Extended Csound

Barry Vercoe, Media Lab, M.I.T., bv@media.mit.edu

The realtime performance capacity of Csound has been greatly extended for efficiency and ease of use.

At the ICMC in Glasgow (1990), I announced and demonstrated a new release of Csound® that was tailored to realtime interactive audio synthesis [Vercoe & Ellis, 1990]. Since that time it has enjoyed widespread use, and has also been extended in various directions by independent users who have sensed a personal need.

User-defined extensions have been an important component of the Csound philosophy; they provide fertile ground for users who desire to grow their own operations. However, a difficulty arises in there being no practical way to include every individual idea for the reliable benefit of all. While most users have exploited the openness of Csound's table-driven language compiler to add their own opcodes and audio-processing modules (see "Adding your own C-modules to Csound" in [Vercoe, 1995]), others have chosen to modify the original source code for their own special use. And while most changes are informed and enlightened extensions that clearly serve specific needs, it is difficult to maintain a "dictionary of common usage" that is also a quick and stable ready-reference for today's hurried composer. Moreover, while I sometimes found the time to review, test, and adopt some of these as suitably robust and reliable additions to the standard distribution (with gratefully acknowledged help from John Fitch at Bath), I have found it daunting to keep up this testing and rewriting while trying to honor my own visions and directions. It is no secret that I eventually stopped reading the boards.

As some have guessed, I have not been inactive. The purpose of this paper is to outline the general direction of Csound's recent development, to demonstrate what this can offer the user, and to describe a format that will hopefully increase its general usefulness and its individual flexibility.

1. Recent Directions

Given the realtime performance additions of 1990, one of the most compelling extensions has been to embrace some of the current practices of MIDI (both General and GS), and to do this without wrapping Csound in the straightjacket of commercial synth-think. What we can first learn from the music industry is the importance of quickly changing the functionality of a predefined instrument. Such parameterization has long been a favorite province of computer music instrument design, but its control and switching has generally been vested in awkward score-file pfields and expensive in-line conditionals. A recent Csound addition is a new class of data called *v-variables* (v1, v2, v3,..). These are either string-set, preset, or program set variables whose values can be modified or invoked at times independent of note events. A set of v-values that determines the character of an otherwise generic instrument is called a *program*, and its variables are normally gathered together in a set-up statement performed during Instr 0 initialization:

```
pgminit     pgmnumber, ival1, ival2, ival3, ....
```

The values are I-time variables that have usually been set by other actions, such as automatic loading and numbering of disk-resident function tables:

```
ival2   ftload      "samples/pianoF#3"
ival3   =           ftlptim(ival2)
```

or perhaps by an M x N array representing several of these:

```
ival3   ftsplit     3,2, 0,ipno1, 58,ipno2, 65,ipno3
```

V-variable programs are associated with instrument templates and their instrument numbers by the sequence:

```
instr       1, 2, 3, 4, 5
vprogs      pgmno1, pgmno2, pgmno3
```

which says that pgmno2 (for instance) can grab any free instrument in the given list. A "grab" can occur at any time during a performance; it is called a "program change", and is also the Csound response to a command by that name arriving on any MIDI channel from either a file or a MIDI device (e.g. a sequencer or a keyboard). There can be as many programs simultaneously active as there are instruments to service them; and in the Csound tradition there can be any number of active copies of each program-defined instrument.

As evident above, stored function tables can now be defined and loaded from within the orchestra. This is typically done at orchestra init time in the header (instr 0), but can in fact be scheduled by placing them inside instruments. The repertoire is small, but of the form:

```
i1      ftgen       0, 0, 1024, 10, 1
i2      ftgen       0, 0, 0, 1, "samples/pianoF#3"
i3      ftload      "samples/pianoD#5"
        ftscale     i2, 6
i4      ftstep      0, i2, 60, i3
```

The first shows how any Csound GEN routine can be invoked in this way, the second and third will perform similar loading, the fourth will rescale one of these (for balance), and the last will create a step function (like GEN17) for fast mapping of things like keyboard and drumset splits. All of these leave the table number unspecified; this will be automatically assigned (from 101) and the symbolic references serve to free the user from previously painful checking. Global symbols like *gi1* would allow direct reference from other instruments.

