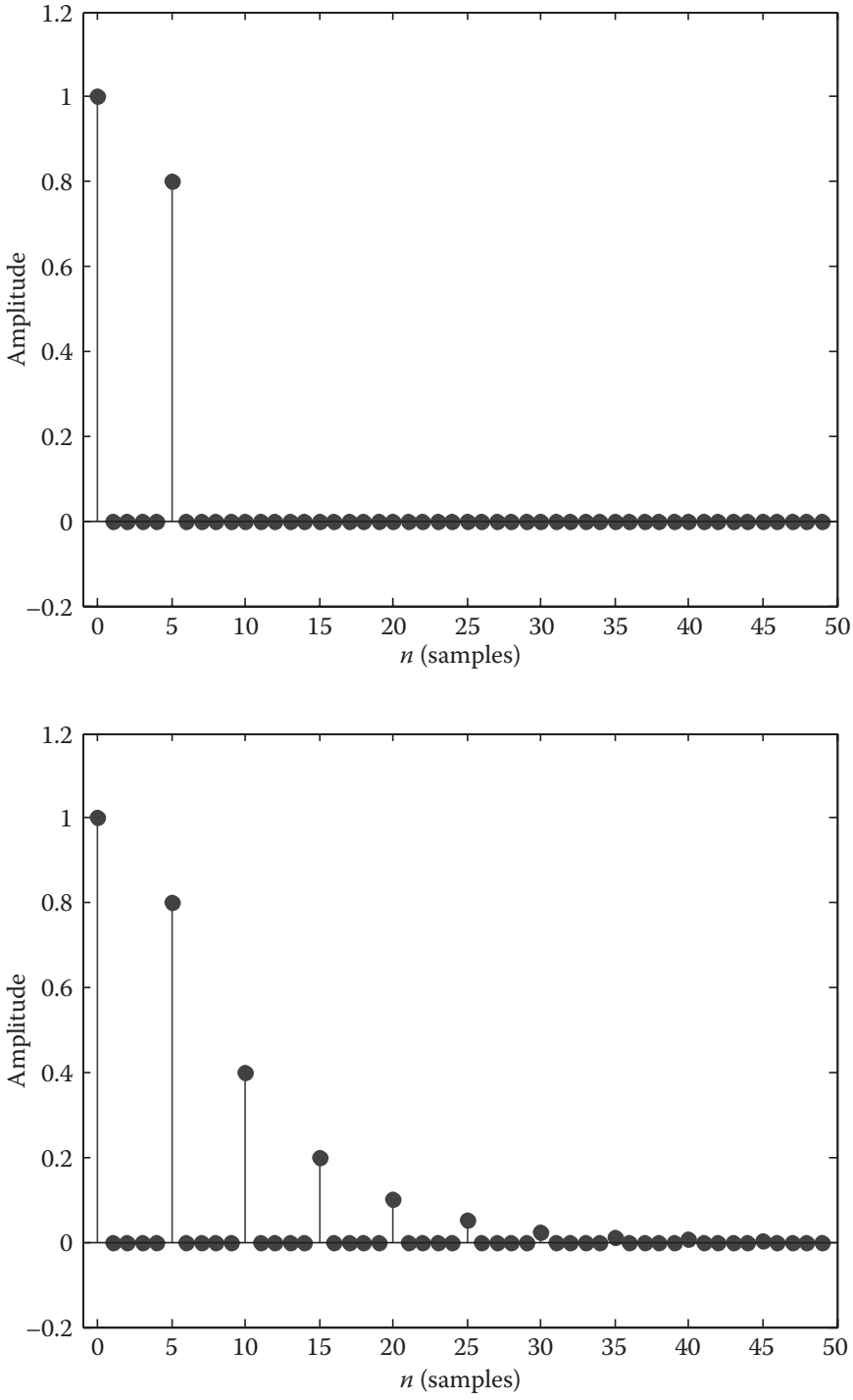


**FIGURE 2.14**
Impulse responses for a comb filter (i.e., delay effect) with a five-sample delay. On top, $g = 0.8$ and no feedback. On bottom, $g_{FF} = 0.8$ and $g_{FB} = 0.5$.

