CSOUND PROGRAMMING

AIST2010 Lecture 5P

CSOUND

CSound is an environment for sound and music computing

- "A sound compiler"
- Non-interactive score-driven
- Real-time interaction

It started with Max Mathews's "Music1" in 1954 at Bell Labs

- "MusicN" series by Bell Labs, Princeton, Stanford, MIT, etc.
- CSound was created by Barry Vercoe in 1985 at MIT Media Lab
- Read about the journey: https://120years.net/music-n-max-mathews-usa-1957/

CSOUND FILE STRUCTURE

In the old days CSound requires two files

- .orc (orchestra) file: how CSound should create sounds
- .sco (score) file: what sounds CSound should create

*Now only one **.csd** file is needed, with 3 sections*

```
<CsoundSynthesizer>
;all code relating to Csound should be encapsulated between
;<CsoundSynthesizer> and </CsoundSynthesizer>
<CsOptions>
;this section tells Csound how to interact
;with various devices and hardware
</CsOptions>

<CsInstruments>
;this section contains instrument definitions
</CsInstruments>

<CsScore>
;this section tells Csound when and how to perform
;instruments defined in the CsInstruments section above
</CsScore>
</CsoundSynthesizer>
```

A VERY SIMPLE MUSIC

```
<CsoundSynthesizer>
<CsOptions>
-odac                   ; real-time output to DAC
</CsOptions>
<CsInstruments>
sr = 44100              ; Sampling rate
kr = 4410               ; Control rate
nchnls = 2              ; Number of channels
0dbfs = 1               ; Max amplitude (full scale 0db) represented by 1

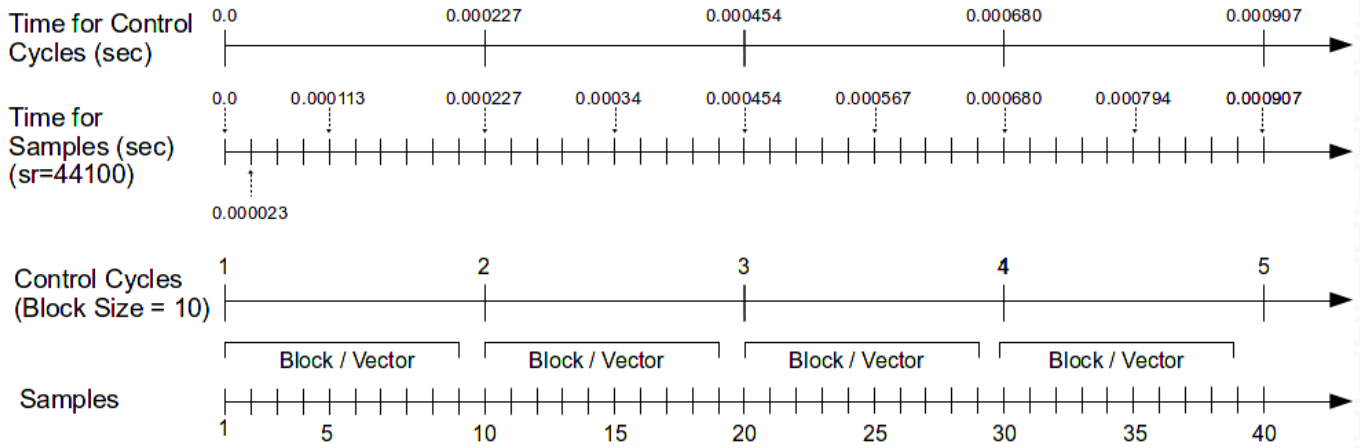
instr 1                 ; Instrument #1
aOut oscili p3, p4      ; Opcode oscili arguments p3,p4, output to aOut
  out aOut              ; Use aOut as the output source
endin                   ; End of instrument #1
</CsInstruments>
<CsScore>
i1 0 1 100              ; Instrument #1, start time 0s, p3=1, p4=100
i1 1 1 200              ; Instrument #1, start time 1s, p3=1, p4=200
i1 2 1 300              ; Instrument #1, start time 2s, p3=1, p4=300
</CsScore>
</CsoundSynthesizer>
```

SAMPLING RATE AND CONTROL RATE

Image from:
http://floss.booktype.pro/csound/a-initialization-and-performance-pass/

The instruments are driven by the sampling rate

- A hidden loop repeats the code in relevant "instrument", during the time specified by the "score"
  - `sr = 44100`



Relationship between sampling rate and control rate

There is also a control rate (k-rate) for data not changing so rapidly

- `kr = 4410` `;usually 10% of sampling rate is sufficient`
- `ksmps = 10` `;it can also be a ratio`

