Something Digital

Miller Puckette
l'Institut de Recherche et Coordination Musique/Acoustique
31 Rue St. Merri, 75004 Paris, France
miller.puckette@ircam.fr

1 Can computers make music?

Suppose we buy a computer with a big disk, a modern programming environment, an audio-quality DAC, or two, or eight, and amplifiers and speakers. In theory we can now make any sound that our speakers can emit (I owe this observation to Richard Moore). We could simply type in a few million samples if nothing easier comes to mind. One major goal of computer music research has been to make this process practical, that is, to find good specification languages for sound. This quest started at Bell Laboratories with the first experiments of Max Mathews and John Pierce, and gave rise to the dozens of "MUSIC" programs by a dozen authors. Later, there were also languages and editors for manipulating scores, sound editing scripts, spectra, "control signals", waveforms, "patches" (as in a modular analog synthesizer), and so on.

But computer music should be something more than the art of shaking a speaker cone in a pre-specified way. A piece of music could, in principle, be reduced to a succession of excitations of the modes of vibration of a listening room, but can we make music from this point of view? Only in the very limited sense in which "tape music" can be considered music. An essential part of "real" music is the live element, the undefinable but undeniable interaction between players and audience which makes music exciting. It is hard to prove this interaction is there (and our psychoacousticians still lack any way to measure it) but any musician will swear to its reality, and to its importance to the music-making process. It exists if only because musicians think it exists; that's enough if it changes the way they play music.

To make live music with the hypothetical Computer Music Workstation of the first paragraph we must make it work in real time: connect one or more gestural input devices to it, compute each sample only slightly in advance of the time it is needed for conversion by the DACs, and make the sample computation dependent in some musically useful way on what has come in recently from the live input. The closer the rapport between the live input and what comes out of the speaker, the longer the audience will stay awake. This rapport is the crux of live computer music. A part of the quest for a better Computer Music Workstation is to make it easy to establish real-time musical control over the computer.

I am against trying to set the computer up as a musical performer. Large software systems which try to instill "musical intelligence" in the computer, while interesting as research projects, are not likely to be musically useful. (An exception might be found in common practice music notation editing, where informed guesses by the computer might save key strokes or mouse clicks.) No concert audience wants to watch a robot play the piano, even if it seems to exhibit some kind of musical "feel." An automatic music performance makes a nice curiosity but nothing more.

The computer is better used as an instrument. A unique one, to be sure; no violin has a programmable user interface. The computer-instrument widens the possibilities of musical expression—human musical expression—in ways which we are only beginning to explore.

2 IRCAM and the Search

At IRCAM, where I work, the development of Computer Music Workstations has occupied researchers for the thirteen years of IRCAM's existence. Tradition has it that Luciano Berio asked Giuseppe ("Peppino") Di Giugno (they met on a train) to build a large analog synthesizer, estimating that a bank of 1000 analog oscillators would be enough to make rich and varied sounds. Peppino, who was originally an atomic physicist, tried this in a thought experiment first. The 1000 analog oscillators would take, perhaps, 100 hours of knob-tweaking to make a single sound. Similar problems of scale would apply to maintenance (250 hours a week, say), and cooling.

Peppino decided that we needed Something Digital. Over the years, he designed and built the 4A, 4C, 4X, and 5A. The 4A was a rather special-purpose oscillator box; the 4C introduced hardware envelope generators and could do a bit of filtering; the 4X was a collection of eight microcodable bit-slice processors; the 5A (of which one prototype was built) introduced floating-point arithmetic and, marvel of modern technology, a jump instruction. We still use the 4X constantly in music productions. IRCAM has three or four, depending on how one counts them, and I have been ceaselessly complaining about them for the five years I have programmed them, lacking anything newer to complain about.

Although the original idea was to build an oscillator bank, by 1985 most of the composers who tried to use the 4X were interested in "signal processing", using the term to mean transforming the sound of a live instrument in some way. This change of focus was the product both of opportunity and of necessity: opportunity since "signal processing" is capable of a richer sonic result than pure synthesis, and since it is easier to create musical connections between a live player and electronics if the electronics are acting on the live sound itself. Necessity, since it was clear that after eight years of refining the digital oscillator, we lacked the software to specify interesting real-time timbral control at the level of detail needed. Signal processing, by contrast, can often yield interesting results from only a small number of control parameters.

The main real-time control problem in live signal processing is to synchronize the live player and the processor. The synchronization must sometimes be accurate to as few as 30 milliseconds, that being the smallest time scale on which an instrument normally articulates sound. For example, the attack of a flute note often lasts only 100 milliseconds; it is frequently necessary to specify which portion of the attack to send to some processing element.