The difference equation and frequency response for a flanger with feedback can be derived similarly to the delay with feedback:

$$\begin{aligned} y[n] &= g_{FB}y\big[n - M[n]\big] + x[n] + (g_{FF} - g_{FB})x\big[n - M[n]\big] \\ H(z) &= \frac{Y(z)}{X(z)} = \frac{1 + z^{-M[n]}(g_{FF} - g_{FB})}{1 - z^{-M[n]}g_{FB}} = \frac{z^{M[n]} + g_{FF} - g_{FB}}{z^{M[n]} - g_{FB}} \end{aligned} \tag{2.27}$$

We can see that when the feedback gain $g_{FB} = 0$, these terms exactly match the basic flanger without feedback, as expected. In addition to the zeros of $H(z)$, which are similarly located to the flanger without feedback, the transfer function has poles at the complex roots of $g_{FB}$. If $g_{FB} < 1$, these will remain inside the unit circle and the system will be *stable*. This is an intuitive result: feedback gains less than 1 mean that the delayed copies of the sound will gradually decay, where a gain of 1 or more means they will grow (or at least persist with significant amplitude) indefinitely.

The effect of feedback is to make the peaks and notches sharper and more pronounced. Its sound is often described as intense or metallic, and as the feedback gain approaches 1, the pitch $f_s/M$ resulting from the delay line can overwhelm the rest of the sound.

### *Stereo Flanging*

A stereo flanger is constructed of two monophonic flangers that are identical in all settings except the *phase* of the low-frequency oscillator. Typically, the two oscillators are in *quadrature phase*, where one leads the other by 90°. In a stereo flanger, the same signal can be used as the input to both channels, or separate signals can be used for each input. The outputs are typically panned fully to the left and right of a stereo mix.

### *Properties*

Because the flanger (with or without feedback) is composed entirely of delays and multiplication, it is a *linear* effect. However, because the properties of the delay line vary over time independently of the input signal, it is *time variant*, unlike the standard delay: shifting the input signal in time by $N$ samples does not necessarily produce the identical output shifted by $N$ samples, since the delay line length may have changed. The basic flanger is always *stable*, where the flanger with feedback is stable if and only if the feedback gain $g_{FB} < 1$.

### Common Parameters

The typical flanger effect contains several controls that the musician can adjust.

### *Depth (or Mix)*

The *depth* control affects the amount of delayed signal that is mixed in with the original. $g = 0$ produces no effect, whereas $g = 1$ produces the most pronounced flanging effect. Higher depth settings ($g > 1$) produce a louder overall sound due to scaling up the delayed signal, but the flanging effect becomes less pronounced: only when the original and delayed copies exactly match in amplitude can perfect cancelation of the notch frequencies occur.

### *Delay and Sweep Width*

The term *delay* is potentially misleading in the flanger since the length of the delay line varies over time under the control of a low-frequency oscillator. The delay control parameter on a flanger affects the minimum amount of delay $M[n]$. The value of the LFO is added to produce larger time-varying values. *Sweep width* controls the total amplitude of the low-frequency oscillator, such that the maximum delay time is given by the sum of the delay and sweep width controls (Figure 2.15).

As the delay is decreased, the first notch becomes higher in frequency. The delay control thus sets the highest frequency the first notch will reach. If it is set to zero, the notches will disappear entirely when the LFO reaches its minimum value: when the original and delayed signals are exactly aligned, no cancelation will take place at any frequency. Similarly, the sum of the delay and sweep width controls determines the lowest frequency the first notch will reach.

