A Project in Computer Music: The Musician's Workbench

Roger B. Dannenberg
Paul McAvinney
Marilyn Taft Thomas
Joshua J. Bloch
Dean Rubine
Marie-Hélène Serra

Computer Music is a broad term that applies to almost any application of computation to music. Because computers are almost infinitely flexible and because music is so many-faceted, the field of computer music encompasses a broad range of research, composition, performance, analysis, education, and recreation. The impact of computers on music is profound, even though computer music is in its infancy.

Historically, the path of Computer Music has been tread by the few brave souls who had enough skill and perseverance to overcome many obstacles, including the building of special-purpose hardware to generate and control sound, the creation of music-oriented software, and the development of new theories for the analysis, representation, and generation of musical sounds. With the advent of low-cost, high-power personal computers along with similar advances in music synthesizers, we are rapidly approaching a new era of computer music in which personal computer systems will replace the large studio installations of the past. Because of the astronomical advances in digital technology, these personal systems will be more powerful than even present day installations costing over a million dollars.

As the price of computing continues to fall, a number of changes in the way we interact with computers are beginning to take place. First, it is practical to dedicate an entire computer to an individual user, providing the computing power necessary for real-time control. Secondly, since computers are no longer very expensive and scarce resources, it makes sense to expend a significant amount of computing resources in order to make the system easy to use, thereby making better use of the computer user's time. For example, modern computers can display pictures, graphs, and give other visual feedback in addition to displaying high-quality text, all of which can help make the computer easier to operate. Third, advances in printer technology are making high-quality document production capabilities more affordable and hence more common.

All these factors apply to the field of computer music and have led to a proliferation of efforts to produce integrated computer music systems based on personal computers. The Musician's

© 1986 Roger B. Dannenberg
To appear in Advances in Computing and the Humanities

Workbench project at Carnegie-Mellon University is one such effort. The goal of the Musician's Workbench is to provide an environment that supports research, composition, performance, and education in music using a personal computer and a real-time synthesizer.

1. Introduction

All computer music work depends upon the ability of computers to make sound. How do they do it? In this introduction, we will attempt to provide a plausible answer without plunging into technical details. The answer begins with the observation that sound is a vibration, and sound is transmitted to our ears as rapid changes in air pressure. The atmosphere normally exerts a constant pressure, which by itself is silent, but vibrations that cause the pressure to go slightly above and slightly below the normal pressure can be perceived as sound.

These pressure variations we call sound can be represented in many ways. Consider the wiggly grooves of a record, the magnetization of recording tape, or the fluctuating voltage inside an audio amplifier. In each case, there is a direct correspondence between the representation (displacement of a record groove, magnetic intensity of a tape, or the magnitude of an electrical voltage) and the physical sound, which is just rapidly changing air pressure. When the change from normal air pressure doubles, so does the corresponding swing in the record groove or voltage in an amplifier. In other words, the deviation of the record groove about its average location is exactly proportional to the deviation of air pressure about its normal value.

Because of the close correspondence between sound and these other representations, it is possible to convert one into the other. For example, wiggles in a record groove are detected and converted into a fluctuating voltage by a phonograph cartridge. This fluctuating voltage can be made larger by an amplifier and used to drive loudspeakers. A loudspeaker converts fluctuating voltage into fluctuations of air pressure, or sound.

1.1. Digital Recording

Digital recording uses numbers (digits) to represent sound vibrations. This is accomplished by using a microphone to produce a fluctuating voltage and then measuring the voltage to produce a stream of numbers. Each number represents the amount of voltage at a single instant in time, which in turn corresponds to the air pressure at the same instant. To capture all of the sound information present in an audio signal, it is necessary to measure the voltage very frequently and very accurately. Compact disks, for example, store a number for each 23-microsecond-long[1] segment of sound, and

[1] 23 microseconds means 23 millionths of a second

the number represents a range of 65536 different pressure levels.

Once a sound has been recorded as a stream of numbers, the sound can be reproduced by using the numbers to determine a voltage. This voltage is updated at the same rate the numbers were originally recorded (every 23 microseconds in the case of compact disks). The resulting voltage is smoothed by a filter and can then be amplified to drive a loudspeaker, reproducing almost exactly the original sound. Figure 1-1 illustrates the sound-to-digital and digital-to-sound conversion process.

1.2. Digital Synthesis

Now, consider the capabilities of computers. Among other things computers are very good at numerical computation; they can be programmed to solve equations and to carry out detailed calculations repetitively. As we shall see, it is possible to program a computer to compute digital representations of sounds. The sound can then be produced in the same way that digital recordings are reproduced. The exciting difference is that computer-generated sounds have no physical origin and are therefore free from the constraints imposed by acoustic instruments. This is quite an important difference! Computer-generated sounds can be controlled and organized with the help of computer programs, giving composers almost unlimited freedoms in the kinds of sounds they can produce.

The generation of sound by computer is called *digital synthesis*. To illustrate the process, let us look at a procedure to generate a very simple sound: a sine tone with a frequency of 440 vibrations per second and lasting 2 seconds. The result will sound something like a tuning fork. Assume that our representation will use 50,000 numbers (called samples) per second of sound. An equation describing each sample is:

$$s_i = sin(2\pi i \frac{440}{50000})$$

The variable i is the number of the sample. By computing the right-hand side of the equation once for each value of i ranging from 0 to 99999, we obtain a stream of numbers that represent 2 seconds of the desired sound. Of course, musical sounds are much more complex to describe, but the principles of the digital synthesis of music are the same: *for each sample, a computation is performed to compute the desired sound pressure, and the resulting stream of numbers is converted into sound.*

The synthesis of interesting musical sounds, however, requires more sophisticated programming techniques. Digital synthesis techniques are like musical instruments; they must be controlled in order to produce specific sounds, melodies, discords, or whatever. Many computer music composers take synthesis techniques as a fixed set of resources and develop control information to give their

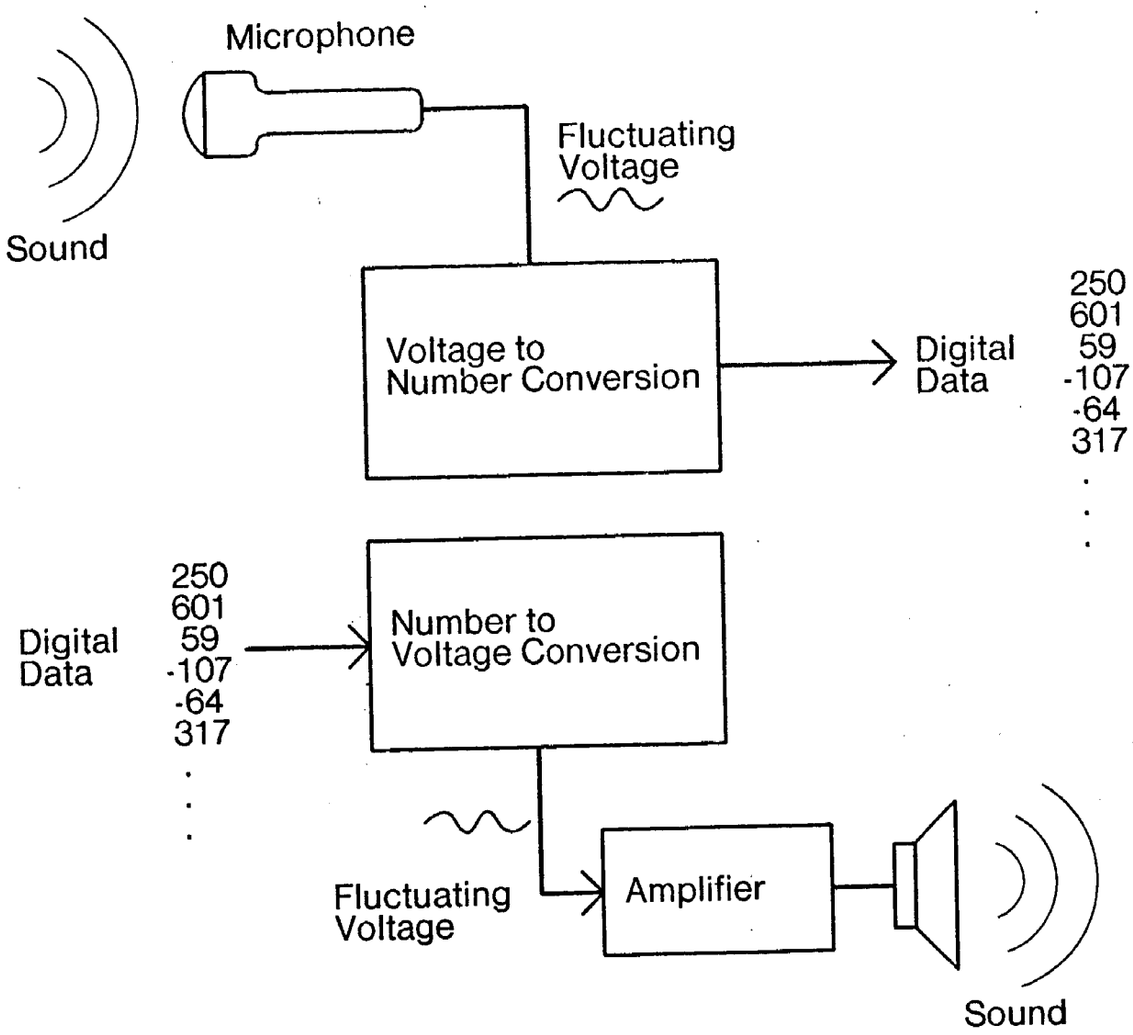

Figure 1-1: Conversion of sound into a digital recording and conversion of the recording back into sound.

