A Project in Computer Music: The Musician's Workbench

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

Abstract. The Musician's Workbench was the focus of a number of interrelated computer music research projects at Carnegie Mellon University. A central issue has been expressive control of music synthesis, performance, and composition. At the synthesis level, new techniques were explored for controlling timbre, making possible the synthesis of expressive performance. To control synthesis parameters, the language Arctic was developed. Arctic emphasizes the composition of behaviors and timing relationships, using hierarchical descriptions to make complex behaviors manageable. To handle still higher levels of abstraction, a structured editor for music notation was designed. The editor is based on the idea that a score can be seen as a collection of events, each described by attribute/value pairs. Multiple hierarchies of interrelated events express "deep" semantic structure as well as the surface features of notation and performance parameters. The highest level of interaction between humans and computers is limited by the ability of computers to interact with humans at a near-human level of understanding of musical concepts. Some preliminary studies of intelligent computer composition and performance are discussed. Many of the issues explored in the Musician's Workbench are still important and open research topics today.

1. Introduction

Computer Music is a term that refers to 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 have reached a new era

of computer music in which personal computer systems have largely replaced the large studio installations of the past. Because of the astronomical advances in digital technology, these personal systems are more powerful than earlier installations that originally cost over a million dollars.

As the price of computing continues to fall, a number of changes in the way we interact with computers have taken 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 have made 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 Workbench project was formed at Carnegie Mellon as one such effort. The goal of the Musician's Workbench was to provide an environment that supports research, composition, performance, and education in music using a personal computer and a real-time synthesizer. Over the years, some of our plans were successful, some are still in progress, and others have fallen by the wayside. This chapter describes our effort and relates it to some of the central issues of computer music research today.

Looking back, the Musician's Workbench project is a reflection of the ideas and technologies available in the 1985-1990 timeframe, but many ideas have yet to reach common practice. The "personal computer revolution", while making much technology very accessible, has been directed largely by market demands. As a consequence, many ideas that are of interest to artists and/or scientists are not supported by readily available hardware and software. This situation creates the main justifications for developing computer music systems in an academic as opposed to a commercial setting. The special needs of artists and scientists can be addressed without concern for monetary profit. Research prototype systems can be used to explore new ideas and demonstrate new concepts. Most of the features of commercial computer systems, including computer music systems, were first seen in research prototypes. Perhaps the most striking example is the pioneering use of digital audio for music by Max Matthews in the 1950's. It took many years before digital audio became viable for commercial music synthesizers and audio recordings (the compact disc).

This chapter begins with an introduction to the technology of producing sounds by computer. Then, we describe our original vision of the integrated music workstation, the "Musician's Workbench". In reality, the Musician's Workbench Project became an "umbrella" or a unifying theme for a number of individual research efforts, each of which addressed some problem area. In retrospect, our analysis of the problems are as interesting as the (still partial) solutions. One of the main goals of this chapter is to convey a sense of the important issues that concern the practitioners of computer music. Although the technology changes rapidly, and interesting discoveries are made continuously, several big issues have remained fairly stable and consistent. In the light of all this, the concept of the Musician's Workbench is as viable today as when we began. We have recently begun a new project, based on what was learned from the Musician's Workbench. We will explain the changes in direction we have made and speculate on the nature of future systems for computer music.

1.1. Computers and Sound

How do computers make sound? In this introduction, we will attempt to provide a plausible answer without plunging into technical details. At first, we will be concerned with the lowest or most detailed level of representation, that is, sound itself. This might be compared to the *image* of printed text as opposed to its content. Higher, more abstract levels of representation will be considered in Sections 3, 4.3, and 5. One of the interesting characteristics of computer music is that many different levels of representation are important. Readers who are familiar with digital audio concepts may wish to skip this section.

Let us begin 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. These representations are called *analog* because the displacement of (say) the record groove is analogous to the change in air pressure.

1.1.1. 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 that 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 discs, for example, store a number for each 23-microsecond-long[1] segment of sound, and the number represents a range of 65536 different pressure levels. [Pohlmann 92]

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 discs). 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.

There are many reasons to use digital representations. First of all, high-precision analog systems are difficult to design and construct. Noise, static, and distortion are all the result of imperfect analog recording, transmission, and reproduction. In contrast, precision in digital systems depends only on the number of digits (bits). The second reason is that digital systems allow perfect reproduction. With analog recordings, each reproduction necessarily suffers a slight degradation. In contrast, digital systems rely on a symbolic representation that rarely suffers any degradation. For example, if you spill coffee on this page, you can still read and reproduce the text perfectly. This is because each symbol (characters of text) retains its exact identity even when slightly distorted or discolored. For the computer musician, the

[1] 23 microseconds means 23 millionths of a second

biggest advantage of digital representation is that they are inherently compatible with digital computers, allowing computers to directly create and manipulate audio information.

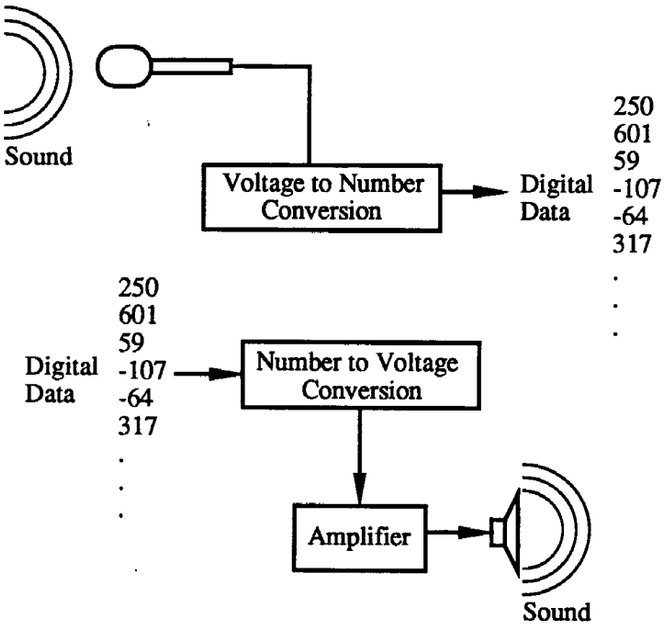

Figure 1-1: Conversion of sound into a digital recording and conversion of the recording back into sound.

1.1.2. Digital Synthesis

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 the compact disc standard of 44,100 numbers (called samples) per second of sound. The formula for each sample s_i is:

$$s_i = sin(2\pi i \frac{440}{44100})$$

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 88,200 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.*

1.2. Relating Representations to Sound

The synthesis of interesting musical sounds, however, requires more sophisticated representations and 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 music character and interest. To give a simple example, the human voice is incapable of holding a perfectly steady pitch. Instead, there are constant subtle variations in pitch and loudness that are important voice qualities. When tones are synthesized it is often desirable to give them similar subtle variations, and a good deal of research has been devoted to the problem of producing such "natural" sounding variations. In fact, computing appropriate control information has turned out to be as difficult as digital sound generation.