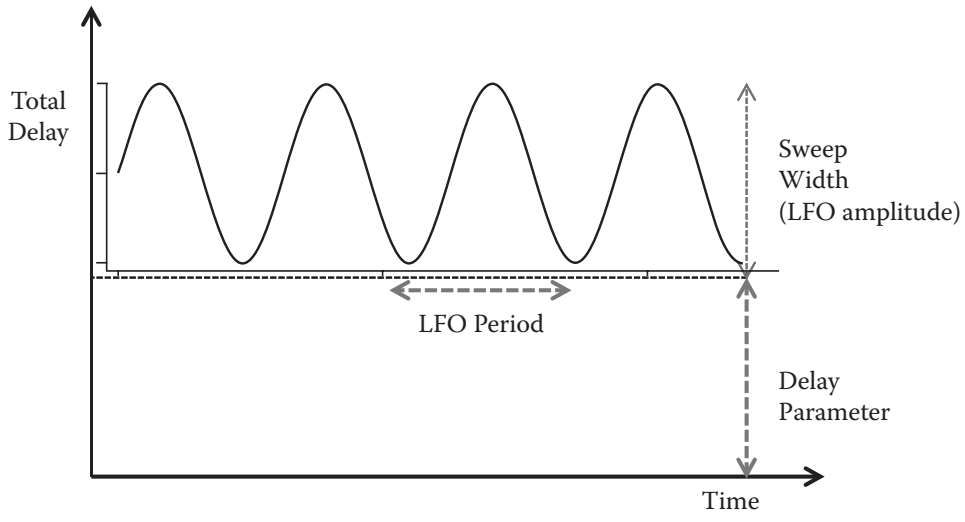


**FIGURE 2.15**
The maximum delay is the sum of the *sweep width* and *delay* parameters. The delay changes over time according to the *sweep rate*.

### *Speed and Waveform*

These controls affect the behavior of the LFO controlling the delay length. The *speed* control sets the LFO frequency and typically ranges from 0.1 Hz (10 s per cycle) to 10 Hz. *Waveform* is usually chosen from one of several predefined values, including sine, triangle, sawtooth, or exponential (triangular in log frequency). Many flangers do not offer this control and always use a sinusoidal LFO. In this case, the total delay $M[n]$ is given by

$$M[n] = M_0 + \frac{M_W}{2}\left[1 + \sin\left(2\pi f_{LFO} n / f_s\right)\right] \tag{2.28}$$

where $M_0$ (in samples) is given by the delay control, $M_W$ (in samples) is given by the sweep width control, $f_{LFO}$ (in Hertz) is given by the speed control, and $f_s$ (in Hertz) is the sampling frequency. We can see that the value of $M[n]$ varies from $M_0$ at minimum to $M_0 + M_W$ at maximum, consistent with the expected behavior of these controls.

### *Feedback (or Regeneration)*

The basic flanger has only a feedforward path, in which the delayed signal is added to the original. In a flanger with feedback, the *feedback/regeneration* control sets the gain $g_{FB}$ between output and input of the delay line. Possible values are in the range [0, 1), i.e., strictly less than 1, to maintain stability. In practice, values close to 1 are rarely used except for special effects. Even when the system is mathematically stable, large gains at the peaks can result in clipping distortion depending on the level of the input.

### *Inverted Mode (or Phase)*

On some flangers, the feedforward gain $g$ or $g_{FF}$ can be altered in polarity. *Inverted mode* is typically selected with a switch; when activated, $g$ ranges from 0 to –1 instead of 0 to 1, in which case, the peaks and notches in the frequency response will trade places; the lowest peak will occur at $f = f_s/2M$ Hz and the lowest notch at $f = 0$ (DC). Because of the notch at DC, the bass response in the inverted mode is poor, producing a thinner sound unless $M$ is very large (which reduces the frequency of the lowest peak). The different color of the inverted mode flanger can be useful in some musical situations.

## Implementation

### *Buffer Allocation*

The flanger, like all delay-based effects, is typically implemented digitally using *circular buffers* (see Chapter 13). Memory allocation and deallocation

are highly time-consuming in comparison to basic audio calculations. Since the length of the delay changes with the phase of the LFO, the buffer is preallocated to be large enough to accommodate the maximum amount of delay at any point in the LFO cycle, for any settings of the delay and sweep width parameters.

The actual length of delay at any time is controlled by the distance between the read pointer and write pointer in the buffer. In a typical implementation, the write pointer will move at a constant speed, advancing one sample in the buffer for each input sample. Moving the read pointer faster than this rate will decrease the amount of delay, while moving it slower will increase the delay.

### *Interpolation*

Since the delay of the flanger changes by small amounts each sample, it will inevitably take fractional values. As discussed previously in this chapter, mathematically exact *fractional delay* involves calculations requiring knowledge of the complete signal extending to infinity in both directions. This is clearly impractical, so approximations based on low order polynomial interpolation are used that are suitable for real-time computation. Interpolation is always used when calculating the delayed signal of the flanger.

