Hyperinstruments

A Progress Report
1987 - 1991

Professor Tod Machover
Principal Investigator

MIT Media Laboratory
Massachusetts Institute of Technology

January 1992

Contents

Introduction

I started the Hyperinstrument Project at the MIT Media Lab in the beginning of 1987. The initial motivations were both general and specific:

- To continue work that I had begun at IRCAM in 1978 concerning the adaptation of computers to the needs of sophisticated real-time musical performance.
- To adopt the MIDI standard for musical input and output so as to concentrate on the technical and scientific problems of collecting, analyzing, and interpreting musical data, as well as on the conceptual and musical questions regarding the implications, possibilities, and meanings of such systems and instruments.
- To produce a powerful (and relatively inexpensive) real-time performance system for use in the premiere of my opera *VALIS* in December 1987 that could allow a small number of musicians (two as it turned out) to equal or exceed the musical result of a traditional opera orchestra.

Since 1987, Hyperinstrument Research has explored these areas and more. This Progress Report is an attempt to give an overview of what we consider to be our most important results over these four years. The sections of this report are organized as follows:

- **Background**: The philosophy and intention of hyperinstrument research, exploring some precedents. Describes the performance system developed for *VALIS*.
- **Year One** (Sep 1988 through Aug 1989): The expansion of Hyperlisp, and the extension of hyperinstrument functionalities for my ensemble composition *Towards the Center*, including real-time rhythmic and timbral shaping tools, and the concept of the "double instrument."
- **Year Two** (Sep 1989 through Aug 1990): The extension of hyperinstrument principles to non-traditional musical instruments, especially hand gesture tracking, and the application of this technique to measuring left-hand conducting gesture and its use in my composition *Bug-Mudra*.
- **Year Three** (Sep 1990 through Aug 1991): Integration of string instruments into the hyperinstrument environment and development of the hypercello, including new techniques for gesture tracking, physical sensing, and acoustic analysis. Development of new hyperinstrument functions for my composition *Begin Again Again...*, and discussion of that work as a musical metaphor for the process of conceiving a new generation of performance instruments.

My principal colleagues in the Hyperinstrument Research Group over the past four years have been Joseph Chung (chief software designer), Andrew Hong (digital signal processing and software design), and Neil Gershenfeld (string instrument sensor design). Their invaluable contributions to the project are documented throughout this report. Other associated researchers have been Robert Rowe (computational music intelligence), Sharokh David Yadegari (digital signal processing), Mary Ann Norris (cognitive analysis of musical performance), David Sturman (hand gesture tracking), and Jim Davis (software design and real-time timbral shaping).

I would like to express our gratitude to the Yamaha Corporation for their generous support of hyperinstrument research since 1988.

Tod Machover

MIT Media Laboratory
January 1992

Background

Musically Intelligent/Interactive Performance and Creativity Systems

1.1 Introduction

Enhanced human expressivity is the most important goal of any technological research in the arts. To achieve this, it is necessary to augment the sophistication of the particular tools available to the artist. These tools must transcend the traditional limits of amplifying human gestuality, and become stimulants and facilitators to the creative process itself.

In music, we believe that the combination of machine-augmented instrumental technique, knowledge-based performance monitoring, and intelligent music structure generation, will lead to such creative enhancement [Machover, The Extended Orchestra, 1986].

We call such systems *hyperinstruments*, and believe that they will become the musical instruments of the future. More importantly, perhaps, by combining the traditional roles of performer and composer, and by adding the unprecedented power of machine intelligence, it is likely that experience with such systems will lead to a gradual redefinition of musical expression itself.

It is our conviction that these hyperinstruments will be an essential component in exploring the future of music in at least three fundamental ways:

1. By providing exploratory tools for professional musicians and composers.
2. By inventing instruments that give full access to existing but untapped resources.
3. By giving unprecedented creative power to the musical amateur.

We have been developing hyperinstruments since 1981 (at IRCAM, Paris) and have been working on our current system since February, 1987 (at the Media Lab, MIT). Already, this system addresses the main categories described above, and has been tested in laboratory conditions as well as on the concert stage. It has proven to be a powerful and reliable musical tool, and has allowed for musical results that would have been unattainable otherwise.

At the same time, our experience with this system has clearly pointed the way to improvements and future developments. This document explains our current achievements, and outlines our proposed research in Hyperinstruments for the next five years.

1.2 Basic Concept

The basic concept of a hyperinstrument is to take musical performance data in some form, to process it through a series of computer programs, and to generate a musical result. Usually, this chain of events is meant to take place in real time; however, there is no reason why the same system could not be used for composing, experimenting, or playing with musical structures and sounds in non-realtime.

The performer uses more-or-less traditional musical gestures, on a more-or-less traditional musical instrument. The instrument itself, however, is "virtual", since the computer system supporting it can redefine meaning and functionality at any point. The performer therefore chooses the definition of each key, note, movement or sound that he produces, and changing such definitions becomes a major part of his musical intention. Traditional musical performance always relates one gesture to one note, and therefore to one sound.

The hyperinstrument allows the musician to change the hierarchical level of his musical interaction, as he can momentarily zoom from playing individual notes to the "macro" level of influencing the genesis of entire musical structures, or on the other hand to the "micro" level of influencing the internal construction of sound itself. While this performance power raises the musician to the level of concert-time computer programmer, the integration of true musical gesture into the act assures that intuitive musical intent is transmitted throughout the system.

At present, hyperinstruments use MIDI generating input devices to collect musical data, since this resolves the problem of how to capture such data reliably. However, MIDI is extremely limited in bandwidth as well as in the scope of musical data that it covers (*i.e.* no information about sound itself, nor about physical movement such as string bow tension, speed and position, etc.). Future work will involve not only intelligent analysis of acoustic instrumental data, but the development of completely new instruments better suited to controlling the enormous resources of computer-generated music.

The "brain" of a hyperinstrument is the computer system that monitors musical data from the input instrument, redefines the controls on that instrument, and acts in accordance with its programmed musical knowledge. These programs range from the highly deterministic (one physical event is tied to a particular musical result), to the computationally flexible (musical result chosen by particular context, and modified according to the specific musical gesture received), to the truly intelligent (analysis by rule of performed music and the machine-choice of an appropriate response).

Category one becomes musically interesting when the library of possible musical results greatly exceeds what is generally available to a performer, but when the mode of access is designed to be conceptually elegant and easily memorizable.

Category two presupposes the computer's ability to "be aware" of what is being performed, and allows for the sophisticated interference of predetermined machine templates with interactive musical decisions.

Category three is the most speculative, and certainly the most interesting. An *agent*-type community of experts makes intelligent sonic and structural deductions from input data, and passes conclusions on to related agents that actually create musical results, such results ranging from spontaneous accompaniment to sonic enhancement of the original musical material. The performer may or may not be aware of which mode gradation is interpreting his actions at any specific time. While all three above-mentioned categories are accounted for in our present system, research in music cognition and the computational coding of machine intelligence will continue to enhance the power of category three.

The final component of the hyperinstrument is the musical result. The goal is to produce music of unprecedented subtlety, complexity, richness, and expressive power that is intimately, but not obviously, linked to the original intent of the performer/composer. Such music is the direct result of commands generated in part two of the system being sent to "music production devices". The basic principles that currently define the generation of this music have been developed over ten years of experimentation in the computer music field. This research has been governed by a conviction that a new unity must be found amongst traditionally distinct categories of musical language [Machover 1986]. At present the real-time musical devices at our disposal are controlled exclusively by MIDI data, and thus only limited access is available to music's diverse parameters. In addition, current technology makes it difficult to incorporate real-time treatment of the original instrument's sound and physical gesture components into the syntactic or synthetic musical result.

Our general goal is to encourage the development of such technology in the context of hyperinstruments so that the entire continuity of musical expression, from gesture to sound to structure, can be treated in a single context.

1.3 *Fusione Fugace*: A First Prototype

Our current research environment for hyperinstruments has been in development at MIT's Media Lab since the beginning of 1987. The work is based loosely on a previous system developed by Machover at IRCAM, Paris [Machover 1982]. This original system was conceived before the advent of MIDI, and was the first solo concert use of a real-time digital synthesizer [Machover 1984].

For the composition *Fusione Fugace*, (commissioned by the Venice Biennale Festival, and premiered there in September 1982) an intelligent interactive environment was designed around the 4X machine [Logue 1983]. A model of chamber music playing was adopted, and the system was manipulated simultaneously by three players/operators, each with distinct tasks. The basic principle was to create one complex program which would contain all of the possibilities for the composition and which would be loaded at the beginning of the performance and not be further modified during that performance.

Musical commands from an organ-type keyboard indicated to a host PDP-11/55 that a musical event was to be initiated. Each key stroke initiated an event on a programmable digital synthesizer (IRCAM's 4X machine). Each event contained a single musical note with 32 separately controllable (frequency, amplitude, waveform, phase, frequency jitter, etc.) oscillators. Two separate controllers, combining an array of buttons, potentiometers, and switches, sent higher level commands to resident software on the host, and providing all parametric information, such as the tuning of each of the 32 partials, envelope characteristics, spectral envelopes, individual wavetables, etc. Yet other controls established modes of transition between contiguous events, or over a series of events.

This system contained all of the data and structures necessary to perform the entire work. The software (written entirely in FORTRAN) was constructed in modular fashion, so that all the sonic possibilities of the piece could be installed and interconnected at will. These programs did not operate at all on the structural level as our present systems do; rather they served to constantly redefine and construct the actual orchestra at the musician's disposal. This was, in fact, a complex task, since the synthesis model used—complete additive synthesis—was far more refined and detailed, and therefore harder to manipulate intelligently, than is any current MIDI-accessible model.

In fact, a major research component of the *Fusione Fugace* project was the integration of concepts regarding spectral fusion and streaming developed by the psychoacoustician Stephen McAdams at Stanford and IRCAM [McAdams 1982]. McAdams algorithms were used to design sound generating modules that provided two categories of control that were unprecedented at the time, and, we believe, still unique (especially since they are so difficult to reproduce using current commercial synthesis technology): (1) the automated control of degrees of spectral fusion, determining, for instance, the relative independence or subjugation of individual partials to a general timbral surrounding; and (2) the passage of sonic material from one timbral state to another, creating a circular continuum from pitch to harmony to inharmonicity to noise, and back again to pitch/frequency.

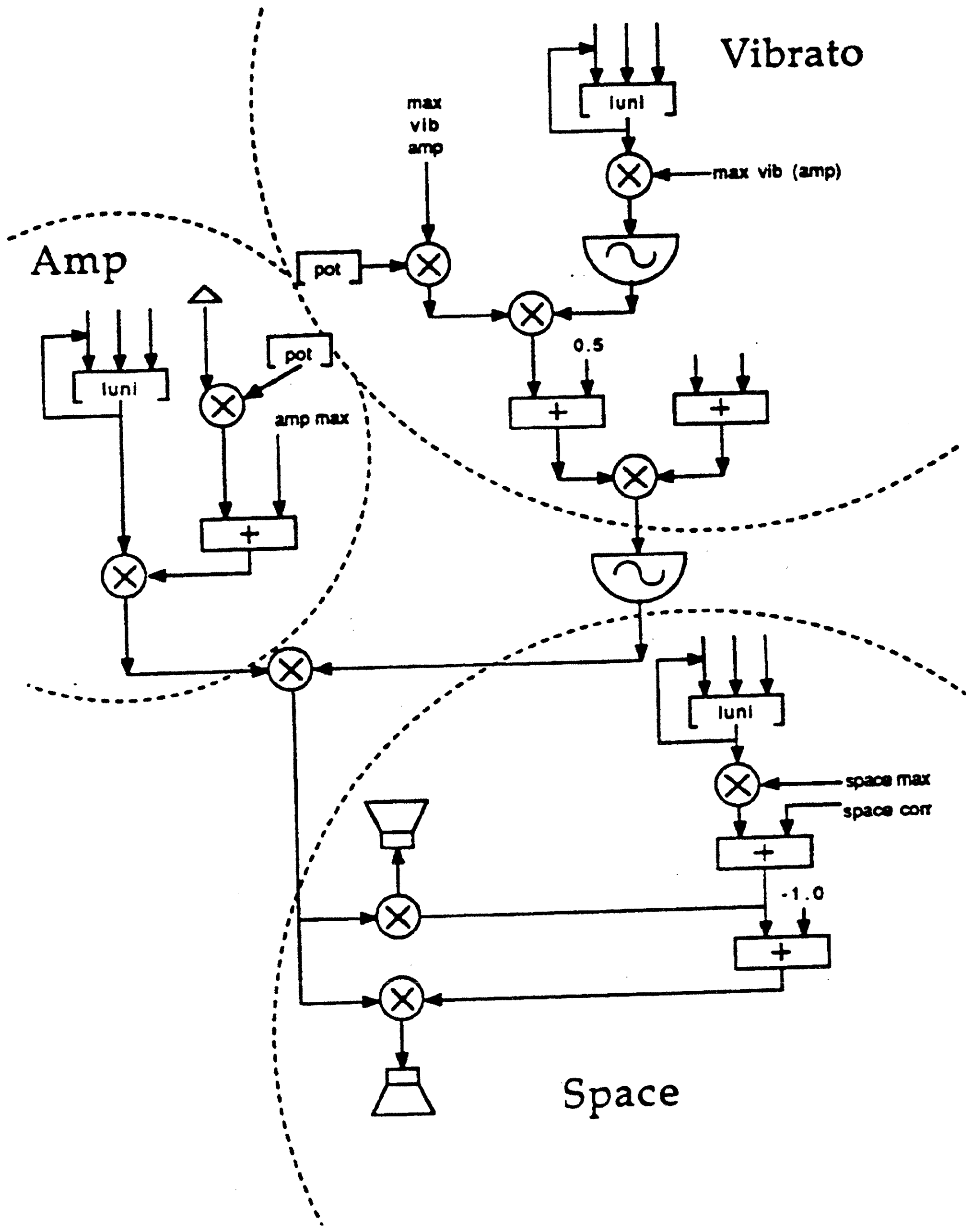


Figure 1.1: *Fusione Fugace* — 4X Real-time Additive Synthesis Algorithm

Since these sonic resources were so complex, great care was taken to allow a three-tiered control over all musical materials: manual, pre-determined by program but "shaped" manually, and fully automated. In the first case, all of the musical attributes were determined in real time performance, with no sonic or structural model to draw on. In case two, data was partially determined in advance but was allowed to be shaped or altered in the real time context. In the third, complete program or memory control was determined in advance, and a complex triggering system was implemented to allow easy access to a large library of events.

This ability to instantly redefine the relationship of performer to machine, and to combine modes of interaction at any one time, was one of the system's most powerful features, and has been improved and incorporated into our present system. Also, although the *Fusione Fugace* system was invented as a real-time concert performance instrument, it served for three years as a very powerful composition and conceptualization tool, and was used in several subsequent projects [Machover 1984].

The Present System: *VALIS* and Beyond

Our present system has been developed by Tod Machover and Joseph Chung. The basic elements were designed in Spring 1987, and implemented between June and November. This system was used as the almost exclusive musical resource for Machover's media opera, *VALIS*, which was commissioned for the tenth anniversary of Paris' Pompidou Center and premiered there in December 1987 [Machover 1987, 1988]. Since February 1988, Professor Machover has led a research seminar at MIT's Media Lab which has expanded and developed various Hyperinstrument concepts and techniques. The results of this research were presented in a two-day public seminar at MIT's Media Laboratory in June 1988.

For the past year, we have concentrated our activities on developing the "brains" of this system (the software that analyzes incoming music and generates musical results), and have spent relatively less time on input procedures or music output devices, since we have, for the time being, adopted existing MIDI standards for the latter two functions. This has allowed us to go somewhat further in the innovative domain of intelligent music processing than has previously been possible.

The current system revolves around two parallel input instruments: electronic keyboards (of the Yamaha KX88 variety), and a series of percussion controllers (Silicon Mallet, Octapad, etc.). MIDI is monitored from each of these two systems, and sent to two independent Macintosh II computers, each equipped with a significant amount of internal memory. Each Mac II runs a version of Allegro Common Lisp (Coral Software Company) a complete version of Common Lisp. Chung has constructed a system called *Real Lisp*, a real-time event processing language in which all of the system's musical functionalities are encoded. Once data is received by Real Lisp and analyzed and processed there, musical events—coded as MIDI data—are sent out of the Mac II's to a variety of digital synthesizers, samplers and outboard processing devices (see Figure 1.2).

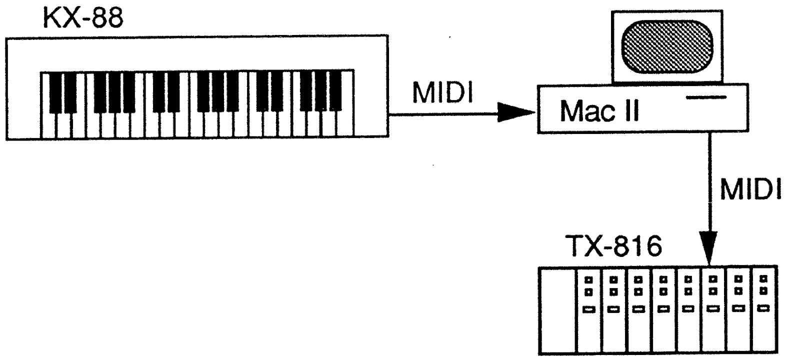


Figure 1.2. Typical Hyperinstrument Hardware Configuration

1.4 Real Lisp: A Real Time Midi Environment in Lisp

The Real Lisp system is based on a simple non-prioritized scheduler which essentially provides two facilities for hyperinstrument programmers: delayed function application and notification of the arrival of MIDI input. Real Lisp also provides various routines for processing MIDI data, and is linked in with ObjectLisp [Beyers *et al*, 1988], the object oriented programming extension provided by Allegro Common Lisp. A simple MIDI delay program written in Real Lisp is shown in Figure 1.3 below:

```
(defobject midi-delay)                           ; define a midi-delay object
(defobfun (exist midi-delay) (args)              ; called on object creation
   (have 'delay 50)                              ; delay in 1/100's second
   (add-midievent-handler self midi-in))         ; cause midi-in to be called
                                                 ; whenever there is MIDI
                                                 ; input
(defobfun (midi-in midi-delay) (event)           ; called on MIDI input
   (out event)                                   ; write event now
   (post delay self (out event)))                ; and again after the right
                                                 ; delay
(defobfun (out midi-delay) (event)
   (midiwrite event))                            ; write out event
```

Figure 1.3. An Example Real Lisp Program

When a *midi-delay* object is created, its *exist* function is called, and *add-midievent-handler* adds the object's *midi-in* function to the functions Real Lisp should call whenever there is MIDI input. A *midievent* handler always takes one argument, the *midievent*. A *midievent* is represented in Real Lisp by a thirty-two bit integer whose lower twenty four bits are the MIDI status byte and the two data bytes.

Once the object is created, Real Lisp will call *midi-in* for every *midievent* it receives from the MIDI controller. *Midi-in* does two things: it calls out which forwards the event to the synthesizer immediately, and it schedules *out* to be applied to the same event *delay* centiseconds into the future. When that amount of time has passed, regardless of what else may be going on in the system, the scheduler will apply *out* to the proper event. The result is a MIDI delay analogous to a digital or analog delay in a conventional sound effects processor.

The basic Real Lisp algorithm is as follows: Wait for either MIDI input or a clock tick. If there is MIDI input, call every function that requested notification via *add-midievent-handler*. If there is a clock tick, check if there are any function applications that were scheduled via *post* for the current tick. If so, apply those functions one by one. Arguments to a delayed function are evaluated when the function is scheduled in the scheduling object's environment. All times are expressed in 1/100's of a second (centiseconds).

1.5 The *VALIS* Hyperinstruments

The most complete instruments so far have been a pair of systems developed for Tod Machover's opera *VALIS* [Machover 1987], and were implemented in 1987. The two systems, a keyboard based system similar to Figure 1.2, and a percussion system which employed the Simmons Silicon Mallet and Roland Octapad as MIDI controllers, were built from essentially the same software. Object inheritance was used extensively to exploit similarities.

Keyboard Based Instrument

The basic software architecture of the keyboard based instrument is shown in Figure 1.4 below. The instrument has four major modes: *through mode,* which is simple play through on a specific timbre; *shadow mode,* a more sophisticated version of *play through* which allows generalized mapping of notes to MIDI channels; *arpeggio mode,* which generates arpeggiations of the chords being played; and *manual mode,* which generates polyrhythmic repetition of held notes.

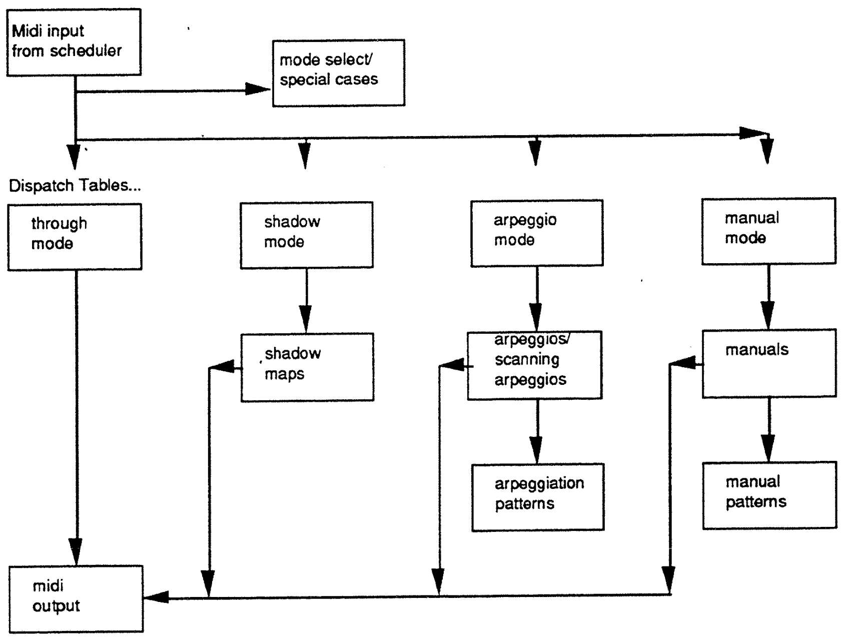


Figure 1.4. Keyboard System Software Architecture

When MIDI input arrives from the scheduler, it is fed into a dispatch table which directs the input to different functions in the instrument depending on the type and values of the MIDI bytes. The instrument has four dispatch tables, one for each mode, and the mode that the instrument is in is entirely determined by which table is active.

Mode selection is handled by a special master dispatch table which is always active and receives MIDI before the normal mode table. Modes are selected via the *program change* MIDI event which can also configure specific state within the mode such as timbres, MIDI channels, arpeggiation patterns etc. In addition, program changes can set up special case mappings where a function not normally associated with any particular mode can be inserted temporarily into the current table. This allows a configuration of the instrument to be almost entirely a certain mode, but with one or two keys that do something completely different such as trigger a pre-recorded sequence. These special case mappings can either be one-shot triggers which revert to the normal mode's mapping after a single use, or they can persist until another configuration is selected.

The system is programmed with software interlocks in such a way that mode switches and special case mappings are guaranteed to maintain consistency. It is impossible for a mode to get a *note-on* event and fail to get the corresponding *note-off.* This hazard can arise when a mode switch is requested while a note is still down. In this case, the mode switch will be deferred until all notes and pedals are released.

Description of Modes

Play Through

Play through simply forwards MIDI events along to the channels specified when the mode was selected. The functions in the dispatch table are very simple.

