RTcmix -- Using CMIX in Real Time

Brad Garton
Dave Topper
Columbia University Music Department
New York, NY 10027 USA
garton@columbia.edu
topper@panix.com

Abstract

Several computer music "languages" intended to facilitate the coding of sound synthesis and signal-processing algorithms have become popular among practitioners of computer music (CSOUND, CMIX and CMUSIC, for example). Until quite recently, however, low-cost computing power was not available to allow these languages to run in real-time; instead the computer musician generally worked by reading and writing samples to and from disk files and listening to the results when computation was complete.

Within the past few years smaller, general-purpose computers have become powerful enough to run digital sound synthesis and signal-processing algorithms in real time. This demonstration will show the capabilities of RTcmix, a set of extensions and modifications to the CMIX computer music language enabling it to function as a real-time programming environment. Included in the RTcmix package is a dynamic event scheduler and the ability to process disk files and external audio signals in real time. Thus RTcmix can be used to create interactive performance works in addition to providing a fast 'turnaround' for composers building tape-music pieces. Additional aspects of the package include a socket-based protocol for control of RTcmix by external processes, and "backwards compatibility" with all existing CMIX features and instruments. The implementation of RTcmix will be discussed in detail, as well as the rationale behind the design of the language. RTcmix is a unix-based language in the public domain, currently running under both Linux and IRIX.

Background

Given the power of today's general-purpose microprocessors, it is now possible to run "standard" software sound synthesis and signal-processing computer music languages (see [Pope, 1993] for a good overview of several of these languages) in real-time. Although this capability has been available to those with sufficiently powerful computing machinery (or with specialized hardware) for some time, it is only recently that smaller, more affordable computers have had enough power to do real-time audio work with any degree of complexity.

RTcmix is a revised version of CMIX, the computer music language designed by Paul Lansky [Lansky, 1997] and popular among musicians and researchers as a robust development and compositional environment. RTcmix can also function as a replacement for CMIX, able to compile "standard" CMIX

instruments and parse pre-existing CMIX scorefiles that were designed to work with the older, disk-based CMIX. In fact, RTcmix uses the same parser and unit-generator library functions as the basic CMIX package, allowing nearly all extant CMIX instruments to run (disk-based) under RTcmix with only a simple recompilation required.

Design Rationale

Maintaining this "backwards compatibility" was important for several reasons. CMIX has been around for over a decade, and a large amount of development work has taken place. Most of the unit-generator library functions are highly optimized for audio work, and nearly all existing sound synthesis and signal processing algorithms have been implemented in CMIX. CMIX also allows a great deal of flexibility in instrument design, with the basic core of the language being nothing more than a set of C functions that can be called at any point in a CMIX instrument -- the instrument designer is limited only by the capabilities of the C programming language. Basic CMIX also includes Minc, a fully-functional parser/programming language for scorefile interpretation.

RTcmix keeps the Minc parser and the unit-generator library intact, retaining the functionality and optimization of disk-based CMIX. RTcmix is implemented in C++, however, so this functionality is enhanced by the object-oriented capabilities of the C++ language. As with CMIX, RTcmix attempts to place very loose constraints on instrument design, allowing the full capabilities of the C++ (or C) programming language to be used.

Features

The primary difference between RTcmix and CMIX is that RTcmix uses a scheduler to call RTcmix instruments. CMIX assumes out-of-time random access, so CMIX instruments can be invoked at any point during scorefile parsing. RTcmix cannot do this, because the main purpose of the language is to run in real time. RTcmix event scheduling is accomplished through a binary tree, priority-queue dynamic heap [Van Wyk, 1991]. The heap is also designed to do "scheduling-on-the-fly", allowing notes to be scheduled at run-time (usually triggered by an external event, such as a MIDI note being depressed). The RTcmix scheduler is called by the Minc parser, which can be addressed through the standard input (screen/keyboard) or through a TCP/IP socket. Each RTcmix instrument can also instantiate an independent set of TCP/IP sockets for updating of parameters during note execution.

The use of internet socket protocols in RTcmix greatly extends the capabilities of the package. The intention was to design a sound-synthesis and signal-processing engine that could be controlled by a wide variety of interface methods and technologies. RTcmix can be used on a single host or across a network, separating asynchronous interface events from the synchronous, compute-intensive operation of RTcmix instruments.