This pitch-variation example illustrates the importance of representations. Consider the waveform representation of the short vocal sound in Figure 1-2, to which a quite audible vibrato (pitch modulation) has been added artificially. In the figure, pitch variation is not readily apparent. Since vibrato takes place over many periods of oscillation, it is even less visible in a more detailed view such as Figure 1-3. Now consider the vibrato function in Figure 1-4, which isolates and displays pitch variation, making the nature of the vibrato quite clear. Figure 1-5 illustrates another parameter, the amplitude envelope of a sound. These sorts of *parametric* descriptions of sounds are used to control synthesis computations. They enable composers to obtain fine control over many details of sounds, varying perhaps just one parameter at a time, and working with an appropriate representation.

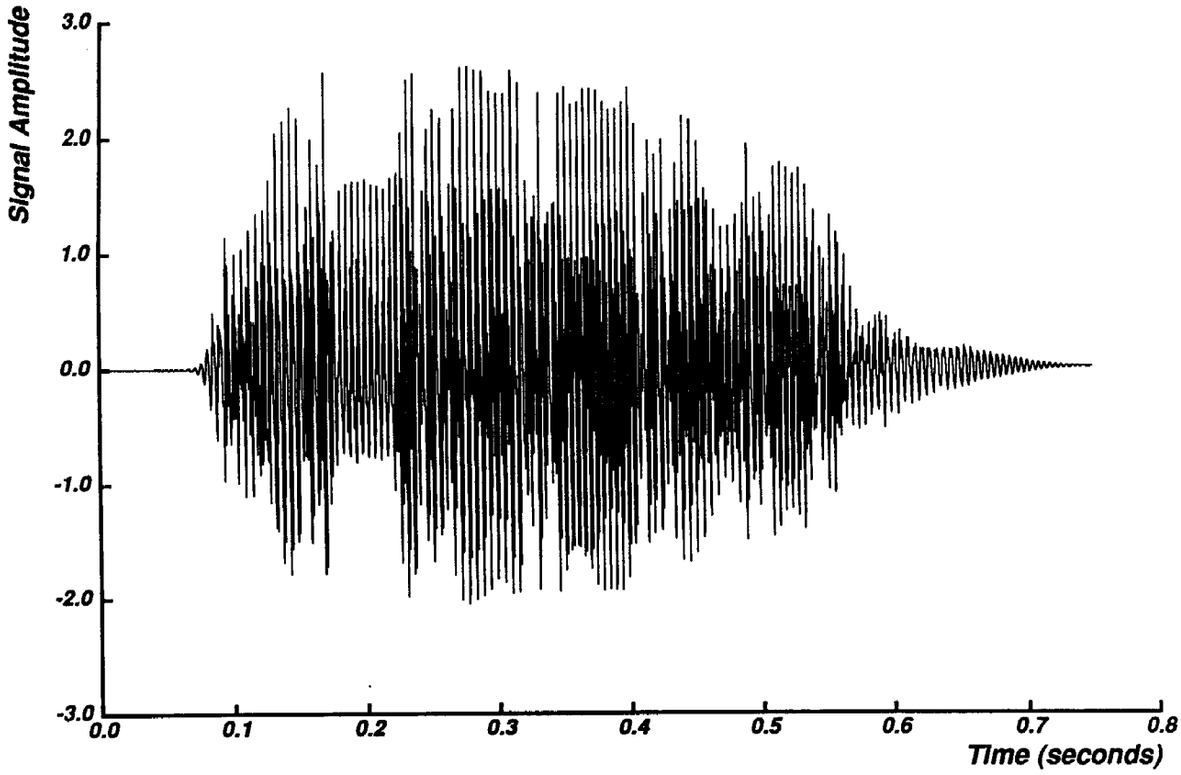

Figure 1-2: A short vocal sound with artificially added vibrato.

1.3. Integrated Systems

In the past, computer music systems alienated many composers and musicians because they required a great deal of technical expertise. With the advent of personal computers and rapidly decreasing hardware costs, it is now more important than ever to design systems that do not require a support staff and a team of scientists to maintain and operate. The primary issues are the human-machine interface, music representation, and real-time capabilities.

Figure 1-6 illustrates the main components we envisioned for the Musician's Workbench, a personal computer music system. The system is built around a computer with software for representing and displaying music. The computer system is augmented with the addition of a music keyboard for input and a computer-controlled music synthesizer for output. There is also the possibility of attaching a printer to the computer in order to print scores. Typical uses would include:

- entering a score to be performed on the synthesizer,
- live performance using the keyboard and synthesizer,

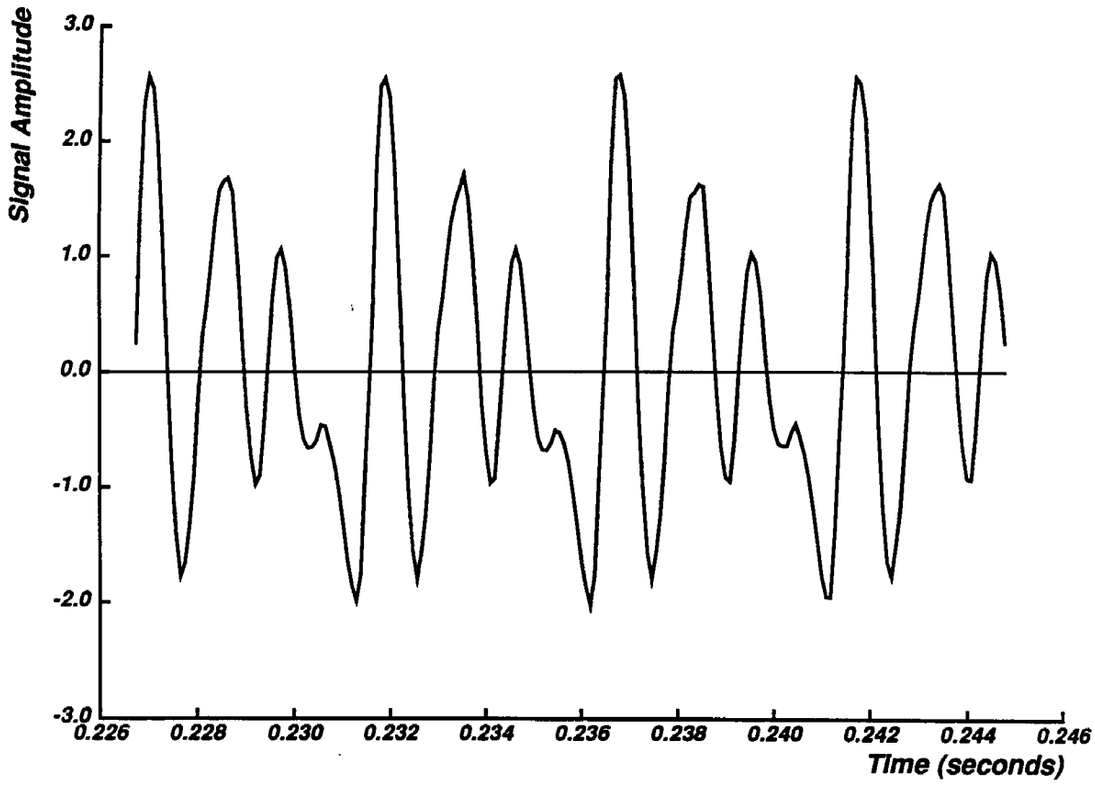

Figure 1-3: A magnified segment of the short vocal sound with artificially added vibrato.

