Computer Systems and Languages for Audio Research

BARRY VERCOE

Massachusetts Institute of Technology, Experimental Music Studio, Cambridge, MA 02139, USA

While most of the professional audio industry is preoccupied with the reproduction of recorded music created by natural instruments, there is a more profound application: creation of original music with digital signal processing. The historic limitations of natural air and string resonances can be overcome by the use of computer sound synthesis. Although computers can generate any sound that can be specified in point-by-point fashion, creative exploration of new timbres and new musical effects requires fabrication of new signal-processing structures at the level of software programming. For both musician and engineer, replacing an instrument by a terminal—or an analog pot by a subroutine—can be very disturbing, particularly if the modes of human–machine interaction are orthogonal to the task. A composer-oriented software system is described which affords intuitive yet flexible control over the most recent methods of digital audio processing.

1 SOFTWARE SYSTEMS

Growing interest in computer music composition and performance has led to the development of both hardware and software systems for efficient investigation of its potential. The first program that could support serious research in digital music synthesis was Music 4, developed in 1963 by Max Mathews at Bell Laboratories. Music 4 established a modular principle for the description of audio processing networks that has enabled composers and researchers to communicate in common terms despite the variety of computer systems they use. The many later variants of Music 4 have provided users with digital audio descriptor languages that are musically intuitive, yet close to the hardware of a particular machine for the sake of speed.

The major software systems currently in use are Music 360, a variant of Music 4, which runs on large IBM systems; Music 10, designed for the DEC PDP-10; Music 11, which runs on any DEC PDP-11 computer; Music 4BF, a FORTRAN version designed for portability; and Mathews' Music 5. Music 5, Music 360, and Music 11 are the most widely distributed, the first two in research centers with large centralized computing. Music 11 suits a different pocketbook, and has induced several music departments such as Eastman and Brooklyn College to buy their own PDP-11 computers exclusively for music composition and research.

Cost-saving advances have included elimination of redundancies. Observation of Music 360 composers, for instance, reveals that up to 50% of music signal processing is aimed at shaping loudness and pitch contours—functions essentially of acoustic control that need not be computed at audio rates. Music 11 was therefore designed with two separate levels of processing, *control signals* and *audio signals*. Slower signals such as vibratos can be calculated at less expensive data rates (at 1 kHz rather than 50 kHz), providing an overall computational saving of about 40%.

2 THE HUMAN–MACHINE INTERFACE

The effectiveness of digital audio processing can be improved by better human–machine communication. For example, the M.I.T. studio has developed a powerful and sophisticated *music score editor*, manipulating data on a screen in standard music notation (Fig. 1). The score editor is effective as a composing tool, not because it is aimed at producing a good final layout, but because it manages the more difficult task of *maintaining the editability* of a work in progress.

Music in the past has been represented by a fluently pictorial yet imprecise symbology. For a computer, however, symbols such as *mp* or *fff* must be given numeric values by way of translation tables. When these values are written out in full, they create a score file that provides very little sense of the music it represents. Most computer music today is painstakingly encoded in just this form. An encoding scheme that incorporates alphabetic characters as well is sometimes used (Fig. 2). This is of some help because it minimizes the amount of visual data required to represent a full score. The preferred notation for much computer music is standard notation, simply because it is so immediate. Moreover, failure to support standard practices of music notation will impair the speed of human–machine communication, and so reduce the artistic effectiveness of the signal-processing system it will invoke.

In the area of digital audio processing itself, defining a signal-processing network by the use of deft sketching motions can be conducive to creative experiment (Fig. 3). People naturally conceive of the "patched module" networks of time-domain synthesis largely through inner imagery—as a collection of boxes arranged spatially and interconnected by wires. When contemplating changes, they find it natural to make boxes appear and disappear as needed, and to imagine plugging and unplugging the connecting wires. The symbolic objects here are typically time-domain signal-processing modules at the level of oscillator, filter, envelope, and so on.

