Multichannel Audio with RTcmix

John Gibson and David Topper

The Open Source software paradigm offers many benefits to users and programmers alike, not the least of which is the ability to take existing code and modify it to suit a particular need. This can be incredibly useful to a musician. Imagine being able to redesign a trumpet or violin to investigate a new musical idea. This is the story of RTcmix and multichannel audio. An open package from the outset, Cmix was made real time in 1997. Enhancements have continued, the latest of which were presented at SEAMUS Y2K at the University of North Texas. Work has centered on two new features: output to multichannel audio cards, and an internal signal-routing scheme.[1]

Before describing these new capabilities, let's take a look at a typical RTcmix score. You control RTcmix by feeding it a script written in a simplified dialect of the C programming language. If RTcmix likes the score, it'll play the result in real time or write it to a sound file. People who have never programmed usually get the hang of script-writing pretty quickly.

This score reads random snippets from a sound file and sprays them around the stereo field:

```
rtinput("/snd/bulldozer.aiff")
file_dur = DUR()
dur = 0.2

for (start = 0; start < 20; start = start +
0.1) {
   instart = random() * file_dur
   amp = random()
   stereo_loc = random()
   STEREO(start, instart, dur, amp, stereo_loc)
}
```

The resulting sound lasts 20 seconds. Every tenth of a second, the STEREO instrument fires off a new "note." For each note, the time to start reading on the input file, and the output amplitude and stereo location, are randomly chosen. The ease with which you can construct loops of this sort is a real strength of RTcmix.

But what if you want to process this 20-second sound? That's when you need the new signal-routing capability. Instead of sending the output of the STEREO instrument to the sound card, you route it to the input of another instrument, REVERBIT, and send that instrument's output to the sound card.

Every electroacoustic musician knows how to use a mixer, so we thought it best to adopt mixer terminology when designing the new signal-routing feature. A "bus" is an internal path in a mixer that carries audio signals from one place to another. For example, the "main mix bus" combines signals from multiple channels so that they can be sent to a stereo mix-down deck; an "aux send bus" routes signals to an effects processor. So we provided RTcmix with a notion of buses.

In RTcmix there are three kinds of bus:

- **in** — input from a sound file or from a real-time audio source (e.g., microphone)
- **aux** — intermediate bus, functioning as either an input or an output, used to connect instruments together
- **out** — output to a sound file or to the sound card

You specify the signal routing using a new function called *bus_config*. We wanted it to be easy to modify old scores to take advantage of the new features, so this bus_config function is the only thing added to the score interface.

Let's modify the score given above so as to reverberate the looped sound. First we tell STEREO to send its output to the "aux 0-1" stereo bus, and REVERBIT to take its input from the same pair of buses:

```
bus_config("STEREO", "in 0", "aux 0-1 out")
bus_config("REVERBIT", "aux 0-1 in", "out 0-
1")
```

Then, following the STEREO loop given earlier, we add some code to run the reverberator:

```
reverb_time = .5
wet_dry_mix = .5
right_chan_delay = .1
low_pass_cf = 2000

REVERBIT(0, 0, dur=20, amp=1, reverb_time,
     wet_dry_mix, right_chan_delay,
low_pass_cf)
```

Changing the signal paths is as easy as editing the "bus_config" lines.

The other new feature in RTcmix is the ability to take full advantage of multichannel sound cards. We're not limited to two output buses. We can have as many as the sound card allows (or more, if writing to a file). Let's say you want to add two output channels to the score we've been writing: have a flanger process the STEREO loop and send the output of that to an additional pair of speakers, while retaining the reverberated output to the first pair of speakers. You need to add this bus_config statement to the ones already given:

```
bus_config("FLANGE", "aux 0-1 in", "out 2-3")
```

And then some lines to run the flanger:

```
reson = 0.1
max_delay = 0.004
depth = 80
speed = 0.5
wet_dry_mix = 0.5
FLANGE(0, 0, dur, 1, reson, max_delay, depth,
   speed, wet_dry_mix, 0, inchan=0,
stereo_loc=1)
FLANGE(0, 0, dur, 1, reson, max_delay, depth,
   speed, wet_dry_mix, 0, inchan=1,
stereo_loc=0)
```

(FLANGE takes mono input only, so you need to process each channel separately.)

