4 Synthesis Fundamentals

Sound synthesis is the generation of a signal that creates a desired acoustic sensation. This chapter begins with the fundamentals of signal generation and presents techniques of additive synthesis, modulation, and noise generation. Several example computer instrument designs are given along with compositional examples.

4.1 COMPUTER INSTRUMENTS, UNIT GENERATORS, AND SOUND-SYNTHESIS TECHNIQUES

In computer music, the term *instrument* refers to an algorithm that realizes (performs) a musical event. It is called upon by a computer program that is interpreting either a score stored in memory or the actions of a performer on a transducer. The instrument algorithm calculates the sample values of an audio signal using inputs, known as parameters, received from the calling program. For example, an instrument designed to play a single, simple tone might be passed parameters controlling the duration, frequency, and amplitude of the tone. Other parameters can be passed that affect other aspects of the sound. When designing an instrument, the musician determines the number and nature of the parameters to be passed. These are based on a choice of which attributes of the sound will be controlled *externally* during the generation of the sound. An instrument can also be designed to accept an audio signal in digital form as an input to be processed by the algorithm.

There are many ways to specify and provide control of computer instruments. At the lowest level, the musician writes a step-by-step program to generate sample values. It can be difficult at this level to have a sense of how to modify the algorithm to produce a particular sound, and the overall musical plan can be obscured by the necessary attention to detail.

The next level of specification improves both conceptual clarity and programming convenience by dividing a complete sound-generating algorithm into smaller, separate algorithms called *unit generators*. Each unit generator has input parameters and at least one output. Each performs a specific function of signal generation or modification, or the combination of signals. Many music languages express synthesis algorithms in terms of unit generators, using them as the building blocks with which instruments are made. The internal algorithm of each unit generator has been determined and encoded by the music systems programmer. The musician's task is to interconnect the inputs and outputs of the unit generators to achieve an overall synthesis algorithm that produces the desired result. The unit generator is a useful concept because it minimizes the amount of knowledge of the inner workings of each algorithm required on the part of the musician, while retaining considerable flexibility for the construction of synthesis algorithms.

Individual unit generators and methods of interconnecting them will be demonstrated in the next three chapters to explain the synthesis of specific sounds.

With signal-generating models, the third level of specification, the musician chooses from a set of available synthesis techniques. In this case, the computer is preprogrammed with the appropriate interconnections of unit generators. The musician selects a technique and specifies the parameters necessary to control it. Available sound synthesis techniques include additive, subtractive, distortion (nonlinear), and granular synthesis. *Additive synthesis,* described in this chapter, is the summation of several simple tones to form a complex one. In *subtractive synthesis* (chapter 6), the algorithm begins with a complex tone and diminishes the strength of selected frequencies in order to realize the desired spectrum. Many of the additive- and subtractive-synthesis instruments use data derived from the analysis of natural sound. Chapter 7 will describe various *synthesis-from-analysis* techniques. *Distortion synthesis* (chapter 5) encompasses several techniques where a controlled amount of distortion is applied to a simple tone to obtain a more complex one. A widely used member of this class of techniques is frequency modulation, which can be thought of as the distortion of the frequency of a tone. Another technique, waveshaping, is the distortion of the waveform of a tone. *Granular synthesis* (chapter 8) assembles its sounds from a multitude of bursts of energy that are too short to be perceived musically by themselves.

The last level of instrument specification is the *physical model* (chapter 9). This method requires extensive technical research on the part of the music systems programmer. The musician is given a model of a sound production process with variable parameters to achieve a particular sound. For example, the software might simulate a violin with a list of parameters given in terms of some of the physical attributes of the modeled instrument. The musician could then alter the character of the tone by changing such parameters as the volume of the body, the bowing point, the placement of the bridge, and so on. The primary benefit of a physical model is to give the musician a means to predict intuitively, to some degree, the effect of timbral modification. For instance, an increase in body volume would be expected to lower the frequency of many of the resonances in the tone.

Physical models have been created of many instruments and of speech. In addition, they have been used for describing processes that modify sounds, such as the specification of reverberation on the basis of the physical characteristics of a room. (See chapter 10.) As it continues to develop, this method is becoming a more widely available means of providing musicians with a more intuitive approach to computer instrument design than with the direct specification of the parameters of a signal-processing algorithm.

4.2 SIGNAL FLOWCHARTS

Unit generators will be used to define the fundamental synthesis techniques presented in this, the central portion of the text. A *signal flowchart,* such as the example in figure 4.1, is a graphical representation of the way in which unit generators are interconnected to form an instrument. The symbols for the various unit generators will be given as they are introduced throughout the text.

