Hyperscore: A New Approach to Interactive, Computer-Generated Music

by

Mary Farbood

A.B. Music
Harvard University (1997)

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 2001

Author ....
Program in Media Arts and Sciences,
School of Architecture and Planning
September 1, 2001

Certified by ....
Tod Machover
Professor of Music and Media
Thesis Supervisor

Accepted by ....
Andrew B. Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Hyperscore: A New Approach to Interactive, Computer-Generated Music

by

Mary Farbood

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on September 1, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science
in Media Arts and Sciences

Abstract

This thesis discusses the design and implementation of Hyperscore, a computer-assisted composition system intended for users of all musical backgrounds. Hyperscore presents a unique graphical interface which takes input in the form of freehand drawing. The strokes in the drawing are mapped to structural and gestural elements in the music, allowing the user to describe the large scale-structure of a piece visually. Hyperscore's graphical notation also enables the depiction of musical ideas on a detailed level. Additional annotations around a main curve indicate the placement and emphasis of selected motives. These motives are short melodic fragments that are either composed by the user or selected from a set of pre-composed material. Changing qualitative aspects of the annotations such as texture and shape let the user alter different musical parameters. The ultimate goal of Hyperscore is to provide an intuitive, interactive graphical environment for creating and editing compositions.

Thesis Supervisor: Tod Machover
Title: Professor of Music and Media

Hyperscore: A New Approach to Interactive,
Computer-Generated Music

by

Mary Farbood

The following people served as thesis readers:

Thesis Supervisor	Tod Machover Professor of Music and Media MIT Media Laboratory

---

Thesis Reader	David Cope Professor of Music University of California at Santa Cruz

---

Thesis Reader	John Harbison Institute Professor of Music Massachusetts Institute of Technology

---

Thesis Reader	Michael Hawley Assistant Professor of Media Technology MIT Media Laboratory

For my parents

Acknowledgments

I would like to thank...

My advisor Tod Machover for giving me a place in his group and for encouraging me in every way possible.

Mike Hawley for being a thesis reader, a friend, and my favorite pianist.

My thesis readers John Harbison and David Cope for their time and patience.

Egon Pasztor for his many important contributions to Hyperscore.

Tristan Jehan and Ed Hammond for being great friends.

Golan Levin for never failing to make me laugh.

Paul Nemirovsky for many entertaining conversations.

My officemate of two years Gili Weinberg for giving me consistently good advice.

David Deveau and Teresa Marrin for making chamber music so fun.

Heather Childress for her help in many matters great and small at the lab.

My dear friends Junne Kamihara, Nicholas Flowers, and Alex Lai for all the support they've given me over the past years.

Bernd Schoner for his contributions to Palestrina and everything else in my life. His constant love and support have given me so much happiness.

My parents and sister Mina for their inestimable love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

How does one become a composer? More specifically, what essential musical concepts and skills need to be learned before it becomes possible to sit down and write something which would be considered an acceptable piece of music? In eighteenth-century Europe, this question had a relatively straightforward answer: rigorous training on orchestral and keyboard instruments coupled with instruction on theory and counterpoint. For most people this required years of concentrated study at an early age.

Music is no less a part of people's lives today, but the nature of musical experiences has changed dramatically. Although few people have the benefit of a rigorous musical education, the rapid advance of technology in the twentieth century has made it possible for people to be musically creative without going through the formal training required of a traditional composer. In the past fifty years, the field of interactive and computer-assisted composition has blossomed. Although there has been a proliferation of computer-based tools that have enabled both professionals and novices to explore musical creativity, the field is still relatively new and open to innovative work.

So how can computers help the potential composer? For one, they are good at expediting repetitive tasks. As a musician, one of the first computer programs I wrote for myself was a row generator. Given a twelve-tone row as input, the program would generate a table with all of the transpositions, retrogrades, inversions, and retrograde inversions of the row. It was an extremely handy tool for analyzing music (at the time, I was studying Alban Berg's *Lulu*, an opera where each character has his or her own characteristic row) as well as composing serial music.

But is a computer also good at filling in missing human knowledge or expertise? Is it possible for a user, untrained in music, to use computers to compose while still maintaining originality and personal style? These are some of the questions which the development of Hyperscore has attempted to address in a creative new way.

1.1 Structural elements in music

Hyperscore is a software tool which provides the user with a freehand drawing interface for composing music. It is designed to be accessible to people of all backgrounds—even those completely untrained in music. The only prerequisite required is the patience to go through the compositional process of listening and editing until a satisfactory result is achieved. The graphical environment provided by Hyperscore allows users to define basic structural elements that are essential for composing a coherent and complete piece of music.

These basic musical structures come in both small and large-scale forms. Elements such as melodic and rhythmic fragments, for example, are by themselves small-scale structures. They can, however, be restated or combined to provide material for "local" changes in the music, such as a slightly-altered restatement of a theme, or contribute to the large-scale structure of a piece in the form of motivic development and transformation throughout an entire work.

Harmony is another musical construct that appears in both local and wide-ranging forms. In the simplest example, harmony can be a single chord without a reference point, without regard to what precedes or follows it. Defining transitions from one chord to another is the first step toward adding harmonic function. This can be as insignificant as an extension of a previous chord, or as far-reaching as a modulation to a foreign key.

A classic example of large-scale harmonic movement is illustrated in the well known sonata form (Fig. 1-1). Sonata form, also known as sonata-allegro form, was the standard first (and often final) movement structure for pieces such as symphonies, concertos, and sonatas in the eighteenth century. The basic structure is relatively simple, but there have been numerous permutations and elaborations of it up through the decline of tonality in the twentieth century [Ran78]. In essence, it is a tripartite form consisting of an *exposition*, followed by a *development*, and ending with a *recapitulation*, which is sometimes followed by a *coda*. The exposition consists of two main sections, the first which states the main theme in the tonic key, and the second section which introduces a contrasting second theme in the dominant key (or relative major in the case of a minor tonic). The development is the central portion of the piece, where there is thematic transformation accompanied by a lack of tonal stability which only reaches resolution in the tonic key at the recapitulation. The recapitulation follows a similar thematic format to the exposition, although the second theme group is restated in the tonic, not dominant key. The coda is a short section at the end which provides extra closing material.

What arguably makes the structure of sonata form compelling is the psychological feeling of building tension and final resolution. The elevation to the dominant key in the exposition is the first step toward heightening the feeling of tension, which is extended in the development, then resolved in the recapitulation.

This idea of tension and resolution is fundamental to music, and in essence is a superset or abstraction of musical constructs such as harmony and melody. It is

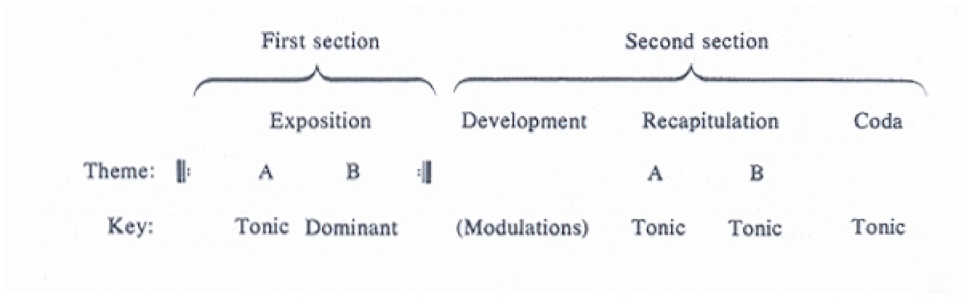


Figure 1-1: Diagram of sonata form [Ran78].

perhaps the most basic and essential (as well as broadly-defined) ingredient in all types of music, and works on both the small-scale and large-scale levels. It can be as simple as a dominant seventh chord resolving to a tonic, or as complex as a subtle combination of harmonic and rhythmic elements leading to the climatic moment of an hour-long symphony.

Another important element in music, although not necessarily structural in nature, is timbre. Roughly defined, timbre is the quality of a sound that makes it distinctive regardless of similarities in pitch, loudness, and duration.[1] In instrumental music, it can be affected by the manner of playing of an individual instrument or by the combined effect of multiple instruments. One example of the former case is the marking *sul ponticello* for stringed instruments, which indicates that the player should bow near the bridge, creating a glassy, brittle sound. An example of the latter is the art of orchestration, which is in part the interweaving of different instrumental timbres to create an overall effect.

All of the musical concepts discussed above have been considered and incorporated to some extent in the design of Hyperscore. While these aren't the only aspects of music that can be addressed, they provide a solid underpinning for the development of an interactive composition system.

1.2 The concepts behind Hyperscore

1.2.1 Musical direction

One reason for having a graphical notation system in the form of freehand drawing is to provide the user with an expressive means of shaping musical direction. Drawing a contour is a simple and intuitive way of depicting the temporal climaxes and denouement of a piece.

[1]I will not get into a discussion about the precise definition of timbre. I'll leave that for the psychoacousticians and audio engineers out there.

The initial idea of having a comprehensive, abstract representation of musical direction stemmed partly from the idea that a musical work is in many ways similar to a written story. The series of events which make up a story is analogous to the musical gestures (or events) within a piece. These musical gestures can be large or small and function as units of tension and resolution, just as plot elements in a story.

Gestures in turn consist of *lines*, which combine and overlap to form a whole. In Hyperscore, these lines are literally depicted by curves drawn by the user. They can either be used as contrapuntal elements that communicate a musical gesture when effectively interweaved, or as a single sweep indicating a larger musical motion. It is through this combination of creating a visual representation of the large-scale structure of a piece and the process of integrating musical lines that the user composes a piece of music in Hyperscore. This representation adequately conveys information regarding both the dramatic arc of the piece as a whole and the placement of individual motivic elements.

1.2.2 Musical context

It is important to mention that in today's world, the concept of "music" is somewhat vague; a sound sculpture consisting of non-discrete frequencies and exotic computer-generated timbres can be considered a piece of music as much as a simple, tonal Minuet. The current versions of Hyperscore have been designed with the intent of eventually producing music for traditional orchestral instruments. As a result, the musical concepts discussed in relation to Hyperscore center on discretely-pitched music. See Section 5.3 for a discussion about nontraditional and purely computer-based applications for Hyperscore.

1.3 Outline

Chapter 2 discusses related work in the field of computer music. An emphasis is made to provide background examples which are relevant to the theoretical and practical aspects of Hyperscore as well as to present an overview of work that has had a direct impact on the conception and development of Hyperscore.

Chapter 3 familiarizes the reader with Markov models, one of the algorithmic tools used in Hyperscore. A project called *Palestrina*, which was developed concurrently with Hyperscore and implements analytical and generative algorithms using Markov chains, is presented. The final two chapters detail the design and development of several versions of Hyperscore and its role in the Toy Symphony project.

Chapter 2

Background and related work

There has been a tremendous amount of work done in the field of computer music to date. The types of systems that have been developed are as varied as they are numerous. Even though only work relevant to Hyperscore will be discussed here, Curtis Road's *Computer Music Tutorial* and Joel Chadabe's *Electric Sound* are recommended for a more complete picture.

2.1 Classification

Roads divides the types of algorithmic composition systems into four categories. The following work includes examples from all of them:

1. Self-contained automated composition systems.
2. Command languages for generation and transformation of musical data and control of musical processes.
3. Extensions to traditional programming languages.
4. Graphical or textual environments designed for music (including music programming languages).

Computational methods used in systems that take some form of musical input can also be classified much the same way Robert Rowe classifies interactive computer "response methods." Although his categorization is used in the context of real-time systems, they are just as applicable in a non-real-time context [Row93]:

- Transformative methods take existing material and apply transformations which result in variants. These variants may or may not be recognizably related to the input.
- Generative algorithms take source material that can be elementary or fragmentary and use sets of rules to produce something new.

- Sequenced techniques use prerecorded or pre-composed material. The variations produced result from performance considerations (i.e. tempo, dynamics).

These categories are not intended to be rigid classification schemes. They are presented here to provide the reader with a framework for placing different types of algorithmic composition systems in context.

2.2 Brief historical survey of automated composition

Research in algorithmic composition was not merely a byproduct of developing technology, but also of the aesthetic trends of the twentieth century. Along with the decline in tonality, there was a marked tendency for formalized and systematic methods of composition in the 1950's. This emerged from the twelve-tone method originally developed by Arnold Schoenberg. Composers such as Olivier Messiaen and Anton Webern extended this method, also known as *serialism*, beyond tone rows to other criteria such as rhythm, timbre, and dynamics. As the complexity of the formalism increased, the amount of preparation required by the composer grew accordingly. Since computer programs sped up the time-consuming labor associated with systematic composition, they served as logical vehicles for furthering this aesthetic [Roa96, p.833].

2.2.1 The Illiac Suite

One cannot speak of the history of computer-generated music without mentioning Lejaren Hiller and Leonard Isaacson's *Illiac Suite for String Quartet.* The piece was written in 1956 using a generate-and-test method applied to a series of randomly generated numbers. Their premise was that the act of composing could be thought of as the extraction of order out of a chaotic environment. The *Illiac Suite* was the culmination of four different musical experiments. The first set of experiments consisted of generating cantus firmi and first-species counterpoint. The second objective of their work was to illustrate how computers might be utilized by contemporary composers. The music produced in this experiment consisted of random chromatic notes restricted by several simple compositional devices such as rhythm and voice-leading. The final set of experiments illustrated their objective of using "the computer as an experimental device to synthesize musical textures based upon mathematical models" which they believed abstracted certain essential elements of musical structure. The music produced in this last experiment was generated from zeroth and first-order Markov chains [HI58] (see Chapter 3 for more on Markov chains).

Hiller and Isaacson's experiments used both *stochastic* (also know as *probabilistic*) and *deterministic* methods of composition. In contrast to deterministic procedures, which guarantee a certain output given some input, stochastic procedures involve randomness. They generate musical events according to probability tables that weight the occurrence of certain events over others. Although tables can define an overall trend, the local details remain mostly unpredictable [Roa96, p.834].

2.2.2 Iannis Xenakis

Iannis Xenakis, another pioneer of automated composition, is especially well known for his work in stochastic music. He saw his methods as the solution to the problems inherent in serial music (what he termed the "fusion of the multimodality of Messiaen and the Viennese school"):

> Linear polyphony destroys itself by its very complexity; what one hears is in reality nothing but a mass of notes in various registers. The enormous complexity prevents the audience from following the intertwining of the lines and has as its macroscopic effect an irrational and fortuitous dispersion of sounds over the whole extent of the sonic spectrum. There is consequently a contradiction between the polyphonic linear system and the heard result, which is surface or mass. This contradiction inherent in polyphony will disappear when the independence of sounds is total [Xen71, p.8].

In the 1960's Xenakis wrote his Stochastic Music Program (SMP), employing formulas that were originally developed by scientists to describe the behavior of particles in gases. SMP uses these formulas to define clouds of sound characterized by the density of notes. The composer interacts with SMP by specifying the following attributes before running the program:

1. Average duration of sections
2. Minimum and maximum density of notes in a section
3. Classification of instruments into timbre classes
4. Distribution of timbre classes as a function of density
5. Probability of playing for each instrument in a timbre class
6. Longest duration playable by each instrument.

The output is meant to be transcribed for performance by traditional instruments and can be altered by the composer in whatever way best fits the musical goal. Xenakis himself had no qualms about modifying and rearranging computer output. Hiller, on the other hand, believed that computer-composed music should not be edited; if the results were unsatisfactory, the program should simply be run again. Xenakis thus favored genuine interaction between computer and composer, while Hiller preferred a consistently-formal musical aesthetic [Roa96, p.845].

2.2.3 The POD system

Unlike SMP, Barry Truax's POD(POisson Distribution) programs were not designed to produce output for acoustic instruments. The program was written in the early 1970's and uses Poisson distributions (a standard probability function) to create a two-dimensional frequency-time score. As the POD programs developed, Truax became increasingly aware of the powerful controls it provided:

> I could say, "I want five sounds per second in this frequency range," and I could do that without having to specify every event. I could suddenly create something from the top down. And this is one of the beauties of algorithmic composition. After all, if you get out only what you put in, why do it? Why bother with the computer if it just produces the specification that you've given it? [Cha97, p.283]

POD's synthesis capabilities, however, were limited in part because of the restrictions imposed by the computational speed of machines available at the time. Polyphonic scores could not be performed in real time, and had to be stored on magnetic tape for later playback [Tru85].

2.2.4 MUSICOMP

MUSICOMP was perhaps the original music software environment. It was developed by Robert Baker and Lejaren Hiller in 1963 and consisted of a library of assembly language routines. The library included functions such as selecting items according to probability distributions, randomly shuffling items, tone row manipulation, and coordination of rhythmic lines. The output of these functions could be printed or formatted for input to sound synthesis programs. Hiller as well as other composers used MUSICOMP to create a number of works for both traditional instruments and computer-generated sound [Roa96, p.848].

2.2.5 Experiments in Musical Intelligence

David Cope's *Experiments in Musical Intelligence* (EMI) is an expert system which successfully generates compositions in the styles of different composers. It uses a non-linear, top-down approach as well as an intricate rules-based system defined according to experience gained in Cope's many years of teaching music theory [Cop91].

Cope started developing EMI in 1981 as a result of "composer's block." The concept behind EMI was the idea that composers have certain musical *signatures*, or frequently-used patterns that define their style. Cope argues that "composers create music by mixing such signatures and using *recombinancy*, or the recombination of elements found in other of their works and in the music of other composers [Cop96]." Just as EMI can generate music in the style of Mozart or Bartok, Cope uses it to generate music his own style.

His most recent system, ALICE (ALgorithmically Integrated Composing Environment), is based on EMI but functions as an interactive composition assistant where input to the database is the user's own musical samples. It is designed to extend user-composed passages, develop "composer created germ ideas" and offer "new ideas when inspiration temporarily wanes." The program can generate anything from a single note to an entire section of music in the user's style [Cop00].

Cope's SPEAC system of musical identifiers (used in EMI and ALICE) is discussed in Section 4.5.2, as it has direct bearing on the generation of harmonic progressions in Hyperscore.

2.3 Graphical environments for composition

In the late 1960's, the advent of the first computer terminals opened up the possibility for graphical interfaces. Max Mathews and L. Rosler began using this new technology to explore graphical applications for music. Utilizing a new computer called the GraphicI, they developed a composition language which substituted graphical input in the place of tediously-punched data cards. The system allowed a musical score to be specified as a group of graphs and provided the means to algorithmically manipulate them by drawing curves on the computer screen (Figure 2-1).

