Larry Polansky
Bregman Electro-Acoustic Music Studio
Department of Music, Dartmouth College
Hanover, New Hampshire 03755 USA
larry.polansky@mac.dartmouth.edu

Live Interactive Computer Music in HMSL, 1984–1992

This is a historical document of my work with live interactive computers and performers in the computer music language Hierarchical Music Specification Language (HMSL) from 1984 to 1992. In this article, I describe eight pieces that were experiments in live interactive computer music, each of which was performed at least several times. The pieces described here are: בראשית *(B'rey'sheet)*; *17 Simple Melodies of the Same Length*; *Simple Actions*; *Cocks crow ...*; *Horn*; *3 Studies ...*; *Slippers of Steel ...*; and *The World's Longest Melody*.

Recent commercial and user-friendly real-time object-oriented music programming environments like the popular Max have helped to make interactive environments more accessible. I hope that the descriptions of my earlier works in this article will give composers and performers ideas for possible experimentation.

A Brief History of HMSL

David Rosenboom and I wrote HMSL in 1980–1983 at the Mills College Center for Contemporary Music (CCM). We completed version 1.1 in 1983. Phil Burk became the third author in 1984. In 1985, HMSL (versions 3.0 and higher) was released for both the Apple Macintosh and Commodore Amiga computers and has been used by many composers since then. It is a general-purpose programming language, not an application, and is a somewhat difficult (though powerful) environment to master because the user must be—or must become—a relatively skilled programmer. HMSL also grew out of interaction with graduate students and colleagues and was created in response to other computer music languages, past and contemporary (like Don Buchla and Lynx Crowe's Patch-IV, David Rosenboom's FOIL, David Anderson and Ron Kuivila's Formula, and Daniel Kelley's MASC, to name just a few).

HMSL has been used intensively by many experimental composers, who have become intimately involved in the development and evolution of the language through their own work and research. This article is an overview of my own work only, not the broad range of music in which HMSL has played a part (see the Appendix). HMSL has been fully documented elsewhere: in the HMSL manual itself (Burk and Polansky 1993) and in a long theoretical overview (Polansky, Burk, and Rosenboom 1990).

HMSL provides a real-time interactive environment for advanced experimentation in computer music composition, performance, and theoretical experimentation. It was intended to provide flexible data structures for the design of hierarchical musical forms, and it incorporates ways of embedding procedures, functions, and stimulus-response algorithms within the hierarchical and polymorphous scheduler. The fundamental design of HMSL has remained fairly consistent since 1980. The most significant extension was Phil Burk's addition of the object-oriented programming environment in 1984.

HMSL is designed with three central criteria in mind: extensibility and experimentation, real-time interactive performance, and compositional intelligence. The language can be extended and modified by individual composers, and it supports a wide range of musical and conceptual experimentation. No musical idea is deemed too outlandish, strange, or peculiar to be precluded by HMSL's design. Our unofficial motto has always been "Everything is possible, but some things may not be easy." HMSL contains a complex, interactive hierarchical scheduler. The user, composer, or performer can define the stimulus/response environment in any way desired. The data structures and concepts of HMSL allow for real-time applications and support experimentation in the use and design of compositional, performance, and improvisational algorithms.

Computer Music Journal, 18:2, pp. 59–77, Summer 1994

HMSL and Community Composition

Some of the motivation for writing and using HMSL comes from a definite desire to continually redefine the nature of composition and of performance itself. The computer music community is in an ideal position to extend the notion of composition. Information, software, and even musical data need no longer be constrained in a regimented and anachronistic fashion. Composition and performance are no longer unitary, individual-based activities in which one person claims, is the main user of, and receives sole credit for a work, idea, piece of software, etc. Musical ideas have always been free-flowing and, to some extent, a product and property of the whole comunity.

Work in HMSL explicitly acknowledges a common desire to engender aspects of community music making. The authors and users have a common interest in the evolution of group music making—collaborative processes that use technology in progressive, co-evolutionary ways, and a de-emphasis of certain aspects of the composer's individual role. Now more than ever we are free to choose among various types of musical interaction, and the fluidity and portability of musical ideas as software have facilitated many interesting, exciting, and unusual musical situations of which our work in HMSL has been just one example. The technology and social/artistic evolution of the computer music community can (through projects like HMSL, Formula, NetJam, and others) approach the level of musical interaction of, for instance, a community of improvisers.

Aspects of My Own Work in HMSL

As both developer and user, I often find myself working on a new piece that significantly extends the capabilities of the language while simultaneously redesigning the language itself. This experience has been common to other advanced users. Version management consequently became a significant issue for these pieces—an interesting experience and one that often made me long for the good old days of pen and paper or even someone else's software (but not for very long). In the pieces described in this article, I have focused on several specific types of experimentation in HMSL.

Different Forms of Human-Machine Interaction

Each piece attempts to explore interactivity in a slightly different way. In each piece, the performer has a specific task or a different way of interacting with the music-making technology and communicating with the machine and other performers. The performance task attempts to introduce something unusual, new, and challenging for the human component.

Compositional and Performance Intelligence

Each piece approaches composition and creativity in a different way, exploring notions of determinacy and indeterminacy (through the use of stochastic algorithms and other procedures) in various compositional parameters, the co-involvement of the performer and the machine in the compositional process, and in many cases, abdicating compositional decisions to the machine.

Development of HMSL and a Community of Composition

Through these pieces I have tried to develop code and ideas that would be useful to other composers. Even when the actual code was not likely to be used by others, I frequently emphasized (through program notes, teaching, lectures, workshops, etc.) the technical and algorithmic aspects of these pieces—trying to make them an open book. My motivations were theoretical and ontological as well as purely musical. Each piece extended HMSL in some way. By "customizing" the language each time, I was consciously contributing to the language's evolution.

Low Hardware Overhead, Inexpensive Systems

All of the pieces involve a minimal, inexpensive, and portable computer music system. The success of the piece does not depend on the availability of expensive sound-producing hardware. The pieces needed to travel (most of them have been performed interna-

Figure 1. בראשית septimal scale for the trope/ cantillation melody.

