Roger B. Dannenberg
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 USA
dannenberg@cs.cmu.edu

Music Representation Issues, Techniques, and Systems

Music offers a challenging array of representation problems. As an art form, music is distinguished by the presence of many relationships that can be treated mathematically, including rhythm and harmony. There are also many nonmathematical elements such as tension, expectancy, and emotion. Music can contain symbolic or structural relationships existing within and between the dimensions of pitch, time, timbre, harmony, tempo, rhythm, phrasing, and articulation. A further source of complexity is that "music" can mean printed notation, performance (instrument control) information, or resulting sounds. Finally, music evolves with every new composition. There can be no "true" representation just as there can be no closed definition of music. These elements combine to make music representation a rich field of study.

Computers allow (and require) a formal approach to the study of music representation. Computer programs demand that every detail of a representation be precisely specified, and the resulting precision allows experimentation and testing of new representations. The knowledge we gain from these experiments is useful in music theory and in music understanding by computer. Computers also allow the representation of dynamic, responsive, or interactive compositions, a concept that was hard to imagine before computing. Better representations also lead to more powerful human-computer interfaces and tools.

This is an overview of some of the issues and techniques that arise in studies of music representation. No attempt has been made to be exhaustive, so the reader is referred to the references and the other articles in this issue of *Computer Music Journal* for more information. In particular, I will omit any discussion of audio representations and transforms used in signal processing (see De Poli, Piccialli, and Roads, 1991), although this is an important part of music representation.

Computer Music Journal, 17:3, pp. 20–30, Fall 1993,
© 1993 Massachusetts Institute of Technology.

Levels of Representation

Musicians deal with many levels of abstraction in music. If a conductor says, "play the downbeat with more conviction," he or she is referring to music structure (a downbeat) and emotional content in the same sentence. It is convenient to think of musical representations at different levels, ranging from the highly symbolic and abstract level denoted by printed music to the nonsymbolic and concrete level of an audio signal. Performance information is an intermediate level. We must consider these levels because there is rarely a unique conversion between any two of them. Each level contains at least some information not available in other levels. In general, there is great interest and value in performing automatic (partial) conversions between levels (Katayose et al. 1989), as in optical music recognition or music transcription.

Hierarchy and Structure

Early computer music systems, especially those intended for music synthesis, represented music as a simple sequence of notes. The Music V score language (Mathews 1969) is a good example. This approach is straightforward, but it makes it difficult to encode structural relationships between notes. For example, applying an amplitude envelope to a group of notes is a tricky operation in Music V and its successors.

MIDI is similar in that it has no mechanisms for describing new structural relationships. However, MIDI has a number of predefined structures. There are 16 channels, which effectively form 16 groups of notes. Each group has a set of controllers such as volume and pitch-bend. This gives MIDI a limited two-level structure.

Many researchers have investigated hierarchical representations. For example, Buxton et al. (1978) describe a system in which "events" can be notes or sequences of events, allowing arbitrarily nested structures. One advantage of hierarchically structured descriptions of music is that transformations such as tempo or pitch can be applied to aggregates of musical objects. In general, hierarchy is a way of representing structure, and it should be no surprise that many composition languages support the notion of hierarchy.

A single hierarchy scheme is still limiting because music often contains multiple hierarchies. Consider beams and phrase marks found in music notation. Notes can have several beams, and beams are often broken to help subdivide rhythms, so a multilevel beam hierarchy arises naturally. Phrase markings, ties, and slurs form another multilevel hierarchy that is completely separate from beaming. Consider some other possible hierarchies: voices (orchestra, strings, violins, first violins, solo violin), sections (movement, section, measure), phrases (which may cut across sections), and chords, all of which are ways of grouping and structuring music. A single hierarchy system is inadequate to represent all these concepts at the same time.

Brinkman (1985) and Dannenberg (1990) describe representations that support multiple hierarchies through named links relating musical events or objects, and through explicit objects that represent instances of hierarchies. The NeXT Music Kit and HyTime support a more indirect approach in which events may be given "tags" to indicate grouping. For example, all notes under a phrase marking may receive the tag "phrase257."

Extensibility

If musical information were well understood and fixed, music representation would be a much simpler problem. In reality, we do not know all there is to know, and the game is constantly changing. For both of these reasons, it is important that music representations allow extensions to support new concepts and structures.

An example of extensibility is seen in Music V, in which the orchestra language allows the composer to assemble unit generators into a collection of synthesis algorithms for synthesizing notes. The most interesting aspect of this arrangement is that every synthesis algorithm determines both the number and meaning of control parameters. The score language allows each note to have a corresponding number of parameters. This is in contrast to MIDI, in which the parameters of voice, pitch, and velocity are fixed. MIDI could be extended fairly simply by designating some MIDI Control Change messages to set additional parameters. For example, Control Numbers 20 through 31 might be dedicated as note parameters P1 through P12, which would then be sent as a prefix to each note that uses extra parameters.