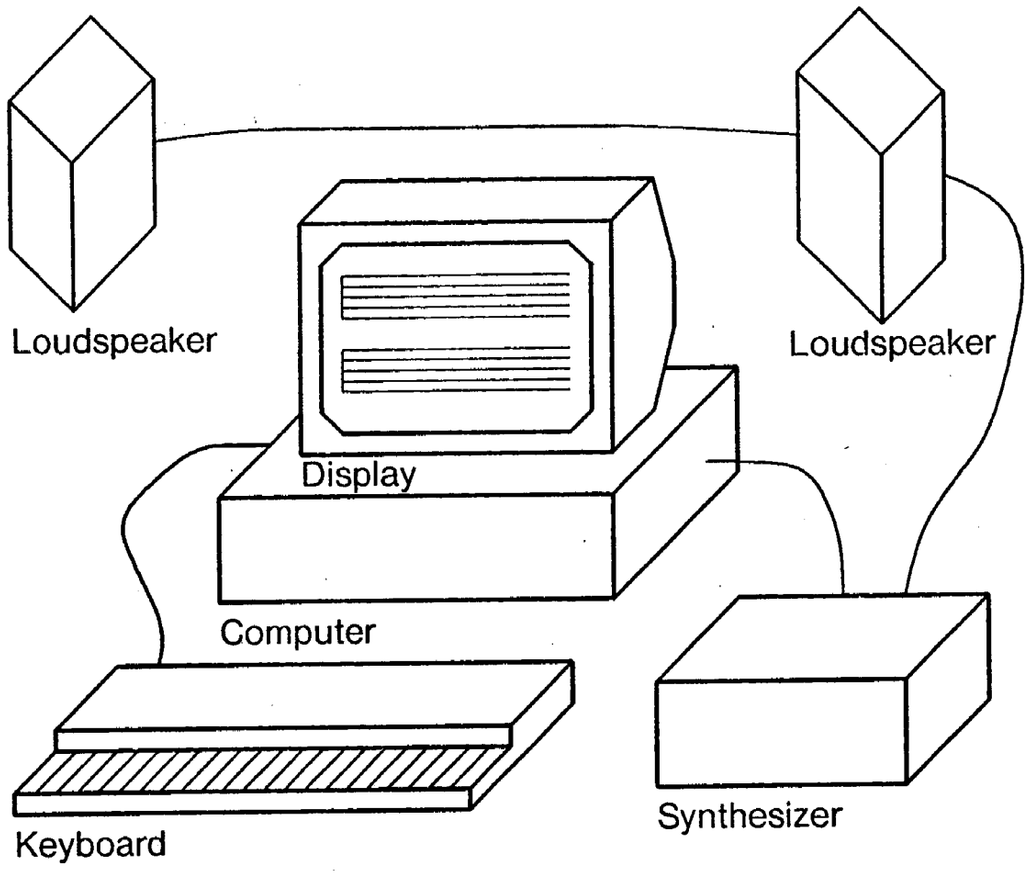

Figure 1-2: "The Musician's Workbench."

and (3) the technique should be easy to control to achieve a desired sound. It is impossible to list all of the useful techniques in this section, but three of the most popular ones are described below. Because of conflicting goals and requirements of different applications, no technique can be said to be best overall, and techniques are often combined, just as an orchestra combines strings, winds, and percussion instruments to achieve a great variety of sound.

2.1. FM Synthesis

FM or *frequency-modulation* synthesis [8] is a particularly efficient and elegant method of generating a wide variety of sounds. The simplest form uses two sine wave oscillators connected as shown in Figure 2-1. The equation describing the resulting sound is:

$$s_i = sin(2\pi f_1 \frac{i}{R} + m_i \cdot sin(2\pi f_2 \frac{i}{R}))$$

In this configuration, an oscillator (or a digital simulation of one as illustrated in the Introduction) whose frequency is f_1 is modulated or affected by another one whose frequency is f_2. (For simplicity, assume that f_1 and f_2 are fixed for the duration of the sound.) R is the sample rate. The magnitude of the modulation effect is controlled by m_i, which can vary from sample to sample. Notice that if m_i is zero, the second oscillator has no effect and the output s_i is a sine tone. However, as m_i is increased, s_i begins to deviate from a sine tone in a systematic and interesting way. Suppose that f_1 and f_2 are equal. Figure 2-2 illustrates the resulting signal for different values of m_i. The result is always a periodic signal, which could be written as a sum of sine tones of frequencies f_1, $2f_1$, $3f_1$, ..., forming a harmonic series. But what does all this sound like? In general, as m_i increases, higher harmonics increase in strength, giving a richer, fuller sound. The ability to control all these harmonics, and thus timbre, in a musically interesting way with only one parameter (m_i) is quite remarkable. Also important is the fact that a sum of many sine tones can result from only two sine oscillators, making FM a computationally inexpensive technique.

By varying the ratio of f_1 to f_2, non-harmonic signals can be generated as well. This can be useful for noisy, metallic, or percussive sounds. More complex sounds can be generated by cascading oscillators; that is, one oscillator modulates another, which in turn modulates yet another, and so on. Alternatively, greater control and complexity can be had by forming the sum of several frequency modulated oscillators. The amplitude of the result is usually controlled by multiplying each sample by a scale factor that can vary from one sample to the next.

FM can be very good for making interesting sounds at low cost; however, it also has some limitations. In particular, FM is not so good for creating realistic reproductions of specific sounds. Given a set of parameters like m_i, f_1, and f_2, we can compute the nature of the resulting sound, but

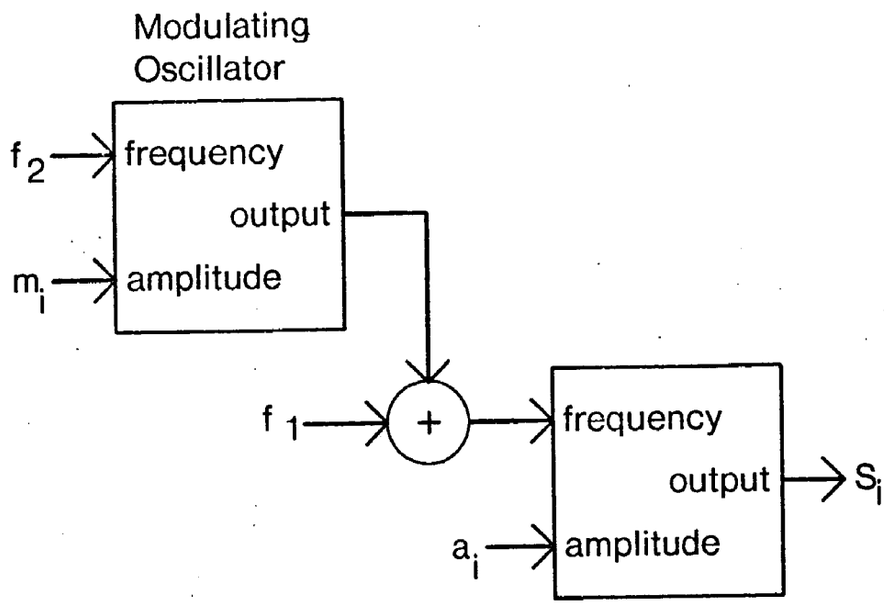

Figure 2-1: Use of two sine wave oscillators to implement frequency modulation synthesis.

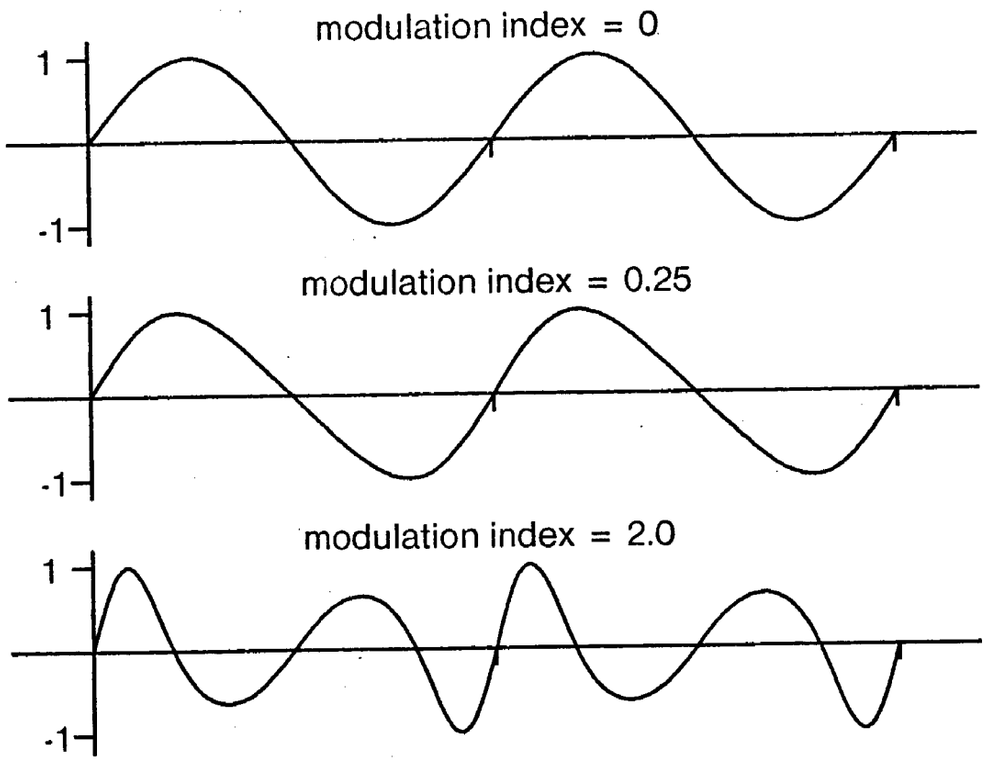

Figure 2-2: Waveforms produced using two different values of *m*, the modulation index.

given an arbitrary sound, there is no direct way to find the set of control parameters that will best reproduce that sound. Thus, one of the problems with FM synthesis is that producing a desired sound can require a large amount of trial and error.

2.2. Sampling Synthesis

The technique known as *sampling* makes it very easy to reproduce many types of sound. The basic idea is to store digital recordings of different sounds in a large digital memory. The stored data can be a recording of a real sound or it can be data generated by some other synthesis technique. Using only this basic idea, sampling would require enormous amounts of memory. Imagine recording every pitch, duration, and loudness combination playable on a piano!

To reduce the amount of memory required, sounds are typically stored in three parts. (See Figure 2-3.) The first part, called the *attack* is a recording of the first few milliseconds of the sound. The second part, called the *sustain* is a portion of the recording that is fairly steady in its sound. The *sustain* section is carefully selected so that it can be played from start-to-end repeatedly to extend the sound to an arbitrary duration. This is referred to as *looping*. Finally, the third part, called the *release* contains a recording of the end of the sound.

