AN INTERACTIVE GRAPHICAL ENVIRONMENT

FOR MUSICAL, ACOUSTICAL

AND PSYCHOACOUSTICAL RESEARCH

Submitted to

NATIONAL SCIENCE FOUNDATION

June 1977

Department of Music
School of Humanities and Sciences
Stanford Universtiy

AN INTERACTIVE GRAPHICAL ENVIRONMENT FOR MUSICAL, ACOUSTICAL AND PSYCHOACOUSTICAL RESEARCH

John M. Chowning, John M. Grey, F. Richard Moore
James A. Moorer, Loren Rush

Center for Computer Research in Music and Acoustics
Artificial Intelligence Lab, Stanford University
Stanford, California 94305

ABSTRACT

Research over the past decade in computer applications to sound synthesis, signal processing, acoustics, and psychoacoustics coupled with recent advances in real-time digital signal processing hardware, make feasible a powerful interactive research environment. The construction of an interaction console and signal processor are now near completion as is the basic support software. Research support is sought for the continued development of higher level interactive software and for the integration of both existing and proposed research programs with this interactive environment. The research programs include: 1) an interactive acoustic manipulation language, 2) interactive recording, editing, processing, and mixing of digitized audio waveforms, 3) transformation of sensory input, 4) interactive real-time psychoacoustic experiments, and 5) high dimensional graphics by means of embedded spaces.

TABLE OF CONTENTS

OVERVIEW 1
Reader's guide 3

PREVIOUS WORK 4
Early Work 4
Interactive Music Compiler 4
Interactive Graphic Spatial Routines 5
Frequency Modulation Synthesis 5
Music Manusctipt Program 6
Psychoacoustics Research Programming Support 7
Analysis and Synthesis 7
Interactive Graphics 7

WORK IN PROGRESS 9
Real-time System Hardware 9
Interaction Console 9
The Systems Concepts Digital Synthesizer 10
The Graphics Console 12
Software Support 13
A Simulation Model of Musical Sound 13
Software Support for the Digital Synthesizer 15
An Interactive Acoustic Manipulation Language 16
Signal Processing Support 21
The Phase Vocoder 22
Nonlinear Synthesis 22
Reverberation Studies 25
The CCRMA Digital Recording Studio 28
Recording Facility and Computer Hardware 29
Record/Playback Program 31
Editing, Processing and Mixing Digitized Audio Waveforms 32
The Sound File Editor 33
Retuning 34
Cross-synthesis 34
Implementation of Existing Techniques 34
Digital Mixing 35

PROPOSED WORK 36
Transformation of Sensory Input 36
Use of the Interaction Console 36
Interactive Real-time Psychoacoustic Experiments 37
Interactive Digital Recording, Processing and Mixing 38
Recording Techniques 39
Archival Techniques 40
High Dimensional Display - Embedded Spaces 41
Program Options 42
An Application 44

APPENDICES

A. A Brief History of CCRMA 47
B. Signal Processing Aspects of Computer Music - A Survey [Moorer, 1977] 51
C. Use of SAIL Process Structure for Musical Purposes 85
D. The Use of the Phase Vocoder in Computer Music Applications [Moorer, 1976] 89
E. Multidimensional Scaling Techniques 98
F. Multidimensional Perceptual Scaling of Musical Timbre [Grey, 1977] 102

BIBLIOGRAPHY 116

BUDGET AND PERSONNEL 121

OVERVIEW

The Stanford Center for Computer Research in Music and Acoustics (CCRMA) is engaged in research into the analysis, synthesis, perception and manipulation of digitized sound. This has been done for the purpose of the computer synthesis of musical sound and for studies in the perception of musical sound.

The status of the facility as it now stands is multi-faceted. As a musical instrument, the computer system is possibly the most flexible of musical instruments. To speak of it as a conventional musical instrument, however, is somewhat misleading because the system is capable of simultaneously producing a large number of independent voices having arbitrary timbral characteristics; it is much more general than a conventional musical instrument. It can generate any sound that can be produced by loudspeakers, modify and transform real sounds entered into the system by means of microphone, remember and modify articulated musical input, and simulate the location and movement of sounds in a variety of illusory reverberant spaces. Equally important, the facility is capable of serving a number of composers and researchers, providing for each a direct control over the medium which was never before possible.

As a research tool, the computer has shown itself to be uniquely useful in generating precisely controlled stimuli for perceptual research. The analysis-synthesis techniques developed here allow for direct experimentation with the sounds of natural instruments. By modifying the sounds of these instruments in systematic ways, then testing the perceptual effects of the modifications, a great deal of information has been produced on the way musical timbre is perceived. Several papers and technical reports have been produced describing the techniques and results of this research [Grey, 1975; Moorer, 1975].

In addition to the normal teaching and research function during the academic year, CCRMA holds special summer workshops for musicians and scientists from outside the university. These workshops have been held every summer since 1969 with students attending from this country and abroad. In a six-week session, the students are able to learn basic computer programming, fundamentals of acoustics and psychoacoustics, and produce a

composition. These summer sessions are limited to 20 people each year. One of the compositions from the most recent summer session was chosen for performance at the 1976 Music Computation Conference in Boston.

Another aspect of CCRMA's work is in the field of archiving recorded sound. Each time an audio tape is copied, the sound quality deteriorates. As an audio tape ages, the sound quality deteriorates. One way to prevent this process is to digitize the sound and store it as binary data on computer tapes. These can be recopied without error any number of times. This promises to be a way to produce any number of "master" quality recordings without degradation of the sound. Extending this further, we can in fact replace all the analog equipment (tape, mixer, processing devices) by digital hardware. This was implemented at CCRMA in 1976. A series of live digital recordings were made which were of extremely high quality. With the computer, this sound can be processed in many ways. Perfectly precise and noise-free splices can be made, passages may be retuned, "dirty" attacks can be edited and cleaned up. In short, the digital recording studio promises higher quality and greater flexibility than was ever before possible. The prototype digital recording studio at Stanford will serve as a model for future installations.