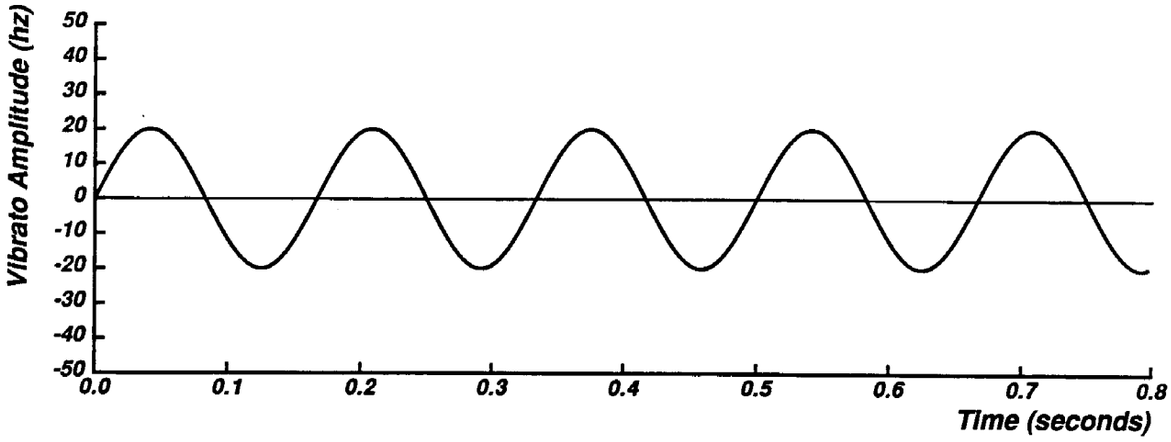

Figure 1-4: The vibrato function in isolation.

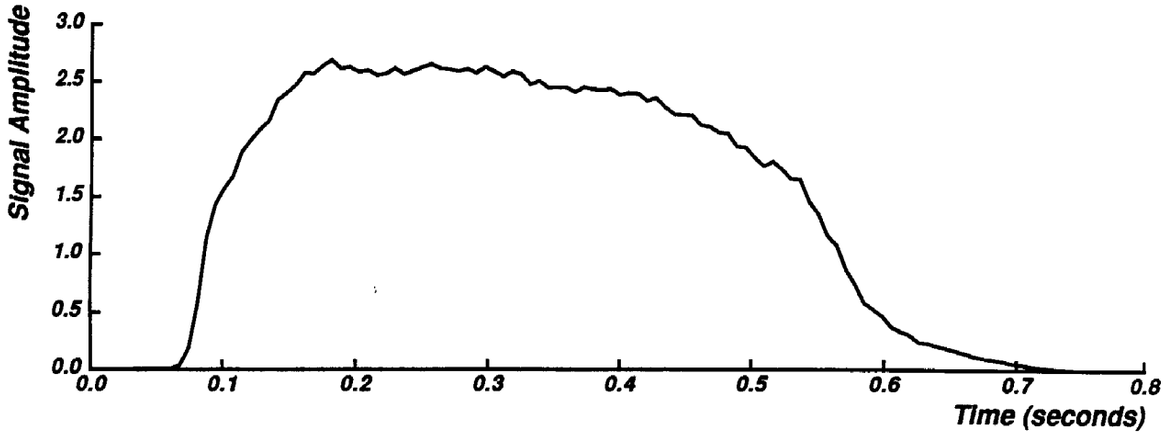

Figure 1-5: The amplitude envelope of the vocal sound.

- listening to computer performances of scores,
- printing music scores and individual parts for various instruments,
- designing new sounds to be played by the synthesizer,
- recording a keyboard performance for analysis, and
- using the computer as a teaching aid for music subjects such as harmony, counterpoint, and ear training.

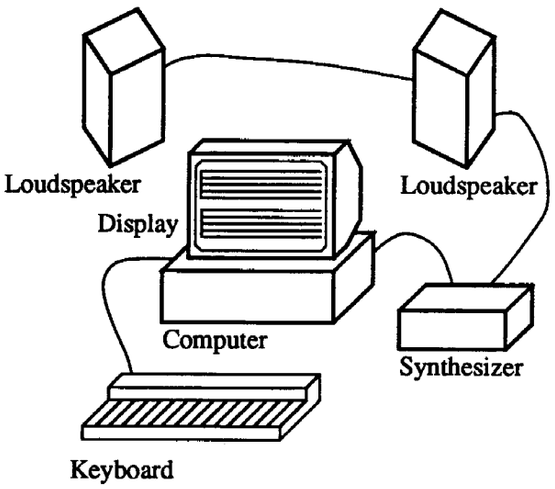

Figure 1-6: The Musician's Workbench.

We will look at a number of issues that influenced the design of the Musician's Workbench. We will generally describe the problems that arose and outline our approach to their solution. In many cases, we will also describe alternatives that have been tried in other computer music systems.

2. Music Synthesis Techniques

The production of musical sounds by computer is a challenge that has led to a surprisingly diverse set of techniques. In every case, the technique is a compromise between three somewhat incompatible requirements: (1) the resulting sound should be musically interesting (this is perhaps the most important and least precise requirement), (2) the sound should be inexpensive to compute, 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 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 [Chowning 73] 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, inharmonic signals can be generated as well. This can be useful for

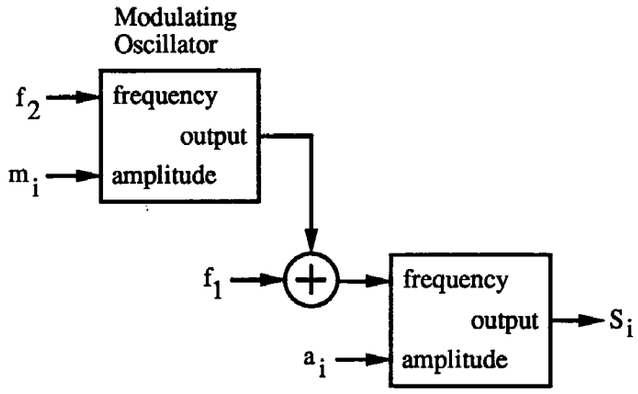

Figure 2-1: Use of two sine wave oscillators to implement frequency modulation synthesis.

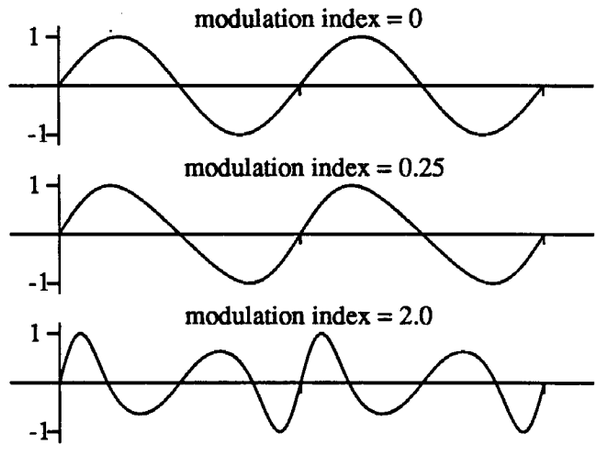

Figure 2-2: Waveforms produced using two different values of *m*, the modulation index.

