

Sound synthesis with Csound

```
sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

instr 1    ;play audio file
aout soundin "fox.wav"
    outs aout, aout
endin

instr 2    ;cross-synthesize
icps = p4
ifn = p5      ; Use the "ahh.aiff" sound and "eee.aiff"
ain1 oscil 0.6, p4, ifn
ain2 soundin "fox.wav"  ; Use the "fox.wav" as modulator

isize = 4096
ioverlap = 2
iwin = 3
kbias init 1

aout cross2 ain1, ain2, isize, ioverlap, iwin, kbias
    outs aout, aout
endin
```

Di Giorgio Zucco

Giorgio Zucco

Programmazione con Csound5

Real time,midi,interfacce grafiche con Csound5

INDICE

Prefazione,credits,dedica

Introduzione

Installazione

Cap.1 Panoramica sul linguaggio

1.1 Panoramica sui nuovi opcode della versione 5

1.2 Ambienti integrati,editor

1.3 Una panoramica di WinXound

1.4 Hello world

Cap.2 Primi passi,orchestra,score

2.1 Ampiezza,frequenza

2.2 Render to file,real time

2.3 P fields

2.4 Metodi di gestione eventi

Cap.3 Tecniche fondamentali

3.1 Signal generators : breve panoramica

3.2 Inviluppi,glissandi

3.3 Tecniche di vibrato,tremolo

3.4 Tecniche di spazializzazione

3.5 Utilizzo delle macro

3.6 Elementi random

3.7 Processi generativi

3.8 Gen : panoramica

Cap.4 Midi e Osc

4.1 Introduzione ai midi-opcodes

4.2 Virtual midi keyboard

4.3 Controller midi esterni

4.4 Leggere file midi

4.5 scale microtonali

4.6 OSC e dispositivi mobili(iphone,ipad)

Cap.5 Interfacce grafiche(Fltk,Max/Msp)

5.1 La libreria Fltk

5.2 Fltk panel

5.3 Knobs,sliders,buttons,counter,Joysticks

5.4 Automazioni,controllo midi

5.5 Libreria grafica con qutecsound

5.6 Csound e Max/Msp

Cap.6 Sintesi additiva e sottrattiva

6.1 Sintesi del suono : panoramica introduttiva

6.2 Sintesi Additiva

6.3 Sintesi sottrattiva

Cap.7 Tecniche di modulazione

7.1 Ring modulator

7.3 Sintesi Fm,Pm

7.4 Waveshape

Cap.8 Sample Playback

8.1 Lettura con diskin2

8.2 Loop player

8.3 Mixing vettoriale

8.4 Lettura soundfonts

Cap.9 Sintesi granulare

9.1 simple grain

9.2 syncgrain,syncloop

9.2 partikkel

Cap.10 Modelli fisici

10.1 modelli di strumenti acustici

10.2 L'opcode Mode e la costruzione di risuonatori

10.3 Scanned synthesis

Cap.11 Spectral Processing

11.1 phase vocoder

11.2 pitch shifting,time stretching,morphing

11.3 harmonizer,spectral blur,spectral filter,spectral delay

Cap.12 I processori di segnale

12.1 Introduzione ai Dsp

12.2 Linee di ritardo

12.3 Chorus,flanger,phaser

12.4 Riverberi

12.5 Tecniche di live electronics

Cap.13 Approfondimenti

14.1 Pwg!

14.2 Generazione di score per Csound

14.3 U.D.O opcodes

14.4 Librerie di sintesi (Chowning,Risset)

PREFAZIONE

INTRODUZIONE

.....siamo nell'era degli studi di registrazione tascabili....nell'era in cui è possibile emulare (con un realismo e qualità sbalorditivi) comodamente sul proprio portatile,macchine di culto come un banco Neve,Ssl,Trident,Harrison,ecc,ecc.....i moderni plugin a convoluzione o per modelli fisici,sono raggiungere il calore e il carattere di un compressore Distressor a un ventesimo del costo della macchina originale....

E sul versante del campionamento?il campione di oboe con i suoi generosi 100kb,che girava sul glorioso Akai S900 a 12 bit,oggi ha una sua interfaccia grafica,è campionato a 24 bit in tutte le sue possibilii articolazioni,è disponibile in tre diverse microfonazioni,ecc,ecc.....

E nel campo della sintesi?la potenza di calcolo dei processori attuali spinge sempre di più le software house musicali a produrre sintetizzatori software sempre più evoluti,qui di seguito la presentazione di un celebre Vst (di cui non farò naturalmente il nome) :

- Sintetizzatore semi-modulare in grado di offrire quantità e possibilità sonore senza precedenti
- Nuove combinazioni di sintesi multiple e tecniche di campionamento
- Straordinari miglioramenti apportati nella nuova interfaccia
- scelta sonora tra 1200 preset
- Straordinari effetti, e gestione del wave morphing
- 16 "macro control" liberamente assegnabili ed una nuova sezione dedicata alla modulation ADSR, per avere un rapido e completo controllo del suono
- Possibilità di suddividere l'inviluppo fino ad un massimo di 68 breakpoint
- Funzioni avanzate dedicate alla gestione del suono in surround per ciascun canale

...da aggiungere ancora :

possibilità di aprire infinite istanze del plugin (con un moderno quad core si arriva tranquillamente a gestire 20-25 istanze di un plugin molto complesso,il tutto in tempo reale naturalmente);

Di fronte a simili prodigi,la domanda dei detrattori dei linguaggi di sintesi è sempre la stessa,ha davvero senso dedicarsi ad un linguaggio di programmazione ormai “antico”,che presuppone conoscenze informatiche,che lavora in differita,con tempi di apprendimento lunghissimi,in cui per realizzare una sinusoide di 4 secondi sono necessari diversi minuti di scrittura codice,debug,compilazione e render finale?

Risposta prevedibile.....ma iniziamo con il fare le dovute precisazioni in merito ad alcune inesattezze che gravitano intorno a questo linguaggio :

- non bisogna essere ingegneri informatici per imparare Csound,la sua sintassi,superate le naturali difficoltà di approccio iniziale,è relativamente semplice,costruire uno strumento Csound richiede una mentalità vicina al concetto di sintesi modulare,ogni oggetto Csound (non proprio tutti...) ha un ingresso e un'uscita...
- chi ha avuto esperienze con linguaggi di programmazione tradizionali,come Java ad esempio,ricorderà la complessità di affrontare argomenti come classi,ereditarietà e polimorfismo,metodi e molto altro,non troverete nulla del genere nella programmazione con Csound.
- Csound permette il real time sotto ogni punto di vista,è possibile costruire il proprio sintetizzatore e suonarlo in tempo reale con la propria master keyboard,controllando ogni parametro del suono tramite i knob,slider,ribbon controller del proprio controller midi,Csound può inoltre processare in tempo reale anche segnali microfonici(e per quanto in questo campo lo standard attuale è sempre Max/Msp o SuperCollider,il live con Csound rimane un campo ancora da esplorare e dalle potenzialità notevoli).
- Csound offre nelle performance live un timing perfetto e latenza bassissima,questo grazie alla possibilità di interfacciarsi con qualunque convertitore audio esterno sfruttando ad esempio i driver Asio.
- Csound è disponibile per qualunque piattaforma e sistema operativo (Mac,Linux,pc)
- Dal punto di vista didattico,oltre alle pubblicazioni ufficiali,esiste una documentazione molto estesa sul web,oltre a numerose librerie,composizioni,tutorials per le tecniche più disparate
- Le recenti versioni di Csound offrono la possibilità di costruire interfacce grafiche,raggiungendo un livello di interattività molto simile Max/Msp,la libreria grafica utilizzata si chiama Fltk (fast light toolkit,molto usata su piattaforme Linux),questi opcode furono introdotti per la prima volta dal musicista ricercatore Gabriel Maldonado nella versione CsoundAV (che tra le varie cose,integrava anche le Opengl per poter realizzare grafica 3d).
- Csound non invecchia....non segue mode....non nasce con presupposti estetico musicali specifici,è uno strumento adatto sia al compositore di natura elettroacustica colta,che al Dj(non sono affatto rari gli esempi di utilizzo in aree elettroniche come house,techno,minimal,industrial,glitch,ecc,ecc...)
- Csound viene mantenuto e aggiornato da uno staff internazionale di altissimo livello,in questo preciso istante gli sviluppatori staranno probabilmente compilando una nuova release.
- Csound può essere utilizzato come motore di sintesi per altri linguaggi come Max/Msp e Open music,tramite questi due ambienti grafici è possibile controllare Csound attraverso dati midi,generare score,fare composizione algoritmica.
-dimenticavo.....tra le tante cose.....è gratis.....

INSTALLAZIONE

Per prima cosa procuriamoci l'ultima release di Csound dal sito <http://csound.sourceforge.net/>,da cui scegliere la versione adatta alla nostra macchina o sistema operativo.

L'installazione è molto semplice,l'eseguibile contiene il pacchetto completo di compilatore,numeroso materiale didattico con vari file di esempio (tra cui lo storico brano "Trapped" di Richard Boulanger, che può essere un ottimo inizio per farci un'idea delle possibilità sonore di questo linguaggio),un editor (per scrivere e compilare i nostri sorgenti Csound) chiamato qutecsound,oltre a numerosi file con estensione .dll che fanno riferimento a specifici nuovi opcode.

Cap.1 Panoramica sul linguaggio

- 1.1 Panoramica sui nuovi opcode della versione 5
- 1.2 Ambienti integrati,editor
- 1.3 Una panoramica di WinXound
- 1.4 Hello world

Cap.1

I moduli di Csound,definizione di opcode

Come viene strutturato un programma in Csound?immaginiamo un intestazione con regole formali (un intestazione,la dichiarazione iniziale di un “*nome*” per un preciso compito da svolgere,un istruzione di **chiusura** del processo da realizzare,infine il comportamento nel tempo di un determinato processo (ad esempio la sua durata).Per fare tutto questo,Csound utilizza delle parole chiave in lingua inglese,chiamate **Opcodes**,si tratta di blocchi di codice scritti in *linguaggio C* atti a definire una particolare funzione all’interno del programma.Immaginiamo in astratto il procedimento con cui preparare una torta(!?!?!?!?)...di cui conosciamo gli ingredienti necessari e il modo in cui verranno utilizzati,per arrivare al risultato finale (si..una torta....),l’ultima delle nostre preoccupazioni sarà quella di conoscere il lavoro di produzione di ogni singolo ingrediente! Per realizzare quindi un suono in Csound,il compito del sound designer sarà quello di conoscere una tecnica di sintesi dal punto di vista teorico,e di utilizzare gli opcodes(“ingredienti”) più opportuni per realizzare il suono di sintesi.

La lista degli opcodes in Csound (aggiornato alla versione 5.14,per conoscere la lista esiste un comando chiamato “-z”,si tratta di un *flag*,una linea di comando il cui utilizzo verrà affrontato nei prossimi capitoli) , 1570 opcodes :

ATSaddd	ATSaddnz	ATSbufread	ATScross
ATSinfo	ATSinterpread	ATSpartialtap	ATSread
ATSreadnz	ATSSinnoi	FLbox	FLbutBank
FLbutton	FLcloseButton	FLcolor	FLcolor2
FLcount	FLexecButton	FLgetsnap	FLgroup
FLgroupEnd	FLgroup_end	FLhide	FLhvsBox
FLhvsBoxSetValue	FLjoy	FLkeyIn	FLknob
FLlabel	FLloadsnap	FLmouse	FLpack
FLpackEnd	FLpack_end	FLpanel	FLpanelEnd
FLpanel_end	FLprintk	FLprintk2	FLroller
FLrun	FLsavesnap	FLscroll	FLscrollEnd
FLscroll_end	FLsetAlign	FLsetBox	FLsetColor
FLsetColor2	FLSetFont	FLsetPosition	FLsetSize
FLsetSnapGroup	FLsetText	FLsetTextColor	FLsetTextSize
FLsetTextType	FLsetVal	FLsetVal_i	FLsetVali
FLsetsnap	FLshow	FLslidBnk	FLslidBnk2
FLslidBnk2Set	FLslidBnk2Setk	FLslidBnkGetHandle	FLslidBnkSet
FLslidBnkSetk	FLslider	FLtabs	FLtabsEnd
FLtabs_end	FLtext	FLupdate	FLvalue
FLvkeybd	FLvslidBnk	FLvslidBnk2	FLxyin
MixerClear	MixerGetLevel	MixerReceive	MixerSend
MixerSetLevel	MixerSetLevel_i	OSCinit	OSClisten
OSCrecv	OSCsend	STKBandedWG	STKBeeThree
STKBlowBotl	STKBlowHole	STKBowed	STKBrass
STKClarinet	STKDrummer	STKFMVoices	STKFlute
STKHeavyMetl	STKMandolin	STKModalBar	STKMoog
STKPercFlut	STKPlucked	STKResonate	STKRhodey
STKSaxofony	STKShakers	STKSimples	STKSitar

STKStifKarp	STKTubeBell	STKVoicForm	STKWhistle
STKWurley	a	abs	active
add	adsr	adsyn	adsynt
adsynt2	aftouch	alpass	alwayson
ampdb	ampdbfs	ampmidi	ampmidid
and	areson	aresonk	atone
atonek	atonex	babo	balance
bamboo	barmodel	bbcutm	bbcuts
betarand	bexprnd	bformdec	bformdec1
bformenc	bformenc1	binit	biquad
biquada	birnd	bqrez	butbp
butbr	buthp	butlp	butterbp
butterbr	butterhp	butterlp	buzz
cabasa	cauchy	cauchyi	ceil
cent	cggoto	chanctrl	changed
chani	chano	chebyshevpoly	chn_S
chn_a	chn_k	chnclear	chnexport
chnget	chnmix	chnparams	chnrecv
chnsend	chnset	chuap	cigoto
ckgoto	clear	clfilt	clip
clockoff	clockon	cngoto	cogoto
comb	compress	connect	convle
convolve	cos	cosh	cosinv
cps2pch	cpsmidi	cpsmidib	cpsmidinn
cpsoct	cpspch	cpstmid	cpstun
cpstuni	cpxpoch	cpuprc	cross2
crossfm	crossfmi	crossfmpm	crossfmpmi
crosspm	crosspmi	crunch	ctlchn
ctrl14	ctrl21	ctrl7	ctrlinit
cuserrnd	dam	date	dates
db	dbamp	dbfsamp	dcblock
dcblock2	dconv	delay	delay1
delayk	delayr	delayw	deltap
deltap3	deltapi	deltapn	deltapx
deltapxw	denorm	diff	diskgrain
diskin	diskin2	disppfft	display
distort	distort1	div	divz
doppler	downsamp	dripwater	dumpk
dumpk2	dumpk3	dumpk4	duserrnd
endin	endop	envlpx	envlpxr
ephasor	eqfil	event	event_i
exitnow	exp	expcurve	expon
exrand	exrandi	expseg	expsega
expsegb	expsegba	expsegr	fareylen
fareyleni	ficlose	filebit	filelen
filenchnls	filepeak	filesr	filevalid
filter2	fin	fini	fink
fiopen	flanger	flooper	flooper2
flooper3	floor	fluidAllOut	fluidCci
fluidCCK	fluidControl	fluidEngine	fluidLoad
fluidNote	fluidOut	fluidProgramSelect	fluidSetInterpMethod
fmb3	fmbell	fmmetal	fmpercfl
fmrhode	fmvoice	fmwurlie	fof
fof2	fofilter	fog	fold
follow	follow2	foscil	foscili
fout	fouti	foutir	foutk
fprintks	fprints	frac	freeverb
ftchnls	ftconv	ftcps	ftfree
ftgen	ftgenonce	ftgentmp	ftlen
ftload	ftloadk	ftlptim	ftmorf
ftsav	ftsavek	ftsr	gain
gainslider	gauss	gaussi	gbuzz
getcfg	gogobel	goto	grain

grain2	grain3	granule	guiro
harmon	harmon2	harmon3	harmon4
hilbert	hrtfer	hrtfmove	hrtfmove2
hrtfstat	hsboscil	hvs1	hvs2
hvs3	i	igoto	ihold
imagecreate	imagefree	imagegetpixel	imageload
imagesave	imagesetpixel	imagesize	in
in32	inch	inh	init
initc14	initc21	initc7	inleta
inletf	inletk	inletkid	ino
inq	inrg	ins	insglobal
insremot	instr	int	integ
interp	invalue	inx	inz
jitter	jitter2	jspline	k
kgoto	ktableseg	la_i_add_mc	la_i_add_mr
la_i_add_vc	la_i_add_vr	la_i_assign_mc	la_i_assign_mr
la_i_assign_t	la_i_assign_vc	la_i_assign_vr	la_i_conjugate_mc
la_i_conjugate_mr	la_i_conjugate_vc	la_i_conjugate_vr	la_i_distance_vc
la_i_distance_vr	la_i_divide_mc	la_i_divide_mr	la_i_divide_vc
la_i_divide_vr	la_i_dot_mc	la_i_dot_mc_vc	la_i_dot_mr
la_i_dot_mr_vr	la_i_dot_vc	la_i_dot_vr	la_i_get_mc
la_i_get_mr	la_i_get_vc	la_i_get_vr	la_i_invert_mc
la_i_invert_mr	la_i_lower_solve_mc	la_i_lower_solve_mr	la_i_lu_det_mc
la_i_lu_det_mr	la_i_lu_factor_mc	la_i_lu_factor_mr	la_i_lu_solve_mc
la_i_lu_solve_mr	la_i_mc_create	la_i_mc_set	la_i_mr_create
la_i_mr_set	la_i_multiply_mc	la_i_multiply_mr	la_i_multiply_vc
la_i_multiply_vr	la_i_norm1_mc	la_i_norm1_mr	la_i_norm1_vc
la_i_norm1_vr	la_i_norm_euclid_mc	la_i_norm_euclid_mr	la_i_norm_euclid_vc
la_i_norm_euclid_vr	la_i_norm_inf_mc	la_i_norm_inf_mr	la_i_norm_inf_vc
la_i_norm_inf_vr	la_i_norm_max_mc	la_i_norm_max_mr	la_i_print_mc
la_i_print_mr	la_i_print_vc	la_i_print_vr	la_i_qr_eigen_mc
la_i_qr_eigen_mr	la_i_qr_factor_mc	la_i_qr_factor_mr	la_i_qr_sym_eigen_mc
la_i_qr_sym_eigen_mr	la_i_random_mc	la_i_random_mr	la_i_random_vc
la_i_random_vr	la_i_size_mc	la_i_size_mr	la_i_size_vc
la_i_size_vr	la_i_subtract_mc	la_i_subtract_mr	la_i_subtract_vc
la_i_subtract_vr	la_i_t_assign	la_i_trace_mc	la_i_trace_mr
la_i_transpose_mc	la_i_transpose_mr	la_i_upper_solve_mc	la_i_upper_solve_mr
la_i_vc_create	la_i_vc_set	la_i_vr_create	la_i_vr_set
la_k_a_assign	la_k_add_mc	la_k_add_mr	la_k_add_vc
la_k_add_vr	la_k_assign_a	la_k_assign_f	la_k_assign_mc
la_k_assign_mr	la_k_assign_t	la_k_assign_vc	la_k_assign_vr
la_k_conjugate_mc	la_k_conjugate_mr	la_k_conjugate_vc	la_k_conjugate_vr
la_k_current_f	la_k_current_vr	la_k_distance_vc	la_k_distance_vr
la_k_divide_mc	la_k_divide_mr	la_k_divide_vc	la_k_divide_vr
la_k_dot_mc	la_k_dot_mc_vc	la_k_dot_mr	la_k_dot_mr_vr
la_k_dot_vc	la_k_dot_vr	la_k_f_assign	la_k_get_mc
la_k_get_mr	la_k_get_vc	la_k_get_vr	la_k_invert_mc
la_k_invert_mr	la_k_lower_solve_mc	la_k_lower_solve_mr	la_k_lu_det_mc
la_k_lu_det_mr	la_k_lu_factor_mc	la_k_lu_factor_mr	la_k_lu_solve_mc
la_k_lu_solve_mr	la_k_mc_set	la_k_mr_set	la_k_multiply_mc
la_k_multiply_mr	la_k_multiply_vc	la_k_multiply_vr	la_k_norm1_mc
la_k_norm1_mr	la_k_norm1_vc	la_k_norm1_vr	la_k_norm_euclid_mc
la_k_norm_euclid_mr	la_k_norm_euclid_vc	la_k_norm_euclid_vr	la_k_norm_inf_mc
la_k_norm_inf_mr	la_k_norm_inf_vc	la_k_norm_inf_vr	la_k_norm_max_mc
la_k_norm_max_mr	la_k_qr_eigen_mc	la_k_qr_eigen_mr	la_k_qr_factor_mc
la_k_qr_factor_mr	la_k_qr_sym_eigen_mc	la_k_qr_sym_eigen_mr	la_k_random_mc
la_k_random_mr	la_k_random_vc	la_k_random_vr	la_k_subtract_mc
la_k_subtract_mr	la_k_subtract_vc	la_k_subtract_vr	la_k_t_assign
la_k_trace_mc	la_k_trace_mr	la_k_upper_solve_mc	la_k_upper_solve_mr
la_k_vc_set	la_k_vr_set	lfo	limit
line	linen	linenr	lineto
linrand	linseg	linsegb	linsegr
locsend	locsig	log	log10

logbtwo	logcurve	loop_ge	loop_gt
loop_le	loop_lt	loopseg	loopsegp
looptseg	loopxseg	lorenz	loscil
loscil3	loscilx	lowpass2	lowres
lowresx	lpf18	lpform	lpfreson
lphasor	lpinterp	lposcil	lposcil3
lposcila	lposcilsa	lposcilsa2	lpread
lpreson	lpshold	lpsholdp	lpslot
lua_exec	lua_iaopcall	lua_iaopcall_off	lua_ikopcall
lua_ikopcall_off	lua_iopcall	lua_iopcall_off	lua_opdef
mac	maca	madsr	mandel
mandol	marimba	massign	max
max_k	maxabs	maxabsaccum	maxaccum
maxalloc	maxk	mclock	mdelay
median	mediank	metro	midglobal
midic14	midic21	midic7	
midichannelaftertouch			
midichn	midicontrolchange	midictrl	mididefault
midiin	midinoteoff	midinoteoncps	midinoteonkey
midinoteonoct	midinoteonpch	midion	midion2
midfout	midipgm	midipitchbend	midipolyaftertouch
midiprogramchange	miditempo	midremot	min
minabs	minabsaccum	minaccum	mincer
mirror	mod	mode	modmatrix
monitor	moog	moogladder	moogvcf
moogvcf2	moscil	mpulse	mrtmsg
mul	multitap	mute	mutex_lock
mutex_locki	mutex_unlock	mutex_unlocki	mxadsr
nestedap	nlalp	nlfilt	noise
not	noteoff	noteon	noteondur
noteondur2	notnum	nreverb	nrpn
nsamp	nstrnum	ntrpol	octave
octcps	octmidi	octmidib	octmidinn
octpch	opcode	or	oscbnk
oscil	oscill	oscilli	oscil3
oscili	oscilikt	osciliktp	oscilikts
osciln	oscils	oscilx	out
out32	outc	outch	outh
outiat	outic	outic14	outipat
outipb	outipc	outkat	outkc
outkcl4	outkpat	outkpb	outkpc
outleta	outletf	outletk	outletkid
outo	outq	outql	outq2
outq3	outq4	outrg	outs
outs1	outs2	outvalue	outx
outz	p	pan	pan2
pareq	partials	partikkkel	partikkelsync
passign	pcauchy	pchbend	pchmidi
pchmidib	pchmidinn	pchoct	pconvolve
pcount	pdclip	pdhalf	pdhalfy
peak	pgmassign	pgmchn	phaser1
phaser2	phasor	phasorbnk	pindex
pinkish	pitch	pitchac	pitchamdf
planet	pluck	poisson	polyaft
polynomial	pop	pop_f	port
portk	poscil	poscil3	pow
powershape	powoftwo	prealloc	prepiano
print	printf	printf_i	printk
printk2	printfks	prints	product
pset	ptrack	push	push_f
puts	pvadd	pbufread	pvcross
pvinterp	pvoc	pvread	pvs2tab
pvsadsyn	pvsanal	pvsarp	pvsbandp

pvsbandr	pvsbin	pvsblur	pvsbuffer
pvsbufread	pvsbufread2	pvscale	pvscent
pvscross	pvsdemix	pvsdiskin	pvsdisp
pvsenvftw	pvsfilter	pvsfread	pvsfreeze
pvsftr	pvsftw	pvsfwrite	pvsgain
pvshift	pvsifd	pvsin	pvsinfo
pvsinit	pvslock	pvsmaska	pvsmix
pvssmooth	pvsmorph	pvsosc	pvsout
pvspitch	pvstanal	pvcstencil	pvsvocab
pvs warp	pvsynth	qinf	qnan
rand	randh	randi	random
randomh	randomi	rbjeq	readclock
readk	readk2	readk3	readk4
readks	reinit	release	remoteport
remove	repluck	reson	resonk
resonr	resonx	resonxk	resony
resonz	resyn	reverb	reverb2
reverb sc	rewindscore	rezzy	rigoto
r ireturn	rms	rnd	rnd31
round	rspline	rtclock	s16b14
s32b14	samphold	sandpaper	scale
scanhammer	scans	scantable	scantu
schedkwhen	schedkwhennamed	schedule	schedwhen
scoreline	scoreline_i	seed	sekere
semitone	sense	sensekey	seqtime
seqtime2	setksmps	setscorepos	sfilist
sfinstr	sfinstr3	sfinstr3m	sfinstrm
sfload	sflooper	sfpassign	sfplay
sfplay3	sfplay3m	sfplaym	sfplist
sf preset	shaker	shl	shr
sin	sinh	sininv	sinsyn
sleighbells	slider16	slider16f	slider16table
slider16tablef	slider32	slider32f	slider32table
slider32tablef	slider64	slider64f	slider64table
slider64tablef	slider8	slider8f	slider8table
slider8tablef	sliderKawai	sndload	sndloop
sdn warp	sdn warpst	soundin	soundout
soundouts	space	spat3d	spat3di
spat3dt	spd ist	specaddm	specdiff
spec disp	specfilt	spec hist	specptrk
specscal	specsum	spectrum	splitrig
sprintf	sprintfk	sp send	sqrt
stack	statevar	st ix	strcat
strcatk	strchar	strchark	strcmp
strcm pk	strcpy	strcpyk	streson
strget	str index	str indexk	strlen
str len	str lower	str lowerk	str rindex
str rindexk	str set	str sub	str subk
strtod	strtodk	str tol	str tolk
strupper	strupperk	sub	sub instr
sub instr init	sum	sv filter	syncgrain
sync loop	sync phasor	system	system_i
tab	tab2pvs	tab_i	table
table3	table copy	table filter	table filteri
table gpw	table i	table icopy	table igpw
table ikt	table imix	table iw	table kt
table mix	tabl eng	tabl era	table seg
table shuffle	table shufflei	table w	table wa
table wkt	table xkt	table xseg	tab morph
tab morpha	tab morphak	tab morphi	tab play
tab rec	tab ref	tab sum	tab w
tab w_i	tambourine	tan	tanh
tan inv	tan inv2	tb0	tb0_init

tb1	tb10	tb10_init	tb11
tb11_init	tb12	tb12_init	tb13
tb13_init	tb14	tb14_init	tb15
tb15_init	tb1_init	tb2	tb2_init
tb3	tb3_init	tb4	tb4_init
tb5	tb5_init	tb6	tb6_init
tb7	tb7_init	tb8	tb8_init
tb9	tb9_init	tbvcf	tempest
tempo	temposcal	tempoval	tigoto
timedseq	timeinstk	timeinsts	timek
times	timeout	tival	tlineto
tone	tonek	tonex	tradsyn
trandom	transeg	transegb	transegr
trcross	trfilter	trhighest	trigger
trigseq	trirand	trlowest	trmix
trscale	trshift	trssplit	turnoff
turnoff2	turnon	unirand	upsamp
urd	vadd	vadd_i	vaddv
vaddv_i	vaget	valpass	vaset
vbap16	vbap16move	vbap4	vbap4move
vbap8	vbap8move	vbapsinit	vbapz
vbapzmove	vcella	vco	vco2
vco2ft	vco2ift	vco2init	vcomb
vcopy	vcopy_i	vdel_k	vdelay
vdelay3	vdelayk	vdelayx	vdelayxq
vdelayxs	vdelayxw	vdelayxwq	vdelayxws
vdivv	vdivv_i	vecdelay	veloc
vexp	vexp_i	vexpseg	vexpv
vexpv_i	vibes	vibr	vibrato
vincr	vlimit	vlinseg	vlowres
vmap	vmirror	vmult	vmult_i
vmultv	vmultv_i	voice	vosim
vphaseseg	vport	vpow	vpow_i
vpowv	vpowv_i	vpvoc	vrandh
vrandi	vsubv	vsubv_i	vtaba
vtabi	vtabk	vtable1k	vtablea
vtablei	vtablek	vtablewa	vtablewi
vtablewk	vtabwa	vtabwi	vtabwk
vwrap	waveset	weibull	wgbow
wgbowedbar	wgbrass	wgclar	wgflute
wgpluck	wgpluck2	wguide1	wguide2
wrap	wterrain	xadsr	xin
xor	xout	xscanmap	xscans
xscansmap	xscanu	xtratim	xyin
zacl	zakinit	zamod	zar
zarg	zaw	zawm	zfilter2
zir	ziw	ziwm	zkcl
zkmod	zkr	zkw	zkwm

Panoramica sui nuovi opcode della versione 5

La versione attuale di Csound è numerata 5.14, dalla storica versione di Barry Vercoe (1985) sono stati fatti passi notevoli dal punto di vista della versatilità del linguaggio, oltre all'impostazione definirei “in differita” delle versioni precedenti (lavorare con orchestra e score separati con render su disco). Le versioni più recenti

offrono innumerevoli strumenti per il real time, costruzione di interfacce grafiche, lettura di qualunque tipo di file audio, integrazione perfetta con qualunque hardware audio e midi e molto altro.