### *Code Example*

The following C++ code fragment, adapted from the code that accompanies this book, implements a flanger with feedback.

```
// Variables whose values are set externally:
int numSamples;     // How many audio samples to process
float *channelData; // Array of samples, length numSamples
float *delayData;   // Our own circular buffer of samples
int delayBufLength; // Length of our delay buffer in samples
int dpw;            // Write pointer into the delay buffer
float ph;           // Current LFO phase, always between 0-1
float inverseSampleRate; // 1/f_s, where f_s = sample rate

// User-adjustable effect parameters:
float frequency_;   // Frequency of the LFO
float sweepWidth_;  // Width of the LFO in samples
float depth_;       // Amount of delayed signal mixed with
                    // original (0-1)
float feedback_;    // Amount of feedback (>= 0, < 1)

for (int i = 0; i < numSamples; ++i)
{
    const float in = channelData[i];
    float interpolatedSample = 0.0;
```

```
    // Recalculate the read pointer position with respect to
    // the write pointer.
    float currentDelay = sweepWidth_ * (0.5f +
                              0.5f * sinf(2.0 * M_PI * ph));

    // Subtract 3 samples to the delay pointer to make sure
    // we have enough previous samples to interpolate with
    float dpr = fmodf((float)dpw
                   - (float)(currentDelay * getSampleRate())
                   + (float)delayBufLength - 3.0,
                   (float)delayBufLength);

    // Use linear interpolation to read a fractional index
    // into the buffer.
    float fraction = dpr - floorf(dpr);
    int previousSample = (int)floorf(dpr);
    int nextSample = (previousSample + 1) % delayBufLength;
    interpolatedSample = fraction*delayData[nextSample]
        + (1.0f-fraction)*delayData[previousSample];

    // Store the current information in the delay buffer.
    // With feedback, what we read is included in what gets
    // stored in the buffer, otherwise it's just a simple
    // delay line of the input signal.
    delayData[dpw] = in + (interpolatedSample * feedback_);

    // Increment the write pointer at a constant rate.
    if (++dpw >= delayBufLength)
        dpw = 0;

    // Store the output in the buffer, replacing the input
    channelData[i] = in + depth_ * interpolatedSample;

    // Update the LFO phase, keeping it in the range 0-1
    ph += frequency_*inverseSampleRate;
    if(ph >= 1.0)
        ph -= 1.0;
}
```

This code example is nearly identical to the code for vibrato. The main differences appear at the end of the example, where feedback is used on the delay buffer and the original (dry) signal is mixed with the output:

```
delayData[dpw] = in + (interpolatedSample * feedback_);
// [...]
channelData[i] = in + depth_ * interpolatedSample;
```

For this reason, the flanger also has feedback and depth parameters where the vibrato example did not. This example code for the flanger could be used

with slight modifications and different parameter values (mainly longer delay time) to implement a chorus. Complete flanger and chorus examples accompany this book, including variable LFO waveform and stereo options.

## Applications

The flanger originated as a way of conveniently simulating the double-tracking effect on vocals, but its application goes well beyond the voice. Flanging is commonly used as a guitar effect (where it can be implemented with analog or digital electronics) and is often applied to drums and other instruments. The frequency of the LFO can be aligned to the tempo of the music for beat-synchronous effects.

### *Resonant Pitches*

Recall that audio signals of a single pitch are typically composed of *harmonically related* sinusoids, i.e., integer multiples of a fundamental frequency. Because the peaks and notches in the flanger frequency response are always uniformly spaced, they can impose a discernible resonant pitch on the audio signal. The effect is similar to being inside a resonant tube whose length changes over time according to the amount of delay [1]. The resonance effect is particularly strong when feedback is used and when the depth control is at its maximum.

### *Avoiding Disappearing Instruments*

The notches in a flanger are spaced at regular intervals in frequency, much like the harmonics of a musical instrument, where the signal consists of regular multiples of a fundamental frequency. If a flanger is applied to an instrument sound and the notches happen to line up precisely with the instrument's harmonics, it is possible for the instrument to disappear entirely. In practice, the effect will never be perfect and the instrument will not be completely eliminated, but strange amplitude modulation effects could take place as the notches sweep up and down. This problem does not occur when flanging is applied to more noise-like signals, such as drums. Flanging can also be used on an entire mix, where the frequency content is likely to be complex enough to avoid these modulation effects.