The goal of OEDIT, our orchestra editor, is to foster ad hoc experimentation through the medium of high-level symbolic programming, in such a way as to exploit speed and directness of human–machine interaction as in investigative utility. However, the use of graphics is not the simplifying step it might seem to be. Representing arbitrary signal-processing networks in this way makes it difficult to specify both *signal* flow and *programming* flow of control. The ideal that we are seeking, of course, is something that is both a signal-processing language and a programming language. Unfortunately the characteristics of the two are not identical, as we will see by examining some programming concepts in more detail.

```
% J. S. Bach invention #6 %

orchestra {
        top     oscin
        bot     oscin
        rev     reverb 1
}
functions {
        f1 0 256 10 1
        f2 0 256 10 1 1 0 1
        i3 0 31
}
score {
$top #fcgd t60

        r16 ='e8ddn16_/
        dnc8 ba16_/
        ag8f g32a/
        g16 b32a b16 g32f g16 e32d/
        e8fg/
        abc/
        d#8 e16dcb/
        e8,er/
        r16'g8ec16_/
        c e32d e16ca#f_/
        f'f8db16_/
        b d32c d16 b g e_/
        e 'e8c a#16_/
        a# 'g8f e16_/
        e d8c b16/
        a#16 e32d e16 a32g a16 b32a/
        b16,dc'b,c'a#/
        b8 r16 bdf/
t60
        b8 r16 bfd/
        bfdb r8/
t50

$bot #fcgd

        =,e8fg/
        abc/
        d# e16dcb/
        e8,e r/
        r16 e8d dn16_/
        dn c8b a16_/
        a g8f g32a/
        g16 b32a b16 g32f g16 e32d/
        e8 'ce/
        fa#c/
        ,,d'bd/
        egb/
        ,,c'ce/
        fga#/
        b,eg/
        f16'ca#c,e'c/
        ,d8ef/
        ,b16 'b32a# b16 f32e f16 d32c/
        d16 f32e f16 d32c d16 b32a#/
        b4,b8/
```

(a)

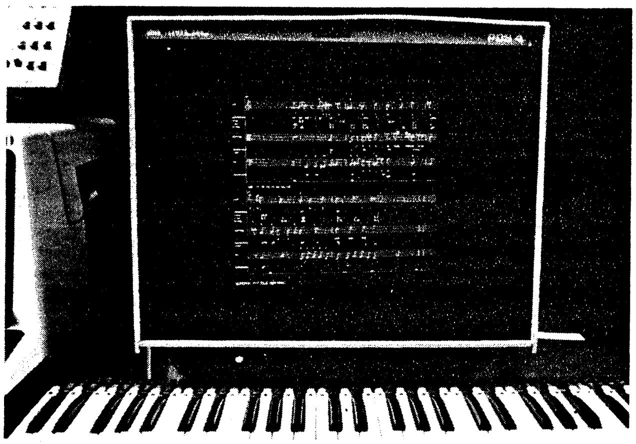
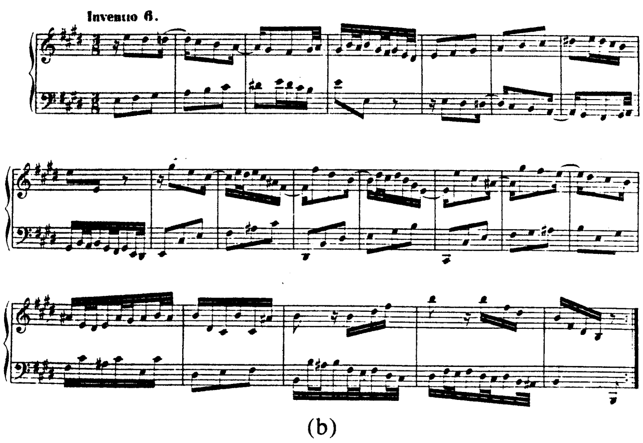


Fig. 1. Musical score editor: manipulating data on a screen in standard music notation.