Besides looping, other techniques are often used to reduce the amount of data required or to represent a wider range of sounds within a given amount of memory. One trick to reduce storage requirements is to use a variable sampling rate, allowing one sound recording to be played at different pitches. Changing the sampling rate has the same effect as changing the speed of a phonograph record. Typically, the speed can only be changed slightly before the sound quality or timbre is noticeably changed, just as changing the speed of a record changes the sound quality as well as pitch. Therefore, recordings are stored for each octave or so, and the one whose original pitch is closest to the desired pitch is used for synthesis.

Another trick is to multiply the recordings by a scale factor to achieve different loudness levels from each recording. Again, this can only be applied to a limited extent because changes in loudness usually involve changes in timbre as well as the overall magnitude of the samples. In other words, when an instrument plays louder, it does not simply produce the same sound with more energy. This explains why someone shouting from far away will sound different from when they speak in a normal voice at close range, even if the actual energy reaching your ears is the same.

The expression below describes a sampling synthesis system:

given an arbitrary sound, there is no direct way to find the set of control parameters that will best reproduce that sound. Thus, one of the problems with FM synthesis is that producing a desired sound can require a large amount of trial and error.

2.2. Sampling Synthesis

The technique known as *sampling* makes it very easy to reproduce many types of sound. The basic idea is to store digital recordings of different sounds in a large digital memory. The stored data can be a recording of a real sound or it can be data generated by some other synthesis technique. Using only this basic idea, sampling would require enormous amounts of memory. Imagine recording every pitch, duration, and loudness combination playable on a piano!

To reduce the amount of memory required, sounds are typically stored in three parts. (See Figure 2-3.) The first part, called the *attack* is a recording of the first few milliseconds of the sound. The second part, called the *sustain* is a portion of the recording that is fairly steady in its sound. The *sustain* section is carefully selected so that it can be played from start-to-end repeatedly to extend the sound to an arbitrary duration. This is referred to as *looping*. Finally, the third part, called the *release* contains a recording of the end of the sound.

Besides looping, other techniques are often used to reduce the amount of data required or to represent a wider range of sounds within a given amount of memory. One trick to reduce storage requirements is to use a variable sampling rate, allowing one sound recording to be played at different pitches. Changing the sampling rate has the same effect as changing the speed of a phonograph record. Typically, the speed can only be changed slightly before the sound quality or timbre is noticeably changed, just as changing the speed of a record changes the sound quality as well as pitch. Therefore, recordings are stored for each octave or so, and the one whose original pitch is closest to the desired pitch is used for synthesis.

Another trick is to multiply the recordings by a scale factor to achieve different loudness levels from each recording. Again, this can only be applied to a limited extent because changes in loudness usually involve changes in timbre as well as the overall magnitude of the samples. In other words, when an instrument plays louder, it does not simply produce the same sound with more energy. This explains why someone shouting from far away will sound different from when they speak in a normal voice at close range, even if the actual energy reaching your ears is the same.

The expression below describes a sampling synthesis system:

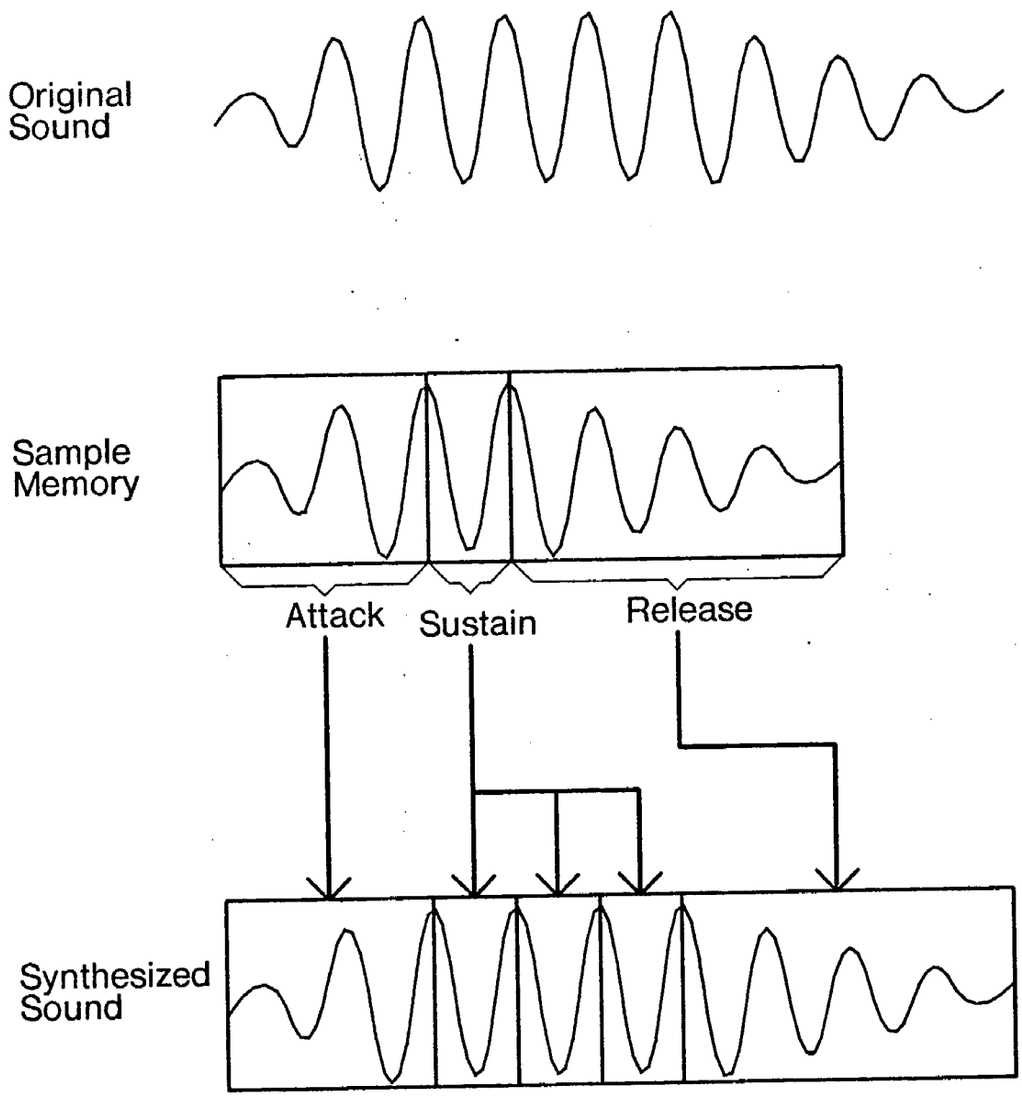

Figure 2-3: Sampling synthesis.

$$
\begin{aligned}
s_i = \ & \textbf{if } 0 \le i < a \textbf{ then } g \cdot attack_i \\
& \textbf{if } a \le i < a + nb \textbf{ then } g \cdot sustain_{(i-a) \bmod b} \\
& \textbf{if } a + nb \le i < a + nb + c \textbf{ then } g \cdot release_{i-a-nb} \\
& \textbf{otherwise } 0
\end{aligned}
$$

Here, the number of samples in the attack portion is represented by a, the sustain has b samples repeated n times, and the release has c samples. The data is scaled by gain factor g, and pitch is varied by changing the sampling rate (not shown in the equation). The expression is illustrated in Figure 2-3.

While these storage-saving techniques are necessary in practice, they tend to give less than perfect results. As implied above, when loud and soft or high and low tones are played on a piano, the sound change is more than just a matter of change in magnitude or frequency. The sound color, or timbre, changes as well. Typically, many recordings must be stored to provide an adequate range of timbres, and there is always a tradeoff between the cost of memory and the quality of synthesis. Another problem with sampling is that sounds are always produced from stored data with only a few control parameters, limiting the extent to which sounds can be molded to the desires of the composer or performer. For example, a convincing crescendo would require prerecorded samples of crescendi at different pitch and loudness levels.

2.3. Wavetable and Summation Synthesis

One of the earliest synthesis techniques is called *wavetable* synthesis. With this technique, a full cycle of a periodic signal is stored in a table, not unlike the sustain section in sampling synthesis. The numbers in the table are output repeatedly in order to produce a sustained tone. Typically, the sampling rate is fixed, and the output frequency is varied by stepping through the table by a variable distance (called the *phase increment*) rather than reading every sample. The greater the distance, the faster the table is read, and the higher the synthesized pitch. To obtain fine frequency resolution, the phase increment does not have to be a whole number; table values are interpolated or the fractional part of the calculated table location is rounded to the nearest integer to select a table value. The table must be sufficiently large so that interpolation or rounding does not introduce enough error to result in audible noise. The output of the table is then scaled by an amplitude factor. The following equations describe the computation:

$$
\begin{aligned}
P_0 &= 0 \\
P_i &= P_{i-1} + f_i \cdot \frac{n}{r} \\
s_i &= a_i \cdot table_{\lfloor P_i \rfloor \bmod n}
\end{aligned}
$$

Unlike previous examples, this one also illustrates how frequency can be varied over time. The

instantaneous frequency at sample i is f_i. It is multiplied by the table size, n, and divided by the sample rate, r, to form a phase increment, which is added to the previous phase P_{i-1} to get a new phase P_i. The phase is converted into a whole number and used to indicate where in the table to obtain the next sample of sound. The *mod n* operation keeps the phase within the bounds (0 to $n-1$) of the table. The result of the table lookup is scaled by the amplitude a_i to form the result s_i.

Figure 2-4 illustrates wavetable synthesis as a three-part computation. First, the phase is computed to form a number that serves as a location in the table. Second, a sample of the waveform is accessed from the table, and third, the sample is multiplied by the desired amplitude. Wavetable synthesis differs from looping in sampling synthesis mainly in that pitch is controlled by the phase increment rather than by varying the sample rate.

Wavetable synthesis has several advantages. First, it is inexpensive to compute (less than FM and about the same as sampling). Second, the resulting signal is produced at a fixed sampling rate, independent of pitch. A fixed sampling rate is desirable because it allows many synthesized signals to be combined into one just by adding the corresponding samples numerically. It is quite practical to generate and combine hundreds of sounds with this technique. Third, it can reproduce any periodic signal. All that is necessary is to obtain and digitize one period of a signal and then load the resulting waveform into a table.