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 given an arbitrary sound, it is difficult to find a set of control parameters that will approximately reproduce that sound [Payne 87]. Thus, one of the problems with FM synthesis is that producing a desired sound can require a large amount of trial and error. We might say that FM synthesis parameters form an inadequate representation system for many sounds of interest.

2.2. Sampling Synthesis

The technique known as *sampling* [Massie 85] 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 two or 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 called *looping*. Finally, the third part, called the *release* contains a recording of the end of the sound. In practice, the *release* is usually approximated by diminishing the amplitude while looping the sustain section.

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 be changed only 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 waveform as well as the overall magnitude of the samples. For example, when an

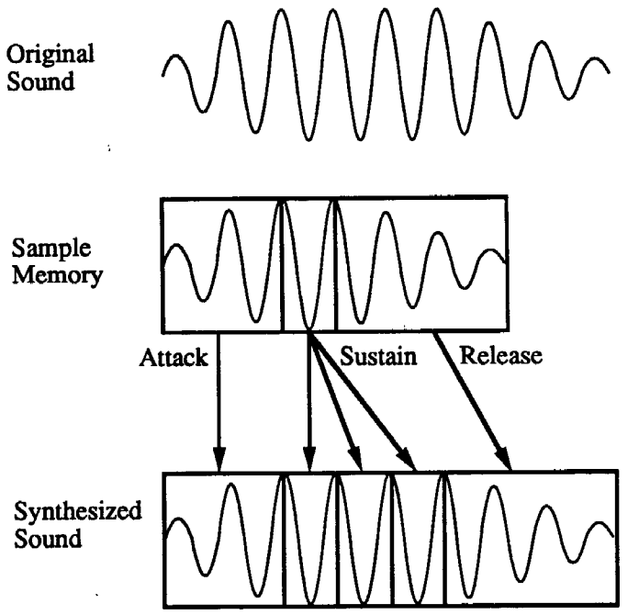

Figure 2-3: Sampling synthesis.

instrument plays louder, the high frequencies usually become disproportionately louder than the low ones. The instrument 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.

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 might require prerecorded samples of crescendi at different pitch and loudness levels. Viewed as a representation system, sampling synthesis is excellent for specific predetermined sounds, but it fails to capture the wide range of variability in musical instrument sounds.

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.

Figure 2-4 illustrates wavetable synthesis as a three-part computation. For each sample, the phase is computed to form a location in the table. The phase is incremented by an amount proportional to the desired frequency. A sample of the waveform is then accessed from the table, and the sample is multiplied by the desired amplitude. Wavetable synthesis differs from looping in sampling synthesis mainly in that a fixed-size table containing one period is typically used for wavetable synthesis, whereas the loop region is arbitrary with sampling synthesis.

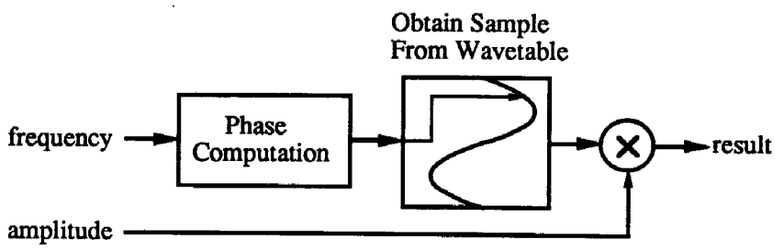

Figure 2-4: Wavetable synthesis.

Wavetable synthesis has several advantages. First, it is inexpensive to compute (typically less than FM and sampling).[2] 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. Second, 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. Third, tables can be stored in a compact form as a table of harmonic amplitudes.

---

[2]Sampling computation is trivial, but there is an added cost associated with transposing sampled audio to vary the pitch.

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 can stretch the control functions to produce a longer sound without lowering the pitch. One can also create hybrid sounds by combining control functions derived from different natural sounds.

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 [Lansky 81, Rodet 84a, Slawson 85].

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 [Adrien 85, Buxton 85].

2.5. Synthesis in the Musician's Workbench

One of the objectives of the Musician's Workbench was 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 seemed adequate. On the other hand, more general techniques like summation synthesis and physical modeling require more computation than we thought would be practical in the near future.

We worked with Dr. Peter Comerford at the University of Bradford, England, to interface his Bradford Musical Instrument Simulator [Comerford 81] 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 was in the early stages of optimization, and we believed that equivalent performance could be achieved with 20 to 30 integrated circuits using VLSI technology. (A recent version of the device uses even fewer parts.) 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 always reads and combines data from two wavetables. Thus, it is possible to interpolate between any two waveforms[3]. 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 developed 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. [Serra 90] Our preliminary results have been encouraging and indicate that the technique is sound. We have succeeded in reproducing many brass and woodwind tones 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.

Table 2-1 describes some classes of instruments and appropriate synthesis techniques using the BMIS. The wavetable and summation techniques were described in Section 2.3. In the BMIS, wavetable synthesis is often done with two interpolated wavetables where the interpolation is based on pitch. For example, the timbre of a set of organ pipes gradually changes from low pitch to high pitch. Summation

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

synthesis in the BMIS often uses non-sinusoidal waveforms. This is an intermediate step between placing all sines in one table (wavetable synthesis) and using a separate table-lookup operation for each sinusoid. Wavetable interpolation, where interpolation is changed over time, is described above, and sampling in the BMIS is accomplished by playing just one cycle of a wavetable. This is appropriate only for very short sounds or for the initial transient portion of a longer sound.

Class of Sound	**Example**	**Technique**
Organ Pipes	Gedackt	wavetable
Plucked & Hammered String	Pianoforte	summation
Winds	Clarinet	wavetable interpolation
Bowed Strings	Violin	wavetable interpolation
Percussion	Snare Drum	sampling

Figure 2-1: Classes of instruments and appropriate synthesis techniques as implemented on the Bradford Musical Instrument Simulator

2.6. Retrospection

We have discussed synthesis techniques in some detail and outlined our plans for a new approach based on waveform interpolation. The big issue for computer music is synthesis and control of musically interesting sounds, and the issue is still with us. Over the last several years, memory prices have dropped dramatically, and memory-intensive sampling synthesizers have become the best approach to synthesizing traditional instruments. This is a case where enough sound "images" can be stored to make reproduction straightforward. For non-traditional sounds, sampling is not favored because it lacks interesting ways to create and manipulate new timbres. Even with traditional orchestral sounds, the lack of expressive control, due the frozen nature of the captured "image" is a problem of the sampling approach, and researchers are therefore continuing to explore new techniques for synthesis.

Our main objective in this area was 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 has been 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*, 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.

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 [Dannenberg 84a, Dannenberg 84b, Dannenberg 86], 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. An important aspect of the problem is maintaining flexibility: computer music composers and performers want to design new instruments and new ways to control them. Thus, it is important to have a notation to describe control systems. Arctic is a very high-level notation for describing *behaviors* or real-time responses to input.