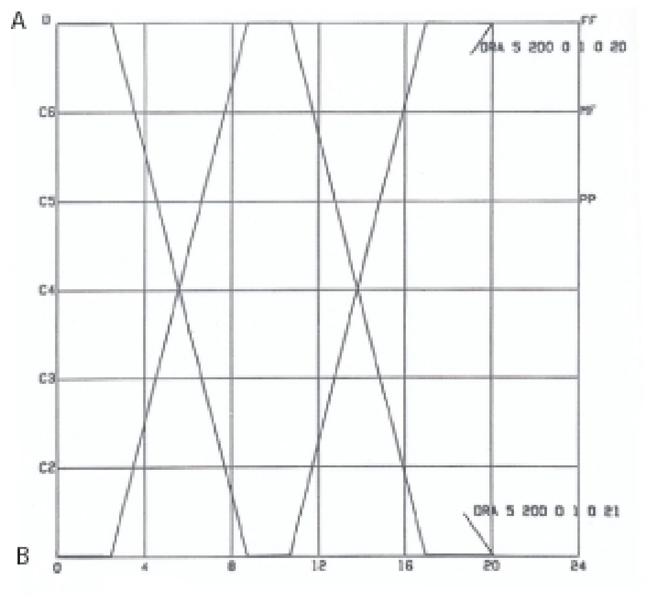


Figure 2-1: Graphical functions used to combine two melodies (letter labels added) [MR68]. Melody A is prominent at the beginning. It gradually loses influence until Melody B is completely dominant in the middle section. Melody A then regains dominance at the end.

Computer graphics have come a long way since the 1960's, and a whole spectrum of graphical composition tools has followed Mathews and Rosler's work. These tools range from notation programs, which are serve as substitutes for the precise work of human engravers, to programs which use abstract graphical forms as input parameters for the production of sound or music.

2.3.1 Common music notation

Common music notation (CMN) is the standard musical notation system which originated in seventeenth-century Europe and is still used today by many (if not most) composers and performers. It is mentioned here in brief because, strictly speaking, it is a graphical system used for composing and recording musical events. Roads notes that CMN has been bypassed by many composers for several reasons. First, it's biased toward the pitch and duration of notes. Second, it does not provide enough ways for describing timbre or for representing spatial elements. Finally, it only displays music at one level; there are no allowances for hierarchical descriptions.

The systems described in the following sections are very different from CMN programs. They use graphical objects as a means of abstracting musical structures and parameters in order to create composition environments that are either 1) more intuitive from a top-down approach, 2) educational, 3) aesthetically pleasing, or 4) fun.[1] Even though these examples lack the precision of CMN in certain respects, they offer a fresh outlook on both notation systems and automated composition.

2.3.2 UPIC

After Xenakis completed his orchestral work *Metastasis* in 1954, he set out to find a comprehensive way to translate graphical design into musical notation. His goal was "to establish—beyond traditional notation—a more universal way of expressing musical thought, readily interpretable by anyone [Loh86]." What eventually resulted was the UPIC (Unité Polyagogique Informatique de CEMAMu), a graphical sound synthesis system realized in the 1970's.

The first version of the system dates from 1977 and uses a large, high-resolution graphics tablet for input. Lower-level forms of input include waveforms and envelopes that are directly drawn onto the table or a set of points which are connected by the computer using interpolation. On the "macrocompositional" level, the Music Page function allows the user to draw a time-frequency score consisting of lines, curves and points (Figure 2-2).

More recent versions of UPIC are real-time systems which can serve as performance instruments. They allow the composer to control the time flow using a mouse, and even permit discontinuous jumps to other regions of the score [Roa96, pp.331-334].

[1]Hyperscore is intended to fit all four of these categories.

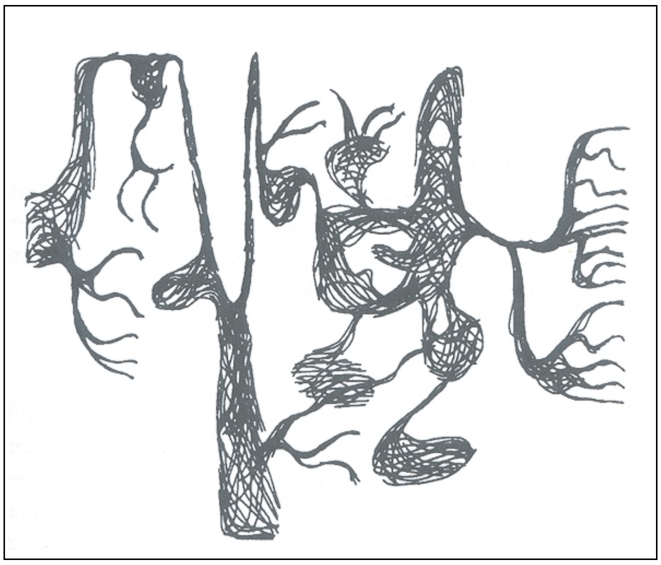


Figure 2-2: A page from Iannis Xenakis's *Mycenae-Alpha* (1980), created with a UPIC system. The horizontal-axis is mapped to time, the vertical-axis to frequency [Xen86].

2.3.3 PatchWork and OpenMusic

PatchWork and OpenMusic are two related projects that have recently emerged from Ircam (Institut de Recherche et Coordination Acoustique Musique). Over the past years, the term "computer-assisted composition" has taken a special meaning at Ircam: "As opposed to the generation and processing of audio signals by the means of DSP hardware or software technologies, computer-assisted composition systems focus on the formal structure of music." Thus the output of one the predecessors to PatchWork and OpenMusic was "no longer a set of set of signals, but rather a symbolic description of a musical score which could also be printed in common music notation. Specialized formal sublanguages were made available to composers which could then be used to define rhythmic patterns, harmonic progressions, polyphonic interrelations and so on [Gir99]."

In this sprit, the PatchWork project, begun early 1990's by M. Laurson, J. Duthen, and C. Rueda, was implemented to serve as a visual programming interface to LISP. Patchwork allows LISP functions to be graphically represented and manipulated as boxes called *patches* (Fig. 2-3a). The programming syntax consists of making connections between boxes. These visual elements can be positioned, moved, and edited. PatchWork is "musically neutral in the sense that it does not make assumptions about what kind of music or musical raw material is to be produced or analyzed with it. The main aim is to give the user basic tools for visual programming and to provide a straightforward correspondence with the base language, Common LISP."

OpenMusic, designed by G. Assayag and C. Agon in 1998, is a superset of PatchWork. It is an object-oriented environment which allows the composer to design sophisticated musical object classes. Patches are symbolized by icons which can be dragged, dropped, and interconnected to implement musical algorithms. One particularly powerful patch, the *maquette* (Fig. 2-3b), enables high-level control of musical material over time (visually represented by the horizontal axis). Objects that can be inserted into maquettes include MIDI files and sound files as well patches representing musical functions. Maquettes can also be embedded into other maquettes, making OpenMusic an effective tool for designing sophisticated hierarchical and temporal musical structures [Gir99].

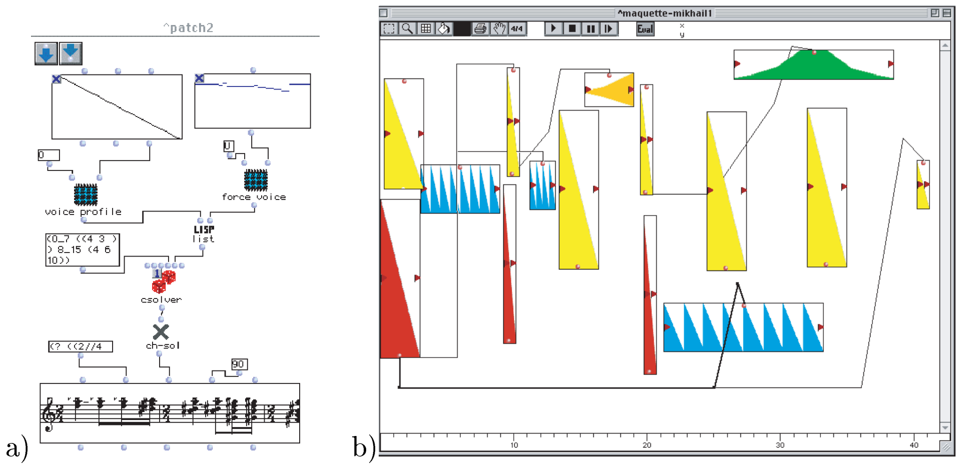


Figure 2-3: a) An OpenMusic patch which harmonizes the beginning of *Syrinx* by Debussy. b) A maquette created by composer Mikhail Malt. Each block's position corresponds to its point in time. The horizontal length corresponds to duration, and the vertical height to intensity. The lines linking blocks represent functional connections [Gir98].

PatchWork and OpenMusic, as well as all of the previously mentioned systems, are intended for use by professional composers or composer-programmers. For a musically or technically untrained user, however, most of them are difficult if not impossible to use effectively. The following sections will outline some graphical composition environments that, unlike the previous examples, are designed for a more general audience, with a particular focus on children.

2.3.4 MetaSynth

MetaSynth is a commercially-available sound synthesis application created by Eric Wenger. It runs on Macintosh machines and uses images as input.[2] Graphical tools are used to "paint" scores, in which the vertical axis represents pitch and the horizontal axis represents time (Fig. 2-4). Each pixel's gray-scale value (brightness) controls the volume level of an individual harmonic (black being silent). The color value is mapped to stereo placement, where red is left, yellow is center, and green is right [Wen01].

The numerous audio techniques integrated into MetaSynth include wavetable synthesis, frequency modulation, granular synthesis, and sampling. The "effects palette" also offers audio processes such as echo, reverb, parametric EQ, and chorus which can be changed and previewed in real time. The resulting "sonic sculptures," many of which have been made available on the web by MetaSynth users, range from repetitive techno to eerie sound-scapes.

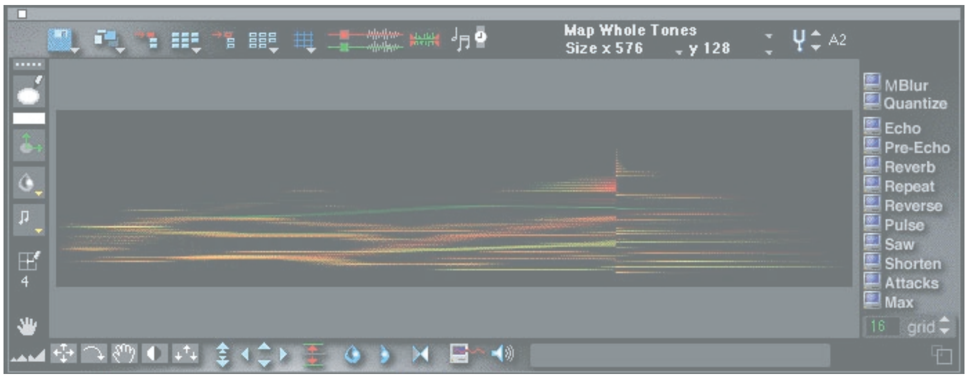


Figure 2-4: A partial screenshot of MetaSynth.

2.3.5 Music Insects and SimTunes

Music Insects (1992), originally designed by Toshio Iwai for the Exploratorium in San Francisco, is a visual environment consisting of animated insects which move and react to dots of color painted on the screen. There are a total of four different insects, each mapped to different instrumental timbres. Musical scales, sounds, and light patterns are triggered as they pass over the dots. The insects by default move in a straight line but can be redirected (left, right, reverse) by placing special blocks

[2]The latest version to be released as of August 2001 is 2.7.

in their paths [Iwa92]. From a personal perspective, Iwai designed Music Insects as a means to overcome his lack of conventional artistic training:

> I haven't mastered any musical instruments, and I can't paint or draw very well. But encountering the computer made me realize that I could express myself very well without the typical process of technical study and practice. I found that the computer awakened my dormant talents. And I knew that a lot of other people feel the same way [Bro97].

SimTunes (1996), the third and most advanced version of Music Insects, is a commercially-available application designed for ages 8 and up and developed in conjunction with the game-simulation company Maxis. Like its predecessor, it uses animated insects and a palette of paint blocks to create an interactive musical environment (Figure 2-5).

Although it is entertaining to watch the insects crawl around the screen and trigger patters, SimTunes is limited by the fact that it's much easier to construct something repetitive than something that evolves in an interesting way over time.

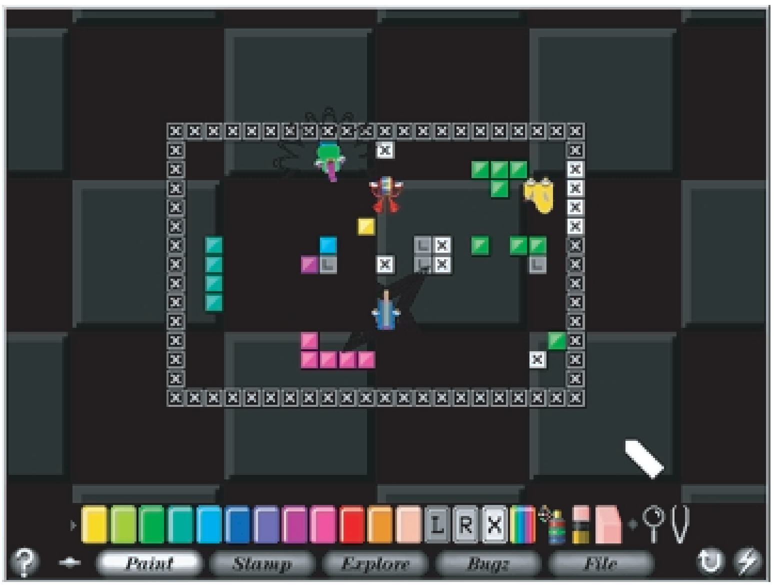


Figure 2-5: Screenshot of a basic SimTunes environment.

2.3.6 Creating Music

Morton Subotnik's *Creating Music* website (last updated 1999)[3] is designed to provide an environment for children to experience musical creativity with the same ease they are "able to enjoy with toys, drawing tools, building blocks, puppets, etc. In addition, special emphasis is placed on the written tradition [Sub99]."

The most composition-oriented of the various activities is called "Sketch Pad" (Figure 2-6). Freehand drawing ("finger painting") is used to sketch musical lines. Different colors correspond to different instruments. The drawings are interpreted as scores in which the horizontal axis is time and vertical axis is pitch. There are also special editing functions that invert, reverse, or duplicate notes in a selected area.

Other activities on the website include "Puzzles," a game where melodies are broken into four scrambled phrases and have to be reordered correctly, "Playing with Music," a conducting game that allows the participant to play with two basic qualities of music performance—tempo and dynamics, "Cartoon Conductor," which enables users to "perform" a scene by sliding the mouse around, triggering animation effects coupled with musical phrases, and finally, "Musical Contours," an activity which centers around the "directionality of melodic materials" or the idea that melodies have a general pitch contour: up, down, flat, or jagged.

The activities are very simple in general and do not offer an in-depth exploration of their respective musical aspects. However, they would probably appeal to younger children.

2.3.7 Impromptu

Impromptu is an example of an interactive teaching tool for both non-musicians and children. It accompanies Jeanne Bamberger's book, *Developing Musical Intuitions* (2000), a project-based tutorial for understanding structures inherent in music. The main graphical element in Impromptu is an icon called a *block*. There are three different types of blocks [Bam00b]:

- *Tuneblocks* consist of melodic segments (phrases, figures, and motives). They can be ordered sequentially or combined and encapsulated to form SuperBlocks.
- *Percussion Blocks* come in two forms: Drumblocks and Pattern Blocks. Both are played by percussion MIDI instruments selected by the user. Drumblocks play a single note and are labeled by the number representing its duration. Pattern Blocks are higher-level objects that contain complete rhythmic patterns.

[3]More advanced versions of the website activities are available on CD-ROM.

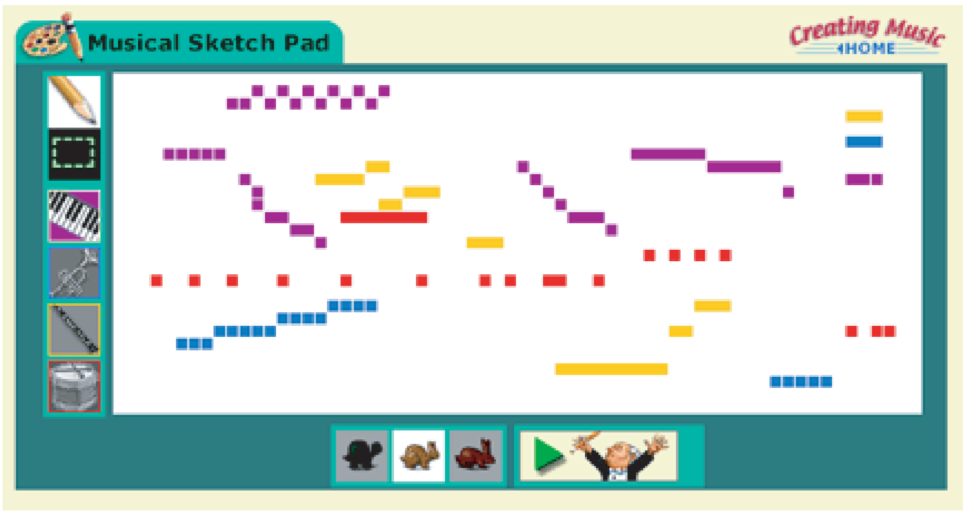


Figure 2-6: Screenshot of the web version of Musical Sketch Pad, an activity in Morton Subotnik's *Creating Music*.

- *Chord Blocks* are primarily used to harmonize melodies in real time and consist of three simultaneously played notes. Each chord is labeled by its function: I, IV, or V.

Impromptu comes with a large number of built-in projects (Figure 2-7) with preprogrammed blocks which are used in conjunction with the book. These projects are thoughtfully constructed and make Impromptu and effective hands-on tool for learning music.

2.4 Work at the MIT Media Lab

The Hyperinstruments group (also know as the Opera of the Future group), directed by Tod Machover, has been a major source of interactive music projects developed at the MIT Media Lab. It has provided the fertile and dynamic environment from which Hyperscore has emerged. With the exception of Golan Levin's *Audiovisual Environment Suite*, all of the work discussed in the remainder of this chapter has originated from this group.

