RTcmix: Recent Developments

John Gibson, Indiana University School of Music (johgibso@indiana.edu)

Abstract: The RTcmix sound synthesis and processing program works by accepting scripts in any of several languages, and then rendering audio in real time. Recent work extends the syntax of RTcmix scripts to support more flexible and extensible means of real-time control. A new editing program, RTcmixShell, makes it easier to get started with RTcmix scripting.

1 Introduction

RTcmix is a program for synthesizing and processing audio in real time, either interactively or off line. The program was derived from Paul Lansky's Cmix by Brad Garton and David Topper and has a long history of use by composers [Garton and Topper 97, Gibson and Topper 00]. It is free software, in the sense that you can download and use it free of charge, and — more importantly — that you can see the source code, modify it, and distribute your changes. RTcmix runs on Linux and Mac OS X. A Windows port is under way.

Recently Douglas Scott and I have contributed important new features to RTcmix, making it more powerful and flexible. This paper offers an explanation of the latest features in RTcmix 4, which make it much easier to control sound in real time. This should be of interest to anyone who has used RTcmix or another text-based audio program, such as Csound or SuperCollider, or anyone who would like to learn what RTcmix can do for a composer or sound designer.

You control RTcmix with a scripting language or via a custom interface program that embeds the RTcmix functionality. (One such program is David Topper's GAIA, Graphical Audio Interface Application [Topper 04].) Three scripting languages are available: Perl, Python and MinC. The latter is RTcmix's own interpreter, modeled on the C programming language.

You can use any text editor to write the scripts. (I like Vim.) For users new to RTcmix, I've written RTcmixShell, a simple editor that provides play and stop buttons, syntax highlighting, and various RTcmix-specific amenities, such as function hinting. (Press a key to see a list of functions; choose one to insert its name and see its argument list.) I plan to add graphical facilities for table construction, audio interface selection, and so on (figure 1).

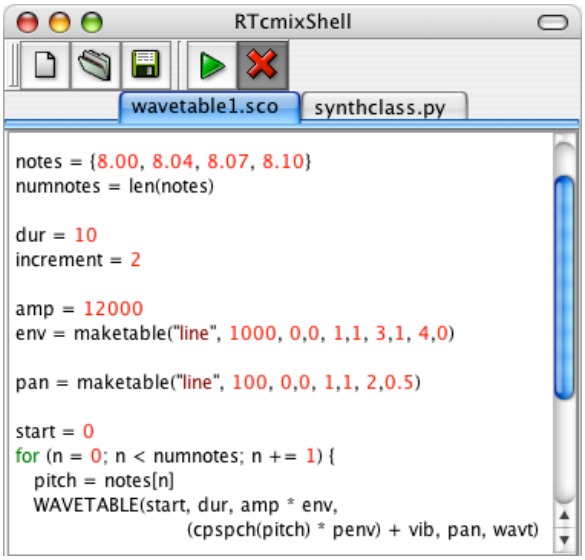

Figure 1: The RTcmixShell script editor

To make sound in RTcmix, you set up tables and other control structures; then you invoke *instruments*, which synthesize audio or process live input or sound files. Instrument *calls* have *arguments* that determine the behavior of a "note." (A *note* is the sound generated by one instance of an instrument, regardless of whether this comes across as one note in the conventional musical sense.) An argument is often a variable whose value you set earlier in the script. Here is a typical instrument call, with some arguments supplied as variables and some as literals.

```
DELAY(start, 1.25, dur, amp,
      deltime, feedback=0.9,
      0.5, 0, pan)
```

The sound output goes to the system audio driver or to a file for inclusion in a

different RTcmix script or another program. RTcmix now supports the ALSA sound driver on Linux, as well as Mac OS X CoreAudio, including multichannel interfaces like the MOTU 828 mkII and Digidesign 192 I/O.

2 Why Bother?

People sometimes ask: "Why would I want to use RTcmix, when there are so many other wonderful tools?" It's true that with the availability of extraordinary commercial programs, such as Max/MSP and the Native Instruments series, it's harder to argue for a text-based approach. But any program can influence your compositional process, and each person must find the set of tools that fits well with his or her way of working. Text-based programs can offer greater generality, flexibility and precision, and sometimes make it easier to express certain ideas. For example, I find it much less confusing to construct a complex loop in one of RTcmix's procedural scripting languages than to connect a bunch of Max/MSP objects in the right graphical order. (It can also be easier to debug a script.) For people who want the best of both worlds, Brad Garton has written rtcmix~, an MSP object that embeds much of the functionality of RTcmix. (His latest work incorporates the version 4 improvements I describe below.)