Read more about timing and pass here:

- http://floss.booktype.pro/csound/a-initialization-and-performance-pass/

VARIABLES

Mainly 3 types of numerical variables, differentiated by the first character

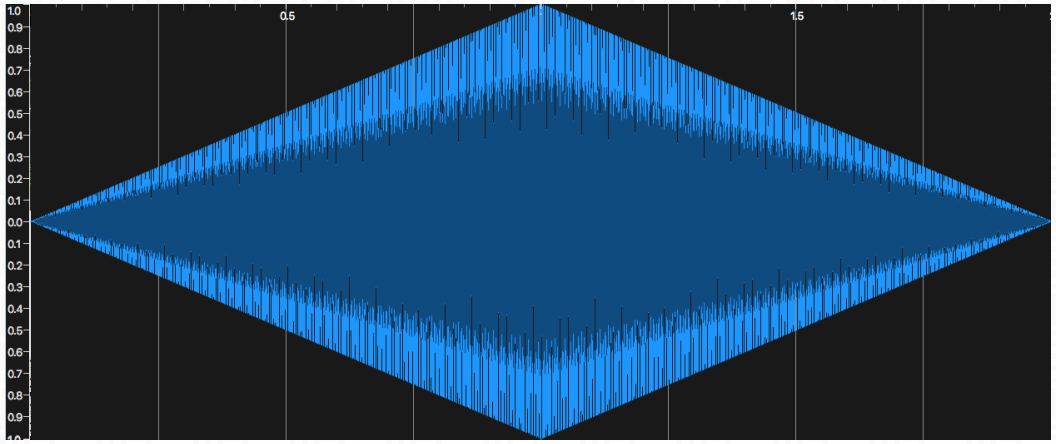
- "**a**"udio variables: one value per sample
  or, view it as a **ksmps** block of values per k-time
- "**k**"ontrol variables: one value per k-time
- "**i**"nit variables: one value per note (in score)

Global variables that can be read by all instruments

- Add "**g**" in front of a/k/i

```
instr 1
  ; ifreq only updated once
  ; during init
  ifreq = 440
  ; kenv is a changing value
  ; updated every k-cycle
  kenv linseg 0, 1, 1, 1, 0
  ; aout is an audio signal:
  ; vector every k-cycle
  aout oscili kenv, ifreq
  out aout
endin
```

Waveform by the above code



INSTRUMENTS AND P-FIELDS

All instruments specified using instr…endin blocks can be called upon in the score

- Different methods to generate sounds, and receive parameters from score
- Besides numbers, you can also use a **"string"** to name the instruments

```
; instruments
instr 1
  aout oscili p4, p5 ;sine
  out aout
endin
instr 2
  aout vco2 p4, p5 ;sawtooth
  out aout
endin
```

```
; score
i1 0 1 0.5 440
; start=0, dur=1, p4=0.5, p5=440
i2 0.5 2 0.8 220
; start=0.5, dur=2, p4=0.8, p5=220
```

OPCODES AND OPERATORS

Opcodes are basic unit generators

- They are like usual functions in other languages, and easily defined by user
  - **opcode** … **endop** block

Two ways to use opcodes, e.g. **`oscili`**

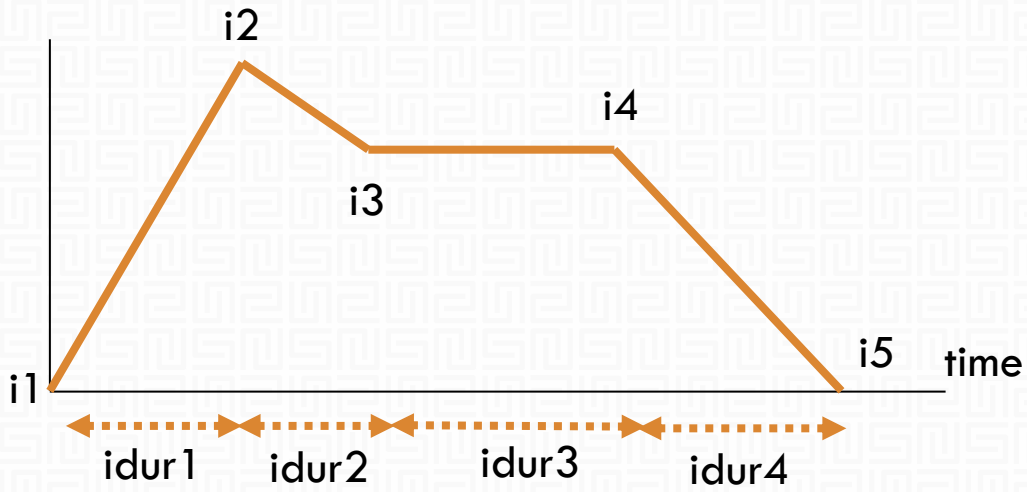
```
aout oscili 1, 440        ;the traditional way
aout = oscili(1, 440)     ;easier for inline opcode
```