### *Flanging versus Chorus*

The flanger and the chorus are nearly identical in implementation, both being based on modulated delay lines. The primary difference is that the chorus uses longer delay times (30 ms is a typical value) to accentuate the perception of multiple instruments playing together. Flanging and chorus

are used in similar situations, but because of the greater delay between copies of the sound and corresponding perception of multiple instruments, chorus is somewhat less likely to be used on complex audio sources such an entire mix.

## Chorus

In music, the chorus effect occurs when several individual sounds with similar pitch and timbre play in unison. This phenomenon occurs naturally with a group of singers or violinists, who will always exhibit slight variations in pitch and timing, even when playing in unison. These slight variations are crucial to producing the lush or shimmering sound we are accustomed to hearing from large choirs or string sections. The chorus audio effect simulates these timing and pitch variations, making a single instrument source sound as if there were several instruments playing together.

### Theory

#### *Basic Chorus*

Figure 2.16 shows a block diagram of a basic chorus, in which a delayed copy of the input signal is mixed with the original. As with the flanger and vibrato effect, the delay length varies with time (modulated delay line). The input/output relationship can be written as

$$y[n] = x[n] + gx\big[n - M[n]\big] \tag{2.29}$$

Notice that this formula is identical to the basic flanger presented in the previous section. In general, the chorus effect is nearly identical to the flanger, using the same structure with different parameters. The main difference is the *delay length*, which in a chorus is usually between 20 and 30 ms, in

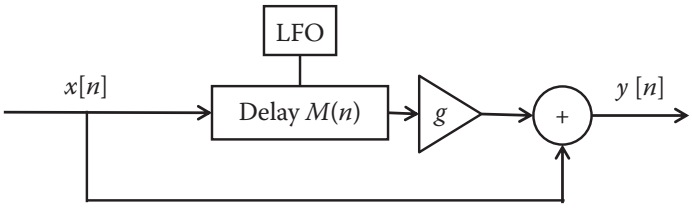


**FIGURE 2.16**
The flow diagram for the chorus effect including its LFO dependence. The delay changes with time.

contrast to delays between 1 and 10 ms in the flanger. We saw previously that the flanger produces patterns of *constructive* and *destructive interference*, resulting in a frequency response characterized by evenly spaced peaks and notches:

$$\left|H(e^{j\omega})\right| = \sqrt{1 + 2g\cos\left(\omega M[n]\right) + g^2} \tag{2.30}$$

with the peaks (points of maximum frequency response) located at

$$\omega_p = 2\pi p/M \quad \text{where} \quad p = 0, 1, 2, \ldots, M-1 \tag{2.31}$$

and the notches (minimum frequency response) at

$$\omega_n = (2n+1)\pi/M \quad \text{where} \quad n = 0, 1, 2, \ldots, M-1 \tag{2.32}$$

Thus, the *comb filtering* produced by the flanger also occurs in the chorus. However, the longer delay $M[n]$ substantially alters its perceived effect. At a sample rate of 48 kHz, a 30 ms delay corresponds to $M = 1440$ samples. There will therefore be 1440 peaks and 1440 notches in the frequency response, each located at intervals of $f_s/M$ (recalling that $\omega = 2\pi$ corresponds to the sampling frequency $f_s$). Thus, a peak will occur every 33.3 Hz, with notches likewise spaced every 33.3 Hz. These are close enough together that the characteristic sweeping timbre of the flanger is no longer perceptible. In particular, any sense that the comb filter has a definite pitch (owing to its regularly spaced peaks) will be lost at such close spacing. Nonetheless, though the sound is different from the flanger, this comb filtering is an important part of the sonic signature of the chorus effect.

Considered a different way, a delay on the order of 20–30 ms begins to approach the threshold where two separate sonic events can be perceived, though a clear perception of an echo requires a longer delay still (100 ms or more). So the chorus can also be heard as two separate copies of the same sound, whose exact timing relationship changes over time as $M[n]$ changes. Both understandings are mathematically correct; the only difference lies in human audio perception.

### *Low-Frequency Oscillator*

In the chorus, as in the flanger, the delay length $M[n]$ varies under the control of a low-frequency oscillator. Several waveforms can be used for the LFO, with sinusoids being the most common. In comparison to the flanger, slower LFO sweep rates (3 Hz or less) but higher LFO sweep widths (5 ms or more) are typically used. As with all modulated delay effects, interpolation is used to calculate the output of the delay line whenever $M[n]$ is not an integer.

