An Introduction to Csound

Michael Gogins
http://michaelgogins.tumblr.com

Irreducible Productions
New York

14 October 2014

Outline

Summary

- Csound is a general-purpose, user-programmable software sound synthesizer (SWSS).
- It's the earliest descendent of Max Mathews' original software synthesizer, MUSIC I, still in common use.
- It's in the same family as cmusic, Max/MSP, Pure Data, and the SuperCollider server.
- Of all SWSS, Csound has the most unit generators, the most interfaces to external systems and other languages, and runs on the most platforms.
- *Csound preserves almost complete backwards compatibility with older scores and orchestras.*

History I

1958 Max Mathews wrote Music I in IBM 704 assembler at Bell Labs. It was the first (or second, if you count the unsophisticated CSIRAC) digital sound synthesizer.

MUSIC I already had the basic design of unit generators wired together. This design is still used because it is simple, efficient, and represents the linear, time-invariant way engineers think about digital signal processing.

1985 Barry Vercoe, at the Massachusetts Institute of Technology, wrote Csound as one of the first MUSIC N systems in the C programming language (another early system was Paul Lansky's Cmix).

History II

1991 John ffitch, at the University of Bath, ported Csound to the IBM personal computer.

1998-2001 I defined the first form of the Csound API. Csound became a shared library with a well defined application programming interface (API), since extended and ported to many languages.

2001 I created CsoundVST, a VST plugin for studio digital audio workstations. Since then Csound has been made into plugins for many other systems.

History III

2003 MIT originally retained control over redistribution of Csound; at the urging of many people including myself, MIT changed the license to the GNU Lesser General Public License, v2. This permits free redistribution of Csound source code and the development of new products, including commercial ones, derived from Csound code.

2011 John ffitch and Chris Wilson contributed innovative code that runs independent portions of Csound's signal flow graph concurrently on multi-core platforms. Csound is the only SWSS that runs concurrently without any need for annotating or rewriting code.

History IV

2012 Steven Yi and Victor Lazzarini ported Csound to Android and iOS.

2013 Edward Costello and Steven Yi ported Csound to Emscripten (C compiled to JavaScript bytecodes); it runs fast enough in Web browsers.

2013 Victor Lazzarini ported Csound to Google's Portable Native Client runtime (PNaCl); it runs at *native speed* in Chrome.

Agenda I

- I'll play a number of Csound pieces, or segments — chosen to demonstrate capabilities more than musical quality.
- I'll list many Csound features.
- I'll give an overview of Csound's internals.
- I'll offer thoughts on future developments and uses.
- I'll try to answer questions.

Examples I

- Karlheinz Stockhausen, *Studie II*, realized by Joachim Heintz.
  - *Environment* CsoundQt.
- Joseph T. Kung, ***Xanadu***, high-resolution version.
  - *Environment* Csound command.
  - *Features* Various delay lines, frequency modulation, all possible high-resolution options.
- Jeff Livingston, physically modeled Spanish guitar.
  - *Environment* Csound command.
  - *Features* Waveguides.
- Ian McCurdy, physically modeled flute, with Cabbage.

Examples II

  - *Environment* Cabbage.
  - *Features* Waveguides, visual interface, VST interface.
- Evan Allen, ***Set Me Free***.
  - *Environment* Csound command?
  - *Other software* Logic, Pro Tools.
- Ian McCurdy, ***Csound Haiku***.
  - *Environment* Csound6 Android app.
  - *Features* Random splines, hsboscil, alwayson opcode.
- Brian Wong, ***syzygr***.
  - *Environment* blue.
- Steven Yi, ***The Living Ocean***.

Examples III

- *Environment* blue.

- Hector Centeno, ***What Is It?***.
  - *Environment* Csound command.
  - *Features* Samples, FLTK opcodes for visual interface, phase vocoder streaming opcodes.
- Prent Rodgers, ***Rattlesnake Ridge***.
  - *Environment* Csound command?
  - *Features* Based on 72-EDO subsets. Most work with ten note scales, using only six notes at a time.
- J. Garret Frierson, ***cs4live-OnToo***.
  - *Environment* Csound for Live.

Examples IV

- *Features* Using samples I created grooves and used the CS4L AlgoSplice, Spectral Freeze, Spectral Masking, and Fluidic (by Matthew Hines) plugins to effect the vocal sample and several of the instrument samples.

- Aaron Krister Johnson, ***Satiesque***
  - *Environment* Csound command?
  - *Features* Here I use a 46-note per octave scale, some digital piano samples, a Csound front-end scoring system I wrote in Python, and some inspiration from very late 19th century and early 20th century French piano music (Satie, Debussy, Ravel).