RTcmix also does real-time signal processing of external audio data, taking input from pre-existing digitized audio disk files or directly from hardware A-D convertors.

Implementation

Each RTcmix instrument is defined as a separate C++ class, inheriting a basic set of variables and member functions (methods) from a base Instrument class. The design of an RTcmix instrument reflects the basic outline of a disk-based CMIX instrument, except that the setup (initialization) portion of a CMIX instrument and the sample-generating loop are separated into two different RTcmix member functions. These member functions must be coded by the instrument designer. The *Instrument::init(p, n_args)* function does the setup work, and is called once when a note using the particular instrument is scheduled. The *Instrument::run()* member function is called repeatedly at run-time. Each time this function is called, it produces up to one buffer's worth of audio samples. These samples are then combined with buffers from all other currently sounding instruments and sent to the D-A convertors to produce sound.

It is worth noting the following about RTcmix instruments:

-- A new Instrument is instantiated and scheduled for each independent note event (the *rtprofile()* function is used to accomplish this). Every note then has an independent state, allowing the note to sound as intended. The constructor for each instrument can be used to establish a socket connection for the instrument, thus allowing independent external parameter control of every note.

-- The number of times the *Instrument::run()* member function is called depends on the length of the note and the size of the output buffer. The scheduler calculates how many output buffers will be needed from a particular note/instrument to produce a given duration. Because the *Instrument::run()* member function will most likely be called many times, the variable "cursamp" is used to keep function tracking (control envelopes, duration-dependent note changes, etc.) consistent. The size of the output buffers for RTcmix is set at run-time, as are most of the other parameters; sampling rate, number of channels, etc. This allows users to trade-off note latency with compute time for different performance circumstances.

-- The scheduling heap contains a list of pointers to *Instrument::run()* member functions, and as each "pops" off the heap the associated *Instrument::run()* member function is called. The only invariant that needs to be maintained in this approach is that every child be indexed by its parent. Heap order is maintained during insertion and deletion by a simple bubble/sift process. This guarantees logarithmic time insertion and deletion, necessary for extensive real-time operation. The index used for heap members is time, expressed as samples. RTcmix instrument scheduling is thus accurate to the sample.

Availability and Performance

RTcmix is in the public domain, and is currently available from http://www.music.columbia.edu/RTcmix (much additional documentation as well as full source code are also available at this site). RTcmix has been ported to SGI platforms running IRIX (5.3 or higher) and Linux running on Pentium machines. Although extensive performance testing of RTcmix is not available at present, approximately 70 non-optimized wavetable oscillators at 44100 samples/sec (mono) could run on a relatively quiescent 133 Mhz R4600 SGI Indy. The nominal input/output latency for a set of 4 simple comb filters processing real-time audio on this same system was about 500 samples (stereo output, .0057 seconds). These numbers could be improved with some judicial optimization of the i/o routines.

Future Extensions

The immediate focus of future RTcmix development work will be to translate more existing CMIX instruments for real-time work. A number of CMIX instruments require semi-random access to an input buffer (granular synthesis-type instruments, for example); input buffering will be added to RTcmix shortly.

Ports to other platforms/operating systems are also a high priority for future work. CMIX itself has been ported to nearly every unix platform available. Because of the concurrent work on both IRIX and Linux in creating RTcmix, most of the OS- and hardware-dependent code has been isolated in a small set of functions.

As stated earlier, RTcmix is meant to be used as a synthesis/signal-processing support engine for a variety of applications. The inclusion of a type of "protocol translator" enabling RTcmix to be controlled by various music-data protocols (MIDI, SKINI [Cook, 1997], ZIPI...) would certainly make this goal much easier to attain.

Bibliography

Cook, P. 1997. "The SKINI 1.0 Protocol." http://www.cs.princeton.edu/~prc.

Dannenberg, R. McAvinney, and Rubine. 1986. "Arctic: A Functional Approach to Real-Time Control." *Computer Music Journal*. Volume 10, Number 4 (Winter, 1986).

Lansky, P. 1997. "The CMIX Home Page." http://www.music.princeton.edu/cmix.

Pope, S. 1993. "Machine Tongues XV: Three Packages for Software Sound Synthesis." *Computer Music Journal*. Volume 17, Number 2 (Summer, 1993).

Van Wyk, J. 1991. *Data Structures and C Programs*. New York: Addison-Wesley.