Common operators are available, with usual operator precedence

```
+ - * / ^ % = += -= *= /= == >= > < <= !=
```

BUILDING ENVELOPES

Lines can be built for affecting amplitude, frequency, or any values as an input for opcodes



A line built using **linseg**

xvar line ia, idur, ib

- A line from **ia** to **ib** for **idur** seconds

xvar linseg i1, idur1, i2, idur2, i3, idur3, …, in

- Line segments from **i1** to **in** using multiple points with durations

xvar expseg i1, idur1, i2, idur2, i3, idur3, …, in

- Exponential segments similar to **linseg**

You can also use an **oscili** *for building an envelope!*

CONDITIONS AND LOOPS

Not common but exists…

- `if <condition> then`
  `…`
  `else`
  `…`
  `endif`

Looping using while

- `while <condition> do`
  `…`
  `od`
- `until <condition> do`
  `…`
  `od`

ADDITIVE SYNTHESIS

```
…
instr 1 ; Trumpet
  ifreq = p4
  aout1 oscili .141, ifreq
  aout2 oscili .2 , ifreq*2
  aout3 oscili .141, ifreq*3
  aout4 oscili .112, ifreq*4
  aout5 oscili .079, ifreq*5
  aout6 oscili .056, ifreq*6
  aout7 oscili .05 , ifreq*7
  aout8 oscili .035, ifreq*8
  aout9 oscili .032, ifreq*9
  aout10 oscili .02, ifreq*10
  out (aout1+aout2+aout3+aout4+aout5\ ;nextline
      +aout6+aout7+aout8+aout9+aout10)/10
endin
…
i1 0 1 783.99
…
```

```
…
instr 2 ; Nokia phone
  ifreq = p4
  aout1 oscili .045, ifreq
  aout2 oscili .079, ifreq*2
  aout3 oscili .158, ifreq*3
  aout4 oscili .224, ifreq*4
  aout5 oscili .158, ifreq*5
  aout6 oscili .126, ifreq*6
  aout7 oscili .158, ifreq*7
  aout8 oscili .045, ifreq*8
  aout9 oscili .032, ifreq*9
  aout10 oscili .025, ifreq*10
  out (aout1+aout2+aout3+aout4+aout5\ ;nextline
      +aout6+aout7+aout8+aout9+aout10)/10
endin
…
i2 0 1 440
…
```

WAVETABLE SYNTHESIS

Simplifying from additive synthesis with constant sinusoidal waves

```
;; a simple instrument
instr 1
  ifreq = cpsmidinn(p4) ;convert from midi no to hz
  aout oscili 1, ifreq, 1 ;1 is table 1
  out aout
endin
```

```
<CsScore>
f1 0 16384 10 .045 .079 .158 .224 .158 \
              .126 .158 .045 .032 .025 ;; f1 is table 1

i1 0 .1 88 ;; in this notation
i1 +  . 86 ;; + means continue from previous note
i1 + .2 78 ;; . means the same duration as previous note
i1 +  . 80 ;; midi numbers are used here for convenience
i1 + .1 85 ;; and can easily be converted
i1 +  . 83
i1 + .2 74
i1 +  . 76
i1 + .1 83
i1 +  . 81
i1 + .2 73
i1 +  . 76
i1 + .4 81
</CsScore>
```

The table is kept in the score

f1 — function table 1 (any number)

0 — start time = 0

16384 — no. of samples for drawing waveforms

10 — sine wave generator

Then 10 amplitudes for the sine waves

FREQUENCY MODULATION

FM is to introduce a extra wave onto `ifreq`*, with controllable parameters*

Basic vibrato…

```
ifreq = 440
;use kvib for changing vib
;kvib linseg 1, 5, 20
;kmod oscili 1, kvib
kmod oscili 1, 10
aout oscili 0.5, ifreq+kmod
out aout
```

And FM?