### *Pitch-Shifting in the Chorus*

The wider sweep width (delay variation) in the chorus has an important consequence on the pitch of the delayed sound. As was shown for the vibrato effect, changing the length of a delay line introduces a *pitch shift* into its output, where lengthening the delay scales all frequencies in a signal down, and reducing the delay scales them up. Recall the formula for pitch shift as a function of LFO frequency $f$, sweep width $W$, and sample rate $f_s$:

$$f_{ratio}[n] \approx 1 - 2\pi f W \cos\left(2\pi n f / f_s\right) \tag{2.33}$$

Given that the cosine function ranges from –1 to 1, we can find the maximum pitch shift for any given set of parameters as

$$f_{ratio,\max} = 1 + 2\pi f W \tag{2.34}$$

For $f$ = 1 Hz and sweep width $W$ = 10 ms, this results in a pitch ratio of 1.063 (6.3% variation), slightly more than a semitone (5.9%) in either direction. This is a noticeable amount of tuning variation between the original and delayed copies of the signal, which can simulate and even exaggerate the natural variation in pitch between musicians playing in unison. Note that in comparison to the vibrato effect, the chorus mixes the original and delayed copies, so a single pitch shift is not heard.

### *Multivoice Chorus*

The basic chorus can be considered a *single-voice chorus* in that it adds a single delayed copy to the original signal. A *multivoice chorus*, by contrast, involves several delayed copies of the input signal mixed together, with each delayed copy moving independently. Figure 2.17 shows a diagram for an arrangement with two delayed copies (*dual voice*). Each individual voice can be analyzed identically to the basic chorus described in the preceding sections, but the sum total of all voices will produce a more complex, richer tone suggest-

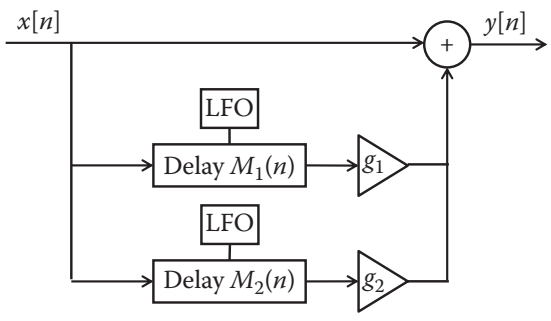


**FIGURE 2.17**
A multivoice chorus diagram.

ing multiple instruments played in unison. The "Implementation" section below discusses control strategies for the voices in a multivoice chorus.

### *Stereo Chorus*

Stereo chorus is a variation on multivoice chorus, where each delayed copy of the signal is panned to a different location in the stereo field. When two voices are used, the delayed signals are typically panned completely to the left and right, with the original signal in the center. In this case, the two voices are usually run in quadrature phase: each LFO has the same sweep rate and same sweep width, but they differ in phase by 90°. When more than two voices are used, they may be spread evenly across the stereo field or split into two groups, with one group panned hard left and the other group panned hard right.

### *Properties*

The chorus is implemented identically to a flanger without feedback, so it shares the same properties. Notably, since delay and mixing are linear operations, the chorus is a linear effect (for any number of voices). The delay $M[n]$ changes over time under the control of the LFO, so the chorus is a time-variant effect: an input signal applied at one time may produce a different result than the same signal applied at a different time if the LFO phase differs. Since the chorus never involves feedback, it is always stable, with a bounded input producing a bounded output.

## Common Parameters

Chorus effects have several user-adjustable controls.

### *Depth (or Mix)*

As with the flanger, the depth controls affect the amount of delayed signal(s) that are mixed in with the original. $g = 0$ produces no effect, whereas $g = 1$ produces the most pronounced chorus effect. Higher depth settings ($g > 1$) make the delayed copies louder than the original, a setting rarely found in practice as it produces a weaker chorus effect than $g = 1$. Some simple chorus effects may not have this control (always setting $g$ to 1). Confusingly, the term *depth* is also sometimes used to refer to sweep width, or the amount of variation in the delay. When examining an existing chorus effect, it is thus important to find out what the depth control means.