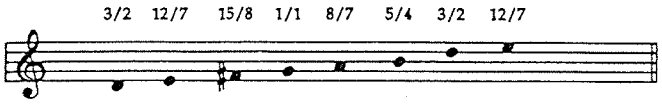
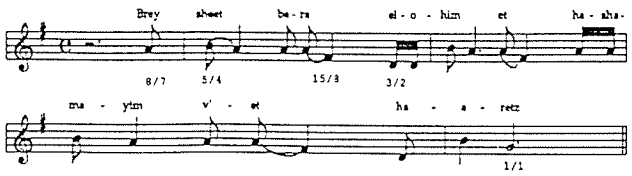


Figure 2. First few measures of בראשית cantillation melody, with tuning ratios and transliterated Hebrew text ("In the beginning, elohim created the heaven and the earth").



tionally), so the hardware configuration was kept small and easily modifiable. I was interested in deemphasizing hardware and the relationship between consumerism and music, a natural occurrence since the advent of MIDI. The interest of a piece should not be related to the economic means of a composer (something Ron Kuivila has jokingly and aptly termed "virtuoso consumerism"). It was a particular challenge to create interesting live pieces with a software emphasis using inexpensive, flexible, and rather humble-sounding equipment.

Morphology

Much of my work, including the design of HMSL itself, investigates morphology, or form. Although this is easily seen in works like *17 Simple Melodies ...*, בראשית, and *3 Studies*, my concern with morphology has taken a very different form in my live interactive work than in my HMSL-assisted scores, partly due to the interesting limitations of live interactive systems with regard to computational speed, predictive ability, and global knowledge.

De-emphasizing Timbre and the Use of Found Sounds

In these particular pieces, I am more concerned with structure, form, machine-performer interaction, and aesthetic/technological evolution than with sound itself. Computer music has most often focused on sound (timbre and synthesis) and avoided issues of structure, form, and musical and philosophical idea. The MIDI standard tends to separate timbre from event in interesting but often restrictive ways. Most of the pieces described in this article use real-time MIDI system-exclusive algorithms (as in *Simple Actions* and *Cocks crow ...*) and other real-time dynamic timbral ideas (בראשית) to expand the boundaries of the MIDI communication standard. (I believe they were among the first such pieces to do so.) Several, like *17 Simple Melodies of the Same Length* and *3 Studies*, deliberately avoid the notion of timbre entirely, treating MIDI instruments simply as found objects.

The Pieces

בראשית *(B'rey'sheet) (In the beginning)*

בראשית (1984; revised 1987, 1989) was one of the first pieces written in HMSL. בראשית is the first of a set of pieces called the *Cantillation Studies*, which are based on computer-aided morphological transformations of the 11th- and 12th-century Masoretic cantillation melodies for the singing of Torah (Shabat morning tropes). It was first performed at the Mills College CCM using the prototype version of HMSL running on a single-board Motorola MC68000-based computer driving a Buchla and Associates model 400 digital oscillator card. It was written for Jody Diamond, who had also studied the cantillation of the Torah, the melodies for interpreting the Biblical nuemes or tropes according to specific liturgical situations.

Each cantillation study is based on successive 17-verse sections of the Torah, named in traditional manner after the first few words of the text. The computer-generated melodic transformations are based on the tropes. In בראשית, the tropes are sung unadorned, and the pitch of the voice is captured by HMSL. Incoming pitches are debounced in software, and a lookup-table algorithm selects from a set of possible target pitches. The modal trope is sung in a just-intoned scale (Figures 1 and 2).

The text is the first 17 verses of the Torah, describing creation and the gradual imposition of cosmological order. With each verse, the statistics of the computer's musical response are changed. The computer "listens" to the melodies and generates its own

events based on what it hears and on where it is in the piece. There is a predefined trajectory of "computer attention." At the beginning of the work, the computer mostly ignores the voice, but its attention gradually and continuously increases until, at the end, it tries to follow the voice closely. This is achieved by constraining the amount of randomness from beginning to end. The piece begins in accompanying statistical chaos in all parameters and ends (it is hoped) in unison with the voice. One variable in the program, called VERSE-#, controls the degree of all change. It begins high (17) and ends at 0. A second performer is responsible for changing the value of the variable (telling the computer where it is in the piece) through a simple graphic interface (the Perform Table of HMSL).

The computer's sonic material is limited to *four sine waves*—a limitation based on my decision to use only the Amiga's local-sound but also one that is related to the text. The Amiga's four DMA sound channels are highly flexible in terms of intonation and timbre. A separate processor is responsible for updating the DAC outputs from a specified memory location, so high-level software can change the output waveform transparently and quickly without interrupting the sound. In בראשית, I was interested in dynamic, point-by-point time-domain waveshape modifications that were "orthogonal" to HMSL's higher-level morphological transforms. The four sine tables are treated as long melodies, and the wavetable modulations have the following parameters: amount of wavetable to be modulated, degree of modulation, type of modulation, and possibility of modulation.

All modulation is time-domain, and the algorithms are derived from the HMSL *shape class*, which has many methods for editing lists of musical data. For example, in בראשית, particular points in the sine table can be replaced with other points (a kind of spectral deformation), or portions of the table can be read out in retrograde, inverted, scrambled, randomized, and so on. The value of VERSE-# determines the percentage of the wavetable to be modulated (a lot at the beginning, none by the end) as well as the amount of deformation of single points from the sine table and the rate of change of the tables. Typically, at the beginning of the piece, all four sine tables are modulated at audio rates, as is the type of modulation itself (chosen by a large CASE statement). The modulation rate and degree decrease as the piece progresses.

One of the most interesting aspects of בראשית is that the computer tunes "on the fly," using a simple intonational trajectory to guide its tuning decisions. The trajectory begins in a complex 17-limit tuning space and ends in a simple 3-limit one (Pythagorean tuning). For each verse, the VERSE-# is the limit for the tuning space, and random tuning values whose prime factors are limited by VERSE-# are chosen for numerators and denominators of just intervals to the just intervals of the melody itself. For example, in the first verse, the computer might select a 51/50 (a small minor second to the 8/7 using prime factors 17, 5, 3, and 2) to harmonize the septimal major second (8/7) in the melody, resulting in an absolute "minor-third" interval of 204/175 (approximately 266.5 cents). This pitch might only last a fraction of a second because in the beginning of the piece, everything changes rapidly, and the tuning process happens simultaneously in all voices. The computer is always in harmony with the voice, but the nature of the harmonic space is stochastic and has a primitive, primarily statistical functionality. At VERSE-# = 7, for example, only prime values of 7 or lower are allowed, and by VERSE-# = 3, an organum-like effect is achieved by the 3-limit—intervals are always some type of just perfect fifth.

