Helsinki Music Tools
Jukka-Pekka Keroune (jpk@elisi.helsinki.fi)
Kai Lasnestic (kpl@elisi.helsinki.fi)
Department of Musicology
P.O.Box 35 (Virrankatu 3)
00014 University of Helsinki

Abstract
Helsinki Music Tools (abbreviated as HMT) is a set of musical software tools. HMT works under the UNIX operating system. HMT consists of more than 50 programs that can be used for musical tasks such as algorithmic composition and music analysis. HMT programs behave like the standard UNIX tools and take advantage of standard UNIX features including input/output redirection. HMT provides no direct means of musical input or output. To compensate this, HMT can be used in combination with other music software such as a synthesis language. The current implementation supports Csound and the CASSA Music Kit.

1 Introduction
HMT consists of UNIX-like commands that are typically called from the shell. The HMT commands behave like their UNIX counterparts. They read and write ASCII formatted data using the standard input/output mechanism. Using ASCII allows to use standard UNIX commands in combination with HMT to perform musical tasks.

The following example demonstrates how HMT commands are used and combined to generate and play a simple melody.

```
$ hrnd 36 96 10 | aagen | asap
```

Here the `hrnd` command prints 10 random numbers distributed uniformly between the minimum and maximum values of 36 and 96. `hrnd` prints the numbers to the standard output stream which is connected via a pipe to the next command, `aagen`. `aagen` reads standard input and generates an event list (called "score"). The score is in turn piped to `asap` which plays it.

`hrnd` is one of HMT's generative commands.

There are nine commands for various stochastic distributions, two stochastic processes and an implementation of generative context-free grammars. These commands can be combined with the UNIX piping mechanism to form more complex random processes.

`aagen` is a tool for creating HMT scores. `aagen` generates a "simple score". A simple score consists of timed events which is turn consist of parameter fields. The format resembles a conventional synthesis language such as Music V or Csound. An example simple score is shown below.

```
$vrc 1 0 1 60 30
$vrc 1 1 1 62 60
$vrc 1 2 1 64 90
```

The example score has five note events that form a melody line of c4, d4, e4, f4 and g4. Events are separated by newlines and parameter fields are separated by spaces and/or tabs. The first field in an event is an event type. The event type of a musical note is "vrc". The second field is an instrument number, an integer greater than or equal to 1. The third field is the starting time (called "attack time") of the event. It is represented in beats by a floating point number. The fourth field is duration of the event. The fifth field is a note number, an integer ranging from 0 to 127. The note numbers are the same as in MIDI. The sixth field is an amplitude value (or velocity), an integer ranging from 0 to 255. Attack times are calculated either from the beginning of the score or from start of the previous event.

`aagen` creates `vrc` events by reading note number values from standard input and filling the other fields with default values. This default action can be changed by using one of the following options:

- `i` standard input is placed in the instrument number field
- `a` standard input is placed in the attack time field
- `d` standard input is placed in the duration field
- `v` standard input is placed in the amplitude field

`aagen` can also be given a template score file as a command line argument. In the following example, `aagen` reads amplitude values from `hrnd` and fills in the other fields with values from `vrc` events stored in the file `t.ssa`. The output is redirected to `u.ssa`.

```
$ hrnd 0 255 10 | aagen -v t.ssa > u.ssa
```
HMT provides commands for score generation and manipulation. For example the following command prints attack times of a given score file.

```
$ aget -a testscore.mn
```

These values can be piped to t command that manipulates numerical data in various ways. agetc accepts the same options as aget.

2 Generative programs

The generative programs give good examples of features common to most HMT commands. Generative programs allow arguments to be constants or variables. For example, the following command line

```
$ hrand 0 \leq 100
```

prints 100 normally distributed random numbers with mean value of 0 and variance of 5. The argument values 0 and 5 are treated as constants. In the following command

```
$ hrand a \leq b 100
```

hrand treats a and b as files containing numeric values. The program reads new mean and variance values from files a and b respectively for each random number it prints. The command terminates when 100 numbers are printed or if either a or b runs out of numbers.

A variable argument may be either a valid UNIX filename or '...' which indicates standard input. Thus variable arguments may be piped from command to command as demonstrated by the following example.

```
$ hrand 0 \leq 10 0 \leq hrvp 60 - 36 96
```

This command line combines the Gaussian distribution command hrand and a random walk command hrvp. hrvp takes four arguments: start, jump, min and max. In the example hrvp uses jump values produced by hrand. This generates a random process called Brownian motion (Dodge and Jerse, 1965).

By default all stochastic distributions and processes produce integer values. Also by default, the seed value for the random number generator is read from the system clock. These settings may be changed with the following options.

