Recent Research in Computer Music at CCRMA

Julius Smith, John Gordon, David Jaffe, Bernard Mont-Reynaud,
Andrew Schloss, Bill Schottstaedt, and Paul Wieneke

Dept. of Music, Stanford University

February 1982

Abstract — We describe some recent research at CCRMA in the areas of real-time signal processing hardware, new techniques for modeling signals, automatic music transcription, and software environments for the composition of music.

Introduction

The Center for Computer Research in Music and Acoustics (CCRMA) is a music lab where art and science join forces for the purpose of advancing the possibilities in music. There is ongoing work in new music, psychoacoustics, musical acoustics, applied artificial intelligence, signal processing, hardware architectures, and digital recording-studio technology.

This paper is intended as a quick introduction to computer music, CCRMA, and the research of the authors. We first review the principal sound synthesis techniques in use today by computer music composers. Next we describe briefly the facility at CCRMA. Finally we present a glimpse of some of the research in progress.

Synthesis Techniques

Sound *synthesis* techniques are classically divided into the following categories:

1) Additive
2) Subtractive
3) Nonlinear

Additive synthesis is based on the famous theorem of Fourier which states that any reasonably smooth bounded function can be expressed as a sum of sinusoids. For this technique, a large number of sinusoidal oscillators is needed, and their amplitudes and frequencies must be controllable over time to allow time-varying spectra. For example, with a sampling rate of 32KHz and a lowest pitch frequency of 20Hz, we need nearly $16000/20 = 800$ oscillators to synthesize an arbitrary periodic waveform to the full available bandwidth. Fortunately, most natural sounds have spectra which diminish rapidly at high frequencies so that the first 20 harmonics (or less) give good results in most situations. Additive synthesis has the advantage of generality but the disadvantage of complexity in both computation and storage. Also, additive synthesis does not easily produce noise-like signals, and it is difficult to add noise in a way which fuses perceptually.

Subtractive synthesis denotes starting with a rich spectrum and removing unwanted spectral components, much as in sculpture. For periodic signals, a pulse train is often used as the input to a digital filter. A periodic pulse train has harmonics of equal amplitude. Thus the digital filter provides all spectral shaping by emphasizing or suppressing some harmonics with respect to others. For noise-like signals, the starting point is often "white noise" such as simulated by a uniform pseudo-random number generator. Regardless of the noise distribution (uniform, Gaussian, Poisson, etc.) the power spectral density of the noise will be flat as long as the samples are uncorrelated. A digital filter may then be used to obtain the desired spectral distribution. Subtractive synthesis is also quite general and especially suited for physically modeling resonant systems such as in speech, reverberation, and orchestral instrumentation. A disadvantage of the subtractive

approach is that digital filters are difficult to manage for most musicians. Also, most implementations use recursive filters in fixed-point arithmetic, and such systems are prone to scaling problems and overflow, especially when using high-order filters.

Nonlinear synthesis refers to methods which compute sound samples by means of some formula which is generally not a linear operation. (Note that Fourier transforms and typical digital filters are linear operators). By far the most prevalent nonlinear synthesis technique, and perhaps the most pervasive technique of any variety, is *FM synthesis*.[3] FM synthesis was originally developed by Prof. John Chowning, director of CCRMA. It works by adding a "modulating" sinusoid to the frequency of a "carrier" sinusoid, exactly as is done for FM radio broadcasts. It differs from FM radio in that the carrier frequency is in the audio band, and the modulating signal is typically a simple sinusoid rather than a complex audio signal. The FM side-bands about the carrier frequency are used directly as partials or harmonics of the sound. Another nonlinear synthesis technique, also developed at CCRMA, is Marc LeBrun's *digital waveshaping* synthesis. It is analogous to an FM technique in which a general waveshape replaces the carrier sinusoid.