All existing RTcmix instruments, which output in stereo, can take advantage of this flexible output bus scheme. You just change which pair of outputs the instrument uses.

A new instrument, MIXN, lets you address all outputs in more complex ways: you can define a path for a mono sound to follow, weaving it among eight or more speakers. Here's how:

```
dist = 10
speakerloc_p(dist,0, dist,45, dist,90,
             dist,135, dist,180, dist,225,
             dist,270, dist,315)
dist = 8
path_p(0,dist,0, 1,dist,45, 2,dist,90,
       3,dist,135, 4,dist,180, 5,dist,225,
       6,dist,270, 7,dist,315, 8,dist,360)
rates(0,1, 5,1, 12,10, 20,1)

MIXN(start=0, instart=0, dur=20, inchan=0,
amp=1)
```

The *speakerloc_p* function defines speaker locations in polar coordinates: distance and angle, relative to a point in the center. (Distance is not in any real unit of measure; it's a computational value that loosely corresponds to reality.) The *path_p* function describes the path the sound will take, also in polar form: time, distance, and angle. We can speed up or slow down the rate of movement with the *rate* function, which takes time / value pairs. So the code given above would spin a sound around the room, provided speakers are actually placed in a circle. A joystick interface for controlling the path in real time is in the works.

So from the user's perspective, not much has changed in terms of scorefile syntax. But a great deal has changed under the hood: the scheduler needed some reworking, and the mechanism by which instruments get and send audio data needed complete rewriting.

RTcmix uses a queue / heap scheduler. Note events are stored on a heap, indexed by start time, then placed on a playback queue until done. For the bus and multichannel implementation, queues were added to support the various bus types and channels. This made it possible to insure that instruments play in the right order. In the first example score above, REVERBIT shouldn't try to make any sound before STEREO has provided it with some input! This setup also paves the way for Symmetric Multiprocessing: instruments on the same queue can execute on multiple processors. Work is currently being done in this area.

The structure of the scheduler offers a high degree of flexibility for interface designers. An *interface* is a program that can send RTcmix commands in real time. Most of these programs have graphic user interfaces, such as Luke Dubois' Virtchla[2], a simulation of a classic analog sequencer. Virtchla takes care of event timing; the RTcmix scheduler plays the notes as soon as it receives them. It's also possible for an interface program to send RTcmix a series of events — in effect, a small scorefile. RTcmix can schedule the events, and then play them while receiving more events in real time.

Linux has proven an excellent platform for development. Its speed and reliability served in all aspects of the project. The basic audio API used by RTcmix running under Linux is provided by the commercial OSS (Open Sound System) driver[3]. Their support of the RME Digi96/8 8-channel audio card made it easy to extend RTcmix from stereo to multichannel output. There is an Open Source audio driver for Linux, called ALSA (Advanced Linux Sound Architecture)[4], which is starting to support multichannel audio cards from MIDIMan, Sonorus and RME, including the 24-channel RME Hammerfall. RTcmix currently cannot use the native ALSA API, but support for ALSA is planned.

Current PC architecture is fast enough to realize many musical ideas that the new RTcmix features encourage. We hope that more computer musicians will join us in exploring and extending RTcmix.

John Gibson
johngibson@virginia.edu
University of Virginia

David Topper
topper@virginia.edu

---

[1] RTcmix runs on the Linux and Irix platforms. A port to Mac OS X is under consideration. The RTcmix source code is available from ftp://din.music.virginia.edu/pub/RTcmix.

[2] Virtchla and other RTcmix interfaces are available from http://www.music.columbia.edu/cmix.

[3] http://www.opensound.com

[4] http://www.alsa-project.org