.

A Module System for Csound

Michael Gogins
michael.gogins@gmail.com

September 27, 2021

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 predefined 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.

It must be admitted that this system was designed for, and works best with, note-based compositions. Every event in the score is assumed to be a Csound 'i' statement with a standard set of pfields. The instruments are otherwise controlled using Csound control channels that are generally assumed to affect all instances of the same instrument definition.

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. Here is an example of a module (with comments added):

```
; All control channels are exported from global variables using chnexport.
gk_STKBowed_vibrato_level chnexport "gk_STKBosed_vibrato_level", 3
gk_STKBowed_bow_pressure chnexport "gk_STKBowed_bow_pressure", 3
gk_STKBowed_bow_position chnexport "gk_STKBowed_bow_position", 3
gk_STKBowed_vibrato_frequency chnexport "gk_STKBowed_vibrato_frequency", 3
gk_STKBowed_level chnexport "gk_STKBowed_level", 3
gk_STKBowed_vibrato_level init 2.8
gk_STKBowed_bow_pressure init 110
gk_STKBowed_bow_position init 20
gk_STKBowed_vibrato_frequency init 50.2
gk_STKBowed_level init 0
instr STKBowed
; Author: Michael Gogins
i_instrument = p1
i_time = p2
i_duration = p3
i_midi_key = p4
i_midi_velocity = p5
k_space_front_to_back = p6
k_space_left_to_right = p7
k_space_bottom_to_top = p8
i_phase = p9
i_frequency = cpsmidinn(i_midi_key)
; Adjust the following value until "overall amps" at the end of solo performance is
   about -6 dB.
i_overall_amps = 74.5
i_normalization = ampdb(-i_overall_amps) / 2
i_amplitude = ampdb(i_midi_velocity) * i_normalization
k_gain = ampdb(gk_STKBowed_level)
a_signal STKBowed i_frequency, 1.0, 1, gk_STKBowed_vibrato_level, 2,
    gk_STKBowed_bow_pressure, 4, gk_STKBowed_bow_position, 11,
    gk_STKBowed_vibrato_frequency
i_attack = .002
i_sustain = p3
i_release = 0.01
xtratim i_attack + i_release
a_declicking linsegr 0, i_attack, 1, i_sustain, 1, i_release, 0
a_signal = a_signal * i_amplitude * a_declicking * k_gain
#ifdef USE_SPATIALIZATION
a_spatial_reverb_send init 0
a_bsignal[] init 16
a_bsignal, a_spatial_reverb_send Spatialize a_signal, k_space_front_to_back,
    k_space_left_to_right, k_space_bottom_to_top
outletv "outbformat", a_bsignal
outleta "out", a_spatial_reverb_send
#else
a_out_left, a_out_right pan2 a_signal, k_space_left_to_right
outleta "outleft", a_out_left
outleta "outright", a_out_right
#endif
prints "%-24.24s i %9.4f t %9.4f d %9.4f k %9.4f v %9.4f p %9.4f #%3d\n", nstrstr(
    p1), p1, p2, p3, p4, p5, p7, active(p1)
endin
```

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; a negative value means "note-off" for instrument instances running with the same absolute value of p1).

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

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

p4 Pitch as MIDI key number (middle C is 60, may be a fraction).

p5 Loudness as MIDI velocity number (*forte* is 80, may be a fraction).

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

p7 Cartesian y coordinate in meters, running from the left of the listener to the right of the listener. This is the same as stereo pan.

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 instruments defined in this repository suppport both plain stereo output and spatialized output using the following logic. It is assumed that the instrument's output signal is in `a_asignal` at unity gain.

```
i_attack = .002
i_sustain = p3
i_release = 0.01
p3 = i_attack + i_sustain + i_release
a_declicking linsegr 0, i_attack, 1, i_sustain, 1, i_release, 0
a_signal = a_signal * i_amplitude * a_declicking * k_gain
#ifdef USE_SPATIALIZATION
a_spatial_reverb_send init 0
a_bsignal[] init 16
a_bsignal, a_spatial_reverb_send Spatialize a_signal, k_space_front_to_back,
k_space_left_to_right, k_space_bottom_to_top
outletv "outbformat", a_bsignal
outleta "out", a_spatial_reverb_send
#else
a_out_left, a_out_right pan2 a_signal, k_space_left_to_right
outleta "outleft", a_out_left
outleta "outright", a_out_right
#endif
```