Csound 5.13 introduce le seguenti caratteristiche :

- PortAudio con supporto per driver Asio (che garantisce bassa latenza per il real time);
- PortMidi per controllare Csound da qualunque controller midi esistente sul mercato;
- Fltk (fast light toolkit) , libreria grafica per oggetti come slider, controlli rotativi, pulsanti, joystick, ecc
- FluidSynth , modulo costituito da vari opcode per la gestione dei file in formato soundfont
- Python opcodes per inizializzare l'interprete del linguaggio python da un'orchestra
- OSC opcodes : Osc (open sound control) è un protocollo elettronico alternativo a quello Midi, permette lo scambio di dati, tra strumenti musicali o computer, ad altissima velocità, attualmente non ancora supportato dalla maggior parte di hardware musicale in commercio;
- ATS, Loris, PVS : sono opcodes di nuova generazione per l'analisi e risintesi anche in tempo reale;
- STK opcodes (Perry Cook's original Synthesis Toolkit in C++) : libreria open source di sintesi per modelli fisici;
- CsoundAPI : Application Programming Interface, le Api di Csound permettono l'utilizzo di questo potentissimo motore di sintesi all'interno dei più svariati linguaggi di programmazione (Java, Lisp, C++, Python, Lua);

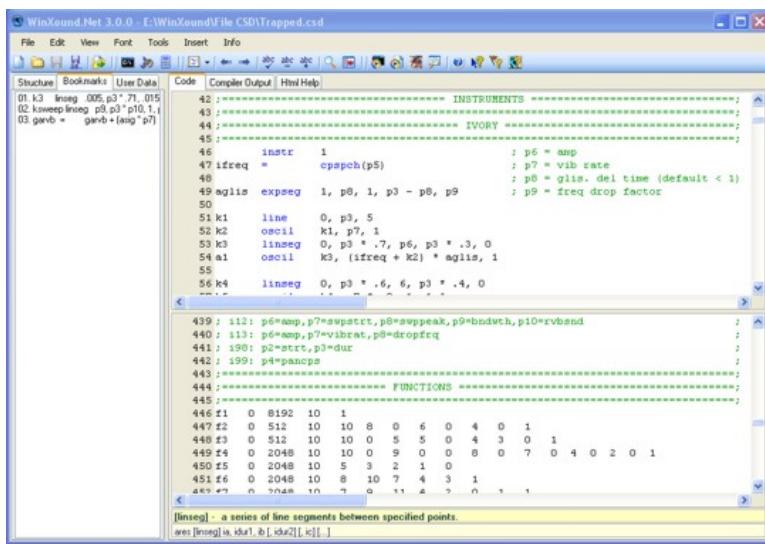
Ambienti integrati, editor

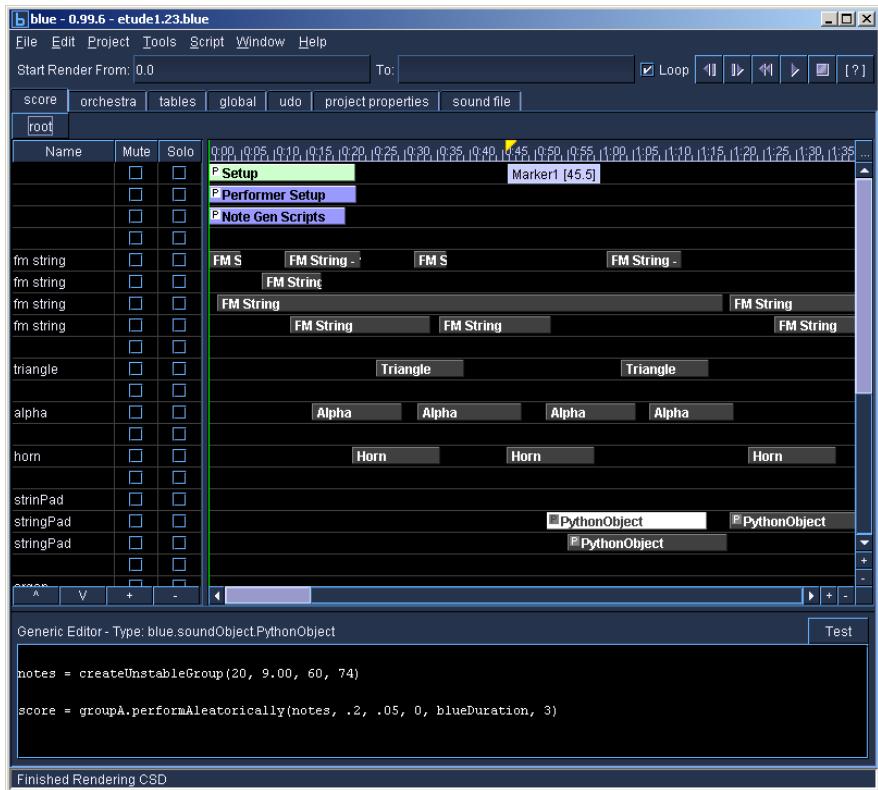
La versione attuale di Csound5 contiene già un eccellente editor chiamato **qutecsound** compreso nel pacchetto d'installazione, qutecsound offre tutte le features tipiche di un moderno editor per Csound e molto altro :

- Sintassi colorata
- Strumenti per ottimizzare e velocizzare le tecniche di analisi e risintesi
- Creazione di widjet (oggetti per interfacce grafiche)
- Accesso immediato alla sintassi di tutti gli opcode
- Tutorials delle tecniche più conosciute

Esistono numerosi altri editor per questo linguaggio, alcuni di questi non sono stati mantenuti e aggiornati negli anni (problema tipico legato al mondo dei software open source), altri sono diventati dei veri e collaudati standard per numerosi csounders sparsi nel mondo. L'elenco seguente è puramente indicativo e non tiene conto di storici editor (come Cecilia e altri) attualmente non più supportati, vale la pena citare :

- Winxound di Stefano Bonetti, all platforms
- Blue di Steven Yi, è un sistema che ricorda l'interfaccia di un sequencer multitraccia, offre numerosi strumenti per la composizione e il controllo degli score, all platforms
- Cabel, una versione sperimentale simile ai sistemi grafici come Max, Open Music, Pure data, attualmente l'ultima release è del 2006 e non si hanno notizie sui possibili sviluppi.
- L'interfaccia di WinXound editor :





Una breve panoramica di WinXound

Si tratta di un ottimo editor sviluppato e in continuo aggiornamento da Stefano Bonetti, inizialmente nato solo per macchine Windows e oggi disponibile anche per sistemi Mac e Linux, è una versione ottimizzata per lavorare con Csound5 (con le vecchie versioni potrebbe essere incompatibile), offre le seguenti caratteristiche :

- scrivere e modificare file Csound, Python, Lua files (csd, orc, sco, py, lua) con sintassi colorata;
- lavora con Csound, CsoundAV (la versione di G.Maldonado), CsoundAC (versione sperimentale per la composizione algoritmica di Michael Gogins), compilatori Python and Lua ;
- permette di interfacciarsi a sua volta con altri editor (come QuteCsound);
- offre tool molto pratici per le tecniche di analisi e risintesi, in cui è possibile importare un file audio (wav,aiff) e creare in pochi istanti un file di analisi con Csound per successivi lavori di risintesi;
- integra il manuale ufficiale di Csound a cui è possibile accedere in modo immediato direttamente nelle operazioni di scrittura del codice;
- importa e converte automaticamente i file separati .orc e .sco in un file .csd
- Linee numerate, è molto importante nei casi in cui il compilatore segnali errori di sintassi
- marcatori;

- integra una collezione di U.d.o opcodes, si tratta di opcode non contenuti nella versione ufficiale di Csound, sono opcode non “compilati” ma scritti direttamente in linguaggio Csound.

Dal menu – File – Settings è possibile configurare l’editor in modo molto dettagliato, ad esempio possiamo associare un software di editing audio esterno (come l’open source Audacity), possiamo indicare la directory di CsoundAV, per le operazioni di analisi e risintesi (Menu – Tools – Csound Analysis) possiamo indicare la directory di una cartella in cui sono contenuti i nostri samples o dove vogliamo che Csound scriva i file di analisi.

Configurazione iniziale :

- Windows : dal menu – File – Settings – Compiler Settings, troverete la voce “*Csound Compiler Default Flags*”, si tratta delle impostazioni iniziali relative ai comandi Csound per il render dei file, di default troviamo :

-B4096 --displays –asciidisplay

Si consiglia di non modificare questa voce, il flag –B riguarda il buffer di memoria hardware, utilizzando schede audio esterne di qualità (firewire o usb) è possibile ridurre la latenza per il real time abbassando questo valore, lasciare inalterate queste impostazioni utilizzando la scheda audio integrata nella motherboard.

- Mac : menu-WinXound-Preferences-Compiler, usare come flags :

-b4000 –B4000 –g

Nei capitoli successivi vedremo come personalizzare queste linee di comando, una possibile linea in grado di leggere un file audio in real time potrebbe essere :

-odac1 -b4000 –B4000 –g

Il numero “1” del flag *odac* (oppure *devaudio*) attiva una porta audio della propria scheda, è possibile indicare un numero compreso tra 0 e 1023, indicando un numero errato Csound genera errore e fornisce la lista dei dispositivi audio trovati sul proprio computer, in questo modo è possibile indicare il numero corretto. Su computer, ad esempio portatili, in cui si utilizza unicamente la scheda audio integrata (e in cui non è stato installato nessun driver di scheda audio esterna), il comando da utilizzare è *–odac1* (è anche possibile scrivere *–o* staccato da *dac*, ”*–o dac*”).

Negli esempi relativi al libro verrà utilizzato il flag *–odac1* indicato nel file .csd in questo modo :

```
<CsOptions>
  -odac1
</CsOptions>
```

(provare a cambiare il numero di porta audio in caso di errore o in caso di assenza di segnale audio in uscita).

Avvertenze

Le versioni di Csound successive alla release 5.14, possono presentare qualche problema con esempi che utilizzano la libreria grafica Fltk, in caso di messaggi di errore (sia in ambiente Mac che Windows) può essere necessario aggiungere il seguente flag tra le opzioni :

```
<CsOptions>
```

```
--old-parser  
</CsOptions>
```

Hello world

L' Hello world è il più semplice programma con cui si inizia lo studio di un qualunque linguaggio di programmazione,un "Hello world" in Java per esempio,una volta compilato restituirà al prompt dei comandi il messaggio..... "Hello world".

Il nostro Hello world con Csound naturalmente sarà un suono (sebbene sia possibile anche con csound stampare un testo su una finestra),con cui inizieremo a capire quelle che sono le caratteristiche fondamentali di Csound.

Su Windows,apriamo l'editor WinXound,come nuovo file scegliamo il formato Csd ,questo formato contiene in un unico file i due grandi blocchi di programma Csound che sono :

Orchestra

Score

Nelle primissime versioni di questo linguaggio bisognava scrivere il codice proprio su due file separati con estensione .orc (orchestra) e .sco (score),il formato csd per la sua praticità è diventato lo standard attuale.

andiamo adesso ad analizzare lo scheletro di un file csd :

```
<CsoundSynthesizer>  
<CsOptions>  
;questo spazio è dedicato ai flag di Csound,i flag sono comandi con cui è  
;possibile dire al compilatore di svolgere numerose operazioni,ad esempio  
;scrivere file su disco,compilare in real time,comunicare con il proprio  
;hardware audio,midi e molto altro;  
</CsOptions>  
<CsInstruments>  
;questo spazio viene chiamato Orchestra,è il blocco di programma in cui definire  
gli strumenti,ad esempio :
```

```

instr1      ;abbiamo creato uno strumento chiamato "1"
;in questo spazio verranno definite le caratteristiche di questo strumento
endin ;chiudiamo le istruzioni per lo strumento "1"

Instr 2

endin

;

instr 999 ;possiamo utilizzare qualunque numero o scrivere numerosi strumenti;
endin

</CsInstruments>

<CsScore>

;in questo spazio scriveremo la score di Csound,è una sorta di partitura con cui
;definiremo eventi temporali in relazione con gli strumenti creati in
;orchestra,sarà possibile ad esempio dire per quanti secondi dovrà suonare un
;certo strumento,a quale secondo dovrà suonare e molto altro.

</CsScore>

</CsoundSynthesizer>
```

Avrete sicuramente notato l'utilizzo del ";" nei commenti al file,in questo modo è possibile commentare il codice di un file csound,il compilatore salterà tutto quello che verrà scritto dopo il punto e virgola.

Dal punto di vista dell'apprendimento di un qualunque linguaggio di programmazione è pratica assai diffusa e consigliata quella di commentare le righe di codice,questo serve sia al principiante che muove i primi passi che al programmatore esperto che deve mettere mano su programmi scritti in passato e di cui magari ricorda ben poco il significato di alcuni passaggi.

Adesso torniamo al nostro scheletro di programma e inseriamo le seguenti righe di codice :

;Esempio Cap1.1_hello world.csd

```
<CsoundSynthesizer>
```

```
<CsOptions>
```

```
</CsOptions>
```

```
<CsInstruments>
```

```
sr = 44100
```

```
kr = 4410
```

```
ksmps = 10
```

```
nchnls = 1
```

```

instr 1

asuono      rand  10000

out    asuono

endin

</CsInstruments>

<CsScore>

i1    0    10

</CsScore>

</CsoundSynthesizer>

```

Adesso salviamo il file appena scritto con WinXound, (File-Save as),alla voce **Tools** troviamo il comando **Compile**,se non abbiamo commesso errori di sintassi Csound avvierà il processo di compilazione in real time,se tutto è andato a buon fine dovrà riuscire a sentire 10 secondi di rumore bianco.

Andiamo a vedere nel dettaglio le righe di codice scritto,per prima cosa vediamo l'Header di Csound,è una parte della programmazione molto importante in cui definiamo ad esempio la frequenza di campionamento e il numero di canali,tuttavia il seguente “header” non è obbligatorio in questo semplice file di esempio perché rappresenta dei valori dati di default,ma se volissimo ad esempio un control rate più basso,un sistema audio ottocentrico o una frequenza di campionamento più alta diventerebbe indispensabile.

```

;Header

sr = 44100 ;sample rate a 44100,è la frequenza di campionamento
kr = 4410   ;frequenza di controllo
ksmps = 10 ;rappresenta il valore di sr/kr
nchnls = 1  ;numero di canali,in questo caso mono
instr 1 ;definiamo uno strumento "1"

```

asuono rand 10000

in questo frammento di codice stiamo creando un generatore di rumore bianco con ampiezza 10000,che cos’è “asuono”? è un comando di Csound?assolutamente no,”asuono” è il nome di una variabile di tipo audio che abbiamo chiamato “suono”,da cosa capiamo che si tratta di un generatore di segnali audio?proprio dalla lettera iniziale “a”. In Csound esistono vari tipi di variabili :

“a” : iniziano con questa lettera tutte le variabili di tipo audio,dopo la lettera “a” è possibile aggiungere qualunque parola o numero,ad esempio a1,a2,a77,a99,asuono,arumore,ecc.

“i” : sono le variabili di inizializzazione,ne parleremo a breve in riferimento alle forme d’onda;

“k” : variabili di controllo,sono variabili che non generano audio ma definiscono comportamenti,ad esempio di un generatore sonoro,potrebbe essere una variabile che controlla l’inviluppo d’ampiezza di un dato suono,oppure la variabile che controlla la frequenza di taglio di un filtro.

“g” : variabili globali,vanno dichiarate in orchestra prima della costruzione degli strumenti,a queste variabili possono accedere tutti gli strumenti.

Adesso torniamo al nostro frammento di codice :

asuono rand 10000

abbiamo quindi definito la variabile “asuono”,ora passiamo a **rand**,si tratta di un opcode che genera forme d’onda casuali ad una certa ampiezza data (in questo caso 10000),il rumore bianco viene spesso impiegato nella sintesi sottrattiva per via della sua complessa densità sonora;

out asuono

endin

“out” è un opcode che collega il suono generato da csound al nostro sistema audio di ascolto,esistono numerosi opcode per gestire l’audio in uscita,out è il più semplice tra tutti (un unico canale mono) ,csound offre anche la gestione di audio multicanale,quadrifonia,ottofonia,sistemi 5.1.

L’istruzione finale “endin” chiude il blocco di programma chiamato “instr 1”,a questo punto dobbiamo dire a csound in che modo gestire questo suono appena creato :

```
<CsScore>
;p1    p2    p3
i1    0    10
</CsScore>
</CsoundSynthesizer>
```

Cosa rappresentano p1,p2,p3??sono i p fields di Csound,indicano i parametri di un determinato suono,uno strumento Csound può avere numerosi pfields (immaginate lo score di un suono in cui dobbiamo definire attacco,durata,ampiezza,frequenza,pan e molto altro),i parametri fondamentali di un suono sono i primi tre p fields :

p1 : nome dello strumento ,ad esempio “i1” farà riferimento allo strumento “instr 1” dell’orchestra

p2 : attacco ,indica a quale secondo (ma potrebbe anche essere un millesimo di secondo) questo strumento inizierà a suonare,indicando p2 = 0 csound farà partire il suono nell’istante zero;

p3 : durata,indica la durata espressa in secondi,l’istruzione del nostro esempio :

```
i1    0    10
```

sta dicendo al compilatore di far suonare “instr 1” (il nostro rumore bianco) per 10 secondi con attacco immediato.

Abbiamo visto un primo semplicissimo esempio di sintesi sonora in Csound,provate a modificare il file appena creato nel seguente modo :

- modificate il valore dell’ampiezza di rand (siate cauti...non mettete valori impossibili come decine di zeri...);
- modificate la durata di questo suono

- provate a creare diverse istanze di instr 1 con vari intervalli di tempo.

Elementi di programmazione

Nel breve esempio “Hello world” abbiamo incontrato alcuni elementi e concetti elementari della programmazione :

- costanti : valore fisso che rimane invariato durante l'esecuzione del programma
- variabili : servono a contenere dati variabili nel tempo durante l'esecuzione del programma
- algoritmo : descrizione della logica in base alla quale un programma elabora dei dati per svolgere un determinato compito

in che modo delle variabili con valori numerici possono essere elaborate? ad esempio con semplici operazioni matematiche, in Csound esiste una numerosa lista di opcodes per realizzare operazioni di calcolo matematico. Vediamo un semplice programma in cui due variabili vengono utilizzate per operazioni matematiche :

Esempio Cap1.2_matematica

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

instr 1

ivarA =      8      ;dichiaro variabili
ivarB =      4

isomma      =      ivarA+ivarB
idifferenza =      ivarA-ivarB
irapporto   =      ivarA/ivarB
imoltiplicazione =      ivarA*ivarB
isenoA      =      sin(ivarA)
icosenoA   =      cos(ivarA)
irnd =      rnd(ivarA)
iradice     =      sqrt(ivarB)
iespress    =      (((ivarA+ivarB)/2)*cos(ivarA*ivarB))/ivarB+sqrt(ivarA)

;visualizza in console, print accetta anche argomenti multipli (print
var1,var2,var3,ecc...)

print isomma
print idifferenza
print irapporto
print irnd
print imoltiplicazione
print isenoA
print icosenoA
print iradice
print iespress

endin

</CsInstruments>
<CsScore>
```

```

i1      0      20

</CsScore>
</CsoundSynthesizer>
```

il programma visualizza nella console di Csound il seguente output :

```

SECTION 1:
new alloc for instr 1:
instr 1:  isomma = 12.000
instr 1:  idifferenza = 4.000
instr 1:  irapporto = 2.000
instr 1:  irnd = 7.788
instr 1:  imoltiplicazione = 32.000
instr 1:  isenoA = 0.989
instr 1:  icosenoA = -0.146
instr 1:  iradice = 2.000
instr 1:  iexpress = 4.080
----- Compiler End -----
```

Un altro aspetto molto tipico di un linguaggio di programmazione è quello delle condizionali,in pratica viene definita una condizione,se la condizione è “vera” si verifica un determinato output (o si verifica un processo,ecc),se la condizione è “falsa” il programma non tiene conto dell’istruzione e passa alla riga successiva.

Il blocco “if-then” :

```

if      "dichiaro condizione"  then
.....processo da realizzare
endif
```

Esempio Cap1.3_if-then

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr      =      44100
kr      =      4410
ksmps =      10
nchnls     =      1

instr 1

ivarA =      5      ;dichiaro variabili
ivarB =      4

if      ivarA >      ivarB then ;se A è maggiore di B suona 440Hz
a1      oscil 10000,440,1
elseif      ivarA <      ivarB then ;se A è minore di B suona 220Hz
a1      oscil 10000,220,1
endif

out    a1

endin

</CsInstruments>
```

```
<CsScore>  
f1      0      4096   10      1  
i1      0      10  
</CsScore>  
</CsoundSynthesizer>
```

Cap.2 Primi passi,orchestra,score

2.1 Ampiezza,frequenza

2.2 Render to file,real time

2.3 p fields

2.4 score

Aampiezza,frequenza

Per generare il rumore bianco dell'esempio precedente,la nostra orchestra utilizzava un opcode (**rand**) controllato da un unico parametro **Aampiezza** :

```
asegnale    rand  10000 ;(10000 = valore di ampiezza)
```

come facilmente potete immaginare,esistono in Csound numerosi altri opcode il cui numero di parametri varia dalla complessità dell'oggetto stesso,prenderemo ora in considerazione uno degli opcode più importanti di Csound,l'opcode “**oscil**” ,andiamo a vedere la sua sintassi nel dettaglio :

```
asig  oscil iamp,ifreq,ifn
```

come si legge la sua sintassi?ci viene in aiuto il manuale ufficiale di Csound,in breve possiamo descrivere l'opcode **oscil** come un modulo sonoro per generare una forma d'onda (ma vedremo in seguito che i suoi utilizzi sono molteplici,ad esempio per la costruzione di segnali di controllo),i suoi parametri sono nell'ordine :

ampiezza ,frequenza,numero di funzione

un possibile esempio dentro un'orchestra potrebbe essere :

```
instr 1      ;nome dello strumento  
  
asuono      oscil 10000,220,1 ;variabileasuono  
  
out   asuono      ;uscita mono della variabile asuono  
  
endin ;chiusura del blocco di programma
```

abbiamo appena creato uno strumento in grado di generare una forma d'onda con ampiezza 10000 e frequenza di 220 hz,di quale forma d'onda stiamo parlando?l'opcode **oscil** è legato ad un particolare tipo di forma d'onda?dalle informazioni che troviamo nel nostro “instr 1” non siamo ancora in grado di capire se il processo di sintesi determinerà un'onda quadra,piuttosto che un'onda a dente di sega,triangolare o altro.

A questo punto introduciamo uno degli elementi fondamentali di questo linguaggio per la sintesi,ossia il concetto di funzione (che in Csound serve per creare una forma d'onda) ,osserviamo il terzo parametro del nostro opcode :

```
asuono      oscil 10000,220,1
```

l'ultimo parametro a cui abbiamo assegnato il valore “1” è legato ad una funzione,in questo caso un metodo di generazione di forma d'onda che troveremo nello score.

0dbfs

Uno dei metodi più semplici e soddisfacenti per gestire il livello di ampiezza di un file Csound,è l'utilizzo dell'opcode **0dbfs**,corrisponde al valore massimo di ampiezza nel dominio digitale prima del clip.

0dbfs = 32767 (corrispondente alla gamma bipolare di un file audio 16 bit o 16 bit AD/DA codec)

L'utilizzo di questo opcode avviene di norma nell'header in questa modalità :

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1
```

in questo modo possiamo assegnare valori di ampiezza in un range compreso tra 0 e 1. I prossimi esempi contenuti nel libro utilizzeranno prevalentemente questa modalità.

Esempio Cap2.1

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1
asuono      oscil .5,220,1
out   asuono
endin

</CsInstruments>
<CsScore>
f1    0        4096  10      1
i1    0        4
</CsScore>
</CsoundSynthesizer>
```

Proviamo a sintetizzare il suono per ascoltarlo, come nell'esempio precedente salviamo il nostro file, sempre da WinXound al menu **Tools** clicchiamo su **Compile** (oppure clicchiamo sul pulsante Verde simile ad un tasto play), se tutto è andato a buon fine potremo ascoltare 4 secondi di una sinusoide con frequenza di 220 hz.

Abbiamo così introdotto un nuovo elemento importante di Csound chiamato funzione (f1 0 4096 10 1), andiamo a vedere nel dettaglio la struttura della nostra funzione :

f1 : è il numero della funzione , l'opcode oscil richiamava questa funzione proprio nel suo terzo parametro;

0 : rappresenta l'istante in cui viene generata la forma d'onda,in questo caso ad istante zero;

4096 : indica il numero di punti con cui sarà rappresentata la nostra forma d'onda ;

10 : indica il metodo di generazione di Csound, chiamato Gen, in questo caso Gen10 (questa Gen serve a creare sinusoidi, ma vedremo che esistono diversi altri tipi di Gen)

1 : questo valore indica che stiamo creando una sola sinusoide .

Per iniziare ad addentrarci sempre di più nella logica di programmazione, proviamo ad utilizzare le poche conoscenze finora acquisite per esplorare alcuni tipi di utilizzi dell'opcode **oscil**, nell'esempio seguente costruiremo uno strumento costituito da tre oscillatori (quindi dichiareremo tre variabili audio), ognuno con una funzione (Gen10) diversa nello score.

(ricordate che dopo il ";" sono dei commenti al codice non considerati da Csound)

Esempio Cap.2.2

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

a1    oscil 1,220,1      ;variabile a1 con frequenza 220 e funzione 1
a2    oscil 1,440,2      ;variabile a2 con frequenza 440 e funzione 2
a3    oscil 1,880,3      ;variabile a3 con frequenza 880 e funzione 3
asomma    sum   a1,a2,a3  ;variabile asomma in cui sommiamo a1,a2,a3

out    asomma*.3      ;uscita audio mono della variabile asomma riscalata(*.3)

endin

</CsInstruments>
<CsScore>

f1    0      4096  10     1      ;una sola fondamentale
f2    0      4096  10     1      1      ;fondamentale + seconda armonica
f3    0      4096  10     1      0      1      ;fondamentale + terza armonica

i1    0      4      ;instr 1 con attacco immediato e durata di 4 secondi
i1    5      4      ;attacco al quinto secondo e durata di 4 secondi

</CsScore>
</CsoundSynthesizer>
```

Anche in questo esempio abbiamo introdotto alcuni nuovi elementi di questo linguaggio, iniziamo dallo strumento in orchestra in cui abbiamo costruito tre variabili audio, in seguito sommate in una nuova variabile chiamata "**asomma**", l'opcode **sum** equivale al frammento di codice :

```
asomma = a1 + a2 + a3
```

esiste un limite al numero di oscillatori possibili di uno strumento? assolutamente no, l'unico limite è dato dalla nostra fantasia o dalla potenza di calcolo della nostra macchina.

Analizziamo adesso la funzione numero 2 della nostra score :

```
f 2 0 4096 10 1 1
```

questa funzione definisce due sinusoidi ,fondamentale + la sua seconda armonica con ampiezza uguale alla fondamentale,è possibile costruire in questo modo anche suoni molto complessi :

```
f 1 0 4096 10 1 1 1 1 1 1 1 10
```

cosa significa il numero “10” alla fine della funzione?indica che l’ultima armonica avrà un’ampiezza dieci volte quella della fondamentale o delle altre armoniche; come vedete le possibilità sono veramente infinite,vediamo adesso altri esempi di funzioni con fondamentale e armoniche :

```
f 1 0 4096 10 1 0.111 0.04 0.02 0.012
```

```
f 1 0 4096 10 1 0 .1 .2 .3 .4 .5 .6 .7
```

il valore 4096,che come abbiamo precedentemente spiegato,indica il numero di punti per definire la tabella della nostra forma d’onda,deve sempre essere una potenza di due,un esempio molto tipico di funzione potrebbe essere :

```
f 1 0 16384 10 1
```

con questa funzione csound scriverà una tabella molto più precisa,esattamente con 16384 punti,provate a sperimentare utilizzando dei valori di tabella molto bassi (ad esempio 16),vi accorgerete come la qualità audio si abbasserà notevolmente (quasi una sorta di downsampling,anche se in realtà la frequenza di campionamento sarà sempre quella dichiarata nell’header.)

Csound offre diverse varianti aggiornate dell’opcode **oscil** (il quale rappresenta,fin dalle primissime versioni di Csound,uno degli strumenti più usati dai musicisti e sound designer,vedremo come sia possibile utilizzare questo opcode nella sintesi additiva,nella costruzione di processori di segnale come l’effetto chorus,come segnale di controllo per spazializzare un suono,per la costruzione di vibrati ed lfo),nell’ordine elenchiamo alcune delle varianti più attuali di questo opcode :

- **poscil, poscil3** : si tratta di oscillatori ad alta precisione,sono molto usati per la costruzione di segnali di controllo e sono preferibili per lavorare con valori di frequenza molto bassi,accettano anche valori negativi,si consiglia di consultare il manuale ufficiale per avere una panoramica più dettagliata di questi opcode;
- **oscili** : questa variante utilizza l’interpolazione lineare ed è molto più adatto per l’utilizzo in real time,ad esempio in situazioni in cui il suo valore è controllato in tempo reale da un oggetto grafico o da un controller midi;
- **oscil3** : come oscili ma utilizza l’interpolazione cubica;

Esercizio : provate a modificare lo strumento precedente sostituendo oscil con uno di questi nuovi opcode,dal punto di vista sonoro non noterete per il momento differenze ma vedremo molto più avanti come la scelta di questi opcode sia preferibile ad oscil.

Render to file

Una delle funzioni fondamentali per un linguaggio di sintesi è quella di poter scrivere e salvare su un file audio il proprio lavoro,Csound permette vari metodi di scrittura per file attraverso l'utilizzo di flag (le linee di comando di Csound) specifici.

Proviamo a modificare uno dei nostri file scritti precedentemente,il flag per il render to file deve trovarsi nello spazio delimitato dai tag :

```
<CsOptions>
-o Nomedelfile.wav
</CsOptions>
```

Il flag “-o” prende come argomento il nome del file scelto,è anche possibile indicare una eventuale directory completa,ad esempio :

```
-o c:/music/samples/Render.wav
```

Csound genera errore se la directory indicata è inesistente,se non specifichiamo nessun percorso il file audio generato verrà scritto nella stessa cartella dove si trova il file .csd.In questi due esempi abbiamo scelto come formato audio il file wav,naturalmente è anche possibile scrivere file in formato aiff semplicemente indicandolo nel nome del file :

```
-o c:/music/samples/Render.aiff
```

Csound offre ancora numerosi flag per il tipo di file audio,per esempio potrebbe essere utile scrivere un file audio a 24 bit o 32 bit :

```
-o Nomedelfile.wav -f ;scrive un file wav a 32 bit(floating point)
```

```
-o Nomedelfile.wav -3;scrive un file wav a 24 bit
```

```
-o Nomedelfile.wav -s ;scrive un file wav a 16 bit (opzionale,default)
```

```
-o Nomedelfile.wav -8 ;scrive un file wav a 8 bit
```

La lista dei formati audio disponibili in Csound è molto ampia,oltre ai formati più usati come wav e aiff,è possibile specificare anche i formati au, avr, caf, flac, htk, ircam, mat4, mat5, nis, paf, pvf, raw, sd2, sds, svx, voc, w64, wav, wavex.

P fields

In questa sezione prenderemo in considerazione la possibilità di controllare i parametri di un suono completamente tramite la score, come possiamo osservare negli esempi precedenti il nostro oscillatore definiva ampiezza e frequenza costante in “instr 1”, in che modo è possibile controllare ampiezza e frequenza dalla score? considerando questi due importanti parametri con ulteriori p fields, esattamente come **p4** e **p5**.

Esempio Cap.2.3

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1
    acorda      pluck p4,p5,p6,0,1      ;modello di corda pizzicata

    out    acorda      ;uscita audio mono della variabile acorda
endin

</CsInstruments>
<CsScore>

;p1   p2    p3    p4    p5    p6
i1    0     2     1     440   220
i1    3     2     .6    880   420
i1    5     2     .5    440   620
i1    7     2     .4    880   820

</CsScore>
</CsoundSynthesizer>
```

Abbiamo introdotto un nuovo metodo di sintesi, l’opcode **pluck** (modello fisico di una corda pizzicata) di cui ne parleremo in modo più approfondito nel capitolo sulla sintesi per modelli fisici, la nostra score in questo esempio è costituita da 6 parametri di cui i primi tre (nome strumento, attacco, durata della nota) gestiscono l’evento sonoro dal punto di vista temporale, i parametri p4, p5, p6 sono invece riferiti alle caratteristiche specifiche di questo opcode :

sintassi di pluck :

```
ares pluck kamp,kcps,icps,ifn,imeth
```

acorda **pluck** ampiezza,frequenza,timbro ,0,1 (“0 e 1” sono valori di default, consultare la guida ufficiale)

nella nostra score abbiamo associato il parametro ampiezza alla variabile p4,il parametro frequenza alla variabile p5 e infine il timbro alla variabile p6,provate a sperimentare modificando i vari parametri della score.Vediamo un esempio che fa uso di numerosi parametri di score per definire il suono :

Esempio Cap.2.4

```
<CsSoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

a1      oscil p4,p5,1
a2      oscil p4,p6,2
a3      oscil p4,p7,3
a4      oscil p4,p8,4

asomma      sum    a1,a2,a3,a4

out      asomma

endin

</CsInstruments>
<CsScore>

f1      0      4096  10      1
f2      0      4096  10      0      1
f3      0      4096  10      0      0      1
f4      0      4096  10      0      0      0      1

;p1    p2      p3      p4      p5      p6      p7      p8
i1      0      4      .1      440     540     640     740

</CsScore>
</CsSoundSynthesizer>
```

L'esempio 2.4 definisce uno strumento formato da 4 oscillatori sinusoidali sommati in una variabile "asomma",ogni oscillatore ha un valore di frequenza definito dalla score (p5,p6,p7,p8).