Music V influenced many other systems, and a frequent modification has been the use of named and typed properties. Examples include event-lists (Decker and Kendall 1984), item-lists (Dannenberg 1990), PLA (Schottstaedt 1983), SMDL (Newcomb 1991), and MODE (Pope 1991, 1992), to name just a few. In these systems, note parameters are represented as a set of properties, usually consisting of name: value pairs, for example, `[Pitch: C4, Instrument: Violin, Duration: 1 Quarter, Dynamic: Forte]`.

One of the disadvantages of extensible structures such as these is the difficulty of providing semantic extensions to deal with additional parameters. Many of the most flexible representation systems are part of a programming environment, and it is up to the composer to implement interpretations for new parameters by programming.

If programming is necessary, one would like at least to write one global description of what it means, for example, interpret a "vibrato" property, rather than modifying many instrument descriptions. Multiple inheritance schemes in object-oriented and frame-based systems have the potential to support such extensions, but in practice, real systems often support certain classes of extensions and break down on others. Designing representations that can be extended with both raw information and semantic interpretations of the data are still an area of research (Böcker and Mahling 1988).

Even fairly closed representations such as MIDI are full of interpretation problems. For example, MIDI specifies how to encode key velocity and pitch bend, but it does not specify quantitatively how these parameters should affect sounds. Should key velocity translate into decibels? Should it map to some subjective loudness measure (Martens 1985)? Should it map uniformly to the dynamic range of a corresponding acoustic instrument? These are all conceivably valid interpretations. The state of the art is that manufacturers provide a set of suggestive defaults, and composers either adjust synthesizer patches to obtain uniform responses or adjust their MIDI data separately for each voice to get the desired results.

Pitch

The seemingly simple concept of pitch is in practice fairly complex. This is because pitch exists as an acoustic property (repetition rate), a psychological percept (perceived pitch), and an abstract symbolic entity relating to intervals and keys. Psychologists have measured the relationship between the perceived pitch or interval size and the fundamental frequency of a tone. The relationship depends on many variables (Krumhansl 1991). A consequence is that a concept as basic as the octave can have many meanings.

At the symbolic level, pitches participate in mathematical structures (Balzano 1980; Shepard 1982). To name a pitch, we need to specify a scale, octave, scale step, and alterations, for example, the pitch C-sharp in the fourth octave of an equal-tempered scale based on A440. Descriptions like this assume that scales establish finite sets of pitch choices. In practice, performers vary pitch for a variety of reasons; soloists often play a little sharp for added salience, performers may retune dynamically to eliminate beating, and ensembles exhibit pitch variations producing chorus effects.

These are more problems of models, or deciding what to represent, rather than how to represent it. I mention these problems here to indicate the richness required of any general pitch representation.

Tempo, Beat, Duration, and Time

The interaction between real time, measured in seconds, and metrical time, measured in beats, is frequently addressed in music representation schemes. Abstractly, there is a mathematical function that maps beat numbers to time and an inverse function that maps time to beats. (Some special mathematical care must be taken to allow for music that momentarily stops, instantaneously skips ahead, or is allowed to jump backward, but we will ignore these details.)

One practical representation of the beat-to-time function is the set of times of individual beats. For example, a performer can tap beats in real time, and a computer can record the time of each tap. This produces a finitely sampled representation of the continuous mapping from beats to time, which can be interpolated to obtain intermediate values. MIDI Clocks, which are transmitted at 24 clocks per quarter note, are an example of this approach.

The Mockingbird score editor (Maxwell and Ornstein 1984) interface also supported this idea. The composer would play a score, producing a piano-roll notation. Then, downbeats were indicated by clicking with a mouse at the proper locations in the piano-roll score. Given this beat information, Mockingbird could then derive a symbolic score.

Another popular technique is the "tempo curve" (Rogers, Rockstroh, and Batstone 1980; Jaffe 1985), a function from beat number to tempo, where tempo is the instantaneous ratio of beats per second, or equivalently, the first derivative of the function from time to beats. (A related function is tempo as a function of time rather than beats.) The tempo curve is a nice abstraction for mathematical specification. For example, logarithmic tempo curves have been found to produce smooth tempo changes that are musically plausible (at least better than linear ones).

There are some interesting issues of numerical precision created by considerations of tempo and rhythm. One is that rounding errors in note durations must not cause noticeable errors. For example, suppose a system represents duration in milliseconds. If a quarter note has a duration of 1,000 msec, then eighth-triplets will have a duration (after rounding) of 333 msec. A series of 30 triplets played against 10 quarters will be skewed by 10 msec. Some early computer music systems, undoubtedly feeling the pressure of memory limitations, developed encodings based on rational arithmetic to avoid round-off error altogether. This proved not to solve all the numerical problems.