(b)

Fig. 2. Alphanumeric encoding scheme: compact yet unnatural.

3 A FLEXIBLE AUDIO PROCESSING LANGUAGE

A mature software system such as Music 11 can be seen to have two main phases of operation, input preprocessing and run-time operation (Fig. 4). Input modalities may be enhanced by front-end graphics, or remain limited to the standard text-editing capabilities of the host system. Two text files must be created, an *orchestra* of signal-processing instruments and a *score* that invokes these instruments at specific times with parameters of frequency, loudness, and so on. Both files will be subject to preprocessing, and can therefore have a syntax convenient to the user. The orchestra language, for instance, will attempt to function in two ways. First it will act as a signal-processing language, invoking the time-domain audio operations (oscillators, filters, envelopes) and passing the signals between operations (audio patching). Second it will serve as a programming language, with threshhold detection and conditional branching (program flow of control). A good orchestra language will attempt to provide clear semantic implications on both counts. It is then the task of the orchestra translator (OTRAN in Music 11) to convey these details to the run-time system.

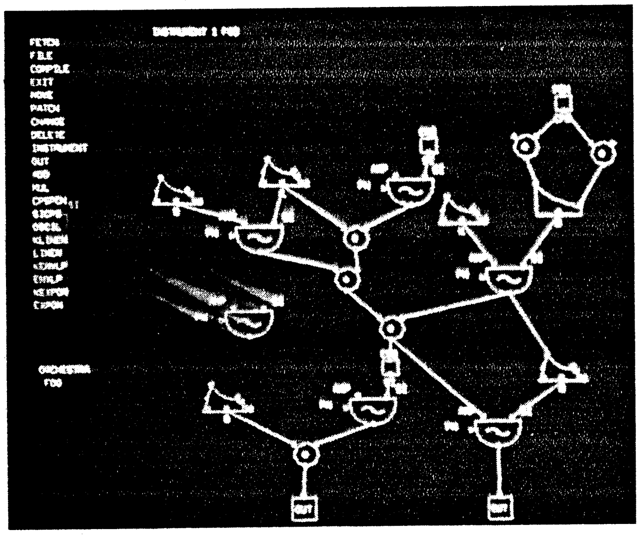


Fig. 3. OEDIT, an orchestra signal-processing network editor: moving a new oscillator into position.

The signal-processing primitives in an orchestra language should exhibit both breadth and depth. There should be a full complement of elementary operations (such as delay one sample), allowing the user to perform primitive signal-processing operations. There should also be numerous high-level modules (filters, envelopes) enabling the researcher to avoid the clutter of detail when appropriate. As an additional principle, the low-level primitives should enable the fabrication of any high-level module (for example, phasor plus table lookup makes an oscillator), with the result that there are no "black boxes" in the language. The second-order RESON filter in Music 11, for example, can alternatively be fabricated from existing language primitives (Fig. 5).

Following orchestra translation and massaging of the score file (for tempo warping), the two resultant files are directed to the orchestra loader and music monitor, respectively (Fig. 4). At run time the loader first places into main memory a description of each instrument in the form of a *transfer vector* (sequence of processing operations) and a *data-space descriptor*. No instruments actually exist at this time. Next the music monitor begins reading the score file: each time a new note is encountered, it requests the loader to provide an *instance* of the instrument (that is, to allocate and initialize some data space), and to mark that instance *active*. Signal

