The Composers Toolbox, a Lisp-Based Precomposition Environment: General System Overview and an Introduction to the Package's New Krayola MIDI Paint Module

D. SKRIEN AND J. HALLSTROM
Colby College, Waterville, ME 04901, U.S.A.
djskrien@colby.edu jfhallst@colby.edu

Abstract—The Composers Toolbox is a Macintosh Common Lisp-based environment designed to facilitate ease of data manipulation during the precompositional sketching process. The package provides a series of tools, many with graphic interfaces, which allow the user to work with both MIDI and sampled sound data in a free and flexible fashion.

Keywords—Composers Toolbox, Lisp, MIDI, Music composition, Computer music, Krayola.

1. OVERVIEW OF THE SYSTEM

The Composers Toolbox project was the result of the authors' frustration with the dearth of options available to composers wishing to freely combine the use of algorithmic and graphics-based approaches in the manipulation of musical event streams. This need for flexible tools was seen as particularly acute during the precomposition process, when the ability to "play" with musical ideas, textures, spatial considerations, etc., in a free and flexible fashion is especially crucial to creative freedom. In designing the Composers Toolbox environment, a concerted effort was made to provide a collection of compositional tools which would allow composers to work quickly and freely, and yet not limit their flexibility to interact with the environment in a variety of modes. This has been accomplished by creating a number of self-contained, open-ended tool objects (many with graphic interfaces) which are supplied with default characteristics, but which can be customized at a variety of levels. In order to maintain compositional flexibility, all tools have been designed to allow free-form implementation of both package-defined and user-defined functions, and an internal interpreter has been created which allows Lisp functions to be entered in place of individual data collections in most situations requiring data input.

The Composers Toolbox environment was developed in, and runs under, Apple Corp.'s Macintosh Common Lisp. It requires a reasonably fast Macintosh computer with a minimum of 5 MB of RAM. At present, the package contains two subenvironments, one devoted to DSP operations on sampled sound files, and one devoted to the manipulation of MIDI event streams. Since the Composers Toolbox was conceived as an addition to, rather than a replacement for, other composition environments, considerable effort has been undertaken to provide for communication with other applications. Accordingly, the system both reads and writes standard AIFF (Audio Interchange File Format) and MIDI files, and is compatible with Apple's MIDI Manager.
attempt to foster continuing dialogue on further options for precompositional systems, and to en-
courage further development of the current tool set, the entire Composers Toolbox environment,
complete with user's manual and copiously documented source code, is available free of charge
(for noncommercial use) to the general computer music community.

2. OVERVIEW OF THE DSP SUBENVIRONMENT:
CT-PAN AND CT-DOPPLER

The principal features are as follows:

- a collection of algorithms which implement DSP functions not commonly available in
  commercial packages,
- configurable graphic tool objects which provide a friendly interface to the algorithms,
- the ability to read and write standard AIFF sound files,
- access to Digidesign's family of Motorola 56000-based DSP cards.

To date, the Composers Toolbox DSP environment includes two modules, both of which have
been designed for speed and ease of use. CT-Pan provides the means to experiment with dynamic
Panning sequences, and CT-Doppler with Doppler shifting sounds. Both modules were written
using a combination of Assembly and Lisp code, and both include user-configurable graphical

![Figure 1. The Pan and Doppler interfaces.](image)
interfaces written entirely in Macintosh Common Lisp. Playback of sound files can be
accomplished from within Lisp by means of a series of functions which provide access to Digidesign's
family of DSP cards. The CT-Pan module has also been set up to run its computations on the
card, and when running in its 56000-access mode can process dynamic panning sequences in real
time. The interface to this module is illustrated in Figure 1.

3. OVERVIEW OF THE MIDI SUBENVIRONMENT

The principal highlights of the MIDI subenvironment are:

- data objects which allow the representation of MIDI data collections as macro-events
  which can be manipulated on a variety of levels,
- extremely powerful and user-extendable search and transformation shells,
- a wide range of predefined data manipulation functions,
- Apple MIDI Manager compatibility,
- the ability to read and write standard MIDI format files.