The tuning algorithms used here are what I have called *paratactical tuning* (Polansky 1987c). All tuning is done in real time in response to input, and no concept of scale is invoked. The pitches used are generated by the machine in response to the voice, not chosen from a set of pitches. This is similar to the way a chorus or string quartet would dynamically tune to itself over the course of a piece, except that the rules are extremely primitive. A natural extension to these algorithms would be to introduce some sort of voice leading or functionality rules, as Phil Burk has done in his adaptive tuning software for HMSL. In בראשית, this was not part of my aesthetic intent.

By delaying or anticipating change in relation to the singer, the mood and macro-rhythm of the per-

formance is varied tremendously. Jody Diamond and I have found through many performances that even though both of our tasks are highly constrained and even somewhat minimalist in performance aesthetic (she just sings the trope, I just change a few variables at more or less specified times), the piece sounds and feels very different depending on our respective timings. Our growing sensitivity to these details made performing the piece enjoyable, exciting, and surprisingly unpredictable.

17 Simple Melodies of the Same Length

17 Simple Melodies was written for composer-performer Daniel Goode in 1987. I wanted a truly portable computer music piece that was more than simply a sequence of note information. It was intended to be a live, interactive, and flexible computer piece for any melodic performers who could generate MIDI pitch data—a piece in which their own equipment would be used. This performance configuration was motivated to some extent by the fact that more and more extraordinary and experimental performers (like Daniel Goode, John Oswald, Ann LaBerge, George Brooks, and others who have performed this piece) were starting to work with MIDI equipment in sophisticated ways: programming their own sounds, working with various input devices, and so on. What tended to be available to these artists was commercially-oriented software that limited experimental resources for composition and performance. I intended *17 Simple Melodies* as an example of an unusual, interactive, and somewhat intelligent piece for computer and performers of conventional instruments. By supplying only a musical form as a kind of "black box" software engine and leaving the timbres of the MIDI synthesizers as well as all the melodic material completely up to the performer, I was inviting what I hoped would be an evolutionary collaboration.

From its inception, *17 Simple Melodies* was distributed to performers on disk with instructions. The disk contains a compiled version of the piece and the source code. The score for the piece is the code itself. Performers were encouraged to modify the source code in any way they chose or to use it as a model for making their own pieces, which would be some sort of collaboration with me.

The form of *17 Simple Melodies* is meant to be a kind of good-natured parody of "classical" artificial intelligence; data are gathered by a kind of "perceptron" (17 melodies of 17 notes each), the data are sorted (the intelligence), and finally, the data are replayed as a kind of simulation. In Section 1, the performer plays 17 melodies, each 17 notes long, signaling the computer through the computer keyboard when ready to play the next melody. (In one performance, John Oswald put the keyboard on the floor and entered these signals with his foot.) The computer informs the performer when it thinks it has heard 17 notes, by displaying a message on the screen. Glissandi, rapid arpeggii, multiphonics, noises, and so on will confuse it, usually in interesting ways. The salience of the original input melody in the computer's recording greatly depends on the type of material the performer plays and the settings of the pitch-to-MIDI converter.

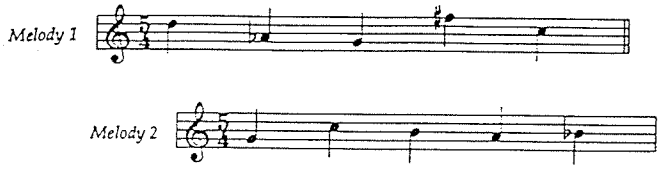
Section 2 is silent, lasting a few seconds while the computer sorts the 17 melodies into three independent lists, each ordered by some metric to the first melody played. All of the melodies are measured with respect to their similarity to the first melody by three different morphological metrics (Polansky 1987b) or melodic distance functions on melodies. The metric from which the three variations are derived is a simple version of what I have called the *ordered combinatorial direction (OCD) metric*:

$$d(N,M) = \frac{\sum_{j=1}^{L-1} \sum_{i=1}^{L-j} diff\,(sgn(N_i, N_{i+j}),\ sgn\,(M_i, M_{i+j}))}{L_m}$$

where N, M are two morphologies or melodies (ordered lists of numbers); N_i, M_i are the i^{th} elements of N and M; *sgn* is a contour function that returns a –1, 0, or 1, depending on whether or not the first element is bigger, equal to, or smaller than the second; *diff* is a binary comparison that returns 0 if the values are equal or 1 if not; and L_m is the binary coefficient of the length of the melodies (in this piece always 17) or the number of pairwise relationships. That is, if the melody is L notes long, then

$$L_m = \frac{L^2 - L}{2}$$

Figure 3. Musical staff notation for two sample melodies to illustrate the OCD metric.



(or, in this case, 136). L_m describes a "half-matrix minus the diagonal."

More simply, this metric sums the difference of corresponding cells (each containing a 1, –1, or 0) in the two L_m contour matrices generated by N and M (Friedman 1985, 1987; Marvin and Laprade 1987; Polansky and Bassein 1990). Each note in the melody is compared to every other note in the same melody—the contour relationship between the second and fifth notes is taken into account as well as the usual linear contour relationship between the second and third notes. The results of this internal network of comparisons form the L_m*-matrix*. If all cells of the matrix generated by N are equal to those generated by M, the metric is 0 and the melodies are considered to be the same in terms of their combinatorial contour. A value of 1 indicates that the two melodies are "as far apart as they can be" in terms of contour; all cells of the matrices are different. Figure 3 shows two simple five-note melodies. The metric for these is as follows.

Contour Matrix for Melody 1:

1	1	–1	1	(D>A♭, D>G, D<F#, D>C)
	1	–1	–1	(A♭>G, A♭<F#, A♭<C)
		–1	–1	(G<F#,G<C)
			1	(F#>C)

Contour Matrix for Melody 2:

–1	–1	–1	–1	(G<C, G<B, G<A, G<B♭)
	1	1	1	(C>B, C>A, C>B♭)
		1	1	(B>A, B>B♭)
			–1	(A<B♭)

In both melodies, the first pitch is lower than the fourth pitch (–1 in the third position of the first row), and the second pitch is higher than the third (1 in the first position, second row). All other aspects of their contours are different. For a five-note melody, L_m = 10, so the OCD metric for these two melodies is the number of corresponding cells that are different divided by 10, or 0.8.

