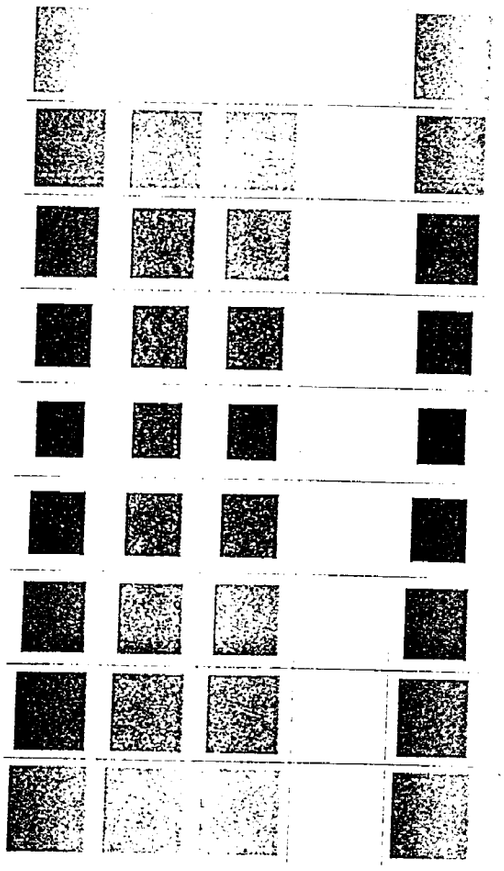
VOLUME NINE
THE COMPUTER MUSIC AND DIGITAL AUDIO SERIES

MUSICPROCESSING



GOFFREDO HAUS EDITOR

ROGER B. DANNENBERG

COMPUTER MUSIC AT CARNEGIE MELLON UNIVERSITY

The Computer Music Project at Carnegie Mellon University has been active since 1983, when I joined the faculty of what was then the Department of Computer Science. A number of research topics have been addressed, and this chapter will survey what has been accomplished. I will also mention directions for future computer music research at Carnegie Mellon. This work would not have been possible without the help of coworkers and students, and I will describe their contributions as the story unfolds. Also, a number of publications have resulted from our research. For the sake of completeness, citations to nearly all papers from the Computer Music Project are included in this chapter.

The title "Computer Music Project" may elicit images of a large organization with many composers and researchers. In reality, projects in the School of Computer Science are often run by one or a small number of faculty, and the Computer Music Project is essentially the efforts of one faculty member and a few very capable coworkers and students.

This is not meant to imply that this is the only computer music work done at Carnegie Mellon. Herb Simon is known for his early studies in

Music and Artificial Intelligence, resulting in the Listener program (Simon 1968), and Paul Dvorak held a joint appointment in electrical engineering and music in the 1970s and constructed a digital music synthesizer. Marilyn Thomas joined the music department with the mandate of starting a computer and electronic music program for undergraduate music majors. This has developed into a set of courses (Dannenberg, McAvinney, and Thomas 1985), including a two-semester sequence in computer and electronic music, currently taught by Reza Vali, and a similar sequence entitled Sound and Recording, taught by Ricardo Schulz. Dr. Thomas has also developed interesting courseware for harmony instruction (Thomas and Monta 1986) and expert systems for harmonizing chorales (Thomas 1985) and composing melody (Thomas, Chatterjee, and Maimone 1989).

REAL-TIME CONTROL

When I started working in computer music, there was neither a big budget nor a well-equipped studio. Paul McAvinney, who had been working on a digital music synthesizer in Pittsburgh, joined me at Carnegie Mellon in 1983. Together, we were certain that real-time synthesis would become the most popular way to synthesize music. Believing that controlling signal processors would soon become more difficult than building or buying them, we decided to focus our limited resources on the problems of expressive real-time control.

In addition to our experience with real-time systems and digital synthesizers, we were very strongly influenced by the concept of scientific workstations. The Computer Science Department was winding up a large project in which we implemented an operating system for a network of personal computers (Dannenberg 1982) with virtual memory, local disk storage, and large bit-mapped displays. Hoping to do for music what had already been done for engineering, we began looking at ways to use workstations for computer music production (Dannenberg 1986e). Note that "workstation" in the computer music world has since come to mean an integrated music production system, but current commercial "music workstations" lack the powerful general-purpose computer that we envisioned.

After a couple of false starts, we proposed to build the "Musician's Workbench," which would consist of special-purpose synthesis hardware, a powerful personal computer, and a suite of software for music representation, timbre design, and real-time performance (Dannenberg 1987). The "Musician's Workbench" was never fully realized, but it served us well by providing a model of how computer music systems could be organized.

Comparisons between what was then the current technology and the system we envisioned motivated a large number of smaller research projects.

One of our earliest ambitions was to realize an intuitive connection between gestures, or physical motion, and sound. This immediately led to three areas of research: the sensing of gestures, the manipulation of gestural control information, and synthesis techniques that lend themselves to timbral control. In the following sections, I will describe these research paths independently, starting at the end of the process: music synthesis.

Spectral Interpolation Synthesis

My investigation of synthesis based on time-domain interpolation began with an unpublished proposal and description written in 1979. The basic principle is quite simple: By interpolating slowly and smoothly between two periodic waveforms, a continuously varying spectrum can be obtained. If the phases of the corresponding harmonics are all matched, then a linear interpolation of waveforms in the time domain produces a linear interpolation of spectra in the frequency domain.

Figure 1 illustrates the analysis/synthesis process using spectral interpolation. Starting with a musical tone, the first step is to perform short term DFTs to obtain an approximation of the "instantaneous" harmonic content of the signal. The DFTs are taken at every period after resampling the signal so that each period of the signal lasts an integral number of samples. This procedure minimizes artifacts due to performing the DFT on a very short section (one period) of the signal.

In the next step, we eliminate many of the spectra from the signal representation. Of course, this causes some loss of information, but the goal is to eliminate only spectra such that, when they are approximated by interpolations between the remaining spectra, the resulting sound is perceptually unchanged.

The last signal in figure 1 shows the result of synthesizing a tone from the remaining spectra. Rather than interpolating to compute a spectrum for each period and then performing an inverse DFT to obtain the corresponding signal, we need only to generate a waveform for each spectrum in the representation. Intermediate waveforms are produced by interpolating between these waveforms. As mentioned above, the phase of each harmonic must remain the same from one waveform to the next in order for waveform interpolation to be equivalent to spectral interpolation. Because of changes in phase, the resulting signal does not always look much like the original, but the phase changes are not perceptible.

The process illustrated in figure 1 is an interesting technique for data reduction of signals, but it does not provide for any control over the evolution of the signal. The solution is to compute or derive spectra in real

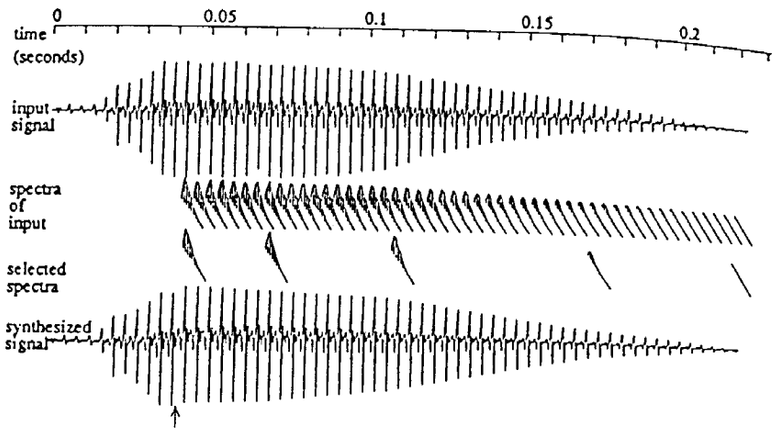


FIGURE 1. Analysis and synthesis by spectral interpolation.

time rather than fetch them from a fixed sequence. A simple way to derive spectra is by table lookup. Imagine a table containing an array of spectra. Each row of the table corresponds to a different pitch, and each column corresponds to a different amplitude level. If we assume that the spectrum an instrument produces is a function of pitch and amplitude, then we can analyze the spectrum of an acoustic instrument at each combination of pitch and amplitude in the table.

A musical tone describes a trajectory through a space defined by the dimensions of pitch and amplitude. To obtain an appropriate spectrum for each point along the trajectory, it is a simple matter to interpolate between the nearest spectra stored in the table. In summary, control information (pitch and amplitude) gives rise to a trajectory, and table-lookup methods can be used to derive a sequence of spectra corresponding to points along the trajectory. Spectral interpolation can then efficiently convert these spectral points into a musical tone with a continuously varying spectrum appropriate for the original control information. This technique can be generalized to more control dimensions if desired.