Several new methods for analyzing natural sounds have been developed. Analysis can be done for insight or for the purpose of resynthesizing the sound accurately, or with modifications to produce new sounds that still preserve lifelike quality. The combination of the phase vocoder and the linear predictor [Moorer, 1976] is capable of analyzing passages with vibrato, glissando, or any other musical nuance with perfect fidelity. One intended use of this technique is not only for musical composition but also for experiments in music perception in context. By analysing entire musical passages, the acoustical data can be modified, changing just the features under study and leaving other features constant.

In computer language design, a powerful interactive interpretive data manipulation language has been developed. This language allows the user to deal with the large amount of coordinated data that the analysis techniques produce in a graphical and interactive manner. The language has a macro facility, so that extreme conciseness is possible. Positional notation in procedure calls has been abandoned in favor of named procedure parameters. This permits "intelligent" defaulting of parameters, such that the program

always attempts to "do what I mean" without the user having to supply specific values for every optional argument.

CCRMA is now in the process of acquiring, through purchase and construction, a large scale real-time computing facility. This include a special purpose all-digital signal processor, and an interaction console, consisting of knobs, buttons, a tablet, one or more organ-like keyboards, and other man-machine interface devices. This will allow instantaneous auditioning of changes in synthesis data, as well as on-line psychoacoustic testing with complex stimuli.

This proposal is concerned mainly with the interactive, graphical, and computer language aspects of the Center's work. It is proposed that the work currently in progress continue, and additionally several other projects relating to real-time interactive use of the computer system be initiated.

Reader's Guide

This proposal is oraganized such that there are three major headings, Previous Work, Work in Progress, and Proposed Work. Within each of these headings the particular research areas of interest are defined or proposed, as the case may be. These areas are concerned with the interactive, graphical, and computer language aspects of the Center's work. The table of contents includes the 'keywords' so that the reader can easily track a topic of interest.

Appendix A consists of a brief history of the Center and a description of the facility, while Appendix C is a description of parallel processing applications. The other four appendices are articles, either published or in process, which constitute useful background for the work which is proposed.

PREVIOUS WORK

The Stanford computer music group has a history stretching over more than twelve years of the creation and use of interactive graphical tools for experimentation with acoustical phenomena, mostly in the context of computer synthesized musical sound. Over the last 5 years, however, the emphasis has been broadened to include psychoacoustics and perceptual testing, as well as advanced digital signal processing techniques. To these ends, a number of powerful interactive graphical programs have been written that help the user conceptualize and manipulate the vast amounts of data that are often required for lifelike sound synthesis.

All of this work to date has been done at the Stanford Artificial Intelligence Laboratory on the PDP-10 time-sharing system. A description of the facility may be found in Appendix A.

Early Work

Interactive Music Compiler

One of the first interactive programs written at Stanford is the MUS10 music synthesis language. This is an ALGOL-based interactive music language patterned somewhat after MUSIC V [Mathews, 1969]. In music languages in general, the program is divided into two parts: the instrument definition and the note list. In MUS10, the instrument definition was in essence a small ALGOL-like program which generated the next sample of the sound. It made use of precoded basic functional modules called "unit generators". An example of a unit generator might be a sinusoidal oscillator. Other unit generators were provided for wave shaping and various kinds of modulations, and other processes. In addition, any construct not provided in the various unit generators could be written directly in ALGOL. For instance, even though there was no unit generator to do digital filtering, this algorithm was easily written as an ALGOL-like subroutine some years ago. The language is a load-and-go compiler, where instrument definitions are compiled into the program's local

storage. The note list then specifies instantiations of these instruments: when they are to begin, how long they are to sound, what are the performance parameters (pitch, amplitude, etc). The note list can be prepared beforehand, or it can be typed in on-line for interactive experimentation. The sound is then computed quickly and played, thus providing the necessary feedback to allow the user to alter the synthesis parameters at will. All statements are compiled. There is no interpretation, although the effect is immediate and highly reminiscent of purely interpretive languages. The process of compilation is entirely invisible to the user.

Soon after MUS10 was in use, graphical features were added, as well as more advanced ALGOL constructs such as IF-THEN-ELSE statements and loop constructs (FOR, WHILE, DO-UNTIL), all of these operating in a load-and go fashion, accessible interactively from the terminal as well as from prepared programs. Waveforms and synthesis functions (envelopes) can be displayed and altered in an on-line fashion, auditioning the resulting sound after each change.

Interactive Graphic Spatial Routines

In addition, two more support programs for preparing synthesis data for MUS10 were written. These included a program for designing synthesis functions and a program for designing spacial patterns [Chowning, 1971]. With this latter program, the user could specify a trajectory that he wished the illusory sound source to follow. The program would display the trajectory with a diagram of the room and its four loud-speakers superimposed within the trajectory. The user could then edit the sound path until the desired shape was attained (see Figure 32 of Appendix B). The program then automatically prepared the required synthesis functions (energy distribution to the four channels, amount of reverberation, doppler shift, etc etc) to simulate that trajectory in synthesizing the tone.