Three lists of the 17 input melodies are made, according to this metric in the pitch and duration dimensions and an equally weighted average of pitch and duration dimensions.

	List 1 (pitch)	List 2 (dur)	List 3 ([pitch+dur/2])
	Melody-p	Melody-d	Melody-pd
(15 more sorted melodies)	...	...	...
	Melody-1	Melody-1	Melody-1

Entries Melody-p, Melody-d, and Melody-pd can be any of the 17 melodies (except the first). There is no necessary correlation between the three lists. They could all be identical or completely different, depending on the relationships between the similarities in the duration and pitch dimensions with the first melody. One could graph this as a two-dimensional space, with the first melody as the origin, and pitch-contour and duration-contour distances to the first melody as the x- and y-axes. List 1 is sorted by x-axes value, list 2 by y-axes value, and list 3 by a simple combination of both.

In Section 3 of the piece, the lists are played back simultaneously on three separate MIDI channels. The performer does nothing, creating a symmetry: in the first part only the performer plays, in the second one no one plays, in the third part only the computer plays. Several variables can be specified by the performer at performance time, including the average number of repeats for each melody, the probability of changing a MIDI preset at any given time in the piece, and the list of MIDI presets available to each channel. The performer has a tremendous degree of control over the piece by working with these parameters and customizing sounds and melodies. The duration of the piece can be greatly affected by the average number of repeats for each melody. The performer specification of MIDI presets adds a distinctive individual voice to the performances. One particularly interesting performance by Daniel Goode used simple sinusoid-like presets for all voices, which blurred the linear polyphony into a more "harmonic" texture.

17 Simple Melodies was both a philosophical and a technological experiment. My intent was to make the simple and straightforward form emerge as the salient feature. There is a conceivable, though extreme, description of the piece, and of pieces like it, that says that hearing it is to some extent irrelevant because it purposefully avoids the decision about what it should sound like. This is, of course, at odds with the prevailing notion that music is, at its most fundamental definition, in some way inextricably tied to sound (I do not believe this to be necessarily true) and also against the more purely fashionable idea that if a composer needs to explain a piece, it cannot be very good. *17 Simple Melodies* is heard, but it is also in some sense pure explanation, both from the composer to the machine and the performer and from the performer and the machine to the audience. I was not so interested in removing sound from music as in refocusing the attention of the performer and audience; the particular timbres might be heard to be irrelevant; there is no great skill or compositional intelligence required to choose a MIDI preset. Some listeners react to electronic music almost exclusively on the basis of timbre. In *17 Simple Melodies* this is like reacting to the color of the chair you are sitting in while listening to the piece—it's an important part of the experience, but not one that I wish to have any part in determining. As a result, this work has confused, angered, and, I hope to some degree, interested many who have heard it and heard about it.

Simple Actions (for Daniel Kelley)

Simple Actions (March 1987) was originally a work for solo performer and computer, using only a mouse for input. It was first written on the Amiga, using only its local sound-producing capacity and an inexpensive reverb/delay unit. The most important aspects of the piece are described in the score *Distance Musics I–VI* (Polansky 1987a):

> [The performer/programmer] designs several very simple ongoing musical events, called "actions." The criteria for simplicity is something like, "the sonic process may be completely described in about one sentence." Thus, glissandi, crescendi, simple stochastic melodies (or simple stochastic generation of parameters in any dimension) and random waveform generation, are all examples of "simple actions." Actions ... can be turned on or off, but once on, they execute repeatedly until turned off....
>
> Actions ... share data as much as possible. If two actions have an 'idea' in common, they use the same data. For example, both a glissando generator and a random interval generator would likely need a variable for "previous pitch" (so that the next pitch could be derived). This variable should be shared by both actions [so] there is as much "parameter passing" ... as possible. Another way of stating this is that each action's parameter passing to *itself* should be interfered with as much as possible by any other action.

I envisioned the piece as an ecology with many little musical organisms, which composer Robert Marsanyi has called "critters," wriggling around on a kind of sonic Petri dish, tripping over each other, altering each others' paths, and in general, creating a complex, unpredictable and hard-to-control musical result. *Simple Actions* is based to some extent on Minsky's "society of mind" concept—that complex intelligences are often created by difficult-to-predict interactions of many simple intelligences (Minsky's "agents") that share a certain informational ecology. Minsky's concept was part of the inspiration for the simplicity of each action's design.

Functioning only as an improviser, the performer has an unusual task. Actions may be turned off and on, voices may be enabled and disabled, and variables may be modified (like range and speed of execution). The piece is not "playable" in a way that might be called "musical" in the usual sense of the word. The performer cannot directly "cause" a particular sound or event, only classes of events and textures. There are hundreds of possible simple interacting processes, or "critters," and it is often nearly impossible to tell what is causing certain effects. The performer often becomes an observer, trying to figure out what

is going on and how to shape it toward something else.

The first version used the Amiga's four local-sound channels with relatively simple timbral and musical processes. These were combined with some unusual timbral events, such as frequency-modulating large blocks of the computer's memory with itself at some phase increment, a process lifted directly from my piece *Mod.Mania*. Later versions of the piece added MIDI in the form of a system-exclusive library written for the Yamaha FBO1, which allows for real-time control of FM voice parameters, such as low-frequency oscillator (LFO) rate, LFO depth, LFO shape, algorithm, and modulation index, without sending MIDI preset changes. As in the later work *Cocks crow*, MIDI "note-on" events are avoided almost entirely. Actual frequencies are specified, and all voice parameters are changed rapidly in real time. The notion of "preset" becomes meaningless. All parameters are modulated by the computer as fast as one wishes. The DEP-5 and the Amiga local-sound are controlled from the screen in similar ways. To the best of my knowledge, this was one of the first such uses of this kind of real-time interactive MIDI system-exclusive implementation. The FBO1 was chosen early on despite its relatively low fidelity because its system-exclusive implementation was well documented and can be controlled in a way that one company now refers to as "dynamic MIDI".