One feature of Arctic is that it represents behavior in a very abstract form that is suited to many levels of hierarchy in music compositions and performances. A behavior can represent a whole composition, a

movement, a theme, a phrase, a note, or even the vibrato within a note. This means that a single formalism and language can apply at many levels to yield a hierarchical description of the musical structure.

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.

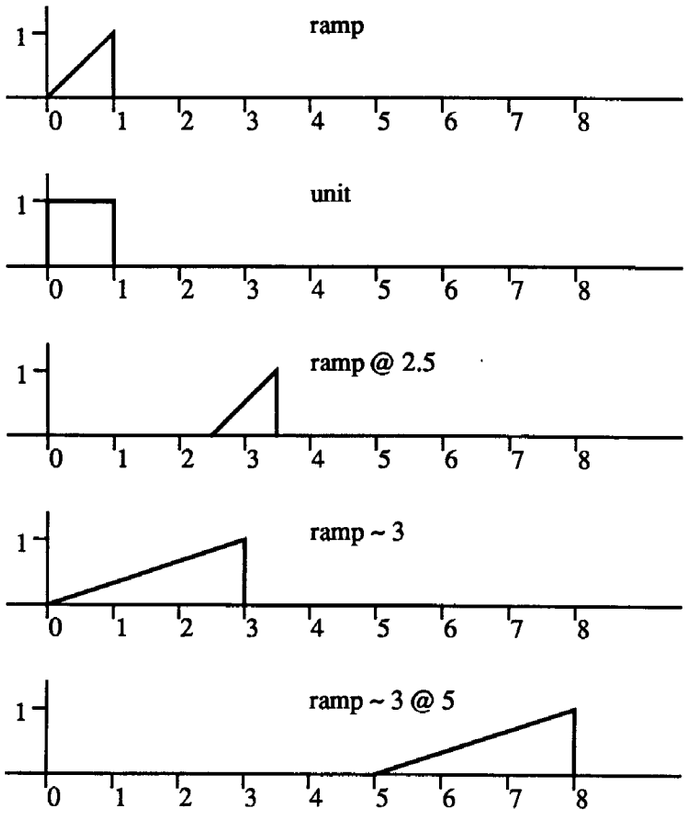

Figure 3-1: Arctic functions.

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. 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 simultaneously. 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.

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.

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 writing *Note(p)* where p specifies a pitch. We can write *random(bottom, top)* to compute a pitch between bottom and top, so [*Note*(*random*(*bottom*,*top*))] will

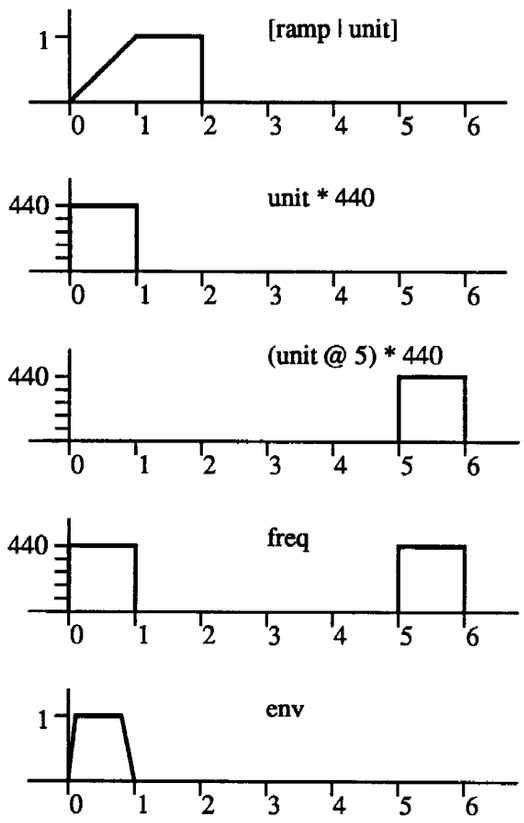

Figure 3-2: More Arctic functions.

play a note with the random pitch. We then embed this expression in a sequence construct.

The program is given below:

```
NoteSequence is [
    Note(random(bottom,top));
    NoteSequence) @ 1 ];
```

The sequence is started by invoking *NoteSequence*, which is defined to be a collection of two behaviors. The first creates a *note* as described above. Then, after a delay of one second, *NoteSequence* is invoked *recursively*; that is, *NoteSequence* causes itself. This generates an endless succession of notes.

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 *Note*, 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 can be much more complex, although based on the same set of principles.

The principal advantage of Arctic over more conventional programming 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. 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. This is illustrated by the behaviors in Figure 3-2.

Arctic is one of several new languages and systems oriented especially toward the problems of music synthesis and composition. Other notable languages are FORMES [Cointe 84, Rodet 84b] and the work by Anderson and Kuivila [Anderson 86, Anderson 90].

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. These results may be used to control the frequency, amplitude, and other sound parameters in a music synthesis program.

It was our plan to develop a real-time version of Arctic that would serve to link control devices such as keyboards and gesture sensors to a real-time synthesizer performing spectral interpolation synthesis. Arctic could also be used for non-interactive compositions. We developed a prototype of a real-time version of Arctic, but it was not integrated with the Musician's Workbench.

3.2. Retrospection

The work on Arctic and real-time control were directed at a central issue of computer music: the expression and control of complex musical behavior. Our work with Arctic has helped us to crystalize the concepts of behaviors, functions, and timing relationships, and this led to a series of interesting languages based on Arctic semantics.

One of the things we anticipated with the Musician's Workbench was a close coupling between

computer and synthesizer. We expected the computer to generate control functions for synthesis algorithms running on special high-speed hardware. This leads to a very flexible system because the method of control is completely programmed. This was a standard approach in research systems, and there are still systems of this type being built, but the dominant configuration of personal computer music systems today is a synthesizer with a dedicated control computer controlling special synthesis hardware. The control computer runs proprietary software and is programmable only to a limited extent. Furthermore, there is a limit to the speed at which information can be sent to the synthesizer control computer. (See Figure 3-3.)

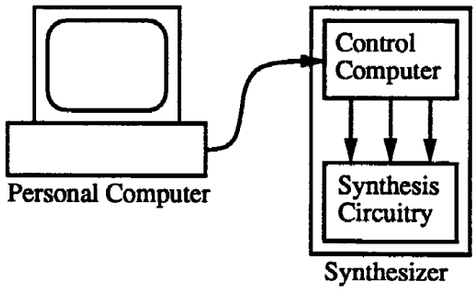

Figure 3-3: A personal computer controlling a music synthesizer with an embedded control processor.

This configuration is likely to continue for some time because the flexibility of a reconfigurable and highly programmable control system is not considered essential by most consumers. For researchers and serious composers, however, control remains a critical issue. It may not be long before computers are so fast that special purpose synthesis hardware is not necessary — one computer will compute control information and synthesize digital audio in one integrated process. Self-contained synthesizers will still exist for convenience, but researchers will turn to fast integrated computers to replace more limiting synthesizers. When that time is reached, it will be much easier to explore various approaches to control problems in music, and languages to express these approaches will become more important.

