Introduction to Computer Music
Week 14 Music Understanding and
the Future of Music Performance
Version 1, 29 Nov 2018

Roger B. Dannenberg

Topics Discussed: **Computer Accompaniment, Style classification Audio-to-Score alignment, Human Computer Music Performance**

1 Introduction

Let's take a step back from details and technical issues to think about where computer music is heading, and maybe where music is heading. This essay is a personal view, and I should state at the beginning that the ideas here are highly biased by my research, which is in turn guided by my own knowledge, interests, abilities, and talents.

I believe that if we divide computer music into a "past" and "future" that the "past" is largely characterized by the pursuit of sound and the concept of the instrument. Much of the early work on synthesis tried to create interesting sounds, model acoustic instruments or reproduce their sounds, and build interfaces and real-time systems that could be used in live performance in the same way that traditional musicians master and perform with acoustic instruments. The "future," I believe, will be characterized by the model of the musician rather than the instrument. The focus will be on models of music performance, musical interaction among players, music described in terms of style, genre, "feel," and emotion rather than notes, pitch, amplitude, and spectrum. A key technology will be the use of AI and machine learning to "understand" music at these high levels of abstraction and to incorporate music understanding into the composition, performance, and production of music.

Much of my research has been in the area of music understanding, and in some sense, the "future" is already here. Everything I envision as part of the future has some active beginnings well underway. It's hard to foresee the future as anything but an extension of what we already know in the present! However, I *have* seen visions of high-level languages enabling sophisticated music creation, of real-time control of sound with gestural control, and of all-digital music production. These visions took a long time to develop, and even if they were merely the logical progression of what was understood about computer music in the 70s or 80s, the results are truly marvelous and revolutionary. I think in another 20 or 30 years, we will have models of musicians and levels of automatic music understanding that make current systems pale by comparison.

The following sections describe a selection of topics in music understanding. Each section will describe a music understanding problem and outline a solution or at least some research systems that address the problem and show some possibilities for the future.

2 Computer Accompaniment

A basic skill for the musically literate is to read music notation while listening to a performance. Humans can follow quite complex scores in real-time without having previously heard the music or seen the score. The task of Computer Accompaniment is to follow a live performance in a score and to synchronize a computer performance. Note that the computer performs a pre-composed part, so there is no real-time composition involved but rather a responsive synchronization. There are now many accompaniment systems, at least in the literature, as well as a few commercial systems.

Accompaniment systems have two basic parts. First, a score-follower matches an incoming live performance signal (usually either audio or data taken directly from a piano-like keyboard using the MIDI protocol) to a machine-readable note-list or score. Second, an accompaniment algorithm uses score location to estimate tempo, position, and decide how to schedule upcoming notes or sound events in the accompaniment score.

The score-following component of Computer Accompaniment can be considered to have two sub-tasks as shown in Figure 1.

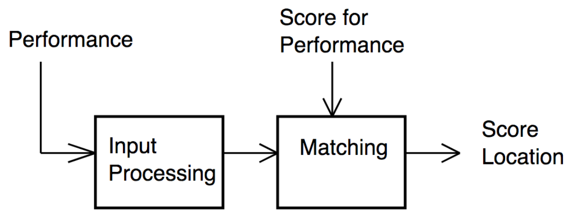


Figure 1: Score following block diagram.

2.1 Input Processing

The first task, the Input Processor, translates the human performance (which may be detected by a microphone or by mechanical sensors attached to keys) into a sequence of symbols, which typically correspond to pitches. With microphone input, the pitch must be estimated and quantized to the nearest semitone, and additional processing is useful to reduce the number of false outputs that typically arise.

2.2 Matching

The Matcher receives input from the Input Processor and attempts to find a correspondence between the real-time performance and the score. The Matcher has access to the entire score before the performance begins. As each note is reported by the Input Processor, the matcher looks for a corresponding note in the score. Whenever a match is found, it is output. The information needed for Computer Accompaniment is just the real-time occurrence of the note performed by the human and the designated time of the note according to the score.

Since the Matcher must be tolerant of timing variations, matching is performed on sequences of pitches only. This decision makes the matcher completely time-independent. One problem raised by this pitch-only approach is that each pitch is likely to occur many times in a composition. In a typical melody, a few pitches occur in many places, so there may be many candidates to match a given performed note.