CT-MIDI’s Principal Low-Level Data Structure: The Gesture Object

Almost all the tools in the CT-MIDI package take advantage of a fundamental data structure
called a Gesture (see Figure 2). Gestures are data objects which serve as repositories for four
types of information:

1. a name for the gesture,
2. six lists of individual MIDI events, each type of event being in a separate list,
3. zero or more subgestures (which are merely other gestures), and
4. an offset time that is used to determine when the system should output the gesture to the
   MIDI Driver in relation to its parent gesture (if any).

In addition to performing the obvious function of referring to things like motives, phrases, etc.,
gestures can do a number of other things. Since a gesture can refer to anything from a single
MIDI event (a patch change message, for instance) to an entire composition, and gestures can
be nested to any level, it is possible to create extremely complex hierarchical structures that can
still be manipulated as single entities.

A gesture’s subgesture may contain, in addition to its own event lists, further subgestures, etc.,
allowing for the nesting of materials to virtually any level, while maintaining complete autonomy
among all the elements. A gesture, together with its subgestures, the subgesture’s subgestures,
etc., may refer to many thousands of MIDI-events. However, it is still regarded by the Toolbox
as a single entity. As such it can be manipulated (copied, transformed, etc.) without dealing
directly with any of its component parts.
Overview of Basic CT-MIDI Functions

The CT-MIDI module provides the means to manipulate gestures and their subgestures as a single unit, in groups consisting of subsets of the MIDI events of the gesture, or as individual MIDI events. For manipulating gestures as a whole, the package provides means to create, delete, and copy gestures, to add or delete subgestures, and to change subgesture offset times. Gestures can also be "agglomerated" by collecting all events from the gesture's subgesture "tree" and putting them together on one level in the event list of the parent gesture. Once a gesture is created, the user can easily add new events to it by appending to the events already in the gesture, merging them with existing events, or shifting the current events to create a time slot for the new events.

One of the most useful features of the CT-MIDI module is its ability to search for events based on a wide variety of criteria and then to transform those events in a variety of ways. The searching tools provided with the module allow the user to find almost any collection of events of a given type in a gesture (including noncontiguous events). Search criteria can take the form of built-in search functions, custom user-defined (Lisp) search functions, or a combination of the two. Once events have been selected, they can be transformed via a wide range of predefined transformation options which can include, as in the search shell, the introduction of user-defined functions.

4. KRAYOLA: A HIGH LEVEL MIDI EVENT "PAINTING" INTERFACE

In an effort to further enhance the user-friendliness and flexibility of the Composers Toolbox, the Krayola MIDI paint interface was recently added to the environment. Krayola functions as CT-MIDI's highest level of graphic interface. It was designed to provide composers with an easy-to-use, functionally robust means of experimenting with musical raw materials (themes, harmonies, timbral flows, etc.). Its simple and intuitive top-level interface functions like a sophisticated "paint" program for MIDI events, with the added capability of allowing the introduction of algorithmic controls over virtually any aspect of MIDI data pattern creation and flow (both horizontal and vertical).

The Krayola Event Entry Interface

At its simplest level, the Krayola event entry interface is quite similar to that of a typical sequencer program in "piano roll" mode. Using a combination of the note placement and arrow tools (see Figure 3), individual notes can be entered in the time/pitch grid and selected either individually or in groups (using the standard Macintosh shift-click or dragged marquee selection methods). Once selected, events can be dragged to change their duration or position in the pitch/time grid or option-dragged to replicate them.

Controls over the Placement of Horizontal Collections of Events

The true power of Krayola emerges in the additional controls it provides for the placement of pitches, rhythms, etc., most of which are available through the Draw options dialog (see Figure 4). In its default state, this dialog contains the options available at the simple note entry level: MIDI channel is set to 1, velocities will all take the value of 60, and durations and offsets will be determined by the place where the mouse is clicked in the pitch/time grid (indicated by the symbol x in various fields of the configuration window). To provide for more complex collections of pitches, durations, etc., the user needs only to alter these default values. By entering multiple integer values into any of the configuration slots, these values will be iterated as the mouse is clicked repeatedly in the main display window. In the case of offsets and durations, the interpreter "watches" the position of the mouse after it has been clicked and invokes a new iteration of the pitch/velocity/channel data when the requested duration/time offset has elapsed. X's (mouse
position-dependent values) can be mixed freely with specifically designated pitches, durations, etc., to provide varying degrees of control over how movement of the mouse affects the placement of events in the window.