After performing the piece solo many times, I expanded it to include other performers. I wrote more "simple actions" that integrated the pitch and loudness of an instrument into the sonic ecology. Although the performer was not directly affected by the data of the other actions (except aurally), the instrument's input data were shared by as many other actions as possible. A year or so later, I began to work with the human voice in a similar fashion and performed the piece with sound poets Chris Mann in Australia and Paul Dutton in Canada. In these versions, Mann and Dutton contributed their own texts (Mann read a previously written one, and Dutton improvised sound poetry in performance). This is the version of the piece that I now prefer, perhaps because I was beginning to get lonely improvising with just a mouse and a computer screen. After more than 20 performances of the solo version, I was beginning to get bored and was finding to my dismay that I could control it. The addition of text and text-sound gave me a new sense of freshness and interest in the work as well as the joy of improvising with a fellow performer.

Simple Actions is a deliberate attempt at musical and cognitive co-evolution of myself and the machine. Like *17 Simple Melodies*, it is only a beginning toward a new form of musical and performance intelligence. The nature of the piece attempts to obviate traditional notions of performer creativity, cleverness, drama, and "musicality" and forces me to rethink the very nature of the composer and compositional, creative, and perceptual intelligence.

Cocks crow, dogs bark, this all men know, but even the wisest, cannot tell, why cocks crow, dogs bark, when they do (with John Bischoff and Melody Sumner)

Cocks crow (1987, 1988) is a live improvisation for three performers and computer, written after *Simple Actions*. It incorporates some new elements into the live interactive environment, such as more sophisticated use of HMSL's hierarchical data structures to determine large-scale forms in real time, more developed use of the real-time system-exclusive MIDI code, including audio rate control of a Roland DEP-5 signal processor, use of computer-generated real-time notation in the form of on-screen text commands to the human performers, and the use of non-electronic sounds, including the voice.

The title is from a Thomas Merton translation of a Chang Tzu poem. It describes my intent in the piece: things happen, and they happen for some reason, but there is really no way to predict or know when or why they happen. The entire piece is an HMSL *collection*—a class of HMSL objects that "know" how to play their component parts. Collections have an internal intelligence, called a *behavior*, for deciding when and how to play components. The behavior is user-defined and can be complex, as can the components of the collection itself. A collection can also contain other collections with equally complex intel-

ligences. In this piece, the main collection decides when specific sections will begin, what instructions to give to the performers, the length of each section, the overall length of the piece, what kinds of parameters will be active during each section, and other things pertaining to the instruments and sounds themselves.

The first version of the piece was performed at the Mills College CCM in June 1987 by Amy Neuberg, Jarrad Powell, and me. HMSL controlled the DEP-5, an FBO1, and some Amiga-generated sounds. At the time, the DEP-5 was one of the only low-priced signal processors that could be controlled in real time via MIDI system-exclusive commands. Its power has now been surpassed by other equally inexpensive and flexible MIDI devices. Neuberg and Powell sang and played into microphones small sound-making objects (like children's toys and small percussion instruments), which they were asked to bring to the performance. Their sounds were fed through the DEP-5, which was controlled rapidly and unpredictably by HMSL. Each of us had a monitor that displayed information about the length of the current section of the work and instructions such as "Jarrad, play quietly," "Amy, make loud sounds," and "Larry, don't do anything in this section." HMSL chose from a large number of these kinds of instructions for each section. The visual aspect of the piece was unusual. The performers sat behind video monitors with only their faces showing in the light of the screens. There was no way to connect the sounds heard with the way they were produced.

My screen displayed the same information as the other screens, as well as the HMSL *shape editor*, which allowed real-time control of the DEP-5 via HMSL's system-exclusive library. Each section of the piece chose a DEP-5 algorithm that was some combination of its signal-processing algorithms. I made a DEP-5 "software instrument" (in HMSL's virtual device interface, or VDI) that used nine dimensions of musical data in time. Each dimension represented a specific DEP-5 algorithm parameter. This data could be edited quickly in real time via the shape editor. Figure 4 illustrates a typical shape editor display, showing the chorus depth dimension. I could play the DEP-5 by drawing trajectories, or *shapes*, in time

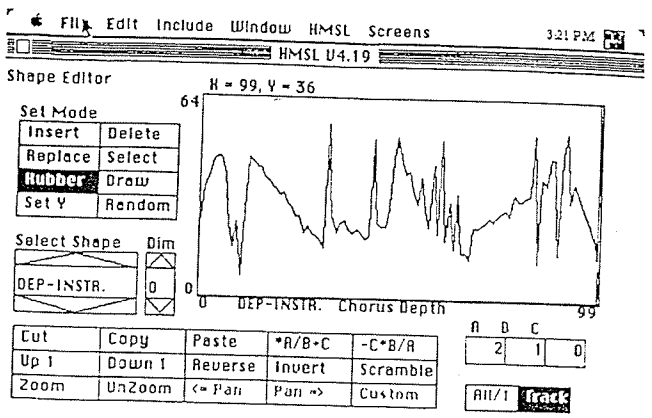


Figure 4. Shape editor screen from HMSL, showing one dimension of the DEP-5 instrument for Cocks crow.

for each parameter. The interaction between the sounds made by Neuberg and Powell and my HMSL/DEP-5 improvisation in HMSL further complicated the sonic output.

I also made several real-time modifiable "instruments" for Amiga local-sound and the FBO1 that augmented the "naturally" produced sounds. The local-sound instrument is called a chorder, an HMSL software instrument that reads five-dimensional shapes with duration, fundamental pitch of a chord, average loudness of the chord, harmonic complexity of the chord, and on/off time ratio for the duration of the chord. Harmonic complexity is defined as the height of a pitch in the harmonic series of the fundamental. The chorder can be played in real time by changing any one of its parameters. The FBO1 instruments were similar to the ones in *Simple Actions*. It is used like a computer-controlled analog synthesizer; no MIDI "note-on" is ever sent to the device.

This first version of the piece was performed once more, at Roulette in New York City, with Philip Corner and Daniel Goode. The next (and final) version of the piece had a personal motivation. In July 1988, I left for Indonesia for more than a year to assist Jody Diamond in a Fulbright-sponsored survey of Indonesian experimental music. Before I left, I did a set of "goodbye" concerts in San Francisco (at the New Langton Arts Center) and New York (at the Experimental Intermedia Foundation) in which I collaborated with many of the Bay Area artists who had been important to me through their friendship and work. Two of these artists were poet-publisher-per-

former Melody Sumner, who founded and directs the important publishing company Burning Books, and composer-performer John Bischoff, an important pioneer of live computer music. I decided to modify the structure of *Cocks crow* to make it a collaboration that incorporated the ideas and work of all three of us.