Metodi di gestione eventi

Abbiamo fin qui osservato il modo in cui gli eventi sonori nel tempo (la nostra partitura) vengono controllati dalla score attraverso i vari p fields,Csound offre numerosi metodi di controllo della score,per esempio potrebbe rendersi necessario definire una velocità metronomica magari variabile tra più valori,oppure con una score molto lunga avere la possibilità di saltare da un punto a un altro nel tempo,vedremo nei capitoli successivi come sia possibile controllare una score di Csound direttamente dall'orchestra.

Il seguente esempio illustra il controllo metronomico :

```

</CsInstruments>
<CsScore>
f1    0      4096  10     1
f2    0      4096  10     0      1
f3    0      4096  10     0      0      1
f4    0      4096  10     0      0      0      1

t    0      120
;p1   p2    p3     p4     p5     p6     p7     p8
i1    0      4      1      440    540    640    740

</CsScore>
</CsoundSynthesizer>

```

l'istruzione "t 0 120" controlla la velocità di questa score, è anche possibile indicare delle variazioni temporali all'interno della stessa score per interpretazioni come rallentando, accelerando ecc, analizziamo il metodo "t" :

```

p1      deve essere zero
p2      tempo iniziale in battiti per minuto
p3,p5,p7, ... valori in beats

t0     260     8       30      18      320

```

con questo esempio il tempo di 260 inizia dall'istante zero e rallenta nel tempo di 8 secondi al tempo 30, in seguito accelera fino al secondo 18 raggiungendo la velocità di 320 bpm.

Lavorando con score molto complesse che si occupano di controllare numerosi strumenti dell'orchestra, diventa molto utile durante l'attività compositiva e di sound design, poter mettere in "mute" un determinato strumento, bypassando la sua lettura, per far questo basta semplicemente trasformare il numero dello strumento in un valore negativo :

```

;valore negativo per "instr 1", mute

i     -1      0       4       .1      440

```

e se volessimo saltare in un preciso istante temporale senza dover dover ascoltare l'intera score? esiste anche per questo un istruzione precisa :

```

a     0      0       22

```

csound salta la lettura fino al p2=22, iniziando la lettura da questo punto (l'istruzione deve essere scritta prima di ogni evento nella score).

La lista di operazioni per gestire la score di Csound sono numerose e si consiglia la documentazione ufficiale, concludiamo con un ulteriore esempio di gestione eventi molto utile :

```

<CsScore>

i1    0      1      10000      220
i1    2      .      .          320
i1    4      .      .          420
i1    6      .      .          520
i1    8      .      .          620
i1    4      .      .          720
i1    6      .      .          820
i2    8      .      28000      100
i2    10     .      .          200
i2    12     .      .          300
i2    14     .      .          400

```

```
</CsScore>
</CsoundSynthesizer>
```

In questo esempio osserviamo come sia relativamente semplice poter evitare noiose ripetizioni di scrittura,i parametri p3 e p4 utilizzano come valore quello espresso nella prima istruzione della score “i1 0 1 10000 440”,il seguente esempio darebbe lo stesso identico risultato :

```
<CsScore>

i1    0      1      10000      220
i1    2      1      10000      320
i1    4      1      10000      420
i1    6      1      10000      520
i1    8      1      10000      620
i1    4      1      10000      720
i1    6      1      10000      820
i2    8      1      10000      100
i2   10      1     28000      200
i2   12      1     28000      300
i2   14      1     28000      400

</CsScore>
</CsoundSynthesizer>
```

Con questo esempio si conclude la prima parte riservata a chi sta muovendo i primi passi con questo meraviglioso linguaggio,abbiamo presentato i concetti fondamentali per iniziare ad addentrarci nelle tecniche specifiche,sintesi del suono e i vari tipi di applicazioni pratiche che seguiranno.

Cap.3 Tecniche fondamentali

- 3.1 Signal generators : breve panoramica
- 3.2 Inviluppi,glissandi
- 3.3 Tecniche di vibrato,tremolo
- 3.4 Tecniche di spazializzazione
- 3.5 Utilizzo delle macro
- 3.6 Funzioni random
- 3.7 Processi generativi
- 3.8 Gen : panoramica

Signal generators : breve panoramica

Conclusa la prima parte introduttiva al linguaggio iniziamo con questo capitolo un percorso che ci porterà ad esplorare le varie tecniche di sintesi ed elaborazione di segnali audio,i *Signal generators* di Csound sono classificati nel *canonical reference manual* in varie categorie che comprendono la sintesi additiva,risintesi,oscillatori base,sintesi Fm,sintesi granulare,sintesi Scanned,opcode per generare inviluppi e strutture di controllo,generatori di numeri casuali,lettura di file audio,sintesi per modelli fisici e molto altro.Il primo esempio di questo capitolo contiene una rapida carrellata sulle principali tecniche di sintesi attualmente conosciute e implementate in Csound,ogni strumento illustrerà una particolare tecnica che verrà approfondita nei capitoli successivi. Il prossimo esempio presenta una rapida successione dimostrativa di alcune tecniche di sintesi implementate in Csound,in ordine ascolteremo :

- oscillatore Vco per onda a dente di sega
- modello fisico di bamboo
- serie di armoniche sinusoidali
- sintesi fm
- sintesi per formanti
- sample player
- modello fisico di corda pizzicata
- rumore bianco filtrato

Esempio Cap.3.1

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1
a1    vco2  .5,220,0,.8 ;sawtooth
outs  a1,a1
endin

instr 2
a1    bamboo      .5,0   ;modello di bamboo
outs  a1,a1
endin

instr 3
a1    buzz   .5,220,8,1 ;serie di armoniche sinusoidali
```

```

outs a1,a1
endin

instr 4
a1    foscil      .5, 220,400,200,2,1      ;sintesi fm
outs a1,a1
endin

instr 5
k1    randomi 260,550,3
a1    fof   .5, 220, k1, 0, 50, 0.003,0.02, 0.007,100,1,2,p3      ;formanti
outs a1,a1
endin

instr 6
a1    soundin    "sample1.wav"      ;sample player
outs a1,a1
endin

instr 7
axcite    oscil 1, 1, 1
a1    wgpluck    440,.5,0.3,0,10,2000,axcite  ;modello di corda pizzicata
outs a1,a1
endin

instr 8
a1    rand  1      ;rumore bianco
a2    tonex a1,3000,8    ;filtro
outs a2,a2
endin

</CsInstruments>
<CsScore>
f1    0      16384 10      1
f2    0      16384 19      0.5    0.5    270    0.5

i1    0      3.5
s
i2    0      3.5
s
i3    0      3.5
s
i4    0      3.5
s
i5    0      3.5
s
i6    0      3.5
s
i7    0      3.5
s
i8    0      3.5

</CsScore>
</CsoundSynthesizer>
```

Il nostro file di esempio contiene 8 strumenti con caratteristiche timbriche molto differenti, la sintassi e il funzionamento dei numerosi nuovi opcode incontrati saranno esplorati nei capitoli successivi, per adesso ci soffermeremo solo sull'uscita audio stereo non incontrata negli esempi precedenti. Csound offre numerosi opcode per scrivere e ascoltare file multicanale (**out,out32,outc,outch,outh,outo,outrg,outq,outq1**, ecc), l'istruzione **nchnls = 2** crea un suono stereo in cui ogni strumento dovrà dichiarare due canali di uscita (left e right):

```

instr 1
asuono .....
outs asuono,asuono
endin

```

in questo esempio l'opcode **outs** prende due argomenti :

```
outs left,right
```

in strumenti più complessi, costituiti da almeno due timbri con caratteristiche diverse, è pratica molto comune indirizzare ai due canali di uscita due timbri diversi, ad esempio :

```
outs achitarra,aflauto
```

Esercizio : provare a costruire uno strumento costituito dalla fusione spettrale di due tecniche di sintesi diverse, indirizzare i due timbri ad un canale diverso e creare una score.

Inviluppi,glissandi

Come abbiamo avuto modo di ascoltare negli esempi precedenti, il suono di ogni strumento era di tipo "statico" con un fastidioso click alla fine di ogni evento definito nella score, prenderemo ora in esame alcune delle tecniche più tipiche per costruire inviluppi di ampiezza, glissandi, variabili di controllo.

Anche su questo argomento vedremo come il numero di opcode dedicati sia molto ampio, il prossimo esempio da una semplice dimostrazione dell'opcode **linseg**, con cui è possibile definire dei segmenti tra valori indicati, immaginate di avere un suono della durata di 10 secondi, in cui i primi 3 secondi la sua ampiezza inizi da un valore iniziale=0 e aumenti fino a raggiungere il valore=12000 per poi ritornare al valore iniziale in un dato spazio temporale.

la sintassi di linseg :

```
variabile linseg valA,durata1,valB,durata2,valC,durata3,valD,ecc.....
```

un esempio tipico di inviluppo :

```
k1 linseg 0,p3/2,10000,p3/2,0
```

ricordiamo che p3 rappresenta la durata? p3/2 rappresenta quindi metà del valore di durata, con questo esempio la variabile k1 (come abbiamo già accennato le variabili di controllo devono iniziare con la lettera "k") assume il valore=0 nell'istante iniziale, sale al valore=10000 per metà del tempo p3 indicato nella score, infine nella seconda metà di valore temporale(p3/2) "spegne" gradualmente il suono raggiungendo nuovamente il valore=0, questo è l'esempio più semplice per creare inviluppi di ampiezza senza fastidiosi click.

Esempio Cap.3.2

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

```

```

kamp linseg 0,p3/2,.5,p3/2,0 ;inviluppo di ampiezza
a1 oscil kamp,440,1 ;l'ampiezza prende come argomento l'inviluppo kamp
out a1
endin

</CsInstruments>
<CsScore>

f1 0 16384 10 1 1
i1 0 8

</CsScore>
</CsoundSynthesizer>
```

analizziamo il seguente frammento di codice :

```

kamp linseg 0,p3/2,.5,p3/2,0
a1 oscil kamp,440,1
```

abbiamo creato una variabile **kamp** che crea un inviluppo di ampiezza formato da due segmenti, la cui durata p3 sarà legata al valore che daremo nella score, la variabile **kamp** viene poi utilizzata come struttura di controllo per la variabile audio **a1**, nei precedenti esempi abbiamo sempre dato un valore fisso al primo parametro di **oscil** (appunto l'ampiezza), questa volta il suo valore sarà determinato dalla variabile **kamp** e quindi il risultato sonoro sarà dinamico nel tempo.

Possiamo utilizzare **linseg** anche per definire dei glissandi, in questo caso la nostra variabile di controllo verrà associata alla frequenza dell'oscillatore :

Esempio Cap.3.3_glissando

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kamp linseg 0,p3/2,.5,p3/2,0
kfreq linseg 220,p3/2,440,p3/2,220
a1 oscili kamp,kfreq,1

out a1
endin

</CsInstruments>
<CsScore>

f1 0 16384 10 1 1
i1 0 8
```

```
</CsScore>
</CsoundSynthesizer>
```

In questo esempio abbiamo costruito un oscillatore controllato da due variabili,kamp definisce un inviluppo di ampiezza mentre kfreq genera un glissando tra le frequenze 220hz e 440hz,l'opcode **oscili** prende come argomenti le due variabili.

Una variante possibile di linseg è rappresentata dall'opcode **expseg** che crea un inviluppo esponenziale,expseg traccia una serie di segmenti esponenziali tra punti dati,la sua sintassi :

```
ares expseg iamp1,idur1,iamp2,idur2,iamp3,idur3.....
```

esempio :

```
kamp expseg 0.01,p3*0.25,1,p3*0.75,0.01
```

expseg non accetta il valore zero ma è possibile utilizzare ugualmente valori molto bassi come 0.001.

nel manuale ufficiale alla sezione *Linear and Exponential Generators* possiamo ancora trovare numerosi opcode per creare inviluppi di vario genere,un altro molto interessante è **loopseg** in cui è possibile creare dei veri e propri loop all'interno dell'inviluppo con *rate* controllabile,sintassi di loopseg :

```
ksig loopseg kfreq,ktrig,ktimed0,kvalue0[,ktimed1][,kvalue1],ecc
```

kfreq : rate dell'inviluppo,controlla la sua durata e accetta anche valori negativi (lettura all'indietro dell'inviluppo);

ktrig : il valore zero attiva il loop;

ktimed0 : punto iniziale(“0” di default);

ktimed1,ktimed2,ecc : le durate non sono rappresentate in secondi ma sono rapportate a kfreq;

kvalue : valore di ogni punto;

Esempio Cap.3.4_loopseg

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kfreq =      p4      ;velocità loop
ktrig =      0       ;attiva loopseg
itimel0 =    0       ;definisce segmenti
kvalue0 =    0
itimel1 =    1
kvalue1 =   .5 ;ampiezza
itimel2 =    1
kvalue2 =    0
```

```

ksig  loopseg      kfreq,ktrig,itime0,kvalue0,itime1,kvalue1,itime2,kvalue2
a1    oscil ksig,220,1
out   a1
endin

</CsInstruments>
<CsScore>

f1    0      16384 10     1      0      1      0      1      0      1

i1    0      4      2
i1    5      5      .2
i1    11     10     8

</CsScore>
</CsoundSynthesizer>
```

Abbiamo creato un inviluppo con valore di attacco = 0, ampiezza massima di .5 e decadimento a valore=0, la durata dell'intero inviluppo è controllata dal parametro kfreq(p4 nella score) che controlla l'intervallo di tempo tra una ripetizione e l'altra.

Affrontiamo ora un argomento molto importante relativo a questo linguaggio di sintesi, le conversioni da frequenza a pitch, leggiamo il funzionamento dei seguenti opcode :

cpspch : conversione di pitch (nota, ottava divisa in 12 parti) in frequenza (hertz)

Esempio Cap.3.5_conversioni

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      1
0dbfs   =      1

instr   1

kamp    =      p4
kfreq   =      cpspch(p5)
a1      pluck  kamp,kfreq,220,0,1

out    a1

endin

</CsInstruments>
<CsScore>

i1    0      1      .5      7.00
s
i1    0      1      .5      7.02
s
```

```

i1    0      1      .5      7.04
s
i1    0      1      .5      7.05
s
i1    0      3      .5      7.07
s
i1    0      .1     .5      7.07
s
i1    0      .1     .5      7.05
s
i1    0      .1     .5      7.04
s
i1    0      .1     .5      7.02
s
i1    0      4      .5      7.00
s

</CsScore>
</CsoundSynthesizer>
```

Il valore 7.00 corrisponde alla nota musicale Do, il primo numero intero indica l'ottava e tramite i decimali è possibile indicare tutte le note contenute nell'ottava (7.00, 7.01, 7.02, 7.03, 7.04....fino a 7.11, a cui segue l'ottava successiva 8.00).

Ora vediamo un esempio molto più complesso in cui oltre a **loopseg** utilizzeremo una variante **lpshold** molto adatta a controllare per esempio strutture melodiche, **lpshold** ha la stessa sintassi di **loopseg** ma ha la caratteristica di mantenere costante ogni valore fino a quando un nuovo punto di interruzione viene rilevato. L'esempio *bassline* utilizza **loopseg** per controllare l'inviluppo di ampiezza, **lpshold** controlla invece una sorta di tabella di valori di frequenza (indicate in pitch tramite la conversione **cpspch**), l'idea del prossimo esempio nasce dal concetto di step sequencer che troviamo in alcune storiche macchine analogiche o anche in strumenti più recenti come *Alesis Andromeda*.

Esempio Cap.3.6_Bassline

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kfreq =      p4      ;velocità loop
ktrig =      0
kamp =       .7
ksig  loopseg      kfreq,ktrig,0,0,1,kamp,1,0 ;inviluppo ampiezza
knote lpshold kfreq,ktrig,1,cpspch(6.01),1,cpspch(7.00),\;tabella pitch
1,cpspch(6.02),1,cpspch(9.00),1,cpspch(6.01),1,cpspch(7.02),\
1,cpspch(9.00),1,cpspch(6.01),1,cpspch(6.02),1
a1    vco2  ksig,knote;vco sawtooth
kmoog linseg 100,p3/2,9000,p3/2,100;frequenza controllo filtro moog
afilt moogladder a1,kmoog,.6;emulazione filtro moog
out    afilt;uscita vco filtrato
```

```

endin

</CsInstruments>
<CsScore>

i1    0    20    .8

</CsScore>
</CsoundSynthesizer>
```

In questo esempio abbiamo un oscillatore che genera un'onda a dente di sega filtrata da un filtro emulazione *Moog* (parleremo in modo approfondito di filtri nella sezione sulla sintesi sottrattiva), per ora limitiamoci ad osservare che l'opcode **out** prende come argomento l'uscita del suono filtrato **afilt**, analizziamo il blocco di codice relativo all'oscillatore :

```
a1    vco2  ksig,knote
```

ksig è la variabile di controllo costruita con *loopseg* ed utilizzata per controllare *vco2* in ampiezza, **knote** è invece la variabile creata dall'opcode *lpshold* che contiene le frequenze della sequenza melodica in stile *analog sequencer*; come avrete notato nella riga codice relativa a *lpshold*, abbiamo utilizzato il simbolo “\” molto utile per la funzione di “andare a capo”.

Esercizio : provare a sostituire l'opcode *lpshold* con *loopseg* per controllare la lista di note

Il prossimo esempio introduce una delle tecniche di inviluppo più note, il classico **Adsr** (Attack , Decay , Sustain , Release) che troviamo in numerosi dispositivi analogici o digitali, questo particolare inviluppo viene spesso utilizzato anche per controllare l'azione di un filtro nella sintesi sottrattiva, Csound offre l'opcode **adsr** (con alcune varianti come *madsr*, *mxadsr* , alcuni di questi opcode non possono essere usati lavorando con il midi) la sua sintassi :

```
kres    adsr    iatt,idec,islev,irel
```

questo opcode realizza degli inviluppi con un range di valori che variano tra 0 e 1, nell'esempio seguente creeremo una variabile di controllo *adsr* e la moltiplicheremo per un valore di ampiezza dell'oscillatore, ad esempio :

```
kadsr adsr  iatt,idec,islev,irel
asuono oscili ampiezza*kadsr,frequenza,funzione
```

Esempio Cap.3.7_adsr

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

iattac =      p6      ;attack
idec   =      p7      ;decay
isust   =      p8      ;sustain
irel   =      p9      ;release
```

```

kadsr    adsr    iattac,iDEC,isust,irel
kamp     =        p4      ;ampiezza
kpitch   =        cpspch(p5) ;pitch
a1       vco2    kamp*kadsr,kpitch,0,.9 ;vco

out      a1
endin

</CsInstruments>
<CsScore>

i1      0      2      .6      8.00    1      .4      .06      .0001
i1      0      2      .6      7.00    1      .4      .06      .0001
i1      3      2      .6      8.03    .1      .9      .02      .0001
i1      3      2      .6      7.03    .1      .9      .02      .0001
i1      6      2      .6      8.06    1      .01     .1      .0001
i1      6      2      .6      7.06    1      .01     .1      .0001

</CsScore>
</CsoundSynthesizer>

```

Tecniche di vibrato,tremolo

In questa sezione affronteremo alcune delle tecniche di modulazione più tipiche,Csound offre alcuni opcode dedicati a specifiche tecniche,ma è anche possibile costruire effetti di modulazione partendo da componenti primarie ad esempio con l'uso di **oscil**.Vediamo una breve descrizione per ogni tipo di tecnica :

vibrato : è caratterizzato da una variazione periodica dell'altezza di una nota,in Csound La frequenza è modulata da un segnale di controllo che utilizza gli opcode **oscil** oppure **lfo**.

lfo (low frequency oscillator) : si tratta di un oscillatore che produce frequenze subsoniche (sotto i 20 hertz),troviamo questo tipo di oscillatore nei sintetizzatori degli anni settanta per modulare oscillatori controllati in tensione (Moog,Arp) ,ma anche nei synth di generazioni successive(Roland,Waldorf,Access),con lfo è possibile creare effetti di tremolo e vibrato (a seconda che sia applicato per modulare la frequenza o l'ampiezza).

Tremolo : si tratta di una variazione periodica di ampiezza,l'oscillatore che modula produce anche qui frequenze al di sotto dei 20 hertz.

il seguente esempio mostra tre tecniche diverse per la creazione di un vibrato,nel primo strumento la frequenza dell'oscillatore portante sarà modulata da un oscillatore di controllo costruito con **oscili** :

```

kvibrato      oscili 1,2,1
a1      pluck  iamp,ifreq+kvibrato,220,0,1

```

è possibile sostituire oscili con lfo :

```

kvibrato      lfo      1,2 ;(sintassi : k1 lfo kamp,kfreq)
a1      pluck  iamp,ifreq+kvibrato,220,0,1

```

questo esempio crea un tipo di vibrato dal carattere freddo e meccanico,adatto per alcuni scopi ma non adatto per ricreare la naturalezza di un vero vibrato a cui siamo abituati dall'ascolto degli strumenti acustici,per ovviare a questo problema abbiamo bisogno di applicare all'oscillatore modulante alcuni elementi di variazione random,a questo scopo esistono **jitter,jitter2** e **jspline** di cui analizziamo la sintassi :

```

kout      jitter  kamp,kcpsMin,kcpsMax

```

(kcpsMin, kcpsMax indicano il range di velocità di variazioni di frequenza casuale).

```
kout    jitter2 ktotamp,kamp1,kcps1,kamp2,kcps2,kamp3,kcps3
```

simile a jitter ma con la possibilità di definire i parametri per tre punti (jitter2 è simile alla somma di tre opcode **randi** ma con differenti ampiezza e frequenza).

Esempio Cap.3.8_Vibrato

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kvibrato      oscili 3,2,1
iamp   =      p4
ifreq   =      cpspch(p5)
a1     vco2    iamp,ifreq+kvibrato,0,.8
out    a1

endin

instr 2

kvibrato      lfo      4,.7
iamp   =      p4
ifreq   =      cpspch(p5)
a1     vco2    iamp,ifreq+kvibrato,0,.8
out    a1

endin

instr 3

kvibrato jitter 6,.1,.8
iamp   =      p4
ifreq   =      cpspch(p5)
a1     pluck   iamp,ifreq+kvibrato,220,0,1
out    a1

endin

instr 4

kvibrato jitter2 6,.5,4,.1,22,.3,12
iamp   =      p4
ifreq   =      cpspch(p5)
a1     pluck   iamp,ifreq+kvibrato,220,0,1
out    a1

endin
```

```

instr 5

kvibrato jspline 5,1,8
iamp = p4
ifreq = cpspch(p5)
a1 pluck iamp,ifreq+kvibrato,220,0,1
out a1

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 1

i1 0 4 .6 8.00
s
i2 0 4 .6 8.00
s
i3 0 4 .6 8.00
s
i4 0 4 .6 8.00
s
i5 0 4 .6 8.00
s
s
</CsScore>
</CsoundSynthesizer>
```

Vediamo adesso alcuni esempi di tremolo, in questo caso l'oscillatore modulante sarà applicato all'ampiezza creando una variazione periodica del volume sonoro.

Esempio Cap.3.9_tremolo

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kenv linseg 0,p3/2,8,p3/2,0
k1 poscil .3,kenv,1
iamp = p4
ifreq = cpspch(p5)
a1 poscil iamp*(k1+1),ifreq,1
out a1

endin

</CsInstruments>
<CsScore>

f1 0 16384 10 1 0 1

i1 0 20 .6 7.00
```

```
</CsScore>
</CsoundSynthesizer>
```

Il seguente frammento :

```
a1      poscil  iamp*(k1+1), ifreq, 1
```

aggiunge il valore “1” alla variabile di controllo creando un segnale unipolare(in questo caso positivo),vedremo nei capitoli successivi come questa modalità sia alla base della sintesi AM.

Tecniche di spazializzazione

Prenderemo ora in esame alcune tecniche base di spazializzazione del suono,per prima cosa affronteremo alcune tecniche di controllo per i segnali stereofonici.Cosa significa moltiplicare un segnale audio per un valore uguale a zero e uno?immaginiamo uno strumento stereofonico,uno dei metodi più semplici per controllare il pan,ossia gestire il suono diviso tra left e right,è quello di moltiplicare il segnale audio di uno dei due canali per un valore uguale a 0 e 1.

Osserviamo la sequente riga di codice :

```
outs aleft*p4, aright*p5
```

cosa accade se il parametro p4 assume un valore = 0?in questo caso il canale audio *aleft* viene soppresso (un qualunque valore moltiplicato per zero darà sempre zero),moltiplicando invece uno dei due canali per un valore = 1 il suono rimarrà attivo,possiamo quindi immaginare come poter disporre il nostro suono all'interno del panorama stereo semplicemente moltiplicando i canali per valori variabili tra 0 e 1(un valore = 0.5 fisserà l'immagine sonora al centro).La prossima orchestra illustrerà alcune di queste tecniche,oltre al movimento fisso nello spazio è interessante creare strutture di controllo per muovere il suono nello spazio,uno degli esempi più tipici è l'effetto autopan a velocità controllabile.

Esempio Cap.3.10_controllare il suono stereo

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1      rand    .6
outs    a1*p4, a1*p5

endin

instr 2

a1      rand    .6
kpan    linseg 0, p3/2, 1, p3/2, 0
kpanl   =        kpan
```

```

kpanr = 1-kpan
outs a1*kpanl, a1*kpanr
endin
instr 3
a1 rand .6
kpan linseg 0,p3/2,1,p3/2,0
kpanl = sqrt(kpan) ; esempio di Curtis Roads
kpanr = sqrt(1-kpan)

outs a1*kpanl, a1*kpanr
endin
instr 4
asound rand .6
kpan linseg 0,p3/2,1,p3/2,0
a1,a2 pan2 asound,kpan

outs a1,a2
endin
</CsInstruments>
<CsScore>

i1 0 2 0 1
i1 3 2 1 0
i2 6 6
i3 14 5
i4 20 5
</CsScore>
</CsoundSynthesizer>
```

Analizziamo il blocco di instr 2 :

```

a1 rand .6
kpan linseg 0,p3/2,1,p3/2,0
kpanl = kpan
kpanr = 1-kpan

outs a1*kpanl,a1*kpanr
```

abbiamo una variabile di controllo che nel tempo p3 attraversa i valori da 0 a 1 per tornare a 0, cosa indica la variabile kpanr = 1 - kpan? immaginiamo i valori possibili della variabile kpan (da 0 a 1), quando kpan assumerà nel tempo p3 il valore = 1, kpanr restituirà il valore di 0 (1 - kpan), in questo istante il nostro rumore bianco verrà quindi soppresso sul canale kpanr.

Lo strumento instr 3 offre invece una variante proposta da *Curtis Roads* in cui l'opcode **sqrt** calcola la radice quadrata di kpan, secondo questa tecnica il risultato sonoro è più preciso e bilanciato sui valori estremi di left e right. Lo strumento instr 4 utilizza un semplice e pratico opcode per gestire canali stereo, si tratta del recente **pan2** la cui sintassi è :

```
a1,a2 pan2 asig,xp (con un terzo parametro "imode" opzionale)
```

dove xp potrebbe contenere il kpan linseg.... dei precedenti instr 2 e instr 3.

Un modo molto semplice per realizzare un effetto di autopan è quello di variare l'esempio di instr 2,sostituendo la variabile kpan linseg 0,p3/2,1,p3/2,0 con la seguente istruzione :

```
kpan oscil .5,.5,1
```

in questo modo otteniamo un movimento circolare continuo con velocità controllata dalla frequenza di oscil.

Esempio Cap.3.11_autopan

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1      rand    .6
k1      oscil   .5,p4,1 ;p4 controlla la velocità di rotazione
kpanl  =       k1
kpanr  =       1-k1
outs    a1*kpanl,a1*kpanr

endin

instr 2
;random pan
a1      rand    .6
kpan    randi   1,1,-1 ;valori random tra 0 e 1 con interpolazione
al,ar  pan2    a1,kpan,2
outs    al,ar

endin

</CsInstruments>
<CsScore>

f1      0        4096    10      1
i1      0        4        3
s
i1      0        4        .5
s
i2      0        20

</CsScore>
</CsoundSynthesizer>
```

Nel prossimo esempio introduciamo alcune idee per utilizzare l'audio **3D**,Csound offre alcuni opcode che fanno riferimento al concetto di "*Head Related Transfer Function*"(**HRTF**),si tratta di un processo in cui si usa un modello artificiale di testa umana per studiare in che modo il suono influisce sul meccanismo dell'udito umano.A causa della distanza e separazione delle orecchie della testa,abbiamo la capacità di percepire un audio di tipo tridimensionale.Le onde sonore arrivano al nostro udito in tempi diversi e il nostro cervello è in grado di calcolare le differenze di tempo con conseguente percezione di direzione all'interno di un campo di 360°intorno alla testa.

Le recenti versioni di csound offrono una versione migliorata dei vecchi opcode hrtf, si tratta di **hrtfmove** e **hrtfmove2**, quest'ultimo permette la riproduzione di audio 3d binaurale (da non confondere con la "stereofonia") adatto per l'ascolto in cuffia, questo opcode prende una sorgente sonora e la dispone nello spazio a 3 dimensioni intorno all'ascoltatore utilizzando la funzione di trasferimento HRTF.

la sintassi di hrtfmove2 :

```
aleft,aright      hrtfmove2      asrc,kAz,kElev,ifilel,ifiler
```

asrc : indica la sorgente sonora da spazializzare;

kAz : valore di azimuth espresso in gradi. I valori positivi rappresentano la posizione a destra, valori negativi sono la posizione a sinistra.

kElev : valore di elevazione espresso in gradi. I valori positivi rappresentano la posizione in alto, i valori negativi la posizione in basso.

ifile (left e right) : file di dati spettrali disponibili in 3 diverse frequenze di campionamento: 44.1, 48 e 96 kHz, ingresso e sr (sample rate) devono corrispondere al sample rate del datafile. I file devono essere nella directory corrente (sono contenuti nella distribuzione ufficiale di Csound5, nel nostro esempio si trovano nella stessa cartella del file sorgente .csd del capitolo 3 e sono campionati a 44.1).

Esempio Cap.3.12_audio 3d

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1    diskin2    "vocal.aiff",1,0,1
kaz   linseg      0,p3,1440 ; 4 rotazioni complete (360° * 4)
aleft,aright   hrtfmove2  a1, kaz,0,"hrtf-left.dat","hrtf-right.dat"
outs  aleft, aright

endin

</CsInstruments>
<CsScore>

i1    0      30

</CsScore>
</CsoundSynthesizer>
```

Utilizzo delle macro

Le macro in Csound consentono di operare diverse semplificazioni sul codice, di definire variabili globali o di importare file all'interno di un orchestra in modo da rendere il codice più chiaro e leggibile, il prossimo

esempio utilizza alcune macro di Csound (ogni macro viene identificata con i simboli # e \$), il nostro editor Winxound le evidenzierà in un colore diverso dagli opcode.