The synchronization problem divides into two components: finding the pitch of the live instrument, and "score following", the art of deducing from the stream of detected notes where the player is in the score and responding appropriately. The first problem is straightforward (but not always cheap) to solve for keyboard instruments and fairly hard for other instruments. In 1984 Lawrence Beauregard built an "instrumented flute". This is a standard concert flute with switches added to the keys so that a computer can detect the fingering. Knowing the fingering reduces the number of pitches the flute might reasonably be playing to three or four. An acoustic pitch detector can use this information to increase its speed and reliability to the point of being usable on stage.

The problem of score following was first solved independently by Vercoe [1] and Dannenburg [2]; Vercoe worked specifically with the Beauregard flute and the 4X. The complicated part of the job is not to find a good matching between the two discrete sets of notes (the live input and the player's part) but to time an accompaniment which stays close to the instrument player without resorting to extreme speedups and slowdowns, or alternatively, to design an accompaniment which can take in or let out slack on short notice because of changes in tempo on the part of the live player.

The first use in a real musical performance of the 4X, the Beauregard flute, and score following occurred in a 1987 concert, in which "Jupiter" by Philippe Manoury and "Aloni" by Thierry Lancino were played. The two pieces had taken roughly three years to realize. This was the epoch in which I learned to curse in French.

The 4X was not completely tamed until 1988. We bought Apple MacIntosh computers for the 4X studios, so that instead of writing real-time control programs to run on the 4X system itself, we can set the 4X up as a MIDI device—typically, a patch looks like 100-500 MIDI controllers—and do all the hard programming in Lightspeed C on the MacIntosh. This is how the MAX program was born [3], which has turned out to be useful in many contexts besides that of the 4X.

I think we can draw a lesson from the 4X experience. The 4X design was guided by low-level thinking about such issues as data bandwidths for driving breakpoint envelopes. A serious software effort aimed at real-time control problems did not start in earnest until 1984, although the 4 series had by then been under development for several years. It was largely the lack of control software that made early 4X music productions so hard.

We also paid dearly for the decision (which was, of course, unavoidable at the time) to build the 4X from discrete components; this meant that we had to write a compiler for the 4X proper and an operating system for the "real time control" portion of the system. In consequence, the 4X development environment was, and is, Neanderthal.

IRCAM is still in the business of designing what can be called "Computer Music Workstations"; see [4] for details about the latest project, which is led by Eric Lindemann. In short, we're building a machine twice as powerful as the 4X for a fifth the price, based on a modern computer (the NeXT), and combining the control and signal processing work in a single off-the-shelf processor (the INTEL i860.)

3 Trends in live electronic music

Music production at IRCAM—yes, I'll say it—represents only one of many possible outlooks on live electronic music. Other people and institutions are working in many different directions. Here is a sampler:

At Berkeley, David Wessel, Adrian Freed, and Keith Gordon [5] have been working to develop a unified software interface between the MacIntosh computer and external DSP hardware. The most interesting aspect of this work (to me at least) is the wide variety of user-level programs which can access the DSP through this interface (in principle, any application which can load and run external object code). For example, a modular real-time control program (MAX springs to my mind) could now be given a collection of "DSP objects" which correspond to real-time DSP code to run: oscillators, filters, and so on. These DSP objects would also be seamlessly integrated into the control environment—no more brick wall between "synthesis" and "control".

Max Mathews at Stanford is working on the "Radio Drum", a new input device designed by R. Boie which continues the direction taken by Mathews's earlier "Sequential Drum". The Boie drum not only can tell where and how hard it has been hit, but where its two drumsticks are in the air (at least when they are fairly close to the drum.) Mathews and Andy Schloss [6] have been experimenting with a wide variety of different ways to use this information to control live music. I'll propose one more: since the location of a drumstick is known before it whacks the drum, drum hits could be predicted slightly before they occur. One could use the drum to make sounds whose "perceptual onsets" come later than their "real onsets". Spoken or sung speech is a good candidate: in the word "drum", for example, the heard beat falls on the vowel, but you have to utter the first two consonants first. One could, for example, preload the phonemes and pitches of a sung phrase, and sequence through them with the drum controlling timing and stress.

Researchers and engineers at STEIM in Amsterdam have been taking a unified approach to physical controller design, centered on an extremely flexible voltage-to-MIDI convertor they designed. The physical devices have used ultrasonic range sensors, stress sensors, photocells, Hall effect switches, and what not. A new and interesting example, conceived by Michel Waisvitz, is called the "web": four strings are strung across a circular frame, meeting at its center. The tensions of the strings at the eight ends are measured and converted to MIDI. To play it, you reach in with both hands and knead the surface that the strings define. The result seems to have a much higher "interesting bandwidth" (whatever that means) than a bank of eight sliders would; the interface has already proved useful for certain kinds of timbral control, and for parametrizing note generation algorithms.

