Growing Music from Seeds: Parametric Generation and Control of Seed-Based Music for Interactive Composition and Performance

by

Alexander Rigopulos

B.S. Music
Massachusetts Institute of Technology
1992

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the Massachusetts Institute of Technology
September 1994

© Massachusetts Institute of Technology, 1994
All Rights Reserved

Signature of Author [illegible]
Program in Media Arts and Sciences
August 5, 1994

Certified by [illegible]
Tod Machover, M.M.
Associate Professor of Music and Media
Information and Entertainment Section
MIT Program in Media Arts and Sciences
Thesis Supervisor

Accepted by [illegible]
Stephen A. Benton, Ph.D.
Chairperson
Departmental Committee on Graduate Students
MIT Program in Media Arts and Sciences

Rotch
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

OCT 12 1994

LIBRARIES

Growing Music from Seeds: Parametric Generation and Control of Seed-Based Music for Interactive Composition and Performance

by

Alexander Rigopulos

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on August 5, 1994
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences
at the Massachusetts Institute of Technology

Abstract

Conventional musical instruments are low-level interfaces for music control: one action or combination of actions on the instrument is required to produce a single musical event (a note). Facile manipulation of such an interface requires considerable expertise. Higher-level interfaces, in which each controlling gesture elicits a complex series of musical events, could grant non-expert musicians the opportunity to experience improvisatory music composition and performance. The primary obstacle in developing such interfaces is a the lack of a versatile high-level computer representation for music. The current ubiquitous representation, the MIDI sequence, is a low-level representation which is poorly suited for feature-based manipulation. This document describes a new, higher-level musical representation scheme, the parametric "seed", which is a versatile medium for real-time, feature-based transformation of music. Discussion includes the conversion processes to and from this seed format, details of existing implementations, related musical interface design issues, and the many implications of this technology.

Thesis Supervisor: Tod Machover, M.M.
Associate Professor of Music and Media

This work was supported in part by the Yamaha Corporation, Sega of America, and Interval Research Corporation.

Growing Music from Seeds: Parametric Generation and Control of Seed-Based Music for Interactive Composition and Performance

by

Alexander Rigopulos

Thesis Readers:

Certified by [illegible]
Mitchel Resnick, Ph.D.
Assistant Professor of Media Arts and Sciences
Learning and Common Sense Section
MIT Program in Media Arts and Sciences

Certified by [illegible]
Evan Ziporyn, Ph.D.
Associate Professor of Music
Music and Theater Arts Section
MIT Department of Humanities

Acknowledgments

I am indebted to my advisor, Tod Machover, for his extraordinary support and guidance throughout the past two years.

Secondly, I must thank Eran Egozy and Damon Horowitz, who did all of the programming for this project, as well as much of the conceptual design. Without their help, this project quite literally would not have been possible.

Additional thanks go to...

My readers, Mitch Resnick and Evan Ziporyn, for offering their valuable perspectives.

My "wife", Jane, and my son, Adrian, for patiently tolerating two years of my excessive absence from the homestead.

My parents, for all of the moral and culinary support.

All the boys of the Hyperinstruments Group, for reinforcing my suspicion that, in fact, there is no line beyond which puerile humor goes too far.

Fumi Matsumoto, for all of those productive "business lunches".

Suzanne McDermott, for taking care of business. A lot of it.

Au Bon Pain, for an inexhaustible supply of caffeine, refined sugar, and saturated fat.

General Chuck Yeager, for his high-altitude companionship throughout the past year, and for his novel ideas about joystick interfaces. In his sagely words: "It's the man, not the machine."

Contents

1 Introduction

The MIDI sequence is the ubiquitous modern representation for computer music scores. The development of MIDI was an important step towards abstracting the computer music score, because it eliminated timbral specification. The MIDI sequence instead preserves only a list of performance events, such as the timing and velocity of key presses and releases. The sequence can thus be reconstituted into sound through any MIDI tone generator or sampler using any desired timbre.

This abstraction was quite powerful in context; no universal computer score representation had preceded it. But although the MIDI protocol was a leap forward, it is still a comparatively low-level representation. Each datum in the note sequence corresponds to a single, specific event in "note-space". Thus, the MIDI sequence is an even more low-level representation than a conventional written score, in which each symbol represents a whole body of implications governed by the relevant performance practice.

A low-level representation of this kind is useful when absolute specificity is desired by the individual who is manipulating the data within the representation. However, absolute specificity can also be encumbering. Often, manipulation of a complex system of low-level variables can be quite vexing, as changing one of the variables does not affect any salient features of the system's behavior. Alternatively, a far more intuitive control paradigm under many circumstances is "feature-based" control, in which the accessible system variables do control the salient features of a system's behavior. These high-level variables are normally composed of webs of the system's hidden low-level variables.

The resulting interface requires only that a user is capable of expressing a desired qualitative change in the state of the system. The user need not be capable of expressing each of the incremental low-level modifications necessary to achieve that change. This type of feature-based interface has vast unrealized potential in music.

There is an enormous population of people who are not skilled musicians, but for whom music is deeply important. For these people, the experience of improvisatory music composition and performance has thus far been utterly inaccessible. Adept performance and composition requires literally years of training. But with the advent of high-level musical instrument interfaces, this barrier will slowly deteriorate.

To facilitate a feature-based interface, a more abstracted music representation system is needed. The purpose of the research described in this document was to build just such a representation system. This system describes music sequences as a fluctuating set of parameters, each of which corresponds to some salient musical property. This set of parameter data, the "seed", is the "parameter-space" equivalent of the input sequence; it will contain all of the information necessary to regenerate the original sequence. The benefit of this design is that any of these parameters can then be manipulated in real-time to dynamically shape features of the music being generated. The resulting interfaces to such a system could offer real-time feature-based control of music to provide non-expert musicians with a form of liberated musical performance and experimentation which they could never experience otherwise.

1.1 Hyperinstruments: Enhanced Expressivity for Experts

Since 1987, the goal of the Media Lab's Hyperinstruments Group under the direction of Tod Machover has been to enhance human expressivity in music performance through the use of technology. This research effort has been conducted through a diverse series of projects collectively known as "hyperinstruments" [Machover92].

During the first several years of development, hyperinstruments were designed exclusively for virtuoso musicians. The hyperinstrument model consists of three basic parts:

- A physical instrument, played by a performer. Sensors on the instrument/performer gather as much data as possible about what the performer is doing physically. Common data types are: (a) a MIDI stream, (b) an audio signal produced by the instrument, and (c) output from sensors designed to measure some physical aspect of the instrumental technique (e.g., sensors on a cellist's bow for measuring bow position with respect to the bridge).

- A computer which analyzes this data to extract higher-level musical information about the performance (e.g. bowing technique from bow position data). The computer then uses this information to control some aspect of either the sound being produced by the computer or the sound being produced by the instrument itself. The manner in which the performance gestures control the computer output is determined by the composer.

- The actual sound-generating or sound-altering devices: synthesizers, computers for sample playback or audio processing, external effects processors, etc. The output from these devices is dynamically shaped by the computer's interpretation of the performance.

One typical result is a concerto-type situation is which a solo performer on a hyperinstrument is accompanied by a computer "orchestra". Consider, as a simple example, a passage in which a hypercellist's bowing technique controls the articulation of accompanying material in the computer score: if the cellist's technique is particularly marcato, the computer may exaggerate this gesture by doubling with percussive attacks; or, if the bowing technique is more legato, the computer may complement this gesture by doubling with softer, sustained tones.

The hyperinstrument, then, effectively has two general stages: an analyzer stage for determining the performer's musical gestures, and a generator stage for mapping these gestures onto the output from the computer. A parametric music generator could play an important role in the function of the latter stage: given a front end which could effectively extract important features of a player's performance, the parametric generator could then amplify the performer's gestures in the accompanying music by using these features to shape parameter envelopes over time. The challenge is to define a parameter hierarchy such that the transformations in the generated music exhibit qualities which bear a transparent relationship to musical ideas being implied by the performer.

1.2 Enhanced Expressivity for Amateurs

Until recently, hyperinstruments have been designed only for expert instrumentalists. The premise was that a virtuosic performer who is in complete control of his/her instrument can have expanded expressivity when given the power to effectively control a greater spectrum of musical sound and activity than is accessible through a conventional instrument.

During the last two years, however, the Hyperinstruments Group has investigated the possible application of hyperinstrument technology to non-expert musicians. The first such project was an interactive percussion system called DrumBoy [Matsumoto93]. Our goal with DrumBoy was to give non-expert percussionists tools for creating and manipulating complex drum patterns. In previous hyperinstruments, high-level musical gestures were analytically extracted from an instrumental performance. In DrumBoy, however, the high-level musical intentions could be directly expressed by the user. Several keys on a keyboard interface were labeled with adjectives describing salient features of drum patterns (complex vs. simple, energetic vs. calm, mechanical vs. graceful, hard vs. soft, etc.).

Fumi Matsumoto and I then developed transformation functions for all of these adjectives. For each adjective-based transformer, a generalized algorithm was needed which could modify any drum pattern in a manner consistent with the implications of that adjective. To accomplished this, we used an agent-based architecture. Each agent performed a very simplistic operation, such as displacing a note in time or determining a note's position with respect to the beat. Multiple agents were then linked together into hierarchical networks. Transformer networks were used to effect complex pattern alterations. Analyzer networks assisted the system in determining which transformer networks would be most effective under a given set of conditions. The challenge was to design these networks such that the resulting transformations would evoke the musical properties implied by a given adjective.

When completed, these simple controls permitted the user to qualitatively express a desired effect without needing to specify the countless discrete edits that might be necessary to achieve that effect. The seed music research is a direct outgrowth of this high-level control principle which was initiated in DrumBoy. The logical extension was to add real-time, feature-based control of pitched instruments as well as (non-pitched) percussion.

In order to do this, however, it was necessary to overcome the most fundamental obstacle in the development of DrumBoy: the unsuitability of the MIDI score representation for feature-based manipulation. This inadequacy results from the fact that while two drum patterns in different musical contexts can sound qualitatively quite similar, their underlying low-level activities might be quite different. While designing the adjective-based transformers, it became obvious that creating *broadly applicable* music transformation algorithms within a low-level, note-space representation would be extremely elusive. Thus, for the considerably more challenging task of real-time control of pitched instruments, a higher-level representation was clearly needed.

2 Related Work

In addition to the hyperinstrument projects, a substantial body of pertinent work in related fields has already been done as well. Some of the most relevant contributors include Clarence Barlow, who has set the most significant precedents in the parametrization of music; Robert Rowe and George Lewis, who have advanced the general state of feature-driven interactive music systems; and David Cope, who has had much success at the task of extracting stylistic traits from a sample of music in order to generate more stylistically similar music.

2.1 Clarence Barlow

Barlow's "Bus Journey to Parametron (all about Çoǧluotobüsişletmesi)" [Barlow81] is a magnificently creative document in which he meticulously details an elaborate system he crafted for parametric composition of music. This system was initially utilized in his seminal work, *Çoǧluotobüsişletmesi,* a piece for piano which was produced entirely through parametric textural shaping of a pre-defined harmonic framework.

His objective when commencing the project was to investigate the implications of composing a piece based upon the dimensions of musical texture. He expands upon the "weaving" implications of the word texture (from Latin, *textura*) by describing a musical texture as containing a number of "strands" running parallel in time. Each of these strands consists of a series of "monochronous events", i.e., events succeeding each other without overlapping . (The strand concept should not be confused with the concept of a "voice" in classical counterpoint; a chord is one monochronous event.)

He defines two temporal realms of musical behavior: the microscopic, in which repeating periods are perceived in the pitch domain, and the macroscopic, in which repeating periods are perceived in the rhythm domain.

The consistency of each strand is governed by eight parameters. The first six are three pairs of two (microscopic and macroscopic equivalents):

1 & 2) harmonic & metric *cohesion,* "dealing with the extent of harmonic or metric agreement between the individual pitches or spans".

3 & 4) melodic & rhythmic *smoothness,* "dealing with the non-angularity of the melodies (increasing angularity leads to pointillism) or non-syncopatedness of the rhythms involved".

5 & 6) chordal & tactile *density,* "dealing with the number of notes forming one event, or the average number of events in a unit time".

The last two are auxiliary parameters:

7) *articulation,* "dealing with the duration of events independent of timing".

8) *dynamics*, "dealing with their loudness".

Finally, an additional parameter, *conformity*, "deals with sameness or near-sameness of the micro- or macro-temporal material in the streams concerned".

The theoretical purity and simplicity of these control axes is attractive, and for Barlow this system was compositionally powerful. It allowed him to compose by independently governing the evolution of the surface features of a generated piece.

However, the parameter set he defined does *not* embody a complete representation for a broad class of musics, because there is no "transform" that exists between note-space and Barlow's parameter-space that can be reciprocated to regenerate the original object in note-space. So there is no mechanism by which any *existing* score can be degenerated and regenerated parametrically to allow feature-manipulation of that original score. Consequently, the applicability of Barlow's system was more or less restricted to his own compositional turf.

2.2 Robert Rowe

Robert Rowe classifies interactive music systems along three dimensions [Rowe93]:

1) The first dimension addresses the linearity of the system response.

Score-driven systems are linearly driven. The system anticipates the performance of a specific musical score as its input. System response is guided by the procession of this score.

Performance-driven systems are non-linearly driven. No fixed score is anticipated. System response is guided solely by perceptual features of the input.

2) The second dimension addresses the system's method of generating its output.

   Transformative systems employ transformation algorithms to produce variants of the input material.