Esempio Cap.3.13_macros

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

#include      "nuovi_opcode.txt"

#define VOLUME  #.6#
#define FREQUENZA      #440#
#define TIMBRO   #220#

instr 1

a1      pluck  $VOLUME,$FREQUENZA,$TIMBRO,0,1
a2,a3  FX_Flanger    a1,2,1.5
adeclick      declick (a2+a3)
outs     adeclick,adeclick

endin

</CsInstruments>
<CsScore>
f1      0        4096    10      1
i1      0        8
</CsScore>
</CsoundSynthesizer>
```

La macro #include importa il file nuovi_opcode.txt che contiene alcune parti di codice per instr 1, con l'istruzione #define indichiamo delle variabili globali (dichiarate prima dei vari instr in orchestra) in questo caso associate alla variabile audio a1 (corda pizzicata).

Elementi random

L'utilizzo di generatori di numeri casuali è sempre stata una delle aree legate, tra le tante cose, alla composizione definita algoritmica. Csound offre un numero notevole di opcode di generazione random e nei prossimi esempi vedremo solo alcune delle tante applicazioni di questi metodi. Il primo opcode preso in esame è **rnd** :

rnd (x)

dove x indica un numero dato, il risultato sarà un numero casuale compreso tra 0 e il valore dato ad x, il prossimo esempio crea un oscillatore con numero di tabella scelto in modo casuale tra le varie indicate nella score, il risultato sarà un timbro diverso ad ogni evento.

Esempio Cap.3.14_rnd(x)

```
<CsoundSynthesizer>
<CsOptions>
```

```

</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

irandom =      rnd(8) ;crea numeri casuali compresi tra 0 e 8
a1      poscil  .6,220,irandom+1
out     a1

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 1
f2 0 4096 10 0 1
f3 0 4096 10 0 1 1
f4 0 4096 10 1 0 0 1
f5 0 4096 10 0 0 1 0 1
f6 0 4096 10 0 1 0 0 0 1
f7 0 4096 10 1 0 0 1 0 0 1
f8 0 4096 10 1 0 1 0 1 0 1 0 1
f9 0 4096 10 0 1 0 1 0 1 0 1 0 1

r      40
i1     0       .1

</CsScore>
</CsoundSynthesizer>
```

Adesso un esempio da considerare “storico” della computer music con cui generare frequenze casuali a partire da un range definito, l’opcode più adatto a questa funzione è **randomh** (esiste anche una variante **randomi** ma interpolata e quindi non adatta a questo scopo), la sintassi di randomh :

kres randomh kmin, kmax, kcps

definito un range di valori possiamo controllare la velocità tra i vari punti generati (kcps).

Esempio Cap.3.15_random melody

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

k1      randomh 60,1200,6
```

```

a1      oscil    .7,k1,1
out     a1
endin

</CsInstruments>
<CsScore>

f1      0        4096    10      1
i1      0        40

</CsScore>
</CsoundSynthesizer>
```

Una possibile variante dell'esempio precedente consiste nel definire una tabella di frequenze precise o meglio ancora di note (pitch), per esempio una scala maggiore, pentatonica, esatonale, cromatica, ecc in cui gli elementi random determineranno le possibili permutazioni della lista data, per definire una tabella di valori (in questo caso le note di una scala) utilizzeremo l'opcode **table** che leggerà i dati da una funzione definita nella score.

ares table andx,ifn

- andx : indice della tabella
- ifn : numero della tabella

Esempio Cap.3.16_random melody2

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

ifunction      =      2
krndmelody    randomh 1,22,8
kpitch  table   krndmelody,ifunction
a1      poscil  .6,cpspch(kpitch),1

out     a1
endin

</CsInstruments>
<CsScore>

;major scale
f1      0        4096    10      1
f2 0 32 -2 6.07 6.09 6.11 7.00 7.02 7.04 7.06
               7.07 7.09 7.11 8.00 8.02 8.04 8.06
               8.07 8.09 8.11 9.00 9.02 9.04 9.06
```

```

10.07 10.09 10.11 11.00 11.02 11.04 11.06

i1      0      20
</CsScore>
</CsoundSynthesizer>
```

In quest'ultimo esempio abbiamo prima creato una struttura di controllo random con range variabile tra 1 e 22 (**krndmelody**), kpitch table legge la funzione f2 nella score accedendo alla posizione dei suoi dati in modo casuale,in seguito abbiamo creato un semplice oscillatore che utilizza come argomento “frequenza” la variabile kpitch(con conversione in pitch in riferimento alle note contenute nella funzione f2);notiamo anche l'utilizzo di una gen nuova adatta a questo utilizzo.

Esercizio : provare a creare una variabile linseg che controlli la velocità di randomh creando accelerazioni e rallentamenti della melodia.

Processi generativi

Partendo dagli ultimi esempi di generazione eventi,prenderemo in esame alcune tecniche per la generazione di eventi complessi in cui uno strumento dell'orchestra (con funzione di controllo e non audio) controlla uno o più strumenti audio,questa tecnica permette di creare strutture con metodi di generazione definiti e non vincolati alla score,che in questo caso determinerà unicamente la durata del processo creato.

Una delle tecniche più semplici per gestire gli eventi tramite l' orchestra è l'opcode **scoreline_i**,permette di definire frammenti di score per qualunque numero di strumenti definiti.Dal punto di vista compositivo,diventa assai pratico poter costruire in questo modo,intere sezioni musicali da sottoporre a iterazioni continue,pur facendo riferimento ad un unico (instr “number”) in orchestra.

Esempio Cap.3.17_ scoreline_i

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

scoreline_i {{
i2    0    .4
i3    .2   .4
i4    .4   .4
i5    .5   .4
i2    .8   .4
} }

endin

instr 2
a1    soundin    "sample1.wav"
out   a1
endin
```

```

instr 3
a1    oscil .5,220,1
out   a1
endin

instr 4
a1    pluck .5,440,200,0,1
out   a1
endin

instr 5
a1    oscil .5,440,1
out   a1
endin

</CsInstruments>
<CsScore>

f1    0      16384 10      1      .7      .6

r     10
i1    0      .9

</CsScore>
</CsoundSynthesizer>
```

Come vediamo dall'esempio ,abbiamo definito un metodo di controllo di eventi gestiti direttamente in orchestra,l'opcode **scoreline_i** contiene nel blocco delimitato dalle parentesi graffe,una serie di istruzioni che controllano gli strumenti 2,3,4,5,questa modalità risulta essere molto efficace per lavorare con chiarezza su una singola sezione e per poter creare numerose istanze della sezione stessa,l'ultimo frammento di codice “**r 10**” permette un loop (in questo caso 10 ripetizioni) dell'istruzione “**i1 0 .9**”.

Vediamo adesso una sorta di melodia di timbri,per prima cosa creeremo un'orchestra con vari strumenti diversi tra di loro controllati da uno strumento particolare con funzione di **trigger**,ossia uno strumento che avrà la funzione di *suonare* gli strumenti creati (seguendo un ordine random),lo strumento trigger utilizzerà l'opcode **schedwhen** in cui è possibile indicare il numero di “instr” da controllare,l'inizio dell'evento e la sua durata, sintassi di **schedwhen** :

```

schedwhen ktrigger,kinsnum,kwhen,kdur

- ktrigger : il valore 0 bypassa l'evento.

- kinsnum : numero dello strumento

- kwhen : inizio dell'evento nella score (equivale a p2 nella score)

- kdur : durata dell'evento (equivale a p3 nella score)
```

Esempio Cap.3.18_random instrument

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
```

```

nchnls = 2
0dbfs = 1

instr 1
a1    vco2  .5,220,0,.8 ;sawtooth
outs  a1,a1
endin

instr 2
a1    bamboo      .5,0   ;modello di bamboo
outs  a1,a1
endin

instr 3
a1    buzz   .5,440,8,1 ;serie di armoniche sinusoidali
outs  a1,a1
endin

instr 4
a1    foscil     .5,220,400,200,2,1      ;sintesi fm
outs  a1,a1
endin

instr 5
k1    randomi   260,550,3
a1    fof      .5,220,k1,0,50,0.003,0.02,0.007,100,1,2,p3      ;formanti
outs  a1,a1
endin

instr 6
a1    soundin   "sample1.wav"      ;sample player
outs  a1,a1
endin

instr 7
axcite    oscil 1, 1, 1
a1    wgpluck   440,.5,0.3,0,10,2000,axcite ;modello di corda pizzicata
outs  a1,a1
endin

instr 8
a1    rand   .5    ;rumore bianco
outs  a1,a1
endin

instr 9
ktrigger  =  1      ;zero bypassa,1 attiva
kinsnum   =  rnd(7)  ;suona lo strumento numero....
kwhen =  0      ;p2,inizio dell'evento
kdur  =  .1      ;per la durata di .3
schedwhen  ktrigger,kinsnum+1,kwhen,kdur
endin

</CsInstruments>
<CsScore>

f1    0      16384 10      1
f2    0      16384 19      0.5    0.5    270    0.5

r      100
i9    0      .1

```

```
</CsScore>
</CsoundSynthesizer>
```

instr 9 controlla gli strumenti da 1 a 8, la scelta avviene in modo casuale (controllata da kinsnum = rnd (7) a cui sommiamo alla fine un valore=1 per evitare il numero 0 che provocherebbe un errore del compilatore in quanto non esiste un instr 0 in orchestra), abbiamo usato nuovamente l'istruzione *r number* per ripetere l'evento della score numerose volte in modo da inizializzare continuamente lo strumento trigger(che sceglierà ogni volta uno strumento in modo casuale).

Ancora una variante di **schedwhen** in cui gli eventi vengono triggerati a intervalli di tempo utilizzando l'opcode **metro** :

```
ktrig metro kfreq
```

Esempio Cap.3.19_trigger metronome

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1
ktrig metro p4      ;intervallo di tempo tra un evento e il successivo
kdur    =      p5      ;durata di ogni evento
schedkwhen ktrig,0,1,2,0,kdur
endin

instr 2
a1      poscil .6,220,1
kenv    line   1,p3,0
out     a1*kenv
endin

</CsInstruments>
<CsScore>

f1      0        4096    10      1       1       1

i1      0        10      1       1
s
i1      0        10      .5      2
s
i1      0        5       60      .1
s

</CsScore>
</CsoundSynthesizer>
```

una piccola variante ad instr 2 :

```
instr 2
kfrq    =      rnd(600)
a1      pluck  1,kfrq+100,440,0,1
```

```

kenv    line    1,p3,0
out    a1*kenv
endin

```

questo tipo di approccio può risultare molto utile per la composizione algoritmica,i prossimi esempi offrono alcuni semplici spunti di processi generativi dove l'unico limite è la fantasia.Vediamo un esempio in cui variare la velocità di generazione di una melodia,lo strumento di controllo restituirà un valore casuale per la durata di ogni evento,mentre il generatore sonoro sarà un semplice oscillatore con funzione (gen) diversa ad ogni evento (tra 16 funzioni definite),l'unica istruzione nella score sarà riferita alla durata dell'intero processo.

Esempio Cap.3.20_Generative1

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

gifn    ftgen  1,0,1024,10,1
gifn    ftgen  2,0,1024,10,1,0,1
gifn    ftgen  3,0,1024,10,1,0,0,1
gifn    ftgen  4,0,1024,10,1,0,0,0,1
gifn    ftgen  5,0,1024,10,1,0,0,0,0,1
gifn    ftgen  6,0,1024,10,1,0,1,0,0,0,1
gifn    ftgen  7,0,1024,10,1,0,0,1,0,0,0,1
gifn    ftgen  8,0,1024,10,1,0,0,0,1,0,0,0,1
gifn    ftgen  9,0,1024,10,1,0,0,0,0,1,0,0,1,1
gifn    ftgen  10,0,1024,10,1,0,0,0,0,0,1,1,0,0,1
gifn    ftgen  11,0,1024,10,1,0,0,0,0,1,1,0,0,0,1
gifn    ftgen  12,0,1024,10,1,0,0,0,1,1,0,0,0,0,0,1
gifn    ftgen  13,0,1024,10,1,0,0,0,1,1,0,0,0,0,0,1
gifn    ftgen  14,0,1024,10,1,0,0,1,0,0,1,0,0,0,0,0,1
gifn    ftgen  15,0,1024,10,1,0,1,0,0,1,0,0,0,0,0,0,0,1
gifn    ftgen  16,0,1024,10,1,1,0,0,1,0,0,0,0,0,0,0,0,1

instr  1
;crea accelerandi e rallentandi
krndtime      linseg .1,p3/4,.12,p3/4,.2,p3/4,.20,p3/4,.1
krnndur random .1,2 ;genera valori casuali per le durate
ktrig1 metro krndtime ;intervallo tra un evento e il successivo
kdur      =      krnndur ;durata di ogni evento
kwhen    =      0 ;attacco (p2)
;schedkwhen permette di indicare il nome dello strumento
schedkwhennamed ktrig1,0,100,"synth",kwhen,kdur
endin

instr synth

```

```

ifunction      =      200      ;tabella con scala maggiore in pitch
krndmelody    =      rnd (21)
kpitch table  krndmelody,ifunction
ifn     =      rnd (15)      ;sceglie un numero di gen in modo casuale
a1     oscili .1,1+cpspch(kpitch),1+ifn      ;valori di pitch in tabella
;random pan
kpan    =      birnd(1)
al,ar  pan2  al,kpan,2
kenv   linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0      ;declick
outs   al*kenv,ar*kenv

endin

</CsInstruments>
<CsScore>

f200 0 32 -2  6.07 6.09 6.11 7.00 7.02 7.04 7.06
          7.07 7.09 7.11 8.00 8.02 8.04 8.06
          8.07 8.09 8.11 9.00 9.02 9.04 9.06
          10.07 10.09 10.11 11.00 11.02 11.04 11.06

i1      0      30

</CsScore>
</CsoundSynthesizer>
```

Adesso un processo generativo con entrate in successione, è possibile controllare un numero infinito di processi sonori utilizzando istanze multiple di **schedkwhennamed** :

Esempio Cap.3.21_Generative2

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

krndtime      linseg .1,p3/4,4,p3/4,.2,p3/4,10,p3/4,.1
krnndur random .02,.1 ;durata random di ogni evento
ktrig1 metro krndtime      ;intervallo tra un evento e il successivo
kdur     =      krnndur ;durata di ogni evento (p3 nella score)
kwhen   =      0      ;attacco (p2 nella score)

schedkwhennamed      ktrig1,0,100,"percussion1",kwhen,kdur
schedkwhennamed      ktrig1,0,100,"percussion2",kwhen+4,kdur
schedkwhennamed      ktrig1,0,100,"percussion3",kwhen+8,kdur
schedkwhennamed      ktrig1,0,100,"percussion4",kwhen+12,kdur
schedkwhennamed      ktrig1,0,100,"percussion5",kwhen+24,kdur
schedkwhennamed      ktrig1,0,100,"water",kwhen+32,kdur
```

```

endin

instr  percussion1
a1      guiro   .5,.01
outs    a1,a1*0
endin

instr  percussion2
a1      cabasa  .5,.07
outs    a1*0,a1
endin

instr  percussion3
a1      sleighbells .5,.01
outs    a1,a1*0
endin

instr  percussion4
a1      tambourine .5,.01
outs    a1*0,a1
endin

instr  percussion5
a1      stix     .5,.1
outs    a1*0,a1
endin

instr  water
a1      dripwater .5,.09,10,.9
;random pan
kpan    =      birnd(1)
al,ar  pan2    a1,kpan,2
outs    al,ar
endin

</CsInstruments>
<CsScore>

i1      0          60

</CsScore>
</CsoundSynthesizer>
```

L'esempio finale incrementa una variabile di un valore definito, il risultato è una successione scalare ascendente e discendente. L'esempio introduce la creazione di veri e propri loop di sequenze utilizzando **loop_lt**:

```

loop_lt idx,incr,imax,label
definito il nome del processo (label) nella modalità :
imax      =      20      ; crea 20 iterazioni
prova:
.....
.....
.....
loop_lt idx,incr,imax,prova
```

l'opcode attiva la funzione di loop quando :

indx = indx + incr

se (indx < imax) si attiva il loop

Esempio Cap.3.22_Generative3

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1      ;strumento di controllo per instr 2,instr 3
indx    =      1      ;(indx = indx + incr,se indx < inumber si attiva il loop)
incr   =      1
inumber =    200    ;numero di iterazioni
instrument1 =     2    ;numero strumento (p1)
instrument2 =     3    ;numero strumento (p1)
iamp    =     .6    ;ampiezza oscillatore
ifreq1  =    1000   ;frequenzal
ifreq2  =     20    ;frequenza2
idelay   =     0    ;attacco (p2)
idur    =     .1    ;durata evento (p3)

sequence:    ;inizializza il processo di loop

ifreq1  =      ifreq1 - 5      ;genera una scala discendente (incremento = 5)
ifreq2  =      ifreq2 + 5      ;genera una scala ascendente (incremento = 5)
idelay  =      idelay + .1    ;intervallo tra ogni evento
event_i "i",instrument1,idelay,idur,iamp,ifreq1;scale con attacco diverso
event_i "i",instrument2,idelay+.1,idur,iamp,ifreq2
loop_lt indx,incr,inumber,sequence      ;genera sequenza

endin

instr 2      ;generatore controllato da instr 1
a1      poscil p4/2,p5,1
kenv   linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0
out    a1*kenv
endin

instr 3      ;generatore controllato da instr 1
a1      pluck  p4/2,p5,p5*2,0,1
kenv   linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0
out    a1*kenv
endin

</CsInstruments>
<CsScore>

f1      0      4096    10      10      9      8      7      6      5      4      3
i1      0      20
```

```
</CsScore>
</CsoundSynthesizer>
```

Adesso un altro esempio con loop_lt, il risultato è una melodia random (basata su una scala con pitch definiti) in continua variazione temporale.

Esempio Cap.3.23_Melodia_infinita

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

gifn    ftgen  1,0,1024,10,1
gifn    ftgen  2,0,1024,10,1,1
gifn    ftgen  3,0,1024,10,1,0,1
gifn    ftgen  4,0,1024,10,1,0,0,1
gifn    ftgen  5,0,1024,10,1,0,0,0,1

instr 1

indx    =      1
incr   =      1
iattack =      0
loop:
idur   =      rnd(1)
iattack = iattack+idur
event_i "i",2,iattack,idur
loop_lt indx,incr,1000,loop

endin

instr 2

iscale =      200      ;tabella con note in pitch
kmelody =     rnd(15) ;sceglie una nota dalla tabella
kpitch table kmelody+1,iscale ;legge la funzione con le note
kamp   =      rnd(1)  ;genera valori random per l'ampiezza
ifn    =      rnd(4)  ;sceglie una forma d'onda diversa ad ogni evento
a1    poscil .1+kamp,cpspch(kpitch),ifn+1 ;oscillatore
aenv   line    .7,p3,0 ;inviluppo per l'ampiezza

krndpan =     rnd(1) ;valori random per pan
aL,aR  pan2    a1,krndpan ;random pan
outs   aL*aenv,aR*aenv;uscita stereo con inviluppo

endin

</CsInstruments>
<CsScore>
```

```

f200 0 16 -2 7.00 7.05 7.06 7.10 8.02 8.00 8.05 8.06 8.10 9.02 9.00 9.05 9.06
9.10 10.02 10.00

i1      0      1000

</CsScore>
</CsoundSynthesizer>
```

In conclusione utilizziamo la tecnica dei loop per generare un elevato numero di sinusoidi, si tratta di un semplice esempio di sintesi additiva in cui vengono generate melodie random a velocità diversa.

Esempio Cap.3.24_200oscil

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

seed 0      ;global random, genera valori diversi ad ogni play del file

instr 1

instrument = 2
inum=1

loop:

iamp random     .1,.6 ;valori per ampiezza
ifreq random    100,12000 ;valori per frequenza
inumber      = p4      ;numero di oscillatori
irate random    .1,20 ;velocità melodia
event_i      "i",instrument,0,p3,iamp/(p4*.3),ifreq,irate
loop_lt      inum,1,inumber,loop
endin

instr 2      ;generatore di sintesi additiva

kfreq randomh   60,p5,p6      ;melodia random
al    poscil    p4,kfreq,1  ;genera un banco di oscillatori

kpan randomi    0,1,4
al,ar pan2 al,kpan

kenv linseg    0,0.02,1,p3-0.05,1,0.02,0,0.01,0    ;inviluppo declick
outs al*kenv,ar*kenv

endin
```

```

</CsInstruments>
<CsScore>

f1      0      16384 10      1

;p4    =      numero di oscillatori

i1      0      10      1
s
i1      0      10      10
s
i1      0      10      20
s
i1      0      10      30
s
i1      0      10      40
s
i1      0      10      50
s
i1      0      10      100
s
i1      0      10      200
s

</CsScore>
</CsoundSynthesizer>

```

Gen : panoramica

Negli esempi fin qui introdotti,abbiamo utilizzato prevalentemente la funzione *Gen 10*,in grado di definire una forma d'onda e le sue armoniche. Esistono in Csound numerosi metodi di generazione di funzioni con cui è possibile importare file audio,definire complessi inviluppi,definire forme d'onda per la sintesi additiva,generare rumore bianco,ecc,ecc.

La *Gen09* può essere considerata come una variante più completa della *Gen10*,permette infatti di definire il numero delle armoniche in qualunque ordine,con ampiezza e fase (espressa in gradi) indipendente,per esempio :

```
f1      0      16384 9      1      1      90
```

indica una sinusoida che inizia con fase di 90 gradi,come notiamo è necessario indicare tre parametri per ogni armonica,il prossimo esempio è composto da 3 armoniche (1,2,3) :

```
f1 0 4096      9 1 1 180 2 .9 90 3 .8 0
```

Nel paragrafo sugli inviluppi di ampiezza,abbiamo introdotto le tecniche più tipiche utilizzando degli opcode specifici a questo scopo,vediamo adesso come definire un complesso inviluppo definendo le sue caratteristiche all'interno di una tabella,introduciamo la *Gen 07* :

```
f  #      time      size      7      a      n1      b      n2      c      ...
```

I parametri *time* e *size* definiscono attacco e grandezza della tabella (che deve essere una potenza di 2),i parametri alfabetici indicano le coordinate di segmenti lineari (i numeri dispari),i parametri n1,n2,n3,ecc indicano la lunghezza di ogni segmento (numeri pari).

L'esempio seguente mostra due diversi utilizzi della *Gen07*,il primo esempio come inviluppo di ampiezza,il secondo esempio come forma d'onda,il terzo strumento utilizza una variante chiamata *Gen08*,si tratta di una funzione in grado di definire curve spline dal carattere particolarmente armonioso e dolce.

Esempio Cap.3.25_Gen07-08

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1      ;gen07 come inviluppo

iamp = .6
ienv = 2
isine = 1

kenv  poscil    iamp,1/p3,ienv
a1    poscil    kenv,440,isine
out   a1

endin

instr 2      ;gen07 come forma d'onda

iamp = .6
iwave = 2

a1    poscil    iamp,220,iwave
out   a1

endin

instr 3      ;gen08 come inviluppo

iamp = .6
ienv = 3
isine = 1

kenv  poscil    iamp,1/p3,ienv
a1    poscil    kenv,440,isine
out   a1

endin
```

```

</CsInstruments>
<CsScore>

f1    0      16384 10      1
f2    0      1024   7      1      1024   -1 ;saw down
f3    0      65     8      0      16     1      16     1      16     0      17     0

i1    0      5
s
i2    0      5
s
i3    0      5

</CsScore>
</CsoundSynthesizer>
```

La *Gen11* viene usata per generare una sintesi additiva simile ad opcode come *buzz* e *gbuzz* (sebbene sia più limitata di questi opcode), permette di definire il numero di parziali cosinusoidali nel seguente modo :

```
f # time size 11 nh [lh] [r]
```

anche in questo caso i parametri *time* e *size* definiscono attacco e grandezza della tabella (che deve essere una potenza di 2), *nh* indica il numero di parziali richieste (deve essere un numero positivo), seguono due parametri facoltativi (in cui è possibile definire l'armonica più bassa da cui iniziare e un coefficiente d'ampiezza).

Esempio Cap.3.26_Gen11

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1      ;gen11
iamp = .6
iwave = p4
kenv adsr .6,.3,1,0
a1 poscil iamp*kenv,220,iwave
out a1
endin
```

```
</CsInstruments>
<CsScore>

f1    0      16384 11      10      1
f2    0      16384 11      10      5
f3    0      16384 11      20      1
f4    0      16384 11      40      10
```

```
i1    0      5      1
s
i1    0      5      2
s
i1    0      5      3
s
i1    0      5      4
s
```

```
</CsScore>
</CsoundSynthesizer>
```

Cap.4 Csound : midi e Osc

- 4.1 Introduzione ai midi-opcodes
- 4.2 Virtual midi keyboard
- 4.3 Controller midi esterni
- 4.4 Leggere file midi
- 4.5 scale microtonali
- 4.6 Csound e Osc (open sound control)

Introduzione ai midi-opcodes

Uno dei principali fattori che spesso ha tenuto lontano aspiranti musicisti elettronici da Csound, è sempre stato quello dell'interattività. Csound nasce come linguaggio non adatto al tempo reale e le ragioni sono molteplici, principalmente il problema era legato alla potenza di calcolo dei computer che rendeva impossibile anche solo la compilazione e ascolto in tempo reale, specie con orchestre e sintesi molto complesse. La storica versione di Csound del 1992 (*Barry Vercoe*) introduceva per la prima volta i midi opcodes per far comunicare Csound con il mondo esterno attraverso il protocollo midi, purtroppo le uniche macchine dell'epoca in grado di poter svolgere una tale mole di calcolo erano solo alcune costose workstation.

Con il graduale incremento della potenza di calcolo dei personal computer, e con l'aggiunta di numerosi nuovi opcode e flag per gestire il tempo reale, Csound nel corso degli ultimi dieci anni è diventato uno strumento a tutti gli effetti interattivo, una delle chiavi di volta è stata la release della versione Direct Csound di *Gabriel Maldonado* (giugno 1998), si trattava di una versione potenziata per il live e sincronizzata alla versione di Csound 3.494, il progetto prese in seguito il nome di CsoundAV (gennaio 2002) con l'ultima release nel 2005 (sincronizzata alla versione di Csound 4.23, con supporto per interfacce grafiche, OpenGL per la grafica e numerose altre innovazioni), in particolare la potenzialità di queste versioni è quella di potersi interfacciare in modo diretto con il proprio convertitore audio (Motu, Rme, Apogee, Focusrite, ecc, ecc) sfruttando ad esempio (nel caso di macchine Windows) i driver ASIO per ridurre la latenza durante le performance live.

La recente versione standard di Csound (5.13) integra quasi totalmente gli opcode di CsoundAV (ad eccezione delle OpenGL) ed è a questa versione che faremo riferimento negli esempi di tutto il libro (questo per ragioni di portabilità e per poter sfruttare le future features di questo linguaggio)

Cosa significa concretamente usare Csound in tempo reale? L'unico limite è naturalmente la fantasia, possiamo controllare i parametri di sintesi del suono muovendo con il mouse degli slider virtuali, oppure usando un controller midi esterno, possiamo attivare o disattivare funzioni specifiche, elaborare un file audio in tempo reale, salvare sul proprio hard disk il file audio della nostra performance live in qualunque formato anche multicanale, possiamo elaborare un segnale captato da un microfono e quindi rendere perfetta l'integrazione tra strumenti acustici ed il mondo elettronico, ecc, ecc.

La lista dei midi opcode di Csound è molto ampia e comprende, tra le tante, le seguenti caratteristiche :

- poter trasmettere dati di pitch, velocity, after-touch, pitch-bend, numero di canale midi, program change, ecc
- definire sistemi di accordatura alternative (come scale microtonali)
- trasmettere dati da controller hardware midi esterni di qualunque tipo
- scrivere orchestre Csound con interoperabilità midi e score
- inviare messaggi tramite la porta midi out a strumenti esterni.

Breve introduzione al protocollo Midi

Virtual midi keyboard

Uno degli strumenti più pratici per gestire dati midi è il controller virtuale introdotto dalla versione Csound5.07 da *Steven Yi*, si tratta di una tastiera midi virtuale con totale controllo di canali midi, program change e slider completamente configurabili, la virtual midi keyboard è disponibile nelle distribuzioni di Csound5 sia per pc che Mac.

Da winxound avviamo il seguente programma :

Esempio Cap.4.1_virtual midi keyboard

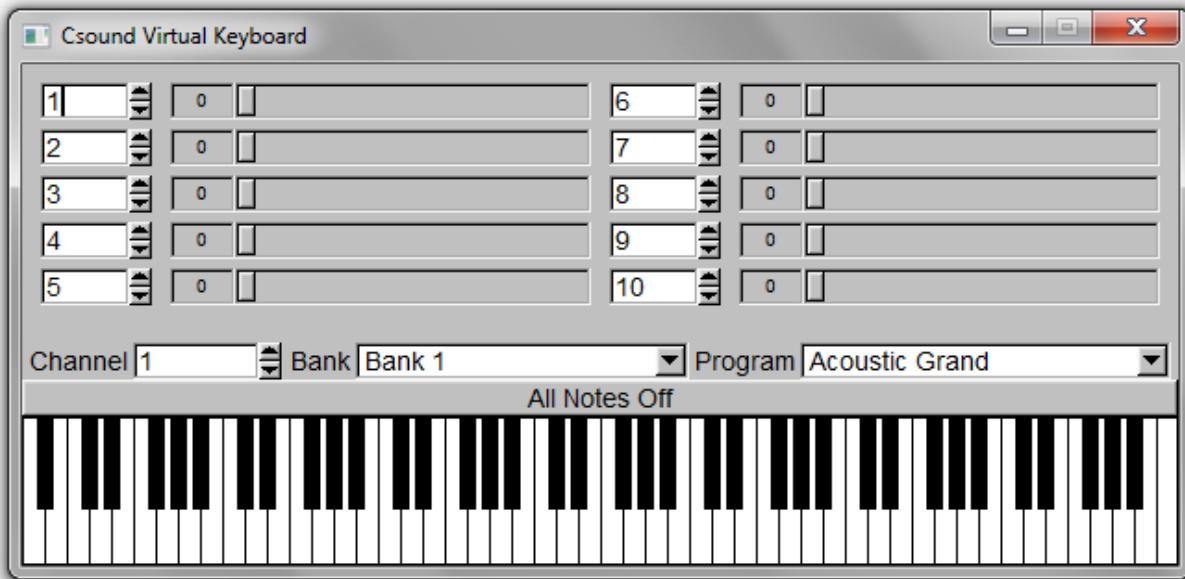
```
<CsoundSynthesizer>
<CsOptions>

-+rtmidi=virtual -M0

</CsOptions>
<CsInstruments>

instr 1
;instrument design.....
endin
</CsInstruments>
<CsScore>
i1 0 3600
</CsScore>
</CsoundSynthesizer>
```

Apparirà il nostro controller virtuale :



Analizziamo ora il breve programma :

```
<CsOptions>

-+rtmidi=virtual -M0
```

I flag di csound devono essere scritti tra questi tag(CsOptions), **-+rtmidi=virtual** attiva la virtual midi keyboard, il flag “**-M**” attiva la porta midi input (il numero 0 richiama in questo caso la tastiera virtuale, ma disponendo di altri controller per esempio le porte midi della propria scheda audio o di altri controller midi usb collegati, bisognerà indicare un numero diverso, ad esempio **-M1,-M2, ecc)**

```
</CsOptions>
```

per poter visualizzare la tastiera Csound richiede la scrittura di almeno un blocco di codice di strumento con relativo endin (nel nostro caso si tratta di uno strumento vuoto), anche se il nostro strumento non produrrà nessun tipo di suono dobbiamo comunque attivarlo dalla score per una qualsiasi durata (p3) in modo da attivare il controller midi virtuale.