At MIT, Barry Vercoe, with whom I studied, has done something brilliant: nothing. You can run essentially the same Music11 orchestra and score in 1990 that ran in 1980 or even 1976 (on a PDP 11/45—remember them?); a piece that ran 60:1 then (i.e. an hour of computation for a minute of sound) can run in real time now on a UNIX workstation. After fourteen years of keeping a music language alive, the result is exceedingly portable; I think it's a good bet that Csound (as it's now called) will remain alive and well for many more years.

Luciano Berio has started Tempo Reale, a computer music studio in Florence, where Peter Otto and Nicola Bernardini (and Audiomatica's Maurizio Cavalli) have brought the TRAILS project to life: a matrix of up to 512 VCAs for doing live sound localization. Along the way, they decided they needed a serious MIDI slider box; the result, now sold commercially, is usable for tasks such as mixer automation. They are now attacking the much harder problem of using it to control TRAILS in real time. This work sets the current record, I think, for attainable live control bandwidth.

Trevor Wishart [7] has finally convinced me that the phase vocoder can give rise to musical results which would be hard or impossible to come by otherwise. His "Vox 5," a tape piece realized at IRCAM, constitutes an undeniable musical proof of the technique. (I'm not pleased to have to admit this. In addition to being somewhat expensive computationally, the phase vocoder is conceptually ugly.) Like it or not, we realizers of Computer Music Workstations are going to have to think about whether our real-time control structures are capable of steering this whale.

Jean-Claude Risset [8] also has made a wonderful piece of music, entitled "Duet for One Pianist." A live pianist plays on a MIDI-equipped piano, which sends the notes to a computer. The computer carries out one of several different transformations on the notes, each rather simple, and sends the transformed notes back to the piano. The result is a quirky and entertaining dialog. This piece loses more than most in a recording: there is simply no comparison between listening to a tape of it and seeing it live. On the tape it is frequently hard to guess what is human stimulus and what is computer response, since the notes sound the same—the only difference is how they look.

One trend I see in the above examples is that the computer occupies less and less of the center stage, and is becoming more and more a tool like any other in the musician's or instrument designer's toolbox. Another trend: we are constantly creating more and more horrendous control bandwidth problems. In comparison, our number crunching needs for sound synthesis or signal processing are increasing quite slowly. But the most interesting trend (to me) is the diversity of fruitful uses people are finding for computers in music making. I will be happy to see that continue.

4 The computer music tradition

The Library of Congress, which archives early sound recordings (among other things), reportedly keeps up some exotic equipment for playing them back. Before tapes existed, for example, there were machines which made magnetic recordings on spools of steel wire, and of course we are now stuck with a collection of old and precious tangles of wire. So the Library maintains a machine which plays wire recordings.

Now consider a live computer music piece for a MacIntosh computer and a Yamaha DX7 synthesizer. Fifty or a hundred years from now, will it be practical to design and build a system that can read a 1990 flexible disk and emulate the MacIntosh and the DX7? This is a great deal more complicated than measuring the magnetic field around a steel wire. The MacIntosh and the DX7 use proprietary software, firmware and hardware; data paths, especially in the DX7, have different formats and sizes; "system exclusive" MIDI messages may be used; the MacIntosh software may require "inits" or "device drivers" to be installed; documentation for the various components of the system is widely scattered and some is secret.

As a related question, suppose that a performer wishes to play a computer music piece from some other performer's repertoire; or suppose that the second performer has invented some musical gesture that the first one would like to borrow, as one jazz player might borrow a "lick" from another. The problem of compatibility comes up again; the two performers are unlikely to have the same equipment. How, then, can we move toward having an established repertoire of computer music, like the repertoire of classical music from which performers now draw? And how can this repertoire outlive the hardware of our day?

A short-term, and partial, answer could be to find and agree on a FORTRAN for computer music: a language which could serve as a common ground, and which future systems designers would feel obliged to support. At a minimum, this Computer Music FORTRAN (CMF?) should be able to describe real-time control structures as well as synthesis algorithms. But like FORTRAN, we can guess that CMF would be disparaged by future software designers; languages can rarely be kept up-to-date and backward compatible at once. In any event, it is hard to imagine any standard far-reaching enough to permit everything a computer musician might want to do. And even if it did, we would still have a wide range of different input devices, whose differences could not be hidden from the software without losing the special capabilities of each one. We need to be able to share (or steal) ideas on a level higher than software, by watching, listening to, and imitating each other.