Frequency Modulation Synthesis

Through the period of 1968 through 1971 a fundamental breakthrough was achieved in the synthesis of complex audio spectra. Until this time nearly all electronic sound synthesis, both analog and digital, was based upon the processing of fixed waveforms. The only

exceptions were a few computer based studies in summation synthesis where the amplitude and frequency components were independently controlled as a function of time. Summation synthesis proved to be be very powerful although expensive and not always intuitive. At Stanford we discovered a means to generate time-dependent spectra by modulating the frequency of a sinusoid by a second sinusoid (frequency modulation synthesis). This technique proved to be economical in computation and yielded a multitude of perceptually different timbres by changing just a few parameters. This technique has been under development ever since and is described in the section 'Work in Progress'.

Music Manuscript Program

Soon after this, work was also begun on a program for music manuscripting. This has developed over the years into one of the most powerful computer-aided manuscripting systems in existence. The user types in, one staff at a time, the notes of the piece. Any special symbols to be used can easily be defined using the light pen and then positioned accurately. The computer provides a wide variety of semi-automatic features, such as positioning the notes on the staff to achieve a reasonable spacing without overlap, aligning of multiple staves to achieve spacial synchrony over the total score, and extraction of individual parts from full orchestral scores [Smith, 1973]. The resulting score can be quickly printed by a Xerographic process, or can be slowly drawn at 4X enlargement by a Calcomp plotter. Subsequent photoreduction produces an extremely high quality manuscript. Using this system, sample pages of scores of exceptional difficulty (such as Ligeti's *San Francisco Polyphony*) have been printed with ease. Several music composition graduate students at Stanford have notated their new works entirely using the manuscripting programs. This not only aids "debugging" of the composition, but produces highly readable copy for the performers - somewhat of a novelty for student compositions.

WORK IN PROGRESS

In this section, we describe hardware under construction, programs under development, and new techniques of sound processing. Of the projects described below, the software development is the most complete at this time. All of the programs described are in use. The continuing development consists of making the system more general and more efficient. The hardware is set for completion within the next six months.

The goal of this work is to establish a comprehensive system for recording, editing, processing, manipulation, storage, and synthesis of sound. We wish to provide a range of turn-around times from batch mode, through interactive mode (both these modes are available now) to real-time mode (which awaits the appropriate hardware).

We shall describe the hardware first, the software next, then the signal processing support, and lastly the digital recording and processing.

Real-time System Hardware

We have currently nearing completion of hardware which will allow real-time synthesis of complex audio signals and real-time interaction at a high level with the synthesis and display routines. The hardware is divided into three parts: the digital synthesizer which is being built out-of-house at Systems Concepts, Inc., the interaction console, which is much simpler and is being built in-house, and a DEC GT-46 graphics computer.

Interaction Console

The interaction console is a microprocessor-based data collection center with provisions for the connections of a large amount of input devices, such as knobs, tablets, organ-like keyboards, button and switch panels, and the like. The microprocessor has basically two interfaces. The first interface is a high speed serial interface with the PDP-10 computer. This interface transmits a 24-bit datum (three microprocessor words) to the PDP-10 in

about 5 microseconds. This datum consists of an 8-bit source identifier and a 16-bit data word. The other interface is what we call the "panel" interface. It connects to 16 EIA connectors, each one of which can access up to eight 8-bit words. This gives each connector the capability of reading 64 bits. This means that a single 64-key organ keyboard will plug into a single slot on the connector panel. Each connector, then, supplies a 3-bit address, an enable signal, a read/write signal, and an 8-bit bidirectional data bus. A box containing 8 potentiometers, each digitized to an accuracy of 8 bits, can plug into single slot also. The devices that are initially planned are one or two organ keyboards, 2 boxes of 8 potentiometers each, and a square array of 64 momentary pushbuttons. The microprocessor will scan these panels repetitively. When changes in state are noted, these changes will be forwarded to the PDP-10 via the high-speed bit-serial line. If all 16 panels are used, a complete scan of all the panels will take about 20 milliseconds. As fewer panels are used, faster interaction is possible.

Since the interaction console forwards only changes to the PDP-10, the main computer can treat these as interrupts, or events. The PDP-10 is relieved of the burden of scanning these devices individually. Additionally, the use of the microcomputer allows more sophisticated coding of the devices. For instance, a panel of momentary push-buttons can be made to behave like toggle switches simply by changing the program in the microprocessor. It is planned to include a light-emmiting diode with each momentary switch so that the internal state, on or off, will be visible.

The Systems Concepts Digital Synthesizer

Systems Concepts, Inc., of San Francisco is now building for CCRMA a large-scale digital synthesizer. The Systems Concepts device has a fixed menu of functional modules: there are up to 256 "generators" and 128 "modifiers". They are interconnected by RAM called "sum" memory. There is, in addition, an associated memory of up to 64K 20-bit words which can be used as up to 32 tapped delay lines for table lookup. Figure 2 shows a simplified block diagram of the device. Each generator reads one datum from sum memory and outputs one datum. Each modifier can read two data and write one datum. These functional modules are time-division multiplexed. The device operates on a 195 nanosecond "tick". Each generator takes one tick and each modifier takes two ticks. Ticks on which