One of the features of this approach is that the selection of spectra that are used to construct waveforms can be controlled continuously by real-time gestures. This is in contrast to simpler table-lookup techniques. Another feature is that waveforms and interpolation functions can be based on the analysis of actual sounds. Extensive research has been devoted to developing practical analysis techniques for spectral interpolation synthesis.

Dr. Marie-Hélène Serra joined the project in 1985 and began working on analysis software for spectral interpolation synthesis (Dannenberg, Serra, and Rubine 1987; Serra, Rubine, and Dannenberg 1988a; 1988b; 1990). She used a DEC Vax computer for both analysis and synthesis to be sure we would not be subject to any limitations imposed by real-time software and hardware. The general research plan has two phases. In the first phase, we wanted to show that we could represent interesting musical tones using spectral interpolation with minimal perceptual changes to the signal. In the second phase, we want to show that we can synthesize tones from a description of their properties, such as the amplitude envelope and pitch contour.

The first phase is now complete. In addition, we examined several alternative synthesis strategies in the course of the research. One of the design choices was how to measure error between the original signal and the synthesized one. We want an error measure that corresponds closely to perceived error so that we can keep the error just under the minimum that is perceptible. We tried several error measures such as the sum of squares of the differences in each harmonic and the maximum difference, on both logarithmic and linear scales. The results here were inconclusive: It seems that any reasonable choice works very well.

Another design choice was whether to use linear interpolation over time between waveforms or to use, at each period, some arbitrary linear combination of the two waveforms. Linear interpolation over time is simple and compact, but we felt that arbitrary interpolation per period might allow fewer spectra to be needed, more than making up for the extra data required to specify per-period interpolation factors.

Dr. Serra developed analysis procedures for both types of interpolation, and both work well. In some cases, there is an advantage to the arbitrary interpolation approach, but this was at best a small advantage. For a real-time system in which new spectra will be computed at a constant rate, the simpler linear interpolation approach seems to work fine.

The study of different types of interpolation was driven by our dissatisfaction with the attack transients of some of our synthesized tones. We thought that per-period interpolation factors might help out, especially during attacks where the spectrum changes rapidly.

When this did not satisfy us, we compared our results to a more traditional summation synthesis approach. We found it somewhat surprising that summation synthesis does no better with attacks than does spectral interpolation. It should be noted that if rapidly varying phase or frequency of partials is permitted, then it is possible to represent arbitrary signals exactly. On the other hand, it is useful for data reduction and for synthesis purposes to use smooth control functions that change slowly. It is this restriction that causes problems (Smith and Serra 1987).

Ultimately, we decided to try to splice sampled attack sounds onto synthesized "steady-state" sounds, so that attacks could include noise and other inharmonic energy. Cross-fades, where the attack is smoothly faded to zero while the steady-state signal is smoothly faded from zero to full amplitude, generally produce some audible effects due to an improper matching of the phase of each partial.

We decided instead to try to synthesize a transition tone that would match the phases in the sampled attack at the beginning and match the phases of the steady-state signal at the end. In between, the phase of each partial would be advanced or delayed while holding the amplitude constant. Furthermore, the length of the transition was optimized to minimize the total amount of phase shift required. Software was developed to compute all of this automatically and construct a synthetic tone, but the results were disappointing. To our great surprise, the very slight phase shifts were noticeable as a peculiar and unnatural quality to the sound.

Our final solution for making transitions from sampled attacks to synthesized steady-state signals was to match the phase and amplitude of each partial at the transition and to make the transition abruptly. No smoothing is necessary if all of the partials are perfectly matched at the boundary. With spectral interpolation synthesis, the phases of harmonics at the end of the sampled attack determine the phases of not only the first but of every waveform used to synthesize the tone. Another requirement for a smooth transition is that the end of the attack must have a nearly pure harmonic spectrum like that of the "steady-state" spectral interpolation signal. Figure 1 illustrates a sampled attack, and an arrow marks the splice point.

Another interesting finding in the study was how much learning is involved in listening. When we first started working with spectral interpolation synthesis, we thought most of our results sounded very good, even in direct comparison to the original sound recordings. As our work progressed, however, our listening skills improved so that even though our results improved, we could hear more imperfections. For some time, I had difficulty understanding Dr. Serra's reservations about certain sounds. It turned out that I simply had not listened enough. This learning effect makes it very difficult and time-consuming to compare and evaluate different techniques.

Work has just begun on the second phase of our research. Software has been written by Chris Fraley to resample signals with vibrato to obtain an integral number of samples per period, using the resampling technique of Smith and Gossett (1984). The software then sorts the periods by amplitude and computes spectra. The output is a table of spectra for various amplitude levels. By processing crescendos played on different pitches, a table mapping pitch and amplitude values to spectra can be obtained. We

have yet to try synthesizing a variety of timbres and subjecting them to critical listening.

All of our work has used non-real-time software synthesis. For real-time synthesis, new digital signal processing chips are fast enough to compute at least several voices using spectral interpolation, but we have yet to do this. One of the difficulties is that waveform tables must be computed from spectral information during the synthesis process. That means the signal processor must be shared between a table-lookup process for producing the tone and a table-generation process that provides the next table.

Real-Time Programming

Spectral interpolation synthesis promises to give us a powerful method for generating tones with time-varying spectra. This immediately raises the question of where to get the control information. In many cases, control information will come from a live performer, but a significant amount of computation is required to transform a performer's gestures into the appropriate control parameters for a synthesis technique like spectral interpolation.

Two systems have been developed in the Computer Music Project for real-time programming. The first is Arctic, a very powerful and very high-level experimental language. The other system is the CMU MIDI Toolkit, a very practical and portable system that has served in a variety of applications.

Arctic

The language Arctic (Dannenberg 1984; 1986a; 1986b; Dannenberg and McAvinney 1985; Dannenberg, McAvinney, and Rubine 1986; Rubine and Dannenberg 1987) was developed as a compact notation for expressing complex linkages between human gestures and the control parameters of signal processors. As we shall see later, the Arctic paradigm is quite useful in other domains as well. Most of Arctic was designed by myself and Paul McAvinney in 1983. Dean Rubine wrote an interpreter for Arctic in 1984 and 1985. I have recently worked on a real-time implementation of Arctic with Hal Mukaino (Dannenberg 1990b; 1991a).

Arctic is in many ways a refinement of Music V (Mathews 1969) and 4CED (Abbott 1981). Recall that Music V has a score language for specifying when and how long notes should occur and an orchestra language that specifies how instruments are composed from networks of "unit generators," or built-in software signal processing modules. The 4CED language allows sequences to be triggered by other sequences, effectively providing a hierarchical score language.

Taking a more abstract view, the score language of Music V is a language in which the programmer specifies the timing of discrete events, namely instantiations of instruments at specified times. In contrast, the orchestra language is one in which "unit generators" are combined to operate on signals, which are nothing more than streams of numbers. Arctic provides for both instantiating computations at specific times *and* defining computations on signals. Thus, Arctic unifies the score and orchestra concepts of Music V, resulting in an elegant and powerful language for real-time control and composition.

To give some idea of how Arctic programs work, two short examples will be presented. The first example illustrates how Arctic might be used to define a score:

```
myscore is [
    trumpet(F4,mp) ~ 1 @ 10;
    trumpet(Bf4,mf) ~ 1.5 @ 11;
    violin(F5,mp) ~ 2.5 @ 10 ];
```

In this example, three notes are described. The "~" notation means to stretch the duration by some factor, and the "@" notation means to shift in time, so *X*~2@10 means "stretch *X* by 2 at time 10." The terms *trumpet* and *violin* are not part of the language per se but are presumably defined elsewhere in the Arctic program to produce notes of the appropriate timbres, starting times, durations, pitches, and loudnesses.

Arctic programs can serve as scores, and it is perfectly possible to write complex procedures for generating notes as well as to write each note out explicitly. For example, one can write a procedure to make trills or to generate a random melody. Furthermore, Arctic is hierarchical so that large scores can be constructed from simpler parts. If `myscore` is defined as shown above, one can write "`myscore` @ 4" to play `myscore` shifted by 4 seconds, or "`myscore` ~ 2" to stretch `myscore` by a factor of two.

Arctic programs can also be used to generate functions of time, which can be used as control functions for a synthesizer. For example, the following Arctic program generates an amplitude envelope and a pitch control for a note when a key is pressed:

```
out ampl, pitch;
env is [ ramp | unit | 1 - ramp ];
key(in p) is [
ampl += env ~ 1/3;
pitch += unit × p ];
```

In this program, `env` is defined to be a function that ramps from 0 to 1 in the first second, then stays at 1 for a second, and finally ramps back down

to 0. When a key is pressed, the envelope is compressed to one-third of its normal duration and added to the output function `ampl`. Simultaneously, the pitch control is set to `p` for a duration of one second.