4. Integrated 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 were to 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 any system design. We anticipated that it would take several years to develop the Musician's Workbench, so we decided to use what was then a large and expensive computer for development. We reasoned that computers of that complexity would be relatively inexpensive by the time we finished our development. 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 was a Sun II workstation, which is comparable to a fairly small personal computer today.

The keyboard and mouse input devices of the computer were augmented with 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 [Ioy 85]. 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 was able to both send and receive MIDI data.

Visual output was provided by a large display that was intended to show music, text, and other graphics. For sound output, the computer was connected to a Bradford Musical Instrument Simulator. This allowed real-time sound synthesis, giving instant aural feedback to composers and students. The combination of high quality, low cost, and programmability of the BMIS was 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 were 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 researcher wants to do something new and unplanned for, he or she 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 contains 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 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 was to be used for music throughout the system. (The common representation exists in prototype form, but never reached the point of everyday use.) A common representation allows, say, a composition program to produce music which can then be printed by a music typography program. The music might 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 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. In fact, the schema was used to encode common practice music notation, and also for encoding performance information such as exact starting times, durations, and loudnesses derived from live performances. The intention was to handle the bulk of musical information using a fixed set of conventions, allowing information to be easily exchanged among many programs. However, there are always cases where conventions are too restrictive, in which case new attributes can be added. These attributes would not have any meaning to most programs, but the user could write his own programs to deal with the new attributes. In this way, the representation can be extended to meet new representation requirements.

To illustrate how this extensibility could be used, imagine composing music using conventional music notation to create a score. A score editor program would let the composer create scores in conventional notation on the computer display. Scores created with the score editor could then be performed by the computer using built-in software to control a synthesizer. Suppose, however, that the score should be performed using special 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.

This last example is perhaps the best representation of our vision of an integrated set of hardware and software tools for music. By taking care to make components flexible and extensible, the resulting system can find a wide range of users and applications. The term "system" is purposefully chosen to indicate that the Musician's Workbench was not a single program or a solution to a single problem. Rather, it was an attempt to build a complex yet integrated computer music environment. As with any large system, we expected new components to be added and old ones to evolve over time. We believe that advanced applications will demand the sort of foundations provided by systems like the Musician's Workbench. In section 5, we will look at a few examples of advanced applications of computers to music.

4.4. Retrospection

One of the most interesting and challenging aspects of the Musician's Workbench was the effort to define a rich music representation that could serve a number of applications, including music notation, performance capture, and synthesis. Prototypes of these applications were developed, but never reached the point where the concept could be properly tested. Nevertheless, there are indications that the complexity of music notation poses some real barriers to the idea of a "universal representation". In particular, work on music notation has convinced us that any representation for notation must contain a wealth of extra-musical information such as key signatures, ties, beams, and so on. Once this "extra baggage" is added, it is very difficult for other programs to deal with all the complex representation conventions. Of course, the extra information can be ignored, but if the score is created or modified without taking notational conventions into account, it might then become impossible for a notation editor to deal with the score.

In short, representations tend to be specialized. A good representation that is general enough for notation, performance, and theoretical work has not been found. The issue of music representation is a central topic in computer music research and will continue to be the focus of many investigations [DePoli 91].

Since the Musician's Workbench project began, there have been a number of music notation systems produced for personal computers. A surprising observation is that a music notation system takes 5 to 10 man-years of programming effort to reach the product stage. Even allowing for the difference between a research prototype and a commercial product, attempting to provide music notation on the Musician's Workbench was overly ambitious. In the area of representations, a standardization effort for music notation has arisen with many similarities to the extensible attribute/value pair approach of the Musician's Workbench [Sloan 89]. The standard is quite large, and it will be some time before we can expect to exchange musical data among diverse applications. However, a very simple representation based on MIDI data has become a very successful standard. It supports the encoding exchange of performance information such as duration and loudness but not music notation information such as page layout and typography.

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 perform 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. These examples are the result of work by the authors and are representative of a wide variety of work in the field of artificial intelligence and music.

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 [Thomas 85]. 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 [Thomas 89]. 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 [Dannenberg 84c, Vercoe 84, Bloch 85, Vercoe 85, Lifton 85]. 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.

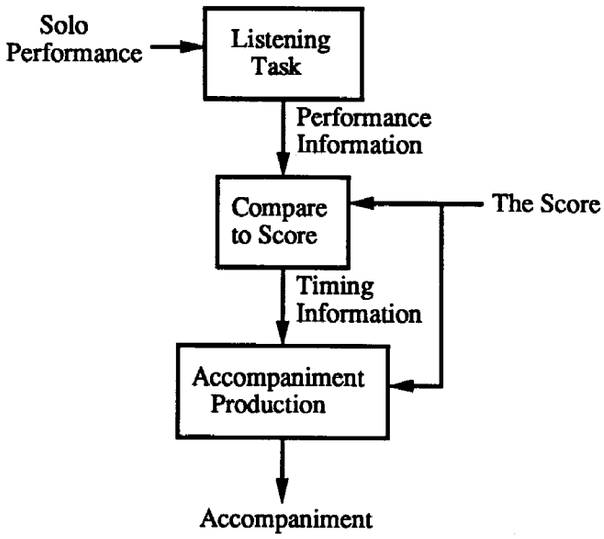

Figure 5-1: Structure of a computer accompaniment system

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 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 [Roads 80]. 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 [Dannenberg 87], and performing works by great composers with musical expression [Sundberg 83]. 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 teaching methods.

6. New Directions

The Musician's Workbench spawned a number of interesting research efforts, including work on synthesis, real-time control, and music representation. These efforts have met with some success and work is ongoing. The research has in fact outlasted the specifics of the Musician's Workbench. In retrospect, one of our mistakes was to tie our project to special-purpose hardware (the BMIS) when we had very limited resources. As a result, we had to deal with many hardware details, and porting the system to a new machine would require additional hardware construction. Furthermore, programming the special-purpose hardware was so difficult that we ultimately used slow software synthesis for all of our research on interpolation synthesis. Similar difficulties have been encountered by many groups, and only a few hardware synthesis systems (other than commercial products) have come into routine use.

We are developing a new workstation-based computer music system using a much faster computer than the Sun II we started with. As explained earlier, our belief is that we will eventually be able to perform real-time music synthesis using a general-purpose processor. Thus, our current effort is toward appropriate software structures to make this possible. In the meantime, we will compute digital audio slower than real-time and store computed samples on a disk memory for subsequent playback at full speed. Advances in magnetic disk storage and optical technology have made this approach much more practical that it was a few years ago. We also have a MIDI interface so we can take advantage of commercial real-time synthesis hardware.

Another mistake we made in the design of the Musician's Workbench was to assume that we could overcome the problems of running in real-time given an operating system with no real-time support. Earlier systems avoided using a general-purpose computer for music control because of the real-time problems when running many programs. Our logic was that software development would be easier and faster on a general purpose computer, and there were no inherent reasons we could not modify the system to support real-time programs. In fact, we expected other researchers might solve the real-time problem, saving us the effort.

As it turned out, the real-time problem still exists on powerful scientific workstations, where Unix is the de facto standard operating system. (There are a few proprietary and thus non-standard solutions available.) Consequently, building a real-time system has the same drawbacks as building special-purpose-hardware: the work is time-consuming, difficult to maintain, and the results are not very portable to new technology.