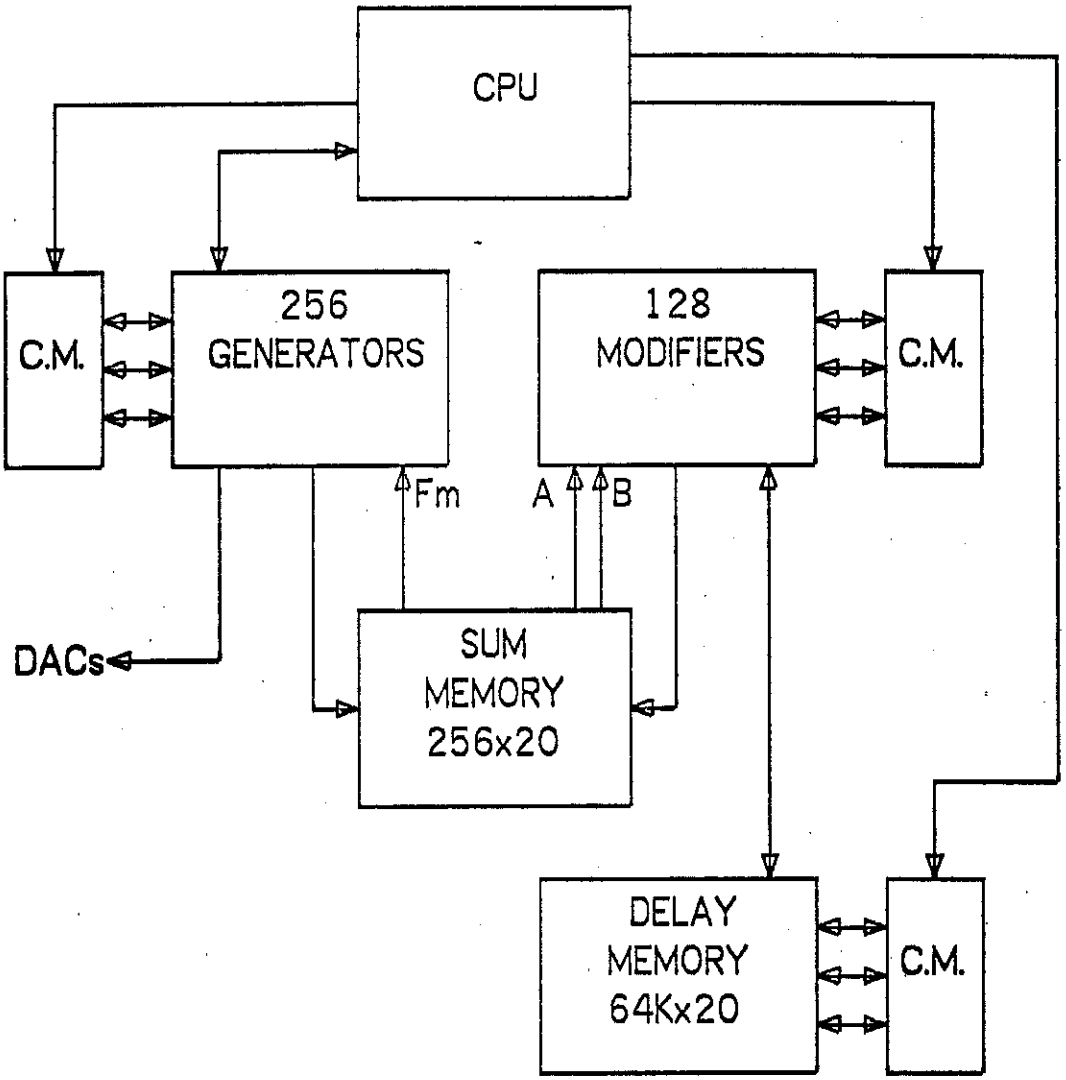

FIGURE 2. Block diagram of the Systems Concepts Digital Synthesizer, designed by Peter Samson. Each generator can synthesize a sinusoid, band-limited pulse train, non-limited square wave, sawtooth, or pulse train. The amplitude of the waveform can also be modulated by a piece of a piece-wise linear or exponential envelope. Each modifier can do two poles or zeros (second-order digital filter section, either numerator or denominator), amplitude modulation, 4-quadrant multiplication, white noise generation, max and min functions, zero-cross pulsing, and in conjunction with the delay memory, can perform the operations of a unit reverberator. The delay memory can be partitioned in to up to 32 independent sections. Each section can be a tapped delay line, for use in unit reverberators, or can serve as a table for generalized function approximation (like square root, reciprocal, arctangent, etc). A generator can also be a data channel which can transfer sample-time data to or from the CPU or the DACs. The sum memory serves as an interconnection matrix for the functional modules. The modules are controled through their control memories (labeled C.M. in the figure) which can be set by the CPU.

computations take place are called processing ticks. There are also update ticks which are used to change the controlling parameters of any of the functional modules. The total of processing ticks plus update ticks (plus 9 for pipelining) determines the overall sample rate. If all the generators are to be used, the maximum sampling rate would be somewhat less than 20 KHz. If only half the generators are needed, the resulting sampling rate could be as high as 40 KHz. These figures would be somewhat lower if many update ticks were needed for a given task.

Each generator and modifier has a control memory that determines its function and holds its running values and parameters. This control memory can be written by the CPU, and this is how the device is operated.

A generator can have several different functions, depending on its "mode", which is a parameter in its control memory. A generator can produce non-band-limited square waves, pulse trains, sawtooth signals as well as pure sinusoids and band-limited pulse trains [Winham and Steiglitz, 1970]. For all of these modes, the input datum from sum memory serves as a frequency modulation control so that any of these waveforms can be frequency modulated by other functional modules. Inside the generator there is a provision for linearly sweeping the frequency in addition to the external frequency modulation input. There is also amplitude control of any of the generated waveforms. The amplitude may be changed either linearly or exponentially. The result of this power is that control functions (like amplitudes and frequencies of sinusoids with time) in piecewise-linear form can be used directly by the hardware. This corresponds nicely to our analysis form for additive synthesis.

A generator can also be used as a data channel to pass sampled data between the sum memory and the CPU. This allows mixing of precomputed or prerecorded sounds with freshly generated or modified sounds. The output can also be stored by the CPU. This means that the device will never be outgrown, because the data can always be shuffled back to the CPU for anything the device cannot do. We consider this provision one of the most important features of the device. Too many synthesizer designs do not allow this most simple process and thus have "hard" limitations.