Another advantage of scripting languages — at least relative to "hard-wired" software synthesizers like Native Instruments' Absynth — is easy extensibility. Using the Python front end to RTcmix, I built a large algorithmic loop-generating program, called Hula, which has its own special-purpose scripting syntax that can be extended by arbitrary Python code [Gibson 04]. Hula lets you create multiple *player* objects, each with its own tempo and note pattern, and makes it easy to specify a large number of randomly changing note characteristics.

Working with scripts in a Unix command-line environment can lead to novel ways of managing work flow. When I compose in RTcmix, I use a *makefile* to describe dependencies between the many scripts and sound files that go together to form the piece. If I make a small change to one script, typing "make" regenerates all necessary sound files, in the correct order. This may seem a minor point, but one of the difficulties most people face when composing computer-based music is organizing their work and keeping track of many details. Scripts can help.

3 Dynamic Instrument Control

The improvements in RTcmix 4 center around dynamic control of instruments. Formerly, serious limitations made it awkward or impossible to control instrument behavior during the course of a note. All the arguments to an instrument call were constant values. Instruments that supported some time-varying parameters did so via the *makegen* table-creation function. You created a table with makegen, giving it an ID number that the instrument expected. One drawback was that instruments tended to expect the same IDs, leading to confusion and unanticipated results. RTcmix 4 introduces a new way of specifying time-varying parameter changes that is clearer and much more flexible. Instrument arguments are no longer limited to constant values. In the DELAY example on the previous page, *deltime* now can be either a constant or a reference to a table, MIDI controller, or other real-time data stream.

A new family of functions creates these time-varying parameter references. This is best illustrated by example (figure 2). The *maketable* and *makeconnection* functions return a reference that you can store into a variable — such as *deltime* — and then pass to an instrument in order to control a parameter continuously during the

```
amp = maketable("line", size=1000, 0,0, 1,1, 9,1, 10,0)
deltime = makeconnection("midi", min=0.01, max=0.5, default=max,
                         lag=80, chan=1, "cntl", "foot")
feedback = makeconnection("mouse", "x", min=0, max=1, default=0, lag)
DELAY(start, inskip, dur, amp, deltime, feedback, ringdur, inchan, pan)
```

Figure 2: *maketable* and *makeconnection*

life of a note.

The *maketable* function lets you create many different types of table, containing lines, curves, splines, waveforms, random numbers, numbers culled from text or sound files, etc. Functions like *addtables, multtables, inverttable, reversetable, shifttable* and *quantizetable* let you combine and reshape tables.

The *makeconnection* function establishes a link to a real-time data stream, providing a convenient way to control parameters via MIDI or mouse input. The incoming data is scaled to fit a range you specify, and is smoothed depending on the value of *lag*. Mouse input provides a convenient way to test the behavior of an instrument when its parameters change, as well as a reasonable way to operate a multichannel panning instrument, of which RTcmix has several. MIDI input currently is limited to controlling the sound characteristics of a note while it plays, rather than triggering new notes. Future work will implement this second possibility. You can use any MIDI channel voice message to control RTcmix instruments.

RTcmix supports real-time connection types via a plug-in system, which means you can write your own plug-in to read data from a USB microscope or any other device. We hope to make plug-ins for Open Sound Control [Wright and Freed 97], as well as Electrotap's Teabox [Allison and Place 04] and other sensor interfaces. The rtcmix~ MSP object employs a connection plug-in to stream control data into RTcmix from MSP inlets.

To some extent, you can treat a dynamic reference as you would a constant. For example, you can add or multiply two references — say, to scale one type of MIDI input by another — or multiply a constant and a reference, as in the following code excerpt. In figure 3, a wavetable oscillator instrument plays a note each time through the loop. The base frequency changes randomly from 100 to 2000 Hz, but the glissando is always up an octave, due to the multiplication of *freq* and *gliss*. The "nonorm" tag suppresses normalization of maketable output, which would scale the line segment so that its values fit between 0 and 1. Tables

```
// exponential curve from 1 to 2
gliss = maketable("curve",
         "nonorm", 100, 0,1,3, 1,2)

// loop for 10 seconds
for (n = 0; n < 10; n += 0.5) {

   // random constant frequency
   freq = irand(100, 2000)

   WAVETABLE(n, dur, amp,
               freq * gliss)
}
```

Figure 3: Combining a constant and a dynamic reference

normally are read using linear interpolation, but it's possible to ask for no interpolation or second-order interpolation when creating the table.