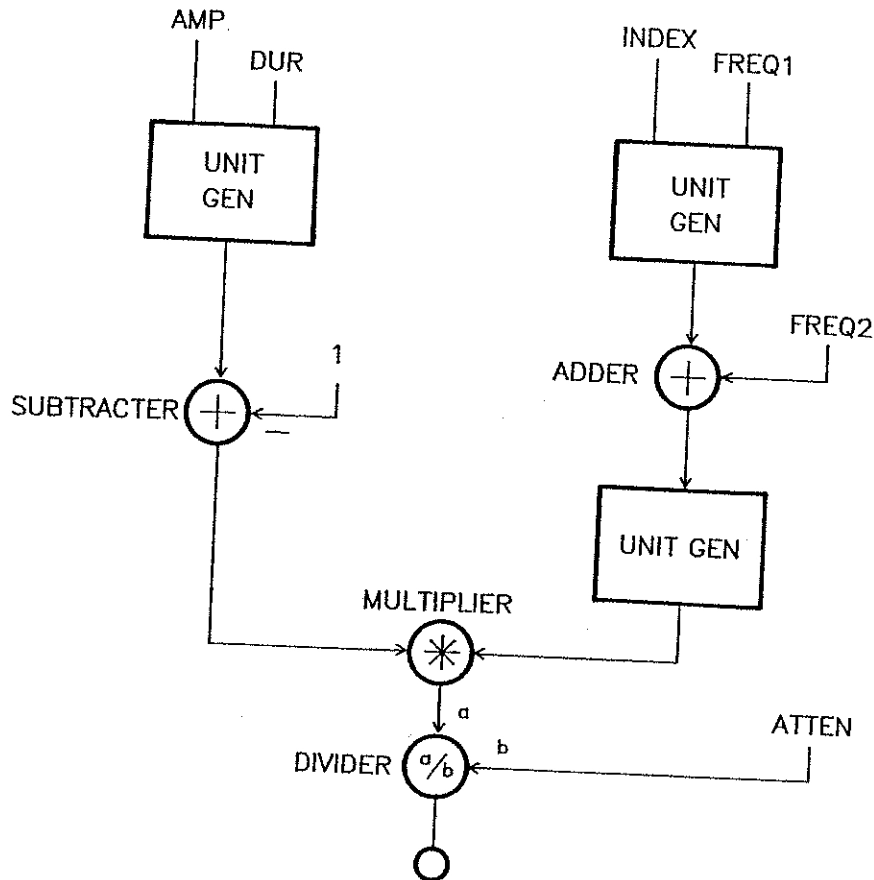

FIGURE 4.1 Example of a signal flowchart.

There are two basic rules that apply to the interconnection of unit generators: (1) An output of a unit generator may be connected to one or more input(s) of one or more other unit generator(s). Thus, an output can drive more than one input. (2) Outputs may never be connected directly together. The direct connection of outputs would result in an ambiguous situation when, as is usually the case, the unit generators provided conflicting numerical values. In the generalized unit generators of the example, the inputs go into the top of the symbol and the outputs emerge from the bottom of the symbol.

Outputs can be combined by mathematical operations. The most common combinatorial operation is addition, represented in a signal flowchart by the symbol for the adder shown in figure 4.1. An adder has two or more inputs, denoted by arrows, and one output. The principal use of an adder is to mix signals together. The operation of subtraction also uses an adder. In this case, the *sign* of the subtrahend is reversed just before it enters the adder. This is indicated by a minus sign placed near the arrow connecting the subtrahend to the adder.

The signal flowchart shown in figure 4.1 includes a multiplier and a divider, as well. Multiplying a signal by a constant with a value greater than 1 increases the amplitude of the

signal; this process is called *amplification.* The reverse process, *attenuation,* is obtained through multiplying by a constant less than 1. Multiplication and division on a general-purpose computer can take substantially longer to perform than addition or subtraction; therefore, the instrument designer tries to minimize the number of these operations. However, many modern systems incorporate special hardware that performs multiplication and division very rapidly so that this consideration becomes less important. The use of division requires special care. To avoid errors, the instrument designer must make certain that the divisor can never assume a value of 0 because the resulting quotient is infinite. On some systems, such an error can cause the synthesis program to cease operation, or at least generate an unexpected sample value, producing an associated "click" in the sound.

The instrument diagrammed in the example flowchart is controlled by six parameters indicated by the mnemonics such as AMP, DUR, and so on. The value of each parameter is passed from the main program to the instrument each time the instrument is called upon to produce a sound. Parameters are best labeled with descriptive mnemonics. For example, the parameter that controls the amplitude of an instrument is often designated AMP.

Every instrument must have at least one output. The flowchart symbol for an output is a small, empty circle usually located at the bottom of the chart. There may be multiple outputs usually corresponding to a multichannel audio system.