At present, the most effective way to build a real-time computer music system is to use a personal computer (as opposed to as scientific workstation). Personal computers such as the IBM PC, Apple Macintosh, Commodore Amiga, and Atari computers have simple operating systems that facilitate writing real-time systems. We have worked extensively with this class of computer as an alternative to the Musician's Workbench. This work, and that of many others has verified our original belief that the computer used for program development should run the real-time control software rather than using a special-purpose control computer.

We are still waiting for real-time support on scientific workstations. Rather than try to solve this problem, we have decided to continue working with personal computers for real-time systems, continue developing non-real-time software for signal processing on scientific workstations, and plan to integrate the software at some future date. It will be particularly exciting when real-time capabilities combined with faster computers allow real-time music synthesis in software.

7. 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. In any case, the discussion served to introduce an important issue in computer music: the problem of new techniques for creating interesting musical sounds. Some of these synthesis techniques have been incorporated into special hardware 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 assemble 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 an early system, the Musician's Workbench, and showed how appropriate hardware, augmented by a collection of software tools, could support musical activities. One of the critical prerequisites of an integrated system

is support for exchanging data among various software tools. We discussed the representation scheme for the Musician's Workbench and used it to illustrate a third big issue: the representation problem.

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. The development of intelligent music systems is an area of continuing importance in computer music.

Based on our experience with the Musician's Workbench, we have begun a new project with similar goals. Because of advances in memory and computation, we are emphasizing the use of software synthesis as opposed to synthesis with special hardware, and we are emphasizing capabilities not available on commerical systems. These include extensible environments for graphical music representation and sophisticated support for music synthesis.

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 (International Computer Music Association, Suite 330, 2040 Polk Street, San Francisco, California 94101, USA) are good sources of information on the state-of-the-art in computer music. Recent research is also treated by Mathews and Pierce [Mathews 89]. For those with a computer science background, an issue of ACM Computing Surveys [Abbot 85] 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 [Schrader 82]. 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 [Lansky 81].

Two collections of early work in computer music are *Music by Computers* [Foerster 69] and *The Computer and Music* [Lincoln 70]. 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 [Mathews 63]) wrote a classic monograph [Mathews 69] 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 [Moorer 77] covers a somewhat wider range of topics but in less detail. A more detailed discussion of practical aspects of digital audio is contained in an article by Blesser [Blesser 78], and a detailed text on synthesis by

computer was written by Moore [Moore 90].

8. 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 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. There will be tremendous economic and pragmatic incentives for many composers to adopt the computer as their primary medium. (Many 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 or supplant the aural, visual, and social qualities of existing media.

One might also wonder if economic pressures will bring an end to live performances of music. A look at the present situation, however, reveals that live performance is still important even after the advent of radios, phonographs, film and television. Many composers of computer music are keenly interested in live performance, often in combination with traditional instruments. Computer technology seems to be neutral with respect to live human interaction: performances can be completely spontaneous and improvised or completely predetermined. If anything, the computer has expanded our potential for live performance and human involvement in music making.

The computer will continue to make many contributions to music. Because new low-cost computers will be as common as television sets, 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.

9. Acknowledgments

The authors wish to thank Georges Bloch, Xavier Chabot, Gareth Loy, F. R. Moore, Allen Newell, Dana Scott, and Herbert 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.

The Authors

Roger B. Dannenberg

is a senior research computer scientist at Carnegie Mellon University. His research interests include programming-language design and implementation, and the application of computer science techniques to the generation, control, and composition of computer music. Dr. Dannenberg is currently studying software and system support for digital multimedia. He frequently performs jazz and experimental music on trumpet or electronically.

Dannenberg received a BSEE from Rice University in 1977, and MS in computer engineering from Case Western Reserve University in 1979, and a PhD in computer science from Carnegie Mellon in 1982. He is a board member of the Pittsburgh New Music Ensemble and the International Computer Music Association.

Paul McAvinney

received a B.A. in English Literature, with minors in philosophy and physics, from Fordham University in 1964. He has been an independent consultant to several Fortune-100 firms in data communications software, hardware, and network simulation. He held technical, marketing, and management positions at Sperry-Rand Corporation (now Unisys), Honeywell Information Systems, and General Electric Company.

McAvinney is the founder and chairman of Sensor Frame Corporation, and inventor of the Sensor Frame, Sensor Cube, Stage Frame, and VideoHarp technologies. He was a Senior Research Programmer at Carnegie Mellon, where in 1983, together with Dr. Roger Dannenberg, he co-founded the Computer Music Laboratory within the Computer Science Department.

Marilyn Taft Thomas

is head of the Department of Music and associate professor of theory and composition. Dr. Thomas is an active composer with works for orchestra, choir, piano, voice and chamber ensembles, and she has received six awards in composition from ASCAP.

Dr. Thomas received a Ph.D. in composition from the University of Pittsburgh. Her computer music research and development of music theory software is internationally known. "MacVoice," an interactive tool for writing four-part harmony, co-authored with Peter Monta, is now in use by many music departments.

Joshua J. Bloch

is a senior systems designer at the Transarc Corporation. He received a B.S. in Computer Science from Columbia University in 1982, and a Ph.D. in Computer Science from Carnegie Mellon in 1990. He held summer positions at the usual assortment of corporate research labs.

Bloch has worked on data replication for high availability, general purpose programming languages for transaction systems, and polyphonic music accompaniment algorithms. He is currently a senior architect and performance analyst of an open, distributed transaction system.

Dean Rubine

is a research computer scientist at Carnegie Mellon University. He received his BS and MS degrees in electrical engineering and computer science from MIT in 1982, and a PhD in computer science from Carnegie Mellon in 1992. Previously, he spent several years at Bell Laboratories, where he worked mainly on the implementation of programming languages.

Rubine has worked on programming languages for real-time control, the analysis and synthesis of musical tones, and the creation of a new musical instrument, the VideoHarp. He now investigates novel forms of human-computer interaction, including multimedia, and has built computer systems able to interpret human gestures.

Marie-Helene Serra

received a diploma of engineering in physics in 1981 from INSA (National Institute for Applied Sciences) in Toulouse, France, and a DEA (Diplome d'Etudes Approfondies) in acoustics from Bordeaux University in 1982. She received her doctoral degree on speech synthesis in 1985 and a diploma of docteur ingenieur from the computer science department at Toulouse University.

Dr. Serra was a visiting scientist at Carnegie Mellon from 1985 to 1987 where she worked on sound synthesis via waveform interpolation. She is currently working as a software engineer at the CEMAMu center in Paris, which is directed by Iannis Xenakis.

References

[Abbot 85] Abbot. C. (editor). *Computing Surveys*. ACM, 1985. No. 2 (June).

[Adrien 85] Adrien, J-M, and X. 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.

[Anderson 86] Anderson, D. P. and R. Kuivila. Accurately Timed Generation of Discrete Musical Events. *Computer Music Journal* 10(3):48-56, Fall, 1986.

[Anderson 90] Anderson, D. P. and R. Kuivila. A System for Computer Music Performance. *ACM Transactions on Computer Systems* 8(1):56-82, February, 1990.