On the other hand, few musical signals are actually periodic. In practice, most musical signals are constantly changing in shape, especially as loudness varies, and attempts to use wavetable synthesis often result in lifeless sounds.

A special case of wavetable synthesis, called *summation* or *additive* synthesis, uses only sine waves. This technique relies on the fact that any sound can be expressed as a sum of sine waves, called *partials*. To create a complex sound, a sine wave for each partial is generated. The frequency and amplitude of each sine can be independently controlled, providing great flexibility but requiring much more computation than other techniques described here. One advantage of summation synthesis is that arbitrary sounds can be analyzed by computer programs to produce a set of frequency and amplitude controls that can then be used to reproduce the original sound through summation synthesis. This control information can be manipulated in many ways to produce new sounds. For example, one could stretch the control functions to produce a longer sound without lowering the pitch. One could also create hybrid sounds by combining control functions derived from different natural sounds.

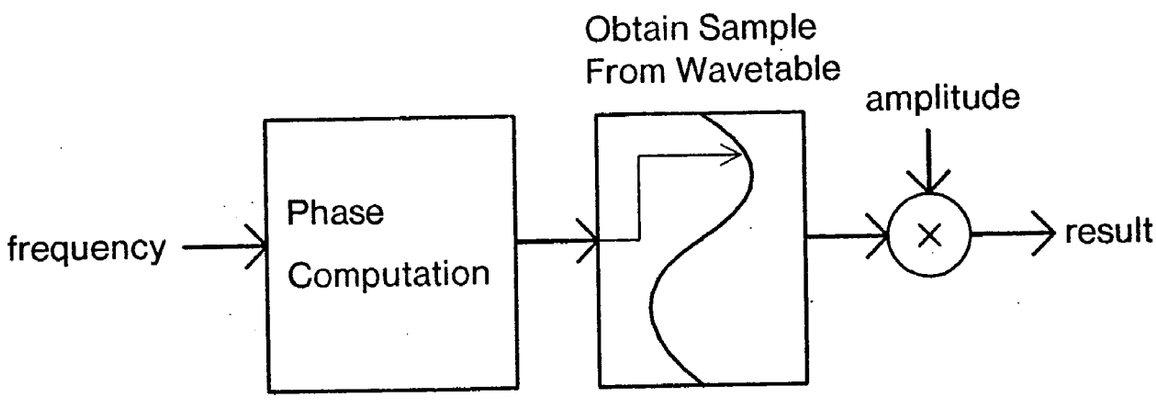

Figure 2-4: Wavetable synthesis.

2.4. Other Techniques

Many other synthesis techniques exist. In particular, subtractive synthesis and physical modeling are important classes of techniques. Subtractive synthesis recognizes that sounds, especially those of the human voice, are modified by resonating cavities. Resonances and other effects can be simulated using digital filters to modify the output of a digital oscillator. Subtractive synthesis is particularly useful for speech sounds, and the technique is particularly powerful when the filters are controlled according to information from the analysis of a real voice [16, 24, 27].

Physical modeling starts with a set of equations describing the behavior of a physical object such as a violin string. Then a program is written to solve the equations, thereby computing behavior as a function of time. The computation can be very expensive, but the result can be extremely lifelike, since many sonic details can be accounted for just by using a set of equations that are faithful to physical reality. As computation costs decline, physical modeling is likely to increase in popularity [2, 7].

2.5. Synthesis in the Musician's Workbench

One of the objectives of the Musician's Workbench is to be able to synthesize a chamber orchestra with enough realism to be useful to a composer. To meet this objective, neither FM nor sampling synthesis seems adequate. On the other hand, more general techniques like summation synthesis and physical modeling require more computation than we think is practical in the near future.

We are cooperating with Dr. Peter Comerford at the University of Bradford, England, in an effort to interface his Bradford Musical Instrument Simulator to a computer workstation. The Bradford Musical Instrument Simulator is a very high quality music synthesis device that performs about 20 million arithmetic operations per second. This device is in the early stages of optimization, and we believe this performance could be achieved with 20 to 30 integrated circuits using currently available VLSI technology. The Bradford Musical Instrument Simulator, or BMIS, is based on wavetable synthesis. The basic unit can generate 64 independently controllable sounds simultaneously, and an arbitrary number of units can be attached to a computer.

As described above, one serious problem with table lookup is that there is often no way to vary the waveform over time, which is important to the production of interesting timbres. The BMIS approaches this problem in two ways. First, it is possible to use several oscillators to produce one sound. For example, if one oscillator produces low partials and another produces high partials of a sound, then the relative mix of high and low partials can be changed as a function of time.

A second technique arises from the fact that the BMIS actually reads and combines data from two wavetables. Thus, it is possible to interpolate between any two waveforms[2]. Thus timbre can be changed dynamically by smoothly interpolating from one waveform to another. It is possible to interpolate from one waveform to a second, then from the second to a third, and so on. In this way, constantly changing timbres can be generated with inexpensive hardware.

We are currently working on a software system that can analyze real instrumental sounds and produce a description of appropriate waveforms and interpolation rates necessary to reproduce the original sound to within a given level of error. Our preliminary results have been encouraging and indicate that the technique is sound. We have succeeded in reproducing a trumpet tone quite accurately using a completely automated system for analysis and resynthesis by interpolated wavetable synthesis. It remains to be shown that the technique is efficient and general enough to satisfy our needs.

Our main objective in this area is to reduce the amount of data required to describe a sound for synthesis. There are several reasons to be concerned about the amount of data used to represent sounds. First, many descriptions must fit into the limited memory size of the Musician's Workbench computer. Second, if the volume of data is too high, the computer might not be able to send the data to a synthesizer fast enough to keep up with a performance. Finally, the synthesizer must also deal with the data, and reducing the amount of data implies less work for the synthesizer in most cases. In the next section, we will talk more about the control of synthesizers to achieve music performance in *real-time*.

3. Real-Time Control

The techniques described in the previous section can be implemented with any computer, but most computers would take more than one second to compute one second of sound. Consequently, early computer music systems computed sounds and recorded them on a disk or tape memory. Once a sound is recorded, no more computation is necessary, so playback at the desired rate is not difficult.

This style of synthesis makes human interaction difficult because the composer cannot hear the results of a change for many seconds or minutes during which sound is computed and stored. To solve this problem, researchers have built special-purpose computers that implement one or more synthesis techniques. These machines, called *digital synthesizers*, can produce sound in *real time*,

---

[2] Interpolation makes sense only when the phases of the components of the two waveforms are equal, but this is easy to arrange.

meaning that the output can be heard without delay. Moreover, a change in a control can be heard immediately, allowing composers and performers to interact with the synthesizer and receive immediate aural feedback.

The synthesis techniques in Section 2 are good examples of the type of computation that can be performed in real time by digital synthesizers. Even the best synthesis techniques, however, will sound dull without the proper control of frequencies, amplitudes, and other parameters. This higher-level control produces musical nuance such as vibrato, legato and crescendo as well as unconventional sounds for which there are no standard terms. Producing this kind of control is as difficult as designing synthesis techniques, and providing musical control in real time is an even greater challenge.

Recently, a number of researchers have recognized real-time control as an important problem, and various programming languages and specialized programs have been designed to simplify the task of controlling synthesizers. The language Arctic [11, 12, 14], designed as part of the Musician's Workbench, will be used as an illustration.

Arctic adopts the view that a real-time control system has a number of inputs, corresponding to buttons, knobs and other sensors, and a number of outputs, corresponding to the control parameters of a synthesizer. The basic function of the system is to respond to inputs by producing appropriate outputs. Arctic is a very high-level notation for describing *behaviors* or responses to input.

In Arctic, behaviors are denoted by expressions. For example, the expression "ramp" denotes a function that increases smoothly from zero to one over the duration of one second, and "unit" denotes a function that jumps from zero to one and then back to zero after one second. These functions are illustrated in Figure 3-1.

Behaviors can be modified by applying operations to them. The expression "ramp @ 2.5" is the same behavior as "ramp," except it starts 2.5 seconds later. The expression "ramp ~ 3" is also similar to "ramp" except the function is stretched by a factor of three. The shift (@) and stretch (~) operators can be applied to any expression and can also be nested. For example,

ramp ~ 3 @ 5

denotes a ramp of length three starting at time five. These functions are also illustrated in Figure 3-1.

Behaviors can be combined in various ways. The simplest combinations are arithmetic, for example "ramp + unit" is the sum of a ramp behavior and a unit behavior, resulting in a ramp from 1 to 2.