4.3 THE OSCILLATOR

The unit generator fundamental to almost all computer sound synthesis is the *oscillator.* An oscillator generates a periodic waveform. The controls applied to an oscillator determine the amplitude, frequency, and type of waveform that it produces. The symbol for an oscillator is shown in figure 4.2. The symbol inside the oscillator (WF in this case) designates the waveform of the oscillator. The symbol can be a mnemonic of a particular waveform or a drawing of one cycle of the waveform. The numerical value that is fed into the left input sets the peak amplitude of the signal. The numerical value applied to the right input determines the frequency at which the oscillator repeats the waveform. Depending on the system, the frequency can be specified in one of two ways: (1) an actual number of hertz, or (2) a sampling increment—a number proportional to the frequency, which will be explained below. The input on the right side of the oscillator, PHASE, determines at which point on the waveform the oscillator begins. PHASE is

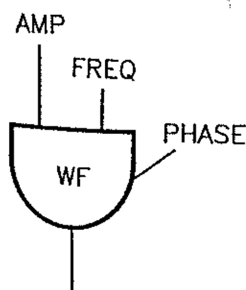

FIGURE 4.2 Flowchart symbol for an oscillator. The phase input is often not used.

usually not specified unless required for an explicit purpose. The output of the oscillator is a sequence of samples which forms a digital signal representing the waveform.

One method of implementing an oscillator algorithm specifies the waveform as a mathematical function of time. Thus, in order to generate a sine wave using this method, the algorithm would have to calculate the value of the mathematical function, sine, on every sample. This method (*direct evaluation*) is prohibitively slow for most functions.

For the sake of efficiency, most digital oscillators use a stored waveform: a waveform that is evaluated prior to the generation of any sound. The computer calculates the value of many uniformly spaced points on a cycle of the waveform, and stores them in computer memory as a block called a *wave table*. Thus, a wave table consists of a long sequence of numbers, each corresponding to the sampled value of successive points on the waveform. Once the waveform has been stored, the oscillator can generate sample values by simply retrieving values from the wave table—a much faster operation for the computer than evaluating the mathematical function of the waveform directly.

To understand the operation of a digital oscillator, consider the wave table in figure 4.3, which contains one cycle of a sine wave stored in 512 numerical entries. Each entry is marked by a numerical address, denoted in this case by integers from 0 through 511. The oscillator algorithm maintains a numerical value, called the *phase*, which indicates the address of the entry currently in use. At the beginning of its operation, the oscillator

0	0.0000
1	0.0123
2	0.0245
⋮	⋮
127	0.9999
128	1.0000
129	0.9999
⋮	⋮
319	− 0.6984
320	− 0.7071
321	− 0.7157
⋮	⋮
509	− 0.0368
510	− 0.0245
511	− 0.0123

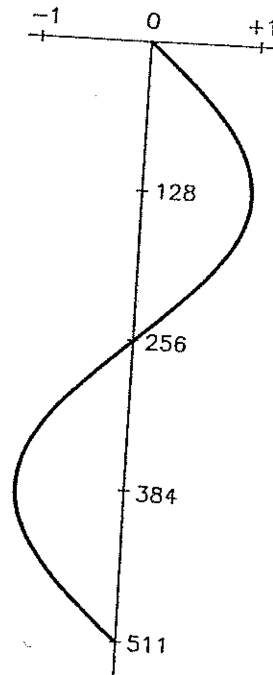

FIGURE 4.3 Wave table containing one cycle of a sine wave.

is given an initial phase value, which denotes the first entry in the wave table to be used. On every sample the oscillator algorithm obtains the current phase value (ϕ) and adds it to an amount that is proportional to the frequency of operation. The new phase value determines the entry used to calculate the next output sample. The amount added to the phase on every sample is called the *sampling increment* (SI): the distance in the wave table between successive entries selected by the oscillator. When the phase value exceeds the number of the last entry in the table, it is "wrapped around" to a point near the beginning of the table by subtracting the total number of table entries from that phase. In this example, the number of the last entry in the table is 511. If $\phi = 512$ after adding the sampling increment, then the oscillator algorithm would modify the phase so that $\phi = \phi - 512 = 0$, thereby returning the phase to the first location of the table. Hence, the oscillator algorithm can be thought of as scanning the wave table in a circular fashion.

The two varieties of digital oscillator commonly encountered in computer music are the *fixed sampling rate oscillator* and the *variable sampling rate oscillator*. The remainder of this section describes the operation of the fixed sampling rate oscillator. In modern practice, variable sampling rates are used for sound modification and will be described in section 10.3A.

Using the wave table in figure 4.3, suppose that the sampling rate is 40 kHz and the oscillator is programmed to scan through the wave table with a sampling increment of 1; that is, one entry at a time. There are 512 entries in the table and the table contains one cycle, so it would take 512 samples to produce one cycle. Therefore, the fundamental frequency of the oscillator would be 40,000 ÷ 512 = 78.13 Hz.

If a tone one octave higher is desired, the oscillator would be programmed to retrieve values from every other entry in the wave table (SI = 2). Because the oscillator would go through the wave table twice as fast, there would be half as many samples per cycle (256), and the fundamental frequency of the oscillator would be 40,000 ÷ 256 = 156.25 Hz. This result is twice as large as the previous example, which makes sense because the wave table is scanned at twice the speed.