Shadow Mode

Shadow mode is a more sophisticated version of play through mode. A predefined *shadow map* which maps notes on the keyboard to MIDI channels is used to assign different timbres to different notes. The mapping is completely general so that any note can play to any combination of the sixteen available MIDI channels. Even though through mode is a special case of shadow mode where all the channels in the shadow map are the same, both modes are kept in order to maximize responsiveness in the common case where simple play through is all that is required.

Manual Mode

Manual mode is a more complex mode where held notes are repeated according to a pre-recorded set of rhythms and accents known as a manual pattern. The manual pattern is selected by the velocity with which the note was attacked. A typical manual pattern is shown in Figure 1.5.

```
#S(PATTERN VELOCITIES #(80 31 44 110 43 92 44 29 110 51 80 92 110 40
                        47 90 55 92 51 29 110 44 110 55 110 125)
           RELEASES #(30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
                      30 30 30 30 30 30 30 30 30 30 30)
           DURATIONS #(60 60 60 60 60 60 60 60 60 60 60 60 60 60 60
                       60 60 60 60 60 60 60 60 60 60 60))
```

Figure 1.5. Example Manual Pattern

In general there are a number of such patterns which are eligible to be selected. The system takes the number of patterns available and divides them equally into the 127 gradations of MIDI note-on velocity. When a note is played, the appropriate pattern is selected, and a manual object is created which plays the note with the velocities, release times and rhythmic values specified in the pattern. This object continues to play until the note is released (or if the sustain pedal is being held and the note is not, until the pedal is released). If the manual object gets to the end of its pattern, it starts over.

As an example, if a middle C were played with a velocity such that the manual pattern from Figure 1.5 were selected, the result would be as follows: produce a note-on on middle C with velocity 80; wait 30 centiseconds and produce a note-off on middle C; wait 30 more centiseconds (for a total of 60) and play a middle C with velocity 31; wait 30 centiseconds and write the note-off; etc.

Our normal organization of manual patterns is to order them so that the lower velocity patterns have longer durations while the higher velocity patterns are faster, forming a continuum of rhythmic values. Complex, yet controllable polyrhythms are achievable by a virtuosic performer.

Arpeggio Mode

The most complex mode is *arpeggio mode*, in which chords held by the performer are arpeggiated according to pre-recorded arpeggiation patterns. A sample arpeggiation pattern is shown in Figure 1.6 below:

```
#S(PATTERN NOTES #(0 2 5 0 2 5 1 3 5 2 2 5 0 2 5 4 0 5 2 2 4 3
                   5 0 3 1 2 5 0 4 2 5 4 1 2 5 3 2)
           VELOCITIES #(80 54 48 55 94 52 57 92 52 94 60 52 88 51
                        92 56 92 59 64 80 90 64 59 84 88 64 72 104
                        54 64 80 64 88 56 80 72 80 104)
           RELEASES #(9 9 8 9 9 8 9 9 8 9 9 8 9 9 8 9 9 8 9 9 9 8
                      9 9 9 8 9 9 8 9 9 8 9 9 8 9 9 8)
           DURATIONS #(9 9 8 9 9 8 9 9 8 9 9 8 9 9 8 9 9 8 9 0 9 8
                       9 0 9 8 9 9 8 9 9 8 9 9 8 9 9 8))
```

Figure 1.6. An Example Arpeggiation Pattern

The first time the performer plays a note after entering arpeggio mode, an arpeggio object is created which initiates the process of arpeggiating whatever notes are currently held down. The instrument numbers the current set of notes from lowest to highest starting with *0*, and updates the numbering every time keys are played and released. The instrument allows the use of the sustain pedal to hold keys.

Arpeggiation takes place as follows: The arpeggio object plays a note by obtaining the note number from the arpeggio pattern, a number between *0* and the most number of notes the pattern was written for—typically less than ten. Next, the instrument uses this number to find one of the notes that are currently held. In the simplest case, the note number is used to find the held note that was assigned that number. If the note number is greater than the number of held notes, however, the modulo operator is used to obtain a number in the right range. The more complex case is when there are more held notes than the number of notes the pattern was written for. The instrument detects this condition, and randomly picks from those held notes whose assigned note modulo the number of notes the pattern was written for is the same as the note number.

In mathematical terms, if *n* is the note number from the arpeggiation pattern, *N* the number of notes the pattern was written for (the highest n in the pattern plus 1), *M* the number of held notes, and *m* the held note chosen:

$$\text{if } N \geq M \text{ then } m = n \bmod N$$

$$\text{if } N \leq M \text{ then } m = \text{random}(M \text{ div } N) \times M + n$$

Where *random(x)* returns a random number between *0* and *x* inclusive, and *div* computes integer division (without rounding).

As an example, if the performer were holding the notes C, D, E and F, and the arpeggiation pattern from Figure 1.6 were being used, the resulting notes from the arpeggio object would be: C E E C E E D etc. Notice that note number 5 "wrapped around" and found held note 1. In this example *M* = 4 and *N* = 6.

Once the proper held note is found, a note-on is generated with a velocity that is a multiplicative average of the velocity the held note was originally played at and the velocity from the arpeggiation pattern. The release times and rhythmic values come from the pattern, just as in manual mode.

In addition, the instrument is sensitive to MIDI aftertouch (pressure applied to the keyboard controller after a note has been played). Aftertouch is translated into timbre by changing the MIDI channels that the arpeggios are played on. When there is no pressure, the arpeggios are heard on channels one and two. As pressure is applied, these channels are expanded one by one until all eight channels, and hence all eight modules of the Yamaha TX-816, are playing. The programs in the TX-816 synthesizer have been carefully crafted so that each module adds a particular richness to the composite sound.

Thus the instrument allows sophisticated yet intuitive control over the character of the resulting sound from gesture derived from traditional keyboard technique. By re-attacking held notes, perhaps with the sustain pedal down, the performer can easily scale the relative prominence of each pitch in a chord while simultaneously emphasizing or de-emphasizing specific notes via aftertouch. The result is an instrument that can be played with great delicacy and subtlety.

Searching Arpeggio Mode

Too often in computer/human musical systems, control over the computer is inadequate, forcing the human to severely compromise musicality for the lack of a responsive partner. In the worst case, a computer can be worse than a tape recording; while it may be equally deaf and uncontrollable, it may also be playing the wrong thing! For our hyperinstrument systems to be true instruments, control over the computer must be analogous to control over a traditional instrument such as a violin. The instrument must be responsive to the performer without requiring any awkward, non-musical gestures.

Unlike manual mode which selects its pre-stored patterns according to note-on velocities, standard arpeggio mode lacks an intuitive, musical method of arpeggio pattern selection. A number of methods were tried such as using the lowest octave to select patterns, using a foot pedal position, or using button pushes (program changes), but they were rejected because they were either non-musical or unable to support the number of patterns desired (over one hundred). On the other hand, for the music we were developing our instruments for, we did not need completely generalized access to the patterns; in fact, it was desired that each pattern be selected only at a particular place in the piece.

Our solution was to select patterns based on the chords that were played. Basically, the chord played is used to search through a database of patterns, and, in general, the first pattern that matches the chord is selected. It is assumed that the hyperinstrument, when used in this mode, has been configured with patterns and chords for a particular piece of music, since the instrument is matching against the particular notes of the chords, not the chord class. The instrument is not performing any sort of chord analysis.

This sub-mode of arpeggio mode is known as searching arpeggio mode since the arpeggio objects search through the pattern database to find the correct arpeggiation pattern. The searching algorithm is complicated by a number of considerations:

Because the performer will play each chord one after another, strong preference should be given for finding adjacent patterns in the database. On the other hand, the performer may make errors such as playing wrong notes or skipping entire passages, in which case the instrument should recover as quickly and gracefully as possible. In addition, performers tend to blend adjacent chords so that for a short period of time, notes from two chords can be held simultaneously. Lastly, one can assume that the performer will not make large numbers of errors, so that the best performance should be achieved for error-free input. The complete searching algorithm is shown below:

1. All the chords and associated patterns are arranged one after another in the order that they appear in the piece. The last chord "wraps around" to the first, forming a circle.

2. Searching arpeggios are either *forward* or *backward*. Forward means that all of the held notes are a subset of the current chord (the chord that is associated with the currently selected arpeggiation pattern). Backward means that some of the notes currently held are part of the preceding chord. When the mode starts, the first pattern is selected and the arpeggio is forward .

3. When a new note is played and the arpeggio is forward, a check is made to see if the note is part of the current chord or part of the next chord. If it is part of the current chord, nothing is done, but if it is part of the next chord, the next pattern is selected, and the arpeggio becomes backward. This is the normal way of changing patterns.

4. When a new note is played and the arpeggio is backward, a check is made to see if the note is part of the current chord or part of the preceding chord. If so, nothing is done.

5. If the note is not a member of any of the sets from 3 and 4, then a linear search is initiated starting from the current chord and continuing until a chord is found that is a super-set of the notes held. In this case, the new pattern is immediately selected, and the arpeggio becomes forward. If no such chord is found, it is assumed that the performer has played a wrong note, and the current pattern is not changed.

6. When a note is released and the arpeggio is backward a check is made to see if the remaining notes form a proper subset of the current chord *i.e.* that there are no longer any hold overs from the preceding chord. If so, the arpeggio becomes forward.

There are a number of advantages to this scheme. Assuming the performer makes relatively few errors, the searching arpeggios will simply walk through the pattern database one by one, flip-flopping between being forward and being backward. The costly linear search will be avoided. On the other hand, when errors are made, the system will tend to recover rapidly. Even if the system finds the wrong chord, it will tend to resynchronize as soon as a distinctive chord or series of chords have been played. Lastly, although the system has a preference for sequentiality, the performer can begin playing at any point and can skip around at will. This is especially important for rehearsals where it is quite common to repeat or omit sections. A system that requires a piece to be played through from the beginning can be terribly cumbersome to a conductor who wants to rehearse only the last two measures!

Percussion Based Instrument

The sister instrument to the keyboard based instrument is a percussion based instrument whose hardware configuration is show in Figure 1.7 below:

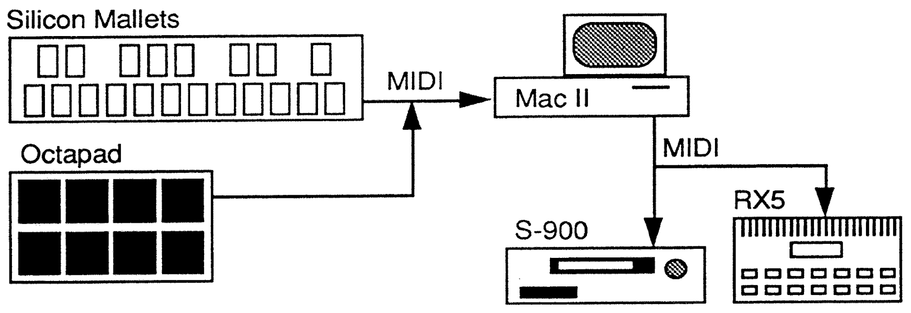


Figure 1.7. Percussion Based Instrument

Object Inheritance

In terms of software, the percussion based instrument is only a small variation of the keyboard instrument. Exploiting a feature of object oriented programming known as object inheritance, we were able to create the percussion instrument simply by specifying the ways in which the instrument differs from the keyboard software. Object inheritance allows sharing of code in a sophisticated way so that changes that are common to both instruments only need to be made once.

There differences in the percussion instrument, stem from differences in the nature of the input controllers and from differences in the nature of the output sounds. The percussion controllers used were the Simmons Silicon Mallet, a three octave MIDI vibraphone, and the Roland Octapad, a similar device with only eight notes. The octapad was included as a command input, and notes struck on the octapad generally do not produce sounds; they are used to communicate commands to the instrument.

Differences in the Controller

The percussion controllers that we employ provide less information than keyboard controllers since they only sense when and how hard a note is struck. They generate note-offs after a fixed time interval, and do not sense if a mallet or drum stick is being held on top of a note. In addition, they do not provide aftertouch data.

Nonetheless, we were able to adapt the keyboard instrument for our percussion controllers by redefining some of the interface functions. Manual mode and arpeggio mode both require some method of selecting the notes which are to be repeated or arpeggiated. Selecting a note in the first place is not a problem, but once a note is played, the corresponding note-off automatically comes a short time later and is totally meaningless. Our solution is to de-select notes via damping, *i.e.* playing the note with a velocity below some setable threshold, employing one's fingers instead of a mallet. We found that performers had no difficulty mastering this technique, and we believe the solution is in keeping with our efforts to make performance gestures as intuitive and musical as possible.

Differences in the Sounds

The other major difference betwen the percussion system and the keyboard based instrument is that the output timbres of the percussion instrument are drum sounds and are essentially unpitched. Furthermore, it is highly desirable to have access to a vast number of sounds—far more than thirty-six keys on the input controller.

Our solution here was to construct a facility for generalized mapping of the notes on our input controller to some set of output sounds. The mapping takes place just before a MIDI note is produced, and is used in manual mode and arpeggio mode as well as through mode (shadow mode was not used in the percussion instrument). A map takes a note about to be played and converts it to any number, including zero, of specific notes and MIDI channels while forwarding the velocity information.

Our standard configuration uses three basic maps: a delicate map composed of sampled wood sounds, a map made up of typical percussion sounds (congas, timbales, shakers), and a map of loud rock and roll percussion (electronic tom toms, crash cymbals, breaking glass). These maps were then layered in every combination making eight maps all together. In the percussion version of arpeggio mode, three keys in the lowest octave were assigned to toggle each of the layers whenever struck. Thus our facility allowed careful organization of a large number of sounds while providing real time control over the selection of sound palettes.

Percussion Arpeggio Mode

The percussion based instrument version of arpeggio mode is sufficiently different to merit a brief discussion. As in the original arpeggio mode, the main difficulty is in the method of selecting the arpeggiation pattern. In this case, however, the searching arpeggio solution would not work; one of the principles of the percussion version is that the choice of sounds, and hence the notes selected, is left to the performer and is unscored. A further difficulty arises because synchronization. In the keyboard instrument, arpeggiation stops when there are no held notes, and restarts as soon as a new note is selected. This allows the keyboard performer to resynchronize the arpeggios with every chord, if necessary, by completely releasing one chord before playing the next. While this is still true of the percussion version, in practice, it is a difficult maneuver to damp all the notes and restart an arpeggio quickly. This remains true despite the addition of a *damp all* key on the octapad.

We solved both problems by reserving the lowest octave on the mallets for pattern selection. Striking one of these keys instantly changes the pattern to the pattern associated with the key, and further, if the velocity with which the key is struck is above a certain threshold, any running arpeggio is resynchronized. Resynchronization in this way is identical to instantaneously damping all the notes, selecting a new pattern, and restriking all the damped notes.

Unfortunately, pattern selection using our new method restricts the number of patterns to eight (three notes of the lowest octave were used for map selection). In order to provide additional variety in our patterns we added some functions on the octapad which temporarily alter the current arpeggiation pattern. These functions alter the dynamics of the next note to be played in the pattern. One key sets the velocity recorded in the pattern to whatever velocity the key was struck with, and another key sets the velocity recorded in the pattern to zero. A third key restores all velocities to their pre-recorded values. All changes are temporary and are discarded as soon as another pattern is selected.

Although both instruments were developed for use in a particular piece of music the instruments were designed to be used, without modification, in a variety of pieces. Their design is such that the data such as arpeggio patterns, percussion maps, shadow maps, etc. are separate from the actual programs and are thus interchangeable. Naturally the sounds in the synthesizers and the synthesizers themselves are equally easy to change. The controllers themselves can be substituted, although this is more difficult as controllers tend to vary widely in their capabilities.

The development of the *VALIS* instruments has uncovered some of the fundamental issues in hyperinstrument design: not only is it necessary that the performer have absolute control over the instrument, it is essential that the control be accessible in a musical way.

1.6 Instruments that Listen

One of the most exciting directions that we have been pursuing is the development of computational models of listening. Most of our efforts in this direction have been inspired by Marvin Minsky's ideas as expressed in his revolutionary book, *The Society of Mind* [Minsky 1986]. A prototype of such a listening system which recognizes musical fragments and can follow scores has been implemented in Real Lisp.

Minsky's theories are formulated around the hypothesis that the mind's cognitive processes are composed of thousands, perhaps millions of simple parallel processes organized in hierarchies. These processes, or *agents* as they are known, are not very intelligent when considered as separate entities, but when many agents are organized into hierarchies of *agencies*, the result can constitute true human intelligence.

Our agent based score follower is organized into a three level hierarchy as shown in Figure 1.8.

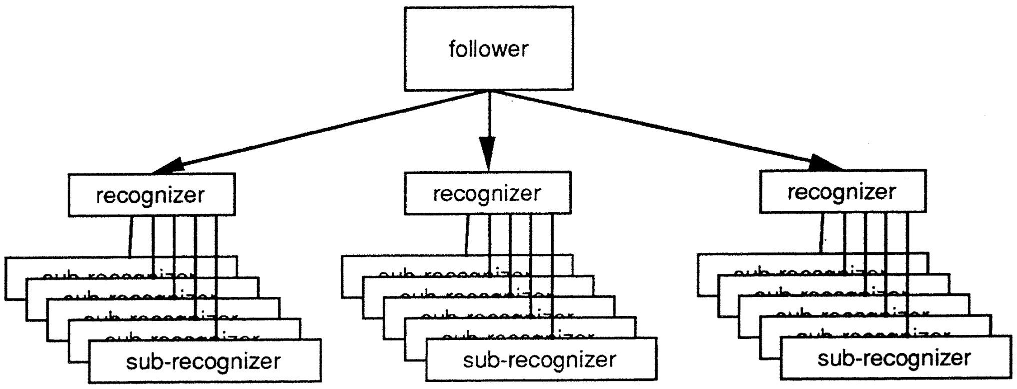


Figure 1.8. Agent Based Score Follower

The score follower works by recognizing a series musical fragments, usually on the order of five to ten notes long. At any given moment in a performance, the score follower knows where the performer is up to the fragment most recently recognized. Since fragments may overlap, the follower can track performances note by note if desired.

Each level of the hierarchy performs a simple, relatively unintelligent task. The *follower* agent has a list of fragments to be recognized and the time at which that fragment is expected, and it is responsible for activating a *recognizer* agent to recognize the fragment some several seconds before the fragment is expected. It is common to have several recognizers active simultaneously.

A recognizer is thus responsible for recognizing a specific musical fragment. It accomplishes this by employing a number of *sub-recognizers*. The recognizer activates a sub-recognizer for every note that is played. The sub-recognizer uses a very simple algorithm to judge whether each of the stream of notes starting with the note it was activated on matches the fragment. If there are too many mismatches, the sub-recognizer will automatically deactivate, freeing its computational resources. If the sub-recognizer succeeds on a sufficient number of notes, it signals its success to its parent recognizer which deactivates itself and all its subordinates after reporting success to the follower.

Although the score follower has not been thoroughly tested, preliminary results are extremely encouraging. The follower tends to be quite robust with respect to variations and errors in performances. Further work in this direction will involve extending *Society of Mind* principles to more generalized models of listening.

1.7 Implementation Considerations

Why Lisp?

Programming in Lisp poses a number of difficulties for real time applications. The most odious problem is Lisp's memory management scheme which involves a process known as *garbage collection* [Abelson 1985]. Normal Lisp programs generally assume that there is an infinite amount of memory available, and tend to consume memory at a rapid rate. When whatever finite memory is entirely consumed, the Lisp system halts the program and spends a significant amount of time (generally between .5 and 5 seconds) ferreting out locations in memory which are no longer needed by the program. Obviously this process, known as garbage collection, is catastrophic for a real time system. In our case, our instruments become completely dead if garbage collection occurs.

Another problem with Lisp is its relative inefficiency in comparison to more traditional programming languages such as C [Kernighan *et al*, 1978]. A comparison between real-time schedulers implemented in Lisp and in C revealed an "order of magnitude" difference in scheduling overhead for the Lisp implementation [Boynton 1987]. This comparison is especially relevant as it was conducted using a scheduling algorithm similar to Real Lisp in the same Allegro Common Lisp environment on the Macintosh II. The C implementation was also on the Macintosh II in Lightspeed C.

Despite these problems, we still feel that Lisp is the best suited environment for our work, mainly because of the ease of prototyping and modifying systems. Lisp has a number of features which facilitate program development. Lisp is interactive which allows changes to be made quickly without the compilation and linking phases present in conventional languages. In addition, Lisp provides high level symbolic debugging tools which allow such features as stepping through programs statement by statement, tracing the invocation of functions, and inspecting the values of variables at any time by name. Furthermore, Allegro Common Lisp has an elegant object system which has proved to be extremely useful in our work.

Although one must program more carefully to avoid the creation and subsequent collection of garbage, we have found the overall programming environment to be superior. In addition, the efficiency of Lisp has continued to improve as its use becomes more widespread. Particularly interesting are recent proposed schemes for real-time garbage collection [Hewitt and Lieberman, 1986], and the continuing decrease in cost of specialized Lisp processing hardware.

Scheduler Complexity

Another thorny issue is that of scheduler complexity. Simply put, there is a trade-off between scheduler complexity and scheduler efficiency. A complex scheduler may have many desirable features such as multiple priorities or preemption (the ability to interrupt a low priority task at any time to execute a higher priority one), but the price is a decrease in scheduler efficiency. The more complex the algorithm, the more time the scheduler must spend computing the algorithm, leaving less time for the actual application.

Scheduler complexity becomes an issue when, for a given time span in the execution of a program, there is not enough time to compute everything scheduled. The simplest example is when two function applications that take several clock ticks to compute are scheduled to execute at exactly the same time. Which ever function is picked to go first will be more or less on time; the other will be late. If the scheduler is simple and has no notion of priorities, it would select a function arbitrarily, whereas a prioritized scheduler would choose the function that had been assigned a higher priority. A common case is where notes are being simultaneously played via MIDI and displayed on a graphics screen. Playing the notes should take priority over displaying them.

On the other hand, the prioritized scheduler must do more work every time there is a clock tick, since it must check each priority level separately for functions that are ready to be applied. In addition, there can be extra overhead associated with scheduling functions in the first place.

Our choice of a simple scheduler was made because nearly all of our scheduled tasks are trying to output MIDI notes, and thus are of equal priority. The most notable exception is the searching arpeggios. It would desirable for searches to happen at a lower priority so that the arpeggio would continue to play even if it were in the process of a search. This is not the case now, and the searches do interfere with note playing if the performer makes many errors. However, our scheduler demands minimum overhead, especially in servicing MIDI input. This is of particular importance since responsiveness is a key feature for any musical instrument.

Year One

Hyperlisp and the Hyperinstrument Environment

2.1 The Hyperinstrument Programming Environment

Over the course of the year, the hyperinstrument programming environment, which includes Hyperlisp [Chung 1988], has been extensively improved and generalized. The focus of this work was in creating a software architecture in which several programmers could cooperate on the same project without interfering with one another, while at the same time, making effective use of previously implemented modules. By exploiting both the object system and package facility available in Allegro Common Lisp, a hyperinstrument coding discipline was established which both eliminates unforeseen interactions between programs written by different people, and maximizes the reusability of existing software.

A great deal of effort was spent identifying modules which would be reused again and again in different hyperinstrument systems. These modules were separated into software layers according to their generality. See figure 2.1.

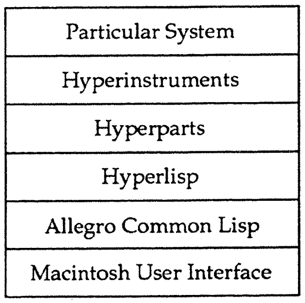


Figure 2.1: Hyperinstrument Software Layers

The most important layer in the diagram above is the *Hyperparts* layer, which can be thought of as a "toolbox" of useful modules and routines. New hyperinstruments are never implemented "from scratch," instead they are built on top of the Hyperparts layer, which contains well-tested solutions for many hyperinstrument tasks. As new systems are developed, modules which are reusable are identified and added to the Hyperparts layer. In this way, we have accumulated an extremely valuable library of software, which eliminates redundant effort while contributing to the overall robustness of our systems.