To show how continuous input might be handled, imagine a knob used to control the amplitude of a 6 Hz vibrato. We could add the following lines to the program listed above:

```
in knob;
go is [pitch += (sin(6) ~ inf) × knob ];
```

The symbol `go` stands for things that should occur at the beginning of the program. In this case, `sin(6)`, a 6 Hz sinusoid, is multiplied by the time-varying value of `knob`, which represents the input from a physical control, and the result is added to the `pitch` control. The sinusoid is normally of finite duration, so it is stretched by infinity to make sure it does not terminate. The `sin` in Arctic is defined so that stretching does not change the frequency.

There is still much work to be done on Arctic. Techniques for running Arctic in real time have just recently been developed, and there is no compiler for real-time Arctic. More experience is needed to determine what extensions to the language are necessary to make real-time programming convenient and practical in Arctic.

The CMU MIDI Toolkit

Even before Arctic was developed, there was a need at Carnegie Mellon for some real-time music software. The CMU MIDI Toolkit was started not as a research project but as a short-term solution to our immediate needs. The CMU MIDI Toolkit (Dannenberg 1986c), or CMT, contains the most basic of music software components:

Adagio is a score language and an associated compiler. In Adagio, a note is represented by a line of text that specifies attributes such as pitch, duration, and loudness. Adagio is quite flexible and is compatible with several different ways of thinking about scores. For example, "Q" stands for a quarter note, but duration can also be indicated by "U87," which means 0.87 seconds. Adagio also supports arbitrary tuning systems. Adagio can record MIDI performances as well as play scores. The result of recording is a text file that can be edited by hand or used as input to another program.

ExGet and ExPut are programs for recording and replaying MIDI system exclusive messages. These programs are typically used to save and restore synthesis parameters for a digital synthesizer.

Moxc is a real-time programming environment that is ideal for writing interactive music programs. Moxc is an extension of the C

programming language and is based on Douglas Collinge's Moxie language.

Also provided are routines (in C) that allow direct production of MIDI output. Other routines are available to read MIDI data from a circular input buffer and to get the current time with approximately millisecond resolution.

Because there was no time to construct anything elaborate, the result is small and simple, but this is CMT's greatest strength. Programmers find that they can begin using CMT almost immediately because there is so little to learn (Dannenberg, Dannenberg, and Miller 1984). Also, CMT is very open-ended and therefore can be used for a wide range of tasks.

An interesting component of CMT is the software called Moxc, which is based on Douglas Collinge's Moxie language (Collinge 1985). The main idea in Moxc is that subroutine calls can be scheduled for execution in the future using a very simple expression; for example,

```
cause (150, play_melody, 15, 23, 0);
```

means "call `play_melody` 150 time units in the future, passing it the parameters 15, 23, and 0." With this one simple feature, it is possible to build very complex real-time systems (Dannenberg n.d.[c]). Because the system is so simple, programs run very efficiently and Moxc software tends to be very reliable, and important consideration for live performances.

Until recently, the CMT score language, called Adagio, and the CMT programming environment, Moxc, were entirely separate systems. This was unfortunate because the score language is quite nice for notating elaborate sequences, while Moxc is nice because it is programmable. For example, it is awkward to notate a melody in Moxc or a trill in Adagio.

The latest version of CMT has to some extent unified Adagio and Moxc: in fact, Adagio is now written using Moxc (Dannenberg 1990a). The consequence of this change is that users can write arbitrary real-time algorithms and invoke them from Adagio scores. An example would be an algorithm that generates a trill or a drum roll from a single command in an Adagio score. In addition, Moxc programs can load and play Adagio score files. This makes it relatively simple to, say, write a program that plays different sequences when triggered by specific events. The new system also incorporates an enhanced scheduler for greater efficiency (Dannenberg 1989f).

The CMU MIDI Toolkit has found a number of applications. One of its most common uses is playing score files that have been generated by another program. For example, scores intended for software synthesis can be "previewed" using MIDI (Beauchamp 1989). Adagio has also been used to play the output of many composition programs. Another class of appli-

cations is interactive compositions (Pennycook 1988), including the use of new controllers and real-time interfaces (Logemann 1989).

A recent addition to the toolkit is software for conducting a score by tapping beats on a keyboard or other MIDI controller (Dannenberg and Bookstein 1991). This software allows a conductor to synchronize a MIDI score with a live performance and is related to work in computer accompaniment described in "Computer Accompaniment," below.

Another new toolkit program is a real-time MIDI mapping program that transforms MIDI input according to a description called a map. For example, a single incoming note can be mapped to a multinote chord output, where the root of the chord and loudness are based on the input note. Inputs can also be mapped to musical sequences, with or without transposition.

Gestural Systems

In thinking about interfaces between musicians and computer music systems, Paul McAvinney came to the conclusion that interaction with a screen using hand gestures would be desirable. Because of limitations of existing touch-screen technologies, he set out to build something better, called the Sensor Frame. Paul continued to develop the Sensor Frame after joining the Computer Music Project and has now left Carnegie Mellon to pursue his interests full-time in optically based gesture sensing at the Sensor Frame Corporation.

The Sensor Frame is a rectangular frame surrounded by a light source and having an optical sensor in each of the four corners. Images of fingers within the frame are captured by the sensors, and triangulation is used to calculate the finger's position. Because of the multiple sensors, it is possible to track several fingers within the framework, allowing the user to control many parameters at once. It is possible to sense grasping and knob-turning gestures by tracking multiple fingers (Dannenberg and Amon 1989).

Dean Rubine's doctoral thesis (Rubine 1991a) examines the software issues of gesture recognition in detail. He has developed pattern classification software (Rubine n.d.) that can learn to identify gestures after observing a small number of examples (say, ten). The results will enable software developers to create gesture-based interfaces with a minimal amount of programming (Rubine 1991c).

In computer music work, gestures are often used to control continuous parameters; for example, a conductor can hold up a palm to mean "play softly," and the amount of arm extension gives some range of expression between "a little softer please" and the traffic police gesture that says "No, you are playing the wrong movement!" When gestures can be identified

from their initial movements, the remainder of the gesture can be tracked in real time to control one or more parameters (Rubine 1991b).

A related development is the VideoHarp (Rubine and McAvinney 1988; 1990), a controller based on the same sensing technology as is used in the Sensor Frame. The VideoHarp was invented by McAvinney and Rubine while working in the Computer Music Project and is now a commercial product of the Sensor Frame Corporation. Unlike the Sensor Frame, which uses multiple sensors and triangulation, the VideoHarp uses a single sensor and consequently has essentially one degree of freedom, the angle between the sensor and the performer's finger(s).

Figure 2 illustrates the geometry of the VideoHarp. The sensor looks through mirrors at the two surfaces that meet at the long edge of the device. Here, a light source produces back lighting to achieve high-contrast images of the player's fingers. In addition to the apparent angle from the sensor to the player's fingers, the angular width of the finger can be used as a coarse estimate of the distance from the sensor to the finger. The VideoHarp can see multiple fingers on each of two playing surfaces.

Software has been written for the VideoHarp that allows the user to play the VideoHarp like a keyboard, to use bowing and strumming motions, to use the VideoHarp surface like a drum, and to control pitch bend and other alterations. One interesting technique is to use bowing motions to step through a score each time the "bow" reverses directions and to use bow velocity to control loudness. This might be called a "sequential bow" (Mathews and Abbott 1980).

MUSIC REPRESENTATION

In our original plans for the Musician's Workbench, a music representation system played a central role. It was our intention to construct a rich facility for representing conventionally notated music, performance information, and experimental notations. The representation, or data structure, would have provided the common language with which various components such as editors and sequencers would communicate. The data structure has been implemented (Dannenberg 1986d; 1990c), and it provides the following features:

Scores. A piece of music is represented by a structure called a score.

Events. A piece of music, or score, is a set of events, data objects that may be created, deleted, or modified. Events are typically used to represent notes and rests, but they are also used to repre-

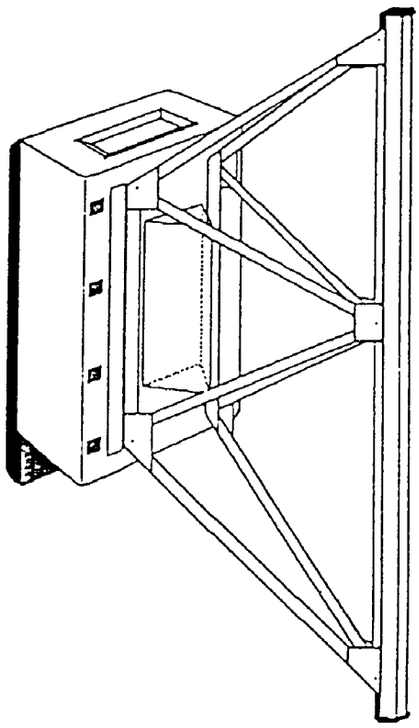


FIGURE 2. The VideoHarp, an optical gesture-sensing computer music instrument.

sent notational symbols like clefs and bar lines. Events can also represent abstract entities such as the set of notes selected for editing.