Imagine a system for film scoring, in which absolute timing is critical. Even if rational arithmetic is used to avoid round-off error in computing beats, the conversion from beats to seconds is subject to round-off error. If we arbitrarily tolerate 1 msec of error per hour, then we need a precision of at least 22 bits, and more if many small durations are summed. Even 32-bit floating-point numbers are not suitable for such calculations.

One advantage of computers is that they can perform calculations using musical data. In the case of tempo, a common task is to fill a certain amount of time with a certain number of beats. Extensions of this idea include synchronizing certain beats with actions on film or video and manipulating multiple changing tempi while still achieving synchronization at certain points. This is essentially a constraint satisfaction problem, yet few representations for time and tempo can express constraints.

Before concluding this discussion of tempo and time, we should observe that mathematically elegant representations do not necessarily guarantee musical validity. Tempo transformations make mathematical sense, but do not always produce musical results (Desain and Honing 1991a). Research is needed to find transformations that retain musical nuance, and it has been suggested that explicit structure is essential (Desain and Honing 1991b; Bilmes 1992). In the programming language area, I have suggested that transformations can be encapsulated into behavioral abstractions that support transformations (Dannenberg 1989), and similar notions have been proposed by Desain and Honing (1992) and Scaletti (1992). For example, the operation to stretch a trill is very different from the operation to stretch a melody. Here again, good representations can support the goal of musical transformations.

Timbre

With many aspects of music, we know what to represent, and the issue is how to represent it. With timbre, we are still learning what to represent. My explanation is that, starting with sound, we picked out the two things we understood, pitch and amplitude, and called everything else timbre. So timbre is by definition that which we cannot explain. As aspects of timbre are isolated and understood, such as spatial location and reverberation, these components come to be regarded separately, leaving timbre as impenetrable as ever.

Taking a less cynical view, real progress has been made toward timbre representation. The classic studies by David Wessel (1979) and John Grey (1975) used multidimensional scaling and refined the notion of timbre space. Although these studies represented timbre in terms of perceptual dimensions, others have represented timbre in terms of control dimensions, such as the set of parameters needed for a particular synthesis algorithm.

The current practice is usually to represent timbre by name or number. Real timbre space has so many dimensions that it is often preferable to locate and name interesting points in the space. MIDI program change messages and the concept of "instrument" found in many software synthesis systems are examples of this approach.

Music Notation

Music notation is a rich source of representational problems. One of the major problems is that music notation is not just a mechanical transformation of performance information. Performance nuance is lost in going from performance to notation, and symbolic structure is lost in the translation from notation to performance. There is evidence of strong relationships between structure and performance nuance (Palmer 1989; Clark 1991), but it seems unlikely that the relationships are strong enough to guarantee lossless transformations.

It seems that music notation rules are made to be broken. Donald Byrd's thesis contains an excellent discussion of notational variations and the impossibility of automatic music notation layout (Byrd 1984). Especially interesting is the fact that his examples, numbering about 100, all come from established, traditional composers and publishers. Two examples are the Henle edition of Chopin's Nocturne, op. 15, no. 2, in which one note head serves both as a regular and a triplet sixteenth, and the Peters/Sauer edition of Brahm's Capriccio, op. 76, no. 1, in which a written dotted half note has an actual duration of 11 sixteenths!

Another issue is that notation is visual and leaves open many layout choices. Thus, notation is partly a graphics design task, and manual layout is necessary. Representations for music notation usually include the representation of musical structure, such as key and time signatures, bar lines, beams, slurs, etc., and also graphical information such as staff position, stem direction, and graphical positioning. In many cases, layout information is implied by musical structure, but manual overrides are sometimes necessary.

The need to represent visual layout information, some of which cannot be generated automatically, has important implications for music notation systems. One is that copying parts from a score cannot be a fully automatic process, a problem that commercial notation systems have largely ignored. Most music notation programs allow the user to convert a score file into a set of parts files. However, complex scores will inevitably have many details that must be manually specified for each part, including page breaks, cues, and other notation that would not appear in the score. A problem occurs if changes are made to the score after parts have been manually corrected. Either the changes must be manually transferred to each part, or alternatively, the parts can be regenerated automatically. In the first case, there is no support for consistency between the score and the parts; in the second case, the manual adjustments to each part are lost and must be redone by hand.

A solution to this problem with parts is to represent parts as views on the score (Dannenberg 1986). A view of a data structure contains a subset of the information in the data structure and sometimes provides alternative or additional data to what is in the data structure. The idea is to keep shared data in one place so that a change in the score will automatically be propagated to the parts, and part-specific layout information can be maintained for each part (view).