The matcher described here overcomes this problem and works well in practice. Many different matching algorithms have been explored, often introducing more sophisticated probabilistic approaches, but my goal here is to illustrate *one* simple approach. The matcher is derived from the dynamic programming algorithm for finding the longest common subsequence (LCS) of two strings. Imagine starting with two strings and eliminating arbitrary characters from each string until the the remaining characters (subsequences) match exactly. If these strings represent the performance and score, respectively, then a common subsequence represents a potential correspondence between performed notes and the score (see Figure 2). If we assume that most of the score will be performed correctly, then the longest possible common subsequence should be close to the"true" correspondence between performance and score.

In practice, it is necessary to match the performance against the score as the performance unfolds, so only an initial subsequence of the entire performance is available. This causes an interesting anomaly: if a wrong note is played, the LCS algorithm will search arbitrarily far ahead into the score to find a match. This will more than likely turn out not to be the best match once more notes are played, so the algorithm will recover. Nevertheless, being unreasonably wrong, even momentarily, causes problems in the accompaniment task. To avoid skipping ahead in the score, the algorithm is modified to maximize the number of corresponding notes minus the number of notes

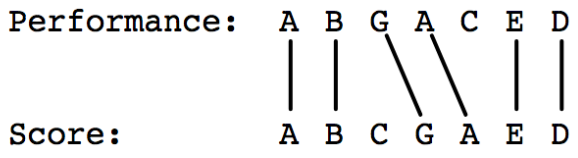


Figure 2: Illustration of the longest common subsequence between a performed sequence and a score sequence.

skipped in the score. Other functions are possible, but this one works well: the matcher will only skip notes when their number is offset by a larger number of matching notes.

The Matcher is an interesting combination of algorithm design, use of heuristics, and outright ad-hoc decisions. Much of the challenge in designing the matcher was to model the matching problem in such a way that good results could be obtained efficiently. In contrast to the other accompaniment systems emerging at the same time, this matcher designed by the author could easily match sequences of 20 or more pitches in real time, even on very slow microprocessors of the day, making it very tolerant of errors. (Consider a naive algorithm that tests all different ways to align performed notes to score notes. The number of possible alignments grows exponentially with the number of notes, so sequences of length 20 performed notes plus 20 score notes have about 10^{12} possible alignments, and even modern computers cannot do this.)

Polyphonic matchers have also been explored. One approach is to group individual notes that occur approximately simultaneously into structures called *compound events*. A single isolated note is considered to be a degenerate form of compound event. By modifying the definition of "matches," the monophonic matcher can be used to find a correspondence between two sequences of compound events. Another approach processes each incoming performance event as it occurs with no regard to its timing relationship to other performed notes. It is important in this case to allow notes within a chord (compound event) in the score to arrive in any order. (Note that the LCS algorithm disallows reordering.) The resulting algorithm is time-independent.

The Matcher performs a fairly low-level recognition task where efficiency is important and relatively little knowledge is required. When matches are found, they are output for use by an Accompaniment Performance subtask, which uses knowledge about musical performance to control a synthesizer. Several systems have been implemented based on these techniques, and the results are quite good. (You can find a video demonstration at http://www.cs.cmu.edu/~rbd/videos.html.)

The Matcher has also been extended to handle trills, glissandi, and grace notes as special cases that would otherwise cause problems, and this version has been used successfully for several concerts. A commercial Computer Accompaniment system, SmartMusic, is derived directly from the author's work and has been used mainly for music education since the 1990's. Other accompaniment systems include Tonara, Antescofo, and Music Plus One.

3 Style Classification

Many computer music applications can benefit from higher-level music understanding. For example, interactive performance systems are sometimes designed to react to higher-level intentions of the performer. Unfortunately, there is often a discrepancy between the ideal realization of an interactive system, in which the musician and machine carry on a high-level musical discourse, and the realization, in which the musician does little more than trigger stored sound events. This discrepancy is caused in part by the difficulty of recognizing high-level characteristics or style of a performance with any reliability.

Our experience has suggested that even relatively simple stylistic features, such as playing energetically, playing lyrically, or playing with syncopation, are difficult to detect reliably. Although it may appear obvious how one might detect these styles, good musical performance is always filled with contrast. For example, energetic performances contain silence, slow lyrical passages may have rapid runs of grace notes, and syncopated passages may have a variety of confusing patterns. In general, higher-level musical intent appears chaotic and unstructured when presented in low-level terms such as MIDI performance data.