Properties. An event has a list of associated properties, each of which has an attribute, or name, and a value. For example, the duration of a quarter note is indicated by a property with attribute DURATION and value QUARTER.

Hierarchies. Events can be related in multiple hierarchical structures. For example, an event may represent a note that is a member of several overlapping phrases. The note may also be a member of a chord. These structural relationships can be represented explicitly as instances of hierarchies.

Views. An event can participate in any number of views, which support multiple graphical presentations of the same underlying data. For example, one window of a graphics display might show a section of an orchestral score, while another window shows a piano reduction of the same musical material. Like scores, views contain a set of events, most of which are normally shared with the score. Events in a view may have view-specific properties; for example, a note in a piano reduction may have its stem directed upward, while the same note in the score may have its stem directed downward. This can be indicated by having a view-specific property with attribute STEM-DIRECTION and value UP. The event would also have an ordinary non-view-specific property with STEM-DIRECTION DOWN.

Incremental update. A view can be associated with one or more regions of a computer display. When visible events of a view are modified, mechanisms provided by the data structure allow view software to modify the display incrementally. This is accomplished by automatic bookkeeping that records what has changed and what views are affected by the change.

Undo. The data structure uses a history mechanism to enable previous versions of the data structure to be restored. Undo can be applied iteratively to undo a sequence of changes (called Undo-more) and recursively to restore a change that was undone (called Undo-less). This undo facility is completely integrated with the incremental mechanism so that changes can be rapidly "undone."

Permanent storage. Scores can be written to a file and read back into memory. Circular data structures and shared structures are properly handled. Assuming music editing exhibits locality of reference, then saving and restoring a score will, as a side effect, rearrange its placement in virtual memory so as to optimize paging performance. This is especially important for large orchestral scores that are likely to take millions of bytes of storage.

Event sets. Events can be collected into sets (implemented as hierarchies). Abstract operations exist for creating and manipulating event sets. For example, one operation creates a set containing every event such that the value of a given property is greater than a given amount. Set union and intersection are also implemented.

Editing operations. A collection of abstract editing operations is provided. For example, one operation takes every event in a given set and increments the value of a given property by a given amount.

This aspect of the project was a very ambitious undertaking. The low-level representation seems to address many important issues in music representation, but years after the first publication of this work, commer-

cial music notation systems are still lacking most of the capabilities we advocate (Dannenberg 1989c), such as support for multiple hierarchies and multiple views. This is an indication of just how difficult a comprehensive music notation system is to design and implement.

Realizing (at last) that the ultimate representation system is beyond the means of a small research group, we hope to at least develop some interesting software for displaying and editing music using notations that are simpler than traditional notation. These will include "piano roll" scores and a graphical notation using traditional symbols. In this way, we hope to study the capabilities of a rich underlying structure without spending years implementing traditional notation.

MUSIC UNDERSTANDING

Real-time computer music systems are now so common that it is hard for many to imagine making music any other way. An entire industry has arisen to develop real-time control software, and a growing number of companies are producing interesting controllers for converting musical gestures into digital form.

As systems become more complex, we need to pay more and more attention to what the system users are trying to do. Understanding musicians on their own terms is a key requirement for the next generation of computer music systems (Dannenberg 1989d). With this in mind, let us look at some of the work on music understanding that has taken place within the Computer Music Project at Carnegie Mellon (Dannenberg n.d.[c]).

Computer Accompaniment

In 1983, after hearing a lecture about a sonar system for sensing a conductor's hand motions (Haflich and Burns 1983), I began to think about the problem of synchronizing computer systems with performing musicians. As both a musician and a scientist, I believed that it could be very difficult to make a machine interpret a conductor's gestures in the same way that a human would. Furthermore, if a conducting-based system were to lose synchronization by as little as half a measure, it would be difficult to know whether to speed up or slow down.

In contrast, synchronizing with other performers is potentially easier for two reasons. First, it is possible to measure the actual starting times of notes, and this can provide accurate synchronization points. Second, pitch sequences provide a reference that can help indicate the present location in the score.

The task of an accompaniment system is to synchronize computer-generated music with a live performance, continuously adapting to the tempo variations of the human performer(s) (Buxton, Dannenberg, and Vercoe 1986). An accompaniment system has two main components, called the matcher and the accompanist. The job of the matcher is to compare incoming pitch sequences to pitches that are notated in the score. Since the performer may omit notes, play extra notes, or play wrong notes, there are many combinations to consider. The accompaniment systems developed at Carnegie Mellon rely on a very powerful pattern-matching algorithm that normally considers every possible correspondence of performed notes to notated ones. This makes the matcher very robust even in the presence of multiple errors.

The second component is the accompanist. When the matcher recognizes a correspondence between the performance and the score, it forwards the time and location to the accompanist. The accompanist is a fairly autonomous process that reads the score and performs the parts designated as belonging to the computer. When reports come from the matcher, the accompanist updates its current estimate of score location and tempo. To avoid abrupt and unmusical output when the location or tempo is updated, the accompanist uses a set of rules that govern the way in which various situations are handled. For example, one rule says that if you are ahead of the other player, it is better to stop and wait a brief period of time than it is to back up and repeat a section of music.

My first accompaniment system was developed in early 1984, and a reimplementation that used a briefcase-size microcomputer made a demonstration possible at the International Computer Music Conference in the fall (Dannenberg 1985). The accompaniment algorithm, a real-time pitch detector, and a crude synthesizer were all operated by an 8-bit microprocessor in the tiny system.

Polyphonic Accompaniment

In the following year, graduate student Joshua Bloch and I implemented a new accompaniment system, and we each developed a matcher for following a polyphonic performance (Bloch and Dannenberg 1985). The detection of multiple pitches in a polyphonic acoustic signal is still largely unsolved, so we use an electronic keyboard as our source of performance information. We also developed a more sophisticated accompanist component.

In 1986, Hal Mukaino and I started work on yet another accompaniment system. This one would eventually incorporate a number of new features including the ability to handle trills, glissandi, and other figures that are

not notated precisely, better recovery mechanisms to handle cases where the matcher fails, and a MIDI-based implementation (Dannenberg and Mukaino 1988).

This system is capable of running two matchers simultaneously, each of which scans a segment of the score, called a matching window, or simply window. Whenever the performer stops (a frequent occurrence in rehearsals) the accompaniment will continue playing. It is reasonable for the performer to start in again either where he or she left off or at the current location of the accompanist. The earlier systems could find a match at only one of these two locations and would get lost if the performer made an entrance at the other location. The new system solves this problem by placing a matcher at each location so that either case will lead to a match and enable the accompanist to immediately resynchronize with the player.

Listening to Improvisations

Computer accompaniment assumes that a score is available and that the performer will play a consistent and recognizable sequence of pitches. In 1987, I began to look at the problem of following traditional jazz improvisations, where a performer's choice of pitches implies an underlying chord progression, but each performance will be different. Early attempts to find salient features that would disclose the chord structure were not successful, so a statistical approach was taken.

The task I chose to solve is just one of many possibilities. I wanted to build a system that could "listen" to a twelve-bar blues solo and identify the beginning of the twelve-bar chord pattern. The basic approach is to look for a correlation between the pitches that are performed and the expectation that a given pitch will be performed on a given beat. A simple search procedure looks for the starting point of the chord progression that yields the highest correlation (Dannenberg and Mont-Reynaud 1987). Figure 3 plots the correlation versus starting point. The correlation peaks at 0 and again at 96 (twelve measures of eighth notes), illustrating that the peak value indeed gives information about the starting point of the chord sequence.

Beat Tracking

The "blues listener" program depends upon being able to parse the performance into beats, since the correlation algorithm must know what pitches are played on each beat. I collaborated with Bernard Mont-Reynaud, a researcher at Stanford's CCRMA, on an implementation of a "blues listener" and used his beat-tracking algorithm that is similar to one proposed by Longuet-Higgins (Longuet-Higgins and Lee 1982).