Views have many other potential applications. Alternative notations for a single piece of music can be represented as views (Buxton et al. 1979). Even repeated sections of music might be represented as views. For example, it is common notational practice to indicate that a section of music is to be repeated. Often, each repetition has its own ending, or notation such as "tacet the first time." This is essentially a view mechanism in operation; each repetition shares most of the underlying information, but there are a few local alterations. If repeats are to be notated once but performed twice, views offer a mechanism for representing performance variations between the first and second repeat.

Carrying this to an extreme, we can imagine views as a general mechanism to represent structure in music. Imagine a representation in which motives are represented only once, and each occurrence is some kind of view, perhaps with local alterations and transformations, of the motive. Such a scheme would quickly lead to many interesting problems. If a note in a view is edited, and then the original note in the motive is deleted, should the view's note be deleted as well? Can the user control such decisions? Can views be nested? These issues have much in common with representational schemes proposed for artificial intelligence, programming languages, and databases.

One important feature of notation is that events are represented left-to-right in increasing time order, but the position is not in exact proportion to time or to beat number. An interesting situation arises when multiple tempi are present simultaneously, especially when the tempi are not related by simple fractions such as a 6/8 measure in the time of a 2/4 measure. In the more complicated situations, beat and tempo information must be combined to form absolute time, which then becomes the basis for left-to-right layout. I know of no music notation systems that can even represent this situation, much less perform reasonable layout. A similar situation would arise if absolute time notations for film, animation, or tape were to be graphically aligned with conventional music notation.

Composers would like to notate not only conventional notation but also new graphical notations. In some ways, any good computer-assisted design or graphics package could support new notation, but it would be nice to have the graphics closely tied to underlying musical structures. Graphical editing should have a corresponding effect on musical parameters, which might then control a music synthesizer. An interesting proposal along these lines was made by

Daniel Oppenheim (1987), and examples of (noneditable) graphics representations have been presented by Dannenberg, Fraley, and Velikonja (1991), Brinkman (1991), Waters and Ungvary (1990), and Schottstaedt (1983).

Another approach to extensible notation is that of Assayag and Timis (1986), who developed an elaborate PostScript library to assist in producing complex musical graphics. The library manages constraints among connected objects so that, for example, beams can be made to terminate at the end of a particular stem and other stems can be made to touch the beam but not go beyond it. The MusScribe notation system (Hamel 1988) also abandons the issue of maintaining consistency between graphical notation and music structure by providing a graphical editor that is optimized for music notation. For example, note heads snap to staff lines and spaces, but horizontal positioning is set manually.

Continuous and Discrete Data

Musical information can be classified as either continuous or discrete. Continuous information changes over time (or perhaps as a function of other variables) and is typically represented by digital sampling, by splines (including piecewise linear functions), or by arbitrary mathematical functions. In contrast to continuous data that fill time intervals, discrete information usually represents events at a point in time. A MIDI Note On event is an example. It is sometimes advantageous to represent discrete events using time intervals rather than points (Honing 1992). There is a natural correspondence to musical notes, which have duration as well as starting time. Also, intervals and sequences of intervals can be appended.

Continuous information is found in the signals of Music V and also in the Groove system (Mathews and Moore 1970). Groove was a real-time, multichannel, continuous information recorder and manipulator; it is highly recommended for further study.

Music representation systems have generally had a difficult time integrating continuous and discrete data. For example, many modern sequencers support smooth changes in controls such as volume, but functions of time are not first-class entities that can be combined with mathematical operations. Another issue is the use of continuous data as parameters to discrete events. If a continuous function is to be used as a pitch contour, should each note sample the function to obtain a constant pitch, or should pitch vary over the course of the note? Finding general representations that incorporate continuous and discrete data is still an active area of research.

Declarative and Procedural Representations

Most of the representations discussed so far are encodings of static information. In contrast, computer programs are often used to encode dynamic behavior, or as an alternative representation of static data. For the most part, visual/graphical editors, such as sequencers and notation editors, manipulate static data. Exceptions include MAX (Puckette 1991), Kyma (Scaletti 1989; Scaletti and Hebel 1991), and numerous patch editors, which describe dynamic behavior.

While visual programming is still in its infancy, textual programming languages have a rich history, and the importance of time in music has led to a number of language innovations. A full treatment of languages for computer music cannot be given here, but we will introduce three very different language approaches to time.

FORMES (Rodet and Cointe 1984) is an object-oriented language in which objects represent behaviors. Each object maintains a start time, a duration, and an operation to be performed at each time step. Objects are organized into hierarchical tree structures, and each object is activated at each time step to provide concurrency.

Arctic (Dannenberg, McAvinney, and Rubine 1986) is a functional language, in which values are functions of time rather than simple scalars. Concurrency arises naturally whenever multiple values are computed because values can span overlapping intervals of time. Arctic-like semantics are found in Canon (Dannenberg 1989), a composition system for generating MIDI data, and in the Fugue (Dannenberg, Fraley, and Velikonja 1992) and Nyquist (Dannenberg 1992) systems for sound synthesis. GTF (Desain and Honing 1992) is a proposal for linking control functions to attributes of discrete events. GTF shares Arctic's functional programming orientation.