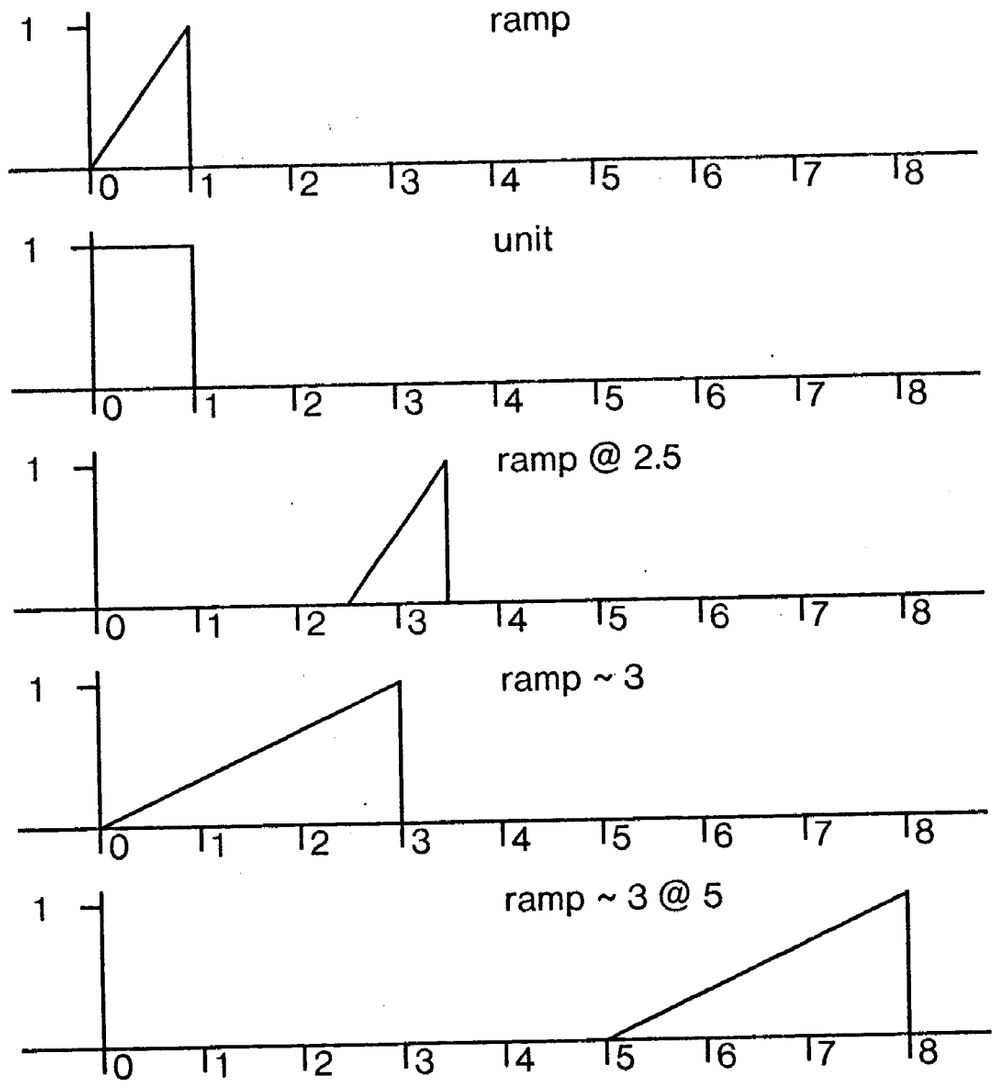

Figure 3-1: Arctic functions.

Multiplication and subtraction are also allowed. (Division is not allowed because nearly all behaviors are zero at some point and division by zero is undefined.)

Another form of combination is called the "collection," and is simply a collection or set of behaviors that happen at the same time. For example, "[ramp ; unit]" is the collection consisting of a ramp and a unit. Collections are useful in music where complex behaviors can often be described as several simpler, concurrent behaviors. If C, E, and G are behaviors that cause corresponding musical pitches, then "[C; E; G]" describes a three-note chord in which all three pitches start at the same time. The expression "[C; E@1; G@2]" denotes a chord in which the entrance of each note is delayed by one second. In other words, the E will begin one second after the C and the G will begin two seconds after the C.

Another way to achieve sequential behavior is to use another form of combination called *sequence*. In a sequence, behaviors occur one after another rather than all together. For example, "[ramp | unit]" is a behavior that ramps from zero to one (a ramp) and then stays at one for one second (a unit), as shown in Figure 3-2.

One often wants to give names to computed values. For example, the Arctic name *freq* could be associated with the frequency control of a synthesizer. To associate names with values, Arctic uses a special operation called *assignment*. Assignment is indicated by the symbol ":=" (and some others we will see later) and means that the thing on the left is defined to have the value of the thing on the right. For example,

freq := *unit* * 440

will define *freq* to have the value 440 for one second. (In Arctic, as in many computer languages, multiplication is indicated by "*".) Arctic values are communicated to the external world by assigning values to names that correspond to computer outputs. For example, *freq* might be a computer output connected to a synthesizer frequency control, and the above assignment to *freq* would set the frequency to 440 for one second.

Another form of assignment is used when a name is to be defined as the sum of several behaviors. The symbol "+=" is used; for example the behavior

[*freq* += *unit* * 440; *freq* += (*unit* @ 5) * 440]

is a collection of two behaviors separated by ";". Each behavior computes a function and adds it to *freq*. Therefore, *freq* will be the sum of the two functions. The two functions and *freq* are illustrated in Figure 3-2.

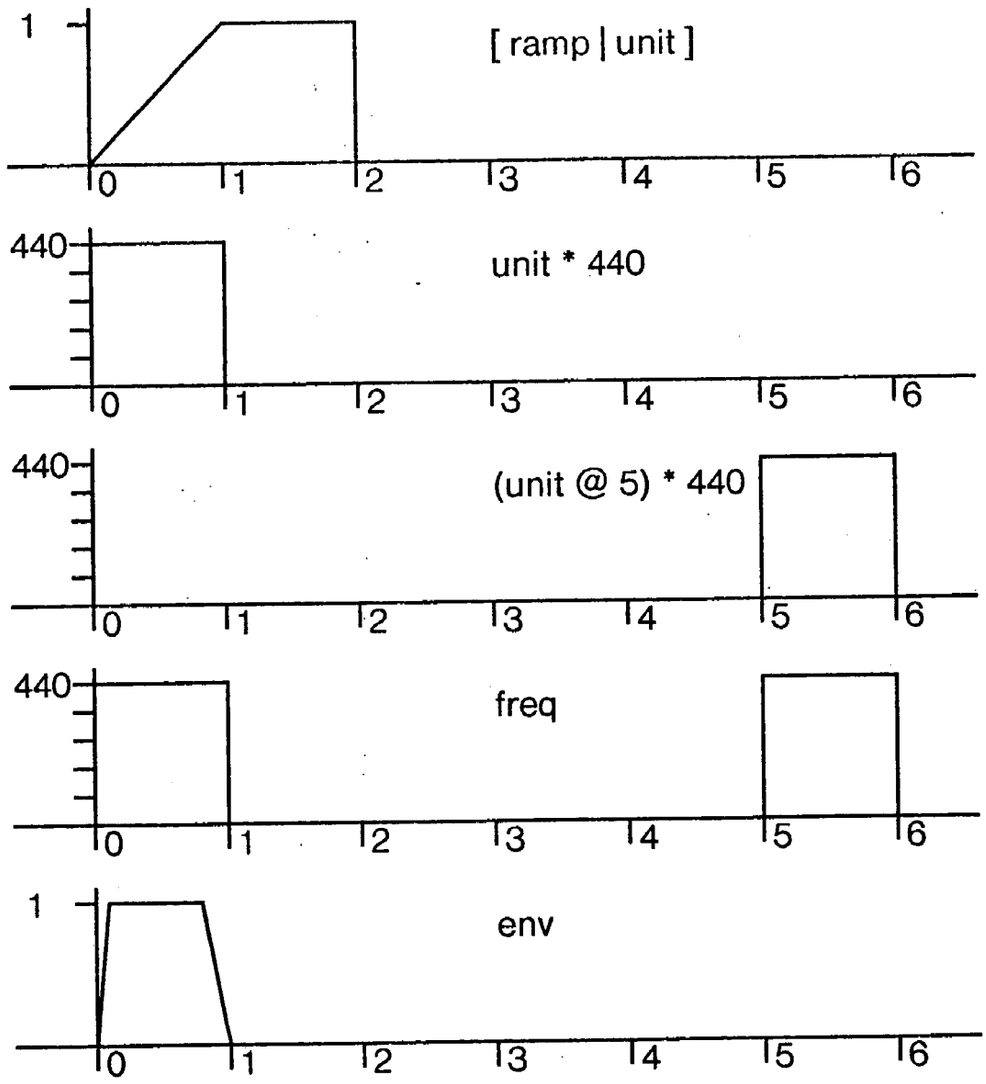

Figure 3-2: More Arctic functions.

Arctic has a number of simple built-in behaviors, but most of the time composers and computer music instrument designers need complex behaviors that are combinations of simple ones. Arctic has a mechanism whereby new behaviors can be defined, named, and then used as if they were built in. The following defines *env* to be a sequential behavior:

env **is** [*ramp* ~ 0.1 | *unit* ~ 0.7 | (*unit* – *ramp*) ~ 0.2];

Now writing *env* in an Arctic program is equivalent to writing the much longer expression after the word "**is**." The behavior of *env* is illustrated at the bottom of Figure 3-2.

Using what we have learned about Arctic, it is now possible to write an Arctic program to control a simple computer music instrument. We must assume that there are some external controls attached as inputs to the computer. The method by which these inputs are sensed and translated into Arctic functions and into the initiation of Arctic behaviors is a part of Arctic itself. The program will respond to two external controls: *key* is an event that causes a tone to be produced, and *note* is a function that tells what pitch to play when *key* occurs. The output of the program will be *freq*, the FM frequency control, *mi*, the FM modulation index, and *amp*, the FM amplitude control. Here is the program:

```
key is [
    freq + = unit * note(time);
    mi + = [ramp ~ 0.2 | unit ~ 0.8];
    amp + = env];
```

Key is defined to be a collection with three parts separated by ";". Whenever *key* occurs as a result of a physical key being pressed, the three functions are computed and used to control the FM synthesis process. To control the frequency, we take the value of *note* at the current time, denoted by *note*(**time**). *Note*(**time**) is multiplied by a *unit* to produce a steady frequency lasting one second. At the same time, the modulation index and amplitude functions are also computed.

In this example, problems would occur if the key were pressed a second time before the first note finished. Although not illustrated here, the Arctic language includes features that allow functions to be cut short as a consequence of some event. It would also be possible to use multiple FM synthesizers to produce several overlapping notes.

Rather than pursuing more realistic and more complex programs for synthesizer control, let us look at how Arctic can also be used as a composition language. As an illustration, we will write a program that plays a sequence of notes whose pitches are random but always lie between the functions *bottom* and *top*. *Bottom* and *top* could be controlled by a performer, or they could be predefined using *ramp* and *unit* functions. Sound will be produced by setting *note* and invoking *key*, which was defined above.

The program is given below.

```
NoteSequence is [
    note + = unit * random (bottom, top);
    key @ 0.1;
    NoteSequence @ 1 ];
```

The sequence is started by invoking *NoteSequence*, which is defined to be a collection of three activities. The first computes a current value for *note* by choosing a random number between *bottom* and *top*. Then *key* is invoked with a slight delay (to make sure the new value of *note* is used to determine the frequency of the sound). Finally, after a delay of one second, *NoteSequence* is invoked *recursively*; that is, NoteSequence invokes itself. This generates an endless succession of *key* events.