To obtain a frequency f_0 using a wave table with N entries, the required sampling increment is

$$\mathrm{SI} = N\frac{f_0}{f_s}$$

For example, given $N = 512$ and a sampling rate (f_s) of 40 kHz, a 2.5-kHz signal would require a sampling increment of 32. In other words, if the oscillator starts at entry 0 in the wave table, sequential entries 0, 32, 64, . . . will be taken from the wave table.

Except for certain select frequencies, the sampling increment will not be an exact integer. For instance, with $N = 512$, generating a 440-Hz tone at a 40-kHz sampling rate requires a sampling increment of 5.632. Suppose, in this case, that the oscillator starts at a phase equal to 0. On the first sample, it retrieves the waveform value from that location. On the next sample, the phase is 0 + 5.632 = 5.632. How does the oscillator treat a phase with a fractional part, if the entries in the wave table are marked by integers? There are three techniques: truncation, rounding, and interpolation.

In *truncation*, the fractional part of the phase is ignored in determining the wave table entry, so that in this case the value is taken from entry 5. To calculate the next phase, however, the oscillator includes in its addition the fractional part of the current phase. Thus, on the next sample, the phase is 5.632 + 5.632 = 11.264, causing the sample to be taken from entry 11. The process continues on each successive sample.

When *rounding* is used, the entry taken is the value of the phase rounded to the nearest integer. Thus, for the example above, the first three wave table values are taken from entries 0, 6, and 11, respectively. Rounding yields a slightly more accurate waveform than truncation, and takes more computation time.

Of the three techniques, *interpolation* gives the most accurate approximation of the waveform. When a phase falls between two integer values, the waveform is calculated as a weighted average of the two entries between which the phase falls. If, as above, the phase is 5.632, the oscillator algorithm interpolates the waveform value as a weighted average of entries 5 and 6. In this case, the phase is 63.2% of the distance between 5 and 6, so the waveform would be evaluated as the sum of 63.2% of entry 6 and 36.8% of entry 5. This process can be thought of as taking the waveform value on a straight line that connects the values of successive wave table entries, resulting in a smoother waveform. Interpolation adds an extra multiplication to the oscillator algorithm and thus increases the amount of computation time.

The inaccuracies introduced in the waveform by any of the three techniques discussed previously evidence themselves as some form of noise or other unwanted signal in the sound. The amount and quality of the noise created depends on the waveform, on the table size, on the value of the sampling increment, and on the technique used. The larger the table size, the better the signal-to-noise ratio. (See section 3.2.) Let k be related to the table size (N) by $k = \log_2 N$. For example, the value $N = 512 = 2^9$ gives $k = 9$. If the entries in the table are stored with sufficient precision to prevent significant quantization noise (see section 3.2), the worst SNR that can occur is given by the approximate expressions $6k - 11$ dB for truncation, $6k - 5$ dB for rounding,[1] and $12(k - 1)$ dB for interpolation.[2] Neglecting for a moment the quantization noise contributed by the data converters, an oscillator using a 512-entry table, for example, would produce tones with no worse than 43, 49, and 96 dB SNR for truncation, rounding, and interpolation, respectively.

The actual SNR of a sound would be determined by combining the quantization noise due to the data converters and the noise resulting from fractional phase. The noise level resulting from fractional sampling increments varies directly with the amplitude of the signal. Thus, unless the noise due to fractional phase is below the level of the quantization noise, the SNR due to this effect is the same on loud sounds as it is on soft sounds.

As might be expected, the expressions above show that methods requiring more computation time or larger table size perform better. The performance of any method can be improved by increasing the table size, and so the digital-oscillator designer is faced with a common compromise: computation speed versus memory size. Many computer music systems make available both truncating and interpolating oscillators to allow the musician to make the compromise between sound quality and computation speed based on the application of a particular oscillator.

4.4 DEFINITION OF THE WAVEFORM

Generally, the musician need not directly specify a numerical value for each location in the wave table. Computer music programs enable a more simple method of entry: either by entering its representation versus time or by specifying which frequency components it contains. The definition of the waveform versus time can be made by specifying the mathematical equation that relates the amplitude of the desired waveform to its phase. The waveform versus time can also be defined by a piecewise linear means. Here, the waveform is defined by specifying a number of representative points on the waveform. These points, called *breakpoints*, are the points where the waveform changes slope. When filling the wave table, the software connects the breakpoints with straight lines. In most programs, breakpoints are specified as a pair of numbers: phase and amplitude at that phase.