- `f` Specify floating point output. This is not recognized by discrete distributions like Poisson.
- `n seed` Specify a seed value to the random number generator.

The last command line argument that specifies the amount of generated numbers is optional for stochastic distributions and processes. Omitting this argument produces an infinite random number series.

Other stochastic commands include linear, triangular, Cauchy, Erlang and Weibull distributions and a simulated 1/f process by the Voss algorithm.

3 Scores and number sequences

The two fundamental data representation formats used in HMT are scores and number sequences. A number sequence is a stream of consecutive numbers separated by spaces, tabs or newlines. Number sequences can represent any musical parameter, such as pitch, timing, amplitude or parameter specific to a synthesis program.

A score represents a collection of musical parameters organized as events. An HMT score may contain any kind of data (not necessarily musical) as long as certain basic syntactic rules are followed.

HMT simple scores follow strict rules and are limited in features. The idea behind simple scores is to provide an easy-to-learn starting point for newcomers and to provide a least common denominator for all scores. Simple scores can be converted to other formats such as MIDI files or synthesis languages (and back) with no loss of information. HMT provides tools for generating and manipulating simple scores. HMT also contains conversion programs between simple scores and other formats including CCMRA Music Kit scores (Boynton and Jaffe, 1989) and Csound (Vercoe, 1991).

HMT also allows the construction of more complex user-defined scores. It is however recommended that such a score be a superset of a simple score and should follow the assertions syntax. HMT has tools to create and modify arbitrary scores but no conversion programs to or from other music file formats are provided. The construction of conversion programs is left to the user. Score conversions from HMT to an other text file format such as Csound score is however an easy task using e.g. the awk programming language (Abel et al., 1988).

The command hzgen generates user-defined score events by specifying default parameter field values with command line arguments. For example the following command line

```
$ hzgen 36 96 5 \{ hzgen \{ 0+1 \} 1 - 128
```

prints a score similar to the example score presented in the first chapter. hzgen accepts both constant and variable arguments as described in the previous chapter. The third argument ('0+1') has a notation specific to hzgen. There '0' represents an attack time value for the first event. This is incremented by 1 for each following event. hzgen accepts any number of arguments.

hzget is a command for retrieving number sequences from score parameter fields. It works as aget (described in chapter 1) but has a different syntax to suit user-defined scores.
HMT contains commands for applying basic arithmetic operations on number sequences. These can be used for transposition or time scale modifications. Also inversion, linear interpolation and retrograde programs are provided.

For example the following command

$ sngst t2.ssa | hadd 12 | sspmv t2.ssa

prints a score where note numbers are transposed by an octave.

4 Context-free grammars

Context-free grammars introduce a third data representation format as an addition to number sequences and scores. A grammar is stored in a text file. An example grammar is shown below.

F --> A B C
A --> 60 62 65
A -->
B --> 72 68 67
C --> 55 52 36
C --> 55 57 60 48

A grammar file is processed by the command hgrar that prints terminal symbols. hgran treats each line as a production that consists of a left-hand side (non-terminal), a production symbol ("->") and a right-hand side (any number of terminals and/or non-terminals). If more than one production has the same left-hand side, one of them is picked at random. If a right-hand side symbol does not exist as the left-hand side in any production, it is treated as terminal and thus printed if the appropriate production is chosen.

Here is an example grammar:

$ hgran grammarfile

where grammarfile contains a grammar. hgran accepts a random number generator seed value as described in chapter 2.

5 Other features

HMT is designed to be easily modifiable by the user. In particular most data conversion programs are written in Bourne Shell and AWK to provide a starting point for modifications. Playback routines, such as the smp command, allow users to write their own data conversion programs for the desired output format. smp also allows to write a header file containing instrument definitions and other data specific to a synthesizer language.

HMT has been written mostly in the C programming language. Some commands, mainly conversion programs, were written in AWK and Bourne Shell. YACC and Lex were used in the generative grammar program. HMT should port easily to most UNIX System V and 4.3BSD implementations. The only major difficulty when porting HMT is the availability of musical I-O routines. The CCRMA Music Kit for example is currently not portable beyond NeXT hardware. Also, UNIX lacks a standard MIDI application programming interface.

6 Summary

HMT combined with standard UNIX tools provides a flexible and programmable music processing environment. Most HMT commands are general-purpose tools that generate or manipulate numerical data. These tools are not particularly "musical". What makes HMT a music software system is the score concept and the score manipulation tools.

HMT contains no sound synthesis or MIDI I-O routines. Instead it requires the use of other music software to perform these tasks. On NeXT computers HMT scores can be synthesized or converted to MIDI files using the CCRMA Music Kit. In other environments a synthesis language can be used.

Perhaps most importantly HMT provides a framework for developing new composition and analysis programs.

References