Formula (Anderson and Kuivila 1986) is a procedural and process-oriented language. It uses the sequential execution of statements to represent sequences, notes, and functions of time. Formula uses processes to obtain concurrent behavior. Of particular interest in Formula are its techniques for real-time scheduling and its support for nested time maps.

HMSL (Polansky, Burk, and Rosenboom 1990) emphasizes hierarchical morphologies, or morphs. The basic morph is a multidimensional array of data; higher-level morphs can contain collections of other morphs. Morphs can provide a declarative, data-intensive representation of structure, but these data can be operated upon and interpreted by programmed procedures. HMSL features graphical interfaces to morphological data and operations.

Resources, Instances, and Streams

There is an interesting distinction between musical entities that are freestanding and those that serve as parameters or updates to some other entity. Often, an entity has both characteristics. Take a note as an example. Some compositions and also languages like Music V treat notes as independent. In Music V, every note produces a copy or instance of a synthesis computation routine (the instrument), which is independent of other notes. MIDI synthesizers in poly mode also support this instance model.

In contrast, notes can also be viewed as updates or control information to some shared resource. For example, in orchestration one must be aware that a clarinet can only play one note at a time, and the line is as important as individual notes. Thus, notes are not independent instances of clarinet tones, but updates to a shared clarinet resource. This is the basis for the resource-instance model (Dannenberg, Rubine, and Neuendorffer 1991).

The resource-instance model, in which every entity is handled by some resource, helps to clarify and explain music representations. For example, in MIDI poly mode it is not specified whether a Note On message is an update to the channel (in which case two notes of the same pitch are possible) or is an update to a key number (in which case two Note On commands of the same pitch cause the note to be retriggered).

It is often convenient to think of a succession of events as a single entity, which we will call a stream. Streams are subject to various operations such as selection of events that satisfy some property, transformations such as transposition, and time deformation. Music Logo (Orlarey 1986), Teitelbaum's performance system (Teitelbaum 1984), MAX (Puckette 1991), and many sequencers, (especially Bars and Pipes (Fey and Grey 1989), use this notation of streams.

A related concept is the sequence, in which components are indexed by ordinal position rather than time. We speak of the seventh element rather than an element at time 3.2 seconds. Gary Nelson's MPL used APL as a basis for a sequence-oriented compositional environment (Nelson 1977). The tone row is a sequence of 12 pitches. PLA (Schottstaedt 1983) and Common Music (Taube 1989) provide a number of sequence generators and use the sequence abstraction heavily.

Protocols and Coding

Once a music representation is adopted, issues of transmitting and storing the representation arise. Transmission, especially in real time, raises questions of network protocols, the conventions by which information is transmitted and received. Storage raises the question of coding, or how the abstract information is converted into specific bit patterns.

MIDI is the most prevalent protocol for the real-time transmission of music information, but it has many weaknesses. MIDI contains no mechanisms for (1) flow control, which would eliminate receive buffer overflow; (2) (forward) error correction, which would enable receivers to detect errors and reconstruct garbled data; or (3) (backward) error recovery, which would provide a mechanism such as retransmission of lost or garbled data. Another limitation of MIDI is that there is no standard way to determine what devices are accessible, to query their status, or to reserve resources. MIDI is based on the transmission of incremental state changes, which means the current state (parameter settings) of a synthesizer depends on its entire message history. If a synthesizer is reinitialized during a performance, there is no way to recover the proper state. Finally, MIDI timing is not explicit. As MIDI data are processed, filtered, and merged, timing distortions occur. There is no way to encode timing specifications or timing distortions so that the intended timing can be reconstructed.

Many of the problems of MIDI are probably justified by economics. In the future, multigigabit network technology will completely change the set of assumptions on which MIDI is based and lead to very different protocols.

In the area of representation coding, two issues are human readability and the encoding of references. Codings can be made "human readable" by using ASCII text (Ames 1985), mnemonic labels, and decimal numbers. Alternatively, codings can be optimized for space efficiency by using binary numbers throughout. References (i.e., pointers) are always problematic when data are encoded for storage in a linear file. A standard technique is to give each entity a unique name, which is then used to encode references to the entity. References tend to make codings harder to work with, but more flexible.

Other Issues

Representations for music databases must make searching efficient. At issue are what kinds of searches are allowed, how search queries are represented, and what kinds of indexes and other data structures can make searching efficient. Brad Rubenstein discusses extensions to a relational database to support music information storage and retrieval in (Rubenstein 1987).