The specification of waveforms in terms of amplitude versus time can, however, sometimes lead to unexpected results. If, at the frequency at which it repeats, the waveform contains any harmonics above the Nyquist frequency, they will be folded over (aliased), thereby producing unexpected frequencies in the sound. Suppose in a system with a 20-kHz sampling rate, a musician specified a sawtooth waveform (figure 4.4a) and used it in an oscillator programmed to produce a tone at a frequency of 1760 Hz. The sixth harmonic of 1760 Hz would be 10,560 Hz, which is above the Nyquist frequency of 10 kHz. Therefore, the sixth harmonic would fold over to 20,000 – 10,560 = 9440 Hz. The seventh harmonic, expected at 12,320 Hz, would sound at 7680 Hz, and so on. Figure 4.4b illustrates the intended spectrum of the sawtooth wave and figure 4.4c

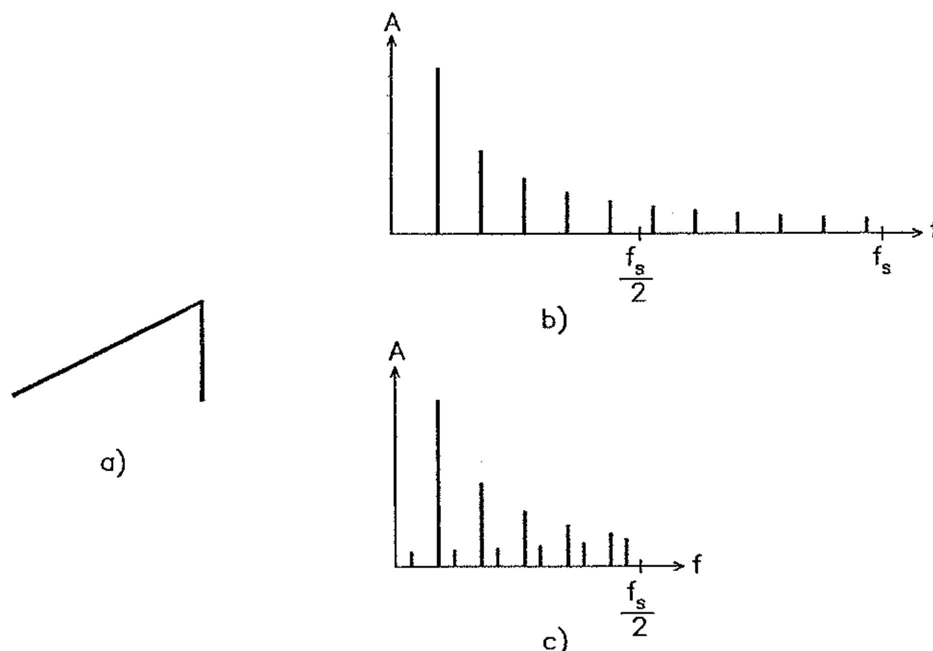

FIGURE 4.4 (a) Sawtooth waveform; (b) its expected spectrum at a fundamental frequency of 1760 Hz; and (c) its actual spectrum after conversion at a 20-kHz sampling rate.

shows how unexpected components appear in the spectrum at the output of the D/A converter. A sawtooth waveform has a significant amount of energy in its upper harmonics, and so the resulting spectrum would not sound completely harmonic. To avoid foldover when specifying waveforms in terms of amplitude versus time, one should define a waveform with little significant energy in the upper harmonics. Generally, this requires the avoidance of waveforms with steep slopes, sharp points, and other abrupt changes of slope or value (see section 2.6).

A safer way to specify a waveform is in terms of its spectrum. Here, the instrument designer specifies the amplitude, the partial number, and, if desired, the phase of each component. The software then calculates and stores a single cycle of the corresponding waveform. The amplitudes of the harmonics are typically described relative to the amplitude of the fundamental. For instance, one could specify a waveform containing a unit-amplitude fundamental with a third-harmonic amplitude 10 dB below the fundamental and a seventh harmonic 22 dB down. When the waveform is defined in terms of spectral content, the musician easily knows the exact value of the highest harmonic contained in the spectrum. Aliasing can thus be avoided by limiting the fundamental frequency of the oscillator accordingly. For example, on a system with a 40-kHz sampling rate, the fundamental frequency of an oscillator producing 10 harmonics should not exceed 2 kHz.

An oscillator that is programmed to have a fixed frequency samples the stored waveform with a constant sampling increment. This process generates a periodic waveform so that the spectrum of the signal contains nothing but exact harmonics. Thus, when describing a waveform in terms of spectral content, using noninteger partial numbers will not result in a signal with an inharmonic spectrum. Suppose an instrument designer, in hopes of obtaining an inharmonic spectrum, specified a fundamental and a partial number of 2.2. When the resulting wave table is sampled by an oscillator, the signal generated would be periodic, and therefore would have a harmonic spectrum. Instead of generating a component at 2.2 times the fundamental, the energy expected at that frequency would be spread throughout the spectrum as harmonics of the fundamental. Usually, this results in a spectrum that is not band-limited, creating the potential for noticeable foldover.

4.5 GENERATING FUNCTIONS OF TIME

Chapter 2 demonstrated that the parameters of musical sound are constantly changing. Thus, in most applications the inputs to an oscillator vary with time; that is, the amplitude and frequency of an oscillator are controlled by functions of time. An oscillator can be used to generate these control functions, but synthesis systems also include envelope generators and other function generators that, because they are tailored for this specific purpose, can synthesize control functions more directly.