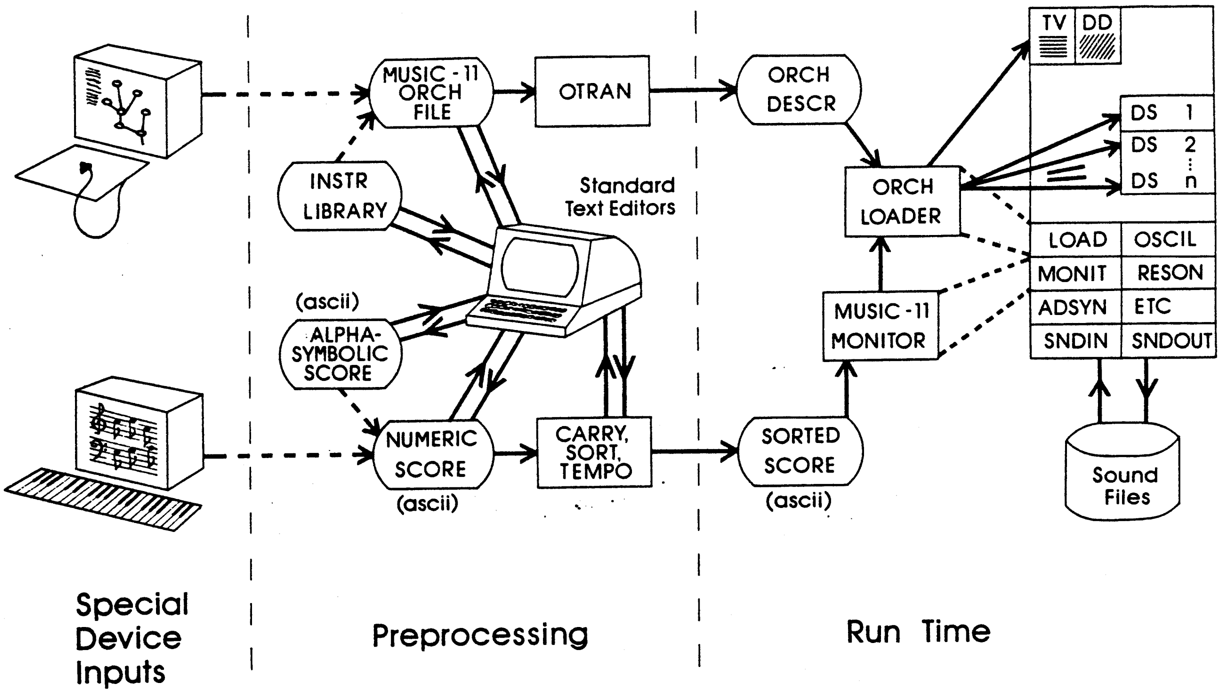


Fig. 4. Phases of Music 11: flexible signal processing on small machines.

processing and audio synthesis can now proceed, and at a fixed pace, typically at 40 or 50 kHz. All the while new event objects are created and old ones retired, in response to the event list found in the score file. The resulting stream of digital audio samples is stored on a large disk for audition at the end of the run.

The observant reader of the above account may well have noticed that there could exist more than one instance of a particular instrument at any one time. Multiple copies of an instrument are created simply by allocating multiple instances of the data space. The technique of using reentrant code for all signal-processing modules in Music 11 has the effect of allowing the score to call for any number or any combination of instruments without the orchestra requiring prior knowledge. The flexibility and the independence this gives to both orchestra and score are important to the creativity and investigative impulse of the composer or researcher. This condones flirting with the likes of seventy-six trombones at one moment, a thousand violins at the next. The instrument copies in Music 11 actually move around in data memory whenever the profile of active instruments is changed. This permits an efficient form of data-space management, allowing considerable signal-processing flexibility on a relatively small computer.

4 PROGRAMMING A RESEARCH PROBLEM

To see how Music 11 may be used for audio research, let us consider the problem of defining and creating acoustic ambience. Current literature indicates that we should pay particular attention to early reflections, since these determine echo density growth patterns and also provide listeners with perceptual cues about the space they believe they are in. To create this illusion, we must model at least the following:

1) The direct signal
2) Initial and continuing recursive wall reflections
3) Initial reflections from specific objects.