On balance, FM synthesis is still the most widely used synthesis technique. This is due to the extreme simplicity in the hardware resources necessary to produce a harmonically rich spectrum. Furthermore, it is simple to adjust gross bandwidth of the spectrum by means of the "FM index". With the use of multiple carriers, it is possible to simulate "formants" (localized regions of emphasis in the spectrum) as in subtractive synthesis. Adding another modulating sinusoid or two allows greater flexibility in shaping the spectral side-bands. The main problem with FM is that specific desired spectra are difficult to obtain, and even when it is theoretically possible to match a given spectrum, no automatic method seems to exist for finding the necessary parameters. Nevertheless, in the hands of a musician with a good ear and considerable patience, FM and its various extensions have consistently provided surprisingly good renditions of many natural sounds.[3,12]

CCRMA Facilities

The facility at CCRMA includes a Foonly F2 computer (emulating a DEC PDP-10), a 16 channel Grinnel display system, 12 high resolution video terminals, a digital music synthesizer/processor called the "Samson Box", a special auxiliary processor called the "Poly", four 16-bit D/A converters (the "DAC"), two 14-bit A/D converters (the "ADC"), and several listening stations (amplifiers, speakers, and conventional recording studio gear). In addition there are the usual complements for a large time-sharing system such as three disk drives (1200 Mbytes total), magtape drive, line-printer, Versatec, and so on.

The Samson Box was designed by Peter Samson and built by Systems Concepts in Berkeley.[11] It is an all-digital synthesizer with provisions for processing digitized data from the outside world as well as synthesis. We mention only its most salient features: Up to 256 basic signals can be generated each with individually "ramped" amplitude and frequency envelopes; the waveforms can be configured for FM modulation or summed into any of 64 "sum-memory" locations. There are 128 "modifiers" used for mixing, filtering, noise generation, and other more specialized functions. As digital filters, they can provide up to 256 poles and/or zeros. There are 32 delay units for reverberation, and we have 48K words of delay memory which can be arbitrarily partitioned among the delay units. Thus the three major synthesis categories plus facilities for sound modification are well supported in the Samson box.

Recent Progress at CCRMA

We now present a few of the current research topics under way. Although most of what we describe has not yet been published, the future issues of the Computer Music Journal will soon provide references.

The Polycephalous Signal Processor

The Poly was designed by James A. Moorer, with extensions by John Gordon. The original need was primarily for a Foonly F2 interface for the DAC and ADC. The requirements of the interface include:

- Memory buffer control
- Packing and unpacking samples into (or from) 36-bit words
- Controlling the sequencing operation of the two peripherals

However, since it was necessary for us to design and build the device in-house, we decided our needs would be better suited if we expanded its capabilities to those of a microprogrammable multiprocessor, including the ability for real-time control.

The Poly allows up to eight independent programs, executed sequentially, to provide a pseudo-parallel processing environment. A program can be as long or short as desired, but should be thought of as a loop. The loop cycles once each time the program is called, and then control is given to the next program. A single run through all eight programs is called a pass. The intent behind the design was to have one pass correspond to the processing of one sample of a digital signal (or one sample for each of several channels), although this isn't required. At the end of each pass, the Poly checks to see if any instructions are waiting to be executed from the F2. If so, it executes them then. This interpolation of instructions into the ongoing stream is one of the ways real-time control can be achieved.

The hardware is all on one board using TTL technology, although the board is a large one — roughly 350 IC's. The architecture can be viewed as a circulating data bus, 36 bits wide, with modules receiving inputs from the bus and/or sending outputs to the bus. The modules consist of a rotator, an ALU, a 16-by-16 bit signed multiplier, and a 256-word scratchpad memory. There is a 64-word FIFO for buffering samples to the DAC, and another to buffer input from the ADC. There is logic for requesting DMA cycles from the F2's memory, and several ways to interrupt the F2. There are also 2 registers which can be used for general communication purposes with the F2. A single instruction, which is 36 bits, specifies a driver of the bus, one or more receivers (latches), a scratchpad memory address, and various operations including several conditional jumps. Instruction cycle time is 100 ns, and instruction memory size is 2K.

The main application for the Poly is as a real-time performing or studio tool in connection with our converters. Real-time mixing of several digital signals while they are being sent to the DAC is very straightforward—as is simple processing of recorded signals coming from the ADC. However, because of the generalized nature of its architecture, the Poly can be used to implement many signal processing algorithms, such as digital filters and spectrum analysis techniques, in real time.

New Methods in Subtractive Synthesis

For his doctoral thesis in electrical engineering, supported by the Hertz Foundation, Julius Smith has been working on signal modeling by means of rational digital filters driven by simply described excitations plus noise. This work may be placed under the heading of "system identification". In the noiseless case, the goal is simply to find the digital filter which optimally transforms a given input signal into a given output signal. This formulation is ideal for music synthesis applications where digital filters are available and the set of basic synthesis waveforms is limited.