Figure 4.5a shows one of the simplest computer instruments. The output of the envelope generator (figure 4.5b) controls the amplitude of the oscillator, so that the instrument produces a fixed waveform enclosed in the envelope (figure 4.5c).

The simplest amplitude envelope (figure 4.6) has three segments: the *attack*, which describes how the amplitude rises during the onset of the tone; the *sustain*, which describes the amplitude of the tone during its steady state; and the *decay*, which describes how the

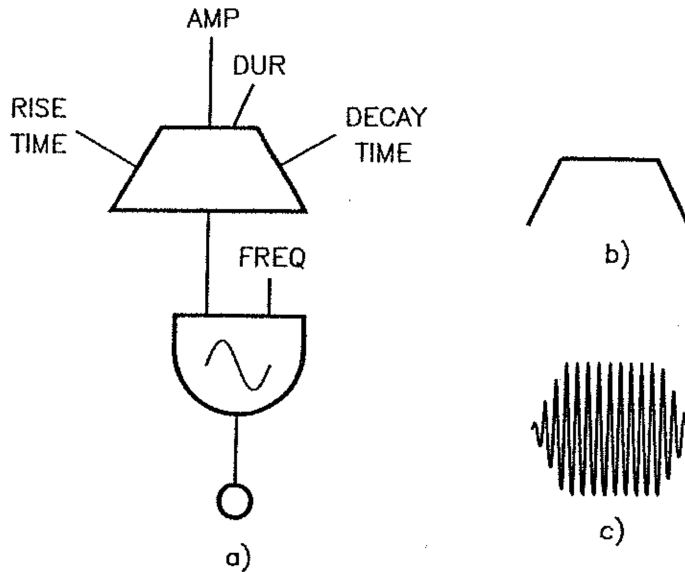

FIGURE 4.5 (a) Simple computer instrument, with its amplitude envelope (b) and its output waveform (c).

tone dies away. An envelope generator has at least four input parameters: rise time which is the duration of the attack segment, amplitude which sets the value at the peak of the attack, total duration of the envelope, and decay time. In addition, the shapes of the attack and decay segments need to be specified. Depending on the type of envelope generator, this can be done in one of two ways. Some envelope generators determine the segment

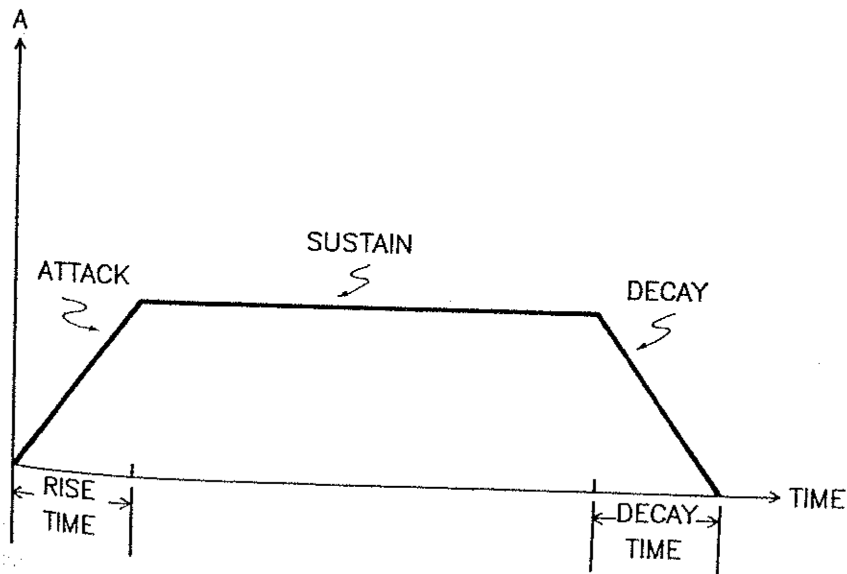

FIGURE 4.6 Simple amplitude envelope.

shape by reference to a function stored in a wave table. In this case, the entire wave table is scanned exactly once in the time of the segment. Other types of envelope generators have predetermined shapes. For example, several languages implement a unit generator called "linen," which realizes envelopes with strictly linear attack and decay segments.

On many systems, an envelope generator can be used as a signal processor. A signal is applied to the amplitude input of the envelope generator. This process results in an output signal that is the input signal encased in an envelope. The instrument of figure 4.7 is identical in function to that of the one in figure 4.5a. Instead of driving the amplitude input of the oscillator with an envelope, a constant (AMP) is applied. This causes the oscillator to produce a waveform with a constant amplitude. Passing this signal through the envelope generator imparts a pattern of amplitude variation onto the waveform. This technique is also used to enclose a digital recording of a natural sound in an envelope (see section 10.3A).