Additionally, we have continued to extend and improve the Hyperlisp layer shown above, which contains all of the functions for programming real-time applications and accessing MIDI. The current version (1.1b) features improved integration with the Allegro Lisp interface and adds the capability of executing background tasks, whereby less important, non-real-time computations can be carried out "in the background," when the machine is not busy servicing more time-critical tasks.

Towards the Center: New Musical Functions

Our new hyperinstruments have been designed to cover a wide range of possible real-time musical interactions, extending from the precise control of minute parametric detail, to the mixing of multiple layers of musical data in a "super-studio" mode, to the shaping of large-scale phrases or sonic shapes, to the enhancement of temporal precision, either for the individual player, or between several players, to the manipulation of compositional elements in such a way as to make musical structure cognitively explicit. A thorough description of our most significant new hyperinstrument designs can be found, along with carefully annotated Hyperlisp code of each application, in [Machover and Chung, 1989].

2.2 Rhythmic Functions

Several recent hyperinstrument designs will provide a concrete example of our design philosophy, and the musical issues that we are attempting to address. The first new application involves "rhythmic amplification," an attempt to enhance rhythmic virtuosity and precision through various real-time machine controls. The most successful technique used a rhythmic grid to create a background, inaudible stream of rapid rhythmic values. The grid was used simultaneously for two purposes: first, to determine the precise start-point of automated events triggered by the keyboard player; and second, to insure that all notes played on the percussion instrument (in this case, drum-like unpitched timbres) were sounded only at grid points (although the window of pull towards these grid points was setable). The percussionist was therefore able to concentrate on articulation and timbre selection, while producing an amazingly rapid array of precise rhythms, synchronized perfectly with all other instruments. This technique worked well when the performer had some liberty in choosing which note to play at which moment, and where the exact ordering was not of primary importance to the musical result [Machover, *Towards the Center*, score measures 531-569].

Another technique for enhancing rhythmic complexity was tested using the same quantizing grid. Rather than forcing live playing to conform to the grid beats, complex rhythmic patterns (mostly articulation and accent patterns based on regular beat subdivisions such as quarter/eighth/sixteenth/thirtysecond notes) were composed and pre-stored in the computer. Specific patterns, which also had pitch components, were triggered by specific notes on the keyboard. These patterns were synchronized to the closest grid-point. The Kurzweil Midiboard's polyphonic afterpressure capability (a powerful control which greatly enhances a keyboard's hyper-sensitivity) was exploited to control the "warping" of these patterns once initiated. Increased afterpressure causes increased speed in the pre-stored pattern. *Return to 0* afterpressure snaps the pattern back into precise sync with the grid. We found this to be a highly powerful tool for rhythmically complex, but synchronizable, ensemble performance, and will certainly further refine these ideas in the coming months [Machover, *Towards the Center*, score measures 169-279].

Various attempts were made to extend and generalize the procedure of using performance gesture to refine and shape pre-stored musical material. The most elaborate new system is keyboard-based. Left-hand keyboard notes trigger a library of musical sequences. Velocity of the trigger note scales the overall velocity of the sequence. The afterpressure on the same trigger note affects timbre: ten different timbral levels are accessed through increased pressure. The right hand is used to further articulate these rapid streams of notes. The velocity value of any white key in the upper keyboard octaves is taken to scale the velocity of the closest sequenced note, thus allowing a very subtle shaping of overall loudness. In addition, black keys in the same octave serve to sustain the closest sequenced note with a distinctive timbre [Machover, *Towards the Center*, score measures 531-587].

Several attempts were made to adapt percussion technique to shape continuous musical parameters, such as timbre. The most successful attempt involved the technique of percussion tremolos, using the KAT controller. We chose two-mallet tremoloing since it is a technique that most players master, achieving fine and separate control of loudness and speed of tremolo. We used these two components as independent controls. Each KAT pad generated one pitch. Speed of tremolo determined timbre as follows: Ten different timbres were available. The slowest tremolo speed selected timbre one. As speed increased, subsequent timbres were added, with two playing at a time. Experiments were made to determine the most effective, gradual transition from pure to inharmonic sounds, to determine how many timbres should overlap, and to determine the proper algorithm for mapping change of tremolo speed onto speed of timbre change, adding a delay so that such changes were neither too abrupt nor too late and disconnected from the performance [Machover, *Towards the Center*, score measures 362-409].

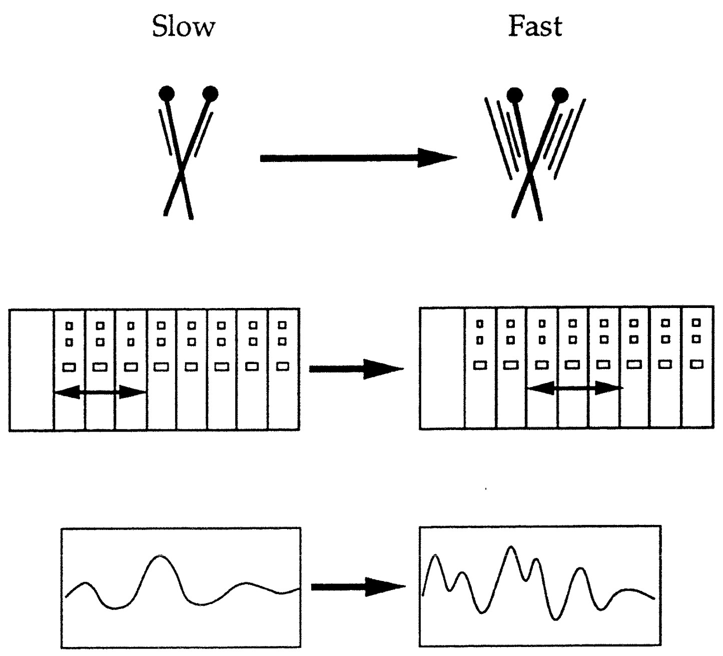


Figure 2.2: Tremolo/Timbre Instrument

2.3 Hyperensemble

Perhaps the most exciting and innovative new hyperinstrument work has been the development of "double instruments." Double instruments are instruments which are played by combining the musical gesture from two performers playing on separate physical controllers. Double instruments represent a new kind of ensemble performance where instead of acoustically combining the sound of two instruments, players can musically combine the gesture of two performances. The instruments are designed so that each musician can influence certain aspects of the music, but both players are required to perform in ensemble to create the entire musical result. In one version of this double instrument concept, the notes on the percussionist's instrument are re-mapped to the pitches which are currently being played on the keyboard. Each octave on the percussion controller sounds the same notes but with different timbres. In addition, the keyboard player uses polyphonic afterpressure to weight the mapping so that certain notes appear more frequently than others [Machover, *Towards the Center,* score measures 310-361].

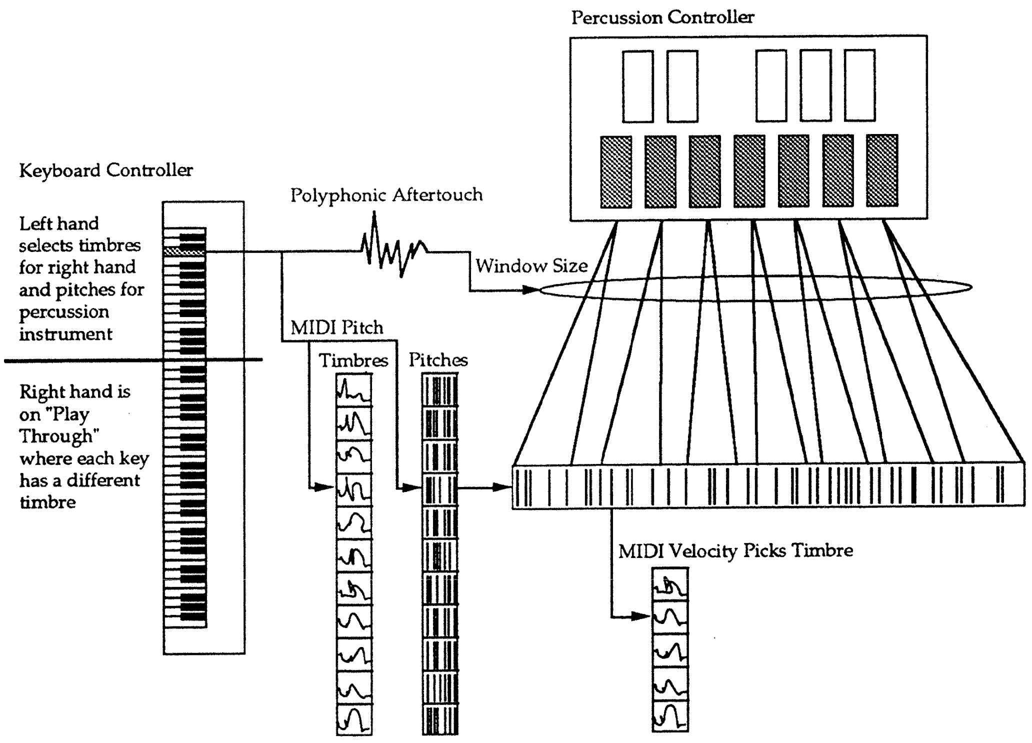

Percussion Controller
Keyboard Controller
Polyphonic Aftertouch
Left hand selects timbres for right hand and pitches for percussion instrument
Window Size
MIDI Pitch
Timbres
Pitches
Right hand is on "Play Through" where each key has a different timbre
MIDI Velocity Picks Timbre

Figure 2.3: Double Instrument

There are essentially two main principles we are trying to explore in our work on double instruments. First, we have found that by creating a sensible division of labor between the two players, each performer can concentrate entirely on his or her respective task with unprecedented precision and expressivity. For example, by restricting the percussionist to seven white keys, the percussionist can produce extremely fast and precise rhythms without having to think about the specific pitches or the distances between the keys. Secondly, we have discovered that double instruments require a new kind of ensemble performance where the two performers must communicate on many levels simultaneously. Double instrument performers must relate their musical gestures not only to the resulting sound as in traditional instruments, but also to the gesture and sound of the other performer.

2.4 Applications of Minsky's Theories to Hyperinstruments

Over the past year, we have vigorously pursued research into the applications of Minsky's theories [Minsky 1986] to models of music cognition and perception, especially in the domain of rhythm. Joseph Chung's Master of Science thesis, "An Agency for the Perception of Musical Beats," [Chung 1989] explores the intricacies of beat and meter perception, and develops explanations of why seemingly simple tasks as foot-tapping are difficult to perform on a computer. He proposes an original theory based on principles of how beat perception might be performed as a process built from *agents* and *agencies*, and has implemented this theory (in non-real-time) in Hyperlisp. From the outset, this work was intended to supplement the musical intelligence of hyperinstruments, and has already led to the development of the rhythmic repetition algorithms used in a new percussion composition/performance system known as *PercMan*.

While the basic theories of beat perception presented in Chung's thesis are quite promising, we have found that the model can still make unlikely choices, or otherwise get "confused." We have concluded that any cognitive model must have higher-level facilities for learning or conditioning certain responses based on success and failure of previous cases. Minsky terms these facilities *censors*. They prevent the same mistake from being made again and again. Even though a model of rhythm perception might be essentially correct, the nature of music is such that it often plays upon the unusual and ambiguous cases. Without the higher-level agents for noticing and suppressing potential mistakes, the model cannot behave in a human-like fashion.

Work on rhythm perceptionis being continued by Chung and David Rosenthal, who has been developing an alternative approach to meter perception based on repetition of musical features [Rosenthal 1989]. Rosenthal is working towards a computational theory of rhythm as a whole, while Chung is concentrating on coding streamlined versions of these models in real-time for use in our performance systems. We are confident that research in these areas will yield results valuable to hyperinstruments in both the long and short terms.

Year Two

Gestural Control of Musical Parameters

3.1 New Technical Developments and Features in the Hyperinstrument Environment

Several modifications and extensions were made to our hyperinstrument software architecture during the past year. Most the these changes were made in order to accommodate the requirements of *Bug-Mudra,* although the overall software environment has been continually optimized and extended. In addition, we are currently in the process of moving the hyperinstrument software to the new version of Macintosh Common Lisp which utilizes the new Common Lisp Object System.

The use of a continuous tape part in *Bug-Mudra* presented us with a unique opportunity to experiment with the possibilities afforded by a *perfect score follower*. *Bug-Mudra*'s tape part consists of a stereo track of musical material that was prepared using a conventional sequencer. The tape was mixed in a single pass and recorded with SMPTE time code such that the beat information in the music is preserved against the time code with sub-frame accuracy. When the tape is played back in performance, the SMPTE is converted to MIDI time code and fed into our hyperinstrument system (see accompanying diagram). Thus, the hyperinstrument "knows" the precise location in the score (in terms of beats) to within 1/100 second and with 100% certainty.

The technical requirement of synchronizing a hyperinstrument (and hence Hyperlisp) to an external clock was realized through the MIDI Time Code (MTC) standard which has recently gained widespread acceptance in industry. MIDI Time Code was chosen because of its direct translation from SMPTE, the industry standard time code for audio and video. In our implementation, we took advantage of the fact that 25 frames per second time code produced at the rate of 4 MTC quarter-frame messages per frame, exactly equals the Hyperlisp clock rate of 100 divisions per second. We exploited this fact by allowing the arrival of MTC quarter-frame messages to drive the Hyperlisp clock on a one for one basis without any phase-locked loops. On the other hand, our implementation can only accept MTC at 25 frames per second (European EBU standard). Our decision not to implement a completely general synchronization system was due to future plans to utilize the Apple MIDI manager instead of our current MIDI driver. The MIDI Manager comes complete with a general synchronization capability, but was not ready in time for *Bug-Mudra*. Other advantages of using the MIDI Manager are described below.

The synchronization process occurs on two levels of software. The MIDI driver was extended to recognize MIDI Time Code data and advance its 100 division per second clock with every quarter frame message received. Additionally, the MIDI driver parses the frame information contained in the MIDI Time Code messages and stores the completed frame numbers, as well as the time at which the last complete frame number was parsed. The process of maintaining synchronization occurs at the Lisp level, and is handled by an object known as a sync-score. The sync-score is an extension of the standard score object described in previous documents, and is designed to play a score in synchrony with a time code stream, given some specified offset from time code frame 0. The sync-score works by periodically checking the running clock against the parsed frame numbers. When a discrepancy is detected, two different actions occur depending on the magnitude of the discrepancy. When the discrepancy is smaller than 1/2 second, the mis-synchronization is corrected for gradually over a period of time (within a maximum of 2.5 seconds). For greater discrepancies, it is assumed that the time code has been re-started from a new position, and the score is re-cued to the new synchronization immediately.

Another change to our software environment was necessary to accommodate the MIDI guitar controllers used in *Bug-Mudra*. Because these controllers produce data on multiple MIDI channels, we added a third dimension to our MIDI dispatching function tables, so that incoming MIDI could be efficiently mapped to handlers through message type, data message 1 (*e.g.* pitch), and logical channel (*e.g.* string). In general, the process of refining and optimizing the critical path of MIDI data flow has continued with particular emphasis on reducing the delay through the system.

Hyperlisp 2.0

We recently initiated a major rewrite of much of our hyperinstrument software to coincide with the forthcoming release of Macintosh Common Lisp 2.0 in which the Objectlisp object oriented programming system has been replaced by the Common Lisp Object System (CLOS). CLOS will be part of the ANSI X3J13 standard and will thus be an integral part of Common Lisp. The main advantages to moving our software to CLOS are greater portability and more maintainable software.

Greater portability has become a serious issue for the hyperinstruments project, as we anticipate the desire to run our software on more powerful platforms than those currently afforded by the Apple Macintosh series. Because Objectlisp is not supported on any other machines besides the Macintosh, the process of moving hyperinstrument software to new machines would have been fairly difficult and would have been different for each system. By moving to CLOS, which will be part of the standard Lisp available on any machine, our future choices of hardware are far less limited. Although we still must translate our existing software from Objectlisp to CLOS, this translation need only occur once. At this point in time, we have translated Hyperlisp (now version 2.0) as well as the most commonly used hyperinstrument modules. The remaining software will be translated "on demand" as required by future systems.

A further advantage to using CLOS is the more maintainable software afforded by the CLOS *generic function* paradigm. Objectlisp is a *prototype* system in which objects are built by defining a prototypical object and creating instances and specializations of that object. While this style of programing tends to be conducive to creative cycles, it suffers from the disadvantage that objects often depend on each other in obscure ways. Changing an object may have many unforeseen repercussions. CLOS, on the other hand, is based on the concept of generic functions, where new functionality is added by defining new *methods* which apply to the specific case at hand. While these two approaches are essentially equivalent in functionality, it is commonly recognized that the generic function paradigm yields more structured and maintainable code which is better suited to software industry standards.

Another feature of Hyperlisp 2.0, currently under development, is the replacement of our existing low-level MIDI driver with Apple's MIDI Manager. In addition to the complete external synchronization implementation previously discussed, the MIDI Manger will enable Hyperlisp to share data and processing with other commercially available Macintosh MIDI programs. In this environment, several MIDI programs can be running simultaneously on the same machine, with the MIDI *patching* occuring through the MIDI Manager. One could run a conventional hyperinstrument in Lisp and record the MIDI output using a conventional sequencing program on the same machine. Alternatively, one could use the graphical MIDI editing facilities provided by the better sequencers to create and edit raw sequence data for later use in Hyperlisp. Integration with the MIDI Manager will yield a richer and more powerful hyperinstrument workstation.

It is worth noting that the Media Laboratory Music and Cognition group has become an *alpha* test site for Macintosh Common Lisp with several benefits including advance notice of new releases and features, as well as special responsiveness to our particular needs. The developers of Macintosh Common Lisp have already made certain modifications to their product in order to interface more easily with Hyperlisp.

Future Platforms

We are continuing to evaluate the possibilities of moving hyperinstruments to more powerful hardware, and, as previously noted, we have anticipated this possibility in the design of our next generation architecture. While it appears that Apple is committed to producing higher performance machines, and has produced two new high end Macintoshes, the 25MHz IIci and the 40MHz IIfx, in the past year, industrial trends indicate the continuing development of low-cost, RISC based, UNIX machines such as the Sun Sparcstation and the Digital DECstation which offer considerably higher performance than Apple's computers. The main problem with these machines is the difficulty of performing real-time tasks under UNIX, though current trends in real-time multi-media programming seem to indicate that good real-time extensions to UNIX are forthcoming. Perhaps the most promising developments in this area continue to come from NeXT corporation.

This year we have experimented with many new hyperinstrument functions, several in an attempt to improve on models experimented with in the previous research period, and several to try totally new mappings between performance/musical information and computationally enhanced result. Most of these functions were attempted in Tod Machover's composition *Bug-Mudra,* both in its initial performances at Tokyo's Bunkamura Theater in January and February 1990, and in subsequent improvements to the programs and concepts during Spring and Summer of 1990.

3.2 New Hyperinstrument Functions

Rhythmic Enhancement

Several experiments were made to explore the concept of rhythmic enhancement or amplification first attempted in Machover's *Towards the Center* and explained in detail in our paper given at the the 1989 International Computer Music Conference [Machover and Chung, 1989]. To give further emphasis to the already precise playing of the percussionist in *Bug-Mudra*, different levels of real-time quantizing were tried in different sections of the piece. All non-pitched percussion in this work is generated from a KAT mallet-percussion controller, connected to three Alesis drum machines. The first and sixth (final) movements of *Bug-Mudra* consist of extremely rapid, synchronized, and difficult to play rhythmic figures (changing meters and accent patterns of sixteenth notes, running at quarter note equals 120). A series of experiments were conducted to measure the ability of the percussionist to play a specifically notated part (*i.e.* no degree of tolerance for specific note events to be delayed to the next waiting position in a quantizing grid) with various degrees of real-time quantization. Our results showed that no greater than 60% quantization is desired with such precisely notated music, *if* the performer is of the highest calibre *and* extremely familiar with the music being played (to the point of almost memorizing the meters and rhythmic patterns). However, for less proficient players, or during the learning process for the composition, higher degrees of quantization are tolerable, and indeed helpful, in assuring proper rhythmic vigor and emphasis. Experiments with section two of the same composition confirmed our hypothesis that a much higher degree of quantization (ca. 80%) is highly effective when a certain "slippage" of timbral selection (a precise note readjusted to conform to the quantizing grid) can be tolerated. In fact this technique is highly effective in a real-time context, liberating the percussionist to concentrate on timbral selection and overall phrase shaping while relinquishing certain control over exact coordination with the underlying time grid.

Several experiments were done with the transformation of extremely rapid rhythmic patterns, approaching tremolos, into various musical results. Several models were made to translate acoustic guitar tremolos into complex timbral extensions. Software was written to account for different styles of tremolo, since this phenomenon is so controllable and subtle with the acoustic instrument. We were able to detect speed of tremolo, synchrony of chord notes versus arpeggios versus independent polyphonic playing (all of which were treated differently), as well as harmonic content of the tremolos. A system was designed to use these effects in real-time, but had to be abandoned because an adequate MIDI-acoustic guitar was not found to be reliable enough for our purposes. Therefore, these hyperinstrument mappings were simulated in the studio using acoustic guitar recordings and hyperinstrument transformations on the Synclavier Direct-to-Disk and Sampling systems, which then were incorporated into the prerecorded section of *Bug-Mudra.*

A more successful real-time application of tremolos was achieved by combining rapid playing with automated rhythmic grids stored in the hyperinstrument system. For this application, the attempt was to create a very complex pattern of synchronization and asynchronization between the three live instrumentalists. The notated music for each player specified particular pitches with long held-note time values, very precise dynamic notation and articulation markings, and general degree of tremolo speed. The hyperinstrument system analyzed the note being played by a particular instrument and chose an appropriate rhythmic grid. These grids were technically similar to our quantizing grids, except that in themselves they varied greatly in relation to the overall internal grid clock, slowing up and speeding down constantly. Each tremolo attack from the live players was coordinated with the appropriate "warping grid", and these grids were in turn synchronized with each other. Therefore, all simultaneous warping patterns were coordinated in a very delicate and complex way automatically by the hyperinstrument system, while the performers retained control over the phrase shaping of these rhythmic phrases (*Bug-Mudra*,Section 4b).

An extension to the automated arpeggio system used in Machover's *VALIS* and *Towards the Center* was employed for the electric guitar in *Bug-Mudra.* A similar procedure was used to send guitar pitches into automatically selected rhythmic templates, which were in turn synchronized with the overall rhythmic grid of the composition. Velocity of each note was measured and used to scale the appropriate events in the resultant arpeggio patterns. Pitch bend of each guitar note was used to control timbre of that note in the resultant pattern, similar to the use of afterpressure in the keyboard version of this system (*Bug-Mudra,* Section 4a).

Hyperlead Guitar

One of the more interesting new guitar-oriented hyperinstruments designed for *Bug-Mudra* is what we term *hyperlead guitar*. In this case, the object was to turn the MIDI-electric guitar into a phrase and articulation shaping instrument, with an impossibly fast succession of diverse pitches which would be handled automatically by the computer. A rapid and dense melody was presequenced, allowing all pitches to be stored ahead of time in memory. During the appropriate section of the piece (*Bug-Mudra,* Section 5), this pattern was generated in exact synchronization with the underlying, but inaudible, time grid. However, all controls for this melody were given from the guitar. Each note played on the guitar sent out MIDI velocity to scale the next event in the MIDI stream; if the live guitar played nothing, no sequenced sound was heard. The left-hand fingering of the guitar was then changed into a complex timbral instrument. A multidimensional timbral space (created with the mixing of many synthetic Yamaha timbres) was mapped onto the frets of the guitar. A simple progression was made from pure guitar timbres to more and more complex ones by moving to higher and higher strings. "Brilliance" or "brightness" was added to the synthetic sound by moving progressively, from any fret position, from the low E-string ("dullest") to the high E-string ("brightest"). In addition, notes played in a close time window (interpreted as varying degrees of "chords") were rewarded with special timbres. Although it took the guitarist a fair amount of practice to learn to coordinate with the sequenced melodic material, and to learn the various articulatory and timbral controls of this instrument, it proved to be highly successful in performance, and is another good example of a satisfying combination of carefully selected modes of machine automation that liberate the performer in another dimension.