   Generative systems use small amounts of pre-stored material and rules for algorithmically generating new material.

   Sequenced systems playback pre-stored sequences which are triggered by the input.

3) The third dimension addresses the system's role in the interactive performance.

   Systems using an *instrument paradigm* amplify musical gestures made by a soloist into the output material. The output can be thought of as an elaborated extension of the solo performance.

   Systems using a *player paradigm* are not subordinate to the soloist. The computer player has a musical voice of its own, which may choose either to reinforce or to disrupt the musical directions of the soloist.

As an example, consider Rowe's own interactive music system, Cypher. It consists of a listener stage and a generator stage. The listener is a real-time performance analyzer which gathers simple perceptual features of the input performance ("density, speed, loudness, register, duration, and harmony"). The listener does not compare the input to a stored score, thus Cypher is a performance-driven system. Each of these perceptual features is then mapped to an output function; when a particular feature of the input exceeds some threshold, the appropriate output function is called. These output functions either transform the input stream, generate material algorithmically, or playback sequences. Thus Cypher combines transformative, generative, and sequenced playback. Finally, Cypher uses the player paradigm, by which features of the system's responses can either combat or comply with the features of the input.

Cypher's usefulness is more or less restricted to the trained composer who is committed to developing functionally interesting "webs" of connections between the input features and output functions. A considerable amount of expertise is needed to use the system effectively, and even under ideal circumstances, the musical relationship between a performer's input and Cypher's output is often quite opaque. Additionally, any one web which relies on generative or sequenced playback is only likely to be useful within the context of a specific piece designed by the composer. I am more concerned with a system that is more generally applicable.

The seed music system, with an accompanying high-level interface of some kind, falls into the categories of a performance-driven system using an instrument paradigm. Its output methods, however, are a *hybrid* of transformative, generative, and sequenced types: the seed is pre-stored, like a sequence; it is converted into a MIDI stream generatively; and it is subject to transformation in the process. Here Rowe's classification scheme is inadequate to completely describe the seed music system. His scheme only addresses scenarios in which the input to the system is a stream of music event data (MIDI) from an instrumentalist. This classification is inadequate in that it neglects the possibility of the input consisting of an entirely different type of performance data. The interface to the seed music system allows a user to directly manipulate the transformative processes of the system, without the barrier of a conventional instrument in the chain.

2.3 George Lewis

George Lewis attacks the challenges of interactive music systems from the perspective of a jazz trombonist. Not surprisingly, he is primarily concerned with developing systems that can participate as equal contributors--not subordinates--in the context of group improvisation; he thus champions the player paradigm almost exclusively [Lewis85].

Similarly, his systems are performance-driven and entirely generative. There are no pre-stored sequences whatsoever. Instead, each system has a battery of (highly probabilistic) algorithmic composing routines which define that system's particular compositional style. The input streams from external performers are only used to help determine which of these distinctive compositional routines are chosen at a given time. Consequently, the system's output *is* responsive to the input, but the system will also defiantly retain its own distinctive characteristics at all times [Rowe93].

Lewis's systems are limited by the fact that his compositional algorithms are only useful in the musical context which he primarily cares about: freeform jazz. His systems also have limited versatility in that they do not make any accommodation whatsoever for the instrument paradigm; they are rather recalcitrant ensemble players.

Yet this same stubbornness might have interesting implications for a high-level control interface: instead of giving a user complete freedom to take the music in any direction, the user is only given the freedom to operate within the stylistic boundaries of a pre-defined musical personality. So for example, a user might have a high-level interface for controlling a Thelonious Monk-esque piano performance. That user could wrestle the performance in a variety of directions, but the output would always be distinctively Monk.

2.4 David Cope

David Cope's work is not concerned with interactive systems, but instead with the problem of representing musical style. Specifically, he has addressed the challenge of extracting a specific composer's distinctive features, or "signatures", through analysis of a large sampling of that composer's works [Cope91].

Cope's EMI (Experiments in Music Intelligence) systems are essentially huge pattern induction engines. The EMI system performs an exhaustive search of number sequences contained in the input sampling of scores, trying to find recurring patterns. A sequence that recurs (even in an approximate form) with sufficient frequency is then said to be a stylistic signature of the given composer. The success of such a system, of course, depends entirely upon how these "number sequences" are derived from the input scores. Some of the sequences are just linearly explicit in the score, e.g., a series of intervals in a melody. But these linearly explicit sequences are not useful for anything other than, say, gathering an inventory of melodic fragments or cadential patterns to which a composer is partial.

So additionally, the EMI system performs a temporally hierarchic score deconstruction (loosely rooted in Schenkerian analysis). Then, sequences on each temporal tier of that hierarchy can be analyzed. Such analysis is clearly necessary for identifying a composer's higher-level signatures, such as proclivities in the treatment of formal structural.

The quality of Cope's methods, of course, can be measured by the success with which his stylistic representation can be used to regenerate music bearing the features of a given composer. This is achieved by superimposing a composer's signature patterns upon a pre-existing structural framework, such as a generic classical sonata form. And in fact Cope's system does indeed perform this task with considerable success.

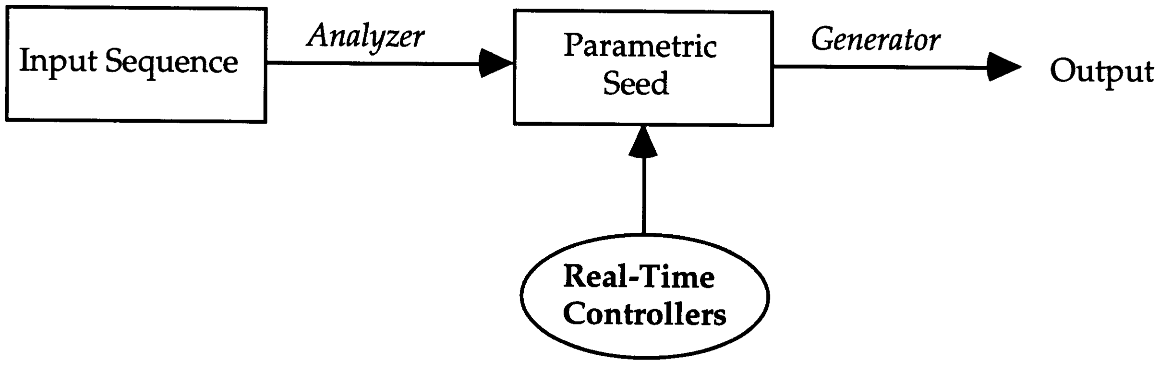
The analytic portion of Cope's methods are not directly relevant to the seed music system. His analysis relies upon exhaustive analysis of large volumes of music, and thus would be ineffective when used to analyze a single seed. The generative portion of his methods, however, is potentially quite applicable to the seed system. Recall the "Monk filter" described in section 2.3. Cope's signature representation might be abstractable into a parametric form, in which it could be used to guide and constrain the seed regeneration process. This would be an essential step towards realizing a system that could restrict its output behavior in such a way as to mimic the characteristic traits of a specific composer.

3 System Design

The primary design principle for the seed music system was that it should provide a platform to facilitate the rapid development of real-time, feature-based musical control interfaces. This required a model which would generate music in real-time as a function of a set of feature-based parameters. These parameters also had to be easily abstractable into higher-level parameters for the sake of interface versatility.

Because the applications of such a system are most relevant in a dynamic, improvisatory context, we initially concentrated upon musical genres in which repetition and improvisation play major roles, namely pop and jazz. Consequently, the parameters described below were chosen specifically to provide a flexible and robust representation for short repeating cycles (on the order of one or two measures) within these genres. (Issues regarding the parametrization of a more universal body of musics are discussed at length in chapter 4.)

Below is an overview of the system's basic architecture:

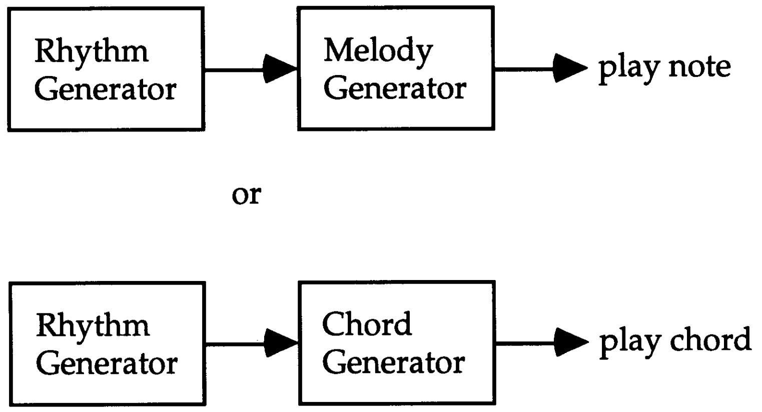


The input sequence is a MIDI file containing the music which the user wishes to manipulate. The analyzer then extracts several musical parameter values from this input as they change over the course of the sequence, producing a set of parameter envelopes, the seed. This seed is the parameter-space representation of the input sequence. The generator then converts this seed back into MIDI output. Real-time controllers are used to tweak parameters settings in order to dynamically change musical features of the generated output.

All of the software for this project was written by Eran Egozy and Damon Horowitz. The development environment was Hyperlisp, created by Joe Chung at the MIT Media Lab [Chung91]. Hyperlisp is an extension of Macintosh Common Lisp that includes a scheduler and a MIDI input/output handler.

3.1 The Generator

The generator is the heart of the system. It consists of three independent components: a rhythm generator, a chord generator, and a melody generator. Twenty-four times per beat, the rhythm generator decides whether or not a note/chord should be played at that instant. If the rhythm generator does decide that a note/chord should be played, it calls upon the melody or chord generator to produce output:



It would have been possible to generalize the melody and chord generators into a single generator. This might be powerful for emulating behavior such as a pianist's rapid shifts between melodic and chordal activity. However, for the sake of simplicity, it was desirable to keep the two generators separate during development .

Several generators can run in tandem, which allows many instrumental layers to be generated concurrently as an ensemble. For example, in a simple jazz context, a rhythm/melody generator can produce a bass line, while a rhythm/chord generator produces a pianist's left hand, while another rhythm/melody generator produces the pianist's right hand.

3.1.1 Rhythm Generator

The rhythm generator is responsible for deciding when a note/chord will be played, as well as the duration of that note/chord and how much dynamic accentuation should be applied to it. The rhythm generator makes these decisions as a function of the following parameters:

- activity
- syncopation
- syncopated-accent
- cross-rhythm
- cross-accent
- cross-meter

Each of these parameters is an integer between 0 and 127 to facilitate control via MIDI, which also uses 7-bit controller values. The rhythm generator also requires some supplementary variables:

- tempo
- simple-compound
- simple-cross-ratio
- compound-cross-ratio
- swing
- half-shuffle

3.1.1.1 Temporal Framework

The tempo variable specifies the number of beats per minute during playback. A beat (at any tempo) is represented as 24 temporal subdivisions, or "slots". The time period between slot increments, then, is 1/(tempo*24). At each slot, the rhythm generator decides, as a function of its parameters, whether or not to call upon the melody/chord generator to produce a new note/chord at that instant. (I will refer to slots as "full" or "empty" depending upon whether the rhythm generator has or has not decided to request a new note/chord in that slot.)

3.1.1.2 Activity

The activity parameter governs the average frequency with which slots are filled. The definition of the activity parameter depends upon the state of a flag variable, simple-compound, which determines whether the metric mode is simple (primary subdivision of beats into two parts) or compound (primary subdivision of beats into three parts). For each of these modes, five activity "tiers" are defined. An activity tier is an activity parameter value at which a single rhythmic pulse level is consistently generated:

SIMPLE MODE ACTIVITY TIERS

Activity Parameter	Pulse Level	Full Slots
0	𝄽 𝄇	none
32	♩ 𝄇	$0 + 24n$
64	♪ 𝄇	$0 + 12n$
96	𝅘𝅥𝅯 𝄇	$0 + 6n$
127	𝅘𝅥𝅰 𝄇	$0 + 3n$

COMPOUND MODE ACTIVITY TIERS

Activity Parameter	Pulse Level	Full Slots
0	𝄽 𝄇	none
32	♩ 𝄇	$0 + 24n$
64	$\overset{3}{\text{♪}}$ 𝄇	$0 + 8n$
96	$\overset{3}{\text{𝅘𝅥𝅯}}$ 𝄇	$0 + 4n$
127	$\overset{3}{\text{𝅘𝅥𝅰}}$ 𝄇	$0 + 2n$

In each table, the third column contains what I refer to as a "slot series", which takes the form $\Phi + Tn$. The n simply represents the series of integers from 0 to infinity. Thus, $\Phi + Tn$ represents a series of slots spaced at intervals with a period, T, and starting at a phase, Φ. Each activity tier has a slot series with a phase of 0 and a period equaling the number of slots between attacks for the rhythm value at that pulse level. (Note that the slot series are only displayed here in tabulated form for the sake of clarity. In the software, the series are derived from a formulaic generalization, as opposed to being retrieved from a table.)

When the activity parameter is at an intermediate value between two tiers, it must temporarily commit to one of the adjacent pulse levels before determining whether or not the current slot will be filled. This selection is made probabilistically as a linear function of the parameter's position between the two tiers. For example, at an activity value of 48 (halfway between 32 and 64), there is a 50% chance that the lower (quarter-note) pulse will be selected and a 50% chance that the upper (8th-note) pulse will be selected. Once a pulse level has been chosen, the slot is tested to see if it is a member of the slot series for that level. If so, the slot will be filled; otherwise, the slot will remain empty.