```
<CsScore>
i1 0 3600 ; strumento 1 attivo per un'ora.
</CsScore>
```

Il primo esempio pratico per controllare Csound via midi sarà la modifica di uno dei primi file .csd dei precedenti capitoli, costruiremo uno strumento con la variabile audio **oscil** la cui frequenza sarà controllata in tempo reale da eventi midi (in questo caso i tasti della tastiera virtuale), per far questo utilizzeremo uno speciale opcode **cpsmidi**.

Esempio Cap.4.2_oscilmidi_A

```
<CsoundSynthesizer>
<CsOptions>
-odac -+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

iamp ampmidi      .6
ifreq cpsmidi
ifunc =    rnd(8)
a1 oscili      iamp,ifreq,ifunc+1
kdeclick linsegr    0,.1,1,.3,.5,.2,0
out a1*kdeclick

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 1
f2 0 4096 10 0 1
f3 0 4096 10 0 1 1
f4 0 4096 10 1 0 0 1
f5 0 4096 10 0 0 1 0 1
f6 0 4096 10 0 1 0 0 0 1
f7 0 4096 10 1 0 0 1 0 0 1
f8 0 4096 10 1 0 1 0 1 0 1 0 1
f9 0 4096 10 0 1 0 1 0 1 0 1 0 1
```

```
i1      0      3600
</CsScore>
</CsoundSynthesizer>
```

La variabile audio “*a1 oscili iamp,ifreq,ifunc+1*” è controllata in ampiezza e frequenza da due nuovi opcode **ampmidi** (contiene il valore della velocity) e **cpsmidi** (numero di nota midi convertita in hertz) che restituiscono ampiezza e frequenza, il terzo parametro di oscil “*ifunc*” viene controllato da un generatore di numeri casuali in modo da generare una forma d’onda sempre diversa ad ogni evento midi, per finire abbiamo applicato all’uscita audio un semplice inviluppo usato come “*declick*”.

Come possiamo facilmente osservare è mutato radicalmente l’approccio dal punto di vista musicale e compositivo, la nostra score non è più definita secondo parametri fissi ma diventa interattiva, in questo modo è possibile “suonare” Csound come un normalissimo sintetizzatore hardware o software commerciale, la nostra score in questo caso controllerà unicamente la durata della nostra performance midi.

Vediamo ora una variante dell’esempio precedente in cui è possibile estendere la durata di una nota suonata dopo il rilascio, la variabile kbend definisce il range per la funzione pitchbend di una master keyboard midi.

Esempio Cap.4.2_oscilmidi_B

```
<CsoundSynthesizer>
<CsOptions>
-odac -+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kbend pchbend      -1,1
iamp  ampmidi      .3
ifreq      cpsmidi
ifunc      =      rnd(8)
a1    oscili      iamp,ifreq*powoftwo(kbend),ifunc+1
irel = 2 ;tempo di estensione della nota
kdeclick linsegr    1,1,0,irel,0
out    a1*kdeclick

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 1
f2 0 4096 10 0 1
f3 0 4096 10 0 1 1
f4 0 4096 10 1 0 0 1
f5 0 4096 10 0 0 1 0 1
f6 0 4096 10 0 1 0 0 0 1
f7 0 4096 10 1 0 0 1 0 0 1
f8 0 4096 10 1 0 1 0 1 0 1 0 1
f9 0 4096 10 0 1 0 1 0 1 0 1 0 1

i1      0      3600
```

```
</CsScore>
</CsoundSynthesizer>
```

È anche possibile specificare un numero di canale midi per uno specifico strumento, per esempio può rivelarsi utile cambiare canale midi dalla propria master keyboard per suonare uno strumento diverso, per questa operazione esiste **massign** :

```
massign ichnl, insnum
massign ichnl, "insname"
```

il secondo permette di utilizzare un “nome” per lo strumento da attivare, massign va inserito come costante in orchestra prima della dichiarazione degli strumenti, esempio :

```
<CsoundSynthesizer>
<CsOptions>
-odac -+rtmidi=virtual -MO
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

massign      1,1 ; assegna canale 1 ad instr 1
massign      2,2 ; assegna canale 2 ad instr 2

instr 1
.....
endin

instr 2
.....
endin
```

Controller Midi esterni

Il prossimo esempio introduce l’utilizzo di particolari opcode adatti ad inviare dati midi tramite un numero di controller e canale midi prestabilito, si tratta di **ctrl7** (di cui esistono versioni più estese come **ctrl14** e **ctrl21**), nel nostro esempio questo opcode sarà associato a un midi slider virtuale, ecco la sintassi di **ctrl7** :

```
kdest ctrl7 ichan,ictlno,kmin,kmax
```

ichan : numero del canale midi (1-16)

ictlno : numero del controller midi (0-127)

kmin,kmax : range di valori

naturalmente **ctrl7** non è legato necessariamente al nostro controller virtuale ma anche a qualunque controller hardware esterno, per usare uno dei tanti midi controller in commercio basterà indicare nel flag **-M** “number” il corretto numero di porta midi in di un altro controller. Come facciamo a sapere il numero esatto con cui la nostra macchina visualizza le varie porte midi? proviamo a dare un numero molto alto al flag **-M**, ad esempio **-M88**, in questo modo Csound restituirà un errore perché sicuramente non disponiamo di 88 controller midi sul nostro computer, sarà però possibile visualizzare nella finestra di console le informazioni precise dei dispositivi midi installati sul nostro computer (questo sistema può essere utilizzato anche per il

flag `--odac` che gestisce le periferiche audio) e quindi sceglieremo il numero di controller midi esterno corretto.

Esempio Cap.4.3_ctrl7

```
<CsoundSynthesizer>
<CsOptions>
-+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

;canale midi = 1 per ogni controller

instr 1

kamp ctrl7 1,1,0,1      ;range di tra 0 e .7,ampiezza globale dello strumento

kfreq1    ctrl7 1,2,20,2000 ;controller 2 con range tra 20 e 2000
kfreq2    ctrl7 1,3,20,2000 ;controller 3 con range tra 20 e 2000
kfreq3    ctrl7 1,4,20,2000 ;//
kfreq4    ctrl7 1,5,20,2000 ;//
kfreq5    ctrl7 1,6,20,2000 ;//
kfreq6    ctrl7 1,7,20,2000
kfreq7    ctrl7 1,8,20,2000
kfreq8    ctrl7 1,9,20,2000
kfreq9    ctrl7 1,10,20,2000

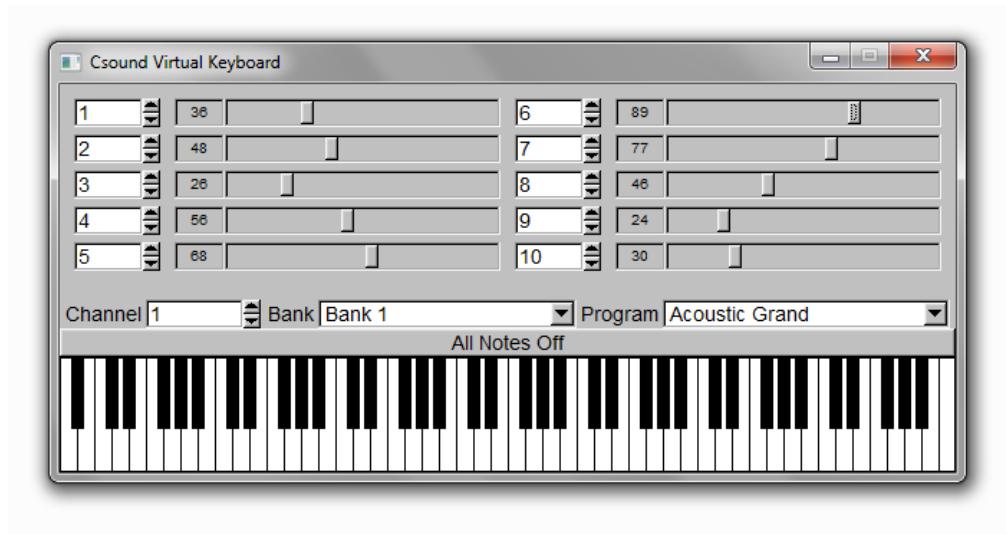
a1  oscili     kamp,kfreq1,1      ;kamp e kfreq gestiti dal controller midi
a2  oscili     kamp,kfreq2,1
a3  oscili     kamp,kfreq3,1
a4  oscili     kamp,kfreq4,1
a5  oscili     kamp,kfreq5,1
a6  oscili     kamp,kfreq6,1
a7  oscili     kamp,kfreq7,1
a8  oscili     kamp,kfreq8,1
a9  oscili     kamp,kfreq9,1
asum sum   a1,a2,a3,a4,a5,a6,a7,a8,a9      ;sommiamo le variabili audio
out  asum/9
endin

</CsInstruments>
<CsScore>

f1    0      4096 10      1
i1    0      3600 ;strumento 1 acceso per un'ora
</CsScore>

</CsoundSynthesizer>
```

In questo esempio abbiamo creato uno strumento in sintesi additiva di semplici sinusoidi, la variabile `kamp` controllerà l'ampiezza globale dello strumento (slider 1), le frequenze saranno controllate dagli slider della virtual keyboard (slider controller dal numero 2 al 10), abbiamo utilizzato la versione interpolata di **oscili** molto adatta a questo scopo proprio per rendere più fluido e musicale il passaggio tra un valore e l'altro.



Come facilmente starete osservando le possibilità creative sono davvero infinite, proviamo adesso a costruire uno strumento con ampiezza, frequenza, vibrato e velocità di autopan controllati in tempo reale dai midi slider virtuali.

Esempio Cap.4.4_vibratomidi

```
<CsoundSynthesizer>
<CsOptions>
-odac -+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

;canale midi = 1
;slider 1 = ampiezza oscillatore vco
;slider 2 = frequenza oscillatore vco
;slider 3 = ampiezza vibrato
;slider 4 = frequenza vibrato
;slider 5 = rate autopan

kamp    ctrl7  1,1,0,.6      ;range di ampiezza tra 0 e 1000
kfreq   ctrl7  1,2,20,2000    ;frequenza oscillatore
kampvib ctrl7  1,3,1,20      ;ampiezza vibrato
kfreqvib  ctrl7  1,4,.1,20    ;frequenza vibrato
kvib    oscili  kampvib,kfreqvib,1    ;vibrato

a1      oscili  kamp,kfreq+kvib,1      ;variabile audio

krate   ctrl7  1,5,.1,3      ;controlla la velocità di autopan
kautopan oscil   .5,krate,1    ;autopan
aleft   =       a1 * (1-kautopan)    ;Pan Left
aright  =       a1 * kautopan  ;Pan Right
outs    aleft, aright

endin
```

```

</CsInstruments>
<CsScore>

f1      0      4096    10      1      0      .4      0      .3
i1      0      3600    ;strumento 1 acceso per un'ora

</CsScore>
</CsoundSynthesizer>

```

Leggere file midi

I midi opcodes oltre a ricevere dati midi da controller esterni possono anche importare e leggere midifile,Csound mette a disposizione il flag -F per leggere un midi file :

```

<CsOptions>
-odac -M0 -F "Nomedelfile"
</CsOptions>

```

Con questo comando stiamo aprendo una porta audio (odac) e midi input (-M) a cui segue il nome del file da suonare in tempo reale da Csound,con le recenti versioni è anche possibile gestire file midi multicanale,con questo esempio il file midi dovrà trovarsi nella stessa cartella del file .csd,è anche possibile specificare l'eventuale directory del file.

Con quale timbro suoneremo il nostro midi file?questo dipenderà dal metodo di sintesi scelto,il prossimo esempio utilizzerà una semplice corda pizzicata (pluck),vedremo anche un esempio di hard disk recording con cui Csound scriverà un file audio stereo della nostra performance midi.

Esempio Cap.4.5_midifile to disk

```

<CsoundSynthesizer>
<CsOptions>
-F "Violin.mid"
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

#define      NOMEFILE      #"sample.wav"#
gaout1,gaout2      init  0
massign      1,1
instr 1
iamp  ampmidi      .6
ifreq  cpsmidi
a1  pluck iamp,ifreq/2,220,0,1
kdeclick  linsegr    0,.1,1,.3,.5,.2,0

```

```

outs  a1*kdeclick,a1*kdeclick

vincr gaout1,a1*kdeclick
vincr gaout2,a1*kdeclick

endin

instr 2

fout  $NOMEFILE,4,gaout1,gaout2

clear gaout1,gaout2

endin

</CsInstruments>
<CsScore>

i1      0      18
i2      0      18

</CsScore>
</CsoundSynthesizer>
```

Analizziamo alcuni frammenti importanti di questo esempio,abbiamo creato uno strumento “1” (corda pizzicata) con ampiezza e frequenza controllate via midi,il secondo strumento svolge funzione di hard disk recording renderizzando sul nostro computer la performance midi appena eseguita(nel nostro caso la lettura di un file midi),notiamo l'utilizzo di variabili al di fuori del blocco strumento,si tratta delle variabili globali (devono iniziare con “ga”) accessibili da tutti gli strumenti dell'orchestra,instr 1 utilizza l'opcode **vincr** per sommare due segnali,quali sono i due segnali di questo esempio?si tratta della variabile a1 (pluck) sommata alla variabile globale.

```

vincr gaout1,a1
vincr gaout1,a1
```

a questo punto il segnale accumulato dall'opcode **vincr** passa ad instr 2 :

```
fout $NOMEFILE,4,gaout1,gaout2
```

l'opcode **fout** scriverà un file audio sul nostro hard disk,il numero 4 indica il tipo di file audio scelto,fout offre numerose possibilità di scelta,dal *canonical reference* :

```

0 - 32-bit floating point samples without header (binary PCM multichannel file)
1 - 16-bit integers without header (binary PCM multichannel file)
2 - 16-bit integers with a header. The header type depends on the render (-o) format. For example, if the user chooses the AIFF format (using the -A flag), the header format will be AIFF type.
3 - u-law samples with a header (see iformat=2).
4 - 16-bit integers with a header (see iformat=2).
5 - 32-bit integers with a header (see iformat=2).
6 - 32-bit floats with a header (see iformat=2).
7 - 8-bit unsigned integers with a header (see iformat=2).
8 - 24-bit integers with a header (see iformat=2).
9 - 64-bit floats with a header (see iformat=2).
```

In conclusione troviamo l'opcode **clear** che serve ad evitare effetti di accumulo riazzerando le variabili globali al valore di zero (equivale all'istruzione gaout = 0).

Scale microtonali

Introduciamo ora un importante opcode con cui sarà possibile definire sistemi alternativi d'intonazione, negli esempi precedenti abbiamo utilizzato l'opcode **cpsmidi** (numero di nota midi convertita in hertz) per controllare i metodi di sintesi attraverso una tastiera midi, immaginiamo di voler suddividere l'ottava della nostra master keyboard utilizzando ad esempio quarti di tono, quindi intervalli più piccoli dei toni e semitonni appartenenti alla nostra cultura occidentale, con Csound è possibile scrivere sistemi d'accordatura inusuali utilizzando l'opcode **cpstmid**, la sua sintassi :

```
icps cpstmid ifn
```

Ifn è legato ad una tabella in cui andremo a definire i rapporti intervallari della nostra scala micro tonale, come **cpsmidi** anche **cpstmid** è nato per ricevere dati da strumenti midi, per poter definire la scala micro tonale è necessario indicare alcuni parametri iniziali come il numero di note della scala, la sua frequenza base, ecc, nell'ordine :

1. *numgrades* : numero di note della scala;
2. *interval* : l'estensione della scala, ad esempio il valore 2 per un'ottava;
3. *basefreq* : frequenza base della scala in Hz;
4. *basekeymidi* : nota midi alla quale la frequenza base è assegnata;

dopo i quattro parametri fondamentali è possibile definire i singoli rapporti (ratios) intervallari, ecco un esempio di tabella semplice per una tradizionale scala di 12 semitonni uguali :

```
f1 0 64 -2 12 2 261 60 1 1.059463094359 1.122462048309 1.189207115003..etc...
```

vediamo adesso un esempio completo di sintesi in tempo reale con l'utilizzo di scale microtonali, le tabelle che definiscono i sistemi di scale alternativi sono messi per comodità nell'orchestra attraverso l'opcode **ftgen**, in ordine troviamo le seguenti scale : Standard tuning, Quarter tones, Decatonic, Pythagorean, Third tones, Detuned, Harmonic, Chinese, Major Triad, Fibonacci, Pentatonic, Bohlen-Pierce, Bharata tuning, Indian shruti, Wendy Carlos Alpha.

Esempio Cap.4.6_ microtonal

```
<CsoundSynthesizer>
<CsOptions>
-+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

#include      "tuning.h"

;101 Standard tuning
;102 Quarter tones
;103 Decatonic
;104 Pythagorean
;105 Third tones
;106 Detuned
;107 Harmonic
;108 Chinese
;109 Major Triad
```

```

;110 Fibonacci
;111 Pentatonic
;112 Bohlen-Pierce
;113 Bharata tuning
;114 Indian shruti
;115 Wendy Carlos Alpha

FLpanel "Virtual Midi keyboard",600,200,90,90;start of container
FLvkeybd      "keyboard.map", 560, 100, 20, 52
FLpanelEnd

FLrun

;strumento 1
instr 1

iamp    ampmidi .6
ifreq   cpstmid 112      ;numero della tabella contenente la scala microtonale
a1      pluck  iamp,ifreq,220,0,1
kenv   linsegr 0,.1,1,1,.1,.1,0
outs   a1*kenv,a1*kenv

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 .9 .8 .7 .6 .5 .4 .3 .2 .1
f0      3600

</CsScore>
</CsoundSynthesizer>

```

Csound e OSC

Il protocollo OSC (*Open sound control*) è un sistema di comunicazione tra strumenti elettronici molto più recente del sistema Midi, è stato sviluppato presso CNMAT da *Adrian Freed* e *Matt Wright*. Si tratta di un protocollo di trasmissione in cui i dati passano attraverso una rete (TCP/IP, internet, ecc), il vantaggio rispetto al sistema Midi riguarda, tra le tante cose, il maggior numero di scambio dati (1024 messaggi) e latenza minore. Tuttavia non dobbiamo necessariamente pensare ad OSC in riferimento al protocollo midi, dato che quest'ultimo rappresenta ancora oggi un valido e stabile sistema di scambio dati, allo stato attuale. Una tipica performance con strumenti elettronici che utilizzano OSC, utilizza controller wireless, quindi strumenti di controllo di nuova generazione (ad esempio i tablet pc) senza l'ausilio di cavi per il collegamento tra gli strumenti, questo porta ad enormi possibilità di ricerca nel campo multimediale, immaginiamo il controllo di un motore di sintesi da parte di un sensore di movimento applicato sul corpo di un attore / danzatore.

Nelle ultimissime release di Csound sono stati introdotti vari tipi di opcode per il controllo tramite OSC, le applicazioni con il mondo hardware spaziano dall'utilizzo dei *tablet pc* fino ai games controller come la *Nintendo Wii remote*. L'esempio seguente utilizzerà gli opcode :

```

ihandle    OSCinit     iport
kans    OSCListen   ihandle,idest,itype[,xdata1,xdata2,...]

```

vediamo una tipica architettura in Csound che utilizza il protocollo OSC :

```

;numero di porta per la comunicazione tra Csound e un dispositivo esterno
giOsc1      OSCinit      5000

instr 1

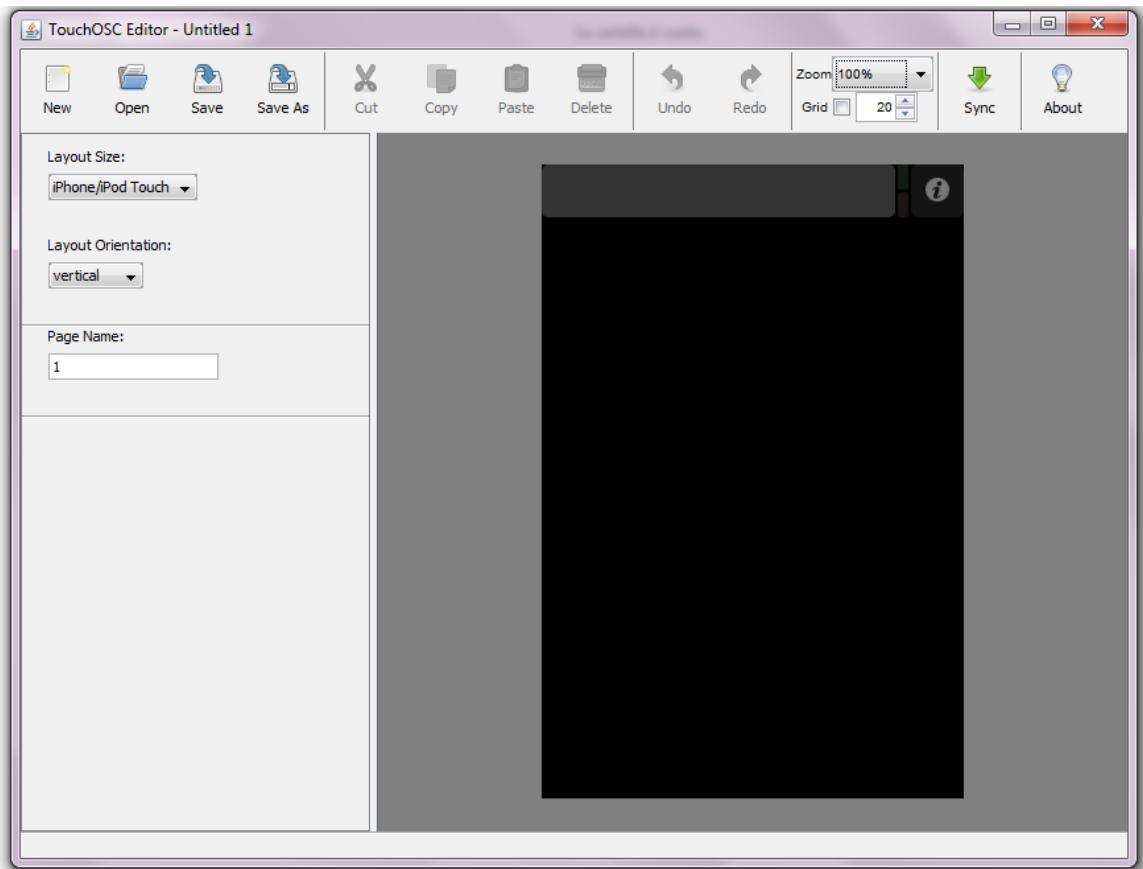
kslider      init  0      ;valore di inizializzazione
k1      OSCListen    giOsc1,"/1/fader1","i",kslider      ;slider di controllo
a1      oscili      iamp,kslider,1
. . . . .

```

Il prossimo esempio pratico sarà basato sul sistema di comunicazione tra Csound ed un dispositivo di controllo mobile come *Iphone, Ipad*.

Questi sistemi (ma anche dispositivi alternativi con il sistema operativo *Android*) utilizzano uno strumento software con cui costruire il proprio controller OSC personalizzato, si tratta di *Touch Osc* (<http://hexler.net/software/touchosc>)

Per prima cosa installiamo sul nostro computer l'editor di Touch Osc e creiamo un nuovo progetto :



Il pannello visualizzato rappresenta lo sfondo su cui disegnare la propria interfaccia di controllo, Touch Osc contiene numerosi widgets come slider, pulsanti, joystick, ecc. Inseriamo un semplice slider scegliendolo tramite il tasto destro del mouse (selezionare Fader V o Fader H).

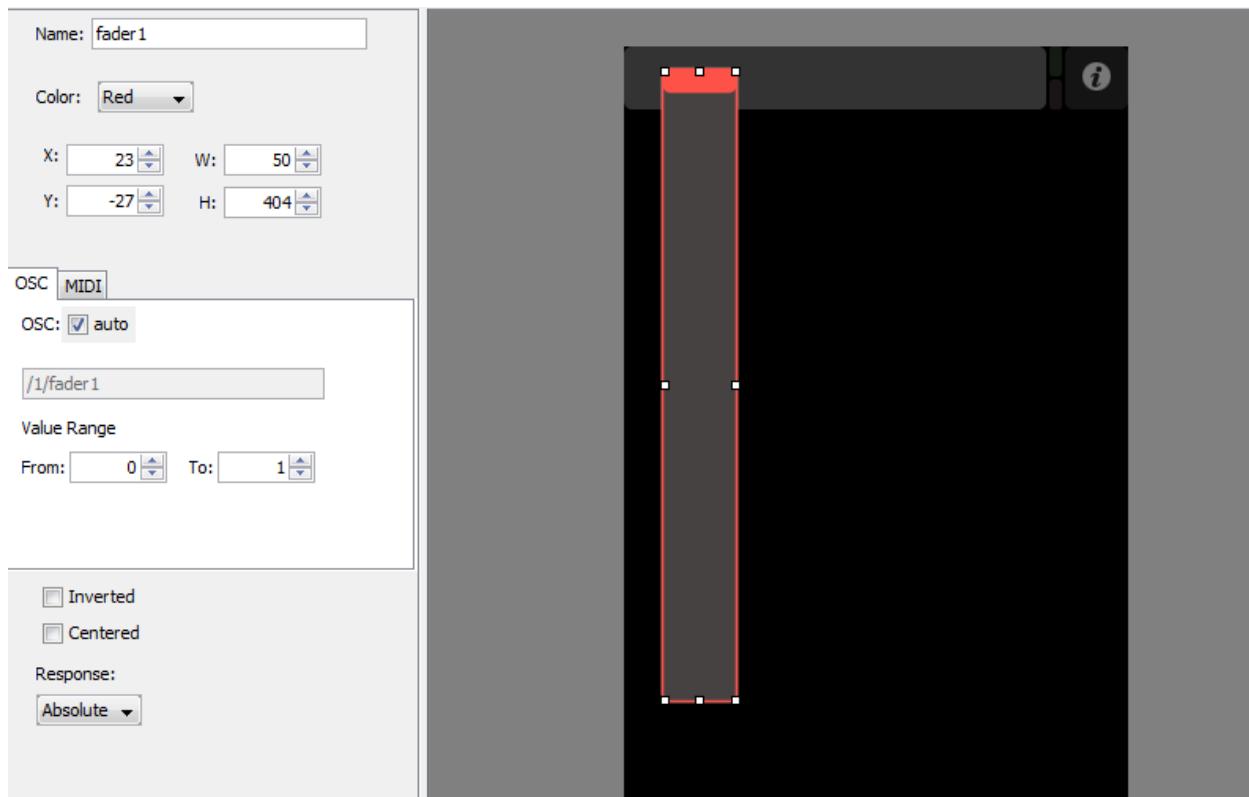
Nel pannello sulla destra possiamo scrivere il nome identificativo dell'oggetto grafico, il quale sarà associato allo strumento Csound di controllo, in questo caso l'oggetto Fader prenderà di default il nome :

/1/fader1

L'orchestra Csound comunicherà con l'interfaccia di controllo nel seguente modo :

```
k1      OSCListen      giOsc1, "/1/fader1", "i", kslider
```

Nella sezione range indichiamo i valori minimo e massimo.



Vediamo adesso un esempio completo, apriamo dall'editor di Touch Osc il file *foscili.touchosc* e attiviamo il processo di invio al dispositivo mobile tramite la funzione *Sync*.



A questo punto apriamo Touch Osc dal nostro dispositivo mobile, nel pannello principale troviamo la categoria *Connections*, apriamo la voce "OSC :" e indichiamo il numero IP del computer in cui verrà eseguito Csound, nella finestra aperta andiamo a modificare i seguenti settaggi :

```
host : IP del computer (esempio 192.168.1.3)
enabled OK

port Outgoing 5000 (il numero di porta con cui il dispositivo comunicherà con
Csound)

port incoming 0 (default)
```

chiudiamo la finestra e torniamo nel pannello principale alla voce *Layout*, dalla prima voce della lista arriviamo alla funzione *Add* con cui Touch Osc effettuerà una piccola scansione dei dispositivi connessi (*Found Host ()*), appena identificato il computer (ricordiamo l'editor in modalità *Sync*) sarà possibile importare il file *foscili.touchosc* all'interno del nostro tablet.

A questo punto attiviamo il nostro controller personalizzato importato dalla lista in *Layout*, se tutto è andato a buon fine dovreste visualizzare sul dispositivo mobile il controller precedentemente disegnato con l'editor.

Analizziamo il seguente file di Csound, apriamo *Cap.4.7_foscili OSC* e proviamo a inviare dati tramite Touch Osc, il tipo di sintesi scelto per esempio è la modulazione di frequenza, tecnica che verrà discussa nei prossimi capitoli.

Esempio Cap.4.7_foscili OSC

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

giOsc1      OSCinit      5000 ;porta di comunicazione con TouchOsc

instr 1

kcar init 0      ;inizializzo le variabili
kmod init 0
kindx init 0
kamp init 0

k1    OSCListen  giOsc1,"/1/fader1","i",kcar
k2    OSCListen  giOsc1,"/1/fader2","i",kmod
k3    OSCListen  giOsc1,"/1/fader3","i",kindx

a1    foscili     .6,1,kcar,kmod,kindx,1

out    a1
```

```
endin

</CsInstruments>
<CsScore>

f1      0      4096   10      1
i1      0      3600 ;strumento attivo per un'ora.

</CsScore>
</CsoundSynthesizer>
```

Abbiamo realizzato un semplice sistema di comunicazione basato su OSC, software come TouchOsc permettono di creare strumenti di controllo creativi, personalizzabili fino all'estremo, superando i limiti evidenti dei controller hardware nati con caratteristiche specifiche.

Cap.5 Interfacce grafiche(Fltk,Max/Msp)

5.1 La libreria Fltk

5.2 Fltk panel

5.3 Knobs,sliders,buttons,counter,Joysticks

5.5 Libreria grafica con qutecsound

5.6 Csound~ e Max/Msp

La libreria Fltk

In questa sezione introdurremo le tecniche per costruire Gui(Graphical User Interface) per rendere interattivi i nostri strumenti virtuali,Csound5 integra una libreria specifica di oggetti grafici chiamata **Fltk** (Fast Light Tool Kit),si tratta di una libreria veloce, performante, multiplataforma e con una sintassi relativamente semplice.Con questa libreria di oggetti possiamo costruire strumenti grafici di controllo per i nostri sintetizzatori,campionatori,processori di segnale.