The shape of the attack and decay portions of the envelope has a great effect on the perceived timbre of a tone. Figure 4.8 depicts the two shapes most commonly encountered in computer music: linear (figure 4.8a) and exponential (figure 4.8c). Because listeners perceive amplitude on a nearly logarithmic scale, a more constant change in loudness will be obtained with an exponential shape than a linear one. Figure 4.8b and d shows how each shape progresses in terms of the logarithmic unit (decibels) which is much closer to how the ear would perceive the progression. A sound with a linear decay will appear to linger on after the beginning of the decay, and then suddenly drop off near the end of the tone. The exponential decay reflects a constant change in decibels versus time and thus will sound as a smooth diminution of amplitude. Natural vibrations almost always die away exponentially.

A true exponential shape can never reach exactly 0, and so, on many systems, the

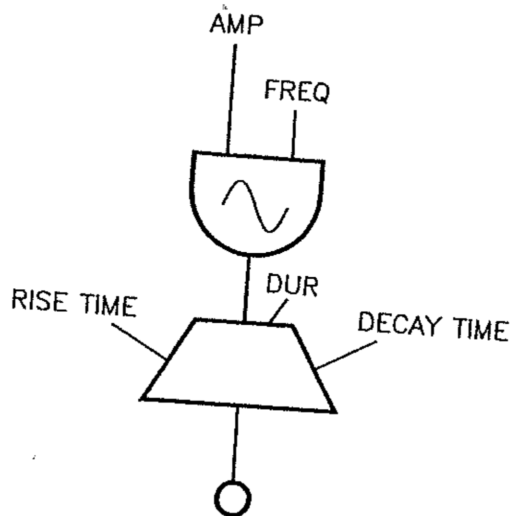

FIGURE 4.7 Another way of imparting an envelope to a signal.

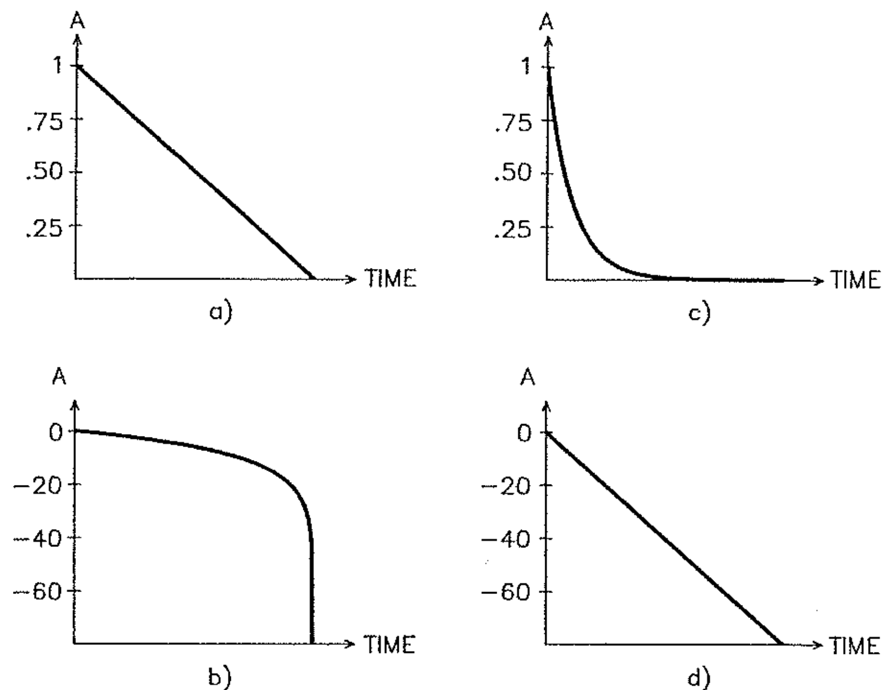

FIGURE 4.8 Decay functions: (a) linear, (b) linear in dB, (c) exponential, and (d) exponential in dB.

musician must specify the maximum and minimum values of the shape. The ratio of the two is important because it sets how quickly the amplitude changes—that is, the rate of change of the segment in dB/second. If it is desired that the minimum value result in an inaudible signal, it may not be a good strategy to make the value arbitrarily small. Suppose that an exponential attack is to last 0.1 second and the ratio is chosen as 1:1,000,000 (120 dB). This is a rate of change of $120 \div 0.1 = 1200$ dB/second. Further assume that the system has 16-bit D/A converters for a dynamic range of about 96 dB. Depending on the amplitude of the tone, the envelope will have to rise at least 24 dB before the converter begins to produce a time-varying signal. Because the envelope rises at 1200 dB/second, there will be at least a 24 dB ÷ 1200 dB/second = 0.020 second additional delay in the onset of the tone. Therefore, the ratio chosen should be no greater than the maximum amplitude value of the system—in the case of 16 bits, 32,768:1; in the general case of N bits, 2^{N-1}:1.