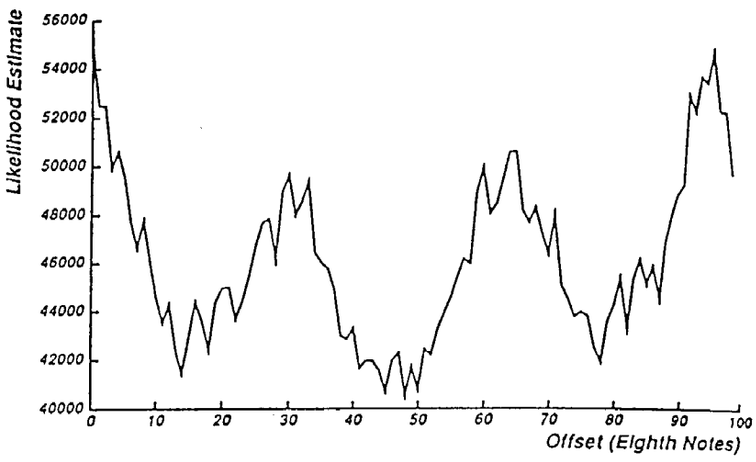


FIGURE 3. Likelihood estimates of the solo starting at different offsets in a twelve-bar blues progression.

Paul Allen, a graduate student at Carnegie Mellon, and I have analyzed and tested this algorithm extensively, concluding that it is not reliable enough for many practical situations. We have been working together on a more robust beat tracker, or "foot tapper," that works in real time and is based on searching multiple hypotheses in parallel (Allen and Dannenberg 1990).

The goal of a beat tracker is to process note onsets in real time, determining whether the note onset represents a downbeat, an upbeat, or some other rhythmic division. The problem is hard because the tempo is not known in advance, and tempo is not necessarily constant.

Our observation was that earlier beat trackers work well as long as the input is unambiguous. However, whenever the program is unsure, for example whether a note onset is on a downbeat or a sixteenth note early, there is some chance of making a mistake. Mistakes tend to lead to disasters because the interpretation of each note onset depends upon the interpretation of the previous one. Errors tend to propagate forward in time.

When faced with ambiguity, our new system does not make a final decision. Instead, it remembers a set of possible decisions. Each alternative is reconsidered on the next note onset. Alternatives that lead to consistent, reasonable interpretations of note onsets are kept, and others

are discarded. For example, our system would reject an interpretation of an eighth note on a downbeat tied to an eighth note triplet, preferring instead the more plausible explanation of two eighths in the presence of a tempo change. So far, the approach looks promising, but implementation still does not perform as well as we should expect. Our study is still in progress.

The Piano Tutor

Many people commented that the computer accompanist would have applications in education. Not only is the accompanist fun to play with and forgiving of beginner's mistakes, but a by-product of the matcher is a list of errors made by the performer.

The Piano Tutor Project (Dannenberg, Sanchez, Joseph, Capell, Joseph, and Saul 1990; Dannenberg, Sanchez, Joseph, Saul, Joseph, and Capell 1990) originated with two music professors at Carnegie Mellon, Marta Sanchez and Annabelle Joseph, who were interested in using computer technology to augment piano teaching and practicing. Working together, we decided to use an expert system for teaching beginners how to play the piano. The expert system embodies knowledge about teaching such as which errors are significant, how to select material, and when to interrupt a performance. A multimedia presentation system gives feedback to the student via videodisc, voice, computer graphics, music notation, and music output.

As it stands now, the Piano Tutor is able to provide intelligent feedback and remediation at a level that is appropriate for up to about one-and-one-half years of study. Some of the Piano Tutor's typical responses to student errors include pointing out wrong notes, identifying irregular tempo or rhythmic errors, and locating notes that were played too short. When a student plays a lesson correctly, the Piano Tutor offers to play an accompaniment as a "reward" and to encourage additional practice. Then, based on the skills the student has demonstrated up to this point, the Piano Tutor selects a new lesson for the student to master.

The Piano Tutor Project is large (at least by the standards of most computer music researchers): The implementation took three years and about ten person years to complete. As of the summer of 1991, we are testing the system with students to see how well they like the system, how fast they learn, and what differences we can observe between these students and students learning by the traditional method. So far, students seem to get tremendous satisfaction from this style of instruction, and the rate and quality of learning are high.

COMPOSITION

The Computer Music Project has focused mainly on the development of computer music systems and techniques. Nevertheless, there has been at least a limited output of music. Notable works have been composed using the facilities and software of the Computer Music Project by Denise Ondishko, who used Arctic, Chris Koenigsberg, who used the CMU MIDI Toolkit, and Peter Velikonja, who has used the CMU MIDI Toolkit in conjunction with software synthesis. Ken Bookstein at New York University has given concerts using polyphonic accompaniment software developed at Carnegie Mellon in conjunction with Yamaha and also the conducting software described in "The CMU MIDI Toolkit," above. Reza Vali, assistant professor of music at Carnegie Mellon, composed a work for chamber ensemble and VideoHarp that was premiered in September 1990. Peter Velikonja composed *Spomin* (Velikonja 1991) using the Fugue synthesis and composition language (see below). The following section describes two compositions languages, and then some specific musical projects at the Computer Music Project will be discussed.

Composition Languages

The Canon Score Language (Dannenberg 1982) is an attempt to apply the semantics of Arctic to the non-real-time problem of expressing music. It was hoped that the result would be easy to learn, extensible, intuitive, flexible, and interactive. With most other language approaches, notes are data, and programs can manipulate the data to perform operations such as copying, transposing, and shifting. In contrast, in Canon even a sequence of notes is a program, but Canon has some very intuitive operators that act on programs as opposed to data to perform transformations. This makes the language more uniform.

For example, a sequence of notes in Canon can be written as:

```
(seq (note C4 Q)
(note D4 H)
(note Ef4 Q))
```

where "`seq`" means "sequence," "`C4`," "`D4`," and "`Ef4`" are pitches, and "`Q`" and "`H`" are durations. Ignoring the parentheses, this notation is similar in form to note lists of Music V (Mathews 1969) and many other commonly used score languages. However, since Canon sequences are actually programs, one can embed commands other than "note" within sequences. For example, here is a sequence containing a sequential repetition ("`seqrep`") of a note twelve times, and a glissando ("`gliss`").

COMPOSITION

The Computer Music Project has focused mainly on the development of computer music systems and techniques. Nevertheless, there has been at least a limited output of music. Notable works have been composed using the facilities and software of the Computer Music Project by Denise Ondishko, who used Arctic, Chris Koenigsberg, who used the CMU MIDI Toolkit, and Peter Velikonja, who has used the CMU MIDI Toolkit in conjunction with software synthesis. Ken Bookstein at New York University has given concerts using polyphonic accompaniment software developed at Carnegie Mellon in conjunction with Yamaha and also the conducting software described in "The CMU MIDI Toolkit," above. Reza Vali, assistant professor of music at Carnegie Mellon, composed a work for chamber ensemble and VideoHarp that was premiered in September 1990. Peter Velikonja composed *Spomin* (Velikonja 1991) using the Fugue synthesis and composition language (see below). The following section describes two compositions languages, and then some specific musical projects at the Computer Music Project will be discussed.

Composition Languages

The Canon Score Language (Dannenberg 1982) is an attempt to apply the semantics of Arctic to the non-real-time problem of expressing music. It was hoped that the result would be easy to learn, extensible, intuitive, flexible, and interactive. With most other language approaches, notes are data, and programs can manipulate the data to perform operations such as copying, transposing, and shifting. In contrast, in Canon even a sequence of notes is a program, but Canon has some very intuitive operators that act on programs as opposed to data to perform transformations. This makes the language more uniform.

For example, a sequence of notes in Canon can be written as:

```
(seq (note C4 Q)
(note D4 H)
(note Ef4 Q))
```

where "`seq`" means "sequence," "`C4`," "`D4`," and "`Ef4`" are pitches, and "`Q`" and "`H`" are durations. Ignoring the parentheses, this notation is similar in form to note lists of Music V (Mathews 1969) and many other commonly used score languages. However, since Canon sequences are actually programs, one can embed commands other than "note" within sequences. For example, here is a sequence containing a sequential repetition ("`seqrep`") of a note twelve times, and a glissando ("`gliss`").

```
(seq (note C4 Q)
     (seqrep (i 12) (note D4 S))
     (gliss Ef4 Ef5)
     (note Ef5 Q))
```

The "gliss" command is not built into Canon, but Canon can be easily extended by defining new functions. The following defines "myphrase" to be a simple sequence of three notes:

```
(defun myphrase ()
    (seq (note C4 Q) (note D4 H) (note Ef4 Q )))
```

Canon supports a variety of transformations, including time shifting, transposition, and stretching. For example, the following will play "myphrase" twice with different transformations:

```
seq (transpose 5 (myphrase))
    (stretch 2.0 (transpose 7 (myphrase))))
```

In addition to fixed transformations such as stretching everything by a factor of 2, Canon allows time-varying transformations to be used. This makes it easy to impose effects such as a crescendo or accelerando to a score.