The result is that the onbeat slots of the upper pulse level will always be filled, but the offbeat slots of the upper pulse level will only be filled probabilistically as a function of the activity parameter's relative position between the upper and lower tiers. For example, given a sufficient window of time, an activity value of 48 (halfway between the quarter-note and 8th-note tiers) will produce and equal distribution of beats with a quarter-note rhythm (♩) and beats with an 8th-note rhythm (♫).

3.1.1.3 Syncopation

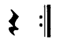
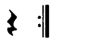
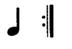
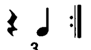
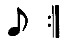
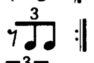
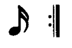
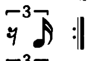
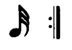
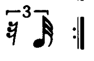
Parametric syncopation is achieved in two ways. First, the syncopation parameter probabilistically inhibits the occurrence of notes at onbeat slot positions and encourages the occurrence of notes in offbeat slot positions. Second, the syncopated-accent parameter softens notes at onbeat slot positions and accents notes at offbeat slot positions. (The syncopated-accent parameter can also be used to *subdue* syncopation by accenting onbeat slots and softening offbeat slots.)

Syncopation does not have identical meaning at all activity levels or meters, so a complimentary pair of syncopated slot series is needed for each pulse level and metric mode:

SIMPLE MODE SYNCOPATION SLOT SERIES'

Pulse Level	Syncopated Pulse	Onbeat Slot Series for Inhibition/Softening	Offbeat Slot Series for Encouragement/Accent
𝄽 𝄇	𝄽 𝄇	none	none
♩ 𝄇	𝄽 ♩ 𝄇	$0 + 48n$	$24 + 48n$
♪ 𝄇	𝄾 ♪ 𝄇	$0 + 24n$	$12 + 24n$
𝅘𝅥𝅯 𝄇	𝄿 𝅘𝅥𝅯 𝄇	$0 + 12n$	$6 + 12n$
𝅘𝅥𝅰 𝄇	𝅀 𝅘𝅥𝅰 𝄇	$0 + 6n$	$3 + 6n$

Compound Mode Syncopation Slot Series'

Pulse Level	Syncopated Pulse	Onbeat Slot Series for Inhibition/Softening	Offbeat Slot Series for Encouragement/Accent
		none	none
		$0 + 48n$	$24 + 48n$
		$0 + 24n$	$16 + 24n$
		$0 + 8n$	$4 + 8n$
		$0 + 4n$	$2 + 4n$

At each pulse level, an onbeat slot series exists that contains the slots which are eligible for inhibition. If the current slot is full, and it is a member of the onbeat slot series for the current pulse level, the syncopation parameter determines the probability of emptying the slot: P[empty slot] = syncopation/127.

At each pulse level, an offbeat slot series also exists which contains the slots which are eligible for encouragement. If the current slot is empty, and it is a member of this offbeat slot series for the current pulse level, the syncopation parameter determines the probability of filling the slot: P[fill slot] = syncopation/127.

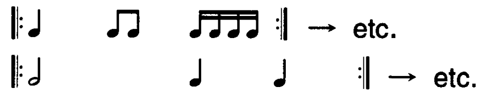
The syncopated-accent parameter is used to either induce or subdue syncopation through dynamic inflections at the appropriate slot positions. An accent equaling the parameter's distance above the center (syncopated-accent - 64) is *added* to all slots in the offbeat slot series and *subtracted* from all slots in the offbeat slot series at the current pulse level. Thus, no accents result when the parameter is centered at 64. When the parameter is increased, offbeat slots are accented and onbeat slots are softened, which induces syncopation. When the parameter is decreased, onbeat slots are accented and offbeat slots are softened, which subdues syncopation.

The accent/softening for each slot caused by syncopation is summed with the accent/softening for each slot caused by cross-rhythm (see section **3.1.1.4**). The result is passed as a key velocity offset to the melody and chord generators.

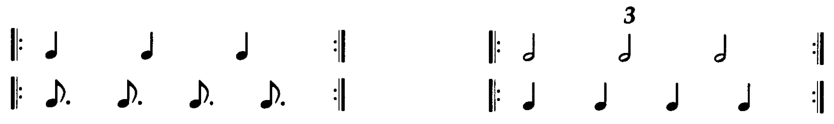
3.1.1.4 Cross-Rhythm

The term cross-rhythm refers to rhythmic oppositions achieved by superposition of repeating temporal periods of different lengths. In my opinion, these rhythmic oppositions fall into three broad classes:

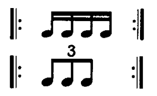
Type 1
Two phrases of different lengths repeat simultaneously:



Type 2
Two simultaneous beats of different lengths can be perceived. For example, the following two figures represent two different ways of hearing a single pattern:



Type 3
Simple and compound beat subdivisions are superimposed:



One could view all of these cases simply as manifestations of the same effect:

- In type 1, the repeating periods are 3 beats and 4 beats, producing a mutual period of 12 beats.

- In type 2, the repeating periods are 1 beat and 3/4 of a beat (or the reciprocal, if you wish), producing a mutual period of 3 beats (or, again, 4 beats if you wish).

- In type 3, the repeating periods are 1/4 of a beat and 1/3 of a beat, producing a mutual period of 1 beat.

Yet each of these cases also has a very different qualitative effect, and so I have chosen to represent them differently within this system:

- Type 1, in which rhythmic oppositions occur between phrases of several beats in length, has no parametric representation. It is accomplished instead by parallel generation, looping multiple seeds of different lengths simultaneously.

- Type 2, in which rhythmic oppositions occur between the beat and another competing period of comparable scale, will be henceforth referred to as cross-rhythms and will be addressed by the cross-rhythm parameter.

- Type 3, in which rhythmic oppositions occur between simple and compound subdivisions of a beat, will be addressed by the cross-meter parameter.

Cross-rhythms are produced in an identical manner as syncopation, both through slot inhibition/encouragement and accentuation. Again, cross-rhythm does not have identical meaning at all activity levels or meters, so a complimentary pair of phased slot series is needed for each pulse level and metric mode:

SIMPLE MODE CROSS-RHYTHM SLOT SERIES'

Pulse Level	Slot Series for Inhibition/Softening	Phase Difference for Encouragement/Accent
𝄽 𝄇	none	none
♩ 𝄇	none	none
♪ 𝄇	0 + 48(simple-cross-ratio)*n*	24
𝅘𝅥𝅯 𝄇	0 + 24(simple-cross-ratio)*n*	12
𝅘𝅥𝅰 𝄇	0 + 12(simple-cross-ratio)*n*	6

COMPOUND MODE CROSS-RHYTHM SLOT SERIES'

Pulse Level	Slot Series for Inhibition/Softening	Phase Difference for Encouragement/Accent
𝄽 𝄇	none	none
♩ 𝄇	none	none
♪ 𝄇	0 + 24(compound-cross-ratio)*n*	12
𝅘𝅥𝅯 𝄇	0 + 12(compound-cross-ratio)*n*	6
𝅘𝅥𝅰 𝄇	0 + 6(compound-cross-ratio)*n*	3

The variables simple-cross-ratio and compound-cross-ratio are ratios used to define the relationship of the desired cross-rhythm to the beat in each metric mode. So, for example, a 3:4 cross-rhythm in simple mode would have a simple-cross-ratio of 0.75. At the 16th-note pulse level, this would produce a dotted-8th feel. At the 8th-note pulse level, this would produce a dotted-quarter feel.

Note that no cross-rhythm slot series are defined for the quarter-note pulse level. This is because cross-rhythms at this pulse level depend upon whether the seed is in duple, triple, etc. meter, and there is no system representation for this aspect of meter. This form of rhythmic opposition is "type 1", as described above.

At each pulse level, one slot series exists which contains the slots which are eligible for cross-rhythmic inhibition. If the current slot is full, and it is a member of this slot series, the cross-rhythm parameter determines the probability of emptying the slot: P[empty slot] = cross-rhythm/127.

At each pulse level, a second slot series (out of phase with the first) also exists which contains the slots which are eligible for cross-rhythmic encouragement. If the current slot is empty, and it is a member of this slot series, the cross-rhythm parameter determines the probability of filling the slot: P[fill slot] = cross-rhythm/127.

The cross-accent parameter is used to induce cross-rhythm through dynamic inflection of the appropriate slot positions. An accent equaling the parameter value is *added* to all slots in the first slot series and *subtracted* from all slots in the second slot series at the current pulse level. Thus, no accents result when the parameter is centered at 64. When the parameter is increased, one cross-rhythmic slot series is accented and the other cross-rhythmic slot series (out of phase with the first) is softened, producing a cross-rhythm.

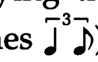
The accent/softening for each slot caused by syncopation (see section **3.1.1.3**) is summed with the accent/softening for each slot caused by cross-rhythm. The result is passed as a key velocity offset to the melody and chord generators.

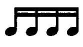
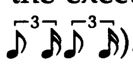
3.1.1.5 Cross-Metrics

Transition between simple and compound meters is accomplished with the cross-meter parameter. This parameter simply determines the probability that the rhythm generator will temporarily use the metric mode *not* indicated by the simple-compound flag: P[swap mode] = cross-meter. In other words, the cross-rhythm parameter controls the likelihood of inducing triplets when in a simple meter and duplets when in a compound meter.

There is one additional restriction on these metric transitions: they can only occur in slots which fall in the series for the pulse level *below* the current slot's pulse level. This is to prevent the possibility of awkwardly rapid transitions between metric modes.

3.1.1.6 Alternative Rhythmic Feels

The swing flag, when enabled, "swings" offbeat 8th-notes by delaying the execution of full slots in the series 12 + 24*n* by four slots (i.e., becomes ).

Similarly, the half-shuffle flag, when enabled, adds a half-shuffle feel by delaying the execution of full slots in the series 6 + 12*n* by two slots (i.e., 
becomes ).

3.1.1.7 Duration

Note duration is managed by the melody and chord generators. However, in order for the melody and chord generators to manage duration dynamically, they must constantly be aware of the current pulse level (see section **3.1.2.5**). To accommodate this need, the rhythm generator passes the activity value and slot number for every slot, full or empty, to the melody or chord generator.

3.1.1.8 Procedure Summary

The rhythm generation procedure can be summarized in the following pseudo-code:

```
choose mode as ƒ(simple-compound, cross-meter)
choose pulse level as ƒ(activity)

choose activity slot series as ƒ(pulse level, mode)
fill slot as ƒ(slot #, slot series)

choose syncopation slot series as ƒ(pulse level, mode)
fill or empty slot as ƒ(slot #, syncopation, slot series)
adjust accent as ƒ(slot #, syncopated-accent, slot series)

choose cross-rhythm slot series as ƒ(pulse level, mode, cross-ratio)
fill or empty slot as ƒ(slot #, cross-rhythm, slot series)
adjust accents as ƒ(slot #, cross-accent, slot series)

delay output as ƒ(swing, half-shuffle)
pass activity to melody/chord generator
if (slot is full)
        request new note/chord
        pass velocity offset to melody/chord generator

increment slot #
repeat loop
```

3.1.2 Melody Generator

When called by the rhythm generator, the melody generator chooses a pitch, a dynamic, and a duration using the following parameters:

- chord-tone-weight
- scale-tone-weight
- ripe-tone-weight
- direction
- leap-probability
- leap-size
- dynamics
- duration

It also requires the following supplementary variables:

- chord-root
- chord-type
- key-root
- key-type
- force-chord-tone
- force-root
- ceiling
- floor

3.1.2.1 Harmonic Framework

The melody generator begins by deciding which pitches are eligible to be the chosen as the next pitch. In order to do this, the melody generator must first check the current harmonic status of the system, which is represented as a chord-root and a chord-type. This status is never *determined* by the melody generator. Instead, it is either determined by the seed or specified in real-time by the user (see section **3.4**). In either case, the harmonic status is then accessible to the melody generator, which makes its choices as a function of that data.

A chord-root is simply the pitch class upon which the current harmony is rooted.

A chord-type is a list of lists, containing the following elements:

- A list of the *required* scale degrees which define a particular type of chord. These scale degrees are listed as semitone intervals with respect to the root note, 0. For example, in a dominant 7th chord type, the required scale degrees would be the major third (4) and the minor 7th (10).

- A list of *auxiliary* scale degrees which define this type of chord. These scale degrees are typically included as part of the specified chord type, but are not essential to categorizing a chord as being one of that type. For example, in a dominant 7th chord type, the auxiliary scale degree would be the perfect 5th (7).

- A list of scale degrees to define a *modal-non-chord* harmonic coloration for that chord type. For example, this category of coloration for a dominant 7th chord type might be the scale degrees which complete the mixolydian mode around that chord: the major 9th (2), the perfect 11th (5), and the major 13th (9).

- A list of scale degrees to define a *chromatic-non-chord* harmonic coloration for that chord type. For example, this category of coloration for a dominant 7th chord type might be the minor 9th (1), the augmented 9th (3), the augmented 11th (6), and the minor 13th (8).

- Lists for any additional groups of scale degrees which a designer believes to be functionally similar in the desired context.

As described here, the method of defining chord-type is inadequate for tonal music. For example, in the key of C major, a D minor chord should have a different set of modal scale degrees in its chord-type than an E minor chord. Otherwise, colorations which would be diatonic over D minor might be chromatic over E minor. Thus, a single minor chord-type does not suffice. Instead, several minor chord-type's must be defined, each of which has coloration lists reflecting the harmonic role of that particular minor chord within a tonal structure. For example, the following three minor chord-type's would be needed within a diatonic context: minor-dorian, minor-phrygian, and minor-aeolian. The *required* and *auxiliary* scale degrees for these chord-type's would be identical, but the *modal-non-chord* and *chromatic-non-chord* scale degrees would be different.