2.4.1 Hyperinstruments

Tod Machover's research at the Media Lab initially focused on using technology to extend the capabilities of acoustic instruments in order to enhance the range of expression for virtuoso performers. The diverse projects encompassed by this goal

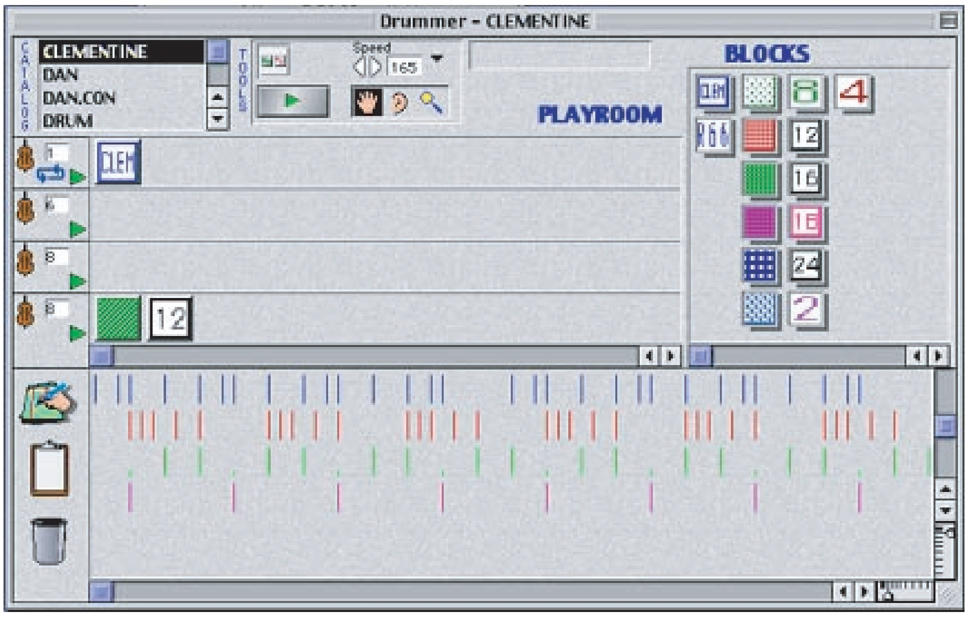


Figure 2-7: Screenshot of Impromptu (sample from CD-ROM).

became known as *hyperinstruments*. The basic idea entailed adding sensors on the instrument and performer to gather as much data as possible about the movements of the player and the musical output. These parameters included audio and MIDI data as well as physical aspects of the player's interaction with the instrument (for example, bow position relative to the bridge of a stringed instrument). A computer was used to analyze the data and extract higher-level musical information which was in turn used to control aspects of the sound being produced by both the computer and the instrument itself [Rig94].

Starting in 1992, Tod Machover and his group began to explore ways of extending the concept of hyperinstruments to include amateur users [Mac92]. Earlier projects included Alexander Rigopulos' *Seed Music* [Rig94], a real-time interactive system that allowed users to improvise by controlling high-level parameters in the music and *Digital Theremins* (1994-95) [Wax95], a series of interactive systems which facilitated music-making for amateurs by using electric-field sensing[4] to measure physical gestures. The most prominent project during this time period was the *Sensor Chair* (1994), a special chair built for the magicians Penn and Teller that used electric-field sensing to determine the hand position of a seated person (see Figure 2-8). Hand positions and gestures were then mapped to a variety of musical

[4]Developed by Neil Gershenfeld and Joseph Paradiso at the MIT Media Lab.

outputs in a piece written by Machover entitled *Penn's Sensor Solo.*

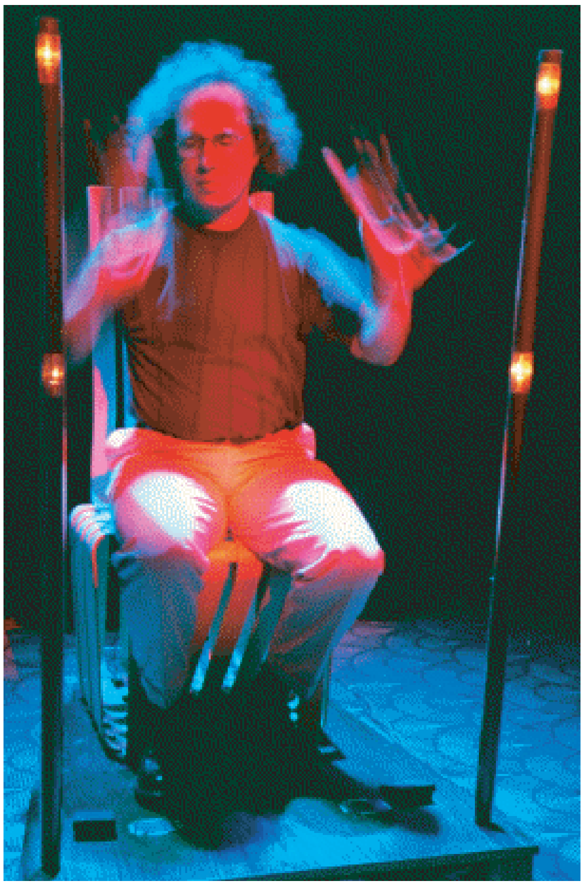


Figure 2-8: Tod Machover playing the Sensor Chair [Kne01].

2.4.2 The Brain Opera

This phase of research culminated with the *Brain Opera* (1996), an interactive music experience on a grand scale. The *Brain Opera* was inspired by Marvin Minsky's book *Society of Mind*, which theorizes that the human mind is a collection of many agencies working together to create a consciousness [Ric98]. The *Brain Opera* consisted of two main sections: a period of audience experimentation and a performance. As many as 200 participants entered a lobby space called the Mind Forest, where they experimented with different hyperinstruments. Among these were the Rhythm Tree, which used touch-sensitive pads to trigger samples, the Gesture Wall, which responded to physical movements with music and animated graphics, and the Singing Tree, which produced musical accompaniment and imagery in response to the user's singing voice. Following this period of exploration in the Mind Forest, three trained musicians—also playing hyperinstruments—performed the Brain Opera piece. The piece incorporated sound samples recorded by audience members in the Mind Forest as well as musical contributions submitted via the internet on the Brain Opera website, thus creating a unique performance every time [Mac96].

2.4.3 Animated graphics for sound and music

Stretchable Music

One of the projects which followed the Brain Opera was Pete Rice's *Stretchable Music* (1997), a system that uses graphical objects for manipulating pre-composed music in real time. The idea behind *Stretchable Music* is to engage passive music listeners in active music-making. Rice's philosophy advocates systems that are not "tools to enable a musically-deficient public," rather "the expression of musical ideas as compositions in an interactive form unique to the medium of computation." By using a mouse, the user can stretch and pull animated objects representing different layers of the music. At various times in the system's temporal framework (predetermined by the composer), animated icons pop up on the screen and offer the user an opportunity to play keyboard or drum solos. Users can navigate between four distinct musical sections by grabbing special advancement icons that appear at key points in the piece [Ric98].

Golan Levin's work

Golan Levin's *Audiovisual Environment Suite*(2000)[5] presents five new interfaces for real-time performance of dynamic visual imagery and sound. His systems are built around the metaphor of an "inexhaustible and dynamic audiovisual 'substance,' which is freely deposited and controlled by the user's gestures."

> Each instrument situates this substance in a context whose free-form structure inherits from the visual language of abstract painting and animation. The use of low-level synthesis techniques permits the sound and image to be tightly linked, commensurately malleable, and deeply plastic [Lev00a].

The results are vivid and dynamic (Figure 2-9). Levin's creative combination of synthesis techniques (which include frequency modulation and granular synthesis) generate accompanying sound which evolves beautifully and corresponds appropriately to the animation.

[5]The author's minor contribution to this work involved the implementation of the low-level real-time audio code.

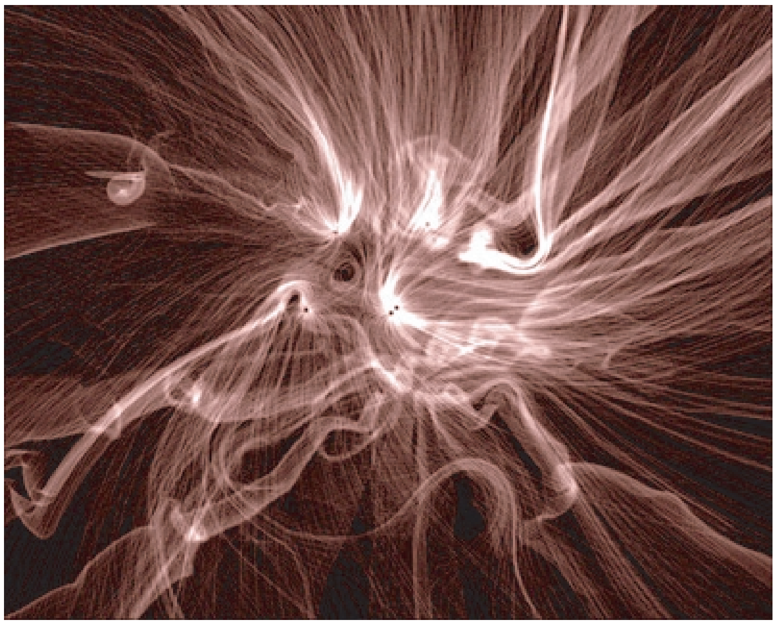


Figure 2-9: *Floo*, one of the instruments in Golan Levin's *Audiovisual Environment Suite* [Lev00a].

2.5 Projects preceding Hyperscore

The last two projects describe the author's previous work in the Hyperinstruments group. They are mentioned in brief to provide a summary of the work accomplished in the year preceding the implementation of Hyperscore. *Palestrina*, a project developed concurrently with Hyperscore, is detailed in the next chapter.

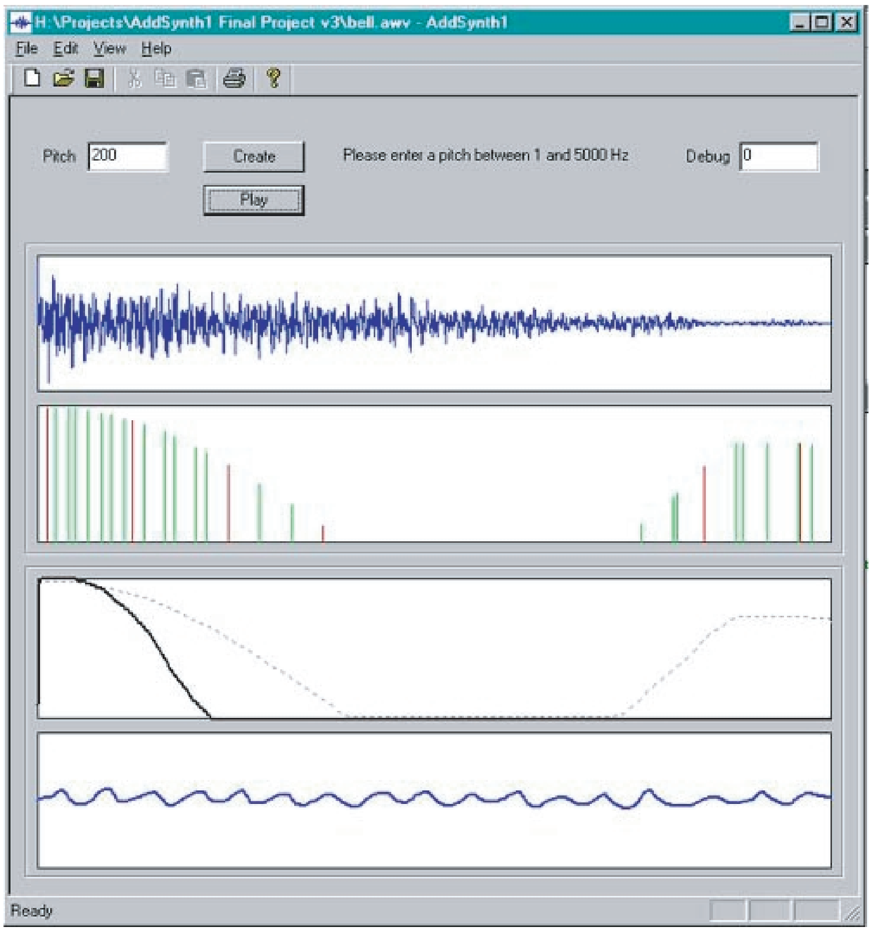
2.5.1 AddSynth

AddSynth (1999)[6] is designed to provide a fun and simple graphical interface for exploring timbre using additive synthesis. The user starts by specifying a base frequency value (between the range of 1 and 5000Hz). The computer then generates a waveform that consists of the fundamental frequency and all of its harmonics up to 5000Hz, with initially identical amplitudes.

There are four display windows (see Figure 2-10), all of which can be used to edit different aspects of the sound:

- *Amplitude:* displays the waveform in the time domain. The user draws a curve

[6]A joint project with Tristan Jehan. Implemented in Visual C++ for Windows NT by the author.



Pitch 200

Please enter a pitch between 1 and 5000 Hz

Debug 0

Figure 2-10: AddSynth. Window 1: The waveform. Window 2: The initial spectrum in the frequency domain. Window 3: The final frequency spectrum. Window 4: Pitch shift curve.

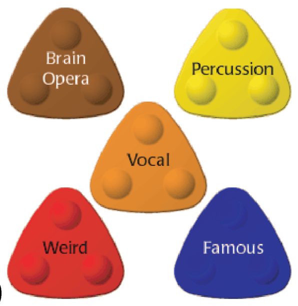
which shapes the amplitude (loudness) envelope of the sound.

- *Frequency spectrum I:* allows the user to increase or decrease the amplitude of harmonics displayed. Right-clicking in the window creates non-harmonic frequencies (colored green). This display represents the *inital* frequency spectrum of the sound.
- *Frequency spectrum II:* specifies the envelope for the *final* frequency spectrum (the black line). The dotted gray line is an outline of the spectrum envelope formed by the window above.
- *Pitch shift:* allows the user to draw a curve which affects pitch. The total vertical range of the window is approximately a minor second.

AddSynth creates the sound by adding the sine waves corresponding to each frequency then normalizing. The values in each sine function are altered to take into account the curves drawn in the windows. The resulting sound morphs from the first frequency spectrum to the second.

2.5.2 The Future Music Blender

Machover's *Brain Opera* found a permanent home in June 2000 at Haus der Musik in Vienna. In its transformation from touring show to museum installation, a completely new part called the Future Music Blender was designed in lieu of a live performance [Tod00].

a)


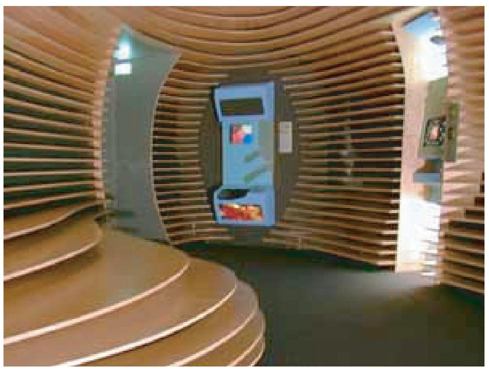
b)


Figure 2-11: a) Five colored chips representing five different categories of sounds. b) A view of the Future Music Blender room with the Blender station in the center and a sound exploration station to the right [Kne01].

As with the original *Brain Opera*, people can explore the Mind Forest and record their own sounds. These sounds are automatically submitted to a central database

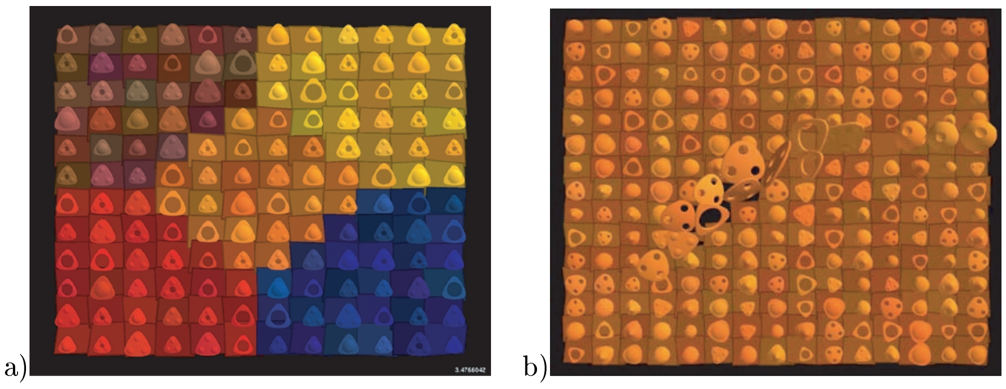
a) b)


Figure 2-12: a) The Main Movement grid consisting of sounds in all categories. b) The Vocal Movement grid as sound samples are being triggered [Kne01].

that contains a large number of preexisting samples. All of the sounds, both user-created and pre-recorded, are represented by ID tags called *chips* which come in different shapes and colors depending on the characteristics of the sound (see Figure 2-11). The sounds recorded by visitors are stored as in the vocal category (represented by orange chips).

The Future Music Blender room, adjacent to the Mind Forest, is littered with thousands of chips of varying colors, each mapped to a particular sample from the database. Visitors can listen to and modify sounds by picking up a chip and inserting it into one of the several sound exploration stations. The changes are simple but clear and effective, and each one causes a corresponding change to the graphical model of the chip which is shown on the station's display.

When satisfied with the sound, the participant can deposit the chip at the *Blender*. This sound then gets added to an active performance database which is graphically displayed as a large projected grid of animated shapes (Fig. 2-12). The participant can sit in a specially-designed Sensor Chair (Fig. 2-8)[7] and "perform" the active sounds by using hand gestures. Every time a sample from the active performance grid is played, its corresponding graphical shape pops out of the display.

There are six "movements" or grid types users can explore. The Main Movement consists of samples from all five categories. Initially, there is an equal distribution of sound types (see Figure 2-12a). When a participant inserts a new chip into the Blender, the corresponding sample is placed in the region with other sounds of its type, thus allowing color/sound regions to grow organically.

If the person in the chair is "favoring" a sound category (for example, play-

[7]This is the same interface used for the Penn and Teller project (see Section 2.4.1).

ing mostly percussion sounds), then a new movement is initiated. There are five "special" movements for the five different categories, each consisting solely of the highlighted sound type (Figure 2-12b). Sounds of "foreign" types deposited in the Blender during these movements are added to the corners of the grid. The special movement eventually returns to the main movement. The amount of time the special movement lasts depends on the activity of the performer in the chair (if very active, the movement continues for a longer time period). Elements such as velocity of motion and three-dimensional position influence the duration and spacialization as well as the attack envelopes of individual samples. Each movement has its own special mappings and quantization values for sample triggering and, in the case of the Brain Opera movement, musical accompaniment (the main Brain Opera theme).

By experiencing the various aspects of the Future Music Blender, visitors can produce constantly evolving sound sculptures that are the direct result of multiple audience choices and contributions. In short, it is a fitting culmination to the Brain Opera's original musical philosophy.

Implementation

The graphics and chip interaction code is written in Python and Java by Brian Knep, and the sound and Sensor Chair code (encapsulated in a DLL) is implemented in C++ by the author. A dual-processor machine running Windows NT serves as the central hub for the blender network. Input from the Sensor Chair determines the musical output which in response calls the graphics routines in the main Java program. The Java and C++ code communicate through a special SoundScape object.

The chair sends a stream of sensor data in the form of MIDI control change messages.[8] The values for the four chair sensors are processed using a linear least-squares fit to estimate the three-dimensional hand position of the user. The coefficients for the equations are determined in advance by a calibration program, also written in C++, which requires as input nine specific hand positions within the perimeter of the four sensors.

Once the hand position is determined, the chair code updates the current gesture information by adding the new data values to the previous values. Data spikes are eliminated by using a simple low-pass filter. Given the updated gesture information, the sound code determines how the musical output is affected. These include everything from triggering samples to switching movements. Movement objects are used to keep track of the many variables governing the current state of the system.