Canon is implemented in the Lisp programming language, and all of the functions and data structures of Lisp are available for use within Canon programs. This means that Canon can be used for a variety of compositional strategies, ranging from traditional note lists to algorithmic composition and artificial intelligence. Canon is interactive, and the output is an Adagio score file, which is compatible with the CMU MIDI Toolkit.

Because Canon is designed to control MIDI, Canon does not give composers direct control over timbre, which is left to the particular synthesizer used to make the sound. After completing Canon, it became apparent that the ideas developed in Canon could be extended to deal with sounds below the level of MIDI commands. A new language, Fugue, was designed and implemented to explore this possibility (Dannenberg and Fraley 1989; Dannenberg, Fraley, and Velikonja 1991).

A primary motivation for Fugue was the recognition that almost none of our sound synthesis research was actually being used for composition. The problem has been that our research programs tend to be special-purpose. They are designed just to demonstrate the ability to make interesting sounds, but no effort has been made to connect the sound-producing software to score languages and composition environments. It is hoped that Fugue will provide a convenient environment for both composition *and* synthesis research so that each activity can benefit from the other.

Fugue is very similar to Canon, except the "note" function in Canon is replaced with a fairly extensive facility for producing sounds using software synthesis. Each function in Canon computes a sound that can be manipulated in turn by some other function. For example, there is an "`osc`" function that implements a table-lookup oscillator, returning the resulting sound. This sound can be modified by a filter function.

In the following expression, the "`osc`" function computes a tone using a default waveform, and "`amp-env`" is a user-defined function that computes an amplitude envelope. The two sounds (even envelopes and other control functions are considered to be sounds) are multiplied together using "`s-mult`." "`lp-env`" is a user-defined filter frequency control function, and "`low-pass`" implements a variable low-pass filter that is used here to filter the enveloped output of the oscillator.

```
(low-pass (lp-env)
          (s-mult (amp-env)
          (osc C4 H)))
```

Using the function-definition facilities of Fugue, the composer can define a personalized orchestra of sounds and sound-manipulation functions. All of the facilities for representing scores that are present in Canon are also available in Fugue, offering the composer a ready-made but extensible score language from which to invoke the new sounds.

The implementation of Fugue was begun by Chris Fraley in 1989, and his work was continued by George Polly in the fall. It is our hope to extend Fugue with more synthesis techniques for composers and also to use Fugue for work on spectral interpolation synthesis.

Composed Improvisation

In 1985, I met Georges Bloch, who was then a graduate student at the University of California at San Diego. With Xavier Chabot, we decided to collaborate on a new piece, and I invited Georges and Xavier to visit Carnegie Mellon. I had in mind trying to apply the pattern matcher used for computer accompaniment to recognize phrases played by an improviser.

We set out to write a piece for keyboard, flute, trumpet, and computer in which each of us, as performers, would be interfaced to the computer, and each of us would control some aspect of the piece. The program/composition, *Jimmy Durante Boulevard,* analyzes the keyboard input for harmonic content, the trumpet input for melodic phrases, and the flute input for pitch contour. All of this information is used in real time to provide source material to a program that manipulates the material according to a high-level plan that is input at the start of the performance.

Jimmy Durante Boulevard was written by the three of us (Chabot, Dannenberg, and Bloch 1986; Dannenberg 1989b) and first performed at the STEIM Studio in 1986. The final preparation of the software and concert arrangements was interesting in that Xavier, Georges, and I were located in San Diego, Pittsburgh, and Paris, respectively. We communicated via electronic mail, bringing a new dimension to the use of computers in music!

My personal musical directions have been strongly influenced by this experience, and I have since written several works of what has been called "composed improvisation." In these pieces, one or more performers interact with a real-time computer music system running the "composition." The live performer improvises, but the computer's responses provide real-time music direction and structure that can be carefully designed by the composer. Thus, composed improvisation can offer the spontaneity of free improvisation but also retain attention to detail, form, and design that are characteristic of traditional Western music (Dannenberg n.d.[b]).

Computer Music and Graphics

The most recent area of artistic research within the Computer Music Project has been the incorporation of real-time computer graphics into computer music compositions (Dannenberg 1989e). This work has evolved from work on composed improvisation: It turned out to be a very small extension to add graphical output to programs that were already generating music in response to live performers. The result is interesting because of the organic connection between image and sound.

Unlike images based on waveforms or spectral content, images can be based on "deep structure" in the music, which may have very little to do with individual notes and much more to do with abstract form and musical organization. Works of composed improvisation are a natural context for this work on graphics because any program that produces a real-time response to performers must have a rich internal representation corresponding to many different levels of abstraction in the music. The program itself is also a representation of the intent of the composer and his or her high-level concepts. These representations provide a starting point for the design of animation and graphics that truly reflect and reinforce the structure of the music.

The most recent of these works is *Ritual of the Science Makers* (Dannenberg 1989g), written in celebration of the twenty-fifth anniversary of computer science at Carnegie Mellon. This work calls for flute, violin, and cello, all interfaced to a real-time program that produces musical and

graphical output. The work is partly composed and partly improvised; the computer program provides a structure within which there is a lot of freedom for the players. The implementation of *Ritual* made extensive use of the CMU MIDI Toolkit.

FUTURE DIRECTIONS

Digital technology continues to race forward at a rate that is almost impossible to comprehend. Portable computer music systems produce in real time today what mainframe computers spent hours computing only a decade ago. As we look forward another decade, we can assume that whatever we do on large systems in non-real time today we will be able to pack in a briefcase and perform in real time by the year 2000.

Today we are seeing a trend toward live performance with computer controlled synthesizers. Since portable synthesizers have limited facilities for storing and manipulating live sound, we see a strong tendency to interact with computers at the note and gestural level rather than the audio level. In the future, we can expect to see portable synthesizers that perform real-time resampling, spectral analysis, and reconstruction of the sounds of live instruments. Composers will no doubt find interesting uses for live audio processing. More complex music understanding systems will evolve and take advantage of the additional computing available.

For non-real time systems, composition languages and systems will run faster and enable much more sound processing. For example, reverberation and spacialization software exists but is used sparingly because of the computation involved. In the future it will be routine to specify individually the spacial location as well as timbre of synthetic instruments.

Computer graphics will make great strides, enabling sophisticated animated images to be controlled in real time using portable systems. Real-time video processing will augment synthetic images, and some amount of real-time image understanding combined with music understanding will enable new forms of composition and theater. This could lead to interactive multimedia performances on the scale of grand opera, or it could enable more personal interactive performances for listeners and viewers to experience at home.

It should be apparent from all of the topics mentioned so far that the Computer Music Project has many active research directions. In the future, we hope to pursue further development of spectral interpolation synthe-

graphical output. The work is partly composed and partly improvised; the computer program provides a structure within which there is a lot of freedom for the players. The implementation of *Ritual* made extensive use of the CMU MIDI Toolkit.

FUTURE DIRECTIONS

Digital technology continues to race forward at a rate that is almost impossible to comprehend. Portable computer music systems produce in real time today what mainframe computers spent hours computing only a decade ago. As we look forward another decade, we can assume that whatever we do on large systems in non-real time today we will be able to pack in a briefcase and perform in real time by the year 2000.

Today we are seeing a trend toward live performance with computer controlled synthesizers. Since portable synthesizers have limited facilities for storing and manipulating live sound, we see a strong tendency to interact with computers at the note and gestural level rather than the audio level. In the future, we can expect to see portable synthesizers that perform real-time resampling, spectral analysis, and reconstruction of the sounds of live instruments. Composers will no doubt find interesting uses for live audio processing. More complex music understanding systems will evolve and take advantage of the additional computing available.

For non-real time systems, composition languages and systems will run faster and enable much more sound processing. For example, reverberation and spacialization software exists but is used sparingly because of the computation involved. In the future it will be routine to specify individually the spacial location as well as timbre of synthetic instruments.

Computer graphics will make great strides, enabling sophisticated animated images to be controlled in real time using portable systems. Real-time video processing will augment synthetic images, and some amount of real-time image understanding combined with music understanding will enable new forms of composition and theater. This could lead to interactive multimedia performances on the scale of grand opera, or it could enable more personal interactive performances for listeners and viewers to experience at home.

It should be apparent from all of the topics mentioned so far that the Computer Music Project has many active research directions. In the future, we hope to pursue further development of spectral interpolation synthe-

sis, real-time implementations of the Arctic language, more sophisticated music understanding systems, and the Fugue language for software synthesis.