To relieve the user of having to select chord-type's manually, an additional structure, a key-type, is used. A key-type is a list of chord-type's corresponding to all scale degrees within a particular mode. The key-root specifies the actual pitch class upon which the key-type is based. Given the key-root, then, any chord-root will correspond to some position in the key-type list. That position contains the chord-type associated with the specified chord-root. To put it in simpler terms, if a D chord (chord-root) of unspecified type is requested the in key of C (key-root) major (key-type), the table would return a minor-dorian chord-type.

3.1.2.2 Pitch Classes

The melody generator interprets a chord-type list by dividing it up into three groups of melodic tones: chord tones, scale tones, and "ripe" (dissonant) tones.

- The chord tone group combines the *required* and *auxiliary* scale degrees plus the chord-root. It is useful for melodies which behave like arpeggios, reinforcing the harmony.

- The scale tone group uses the *modal-non-chord* scale degrees. When combined with the chord tone group, the scale tone group fills out the mode, enabling normal stepwise melodic motion. The scale tone group also introduces the possibility of mild dissonances caused by non-chord tones occurring on beats.

- The ripe tone class group used the *chromatic-non-chord* scale degrees. When combined in equal proportions with both of the other tone groups, it is useful for chromatic stepwise motion. In greater proportions it is useful for dissonant harmonic inflection of the melody.

The melody generator first designates one of these tone groups as the group of "eligible" pitches for the next note. It makes this decision probabilistically. The role of the chord-tone-weight, scale-tone-weight, and ripe-tone-weight parameters is simply to weight the respective likelihoods of each tone group being selected.

Normally, at least one of these three parameters should remain constant. If all three are changing, some unintuitive behavior may result. For example, if all three parameters are at 10, and then all three parameters are moved to 100, there is no change in the system's output. The *balance* between them must change to achieve an effect. One simple implementation is to fix the chord-tone-weight parameter at 64 and allow the scale-tone-weight and ripe-tone-weight to change freely.

Two additional variables affect the selection of the tone group: the force-chord-tone flag and the force-root flag. At slot 0, the force-chord-tone flag is checked. If it is set to 1, the chord tone group is forced for that slot, regardless of the probabilistic tone group selection. This is useful for making melodies "safe" from the dissonances induced by non-chord tones falling on beats. Similarly, if the force-root flag is set to 1 in *any* slot, the chord-root pitch is chosen as the only eligible pitch, and no tone group is used. This is useful when the melody generator is being used to produce bass lines, and the line is required to reinforce the harmony with roots on chord changes.

3.1.2.3 Melodic Motion

Once the group of eligible pitches has been determined, the melody generator must choose a specific note. First, the direction of motion is determined by the direction parameter: P[move up] = direction/127. Then, the leap-probability parameter determines whether the new note will move by step or by leap away from the previous note: P[leap] = leap-probability/127. Motion by step involves moving to the nearest eligible pitch in the chosen direction. Motion by leap involves leaping by some interval in the chosen direction and then selecting the nearest eligible pitch. The size of this interval, in semitones, is equal to the value of the leap-size parameter. A leap-size of 0 is used to produce repeated notes. (Note that if the eligible pitch group is chord tones, "stepwise" motion to the nearest eligible pitch might actually result in a small leap.)

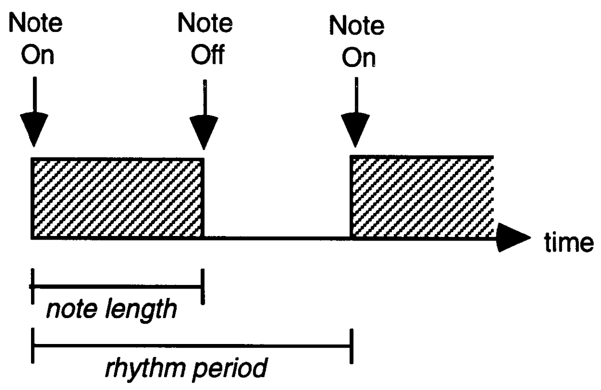
Additionally, it is usually desirable for safety's sake to establish some registral boundaries for the melody generator. Otherwise, a careless (or over-zealous!) user could easily lead the melodic line right out of bounds. To do this simply requires specifying a ceiling note number and a floor note number. If any steps or leaps attempt to cross either of these boundaries, they are "reflected" back into play by a temporary reversal of direction.

3.1.2.4 Dynamics

The rhythm generator passes a dynamic offset to the melody generator which is a function of the syncopated-accent and cross-accent parameters (see sections **3.1.1.3** and **3.1.1.4**). The melody generator simply sums this offset with the dynamics parameter to determine the key velocity of the new note.

3.1.2.5 Duration

Duration in the context of this system is defined not as an absolute property, but instead as a relative property, equivalent to the duty cycle of a square wave. Absolute duration will be referred to as note length, and the time between successive attacks will be referred to as the rhythm period:



The duration parameter is defined as 64(note length/rhythm period). Thus, a duration value of 64 corresponds to a *legato* succession of attacks, i.e., the ratio of note length to rhythm period is exactly 1. Lower duration values will produce detached phrases, whereas higher duration values will result in note overlap.

A system designed for responsive real-time manipulation of music inherits certain complexities when dealing with note length. Determining a note's length as a function of a duration parameter requires knowledge of the arrival time of the subsequent note's attack (the rhythm period). In a probabilistic, real-time system, it is not possible to know this information for certain, because it is subject to change at the whim of the user. Consequently, the system cannot make a final decision about the length of a note at the time that the note is turned on.

For example, consider an activity of 64, corresponding to an 8th-note pulse level which has a rhythm period of 12. At the start of the beat, slot 0 is filled. If the activity remains constant, the next attack should arrive at slot 12. Thus, a duration value of 32 (50%) would produce a note length of 6 slots. But what if the user suddenly changes the duration or activity values? The note length of 6 slots becomes obsolete.

To compensate, the melody generator must first of all remember all lingering notes. Secondly, the melody generator must monitor the changing duration and activity. The activity, which determines the rhythm period, is passed from the rhythm generator in every slot (see section **3.1.1.7**). Finally, the melody generator must terminate lingering notes when they expire according to the following relationship: note length = (rhythm period*duration)/100. A simple loop for managing note lengths is expressed in the following pseudo-code:

```
if (activity or duration has changed)
    compute new note lengths for each note in stack
for (each note in stack)
    if ((time since note-on) > note length)
        send (note-off)
        remove note from stack
```

Note that at a quarter-note pulse level with maximum duration, the greatest possible note length is a 30 slots (1.27*24), or a quarter-note tied to a sixteenth note. In order to generate larger note lengths, the lowest activity level is used (no pulse). Because the lowest activity level has an infinitely large rhythm period, indefinitely long sustained tones can also be generated.

3.1.2.6 Procedure Summary

The melody generation procedure can be summarized in the following pseudo-code:

```
choose tone group as ƒ(chord/scale/ripe-tone-weights)
if (slot = 0) and (force-chord-tone = 1)
    reset tone group to chord-tone
if (force-root = 1)
    reset tone group to chord-root

choose direction of motion as ƒ(direction)
choose motion by step or leap as ƒ(leap-prob)
if (stepwise)
    move in chosen direction to nearest pitch in tone group
    if (new note > ceiling) or (new note < floor)
        move in opposite direction to nearest pitch in tone group
```

[continued on next page]

```
if (leapwise)
        leap by (leap-size) in chosen direction
        if (leap destination > ceiling) or (leap destination < floor)
                leap in opposite direction
        move to nearest pitch in tone group in either direction

choose velocity as ƒ(dynamics, rhythm generator offset)
send note-on
send note-off's for expired notes as ƒ(duration, activity, slot #)
```

3.1.3 Chord Generator

When called by the rhythm generator, the chord generator constructs a chord and chooses a dynamic and duration for that chord using the following parameters:

- size
- register
- density
- density-scaling
- chord-tone-weight
- color-a-weight
- color-b-weight
- dynamics
- duration

It also requires the following supplementary variables:

- chord-root
- chord-type
- key-root
- key-type
- force-root
- force-required

3.1.3.1 Harmonic Framework

Before attempting to build a chord parametrically, the generator must first have information about the function of all pitch classes within the current harmony. This is determined by the chord-root, chord-type, key-root, and key-type variables.

(For a complete discussion of these variables, see section **3.1.2.1**.)

3.1.3.2 Chord Range

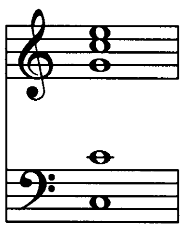
The size and register parameters determine the pitch range within which the chord will be generated. The size parameter value corresponds to the maximum distance in semitones between the lowest and highest notes of the chord. The register parameter value corresponds to the MIDI note number of the "center" pitch of the chord (the pitch that lies of the center of the chord's range). So, for example, a C augmented triad rooted upon middle C would have a size value of 8 (the distance in semitones between C and G#) and a register value of 64 (the MIDI note number of the E above middle C).

3.1.3.3 Chord Voicing

The density parameter determines how densely the chord will be voiced within this range. An "expected interval" value is derived from the density parameter by inverting it (subtracting from 127) and adding 1. The expected interval is the approximate distance, in semitones, between notes in the chord. So, for example, an augmented triad would have an expected interval value of 4 (a density value of 124). With an expected interval of 1 (a density of 127), then, every eligible note within the range would be included in the chord. An expected interval of 128 (a density of 0) would force a one-note "chord" by exceeding the maximum possible range achievable within the MIDI protocol. (Similarly, whenever the expected interval value is greater than the size parameter, a one-note chord will be generated.)

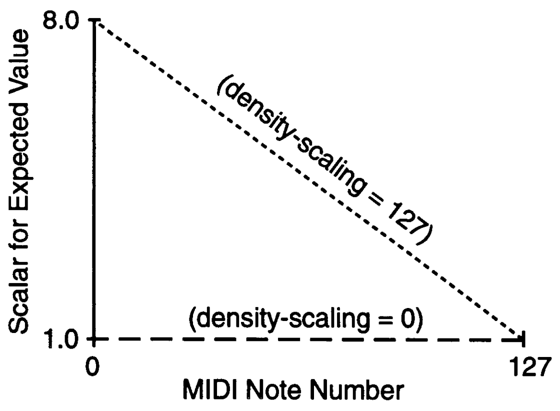
Chords are constructed from the bottom up. The chord generator starts at the bottom of the range, adds the nearest eligible pitch to the chord, leaps upward by the expected interval, adds the nearest eligible pitch, leaps upward, etc., until it reaches the upper limit of the chord range.

One important flaw in this algorithm as stated is that it assumes an equal distribution of notes throughout the chord. Often, this is not the case in music. Chords are typically voiced more densely in higher registers, so chords with wide ranges often have unequal registral distributions of notes. For example, a common voicing of a C major chord might look like this:



In this example, the expected interval value of this chord decreases from about 12 at the bottom of the chord to about 4 at the top. Since a single density parameter is used to construct a whole chord, the expected interval must change as a function of register as the chord is being constructed.

To accomplish this, each time a note is selected the expected interval value is multiplied by a scalar that varies as a function of register. This function is linear, and the slope of this line is controlled by the density-scaling parameter:



So as the chord generator leaps through the chord range choosing notes for the chord, it constantly adjusts the expected interval to maintain more spacious voicing in lower registers.

Often it is desirable to be able to require a root-position chord. This is the role of the force-root flag. When the chord generator begins at the bottom of the chord range, it checks the force-root flag. If the flag is set to 1, it will seek out the nearest chord-root pitch and use that pitch at the bottom of the chord, regardless of the which pitches are currently eligible.

3.1.3.4 Harmonic Coloration

As described above, the chord generator leaps through the chord range selecting pitches with which to construct the chord. Each time the chord generator selects a pitch, it first decides whether the pitch should be a chord tone or a non-chord tone. The chord generator understands three classes of pitches:

- Chord tones, which includes the *required* and *auxiliary* scale degrees as defined in the chord-type, as well as the chord-root.

- Color A tones, which includes the *modal-non-chord* scale degrees as defined in the chord-type.

- Color B tones, which includes the *chromatic-non-chord* scale degrees as defined in the chord-type.

When a new pitch is chosen, the chord-tone-weight, color-a-weight, and color-b-weight parameters weight the respective likelihoods of a pitch from the corresponding tone group being selected. (This is identical to the function of the melodic tone weighting parameters as described in section **3.1.2.2**.) Color A is used for mild, consonant coloration, whereas color B is for more dissonant coloration.

Because the pitches are chosen entirely probabilistically, there is some chance, even with very low (but non-zero) color-a/b weights, that *none* of the required pitches for a particular chord-type will be selected for a chord. It is desirable under some circumstances to prevent this occurrence. This is the role of the force-required flag. If the flag is set to 1, once the chord has been constructed, the chord generator sweeps back through the chord, surveying it for the presence of the required pitches. If any of the required pitches is absent, the nearest non-chord neighbor will be "nudged" over to the required chord tone.

3.1.3.5 Dynamics

The chord generator uses the exact same method as the melody generator for determining key velocities (see section **3.1.2.4**).

3.1.3.6 Duration

The chord generator uses the exact same method as the melody generator for managing durations (see section **3.1.2.5**).