Grid objects, also part of the sound code, are used to maintain the current performance array of sound samples. The Java code receives messages from tag readers (embedded inside the Blender and sound exploration stations) when a new chip has been inserted in the Blender or when a chip's sound has been altered at

[8]For more information on the chair hardware, see [PG97].

an exploration station. The Grid object in turn updates the sample array and calls the appropriate SoundScape functions which update the graphical display.

Afterwards

The Future Music Blender was a remarkable experience in many ways; it was both a dynamic project with a terrific venue and a learning experience. Most of all, it was rewarding to see how visitors enjoyed exploring the many creative facets of the exhibit.

It was also served as an appropriate precursor to Hyperscore. The initial concept of musical gestures came from the Sensor Chair's literal mapping of physical gesture to music. Furthermore, the programmer in a sense became the composer behind the composer. Although the person sitting in the chair did have the decision-making power to choose from any of the sounds available, (I) the programmer was the one who had to make the initial decisions about how the sound grids were arranged and how the gestures were mapped to effects. Likewise, Hyperscore's implementation provided an opportunity to make critical decisions about the musical role of the computer. All this is to say that both projects afforded the pleasure of an interesting and creative implementation process, not just an enjoyable result.

Chapter 3

Markov Models and Palestrina

The beauty and descriptive power of probabilistic network architectures and their graphical representation have been widely appreciated in the machine-learning community[Jor98]. Graphical models integrate essentially all known machine learning, function approximation, and prediction tools in a single framework. Hyperscore and *Palestrina* use Markov chains, a subclass of graphical models to provide the necessary structure for representing the compositional process of generating harmony and counterpoint. Markov chains also offer the flexibility of integrating deterministic and probabilistic rules in the same model and can be trained on experimental data.

Markov chains are probabilistic systems where the likelihood of a future event or *state* is determined by the event or events that immediately precede it. The probabilities in a Markov chain can be laid out in a table called a *state-transition matrix* (see Tables 3.1 and 3.2). The *order* of a Markov chain represents how many previous states need to be considered for each transition to a new state. States in a *zeroth-order* chain are independent and do not take into account any previous states; states in a *first-order* chain are determined only by the immediate predessor; events in a *second-order* chain are determined by the two preceding states, and so on [Roa96, p.878].

Palestrina is a program that generates and analyzes first-species counterpoint. Palestrina-style counterpoint was chosen as the rule-base for this experiment because of its simple and well-defined structure. The results of this project show 1) how Markov chains can adequately capture the rules of species counterpoint 2) how species counterpoint can be synthesized given a cantus firmus and 3) how such rules can be inferred from musical examples.[1]

[1]Material in this chapter will appear in [FS01].

3.1 Historical background of species counterpoint

Sixteenth-century counterpoint, with which the name of Palestrina has almost become synonymous, has long been held by musicians and theoreticians as an exceptionally elegant form of composition that has both musical and pedagogical value. Modeled after this style, species counterpoint was first introduced by J. J. Fux in his 1725 treatise, *Gradus ad Parnassum.* Fux codified the study of Palestrina-style counterpoint by presenting five categories of instruction, called species. Each of these categories specified different rhythmic interactions: whole notes against whole notes (first species), half notes against whole notes (second species), quarter notes against whole notes (third species), half notes tied over the bar line against whole notes (fourth species), and a mixture of rhythmic values against whole notes (fifth species).

Gradus was a standard counterpoint text studied by countless musicians during the eighteenth and nineteenth centuries. However, from a stylistic viewpoint, it did not present an adequate approximation of Palestrina's music [SS89]. This goal was eventually realized by Knud Jeppesen, who believed that counterpoint must be studied with as much correspondence between written exercises and composition as possible [Jep31]. In other words, counterpoint had to be learned within the context of a specific musical style. In essence Jeppensen's presentation raised the practice of species counterpoint from mere "academic ritual" to a theory of composition [SS89] Regardless of whether one agrees or disagrees with this pedagogical view, the fact that Jeppesen did successfully imitate the Palestrinian idiom makes his formulations ideal for this project. His deep knowledge and understanding of Palestrina's style provide invaluable insight into the theory behind the practice.

3.2 Prior work in automated species counterpoint

3.2.1 ILLIAC revisited

As mentioned in the previous chapter, Hiller and Isaacson's experiments with the ILLIAC included routines that used some basic counterpoint rules to generate four-part first species counterpoint given a cantus firmus. Their objective, however, was not to compose great counterpoint—it was to "demonstrate that technical musical concepts could be translated into computer language to produce musical output [HI58]." Since their initial experiments in the 1950's, there have been a number of other people who have implemented systems that generate species counterpoint.

3.2.2 Kemal Ebcioğlu (1980)

Kemal Ebcioğlu created a rule-based system to generate fifth-species counterpoint given a cantus firmus that encapsulates each rule in a simple LISP function [Kem80]. An enumeration algorithm is used to generate as many solutions as possible given a time limit. The enumeration algorithm uses additional rules to output the more "musical" solutions first (added due to the fact that solutions blindly following the rules were deemed "grossly insufficient"). These include rules for producing better musical contours such as the following:

> If the melody skips, and if the notes within the scope of this skip have not already been sounded, then they must eventually be sounded before the end of the melody.

The system, in the author's own words, seems to output solutions which "compare quite well to that of an average conservatory student."

3.2.3 David Lewin (1983)

David Lewin has written a program that generates first-species counterpoint using his own "Global Rule" which resembles Ebcioğlu's musical countour rule [Lew83]:

> For every note X of the counterpoint line lying above (below) the cadence tone, some note lying one step lower (higher) than X must appear in the line at some point subsequent to X.

His program generates the counterpoint backwards from the cadence note in order to more easily implement the Global Rule. Although this method is more computationally efficient, it is, as argued by Robert Gjerdingen, an unmusical approach since it treats the counterpoint as a solution to a problem rather than an "aesthetic utterance."

3.2.4 Robert Gjerdingen (1988)

Gjerdingen approaches the problem from a musician's point of view, as opposed to a programmer's [Gje88]. He remarks that theories of music often "view the concrete only as the end product of abstract operations and transformations."

> Melodies are thought to result from ornamenting or prolonging triads, triads from prolonging higher-level harmonic functions, and those functions from prolonging a central tonality. Yet what if concrete, stylistically specific melodies are the very stuff of music? What if musical knowledge and experience begins and is rooted in the concrete?

His system, PRAENESTE, is based on the idea that, just like a sixteenth-century singer improvising on a cantus, the computer should work forward in time without the benefit of being able to back up and start over. In response to each new contrapuntal situation, PRAENESTE uses a small number of concrete musical schemata to provide for itself a selection of correct melodic patterns. Given the preceding material, it selects a single melodic path that best satisfies a number of the higher-level aesthetic constraints. For PRAENESTE, being correct is a matter of course; its main concern is with style (as described by Jeppesen). PRAENESTE is quite successful, and the results resemble Jeppesen's examples (especially those of the first three species) remarkably. In the case of fifth species, however, the musical differences are still considerable.

3.2.5 William Schottstaedt (1989)

William Schottstaedt implemented all five species with up to six voices [Sch89]. His program follows Fux's guidelines and rules as closely as possible. These rules were extended and modified until the results acceptably resembled Fux's examples. Since most of the rules are not absolute, a penalty system is used to assign them relative degrees of importance. If at any given time the total penalty exceeds a certain amount, that particular path is abandoned. For example, an absolute rule such as the restriction of parallel fifths has a infinite penalty. A "very bad infraction" like direct motion to an octave has a penalty of 200, while a stylistic infraction such as a repeated pattern of three notes only has 4. The penalty values arise from both Fux's commentary as well as experience running the program.

The search time is optimized by taking the branch of the search tree which has the least penalty. If the path fails, then the program backs up and tries the next best branch. Unfortunately, the first solution found may not be very good, in which case a new search is made with a different starting interval in order to maximize the chances of finding a truly new solution. Even with this approach, the compute-time for fifth species solutions still remains high. Schottstaedt also admits that the program has no provision for starting a melody with a rest. Neither does it favor invertible counterpoint and imitation. It tends to "let voices get entangled in each other" and make "inadequate judgments about overall melodic shapes."

The approach used in *Palestrina* is considerably different from the cited work, because it (a) uses a probabilistic description of the problem and (b) infers its model from data which ensures that the result is musical assuming that the training examples themselves are musical.

3.3 Synthesizing the counterpoint

3.3.1 Dynamic Programming

Palestrina uses a forward dynamic programming approach to find the most likely solution out of a multitude of possible solutions given a cantus firmus [Ber95].

The first step is the construction of a state trellis diagram where the states describe the possible chromatic notes in the counterpoint (Fig. 3-1). The counterpoint is constrained to be above or below the cantus and the permissible note range spans

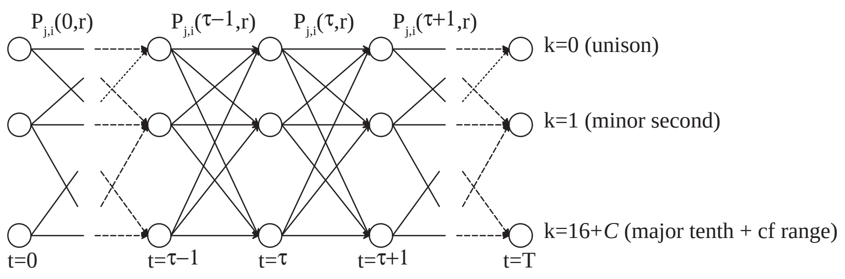


Figure 3-1: Trellis state diagram. The vertical states n correspond to note intervals relative to the cantus note at time t. The time t corresponds to time steps going forward. $P_{j,i}(t,r)$ denotes the probability of transition from state i at time t to state j at time $t+1$ with respect to rule r.

a major tenth above (or below) the highest (or lowest) cantus note. Hence the number of states in the trellis equals the chromatic range of the cantus plus 16 (a major tenth is equivalent to 16 minor seconds). Musical constraints on the counterpoint are then formalized by a mix of transition probabilities and state probabilities conditioned on the cantus. For example, the unconditional likelihood of a harmonic interval and the probability of a melodic interval given the previous melodic interval are specified.

While most rules can be realized based on a first-order Markov chain, some require a second-order system. A basic first-order structure has been implemented, but the model also allows for second-order considerations by looking one step ahead and choosing optimal solutions with respect to the next time step.

In the synthesis application, the most likely path through the trellis structure given the underlying cantus and transition rules is calculated. Evaluating every possible path would be computationally prohibitive. However, the problem is made tractable by using a dynamic programming approach commonly known as the Viterbi algorithm, which searches through the tree iteratively. The Viterbi algorithm is based on the insight that the most likely past state sequence leading into any one state is independent from future states.

Going forward in the chain, previous path probabilities are multiplied with the transition probability of the new update, and the path that generated the highest joint probability is stored. At any time step τ, the only considerations are the path leading into state $\tau - 1$, the probability associated with that sequence, and the transition probabilities at time τ. After the last transition occurs, the sequence with the highest probability is selected as the solution [Ber95].

3.3.2 Probabilistic counterpoint rules

Each rule is implemented as a probability table where illegal transitions are described by probability zero. The transition probabilities for generating a counterpoint line are obtained by multiplying the individual values from each table, assuming the

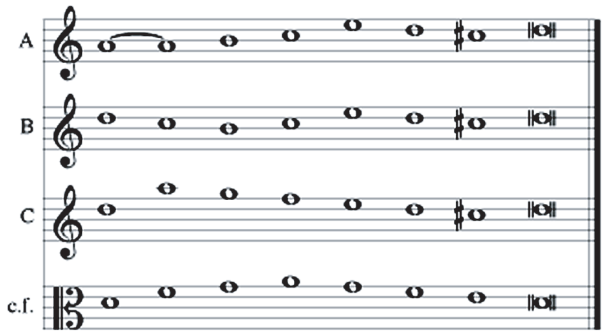


Figure 3-2: Counterpoint examples for a given cantus firmus. Example A was composed by a human (David Lewin). Example B was machine generated retaining the same climax point. Example C was machine generated with a different climax point.

rules are independent:

Table 3.1: Probability of harmonic intervals between cantus and counterpoint for notes other than the first and last. *Top row:* Estimated from human-generated counterpoint (12 examples). *Middle row:* Edited probability table used in the generating program. *Bottom row:* Estimated from machine-generated counterpoint (44 examples).

	P1	m2	M2	m3	M3	P4	d5	P5	m6	M6	m7	M7	P8	m9	M9	m10	M10
Human counterp.	0	0	0	.133	.133	0	0	.060	.181	.349	0	0	.072	0	0	.145	.072
Edited table	0	0	0	.1428	.1428	0	0	.1428	.1428	.1428	0	0	.0004	0	0	.1428	.1428
Machine counterp.	0	0	0	.100	.104	0	0	.054	.118	.409	0	0	.133	0	0	.172	.065

- The **harmonic table** determines whether the interval between the counterpoint note and its respective cantus note at any time τ is permissible. The likely intervals—third, sixth, tenth—are assigned higher probability values, whereas the forbidden intervals are assigned zero (Table 3.1). For the first cantus note, only perfect intervals are permitted, and for the last, only unisons and octaves.
- The **melodic table** describes the likelihood of a melodic interval created by two consecutive counterpoint notes being followed by any other melodic interval (Table 3.2). This table not only implements the rules determining legal melodic intervals and melodic patterns, but also has a profound effect on the shape of the line as a whole, since stepwise motion (among other such desirable behavior) is heavily weighted.
- The **cadence table**, in conjunction with the chromatic table, ensures that there is an appropriate cadence at the end of each example. Only stepwise motion is permitted to the final note, and stepwise motion to the penultimate note is given very high probability.
- The **chromatic table** gives very low weight to chromatically-altered notes (except the penultimate)—low enough that they will not be used unless there are no other

Table 3.2: Melodic transition rules. The first column indicates the previous melodic interval in the counterpoint line. The top row indicates the next melodic interval. The entries indicate the probability of each transition. The top table shows the probabilities used in the generating program. The bottom table shows probabilities estimated from the machine-generated counterpoint (44 examples).

Table edited for use in the generating program

	m2u	M2u	m3u	M3u	P4u	P5u	m6U	P8u	m2d	M2d	m3d	M3d	P4d	P5d	P8d	P1
m2u	0	.45	.2	.2	0	0	0	0	.035	.035	.025	.025	.01	.01	.009	.001
M2u	.45	.45	.03	.03	0	0	0	0	.01	.01	.005	.005	.004	.004	.002	.001
m3u	.45	.45	.03	.03	0	0	0	0	.01	.01	.005	.005	.004	.004	.002	.001
M3u	.35	.35	.05	.05	0	0	0	0	.05	.05	.025	.025	.025	.025	.024	.001
P4u	.065	.065	0	0	0	0	0	0	.4	.4	.02	.02	.01	.01	.009	.001
P5u	0	0	0	0	0	0	0	0	.52	.52	.01	.01	.01	.01	.009	.001
m6u	0	0	0	0	0	0	0	0	.51	.51	.025	.025	.01	.01	.009	.001
P8u	0	0	0	0	0	0	0	0	.51	.51	.025	.025	.01	.01	.009	.001
m2d	.05	.05	.005	.005	.002	.002	.002	.002	0	.467	.2	.2	.015	0	0	.001
M2d	.05	.05	.005	.005	.002	.002	.002	.002	.367	.367	.1	.1	.015	0	0	.001
m3d	.366	.366	.005	.005	.002	.002	.002	.002	.05	.05	.1	.1	0	0	0	.001
M3d	.366	.366	.005	.005	.002	.002	.002	.002	.05	.05	.1	.1	0	0	0	.001
P4d	.53	.53	.02	.02	.02	.02	.039	.02	0	0	0	0	0	0	0	.001
P5d	.454	.454	.03	.03	.01	.005	.005	.011	0	0	0	0	0	0	0	.001
P8d	.454	.454	.03	.03	.01	.005	.005	.011	0	0	0	0	0	0	0	.001
P1	.1	.1	.08	.08	.05	.05	.04	.03	.1	.1	.08	.08	.05	.04	.02	0

Table estimated from computer generated examples

	m2u	M2u	m3u	M3u	P4u	P5u	m6U	P8u	m2d	M2d	m3d	M3d	P4d	P5d	P8d	P1
m2u	0	.280	0	.080	0	0	0	0	.440	0	.160	0	0	0	0	.040
M2u	.088	.235	.029	0	0	0	0	0	0	.412	.029	.059	.029	.059	0	.059
m3u	0	.227	0	.045	0	0	0	0	.682	0	0	0	0	0	0	.045
M3u	.154	0	.077	0	0	0	0	0	0	.538	0	.077	.077	.077	0	0
P4u	.048	.238	0	0	0	0	0	0	.333	.286	.095	0	0	0	0	0
P5u	0	0	0	0	0	0	0	0	0	.833	0	.167	0	0	0	0
m6u	0	0	0	0	0	0	0	0	.500	0	.250	0	0	.250	0	0
P8u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m2d	.581	0	.054	0	.027	0	.014	0	0	.189	0	.041	.041	0	0	.054
M2d	0	.070	0	.085	.099	0	.028	0	.423	.169	.113	0	0	0	0	.014
m3d	.235	.353	.176	0	0	.059	0	0	.059	.118	0	0	0	0	0	0
M3d	0	.143	0	0	.143	0	0	0	.429	0	.143	0	0	0	0	.143
P4d	0	0	.875	0	0	.125	0	0	0	0	0	0	0	0	0	0
P5d	0	.200	.200	.400	.200	0	0	0	0	0	0	0	0	0	0	0
P8d	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P1	.133	.133	.067	.067	.200	.067	0	0	.200	.133	0	0	0	0	0	0

solutions. For the penultimate note, only chromatically-altered notes, with the exception of Eb, are permitted, in order to provide a leading tone (the E-based Phyrgian mode does not have a raised leading tone).

- The **good parallel motion table** prevents too many consecutive parallel thirds, sixths, and tenths. The probability that three in a row can occur is small, and the probability that four or more can occur is minute.
- The **approaching motion table** prohibits approaching a perfect interval in direct motion (also known as "hidden" fifths or octaves).
- The **departing motion table** prohibits leaving a unison in direct motion. This also has the added advantage of preventing the overlap of voices.
- The **general motion table** assigns contrary motion high probability, oblique and direct motion low probability.
- The **leap and climax tables** return probabilities of zero or one. The leap table returns zero if both the cantus and the counterpoint are leaping simultaneously in the

same direction and have done so previously (only one such occurrence is permitted). The climax table determines whether a pitch is acceptable given restrictions set by the chosen climax position of the counterpoint.

