.

A Module System for Csound

Michael Gogins
michael.gogins@gmail.com

May 3, 2017

1 Introduction

This document presents a system for writing Csound orchestras that I have developed over a number of years to support my work as a composer. This document comes with a complete Csound piece that illustrates all features of the system, `module_system_example.csd`. The main purposes of the system are:

1. To enable the maximum possible reuse of Csound instrument and user-defined opcode definitions. Any instrument or opcode created in one piece can immediately be used without modification in any other piece whose score uses the same pfield conventions. Any instrument can be used without modification for standard Csound score input, for MIDI file input, for MIDI real-time input, or for real-time API input from a host application, whether that input consist of note on/note off pairs or whether that input consist of notes with defined durations.

2. To enable the use of an arbitrary number of real-time control channels for instrument definitions.

3. To enable moving modules between various user interface and external control systems, such as CsoundQt, the Csound for Android app, or csound.node.

4. To implement a comprehensive, flexible, and high-quality system of spatialization, including Ambisonic periphony, arbitrary speaker arrangements, spatial reverberation including diffuse and specular early reflections from various surfaces, and distance cues including attenuation, filtering, Doppler effects, and head-related transfer functions. The spatialization system is adapted from Jan Jacob Hofmann's excellent work.

These goals are achieved by strictly following, insofar as the Csound orchestra language permits, that fundamental principle of software engineering known as *encapsulation* or *data hiding*.

A Csound instrument definition or user-defined opcode definition is called a *module*. These modules are "black boxes" that expose only the following standard interfaces for interactions with the rest of Csound. As a result, modules may be defined in include files (**.inc** files) and `#included` as required by the Csound orchestra.

1. Standard pfields. Please note, in comparison with my past convention for pfields, I have changed slightly the meaning of the spatial pfields to more closely follow the conventions of Csound's spatialization opcodes. Also note that the exact same set of pfields and instrument definitions may be used either for score-driven performance, or for real-time, MIDI-driven performance, if the `--midi-key=4 --midi-velocity=5` command-line options are used. The first three pfields are mandatory except for `alwayson` modules, which do not require any pfields.

   p1 Instrument number or MIDI channel number (may be a fraction; the fractional part can be used as a globally unique identifier for the note).

   p2 Start time in beats (by default, a beat is 1 second).

   p3 Duration in beats (a negative value indicates an indefinite duration).

   p4 Pitch as MIDI key number (may be a fraction).

   p5 Loudness as MIDI velocity number (may be a fraction).

   p6 Cartesian x coordinate in meters, running from in front of the listener to behind the listener.

   p9 Cartesian y coordinate in meters, running from the left of the listener to the right of the listener.

   p8 Cartesian z coordinate in meters, running from below the listener to above the listener.

   p9 Phase of the audio signal in radians (in case the instrument instance is, e.g., synthesizing a single grain of sound; this can be useful for phase-synchronous overlapped granular synthesis).

2. Standard outlet and inlet ports. All modules must send or receive audio signals only via the signal flow graph opcodes. The following block of code can be used without modification at the end of almost any instrument definition:

   ```
   absignal[] init 16
   absignal, aspatialreverbsend Spatialize asignal, kfronttoback, klefttoright,
       kbottomtotop
   outletv "outbformat", absignal
   outleta "out", aspatialreverbsend
   ```

   That is what enables the same modules to be used in any sort of Csound orchestra and for any audio output file format or speaker rig. The B-format encoded audio signal can be decoded to mono, stereo, 2-dimensional panning, or full 3-dimensional panning using first, second, or third order decoding.

3. Standard control variables. The modules themselves do not define or directly use input or output channels or zak variables. They use k-rate global control variables with the following naming convention: `gk_InstrumentName_variable_name`.

These variables must be declared just above the instr or opcode definition and initialized there with default values. Normally the value is expected range between 0 and 1, as if they were VST parameters, but this is not necessary; if the range is not [0,1] then comment to document the allowable range.

In addition to only using these standard interfaces, all modules that use function tables must define their own tables within themselves using the `ftgenonce` opcode. Then there is no dependence of the module upon the external score or orchestra header.

Usually, a modular Csound piece will generate a score in the orchestra header, `#include` a number of instrument and effects processing patches, and connect their outlets and inlets using the signal flow graph connect opcode. The signal flow graph will terminate in a module that will perform Ambisonic decoding to the specified speaker rig and/or output soundfile.

Finally, any external controls, such as CsoundQt widgets, OSC signals, MIDI controllers, or whatever, will be received an another custom input module and mapped to the relevant global control variables.

1.1 Real-Time Notes

The exact same instrument definitions can be used for score-driven, MIDI-driven, or API-driven performance; and for notes with predefined durations, or notes that come in note-on/note-off pairs.

To enable this interoperability, bear in mind that the MIDI system in Csound creates `i` statements for MIDI note-on events with an indefinite duration. If the MIDI interop command-line options or opcodes are used, then the MIDI key number is filled in to a configurable pfield (here, p4) and the MIDI velocity number is filled in to another configurable pfield (here, p5). Also, it may well be desirable to create an application that uses the API to perform a MIDI-type real-time performance driven by note-on/note-off pairs. For real-time interoperability to work you must:

a) For API-driven note-on/note-off usage only, create a fractional instrument number to act as a globally unique identifier for each note on `i` statement.

b) For API-driven note-on/note-off usage only, to turn the note off, create a new `i` statement that is identical to the note-on `i` statement except that the sign of p1 is negative.

c) In the instrument definition, detect if p3 is -1 and, if so, reset it to a large value such as 1000000. When the note-off event occurs, Csound will end the instrument instance but continue the release section of any releasing envelope generators. Using a large positive p3 for indefinitely held notes ensures that envelopes and other features that do arithmetic expecting a positive p3 will continue to work properly.

2 An Example

References