Current literature is helpful on the geometry of source imaging. It also documents the effects of wall absorption, air absorption, and so on, and we should use these data in our simulation.

We now approach the simulation problem by sampling the ambient space in several directions around the listener (Fig. 6). For each direction we create a model of

```
        instr   1               ; instrument with fabricated reson:
la1     init    0                       ;clear feedbacks
la2     init    0                       ;  at start only
i3      = exp(-6.283185 * p6 / 10000)           ; set coef 3
i2      = 4*i3*cos(6.283185 * p5/10000)/(1+i3)  ; set coef 2
i1      = (1-i3) * sqrt(1 - i2*i2/(4*i3))       ; set coef 1
a1      rand    p4                      ; source signal
la3     =       la2                     ; feedback 2
la2     =       la1                     ; feedback 1
la1     = i1*a1 + i2*la2 - i3*la3       ; 2nd order difference eqn
        out     la1                     ; output to chnl 1
        endin


        instr   2               ; this instr does same as above
a1      rand    p4                      ; source signal
a1      reson   a1,p5,p6,1              ; 2nd order recursive filter
        out     a1                      ; output to chnl 1
        endin
```

Fig. 5. Second-order recursive filtering, built from language primitives or invoked as a higher level module.

the important effects. The size of objects in the room and their acoustic shadows will result variously in low-pass and high-pass filtering effects. Signals traveling from one directional sample to another will incur time delays that depend on the distance between those points. The activity in each sample direction can be modeled by a pipe into which each simulated reflection is added at some distance from the output end. The idea of a pipe with arbitrary add-ins allows us to model the reflections uniquely for any number of source locations in the simulated space. The community of pipes, connected so as to send audio samples to one another, creates a signal-processing network that may be viewed as a matrix of comb filters. The feedback coefficients necessary to achieve a smooth and stable response can be chosen with the help of matrix algebra.

In the Music 11 representation (Fig. 7) PIPDEF defines a pipeline, PIPAD represents signal add-in, and PIPRD reads the signal as it spills out the end. PIPADV is a signal add-in whose distance is time-varying, in accordance with the slower control signal k_1. The parameter values, filter settings, and delay times are derived from actual modeling of a source and listener position.

A Music 11 ambience instrument such as this is not necessarily handwritten, but can be generated by a higher level program called ROOM. With its help we are able to specify the dimensions of a rectangular room, the positions of reflective objects inside the room, several different source locations, and a listener position. The source locations are excited with an acoustic signal, and the reverberant response of the stimulated room is then generated in four channels.

One aim of this particular model is to determine a

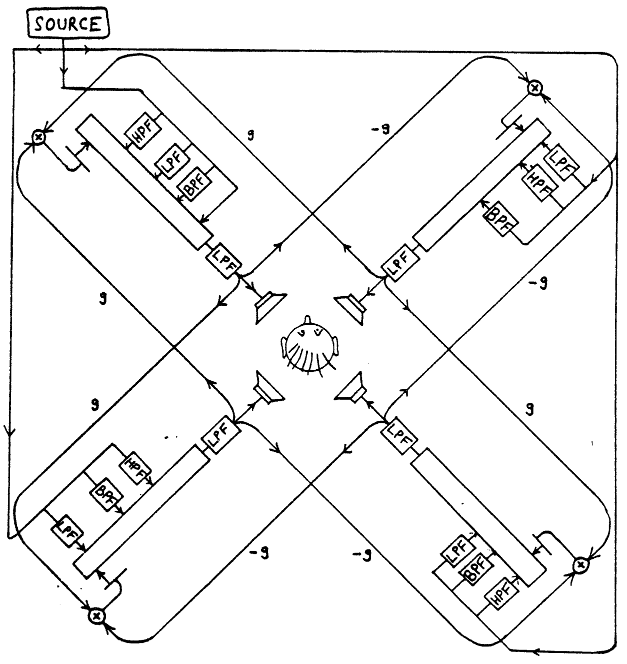


Fig. 6. Four-channel ambience simulator. Activity in each sample direction is modeled by a pipe, into which each simulated reflection is added at some distance from the output end. The community of pipes creates a network that functions as a matrix of comb filters.