This version of the work used the same electronic sounds as the first, and all sound ran eventually through the signal processor. Instead of playing "found" sounds from small objects, though, Sumner and Bischoff participated fully in the computer network. My machine still functioned as a kind of supervisor, but its real-time instructions consisted of computer data sent to Bischoff's machine to be processed and text instructions sent to Sumner's screen telling her what and how to read. Sumner's 17 instructions pertain to how loudly she reads the 17 sections of her text:

Remaining without change
Rising steadily
Rising, then falling
Rising, then rising more slowly
Rising, then steady
Steady, then rising
Rising with continued acceleration
Steady, then falling
Rising, then rising more quickly, then steady
Falling steadily
Falling, then falling more slowly
Sinking or falling with increased intensity
Falling, then steady
Rising, then falling, then rising
Falling, then falling more
Rising, then falling, then rising, then falling
Falling, then rising, then falling

The text (Sumner 1988) was created especially for this piece in response to my description of the work as 17 unpredictable "sonic state changes." (Parts of the text were from a work entitled *Arms and Armour*, and another fragment was taken with revision from *Weather* by Gayle Pickwell). "Each of the 17 sections depicts uncertainty in a dramatic situation, superficially, of course, each depicts a change in season or weather" (Sumner 1989). One of Sumner's texts, which describes barometric pressures, consists of the above instructions. The following are excerpts from the text; the first is a complete section, the second an excerpt from another section:

> He stands looking out the glass doors toward the sundecked expanse of lawn tending to brown in patches and sporting a few ripe dandelion manes. The body on the young man is lean and tense. His feet are small and perfectly shaped and the skin all over his body is like baby's flesh. The woman loves the young man's feet. The woman loves his body. She doesn't know what to do with him.
>
> My dear friend,
> It has been 90 degrees here for a week. Humid. Do your fingers seem to stick to things? I'm not talking about love. I *know* you can't stand another four years of Latin and Greek. I'm not overly fond of it myself. The game is called highway and it ends when you arrive. Summer would be so much fun with you. We could chop onions and go to the store together without feeling like thieves. I liked my fortune too—though I wonder if you are ever really getting what is sent. There have been some changes: I moved physically and spiritually. (Through the medium of inconstancy we sometimes escape our fate.) This next part is personal. Please read it carefully. It has something to do with the way you feel: You said, "I can imagine us moving the day's pleasures along at a brisk pace ... peering at our dandelions while the broken mower festers in the evening." Your letter was painfully uplifting. I am so depressed. The flowers in the well are beautiful. My clock just broke. May our friendship be eternal.

In each section of the work, Sumner's computer monitor displayed the number of the text she was to read. Pitch and loudness information from Sumner's voice was read by my computer and sent to Bischoff's computer via MIDI. Every time my computer decided to start a new section of the piece (and displayed its length so that Sumner could time how fast she read), numerical "presets" were passed to Bischoff's machine so that he could alter global sonic parameters. His computer ran a version of the software for his work, *The Curve Behind the Line*. The

Figure 5. The performance configuration for Cocks crow.

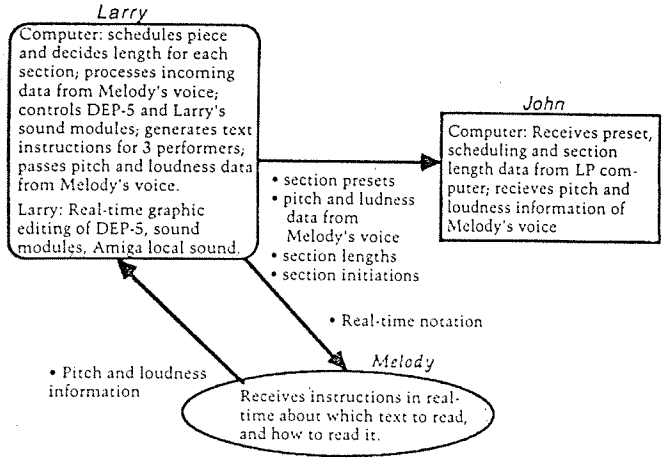
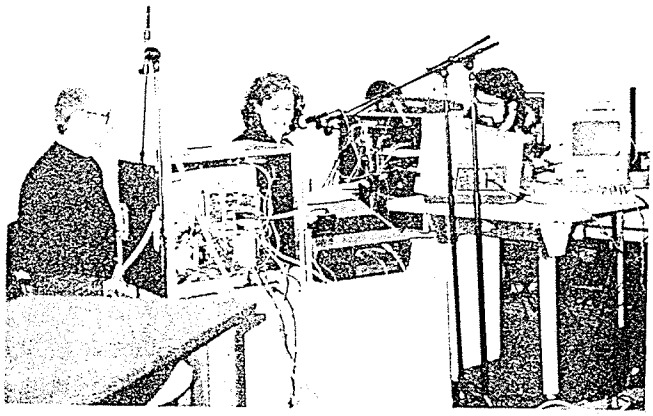


Figure 6. John Bischoff, Melody Sumner, and Larry Polansky performing Cocks crow *at the New Langton Arts Center in San Francisco. Photo by Jody Diamond.*



following are excerpts from a description of this software (Bischoff 1988):

> Eight real-time processes trigger a counterpoint of eight independent rhythmic lines, each one regularly altering its neighbor's tempo at distinct points in its own rhythmic behavior. Each process is represented by a separate channel of MIDI activity; ... eight curves (generated using 1/f numbers as successive goals) are carried on simultaneously, each one advancing step by step. The unfolding of all eight is the activity upon which the musical action of the piece is based.... Each channel has three states: stationary, ascending and descending.
>
> For this version (*Cocks crow*), the pitches played by all eight channels are read from an eight-note buffer which is continually fed by an incoming MIDI note stream. This stream originates from a MIDI pitch follower on the speaker's voice (Melody) and is passed through and modified by the other computer system (Larry) before reaching my input. The loudness of each note varies under direct control of the curve for each channel; the current step in the curve is used as the MIDI velocity byte for the note.
>
> On receiving a program change message (from Larry's computer) at the start of a new section, a new ictus table (which controls the rhythms of each channel) is made current and all down counters are initialized (values that represent the fundamental ictus of each channel) with a value from the new table. This has the effect of instituting a new unified tempo for all channels and synchronizing them. In addition, half the channels are assigned a new voice timbre. Both effects help to create the feeling of a new, unified perspective at the commencement of each new section.