3.4 Analysis of counterpoint

Palestrina can infer a probabilistic description of counterpoint rules from existing compositions. For the purpose of experimentation, a set of first-species counterpoint strictly conforming to the rules were composed. Twelve examples were composed by humans and 44 examples were generated by a machine using the algorithm described above. This database of species-counterpoint examples was used as training data for a Markov model. The framework was set up for a state sequence identical to the last section. The transition tables were then inferred by counting the number of occurrences of certain transitions and renormalizing. Tables 3.1 and 3.2 compare tables which were used in the generating algorithm with those that were inferred from human-generated examples and machine-generated examples.

3.5 Experimental results

Much of the experimentation centered around deciding appropriate transition probabilities for the rule tables. While it was easy to make transitions legal or illegal, based on the outlined rules, it was more interesting and time-consuming to weight the probabilities properly in order to get musical results. Subtle changes in the table values resulted in very different solutions. Also probabilities had to be weighted not just with respect to other values in the same table but with respect to competing rules. For example, one possible situation where there are two acceptable solutions might be the following: (1) the penultimate note is approached in stepwise motion (ideal) but four consecutive parallel sixths result (acceptable, but not ideal) vs. (2) the penultimate note is not approached in stepwise motion (acceptable, but not ideal) but there are fewer consecutive parallel sixths. This is just one example of an aesthetic decision which can be reflected in the probability values.

During the course of implementation, it became apparent that some of the earlier tables were unnecessary. For example, there was initially a table which restricted parallel fifths and octaves. This was later rendered superfluous by the *approaching motion* table listed above (parallel motion is a special case of direct motion).

Another influential aesthetic factor was climax placement. As recommended by Jeppesen, the climax note should be unique as well as higher in pitch than all other notes. The program either tries each possible climax time until a satisfactory (if any) solution is reached, or the user specifies where the climax point should be. This rule helps produce surprisingly musical results.

3.6 Conclusions and Future Work

Palestrina shows effectively that Markov chains can be used to compose a first-species counterpoint line given a cantus firmus. Not only does the composed line comply with the strict rules of sixteenth-century counterpoint, but the results are also musical and comparable to those created by a knowledgeable musician. The program was capable of finding solutions identical to those presented by Jeppesen as well as some of the student compositions by the author.[2] It furthermore shows how some of the more complex transition tables can be estimated from given counterpoint examples and that these probability tables are close to the data available for estimation.

The results produced by *Palestrina* are also potentially useful for incorporation into future versions of Hyperscore. Markov chains like the melodic table (3.2) adequately describe the contour of a musical line in a very compact manner while providing a simple format for making both large and subtle changes. Such a structure would be ideal for creating melodic lines based on abstract descriptions such as "smooth" (as in the case of *Palestrina*) or "jagged." A separate table for each quality could be created by either hand-coding values or generating values by analyzing appropriate source material.

The ultimate goal of this work is to infer rules from authentic Palestrina works and to compose counterpoint based on the inference model. *Palestrina* also includes a module that parses data in MIDI files and converts them into a convenient format for analysis in Matlab.[3] A MIDI database consisting of 30 three-part movements from Palestrina masses has been created, yet the evaluation of the data must be left for future work.

[2]...graded A!

[3]The program itself was implemented in C++, not in Matlab.

Chapter 4

The Hyperscore application

4.1 Inception

Hyperscore was initially conceived as the software component to a large physical structure designed by Margaret Orth called the "Big Thing"[1] (see Figure 4-1). The Big Thing was intended to be a gigantic construction kit[2] used by children to compose and perform their own music. A collection smaller objects called *chunks* and *operator poles* were attached to it. Each chunk contained a musical fragment (rhythmic, melodic, harmonic) and could be updated or completely altered at a separate station called the *Poking Area* before being plugged back into the main structure. Operator poles, as the name suggests, were slender poles on which chunks could be stacked. They determined musical functions and transformations between the chunks. Software was intended to analyze the configuration of chunks and operator poles on the Big Thing and generate appropriate music.

The project was never completed as originally conceived for several reasons. First, the physical structure was almost completely designed before any decisions regarding the music were made. Although it was possible to implement something that would produce music or sound given Big Thing input, it was difficult to come up with something entirely new and musically creative given the restrictions imposed by the physical form. Furthermore the Big Thing was designed as a performance instrument which children could use to tweak their compositions in real time (e.g. modify playback volume or timbre by pulling or bending parts of the physical structure). The confusion resulting from an attempt to make a composition and performance tool out of a physical structure caused many roadblocks in the development of the project. In the end, some software was written and a hardware prototype built for the Poking Area, but the project never got beyond this initial stage.

The Big Thing was originally intended to be a part of Tod Machover's Toy Symphony, a (still ongoing) project designed to enable children, virtuoso soloists,

[1]The author is in no way responsible for this name.

[2]Large enough to be clearly seen on stage by an audience.

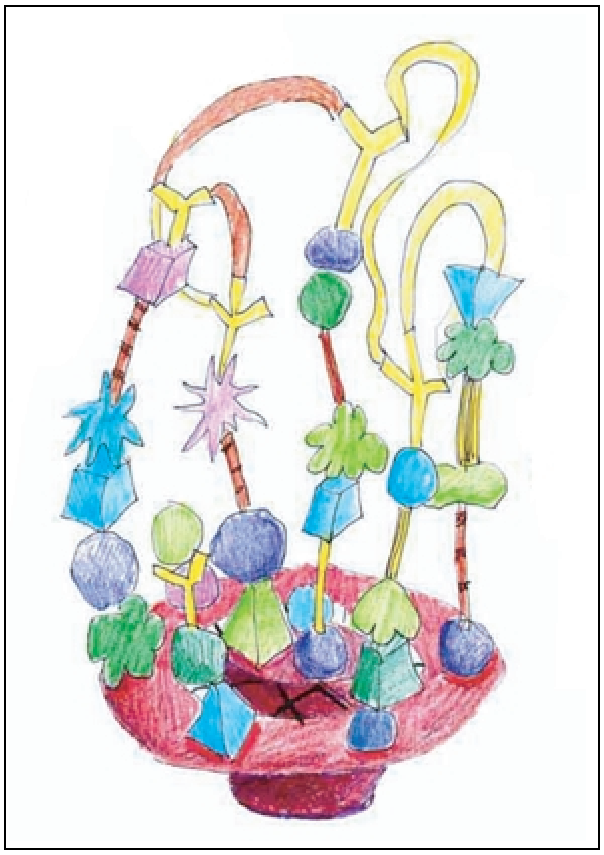


Figure 4-1: A sketch of the Big Thing by Margaret Orth (1999).

composers, and symphony orchestras around the world to interact and perform together through the medium of technology (see Section 5.2 for further discussion). The Big Thing was a central part of Toy Symphony and was supposed to highlight the compositional aspects of the project. Eventually the idea of using a purely software-based, graphical interface eclipsed the notion of using a large physical structure. It became increasingly clear that the functionality of a composition tool would be much more effective without the limitations imposed by the Big Thing structure. So an entirely new project called *Hyperscore* took shape.

Like the Big Thing, Hyperscore was developed with the goal of providing a compositional tool for the Toy Symphony project. Unlike the Big Thing, Hyperscore is a purely compositional tool and is not designed for interactive performance or improvisation. While the first two versions of Hyperscore are "batch mode" programs, where the user prepares the input data, waits for the computer to execute, then either accepts or rejects the entire output, the later versions are interactive and are intended to mirror traditional composition methods with an interface requiring a continuous cycle of sketching, listening, and editing.

One of the first conceptual questions that needed to be addressed was exactly how much of the compositional process should be automated, and how much should be human controlled. Since Hyperscore was supposed to be accessible to all users regardless of background, it was immediately apparent that the computer had to take care of some of the technical aspects of composition. But what exactly that entailed was difficult to define.

The basic conclusion reached after much consideration was that there are certain things related to composition that most people, even those without any musical training, find relatively easy to do. Very short, simple bits of music are not hard to compose—for example, making up melodic fragments or tapping out a spontaneous rhythm. On the opposite end of the spectrum, forming a mental picture of the large-scale structure or dramatic arc of a piece is not too difficult either. It doesn't take a professionally-trained composer to come up with something like, "I'd like a very calm section here, and then something crazy and complicated with a big climax in the middle, then another calm section." What *is* hard is coming up with all the structure in between the two extremes of melodic fragments and grand strategic design.

Thus one of the important concepts guiding the implementation of Hyperscore was the idea that the user should be allowed to determine these two extremes—the motivic elements and the large-scale structure of the piece—and let the computer take care of everything in the middle. However, exactly what "everything in the middle" entailed turned out to be a complicated matter.

4.2 Version 1: Fall 2000

4.2.1 Overview and internal structures

The very first Hyperscore prototype was mostly an attempt at building software infrastructure and experimenting with some basic musical ideas. The user is presented with a drawing window and nine pre-composed motives to select from (see Figure 4-2). The window only permits the user to draw a function (i.e. a single y-value for each x-value) and any combination of the nine motives can be selected or deselected by clicking on their corresponding images. The computer then generates a short piece based on the drawing and the selected motives. The user can click again on the Generate button to create another version of the piece given the same input.

The program plays back the piece using either the MIDI synthesizer on the computer's internal soundcard or an external MIDI device (for higher sound quality). Instrument sounds with precise attacks are used for all of the voices—either pizzicato or harp (see Fig. 4.1). The lower-level MIDI functions are implemented using the Rogus McBogus MIDI library [DP96]. The higher-level playback and score manipulation functions are encapsulated in the class ScoreObj. Each ScoreObj can

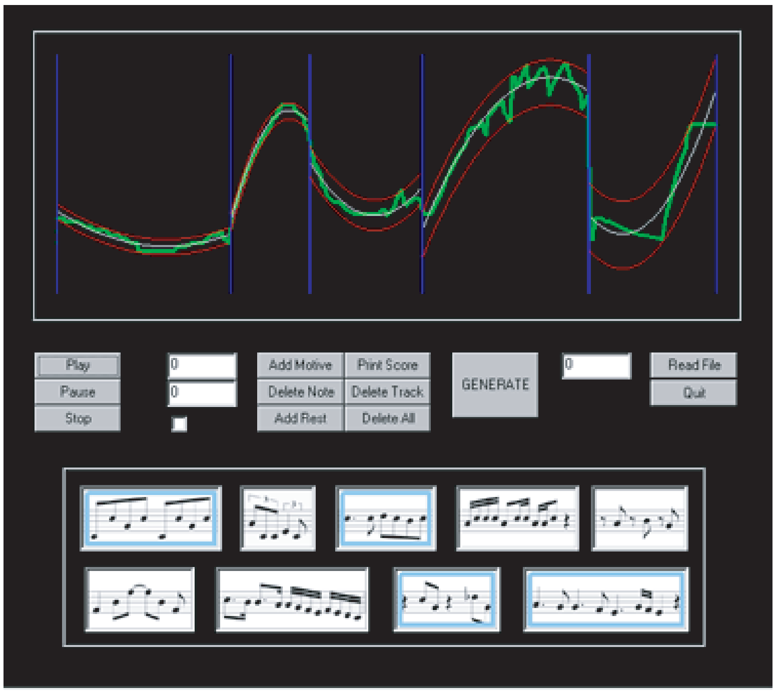


Figure 4-2: A screenshot of the first version of Hyperscore. The green line is drawn by the user, the blue, red, and white lines are added by the computer. The blue lines show the computed section divisions, the red lines show the min and and max boundaries of the section, and the white lines represent the estimated curves.

have an arbitrary number of tracks. In this version of Hyperscore, the generated score has five tracks. The motives are also ScoreObj objects, but have only one track. Each track consists of a list of note events with the following data members:

- *Pitch*. A MIDI value ranging from 0 to 127.
- *State*. Defined as ATTACK, TIE, or REST.
- *Duration*. Length of the note.
- *Velocity*. How loud the note should be played; a MIDI velocity value between 0 and 1027.

4.2.2 The generating algorithm

The generating function creates five independent musical lines for each piece, all of which are generated sequentially (as opposed to in parallel). The first voice, or *tenor*, is generated in a different manner than the other four voices. The computer randomly chooses one of the user-selected motives, and adds it to the tenor line's corresponding track. It continues randomly selecting motives and appending them until some maximum length is reached. This maximum length, which can be adjusted internally, is coded in *ticks*, the shortest possible subdivision allowed for any playable note (one tick is approximately 22 milliseconds). One quarter note is equivalent to 96 ticks, an eighth note is equivalent to 48 ticks, and so on.

The tenor line is intended to serve as the anchor for the piece. It insures that at any point in time, there is some active motive playing. The other voices serve as elaborations of this main line. As in the case of the tenor line, a user-selected motive is randomly chosen for the generation of the non-tenor lines. At this point, however, the generation methods diverge and there are three possible sequences that can be added:

1. The randomly selected motive.

2. A *counter-motive* to the selected motive.

3. A randomly-chosen series of rests.

A counter-motive, as the name indicates, is a matching melodic fragment that corresponds to a given motive. The idea of the counter-motive was conceived with Bach-style two-part counterpoint in mind. In a two-part invention, for example, there is often a sense of motoric completeness: when one line pauses or rests, the other line moves. In concept, counter-motives would be generated on the fly by the computer for each user-composed motive. The counter-motives in this case have been human-composed to match the nine pre-composed motives. Figure 4-3 shows two pre-composed motives and their corresponding counter-motives.

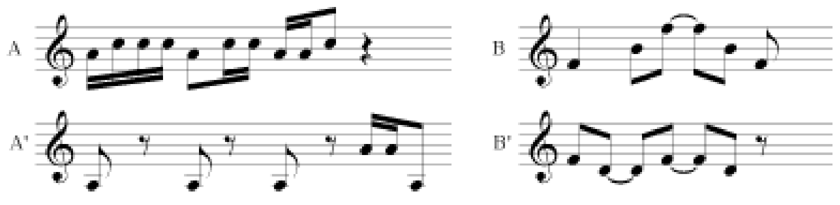


Figure 4-3: Two motives used in Hyperscore and their corresponding counter-motives.

The range of values possible for the random series of rests has been chosen based on experimentation. A minimum of zero and a maximum of six rests can be added in any one sequence. The duration of the rests is always an eighth note. For selected motives consisting of regular patterns of eighths and sixteenths (really all but one), the blocks of rests either fit metrically or add interesting syncopations. If the one motive with triplet values is selected, however, some very strange and complex rhythmic textures can result.

The choice among the three possibilities listed above is based on the texture (or bumpiness) of the curve at the position corresponding to the section in the timeline of the piece where the material is being generated (see Section 4.2.3). If the curve is very smooth, there is a higher chance of rests being added. If the curve is jagged, then it is likely that the counter-motive is used. If the curve is somewhere in the middle, then the original motive will most likely be used. There are no deterministic thresholds in making these decisions. The texture measurement can only influence the overall effect produced by the combination of five voices.

After the four remaining voices are generated in the manner described above, the octave ranges are scaled for each voice in order to widen the sonority range (Table 4.1). Prior to this change, the range of all the voices correspond to the pitch range of the original motives.

Table 4.1: A table showing transposition values in chromatic steps and octaves, velocity multipliers, and instrument assignments for each track.

Track	Chromatic steps	Octaves	Velocity scale	Instrument
0 (tenor)	0	no change	.85	harp
1	+12	up one	.70	harp
2	0	no change	.90	pizzicato
3	-12	down one	.90	pizzicato
4	-24	down two	1.0	pizzicato

Finally, the MIDI velocity values for each track are scaled in a two-step process. The base velocity is determined by the corresponding y-value from the drawing—the higher the y-value, the louder the note. The minimum velocity value (where y is zero) is set at 45 (which in effect prevents the note from being inaudible) and the maximum value is set at 127. All of the notes in each track are then multiplied by a scale constant (see Table 4.1) that takes into account the actual perceived loudness. For example, the only voice to be transposed up is scaled down the most, and the voice transposed to the lowest register is not scaled at all.

4.2.3 VectorAnalyzer

The curve analysis algorithm, also known as VectorAnalyzer, was written by Egon Pasztor[Pas01]. He subsequently joined the author fulltime on the project and is responsible for the beautiful graphics in Versions 2, 3, and 4. However, the hierarchical segmentation information produced by VectorAnalyzer (as described below) is not used to its full advantage until Version 4. Its primary function in this version is to describe the texture of the curve.

VectorAnalyzer attempts to fit different regions of the curve to linear and parabolic segments, and builds a hierarchical description of how the curve is constructed from such pieces. The curve is stored as a list of connected two-dimensional control points (x,y). Each control point is the same distance from the next (usually a few pixels). VectorAnalyzer begins by transforming each curve from its control-point representation into a single array of values which constitute the slopes of successive equal-length line segments. This data, a single vector of numbers, can be plotted as y-data versus x-axis. Any subset of these points can be fitted with a parabola that best approximates the general shape formed by the points using a least-squares fit.

The algorithm operates recursively; it begins by trying to fit the entire range of points (perhaps poorly) with one all-encompassing parabola. At each successive stage, the code takes a sub-range of data, also fitted with a parabola, and looks at the error (i.e. how far away the actual y-values are from the parabola values). It then considers, one by one, all of the possible ways of breaking the sub-range into *two* parabolas. That is, if its sub-range consists of 50 points, it will begin by considering fitting the first three points and the last 47 points with two separate parabolas.

It examines how well these parabolas fit, trying to determine if the potential bifurcation is "worth it." When breaking one parabola into two, the two-parabola fit has to be considerably better than the one-parabola solution to be selected. But from the point of view of this algorithm, selecting a two-parabola model increases the complexity of the resulting description by introducing a segment break. Thus, a heuristically chosen threshold is applied, and the two-parabola fit must be better than the one-parabola fit by greater than this threshold for the break to be considered.

The code then moves on, considering the first four points and the last 46 points with two separate parabolas, then the first five points and the last 45 points, and so on. If any of these two-parabola fits is heuristically seen to be "worth it," then the best breaking point is chosen and the algorithm calls itself recursively, breaking its interval into two intervals, each fit with its own parabola. In this way, the original global one-parabola fit is broken into two segments at the best point, the left and right halves further broken (possibly), and so on.

The algorithm produces a binary-tree of segments where the root is the complete curve, the top left and right children locate the point at which the curve is best broken into two sections, and so forth. This binary tree is then flattened to some heuristically chosen depth because the music-generation code requires a curve description that is presented as a list of segments rather than a tree. The texture measurement is the mean-squared error value of the segment containing the x-value corresponding to the position in the piece.

4.2.4 Analysis of Version 1

The results generated by this version of Hyperscore were interesting mostly from a rhythmic point of a view. As a texture generator, it worked quite well. It was entertaining to be able to scribble a line and hear something actively responding to the shape of it. But it lacked many important features that make music interesting. For example, there was no sense of phrasing, and no harmonic movement. Although there was some *feeling* of harmony, it was mainly a result of the general consonance created by a largely diatonic set of motives.

4.3 Version 2: Spring 2001

A feature lacking in Version 1 that had always been part of the grand Hyperscore-design scheme, was the ability to annotate the main curve, or *spine,* with other curves. Although exactly *how* the annotations would affect the music was a bit nebulous, it was assumed that they would convey some information about the motivic material used at the points in the piece corresponding to the annotated positions on the curve.