We have recently begun to construct an integrated music system incorporating Fugue for signal processing, our music representation software (see "Music Representation," above) for music editing, an instrument editor (Dannenberg, Rubine, and Neuendorffer 1991), and a signal editor. The system will include MIDI and digital audio interfaces.

On the artistic front, we hope to make more research results available for composition and performance. For example, integrating computer accompaniment software with the CMU MIDI Toolkit will allow pattern recognition and score following to be called upon for a variety of applications in interactive experimental music. Incorporating our signal processing research results into Fugue will make them easier to use by nonexperts.

Composition and performance will continue, with an emphasis on real-time interactive systems. The integration of computer music, computer graphics, and video is particularly attractive at this point in time.

CONCLUSION

The Computer Music Project at Carnegie Mellon University is a part of the School of Computer Science, and our mission has been to conduct computer science research in directions that will contribute to the art of music. The demands of music are high, and they have led us to innovative work in computer science. Among our contributions are new languages for expressing temporal behavior, new synthesis techniques, technology for gesture sensing and processing, music understanding techniques including real-time score following and computer accompaniment, schemes for music representation, and the development of systems for real-time interactive computer music.

Much of our work has been predicated on the belief that real-time music systems are of primary importance and that the main problem is not fast computation but the ability to express and control real-time behavior. We have focused on control, developing gesture-sensing devices, software and languages for real-time computer music, music understanding systems, and a synthesis technique offering fine control over the time-varying spectrum. As part of the worldwide community of computer music researchers, we will be interested to see the impact of this technology upon the music of the future.

ACKNOWLEDGMENTS

I would like to thank the School of Computer Science, the Music Department, the Studio for Creative Inquiry (formerly the Center for Art and Technology), and the Information Technology Center for their support over the years. The research described in this chapter has been supported by DARPA (Department of Defense), Yamaha, Inc., the Markle Foundation, NeXT, Inc., Commodore-Amiga, Inc., Hughes Aircraft Co., the IBM Corporation, and Apple Computer Company.

REFERENCES

Abbott, C. "The 4CED Program." *Computer Music Journal* 5,1 (Spring 1981): 13–33.

Allen, P. E., and R. B. Dannenberg. "Tracking Musical Beats in Real Time." In *ICMC Glasgow 1990 Proceedings*, ed. S. Arnold and G. Hair. San Francisco: Computer Music Association, 1990.

Beauchamp, J. "The Computer Music Project at the University of Illinois at Urbana-Champaign: 1989." In *Proceedings of the 1989 International Computer Music Conference*, ed. T. Wells and D. Butler. San Francisco: Computer Music Association, 1989.

Bloch, J. J., and R. B. Dannenberg. "Real-Time Computer Accompaniment of Keyboard Performances." In *Proceedings of the International Computer Music Conference, 1985*. ed. Barry Truax. San Francisco: Computer Music Association, 1985.

Buxton, W., R. B. Dannenberg, and B. Vercoe. "The Computer as Accompanist." In *Proc. CHI'86 Human Factors in Computing Systems*, ed. M. Mantei and P. Orbeton. New York: Association for Computer Machinery, 1986.

Chabot, X., R. Dannenberg, and G. Bloch. "A Workstation in Live Performance: Composed Improvisation." In *Proceedings of the International Computer Music Conference, 1986*, ed. P. Berg. San Francisco: Computer Music Association, 1986.

Collinge, D. J. "MOXIE: A Language for Computer Music Performance." In *Proceedings of the International Computer Music Conference, 1984*, ed. William Buxton. San Francisco: Computer Music Association, 1985.

Dannenberg, F. K., R. B. Dannenberg, and P. Miller. "Teaching Programming to Musicians." In *Proceedings of the Fourth Symposium on Small Computers in the Arts*, ed. D. Mansfield. Washington, D.C.: IEEE Computer Society Press, 1984.

ACKNOWLEDGMENTS

I would like to thank the School of Computer Science, the Music Department, the Studio for Creative Inquiry (formerly the Center for Art and Technology), and the Information Technology Center for their support over the years. The research described in this chapter has been supported by DARPA (Department of Defense), Yamaha, Inc., the Markle Foundation, NeXT, Inc., Commodore-Amiga, Inc., Hughes Aircraft Co., the IBM Corporation, and Apple Computer Company.

REFERENCES

Abbott, C. "The 4CED Program." *Computer Music Journal* 5,1 (Spring 1981): 13–33.

Allen, P. E., and R. B. Dannenberg. "Tracking Musical Beats in Real Time." In *ICMC Glasgow 1990 Proceedings*, ed. S. Arnold and G. Hair. San Francisco: Computer Music Association, 1990.

Beauchamp, J. "The Computer Music Project at the University of Illinois at Urbana-Champaign: 1989." In *Proceedings of the 1989 International Computer Music Conference*, ed. T. Wells and D. Butler. San Francisco: Computer Music Association, 1989.

Bloch, J. J., and R. B. Dannenberg. "Real-Time Computer Accompaniment of Keyboard Performances." In *Proceedings of the International Computer Music Conference, 1985*. ed. Barry Truax. San Francisco: Computer Music Association, 1985.

Buxton, W., R. B. Dannenberg, and B. Vercoe. "The Computer as Accompanist." In *Proc. CHI'86 Human Factors in Computing Systems*, ed. M. Mantei and P. Orbeton. New York: Association for Computer Machinery, 1986.

Chabot, X., R. Dannenberg, and G. Bloch. "A Workstation in Live Performance: Composed Improvisation." In *Proceedings of the International Computer Music Conference, 1986*, ed. P. Berg. San Francisco: Computer Music Association, 1986.

Collinge, D. J. "MOXIE: A Language for Computer Music Performance." In *Proceedings of the International Computer Music Conference, 1984*, ed. William Buxton. San Francisco: Computer Music Association, 1985.

Dannenberg, F. K., R. B. Dannenberg, and P. Miller. "Teaching Programming to Musicians." In *Proceedings of the Fourth Symposium on Small Computers in the Arts*, ed. D. Mansfield. Washington, D.C.: IEEE Computer Society Press, 1984.

Dannenberg, R. B. "Resource Sharing in a Network of Personal Computers." Ph.D. diss., Carnegie Mellon University, 1982.

——. "Arctic: A Functional Language for Real-Time Control." In *1984 ACM Symposium on LISP and Functional Programming.* New York: Association for Computer Machinery, 1984.

——. "An On-Line Algorithm for Real-Time Accompaniment." In *Proceedings of the International Computer Music Conference, 1984,* ed. William Buxton. San Francisco: Computer Music Association, 1985.

——. "Arctic: A Functional Language for Real-Time Control." *IEEE Software* 3,1 (Jan. 1986a): 70–71.

——. "Arctic: Functional Programming for Real-Time Systems." In *Proceedings of the Nineteenth Hawaii International Conference on System Sciences,* ed. B. Shriver. N. Hollywood: Western Periodicals, 1986b.

——. "The CMU MIDI Toolkit." In *Proceedings of the International Computer Music Conference, 1986,* ed. P. Berg. San Francisco: Computer Music Association, 1986c.

——. "A Structure for Representing, Displaying, and Editing Music." In *Proceedings of the International Computer Music Conference, 1986,* ed. P. Berg. San Francisco: Computer Music Association, 1986d.

——. "Workstations for Computer Music at Carnegie-Mellon." In *1986 University AEP Conference,* ed. F. Dwyer. Milford, Conn.: IBM Academic Information Systems, 1986e.

——. "Systemes pour informatique musicale a l'universite de Carnegie-mellon." In *Actes du Symposium Systemes Personnels et Informatique Musicale,* ed. J.-B. Barrière, F. Armand, and C. Marquet. Paris: IRCAM, 1987.

——. "The Canon Score Language." *Computer Music Journal* 13/1 (Spring 1989a): 47–56.

——. *Jimmy Durante Boulevard.* Compact disc accompanying *Current Directions in Computer Music Research,* ed. M. V. Mathews and J. R. Pierce. Cambridge, Mass.: MIT Press, 1989b.

——. "Music Representation Issues: A Position Paper." In *Proceedings of the 1989 International Computer Music Conference,* ed. T. Wells and D. Butler. San Francisco: Computer Music Association, 1989c.

——. "Music Understanding." *Computer Science Research Review 1987/1988,* ed. C. Copetas. Pittsburgh: Carnegie Mellon School of Computer Science, 1989d.

——. "Real Time Control for Interactive Computer Music and Animation." In *The Arts and Technology II: A Symposium,* ed. N. Zahler. New London, Conn.: Connecticut College, 1989e.

——. "Real-Time Scheduling and Computer Accompaniment." In *Current Directions in Computer Music Research*, ed. M. V. Mathews and J. R. Pierce. Cambridge, Mass.: MIT Press, 1989f.