System identification is discussed principally in the literature on automatic control, and to some extent in the signal processing literature. In speech processing, much use is made of *linear prediction* techniques, and these are perhaps more widely known. One may view the system identification paradigm as a generalization of the standard linear prediction framework in three ways:

- Filters designed include *zeros* as well as poles.
- A known component of the driving input signal can be specified rather than assuming white noise or a pulse train.
- The unknown component of the driving input signal can be assumed to obey certain statistics. Alternatively, these statistics can be estimated.

Within this general setting, progress has also been made in tuning the *error criterion* of the identification algorithms to the special needs of audio signal processing. The current tableau of error criteria in system identification is actually quite limited and often even inappropriate for audio work. For example, all the prevalent identification methods minimize some type of squared prediction error. However, a look into audio equipment specifications suggests that users prefer a minimization of the *worst case error* in the frequency domain. Furthermore, this error is usually specified in dB. When the spectral match applies to a short time window, the phase of the spectral match is unimportant. Thus for audio spectral matching, it is preferable

to minimize the *Chebyshev norm of the log magnitude spectral error*. Unfortunately, the estimation problem becomes highly nonlinear in this circumstance. However, by approximating the logarithm appropriately, it is possible to obtain a method which is theoretically guaranteed to converge to a best approximation (a feature which is typically crucial in practice). Such a method has been developed based on the Remez multiple exchange algorithm. Further refinements include approximation of an appropriately smoothed spectrum over an exponentially spaced frequency grid (corresponding to the resolution of the ear), and linear weighting according to "equal loudness contours" in the frequency domain.

David Jaffe has been working on the modeling of plucked string tones based on a subtractive synthesis algorithm devised by Kevin Karplus and Alex Strong.[5] The algorithm consists of repeatedly filtering (using a simple two-point average) a wave table which has been preloaded with white noise. It is equivalent to the ringing of a high-order all-pole digital filter which has been excited with a noise burst. The tone produced is remarkably suggestive of a plucked string. It has been said to sound like a (very large) harpsichord or an auto-harp in certain situations.

Although the algorithm is strikingly good in its canonical form, certain enhancements are desirable for maximum musical expressivity. Partial control of "dynamic level" has been obtained with a time-varying one-pole lowpass filter which controls gross bandwidth. A natural "mute" or string-damping parameter has been obtained which uniformly accelerates the decay of all partials. These modifications can produce diverse yet natural changes in the overall timbre of the sound. Precise tuning of the fundamental frequency (quantized by the wave-table length) has been made possible by the addition of a filter whose phase-delay adds a fraction of a sample-period to the effective period of the synthesized waveform. Jont Allen has suggested that by using more elaborate phase-delay functions, it is possible to simulate stiff strings. Simulations of body resonance and sympathetic string vibration have improved considerably the realism and musical usefulness of the instrument. As an example, the effect of sympathetically vibrating strings was created using multiple copies of the string instrument tuned to different pitches. Then instead of a noise burst, a small amount of the output of the "played" string is fed into each of the "open" strings. The effect is a kind of reverberation such as occurs in real stringed instruments. Since the reverberator frequency does not change from note to note, each note in the range of the instrument has a unique timbre instead of the annoying uniformity too often characteristic of computer instruments.

High Level Software for Manipulating Musical Sound

Recent work by Chris Chafe, Loren Rush, and Andrew Schloss at CCRMA, in conjunction with Scott Foster and Bernard Mont-Reynaud at Systems Control Inc., supported by a joint University-Industry grant from the National Science Foundation, is directed toward the development of an intelligent music recognition system.[2,4] The system is intended to take real signals as input, and produce a score, as well as performing various kinds of analysis as requested by the user. A major application will be the realization of an *intelligent editor of musical sound* having the capability of automatic music transcription.

For the recording studio, there is a need for a "sound editor" which allows specification of sound events in more musical terms. Searching for, say, a trumpet entrance, change of key, or accelerando is a currently a tedious process. One might wish to specify, for example, "take me to the fifth note of the third bar after the violin enters," in order to perform some surgery on the sound samples there. Such an operation relies on the ability to recognize notes and instruments in the sound, and to establish the musical structure of the piece. Acoustical and musical expertise are both necessary to achieve these purposes.

The research has been divided into two levels. The lower level signal processing produces a note list from the digitized signal. The higher level operates on these note lists to produce data structures which reflect the musical content in a useful fashion.