Research into the use of neural nets for music (Lischka 1987; Todd and Loy 1991) has led to the consideration of representations in which information is distributed over collections of artificial neurons rather than being stored in a discrete data structure. An important issue in this work is how data are presented to neural nets. For example, pitch can be specified in terms of frequency, pitch-class plus octave, or the set of triads in which the pitch participates (Hild, Feulner, and Menzel 1992). Time in neural networks has been represented by using time-dependent cells (Scarborough, Miller, and Jones 1989), by successive iterations of a network with feedback (Jordan 1986), and by spatial distribution (Sejnowski and Rosenberg 1986).

Cognitive studies such as that of Simon and Sumner (1968) have investigated how humans represent musical information. Increasingly, music theory is used as a basis for hypothesis forming (Krumhansl 1991), and it seems that perceptual studies are important for new representations and to validate old ones. The issue then is how representations can incorporate results from music psychology research.

Further Information

Many music representation issues are special cases of more general problems addressed by computer science, so a good computer science background is important for those wishing to contribute to this field. Especially important is a knowledge of fundamental data structures such as lists, trees, and graphs, and algorithms for manipulating them (Horowitz and Sahni 1976). Beyond this, the subfields of artificial intelligence and database systems are good sources of information on representation. Brachman and Levesque (1985) edited a collection of important papers on knowledge representation that have much relevance to music representation. Date's book on database systems offers a wealth of material on information organization, representation, and access (Date 1991).

Almost all computer music research touches on some aspect of representation, so a good way to learn more is simply to read back issues of *Computer Music Journal* and *Proceedings of the International Computer Music Conferences*, available from MIT Press and the International Computer Music Association, respectively. Many specific references have already been made. Honing's article on time and structure in music (Honing 1992) is especially recommended.

Music representation issues are discussed informally and sporadically via electronic mail. Readers are invited to join an on-line discussion by sending electronic mail to the author at dannenberg@cs.cmu.edu.

Conclusion

Music is a fertile field for the study of representations. It contains complex structures of many interrelated dimensions, including time. Music, music representations, and music theory are all evolving together, creating a continuous flow of new challenges. I have attempted to present some of the current issues and research directions in music representation. Comments and further discussion are invited.

Acknowledgments

The author wishes to thank Stephen Travis Pope, Carla Scaletti, and especially Henkjan Honing for their comments on an earlier draft of this text.

References

Ames, C. 1985. "The ASHTON Score Transcription Utility." *Interface* 14:165–173.

Anderson, D. P., and R. Kuivila. 1986. "Accurately Timed Generation of Discrete Musical Events." *Computer Music Journal* 10(3):48–56.

Assayag, G., and D. Timis. 1986. "A ToolBox for Music Notation." In *Proceedings of the 1986 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 173–178.

Balzano, G. J. 1980. "The Group-Theoretic Description of 12-Fold and Microtonal Pitch Systems." *Computer Music Journal* 4(4):66–84.

Bilmes, J. 1992. "A Model for Musical Rhythm." In *Proceedings of the 1992 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 207–210.

Böcker, H.-D., and A. Mahling. 1988. "What's in a Note?" In *Proceedings of the 1988 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 166–174.

Brachman, R. J., and H. J. Levesque, eds. 1985. *Readings in Knowledge Representation*. Los Altos, California: Morgan Kaufmann.

Brinkman, A. 1985. "A Data Structure for Computer Analysis of Musical Scores." In *Proceedings of the 1984 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 233–242.

Brinkman, A. 1991. "Computer-Graphic Tools for Music Analysis." In *Proceedings of the 1984 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 53–56.

Buxton, W., W. Reeves, R. Baecker, and L. Mezei. 1978. "The Use of Hierarchy and Instance in a Data Structure for Computer Music." *Computer Music Journal* 2(4):10–20. Reprinted in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press, pp. 443–466.

Buxton, W., R. Sniderman, W. Reeves, R. Patel, and R. Baecker. 1979. "The Evolution of the SSSP Score-Editing Tools." *Computer Music Journal* 3(4): 14–25. Reprinted in C. Roads and J. Strawn, eds. 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press, pp. 376–402.

Byrd, D. 1984. "Music Notation by Computer." Ph.D. thesis, Computer Science Department, Indiana University. Ann Arbor, Michigan: University Microfilms (order no. DA8506091).

Clark, E. F. 1991. "Expression and Communication in Musical Performance." In J. Sundberg, L. Nord, and R. Carlson, eds. *Wenner-Gren International Symposium Series, Vol. 59. Music, Language, Speech and Brain*. London: Macmillan, pp. 184–193.

Dannenberg, R. B. 1986. "A Structure for Representing, Displaying, and Editing Music." In *Proceedings of the 1986 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 153–160.

Dannenberg, R. B. 1989. "The Canon Score Language." *Computer Music Journal* 13(1):47–56.