In this first attempt to experiment with annotations, the graphical interface of Hyperscore was completely redesigned and the concept of "pens" introduced. Each motive is mapped to a unique color indicated by a pen icon (Figure 4-4). The red pen is the only one without a corresponding motive—it is used to draw the main curve of the piece, which is now two-dimensional instead of one-dimensional (the starting point can be anywhere on the screen).

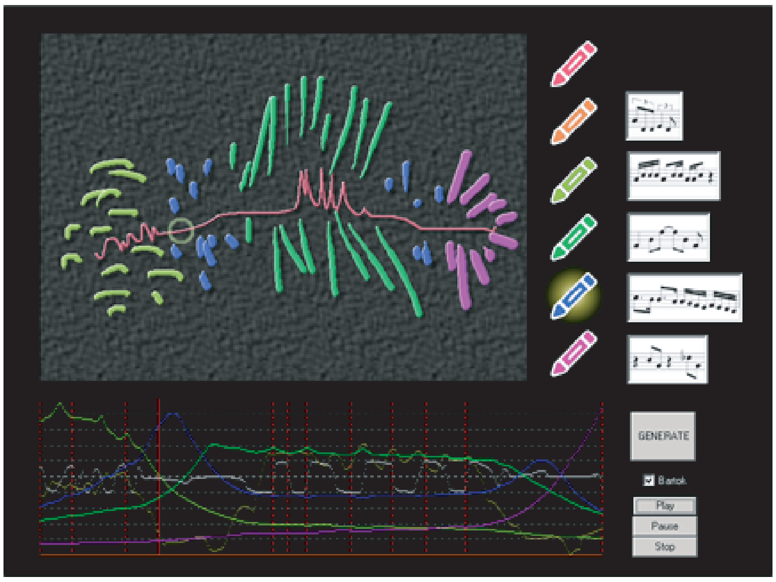


Figure 4-4: A screenshot of the Hyperscore Version 2. The red pen, which has no motive attached to it, is used to draw the main curve. The other pens are used to make annotations to the red line. The graphs below the drawing window indicate the prominence of each motive (colored lines) as well the curviness of the main line (white line) at any given point in time. The halo is a snapshot of a blinking cursor moving along the spine during playback. See Figure 4-5 for a musical realization of the this drawing.

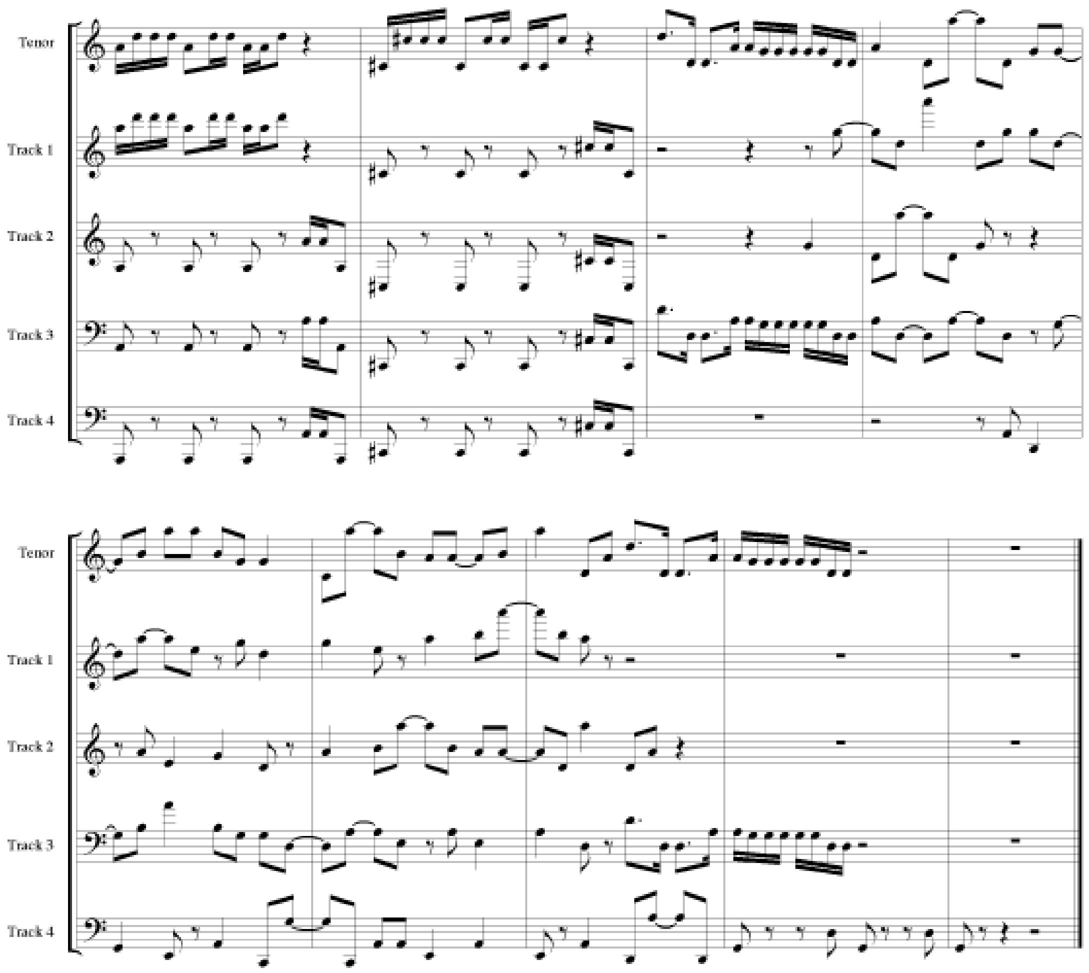


Figure 4-5: One musical realization of the Version 2 drawing shown in Figure 4-4.

The basic generation scheme in Version 2 is similar to the one described previously—there is a continuous tenor line and four other independent lines. However, the method by which the motives are chosen differs greatly from the random process used in Version 1; it is entirely dependent on the annotations to the spine. At every point, each color has a value which indicates how influential it is at that particular position along the spine. The less numerous and farther away the annotations are, the less influential. In effect, this allows the user to make broad decisions about what motivic material should appear at certain sections in the piece, leaving the lower-level decisions up to the computer.

4.3.1 Localized texture measure

The decision-making process for the non-tenor lines (i.e. choosing between motive, counter-motive, and rests) in Version 2 uses a new method for determining the texture of the curve. Instead of using an error metric, which results from the sectioning of the parabola-fitting algorithm, a new localized texture measure is applied [Pas01].

Localized curvature is highest where a curve is very "bendy," lowest where the curve is straight. As in the previous version, the spine is stored as a list of connected two-dimensional control points (x,y). To analyze the curviness, the spine is represented as an array of values corresponding to the two-dimensional angles of the lines connecting each control point. For example, a curve consisting of 50 points is made up of 49 line segments; the angle of each line segment is measured, and these 49 values make up the array. This array of angles is continuous, that is, the angle values are not bounded in $[0 \ldots 2\pi]$. Instead, the angle of the first line segment is stored in $[0 \ldots 2\pi]$, and the angles of successive line segments are determined by incrementing the previous angle by the amount that the curve turns right or left. For example, a curve spiraling counterclockwise and outwards from a point would not be represented by an array of angles beginning with 0, rising smoothly to 2π, then discontinuously jumping back to 0 and rising again. Instead, it would be represented by an array of angles beginning with 0 and rising smoothly beyond 2π, beyond 4π, and so on, depending on how many rotations the curve has.

An array of bounded-interval standard deviations of these angles is then computed. The interval length is hard-coded to be a visually short distance, (about ten control points). Thus, for our example curve of 49 angles, the standard deviation of the first ten points (values 0 through 9 inclusive) would be taken, then the standard deviation for the next ten points (1 through 10) would be taken, and so forth. Thus the array of 49 angles becomes an array of 40 standard-deviations.

4.3.2 Harmony generator

Another new feature of Version 2 is the harmony generator. The algorithm implemented uses *hierarchical* Markov chains to handle different layers of organization [Roa96, p.880]. One set of Markov chains is used to generate a series of higher-level harmonic functions, and another set is used to generate the actual chords.

There were a number of different Markov chains tested, some more complicated than others. One particularly effective but simple method used a chord function table consisting of transitions between three types of chords: *tonic*, *dominant*, and *subdominant* [Sch01]. Table 4.2 shows a set of experimental transition values used for one time step. The values in the table were determined mostly from personal observation.[3] For example, in traditional harmony, a transition from a dominant to a subdominant(as in **V** to **IV**) can be very awkward, and thus is restricted in the table. On the other hand a chord with a tonic function can be followed by chords of any other function; e.g. it can be extended with another tonic-function chord, followed by a subdominant which prepares for an eventual dominant, or followed directly by a dominant.

	Tonic	Subdominant	Dominant
Tonic	.4	.3	.3
Subdominant	.1	.6	.3
Dominant	.7	0	.3

Table 4.2: Harmonic function state transition matrix.

The chord functions are selected based on two conditions:

1. The time t at which the chord occurs. The highest value for t corresponds to the final chord in the piece.

2. The function of the previous state (Table 4.2).

Thus the probability of a chord function at time t is

$$p(f_i|f_{i-1},t) = p(f_i|f_{i-1})p(f_{i-1}|t) \tag{4.1}$$

For major and minor keys, triadic chords are used to determine the actual note values of the selected chord type. Table 4.3 shows the chords available in each category for a major key.

Regardless of the type of harmony involved, a chord is chosen according to two criteria:

1. The time t at which the chord occurs.

2. The relative frequency at which the chord would appear regardless of the circumstances (i.e. the zeroth-order probability).

[3]This is in no way claiming that the rules of traditional harmony have been satisfactorily encapsulated in these simple tables—far from it. However, some basic principles can go a long way in generating something quite coherent.

This results in the following equation for the probability of a chord given a particular function at time t:

$$p(c_j|f_j,t) = p(c_j|f_j)p(f_j|t) \tag{4.2}$$

where L depends on the number of chord possibilities for a given function (Table 4.3).

Function	Possible chords for each function
Tonic	$\mathbf{I}$, $\mathbf{I^6}$, $\mathbf{vi}$, $\mathbf{vi^6}$
Subdominant	$\mathbf{ii}$, $\mathbf{ii^6}$, $\mathbf{ii^7}$, $\mathbf{ii^5_6}$, $\mathbf{IV}$, $\mathbf{IV^6}$, $\mathbf{vi}$, $\mathbf{vi^6}$
Dominant	$\mathbf{V}$, $\mathbf{V^6}$, $\mathbf{V^6_4}$, $\mathbf{V^7}$, $\mathbf{V^5_6}$, $\mathbf{V^2}$, $\mathbf{vii^7}$, $\mathbf{vii}$, $\mathbf{vii^6}$, $\mathbf{vii^5_6}$

Table 4.3: Harmonic functions and their respective chords.

Aside from the traditional major and minor keys, there is an additional type of harmony available dubbed "Bartok." Unfortunately, it does not reproduce truly Bartok-esque harmony. But what it does do is create chords based on fourths instead of thirds. The tonic, dominant, subdominant labels don't quite apply, but they are still used to impose functionality, as in the case with regular major and minor.

The generated lines are altered to fit the chords in the newly-created progression. First a *beat* duration is chosen to determine when chord changes happen. Since the motives are played at a fixed tempo in this version, the duration value is hard-coded as a half note. The notes in each voice are accordingly altered by comparing the original pitch of each note to its corresponding chord tones (i.e. falling at the same point in time). The pitch is then changed to the closest chord available, "closest" meaning the nearest pitch regardless of the octave. This somewhat rough method of conforming the piece to a generated progression is used (for the most part) until Version 4.

One other harmonic model was tried (albeit briefly) which experimented with the concepts of transposition and modulation. The idea was that the function of a given chord could be altered if a sudden change in curve texture appeared. For example, if a subdominant function were morphed into a dominant function because the spine became extremely jagged, then the chord in question would turn into a secondary dominant which would resolve to a new tonic. As a result, the root of the current tonic could potentially be in a constant state of flux. Due to other implementation matters taking precedence, this idea was not sufficiently explored.

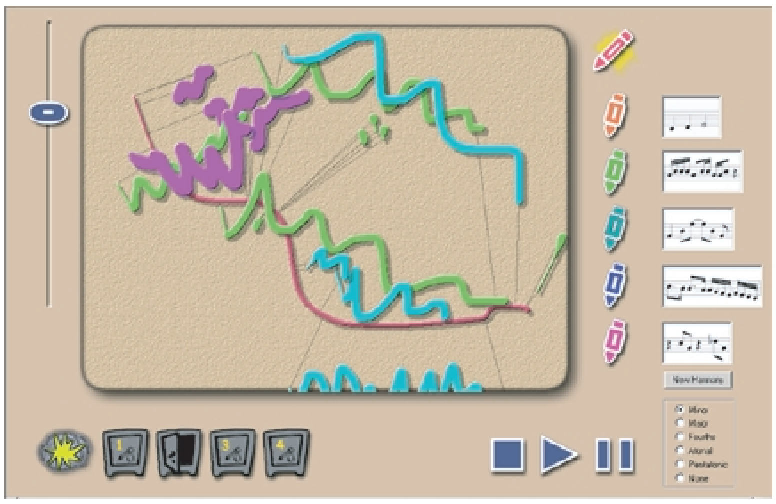
4.3.3 Analysis of Version 2

While the harmony generator did add more depth to the music composed, it was too simplistic to work with anything longer than half a minute of music since the progressions tended to meander without any sense of phrasing or direction. Nevertheless, the algorithm was sufficient to produce some very satisfactory progressions on occasion. In essence, it was a hack, but a useful one that served as a placeholder for harmonic function while other parts of the program were being developed.

It also became clear, after some user-testing, that people were often confused by what the computer was doing in response to the drawing. The bumpiness to rhythmic activity map in Version 1 was actually a little clearer because the segmentation algorithm provided a better visualization of the texture in a given area. The way the annotations were interpreted was also slightly obscure, especially to people who had no musical training and couldn't pick out the motives easily. For people who were more musically inclined, the sudden introduction of counter-motives was often confusing because, unless told explicitly, they had no idea that musical material aside from what was displayed the screen was going to be used.

4.4 Version 3: Summer 2001

After considering the problems encountered in Versions 1 and 2, a completely different approach was devised for mapping motives to annotations which involved more human decision making than machine generation. Instead of *influencing* the decision-making process of the computer, the annotations deterministically *dictated* it.



[✓ Minor] Major Fourths Atonal Pentatonic None

Figure 4-6: A drawing made with Hyperscore Version 3, harmonized in a minor key. The tempo knob is to the left. See Figure 4-8 for the musical realization.

4.4.1 Annotation functions

The new generation algorithm works by assigning each annotation a start and end point which indicate when a motive should begin playing and approximately when it

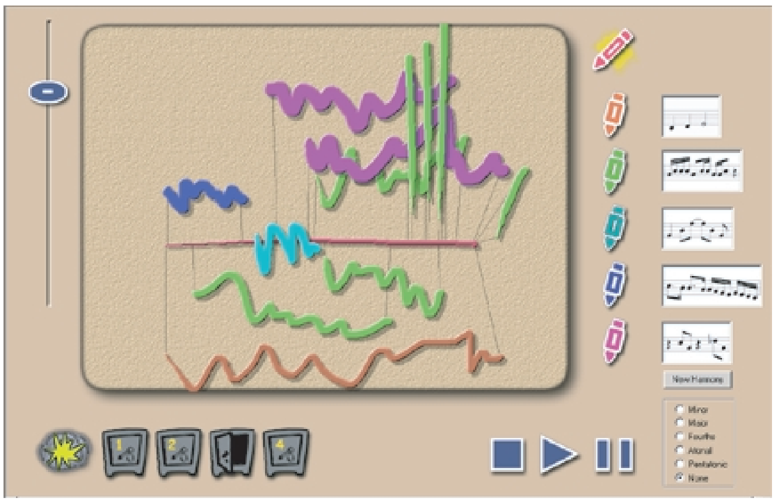


Figure 4-7: Another sketch made with Hyperscore Version 3. See Figure 4-9 for the musical realization.

should end. The motive (mapped to annotation color like the last version) is looped until the end point is reached. Even if the end marker is reached without the last iteration having finished, the motive is still completed, extending it a bit beyond the boundary.

The start and end points are displayed graphically as little black threads extending from the ends of the annotation to their respective points on the spine (see Figures 4-6 and 4-7). As in the previous versions, each curve (both the spine and the annotations) is stored as a list of equidistant, two-dimensional control points (x,y). In order to find where the black tethers attach to the spine, each control point on the spine is checked sequentially for its distance from the start and end points of the annotations. When the closest control point is found, a black line is drawn from the point to the corresponding annotation start or end. The annotations are made symmetric: the point corresponding to the *earlier* point on the spine is termed the start, even if the annotation was actually drawn the opposite way. These curve positions are recomputed every time the spine or any annotation is moved, to keep the graphics consistent.

The average distance from any given annotation to the spine determines the pitch register of the corresponding motive. The farther above the spine the annotation is, the higher the motive is transposed, the farther below the spine the lower the pitch. An annotation drawn at the level of the spine itself results in no transposition up or down.