Hyperchords

Another hyperinstrument implementation that was experimented with but finally rejected was what we termed *hyperchords*. This design was an attempt to liberate the guitarist from the role of selecting groups of pitches so that he could concentrate on articulating and shaping them. This technique was originally designed for Section 6 of *Bug-Mudra*. The idea was to have the automated score-follower detect the exact beat of the piece, choose an appropriate six-note chord (one for each string of the guitar) and to send these notes automatically to the appropriate guitar string for the pre-specified duration. Therefore, just as hyperlead guitar allowed the guitarist to concentrate on timbre and articulation, we expected a similar system to work here. Each string, regardless of left hand fingering, would only play one note at a time, specified by the automated chords. These notes would only sound, however, if the string was actually played by the right hand, and then scaled with the exact velocity of the performance. In addition, a similar timbral mapping, concentrated onto the lower frets of the instrument to avoid the necessity of jumping all over the guitar neck recklessly, allowed any performed note to be given a special timbre. This hyperinstrument was designed as an accompanying instrument, rather than as a primary, melodic one. It didn't feel correct and satisfying to the performer, so the idea was abandoned during the rehearsal period for the piece. We believe that such an idea works when the dimension of automation is sufficiently constrained (such as the single presequenced melodic line described above), but rapidly becomes difficult to control when too many events are predetermined. In this case, we found that the rate of harmonic change (approximately six new pitches sent to the guitar every 1/2 second) was too dense and rapid to allow the guitarist to become familiar with exactly which pitch would be available to him at a specific point in the piece. Therefore, although the guitarist was given free reign to articulate these pitches as he wanted, sophisticated choice became impossible since he did not sufficiently know what pitch would be on what string at what time. Rather than being liberating, this in fact caused the guitarist to lose control over articulation, and play the chord notes in semi-random fashion. It was interesting to note that although the pitches themselves were by definition appropriate (being sent only at the right time by the computer, to coordinate with the general harmonic schema of the piece), the effect produced was of a lack of interesting articulation, loss of interest on the part of the player, and a feeling of disconnection with the general musical texture. Through this experiment, we learned important principles about the types of automation which seem most fruitful, and more about the limits of human players to incorporate and shape complex presequenced material.

Other New Techniques

Two other hyperinstrument techniques were used for the first time in *Bug-Mudra*. The first is the use of a perfect time synchronizer, through employment of a presequenced tape, to allow for the system to be aware of precisely where the score is at any particular moment, with extreme precision. This allowed us to make many musical changes, of hyperinstrument modality or timbral selection for instance, at extremely precise moments in the score. At certain points, each note on a particular instrument could be given a special multidimensional articulation, appropriate to that moment in the piece. The other novelty is the use of the DHM Dataglove to shape many overall parameters in real-time performance. The technical implementation of this system, and its musical use and future development is described below.

3.3 Experiments with Non-Standard Interfaces

Hand Tracking Devices

Following our established research paradigm of focusing experimental work on solving a particular real-world problem, Professor Machover designed and composed *Bug-Mudra* with the idea that a hand gesture tracking device would be used to shape and mix timbres as an integral part of the live performance. The basic model was to implement the conductor's traditional role of shaping dynamics and timbre with the left hand in a direct manner so that the conductor could take a more active role in the performance than is currently possible with electronic instruments. Thus, we focused our efforts on the control of sound mixing and timbral shaping with the hand and purposefully avoided conventional MIDI controller applications, such as "air piano." The rationale for this approach was that existing keyboard controllers and drum controllers serve their respective purposes quite well, and we found it much more interesting to explore the kinds of control for which existing MIDI controllers do not exist. By its nature, hand gesture is continuous and flowing and maps most directly to control of similarly continuous parameters.

Preliminary research for *Bug-Mudra* uncovered a few existing systems for tracking body dynamics that might be suitable for a hand-controller. In the mid-80's, researchers at the Media Laboratory had experimented with LED's mounted on a glove and a video camera that would track finger position from the LED's. A more recent visual system, *Mandala*, was developed by Vincent John Vincent and Francis MacDougall in 1987, and uses video silhouette imaging to create a *virtual* instrument in the air. While both of these systems gave promising results for large scale movement under somewhat idealized conditions, neither could possibly provide the fine, multi-dimensional data necessary to track the subtle movements of the hand.

We performed extensive experiments with the VPL Dataglove, which has become popular as an input device for virtual reality systems. In general, we found that it lacks the consistent and fine control needed for a virtuosic musician. Test results showed that it could only yield five bits of useful data per finger joint, and that the data for specific hand positions tended to be inconsistent and irreproducible over time.

We chose to use the Dexterous Hand Master (DHM), manufactured by Exos, Inc. The DHM was originally developed as part of the MIT/UTAH hand project for telemanipulation of robots, and is constructed of aluminum alloy joints connected by rotating hall-effect sensors. The DHM measures 20 degrees of freedom, and our tests showed that it could provide sampled data corresponding to joint angle with 10 to 12 bits of accuracy per joint at rates exceeding 100 samples per second.

The *Bug-Mudra* System

The complete hardware configuration for *Bug-Mudra* is shown on the following pages. The gestural mixer portion is implemented on two computers, an IBM PC-Compatible 80386 machine and a Macintosh. The PC performs the first level of gestural analysis, parsing the enormous volume of raw data produced by the DHM into a relatively small set of commands and parameters. The Macintosh runs Hyperlisp and is responsible for interpreting the commands and parameters and turning them into control of the two DMP-7's.

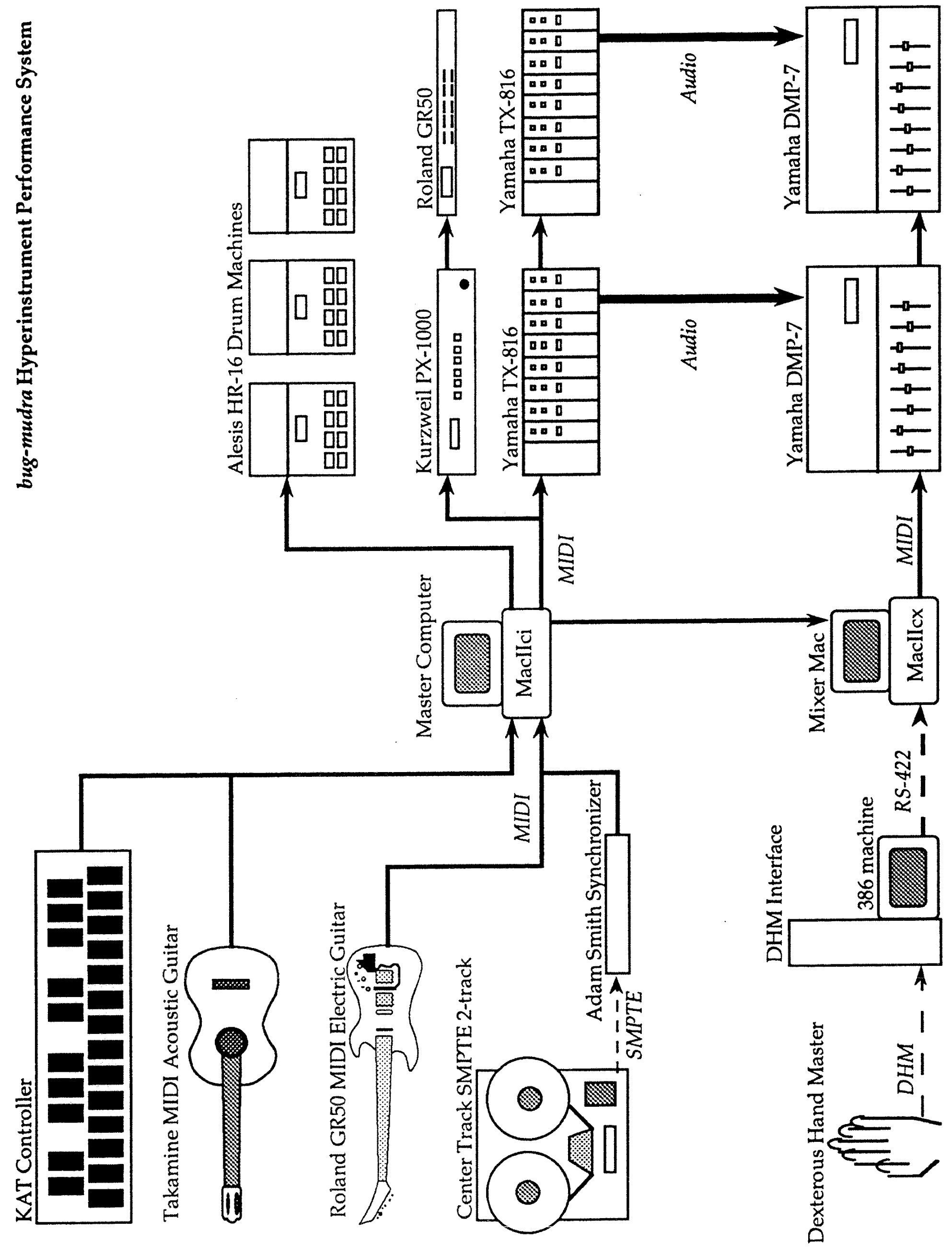


Figure 2.1: *Bug-Mudra* Performance Hyperinstrument

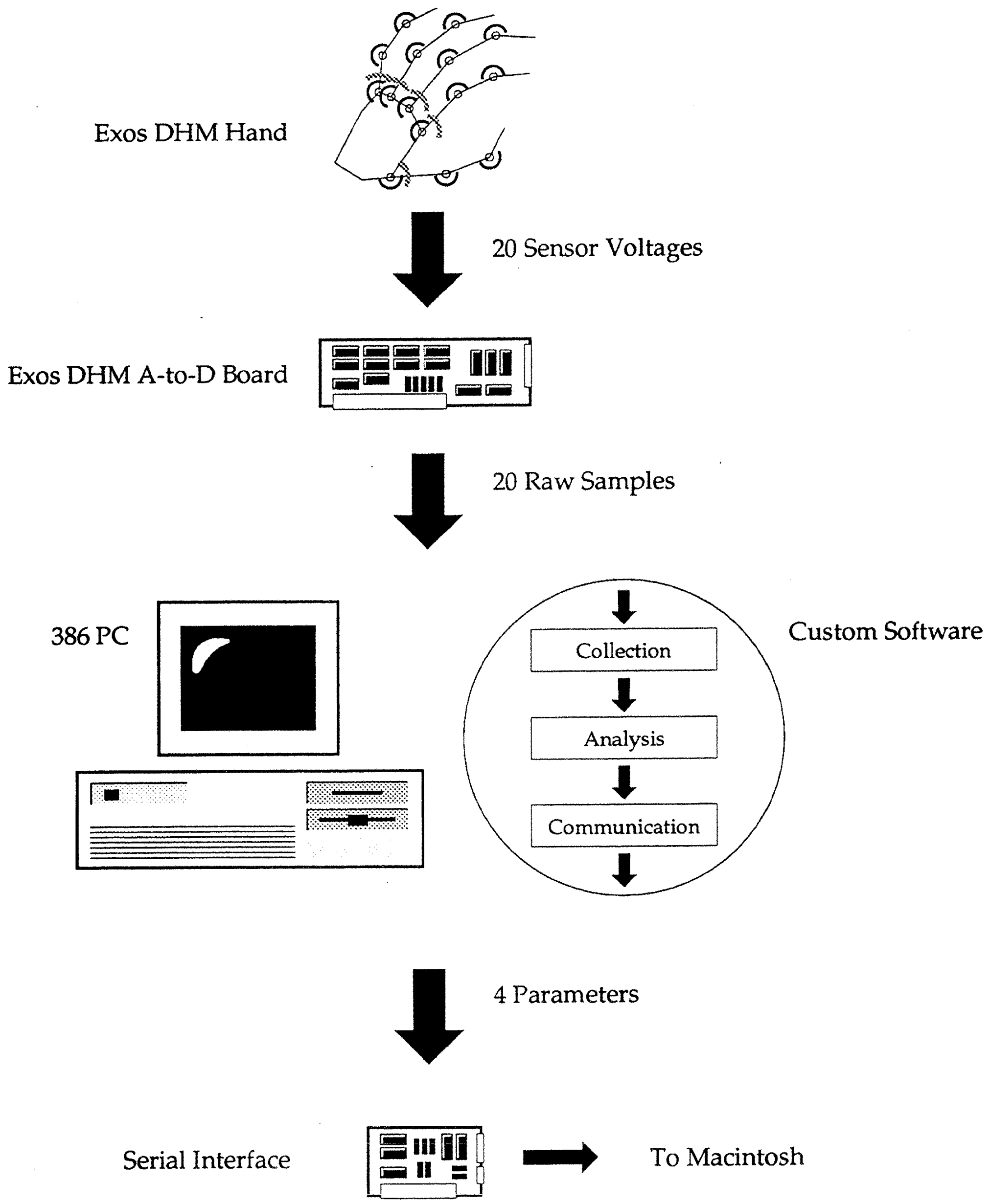


Figure 2.2: Gesture Interpreter

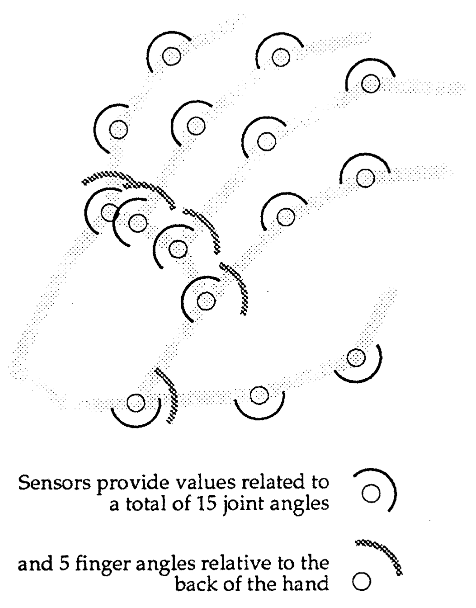

Sensors provide values related to a total of 15 joint angles
and 5 finger angles relative to the back of the hand

Figure 2.3: DHM Measurement Domain

The Dexterous Hand Master connects directly to an Analog-to-Digital board in the PC which samples data from each of the twenty hall-effect sensors. The program *ExosIsr*, reads this sampled data and fills a buffer with 20-measure sample frames. The board supplies each measurement as a 12-bit signed integer, with a range from -4096 to 4095, corresponding to the value of the sampled hall-effect sensor. ExosIsr rounds off one bit before storing the sample as a 12-bit unsigned integer with a range from 0 to 4095. Once all twenty sensors on the DHM are sampled and stored as a complete frame, the process is repeated. Although the sample rate can be set within a wide range of values, after extensive empirical testing, it was determined that for *Bug-Mudra*, a collection rate of 10 frames (200 sensor samples) a second was optimal. Once initialized and started, ExosIsr continues to execute in the background, filling the collection buffer with samples.

As ExosIsr collects samples, another program *Mudra* analyzes these samples and reduces the raw data into useful information. Each frame of raw samples is normalized to a set of curves, which are determined during an initial calibration procedure when the operator of the DHM is asked to flex the hand to minimum and maximum extensions so that the minimum and maximum values for each of the sensors can be determined. With these values, a calibration function is determined to provide an exponential output characteristic based on:

$$y = \exp(A \times (\mathrm{Lskew}(x) + 1/2))$$

in which x is the raw sensor sample and *Lskew* is a linear skew function that normalizes x. Figure 2.4 shows a typical input/output function. Notice that A determines the slope of the function in the middle range and therefore, the sensitivity of the sensor within the middle range.

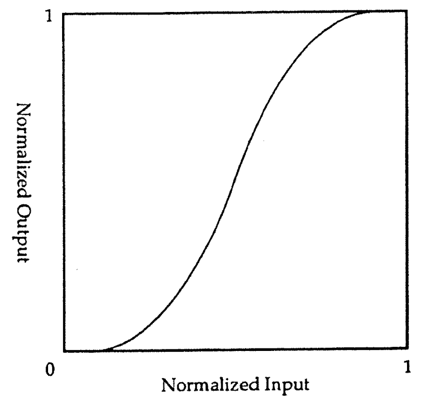


Figure 2.4: Sample Calibration Curve

A function of this form was used to create the calibration curve because it can emphasize the sensitivity in the crucial range where the joint is neither open nor closed, detenuating the boundary regions where the human finger has less control. In practice, a small factor *A* creating only a slight emphasis in the middle range provided the best results.

Gestural Language

Our original intention for first level analysis of the normalized glove data was that a certain number of hand positions called *postures* would be recognized which would be used to set the analysis mode. The various hand movements in between recognized postures would be transmitted as a series of mode specific parameters. This scheme would enable the analyzer to interpret the same finger movements quite differently depending on the mode. Postures are learned by recording a frame of normalized joint values while the DHM user is asked to hold the hand in position. Recognition occurs by comparing current joint values with those of each successive posture stored in a lookup table. If the values differ only within the error range, then current hand position is recognized as that posture.

In practice, this method failed to provide consistent recognition. Because individual joint angles within a given hand posture can differ greatly, an error range significantly large enough to compensate for this difference severely limited the number of distinguishable postures. Instead, we opted for using a single mode system whose four parameters indicated the amount of curl of the thumb and the first three fingers. This scheme caused more of the interpreting work to occur on the Macintosh than we originally envisioned.

3.5 The Gesture Mixer Instrument

Communication takes place between the analysis program on the PC and the interpreter program on the Macintosh via a RS-232 serial byte stream at 9600 baud. Each byte sent from the PC to the Macintosh is either a mode byte or a parameter byte. A mode byte has the high order bit set, and a parameter byte has this bit clear, giving 128 possible modes (although we use only one) and 128 possible parameter values. Parameters are not explicitly numbered or identified.

The serial stream is read on the Macintosh by the *gesture-driver* object. As mode and parameter bytes arrive from the PC, the gesture-driver forwards this data to the current *gesture-handler*. This handler is a mini-program representing the current mode of interpretation of the glove data and how this data is mapped to mixer movements. The collection of handlers is managed by the top-level object, known as the *gesture-mixer-instrument*. This object is responsible for installing the correct handlers at the appropriate times during the piece under the direction of the master hyperinstrument computer (*i.e.* the other Macintosh). Mode control is implemented through MIDI program change messages. In certain modes, the mixer Macintosh also interprets continuous controller data from the main Macintosh as an additional input to the mixer, allowing hyperinstruments on the main Macintosh to influence the mixing, when desired.

Mix Master

We found it useful to work at a level of abstraction above that of the basic mixer. Whereas in traditional mixing, the correspondence between musicians and sound sources tends to remain fixed, in a hyperinstrument piece sound sources (*i.e.* synthesizers) will generally be grouped in many different combinations at different times. This demands that the mixer provide the ability to group sounds based on their function in the piece, and not based on the physical inputs to the mixer. We therefore defined a new abstraction of the mixer which we called a "mix-master." In this abstract model of mixing, a mix consists of a set of *units*, each of which is some selection of any of the mixer inputs on either DMP-7. A single unit acts like a "super fader" and affects all the inputs it contains. For example, at one point, one musician might be playing a hyperinstrument which generates notes on the first four banks of one TX816 synthesizer and a drum synthesizer, while a second uses the remaining four banks of the TX816 and a sampler. In this case there would be two units, one for each musician. Perhaps later, the first would be using all eight banks of the synthesizer, while the second played through the drum machine, and the sampler was run by an autonomous hyperinstrument. The mix-master abstraction allows us to control the sound sources as the piece unfolds without needing to remember which actual inputs the musicians are using.

Normally each input in a unit has the same volume, pan, etc, but one may also specify relative offsets for each of these, so that, for instance, the first input of the unit will always be louder than the remaining ones. This allows diverse sound sources to be combined without one overpowering the others. In addition, the units can be assigned a minimum and maximum level. Since a unit combines many inputs, we define a new mixing attribute called spread, which is a relative angular offset in panning. When spread is at a minimum all inputs are panned in the same place, at a maximum, the inputs are spread through the full angular range, and pan has no audible effect

The units for a given mix along with the rest of state of the mixer are combined to form a mix-master *setup*. A setup is a complete specification for the state of the mixer, including the equalizations and internal effects for the mixers. The mix-master can be reconfigured for a new section of the piece by asking it to adopt a new setup. The new setup can be installed instantaneously or over a specified duration of time, thus allowing for gradual changes.

Gesture Mixer Modes

As previously mentioned, the mode of the gesture-mixer-instrument is dictated by the currently installed gesture-handler, a mini-program which maps the hand data from the PC into mix-master manipulation commands. *Bug-Mudra* required two main gesture-mixer-instrument modes, a basic grouped mixing of inputs, and a timbral mixing mode basedon the shape of the hand.

The grouped mixing mode is analagous to grouped VCA mixing on a conventional mixer, with the addition of grouped panning, equalization, effects, etc. We experimented with different models of mixing with hand gestures, primarily concentrating on *relative* vs. *absolute* mappings between finger positions and mix levels. In the relative mixing model, the range of finger motion is divided into three zones: mix level increasing, mix level decreasing, and neutral. When the finger is in the neutral zone, the unit associated with the finger remains unchanged. If the finger is uncurled into the increasing zone, the mix level begins increasing at a rate depending on the amount of curl. Likewise, when the finger is in the decreasing zone, the mix level is decreased.

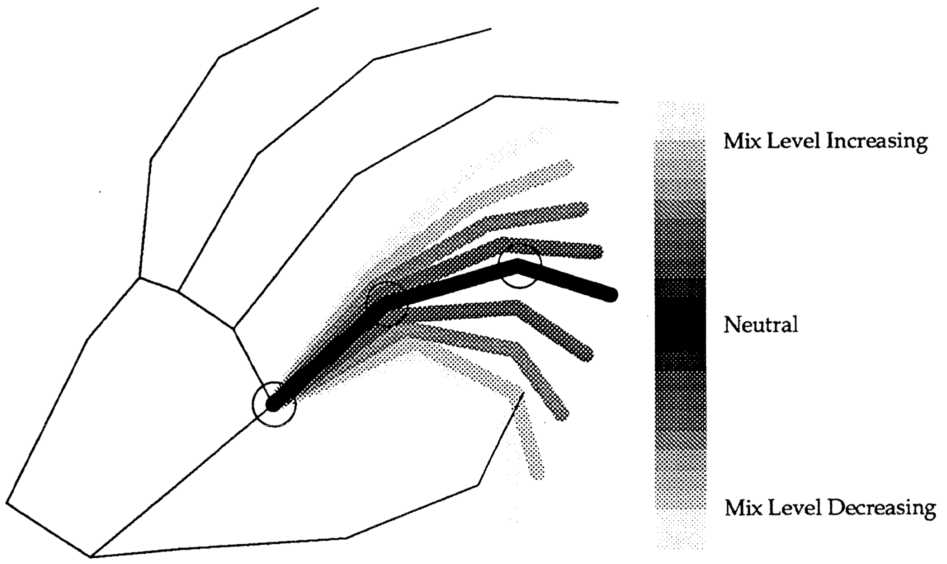


Figure 2.5: Relative Mapping

We found that in performance, particularly when there is no visual feedback of the levels being mixed, it is difficult to control more than one level simultaneously with relative mapping. Additionally, careful control over the rate of change of the mix levels requires rather delicate movements which are not suitable for a stage performance. We opted instead to use an absolute mapping model where finger curl is absolutely mapped to mix level. This offers the advantage that the performer always knows exactly the level of each unit, and can obtain a known mix (*e.g.* all units silent) immediately. The rate of change of mix level is also more easily controlled since it is exactly the rate of change of the fingers themselves.