In other words, if the macro `USE_SPATIALIZATION` is defined in the orchestra header, the instrument will send Ambisonic B-format audio to a 16 channel outlet named `outbformat`, along with a mono signal to `out` that can be used as a reverb send; if `USE_SPATIALIZATION` is not defined, the instrument will send stereo audio to `outleft` and `outright`.

This 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.

Of course, the declaration of audio connections and "alwayson" instruments will be different for plain stereo versus Ambisonic orchestras. See `SpatializedDrone.csd` for an example of an Ambisonic piece, and comments in `Spatialize.inc` for documentation of the spatialization system.

3. Standard control variables. The modules themselves do not define or directly use input or output channels or zak variables. They use global control variables with the following naming convention: `gk_InstrumentName_variable_name`; `i`-rate variables may also be used. These variables must first be defined using the `chnxport` opcode, and then initialized with a default value, just above the instr or opcode definition. If the range is not [0,1] then comment to document the expected range.

   In addition to using only these standard interfaces, all modules that use function tables must define their own tables using the `ftgen` opcode just above the `instr` block Then there is no dependence of the module upon the external score or orchestra header. The funtion table names must follow the same naming convention.

   Usually, a modular Csound piece will generate a score in external code or in the orchestra header, `#include` a number of instrument and effects processing patches, connect their outlets and inlets using the signal flow graph `connect` opcode, and turn on effects modules with the `alwayson` opcode. For spatialization, 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 use the Csound API to set control channel values and these will automatically and efficiently be mapped to the aforementioned global 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 for MIDI note-on events, Csound creates `i` statements 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, use the `xtratim` opcode to extend the duration of the instrument instance by the sum of the attack and release times, and create a de-clicking envelope using p3 for the sustain section along with the attack and release times you have assigned. When the instance stops performing, either because p3 has elapsed or because a note off event has been received, Csounfd will end the instrument instance but continue the release section of any releasing envelope generators.

2 An Example

```
<CsoundSynthesizer>
<CsLicense>

Author: Michael Gogins
License: Lesser GNU General Public License version 2

This file demonstrates a module system for Csound.

</CsLicense>
<CsOptions>
-d -m163 -odac
</CsOptions>
<CsInstruments>

; Initialize the global variables.

sr = 48000
ksmps = 100
nchnls = 2
0dbfs = 1

; Connect up instruments and effects to create a signal flow graph.

connect "STKBowed",     "outleft",      "ReverbSC",     "inleft"
connect "STKBowed",     "outright",     "ReverbSC",     "inright"

connect "Harpsichord",  "outleft",     "ReverbSC",      "inleft"
connect "Harpsichord",  "outright",    "ReverbSC",      "inright"

connect "ReverbSC",     "outleft",     "Compressor",    "inleft"
connect "ReverbSC",     "outright",    "Compressor",    "inright"

connect "Compressor",   "outleft",     "MasterOutput",  "inleft"
connect "Compressor",   "outright",    "MasterOutput",  "inright"

; Turn on the "effect" units in the signal flow graph.

alwayson "ReverbSC"
alwayson "Compressor"
alwayson "MasterOutput"

#include "Harpsichord.inc"
#include "STKBowed.inc"
#include "ReverbSC.inc"
#include "Compressor.inc"
#include "MasterOutput.inc"

; Override default values of control channels.

gk_Reverb_feedback init .9
gk_Compressor_threshhold init .6

</CsInstruments>
<CsScore>

; Not necessary to activate "effects" or create f-tables in the score!
; Overlapping notes to create new instances of instruments.

i 1 1 5 60 85 .25
i 1 2 5 64 80 .25
i 2 3 5 67 75 .75
i 2 4 5 71 70 .75
e 10
</CsScore>
</CsoundSynthesizer>
```