### *Delay and Sweep Width*

The *delay* parameter on the chorus controls the minimum amount of delay $M[n]$. Typical values are on the order of 20 to 30 ms, and this setting represents

one of the primary differences between flanger and chorus. The *sweep width* (which is sometimes called *sweep depth*, but should not be confused with *depth/mix*) controls the amount of additional delay added by the LFO. In other words, sweep width controls the amplitude of the LFO, and the maximum delay is given by the sum of delay and sweep width. The relationship between the delay and sweep width parameters is depicted in Figure 2.15. Typical values for sweep width range from 1–2 to 10 ms or more. A larger sweep width will result in more pitch, creating a warbling effect, whereas changing the delay parameter will not affect the pitch modulation.

### *Speed and Waveform*

As in the flanger, the *speed* (or *sweep rate*) sets the number of cycles per second of the LFO controlling the delay time. In addition to producing a more quickly oscillating chorus effect, higher speed will produce more pronounced pitch modulation for the same sweep width, since the delay line length will be changing more quickly over time. Typical values in the chorus are slower than in the flanger, ranging from roughly 0.1 to 3 Hz.

The *waveform* control selects one of several predefined LFO waveforms, including sine, triangle, sawtooth, or exponential (triangular in log frequency). In addition to controlling how the voices move in time, each waveform will affect the type of pitch modulation. For example, the derivative of a sine wave is a cosine, which is always changing, so the pitch is always changing as well. A triangular waveform, though, has only two discrete slopes, so the pitch will jump back and forth between two fixed values. Many chorus effects do not offer a waveform parameter and always use a sine LFO. As with the flanger, this results in a total delay $M[n]$ given by

$$M[n] = M_0 + \frac{M_W}{2}\left[1 + \sin\left(2\pi f_{LFO} n / f_s\right)\right] \tag{2.35}$$

where $M_0$ (in samples) is given by the *delay* control, $M_W$ (in samples) is given by the *sweep width* control, $f_{LFO}$ (in Hertz) is given by the speed control, and $f_s$ (in Hertz) is the sampling frequency. $M[n]$ thus varies from $M_0$ to $M_0 + M_W$.

### *Number of Voices*

As discussed in the previous section, a multivoice chorus uses more than one delayed copy of the input sound, simulating the effect of more than two instruments being played in unison. Many chorus units give the option of choosing the number of voices. In a simple implementation, each voice could be controlled by the same LFO, but with a different phase. The delay time will be different for each voice since they are at different points in the waveform, but they will remain synchronized to one another over time. More complex implementations can use different LFO waveforms and speeds for each voice.

### *Other Variations*

When multiple instruments play in unison, the variations between them are likely to be more random than periodic. Instead of using a fixed-speed LFO, the delay time between voices could be changed in a more irregular, quasi-random fashion (keeping in mind that abrupt changes in delay will produce audible pitch artifacts). Another variation is to modulate the amplitude of each voice to model the fact that musicians playing in unison will not all have the same relative loudness.

## Summary: Flanger and Chorus Compared

The chorus and flanger effects are nearly identical in structure. The main differences are in the parameter settings: the chorus uses a longer delay time than the flanger and often a larger sweep width. These together produce more sense of separation between the original and delayed copies of the signal and more pitch modulation. The chorus tends to use a lower speed or sweep rate than the flanger, though there is a significant area of overlap. One structural difference is that the flanger can use feedback to produce a more intense effect, whereas this is almost never found in the chorus. On the other hand, the chorus can use more than one delayed copy of the sound (multivoice chorus), where the flanger uses only one copy (except in a stereo flanger). When chorus and flanger effects are implemented in stereo, the same procedure is used in both cases, panning one delayed copy to the left and one to the right, though a multivoice chorus with more than two delayed copies allows further variations on this procedure.

### Problems

1. Suppose we want to delay a signal by 2.5 samples. Briefly explain two different methods of calculating the new signal and their relative advantages and disadvantages.
2. Consider a signal $x[0] = 0.8$, $x[1] = 0.4$, $x[2] = 0.1$, $x[3] = -0.15$, $x[4] = -0.4$. Use zero-, first-, and second-order interpolation to estimate the value $x[1.7]$.
3. a. Draw a block diagram for delay with feedback. Label the input $x[n]$, output $y[n]$, and any other commonly used parameters.
   b. Under what conditions is the system stable? Why?
4. A delay without feedback produces notches at 300 and 900 Hz. List at least three other frequencies where notches will also be present,