To produce a rapid sequence of notes, we could write *NoteSequence* ~ 0.2. The stretch operation applies to all components of *NoteSequence* and in this case would even be passed on to *key*, so everything would take 0.2 as long (five times faster), generating five notes per second. Again, this is only a short example to illustrate the flavor of Arctic. Real compositions in Arctic are much more complex, although based on the same set of principles.

The principal advantage of Arctic over more conventional languages is that it provides a compact notation that gives the programmer explicit control over timing. Arctic also simplifies the expression of behaviors that are composed of many concurrent activities. Another advantage of Arctic is that functions of time can be named and manipulated as single objects rather than a succession of values, one for each point in time. For example, we saw in the previous section how sound could be synthesized sample by sample. Normally, programs that do synthesis deal explicitly with the fact that samples are computed one after another. In Arctic, however, one word can refer to a sequence of samples, leading to a much more compact and convenient notation for describing behavior.

Arctic is one of several new languages and systems oriented especially toward the problems of music synthesis and composition. Other notable languages are FORMES [9, 25, 10] and the work by Anderson and Kuivila [3, 4].

3.1. Arctic and The Musician's Workbench

Arctic has been implemented as an interpretive system on our Musician's Workbench prototype. As Arctic programs are typed into the computer, they may be evaluated immediately to produce results that are plotted on the screen. Programs have been written to perform software synthesis using Arctic programs to determine frequency, amplitude, and other sound parameters. We are currently working on a real-time version of Arctic and hope to use it to control the Bradford Musical Instrument

Simulator. In this way, we will be able to use the Musician's Workbench as a performance instrument.

4. Computer Music Systems

We have seen how sound can be synthesized, and we have looked at Arctic as a language that can be used to control the synthesis process. In this section, we will consider how these and other components of the Musician's Workbench can be housed in an integrated system. We will begin by looking at the hardware components of the Musician's Workbench.

4.1. Musician's Workbench Hardware

The speed and capacity of the computer is an important consideration in the overall design. Because this is a large project that will take several years to develop, it makes sense to use a large and expensive computer for present development. By the time the system is ready for use, the cost of the computer will be much lower than at present. In contrast, if an easily affordable computer system were chosen for development, the development would be more difficult and the result after several years would be a system whose capabilities were far behind the state of the art. The development system executes about one million instructions per second, has two million bytes (characters) of main storage, and has a large display with a resolution of about 80 pixels (points of light) per inch.

The computer includes a character keyboard and a pointing device, called a "mouse" for input. A mouse is a small box that, when slid around on a table top, causes a cursor to move correspondingly on the computer screen. Buttons on the mouse can be pressed so that the user can easily point to and select images on the computer screen. The keyboard and mouse input devices are augmented by a music keyboard and an interface that allows the use of non-keyboard instruments and special controllers such as knobs, joysticks, trackballs, and so on.

The task of connecting music keyboards and other control devices to a computer has been greatly simplified by a standard interface called MIDI, an acronym for Musical Instrument Digital Interface [19]. The MIDI standard specifies a physical scheme for making connections and transmitting data from one device to another. MIDI also specifies how to encode data; for example, one code specifies that a key has been released, and another indicates that the volume pedal has been moved. These codes are easy to read into a computer. MIDI can also be used to control synthesizers, so the Musician's Workbench must be able to both send and receive MIDI data.

Visual output is provided by a large display that can be used to show music, text, and other graphics. For sound output, the computer will be connected to a Bradford Musical Instrument Simulator. This will allow real-time sound synthesis, giving instant aural feedback to composers and

students. The combination of high quality, low cost, and programmability of the BMIS is consistent with our goals of providing a computer music system that can serve many functions.

4.2. Musician's Workbench Software

In designing software for the Musician's Workbench, flexibility and extensibility have been guiding principles. It is impossible to foresee all the possible applications of the Musician's Workbench, so it is impossible to incorporate all the necessary functions into the initial software. On the other hand, when a composer, researcher, or educator wants to do something new and unplanned for, he should not have to start from scratch. Instead, there should be ways to modify or extend what already exists with a minimal amount of additional effort.

This flexibility can be achieved using a combination of methods. A program library will save time for programmers by allowing them to reuse existing software. For example, the library will contain programs for common operations like collecting data from keyboards and sensors, manipulating musical information, and controlling the synthesizer to produce sounds. Secondly, languages like Arctic will provide a very powerful music-oriented notation for solving problems. Programs written in Arctic should be easier to read and hence easier to modify and reuse than programs written in other languages. Third, a common representation will be used for music throughout the system. This will allow, say, a composition program to produce music which can then be printed by a music typography program. The music could also be examined on the display, using a music editor, or auditioned by running a music synthesis program. Whenever there are large collections of programs with compatible input and output formats, there is the potential for combining and using the programs in useful but unanticipated ways. These programs act as a collection of tools, hence the name Musician's Workbench.

4.3. Music Representation

If this approach is to succeed, the representation scheme for music must be designed carefully. Although Western music of the eighteenth and nineteenth centuries uses a fairly standard notation and set of conventions, modern music is more complex in several ways. First, modern composers often invent new symbols to represent unconventional sounds. These are usually accompanied by a printed set of instructions for performers. Secondly, notations of the past dealt with music at only one level, whereas computers allow us to work with music at the level of digitally recorded audio, at the level of performance interpretation, at the level of common music notation, at the level of structural outlines, or even at the level of computer programs. Third, it is sometimes desirable to represent "deep structure," or information about form and relationships (semantics). These concerns have led

to the design of a new representation schema for music within the Musician's Workbench.

The representation schema is based upon the notion that a score, or musical work can be seen as a collection of events. An event can be a note, a phrase, a typographical symbol, or almost any kind of entity, and relationships are special kinds of events. Each event has a list of attribute/value pairs. An attribute is a name like "pitch" or "duration" and a value is a symbolic or numeric piece of information like "A4" or "3 seconds." Thus, a note might be represented by the following attribute/value pairs:

```
class: note
start: 10.0
duration: 0.75
pitch: A4
instrument: trumpet
```

New attribute/value pairs can be added without restriction so that arbitrary information can be associated with events.

Relationships between events can also be expressed because events can be values. In the following example, two events labeled event 2 and event 3 are part of a phrase represented by event 1. The relationship is indicated by making event 1 be the value of the phrase attribute of events 2 and 3:

```
event 1:: class: phrase

event 2:: class: note
          phrase: event 1
          pitch: A4
          ...

event 3:: class: note
          phrase: event 1
          pitch: B4
          ...
```

We say that event 1 represents an instance of a phrase hierarchy whose members are events 2 and 3. Multiple hierarchies can be represented and new hierarchies can always be created simply by creating new attribute/value pairs.

The resulting schema is quite flexible. Current research is focused on conventions for encoding common music notation into this schema, and for encoding performance information such as exact starting times, durations, and loudnesses derived from live performances. These conventions will handle the bulk of musical information and allow information to be easily exchanged among many programs. However, there are cases where the conventions will be too restrictive, in which case new

attributes with no predefined meaning can be used. These attributes would not have any meaning to most programs, but the user could write his own programs to deal with the new attributes.

To illustrate how this extensibility can be used, suppose you want to use conventional music notation to create a score. The Musician's Workbench will provide a program, called a score editor, that lets the composer create scores in conventional notation on the computer display. Scores created with the score editor can be performed by the computer using built-in software to control a synthesizer. Suppose, however, that you want to design your own synthesizer control software. One way this can be accomplished is to write a short program that translates a score from the standard representation used by the score editor into an Arctic program. For example, the event:

```
event 23:: class: note
           start: 23.04
           duration: 0.62
           frequency: 440
           behavior: BuzzSound
           loudness: 85
           brightness: 18
```

would be translated into:

```
(BuzzSound (440, 85, 18) ~ 0.62) @ 23.04
```

Notice how the start and duration attributes are used to shift (@) and stretch (~) the Arctic behavior. These attributes are adjusted automatically by the score editor as graphic images of the notes are manipulated on the screen. Other attributes like "behavior" and "brightness" are not conventional attributes. They can be created and set using special editing commands, but they are not used to affect the display. They will, however, affect the resulting sound as desired. In this way, a composer can edit scores graphically without giving up flexibility that is essential to the creative process, nor is it necessary to write a new editing program just to deal with a few unconventional attributes.

The Musician's Workbench is a collection of hardware and software tools that are designed to work together. By taking care to make the components of the Workbench flexible and extensible, the resulting system should find a wide range of users and applications. The term "system" is purposefully chosen to indicate that the Musician's Workbench is not a single program or a solution to a single problem. Rather, it is an attempt to build a complex yet integrated computer music environment. As with any large system, we expect new components to be added and old ones to evolve over time. We will consider the Musician's Workbench a success if it finds a wide enough application to result in an overall savings of effort. We believe that advanced applications will demand the sort of foundations provided by the Musician's Workbench. In the next section, we will

look at a few examples of advanced applications of computers to music.

5. Artificial Intelligence and Music

As computers become more powerful and more sophisticated, there is a trend toward the production of intelligent programs that interact with their users at a very high level and take on difficult tasks. The field within computer science called *artificial intelligence* studies ways in which computers can be programmed to use knowledge in solving problems. A particularly successful development of artificial intelligence research has been the concept of the *expert system*, a program that has specialized knowledge in a narrow domain.

Within its domain of expertise, an expert system can achieve a high level of performance at problem-solving. This section will describe two programs that could be considered expert systems. The first is a program that harmonizes chorale melodies in the style of Bach, and the second can accompany human musicians in a live performance.

5.1. Vivace: a Harmony Expert

Vivace is an expert system that applies knowledge about harmony and good voice leading to the task of composing four-part chorales [28]. In its current form, *Vivace* is capable of taking a melody and composing a chorale for soprano, alto, tenor, and bass voices in any major or minor key using any standard meter. To accomplish this task, *Vivace* employs a combination of techniques; the important ones will be discussed below.

The primary technique is the use of rule-based programming. *Vivace* consults sets of rules to determine what notes are allowed at each decision point. There are, for example, harmonic rules governing the selection of chords within each phrase, inversion rules affecting the arrangement of notes within each chord, doubling rules governing the note to emphasize within each chord, and melodic rules pertaining to each of the four voices as it moves from chord to chord. There are also prescribed pitch ranges for each voice and specific constraints regarding the tempo and the meter of the piece. These sets of rules constitute *Vivace's* knowledge of harmony.