A modifier can be used to synthesize a second-order section of a digital filter (either two poles or two zeros but not both), amplitude modulation, 4-quadrant multiplication, max and min functions, zero crossing pulser, random noise (by the linear congruential method [Knuth, 1969]) and several others. When used in conjunction with the delay memory, unit reverberators or table lookup can be realized. The table lookup feature we consider quite important because this allows one to do general purpose functional approximation for getting functions that are not hard-wired into the device, such as division (reciprocal table), square roots (for energy normalization), arc-tangents, and numerous others. This is the "escape" that allows quite a bit of generality, with the only limit being the restriction to a maximum of 32 such functions that can be accessed at once.

Most of the data paths through the machine are 20-bit fixed-point paths. The sine table in the generators is effectively 8192 locations long. The multiplies in the modifiers are 20x20 multiplies. Both sum memory and delay memory are 20-bit fixed-point memories.

The sum memory is so named because it is cleared at the beginning of a pass through all the functional units (one sample) and each write into this memory adds into the location rather than replaces. Each modifier has the option of adding into memory or replacing, but the generators only add into sum memory. There are actually two sum memories, one for this pass and one for last pass. Reading from sum memory usually comes from the last pass, so there is a one sample delay in passing data from one functional module to the other, but ordering of the modules is not critical when using this mode. Since the modifiers can read from either sum memory (this pass or last pass), they can be cascaded with no delay. When using this feature, care must be given to the ordering of the units. The programmer must be assured that the previous unit has deposited its output in the memory before the next unit is to read it. Since the machine is extensively pipelined, the data is deposited several ticks after it is read.

The Graphics Console

The DEC GT-46 graphics console is a general-purpose PDP-11 computer with a vector display and a small amount of disk storage. This will be interconnected with the PDP-10 with another high-speed bit-serial line, much like the interaction console, except that the

primary direction will be from the PDP-10 to the graphics console, rather than the other way around; the graphics console is mostly an output device.

The point of having a stand-alone computer for a graphics console independent of the main computer is that this allows the possibility of doing extensive graphics, such as rotations of spacial configurations or "moving windows" on actual sound waveforms concurrent with complex sound synthesis and high-level interactive input. This way each part of the process (the input, the display, and the synthesis) is done by entirely different, distributed, computing devices, all coordinated by the PDP-10.

Software Support

Rather than write an entirely new language from scratch, we have chosen to augment an existing language. That language is SAIL, an extended version of ALGOL [Reiser, 1976]. The augmentations are principally in the form of interactive top-level command loops. SAIL itself provides a rich framework for research, allowing parallel processes, records and references, and events and interrupts, as well as the standard sorts of control structures available in a modern high level language. The features of SAIL make it suitable for use as a simulation language, which as we shall see below is a convenient way to represent musical structures.

A Simulation Model of Musical Sound

Let us describe one possible division of musical synthesis to illustrate the use of parallel processes to simulate the production of musical sound. Figure 3 shows such a division which carries the process of music production from the highest level (the piece itself) through the intermediate levels (the voices and the notes in each voice) through the lowest acoustical levels (the partials of each note and the line segments that make up each partial). Appendix C gives a more detailed description of the use of parallel processes in SAIL.

For purposes of illustration, we will use the additive (or Fourier) model of sound synthesis [see Chowning et al., 1975] in which a tone is represented by a collection of nearly harmonic