Fast Light Tool Kit si divide principalmente in *Containers* e *Valuators* :

Containers :

- Panels
- Scroll areas
- Pack
- Tabs
- Groups

Valuators :

- Sliders
- Knobs
- Rollers
- Text fields
- Joysticks
- Counters

Osserviamo lo scheletro di un file csd per capire le regole base di utilizzo di oggetti grafici,ogni opcode della Fltk deve essere inserito dopo l'intestazione iniziale,non è possibile utilizzare Fltk all'interno di uno strumento.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

FLpanel      "Spazio per testo",iwidth,iheight,ix,iy
FLpanelEnd

FLrun      ;inizializza il thread grafico

instr 1
;blocco di codice per strumento
endin

</CsInstruments>
<CsScore>
f0    3600  ;durata della performance in real time
</CsScore>
</CsoundSynthesizer>
```

quella che vediamo è la struttura base per costruire una finestra all'interno di uno strumento Csound,**FLpanel** e **FLpanelEnd** delimitano il *contenitore*,è possibile inizializzare numerosi contenitori dei quali bisognerà definire le dimensioni e le coordinate nello spazio,definiti i contenitori si procede all'avvio

del motore grafico con **FLrun**, esattamente come abbiamo già visto nella sezione relativa al midi, anche in questo caso la funzione della score è unicamente quella di tenere “accesa” l’istanza del nostro strumento (f0 3600).

Fltk panel

Vediamo adesso un esempio pratico di **FLpanel** con cui disegnare due finestre della stessa grandezza e colori diversi, la sintassi di FLpanel :

```
FLpanel "testo", iwidth, iheight[, ix][, iy][, iborder][, ikbdcapture][, iclose]
```

I parametri fondamentali sono i primi quattro (dimensioni e coordinate), gli altri parametri facoltativi sono :

iborder : è possibile scegliere il tipo di bordo (da 0 a 7).

Ikbdcapture : cattura eventi da tastiera, con valore = 0 viene disabilitata la funzione

IClose : settato a valore = 1 impedisce alla finestra di chiudersi tramite il pulsante close, funzione molto importante specialmente quando si opera con molte finestre.

Per modificare il colore della finestra utilizziamo l’opcode :

```
FLcolor ired, igreen, iblue
```

FLcolor imposta i colori primari in valori RGB (si consiglia di consultare una tabella rgb per impostare i vari colori).

Esempio Cap.5.1_FLpanel

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

iwidth      =      500    ; larghezza
iheight     =      500    ; altezza
ix          =      50     ; coordinate x e y per la posizione nello spazio
iy          =      50
iborder     =      5      ; bordo con linea nera
ikbdcapture =      0      ; disabilita cattura eventi da tastiera
iclose      =      1      ; disabilita il pulsante close della finestra
; prima finestra
FLcolor     0,250,0    ; colore della finestra definito in RGB
FLpanel    "FLTK panel 1",iwidth,iheight,ix,iy,iborder,ikbdcapture,iclose
FLpanelEnd

; seconda finestra
FLcolor     110,180,210
FLpanel    "FLTK panel 2",iwidth,iheight,ix+520,iy,iborder,ikbdcapture,iclose
FLpanelEnd

FLrun ;inizializza il thread grafico

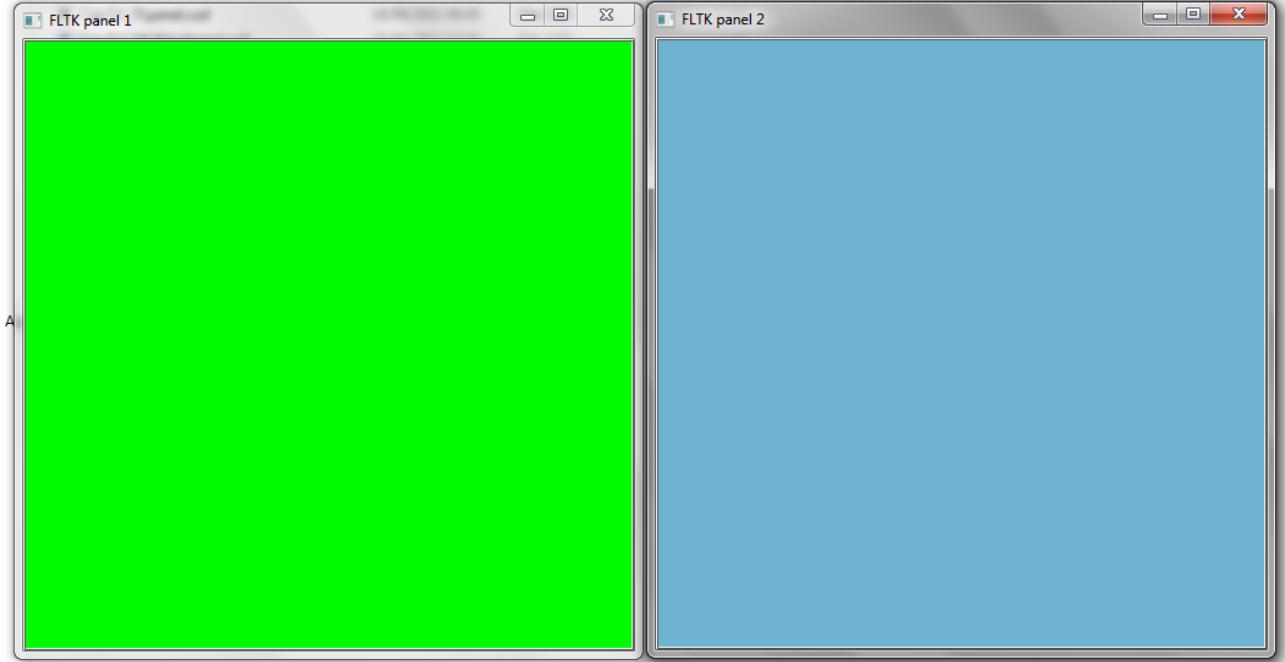
instr 1
; .....
endin

</CsInstruments>
```

```

<CsScore>
f0    3600 ;durata della performance in real time
</CsScore>
</CsoundSynthesizer>

```



Introduciamo ora l'opcode **FLtabs** con il quale è possibile organizzare diverse finestre all'interno di un unico contenitore (**FLpanel**), le varie finestre sovrapposte saranno richiamate tramite pulsanti , la struttura per visualizzare due finestre sovrapposte e richiamabili può essere rappresentata dal seguente schema :

```

FLpanel "Multipanel",500,500,100,100
FLtabs itabswidth,itabsheight,ix,iy
;prima finestra
FLgroup "Panel1",itabwidth,itabheight,ix,iy
FLgroup_end
;seconda finestra
FLgroup "Panel2",itabwidth,itabheight,ix,iy
FLgroup_end
FLtabsEnd
FLpanelEnd
FLrun

```

FLtabs è un' interfaccia utile per visualizzare le diverse aree che si alternano nella stessa finestra (creata inizialmente da **FLpanel**), gli opcode **FLgroup** descrivono l'area all'interno della finestra principale, osserviamo come l'intera struttura sia aperta e chiusa dal blocco di codice con **FLpanel** e **FLpanelEnd**,

ora un esempio completo :

Esempio Cap.5.2_Multiwindows

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

FLpanel "Multiwindows",500,500,100,100

itabswidth = 500
itabsheight = 500
ix = 5
iy = 5
FLtabs itabswidth,itabsheight,ix,iy

;prima finestra
itablwidth = 500
itablheight = 500
itablx = 10
itabley = 40
FLgroup "Panel1",itablwidth,itablheight,itablx,itabley
FLgroup_end

;seconda finestra
itab2width = 500
itab2height = 500
itab2x = 10
itab2y = 40
FLgroup "Panel2",itab2width,itab2height,itab2x,itab2y
FLgroup_end

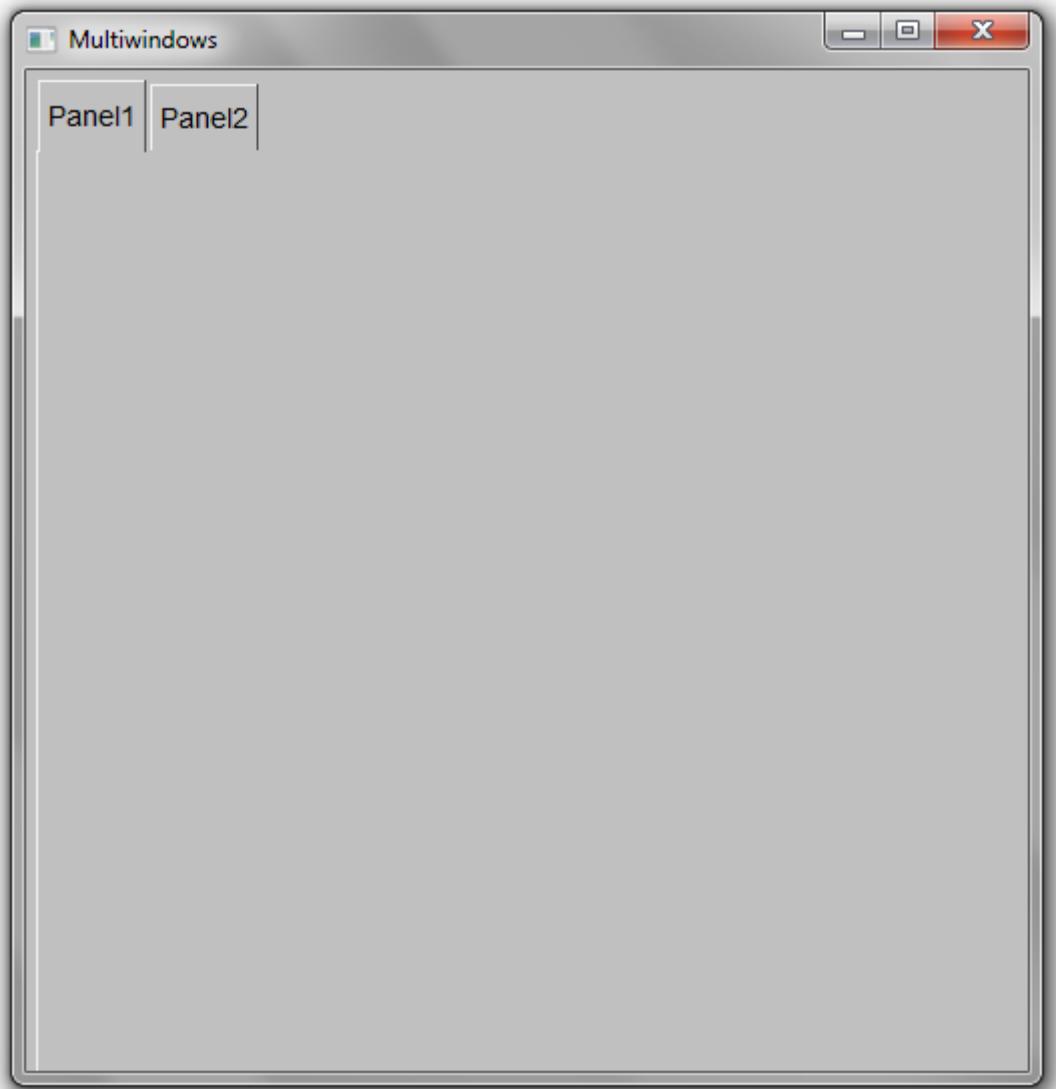
FLtabsEnd
FLpanelEnd

FLrun

instr 1

endin

</CsInstruments>
<CsScore>
f0 3600
</CsScore>
</CsoundSynthesizer>
```



Knobs,sliders,buttons,counter,Joysticks

Introduciamo adesso una serie di oggetti molto utili per costruire strumenti grafici per il controllo in tempo reale dei nostri strumenti csound,Fltk offre una serie di unità base come pulsanti,slider,contatori,ognuno con caratteristiche e possibilità visuali molto ampie,il prossimo esempio mostra l'utilizzo di **FLknob** e **FLvalue** con cui visualizzare il valore in uscita dal controllo rotativo :

Esempio Cap.5.3_FLknob

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

FLpanel      "Knob type",450,200,100,200,1,-1,-1

;ihandle FLvalue "label",iwidth,iheight,ix,iy
ival1 FLvalue "",60,30,35,125
ival2 FLvalue "",60,30,35+100,125
ival3 FLvalue "",60,30,35+200,125
ival4 FLvalue "",60,30,35+300,125

imin = 1
```

```

imax = 100
iexp = -1
iwidth = 75
ix = 20
iy = 20

;kout,ihandle FLknob "label",imin,imax,iexp,itype,idisp,iwidth,ix,iy
gk1,ih1 FLknob "3-D",imin,imax,iexp,1,ival1,iwidth,ix,iy
gk2,ih2 FLknob "pie",imin,imax,iexp,2,ival2,iwidth,ix+100,iy
gk3,ih3 FLknob "clock",imin,imax,iexp,3,ival3,iwidth,ix+200,iy
gk4,ih4 FLknob "flat",imin,imax,iexp,4,ival4,iwidth,ix+300,iy

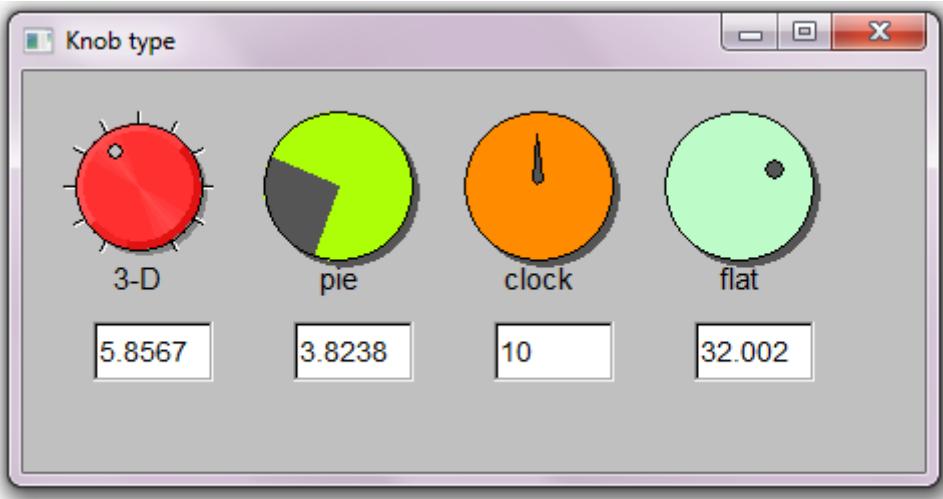
;FLSetTextSize isize,ihandle
FLSetTextSize 15,ih1
FLSetTextSize 15,ih2
FLSetTextSize 15,ih3
FLSetTextSize 15,ih4
;FLSetTextType itype(0-5),ihandle
FLSetTextType 2,ih1
FLSetTextType 2,ih2
FLSetTextType 2,ih3
FLSetTextType 2,ih4
;FLSetAlign ialign(1-9),ihandle
FLSetAlign 3,ih1
FLSetAlign 3,ih2
FLSetAlign 3,ih3
FLSetAlign 3,ih4
;FLSetColor ired,igreen,iblue,ihandle
FLSetColor 255,48,48,ih1
FLSetColor 173,255,7,ih2
FLSetColor 255,140,0,ih3
FLSetColor 189,252,201,ih4
;set color Value box
FLSetColor 255,255,255,ival1
FLSetColor 255,255,255,ival2
FLSetColor 255,255,255,ival3
FLSetColor 255,255,255,ival4
;*****
FLpanelEnd
FLrun

instr 1

endin

</CsInstruments>
<CsScore>
f0 3600
</CsScore>
</CsoundSynthesizer>

```



Prendiamo in esame il seguente frammento di codice per capire come passare il valore di Knob all'oggetto **FLValue**:

```
ival1 FLvalue "",60,30,35,125
gk1,ih1 FLknob "3-D",imin,imax,iexp,1,ival1,iwidth,ix,iy
```

le proprietà di **FLvalue** riguardano unicamente le dimensioni e la posizione nello spazio,**FLknob** è un oggetto più complesso con altri parametri legati al tipo di controllo rotativo da visualizzare (3D,clock,flat,pie) e in cui è viene indicato il range di valori per il controllo,il parametro **iexp** indica i valori in uscita :

iexp = 0 (valore lineare)

iexp = -1 (valore esponenziale)

(l'utilizzo di questo parametro iexp è identico per tutti gli altri tipi di oggetti grafici di controllo)

Come possiamo osservare nell'esempio precedente,FLknob viene indicato con due variabili :

gk1 (globale,sarà visualizzato da tutti gli strumenti dell'orchestra)

ih1 (inizializzazione,nome indicativo per le proprietà grafiche di questo oggetto)

esistono numerosi altri opcode per controllare ogni singolo aspetto grafico dell'oggetto come il colore,oppure per cambiare font e dimensioni per il nome indicativo dell'oggetto,il frammento seguente :

```
gk1,ih1 FLknob "3-D",imin,imax,iexp,1,ival1,iwidth,ix,iy
FLsetColor 255,48,48,ih1
FLsetTextSize 15,ih1
```

I prossimi esempi illustrano gli altri oggetti di questa libreria grafica come slider,bottoni,joystick e molto altro,la sintassi e l'utilizzo dei prossimi opcode non si discosta dall'esempio precedente con FLknob.

Esempio Cap.5.4_FLslider

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

FLcolor 224,255,255
FLpanel "Horizontal slider",630,350,100,100,1,-1,-1

;ihandle FLvalue "label",iwidth,iheight,ix,iy
ival1 FLvalue "",60,30,540,20
ival2 FLvalue "",60,30,540,70
ival3 FLvalue "",60,30,540,120
ival4 FLvalue "",60,30,540,170
ival5 FLvalue "",60,30,540,220
ival6 FLvalue "",60,30,540,270

;"plastic" looking sliders(itype = 20)
imin = 1
imax = 100
iexp = -1
iheight = 30
iwidth = 500
ix = 20
iy = 20
;kout,ihandle FLslider "label",imin,imax,iexp,itYPE,idisp,iwidth,iheight,ix,iy
gk1,ih1 FLslider "",imin,imax,iexp,1,ival1,iwidth,iheight,ix,iy
gk2,ih2 FLslider "",imin,imax,iexp,3,ival2,iwidth,iheight,ix,iy+50
gk3,ih3 FLslider "",imin,imax,iexp,5,ival3,iwidth,iheight,ix,iy+100
gk4,ih4 FLslider "",imin,imax,iexp,21,ival4,iwidth,iheight,ix,iy+150
gk5,ih5 FLslider "",imin,imax,iexp,23,ival5,iwidth,iheight,ix,iy+200
gk6,ih6 FLslider "",imin,imax,iexp,25,ival6,iwidth,iheight,ix,iy+250

FLpanelEnd
;*****
FLpanel "Vertical slider",330,350,750,100,1,-1,-1
;ihandle FLvalue "label",iwidth,iheight,ix,iy
ival7 FLvalue "",50,30,10,270
ival8 FLvalue "",50,30,60,270
ival9 FLvalue "",50,30,110,270
ival10 FLvalue "",50,30,160,270
ival11 FLvalue "",50,30,210,270
ival12 FLvalue "",50,30,260,270

iwidth2 = 30
;kout,ihandle FLslider "label",imin,imax,iexp,itYPE,idisp,iwidth,iheight,ix,iy
gk7,ih7 FLslider "",imin,imax,iexp,2,ival7,iwidth2,iheight+200,ix,iy
gk8,ih8 FLslider "",imin,imax,iexp,4,ival8,iwidth2,iheight+200,ix+50,iy
gk9,ih9 FLslider "",imin,imax,iexp,6,ival9,iwidth2,iheight+200,ix+100,iy
gk10,ih10 FLslider "",imin,imax,iexp,22,ival10,iwidth2,iheight+200,ix+150,iy
gk11,ih11 FLslider "",imin,imax,iexp,24,ival11,iwidth2,iheight+200,ix+200,iy
gk12,ih12 FLslider "",imin,imax,iexp,26,ival12,iwidth2,iheight+200,ix+250,iy

FLpanelEnd
;*****
;FLSetColor ired, igreen,iblue,ihandle
FLSetColor 255,48,48,ih1
FLSetColor 173,255,7,ih2
FLSetColor 255,140,0,ih3
FLSetColor 189,252,201,ih4
FLSetColor 255,48,48,ih5
FLSetColor 173,255,7,ih6
FLSetColor 255,48,48,ih7
FLSetColor 173,255,7,ih8

```

```

FLsetColor 255,140,0,ih9
FLsetColor 189,252,201,ih10
FLsetColor 255,48,48,ih11
FLsetColor 173,255,7,ih12

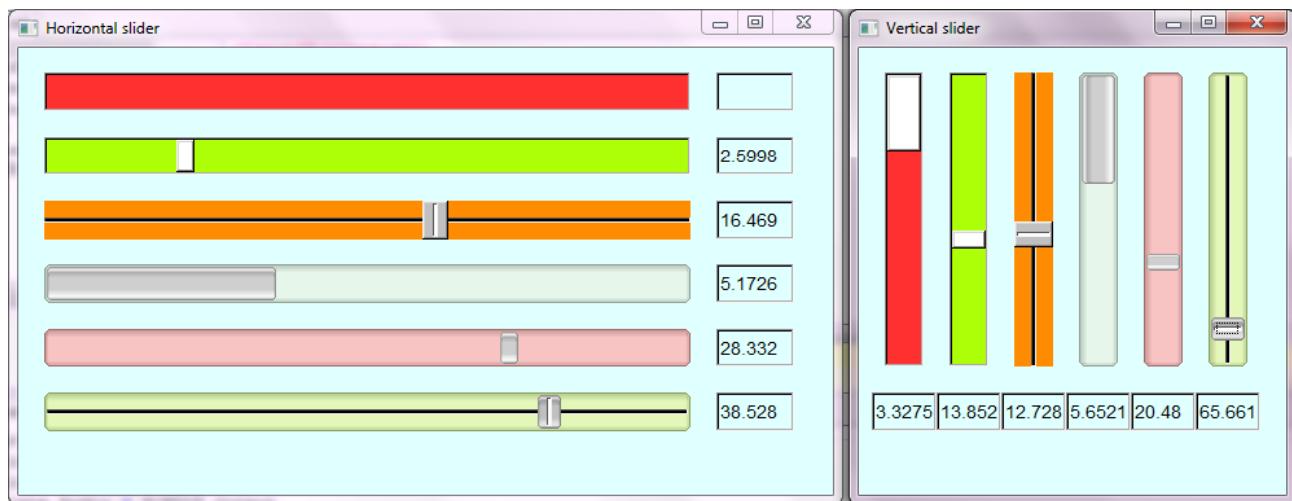
FLrun

instr 1

endin

</CsInstruments>
<CsScore>
f0 3600
</CsScore>
</CsoundSynthesizer>

```



Il prossimo opcode **FLbutton** necessita di una stringa per il parametro label (esempio : "@+8->"), con cui scegliere un simbolo grafico per il bottone (fare riferimento al canonical reference per l'elenco completo) :

Esempio Cap.5.5_FLbutton

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

FLcolor 255,255,255

FLpanel "FLbutton",450,500,400,100,1,-1,-1

ion = 1
ioff = 0
iheight = 50
iwidth = 80
ix = 20
iy = 20
iopcode = -1

;kout,ihandle FLbutton "label",ion,ioff,itype,iwidth,iheight,ix,iy,iopcode
gk1,ih1 FLbutton "@+8->",ion,ioff,1,iwidth,iheight,ix,iy,iopcode
gk2,ih2 FLbutton "@+8>",ion,ioff,1,iwidth,iheight,ix+80,iy,iopcode
gk3,ih3 FLbutton "@+8<<",ion,ioff,1,iwidth,iheight,ix+160,iy,iopcode

```

```

gk4,ih4 FLbutton "@+8>>",ion,ioff,1,iwidth,iheight,ix+240,iy,iopcode
gk5,ih5 FLbutton "@+8<-",ion,ioff,1,iwidth,iheight,ix+320,iy,iopcode
gk6,ih6 FLbutton "@+8square",ion,ioff,1,iwidth,iheight,ix,iy+60,iopcode
gk7,ih7 FLbutton "@+8UpArrow",ion,ioff,1,iwidth,iheight,ix+80,iy+60,iopcode
gk8,ih8 FLbutton "@+8arrow",ion,ioff,1,iwidth,iheight,ix+160,iy+60,iopcode
gk9,ih9 FLbutton "@+8returnarrow",ion,ioff,1,iwidth,iheight,ix+240,iy+60,iopcode
gk10,ih10 FLbutton "@+8circle",ion,ioff,1,iwidth,iheight,ix+320,iy+60,iopcode
gk11,ih11 FLbutton "@+8menu",ion,ioff,1,iwidth,iheight,ix,iy+120,iopcode
gk12,ih12 FLbutton "@+8line",ion,ioff,1,iwidth,iheight,ix+80,iy+120,iopcode
gk13,ih13 FLbutton "@+8+",ion,ioff,1,iwidth,iheight,ix+160,iy+120,iopcode
gk14,ih14 FLbutton "@+8||",ion,ioff,1,iwidth,iheight,ix+240,iy+120,iopcode
gk15,ih15 FLbutton "@+8<->",ion,ioff,1,iwidth,iheight,ix+320,iy+120,iopcode

FLcolor -1 ;disattiva il colore precedente

gk16,ih16 FLbutton "",ion,ioff,1,iwidth+50,iheight-10,ix,iy+220,iopcode
gk17,ih17 FLbutton "",ion,ioff,2,iwidth+50,iheight-10,ix,iy+260,iopcode
gk18,ih18 FLbutton "",ion,ioff,3,iwidth+50,iheight-10,ix,iy+300,iopcode
gk19,ih19 FLbutton "",ion,ioff,4,iwidth+50,iheight-10,ix,iy+340,iopcode
gk20,ih20 FLbutton "",ion,ioff,21,iwidth+50,iheight-10,ix,iy+380,iopcode
gk21,ih21 FLbutton "",ion,ioff,22,iwidth+50,iheight-10,ix,iy+420,iopcode

;kout, ihandle FLbutBank itype, inumx, inumy, iwidth, iheight, ix, iy,iopcode
inumx = 10
inumy = 10
itype = 21 ;(1-4),(21-24 plastic)
gk22,ih22 FLbutBank
itype,inumx,inumy,iwidth+130,iheight+130,ix+200,iy+220,iopcode

FLpanelEnd

FLrun

instr 1

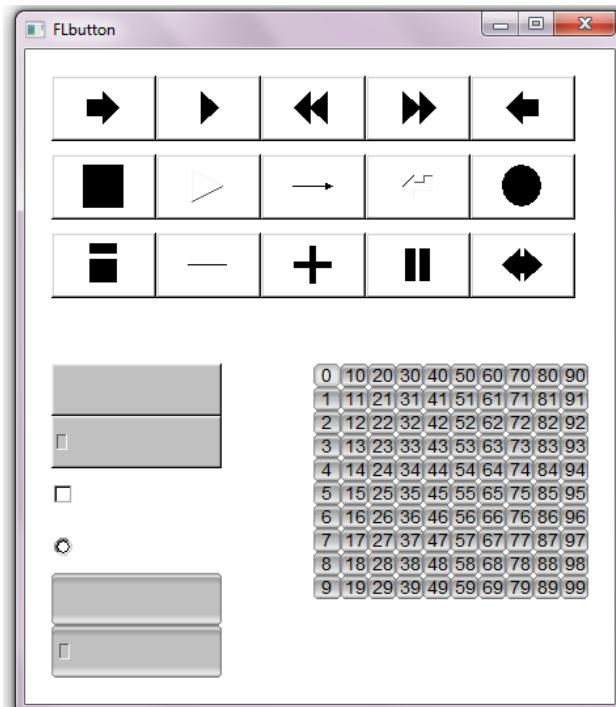
endin

</CsInstruments>
<CsScore>

f0      3600

</CsScore>
</CsoundSynthesizer>

```



Automazioni,controllo midi

Nella costruzione di interfacce per il controllo in tempo reale, diventa assai utile poter automatizzare ad esempio uno slider (immaginiamo il fader che controlla la frequenza di taglio di un filtro), oppure avere la necessità di controllare il movimento di oggetti grafici tramite un controller midi, negli esempi seguenti approfondiremo alcune delle tecniche più utili a questi scopi.

Il prossimo esempio introduce l'opcode **FLjoy**, si tratta di una sorta di joystick molto adatto, tra i vari utilizzi possibili, alla sintesi vettoriale, **FLjoy** utilizza due variabili in ingresso (asse x-y) e in questo esempio vedremo come automatizzare in modo random la traiettoria di un punto nello spazio, oltre ai parametri per definire la posizione nello spazio e la grandezza dell'oggetto troviamo :

kx,ky,ihandlex,ihandley FLjoy "label",iminx,imaxx,iminy,imaxy,iexpx,iexpy,idispx,idispy,iwidth,iheight,ix,iy

iminx,imaxx,iminy,imaxy : valore massimo e minimo per asse x e y;

idispx,idispy : variabili a cui assegnare la visualizzazione dei valori in uscita tramite l'oggetto **FLvalue**

idispx,idispy : valore in uscita lineare (settare a 0) o esponenziale (-1)

Esempio Cap.5.6_FLjoy random move

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

FLpanel "FLjoy",460,500,150,50,1,-1,1
ihvl1 FLvalue "X",60,25,50,420;display per valori di x e y
```

```

ihv2 FLvalue "Y",60,25,120,420

gkx, gky, gihx, gihy FLjoy "",1,360,1,360,-1,-1,ihv1,ihv2,350,300,50,20
gk2,islider FLslider "Rate",.1,20,-1,5,-1,400,30,25,360
gk5,ihandle FLbutton "Auto",1,0,2,120,30,300,420,0,1,0,0

FLsetVal_i 1,islider ;valore iniziale per rate

FLpanelEnd
FLrun

instr 1

if (gk5 == 1)kgoto CONTINUE ;se gk5 = 1 l'istruzione salta a "continue:"
if (gk5 == 0)kgoto STOP ;se gk5 = 0 l'istruzione passa a "endin"

CONTINUE:

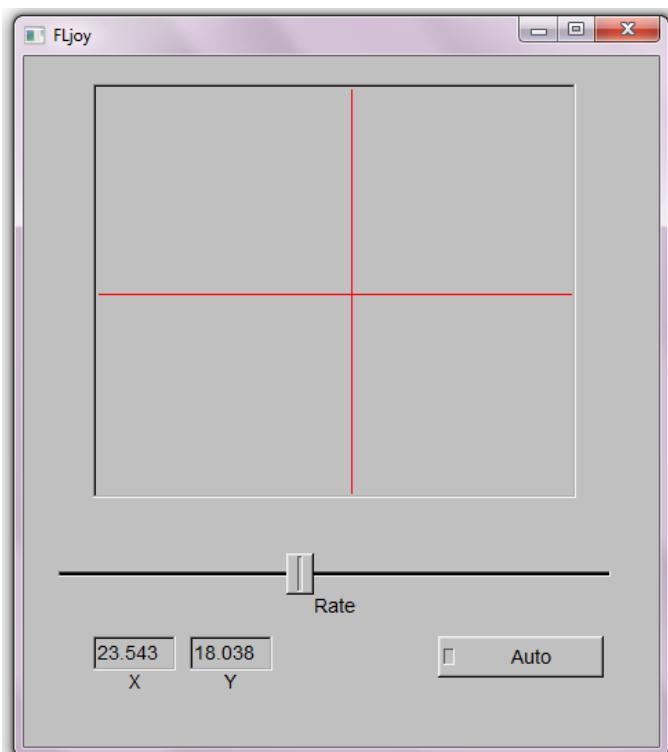
krand randomi 1,360,gk2 ;velocità del movimento di FLjoy
krand2 randomi 1,90,gk2
FLsetVal 1, krand,gihx ;valori random per asse x
FLsetVal 1, krand2,gihy ;valori random per asse y

STOP:

endin

</CsInstruments>
<CsScore>
i1 0 3600
</CsScore>
</CsoundSynthesizer>

```



Possiamo notare nello strumento instr 1 l'utilizzo di una struttura di controllo molto semplice e tipica di importanti linguaggi di programmazione, se "è vera" la condizione dichiarata (in questo caso il valore numerico 1 e 0) il programma salta (goto) nel punto specificato; in questo caso premendo il pulsante

“auto”(gk5) si passa al blocco di programma in cui due funzioni random (per x e y di **FLjoy**) controllano in modo casuale il movimento del punto nello spazio la cui velocità sarà determinata dal valore dato dallo slider “rate”.

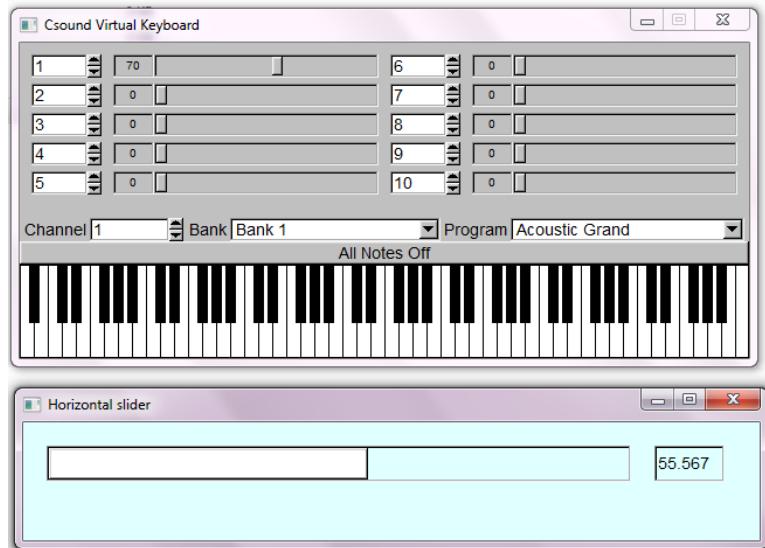
Ora vediamo in che modo sia possibile controllare il movimento di uno slider tramite dati midi, nel prossimo esempio useremo per comodità e chiarezza la Csound virtual keyboard, tuttavia è possibile svolgere lo stesso tipo di controllo tramite un qualunque controller midi hardware.

L’istruzione seguente utilizza **changed** per passare in tempo reale i dati midi del controller k1 allo slider, changed invia messaggi di trigger quando i dati della variabile (in questo caso il controller midi) a cui è associato sono cambiati.