First, the signal is partitioned by the low level into a series of individual notes using segmentation on the basis of amplitude and/or pitch. The note list is created by signal processing algorithms which track spectral harmonics. The current implementation works reliably for one voice. It is hoped that system identification techniques can help extend the system to polyphonic contexts.

The high-level analysis begins by detecting rhythmic or melodic accents in the note list, and noticing quasi-periodicities. This information is gradually refined and cross-checked until an hypothesized relationship between real time and musical time is established throughout the piece (the "tempo line"). Note values are

assigned as a part of this process, relying on a combination of heuristics for hypothesis formation, evaluation, and mutual reinforcement. The key and time-signature are also determined, and a musical score is produced. The system currently expects music of the 18th century, but expansion is straightforward in terms of a more complete knowledge base. Knowledge sources applicable to diverse instruments and/or musical styles can be incorporated as they are developed.

Currently, the upper and lower levels of the system are largely independent. As the system becomes more sophisticated and is called upon to deal with polyphonic input, the interaction and interdependence between high and low levels will increase. This is because more difficult multi-voiced sources are not easily analyzed by fixed signal processing methods. It is becoming necessary to implement "goal driven signal processing," as the higher level analyses raise questions regarding the content of the sound.

LISP as a Composition Environment

As composers gain experience with the computer at CCRMA, they tend more and more to program actual composition or "performance" processes rather than specifying each individual note. One language which seems well suited to a programming-intensive environment for composing is LISP. The dialect in use at CCRMA is MACLISP, coded for the PDP-10. To date, the primary users of LISP for note list generation have been Marc LeBrun and Paul Wieneke.

"Notes" are usually described in the form of parameter lists which control routines that are analogous to musical instruments. Phrases involving a succession of notes, movements, parts, and entire pieces can also be represented by single-level or multi-level lists, the fundamental data structure in LISP; thus the language has a large number of primitives to deal with the important units of a computer score. It has the additional advantages of being an interpreter, and being highly modular. The latter is especially useful because one can easily isolate code which is of general use from that which is specific to a particular piece or user. This can then be compiled into machine code and then loaded at will into someone's personal LISP environment.

Few pieces of music can be expessed purely as algorithms. Most situations require a "hands on" control of a variety of compositional processes. One approach has been to supply these processes with a sort of "meta score" with which one can "tune" the functions that produce the final note list. Such a meta score can be expressed entirely in LISP symbolic expressions which serve as function arguments. This greatly simplifies most i/o, scanning, and parsing problems, of course, but it also gets to the heart of LISP's attributes for symbolic manipulation. Data and functions are expressed in the same general form. One can treat functions as data until they are explicitly evaluated. LISP Functions don't specify the type of their arguments. Since arguments can be numbers, lists, or other functions, one can provide information in an input score not only by the values for items, but also by their structure.

Many of the essentials in LISP note-writing programs, such as basic control structures and scheduling, are based on techniques used by Bill Schottstaedt in his Pla interpreter. Pla was the first score generating language at CCRMA to make extensive use of list processing and it will accept LISP forms in many situations.

Distilling one's musical experience into computer code is of immense value simply as a learning experience, just as is writing idiomatic music for performers. The flexibility and power of LISP has been brought to bear successfully on the computer simulation of real world events throughout its history. Music is just one of the newer applications.

Pla — A New Composition Environment

During the last three years, Bill Schottstaedt has been developing a powerful compositional system named Pla. The goal is to build a fully integrated, programmable system which allows the composer to represent his musical thoughts in almost any arbitrary form, with easy access to the synthesizer and graphical output. The basic parts include an interpreter called Pla which has its own note list editor, a music compiler named Sambox, and a library of "instrument" building routines.

The language that Pla interprets looks much like Sail, a high level Algol language. Numerous music related features have been made a part of the basic language, including obvious things like frequency names and envelopes. Parallel processes called "voices" are the basic music producing entities. Voices can create other voices and can affect voice scheduling explicitly. A message passing system provides a flexible

communication path between voices. Groups of voices can be packaged as a "Section", which can then be recalled, ornamented, and edited in whatever way the composer desires.

The output of Pla is usually a list of notes, each note being an instantiation of a music compiler "instrument". The instruments are synthesizer patches written in Sail and loaded into the compiler. Because instruments often have dozens of parameters, it can quickly become a bother to try to remember what each parameter is doing. Pla provides a way to name parameters, and the music compiler provides ways to handle default parameter values, but the note list always seems to become a morass of numbers. A knowledgeable editor is obviously needed.