——. "Software Support for Interactive Multimedia Performance." In *Proceedings of The Arts and Technology 3: A Symposium*, ed. D. Smalley, N. Zahler, and C. Luce. New London, Conn.: Connecticut College, 1989g.

——. "Recent Developments in the CMU MIDI Toolkit." In *Proceedings of the International Seminar Ano 2000: Theoretical, Technological and Compositional Alternatives*, ed. C. Sandoval. Mexico City: University of Mexico, 1990a.

——. "A Run-Time System for Arctic." In *ICMC Glasgow 1990 Proceedings*, ed. S. Arnold and G. Hair. San Francisco: Computer Music Association, 1990b.

——. "A Structure for Efficient Update, Incremental Redisplay, and Undo in Display-Oriented Editors." *Software: Practice and Experience* 20.2 (Feb. 1990c): 109–32.

——. "Expressing Temporal Behavior Declaratively." In *CMU Computer Science: A 25th Anniversary Commemorative*, ed. R. Rashid. Reading, Mass.: Addison-Wesley, 1991a.

——. "Computer Accompaniment and Music Understanding." In *Proceedings of the 1991 KlangArt Kongress*, ed. B. Enders. Osnabrueck, Germany: Universitat Osnabrueck, n.d.(a).

——. "Extending Music Notation through Programming." *Contemporary Music Review*. n.d.(b).

——. "Software Techniques for Interactive Performance Systems." In *International Workshop on Man-Machine Interaction in Live Performance*, ed. Pisa: Scuola di Studi Superiori Universitari e di Perfezionamento, n.d.(c).

Dannenberg, R. B., and D. Amon. "A Gesture Based User Interface Prototyping System." In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*. New York: ACM, 1989.

Dannenberg, R. B., and K. Bookstein. "Practical Aspects of a MIDI Conducting Program." In *ICMC Montreal 1991 Proceedings*, ed. B. Alphonce and B. Pennycook. San Francisco: Computer Music Association, 1991.

Dannenberg, R. B., and C. L. Fraley. "Fugue: Composition and Sound Synthesis with Lazy Evaluation and Behavioral Abstraction." In *Proceedings of the 1989 International Computer Music Conference*, ed. T. Wells and D. Butler. San Francisco: Computer Music Association, 1989.

Dannenberg, R. B., C. L. Fraley, and P. Velikonja. "Fugue: A Functional Language for Sound Synthesis." *Computer* 24,7 (July 1991): 36–41.

Dannenberg, R. B., and P. McAvinney. "A Functional Approach to Real-Time Control." In *Proceedings of the International Computer Music Conference, 1984*, ed. [illegible] San Francisco: Computer Music Association, 1985.

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

Dannenberg, R. B., P. McAvinney, and M. T. Thomas. "Carnegie-Mellon University Studio Report." In *Proceedings of the International Computer Music Conference, 1984*, ed. William Buxton. San Francisco: Computer Music Association, 1985.

Dannenberg, R. B., and B. Mont-Reynaud. "Following an Improvisation in Real Time." In *Proceedings of the International Computer Music Conference, 1987*, ed. J. Beauchamp. San Francisco: Computer Music Association, 1987.

Dannenberg, R. B., and H. Mukaino. "New Techniques for Enhanced Quality of Computer Accompaniment." In *Proceedings of the 14th International Computer Music Conference*, ed. C. Lischka and J. Fritsch. San Francisco: Computer Music Association, 1988.

Dannenberg, R. B., D. Rubine, and T. Neuendorffer. "The Resource-Instance Model of Music Representation." In *ICMC Montreal 1991 Proceedings*, ed. B. Alphonse and B. Pennycook. San Francisco: Computer Music Association, 1991.

Dannenberg, R. B., M. Sanchez, A. Joseph, P. Capell, R. Joseph, R. Saul. "A Computer-Based Multi-media Tutor for Beginning Piano Students." *Interface* 19 (1990): 155–73.

Dannenberg, R. B., M. Sanchez, A. Joseph, R. Saul, R. Joseph, P. Capell. "An Expert System for Teaching Piano to Novices." In *Proceedings of the International Computer Music Conference, 1990*, ed. S. Arnold and G. Hair. San Francisco: Computer Music Association, 1990.

Dannenberg, R. B., M. H. Serra, and D. Rubine. "Comprehensive Study of Analysis and Synthesis of Tones by Spectral Interpolation." *Journal of the Acoustical Society of America* 82, supp. 1 (Fall 1987): S69.

Haflich, S. M., and M. A. Burns. "Following a Conductor: The Engineering of an Input Device." Paper presented at the 1983 International Computer Music Conference, Oct. 7–10, 1983, Rochester, N.Y.

Logemann, G. W. "Experiments with a Gestural Controller." In *Proceedings of the 1989 International Computer Music Conference*, ed. T. Wells and D. Butler. San Francisco: Computer Music Association, 1989.

Longuet-Higgins, H. C., and C. S. Lee. "The Perception of Musical Rhythms." *Perception* 11 (1982): 115–28.

Mathews, M. V. *The Technology of Computer Music*. Cambridge, Mass.: MIT Press, 1969.

Mathews, M. V., and C. Abbott. "The Sequential Drum." *Computer Music Journal* 4,4 (Winter 1980): 45–59.

Pennycook, B. W. "Praescio-II: Amnesia, toward Dynamic Tapeless Performance." In *Proceedings of the 14th International Computer Music Conference*, ed. C. Lischka and J. Fritsch. San Francisco: Computer Music Association, 1988.

Rubine, D. "The Automatic Recognition of Gestures." Ph.D. diss., Carnegie Mellon University, 1991a.

——. "Integrating Gesture Recognition and Direct Manipulation." In *Proceedings of the Summer 1991 Usenix Conference*, ed. C. Carr. Berkeley: USENIX Association, 1991b.

——. "Specifying Gestures by Example." *Computer Graphics* 25,4 (July 1991c): 329–37.

——. "Criteria for Gesture Recognition Technologies." In *Neural Networks and Pattern Recognition in Human Computer Interaction*, ed. Russell Beale and Janet Finlay. Chichester, UK: Ellis Horwood, n.d.

Rubine, D., and R. B. Dannenberg. *Arctic Programmer's Manual and Tutorial*. Technical Report CMU-CS-87-110. Pittsburgh: Carnegie Mellon University, 1987.

Rubine, D., and P. McAvinney. "The VideoHarp." In *Proceedings of the 14th International Computer Music Conference*, ed. C. Lischka and J. Fritsch. San Francisco: Computer Music Association, 1988.

——. "The VideoHarp." *Computer Music Journal* 14,1 (Spring 1990): 26–41.

Serra, M., D. Rubine, R. B. Dannenberg. "The Analysis and Resynthesis of Tones via Spectral Interpolation." In *Proceedings of the 14th International Computer Music Conference*, ed. C. Lischka and J. Fritsch. San Francisco: Computer Music Association, 1988a.

——. *A Comprehensive Study of the Analysis and Synthesis of Tones by Spectral Interpolation*. Technical Report CMU-CS-88-146. Pittsburgh: Carnegie Mellon University, 1988b.

——. "Analysis and Synthesis of Tones by Spectral Interpolation." *Journal of the Audio Engineering Society* 38,3 (March 1990): 111–28.

Simon, H. A. "Perception du pattern musical par AUDITEUR." *Sciences de l'art* 2 (1968): 28–34.

Smith, J. O., and P. Gossett. "A Flexible Sampling-Rate Conversion Method." In *Proceedings of the IEEE Conference on ASSP*. New York: IEEE, 1984.

Smith, J. O., and X. Serra. "PARSHL: An Analysis/Synthesis Program for Non-Harmonic Sounds Based on a Sinusoidal Representation." In *Proceedings of the 1987 International Computer Music Conference*, ed. J. Beauchamp. San Francisco: Computer Music Association, 1987.

Thomas, M. T. "Vivace: A Rule Based AI System for Composition." In *Proceedings of the International Computer Music Conference, 1985*, ed. Barry Truax. San Francisco: Computer Music Association, 1985.

Thomas, M. T., S. Chatterjee, M. W. Maimone. "Cantabile: A Rule Based System for Composing Melody." In *Proceedings of the 1989 International Computer Music*

Conference, ed. T. Wells and D. Butler. San Francisco: Computer Music Association, 1989.

Thomas, M. T., and P. Monta. *MacVoice 1.0 User's Manual.* Santa Barbara, Calif.: Kinko's Academic Courseware Exchange, 1986.

Velikonja, P. *Spomin.* Sound recording (CD or cassette) accompanying *Computer* 24,7 (July 1991).