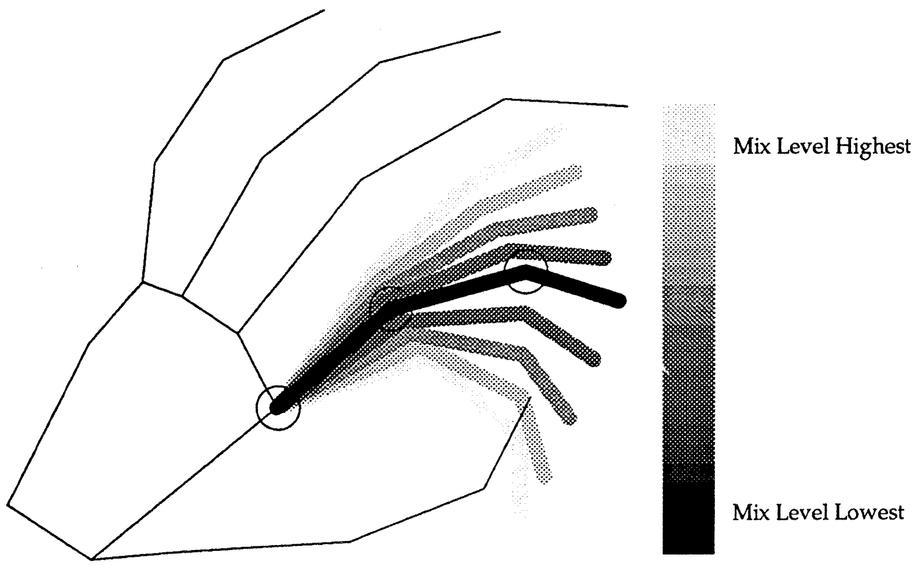


Figure 2.6: Absolute Mapping

The group mixing model is used in various movements throughout *Bug-Mudra*. Because of the re-assignability of units described above, the mapping between fingers and musicians (*i.e.* electric guitar controlled by the index finger, and percussion controlled by the second finger) can remain constant even though the actual sound sources mixed are different. The second major mode used in Bug-Mudra is known as timbral mixing, where a complex timbre generated from the percussion instrument is mixed and manipulated by the shape of the hand.

The timbral mixing mode is an extension of group mixing, where the three fingers are assigned units corresponding to the harmonicity of a complex additive spectra generated on the Yamaha TX-816's. The units are assigned so that the purest tones are associated with the index finger, more complex sounds associated with the second, and the most complex associated with the third finger. The thumb is used to control panning of the purest tones. Thus, the shape of the hand is used to mold the swirl the harmonicity of the complex timbre. When the index finger is held straight and the others bent down, the sound is pure, while the opposite position with the third finger straight yields the most inharmonic timbre.

3.6 Future improvements

Our limited implementation of gestural control immediately suggests a number of possible improvements. Both posture and gesture recognition—robust and stable enough to be used in real-time performance—will add power and functionality to the system. The current four-finger tracker lacks the potential to go beyond the mix-master model. On the other hand, a full gestural controller should be able to control discrete, continuous, switched, and modal setups. In effect, a detailed sign-language could be created as a means of control.

Additionally, these analysis techniques, both existing and planned, could be extended to measure and track other body dynamics. Instead of just the fingers, the movement of the wrist, the arm, or even the whole body could be used to control hyperinstruments. A generalized feature analysis system would be limited only to the availability of devices to measure body or body-part position. We have no doubt that the PC executing the Mudra software has the speed and power to handle the expanded gestural control. We estimated that the current software utilizes less than 20% of the computing power of the 25-MHz 80386 machine.

Year Three

The String Hyperinstrument

4.1 Hyperinstrument Architecture

The software work of 1990/91 focused on a number of specific goals, especially the development of systems which use non-MIDI string instrument controllers. The software goals can be summarized as follows:

- Re-implementation of development environment in CLOS.
- A software architecture for interpretation of continuous, non-event oriented data.
- Support for DSP coprocessors and data acquisition cards.
- Support for latest generation of MIDI hardware and software.
- Maximum portability to other platforms, especially high-performance UNIX workstations.

Common Lisp Second Edition

The Hyperlisp development environment has been completely re-implemented for Common Lisp Second Edition which uses the Common Lisp Object System. We are now programming on the Macintosh IIfx computer (40 MHz 68030 CPU). In general, we are very pleased with the new object system which is simultaneously more powerful in modelling musical constructs, and more efficient than Objectlisp. The re-implementation of our Objectlisp code also afforded us the opportunity to clearly define the relationships between modules, resulting in a very clean and stable high-level layer of code for building new hyperinstrument systems.

Interpretation of Continuous Data

Perhaps the largest programming effort in the general development environment during the past year was the creation of an architecture for acquiring and interpreting several streams of continuous data coming from specially built string instrument sensors. The nature of this gestural data differs from MIDI in two important ways. First, MIDI data from a controller is very definite, leaving little or no room for interpretation. A MIDI note-on message *means* that a new note has been played. Even if a MIDI controller is not 100% accurate, e.g. a pitch-tracking guitar controller, the controller is entirely responsible for interpreting the gestures, and there is no protocol for expressing ambiguity. Secondly, MIDI data is generally event oriented so that new information is transmitted when the controller determines that a new event has occurred and is not directly related to a previous event. Even MIDI continuous controller data is usually treated as a "level" setting protocol. It is rarely meaningful to interpret the shape over time of this data.

Conversely, the string instrument data is entirely uninterpreted. It is the result of raw measurements of various physical parameters such as bow position, wrist angle, sound volume, etc. The interpretation of this data must take place in Hyperlisp by correlating several different streams. In addition, the parameters must be periodically sampled by Hyperlisp and tracked over time; the instantaneous reading of any single stream is not very meaningful. To further complicate matters, the data is not entirely accurate. Noise, hysteresis, and other unpredictable interactions contribute to the unreliability of each data stream in different ways.

Our initial approach to these problems was aimed at developing a kind of "super-MIDI" protocol in which a single software module would be responsible for sampling and interpreting the data and producing a set of standard messages. The advantages to this system are similar to the advantages of MIDI: the raw data is converted to a high-level, abstract language which can be analyzed in a uniform way.

Eventually, we abandoned this approach for a number of reasons. First, we realized that different kinds of hyperinstrument programs need to be capable of interpreting the raw data in radically different ways. It is not at all obvious that a super-MIDI language for strings is practical or even possible. Secondly, we realized that it is not generally necessary or even desirable to fully interpret the data before using it. It is often the case that the information one is most interested in, such as the subtle expressive nuances of string playing, are eliminated in the process of reducing the data into an abstract form. Instead, the raw data can be passed through non-interpretive computation such as digital filters, scalers, etc. and mapped directly into the hyperinstrument program.

The architecture we chose to implement left the responsibility of data interpretation to the specific hyperinstrument program, essentially maximizing the flexibility of the different systems while minimizing overhead computation. An object known as a *hypercello* is responsible for sampling all of the data and storing the samples in circular buffers representing five seconds of data. The ultimate interpretation of the gestural data into a final musical result is accomplished by the currently active hyperinstrument program. These programs are known as *modes*, and they inherit from the *cello-mode* object. A library of objects was implemented to perform useful interpretation tasks, such as detecting tremolos, finding attacks, measuring bow range over time, etc., which are available as *mix-ins* to the modes. A mode uses the library objects by CLOS object inheritance, and the functionalities of the different library objects can be mixed together in any combination. The relationships between these objects are shown in Figure 4.1.

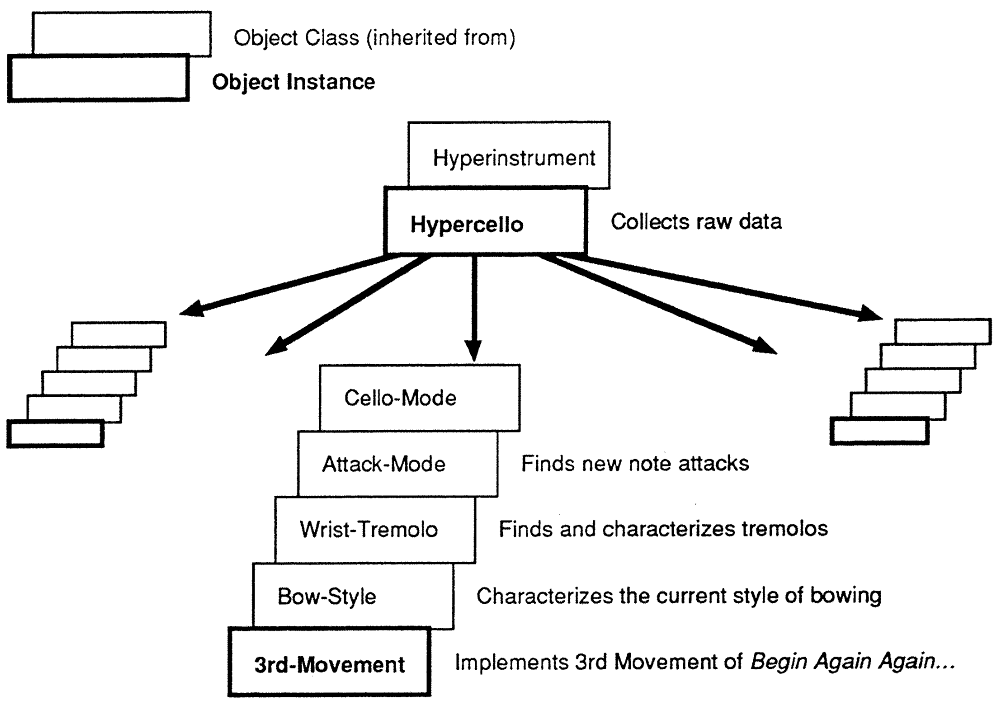

Object Class (inherited from)
Object Instance
Hyperinstrument
Hypercello
Collects raw data
Cello-Mode
Attack-Mode
Finds new note attacks
Wrist-Tremolo
Finds and characterizes tremolos
Bow-Style
Characterizes the current style of bowing
3rd-Movement
Implements 3rd Movement of Begin Again Again...

Figure 4.1: String hyperinstrument software architecture.

In the above example, taken from *Begin Again Again...*, the *3rd-Movement* mode uses the mix-in interpretation objects *Bow-Style, Wrist-Tremolo,* and *Attack-Mode,* to extract some basic interpretations from the data collected by the hypercello object. These interpretations are used along with further processing on the data streams by the mode object to create the desired mapping. Other modes in the piece use different combinations of the mix-in objects according to their requirements. The specific techniques used in the interpretation objects and the modes are described in later sections.

Data Acquisition and DSP Coprocessor Support

The design of hyperinstrument systems for strings necessitates the ability to acquire data from non-MIDI sources, in particular, analog sensing data from specially built hardware and analog audio signals. A number of commercial products exist for digitizing these types of data, and we have chosen to interface two widely used and relatively inexpensive devices which plug into the NuBus slots of the Macintosh II family computers. Analog sensing data is acquired through the General Instruments Macadios II/16 card, and audio is digitized and analyzed on the Digidesign Audiomedia card. These are interfaced in Hyperlisp through two objects, *macadios* and the *sound-axe.* The macadios object supports polled reads of any of its sixteen channels, and can set various control voltages and digital signals. The control of the card is implemented through writing and reading memory mapped locations.

The Audiomedia card is a complete DSP computer based on the Motorola 56001 microprocessor (see Section 4.4). The sound-axe object uses drivers in the card's ROM to down-load a DSP program to the card and communicate with it. Communication takes place synchronously through the card's host-port, or alternatively, through direct memory access to the card.

Support for Latest MIDI Systems

Recent commercial introduction of multiple port MIDI routers, such as the Mark of Unicorn MIDI Time Piece and Opcode Systems Studio 5, have allowed direct connectivity with up to 16 MIDI output ports for a total of 256 MIDI channels. Hyperlisp now supports these extra capabilities by extensions to our own custom MIDI driver. Hyperlisp also supports the Apple MIDI Manager, and the software interface between the low-level MIDI routines and the rest of Hyperlisp have been clearly defined. At present, the MIDI Manager cannot be used with the multiple port routers, so Hyperlisp must be pre-configured to use either one system or the other.

Portability to Future Platforms

Portability remains one of the most important goals, particularly as low-cost, high-performance RISC workstations become increasingly suitable for multi-media applications. The hyperinstrument environment is now clearly separated from the low-level MIDI driver layer which must define a small number of primitive functions for MIDI reading and writing, and a 1/100 second timestamping clock. With minimal effort, it should be possible to run Hyperlisp in any complete Common Lisp, Second Edition implementation, in particular, on a UNIX workstation in Lucid Common Lisp. We intend to make a version which can run on the Digital DECstation and Silicon Graphics Indigo workstations in the coming months.

The Hypercello

During 1990/91, much effort was made to create an integrated hyperinstrument system centered on string instruments. This project has expanded both the technical and musical bases of the hyperinstrument system. Research this year has concentrated on the cello, although subsequent work will extend and expand the model to all members of the string family. Working on string instruments has been especially interesting since they have provided the model for the first hyperinstrument to incorporate many aspects of the control instrument's acoustic signal. In addition, the attempt has been made to measure performance gesture of a much higher degree of complexity than in previous cases. Experiments were made with different kinds of cellos, from a Stradivarius acoustic instrument, to an electroacoustic Raad instrument combining wood construction with built-in amplification and custom-designed sensing technology, to a Zeta solid-body instrument. Also, since this year's project was the first to attempt to invest control of an entire musical discourse into a single solo hyperinstrument, the number and type of musical parameters controlled, and the level of mapping of performance intention to sonic result, were significantly more sophisticated than in previous models. In working on the development of the hypercello, our group was fortunate to have the collaboration of cellist Yo–Yo Ma, who followed the project closely beginning in Fall 1990, through his premiere performance of Tod Machover's new hypercello composition, *Begin Again Again...*, at the Tanglewood Festival in August 1991. Mr. Ma was instrumental in helping to evaluate the efficacy and invasiveness of sensing technology, the appropriateness of mapping gesture to musical result, and the integration of hyperinstrument control to musical intention and performance expressivity.

4.2 System Configuration and Basic Functionalities

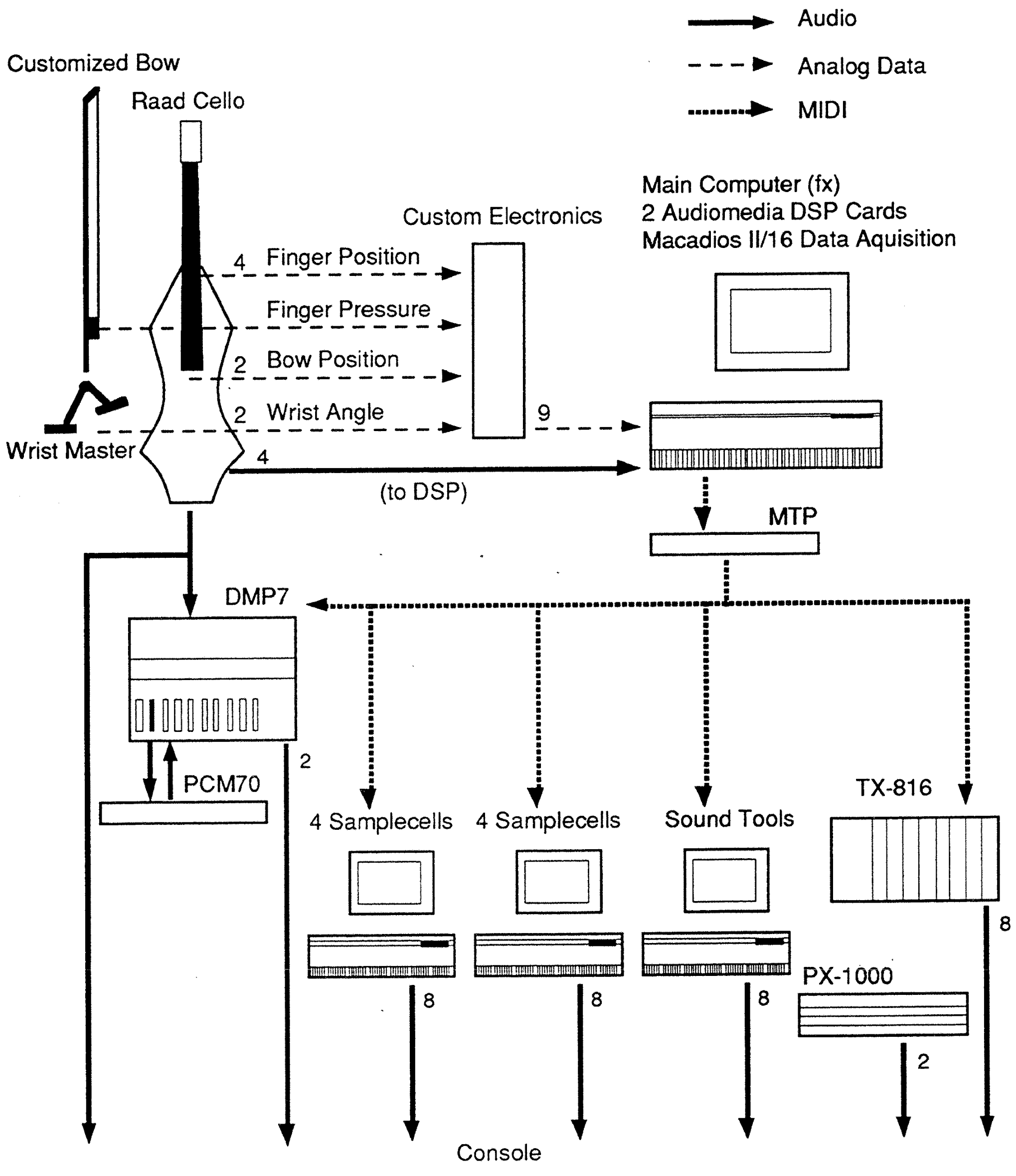


Figure 4.2: Hypercello system.

Figure 4.2 depicts the hypercello system configuration and the major data paths. The system is comprised of three main parts: the instrumented Raad Cello and bow, the sound generation and sound processing devices, and the main computer which runs the Hyperlisp system and maps in between the cello and the musical result. In general, information from the cellist is interpreted and transduced three times before becoming recognizable audio: from physical motion to analog sensing data, from the digitized sensing data to MIDI, and from MIDI to audio. The final transformation from electronic audio to acoustic is made through conventional stereo loudspeakers using a concert hall sound system with a multichannel mixing console.

The first stage of transformation, from physical motion to analog sensing data requires the integration of existing measurement devices and the construction of new devices for measuring as many of the important gestural parameters of cello playing as possible. The parameters measured are listed below, and discussed in detail in the following sections:

- Right hand wrist angle in two dimensions (flexion and deviation)
- Finger pressure on the bow
- Bow position in two dimensions
- Left hand finger position on strings
- Loudness of each of the four strings
- Pitch of each string

The measured parameters are brought into the Hyperlisp system through two means. The Digidesign Audiomedia card is used for the loudness and pitch tracking parameters, and the General Instruments Macadios II/16 data acquisition card is used for all the others. The hypercello object communicates with both cards and samples the parameters once every 10 milliseconds (100 Hz).

The second stage of transduction is the mapping of the digitized parameters to MIDI control of the sound production and processing devices. As discussed above, this mapping is determined by the currently active cello-mode object which generally uses a combination of library objects to assist in interpreting the data. The cello-modes also use a number of library objects to create the desired MIDI output, including sequencer processes, mixing and digital effects managers, and event triggering objects.

The last stage of transduction in the hypercello system is the mapping from MIDI data to electronic audio. This stage consists of a large array of sample playback devices, synthesizers and digital effects processors connected to Hyperlisp through the Mark of the Unicorn MIDI Time Piece (MTP). The MTP is an eight port MIDI router, allowing access to sixty-four MIDI channels. The other equipment used is as follows:

- 8 Digidesign SampleCell sample playback units with 8 megabytes of RAM each
- Yamaha TX-816
- Yamaha DMP7
- Lexicon PCM-70
- Kurzweil PX-1000

4.3 Developments in Cello Sensing Technology

The goal of the cello sensor development was to develop techniques to unobtrusively measure (1) the physical inputs from the player to the cello, and (2) gestural information that gives clues about the player's intent. The final choice for the set of parameters that was used evolved through the interaction between the technical sensor research, cellist's evaluation of the sensors, and the composer's musical goals; it included the bow pressure, bow position (transverse to the strings), bow placement (distance from the bridge), bow wrist orientation, and finger position on the strings.

Interfacing a virtuosic player led to some unusual constraints on the sensors, including:

- The absolute calibration does not matter, but the relative precision must be very good.
- The response must settle within ~ 1 msec.
- The sensors must be easy to install and remove,
- Reliable and robust,
- Lightweight.

In many cases, very simple inexpensive solutions proved to meet these requirements better than much more complicated approaches. The implementation and context of each of the sensors will be described in detail.

Bow Position and Placement

Many standard techniques for measuring distances in this range (1 mm – 1 m) were investigated and rejected, including:

- *Acoustic sonar* (measuring the propagation time of an ultrasonic sound pulse in air). The problem with this is that it is (1) bulky, and (2) very difficult to do it without coupling some energy into the audible range.
- *Acoustic phase* (measuring the phase shift in the reception of an audio signal). This has the same problems as sonar.
- *Infrared strength* (measuring the fall-off in signal strength between an IR diode and detector). This can be done compactly, but requires maintaining a direct line-of-sight between the source and detector, and is sensitive to stray reflections.
- *Inductance proximity* (measuring the eddy-current coupling out of a coil). This is limited to distances that are too short (~ 1 cm).
- *Microwave reflectivity* (measuring the reflected microwave signal strength) This requires a well-characterized target to be reliable.

Following this unsuccessful search, we developed a new technique that proved to be simple and effective (a patent is being filed). The idea is illustrated in Figure 4.3.

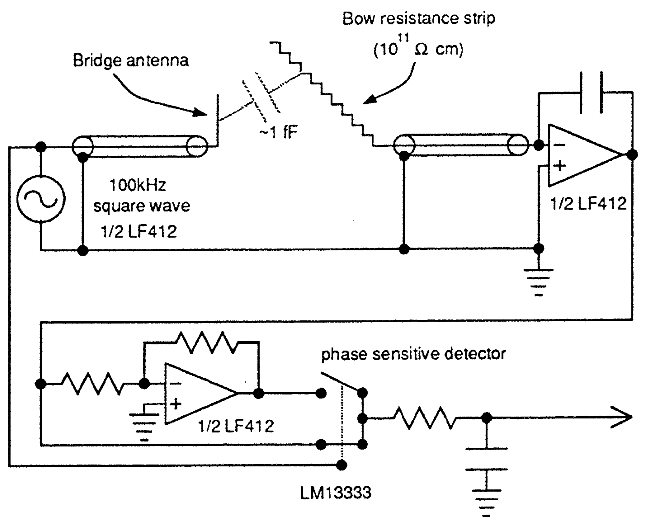


Figure 4.3: Diagram of bow position measurement device.