Machine learning has been shown to improve the performance of many perception and classification systems (including speech recognizers and vision systems). This section describes a machine-learning-based style classifier for music. Our initial problem was to classify an improvisation as one of four styles: lyrical, frantic, syncopated, or pointillistic (the latter consisting of short, well-separated sound events). We later added the additional styles: blues, quote (play a familiar tune), high, and low. The exact meaning of these terms is not important. What really matters is the ability of the performer to consistently produce intentional and different styles of playing at will.

The ultimate test is the following: Suppose, as an improviser, you want to communicate with a machine through improvisation. You can communicate four different tokens of information: lyrical, frantic, syncopated, and pointillistic. The question is, if you play a style that you identify as frantic, what is the probability that the machine will perceive the same token? By describing music as a kind of communication channel and music understanding as a kind of decoder, we can evaluate style recognition systems in an objective scientific way.

It is crucial that this classification be responsive in real time. We arbitrarily constrained the classifier to operate within five seconds.

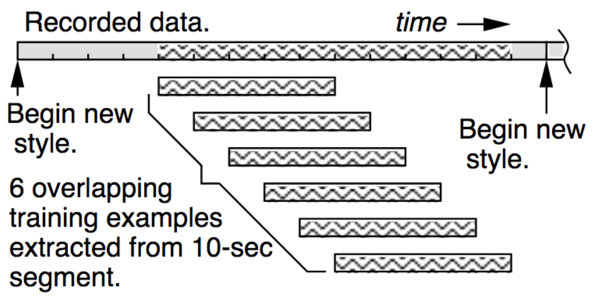

Recorded data.
time
Begin new style.
6 overlapping training examples extracted from 10-sec segment.
Begin new style.

Figure 3: Training data for a style recognition system is obtained by asking a human performer to play in different styles. Each style is played for about 15 seconds, and 6 overlapping 5-second segments or windows of the performance are extracted for training. A randomly selected style is requested every 15 seconds for many minutes to produce hundreds of labeled style examples for training.

Figure 3 illustrates the process of collecting data for training and evaluating a classifier. Since notions of style are personal, all of data is provided by one performer, but it takes less than one hour to produce enough training data to create a customized classifier for a new performer or a new set of style labels. The data collection provided a number of 5-second "windows" of music performance data, each with a label reflecting the intended style.

The data was analyzed by sending music audio from an instrument (a trumpet) through an IVL Pitchrider, a hardware device that detects pitch, note-onsets, amplitude, and other features and encodes them into MIDI. This approach was used because, at the time (1997), audio signal processing in software was very limited on personal computers, and using the MIDI data stream was a simple way to pre-process the audio into a useful form. We extracted a number of features from each 5-second segment of the MIDI data. Features included average pitch, standard deviation of pitch, number of note onsets, mean and standard deviation of note durations, and the fraction of time filled with notes (vs. silence).

Various style classifiers were constructed using rather simple and standard techniques such as linear classifiers and neural networks.

While attempts to build classifiers by hand were not very successful, this data-driven machine-learning approach worked very well, and it seems that our particular formulation of the style-classification problem was not even very difficult. All of the machine-learning algorithms performed well, achieving nearly perfect classification in the 4-class case, and around 90% accuracy in the 8-class case. This was quite surprising since there seems to be no simple connection between the low-level features such as pitch and duration and the high-level concepts of style. For example the "syncopation" style is recognizable from lots of notes on the upbeat, but the low-level features do not include any beat or tempo information, so evidently, "syncopation" as performed happens to be strongly correlated with some other features that the machine learning systems were able to discover and take advantage of. Similarly, "quote," which means "play something familiar" is recognizable to humans because we might recognize the familiar tune being quoted by the performer, but the machine has no database of familiar tunes. Again, it must be that playing melodies from memory differs from free improvisation in terms of low-level pitch and timing features, and the machine learning systems were able to discover this.

You can see a demonstration of this system by finding "Automatic style recognition demonstration" on `http://www.cs.cmu.edu/~rbd/videos.html`. Since this work was published, machine learning and classifiers have been applied to many music understanding problems including genre recognition from music audio, detection of commercials, speech, and music in broadcasts, and the detection of emotion in music.