Controls over the Placement of Vertical Collections of Events

The Draw parameters interface also provides straightforward options for control over the creation and placement of chords. Placing note values in parentheses causes those values to be drawn simultaneously. If an X (mouse position dependent value) is entered as the first value in a parentheses-delimited group, all subsequent values will be interpreted in terms of their relative position to the point at which the mouse was clicked (e.g., (X 2 – 5) will result in a three-note chord with pitches two semitones above and five semitones below the value corresponding to the point where the mouse was clicked). Placing an X after a series of values will cause the those
values to be interpreted as literal MIDI numbers (e.g., (50 62 X) would result in a three-note chord containing the pitches 50, 60, and the value corresponding to the point where the mouse was clicked). Sequences of chords (both literal and mouse position-controlled) and individual pitches can be freely combined.

Controlling Rhythm and Duration

By entering multiple offsets and durations in their respective slots, it is possible to create mouse-position-controlled rhythmic patterns. Used in conjunction with the above options for creating chord and individual pitch sequences, it is possible to use these patterns to control both the horizontal and vertical flow of the composition in very sophisticated ways.

Algorithmic Controls

Any (and all) control parameters can take Lisp functions as their arguments. The environment provides options either for evaluating the function on a one-time basis (with the results of the evaluation placed in the parameter slot for further editing), or for invoking the function repeatedly as the mouse is clicked in the main display window (including allowing the value that results from the click to be used as input to the function). Both hand-entered patterns and function calls can be caused to restart the parsing of data at each mouse click, or to continue to cycle through the generated values in subsequent clicks. Functions and hand-entered values can be mixed within individual slots to provide virtually limitless controls over the painting of pitch/time collections.

Display Controls in the Main Event Window

Krayola provides a wide variety of options for controlling the display, selection, and editing of gestural data. By grouping a series of gestures under a single parent gesture and then displaying the parent in the main Krayola window, it is possible to view and edit multi-“track,” multichannel event streams simultaneously (including the ability to cut, copy, etc., across multiple gestures). On systems utilizing a color or grayscale monitor, events can be “colorized” either by channel or by gesture, providing a clear and concise view of the relationships existing in the data being edited. When working with large amounts of data, or when fine-tuning individual events, the interface allows time slices to be selected and zoomed in or out, including a “full zoom out” option which will fit all data associated with a gesture (and if requested, its subgestures) in a single window frame, thereby allowing the user to “step back” at any time and view an entire work in a single window “pane.”

Displaying Subsets of Data: The Display/Select Interface

Using the display/select options interface (see Figure 5), subsets of data can be isolated for further editing. This interface allows the user to build complex collections of “and/or” clauses (which can include both system and user-defined algorithmic controls) which will be applied to any data which is currently being displayed in the main window (including the collecting of noncontiguous data). Once the user has isolated the data he/she wishes to display, the interface can be switched to “select” mode to gather notes for editing (see Figure 5).

Editing Controls

One of the Composers Toolbox’s most powerful features is its transformation shell. In contrast to a typical MIDI sequencer program which is only able to do simple transformations of data (transpose, reverse, scale time, etc.), the Composers Toolbox provides the means for introducing algorithmic controls over data transformation in virtually limitless ways. Once the user has isolated and selected the data he/she wishes to transform, the edit interface can be invoked
Figure 5. The display and select control dialogs.

Figure 6. The transform options dialog.

(see Figure 6). In addition to standard system-defined transformation procedures, the interface provides the means to introduce user-defined functions to control the transformation process.

5. CONCLUSIONS

The addition of the Krayola MIDI paint module to the already robust collection of tools contained in the Composers Toolbox extends the usefulness of the system by providing a link between the virtues of a typical graphics-based sequencer program and those of a full-fledged algorithmic composition language. Future plans include the extension of the Krayola paradigm to include the display and manipulation of whole gestures as objects in the same way that it is currently able to work with the events they contain.