There is a short (~ 3 cm) drive antenna oriented vertically on the cello bridge, and a thin receiving antenna (~ 15 cm) mounted horizontally on the bow. At the drive frequency (~ 100 kHz) the corresponding wavelength is 3 km, so the coupling between the antennas is purely capacitive; this capacitance will depend on the distance between the two antennas. The capacitance is on the order of 1 fF; the drive frequency is chosen by the trade-off between decreasing impedance and increasing noise figure as the frequency is increased. The electric field potential was determined by numerically solving Laplace's equation with a relaxation technique. Figure 4.4 shows a two-dimensional slice of the potential for a typical configuration.

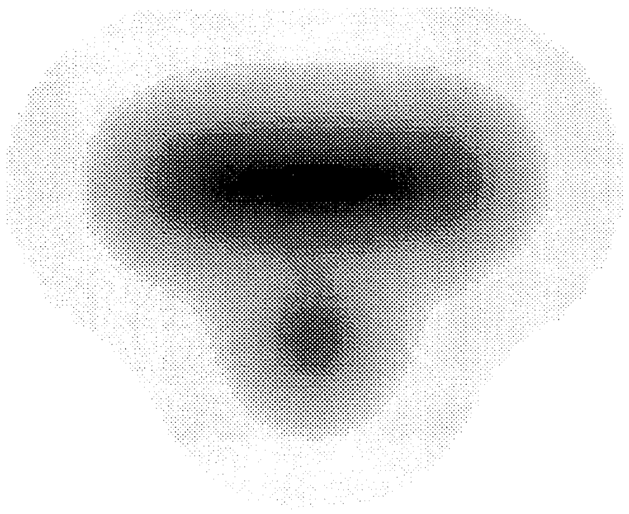


Figure 4.4: Cross section of electric field potential around modified cello.

The capacitance was found by integrating around the antennas to find the charge. Figure 4.5 shows the capacitance versus distance.

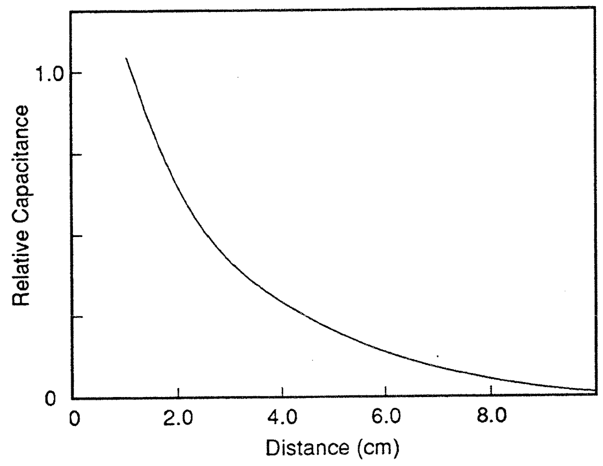


Figure 4.5: Relative capacitance measured using bow position device.

In order to determine the bow position as well as placement, a resistive strip (M–411, 101^1 Ω–cm, Mitech, Twinsburg, Ohio) was used for the receiving antenna with the resistance was chosen to be roughly comparable to the impedance between the antennas. The impedance between the antennas is measured with a simple phase-sensitive detection scheme in order to minimize interference. The impedance will vary both as the bow is moved further from the bridge (the capacitance between the antennas), and as the bow is moved transverse to the strings (the resistance of the receiving antenna). These two degrees of freedom can be separated by either the phase shift (one is 90 degrees out of phase with respect to the other), or by measuring out of both ends of the receiving antenna. The latter approach was chosen because the drive circuitry is simpler, although it does require one more cable. Miniature coaxial cables (47 mil OD, 29.5 pF/ft, AS 450–3650SR, Cooner, Chatsworth, CA) were used to maintain isolation without significantly loading the bow. The position calibration was done by numerically fitting a surface to the measured signals as a function of the bow position.

Bow Pressure

An initial decision was made to measure the pressure under the player's fingers, rather than by the tension in the bow hairs. This was because (1) it does not require re-hairing the bow, and (2) the finger pressure contains the same information as the bow hair tension, but can also be controlled independently (depending on whether the fingers torque or compress the bow). Here again, a number of standard methods were examined and rejected, including:

- *Piezoelectric.* Even with preamps with $10^9\,\Omega$ input impedance, the low frequency response still rolls off on the order of 1 Hz.
- *Force-Sensing Resistors.* These are commonly used in the music industry to sense force; the mechanism is the thickness dependence of the conductivity of a insulator/conductor composite just below the percolation threshold. The problem with these devices is that the relation between conductivity and pressure is both noisy and hysteretic.
- *Piezoresistors.* Aside from FSR's, piezoresistance is achieved with both foil materials and doped silicon. These materials are that they are usually designed to work in flexion rather than compression (for strain gauges), and it is difficult to shape them to fit the bow geometry.

A capacitance measurement was used once again to measure the bow pressure. A capacitor was built using a thin (35 mil) cellular urethane foam (4701–59–25035–1648, Poron, Rogers, CT), with a Young's modulus (9 PSI, 20% deflection) chosen to match typical forces from a player. The great advantage of this scheme was that the response range is determined simply by choosing a urethane foam with the desired modulus, the geometry is selected by cutting the foam to the desired shape, and the device can be easily bent to match the bow circumference. Copper tape (1181, 3M, Austin, TX) was used for the electrodes, and a third grounded shield electrode was needed to isolate the player.

The size of the sensor need not closely match the expected contact area. Assume a square sensor of side length L, with a thickness D, and then assume that it is depressed a distance d in a centered square area of side length l. If the potential between the electrodes is V, then the charge is found by integrating the divergence of the electric field around the electrodes. Neglecting fringing fields ($d,D << l,L$), the charge on one electrode is

$$\begin{aligned}4\pi Q &= \int \nabla \cdot \vec{E} dv \\ &= \int \vec{E} \cdot d\hat{a} \\ &= \frac{V}{D}(L^2 - l^2) + \frac{V}{d} l^2 \quad .\end{aligned}$$

The capacitance is then

$$\begin{aligned}C &= \frac{Q}{V} \\ &= \frac{1}{2\pi}\left[\frac{A}{D} - \frac{a}{D} + \frac{a}{d}\right] \\ &= C_0\left[1 + \frac{a}{A}\left(\frac{D}{d} - 1\right)\right] \quad ,\end{aligned}$$

where C_0 is the undeflected capacitance $A/2\pi D$. If the ratio of the deflection area to the total area is equal to the ratio of the undeflected thickness to the deflected thickness, the capacitance will increase by a factor of roughly 2.

String Position

The goal here is to measure the position of the strings at audio frequencies with very little crosstalk between the strings; this was done by inserting a piezo-electric polymer sheet under each of the strings where it contacted the bridge. The polymer used, PVDF (Sputtered NiCu, 28 μ Penwalt Atochem, Valley Forge, PA), was folded into rolls 5 layers thick in order to increase the signal strength; the output (~ 10 – 100 mV) was directly preamplified and the digitized.

These sensors are very easy to install, and the high-frequency response is extremely good, but they have a few limitations: (1) There is a small change in the coupling of the string to the bridge, which slightly affects the cello's sound, (2) there is a fundamental rocking mode of the bridge which leads to cross-talk between the strings when the instrument is bowed (when the strings are plucked the rejection is very good), and (3) the response is a function of force, rather than position, so symmetry information about the direction of bowing is lost. In a future revision of the instrument it is anticipated that induction or optical pickups will be mounted at the end of the fingerboard for an absolute, non-contact position measurement.

Finger Position

To measure where the strings are being stopped on the fingerboard, it is essential that the sensor not interfere with the motion of the left hand, and that the sensor's mechanical properties be well matched to those of the finger board in order to not affect the sound. Two initial approaches that were rejected were:

- *Sonar* (timing the transit time for pulses piezo-electrically launched down the strings). The times for this are convenient (~ 1 – 10 msec), but the complications introduced by the bowing of the string made this infeasible.

- *Fingerboard acoustics* (acoustically locating the point at which energy is coupled from the vibrating string into the finger board). This also works for a single, clean note, but the signal processing is prohibitive when the instrument is being played aggressively.

The final approach selected was to use strips of a thin (15 mil) hard solid-filled conductive thermoplastic sheet (M–411, 10^{11} Ω–cm, Mitech, Twinsburg, Ohio). The (metal-wrapped) cello strings were grounded, and then the resistance was read out when the string contacted the strips. The strips were mounted with a removable transfer laminating adhesive (3M, St. Paul, MN) that is similar to that used on *Post-it Notes* which permitted easy re-positioning, and electrical contact was made with a copper tape with an electrically conducting glue (1181, 3M, Austin, TX). This arrangement did not interfere with the playing of the instrument, although over time the impact of the strings abraded the thermoplastic, changing the calibration.

Wrist angle

The angles of the player's wrist (flexion and deflection) were measured with a device developed by Exos Inc. (Burlington, MA). This strapped onto the wrist and measured the joint angles with appropriately mounted magnets and hall effect sensors. A magnet measurement was used, instead of a potentiometer or optical encoder on the rotation joint, because it permitted the mechanical properties of the joint to be designed independently of the sensor. The hall sensors (PK 8796 0 SS9, Honeywell Micro Switch, Freeport, IL) had on-board conditioning circuitry so that the output could be directly digitized.

4.4 New Techniques for Real-time Digital Signal Processing of the Acoustic Cello

Many new digital signal processing techniques were developed to analyze and evaluate different aspects of the acoustic cello signal. One goal of using a signal processing subsystem was to develop an efficient and robust means of measuring the pitch of each note played on the cello.

Experimentation with Various Pitch Detection Systems

Before finalizing on the current hybrid DSP/physical-sensor scheme, we experimented with a number of pitch-tracking systems. Originally, we planned to use a Zeta cello as the controller, hoping that it would automatically provide pitch information. The Zeta is equipped with an external controller (built by Ivl Technologies) that maps the electro-acoustic signal from the cello body to MIDI note messages. Within a few months of use, it was determined that not only is its ability to supply MIDI pitch values worse than we expected, but its electro-acoustic sound is of rather poor quality.

Giving up on the Zeta cello, we moved our efforts to using the Raad cello with external pitch-trackers. The Roland CP–40 Pitch to Midi Converter was our first attempt at pitch tracking on the Raad. The device works well with very simple vocals and cello timbres, but its performance degrades rapidly when given any sort of harmonically complex timbre; the CP–40 was too unstable for use with cello sounds.

We then attempted to develop a general software-based pitch tracker that could analyze the electric signal from the Raad cello. S.D. Yadegari developed a pitch detection program based on a linear FFT [Yadegari 1991]. This algorithm first detected a certain number of peaks in the FFT bin and then used perception-based theory to determine pitch by examining peak values.

After executing an FFT on the incoming signal, an average and standard deviation of the values of the magnitude information in the FFT frame is taken. Then, a threshold value (a program parameter, set according to the timbre of the signal and the noise of the environment) is calculated according to a threshold factor. Magnitudes lower than the threshold are considered to be noise and are filtered out. All peaks are detected and an adjustment is made to the location and magnitude of the peaks according to an algorithm proposed in [Serra 1989]. The program then uses the peaks and their values to detect the position of the pitch according to the interdifference spaces between them.

An assumption is made that human pitch detection may be performed by detecting the pitch of the "missing fundamental" by using the beating between partials as a strong cue. If so, using the interdifference method for pitch detection has the advantage of finding the beating value as well as the highest common divisor in a very fast way. Every peak as well as every interdifference between two adjacent peaks suggest a new pitch. If a suggestion has already been made within a quantization factor (0.2 was used), then the new suggestion is averaged with the old suggestion weighted by their energies; and a counter is incremented. The suggestion with the highest number of counts wins as the pitch. Ties are broken according to the kept values.

Several factors made the hyperinstrument group abandon this method of pitch tracking as a viable real-time approach to acoustic cello analysis. First, although quite fast, the algorithm was not implemented on a platform compatible with the general hyperinstrument environment. Second, the current version of this pitch tracker keeps very little context from one FFT frame to another. Therefore, the pitch printing procedure keeps a single pitch context and deletes any single stray pitch which often occurs at very fast pitch changes.

With no success in building a robust signal-based pitch detector, the system to measure fingerboard positions was designed (see section 4.3). But tests with the fingerboard sensors proved that the design constraints (i.e., limited types of materials could be used for the fingerboard sensors) reduced the accuracy of this system; measurements of finger position were stable to within two or three semitones. At this point, we decided to focus our efforts at building a software-based pitch tracker that would take advantage of both the approximate fingerboard position data and the characteristic shape of a bowed-string waveform to fine tune its pitch measurements.

The Bowed String

The characteristic shape of a bowed string waveform is due to the effects of alternating static and sliding friction of the bow against the string. As the bow catches the string, the string is displaced until the static friction is overcome by the tensile force. The string jumps back suddenly, sliding underneath the bow and vibrating until the force of static friction once again overcomes the vibrational force. The dynamics of the static/sliding friction are coupled directly to the vibration of the string. Therefore, the period of the string's waveform is determined by the natural frequency of the string and the periodicity is determined by such factors as bow pressure and location.

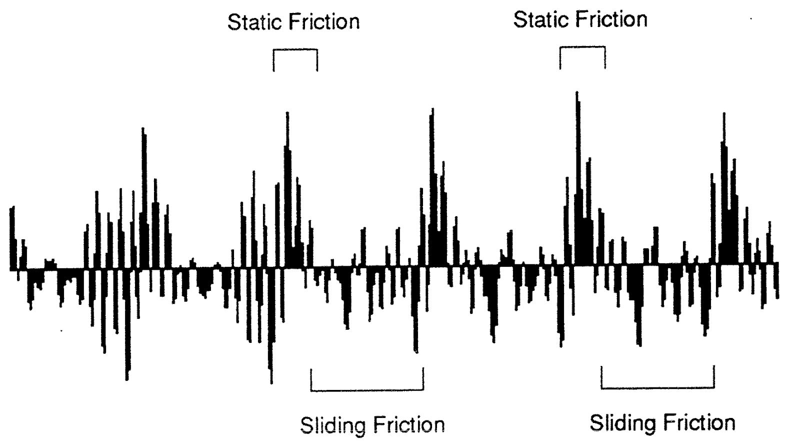


Figure 4.6: A typical cello waveform, with the static and sliding modes labeled.

We realized that we could use our understanding of this characteristic waveform shape to our advantage when developing our DSP subsystem.

The DSP Subsystem

Two Audiomedia cards (Digidesign, Inc., Menlo Park, CA), each equipped with a 22.578 MHz Motorola DSP56001 processor and two channels of CD-quality input and output (16-bit, 44.1 kHz), were used to implement the signal processing subsystem inside the Macintosh IIfx. A single program was developed to analyze energy and pitch, operating on both cards and tracking two channels of audio input for each card. A total of four input channels provided individual processing for each of the four strings on the cello.

The Pitch Tracker

The pitch tracker works in the time domain, taking advantage of the unique shape of the bowed-string waveform. The signal from each audio channel is first decimated to reduce the sampling rate to 22.05 kHz and then pre-filtered to enhance the fundamental frequency and reduce the strength of harmonics.

DSP Designer 1.2 (Zola Technologies, Inc., Atlanta, GA) was used to design four different low-pass filters, each corresponding to one string on the cello. The cutoff frequencies corresponding to the three lowest strings (C, G, D) were chosen to allow for one octave ranges, while the cutoff frequency for the highest string (A) was chosen to allow for a two octave range:

C string:	130 Hz	(C2 = 65.41 Hz	C3 = 130.81 Hz)
G string:	200 Hz	(G2 = 98.00 Hz	G3 = 196.00 Hz)
D string:	300 Hz	(D3 = 146.83 Hz	D4 = 293.67 Hz)
A string:	880 Hz	(A3 = 220.00 Hz	A5 = 880.00 Hz)

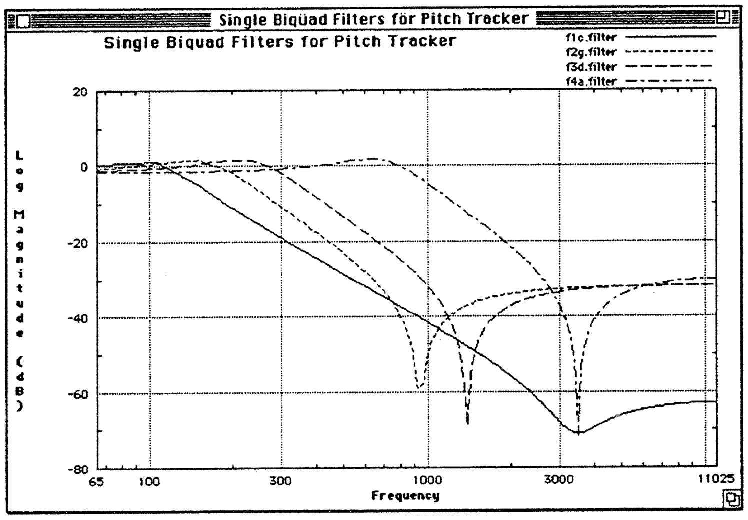


Figure 4.7: A screen dump from DSP Designer showing the four filter responses.

While operating on two channels, an Audiomedia card can execute about 120 multiply-accumulate operations per single 44.1 kHz sample. With the overhead of interrupt service routines providing multiple processes along with decimation, filtering, and tracking for two separate audio channels per card, this benchmark number is easily halved. In order to guarantee real-time processing with minimal latency, we decided to keep our low pass filters as simple as possible. We opted for single-biquad IIR filters providing 12 dB per octave of slope.

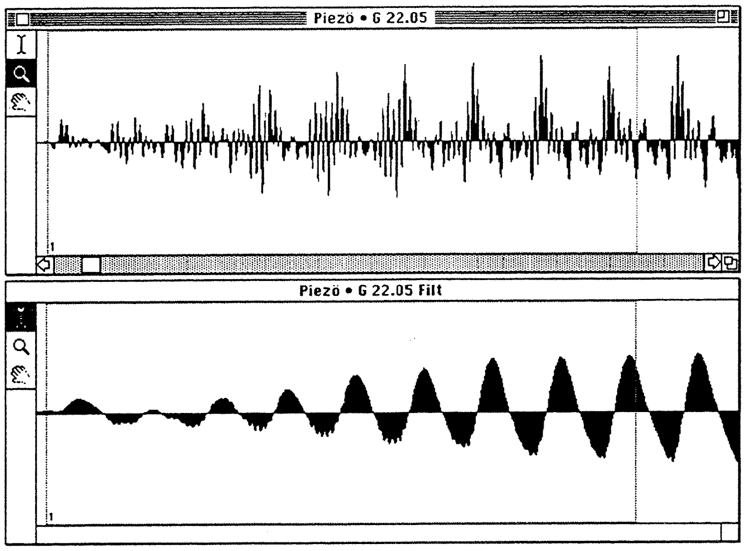


Figure 4.8: The effect of the G string filter on an open G note is quite visible.

The pitch tracker uses a period-length estimate P_f calculated from the fingerboard position as a starting point to compare sections of the time-domain, filtered waveform. Given the period estimate, the pitch tracker finds the largest peak in the waveform, looking back at the most recent filtered samples collected in the time period

$$P_{max} = P_f \times 1.15$$

where P_{max} is the maximum possible period given the estimate. Once it finds this peak, located at T_1, it looks backs in the range P_r, determined by P_{max} and the minimum possible period

$$P_{min} = P_f \times 0.85$$

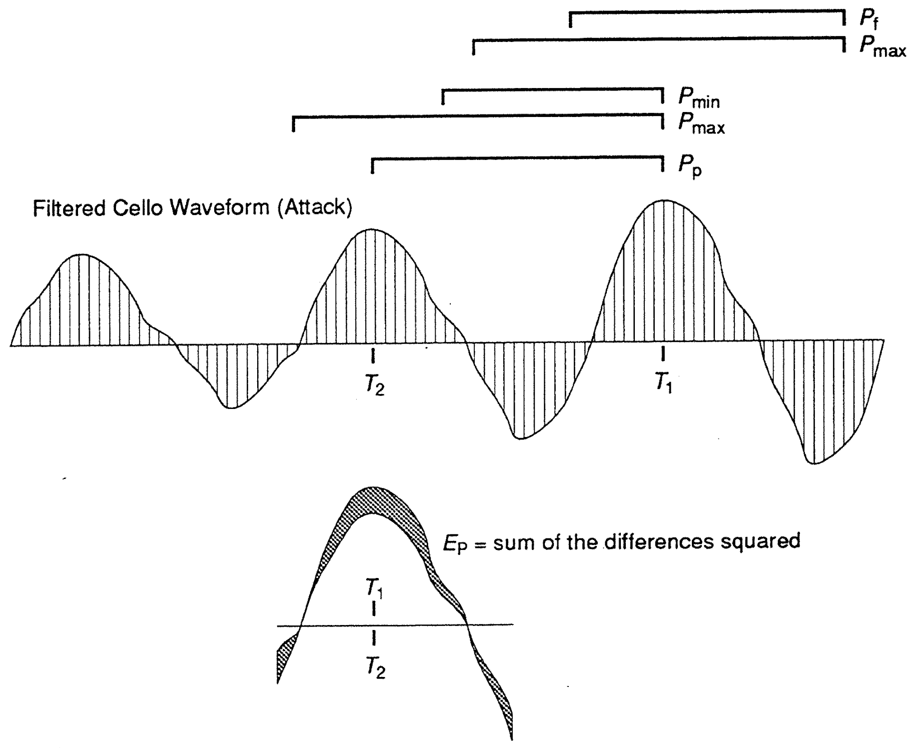
for the maximum sample value with the same direction as the peak at T_1. The shape of the waveform around this second peak, T_2, is compared to the shape of the waveform around T_1. Normalizing for amplitude and measured period

$$P_p = T_2 - T_1$$

the square of the differences for each of the correlated sample values of the waveforms around the two peaks is accumulated into an error value E_p.

The pitch tracker also attempts to correlate sections of the waveform for first and second harmonics, comparing the waveform at T_1 with that at $.5T_2$ and $.3T_2$.

This process is constantly repeated, and the P_p corresponding to the lowest E_p is saved, until a new P_f is provided by the host processor. When the host processor queries the pitch tracker for the current pitch, the pitch tracker returns both the lowest E_p and its corresponding P_p for the most recently specified P_f.



P_f
P_{max}
P_{min}
P_{max}
P_p
Filtered Cello Waveform (Attack)
T_2
T_1
E_p = sum of the differences squared
T_1
T_2

Figure 4.9: Pitch tracker parameters.

Energy Tracker

The energy tracker provides a reading of the volume level for each of the two input channels on an Audiomedia card by calculating the sum of all the sample magnitudes in a window of incoming samples. This process executes continuously for each decimated input sample. After normalizing the input sample, the sample value is added to the current energy level and the magnitude of the oldest sample in the energy window is subtracted from the current energy level. This value is stored until the energy tracker services the next input sample interrupt.

In order to prevent errors from the accumulation of DC offset and other types of noise, the energy tracker normalizes the output of the energy tracker each time the host processor queries the DSP56001 for the current energy value.

Bow Direction Tracker

Along with pitch and energy trackers, a bow direction tracker was implemented to work as an integral part of the energy tracker. Using the same windows as the energy tracker, the direction tracker measured and stored the peaks in the incoming sample while keeping the average signed value (as opposed to the energy tracker's magnitude measurements) of the samples in the input buffer. Taking advantage of the characteristic shape of a bowed string, by comparing the direction of the maximum peaks relative to the window average, the tracker could determine the direction of the bow stroke.