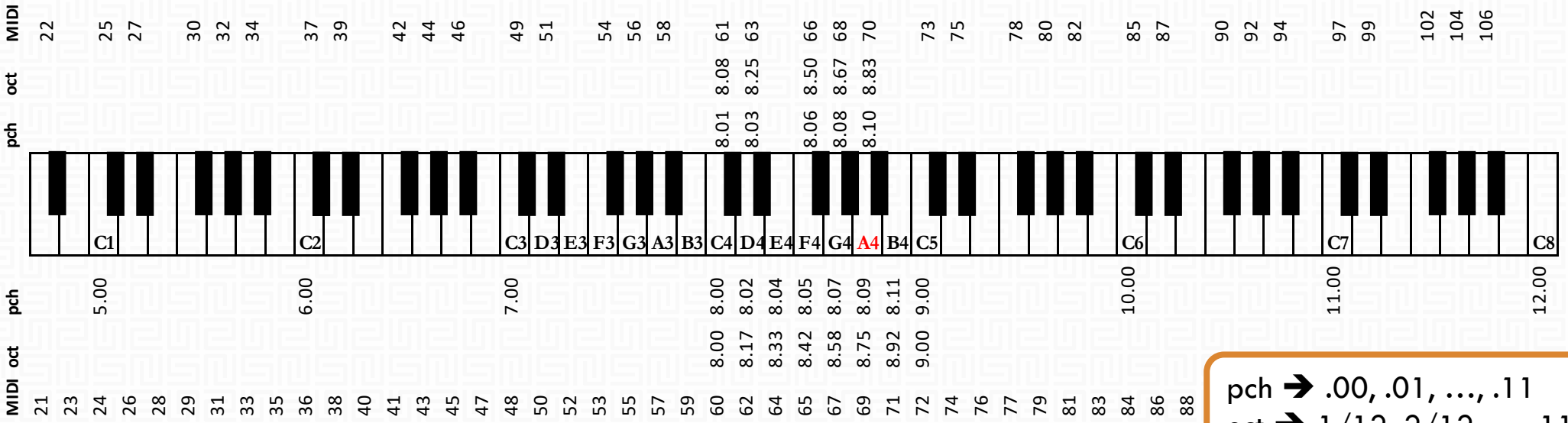
```
icarfreq = 440
iratio = 2
; mod freq depends on ratio
imodfreq = icarfreq*iratio
index = 5
; fmod amp depends on index
imodamp = imodfreq*index
kmod oscili imodamp, imodfreq
aout oscili 0.5, icarfreq+kmod
out aout
```

CSOUND WITH MIDI

CSound by defaults maps MIDI channels 1–16 to instr 1–16

- In ***Configure >> Run***, choose the appropriate MIDI input interface
  - Or you can use the virtual keyboard as well
- ```
  instr 1
    kfreq cpsmidib 2 ;cpsmidib gets midi note messages with pitchbend
    ; a k-type variable is needed since the frequency may change
    iamp ampmidi 0dbfs ;ampmidi gets midi velocity, normalize to 0dbfs
    aout oscili iamp, kfreq
    out aout
  endin
  ```
- An extra line is needed in the score:
  - `f0 300 ; turn on engine for 5 mins, or use any timing you prefer`
- There are many more in CSound MIDI for you to discover!
  ```

HOW TO EASILY NOTATE PITCHES?

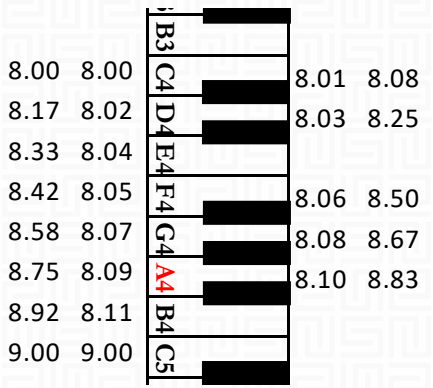


pch ➔ .00, .01, …, .11
oct ➔ 1/12, 2/12, …, 11/12

Pitch converters

- `octpch(pch)` pch to oct
- `cpspch(pch)` pch to cps
- `pchoct(oct)` oct to pch
- `cpsoct(oct)` oct to cps
- `octcps(cps)` cps to oct
- `cpsmidinn(nn)` midi# to cps
- `pchmidinn(nn)` midi# to pch
- `octmidinn(nn)` midi# to oct

cps = cycle per second = Hz



CSOUND API

CSound allow easy collaboration with other programming environments

- CSound API (*in* C)
- CSound on iOS/Android
- CSound in MaxMSP or PureData
- CSound as VST
- Web-based CSound

READ FURTHER

*Chapter 3, "Fundamentals", **CSound**.*

*Chapter 9, "MIDI Input and Output", **CSound**.*

Useful websites:

- https://csound.com/get-started.html
- https://csound.com/docs/manual/index.html
- http://floss.booktype.pro/csound