The duration of the attack and decay segments also has a great influence on timbre. In acoustic instruments, the attack is normally somewhat shorter than the decay. A very short attack is characteristic of percussive sounds, whereas longer attacks are found in acoustic instruments, such as the pipe organ, which produce sound by splitting a stream of air across the edge of a surface. Many acoustic instruments have longer attacks on lower pitches. Instruments that must build up a long column of air such as the tuba tend to have longer attacks. Synthesizing tones with short decays and relatively long attacks

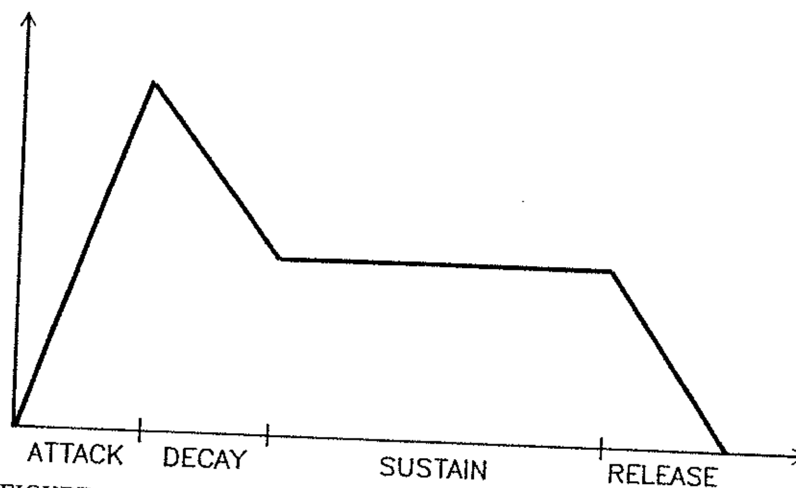

FIGURE 4.9 ADSR envelope.

produces an effect similar to playing a tape recording backwards. Of course, this may be desirable under some circumstances.

A refinement to the simple envelope generator shown in figure 4.6 is the insertion of a fourth segment between the attack and sustain. An envelope of this type (figure 4.9) is called ADSR, representing its segments—attack, decay, sustain, and release. The ADSR shape is an attempt to imitate the envelopes found in acoustic instruments and is commonly used in inexpensive electronic keyboard synthesizers. Here, the tone remains in the sustained state until the key is released.

Envelope generators on computer music systems vary in the complexity of the types of envelopes they can realize. In many systems, the available envelope generators permit envelopes with only two or three breakpoints. When the complexity of a desired envelope exceeds the capabilities of the available envelope generators, an oscillator can be used. Figure 4.10 illustrates the realization of an amplitude envelope in this way. The waveform referenced by the envelope-generating oscillator is the desired envelope shape; the frequency of the oscillator is chosen to be the inverse of the duration of the tone so that the envelope will be generated once. To obtain a smoother envelope, an oscillator that interpolates its output value between successive wave table entries (see section 4.3) is generally used.

Musicians have also used this configuration to realize musical events that are repetitions of a tone, by programming the envelope-generating oscillator to go through several cycles during the duration of the event. For example, setting the frequency of the oscillator to 3 ÷ duration produces three repetitions.

A serious disadvantage of using an oscillator instead of an envelope generator is that the attack and decay times will be altered when the duration is changed. Unless the shape of the waveform is compensated, this will cause quite noticeable differences in timbre over a range of durations.

The first use of envelope generators was to synthesize functions of time that controlled the amplitude of an oscillator. In computer music, other functions are needed to

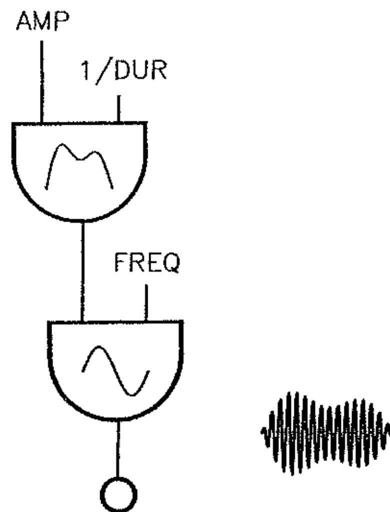

FIGURE 4.10 The use of an oscillator as an envelope generator.

control other parameters of a sound such as the frequency variation of an oscillator. As a result, many systems implement interpolating function generators to provide greater flexibility in realizing functions of time. These are often represented on a flowchart by a rectangle with a mnemonic or picture of the function inside. In using these, the musician specifies the functions of time by listing representative points on the function. For each point a pair of numbers is given: functional value and time elapsed since the previous point. (Some systems use the convention: functional value and time elapsed since the start of the function.) During synthesis, the function generator calculates values by interpolating between the breakpoints. The interpolation can be either linear or exponential, depending on the particular function generator used. For example, one could obtain a smooth glissando by specifying exponential interpolation for a function driving the frequency input of an oscillator. In this case, the function values would be given as the frequencies at the end points of the glissando.