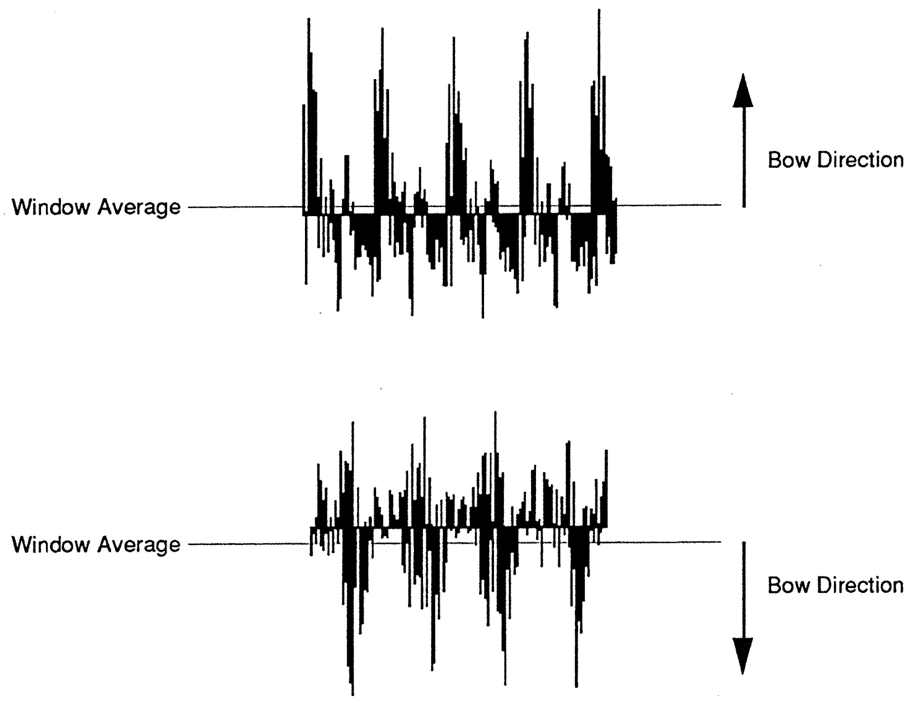


Figure 4.10: The bow direction tracker takes advantage of asymmetries.

Use of the window average as the zero value (as opposed to the actual sample value of zero) for determining the direction of the waveform peaks greatly reduced the detrimental effect of subharmonics and DC-offsets. This simple algorithm proved to be efficient and robust when used with piezo-electric pickups that were shaped so that the voltage changes were directional. For the final concert subsystem, we decided to remove the direction tracker from the signal processing subsystem in order to give the filtering pitch tracker more processing time, especially since bowing direction could be readily determined by the wrist sensor (see section 4.3).

Concurrent Processing

From the beginning, the DSP subsystem was designed to provide a framework for implementing different trackers that execute concurrently. In the current configuration, the pitch tracker runs as the main process on the DSP56001, while the energy tracker and other processes (*i.e.*, decimation and filtering) run at the interrupt level. Each sample that is received in either channel interrupts the main process. Bookkeeping and final calculations for the energy tracker, as well as prefiltering for the pitch tracker, are executed at a sample to sample level.

Host communication works at a second interrupt level. The host processor interrupts the DSP56001 whenever it needs to exchange data. While pitch and energy are being calculated constantly, the trackers only return data when queried by the host.

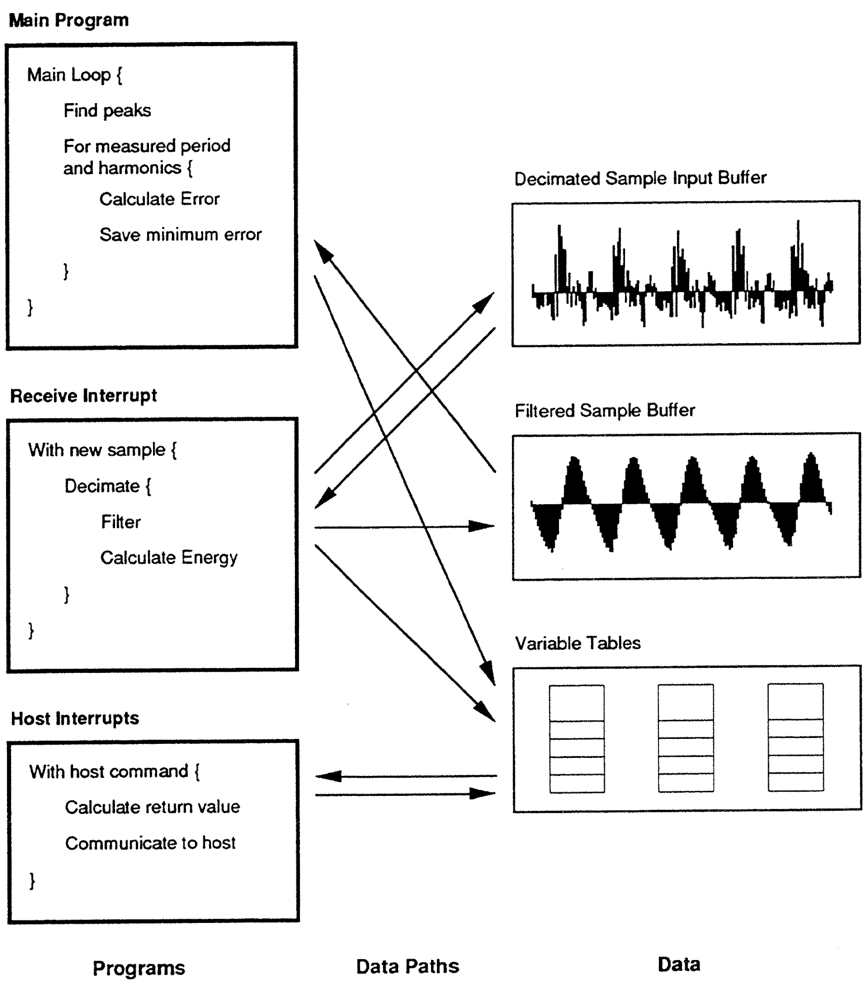


Figure 4.11: Concurrent processes within the DSP subsystem.

Furthermore, the host can signal the DSP subsystem to suspend execution of certain tasks in order to give other tasks more computing cycles or to facilitate debugging (*e.g.*, the state of the pre-filtered and post-filtered buffers can be frozen for analysis).

4.5 Gesture Tracking and Interpretation

Because of the great diversity in the kinds of data we receive, ranging from physical measurements of bow pressure or position, to samples of digital audio from the cello strings, it is not surprising that we could find no one simple method for correlating this data into a simple, abstract form. In addition, one is generally interested in a small subset of the feature which are possible to extract from the data streams. As previously mentioned, we implemented a number of interpretation objects which could be "mixed in" to the inheritance list of cello-modes to enable them to be responsive to specific interpretations of the data. The main interpretation objects are as follows:

- Attack-mode: looks for attacks by analyzing the loudness of the cello.
- Wrist-tremolo: detects tremolos in the wrist and measures the period.
- Bow-style: classifies the bowing style as either normal, tremolo, bounce bow (spiccato), or pluck.
- Bow-range: measures how much bow is being used over time.
- Cello-trigger: uses DSP pitch tracking to trigger events when certain pitches are played.

The interpretation objects are generally designed to be used under many different conditions and so possess a number of parameters that are adjusted for each cello-mode which inherits from it. In addition, the cello-modes themselves have full access to five seconds worth of each data, and can perform additional interpretation processing within itself.

Attack-Mode

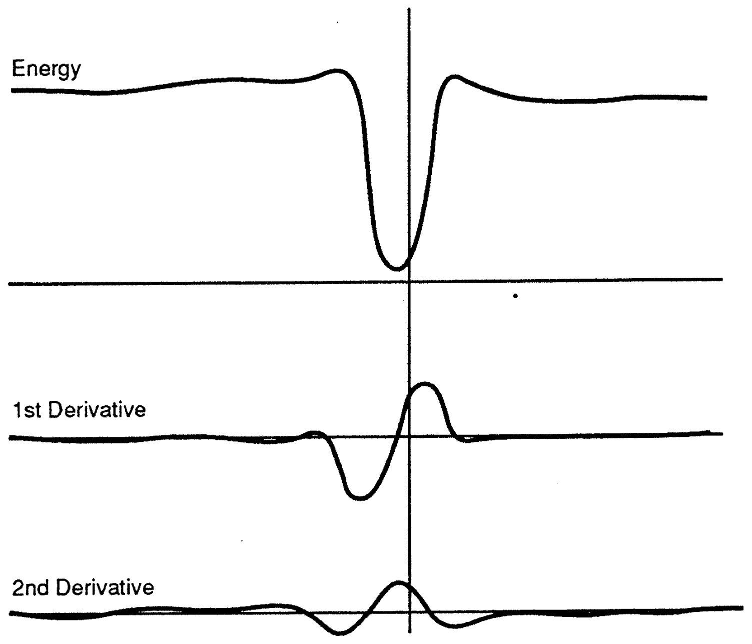


Figure 4.12: Attack Analysis.

Figure 4.12 depicts the calculation made in determining if a new note has been played by analyzing the loudness or energy of a string. The algorithm tries to find a steeply rising energy value, especially when directly preceded by a steep fall. The first and second backward differences of the sampled energy values are calculated, and an attack is interpreted when both the first and second differences are sufficiently positive. This condition generally occurs when the first derivative rises sharply, indicating a steep increase in the energy. Most notably, this condition occurs when a new note is bowed on the same string. In this case, as in Figure 4.12, the energy is generally steady until the new note is played. At this time, the stopping and restarting of the bow causes a rapid dip in the energy immediately followed by an equally rapid ascension. In this case, the first difference swings sharply from negative to positive. If the first derivative rises fast enough *and* high enough, the attack is detected.

Wrist-Tremolo

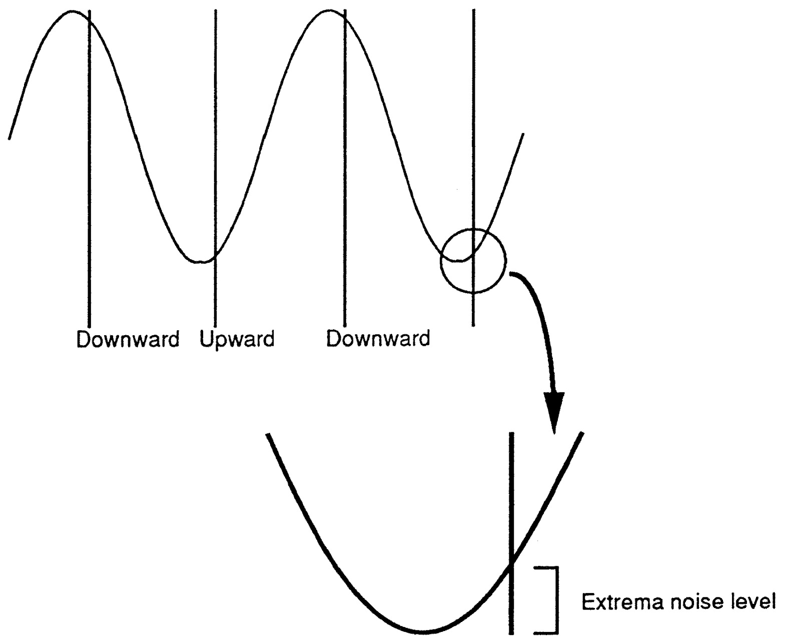


Figure 4.13: Wrist-tremolo detection.

Figure 4.13 depicts the algorithm used to detect tremolos and measure tremolo period by analyzing the wrist flexion data. The wrist-tremolo object is in either one of two states: upward or downward. If the object is in the upward state, then flexion data is rising and the object is waiting until the flexion data begins to fall. Likewise, a downward state means that the data has been falling and the algorithm is waiting until the flexion begins to rise. When the reversal of direction has been detected, the object informs its user of the time period since the last change and changes state.

False detections due to small reversals in the data are eliminated by specifying a tolerance called *extrema noise* which must be exceeded before a reversal is counted. The algorithm also accepts a time-out parameter which is used to inform the user that no reversal has been detected during the time-out period.

Bow Style

The bow-style object is a library object mix-in which itself uses the attack-mode and wrist-tremolo mix-ins, simply by specifying the two objects in its inheritance list. Bow-style uses and correlates data from flexion, bow-position and DSP energy analysis to determine the basic style of bowing. The following list describes each style and the algorithm that bow-style uses to detect it.

- *Tremolo*: Tremolo is detected by using the wrist-tremolo object described above. When four consecutive alternations are seen without any intervening time outs, bow-style interprets a tremolo.
- *Spiccato*: The spiccato or "bounce-bow" technique involves a rapid succession of notes caused by bouncing the bow across a string. This is detected with the help of the attack-mode object described earlier. When a rapid succession of three attacks are seen, and when the wrist has remained relatively still, the spiccato style is detected.
- *Pluck*: A pluck is distinguished from the bow position data. A cellist plucks a string with the right hand, causing the bow to fall out of range of the position sensor. Thus, an attack that occurs with the bow out of range is taken to be a pluck.
- *Normal*: Normal playing is defined as any notes that are played when the above conditions do not apply.

Bow Range

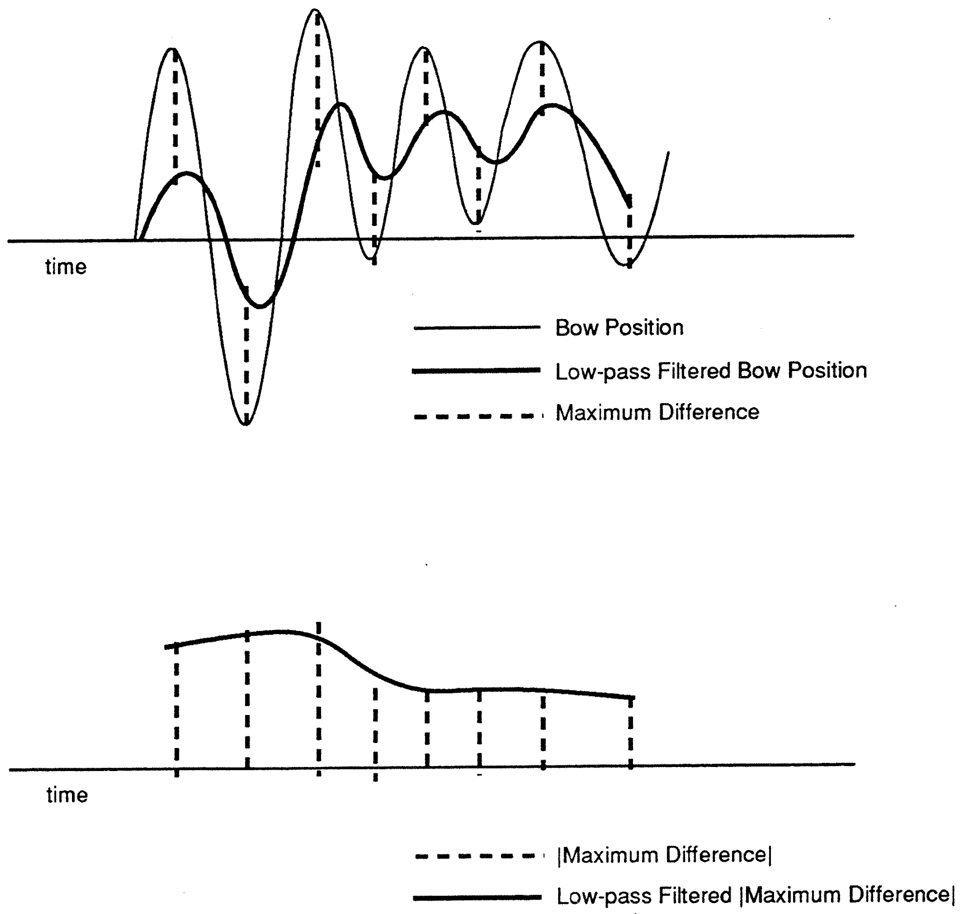

time
Bow Position
Low-pass Filtered Bow Position
Maximum Difference
time
|Maximum Difference|
Low-pass Filtered |Maximum Difference|

Figure 4.14: Bow-range object.

The bow-range object determines how much bow is being when the cellist is playing. This measurement must attempt to distinguish between genuinely using a large amount of the bow, and the necessary deviations which occur as the cellist plays different rhythms and selects different positions on the bow to play on.

The algorithm takes advantage of computationally inexpensive single-pole infinite impulse response (IIR) low-pass filters of the form:

$$y[\mathrm{n}] = s\, x[n] + s/(1+s)\, y[n-1]$$

Where $x[n]$ n^{th} input sample, $y[n\text{-}1]$ is the previous output sample of the filter, and s is the smoothing factor. Typical values for s range between 2 to 5 for slight smoothing to 40 to 50 for heavy smoothing.

Two low-pass filters are used to compute the bow range. The first filter is a medium filter which smooths the raw bow-position data (left-right data) producing a general contour of the position but with much smaller extremes. The difference between the filtered and unfiltered position data is calculated, and the maximum of the absolute value of this difference is computed periodically. These maximum values are themselves smoothed in the second low-pass filter providing a stable yet accurate measure of the range of bow usage.

4.6 The Hypercello and *Begin Again Again...*: Musical Mapping of Performance Gesture

This year we experimented with many new hyperinstrument functions, several in an attempt to improve on previous models, several to give the solo hypercello sufficient control over multiple complex parameters, several to establish a new equilibrium between continuous and discrete parametric control, and several to try totally new mappings between performance/musical information and computationally expanded result. In many ways, the hyperinstrument functionalities developed this year were quite different than those developed since 1988 [Machover 1988; Machover 1989; Machover 1990]. This is mostly due to the fact that the type of reliable control from the various cello controllers was in a somewhat different domain than that of previous controllers: with commercial MIDI controllers, pitch and velocity measurement were assured (although too quantized to be sufficiently expressive), whereas continuous control information was very limited; with the hypercello, pitch tracking was good enough to provide triggers and "focus pitches" but not good enough for note-by-note following, whereas continuous control (especially from the bow) was sophisticated, varied, and reliable. Therefore, we concentrated on real-time controls that would emphasize the strengths of the hypercello and would minimize the importance of tracking problems for which we do not yet have satisfactory technical or theoretical solutions.

Most of this year's new hyperinstrument functions were designed for Tod Machover's composition *Begin Again Again...*, many being implemented for the final performance version of the piece (Summer 1991, with subsequent improvements for the Montréal and Tokyo versions in Fall 1991) and several being evaluated and eventually rejected as hyperinstrument implementations (described below). As with Machover's hyperinstrument composition *Towards the Center* [Machover 1990], *Begin Again Again...* is a completely "live" composition; although there are sequences which are triggered during the composition, no musical material is pre-recorded, all being controlled during the performance. This piece is the most advanced hyperinstrument composition so far, due to its length, complexity and quantity of real-time analysis, transformation and synthesis. The live format of the piece also means that every level of music—from each independent detail to the overall shape of phrase and even structure—can be accessed and influenced by the hyperinstrument performance controls. The following are the most effective new hyperinstrument functionalities developed for *Begin Again Again...*, ordered from those most closely tied to traditional cello-playing technique, to those with the most novel mapping of cello control to sonic result.

Articulation Mapping

The most direct mapping of cello playing uses the technique called *articulation mapping*. In this case, it was decided to develop a series of distinctive cello performance techniques which could be used as discrete cues to throw the hyperinstrument system into a change-of-state. The techniques which are searched for by the system are: tremolo, bounce bow (in a single direction), pizzicato, left hand pizzicato, legato, and staccato. The system was fine-tuned to detect each of these states as quickly as possible, while not being so "jumpy" that no individual state could be sustained. This technique is used in the *Playful* section of *Begin Again Again...*. The music is written in such a way that, while the score is meant to be played very rhythmically and precisely, each short phrase is to be played with a different technique. The effect is to establish a sense of continuity through rhythmic regularity, juxtaposed with a polyphonic discontinuity stemming from the hyperinstrument-enhanced articulatory irregularity. The performer must be careful to exaggerate the differences between each style of playing: *i.e.* tremolo is to be played with as loose a wrist as possible (since wrist movement and note-onset-energy are the salient trigger cues); bounce bow is to be played with as stiff a wrist as possible (so as to differentiate it from tremolo) and with as dry and non-resonant a tone as possible (to make each new note-on very distinct, differentiating it from fast-note staccato or legato playing). In the *Playful* section, there is no accompaniment material generated. Rather, the direct acoustic cello sound is sent directly to a series of digital signal processing modules (*i.e.* Yamaha DMP–7, Lexicon PCM–70, on-board DSP processing, etc.). A different processing technique is associated with each possible playing technique (*i.e.* chorusing/flanging with tremolo; echo with pizzicato; spatial movement and delays with bounce bow, etc.). In addition, several modes apply control of the transformation parameters by continuous performance data once the transformation mode is initiated (*i.e.* finger pressure on bow controls depth of flange/chorus for tremolo; energy of pizzicato controls length of echo decay, and speed of delays, etc.). Experiments were made using a score follower and event detector in conjunction with the articulation mapping mode. In this case, parametric data for the various instrumental transformations were updated (and in fact increased in intensity and perceptibility) at predetermined points in the score. However, it was decided that it was a more powerful model to have the transformations remain static with a larger degree of control over their parameters left to the performer. In this way, the performer is always aware of what playing modifications can initiate a particular transformation, and what physical actions can change those transformations. The hyperinstrument therefore becomes easier to adapt to a particular playing style, and the section of music becomes "interpretable"—both in the individual moment and as an overall dynamic form—by each performer and in each different performance.

Trigger and Control

A more sophisticated extension of the previous technique is called *trigger and control.* Here a score follower and event tracker are used. Each cello phrase is monitored, both for its pitch content and gesture control information. There is no timing information provided by the computer; all is controlled by the speed and phrasing of the performer. A combination of direct transformation for cello input, transformation combined with elements from a large prerecorded database of cello timbres, and computer generated accompaniment material is all made available. Each cello phrase has an entirely different accompanying texture. Most of the computer material is subsequently modified (in time, texture, degree of transformation, etc.) by gesture control of the cellist. This technique is most noticeable in the *Introduction* and *Delicate and Varied* sections of *Begin Again Again....* Since many of the accompaniment figures exceed the live cello phrases in duration, this technique allows the cellist to layer up many different textures while continuing to be able to influence and modify each layer. A very important parameter controlled by the performer is the sonic balance between the three types of musical material: direct transformation, prerecorded samples, and synthetic sound.

Texture Select and Emphasize

One of the most effective new hyperinstrument techniques is *texture select and emphasize.* This function, found in the *Energetic* section, makes use of the multiple control parameters available in cello performance, as well as the way that these parameters are coupled and interdependent. The section is to be played in a rapid and rhythmic fashion. A computer accompaniment follows the cello part, doubling much of its musical material. Using many prerecorded cello sounds (delicate, regular, and aggressive tremolos; pizzicato and snap pizzicato; bounce bow; *col legno battuto*, etc.), a multi-layered accompaniment was pre-sequenced and synchronized with the live cello line. Through performance articulation (concentrating on bowing technique), the cellist controls every aspect of this accompaniment. Each line of accompaniment material is associated with a different prerecorded timbre. The position of the cello bow is measured from the bow's frog to tip at each moment. Each part of the bow is associated with one of accompanying musical lines. The cellist can activate the desired timbre by playing in the correct part of the bow. The louder the playing, the more that particular timbre is emphasized. The more bow that is used (per unit time), the more the musical lines adjacent to the principal line are emphasized. The more bow finger pressure is measured, the more abrupt is the selection of the most prominent musical line. The combination of these parameters makes it possible for the cellist to use familiar playing techniques (accent, amount of bow, frog-to-tip bow position) to modify the accompaniment. In addition, the "feel" of the texture select and emphasize mode seems to be—after much experimentation—just about right: the cellist controls a continuum from delicate to aggressive, sparse to dense, and pointilistic to resonant by using a similar intuitive continuum in his playing. The only adjustment made to this implementation during research and rehearsals with Yo-Yo Ma was the physical distribution of musical lines and timbres along the bow (which seems to be player-dependent). The sensitivity and smoothness of computer reaction seemed to be fairly player-independent, and was tuned by Tod Machover and Joe Chung between sessions with Mr. Ma.