suitable acoustic ambience generator for a new media theater, part of a new building devoted to arts and media technology currently under construction at M.I.T. There have been many attempts to gain control over acoustic listening environments. The *Espace de Projection* at IRCAM, Paris, actually uses moving wall panels to change the reverberant characteristics of the performance space. We expect that the advent of low-cost digital technology, coupled with methods as above, will allow us to find a completely digital solution to this intriguing problem.

5 FINDING THE RIGHT HARDWARE FOR AUDIO RESEARCH

Using Music 11 for research in an audio lab environment requires a medium-sized PDP-11. Although the software will run on any PDP-11, the cheaper LSI-11 versions are too slow for active research. A typical ambience simulation of 10 s of sound may take 5 min to compute on a PDP-11/50. All Music 11 computation is done in 32-bit floating point to simplify amplitude scaling problems and to encourage experimentation. Ambience simulation will tend to use all 16 bits of memory address space so that, for 16-bit computer systems, separate instruction and data space is advisable. For those who might contemplate a system today, a PDP-11/44 computer with 300 Mbyte of disk space, four channels of quality analog-to-digital and digital-to-analog conversion, and the Unix time-sharing system with Music 11 would provide the cleanest, fastest route to a ready-made research environment.

```
            instr 13                                ; AMBIENCE SIMULATOR

CHNL1:      pipdef  .069                            ; DELAY SPACE, CHNL 1

                                                    ; IF SOURCE INACTIVE,
    if gk1 = 0 kgoto  FEEDBACK1                     ;   SKIP SOURCE AND
                                                    ;   EARLY REFLECTIONS

            outq1   ga1/9.2                         ; SEND SOURCE/DISTANCE
                                                    ;   TO SPEAKER 1

    a1      reson   ga1*.024, 214, 200, 1           ; EARLY REFLECTIONS:
            pipad   a1, .0017                       ;   DIRECT MODELING OF
                                                    ;   ROOM RESPONSE,
    a1      reson   ga1*.027, 435, 278, 1
            pipad   a1, .0097
                                                    ;   INCLUDING WALLS
    a1      atone   ga1*.03, 600                    ;   AND OBJECTS
            pipad   a1, .0162

    a1      tone    ga1*.043, 9800
            pipad   a1, .026

FEEDBACK1:                                          ; LATE RESPONSE:
    k1      randi   .001, 3.1, .06                  ;   RECURSIVE MODELING
            pipadv  .7*(la2 + la3), .068+k1         ;   OF AMBIENCE WITH
                                                    ;   RANDOM FLUCTUATION
    a1      piprd                                   ; READ PIPE 1
            outq1   a1                              ;   SEND TO SPEAKER 1


CHNL2:      .
CHNL3:      .                                       ; DITTO CHANNELS 2,3,4
CHNL4:      .


                                                    ; AIR ABSORPTION:
    la1     tone    a1, 14000                       ;   LOWPASS FILTER
    la2     tone    a2, 13000                       ;   ALL SIGS PRIOR TO
            .                                       ;   NEXT FEEDBACK
            .

            endin
```

Fig. 7. Music 11 code for the ambience simulator. Parameter values, filter settings, and delay times are derived from actual modeling of a source and listener position.

