COSC 4P98 Lecture notes: **CSound**
October 30, 2017
B. Ross

- Reference: mostly chapter 1 of *The CSound Book*, ed. R. Boulanger, MIT Press, 2000.
  - http://csounds.com/chapter1/index.html
  - note: [1.1] means Fig 1.1 in chapter 1
  - Csound examples (local Brock access): http://www.cosc.brocku.ca/Offerings/4P98/csound/local/

- Csound: a sound generation, audio processing, music production system
  - data-flow programming
  - event sequencing
  - Over 20 years old
  - developed by Barry Vercoe (MIT Media Lab)
  - Descended from Music V by Max Matthews in late 1960's (grandfather of computer music)
- Over 1200 uGens (functions)
  - user extendable
  - lots of GUI front-ends, real-time interfaces
- Advantages:
  - power: limited only by time to render compositions
  - batch mode lets you render very complex sounds
  - real-time with midi is possible
  - compositions never become obsolete due to hardware advances
    - like a "program" output
  - Free!
  - Portable!
  - Large user community (university music departments)
- Disadvantages:
  - not as algorithmic as other systems (eg. Supercollider)
  - tends to be a data flow language with modules and some calculations
  - few control primitives (if-then-else)
  - batch-mode tweaking is long, tedious, "trial and error"
  - too large? Difficult to master large library of primitives.
- You should consider Csound if:
  - You are interested in technical study of sound design.
  - You are interested in abstract soundscapes.
  - You are interested in experimentation: microtonal intervals (outside of 12-note scale), polyrythms, algorithmic composition.
- You might not consider Csound if:
  - You are mainly interested in producing conventional music (pop, etc.)
  - You are mainly interested in recording acoustic instruments in a studio.
  - You need to create the music quickly.
  - You want to use standard music notation.

COSC 4P98 Lecture notes: **CSound**
October 30, 2017
B. Ross

- Files:
  - orchestra file (".orc")
    - define your instruments, sound generation
  - score file (".sco")
    - define your performance (notes, events over time)
  - (optional) unified format (".csd")
    - combines orc and sco together
  - (optional) Other files
    - midi
    - sample files
    - others

ORC files: define your instruments

1. header [1.1]
   - define sample and control rates
   - sample: (eg.) 44100
   - control: (eg) 4410
     - much slower, prefix with "k"
     - if its too fast, render time is needlessly lengthened
2. Instrument section [1.2, 1.3, 1.4]
   - each given unique instrument number
   - surround the definition by: instr … endin
   - Syntax:

```
outpt   opcode      args            ; comment

eg.

a1      oscil   10000, 440, 1       ; oscillator playing sine (1) at 440 Hz, 10000 ampl.

                                    ; output sent on "a1" channel (label)
```

- Use labels to route output (akin to wires, patch cables) to send to other opcode inputs
- permits creativity: any argument to a module can be computed from elsewhere!

SCO files: define your composition

1. Table section [1.5]
   - generate waves in function tables (f-tables)
   - done via: (a) builtin GEN routines; (b) sound files (wave, aiff)
   - Different GEN routines generate different shapes (over 40 of them!)
   - eg. GEN 1: wave file
   - eg. GEN 9, 10: combine sine waves (ie. inverse DFT)

```
f       111     0       16      10      1
```

; table 111; load at time 0; 16 entries; GEN 10; full strength = 1; no phase shifts

eg. square wave…

f 112 0 1024 9 1 3 0 3 1 0 9 .333 0

; size=1024; GEN 9; harm 1; ampl 3 phase 0; harm 3; ampl 1; phase 0; harm 9, ampl 1/3, phase 0…

[1.10, ... ,1.15]

2. Note (event) list
    - first 3 fields are always:

      p1 p2 p3

    … reserved for: i# start-time duration

    - other p-fields are fed to instrument referred to, and used any way desired or needed by instrument
        - call them p4, p5… in instrument definition
    - fractional times (seconds)
    - times can be in any order (CSound sorts them)
        - can separate events into instrument-specific sections
        - or have mixed instruments sections… whatever is convenient
    - p4 and beyond are always instrument specific parameters
    - character "." means, use value from line above in same field [1.59, 1.60]
        - "<" means do linear interpolation from earlier values
    - Other statements exist to help set up scores, for example:
        - s – end of section
            - all the table & instrument defns following it are taken together
            - timing is relative to start of the section (ie. time starts at 0)
        - t – tempo
            - can set tempo (speed)

Clipping

- audio values from simultaneous signals are added together to generate a mix of instruments/sounds
- if result is > 32767, there is clipping (overflow)
    - harsh distortion
- Only solution: scale down formulas in CSound definitions, and re-render
- Remember: Csound is applying arithmetic computations. Overflow is often common!

Data rates

- i-rate: note rate (slow)
    - use for note durations, some parameters
- k-rate: control rate (kr, slow too, but faster than i-rate)
    - use for envelopes

COSC 4P98 Lecture notes: **CSound**
October 30, 2017
B. Ross

- a-rate: audio rate (sr, fast – akin to sample rate)
    - filters, oscillator updates
- Your variables in Csound should be prefixed with an I, k, or a
    - this letter will determine how often CSound updates it, according to respective rate above.
    - Yes, this is old fashioned (like early FORTRAN), but that's life.

Examples

- Chapter 1 of The CSound Book is online. Worth working thru different exercises and examples.
- Other tutorials are online as well.
- The CSound book has tons of chapters and examples.
- CSound is a system for **experimentation**. Try things out. Try really unusual ideas! You can get results that you simply cannot obtain with any other system.

Granular synthesis in CSound

- Two operators: granule, grain
- eg.

granule xamp, ivoice, iratio, imod, ithd, ifn, ipshift, igskip, igskip_os, ilength, kgap,

igap-os, kgsize, igsize_os, iatt, idec, [others…]

- xamp: output amplitude
- ivoice # output streams
- iratio: speed of sampling pointer, eg.0.1 means stretch by factor of 10
- imod: direction of pointer (+1, -1, 0)
- ithd: skip sample if amplitude below this (stops silence in wave)
- ifn: ftable #
- ipshift: pitch shift control: 0 = random +/- 1 octave (12 notes)
    - others control pitch shift amounts
- igskip, igskip_os, ilength: control where in ftable (wave table) to process
- kgap, igap_os: gap size, random offset between grains
- kgsize, igsize_os: size of grains, with random offset
- iatt, idec: attack and delay of grains (% grain size)
- etc.

- Note: all the above parameters may be read from tables (samples?), computed by other instruments, calculated with arithmetic expressions,…
- They can change over time. Nothing needs to be static!