3.1.3.7 Procedure Summary

The chord generation procedure can be summarized in the following pseudo-code:

```
for loop: from  (register  -  0.5(size))
          to    (register  +  0.5(size))
          step  (expected interval as ƒ(density, density-scaling, last key #))

                if (first time through the loop) and (force-root = 1)
                     select nearest chord-root
                else choose pitch group as ƒ(color-a, color-b)
                     select the nearest note of that type

if (force-required = 1)
          verify required notes in chord

choose velocity as ƒ(dynamics, rhythm generator offset)
send all note-on's
send all note-off's for expired chords as ƒ(duration, activity, slot #)
```

3.2 The Input Sequence

The input sequence is simply a MIDI sequence containing the basic musical materials with which the user wishes to experiment. This sequence can be as short as a measure, or as long as a whole piece.

In the case of a very short sequence, the probable intent is that once the sequence is converted into a seed, the seed will "loop". So typically, short input sequences are more "groove"-like in character. Then, as seeds, these grooves may be cumulatively transformed and molded over a long period of time. Additionally, different seeds may be rapidly recalled from a database. Recalling a similar seed might nudge the music in a new direction, whereas recalling a contrasting seeds could veer the music more abruptly. The user thus has much control over the shape and evolution of the piece as a whole.

In the case of a very long sequence, the probable intent is that its seed will reproduce the sequence only once. Thus, the user sacrifices control over the large-scale form of the piece. Instead, the user concentrates upon expressively inflecting this regeneration in order to evoke novel and personal realizations of the piece.

The input sequence consists of two components: 1) the MIDI note sequence, and 2) an accompanying "harmony track", which demarcates the desired harmony (chord root and type) underlying each portion of that MIDI sequence.

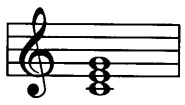
3.2.1 The Note Sequence

The note sequence is simply a MIDI score: it contains a list of MIDI note numbers, velocities, and on/off timings. The input sequence should *not* be a human performance of a score. Namely, it is assumed that the tempo is constant throughout, and that all rhythms are quantized.

Additionally, the musical content of the sequence should be homophonic, i.e., it should contain a single musical "strand" as described by Barlow (see section **2.4**). Multiple strands in a single input sequence will result in a seed which represents a musical texture containing multiple musical entities. In order for each of these independent musical entities to have its own seed, the various strands should be separated into multiple input sequences.

3.2.2 The Harmony Track

Parallel to the note sequence is another track containing note numbers which mark changes in chord root and controller values which mark changes in chord type. The role of the harmony track is to give the analyzer information about the harmonic functions of all pitches in the sequence so that it can extract envelopes describing the relative balance of chord tones and non-chord tones. Without a chord root and type as a harmonic reference for each region of the input, this analysis is not possible. For example, imagine that the following chord appears in the input sequence:



This chord could be interpreted as a C major chord with no non-chord tones. Or it could be interpreted as an Ab major 7th chord with chromatic coloration and no root. Or it could be interpreted a hundred other ways. Thus a chord root and type are necessary for determining the harmonic make-up of a group of pitches. (Of course, expert systems exist which can make intelligent guesses within a specific genre and context about probable chord root and types given a set of pitches. But such heuristics are beyond the scope of this thesis.)

3.3 The Analyzer

The role of the analyzer is exclusively to convert an input sequence into an equivalent parametric seed. The analyzer does not perform in real time; it operates off-line. Thus, the analyzer is used in *preparation* for interaction with the seed music system, not as part of that interaction.

In theory, the analyzer could be adapted to work in real-time. This would allow, for example, a keyboard performer to constantly send a stream of MIDI notes through the analyzer, resulting in continuous generation of new seeds.

But for this system I have been more concerned with real-time *manipulation* of music, not real-time *accumulation* of the music to be manipulated.

In the first stage of analysis, the harmony track of the input sequence is directly transmitted to the seed so that it may later be reused during the regeneration process. The second stage of analysis is rhythmic parameter extraction. The final stage of analysis is the melodic or chordal parameter extraction.

If any simultaneous note-on's occur in the input sequence, the sequence must be converted into a chordal seed. In the absence of any simultaneous note-on's, the user must specify whether the input sequence should be converted into a chordal seed or a melodic seed.

3.3.1 Rhythm Analysis

The rhythm analyzer extracts envelopes for the activity, syncopation, and cross-meter parameters. No envelopes are extracted for cross-rhythm, as any cross-rhythm in a given meter can be "manually" reproduced by activity and syncopation envelopes. The cross-rhythm parameters are thus only used in the control stage of the system, not in the seed representation itself. (Note that the correct activity envelopes alone could produce syncopation, but in the control stage, the seed is more intuitively responsive when it does contain syncopation envelopes.) No envelopes are extracted for cross-accent or syncopated-accent, because dynamic contours are handled entirely by the melody and chord analyzers. The metric mode (simple or compound) for analysis can be specified by the user, or it can be determined automatically through a comparative tally of note-on's occurring in the simple and compound activity slot series.

Most configurations of rhythm parameter settings generate rhythms probabilistically. Thus, there is not a one-to-one correlation between most coordinates in parameter-space and the resulting rhythms in note-space. Although the "cloud" of resulting rhythms will be qualitatively quite similar, the exact rhythm cannot be predicted.

However, there is a small number of parameter-space coordinates for which the generator does behave 100% deterministically. For example, on the activity axis, these deterministic nodes are the activity tiers, where the corresponding pulse level will always be produced. On the syncopation axis, these nodes are the upper and lower boundaries of that axis, where a syncopation will be induced with a "probability" of exactly 1 or 0. It is these deterministic sites which the analyzer uses to create a seed which will reliably reproduce the rhythms in the input sequence. In order to perform the parameter extraction, the rhythm analyzer needs only to sequentially parse the input into fragments, each of which can be reproduced by a deterministic set of parameters nodes. This process is simply a pattern-matching operation using the activity and syncopation slot series.

3.3.2 Chordal Analysis

The chord analyzer can extract chord parameter envelopes which will reproduce chordal figurations which are extremely similar to those in the input, but it cannot *exactly* reproduce some chords reliably. This is because of two significant limitations in the chord generator: probabilistic selection of non-chord tones, and a linear density scaling function.

When the color-a and color-b parameters are at 0, a stationary set of chord parameters will exactly reproduce the same chord repeatedly. When coloration is used, however, some randomness enters the generation algorithm. As each pitch of a chord is selected from the bottom up, the color "dice" are rolled to determine whether a chord-tone or color tone will be chosen next. Thus, given a stationary set of chord parameters that includes a non-zero coloration parameter, repeated chords will have approximately equivalent proportions of chord and non-chord tones, but the voicings of those tones will fluctuate.

Additionally, the spatial distribution of notes within a chord is governed by a linear scaling function. Thus, the density distribution can either be constant throughout the chord, or the distribution can linearly increase in higher registers. Any chord containing a density distribution which is not of this type cannot be exactly reproduced by the generator.

All other chordal parameters (size, register, density, dynamics, and duration) are deterministic and versatile, and can thus reliably reproduce the corresponding features of any chord. The randomness of coloration voicing and the density distribution limitations discussed above are not significant drawbacks in musical contexts which do not rely deeply upon voice leading, e.g., most pop music. These limitations are, however, significant drawbacks in musics in which voice leading plays a central role, e.g., Western classical music (see section **4.3**).

3.3.3 Melodic Analysis

The melodic parameters chord-tone-weight, scale-tone-weight, ripe-tone-weight, direction, leap-probability, leap-size all have deterministic nodes at the extreme boundaries of their axes, and the melodic parameters dynamics and duration are uniformly deterministic. Thus, given a specific starting pitch, melodic parameter can reliably reproduce an input melody.

One problem, however, is that the melody parameters only describe a melody's shape; they do not describe the registral position of this shape. Thus, in order to exactly reproduce an input melody, the seed must somehow contain the note number of the first note in the melody.

This problem is solved by the analyzer in the following way: Because the melody parameters only address melodic motion, no melodic parameter information is needed for the first note of the seed, because it has not moved *from* anywhere. This property of a melodic seed can be exploited to provide an absolute reference point on which the melody can begin. The analyzer can place a "fictitious" note *before* the input sequence, and use that note as a reference point from which the melody parameters can deterministically reproduce the first "real" note.

So, the leap-probability parameter for the first real note is 127, and the direction parameter for the first real note is 127. If the fictitious note has MIDI note number 0, then the desired MIDI note number for the first real note of the melody is simply equal to the leap-size parameter.

3.4 The Seed

The seed is the consolidated rhythm, melody, and chord parameter envelopes, as well as the underlying chord changes, which will reproduce the input sequence when fed to the generator.

3.4.1 Why the Seed Representation?

What, you may wonder, is the purpose of going to all of this trouble to convert a sequence into a representation which, when reconstituted, will simply reproduce the sequence? The purpose and the power of the seed representation is that, unlike a score, music in this form is extremely susceptible to easy transformation. Simply changing a number, like the syncopation value, will elicit an expected musical response. Performing such operations on a score is a far more complicated task. (Parameters such as syncopation represent "mid-level" musical properties. In order to have direct, simple control over higher-level adjectival properties, such as "quirkiness", additional parameter abstraction is necessary (see section **3.5.2**).)

An additional benefit of the parametric representation is that it lends itself to inter-seed interpolation. Instead of using "adjectival" axes for transforming a musical pattern, a user may wish instead to make a musical pattern "more like" another pattern. This mechanism suggests a powerful model for "navigation" within musical space. A feature-based representation is ideal for this sort of transitional blending (see section **2.5.3**).

3.4.2 The Seed's Harmony Track

Normally, the harmony track contained in the seed (as defined in the input sequence) will sequentially update the chord-root and chord-type variables automatically as part of the generation process. Under some circumstances, a user may wish to have harmonic control independent of the harmony changes contained in the seed. In such a cases, harmony changes can be specified in real-time by a user. These user-specified harmony changes can be superimposed upon the seed's internal harmony track. Or, alternatively, the seed's internal harmony track can be disabled completely, and the user-specified harmony changes can be used exclusively.

Similarly, in the cases in which multiple seeds are generating in tandem, the parallel harmony tracks will either be superimposed upon each other, or all but one seed's harmony track will be disabled.

3.5 Real-Time Control

The parametric system I have described comes to fruition in the control stage. One of the many powerful features of the system is the flexibility with which it can be adapted to virtually any type of command interface. An interface could be as low-level as knobs for direct manipulation of parameters, or as high-level as, say, a dance performance analyzer which tunes parameters in response to a dancer's performance. The role of the interface designer is to find an intuitive set of control axes for a specific context, and then define the mappings between these axes in the interface and parameter groups in the system, such that gestures expressed by the user will effect intuitive responses from the system.

3.5.1 Range Calibration

Many of the parameters used by the analyzer/generator are defined such that any parameter value will correspond explicitly to some dimension in the output. For example, the value of the size parameter (see section **3.1.3.2**) corresponds exactly to a maximum semitone span for a chord.

However, the ranges of the parameters, defined for conceptual clarity, often do not correspond to ranges which are practical for manipulation in a user interface. For example, consider the density parameter (section **3.1.3.3**). Under normal circumstances, a practical use of this parameter would only include densities ranging from about 111 (a *very* sparsely voiced chord, with an expected interval of 1.5 octaves between notes) to 127 (a densely voiced block chord).

Consequently, the first task in any user interface design is to calibrate the range scales for the parameters which will be addressed by that interface. For example, a comfortable user interface for controlling chord density would *not* give the user control over a range from 0 to 127, but instead over a range from 111 to 127 as described above.

3.5.2 Parameter Combination

Under some circumstances, a user might want direct control over a single parameter. For example, a composer might use this system to tweak specific features of region of a composition.

Under many circumstances, however, direct control over a single parameter is not desirable. A user might simply want one interface axis that corresponds to a qualitative effect in the generated music. This effect would be induced by simultaneous manipulation of several parameters. For example, a user might want to add a bit of "quirkiness" to the music by moving a joystick in one direction. Increased "quirkiness" might be defined as an increase in both dissonant harmonic coloration (color-b) and metric irregularity (syncopation, syncopated-accent, cross-rhythm, cross-accent, cross-meter). Additionally, each of these parameters may need to be added in different proportions to achieve a specific effect.

Part of the task of interface design, then, is to group parameters into combinations which will yield intuitive musical effects. Typically this involves categorizing parameters into different adjectival groups. For example, consider an extremely high-level interface, which abstracts all control onto only three axes. Given only three axes with which to work, the designer must make an effort to find underlying similarities between otherwise unrelated parameters. One realization might be the following three groupings:

- A *discordance* axis, along which parameters governing syncopation, cross-rhythm, cross-metrics, harmonic coloration, and timbral percussiveness and inharmonicity are increased.

• An *activity* axis, along which parameters governing rhythmic activity and harmonic rhythm are increased.

• A *magnitude* axis, along which parameters governing dynamics, ensemble size, and the grandiosity of the orchestration

(For parameters describing timbre, harmonic rhythm, and ensemble qualities, see chapter 4.) Each of these axes contains a group of parameters which will collectively elicit an intuitive, characteristic system response. Thus, a user will quickly learn to understand the consequences of displacing the interface along one of these axes.

3.5.3 Parameter Modulation and Seed Interpolation

Manipulation of seeds falls into two broad classes: transformation through direct parameter modulation, and transformation through inter-seed interpolation.