In addition, there are metarules governing the way in which the primary rules are implemented. Often, these metarules determine the order of preferences for a group of notes that may be chosen. For example, there is a rule that says the bass (lowest voice) should, whenever possible, move in contrary motion to the soprano (highest voice). *Vivace* selects first any note which meets this criterion from the list of available pitches. If other rules prevent the selection of this pitch, *Vivace* chooses the next closest note in contrary motion before trying any of the pitches which cause it to

move in parallel with the soprano.

The second technique is called "backtracking." Whenever *Vivace* must make a decision among several alternatives, it chooses the preferred one and moves on to the next decision. Whenever *Vivace* reaches a point where no further progress can be made, it returns, or "backtracks" to a previous decision point and makes an alternate choice. If *Vivace* considered all possibilities at each decision point, it would take a very long time to find an acceptable solution to its problem. Therefore, *Vivace* uses a large body of knowledge to make reasonable choices and to detect when an alternate decision is needed as soon as possible. Pieces of knowledge that guide decision-making in this way are called *heuristics*.

Vivace must use additional knowledge to avoid an excessive amount of backtracking. A modular approach determines the order in which it solves various subtasks. For example, *Vivace* begins by analyzing the melody and choosing a suitable chord progression. It then alternates between selection of the next bass voice with selection of the next inner (alto and tenor) voices. This approach reduces the amount of backtracking The entire process, modeled after the human method of chorale writing, includes the following modules:

Melody Writer
Harmonic Rhythm Selector
Phrase Shaper
Chord Designator
Bass Writer
Voice Assigner
Nonharmonic Tone Adder

In terms of performance, *Vivace* produces musical compositions in far less time than would be required by a human. *Vivace's* knowledge is growing through the addition of new rules and new program modules to increase its musical vocabulary and to give it new capabilities such as melody writing. These program changes must be made by human experts and require careful testing to make sure each change has the desired effect.

Vivace has given new insights into the process of writing chorales. By providing a formal framework within which knowledge can be added and tested, *Vivace* has led researchers to discover new knowledge about the chorale-writing process that is not available from any textbook on harmony. Expert systems like *Vivace* often help us to understand and formalize our own intuitive knowledge.

5.2. Computer Accompaniment

The second example of intelligent music systems is the computer accompaniment of human musicians [13, 29, 6, 30, 17]. This technique was developed to allow computers to fill the traditional role of performer or accompanist in live performances. Previously, live performances with computers used the computer as a musical instrument controlled and synchronized by a human player, or as an elaborate music box, which human players had to follow in order to maintain synchronization.

In contrast, a computer accompaniment system listens to the live performer and is capable of adjusting the tempo and making other changes in real time. Just as with human accompanists, the composer provides the computer with a score containing a description of what the human *soloist* is to perform as well as an accompaniment score. The accompanist does not improvise or compose music, but it must play its part in synchrony with another player.

Computer accompaniment involves several concurrent tasks as shown in Figure 5-1. The first task listens to the human performer and converts sound into a symbolic representation to be used in the next task. This representation typically contains only important information such as the starting times and pitch names of each note.

The second task takes this information and compares it to information in the score, searching for a correspondence between the live performance and the composed score. Because mistakes as well as intentional variations are inevitable in a performance, the comparison must be tolerant of slight variations from the ideal. A match between the actual performance and the written score tells the accompanist the location of the performer in the score. This information is passed to the third task and is necessary for the accompanist to stay synchronized with the performer.

The third task determines the timing of all accompaniment notes. This timing is "flexible" in that it may be necessary to speed up, slow down, move forward, or move backward in time to stay with the live performer. An adjustable clock is used to keep time. If the accompaniment falls behind, the clock is set forward and its speed is increased slightly. If the accompaniment gets ahead, the clock is set back and its speed is decreased. The clock allows the accompanist to perform accompaniment notes that occur between notes of the solo. Thus, the computer does more than just respond to inputs; it also maintains an internal sense of time and performs its part accordingly.

The portion of the computer accompaniment system that adjusts the clock uses knowledge about accompaniment to make its changes as musical as possible. For example, if the accompaniment system falls slightly behind, it is better to rush to catch up than it is to skip any notes. On the other

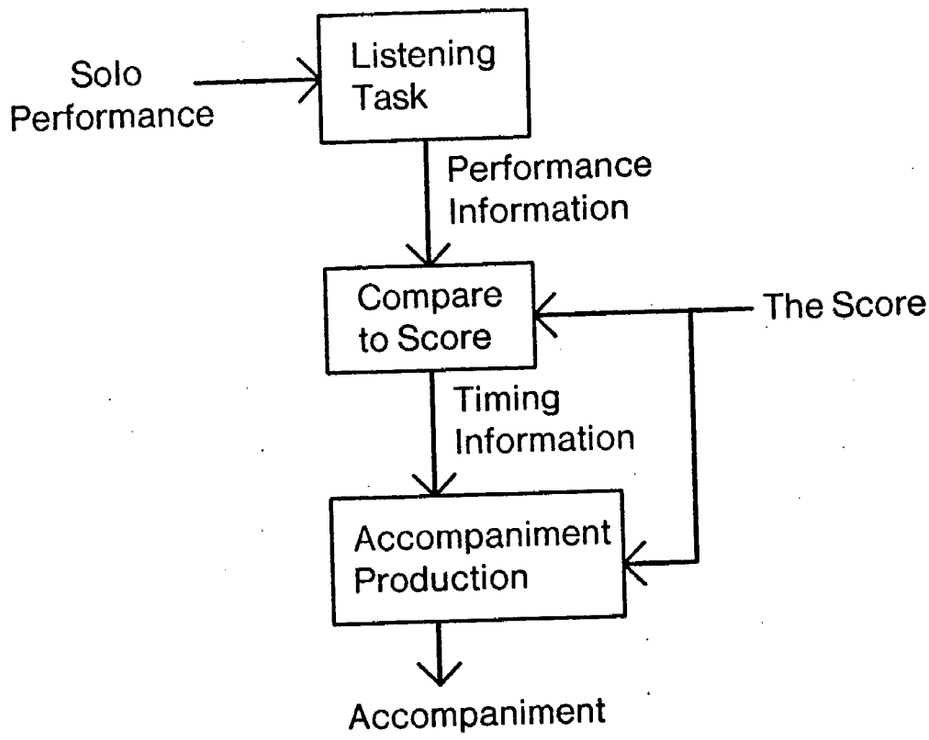

Figure 5-1: Structure of a computer accompaniment system

hand, if the accompaniment is far behind, it makes sense to skip notes rather than play an entire phrase at breakneck speed. Rules like these are used to represent accompaniment knowledge and give rise to an intelligent musical accompanist.

5.3. Future Work

Artificial intelligence applications to music continue to attract researchers [23]. In the future, we can expect to see computer programs that are competent in writing a fugue, joining a "jam session" consisting of humans and computers, and performing works by great composers with musical expression. In fact, all of these tasks have been attempted already, but the determination of "competence" is very subjective. Certainly, much work is left to be done before computers can provide the level of performance achieved by humans. One of the great by-products of this research is a better understanding of music and musical thought. In the case of at least one program, *Vivace*, this improved understanding has led to better ways of teaching.

6. Summary

In the Introduction, we saw how sound could be converted into numbers and back into sound. This change in representation allows computers to work with sound and to create sounds from programmed instructions. In Section 2, we examined several techniques for creating musical sounds. Each technique represents some trade-off among the goals of high-quality sound, low cost, and ease of use.

Some of these synthesis techniques have been incorporated into special computers called digital synthesizers, which can produce sound in real-time, that is, without delay. In many cases, it is not so much the synthesis technique, but the careful control of frequency, amplitude, and other control parameters that makes for effective and musical sounds. As a consequence, an important area of current research is the development of better ways to control these synthesizers. The language Arctic was considered to illustrate how a computer language can help to achieve sophisticated control of a synthesizer in real time.

Given the falling costs associated with computers and digital synthesizers, it is now practical to contemplate powerful personal computer music systems. This represents a radical departure from the systems of the past that were operated in major studios. Section 4 discussed the design of one such system, the Musician's Workbench, and showed how appropriate hardware, augmented by a collection of software tools, could support musical activities.

In Section 5, we looked at two systems in which artificial intelligence techniques were applied to

musical applications. One of these is an expert at harmonizing chorales, and the other can accompany a human performer in a live performance. In addition to providing more powerful or easier to use computer music systems, the design of intelligent music systems can increase our own knowledge about music.

For more information on these and other topics, there are several sources of information. The Computer Music Journal (MIT Press) and the Proceedings of the International Computer Music Conferences (Computer Music Association, P.O. Box 1634, San Francisco, California, USA 94101) are good sources of information on the state-of-the-art in computer music. For those with a computer science background, an issue of ACM Computing Surveys [1] is devoted to computer music. A less technical introduction to the techniques and compositional ideas underlying electronic music can be found in a book by Barry Schrader [26]. Although this book discusses electro-acoustic music, most of the material is applicable to computer music as well. Also of a non-technical nature is an article by Paul Lansky [16].

Two collections of early work in computer music are *Music by Computers* [15] and *The Computer and Music* [18]. Both contain contributions, mostly about technical matters, by early computer music researchers. Max Mathews, often considered the founder of computer music (Mathews reported work on computer music over 20 years ago [20]) wrote a classic monograph [21] on the subject which is still a useful discussion of digital to analog conversion, analog to digital conversion, and software synthesis techniques. An introductory article by Moorer [22] covers a somewhat wider range of topics in less detail. A more detailed discussion of practical aspects of digital audio is contained in an article by Blesser [5].

7. Conclusion

We have looked at a broad spectrum of topics in computer music as they relate to the Musician's Workbench, but this chapter is by no means complete as a survey of the field. We have ignored many important areas, including musical perception, aesthetics and compositional considerations, many signal-processing techniques and music theory.