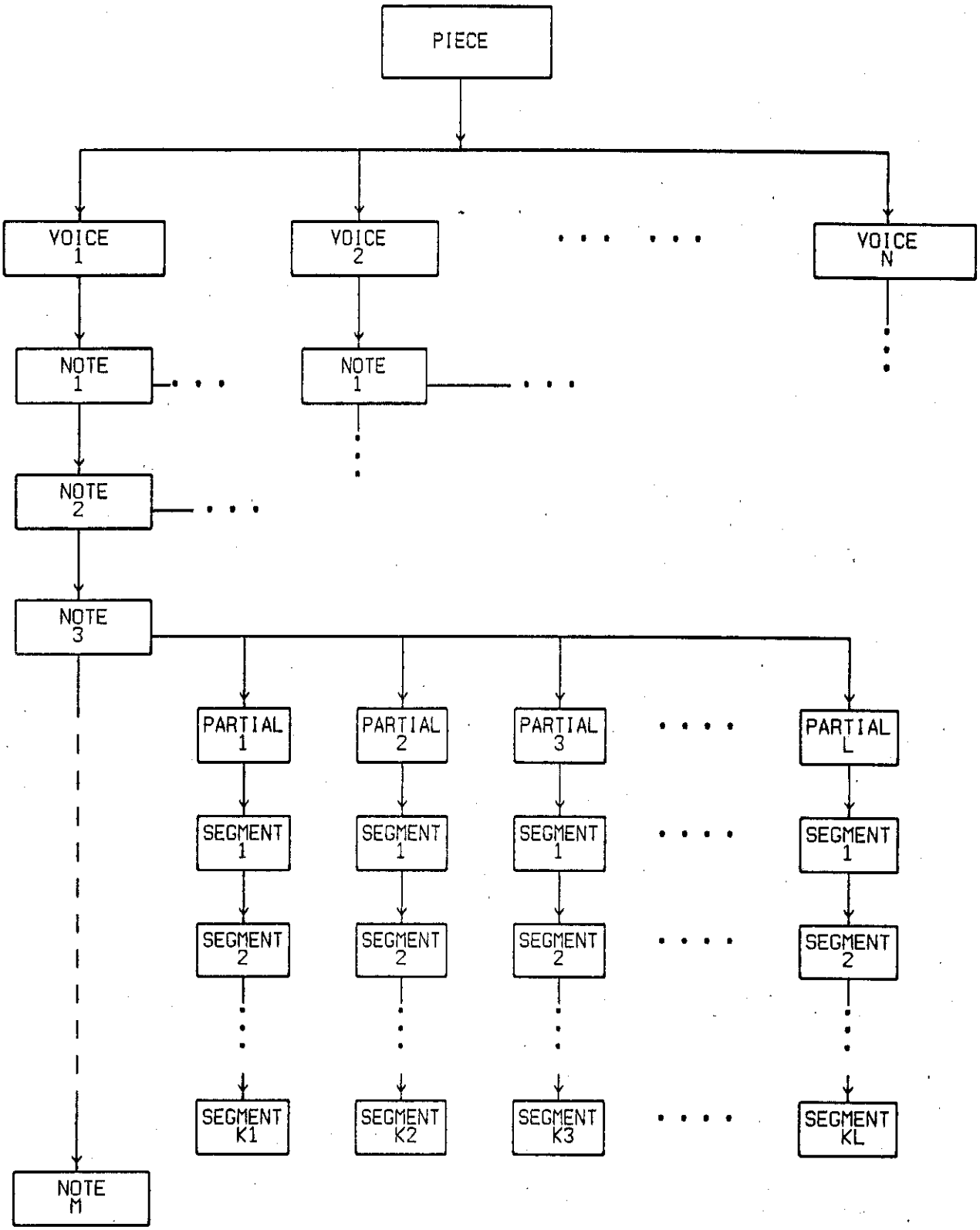

FIGURE 3. Diagram of a parallel process model of one kind musical sound from the highest level (the piece itself) through various intermediate levels (voices, notes within a voice) to the acoustical level (the partials of each note and the line segments for each partial).

sinusoidal partials, each one with time-varying amplitude and frequency. These amplitude and frequency contours are generally approximated by piecewise-linear functions, where each function consists of a number of straight line segments. A simple but very powerful model was developed by Grey [1975] that uses 5 to 7 line segments for each function. To synthesize the tone, then, a series of sinusoidal oscillators, currently software but soon to be hardware oscillators, are computed and added together to produce the sampled-data waveform for that note. The outputs of all the oscillators over all the simultaneously-sounding notes must be added together to produce a single data stream to be presented to the digital-to-analog converter and thus to the listening ear.

In this particular division of music synthesis, the top of Figure 3 is the piece itself. The piece sprouts N parallel processes, each one of which represents one parallel voice. For instance, in a string quartet, the piece might sprout 4 voices representing each part as a separate voice. Each voice then would sprout, in time order, each note of that voice. The use of a simple calander queue allows the interleaving of the various notes of the voices. Each voice would execute a WAIT_UNTIL(TIME) operation to advance to the time of the next note, then the note would be sprouted. The notes must be sprouted rather than just executed as subroutine calls because with some instruments (bells, for instance) notes overlap heavily such that consecutive notes of even the same voice must be considered to be parallel processes. Each note then starts up the numerical routines necessary to generate the tone of that particular instrument.

Figure 3 is drawn as a hierarchical structure, but it is in fact a directed graph. For instance, the piece may execute a JOIN instruction to wait until some of its voices have terminated, then perhaps begin other voices. Each voice can execute a JOIN with some of the notes so that the oscillators used by the notes can be recycled and used by other notes.

One of the great advantages to using a general-purpose language for musical structures, rather than building special-purpose languages, is the already existing rightness of available control structures. For example, in some more modern music, it is convenient to think of one voice "controlling" certain aspects of other voices. This could be done in SAIL in many ways. Perhaps the simplest way is through global variables that one voice sets and another voice reads. The EVENT structure provides a general inter-process communication

system with methods for creating and interrogating events in a unified manner.

Software Support for the Digital Synthesizer

The support packages for the digital synthesizer are set up in two distinct subroutine packages: what we call the "lower-level" routines and the "intermediate-level" routines.

The lower-level routines. These consist of subroutines to allocate and deallocate modules of the synthesizer, to set and modify parameters of allocated units, and to set run modes and controlling information. The routines are unified into a small set of calls: GET, GIVE, BIND, and SET_FIELD (plus a few other less important ones) [Loy, 1977]. GET and GIVE are used to allocate and deallocate modules. GET takes as argument an identifier specifying what kind of module is desired. BIND takes three arguments: the unit name, the field name, and the datum. This assembles the triple into a command word for the synthesizer and places it into the output stream. A primary function of the SET_FIELD call is to specify timing of the output stream. Normally, the commands will be read and interpreted by the synthesizer as fast as possible, which is not always desirable. To specify that a particular command is not to be executed until a certain time, a "linger" command must be given using the SET_FIELD call. This specifies that the synthesizer is not to read any more commands from the data stream until the sample number specified in the linger command is reached. Then command processing begins again.

Since the input command format of the synthesizer has options for packing more than one command to a word, the low level routines provide an optimising formater where this repacking is done automatically as much as possible. This assures a minimum command data rate at the cost of some obscurity in the resulting command stream.

The intermediate-level routines. This package organizes the basic-level calls into more coherent and compact routines which are designed around the concepts we have found useful in music synthesis experimentation. These routines use both the SAIL process structure for interleaving parameter settings and the low-level routines for packing and optimising the output stream.

Perhaps the most useful way to describe how one actually uses these routines is to give some examples of some of the calling sequences and what they do.

GOSC starts an oscillator at a given frequency and sprouts a parallel process to feed it successive line segments of an amplitude envelope. It takes as arguments the starting time and duration of the note, the frequency, the address in sum memory where it is to deposit its output, and a record defining the amplitude function. It returns the number of the oscillator and the process item of the new parallel process.