At some later date, emerging new hardware will favor other solutions, including real-time processing. Since music lends itself to processing in arrays (as is done in software in both Music 5 and Music 11), it is possible to implement real-time synthesis on standard array processor hardware. In 1979 the M.I.T. studio acquired a small but very fast array processor, an Analogic AP400. This is a 24-bit fixed-point machine with a computation rate approaching 10 million multiply-adds per second. Because it was designed for fast-Fourier-transform work, its potential for time-domain processing might at first seem only marginal, but we have managed to implement a substantial portion of Music 11 on this device. It can now be invoked to do such things as sound-test a Music 11 instrument in real time with parameters controlled by slider pots, or add reverberation to a sound file in real time during digital-to-analog conversion. The recently announced AP500, with floating point and flexible control processing, appears even better suited. The combination of a microprocessor-based personal computer with an array processor of this sort would provide a flexible, cost-effective tool for personal real-time audio research. We are beginning to use such a system at M.I.T.

Recent advances in the art of connecting custom microprocessors suggest other possible audio processing structures for the future. Concepts such as the butterfly switch arrangement, recently developed by Bolt Beranek and Newman for manipulating voice data channels on the ARPANET, have reduced multiprocessor interconnection problems to manageable proportions. If packet communication of audio signals can provide the generality required of a good music processor, then such an arrangement could prove very useful to future real-time audio processing.

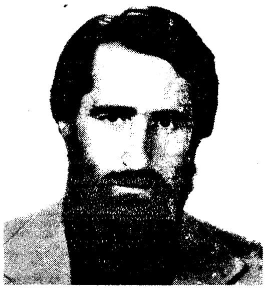
Other alternatives include custom very-large-scale integration, tailored to the specific needs of quality audio processing and exploiting massive computational parallelism. At least one research program is currently under way to determine the kind of multilevel control structure that would be necessary to coordinate and manage such parallelism in acoustically significant and musically meaningful ways.

6 FUTURE NEW INSTRUMENTS

Music processors of the future will likely be responsive to human control in a variety of gestural and acoustic input modalities. There will be whole classes of new instruments, some for studio use, some for stage use, and others for the home market. We could even imagine a new amateur music-making activity, not dependent on the acquired dexterity of playing a two-keyboard 64-button home organ, yet more artistically demanding than merely selecting an LP from a shelf. The user would "orchestrate" an existing score by selecting and assembling the timbral and spatial attributes for a personalized home-audio performance.

Participating in a musical performance with such high-level decision making would be quite desirable to amateur music lovers. For this activity to result in sounds that are as aesthetically effective as those of commercial productions would require that the large computational loads (about 200 million multiply-adds per second) be managed at prices that appeal to the consumer market. Representation of current signal-processing methods in very-large-scale integration will of course hasten that day. Equally effective will be the discovery of new signal-processing methods that concentrate on the perceptually important aspects of music synthesis. That area of research is receiving much attention right now, and the skills are moving almost as rapidly as the supporting technology. To those of us in the field of computer audio research and computer music composition, the potential of digital audio seems very large indeed.

THE AUTHOR



Barry Vercoe was born in New Zealand and gained degrees in music and in mathematics from the University of Auckland. Active as a choral conductor and composer, he came to the U.S. in 1962 to study composition with Ross Lee Finney at the University of Michigan, where he completed a doctorate.

After teaching at Oberlin Conservatory, he was Ford/CMP composer-in-residence in Seattle-Tacoma during 1967–68, and went on to do research at Princeton University in the area of digital audio synthesis. After teaching briefly at Yale University, he joined the faculty at M.I.T., where he founded the Experimental Music Studio in 1973.

Dr. Vercoe is currently Associate Professor of Music and Technology at M.I.T., and an associate member of the Laboratory for Computer Science. His Music 360 and Music 11 digital sound synthesis languages are in worldwide use. He has received awards from the National Endowment for the Arts, the Massachusetts Council on the Arts and Humanities, and the Guggenheim Foundation, and has served on the National Executive Committee of the American Society of University Composers. In 1976 he organized the First International Conference on Computer Music, in conjunction with the ISCM World Music Days festival in Boston, and has served as U.S. advisor to the UNESCO Joint European Studies Commission on Technology in the Arts.