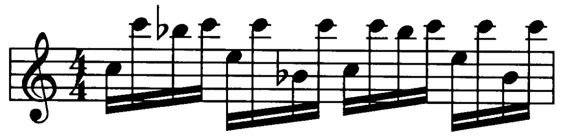
The simplest method of transforming a seed is to directly modulate (boost or cut) the value of some parameter or group of parameters to "compositionally filter" the generated output. The result is the emergence of "adjectival" control axes: one control axis elicits a qualitatively common transformation on any seed ("more/less syncopated").

A more subtle method of transforming a seed is to interpolate its parameter values with those of another seed. This is another of the many great benefits of a parametric system. In addition to wanting to boost or cut the values of specific parameter groups within a seed, and user may also express the desire to make one seed "more like" another seed.

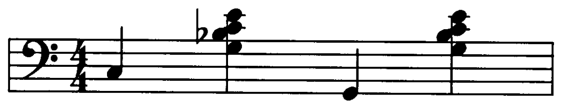
Interpolation between any two seeds is by no means a simple problem, however. Of course, it is relatively easy for a *human* to listen to two dissimilar musical patterns and internally imagine a hybrid pattern which contains the salient features of both source patterns. But this sort of operation involves intelligent, sophisticated recognition and manipulation of those features and underlying patterns. Simple parametric interpolation (say, a linear weighted average of parameter values as a function of "position" between two seeds) often will not produce an intuitive hybrid pattern if the source patterns are dissimilar. However, the parametric representation *does* take us substantially closer to this goal.

Let us take as an example the following two relatively dissimilar figurations of a C dominant 7th chord. The first is an Alberti-like figure, and the second is a march-like figure:

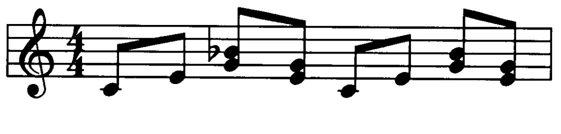
1)



2)



A median linear interpolation of the <u>activity</u>, <u>size</u>, and <u>register</u> parameters would yield the following hybrid figuration:

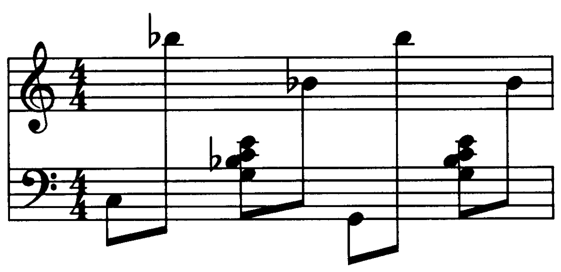


While this is certainly a plausible interpolation, it is hardly an interesting one, as the salient features of both source patterns have been "watered down". A more powerful interpolation model can be used instead, which combines the pure interpolation and a type of parameter-interleaving.

The first step of this process simply involves linear interpolation of the rhythm parameter envelopes of the two boundary seeds, producing a blended rhythm profile for the intermediary seed. The second step involves interleaving chord/melody parameter values from the boundary seeds within the intermediary seed.

On the onbeat slot positions of the intermediary seed's current pulse level, the chord/melody parameter values from the boundary seed having lower activity are copied directly into the intermediary seed. Likewise, on the offbeat slot positions of the intermediary seed's current pulse level, the chord/melody parameter values from the boundary seed having higher activity are copied directly into the intermediary seed. This process avoids diluting the salient chord/melody properties of the boundary seeds during rhythmic interpolation.

This parameter-interleaving method usually produces a more interesting blend of two boundary patterns than would a pure interpolation. For example, a median interpolation using this method on the march and Alberti figures from above would look like this:



Most people would agree that this is a more natural composite of the two boundary seeds than was the result obtained through pure interpolation. Naturally, this method will not work perfectly with all seeds, but the interleaving technique is a powerful initial model for superior interpolation.

3.5.4 Stable and Unstable Transformation

Additionally, two distinct modes of transformation, "stable" and "unstable", apply to both the modulation and interpolation control types.

Consider a joystick interface. When the user applies no force to the joystick in any direction, it rests in the center position. An applied force in any lateral direction will displace the joystick in that direction. When the force is removed, the joystick returns to center.

An unstable transformation is only sustained for as long as the control gesture expressed by the user. So, for example, when the user displaces the joystick along the "quirkiness" axis, the appropriate parameters are modulated by an amount proportional to the magnitude of the displacement. When the joystick is returned to center, the "quirkiness" parameter offsets disappear, and the seed returns to its default stable state. (In other words, a position vector in the interface corresponds to a position vector in parameter-space.) Unstable transformation is most useful for "inflection" gestures, where the user only wishes to depart briefly from the central musical materials for expressive effects.

A stable transformation is preserved even after the control gesture has ceased. So, for example, when the user displaces the joystick along the "quirkiness" axis, the magnitude of the displacement determines the *rate* at which the appropriate parameters will be modulated. When the joystick is returned to center, the "quirkiness" parameter offsets are preserved, and the stable output of the seed is now, well, quirkier. The seed has departed from its default state. (In other words, a position vector in the interface corresponds to a velocity vector in parameter-space.) Stable transformation allows cumulative transformation, gradual or abrupt, over long spans of time. Thus is most useful for "musical navigation", i.e., the traversing of musical spaces, guided by a set of feature-based axes.

3.5.5 Automated Pattern Variation

One of the glaring inadequacies of most automated computer accompaniment systems is their use of looping accompaniment sequences. Sequence repetition has limited utility. For example, in improvisatory musics, a performer will constantly insert minute variations into a repeating groove pattern: an added riff here, a small fill there. In such contexts, strict pattern repetition has a blatantly artificial quality.

Another of the attractive features of the seed system is the ease with which strict repetition can be eliminated. A seed is composed entirely of parameters resting on deterministic nodes. Displacement from these nodes, even a very slight displacement, moves the generator into a state of probabilistic generation. The randomness, however, is constrained by the algorithms employed by the generator. The parametric representation is sufficiently sound that two proximal coordinates in parameter-space produce two qualitatively similar behaviors in note-space.

So under circumstances where automated variation is desirable, noise can be added to a seed in order to "jiggle" the parameters off of their deterministic sites, causing fluctuations in the output. The resulting fluctuations do not contrast drastically with each other. In fact, these variations are remarkably similar to the types of variations exhibited by human performers. Thus, even in the absence of user-guidance, the seed system can exhibit some musicianship of its own. In effect, the system takes up some of the real-time control workload.

Under circumstances where several seeds are generating in parallel, often it will not be desirable for the user to "pilot" all of the seeds simultaneously; some of the seeds will be accompanimental. In such cases, automated variation serves the function of maintaining interesting musical behavior in the accompaniment.

3.5.6 High-Level and Low-Level Control

Before computers, there were essentially only two musical control interfaces. The first, and by far the most ubiquitous, was the low-level interface: the musical instrument. For an elite few, however, there was also a high-level interface: orchestral conducting.

Since the advent of computers, this spectrum of control abstraction has exploded in both directions. Now, at the lowest level, a musician or sound engineer can, for example, precisely adjust the spectral characteristics of a sound by tuning a synthesis parameter. The highest level is inhabited by interfaces such as those discussed in this document, which can effectively reduce musical control to a handful of axes.

Although I clearly advocate the exploration of high-level musical interfaces, I also believe that it is important to consider the drawbacks of high-level control. Every location in the control spectrum has its relative strengths and weaknesses.

Some benefits of high-level control interfaces:

- They are easy to learn. A user can quickly develop mastery of the smaller number of control axes. With a high-level musical interface, a user can rapidly acquire expressive power that would literally require years of training on a conventional instrument.

- They expand the realm of control. Because the underlying system is doing much of the work, the user is free to govern a larger and more complex system of variables. A high-level musical interface can allow a single user to manipulate a vast musical palette by shaping the behavior of a whole musical ensemble in real-time. Even a virtuoso instrumentalist on a conventional instrument is confined to the limits of that instrument.

Some benefits of low-level control interfaces:

- They allow specificity. A user can precisely adjust the most refined detail of a system. High-level systems, by relieving the user of some of the work to be done, also rob the user of the power of exactitude.

- They require, and thus teach, sophisticated understanding of a system's mechanics. The process of learning to play a conventional instrument, or learning to compose a piece one note at a time, gives a musician a profound understanding of musical structures that could never be acquired simply through the manipulation of a few high-level control axes.

There is a danger, then, of developing high-level interfaces which function as a musical "crutch", masking the subtleties and complexities of music composition and performance from the user. There is a danger that such an interface suppresses a user's individuality by automating too much of the creative process. These hazards must be contemplated when designing intelligent musical instruments.

One important criterion by which we can evaluate an interface for these deficiencies is the degree to which the quality of the system's output is determined by the skill of the user. A failure, by this criterion, would be a system in which all users sound alike; a novice user will sound comparably skilled to an experienced user.

Clearly, this must be avoided. When under the command of an inexperienced (or lazy!) user, the system must *not* continue to produce good music. Otherwise, what skill is there to be acquired? What expressive channel does the instrument provide? None.

In other words, the system's learning curve must not plateau too quickly. While it is important for initial, rudimentary skills to be accessible immediately, it is equally important for more sophisticated skills to be accessible only through sustained, attentive effort. The system must not do too much of the work.

3.5.7 Initial Experiments with Interfaces

Three simple performance interfaces have been developed in order to demonstrate some of the basic principles of the system. Each of the first two employs a pair of joysticks. The third is purely keyboard-based.

The two-joystick interface was chosen as a starting point because of the ease with which many axes can be rapidly varied in a precise and unambiguous fashion, and because the dexterity required to do so is not beyond the bounds of most people's comfortable skills. The keyboard interface was chosen simply to investigate the implications of using a conventional instrument as a front-end to the seed music system.

3.5.7.1 Joystick Interface 1: Chord-Control Model

The intent of the first joystick model was to give the user control of both types of seed transformation (interpolation and modulation) in a groove-type context, where the user could shape various chordal figurations over a repeating accompaniment.

In this model, the chord generator continuously produces chordal figurations in a piano voice. An accompaniment sequence (bass and drums) loops beneath these generated chordal figures. Additionally, a simple harmony track swaps the chord root and type back and forth between Bb7 and Eb7 every other measure.

In this model, the left joystick is used for seed interpolation. Four different seeds are assigned to the four joystick directions (forward, back, left, right). These four seeds have distinct characters: one is a low-register march figuration, one is a high-register Alberti-type figure, one is a wide sweeping arpeggio, and the other is a more restrained and lyrical figure. When the joystick is centered, a "flat" seed applied to the generator. By rotating the joystick around the periphery, the user interpolates rapidly between these various seeds in order to create novel figurations improvisationally.

The right joystick is used for direct parameter modulation. Rhythmic effects are assigned to one axis: pulling back boosts <u>syncopation</u> and <u>syncopated-accent</u>, and pushing forward boosts <u>cross-rhythm</u> and <u>cross-accent</u> (with a <u>simple-cross-ratio</u> of 0.75). Harmonic effects are assigned to the other axis: pushing left boosts <u>color-a</u>, and pushing right boosts <u>color-b</u>. Two thumb triggers on the joystick boost the <u>cross-metric</u> parameter. (The triggers also adjust <u>activity</u>, so that one trigger yields rapid cross-metrics of the higher pulse level, and the other trigger yields cross-metrics of the lower activity level.) (For an interface schematic, see the **Appendix**.)

Both the interpolation and the modulation controls are unstable; any position of the left joystick always corresponds to the same interpolation balance, and any position of the right joystick always corresponds to the same parameter adjustments.

3.5.7.2 Joystick Interface 2: Melody-Control Model

In the first joystick interface, the user is limited by the small selection of seeds which are assigned to the joystick axes. With the second joystick interface, the intent was to have a much wider range of musical possibilities available to the user, set in the context of jazz piano melodic improvisation.

The second joystick interface is interesting in that it uses no seed. Or, rather, the seed it uses contains only a harmony track (12-bar blues) and no parameter envelopes. All of the control is based upon direct modulation of the melody parameters. A bass and a drum sequence, as well as a chordal left-hand piano sequence, serve as an accompaniment. The generator is used to produce a pianist's right-hand improvisation.

The left joystick is used to affect rhythm. Forward displacement increases activity, and backwards displacement decreases activity. Displacement to the right adds syncopation. Displacement to the left adds cross-rhythm. Thumb triggers are used to induce cross-metrics (triplets) and glissandi (induced with a large activity spike).

The right joystick is used to affect melodic shape and pitch inflection. The forward/back axis controls melodic direction. The left/right axis controls tone group balance, with chord tones being at the left-most extreme, ripe tones being at the left-most extreme, and scale tones being centered. A finger trigger is used to induce leaps. A thumb trigger simply starts and stops the generator. A second thumb trigger uses a unique feature designed especially for this interface. When this trigger is pressed, the generator finds the most recent leap. Until the trigger is released, the generator then loops the parameter envelopes which were performed in the window between when that leap occurred and when the trigger was pressed. This allows the user to indefinitely repeat a "lick" which he/she is happy with. (For an interface schematic, see the **Appendix**.)

All parameter modulation is, of course, unstable. Because of the absence of any seeds, this interface requires considerably more practice to control comfortably than does the first joystick interface. The benefit, of course, is that mastery of this interface gives a user a greater variety of available expressive gestures than is accessible through the first joystick interface.

3.5.7.3 Keyboard Interface

The third interface is purely keyboard based. The intent of this interface was to demonstrate the power of feature-based control within the context of existing keyboard-based automated accompaniment systems.

First of all, it includes keys for changing the harmony track in real-time. This is a standard feature in all existing auto-accompaniment products.