Dannenberg, R. B. 1990. "A Structure for Efficient Update, Incremental Redisplay and Undo in Display-Oriented Editors." *Software: Practice and Experience* 20(2):109–132.

Dannenberg, R. B. 1992. "Real-Time Software Synthesis on Superscalar Architectures." In *Proceedings of the 1992 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 174–177.

Dannenberg, R. B., C. L. Fraley, and P. Velikonja. 1991. "Fugue: A Functional Language for Sound Synthesis." *IEEE Computer* 24(7):36–42.

Dannenberg, R. B., C. L. Fraley, and P. Velikonja. 1992. "A Functional Language for Sound Synthesis with

Behavioral Abstraction and Lazy Evaluation." In D. Baggi, ed. *Readings in Computer-Generated Music*. Los Alamitos, California: IEEE Computer Society Press.

Dannenberg, R. B., P. McAvinney, and D. Rubine. 1986. "Arctic: A Functional Language for Real-Time Systems." *Computer Music Journal* 10(4):67–78.

Dannenberg, R. B., D. Rubine, and T. Neuendorffer. 1991. "The Resource-Instance Model of Music Representation." In *Proceedings of the 1991 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 428–432.

Date, C. J. 1991. *An Introduction to Database Systems*. Reading, Massachusetts: Addison-Wesley.

Decker, S., and G. Kendall. 1984. "A Modular Approach to Sound Synthesis Software." In *Proceedings of the 1984 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 243–250.

De Poli, G., A. Piccialli, and C. Roads, eds. 1991. *Representations of Musical Signals*. Cambridge, Massachusetts: MIT Press.

Desain, P., and H. Honing. 1991a. "Tempo Curves Considered Harmful." In *Proceedings of the 1991 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 143–149.

Desain, P., and H. Honing. 1991b. "Towards a Calculus for Expressive Timing in Music." Technical Report. Utrecht, The Netherlands: Center for Knowledge Technology.

Desain, P., and H. Honing. 1992. "Time Functions Function Best as Functions of Multiple Times." *Computer Music Journal* 16(2):17–34.

Fey, T., and M. J. Grey. 1989. *Using Bars and Pipes*. Decatur, Georgia: Blue Ribbon Bakery, Inc.

Grey, J. M. 1975. "An Exploration of Musical Timbre." Department of Music Report STAN-M-2. Stanford, California: Stanford University.

Hamel, K. 1988. *MusScribe Reference Manual*. Richmond, British Columbia, Canada: SoftCore Music Systems.

Hild, H., J. Feulner, and W. Menzel. 1992. "HARMONET: A Neural Net for Harmonizing Chorales in the Style of J.S.Bach." In J. E. Moody, S. J. Hanson, and R. P. Lippmann, eds. *Advances in Neural Network Information Processing Systems*. Los Altos, California: Morgan Kaufmann.

Honing, H. 1992. "Issues in the Representation of Time and Structure in Music. Music, Mind, and Machine: Studies in Computer Music, Music Cognition, and Artificial Intelligence." Amsterdam, The Netherlands: Thesis Publishers. Also in *Proceedings of the 1990 Music and the Cognitive Sciences Conference*. London: Harwood Academic Publishers.

Horowitz, E., and S. Sahni. 1976. *Fundamentals of Data Structures* in Pascal. New York: Computer Science Press.

Jaffe, David. 1985. "Ensemble Timing in Computer Music." *Computer Music Journal* 9(4):38–48.

Jordan, M. I. 1986. "Serial Order: A Parallel Distributed Processing Approach." Technical Report 8604, Institute for Cognitive Science. San Diego: University of California

Katayose, H., H. Kato, M. Imai, and S. Inokuchi. 1989. "An Approach to an Artificial Music Expert." In *Proceedings of the 1989 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 139–146.

Krumhansl, C. L. 1991. "Music Psychology: Tonal Structures in Perception and Memory." Annual Review of Psychology, pp. 277–303.

Lischka, C. 1987. "Connectionist Models of Musical Thinking." In *Proceedings of the 1989 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 190–196.

Martens, W. L. 1985. "PALETTE: An Environment for Developing an Individualized Set of Psychophysically Scaled Timbres." In *Proceedings of the 1985 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 355–365.

Mathews, M. V. 1969. *The Technology of Computer Music*. Cambridge, Massachusetts: MIT Press.

Mathews, M. V. and F. R. Moore. 1970. "A Program to Compose, Store, and Edit Functions of Time." *Communications of the ACM* 13(12):715–721.

Maxwell, J. T., and S. M. Ornstein. 1984. "Mockingbird: A Composer's Amanuensis." Byte 9(1):384–401.

Nelson, G. 1977. "MPL: A Program Library for Musical Data Processing." *Creative Computing* 3: 76–81.