SHAPE multiplies an arbitrary signal by an amplitude envelope. It claims a modifier, sets up its parameters, then sprouts a parallel process to feed the modifier the amplitude envelope. The arguments include the address in sum memory of the input to the shaper and the destination address for the output.

REV starts up a unit reverberator of either the comb or the all-pass form. The arguments include the gain coefficient, the delay line length, and the input and output addresses.

Similar routines exist for filtering, mixing, random noise, and many others. Thus setting up a complex instrument, such as an additive synthesis instrument, just involves calling the above routines several times to start up the appropriate modules and connect them together through the interconnection memory.

Thusly, the programmer is relieved of the burden of having to remember all the things prerequisite to making (for instance) an oscillator work. Parameters are intelligently defaulted wherever possible.

An Interactive Acoustic Manipulation Language

We have constructed a language for the interactive manipulation of acoustic data. This language provides a flexible access to the acoustic data in order to evaluate its perceptual correlates. The data is obtained from an analysis of actual recorded materials, and models the sound waves in some convenient form. For instance, data used extensively in past

research was the time-variant amplitude and frequency functions for harmonics of selected instrument tones. The language was derived as a much needed extension of an earlier program for acoustic data manipulation in order to provide greater flexibility and a wider range of application.

The basic function of the language is as a tool for the modeling of perceptually salient features of waveforms. To facilitate this, it allows for the active manipulation of acoustic data to be used in constructing a synthetic sound. Perceptual measurements can evaluate the significance of acoustic data manipulation; the researcher can then relate findings to a model for critical features of timbre. A second provision of this research tool is the access to analytic procedures for reprocessing the acoustic data in forms which may give further insights into component attributes. The availability of interactive feedback with a graphic orientation greatly assists research efforts.

A simple hypothetical example of this is given. Suppose that the wave were represented as a set of harmonics which varied in frequency and amplitude. Let us suppose that the investigator was interested in the frequency domain of the signal. Various analytic searches can be performed through the set of frequency functions to trace correlated activity - and, for instance, some overall frequency contour superimposed upon all partials might be uncovered. The researcher could then vary the shape, extent and representational complexity of this factor in exploring its salience and possiblities for representation as a modeled feature of synthesis.

The language as such is a top-level interpretive package written in SAIL for a SAIL environment. This use of an existant language as the base for a higher level interpreter has, of course, taken its toll in the efficiency of the various operations, but provides a tremendous degree of flexibility that for the time being is well worth the price. In addition, since the extensions are written entirely in SAIL, this assures that changes can easily be made in the higher level language itself, thus allowing programmers who are not highly skilled at PDP-10 assembly language to make modifications. Also, the interactive source-language debugging system available for SAIL can be brought to bear on lingering bugs [Reiser, 1975].

The interactive upper level of the language is an amalgam of ALGOL, APL-like vector structures, and music languages such as MUSIC V. The ALGOL-like syntax is used for conditionals, loop constructs, declarations, assignment statements, and the like. The arrays are like those in APL, where the sizes can change dynamically, and simple operations can be performed elementwise on the arrays. We have, however, the additional information that these are, in fact, sampled data functions, so several other operations (interpolation, decimation, piecewise-linear approximation) are available that are not used in classical vector processing.

For any operations that are not immediately available, the researcher can just write a SAIL subroutine and easily incorporate it into the package. (SAIL [Reiser, 1976] is an extended dialect of ALGOL.) This is also how efficiency can be achieved. Since the interpretive top level of the system is inherently inefficient, highly repetitive tasks can easily be recoded in SAIL and incorporated into the system.

Internally, the routines are organized into procedures which receive messages defining their calling sequences. Each procedure can then determine for itself how to interpret this message. This is much like the message procedure system in SMALLTALK [Goldberg and Kay, 1976], though it is implemented in SAIL rather than in an entirely special-purpose language. A price is paid here in terms of efficiency. Indeed, part of the ongoing work on this language are methods for improving the efficiency of the system while maintaining as much of the flexibility as possible.

The language provides the researcher with a way to interactively construct particular schemes for tests on the acoustic information. By being display oriented, the research process is greatly assisted. The power of research intuition is increased further by the flexibility of interactive modes - including the possibilty for hand editing acoustic data. Schemes for analysis or modification found successful in one case may be extracted from interactive commands and preserved for other test cases. Of course, other schemes may be directly programmed in the language and tested on a variety of data. Interaction is made easier by an interpreter that allows the abbreviation of commands, defaults of unspecified arguments and on-line formulation of macros. We outline below the specific benefits of this language.

1. *Construction and call of arbitrary analysis and modification procedures.* Any technique of signal processing may be accessed programatically or interactively with this language through unified interfacing conventions. This facilitates the arbitrary combination of analysis techniques and modification -procedures, allowing the researcher to test any particular hypothesis about the acoustic data - both in terms of acoustic manipulation and in the light of further analysis of data.

2. *Construction of arbitrary synthetic models.* Closely related to the first point, arbitrary models for synthesis may be formulated. Manipulation of the structure itself of the acoustic data may be performed in view of hypotheses of the researcher. In this way, the model for the stimuli will change from the original mathematical model to one more based upon significant features of the sounds. For instance, we might decide that frequency contours should be specified by a smooth "baseline" pitch curve and an additional perturbation which was strictly random in nature. Using this language a number of such hypotheses can be developed and tested very quickly.