Examples V

- Michael Gogins, ***Ladon***.
  - *Environment* Csound6 Android app.
  - *Features* HTML5-defined user interface, Lua opcodes, silencio Lua library for algorithmic composition including ChordSpace based on Tymoczko's work, signal flow graph opcodes.
- Ian McCurdy, ***sonic lava lamp***.
  - *Environment* Cabbage.
- Alexander Tape, ***testsketch***.
  - *Environment* Csound for PNaCl.
- Sean Costello, ***otis***.
  - *Environment* Csound command.

Examples VI

- *Features* Homecooked granular (at the beginning, which is a few tens of thousand of note events of little sine waves), FM, additive, and my own additive technique which modulates each partial with bandlimited noise.
- *Other software* Common Music.

Features of Csound I

- This is an incomplete list!...
- Csound is a general-purpose, unit-generator based, user-programmable software sound synthesis system.
- Csound runs an orchestra language and a score language. *However, scores can optionally be generated directly in orchestra code*.
- Csound renders soundfiles off-line using all facilities of the libsndfile library.
- Csound renders audio in real time using all facilities of PortAudio. *However, Csound optionally has interfaces to various native audio interfaces*.

Features of Csound II

- Csound can receive and send MIDI messages using a set of opcodes. *However, Csound can optionally receive MIDI input in the regular score fields*.
- Audio is processed in fixed length blocks of sample frames. *However, notes and other events can optionally be scheduled with sample-frame accuracy*.
- Audio is processed with double-precision floating point numbers. This is more precise than most other SWSS. It makes a marginally audible difference in some special cases (e.g. feedback filters).

Features of Csound III

- The signal flow graph can optionally be analyzed after it is compiled into independent sub-graphs that are scheduled and run in parallel. There is no need for users to write special code or annotate existing code. Speedups are significant for certain types of pieces.
- Csound performs dynamic allocation of new instrument instances as required by score events. *However, Csound can optionally be run with fixed size pools of instrument instances*.
- Csound has a user-defined software bus, for both audio and control signals, with both internal and external connections.

Features of Csound IV

- Csound has the most extensive library of unit generators. In addition, many unit generator families also provide lower-level toolkits, e.g. there is table and phasor for building oscillators, various filters plus a general-purpose transposed form-II digital filter lattice, a whole bunch of low-level opcodes for time/frequency analysis, morphing, and resynthesis, and so on.
- Csound automatically scans for, and loads, independently compiled plugin unit generators and function table generators. *However, Csound can optionally be configured to load, or not load, specific lists of plugins*.
- Csound can load VST plugins and DSSI plugins.

Features of Csound V

- Csound can run as VST plugins and DSSI plugins.
- Csound can interface with an external JACK network.
- Csound can internally manage connections in an external JACK network. This enables Csound to treat an external JACK synthesizer as a Csound instrument.
- Csound has two facilities for loading and playing SoundFonts.
- Csound has several facilities for time/frequency analysis/resynthesis: pvoc, adsyn, ats, Loris, and especially the phase vocoder streaming (pvs) opcodes by Victor Lazzarini.

Features of Csound VI

- Csound runs all of Perry Cook's STK instruments as opcodes.
- Csound has several facilities for granular synthesis.
- Csound has numerous chaos generator opcodes.
- Csound has high-quality reverb opcodes.
- Csound can run operating system commands.
- Csound can read and write files of any kind, including soundfiles in addition to the main output soundfile.
- Csound has opcodes that can run Lua and Python code.
- Csound code can define opcodes in Lua. These run as fast as compiled C code and can do audio DSP.

Features of Csound VII

- The Faust DSP language can generate Csound opcodes. There are opcodes for embedding Faust code in Csound code, and generating Faust-based opcodes on the fly.
- Csound has low-level UDP socket opcodes.
- Csound has OSC opcodes.
- Csound has low-level serial port opcodes.
- Csound has several systems for spatializing audio, including Ambisonics and vector-based audio panning.
- Csound has a built-in graphical user interface toolkit based on FLTK.

Features of Csound VIII

- Developers have created many graphical front ends for Csound, including CsoundQt, blue, WinXSound, Winsound, Cabbage, and so on.
- A number of algorithmic composition systems interface more or less directly with Csound: CsoundAC and silencio by myself, blue and Score by Steven Yi, Common Music by Rick Taube, AthenaCL by Christopher Ariza, PythonScore by Jacob Joaquin, ACToolbox by Paul Berg, PWGL, OpenMusic from IRCAM, Strasheela by Torsten Anders, Surmulot by Stephane Rollandin, Processing, and others.
- Csound can read and perform MusicXML score files.
- Csound has opcodes for reading and writing image files.