To make this possible, there must be a direct and comprehensible relationship between the controls we use and the sounds we hear. (This would not be a bad thing from the audience's point of view either.) A performer who pushes a button to start a sequence is not showing us how the music was really made; all we learn about the music is what our ears can tell us. But if the performer's actions correspond more closely to the sounds themselves, then we can see something about the music's gestural content, and our own music can be better informed by it. In this way we could evolve a stronger and more meaningful tradition for making computer music. Moreover, the more directly a piece of computer music follows from its performance, the more easily that piece could be adopted into the repertoire of some other performer. The new performer might even be free to personalize the sounds themselves, as long as their interrelationships remained true to the written piece. In the limit, a computer music piece could even be written on paper; it would be entirely up to the performer to bring it to life.

The computer music workstation should therefore depend less on special-purpose synthesis algorithms, less on the behavior of some 11-bit logarithmic envelope generator found in the Bozotronics BZ23A90, and more on straightforward, what-you-play-is-what-you-hear synthesis and control structures, even if these appear to be more expensive. They are more likely to give rise to a sense of current musical practice which can reach across incompatible machine boundaries, or even across years.

5 How to design and build a Computer Music Workstation

We can easily specify the hardware platform needed for making real-time computer music. First, we should be able to do at least 1000 oscillators (to use Berio's units) which implies, roughly, a minimum of 500 million attainable arithmetic operations. Of course, if we can have more than that nobody will complain, and some algorithms which have been proposed actually require more, so this figure is only a minimum. There is some disagreement about the necessary disk performance; I think the minimum we could ask for is 8 channels of sound to or from disk. I don't think it's proper to insist that a Computer Music Workstation also function as a multitrack tape machine, since that function has little to do, or should have little to do, with making the music itself. The problem of conversion between analog and digital sound formats, which used to dominate discussions of this sort, is now minor.

In addition to needing raw hardware power we need generality. Ideally, everything would happen on a single processor type (perhaps many individual processors) running a single operating system. In the 4X system there are three separate environments: one for development, one for "real-time control" and one for sample crunching. In IRCAM's current project, there are two: UNIX (Mach, really) and real-time. In some future project, we can hope that an operating system which supports such niceties as graphics and a networked file system can also do all the real-time work.

And of course we need software. It is unlikely that some knight will come up with a comprehensive Computer Music Software Package. But it does seem conceivable that we can define a standard for inter-object communication that would allow many different software packages to refer, in real time of course, to objects defined by each other. Good communication standards, especially real-time ones, would allow individual software packages to stay manageably small and portable.

What hopes does the near future of the technology raise? The 500 million operations per second we need should be available Real Soon Now for about $80000 (that would buy a 500-attainable-megaflop configuration of the IRCAM Musical Workstation; the smallest configuration is expected to cost $20000). This is still too high; a reasonable entry cost might be $3000. We can hope for prices to drop to that point sometime between 2000 and 2005. Until then, there is little hope that profit-making organizations will bring out a 4X-like machine, so the task will still fall to government agencies such as IRCAM. (It follows that Computer Music's next decade will be led by Europe.)

In yet another decade, or perhaps two, the Computer Music Workstation will be nothing more than a computer; general purpose home computers will provide all the power we need. Of course, we can confidently suppose that new research will give rise to ever newer and more unattainable computation requirements. But an important milestone will have been reached: a computer that musicians can afford will be capable of acting as a musical instrument.

References

1. Vercoe, B. 1984. "The Synthetic Performer in the Context of Live Musical Performance", Proceedings, International Computer Music Conference (Paris, France), P. 185. San Francisco: The Computer Music Association.

2. Dannenburg, R. 1984. "An On-line Algorithm for Real-Time Accompaniment", Proceedings, ICMC (Paris, France), P. 187.

3. Puckette, M. 1986. “The Patcher”, Proceedings, ICMC (Cologne, Germany), P. 420.

4. Lindemann, E., Starkier, M., and Dechelle, F. 1990. “The IRCAM Musical Workstation: Hardware Overview and Signal Processing Features”, Proceedings, ICMC (Glasgow, Scotland), P. 132.

5. Freed, A., and Gordon, K. 1990. “DSP Driver Software for Performance-oriented Music Synthesis Systems”, Proceedings, ICMC (Glasgow, Scotland), P. 79.

6. Schloss, A. 1990. “Recent Advances in the Coupling of the Language MAX with the Mathews Boie Radio Drum”, Proceedings, ICMC (Glasgow, Scotland), P. 398.

7. Wishart, T. 1988. “The Composition of Vox-5”, Computer Music Journal 12(4), P. 21. Cambridge, Massachusetts: The MIT Press.

8. Risset, J.-C. 1990. “From Piano to Computer to Piano”, Proceedings, ICMC (Glasgow, Scotland), P. 398.