4 Audio-to-Score Alignment

The off-line version of score following is called *audio-to-score alignment*. *Off-line* means that we do not need to output alignment or location data in real-time. Instead, we get to observe the entire audio stream and the entire score before computation begins. This means that we can look ahead into the data for both audio and score. As a result, ambiguous information can be resolved by finding clearer matches happening both earlier and later.

This turns out to be a very big advantage. Earlier we said that computer accompaniment system work in real-time and perform quite well, which is true, but that is partly because accompaniment systems tend to be built for monophonic instruments (those that produce one note at a time). Dealing with polyphony (multiple notes) or something as complex as an orchestra or even a popular music recording poses a very challenging problem to detect notes accurately. Our ability to do real-time score following is greatly diminished when the input consists of *polyphonic music audio*.

These problems are compensated for by the additional power of using look-ahead in a non-real-time audio-to-score alignment system. This section outlines the basic principles that are used and then presents some applications and opportunities for this technology.

4.1 Chroma Vectors and Similarity

In a monophonic score follower or polyphonic MIDI score follower, we normally use note pitch as the "feature" that we match between performance and score. With polyphonic audio, extracting pitches is difficult and unreliable, but another feature, called the *chroma vector*, has been shown to be quite useful and robust. The *chroma vector* is related to the spectrum. You might think the spectrum would be a good feature itself since it requires very little interpretation and certainly contains all the necessary information about the performance. The problem with the spectrum is that just by playing a little louder (brighter) or changing the microphone location, we can get fairly large shifts in the spectrum. Also, to align audio recordings to symbolic representations, we would need a way to determine the spectrum from symbolic representations, i.e. we need to synthesize a band or orchestra in a way that accurately reproduces the spectrum of the recorded version. Our symbolic descriptions do not contain enough information and our synthesis methods are not good enough to reproduce spectra.

The chroma vector divides the spectrum into semitones, i.e. we look at frequency bands for C1, C#1, D1, ... B1, C2, C#2, D2, ... B2, and all the way to the top octave. Then, we take all the magnitudes in the spectrum for pitch class C, namely the frequency bands for all the octaves of C: C1, C2, C3, etc., and add the magnitudes together to get one number we will call C. Then we do the same for C-sharp: C#1, C#2, C#3, etc., and then D1, D2, D3, etc., and all the other pitch classes. The result is a vector of 12 elements: C, C#, D, D#, E, F, F#, G, G#, A, A#, B. To finish processing, the chroma vector is often normalized to have a sum-of-squares magnitude of one so that scaling the amplitude of the source signal has no effect on the chroma vector.

You can think of chroma vectors as a summary or projection of the detailed spectrum into a simpler representation. A sequence of chroma vectors over time is analogous to a spectrogram and we call it a chromagram.

Because this projection averages a lot of information and totally collapses octave information, it tends to be a robust indicator of melody and harmony, but at the same time, chroma vectors tend to have little to no information about timbre, so you get roughly the same chroma vector whether you play a chord on a piano or with an orchestra or synthesize the sound.

4.2 The Similarity Matrix and Dynamic Time Warping

Now that we have a robust representation of polyphonic music performances, we can compare a performance to a score. To get a chromagram from a score, we can either synthesize the score and compute the chromagram from audio, or we can make a direct "chroma-like" representation by simply representing any note with pitch class C as the vector 100000000000, C# as 010000000000, and so on. We could scale these vectors according to loudness indicated in the score. For chords, we just add the vectors from the notes of the chord. After normalizing the sums, we get something compatible with and similar to chromagrams.

To compare a performance chromagram to a score chromagram, we typically construct a full "similarity matrix" that compares every chroma vector in the performance to every chroma vector in the score. (Actually, we only have to compute 1/2 of this matrix since we use a commutative similarity function.) One way to compare vectors x and y is to use their Euclidean distance (distance is the opposite of similarity, so when we say "maximize similarity" it is equivalent to saying "minimize distance").

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Figure 4 illustrates the similarity matrix comparing a MIDI-based chromagram (horizontal) to a chromagram of the recording of the same song by the Beatles (vertical). Notice that there is a dark narrow path near the diagonal from lower left (0, 0) to the upper right. Since dark represents high similarity (smaller distance), this is the path where the MIDI-based chromagram best aligns with the score-based chromagram.