Second, it includes keys for database recall. Seeds in the database are sorted into "clusters". One cluster contains a group of compatible chordal seeds, a group of compatible melody seeds (bass grooves), and a drum sequence. By "compatible", I mean that frequent transition between these seeds through database recall will be relatively smooth. Rapid recall of different seeds within a cluster will be used for improvisatory inflections of the accompaniment. Recall of a new cluster produces a greater contrast, and is used for procession from one section of a piece to the next.

Third, it contains transformer keys, which are used to directly modulate parameters. the velocity with which these keys are stuck determines the magnitude of the modulation of the corresponding parameters. (High key velocities boost values. Low key velocities cut values.) Normally, this modulation is unstable. The transformation disappears as soon as the transformer key is released.

Fourth, it includes keys for database storage. A recording mode can be enabled, in which the transformer keys are stable, i.e., transformations are cumulative. When the user is pleased with a stably transformed seed, it may be stored for future real-time recall.

Finally, it includes a real-time keyboard performance analyzer. Normally, when a pedal is depressed, the keyboard is simply used for improvisatory melodic playthrough over the accompanimental seeds. When the real-time analyzer mode is enabled, however, parametric "tendencies" in the melodic performance are translated into automated seed transformation. Thus, when the soloist gets more excited or more calm, more "in" or more "out", the accompaniment follows. (This mimicry, although often a lot of fun, is hardly an adequate model for interactive automated accompaniment. This is addressed in section **4.2**.)

Another feature of the database is that the seeds used contain parameter envelopes which are not on deterministic parameter nodes. Thus, they work quite well as autonomous accompanists.

4 Future Work

The system described in this document is not yet a complete platform for realizing its countless potential applications. An abundance of additional work must be done.

4.1 Timbre Parameters

One conspicuous omission from this document is any attention whatsoever to timbre. Control of timbre, as pertinent to this project, falls into three broad classes:

1) *Orchestration,* in which generator output is continuously redirected to different instruments as a function of the relevant timbre parameters. Different synth patches are methodically organized into a multidimensional timbre space which indexes properties such as percussiveness, sustain, brightness, inharmonicity, etc.

2) *Tone,* in which the timbral properties of a single synth patch are fine-tuned in real-time as a function of the relevant timbre parameters. This type of control is used to achieve expressive effects through the shaping tone color.

3) *Effects*, in which the properties of ubiquitous signal processing effects (reverberation, delay, flange, etc.) are tuned in real-time as a function of the relevant timbre parameters.

I avoided the issue of timbre during development of the seed music system for several reasons. First of all, timbre is a large enough problem, even in this restrained context, to be a thesis topic of its own. I wanted to concentrate exclusively on the problem of codifying pitches and rhythms. Additionally, to date, there is no ubiquitous high-level model for sound synthesis. Any timbre parameters I had defined would simply have been mappings designed to accommodate the idiosyncrasies of a specific synthesis algorithm. However, a complete parametric music generation system *will* eventually require a comprehensive timbre representation of some kind.

4.2 Ensemble Parameters

All of the parameters discussed in this document deal with musical behavior *within* a single seed. None of them addresses the many aspects of musical relationships between the respective constituents of an ensemble. Barlow tried to address this issue by defining "strand conformity" (see section **2.4**), i.e., the "sameness" or "differentness" of two strands at any moment. A more complete system will include a parametric representation for these relationships.

Under many circumstances, ensemble conformity is desirable. A single control gesture can be amplified by accordant responses in all strands. But many musical properties depend upon lack of conformity between ensemble constituents: Increased rhythmic activity in one strand has unique properties when contrasted by decreased activity in another strand; contrary melodic motion unifies a melodic structure; polyrhythm relies upon rhythmic oppositions between strands; harmonic departure in one strand functions in the context of harmonic grounding in another. This abundance of inter-seed relationships must be codified.

4.3 Percussion Parameters

The seed system has not yet been applied to ensemble percussion. I believe that a parametric representation will greatly improve the quality of the high-level control of ensemble percussion which was developed in the DrumBoy project (see section **1.1.2**). The size of the ensemble, the character of the instrumentation, the complexity and activity of the patterns, could all be controlled parametrically.

The main new task in developing a percussion generator would be to define rules for the cases in which multiple instruments function as a single instrument. For example, the kick and snare drums in a rock kit function as a single strand. Knowledge of their complimentary behaviors within a meter would have to be built into the rhythm generator, so that when, say, the activity is increased within a kick/snare strand, the generator will know whether the kick or snare should be selected to play the additional notes.

4.4 Harmonic Parameters

Other than the simple model for harmonic coloration (see section **3.1.3.4**), there is no current method for parametrically governing harmonic behavior. The harmonic lattice of the generator is defined either by the harmony track present in the seed or by harmony changes given explicitly by the user.

A framework for defining genre-specific harmonic heuristics should be constructed, so that a harmony track can be transformed parametrically. For example, a parameter for harmonic rhythm should be capable of accelerating or retarding harmonic pacing by adding chord changes to the harmony track or removing chord changes from the harmony track which are idiomatically appropriate. The role of various progressions of chords (e.g., harmonic departure or grounding) must similarly be defined parametrically.

Another major inadequacy of the current system is that voice-leading within a seed is impossible. The chord generator has no knowledge of the preceding chord that was generated. Once given memory of the preceding chord, then the generator would need a framework for defining idiomatic voice-leading rules. Chordal continuity could also be represented parametrically, so that the degree to which the chord generator seeks "paths of least motion" for its pitches could be controlled.

Additionally, there is currently no inter-seed communication which would allow pitch choices in one seed to affect those chosen in another seed. For example, voice-leading between a pair of melodic seeds is not possible. Similarly, block chord voicing as a function of the melody pitch (common practice in jazz) pitch is not possible. This type of strand interdependency is essential in many musics, and it must be incorporated into the generators for the seed system to be versatile in many musical contexts.

4.5 Seed Length Parameters

None of the existing parameters addresses the stability of seed length. Often it is desirable for a repeating figure to stretch or compress itself for rhythmic and melodic interest. Some representation is needed to govern this type of instability.

4.6 Database Navigation

The power of the seed music system would be greatly expanded if it were to operate in conjunction with a sophisticated database recall system. The database could be organized as a multidimensional music space into which seeds are automatically sorted by an indexer. Unfortunately, the problem of designing an effective sorting method for the indexer is a complicated one. Simplistic sorting methods, such as deriving coordinates from the average values of parameter envelopes, are clearly inadequate. "Meta-parameters" must be defined: high-level parameters which describe the *shapes* of the lower-level parameter envelopes. These interdependent parameter contours determine the salient features of a seed, and so these contours must be the primary concern of the indexer.

If a convincing feature-based sorting method were developed, however, the resulting axes would provide a powerful model for navigation through musical space. A database could contain seeds that would function as nodes in the space. Smooth transition between these nodes would simply employ seed interpolation.

4.7 Performance Parameters

The current seed music system is primarily designed to make compositional decisions; it determines which pitches and rhythms should be played at a given instant. These compositional decisions are guided improvisationally, so the compositional act does have a performance dimension to it. But the current parameters almost entirely neglect the issues associated with expressive performance.

A note generator is only half of an ideal interactive music system. A more complete system would also include a sort of output filter that would be guided by a set of parameters designed for expressive performance of any score. These parameters would govern the innumerable degrees of freedom in musical performance, such as pitch inflection, vibrato, tone color, and the microscopic and macroscopic uses of tempo, dynamic, and articulation. A control mechanism of this type would have applications not only in expressive performance of pieces composed in real-time, but also in expressive performance of pre-composed pieces.

4.8 Musical Intelligence

The current system has no real musical intelligence or musical identity of its own. In the absence of a person to control it, it lacks an agency for generating interesting transformations of parameter envelopes. For example, the so-called "melody generator" cannot actually *compose* a melody. It can only reproduce a melody from a seed. Without a seed, and in the absence of human guidance, it will generate truly loathsome (or truly comical, depending upon your mood) melodies.

A possible future direction for the seed music system is the development of compositional facilities which are designed to exploit the implicit musical materials contained within a seed. A composing agency would recognize implications of certain envelope behaviors, and automatically induce musically plausible and interesting cumulative transformations of those envelopes. The machine would thus become capable of composing music of its own. Needless to say, intelligence of this sort is beyond the current compass of this system.

5 Applications: Reaping the Harvest

The seed music system I have described is just the beginning of a model for music generation which I believe will eventually transform people's relationship with music. Although the scope of the existing system is still somewhat limited, the inherent principles are solid, and the system will serve as an effective platform for future expansion. I am confident that a system of this type will eventually pave the way for an inestimably large variety applications.

5.1 A Palette of Seeds for a Compositional Canvas

Composition is a labor-intensive process. A composer can easily spend several hours milling over the details of a few seconds of music. This piecemeal assembly of a musical passage gives the composer the power of absolute specificity. This specificity, however, is also a form of confinement. It prohibits the composer from having the freedom to experiment quickly and liberally with broad gestures in large volumes of music.

One immediately obvious outgrowth of the seed concept is an entirely new model of the compositional process. Consider a "parametric sequencer", if you will. A composer could start simply by assembling a collection of musical materials which he/she wishes to employ within a composition: small snippets of MIDI sequences containing rhythms and textures of interest. These scraps would then be converted into a palette of seeds. Small groups of these seeds could be placed sequentially in time to demarcate regions of a piece, creating a sort of compositional canvas. Transformation envelopes could be sketched in by the composer. The composer could then experiment with different implications of the seed materials by dynamically massaging these envelopes into new shapes. The interpolation techniques alone have powerful implications as a compositional device for gradual blending from one musical space into another. A few minutes of assembly would yield a vast and complex musical landscape with which to dynamically experiment. An environment of this sort would encourage a new compositional paradigm, in which the composer--expert or amateur--is free to construct a piece not one note at a time, but one compositional conception at a time.

5.2 Subservient Recordings: Playing Your CD-Player

During this century, audio recordings have become the central medium through which people hear music. The act of listening to a piece of music launches a complex set of internal processes is the ear of the listener. Abstract musical structures are churned about, built up and broken down, as the listener's mind forces itself to construct an internal representation of the experience. In this sense, listening to music is an active process. Nonetheless, the audio recording is a fixed object over which the listener has no external control. And thus in another sense listening is a purely passive experience.

In the future, however, the audio recording will cease to be a fixed object. The CD-player will become a musical instrument. Consider, for example, a pop composer working on his/her new album. After completing all of the tunes, that composer then works with his/her favorite music interface designer in post-production. The designer converts all of the arrangements into their equivalent parametric representations, and the composer and designer work together to define the control axes which the composer wants to become, in effect, part of the piece. By defining these control axes, the composer is defining the breadth and type of improvisational freedom he/she will be transferring to the listener: which alternative instrumentations might merit exploration, which harmonic liberties can be probed, which "rooms" are "off-limits". In this sense, another new compositional paradigm is implied: the role of the composer is not to choose all of the notes, but to construct the most interesting musical spaces for exploration.

If you pop this CD into your player and simply press "play", the CD flies on auto-pilot. It will play the "core" realization of the album that the composer intended the from start to finish. But as soon as you pick up the joysticks, or whatever physical interface is plugged into the back or your player, you can perturb this core and become an improvisational contributor to the compositional act.

Let us consider another scenario: Suppose a composer does *not* want to relinquish any compositional freedom to his/her listeners. (You can be certain that this will often be the case.) Instead, the CD-player will not allow the user to *compositionally* alter a recording, but instead the CD-player will accept real-time performance parameters from the user and use them to re-interpret the performance of that composition. So, for example, when a fan of Metallica buys the group's latest clarinet concerto, that person will not just be the passive listener, but the conductor as well.

There is no reason that this sort of experience needs to be restricted to a single user, either. Imagine several separate interfaces connected to a single CD-player. An ensemble of several users could jam with each other, with each individual piloting some portion of the recorded ensemble.

5.3 Responsive Accompaniment for Instrumentalists

The seed music system does not entirely neglect trained instrumentalists! For example, consider Machover's hyperinstrument paradigm (see section **1.1.1**). Machover has designed at least a half-dozen hyperinstruments for virtuoso musicians. Most of the accompanying compositions take the form of concerti for the hyperinstrument soloist and a computer "orchestra" (plus, in some cases, a chamber orchestra). Real-time performance analyzers use various sensor data to interpret high-level performance gestures made by the soloist through his/her instrument. These performance gestures then control various aspects of the computer's performance of its score.

However, to date, in *none* of these pieces does the computer actually compose any music. All of the possible realizations of the piece exist in the computer in one form or another, and the performer coaxes out the most fitting fragments of these realizations through features of his/her playing. If the computer's score were instead represented as seeds, and real-time performance analyzers were mapped to control axes of a generator, the breadth of possible expressive realizations of that score could be expanded dramatically.

5.4 Music Teaching Systems

One of the features of the parametric generator is that when some parameter is modulated, it will transform a specific property of the active seed, regardless of the contents of that seed. This feature has potential applications in interactive music education. For example, a student could rapidly gain an intuitive grasp of the meaning of rhythmic syncopation by using a knob to add syncopation to and remove syncopation from a wise variety of musical textures. Hopefully, a high-level controller of this type would not function as a buffer between the student and the mechanics of syncopation, but instead it would function as an intuitive and inviting entryway into these lower-level processes.

5.5 Sensitive Soundtracks

Interactive environments will eventually require music systems which are capable of responding dynamically to changes in these environments. Pre-composed music is inadequate for this task, as no finite set of sequences can effectively accompany the essentially limitless set of environment states in sophisticated interactive contexts.