[Blesser 78] Blesser, B. A. Digitization of Audio: A Comprehensive Examination of Theory, Implementation, and Current Practice. *Journal of the Audio Engineering Society* 26(10):739-771, October, 1978.

[Bloch 85] Bloch, J. J. and R. B. Dannenberg. Real-Time Computer Accompaniment of Keyboard Performances. In B. Truax (editor), *Proceedings of the International Computer Music Conference 1985*, pages 279-290. International Computer Music Association, 1985.

[Buxton 85] Kitamura, J., W. Buxton, M. Snelgrove and K. 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.

[Chowning 73] Chowning, J. M. The Synthesis of Complex Audio Spectra by Means of Frequency Modulation. *Journal of the Audio Engineering Society* 21(7), September, 1973.

[Cointe 84] Cointe, P. and X. Rodet. Formes: an Object & Time Oriented System for Music Composition and Synthesis. In *1984 ACM Symposium on LISP and Functional Programming*, pages 85-95. ACM, New York, 1984.

[Comerford 81] Comerford, P. J. Bradford musical instrument simulator. *IEE Proc.* 128, Pt. A(5):364-372, July, 1981.

[Dannenberg 84a] Dannenberg, R. B. Arctic: A Functional Language for Real-Time Control. In *1984 ACM Symposium on LISP and Functional Programming*, pages 96-103. ACM, New York, August, 1984.

[Dannenberg 84b] Dannenberg, R. B. and P. McAvinney. A Functional Approach to Real-Time Control. In W. Buxton (editor), *Proceedings of the International Computer Music Conference 1984*, pages 5-15. International Computer Music Association, 1984.

[Dannenberg 84c] Dannenberg, R. B. An On-Line Algorithm for Real-Time Accompaniment. In W. Buxton (editor), *Proceedings of the International Computer Music Conference 1984*, pages 193-198. International Computer Music Association, 1984.

[Dannenberg 86] Dannenberg, R. B., P. McAvinney, and D. Rubine. Arctic: A Functional Language for Real-Time Systems. *Computer Music Journal* 10(4):67-78, Winter, 1986.

[Dannenberg 87] Dannenberg, R. B. and B. Mont-Reynaud. Following an Improvisation in Real Time. In J. Beauchamp (editor), *Proceedings of the 1987 International Computer Music Conference*, pages 241-248. International Computer Music Association, San Francisco, 1987.

[DePoli 91] G. De Poli, A. Piccialli, and C. Roads (editor). *Representations of Musical Signals*. MIT Press, Cambridge, Mass., 1991.

[Foerster 69] Foerster, H. V., and J. W. Beauchamp (editors). *Music By Computers*. John Wiley & Sons, Inc., 1969.

[Lansky 81] Lansky, P. and K. Steiglitz. The synthesis of timbral families by warped linear prediction. *Computer Music Journal* 5(3):45-49, Fall, 1981.

[Lifton 85] Lifton, J. Some Technical and Aesthetic Considerations in Software for Live Interactive Performance. In B. Truax (editor), *Proceedings of the International Computer Music Conference 1985*, pages 303-306. International Computer Music Association, 1985.

[Lincoln 70] Lincoln, H. B. (editor). *The Computer and Music*. Cornell University Press, Ithaca, N.Y., 1970.

[loy 85] Loy, G. Musicians Make a Standard: The MIDI Phenomenon. *Computer Music Journal* 9(4):8-26, Winter, 1985.

[Massie 85] Massie, D. C. The Emulator II Computer Music Environment. In B. Truax (editor), *Proceedings of the 1985 International Computer Music Conference*, pages 111-118. International Computer Music Association, 1985.

[Mathews 63] Mathews, M. V. The Digital Computer as a Musical Instrument. *Science* 142:553-557, 1963.

[Mathews 69] Mathews, M. V. *The Technology of Computer Music*. MIT Press, Boston, 1969.

[Mathews 89] Mathews, M. V. and Pierce, J. R. (editor). *System Development Foundation Benchmark Series: Current Directions in Computer Music Research*. MIT Press, 1989.

[Moore 90] Moore, F. R. *Elements of Computer Music*. Prentice Hall, New Jersey, 1990.

[Moorer 77] Moorer, J. A. Signal Processing Aspects of Computer Music - A Survey. *Proceedings of the IEEE* 65(8):1108-1137, July, 1977. A revised and updated version is included in *Digital Audio Signal Processing: An Anthology*, ed. by John Strawn, William Kaufmann, Inc., 1985.

[Payne 87] Payne, R. G. A Microcomputer Based Analysis/Resynthesis Scheme for Processing Sampled Sounds Using FM. In *Proceedings of the 1987 International Computer Music Conference*, pages 282-289. Computer Music Association, 1987.

[Pohlmann 92] Pohlmann, K. C. *The Compact Disc: a handbook of theory and use*. A-R Editions, Madison, WI, 1992.

[Roads 80] Roads, C. Artificial Intelligence and Music. *Computer Music Journal* 4(2):13-25, Summer, 1980.

[Rodet 84a] Rodet, X., Y. Potard and Jean-Baptist Barriere. The CHANT Project: From Synthesis of the Singing Voice to Synthesis in General. *Computer Music Journal* 8(3):15-31, Fall, 1984.

[Rodet 84b] Rodet, X. and P. Cointe. FORMES: Composition and Scheduling of Processes. *Computer Music Journal* 8(3):32-50, Fall, 1984.

[Schrader 82] Schrader, B. *Introduction to Electro-Acoustic Music*. Prentice Hall, New Jersey, 1982.

[Serra 90] Serra, M., D. Rubine, R. B. Dannenberg. Analysis and Synthesis of Tones by Spectral Interpolation. *Journal of the Audio Engineering Society* , 1990. to appear.

[Slawson 85] Slawson, A. W. *Sound Color*. University of California Press, Berkeley, 1985.

[Sloan 89] D. Sloan. Precis of the Standard Music Description Language. In *Proceedings of the 1989 International Computer Music Conference*, pages 296-302. Computer Music Association, 1989.

[Sundberg 83] Sundberg, J., A. Askenfelt, L. Fryden. Musical Performance: A Synthesis-by-Rule Approach. *Computer Music Journal* 7(1):37-43, Spring, 1983.

[Thomas 85] Thomas, M. T. Vivace: a rule based AI system for composition. In B. Truax (editor), *Proceedings of the International Computer Music Conference 1985*, pages 267-274. International Computer Music Association, 1985.

[Thomas 89] Thomas, M. T., S. Chatterjee, Mark W. M. Cantabile: A Rule-Based System For Composing Melody. In T. Wells and D. Butler (editor), *Proceedings of the 1989 International Computer Music Conference*, pages 320-323. International Computer Music Association, 1989.

[Vercoe 84] Vercoe, B. 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.

[Vercoe 85] Vercoe, B. and M. Puckette. Synthetic Rehearsal: Training the Synthetic Performer. In B. Truax (editor), *Proceedings of the International Computer Music Conference 1985*, pages 275-278. International Computer Music Association, 1985.

Table of Contents

List of Figures

List of Tables