To find the alignment path automatically, we look for a path from lower left corner to upper right corner in the similarity matrix that minimizes the sum of distances along the path. This is a standard problem called dynamic time warping and uses the same general dynamic programming approach used in the longest common subsequence (LCS) problem. Once the best path is computed, we have a mapping from each score position to the corresponding score location. (Technically, the MIDI file used in this example is essentially a note list and not a score in the sense of common practice music notation. It might be better to say this is *signal-to-symbol* alignment, but symbol is a vague term, so it is common to use *score* to refer to any representation of notes or sound events with some indication of timing and properties such as pitch.)

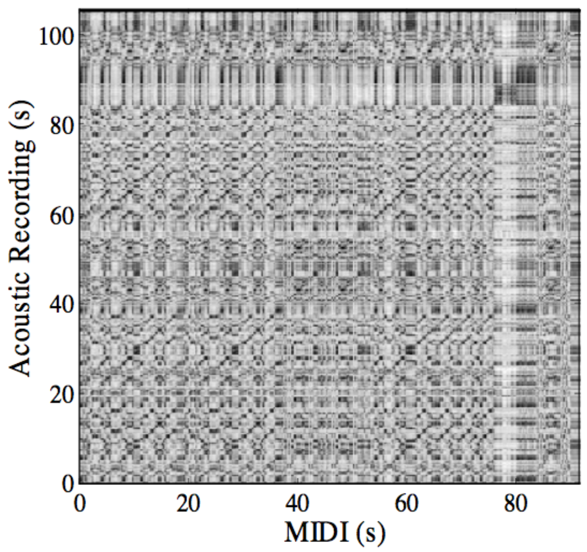
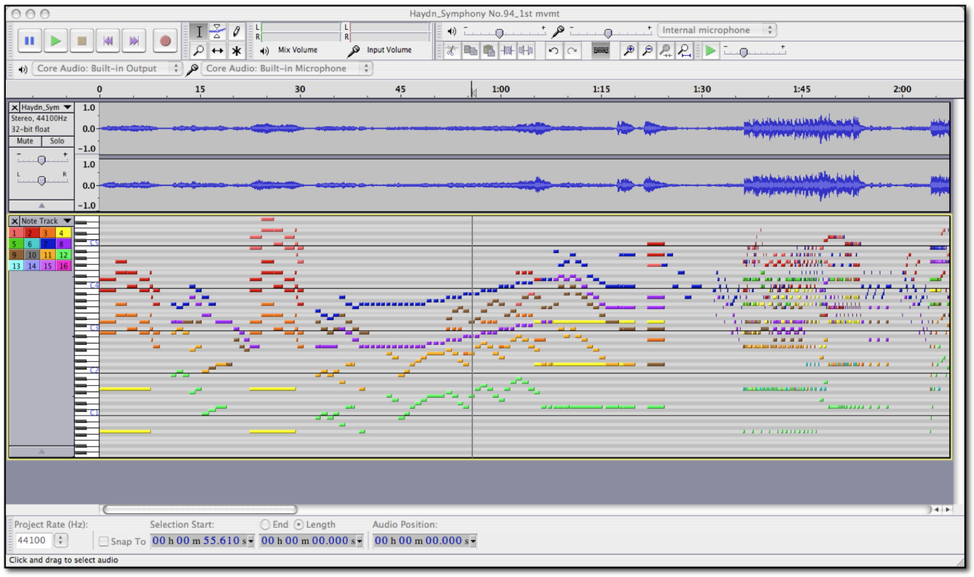


Figure 4: A similarity matrix illustrating data used for audio-to-score alignment. Darker means more similar.

4.3 Applications of Audio-to-Score Alignment

One application of audio-to-score alignment is to make navigation of audio easier. Figure 5 shows an experimental version of Audacity that was extended by the author to “warp” the tempo of a MIDI file to align to an audio recording. This is *not* simply a synthesized version of the MIDI file. The MIDI file came from a MIDI library, and the audio is a commercial recording of a Haydn symphony. By aligning these two different representations, we can immediately see much more of the music structure. For example, if one were looking for the places where the whole orchestra plays a long note together, it is much easier to find those spots with the MIDI file as a reference.



Project Rate (Hz): 44100
Snap To
Selection Start: 00 h 00 m 55.610 s
End / Length: 00 h 00 m 00.000 s
Audio Position: 00 h 00 m 00.000 s

Figure 5: Alignment of MIDI in a piano-roll display to audio in an experimental version of the Audacity audio editor.