To solve this problem, one need only define a thorough set of mappings between the relevant environment variables and the seed music system. Any combination of the seeds can be transformed over time as a function of these environment variable in order to change desired qualities of the music being generated.

Consider a video game in which the player navigates his/her way through some non-linear narrative. The player might pass through an inestimably large number of plot permutations: alternating tension and release, climax and resolution, success and failure, etc. The pace of these changes is governed the user. The direction of these changes is governed by the user. The accompanying music must also (indirectly) be governed by the user if the entire experience is to remain cohesive, accordant, and unified. Parametric music systems will inevitably play an increasing role in interactive environments which include important real-time sound components.

5.6 Beyond Music

The interface ideas suggested within this document are by no means relevant only to the problems of musical control. Consider the basic model: a complex system of low-level variables is abstracted into a high-level parameter representation. These parameters correspond to salient perceptual features of that system, and can even be combined further to describe higher-level properties of the system. Feature manipulations in parameter space then elicit the desired low-level changes in the system to achieve the desired output.

This basic principle has broad applications in a number of contexts. For example, consider a digital graphic art system. An artist, expert or amateur, would begin by sketching some primitive graphic materials. Salient properties of that image (colors, textures, shapes) could then be freely manipulated. Dozens of complex, transformed images could be quickly generated, allowing the artist to rapidly navigate through a landscape of visual aesthetics. Such a system might also be used for improvisatory real-time manipulation of animated computer graphics. Real-time gestural control of images ("image conducting") could spawn a wholly new artistic medium for improvisatory graphic performance.

Additionally, parametric representations in different mediums would promote inter-medium communication. For example, perceptual variables in a music system might be mapped to perceptual variables in an animated computer graphics system to construct a sort of synthetic synaesthesia.

5.7 Conclusion

As a composer in the computer age, I am affected by the lure of machine-assisted composition. I have watched dozens of composers spend months or years of their lives fidgeting with their computers so that the computers might compositionally emancipate them in one way or another. Often, these systems indeed prove to be quite useful to those individuals. Unfortunately, much of their endless tinkering is effectively wasted labor, because many of them are working in isolation to achieve a common goal: high-level expressive control of their musical ideas. I am hopeful that the seed music system described in this document will eventually evolve into a platform which offers liberated, enriching compositional pathways to a large number of musicians. Additionally, as an only marginally skilled instrumental performer myself, I am admittedly rather seduced by the prospect of finding a more expressive mode of improvisational composition and performance in which I can excel. I am confident that I share this need with many others for whom music is a centrally important part of life.

Appendix

The diagrams on the following pages illustrate the features of the two joystick interfaces described in sections **3.5.7.1** and **3.5.7.2**.

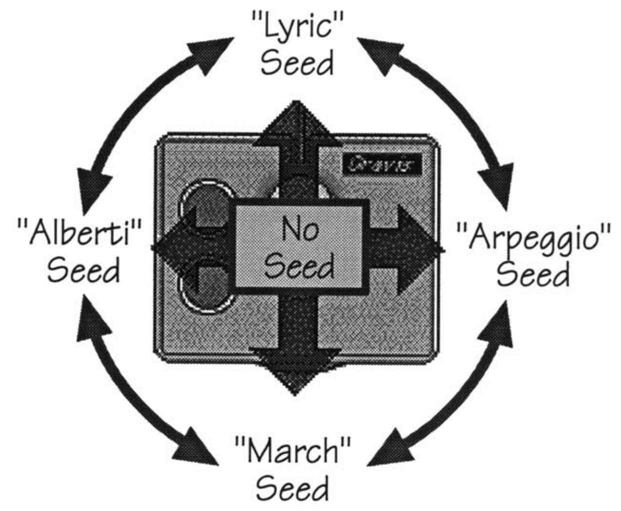
The model of joystick used for these interfaces is the MouseStick®II, by Advanced Gravis Computer Technology Ltd. The joystick images shown on the following pages are bitmaps copied directly from PICT resources in the MouseStick II Control Panel, Copyright © Advanced Gravis Computer Technology Ltd.

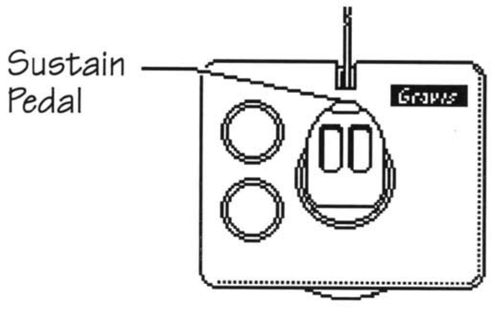
A video demonstration of both joystick interfaces is available from:

Hyperinstruments Group
Room E15-496
MIT Media Laboratory
20 Ames St.
Cambridge, MA 02139
(617) 253-0392

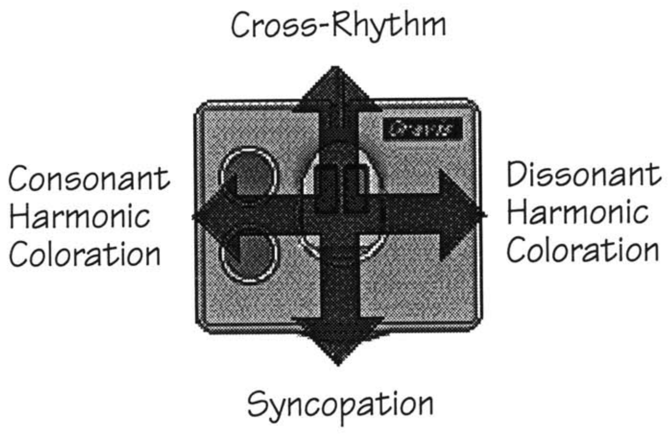
Joystick Interface 1:
Chord-Control Model

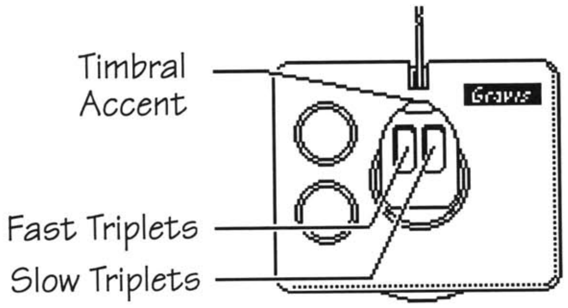
LEFT HAND:





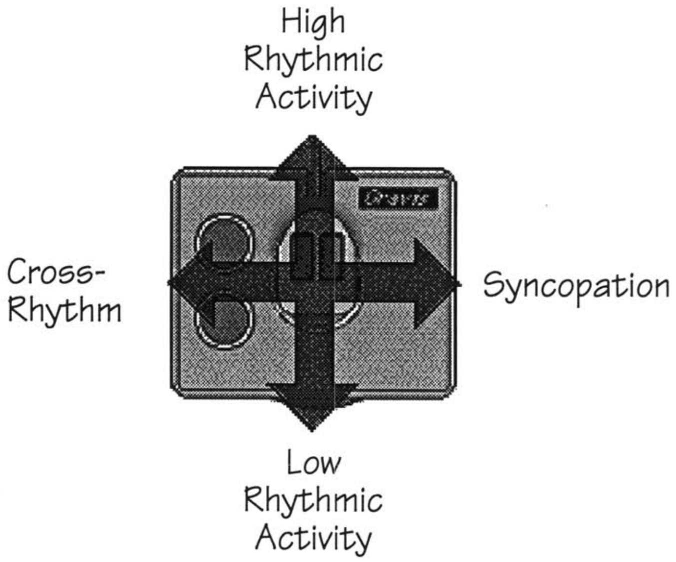
RIGHT HAND:

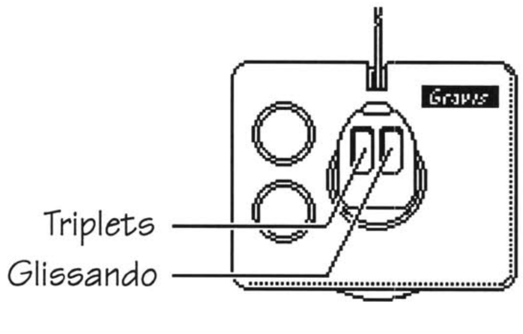

Cross-Rhythm
Consonant Harmonic Coloration
Dissonant Harmonic Coloration
Syncopation
Gravis


Timbral Accent
Fast Triplets
Slow Triplets
Gravis

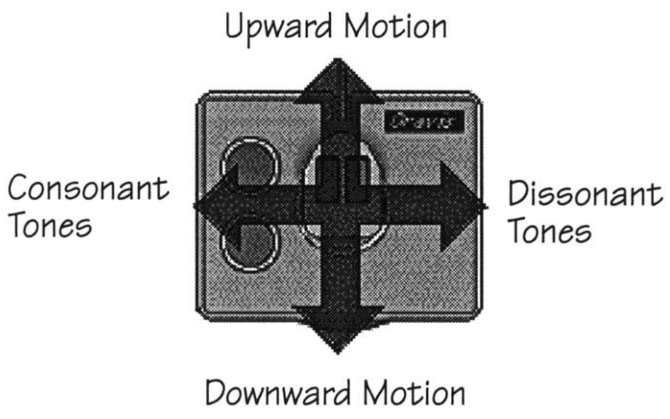
Joystick Interface 2: Melody-Control Model

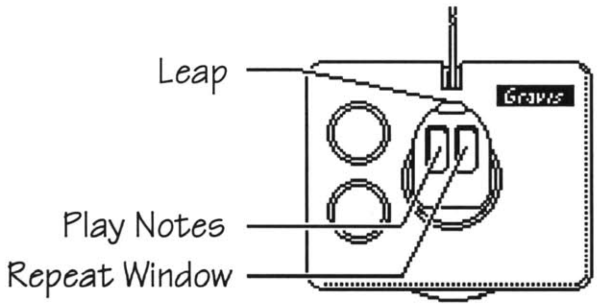
LEFT HAND:





RIGHT HAND:


Upward Motion
Consonant Tones
Dissonant Tones
Downward Motion
Gravis


Leap
Play Notes
Repeat Window
Gravis

Bibliography

[Arom91]	Arom, S., "The notion of relevance" *African Polyphony and Polyrhythm,* Cambridge, England: Cambridge University Press, 1991, pp. 137-157.
[Barlow81]	Barlow, C., "Bus Journey to Parametron (all about Çogluotobüsisletmesi)" *Feedback Verlag,* 21-23, 1981.
[Chung91]	Chung, J., *Hyperlisp Reference Manual,* available from the Media Laboratory, Massachusetts Institute of Technology, 1991.
[Cope90]	Cope, D., "Pattern Matching as an Engine for the Computer Simulation of Musical Style" *Proceedings of the ICMC 1990,* pp. 288-291.
[Cope91]	Cope, D., *Computers and Musical Style,* Madison, WI: A-R Editions, 1991.
[Dannenberg87]	Dannenberg, R.B., and Mont-Reynaud, B., "Following an Improvisation in Real-Time" *Proceedings of the ICMC 1987,* pp. 241-247.
[Lerdahl88]	Lerdahl, F., "Cognitive Constraints on Compositional Systems" *Generative Processes in Music: The Psychology of Performance, Improvisation, and Composition,* Sloboda, J., ed. Oxford: Clarendon Press, 1988, pp. 231-259.
[Lewis85]	Lewis, G., "Improvisation with George Lewis" *Composers and the Computer,* ed. Roads, C., Los Altos, CA: William Kaufmann, Inc., 1985.
[Loy89]	Loy, G., "Composing with Computers -- a Survey of Some Compositional Formalisms and Music Programming Languages" *Current Directions in Computer Music Research,* Matthews, M.V. and Pierce, J. R., eds. Cambridge: MIT Press 1989, pp. 291-396.
[Machover92]	Machover, T., *Hyperinstruments: A Progress Report,,* available from the Media Laboratory, Massachusetts Institute of Technology, 1992.
[Matsumoto93]	Matsumoto, F., *Using Simple Controls to Manipulate Complex Objects: Applications to the Drum-Boy Interactive Percussion System,* Master's Thesis, MIT Media Lab, 1993.

[McMullen87] McMullen, Neil., *Seeds and World Agricultural Progress* , Washington, D.C. : National Planning Association, c1987.

[Minsky75] Minsky, M., "A Framework for Representing Knowledge" (1975) *Readings in Knowledge Representation*, Brachman, R.J. and Levesque, H.J., eds. Los Altos: Morgan Kaufman Publishers, Inc., 1985, pp. 245-262.

[Rosenthal92] Rosenthal, D., *Machine Rhythm: Computer Emulation of Human Rhythm Perception,* Ph.D. Thesis, MIT Media Lab, 1992.

[Rowe91] Rowe, R., *Machine Listening and Composing: Making Sense of Music with Cooperating Real-Time Agents,* Ph.D. Thesis, MIT Media Lab, 1991.

[Rowe93] Rowe, R., *Interactive Music Systems,* Cambridge, MA: MIT Press, 1993.

[Wessel91] Wessel, D., "Improvisation with Highly Interactive Real-Time Performance Systems" *Proceedings of the ICMC 1991,* pp. 344-347.

[Wu94] Wu, M., *Responsive Sound Surfaces,* M.S. Thesis, MIT Media Lab, 1994.

[Zicarelli87] Zicarelli, D., "M and Jam Factory" *Computer Music Journal* 11:4 1987, pp. 13-29.