Given the broad range of applications of computers in music, it is interesting to speculate on the role of the computer in music and on the impact of new developments. Because of the pervasive influence of computers in music, computers may cause a change in Western music comparable to the formation of the orchestra. There are, in fact, some striking parallels. The modern orchestra has provided a powerful and flexible "instrument" for the composer. As a new medium, the orchestra has doubtless had a major impact on musical thinking and musical practice. Like the orchestra, the

computer is a powerful and flexible new medium. It has a set of distinctive characteristics that will have profound effects on music of the future. Perhaps the most important characteristic of computers is that they manipulate symbols. Unlike electronic media and recording, computers can deal with representations of music at many levels of abstraction. Music can be represented as a set of related structures, as a program, as a graphic image, or as digital audio. This flexibility of representation will lead to new techniques for making music and new conceptualizations of the resulting product.

Computers are also capable of great precision and repeatability. Sounds can be modified so slightly that the change is barely perceptible, yet any change can be reproduced exactly time after time. This promotes a style of incremental change and development of musical ideas because ideas, techniques, and knowledge can be stored and recalled later with no loss of information. Precision also facilitates studies of musical perception, adding to our understanding of human capabilities and limitations.

Computers are inexpensive compared to human resources. No composer can afford a private orchestra, but even today many composers own personal computer music systems. As computers increase in speed and power, we will reach a point where the computer provides musical resources that compare in flexibility and interest to those of the orchestra. At this point, there will be tremendous economic and pragmatic incentives for many composers to adopt the computer as their primary medium. (Some would argue that this point has already been reached.) This is not to say that all composers will suddenly abandon traditional forms and begin writing computer music; rather, we can expect increasing numbers of composers to write computer music as a substantial part of their creative work.

Computers will provide composers and listeners with a personal "orchestra" of sounds, techniques, and processes. The resulting experimentation and alternative musical experience is bound to affect the practice of music. The prospect of such a revolution in music is somewhat frightening, and one wonders whether the overall result will be beneficial. One might speculate that traditional music would be rendered obsolete, but this seems highly unlikely given the continued popularity of virtually every known musical form. Each one of these forms survives because it offers the listener something that is neither duplicated nor improved upon by other forms. Although the computer offers many advantages, it is unlikely to completely duplicate the aural, visual, and social qualities of existing media.

One might also wonder if economic pressures would bring an end to live performances of music. A

look at the present situation, however, reveals that live performance is still the mainstay of orchestras and other ensembles even after the advent of radios, phonographs, film and television. In fact, many orchestras take advantage of these relatively new technologies. We should expect a period of exploration and integration of computer music with other forms, but it seems unlikely that computers will bring an end to music as we know it today.

It seems almost certain, however, that the computer will continue to make many positive contributions to music. Because new low-cost computers will be as common as television sets, there is a great likelihood that many non-musicians will have the resources to try their hand at composition and performance. Likewise, skilled composers will be able to practice and perfect their craft, getting constant aural feedback without struggling to have their compositions read by an orchestra. The nature of composition may change so that compositions interact dynamically with the listener, allowing every listener to obtain a personalized performance. Furthermore, intelligent interfaces may allow amateur performers and conductors to achieve more professional results, increasing the rewards of their involvement in music. The potential for personalization, popularization, and growth of music through the medium of computers promises a new music renaissance. We hope that the Musician's Workbench and systems like it will help musicians, educators, and researchers to take full advantage of computers in their work.

8. Acknowledgments

The authors wish to thank Gareth Loy, Dick Moore, Allen Newell, Dana Scott, and Herb Simon for their influential insights into the nature of computers and computer music. Thanks are also due to the Music Department, the Computer Science Department, and the Center for Art and Technology at Carnegie-Mellon University for their support of this project.

References

[1] Curtis Abbot, editor.
Computing Surveys.
ACM, 1985.
No. 2 (June).

[2] Jean-Marie Adrien and Xavier Rodet.
Physical Models of Instruments: A Modular Approach, Application to Strings.
In *Proceedings of the 1985 International Computer Music Conference*, pages 85-96.
Computer Music Association, 1985.

[3] David P. Anderson and Ron Kuivila.
Accurately timed generation of discrete events.
June, 1985.

[4] David P. Anderson and Ron Kuivila.
Continuous abstractions for discrete event languages.
June, 1985.

[5] Barry A. Blesser.
Digitization of Audio: A Comprehensive Examination of Theory, Implementation, and Current Practice.
Journal of the Audio Engineering Society 26(10):739-771, October, 1978.

[6] Joshua J. Bloch and Roger B. Dannenberg.
Real-Time Computer Accompaniment of Keyboard Performances.
In *Proceedings of the 1985 International Computer Music Conference*, pages 279-290. Computer Music Association, 1985.

[7] John Kitamura, William Buxton, Martin Snelgrove and Kenneth C. Smith.
Music Synthesis by Simulation using a General-Purpose Signal Processing System.
In *Proceedings of the 1985 International Computer Music Conference*, pages 155-158. Computer Music Association, 1985.

[8] John M. Chowning.
The Synthesis of Complex Audio Spectra by Means of Frequency Modulation.
Journal of the Audio Engineering Society 21(7), September, 1973.

[9] P. Cointe and X. Rodet.
FORMES: An Object and Time Oriented System for Music Composition and Synthesis.
In *1984 ACM Symposium on LISP and Functional Programming*, pages 85-95. ACM, August, 1984.

[10] Pierre Cointe and Xavier Rodet.
Formes: an Object & Time Oriented System for Music Composition and Synthesis.
In *1984 Symposium on LISP and Functional Programming*, pages 85-95. ACM, 1984.

[11] Roger B. Dannenberg.
Arctic: A Functional Language for Real-Time Control.
In *1984 Symposium on LISP and Functional Programming*, pages 96-103. ACM, 1984.

[12] Roger B. Dannenberg and Paul McAvinney.
A Functional Approach to Real-Time Control.
In *Proceedings of the 1984 International Computer Music Conference*, pages 5-16. Computer Music Association, 1984.

[13] Roger B. Dannenberg.
An On-Line Algorithm for Real-Time Accompaniment.
In *Proceedings of the 1984 International Computer Music Conference*, pages 193-198. Computer Music Association, 1984.

[14] Roger B. Dannenberg.
Arctic: Functional Programming for Real-Time Systems.
In *Proceedings of the Nineteenth Hawaii International Conference on System Sciences*, pages 216-226. IEEE, 1986.

[15] Heinz Von Foerster and James W. Beauchamp, ed.
Music By Computers.
John Wiley & Sons, Inc., 1969.

[16] P. Lansky and K. Steiglitz.
The synthesis of timbral families by warped linear prediction.
Computer Music Journal 5(3):45-49, Fall, 1981.

[17] John Lifton.
Some Technical and Aesthetic Considerations in Software for Live Interactive Performance.
In *Proceedings of the 1985 International Computer Music Conference*, pages 303-306.
Computer Music Association, 1985.

[18] Harry B. Lincoln, ed.
The Computer and Music.
Cornell University Press, Ithaca, N.Y., 1970.

[19] Gareth Loy.
Musicians Make a Standard: The MIDI Phenomenon.
Computer Music Journal 9(4):8-26, Winter, 1985.

[20] M. V. Mathews.
The Digital Computer as a Musical Instrument.
Science 142:553-557, 1963.

[21] M. V. Mathews.
The Technology of Computer Music.
MIT Press, Cambridge, Massachusetts, 1969.

[22] J. A. Moorer.
Signal Processing Aspects of Computer Music – A Survey.
Proceedings of the IEEE , July, 1977.

[23] C. Roads.
Artificial Intelligence and Music.
Computer Music Journal 4(2):13-25, Summer, 1980.

[24] Xavier Rodet, Yves Potard, Jean-Baptiste Barriere.
The CHANT Project: From the Synthesis of the Singing Voice to Synthesis in General.
Computer Music Journal 8(3):15-31, Fall, 1984.

[25] X. Rodet and P. Cointe.
FORMES: Composition and Scheduling of Processes.
Computer Music Journal 8(3):32-50, Fall, 1984.

[26] Barry Schrader.
Introduction to Electro-Acoustic Music.
Prentice Hall, New Jersey, 1982.

[27] A. W. Slawson.
Sound Color.
University of California Press, Berkeley, 1985.

[28] Marilyn Taft Thomas.
Vivace: a rule based AI system for composition.
In *Proceedings of the 1985 International Computer Music Conference*, pages 267-274.
Computer Music Association, 1985.

[29] Barry Vercoe.
The Synthetic Performer in the Context of Live Performance.
In *Proceedings of the 1984 International Computer Music Conference*, pages 199-200.
Computer Music Association, 1984.

[30] Barry Vercoe and Miller Puckette.
Synthetic Rehearsal: Training the Synthetic Performer.
In *Proceedings of the 1985 International Computer Music Conference*, pages 275-278.
Computer Music Association, 1985.

Table of Contents

1. Introduction 1
    1.1. Digital Recording 1
    1.2. Digital Synthesis 2
2. Music Synthesis Techniques 4
    2.1. FM Synthesis 6
    2.2. Sampling Synthesis 9
    2.3. Wavetable and Summation Synthesis 11
    2.4. Other Techniques 14
    2.5. Synthesis in the Musician's Workbench 14
3. Real-Time Control 15
    3.1. Arctic and The Musician's Workbench 21
4. Computer Music Systems 22
    4.1. Musician's Workbench Hardware 22
    4.2. Musician's Workbench Software 23
    4.3. Music Representation 23
5. Artificial Intelligence and Music 26
    5.1. Vivace: a Harmony Expert 26
    5.2. Computer Accompaniment 28
    5.3. Future Work 30
6. Summary 30
7. Conclusion 31
8. Acknowledgments 33

List of Figures

Figure 1-1: Conversion of sound into a digital recording and conversion of the recording back into sound. 3
Figure 1-2: "The Musician's Workbench." 5
Figure 2-1: Use of two sine wave oscillators to implement frequency modulation synthesis. 7
Figure 2-2: Waveforms produced using two different values of *m*, the modulation index. 8
Figure 2-3: Sampling synthesis. 10
Figure 2-4: Wavetable synthesis. 13
Figure 3-1: Arctic functions. 17
Figure 3-2: More Arctic functions. 19
Figure 5-1: Structure of a computer accompaniment system 29