Newcomb, S. R. 1991. "Standard Music Description Language Complies with Hypermedia Standard." *IEEE Computer* 24(7):76–80.

Oppenheim, D. V. 1987. "The P-G-G Environment for Music Composition: A Proposal." In *Proceedings of the 1987 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 40–48.

Orlarey, Y. 1986. "MLOGO: A MIDI Composing Environment." In *Proceedings of the 1986 International Computer Music Conference*. San Francisco:

International Computer Music Association, pp. 211–213.

Palmer, C. 1989. "Mapping Musical Thought to Musical Performance." *Journal of Experimental Psychology* 15(12).

Polansky, L., P. Burk, and D. Rosenboom. 1990. "HMSL (Hierarchical Music Specification Language): A Theoretical Overview." *Perspectives of New Music* 28(2):136–179.

Pope, S. T. 1991. "Introduction to MODE: The Musical Object Development Environment." In S. T. Pope, ed. 1991. *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology.* Cambridge, Massachusetts: MIT Press, pp. 83–106.

Pope, S. T. 1992. "The Interim DyanPiano: An Integrated Computer Tool and Instrument for Composers." *Computer Music Journal* 16(3):73–91.

Puckette, M. 1991. "Combining Event and Signal Processing in the MAX Graphical Programming Environment." *Computer Music Journal* 15(3):68–77.

Rodet, X., and P. Cointe. 1984. "FORMES: Composition and Scheduling of Processes." *Computer Music Journal* 8(3):32–50. Also in S. T. Pope, ed. 1991. *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*. Cambridge, Massachusetts: MIT Press.

Rogers, J., J. Rockstroh, and P. Batstone. 1980. "Music-Time and Clock-Time Similarities Under Tempo Changes." In *Proceedings of the 1980 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 404–442.

Rubenstein, W. B. 1987. *Data Management of Musical Information*. Memorandom No. UCB/ERL M87/69. Berkeley: University of California.

Scaletti, C. 1989. "The Kyma/Platypus Computer Music Workstation." *Computer Music Journal* 13(2):23–38. Also in S. T. Pope, ed. 1991. *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*. Cambridge, Massachusetts: MIT Press.

Scaletti, C., and K. Hebel. 1991. "An Object-Based Representation for Digital Audio Signals." In G. De Poli, A. Piccialli, and C. Roads, eds. *Representations of Musical Signals*. Cambridge, Massachusetts: MIT Press, pp. 371–389.

Scaletti, C. 1992. "Polymorphic Transformations in Kyma." In *Proceedings of the 1992 International Computer Music Conference.* San Francisco: International Computer Music Association, pp. 249–252.

Scaletti, C., and K. Hebel. 1991 "An Object-Based Representation for Digital Audio Signals." In G. DePoli, A. Piccialli, and C. Roads, eds. *Representations of Musical Signals*. Cambridge, Massachusetts: MIT Press, pp. 371–389.

Scarborough, D., B. O. Miller, and J. A. Jones. 1989. "Connectionist Models for Tonal Analysis." *Computer Music Journal* 13(3):49–55. Also in P. Todd and D. G. Loy, eds. 1991. *Music and Connectionism*. Cambridge, Massachusetts: MIT Press, pp. 54–63.

Schottstaedt, W. 1983. "Pla: A Composer's Idea of a Language." *Computer Music Journal* 7(1):11–20.

Sejnowski, T. J., and C. R. Rosenberg. 1986. "NETtalk: A Parallel Network that Learns to Read Aloud." Technical Report JHU/EECS-86/01. Baltimore, Maryland: Johns Hopkins University.

Shepard, R. N. 1982. "Geometrical Approximations to the Structure of Musical Pitch." *Psychological Review* 89(4):305–333.

Simon, H. A. and R. K. Sumner. 1968. "Pattern in Music." In B. Kleinmuntz, ed. *Formal Representation of Human Judgment*. New York: John Wiley and Sons.

Taube, H. 1991. "Common Music: A Music Composition Language in Common Lisp and CLOS." *Computer Music Journal* 15(2):21–32.

Teitelbaum, R. 1984. "The Digital Piano and the Patch Control Language System." In *Proceedings of the 1984 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 213–216.

Todd, P. M., and D. G. Loy, eds. 1991. *Music and Connectionism*. Cambridge, Massachusetts: MIT Press.

Waters, S., and T. Ungvary. 1990. "The Sonogram: A Tool for Visual Documentation of Musical Structure." In *Proceedings of the 1990 International Computer Music Conference*. San Francisco: International Computer Music Association, pp. 159–162.

Wessel, D. L. 1979. "Timbre Space as a Musical Control Structure." *Computer Music Journal* 3(2): 45–52. Also in C. Roads and J. Strawn, eds. *Foundations of Computer Music*. Cambridge, Massachusetts: MIT Press, pp. 640–657.