4 LFOs and Randomness

Two other functions create dynamic data internally, rather than pulling the data from outside of RTcmix, as does makeconnection. The *makerandom* function generates a stream of random numbers, using one of several distribution types. The numbers emerge at a frequency that you specify as a constant or as another dynamic reference. The following code creates a reference, *amp*, to a stream of random numbers that range from –3 to 2.5, using a Gaussian distribution and a frequency that moves from 0 to 20 and back. The minimum

```
// line from 0 to 20 to 0
freq = maketable("line", "nonorm",
           100, 0,0, 1,20, 2,0)

amp = makerandom("gaussian", freq,
        min = -3, max = 2.5, seed)
```

Figure 4: *makerandom* with dynamic frequency

and maximum values can also be set using a dynamic reference, rather than a constant.

The *makeLFO* function generates a cyclical stream of numbers, using a standard waveform (sine, sawup, sawdown, square or triangle) or any waveform supplied as a table reference. The frequency and amplitude of the LFO can be constant or dynamic. A variation of makeLFO allows specification of a value range

rather than an amplitude. In figure 5, LFO frequency changes randomly (twice a second) to values between 1 and 9 Hz. The sine wave is bipolar, with amplitudes ranging from -0.1 to 0.1.

```
freq = makerandom("linear", 2, 1, 9)

vib = makeLFO("sine", freq,
              min = -0.1, max = 0.1)
```

Figure 5: *makeLFO* for random-speed vibrato

We could use the *vib* reference returned by makeLFO as a multiplier for the frequency input of WAVETABLE, thus creating sub-audio rate frequency modulation.

5 Massaging the Data

Additional functions let you condition the data generated by the functions described above. The *makeconverter* function takes a dynamic reference and applies one of RTcmix's converters to the incoming data. As an example, we often want to work with gain in terms of decibels, rather than linear amplitudes. But most RTcmix instruments accept the latter. Figure 6 shows how you might handle this problem. We

```
gain = maketable("line", 1000,
          "nonorm", 0,0, 1,90, 4,0)

gain = makeconverter(gain, "ampdb")

WAVETABLE(start, dur, gain, freq)
```

Figure 6: *makeconverter* for amplitude conversion

construct a table containing a decibel ramp from 0 to 90 dB and back. Since WAVETABLE wants linear amplitude instead, we route the *gain* reference through the decibel-to-linear amplitude converter, *ampdb*, before passing it on to the instrument. RTcmix also supports conversion between its various pitch formats: cycles-per-second, linear octaves, octave-point-pitch class, and MIDI note number.

The *makefilter* function creates control data filters of various sorts. They operate in a manner analogous to that of makeconverter. The filter types currently available include *fitrange*, which scales incoming data to a range that can change dynamically; *quantize*, which snaps values to a grid defined by a (possibly changing) quantum; and *smooth*, which applies a dynamically adjustable first-order low-pass filter to the data, so as to smooth over any discontinuities.

6 Project Status

Most RTcmix instruments have been rewritten to take advantage of the new features while remaining fully compatible with older scripts that use the makegen function. RTcmix 4 is still under development and will be available for download in the spring of 2005, from ftp://presto.music.virginia.edu/pub/rtcmix. People interested in the bleeding edge may download the nightly snapshots, though you must compile these yourself. Fortunately, we have greatly improved our build system for this release, so compiling is much easier than it used to be. We encourage everyone to try RTcmix, and we welcome all suggestions for improvements.

References

[04] Allison, J. T. and T. Place (2004). "Teabox: A Sensor Data Interface System," *Proceedings of the 2004 International Computer Music Conference*.

[97] Garton, B. G. and D. Topper (1997). "RTcmix — Using CMIX in Real Time," *Proceedings of the 1997 International Computer Music Conference*.

[04] Gibson, J. (2004). "Loop-based Composition with RTcmix and Hula," *Proceedings of the 2004 International Computer Music Conference*.

[00] Gibson, J. and D. Topper (2000). "Multichannel Audio with RTcmix," *Journal SEAMUS,* Volume XV, Number 1.

[04] Topper, D. (2004). "GAIA: Graphical Audio Interface Application," *Proceedings of the 2004 International Computer Music Conference*.

[97] Wright, M. and A. Freed (1997). "Open Sound Control: A New Protocol for Communicating with Sound Synthesizers," *Proceedings of the 1997 International Computer Music Conference*.