Bow Mixing

A very useful technique, although one quite hard to perfect (musically and conceptually, rather than technically) is *bow mixing,* for which there were many applications in the *Begin Again Again…* project. The principle of this piece (applied "again and again") is to start with a single cello line and to build up complexity both "horizontally" (by continuously varying the composition of the cello phrases) and "vertically" (by enriching textural superpositions with simultaneous commentaries on instantaneous cello material). The latter technique results in dense textures which must somehow be shaped and modified *by the cellist,* while the cellist is playing. Two problems are posed here: how does the cellist control such mixing operations—similar to conducting—while playing difficult and intricate music; and how does the cellist actually hear the resultant mix of all the simultaneous lines and textures *as the audience would hear it.* Both of these problems turn out to be non-trivial. For the first problem, we developed several models that used bowing parameters (as well as *score following*) to allow the cellist to shape accompaniment textures while playing. The most successful of these was for the *Fluttering* section. Here, a four-part texture is generated by the computer. The hypercello part is composed as a combination between a solo and an accompanying figure, *i.e.* in terms of pitch content (moving to and from "core pitches" of the computer part), articulation (tremolo or not), and rhythm (carefully notated changes of tempo), this cello line moves in and out of the foreground of musical texture. This fact means that an integral part of the performance of this musical material becomes the *relationship* of solo playing to hyper-control, which is determined by the player. Since the mediation of this continuum is non-discrete, the performer can always be modifying, even when this is not a primary concern. Controls were developed to allow different cello parameters to affect different layers of the accompaniment: bow position determines which accompaniment layer is most prominent; bow-finger pressure adds layers of extra timbre (more and more "metallic") to a specific accompanying line; amount of bow used and energy determine the amplitude of the selected accompanying line; tremolo speed controls the amount of transformation of the cello solo part, introducing echo and reverb effects to the acoustic signal as well as adding a synthetic "shadowing" part. The *Fluttering* mode was successful because the cello part itself was only moderately difficult (leaving mental room for extra controls), and because the controls were imagined as an integral part of the normal instrumental performance of the cello line. Other implementations were less successful (such as *Warm and Singing* and *Very Rapid*) because the cello part was already at the limit of performability (very difficult!) and because the accompanying textures were too complex to allow for meaningful controls to be defined so simply (at least at present). We hope to improve such a bow mixing model, and to apply it to complex textures, in future versions of the string hyperinstrument. As for the ability of the soloist to hear the total sound mix on stage as the audience would hear it, this also remains to be solved. To do so, we will need an appropriate model of artificial acoustic ambience to give the player a much more realistic idea of how different sound levels interact once injected into a particular acoustic space. This is a valuable research topic for the coming years.

Automated Desynchronization

Since the pitch tracker for the cello was somewhat less reliable than the MIDI data available in previous hyperinstrument systems (using electronic keyboard, electronic percussion, or electric guitar controllers), the hypercello could not be used as a complete MIDI note generator (*i.e.* turn off entirely any direct sound from the acoustic cello, and reproduce accurately every pitch played with a synthetic timbre). Therefore, some of the more elaborate rhythmic amplification and synchronization techniques [Machover 1989; Machover 1990] were not appropriate or applicable to the present project. However, certain techniques to increase rhythmic control were developed. The most effective of these is *automated desynchronization*, used in the *Jaggedly Aggressive* section. Here the general tempo (quarter note equals 160 b.p.m.) is extremely fast. The computer part is carefully coordinated to the cello playing, which injects accents into the accompaniment and generates a booming bass line to define harmonic movement. At certain points in the movement, we wanted the cello to generate dense clusters of notes around a certain pivot note, and to control the gradual desynchronization of these notes—by acceleration—from the steadily rhythmic accompaniment. It was thought that the live cello could either perform the necessary rhythmic complexifications or play the desired disjunct pitch permutations, but not both, and, in any case, never the clustered pitches required in the score. Therefore, the following hyperinstrument was implemented: the cello, at certain moments in the movement, plays very rapid repeated-note figures; instructions in the score indicate that these figures are to begin in the tempo of the accompanying computer part, but are to deviate "jaggedly" to a faster tempo and then return to the original during the duration of each individual pitch; each cello note played is recognized by the pitch tracker and a library of predetermined clusters centering around the performed pitch is selected; each time a new note is played, it is detected by the energy tracker and the next note in the *cluster library* is sounded, in exact synchrony with the cellist-performed event; this synthetic cluster is played with the loudness of the triggering cello note; the timbre of the triggered note is complexified according to finger pressure of the bow hand.

Timbre Instrument

We have worked on various versions of gesture-control systems for generating, defining and shaping continuous *spectral music* [Machover 1981; Machover 1982; Machover 1989; Machover 1990]. Such timbral music is one of the major frontiers of computer-controlled sound, since current techniques allow for a new definition of sonic objects that no longer remains invariant with the physical limitations of a particular performance medium. In fact, the transformability of sound is one of the truly innovative materials available to the contemporary composer. Neither appropriate control structures nor satisfying organization principles have yet been developed for such timbral transformations, making this area a very fruitful one for future research. It was thought that the cello would provide an ideal set of gestural and musical controls for such timbral transformations, and so various versions of a *timbre instrument* were developed for the *Begin Again Again...* project. Previous hyperinstrument controllers have either been too rigid, insensitive or discontinuous to be satisfactory for such an application (keyboards, percussion, guitar, etc.), or too imprecise and relative in measurement (DHM glove controller). The cello combines many correlated degrees of freedom in bowing technique: continuous parametric control in the instrument and in the bow; precision in bow movement and pressure as well as in left hand finger position; and great variety of timbral variation in the instrument's own sound production. In short, the cello seemed to provide the ideal combination of continuous control and discrete selection.

The initial general conceptual model for the timbre instrument was to consider a complex cello sound, or series of sounds, which could be modified, fused, defused, or otherwise altered, using a real cello as the control instrument. These virtual cello spectra would be totally synthetic (partly computer-synthesized and partly sampled) so that there would be no limits to the type and number of transformation operations which could be performed on them. As a first approximation of such a timbre instrument, we developed a system for expressive control of timbre in real-time using additive synthesis with ordinary MIDI devices. This prototype was based on Tod Machover's work in developing the first real-time digital music performance system, which used a 4–X based additive synthesis model and was implemented for Machover's composition *Fusione Fugace* [Machover 1982]. Additive synthesis corresponds to the Fourier decomposition of a tone. A timbre is a set of partials, each partial being a multiple (not necessarily by an integer) of a fundamental frequency at some amplitude. This required that we be able to play a set of pitches at arbitrary frequencies, not necessarily the same as the chromatic pitches of the MIDI specification. We implemented this by using pitch bend to achieve the required detuning. Since pitch bend affects all notes on a given channel, this required that we use a separate channel for each note.

Additive timbres typically require more than 16 partials (72 would be more common) to sound rich and natural. This is because each partial is usually a pure sine tone. We were able to achieve rich timbres with fewer partials by using richer voices for the partials. We still needed to use more than 16 channels, and found the Mark of the Unicorn Midi TimePiece to be a useful device for accessing more than 16 MIDI channels at a time. Our system included a graphical interface for specifying timbre. A timbre is graphed as a set of lines, where the X axis shows the ratio and the height the amplitude. The composer can edit the timbre with the mouse, moving the lines to change the partials. The composer may also type in values with the keyboard if more precision is required.

To allow for variation in the timbres, we defined a set of *timbre mutators*. Each mutator is a procedure which can alter an additive timbre in some way. One example is to detune some of the partials. Another example is a mutator which increases the relative volume of an arbitrary partial while lowering the volume of all others, thus emphasizing part of the spectrum. At any time there may be many mutators running, each changing some aspect of the timbre. Some mutators run for a fixed length of time, and others run until explicitly stopped. One specially noteworthy mutator was the *timbre interpolator*. This mutator was able to gradually transform one timbre until it became identical to another. To support the mutators, we implemented a Lisp tool kit for specifying piecewise linear functions. An example might be a function which specifies the amount of detuning over time. We also implemented a graphical editor for the piecewise linear graphs.

The timbre instrument was designed to be controlled from the cello. The interface to the cello assigned a different function to each string. The C string selected a fundamental frequency for a note to be played. A downbow started a note, and an upbow stopped it. While bowing down, the loudness of the note was set from the amplitude of the bowing. The note continued until the next upbow. The G string selected a timbre from a preset library. One timbre was assigned to each chromatic pitch. If a note was being played when a new timbre was selected than a timbre interpolating mutator was started. The D string selected a mutator, in the same way that the G string selected a timbre, and the A string was used to supply additional parameters to the mutator.

This system was intended to be used in the piece *Begin Again Again...*, but was not completed in time. Although a prototype was developed and successfully tested, a sufficient method was not found to make the model computationally efficient enough to support the complexity and density of operations needed to render it viable in a musical context. The model which we finally did use for the *Timbral Sea* section of *Begin Again Again...* is not the most technically complex of the systems experimented with, but it proved to be the most robust, controllable, and computationally efficient. In addition, its musical results proved to be the most satisfying. For this timbre instrument, the direct, amplified sound of the live cello is minimized; in each performance, the cellist has the option of mixing his or her live sound as desired with the computer output (in the Tanglewood premiere, Yo–Yo Ma's cello sound was slightly less prominent than the computer part; in Tod Machover's Montréal performance, the live cello sound was virtually eliminated so that the hypercello could become a control instrument). A very complex set of timbral materials was created and stored on the computer. These were organized into four sections, with an ongoing musical continuity holding them together. Each of these sections is distinct in character (aggressive and descending; quasi-harmonic spectra with occasional bass pedal tones; overlapping timbres with pivot frequencies; spectral arpeggios), and lead to a final (fifth) section that uses the main variation theme of the work to regroup all of the timbral materials into a single fused timbral pedal point, which resolves into a simple D in the mid-range of the cello: the same note with which the work began. The performer is given instructions about how to control the shape and texture of each of these sections, hyperinstrument controls being provided through the cello as follows: speed of tremolo controls overall speed of unfolding timbres (each section can vary enormously in duration); cello energy determines amplitude of computer part; "brilliance" of cello sound (*i.e.* closeness of bow to cello bridge, or *sul ponticello*) shapes a spectral envelope, such that high frequencies are emphasized as the bow moves closer to the bridge; cello pitch provides an index into the frequency space of the resultant spectrum, allowing the instrumentalist to select which region of the spectrum will be influenced. While this model is actually rather simple conceptually, it proved to be a powerful tool for controlling and shaping very complex textures in real-time.

4.7 Musical Form and Hyperinstrument Functionality in *Begin Again Again...*

In many ways, *Begin Again Again...* is the most complex hyperinstrument work we have produced so far. In musical terms, this is apparent on various different levels. Machover's works from 1988 through 1990 have all been based on distinct-movement musical forms, with a clear sense of developmental direction from beginning to end of the work. Machover's music before 1987 was characterized by less clear movement distinctions, and a greater concentration on continuous musical development, resulting often in several layers of musical processes to be developed simultaneously. To encourage familiarization with each element of the musical discourse, and to allow very long musical forms to be built of recognizable parts, Machover experimented with the accumulation of many short movements, between and through which development takes place, beginning with *VALIS* in 1987.

Musical and Technical Variation

Begin Again Again... represents a new level of sophistication in Machover's formal structures. The entire piece is based on a set of variations. The theme of these variations is actually a series of fragments heard in the *Introduction* to the piece, consisting of the opening melodic phrase, and the ending phrase, which sweeps from the low register of the cello up the highest, and resolves on the initial melodic material. Thus it is a sort of double variations, one based on melodic material, and the other on timbral material, yet both are intertwined as becomes more evident as the piece develops. The melodic material contains the fundamental musical and expressive character on which the piece is based: centering around the cello's open D string (the instrument's most resonant note), the melody begins in upward movement, falling back to where it began. This process is repeated over and over, and both of its aspects are constantly exaggerated. The ascendant intervals become both more consonant (minor 2nd to major 2nd to minor 3rd, etc.), while the pull downwards is also more pronounced. Each time the phrase begins to ascend without constraint it is brought back to the lower register of the cello with even greater gravity. From the very outset of the work, *Begin Again Again...* establishes itself as being *about* this struggle to ascend, to turn minor into major, and to persist against the forces of gravity which constantly pull the soaring melody back to earth. As Yo–Yo Ma has observed, the musical dialectic established in *Begin Again Again...* could be viewed as an expansion and commentary on Bach's *Suite No.2* for Solo Cello in D minor, especially the *Sarabande* movement.

Besides the musical and psychological importance of this struggle against gravity, another equally important compositional feature is established at the very outset of *Begin Again Again....* The whole work is a set of variations. Formally speaking, there are ten variations, which are fairly well demarcated in terms of their tempi, textures, and atmosphere. In addition, the variations group together into larger units, and the entire piece can be seen as a through-composed structural transformation of complexification and reunification, a large two-movement form. At the same time, the true variation procedures are active on a much smaller scale. The real musical cell that is varied is the opening D–Eb–D. This phrase is expanded, is turned into a structural pivot which becomes embellished, and eventually becomes a kind of Schenkerian generative tree that spans whole variations at a time. In *Begin Again Again...*, Machover seeks a new clarity of form while taking care to establish the closest possible connection between each detail in the piece, the variation procedures of each movement, and the overall structure of the piece. For this reason, *Begin Again Again...* has a perceptual directness as well as a richness of discourse that surpasses previous hyperinstrument compositions.

Complex Sonic World

The sonic world of *Begin Again Again...* is also more complex than hyperinstrument pieces of the last several years. While purely synthetic sounds remain prominent, FM and Additive Synthesis instruments play mostly a background role, either clarifying rhythmic detail or filling out complex timbres. Much more prominent is a large database of stored sounds created from recordings of the cello, made by Yo–Yo Ma and Tod Machover. These sounds are designed to create a continuum from single cello notes, to superimposed cello "drone" sounds that can imitate pseudo-cello timbres (harmonic or inharmonic), to complexes of noise that are so dense that the individual cello elements are hardly perceptible. These sound complexes allow for a musical expression that can move from single line to saturated texture, all based on a solo instrument.

Changing Functionality

As in all Machover's hyperinstrument pieces, the functionalities of the hypercello change with each section of *Begin Again Again....* Although some of these changes were determined for practical reasons resulting from the necessary control of a particular synthesis parameter at a specific moment in the piece, choices were more often made to reflect the formal evolution of the music. In the beginning of the piece, the solo, independent quality of the cello is emphasized. The direct cello sound is most prominent. All timing information comes from the cello, not from the computer. Little by little, the cello releases subsidiary layers of material that grow directly out of his playing: bow pressure causes a note to be sustained, tremolo causes a timbre to be changed, an accent makes a special event be triggered. The overall impression is of the cello leaving sonic traces—remembrances. As the piece progresses, these traces become more and more prominent. At first, the traces simply linger longer before dying away. Then, the traces take on a life of their own, remaining as more than simple resonances, but as musical figurations that pulsate and dance before disappearing. By the *Energetic* section, these background traces have become quite active, even though the solo cello is still in control, shaping and molding the accompaniment with a flick of the wrist or squeeze of the finger. By the *Jaggedly Aggressive* section, there is clearly a sort of struggle going on. The solo cello is felt to be struggling, both to keep up with the incessantly rapid tempo of the computer part, and to make itself heard over the many layers that are beginning to accumulate. This process is exaggerated with the *Warm and Singing* section, where the cello almost ignores the incredibly dense texture that has developed out of its own sound. All the layers of accompaniment are being generated out of the live cello playing, but there are now so many of them that the cello can no longer hope to be in control. The hypercello controls allow the cello to fade in and out of this texture, alternatively playing its own expressive melody and shaping and massaging elements of the computer part. At the climactic *Very Rapid* section, the cello basically abandons any hope of control. The solo cello has finally broken out of the gravitational restraints of the piece's opening melodic fragment, using a chromatic two-note figure to ascend through the entire cello range. In doing so, the accompaniment has been simplified into a more rhythmically and harmonically homophonic texture. The cello takes advantage of this to ride lyrically on top of the enormous texture, singing its climactic melody while again generating layers of sonic residue. At the greatest moment of climax, this fragile moment of equilibrium explodes, as if too much energy has been unleashed to ever be contained. The cello then sets out on the reverse process of that with which the piece began; reorganizing and reunifying the many musical strands of the piece to reach again a possibility of simple discourse. The *Timbral Sea* is a section of reconciliation, of floating and discovering, and of choosing what is most relevant for the musical discourse which is to follow. In a very real sense, the image is of a solo acoustic cello being reconstructed after it has been shattered, with each partial, each note, each string, and each bit of wood being put back in place one by one. When this is done, a moment of perfect balance is achieved between the solo instrument and the computer accompaniment that has been unleashed. This equilibrium becomes the context for the final set of variations (a set of variations-within-variations) that begin with *Lyrical* and run until the *Coda*.

There is no longer a conflict between solo cello and accompaniment; the two are inextricably linked, with the tremblings of the one being reflected in the reactions of the other, and vice versa. The hypercello and its extended sonic world—through the struggle and reconciliation of the first part of the work—have finally become one, a new kind of instrument.

The Piece as Hyperinstrument Metaphor

This dramatic description of the development of hyperinstrument functionality in *Begin Again Again...* shows how the metaphor of Pandora's Box is always present in this composition. The cellist naïvely explores the extended implications of his new technologically enhanced instrument, but in doing so unleashes far more layers of musical material than he can control. He realizes this with a shock, and begins the process of simplifying and clarifying. There are, therefore, two large beginnings in this piece. Both are undertaken with enthusiasm and a certain optimism, but the second one also contains a new awareness of the importance of each choice made, and of the absolute necessity of choosing.

This brief explanation shows that *Begin Again Again...* is not only a piece that uses a carefully chosen series of modality transformations of the hypercello itself to underline the basic musical considerations of the composition. It is more than that: *Begin Again Again...* is a piece *about* building an instrument, and takes as its actual subject the delight in finding the many ways of extending a traditional instrument, the difficulty in containing this explosion of musical layers, the discipline and choice needed to refine this multitude of possibilities into a new and meaningful unity, and the beautiful and tenuous fragility, and expressive possibility, of the new instrument once it exists. The piece explores the implications of turning a monophonic instrument (the cello) into a polyphonic one (the orchestra and beyond). A single person can create a series of materials that he can no longer control in any meaningful way. This doesn't mean that one should retreat to the boundaries of our physical bodies or instruments. But it does mean that the exploration of intentionality—*why* one chooses to do this and not that—takes on an ever greater meaning as our technological prostheses become increasingly sophisticated.

Begin Again Again... is, unlike any project we have yet attempted, a musical work *about* the process and implications of designing hyperinstruments.

Bibliography

Abelson, Hal & Gerald Sussman. *Structure and Interpretation of Computer Programs*. Cambridge: MIT Press, 1985.

Atherton, David. MIDI Gesantkunstwerk. MIT M.S. thesis, 1989.

Chung, Joseph. Hyperlisp reference manual. Media Lab technical report, 1988.

Chung, Joseph. An agency for the perception of musical beats. MIT M.S. thesis, 1989.

Chung, Joseph. A development environment for string hyperinstruments, Proceedings of the International Computer Music Conference, Montréal, 1991.

Davis, Jim. Beat it. Media Lab technical report, 1988.

Ellis, Dan. Some software resources for digital audio in Unix. Media Lab technical report, 1991.

Gargarian, Greg. Logo music: a synthetic approach to musical thought. Media Lab technical report, 1989.

Gershenfeld, Neil. Sensors for real-time cello analysis and interpretation, Proceedings of the ICMC, Montréal, 1991.

Gialanze, M. The musicglove. Media Lab technical report, 1989.

Hong, Andrew. Mach 5: a development environment for timbral hyperinstrument controllers. MIT B.S. thesis, 1989.

Hong, Andrew. Non-linear Analysis of Cello Timbres. MIT M.S. thesis proposal, 1991.

Kopyc, D. Fantastic electronic sounds from runs, skips, and twirls. MIT B.S. thesis, 1989.

Levin, G. Frequency modulation waveform approximations: FM approximations of periodic waveforms from the Fourier spectra of arbitrary sampled systems. Westinghouse competition finalist, 1988.

Logue, Joan. "Tod Machover: Video Portrait," featuring *Fusione Fugace*. Paris: Centre Georges Pompidou, 1983.

Machover, Tod. *Fusione Fugace*, published score. Milan: Ricordi Editions, 1982.

Machover, Tod. Computer music with and without instruments, *Contemporary Music Review*, Vol 1, No 1. Paris: Christian Bourgois, 1984.

Machover, Tod. Thoughts on computer music, *Composers and the Computer*, Curtis Roads, ed. Los Altos: William Kauffman, 1985.

Machover, Tod. The extended orchestra, *The Orchestra: Origins and Transformations*, Joan Peyser, ed. New York: Charles Scribners, 1986.

Machover, Tod. *Spectres*, compact disc recording. New York: Bridge Records, 1986.

Machover, Tod. A stubborn search for artistic unity, *The Language of Electroacoustic Music*, Simon Emmerson, ed. London: Macmillan, 1986.

Machover, Tod. Etre compositeur aujourd'hui, *Inharmoniques*, Vol 1, No 1. Paris: Christian Bourgois, 1987.

Machover, Tod. *VALIS*, published score. Milan: Ricordi Editions, 1987.

Machover, Tod. *VALIS*, compact disc recording. New York: Bridge Records, 1987.

Machover, Tod. Hyperinstruments: musically intelligent/interactive performance and creativity systems, proposal to the Yamaha Corporation, 1988.

Machover, Tod & Joe Chung. Hyperinstruments: musically intelligent/interactive performance and creativity systems, Proceedings of the ICMC, 1989.

Machover, Tod. *Towards the Center*, published score. Milan: Ricordi Editions, 1989.

Machover, Tod. *Flora*, compact disc recording. New York: Bridge Records, 1990.

Machover, Tod. *Begin Again Again...* Musical Score. Milan: Ricordi Editions, 1991.

Machover, Tod. Interview with Jas Morgan, to appear in *Mondo 2000*.

Minsky, Marvin. *The Society of Mind*. New York: Simon and Schuster, 1986.

Norris, M. A. & David Rosenthal. Effect of varying loudness and duration on rhythm perception. MIT Media Laboratory Internal Memo, 1991.

Norris, M. A. The cognitive mapping of musical intention to performance. M.S. thesis, MIT Media Laboratory, 1991.

Romero, M. Objects to interpret real-time performances in a hyperinstrument context. MIT B.S. thesis, 1989.

Rosenthal, David: A model of the process of listening to simple rhythms, *Music Perception*, 6(3), 315-328, 1989.

Rosenthal, David. Emulation of human rhythm perception, to appear in *Computer Music Journal*, 1992.

Rowe, Robert. *Floodgate*, composer's manuscript, 1989.

Rowe, Robert. Machine listening and composing: making sense of music with cooperating real-time agents. Ph.D. thesis, MIT Media Laboratory, 1991.

Rowe, Robert. *Interactive Music Systems*. Cambridge: MIT Press, 1992 (forthcoming).

Sturman, David, David Zeltzer & S. Pieper. Hands on interaction with virtual environments, UIST 1989: ACM SIGGRAPH/SIGCHI Symposium on User Interface Software and Technology, 1989.

Yadegari, S. D. Using self-similarity for sound/music synthesis. M.S. thesis proposal, MIT Media Laboratory, 1991.

Yadegari, S. D. Using self-similarity for sound/music synthesis, Proceedings of ICMC, Montréal, 1991.