On another front, there have been additions to the sensing and control repertoire. The spectral data types have been revised and extended to include a robust pitch tracker *specptrk*, which uses spectral templates to follow lines in some polyphonic contexts. A less robust but

much more fun tracker is imbedded in a harmonizer module *harmon*. This has the ability to follow your voice and add formant-preserving copies either in parallel or according to knowledge-based guides.

An area I hope to encourage is that of run-time interactive event generation. A Csound orchestra can now be repetitively invoked from a Cscore program, and both score-format and MIDI-format events can be thrown at a running orchestra from timer based co-processes. I expect to spend much additional time in this domain.

2. Playgrounds

The long-term potential of realtime music synthesis with auditory-based sensing and control is difficult to guage and anticipate. Although modern computers are on the brink of providing enough horsepower to make this really interesting, the industry has not seen fit to automatically provide audio I/O with the sound quality we need. While I am impressed with the power of my new Indigo 2, I am frustrated that the audio quality has not kept pace.

I have been fortunate to gain the cooperation of a major DAC manufacturer who cares about these things. Analog Devices (Norwood MA) is also the manufacturer of the industry's currently favorite DSP, the 21060 SHARC (rated at 120 megaflops), and I have found the combination of high-quality audio and heavy horsepower an appealing pair. With their dual interests nicely matching my own needs, I have found ADI's experimental boards to provide a supportive environment for Csound exploration, and many of the innovations described above had their birth and gained their maturity on these platforms. Fortunately, the SHARC C-compiler generates efficient and robust DSP code, so there has been no need to hand-compile any of the Csound you know so well. However, when really pushing the envelope of realtime performance of some of my experimental ideas, I have sometimes found additional horsepower useful, with the result that my own playground is quite strewn with assembler-coded accelerators of flangers, chorusers, and the like. At some future time when off-the-shelf computers provide similar levels of power and sound quality, these Csound experiments will migrate to fully portable C code. But in the meantime this is my favorite playground.

Since everyone deserves a playground, I have given thought to how individual innovation can be supported in a language that also tries to provide a standard. The "add your own C-module" concept is not wrong; it is just impractical to maintain. I have therefore recently split Csound into three parts, with a table-driven translation associated with each part. Part I is the standard public Csound, including the syntax checkers, the orchestra translator and loader, the music monitor and MIDI manager, and the standard opcode repertoire stored in entry1. Part II is my experimental playground, with modules in constant change, sometimes in DSP assembler, and their non-stationary opcodes stored in entry2. Part III is the experimental playground in which anybody else can float their own opcodes in entry3. A standard distribution will comprise all three entry modules, with entry1 complete and entry2 and 3 unfilled. Individuals can add their own modules to entry3, can post and exchange their ideas with others, and generally spread their innovations around. Some individuals may get copies of entry2 opcodes (given the required hardware), so as to exploit my realtime developments; an entry2 user could also participate in entry3 exchanges. Over time there will be a natural migration of entry2 and entry3 modules to the entry1 standard set.

Meanwhile, the speed and audio quality of custom hardware has inspired a raft of experimental developments. In the MIDI area, MIDI files are preprocessed to gain forewarning of program changes and drumset keylists, enabling Csound to preload all wavetables that will be required before synthesis begins. The information is conveyed to Csound via a custom *sysex* imbedded in the Format 0 MIDI file. And whereas normal MIDI is limited to 16 channels per cable, port, or MIDI stream, Csound recognizes up to 6 virtual ports (maximum capacity 96 channels) over a single input device. Further, since musical affect and sensitive performance is primarily conveyed through MIDI controllers, their values (0 - 127) are automatically mapped to a more appropriate range; Csound instruments can reference them symbolically as c1, c7, c10, etc. within the incoming channel. These and similar innovations have proven to provide efficient communication and control in realtime performance situations.

The hardware supporting these experiments has been custom boards with a SHARC DSP, high quality audio I/O, memory and host interface. The first was an ISA card, heavily loaded with SRAM and DRAM; the latest is a PCI card which leaves wavetables on host DRAM and accesses these during performance over PCI DMA. Because this is such an inexpensive Csound accelerator, I am hoping that Analog Devices will make these boards available to a wider community.

3. References

Vercoe, B. & Ellis, D. "Real-Time CSOUND: Software Synthesis with Sensing and Control," *ICMC Proceedings*, Glasgow (1990), pp. 209 - 211.

Vercoe, B. *Csound: A Manual for the Audio Processing System and Supporting Programs with Tutorials*. M.I.T. Media Lab, 1995.