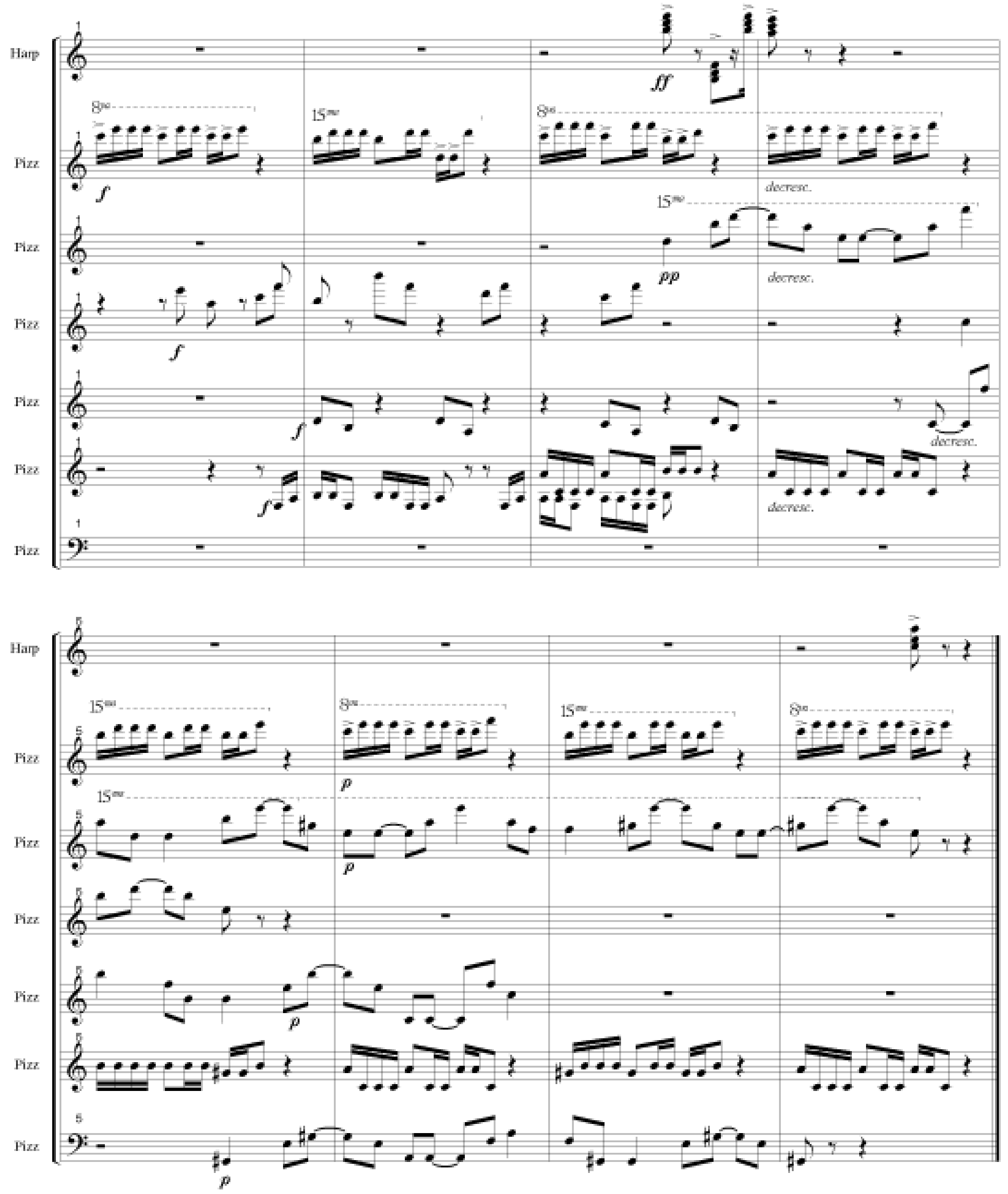


Figure 4-8: A short musical realization of the drawing shown in Figure 4-6 generated by Hyperscore. The second and third lines from the top (the first two "Pizz" staffs) are much softer than the other lines except for the notes that are accented. The reason behind this the fact that the notes are so high that they are not only extremely soft, but difficult to detect as pitches. This factor as well as the y-values of the spine are the source of the dynamic markings. Note that the harmony type is "none."

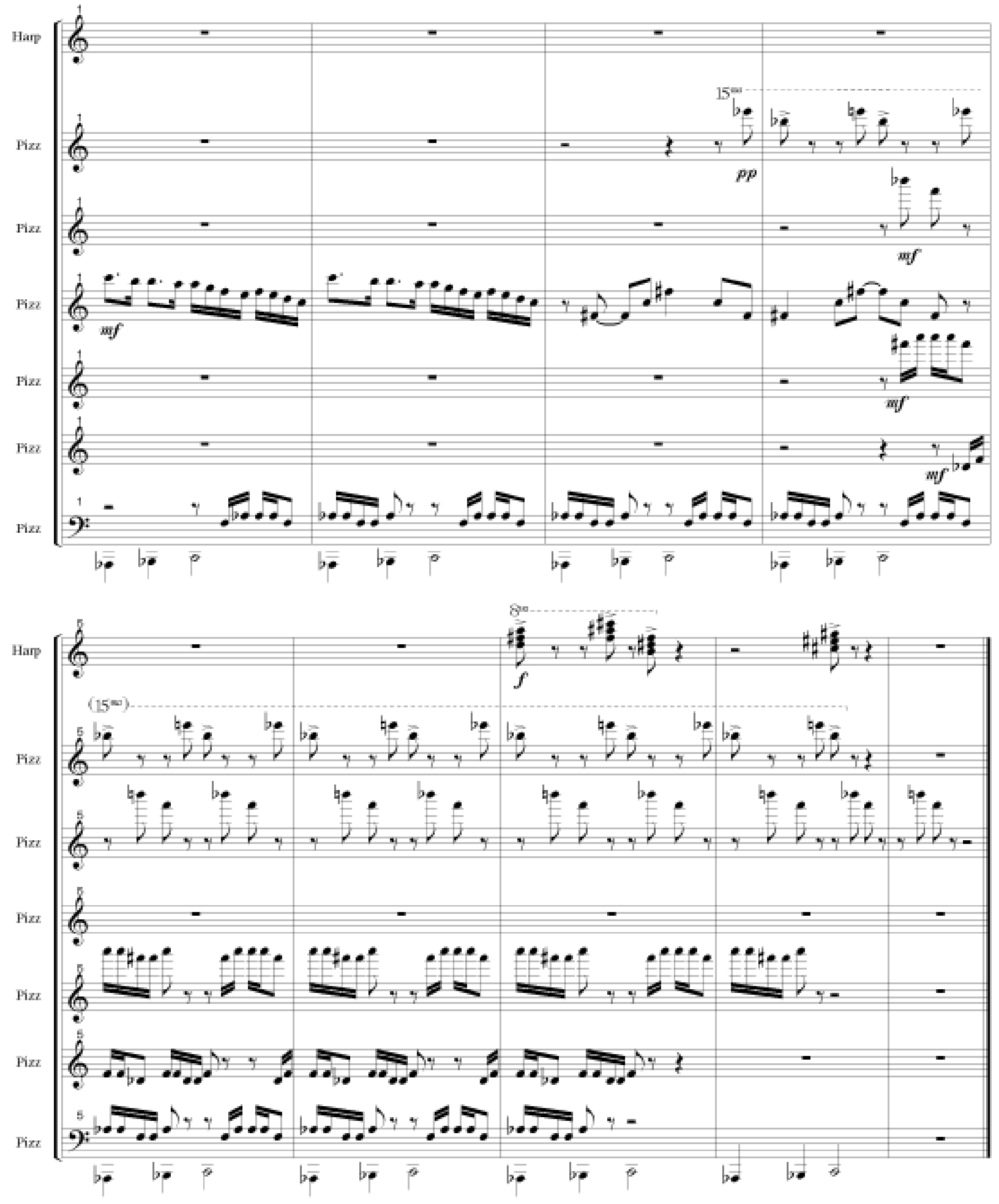


Figure 4-9: A short musical realization of the drawing shown in Figure 4-7.

The bumpiness of the annotation determines the timbre of the motive playback. This texture value is calculated in the same way the curviness of the spine is determined in Version 2. The only difference in this case is that the localized curvature values are all averaged to come up with a single measure. This value is compared to some heuristically chosen threshold in order to choose the timbre.

If an annotation is drawn as a vertical "spike" relative to the spine, then an accented chord is generated rather than a motive. This "spike" is also assigned a different timbre.

There are three main instruments used for playback purposes. Originally, a sustained string sound was intended to be mapped to smooth annotations, and pizzicato to bumpy ones. Unfortunately, the reality of the matter is that MIDI sustained strings generally sound awful. So another plucked instrument (e.g. guitar) was used as a substitute. The third instrument, harp, was mapped to the accented chords created by the spiked annotations.

4.4.2 Other internal and external changes

The ScoreObj class was overhauled so that it would easily allow parallel and non-sequential insertion of musical material into tracks. The main ScoreObj was extended to 30 tracks and three MIDI channels (ten tracks per channel, each channel mapped to a different instrument). A MIDI queue was added to keep track of all the currently playing notes. This was necessary in order to cease playback immediately at any time (issuing an all-notes-off message—127 note-off messages for each of the three channels—was creating an intolerable delay).

There are also three new harmony types to choose from:

- *atonal* randomly generates the notes in each chord.
- *pentatonic* forms chords that are based on the pentatonic scale (or "black note" scale.)
- *no harmony* does not impose any harmonization scheme on the motives. The melodic intervals between the notes in each reproduction of the motive remain identical to the original. This fixed structure gets shifted chromatically up or down by an amount that corresponds to the distance between the annotation and spine (see Figures 4-7 and 4-9 for an example of a score which uses *no harmony*).

The score now regenerates automatically in response to any change made to the spine or annotations. A "New Harmony" button has been added in conjunction with an updated harmony generator that gives the user the option of saving chord progressions instead of ending up with a new one whenever the score is regenerated. A tempo knob has been created to adjust the relative playback speed of the piece as a whole.

Spine and annotation editing capabilities have also been implemented. Right-clicking on a highlighted curve deletes it and left-clicking shifts the entire object. "Safe" icons have been added to flip quickly to a new project window without having to use menu items to open a new file. Clicking on a different safe automatically saves the current working environment before opening the new one.

4.4.3 Analysis of Version 3

There was no doubt that the mini-pieces composed by this new version far superseded any of the ones generated by the earlier versions. A piece could be built up slowly from a couple of annotations to a complex, interweaved combination of dozens of annotations. What was lacking still, though, was the real sense of tension and release on a global level. On a local level, a clever combination of motives at different pitch ranges as well as an adequate chord progressions generated by the computer could create effective moments of tensions and resolution. However, while fine control was possible, it was still awkward to make detailed changes, given the limited editing and musical capabilities. From another perspective, there was arguably *too much* unstructured decision-making left to the human. It was entirely up to the user to structure the motives in such a way that resulted in something musically interesting. While it this would not be a difficult task for a composer, a musically-untrained user might have some trouble.

4.5 Version 4: Work in Progress

At the writing of this thesis, Version 4 is only partially implemented. It attempts to solve many of the problems that have come up in the earlier versions, although there are still conceptual decisions that have to made before it can be completed.

Perhaps the most immediately obvious change is the fact that the motives are now editable (Figures 4-10 and 4-11). The new motive-editing view enables the user to select the duration of a note by using the arrow keys and then add the note to the motive by clicking on the graph. The horizontal axis represents time and the vertical axis pitch. There will also be a chromatic versus diatonic mode; in the diatonic mode, a chromatic note can be added much in the same way a sharp or flat is notated in standard music notation. This change will be indicated by the coloring of the note icon. The user will also be able to hold down the mouse and drag it across the screen to create automatic scale patterns.

4.5.1 Physical stroke model

A new type of editing introduced in this version allows curves to be altered in a natural way without having to delete and redraw the entire object [Pas01]. The left mouse button is still used for shifting the entire curve at once, but the right mouse

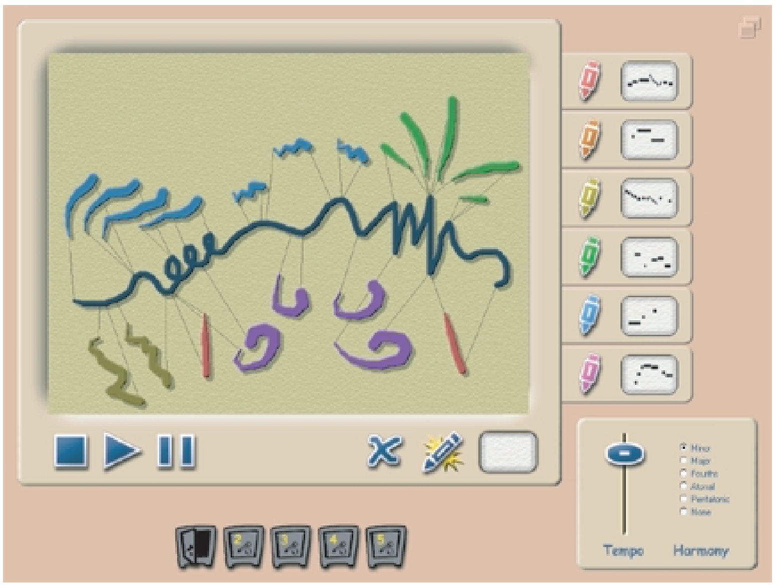


Figure 4-10: The main view of Hyperscore Version 4.

button is now mapped to this new feature (the delete key instead of right-click now removes a motive).

Professional drawing packages like Illustrator represent curves as linked Bezier[4] segments, and the user is allowed to explicitly click-and-drag the vertices of the Beziers' control polygons. This type of control-point curve editing can become tedious. In Hyperscore, the user is able to simply grab the curve and pull. Physically simulated, the curve responds in a natural manner, and small or large changes can be made in the same way one would drag a rope.[5]

As in previous versions, each curve is stored as a list of connected two-dimensional, equidistant points. The line segment between each pair of control points is modeled as a spring with a fairly stiff spring constant. Also, each internal joint (each control point that is not the first nor the last one) has an angular spring that keeps the joint angle around a certain value.

The simulation is done in the standard way physical systems are modeled: the control points are masses with position and velocity states. The springs exert forces on the masses, which are integrated over time using a standard Runge-Kutta fourth-order ordinary differential equation solver [Ger99]. At each progressive time step, the quiescent angle of each of the angular springs is altered toward the current angle. Thus, if the curve is bent and held for a time with the mouse, upon release it will

[4]A Bezier curve in its most common form is a simple cubic equation.

[5]Golan Levin used spring simulations in his *Audiovisual Environment Suite* to create responsive curves that moved in a natural, intuitive manner [Lev00b].

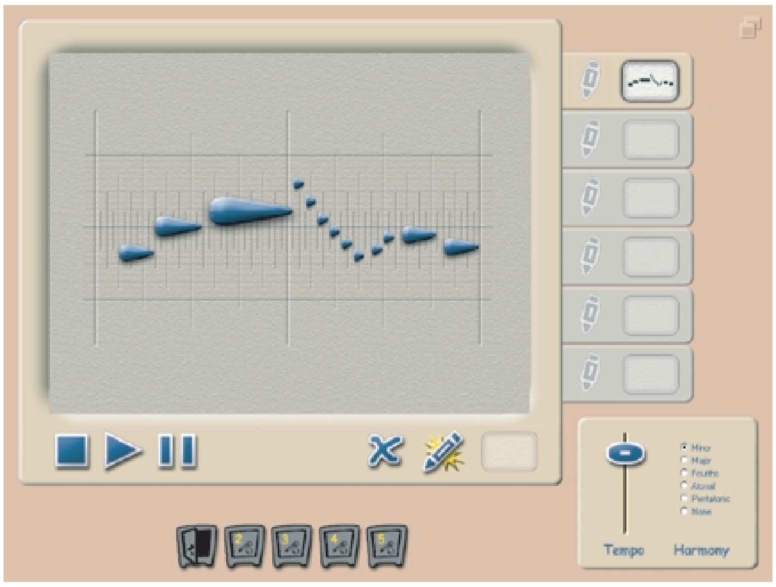


Figure 4-11: The motive-editing view of Hyperscore Version 4.

have solidified closer to the state at which it was held.

4.5.2 An improved harmonic progression generator

A completely new harmonic progression generator is being implemented which takes qualitative input from the contour of the spine. For example, if there is a rising segment with a clear climax point in the spine, then this will be reflected in the chord progression the computer chooses.

The structure behind the chord progression is David Cope's SPEAC system. Cope developed the SPEAC system as an alternative to traditional harmonic analysis and is based on ideas derived from the work of Heinrich Schenker [Cop96]. It provides a level of abstraction for describing the motions of notes, harmonies, and motives. SPEAC is an acronym for five identifiers:

- *statement* (**S**). Exists "as is" and is not the result of other activity. They typically occur near the beginning of musical passages.
- *preparation*(**P**). Precedes statements or other identifiers and are not independent.
- *extension* (**E**). Follows any identifier other than another extension or a preparation.
- *antecedent* (**A**). Causes a significant implication and requires a resolutions (typically a *consequent*).

- *consequent* (**C**). Appears in response to an antecedent motion and is often the same chord found in a *statement*.

Users will be able to change individual chords or groups of chords by highlighting the region and then clicking on a chord-edit icon. This will then generate a replacement chord or group of chords that still fits the function described by the SPEAC progression generated with input from the curve shape.

4.5.3 New mappings

Special motive transformation functions have been implemented but are not currently mapped to graphical representations. These include augmentation, diminution, retrograde, and inversion. The current idea is that these motive transformations will be indicated by subtle changes in the shape of the annotations. Timbre, formerly mapped to annotation shape, will have to be indicated in a different way—perhaps by an etched pattern on the curve surface.

As this is an ongoing project, there is still much that is undecided. The next (and final) chapter in this thesis will attempt to discuss in more detail future designs that go beyond the scope of this current version, as well as Hyperscore's role in the Toy Symphony Project.

Chapter 5

Conclusions and Future Work

5.1 Assessment of results

Hyperscore successfully presents a unique interface for visualizing musical elements. But there is still much to be desired in terms of how the computer supports the user musically. Most of the musical as well as curve-analysis algorithms are clever hacks which serve as placeholders for future, better algorithms.[1] Once these algorithms are improved and are truly able to shape the direction of the music, Hyperscore will have fulfilled its most fundamental goal.

5.1.1 Improvements

Many of the problems encountered in earlier versions are being addressed in Version 4. These include the lack of a motive-editing interface, the inability to control how (if at all) the motive is transformed when annotated in the score, and the poor quality of the harmony generator. However, there still remains the basic question of deciding exactly how graphical features map to musical parameters—and how much is automated by the computer.

Versions 1 and 2 were in some ways too automated—it wasn't entirely clear what the computer did in response to what was drawn. While there is a degree of high-level control, there is no real sense of *composing* a piece. Version 3 was an attempt to balance in the other direction with more one-to-one mappings between annotations and motives. The user draws an annotation and can expect to hear something immediately recognizable as a result of it.

Perhaps the key here is for certain users (people with the least musical knowledge or skill) to allow the computer to do more of the work. One solution is to have a special "computer pen." Drawing annotations with this pen give the computer license to add extra material that is not necessarily controlled by the user at all.

[1]When there is a demo or sponsor meeting at the Media Lab, hacks implemented due to time constraints often end up overstaying their welcome.

Thus if a user is simply dissatisfied with a particular section and cannot see a way of improving it, adding a "computer pen" annotation will add musical material that is completely out of the user's hand and leaves that section of the music open to the "discretion" of the Hyperscore generating algorithm.

Another special editing feature planned for future versions is the "unique" motive-editing option. This type of editing would allow a motive to be altered locally and would apply only to the annotation selected. Unlike the global motive-editing view, the motive mapped to the color of the annotation would not change, thus leaving other annotations which use that same color untouched. An annotation updated in this manner might be indicated by a colored-lining or flag-like mark.

Most importantly, Hyperscore needs to be able to find an effective way of mapping the spine to the gestural or "storyline" elements of the piece. This will require a meaningful correlation between the hierarchical parsing of the line and the corresponding (mostly) hierarchical structures in music.[2] In order to present the user with a better visualization of this structure, zoom and scroll views will be added to the interface to enable easy switching between these levels of abstraction.

5.1.2 User response

During the course of many Media Lab demos, a considerable number of users from diverse musical and technical backgrounds had a chance to experiment with Hyperscore. The deficiencies in Versions 1 and 2 were immediately apparent from their reaction: even after an explanation of what was going on, many of them remained confused. On the other hand, most people picked up how Version 3 worked immediately, once given a quick demo and shown some sample compositions.