A recent addition to the Pla interpreter is a programmable note list editor patterned after the E editor by Arthur Samuels of the Computer Science Dept. at Stanford. In addition to the text window that E provides, the composer can call up as many as three other windows, each holding a different representation of the score. The contents of each window can be customized to suit the composer, but the default is that one window has a graph of the frequency of each note versus the time in seconds, another window has a common musical score, a third has the text of the note list, and the fourth shows the progress of any sub-jobs that are running. The latter includes primarily calls on the music compiler and the synthesizer from the editor. The composer now has all the flexibility of the Pla environment coupled with a text and graphics editor.

Conclusion

This has been a brief introduction to some of the issues in computer-assisted music. Even at CCRMA there are many more current research enquiries than we could present here, such as simulation of concert halls, synthesis of bowed strings, the singing voice, and ambient space illusions, to name a few.

For the future, we are eagerly awaiting faster processors and larger more reliable disks. Music is one of the more demanding applications of digital technology, and there is much to be done with improved capacity when it becomes available.

References

The list of references below is a sampling of prior and future publications from CCRMA. For more general references, the reader is referred to the Computer Music Journal.

1. Chafe, Chris. "Modelling String Sounds." In Roads, C., and J. Strawn, Eds. *Computer Music*. Cambridge: MIT Press, Forthcoming.

2. Chafe, Chris, Bernard Mont-Reynaud, and Loren Rush. 1982. "Toward an Intelligent Editor of Digital Audio: Recognition of Musical Constructs." *Computer Music Journal* Vol. 6 No. 1 (to appear).

3. Chowning, J. 1973. "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation." *Journal of the Audio Engineering Society* 21(7):526-534, 1973. Reprinted in Roads, C., and J. Strawn, Eds. *Computer Music*. Cambridge: MIT Press, Forthcoming.

4. Foster, Scott, and Andrew Schloss. 1982. "Toward an Intelligent Editor of Digital Audio: Signal Processing Methods." *Computer Music Journal* Vol. 6 No. 1. (to appear).

5. Jaffe, David, and Julius Smith. "Extensions of the Karplus-Strong Plucked String Algorithm." Submitted to *Computer Music Journal*.

6. Moorer, James A. 1977. "On the Transcription of Musical Sound by Computer." *Computer Music Journal* 1(4):32-38. Reprinted in Roads, C., and J. Strawn, Eds. *Computer Music*. Cambridge: MIT Press, Forthcoming.

7. Moorer, James A. 1977. "Signal Processing Aspects of Computer Music – A Survey." *Proceedings of the IEEE* 65(8):1108 - 1137. Reprinted in Roads, C., and J. Strawn, Eds. *Computer Music*. Cambridge: MIT Press, Forthcoming.

8. Moorer, James A. 1978. "The Use of the Phase Vocoder in Computer Music Applications." *Journal of the Audio Engineering Society* 26:42–45.

9. Moorer, James A. 1979. "About this Reverberation Business." *Computer Music Journal* 3(2):13-28. Reprinted in Roads, C., and J. Strawn, Eds. *Computer Music*. Cambridge: MIT Press, Forthcoming.

10. Rush, L., James A. Moorer, and D. G. Loy. 1976. "All-digital sound recording and processing." Paper presented at the *55th convention of the Audio Engineering Society*, Oct. 29, 1976.

11. Samson, P. R. "Architectural Issues in the Design of the Systems Concepts Digital Synthesizer." In Roads, C., and J. Strawn, Eds. *Computer Music*. Cambridge: MIT Press, Forthcoming.

12. Schottstaedt, Bill. 1977. "The Simulation of Natural Instrument Tones using Frequency Modulation with a Complex Modulating Wave." *Computer Music Journal* 1(4):46-50. Reprinted in Roads, C., and J. Strawn, Eds. *Computer Music*. Cambridge: MIT Press, Forthcoming.

13. Sheeline, Kip. "The Psychoacoustics of Sound Localization: An Overview." In Roads, C., and J. Strawn, Eds. *Computer Music*. Cambridge: MIT Press, Forthcoming.

14. Smith, Julius. "Introduction to Digital Filters." In Roads, C., and J. Strawn, Eds. *Computer Music*. Cambridge: MIT Press, Forthcoming.

15. Smith, Leland. 1972. "Score: A Musician's Approach to Computer Music." *Journal of the Audio Engineering Society* 20:7-14.