Features of Csound IX

- Csound has been ported to Unix, Linux, Windows, OS X, Android, iOS, JavaScript, PNaCl, and Raspberry Pi.
- The Csound API has interfaces in C, C++, Lisp, Java, JavaScript, Go, Clojure, Python, Lua, and Haskell.
- Csound plugins exist for VST, DSSI, AudioUnits, Max/MSP, Live, and Pure Data.
- Csound supports live coding (incremental compilation of instruments and scores during performance without audio glitches).

Internals I

- Csound consists of two “languages,” and also *embeds* other languages and *is embedded in* other languages.
- I will ignore the score language except to say that it is lists of events (e.g. notes) with some macro capabilities and primitive looping.
- The orchestra language is Turing-complete and has an assembler type syntax: `[result [, result]] opcode [p1 [, p2 ] ...]`
- The orchestra compiler produces an abstract syntax tree which may be modified after compilation and before running. The compiler phases are:

Internals II

- Pre-process macros.
- Lex using Flex-generated lexer.
- Parse using Bison-generated parser.
- Analyze semantics.
- Optimize.
- Compile to a performance-ready runtime state. Instruments are defined as templates to be instantiated. There is roughly one semantic action for each opcode routine.

- The runtime phases are:
  - Run the “init” routines in the orchestra “header”, conceptually instrument 0.

Internals III

- For each audio sample block call `csoundPerformKsmps()`:
  - call `sensevents()` to dispatch all pending file-driven, real-time, and internally produced score events. When events are first dispatched they activate an inactive instance if one is available, or create a new instance.
  - When an instrument instance is first activated, “init” the instance by iterating its opcode list and calling all init routines.
  - Call `kperf()`:

Internals IV

- If multi-threaded and the runtime state has changed call `dag_build` to rebuild the concurrent layers.
  - For each active instrument instance in numerical order (named instruments are assigned numbers as they are compiled):
  - “Perform” the instrument instance by iterating its opcode list and calling all performance routines.
  - If the opcode processes audio, the performance routine must internally loop over each audio sample frame.

Internals V

- For sample frame-accurate scheduling, the audio sample frame loop might start after the 0th frame in the block and at some point begin to output zeros until the end of the block.

- When all scheduled events have been handled, performance ends.
- New score events may be sent to Csound at any time during performance. These go into the current time of the event list, or ahead of the current time.

Internals VI

- New instrument definitions may be sent to Csound at any time during performance. These are compiled into new instrument templates, which are inserted into the master runtime state. Instrument instances that have not finished handling an event are permitted to finish before being replaced.

The Future of Csound I

- At some point, someone might take advantage of the `csoundCompileTree()` function to provid a new syntax, perhaps based on an existing general-purpose programming language, to build an AST that will compile down to Csound runtime states.
- John ffitch and others plan increased use of GPUs for much higher performance.
- I plan to add WebGL to Csound for Android and also to either CsoundQt or a new desktop front end.
- Steven Yi is adding a user-programmable type system to Csound.

The Future of Csound II

- Csound may implement some aspects of functional programming.
- I may embed Lisp and/or Scheme in Csound, if I can only find runtimes that embed easily on Android and the desktop.
- In general, Csound will become an integral part of the HTML5 programming universe. Csound will integrate with Web Audio, and perhaps with JavaScript game engines.
- I expect some Csound front ends to pick up most features of studio digital audio workstations, including scoring, overdubbing, and looping, while remaining completely open source and user-programmable.

The Future of Csound III

- *Csound will continue to provide almost complete backward compatibility for older Csound pieces.*

Questions

- What are are your questions about Csound?
- After that, I have a few questions for you...

Resurces and References I

- Developers' Csound home page, with important links, 2014.
- Csound downloads, 2014.
- Joachim Heintz (Ed.), ***FLOSS Manuals: Csound***, 2014.
- Richard Boulanger's Csound home page, 2014.
- My blog, for CsoundVST downloads and other things, 2014.
- *Richard Boulanger (Ed.),* ***The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming***, 2000.
- SoundCloud, search for tracks tagged "csound", 2014.

Resurces and References II

- SoundCloud, Csound group, 2014.
- YouTube, search for tutorials, demonstrations, and pieces tagged “csound”, 2014.