Users who sat down with Version 3 for extended periods of time clearly had a lot of fun and enjoyed listening to how the musical output responded to their drawings. One very positive comment (made by a composer) took note of the fact that while the mappings from graphics to music were very direct, the results were still often surprising and interesting, unlike the entirely predictable output of a sequencer-like program such as Jeanne Bamberger's Impromptu.[3]

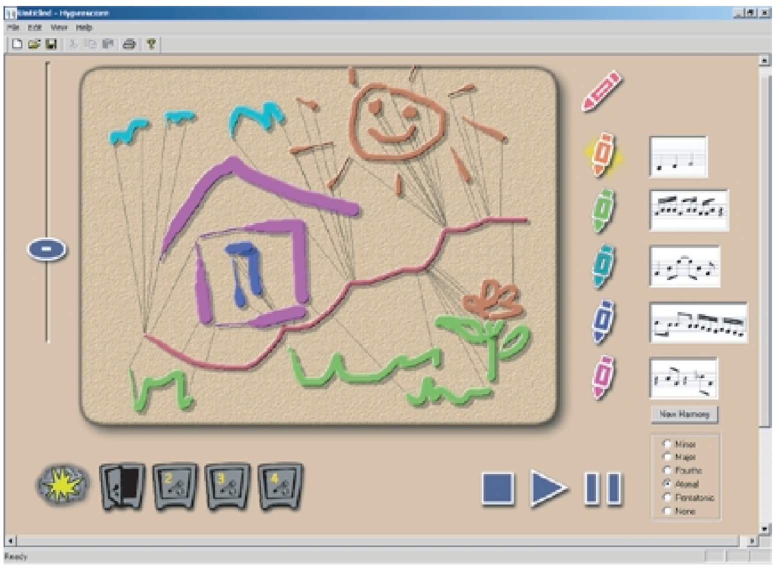
However, users often found it frustrating to have to play back the entire composition in order to listen to a change made in a very small region. An improvement purposed by Egon Pasztor for future versions entails a movable playback region denoted by start and endpoint markers that can be shifted around the spine.

Finally, there was always the occasional person who would draw a house, or a person, or some other identifiable image in order to "see how it sounds." One such problematic sketch is shown in Figure 5-1. Needless to say, the musical results were usually incoherent. The best defense against this type of misguided creativity is that

[2]Something along the lines of David Cope's use of SPEAC and augmented transition networks comes to mind.

[3]This is in no way to disparage Bamberger's work. Unlike Hyperscore, the design of Impromptu focuses on pedagogical goals.

Hyperscore is not designed as a sketch pad, but as a composition tool. The graphical "meaning" is irrelevant to the musical intent—it serves as a form of notation. A user whose goal is to compose a piece of music will see this quickly and will abandon attempts at creating landscape paintings in favor of getting musical results.



Minor
Major
Fourths
[✓ Atonal]
Pentatonic
None

Figure 5-1: Example of a problematic drawing made by a test user.

5.1.3 Amateur versus professional users

During the course of user testing and from personal experience composing examples, some differences and similarities between the responses of amateurs and professional users became apparent. Both types clearly enjoyed having the ability to shape the piece at a macro level. However musicians, particularly composers, tended to want more control of the inner workings. While this level of control could be optional, it still had to be there. Broadly speaking, it seemed that the requirements of a musician were a superset of the amateur's.

If Hyperscore is to be a truly effective composition tool for professionals as well as amateurs, there need to be more precise editing features as well as deeper hooks into the compositional process. The musician should have the choice of being able to access these details just as the amateur should have the choice of letting the computer take care of as many details as necessary.

5.2 Toy Symphony

Hyperscore's most significant future application is its role in Toy Symphony, an international music performance and education project led by Tod Machover. The goal of Toy Symphony is to introduce children to creative music-making through the use of specially designed *Music Toys.* These toys will enable them to engage in sophisticated listening, performance, and compositional activities normally accessible only after years of study.

> Toy Symphony proposes a new, integrated approach to music education and creativity. Coherent musical concepts weave together intensive hands-on workshops, online activities for home or classroom, technological explorations, and a culminating public concert. In this way, Toy Symphony brings together kids and professionals, old and new instruments, and brand new music composed by novice and expert, linking the past, present and future of music for both children and adults to understand and enjoy [MM01].

5.2.1 Workshop and performances

The Toy Symphony project will tour around the world starting in 2002.[4] A complete set of Music Toys will be used in each host city to help children create sounds and compositions for both traditional instruments and the toys themselves.

For a period of a week, a series of workshops will be held, introducing a large number of children to musical expression and composition using Music Toys. Rehearsals with a smaller group of about 50 children playing Music Toys, a Hyperviolin soloist, and the orchestra are followed by a final concert presented to the general public. Pieces composed for the concert, featuring a combination of Music Toys and orchestra, are used as pedagogical models in the workshops and include works by Machover and other contemporary composers.

5.2.2 Music Toys and the Hyperviolin

The Hyperscore software environment, although not an actual physical device, is one of the four Music Toys. The others are performance instruments as well as learning tools. The Hyperviolin, unlike the Music Toys, is designed exclusively for a professional musician.

Beatbugs

The Beatbugs are handheld digital instruments that provide a formal introduction to mathematical concepts in music through an expressive and rhythmic group expe-

[4]The premiere will be in February 2002 with the Deutsches Symphonie-Orchestra conducted by Kent Nagano.

rience (Figure 5-2a). Multiple Beatbug players can form an interconnected musical network by synchronizing with each other, trading sounds, and controlling each other's musical patterns. Interaction among players enriches the musical experience and encourages collaboration and social play [Tod01].

Music Shapers

Music Shapers are soft, squeezable instruments, which allow players to mold, transform, and explore musical material and compositions (Figure 5-2b). Using capacitive sensing and conductive embroidery to measure physical contact, they give children the opportunity to influence high-level musical parameters such as contour, timbre, density, and consonance [MM01].

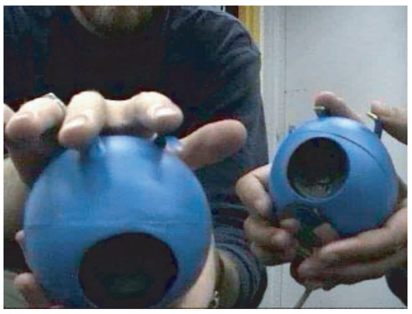
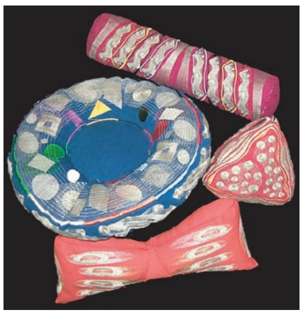
a)  b) 

Figure 5-2: Current Music Toys. a) Prototype of the Beatbugs. Bending the antennae changes rhythmic speed and timbre [Aim01]. b) Music Shapers designed by Margaret Orth [Ort01].

Voice Transformers

Voice Transformers use vocal input to control and create music. Users sing into one end of the device and hear a transformed version of the input coming out of the other end. The generated sound is based on natural forms of expression such as pitch, loudness, brightness, and timbre.

Hyperviolin

The Hyperviolin is the latest hyperinstrument designed for professional players. It features multi-channel audio analysis software and wireless hardware technology embedded in an enhanced violin bow and in the performer's shoes. The Hyperviolin is being developed with Joshua Bell as the primary soloist.

5.2.3 The role of Hyperscore

Hyperscore serves as the principle composition tool in Toy Symphony and is at the core of the creative workshop activities. The end goal is for children to use Hyperscore to compose a three-minute piece for string orchestra. The children will either learn how to use Hyperscore at the workshops in the week before the concert, or considerably earlier if they download it from the web (see Section 5.2.4). There are three types of Hyperscore workshops children will engage in. The first workshop is led by a composer or music educator and introduces compositional techniques to the children using Hyperscore. The lab workshops are times set aside for children to come in on their own (perhaps with a teacher or composer around to provide help) and create their own three-minute pieces. The final workshop is a mini-concert where children get to play their pieces for each other. Out of the (hopefully) many pieces composed, one will be chosen to be performed by the orchestra at the actual concert.

5.2.4 Hyperscore online

An online version of HyperScore will be available on the Toy Symphony website to give children throughout the world the possibility of creating and sharing individual and group compositions. Children will be able to "post" their compositions, much like people post their MetaSynth compositions on the web. The web version of Hyperscore will be for the most part identical to the final in-house version with the exception of more elaborate input and editing devices.

5.2.5 Input methods

Since the mouse is not an ideal drawing tool, there are several other input methods being considered. Commercial devices such as Wacom tablets or touch screens are perhaps the simplest solutions. Another method is camera input. Children could then draw on *any* surface. The computer would take a snapshot of the drawing and use it as initial Hyperscore input. This would, however, require specialized hardware such as a frame grabber card and camera as well as processor-intensive code to parse the input images with accuracy.

Another possible method is the use of audio input for motive composition. The user would hum or sing short melodies into a microphone and the computer would approximate the pitches to the nearest chromatic note and quantize the rhythmic values to some user-specified duration.

The Music Toys themselves could very easily be used as input devices. The Shapers and the Beatbugs either output serial or MIDI data and can be easily hooked up to a Hyperscore machine. The Beatbugs could be used to specify rhythmic patterns in motives[5] and the Shapers could be used as editing tools to shape or warp

[5]Perhaps in future versions of Hyperscore, there will be different categories of motives, in which

the spine and annotations. This works especially well with the physical stroke model discussed in Section 4.5.1. Precise Bezier-segment editing, for example, wouldn't make sense for a Shaper (it makes more sense to use a mouse to select and move control points) but a physical model is a natural match for a tangible interface—squeezing and twisting the Shaper would make the curve contract and twist in the same way [Pas01].

5.2.6 Requirements for Toy Symphony

Before Hyperscore is ready for Toy Symphony, a decision has to made concerning the structure imposed upon the final version used by children. Since they have the explicit goal of composing a short piece for string orchestra, the program has to provide them with an environment that ensures that the final output will work for a real orchestra without a considerable amount of human editing. For example, each instrument has a range restriction as well as technical restrictions which need to be considered. The piece must not be too difficult since in some cases there will be only one rehearsal before the performance.

There also needs to be a module that converts the internal score representation into standard music notation. It is quite possible that at this stage, given the what the child has done, the piece will be altered in ways that make it more amenable to performance by real musicians. If the program restricts the number of possible simultaneous lines to a reasonable maximum number of *divisi* parts, this task might be greatly simplified.

5.3 Beyond Toy Symphony

There are many possible directions Hyperscore can take after Toy Symphony. In its current incarnation, Hyperscore is intended to help the user compose music meant for performance by real musicians. In the future it would be nice to have the option of having the computer playback be the end goal, not just helpful feedback. Since MIDI is not ideal for expressive output, using audio, or a combination of MIDI and audio, is probably the best direction to take. If audio were indeed a component, there is no reason why the current motivic construction format couldn't remain the same. The interface for motive input would have to change, however—a piano-roll notation for non-pitched sounds would not be ideal.

One of the more intriguing ideas is a *reverse* Hyperscore, where the input is a piece of music (in MIDI format, for example) and the output is a Hyperscore sketch complete with editable spine and annotations. Arguably, this is a much harder task than the current graph-to-music approach. There would have to be some concrete method of breaking a piece down into basic motivic elements, perhaps by doing a statistical analysis of recurring rhythmic, melodic, and harmonic patterns.

case one possibility is a purely rhythmic one.

In short, there are many diverse and fascinating possibilities for improving and modifying Hyperscore. It is the author's hope that this thesis presents only the beginning of a project that will, in the future, be a truly innovative addition to the field of computer-assisted composition.

Bibliography

[Aim01] Roberto Aimi. Website, http://www.media.mit.edu/˜ roberto, 2001.

[Bam00a] Jeanne Bamberger. *Developing Musical Intuitions.* Oxford University Press, New York, 2000.

[Bam00b] Jeanne Bamberger. *User's Guide for Impromptu.* Oxford University Press, New York, 2000.

[Ber95] Dimitri Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont Massachusetts, 1995.

[Bro97] Azby Brown. Portrait of the artist as a young geek. *Wired*, May 1997. http://www.wired.com/wired/archive/5.05/ff_iwai_pr.html.

[Car98] Carlos Agon, Girard Assayag, Olivier Delerue, and Camilo Rueda. Objects, Time and Constraints in OpenMusic. In *Proceedings of the International Computer Music Conference*, Ann Arbor, Michigan, 1998. http://www.ircam.fr/equipes/repmus/RMPapers/ICMC98a/OMICMC98.html.

[Cha97] Joel Chadabe. *Electric Sound: The Past and Promise of Electronic Music.* Prentice-Hall, Upper Saddle River, New Jersey, 1997.

[Cop91] David Cope. *Computers and Musical Style.* A-R Editions, Madison, Wisconsin, 1991.

[Cop96] David Cope. *Experiments in Musical Intelligence.* A-R Editions, Madison, Wisconsin, 1996.

[Cop00] David Cope. *The Algorithmic Composer.* A-R Editions, Madison, Wisconsin, 2000.

[DP96] Ben Denckla and Patrick Pelletier. The technical documentation for *Rogus McBogus*, a MIDI library, 1996. http://www.media.mit.edu/hyperins/rogus/home.html.

[FS01] Mary Farbood and Bernd Schoner. Analysis and Synthesis of Palestrina-Style Counterpoint Using Markov Chains. In *Proceedings of the International Computer Music Conference*, Havana, Cuba, 2001.

[Ger99] Neil Gershenfeld. *The Nature of Mathematical Modeling.* Cambridge University Press, New York, 1999.

[Gir98] Girard Assayag. Computer assisted composition today. In *First Symposium on Music and Computers*, Corfu, Greece, October 1998. http://www.ircam.fr/equipes/repmus/RMPapers/Corfou98/.

[Gir99] Girard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon and O. Delerue. Computer Assisted Composition at Ircam: PatchWork and OpenMusic. *Computer Music Journal*, 23(3), 1999. http://www.ircam.fr/equipes/repmus/RMPapers/CMJ98.

[Gje88] Robert Gjerdingen. Concrete musical knowledge and a computer program for species counterpoint. In *Explorations in Music, the Arts, and Ideas: Essays in Honor of Leonard B. Meyer*. Pendragon Press, Stuyvesant, 1988.

[HI58] Lejaren Hiller and Leonard Isaacson. Musical composition with a high-speed digital computer. *Journal of the Audio Engineering Society*, 1958.

[Iwa92] Toshio Iwai. http://www.iamas.ac.jp/˜ iwai/artworks/music_insects.html, 1992.

[Jep31] Knud Jeppesen. *Counterpoint, the Polyphonic Vocal Style of the Sixteenth Century.* Dover Publications, New York, 1931.

[Jor98] Michael Jordan, editor. *Learning in Graphical Models.* MIT Press, Cambridge, Massachusetts, 1998.

[Kem80] Kemal Ebcioğlu. Computer counterpoint. In *Proceedings of the International Computer Music Conference*, New York, 1980.

[Kne01] Brian Knep. Blepco website: http://www.blep.com, 2001.

[Lev00a] Golan Levin. http://www.media.mit.edu/˜ golan, 2000.

[Lev00b] Golan Levin. Painterly interfaces for audiovisual performance. Master's thesis, MIT Media Lab, 2000.

[Lew83] David Lewin. An Interesting Global Rule for Species Counterpoint. *In Theory Only*, 6(8):19–44, 1983.

[Loh86] Henning Lohner. The UPIC System: A User's Report. *Computer Music Journal*, 10(4):42–49, 1986.

[Mac92] Tod Machover. Hyperinstruments. A Progress Report 1987-1991. Technical report, MIT Media Laboratory, 1992.

[Mac96] Tod Machover. *Brain Opera*, 1996.

[MM01] Tod Machover and Ariane Martins. Toy Symphony summary document, 2001.

[MR68] M. V. Mathews and L. Rosler. Graphical Language for the Scores of Computer-Generated Sounds. *Perspective of New Music*, 6(2):92–118, 1968.

[Ort01] Margaret Orth. Website, http://www.media.mit.edu/˜ morth, 2001.

[Pas01] Egon Pasztor, 2001. Personal communication.

[PG97] Joseph A. Paradiso and Neil Gershenfeld. Musical applications of electric field sensing. *Computer Music Journal*, 21(2):69–89, 1997.

[Ran78] Don Michael Randel. *Harvard Concise Dictionary of Music.* Harvard University Press, Cambridge, Massachusetts, 1978.

[Ric98] Peter Rice. Stretchable music: A graphically rich, interactive composition system. Master's thesis, MIT Media Lab, 1998.

[Rig94] Alexander Rigopulos. Growing music from seeds: Parametric generation and control of seed-based music for interactive composition and performance. Master's thesis, MIT Media Lab, 1994.

[Roa89] Curtis Roads, editor. *The Music Machine: Selected Readings from Computer Music Journal.* MIT Press, Cambridge, Massachusetts, 1989.

[Roa96] Curtis Roads. *Computer Music Tutorial.* MIT Press, Cambridge, Massachusetts, 1996.

[Ros88] Charles Rosen. *Sonata Forms.* W. W. Norton, New York, 1988.

[Row93] Robert Rowe. *Interactive Music Systems: Machine Listening and Composing.* MIT Press, Cambridge, Massachusetts, 1993.

[Sch89] William Schottstaedt. Automatic counterpoint. In Matthews and Pierce, editors, *Current Directions in Computer Music Research.* MIT Press, 1989.

[Sch01] Bernd Schoner, 2001. Personal communication.

[SS89] Felix Salzer and Carl Schachter. *Counterpoint in Composition.* Columbia University Press, New York, 1989.

[Sub99] Morton Subotnik. Creating music, http://www.creatingmusic.com, 1999.

[Tod96] Tod Machover and Teresa Marrin Nakra. Brain Opera website: http://brainop.media.mit.edu, 1996.

[Tod00] Tod Machover, Brian Knep, Mary Farbood, Peter Colao, Alex Westner, Freedom Baird. Future Music Blender, 2000.

[Tod01] Tod Machover et al. Hyperinstruments website: http://www.media.mit.edu/hyperins, 2001.

[Tru85] Barry Truax. The PODX System: Interactive Compositional Software for the DMX-1000. *Computer Music Journal*, 9(1):29–38, 1985.

[Wax95] David Waxman. Digital Theremins: Interactive Musical Experiences for Amateurs Using Electric Field Sensing. Master's thesis, MIT Media Lab, 1995.

[Wen01] Eric Wenger. Metasynth website, 2001. http://www.uisoftware.com/PAGES/ms_presentation.html.

[Xen71] Iannis Xenakis. *Formalized Music*. Indiana University Press, Bloomington, 1971.

[Xen86] Iannis Xenakis. Page from *Mycenae-Alpha* (1980). *Computer Music Journal*, 10(4), 1986.