Cocks crow is a complicated and somewhat unpredictable network of two computers, voice, and digital signal processor (Figure 5). HMSL "scheduled" the whole piece, determining at the beginning of each section which text Sumner would read, how she would read it, what parameters would be passed to Bischoff, and what parameters my own sound-producing devices would use. Figure 6 shows the three of us in a performance in San Francisco.

Bischoff and Sumner were completely responsible for their own parts of the piece. Given the same "architecture," two other performers would create very different works. *Cocks crow* is a kind of artistic parallel processor; there is a simple, well-defined communication protocol between the three components, but they all function independently. The piece goes out of its way to "not determine" what will happen and to set up a system that is so interrelated that the result is often either unintelligible, static, or simply out of control. My intent was to implement a technology that would, to some extent, do what musicians already know how to do: use a simple

improvisational structure and set of predefined performance relationships to produce strange, sometimes beautiful, sometimes awkward and "non-musical," but, I believe, fundamentally new music and sounds.

Horn (for Chris Bobrowski)

Horn (1990) is for French horn in just intonation, computer, and two Yamaha FBO1s, using 16 preset sine-wave voices. The HMSL system-exclusive library is used to send the FBO1's just pitch ratios for each note. *Horn* is one of the few live *non-interactive* works I have written in HMSL. The only communication between performer and machine is the computer's real-time display of "how far along the piece is." The piece is composed in real time by HMSL. *Horn* is a realization or "orchestration" of an earlier piece called *Psaltery* (Polansky 1990), which is based on a continuous modulation between three closely related harmonic series.

The harmonic scheme is a simple ordering of the harmonic series up to the 17th partial by a measure of prime complexity, as shown in Table 1. According to this scheme, harmonics enter in the following order: 1, 2, 4, 8, 16, 3, 6, 12, 9, 5, 10, 15, 7, 14, 11, 13, 17. The primes 2 and 1 are assumed for the higher partials. That is, the 6th and 12th partials are only considered to be powers of 3 in this ordering. The three series are related as I:V:III (referred to by Roman numerals to distinguish them from particular partials, notated as Arabic numerals). *Horn*'s structure follows the harmonic progression of these three series. First, the 17 pitches of the first series, I (on F), enter individually in the above order. Next, they are replaced one by one by pitches from V (A), in a modulation-by-replacement algorithm described below. V is then replaced in the same way by III (C), and then III is replaced by I again, whose pitches drop out from the highest harmonic (17) until there is only the fundamental remaining. In other words, the piece has the harmonic form I-V-III-I. *Horn* is 17 minutes long.

The computer's modulation algorithm has two simple rules:

Table 1 Harmonic Scheme Used in *Horn*

Primes Used	*Harmonics*
{2}	1, 2, 4, 8, 16
{3}	3, 6, 12
{3, 3}	9
{5}	5,10
{5, 3}	15
{7}	7, 14
{11}	11
{13}	13
{17}	17

1. Pitches from the new series enter from the "top down," in reverse order of prime complexity (thus, the first pitch to enter is the 17th partial).
2. They replace the closest remaining pitch (in absolute frequency) from the old series.

Rule 1 implies that the new series will be more salient as new pitches enter, since difference tones and phantom fundamentals are implied by higher harmonics of the new series. Rule 2 tries to ensure that the modulation will be as "smooth" as possible. Because old pitches cohabitate with their replacement neighbors for the transition period, there is a great deal of close dissonance, beating, and other acoustical phenomena created by these series's crossing each other harmonically. The modulation algorithm for the horn part follows the same general principles but is modified slightly for purely orchestrational reasons (to make the pitches lie more comfortably in the horn's range).

The piece consists of stochastic arpeggii for the computer and horn. Each of the 68 measures contains a set of pitches that can be played by both. The software uses probabilities that favor stepwise melodies but also allows for skips and leaps. Probabilistic algorithms for dynamics, rhythmic variation, and whether or not to play are built into the software. When a new note is introduced, it is gradually accented more and more by both computer and horn, and the exiting pitch is played softer and softer, so that new pitches cross-fade with old ones.

Figure 7. One page of the score for Horn.

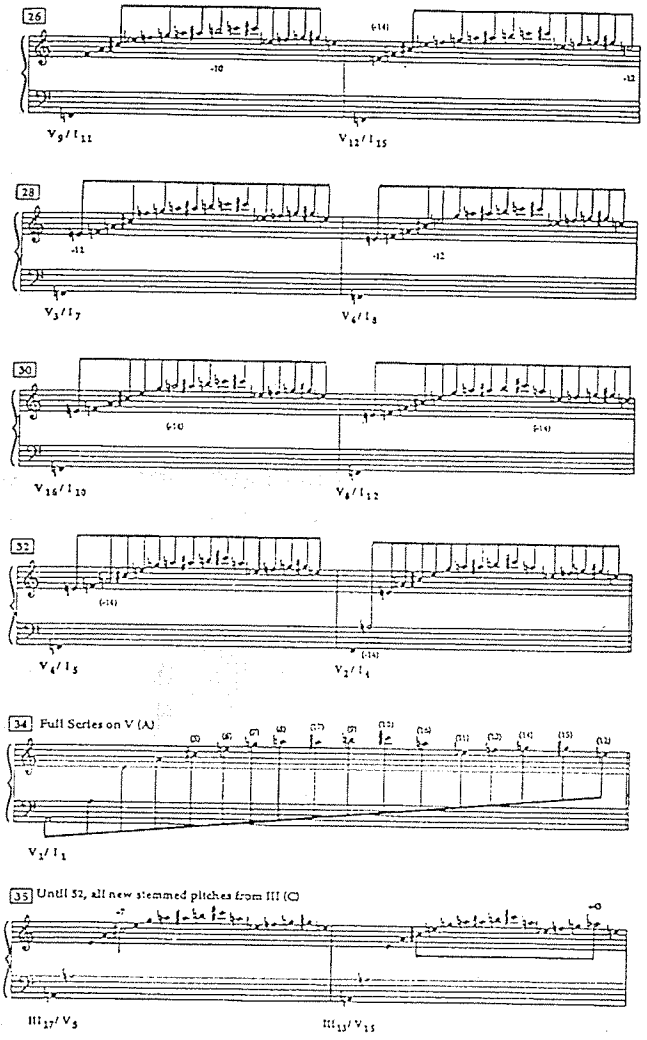


Figure 7 is a one-page excerpt from the horn part. At this point, the series V (on the major third) is about halfway through replacing I (on the fundamental). The fundamental (I) is written as sounding F (concert B♭). The numbers beneath each measure show which pitch is entering and which pitch is leaving. For example, in measure 26, V_9/I_{11} signifies that the ninth partial of V replaces the 11th partial of I. Stems up indicate the new series, no stems the old one. White notes are the entering pitches. Numbers in parentheses indicate cents deviations from 12-tone equal temperament. Measure 34 is the full series on V (parenthetical numbers in that measure show the number of the partial).