```
k1 ctrl17 1,1,1,100 ;controller midi 1,canale midi 1,range tra 1 e 100  
ktrig changed k1  
FLsetVal ktrig,k1,gih1 ;ktrig invia i dati di k1 allo slider gih1
```

Esempio Cap.5.7_Fltk midi controller

```
<CsoundSynthesizer>  
<CsOptions>  
-+rtmidi=virtual -M0  
</CsOptions>  
<CsInstruments>  
  
FLcolor 224,255,255  
FLpanel "Horizontal slider",630,100,300,300,1,-1,-1  
  
ival1 FLvalue "",60,30,540,20  
imin = 1  
imax = 100  
iexp = 0 ;i valori di tipo lineare  
iheight = 30  
iwidth = 500  
ix = 20  
iy = 20  
gk1,gih1 FLslider "",imin,imax,iexp,1,ival1,iwidth,iheight,ix,iy  
  
FLpanelEnd  
FLrun  
  
instr 1  
  
k1 ctrl17 1,1,1,100 ;controller midi 1,canale midi 1,range tra 1 e 100  
ktrig changed k1  
FLsetVal ktrig,k1,gih1 ;ktrig invia i dati di k1 allo slider gih1  
  
endin  
  
</CsInstruments>  
<CsScore>  
f0 3600  
i1 0 3600  
</CsScore>  
</CsoundSynthesizer>
```



Nella costruzione di gui per Csound diventa importante poter salvare le proprie configurazioni specialmente di fronte a progetti complessi con numerosi slider, knob, il prossimo esempio utilizza un pratico sistema per memorizzare dei preset su un file di testo da cui sarà possibile richiamare o riscrivere i settaggi salvati, i vari snapshots saranno numerati e richiamati tramite un banco di bottoni creato con **FLbutBank**;

per questo tipo di operazione “scrittura dati su file” vengono utilizzati i seguenti moduli :

FLloadsnap e **FLgetsnap** i : permette di accedere tramite un banco di bottoni numerati ad un file di testo con snapshots precedentemente salvati.

FLsavesnap e **FLsetsnap** i: salva la configurazione della gui su file di testo numerando gli snapshots.

Esempio Cap.5.8_FLsavesnap

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

#define PRESETS #"preset.txt" ;file di testo per salvataggio preset
FLcolor 224,255,255
FLpanel "Snapshot",630,110,100,100,1,-1,-1

ival1 FLvalue "",60,30,540,20
gk1,ih1 FLslider "",1,100,-1,1,ival1,500,30,20,20

FLpanelEnd

FLcolor 255,255,255
FLpanel "Snapshot",500,295,750,110,1,-1,-1;start of container

gkget,ih FLbutBank 31,8,8,260,280,230,0,0,1,0,0
FLcolor -1
gksavesnap,ih FLbutton "Save preset",1,0,31,150,40,40,10,0,2,0,0
FLsetFont 2,ih
FLsetTextColor 0,0,255,ih
FLcolor -1

gkstore,ih FLcount "preset number",0,41,1,5,1,150,30,40,80,-1
FLsetFont 4,ih
```

```

FLsetTextColor 0,0,255,ih
FLpanelEnd
FLsetColor 255,48,48,ih1

FLrun

instr 1

FLloadsnap $PRESETS ;file di testo in lettura
inumsnap FLgetsnap i(gkget) ;apre un precedente preset numerato tramite ;il
banco di pulsanti

endin

instr 2

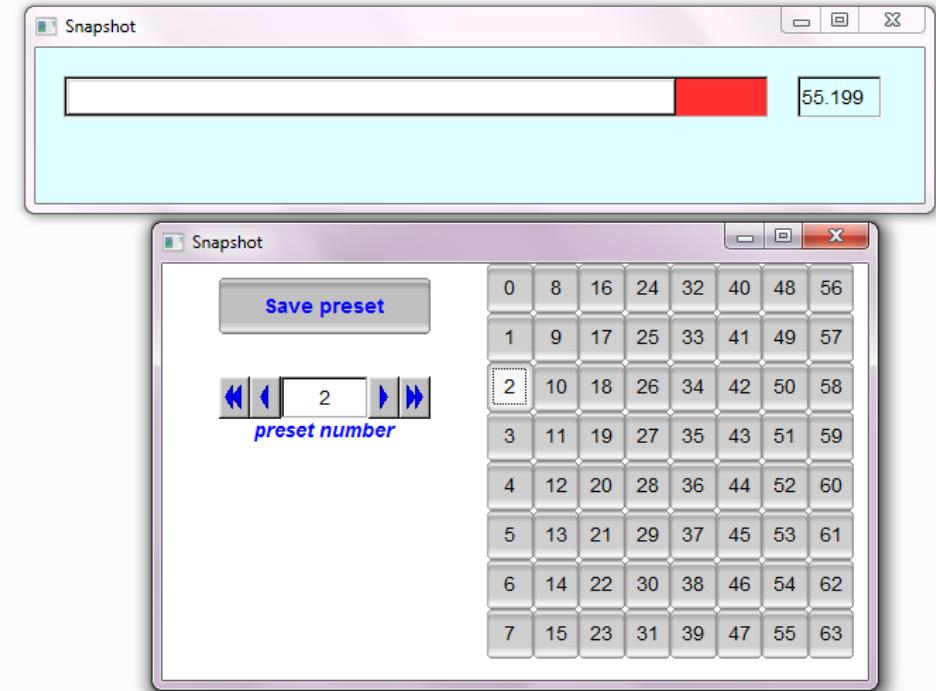
inumsnap,inumval FLsetsnap i(gkstore) ;memorizza un preset numerato ;tramite il
contatore

FLsavesnap $PRESETS

endin

</CsInstruments>
<CsScore>
f0 3600
</CsScore>
</CsoundSynthesizer>

```



Nel pannello costruito per memorizzare i vari preset,selezionare un numero tramite il contatore e poi premere il pulsante Save preset,apparirà una finestra di avviso con la richiesta di riscrivere il file di testo o di annullare,nel nostro esempio il nome del file di testo viene dichiarato all'inizio del file csd :

```
#define PRESETS #"preset.txt"#
```

Se non viene indicata nessuna directory il file verrà scritto nella stessa cartella in cui è contenuto il file .csd, è quindi fondamentale non rimuovere o spostare il file “preset.txt” per poter accedere alle configurazioni salvate o per aggiornare il file con nuovi snapshots.

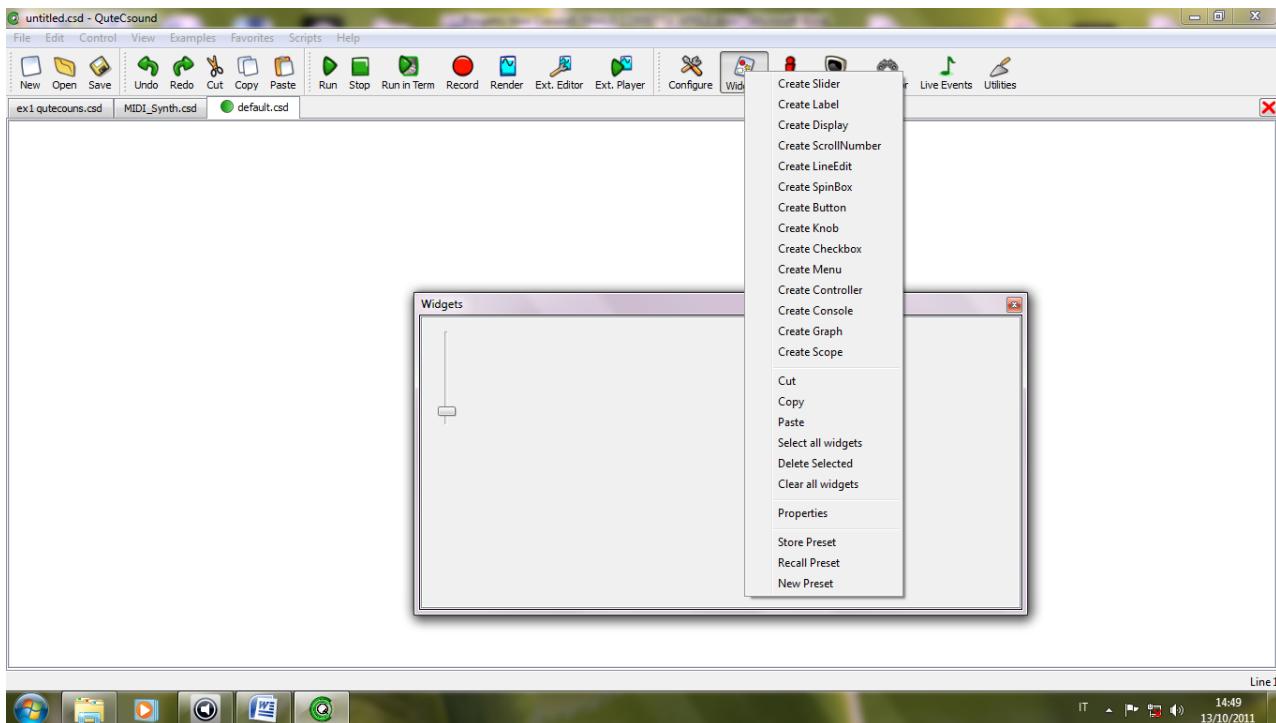
Libreria grafica con QuteCsound

QuteCsound è un ambiente integrato distribuito con il pacchetto d’installazione di Csound, si tratta di un potente editor sviluppato da Andres Cabrera, con le caratteristiche tipiche dei moderni editor per linguaggi di programmazione, come sintassi colorata, strumenti di auto completamento del codice, tools per analisi e risintesi su file audio e molto altro. Una delle sue caratteristiche più innovative e recente è l’integrazione di widgets per poter costruire interfacce grafiche in modo decisamente più semplice e intuitivo rispetto alla libreria Fltk, rispetto a quest’ultima è possibile costruire la propria gui senza scrivere codice all’interno del file .csd, basta selezionare e posizionare i vari oggetti nello spazio e tramite opportuni opcode è possibile inviare i vari dati di controllo ad un file Csound;

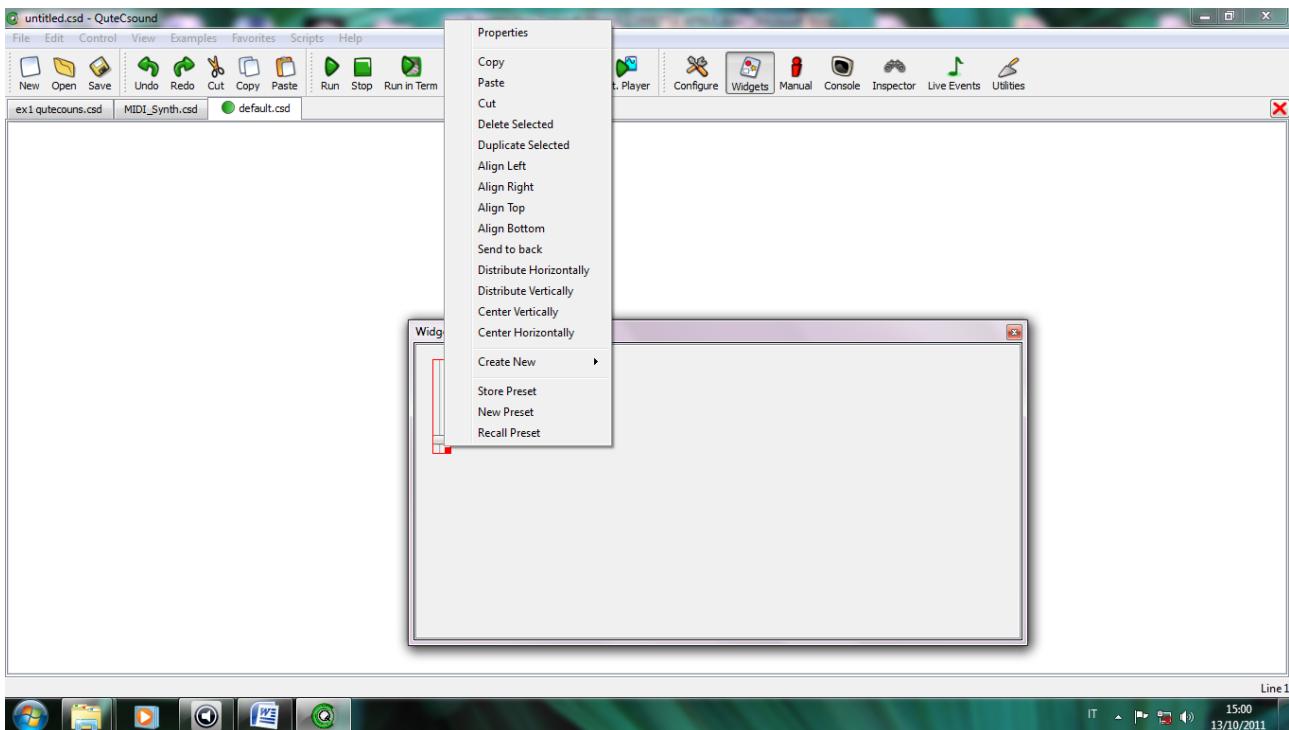
Esistono comunque vari pro e contro valutabili sotto molti punti di vista, bisogna considerare che le gui create con QuteCsound possono essere avviate unicamente tramite quest’editor, aspetto che rende il codice sicuramente meno portabile rispetto alla libreria Fltk (che opera sotto forma di opcodes all’interno di un unico file), i vantaggi di questo sistema sono sicuramente :

- rendering su thread separati;
- maggiore chiarezza nella scrittura di codice Csound
- costruzione dell’interfaccia molto simile a sistemi come Max, Pd

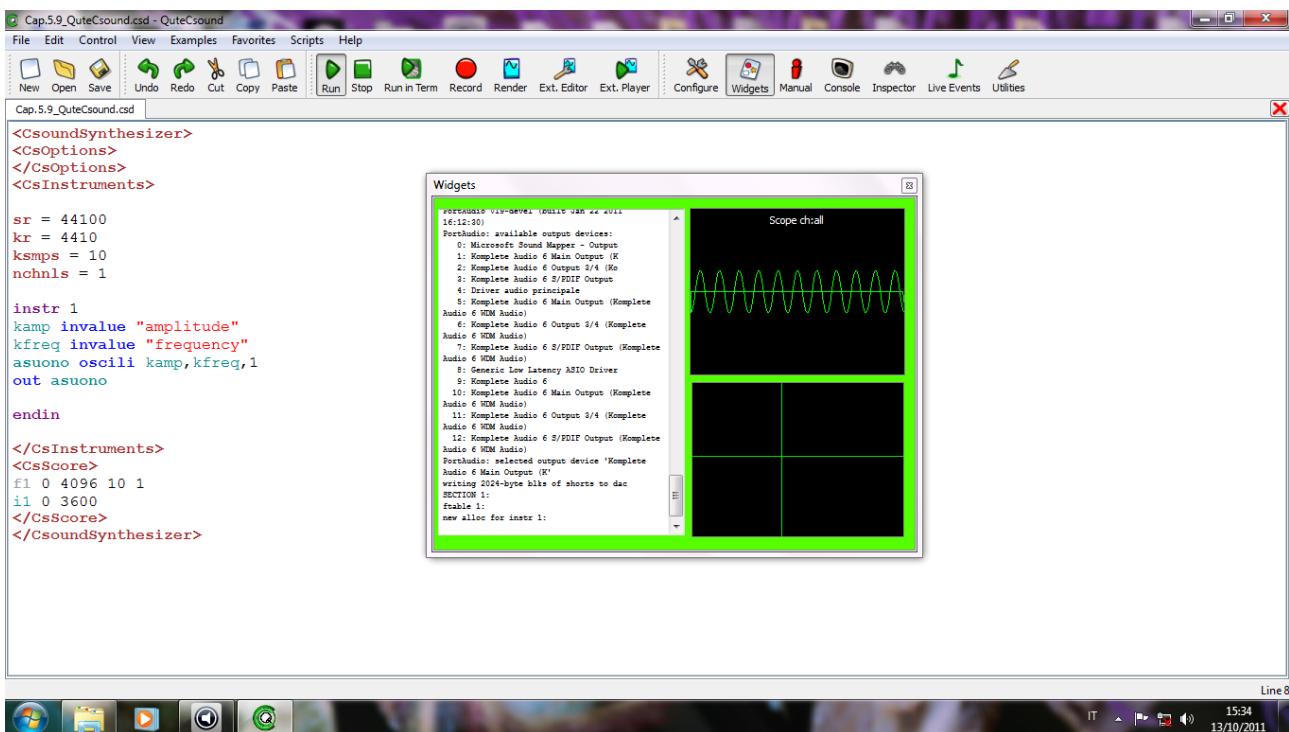
Apriamo adesso l’editor QuteCsound e selezioniamo un nuovo progetto .csd, premendo il pulsante Widgets si aprirà una nuova finestra che conterrà gli oggetti grafici della nostra gui, con un semplice clic destro del mouse si aprirà un menu con l’elenco dei possibili widgets inclusi nell’editor :



Ora in alto scegliamo dal menu Edit la voce Widget Edit Mode, in questo modo entriamo in modalità “modifica widjet” (l’oggetto sarà visualizzato con un contorno rosso) per variare la posizione degli oggetti nello spazio, modificare la dimensione, il colore e molto altro, avvicinando il puntatore del mouse all’oggetto grafico e cliccando sempre con il tasto destro entriamo nel menu proprietà dell’oggetto :



Apriamo adesso il file "Cap.5.9_QuteCsound.csd" con QuteCsound, mettiamo in render il file premendo il pulsante **Run** dal menu in alto dell'editor, se non viene visualizzata nessuna finestra grafica è necessario premere **Widjet** e se occorre ridimensionare la finestra, apparirà la seguente gui introduttiva :



Per prima cosa analizziamo il semplice strumento Csound, introduciamo l'opcode **invalue** con cui Csound comunica con l'interfaccia grafica :

```
kamp inval "amplitude"
kfreq inval "frequency"
```

le stringhe "amplitude" e "frequency" sono i nomi delle variabili che abbiamo associato all'oggetto controller della nostra gui, controller è molto simile all'oggetto **FLjoy** della libreria Fltk e anche in questo

caso svolge la funzione di joystick, osserviamo le proprietà dell'oggetto controller facendo clic con il pulsante destro del mouse, dal menu selezioniamo **properties**:

Horizontal Channel name = amplitude

Vertical Channel name = frequency

In questo modo i valori generati dall'oggetto controller, passano alle variabili kamp e kfreq del generatore audio:

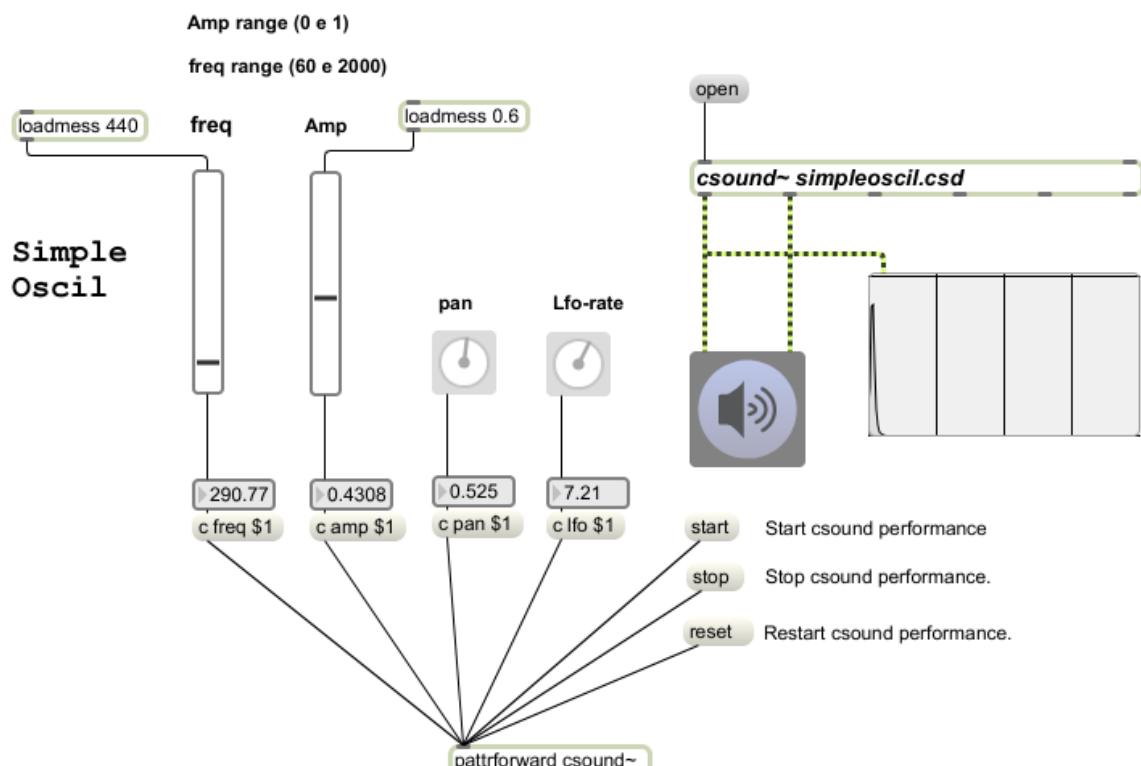
asuono oscili kamp, kfreq, 1

in questa semplice gui introduttiva di QuteCsound troviamo anche due pratici oggetti che non richiedono nessun tipo di programmazione, si tratta degli oggetti Scope e Console, il primo è un pratico oscilloscopio mentre il secondo visualizza la finestra console di Csound durante il rendering, questi due oggetti non sono legati al codice di programma del file .csd, pertanto la creazione e utilizzo è unicamente di tipo grafico.

Csound~ e Max/Msp

L'integrazione tra Csound e Max/Msp rappresenta una delle tecniche più complete e versatili per la costruzione di interfacce grafiche, la comunicazione tra i due linguaggi avviene per mezzo di un Max object chiamato csound~ sviluppato da *Davis Pyon* e disponibile per ambienti Windows e Mac. Questo paragrafo non tratterà nello specifico l'ambiente di programmazione Max/Msp, pertanto si rimanda alla documentazione ufficiale. Per gli utenti che utilizzano PureData esiste una simile external chiamata csoundapi~ sviluppata da *Victor Lazzarini*. Per prima cosa procuriamoci l'external Csound~ dal sito <http://davixology.com/csound~.html>. (percorso Win : C:\Programmi\ Cycling '74\Max\ Cycling '74\ msp-externals; percorso Mac : Applicazioni\Max\ Cycling '74\ msp-externals).

Adesso apriamo l'esempio Cap.5.10_simpleoscil_csoundmax.



l'oggetto csound~ contiene l'argomento “*simpleoscil.csd*” che corrisponde al nome del file csound per il real time.L'oggetto Max utilizza di default un segnale di uscita stereo (ma è possibile aumentare il numero dei canali sia per l'output che per l'input.La comunicazione tra i due linguaggi avviene per mezzo di canali bus definiti con le variabili :

freq,amp,pan,lfo (oggetto *message*,esempio : c freq \$1)

le quattro variabili assumono come valore di input i dati provenienti dagli oggetti *slider* e *dial*,opportunamente settati con il corretto range di valori tramite le proprietà oggetto (*Inspector*).I quattro controller inviano dati in tempo reale all'oggetto csound~ tramite *pattrforward csound~*.

Una volta avviato il motore di Csound tramite il pulsante *Start*,apriamo la finestra Max window di Max/Msp,se tutto è andato a buon fine dovremmo ottenere il seguente output :

Object	Message
csound~	csound~ v1.1.0
csound~	csound simpleoscil.csd -g -+rtmidi=null -M0
csound~	PortMIDI real time MIDI plugin for Csound
csound~	PortAudio real-time audio module for Csound
csound~	virtual_keyboard real time MIDI plugin for Csound
csound~	0dBFS level = 32768.0
csound~	Csound version 5.14 (double samples) Oct 16 2011
csound~	libsndfile-1.0.20
csound~	Reading options from \$CSOUNDRD: C:\Program Files (x86)\Csound\csoundrc
csound~	UnifiedCSD: simpleoscil.csd
csound~	STARTING FILE
csound~	Creating options
csound~	Creating orchestra
csound~	Creating score
csound~	orchname: C:\Users\Giorgio\AppData\Local\Temp\cs8465.orc
csound~	scorename: C:\Users\Giorgio\AppData\Local\Temp\cs8467.sco
csound~	RAWWAVE_PATH: C:\Program Files (x86)\Csound\samples\
csound~	rtaudio: PortAudio module enabled ... using callback interface
csound~	orch compiler:
csound~	instr 1
csound~	Elapsed time at end of orchestra compile: real: 0.451s, CPU: 0.450s
csound~	sorting score ...
csound~	... done
csound~	Elapsed time at end of score sort: real: 0.451s, CPU: 0.451s
csound~	Csound version 5.14 (double samples) Oct 16 2011
csound~	midi channel 1 using instr 1
csound~	midi channel 2 using instr 1

Analizziamo adesso il file di Csound aprendo *simpleoscil.csd* dal capitolo 5 ,la comunicazione avviene per mezzo di canali bus (opcode **chnget**),viene applicato un piccolo portamento ad alcune variabili (**portk**) per evitare alcuni problemi di noise durante il controllo tramite Max.

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

k1    chnget      "amp" ;riceve dati da Max/Msp
k2    chnget      "freq"
k3    chnget      "pan"
k4    chnget      "lfo"

;portamento per eliminare possibili rumori durante l'azione degli slider
kamp  portk k1,.01
kfrq  portk k2,.01
kpan  portk k3,.01

klfo  lfo   12,k4

a1    poscil      kamp,kfrq+klfo,1  ;sine+lfo

al,ar pan2  a1,kpan      ;pan

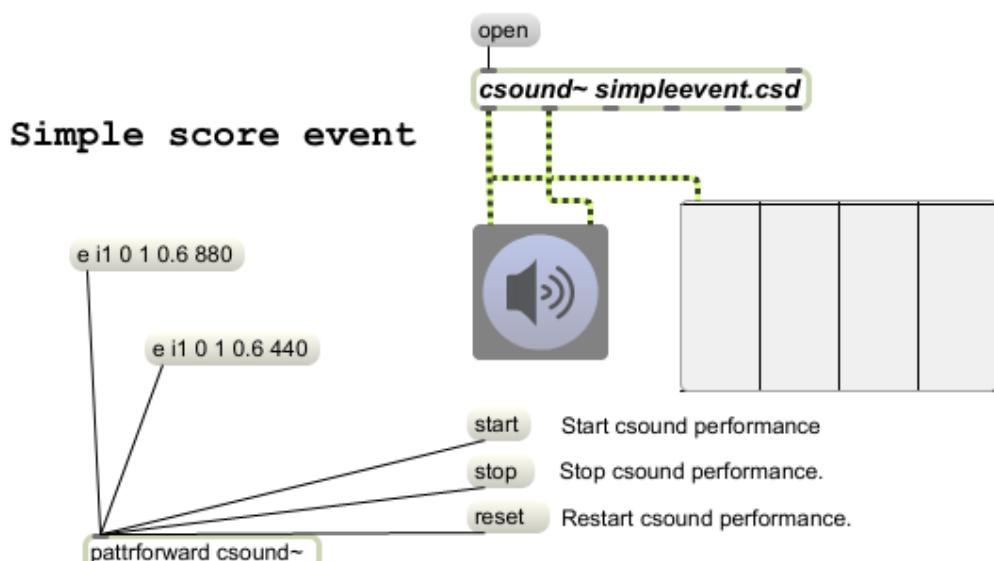
outs  al,ar ;uscita

endin

</CsInstruments>
<CsScore>
f1    0      4096  10      1
i1    0      3600
</CsScore>
</CsoundSynthesizer>

```

Il seguente esempio utilizza Max con funzione di score per Csound, la gestione eventi di questo tipo non utilizza i canali **chnget**. Esempio Cap.5.11_simple_csound_max_score.maxpat



```

"simpleevent.csd"

<CsOptions>
</CsOptions>
</CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
Odbfs = 1

instr 1

a1    poscil      p4,p5,1      ;sine
outs  a1,a1 ;uscita
endin

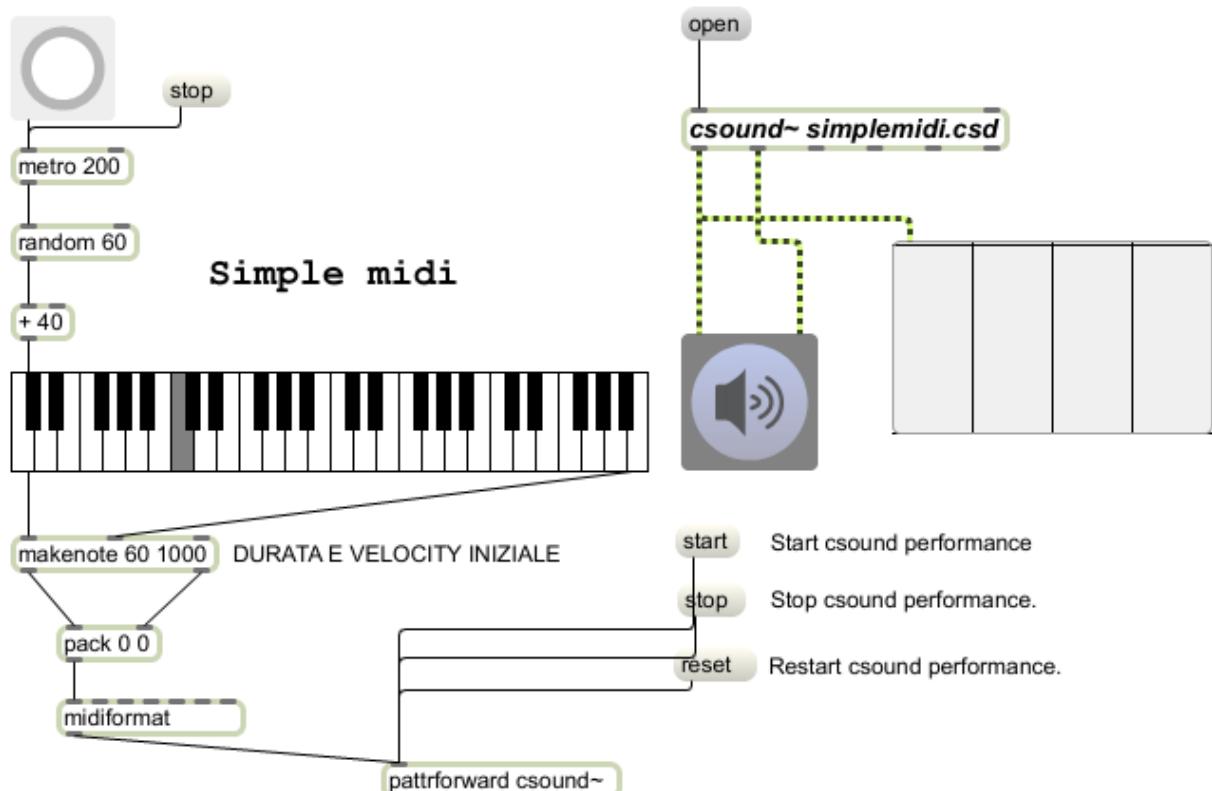
</CsInstruments>
<CsScore>

f1    0      4096  10      1
i1    0      3600

</CsScore>
</CsOptions>

```

L'esempio finale gestisce il controllo di eventi midi, l'oggetto Max *kslider* invia dati di pitch e velocity,l'oggetto *makenote* riceve questi dati e aggiunge il parametro durata per l'evento midi,i tre parametri giungono all'oggetto csound come unico messaggio Midi per mezzo di *midiformat*.