In studio recordings of new music, scores are often prepared using music notation software, so it is often the case that machine-readable common practice notation is available. In principle, that information could be used to display music notation to aid navigation in a future audio editor.

Alignment has also been used in systems to identify “cover songs,” which are versions of popular songs performed by someone other than the original recording artists. Another interesting possibility is to use alignment to provide meta-data or labels for audio. This data can then be used to train machine learning systems to identify chords, pitches, tempo, downbeats and other music understanding tasks.

5 Human Computer Music Performance

Another interesting application for computers in music is live performance of popular music such as rock, jazz, and folk music. By and large, “popular” music has a definite form and steady tempo, but players often have the possibility to interpret the score rather freely. A consequence is that players synchronize more by beat than by note-by-note score following. In fact, many times there is no note-level score for guitars or drums, and even vocalists sing expressively without strictly following notated rhythms.

Computers are often used in such music, but computers play sequences at a strict tempo, and human musicians are forced to play along and synchronize to the computer (or even a digital audio recording, sometimes referred to as a *backing track*). This type of performance is restrictive to musicians, who often end up focused more on keeping with the computer than playing musically, and it is usually not enjoyable to have a completely inflexible tempo.

As an alternative, imagine a system where computers could follow humans, much like in computer accompaniment systems, only now with popular beat-based music. One could imagine a computer using beat-tracking algorithms to find the beat and then synchronize to that. Some experimental systems have been built with this model, but beat tracking is not very reliable. An alternative is simple foot tapping using a sensor that reports the beat to the computer.

In addition to beats, the computer needs to know when to start, so we can add sensors for giving cues to start a performance sequence. Just as human conductors give directions to humans: play something now, stop playing, play louder or softer, go back and repeat the chorus, etc., we can use multiple sensors or different gestures to give similar commands to computer performers.

We call this scenario Human-Computer Music Performance (HCMP), with terminology that is deliberately derived from Human-Computer Interaction (HCI) since many of the problems of HCMP are HCI problems: How can computers and humans communicate and work cooperatively in a music performance? The vision of HCMP is a variety of compatible systems for creating, conducting, cueing, performing, and displaying music. Just as rock musicians can now set up instruments, effects pedals, mixers, amplifiers and speakers, musicians in the future should be able to interconnect modular HCMP systems, combining sensors, virtual performers, conducting systems, and digital music displays as needed to create configurations for live performance.

Some of the interesting challenges for HCMP are:

- Preparing music for virtual players: Perhaps you need a bass player to join your band, but you only have chord progressions for your tunes. Can software write bass parts in a style that complements your musical tastes?
- Sharing music: Can composers, arrangers, and performers share HCMP software and content? How can a band assemble the software components necessary to perform music based on HCMP?
- Music display systems: Can digital music displays replace printed music? If so, can we make the displays interactive and useful for cueing and conducting? Can a band leader lead both human and virtual performers through a natural music-notation-based interface?

Clearly, music understanding is important for HCMP systems to interact with musicians in high-level musical terms. What if a band leader tells a virtual musician to make the music sound more sad? Or make the bass part more lively? HCMP is an example of how future music systems can move from a focus on instruments and sounds to a focus on music performance and musical interaction.

6 Summary

Music understanding encompasses a wide range of research and practice with the goal of working with music at higher levels of abstraction that include emotion, expressive performance, musical form and structure, and music recognition. We have examined just a limited selection of music understanding capabilities based on work by the author. These include:

- computer accompaniment—the computer as virtual interactive and responsive musical accompanist,
- automatic style recognition—the detection of different performance styles using machine classifiers,
- audio-to-score alignment—the ability to match sounds to symbolic representations of music event sequences, and
- Human-Computer Music Performance—an evolving practice of bringing virtual performers into collaboration with humans in live performance.

All of these tasks rely on new techniques for processing musical information that we call *music understanding*.

7 Acknowledgments

Thanks to Shuqi Dai and Sai Samarth for editing assistance.

Portions of this work are taken almost verbatim from *Music and Computers, A Theoretical and Historical Approach* (http://sites.music.columbia.edu/cmc/MusicAndComputers/) by Phil Burk, Larry Polansky, Douglas Repetto, Mary Robert, and Dan Rockmore. Other portions are taken almost verbatim from *Introduction to Computer Music: Volume One* (http://www.indiana.edu/~emusic/etext/toc.shtml) by Jeffrey Hass. I would like to thank these authors for generously sharing their work and knowledge.