3 Studies (For the Downtown Ensemble)

3 Studies (June 1990) grew out of an experiment called *Duet*, for Nick Didkovsky and me playing electric guitars, in which we each improvised eight melodies into the computer. One of us played "source" melodies, and the other "target" melodies. Using some *mutation functions* (Polansky and McKinney 1991; Polansky 1992a) that are closely related to the metrics described above, the source melodies were gradually changed into the target melodies. I intended this piece to move away from "art" and "performance" and into a kind of pure experiment. It had very little (if any) theatrical interest and was even more austere in terms of sonority and musical form than *17 Simple Melodies*. However, certain things bothered me about the work; it wasn't simple enough, and the fundamental idea of the work, one melody gradually being changed into another according to some distance function, was not clearly communicated to the listener.

I decided to write a piece that was as explainable as possible, without sacrificing the complexities of melodic analysis (metrics) and generation (mutations). I liked the experiment form and wrote *3 Studies* to be illustrative of these concepts, simple, and elegant from a performance standpoint. Although the ideas of melodic distance that I was working with were simple and fundamental—measuring the statistics of contour and interval magnitude between two shapes—they were difficult to explain in words. In *3 Studies*, I wanted to use immediately understood musical elements as the medium for explanation. I was interested in translating a notion of similarity from one domain (melodic contour) into simpler, aurally obvious ones.

In *3 Studies*, the metric distance between melodies, improvised by the performers and heard by the audience, is manifest in simple musical ways. The form of the piece is simple. One or two performers play melodies into the computer. Another performer operates the computer via an HMSL graphic interface

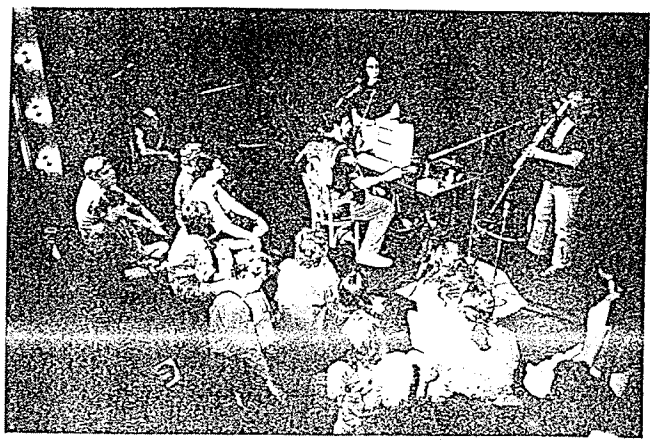


Figure 8. Iris Brooks and I Wayan Sadra, flutes, and Larry Polansky, computer control, performing 3 Studies at the Composer-to-Composer seminars in Telluride, Colorado, in 1990. Photo by John Fago.

that captures the melodies and plays them back either as source or target melodies. The piece uses any eight-voice MIDI synthesiser for output. Timbres are left to the performers. The performers and the computer operator communicate verbally and audibly so that the audience can follow along. For example, the computer operator says things like "Please play a source melody now," "Please play a target melody," "Please play a target melody that is closer to the source," "I'll now play back the source melody so you can hear it," and so on. When a target melody is played, the computer takes a metric on it and the current source melody. The source melody may change several times over the course of the piece and may be played back occasionally so that everyone knows what it is. The metric used is once again the OCD (as in *17 Simple Melodies*), so only the combinatorial contour of the source and target is compared, not absolute pitch, interval magnitude, or anything having to do with durations. The metric value, ranging from 0 to 1, produces simple and audible variations in an ongoing sonic fabric.

The three studies, called *Rhythm*, *Melody*, and *Harmony*, may be performed in any order. In *Rhythm*, eight MIDI voices are set to a common pulse and play continuously throughout the piece. The metric between the source and target determines the range of possible stochastic displacement off the pulse for each voice. In other words, if the source and target melodies are quite dissimilar, the eight voices will create a complex, non-synchronous fabric. The closer the target and source become, the closer the eight voices approach the pulse. The metric value also determines the harmonic fabric; for large values the eight voices will tend to select pitches from an extended altered dominant 13th chord, and for smaller values pitches from the related tonic major seventh are used. When the performers finally match the target and source exactly, all eight voices line up exactly on the pulse in a major seventh chord.

In *Melody*, eight simple melodies form the background, "manifesting" fabric. The pure form of each of these melodies is an ascending chromatic scale of about one octave. The metric values are directly linked to a contour mutation value for each of the eight voices. That is, the greater the metric value between source and target, the more the contours of each of the voices will stray from strict linear ascension. As in *Rhythm*, when the value is small, the voices lock into a unsion. I will not go into the details here of the mutation functions used; it is sufficient to say that each metric function has corresponding mutation functions that are like inverse metrics—they generate target melodies that are a given metric value from a source. I had used metric and mutation functions separately in various pieces before. In *Melody*, these two concepts were used together, and in fact linked, for the first time (and in real time!).

Harmony is my favorite of the three because it most fully achieves the aural simplicity of the manifestation of distance. The sonic fabric is eight sustained tones, which are constrained to be within a major third of each other (plus or minus two half-steps of a fundamental!). The metric values stochastically control the pitch bend of each voice so that the farther apart the source and target, the wider the eight-voice chord tends to spread. When they are very close together, the resultant chord tends to be a dense, microtonal cluster. When they are exactly the same, a unison emerges.

Like *17 Simple Melodies*, *3 Studies* is portable and distributable as software. It can be performed by anyone with a Macintosh, a simple MIDI synthesizer, and a pitch-to-MIDI converter. Figure 8 shows a performance of *3 Studies* at the Composer-to-Composer seminars in Telluride, Colorado, in 1990.