3. *Interactive mode.* The fact that this is an interactive language allows for greater feedback potential. Researchers may try out ideas formulated on-line, on the basis of a specific context of ongoing results. This allows for paths of processing unanticipated at the outset. The ease of interaction is increased by a more flexible interpreter - one which allows abbreviations in commands as well as defaulted arguments so that the researcher need not type formal and full lines of code in this mode. The construction of macros also increases on-line potential. Positional notation of procedure calls has been abandoned for named arguments. This allows the routines to have extensive calling sequences which can be defaulted in an intelligent manner. For instance, if the array name is not specified, the last array mentioned can be used. The display routines can decide on their own how to display a given function, or the defaults can be overridden by including specifications in the calling sequence. We believe that this feature of intelligent defaulting is one of the most powerful aids to concise interactive computing in use today. The ability to either let the program decide how to do things, to override its decision when necessary, or to alter the default settings easily constitutes a powerful set of conveniences that seems to allow the user to spend most of his time using the results of the programs rather than fighting the syntax. For instance, one does not have to remember how many arguments a function has: one can

just call the function and most likely everything will work out all right. Additionally, in interactive mode, all commands typed from the keyboard are stored. These commands can then be, for instance, used subsequently as macro-operations.

4. *Construction of macros.* The grouping of command sets, either on-line or in source programs, into macros allows the quick repetition of processes on other data. A process tested on one harmonic, for example, may be extracted from the command stream and given a name; it may be called thereafter, accepting appropriate arguments if necessary, for all other harmonics; it may be saved for other cases in the future by output to a source program. Macros may be nested to any level and can be edited just like any other program on the computer.

Macros may be abbreviated by as little as a single keystroke, thus providing extremely concise calling seguences (at the expense of mnemonic labeling!). For the experienced user, this provides an excellent "impedence match" (in the terms of Swinehart [1974]) while still allowing the novice user to use the longer more mnemonic labels. Each user can have his own set of macros which are easily loaded at start-up time. This macro facility provides a rudimentary "extensibility" to the language.

5. *Graphic environment.* The display of information is an important feature of this research tool. Graphic provisions are inherent in the language - featuring the arbitrary display of information in any form, up to the possibility of showing related sets of functions in three-dimensional form. For example, the amplitude functions for the set of partials of a tone may be displayed in an amplitude by frequency by time format. Manipulations and analysis may be performed with this format of feedback. This is a very important aid to research, visually suggesting important acoustic activity and ways of processing such activity for the testing of various hypotheses.

Functions can be viewed individually or in collections. When viewed as collections, the three-dimensional representation is most convenient, but an alternaite spectrographic representation is available for the special case of amplitude and frequency curves. Three dimensional representations may be viewed from any angle or perspective. One can "zoom in" to observe interesting features in more detail.

6. *Hand editing of data.* In addition to the arbitrary application of signal processing techniques to acoustic data, actual hand editing may be performed. Among the possible modes for input are light pen and teletype commands; macros can be constructed for higher level manipulations with appropriate feedback. Being implemented are hand inputs via sketch pad and arbitrary configurations of knobs, buttons and slide potentiometers. Note that this editing is done in conjunction with the graphical representation of the data. For instance, we may choose to edit one particular function being displayed in a three-dimensional plot. The entire display can be maintained to help give a context for the proposed change. In additional, the original unmodified function can be displayed superimposed on the function under modification.

These manipulations are very similar to the kinds of manipulations that were possible with the "visible speech" system of the Haskins Labs [Cooper et al., 1951]. The differences are that the use of the computer allows tremendous precision in storage and modification of these "visible" curves. In addition, the meanings of the curves are not limited by the hardware as was the case with the visible speech system. We can assign functions to control any aspect of the sound production we wish, since this is all done through software and therefore easily modified. Further, the patterns are directly the result of an analysis procedure, thus assuring a high degree of fidelity in the synthetic tones when required.

Signal Processing Support

Without a powerful base of signal processing routines, all this graphical and interactive interfacing would have little use in sound analysis and synthesis application [see Chowning et al., 1975]. New techniques are constantly being developed to extend the domain of sounds that can be handled. Some of the more recent discoveries include the use of the phase vocoder for music analysis, a unification in nonlinear synthesis techniques, and new methods of reverberation.

NATIONAL SCIENCE FOUNDATION

WASHINGTON, D. C. 20550

MAR 2 1 1978

Dr. Richard Lyman, President
c/o Office of the Research
Administrator
Stanford University
Stanford, California 94305

Proposal/Grant No.
MCS77-23743

Dear Dr. Lyman:

It is a pleasure to inform you that $80,509 is awarded to Stanford University for twelve months support of the initial level of effort of the project entitled, "An Interactive Graphical Environment for Musical, Acoustical and Psychoacoustical Research," as outlined in the above-numbered proposal. The project has been approved on scientific merit for approximately two years of effort. This project is under the direction of John Chowning and Loren Rush, Department of Music. This grant is effective March 15, 1978 and unless otherwise amended, will expire on August 31, 1979.

An unfunded period of six months is included in the award period to provide flexibility in the performance of the project.

Contingent on the availability of funds and the scientific progress of the project, it is our intention to continue support at approximately the following level:

2nd increment $85,949

The request for continued support should be submitted in accordance with Section 253 of NSF 77-47.

This grant is subject to the provisions of FL 118, "Grant General Conditions."

The cognizant NSF Program Officer is Damon Barrett, Intelligent Systems Program, Telephone: Area Code 202-632-5743.

The cognizant NSF Grants Manager is Aileen Suie, MPS/MSS/SS Branch, Telephone: Area Code 202-632-5940.

Sincerely yours,

/s/ William B. Cole, Jr.

Grants Officer

Enclosures

PRINCIPAL INVESTIGATOR/PROJECT DIRECTOR COPY