```

"simplemidi.csd"

<CsSoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
Odbfs = 1

massign 1,1 ;assegna instr 1 a canale midi 1

instr 1

iamp ampmidi .6
ifreq cpsmidi
a1 pluck iamp,ifreq,440,0,1
kdeclick linsegr 0,.1,1,.3,.5,.2,0
outs a1*kdeclick,a1*kdeclick

endin

</CsInstruments>
<CsScore>

i1 0 3600

</CsScore>
</CsSoundSynthesizer>

```

Un interessante ed estremamente potente campo di azione, è l'utilizzo dell'ambient Max all'interno del noto sequencer *Ableton live*. Esiste un modulo di connessione tra i due mondi chiamato Maxforlive, si tratta di un tool di oggetti nati espressamente per la programmazione di plugin su ambiente *Ableton*.

Immaginiamo la possibilità di integrare la potenza di sintesi di Csound, con l'incredibile arsenale di oggetti Max, il tutto in un pratico plugin da utilizzare all'interno di *Ableton live*. Per sperimentare in tal senso il primo passo sarà quello di sostituire l'oggetto *dac~* di Max/Msp con il nuovo oggetto *plugout~*.

Partendo da questa semplice idea, citiamo un interessante progetto diretto dal *Dr. Richard Boulanger*, che prende il nome di CsoundforLive (<http://www.csoundforlive.com/>).

Si tratta di un incredibile pacchetto di 120 plugin (commerciali, tuttavia sono anche stati rilasciati alcuni pacchetti free, tra cui un plugin molto pratico per renderizzare file csound all'interno di *Ableton*) dei più disparati tipi, si va dalla sintesi granulare, al processamento spettrale, ai campionatori, ai processori multi effetto, riverberi e molto altro. Questo, per certi versi, rappresenta una sorta di nuovo rinascimento di questo potente linguaggio, forse troppo spesso (e per responsabilità ben precise) legato unicamente al mondo accademico e di ricerca. Con CsoundforLive è possibile avere oggi tutta la potenza di Csound all'interno di un pratico sequencer audio-midi, il tutto in tempo reale, interattivo e con una stabilità e qualità audio senza compromessi.

Cap.6 Sintesi additiva e sottrattiva

6.1 Sintesi Additiva

6.2 Sintesi sottrattiva

Sintesi del suono

La sintesi del suono comprende tutte quelle tecnologie di emulazione per produrre il suono in modo artificiale, il tipo di algoritmo e metodologia impiegata determinerà il timbro di un certo tipo di sintesi. Essendo Csound uno strumento virtuale, il cui utilizzo è basato sulla costruzione di algoritmi informatici, parleremo di sintesi digitale del suono.

Non ci soffermeremo in modo dettagliato sulla storia delle tecniche di sintesi, tuttavia è importante fare una sorta di catalogazione delle tecniche più usate e conosciute ai giorni nostri. Il vantaggio di utilizzare uno strumento virtuale come Csound è proprio quello di non avere limiti dal punto di vista delle possibilità sonore, al contrario dei sintetizzatori hardware (non solo analogici ma anche i modelli virtual analog) che nascono per produrre solo alcuni tipi di sintesi.

Vediamo adesso un possibile elenco delle tecniche di sintesi più conosciute, nei prossimi capitoli vedremo come realizzare questi timbri tramite Csound :

- Sintesi additiva
- Sintesi sottrattiva
- Sintesi per modulazione d'ampiezza (AM)
- Sintesi per modulazione di frequenza (FM)
- Sintesi granulare
- Sintesi per modelli fisici
- Sintesi per distorsione non lineare (Waveshaping)
- Sintesi vettoriale
- Sintesi modulare

Da un punto di vista storico, possiamo ricordare che tra le tecniche più antiche troviamo la sintesi additiva (in breve la costruzione di un timbro complesso mediante la somma di semplici sinusoidi, ricordiamo "Studio 1" e "Studio 2" di K. Stockhausen) e sottrattiva (la costruzione di un timbro mediante filtraggio di uno spettro complesso, un esempio storico è il brano "Notturno" di B. Maderna). Tecniche più recenti sono la sintesi per modelli fisici e la sintesi granulare. Da considerare che una tecnica di sintesi può dividersi ulteriormente in sottocategorie molto specifiche, per esempio nel caso della sintesi per modelli fisici troviamo applicazioni nella costruzione di risuonatori e nella modellazione virtuale di strumenti acustici o della voce.

Sintesi Additiva

La sintesi additiva è uno dei metodi di sintesi più importanti dal punto di vista didattico, in genere una delle prime con cui uno studente di sound design si approccia. Questo tipo di sintesi è basata sull'applicazione del teorema di Fourier, secondo cui un suono complesso può essere scomposto in una somma di suoni elementari con determinata ampiezza e frequenza, di tipo sinusoidale o cosinusoidale. Le prime tecniche di sintesi additiva impiegavano naturalmente l'utilizzo di oscillatori analogici, il limite era proprio quello di essere limitati nell'esplorazione sonora proprio dal numero degli oscillatori in dotazione nei centri di ricerca. Con l'utilizzo di un linguaggio per la sintesi del suono la creazione di spettri complessi mediante questa tecnica è certamente più semplice, tuttavia rimane una tecnica di sintesi dalla costruzione assai lenta e preferibilmente adatta al tempo differito, proprio per l'enorme mole di dati di cui ha bisogno. Uno dei problemi più grandi è quello di dover indicare per ogni componente armonica data, la sua frequenza, ampiezza e inviluppo, e per certi timbri complessi (come quelli di emulazione) il numero di sinusoidi impiegate è estremamente alto. Sono questi alcuni dei motivi per cui questo tipo di sintesi è stata nel tempo sostituita da altre tecniche (come la modulazione di frequenza).

Tuttavia grazie alla potenza delle moderne CPU, e all'utilizzo creativo di linguaggi per la sintesi come Csound, la sintesi additiva (oltre a rimanere un eccellente primo approccio didattico alla sintesi del suono) può essere ancora un campo da esplorare per la ricerca di timbri complessi e variabili nel tempo.

Una delle tecniche più comuni e semplici da realizzare è la sintesi additiva a spettro fisso con sinusoidi in rapporto armonico o inarmonico. Si tratta di costruire un banco di otto oscillatori sinusoidali con una frequenza base, lasciando inalterata la frequenza del primo oscillatore (in questo la fondamentale) moltiplichiamo la frequenza delle altre istanze di **oscil** (oscillatore con interpolazione identica ad oscili ma

ad alta precisione) per dei rapporti numerici (le parziali) ottenendo le armoniche superiori, gli otto oscillatori vengono infine sommati in un'unica variabile per l'uscita audio.

Esempio Cap.6.1_simple_additive

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

iamp = p4      ;ampiezza
ibasefreq = p5    ;frequenza base

;otto oscillatori
a1  poscil  iamp*2,ibasefreq,1      ;frequenza fondamentale
a2  poscil  iamp*.9,ibasefreq*1.2,1
a3  poscil  iamp*.8,ibasefreq*1.4,1
a4  poscil  iamp*.7,ibasefreq*1.6,1
a5  poscil  iamp*.6,ibasefreq*1.8,1
a6  poscil  iamp*.5,ibasefreq*2,1
a7  poscil  iamp*.4,ibasefreq*2.2,1
a8  poscil  iamp*.3,ibasefreq*2.4,1

aout sum   a1,a2,a3,a4,a5,a6,a7,a8 ;somma degli oscillatori
aout = aout/8
kenv mxadsr     1,.5,.8,.8

outs aout*kenv,aout*kenv ;uscita con inviluppo

endin

</CsInstruments>
<CsScore>

f1    0      16384 10      1
i1    0      10      .6      220

</CsScore>
</CsoundSynthesizer>
```

Adesso una variante di questo modello in cui le parziali e i valori di ampiezza di ogni parziale, vengono definiti in una funzione (opcode **table**), lo scopo è quello di ottenere un modello più flessibile in cui basta sostituire i valori della funzione per ottenere nuovi spettri armonici.

Esempio Cap.6.2_simple_additive_table

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

iamp = p4      ;ampiezza
ibasefreq = p5    ;frequenza base
iamptab = p6     ;tabella ampiezze
ipartial = p7    ;tabella parziali

;tabella delle parziali
ifreqtable1 table 1,ipartial
ifreqtable2 table 2,ipartial
ifreqtable3 table 3,ipartial
ifreqtable4 table 4,ipartial
ifreqtable5 table 5,ipartial
ifreqtable6 table 6,ipartial
ifreqtable7 table 7,ipartial

;tabella ampiezze
iamptable1 table 1,iamptab
iamptable2 table 2,iamptab
iamptable3 table 3,iamptab
iamptable4 table 4,iamptab
iamptable5 table 5,iamptab
iamptable6 table 6,iamptab
iamptable7 table 7,iamptab

;otto oscillatori
a1  poscil   iamp*2,ibasefreq,1      ;frequenza fondamentale
a2  poscil   iamp*iamptable1,ibasefreq*ifreqtable1,1
a3  poscil   iamp*iamptable2,ibasefreq*ifreqtable2,1
a4  poscil   iamp*iamptable3,ibasefreq*ifreqtable3,1
a5  poscil   iamp*iamptable4,ibasefreq*ifreqtable4,1
a6  poscil   iamp*iamptable5,ibasefreq*ifreqtable5,1
a7  poscil   iamp*iamptable6,ibasefreq*ifreqtable6,1
a8  poscil   iamp*iamptable7,ibasefreq*ifreqtable7,1

aout sum   a1,a2,a3,a4,a5,a6,a7,a8 ;somma degli oscillatori
aout =   aout/8
kenv mxadsr      1,.5,.8,.2

outs aout*kenv,aout*kenv ;uscita con inviluppo

endin

</CsInstruments>
<CsScore>

f1  0      16384 10      1
f3  0      -7      -2      1      .9      .8      .7      .6      .5      .4      .3      ;ampiezze

f2  0      -7      -2      1      1.1      1.2      1.3      1.4      1.5      1.6      1.7      ;parziali
f4  0      -7      -2      1      1      2      3      4      5      6      7
f5  0      -7      -2      1      1.2      1.4      1.6      1.8      2      2.4      2.6
f6  0      -7      -2      1      2      4      5      7      8      10      11
f7  0      -7      -2      1      1.3      1.5      1.7      1.9      2.11      3.13      4.15

```

```

i1    0     4     .5    220   3     2
s
i1    0     4     .5    220   3     4
s
i1    0     4     .5    220   3     5
s
i1    0     4     .5    220   3     6
s
i1    0     4     .5    220   3     7
s

</CsScore>
</CsoundSynthesizer>
```

Una variante creativa dell'esempio precedente può essere quella di realizzare veri e propri arpeggi di armonici per ogni oscillatore, per far questo utilizziamo l'opcode **randomh** per leggere a determinate velocità l'indice di ogni tabella, il risultato è uno spettro dinamico in continuo movimento.

Esempio Cap.6.2_simple_additive_table_B

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

iamp = p4
ibasefreq = p5      ; frequenza base
iamptab = p6        ; tabella ampiezze
ipartial = p7        ; tabella parziali

k1    randomh    1,7,4 ; indice della tabella letto in modo casuale
k2    randomh    1,7,5 ; simile ad un arpeggio di armonici con rate indipendente
k3    randomh    1,7,6
k4    randomh    1,7,1
k5    randomh    1,7,.5
k6    randomh    1,7,.6
k7    randomh    1,7,.7

; tabella delle parziali
kfreqtable1 table k1,ipartial
kfreqtable2 table k2,ipartial
kfreqtable3 table k3,ipartial
kfreqtable4 table k4,ipartial
kfreqtable5 table k5,ipartial
kfreqtable6 table k6,ipartial
kfreqtable7 table k7,ipartial

; tabella ampiezze
iamptable1 table 1,iamptab
iamptable2 table 2,iamptab
iamptable3 table 3,iamptab
iamptable4 table 4,iamptab
iamptable5 table 5,iamptab
```

```

iamptable6  table 6,iamptab
iamptable7  table 7,iamptab

;otto oscillatori
a1  poscil    iamph*2,ibasefreq,1      ;frequenza fondamentale
a2  poscil    iamph*iamptable1,ibasefreq*kfreqtable1,1
a3  poscil    iamph*iamptable2,ibasefreq*kfreqtable2,1
a4  poscil    iamph*iamptable3,ibasefreq*kfreqtable3,1
a5  poscil    iamph*iamptable4,ibasefreq*kfreqtable4,1
a6  poscil    iamph*iamptable5,ibasefreq*kfreqtable5,1
a7  poscil    iamph*iamptable6,ibasefreq*kfreqtable6,1
a8  poscil    iamph*iamptable7,ibasefreq*kfreqtable7,1

aout  sum    a1,a2,a3,a4,a5,a6,a7,a8 ;somma degli oscillatori
aout =      aout/8
kenv  mxadsr     1,.5,.8,.2

outs  aout*kenv,aout*kenv ;uscita con inviluppo

endin

</CsInstruments>
<CsScore>

f1    0      16384 10      1
f3    0      -7      -2      .9      .8      .7      .6      .5      .4      .3      .2      ;ampiezze
parziali

f2    0      -7      -2      1      1.1      1.2      1.3      1.4      1.5      1.6      1.7      ;parziali
f4    0      -7      -2      1      2      3      4      5      6      7      8
f5    0      -7      -2      1      1.2      1.4      1.6      1.8      2      2.4      2.6
f6    0      -7      -2      1      2      4      5      7      8      10      11
f7    0      -7      -2      1      1.3      1.5      1.7      1.9      2.11      3.13      4.15

i1    0      10      .6      440      3      2
s
i1    0      10      .6      440      3      4
s
i1    0      10      .6      440      3      5
s
i1    0      10      .6      440      3      6
s
i1    0      10      .6      440      3      7
s

</CsScore>
</CsoundSynthesizer>

```

In questo esempio abbiamo costruito un modello di sintesi additiva utilizzando un'unità base come l'opcode **poscil**, lo spettro complesso è stato generato dalla somma di tante sinusoidi, tuttavia esistono in Csound altre unità di generazione sonora in grado di generare la sintesi additiva in modo più semplice, uno di questi è l'oscillatore **buzz**, in grado di generare una serie di armoniche sinusoidali in rapporto armonico tra di loro. La sua sintassi è simile ad opcode come **oscil**, **poscil**, **oscili** ma aggiunge un parametro relativo al numero delle armoniche da ottenere.

Esempio Cap.6.4_Additive synthesis (buzz)

```

<CsoundSynthesizer>
<CsOptions>
```

```

</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kamp = p4
kfreq = p5
knh randomh 1,20,5 ;20 armoniche controllate in modo random
ifn = 1
a1 buzz kamp,kfreq,knh,ifn
aenv linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0
out a1*aenv

endin

</CsInstruments>
<CsScore>

f1 0 16384 10 1
i1 0 10 .6 220

</CsScore>
</CsoundSynthesizer>

```

Uno degli approcci moderni per la sintesi additiva, è quello di realizzare delle strutture globali per controllare il numero di armoniche da ottenere, il vantaggio nel lavorare con macro-strutture è quello di rendere la scrittura del codice più essenziale, per poter intervenire con pochissimi interventi sulla generazione sonora. Immaginiamo uno strumento Csound (un vero e proprio opcode personalizzato) in grado di controllare un secondo strumento (un oscillatore sinusoidale), con cui leggere le armoniche da una funzione (simile ad uno dei primi esempi con **table**), in cui il controllo per ogni armonica generata avviene in modo automatico, con uno schema del tipo :

strumento 1 :

- genera uno spettro di 22 armoniche per lo strumento 2

strumento 2 :

```

tabella table indice,numero tabella ( con 22 armoniche)
asuono oscili iamp,frequenzabase*tabella

```

uscitaasuono

In pratica, lo strumento “1” controlla in modo automatico l’indice della tabella contenente le parziali, generando 22 istanze di “asuono oscili iamp, frequenzabase*tabella”, ogni oscillatore avrà un indice di tabella in senso numerico progressivo, per esempio una tabella con cinque parziali :

```
f2 0 -5 -2 1 2 3 4 5
```

per realizzare questo processo di generazione automatica (una sorta di loop con numero di iterazioni definito) ho costruito un semplice opcode grazie alla tecnica dell’incapsulamento (molto simile al concetto di sub patch in Max/Msp), in cui uno strumento complesso viene racchiuso in un opcode, questa tecnica prende il nome di *user defined opcode* (UDO) e verrà ampiamente discussa nei capitoli finali.

Osserviamo la sintassi di **Additive** in cui non vi è nessun argomento in ingresso :

```
Additive    iamp,ifreq,instrument,ipartials,itab
```

- iamp : ampiezza globale (relativa a p4) dello strumento da controllare
- ifreq : somma di frequenze (p5),in base al numero di istanze da generare
- instrument : numero dello strumento da controllare (instr2,instr3,ecc)
- ipartials : numero di armoniche da generare
- itab : tabella con le armoniche

ispirandomi al lavoro del ricercatore *Iain McCurdy*, ho iniziato ad esplorare le possibilità timbriche generate dall'utilizzo di frequenze modali di materiali utilizzandole come tabelle di parziali per la sintesi additiva, un elenco completo delle frequenze modali si trova nel canonical reference alla voce “*modal frequency ratios*”.

Esempio Cap.6.5_Additive_opcode

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

gismall_handbell ftgen 1,0,-22,-2,1, 1.0019054878049, 1.7936737804878,
1.8009908536585, \
2.5201981707317, 2.5224085365854, 2.9907012195122, \
2.9940548780488, 3.7855182926829, 3.8061737804878, 4.5689024390244,
4.5754573170732, \
5.0296493902439, 5.0455030487805, 6.0759908536585, 5.9094512195122,
6.4124237804878, \
6.4430640243902, 7.0826219512195, 7.0923780487805, 7.3188262195122,
7.5551829268293

#include    "Additive.h"

;Additive    iamp,ifreq,instrument,ipartials,itab

instr 1      ;genera parziali per strumento 2

itab =      p7      ;tabella con le parziali
ipartials =   p6      ;numero di armoniche da generare
instrument =  2       ;controlla instr 2

Additive    p4,p5,instrument,ipartials,itab

endin

instr 2      ;generatore di sintesi additiva

itable      table p6,p7 ;tabella con parziali
a1    poscil      p4,p5*itable,100 ;instr 1 genera un banco di oscillatori
kenv  linseg      0,0.02,1,p3-0.05,1,0.02,0,0.01,0 ;inviluppo declick
outs  a1*kenv,a1*kenv
endin
```

```

</CsInstruments>
<CsScore>

f100 0 16384 10 1

;p4 = amp
;p5 = frequenza base
;p6 = numero di armoniche da generare
;p7 = tabella con le parziali

i1 0 4 .5 120 22 1
s

</CsScore>
</CsoundSynthesizer>
```

Come notiamo si ha una grande flessibilità nel riutilizzo del codice per altri spettri in additiva,abbiamo realizzato uno strumento formato da 22 sinusoidi con rapporti armonici basati sul modello di *small_handbell*.Senza l'utilizzo di un processo generativo come **Additive**,l'unico modo per ottenere lo stesso timbro è quello di riscrivere lo strumento con 22 istanze di oscillatori (a1 oscili,a2 oscili,a3 oscili....ecc..) come nei primi semplici esempi.

Vediamo adesso lo stesso concetto applicato ad uno strumento per il controllo midi in tempo reale della frequenza base,l'oggetto grafico **FLcount** selezionerà la tabella partendo da un database di 12 tabelle contenute nel file *partials.h*.

Altri parametri globali riguardano l'inviluppo di tipo classico Adsr ed una sorta di effetto tremolo applicato all'ampiezza globale dello spettro generato,lo strumento include anche un riverbero (**reverb**) e la possibilità di estendere la durata di una nota dopo il rilascio.

Provate a personalizzare questo strumento aggiungendo altre tabelle con rapporti armonici (magari con altri modal ratios di materiali?o con la serie di Fibonacci?....)

Esempio Cap.6.6 _Additive_midi_opcode

```

<CsoundSynthesizer>
<CsOptions>
-+rtmidi=virtual -M1
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2
0dbfs = 1

gasig init 0 ;variabile globale riverbero

#include "partials.h" ;file con tabelle delle parziali

FLpanel "Additive synthesis",600,500,300,100,4,-1,1
FLvkeybd "keyboard.map",560,90,20,400

gk1,ihb1 FLbutton "On/Off",0,1,2,100,30,20,20,-1
gk3,ihb4 FLbutton "Exit",1,0,21,100,30,20,100,0,3,0,0

;flbox display per FLslider
ihval1 FLvalue "",60,30,520,20
ihval2 FLvalue "",60,30,520,70
ihval3 FLvalue "",60,30,520,120
ihval4 FLvalue "",60,30,520,120+60
```

```

ihval5      FLvalue "",60,30,520,120+110
ihval6      FLvalue "",60,30,520,120+160
ihval7      FLvalue "",60,30,520,120+210

;kout,ihandle FLslider "",imin,imax,iexp,itype,idisp,iwidth,iheight,ix,iy

gkamp,gih1  FLslider   "amplitude",0,.3,0,25,ihval1,355,30,143,20
gksustain,gih2  FLslider   "extra-release",0,4,0,25,ihval2,355,30,143,70
gkampdev,gih3  FLslider   "amp deviation",.1,5,0,25,ihval3,355,30,143,120
gkrev,girev FLknob    "reverb",0,1,0,1,-1,60,40,150
gkmodal,gihmodal FLcount   "partials",1,12,1,12,1,110,30,15,252,-1
gkatt,gih4  FLslider   "attack",.001,1,0,5,ihval4,355,30, 143,180
gkdec,gih5  FLslider   "decay",.01,1,0,5,ihval5,355,30, 143,230
gksus,gih6  FLslider   "sustain",.01,1,0,5,ihval6,355,30, 143,280
gkrel,gih7  FLslider   "release",.01,1,0,5,ihval7,355,30, 143,330

FLpanel_end
FLrun
;valori iniziali
FLsetVal_i .2,gih1      ;init value amp
FLsetVal_i 1,gih2      ;init value dur
FLsetVal_i 2,gih3
FLsetVal_i 0.001,gih4  ;init value a
FLsetVal_i 0.3,gih5  ;init value d
FLsetVal_i 0.3,gih6  ;init value s
FLsetVal_i .5,gih7  ;init value r
FLsetVal_i 1,gihmodal

#include "additive.h"      ;opcode per sintesi additiva

instr 1      ;strumento di controllo per instr 2

idur = i(gksustain)      ;durata dell'evento
instrument = 2      ;instr 2

;gkmodal sceglie la tabella con le parziali, se la condizione è "vera"
;vengono generate le parziali basate sulla tabella scelta, ampiezza globale

if i(gkmodal)=1      then
ipartial = 6      ;numero di parziali da generare
Additive_midi p4,p5,instrument,ipartial,idur      ;controlla instr 2
elseif i(gkmodal)=2      then
ipartial = 6
Additive_midi p4,p5,instrument,ipartial,idur
elseif i(gkmodal)=3      then
ipartial = 6
Additive_midi p4,p5,instrument,ipartial,idur
elseif i(gkmodal)=4      then
ipartial = 6
Additive_midi p4,p5,instrument,ipartial,idur
elseif i(gkmodal)=5      then
ipartial = 6
Additive_midi p4,p5,instrument,ipartial,idur
elseif i(gkmodal)=6      then
ipartial = 6
Additive_midi p4,p5,instrument,ipartial,idur
elseif i(gkmodal)=7      then
ipartial = 22
Additive_midi p4,p5,instrument,ipartial,idur
elseif i(gkmodal)=8      then
ipartial = 5
Additive_midi p4,p5,instrument,ipartial,idur
elseif i(gkmodal)=9      then

```

```

ipartials = 7
Additive_midi p4,p5,instrument,ipartials,idur
elseif i(gkmodal)=10 then
ipartials = 24
Additive_midi p4,p5,instrument,ipartials,idur
elseif i(gkmodal)=11 then
ipartials = 9
Additive_midi p4,p5,instrument,ipartials,idur
elseif i(gkmodal)=12 then
ipartials = 20
Additive_midi p4,p5,instrument,ipartials,idur
endif

endin

instr 2

if gk1 = 1 then ;On/Off strumento
turnoff
endif

itable table p6,i(gkmodal) ;tabella con parziali
kamp randomi 5-gkampdev,gkampdev,5 ;variazioni random ampiezza
a1 oscili p4*kamp,p5*itable,100 ;oscillatore sinusoidale
kenv madsr i(gkatt),i(gkdec),i(gksus),i(gkrel),0,i(gksus) ;inviluppo

outs (a1*kenv)*gkamp,(a1*kenv)*gkamp

vincr gasig,(a1*kenv)*gkamp ;uscita globale di a1 per il riverbero
endin

instr 3
exitnow
endin

instr 4 ;riverbero

aL,aR reverbsc gasig,gasig,gkrev,12000,sr,0.5,1

outs aL,aR

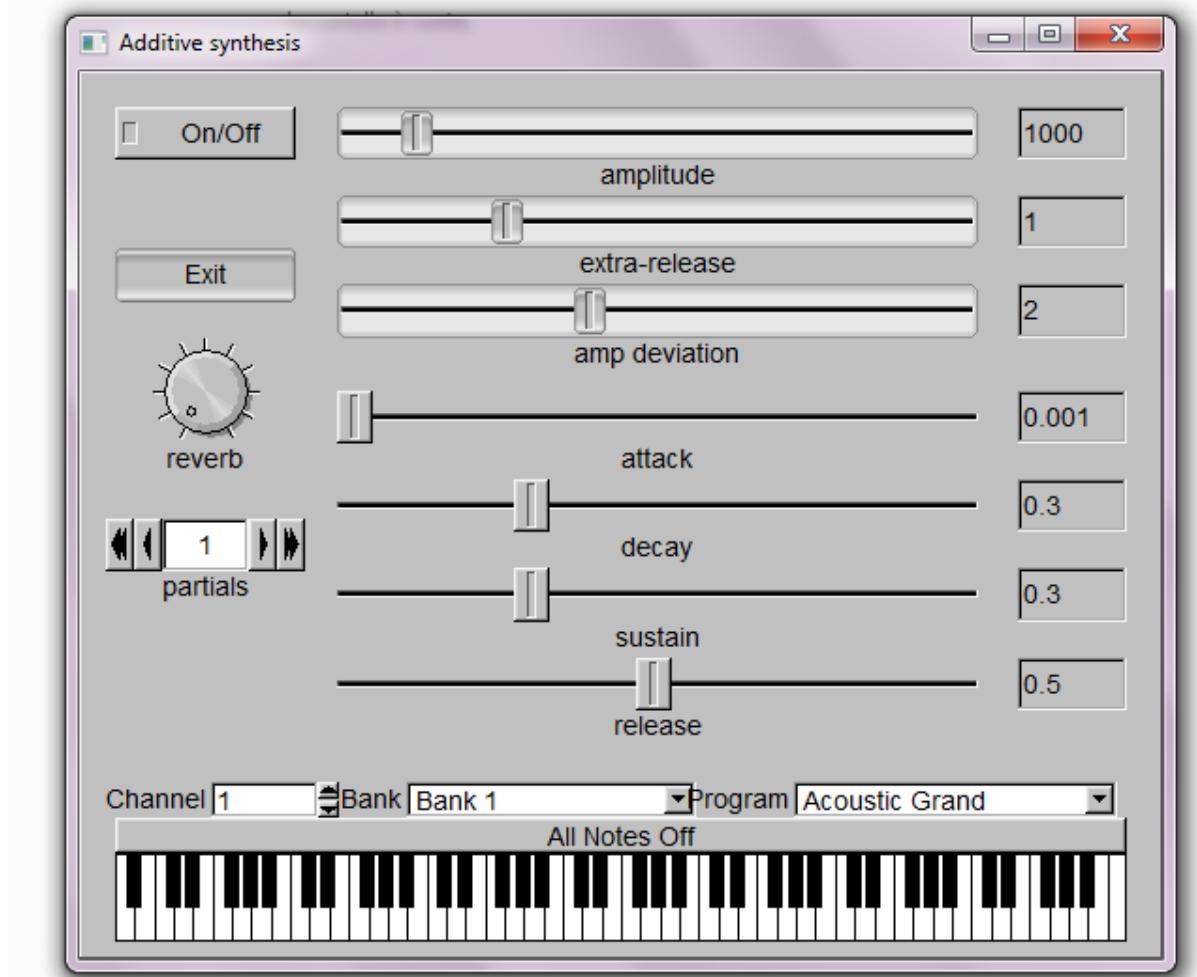
clear gasig ;azzera la variabile gasig per evitare accumulo
endin

</CsInstruments>
<CsScore>

f100 0 1024 10 1
f0 3600
i4 0 3600

</CsScore>
</CsoundSynthesizer>

```



Sintesi sottrattiva (panoramica dei filtri)

In questa sezione verranno introdotti i filtri di Csound con relative applicazioni, la sintesi sottrattiva è un metodo in cui vengono attenuate delle frequenze da uno spettro armonico complesso, anche questa è una delle tecniche di sintesi più antiche e ancora oggi molto usata (specialmente nei sintetizzatori commerciali sia hardware che software). I vari tipi di filtri vengono classificati principalmente in :

- filtro passa basso : permette di ascoltare unicamente le frequenze al di sotto di un valore detto "frequenza di taglio";
- filtro passa alto : attenua le frequenze al di sotto di una frequenza di taglio data
- filtro passa banda : vengono fatte passare solo le frequenze definite in una zona delimitata da due valori (larghezza di banda), attenuando le altre frequenze esterne.
- filtro risonante : serve ad enfatizzare una certa frequenza definita
- filtro notch : viene utilizzato per eliminare il passaggio di frequenze da una larghezza di banda definita.

Esistono molti altri tipi di filtro con caratteristiche più specifiche e quindi meno versatili, in genere usati per la costruzione di processori effetti come i filtri allpass, comb, filtro per formanti (fofilter) e altri.

Uno degli esempi storici di sintesi sottrattiva è il filtraggio di rumore bianco, nel prossimo esempio introduciamo l'opcode **tonex** :

```
ares tonex  asig,khp [,inumlayer]
```

questo opcode realizza un banco di filtri passa-basso collegati in serie (significa che l'uscita del primo filtro entra nel filtro successivo e così via), il funzionamento in Csound è simile a quello di altri filtri (o processori di segnale), una volta indicato il segnale in ingresso abbiamo altri due parametri :

khp : frequenza di taglio del filtro

inumlayer (opzionale, un solo filtro di default) : il numero dei filtri passa-basso in serie.

Esempio Cap.6.7_Lowpass filter(nogui)

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kfreq randomi    100,20000,1
ilayer      =      p4      ; filtri in serie
aout  rand .6    ; rumore bianco
afilt tonex aout,kfreq,ilayer ; banco di filtri
out   afilt

endin

</CsInstruments>
<CsScore>
```

```

i1      0      10      1
s
i1      0      10      10
s
</CsScore>
</CsoundSynthesizer>
```

Esempio Cap.6.7_Lowpass filter

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

FLcolor 255,255,255
FLpanel "tonex bank filter",580,140,300,300,4,-1,1

gk1,ihb1      FLbutton      "Play",1,0,21,100,30,20,20,0,1,0,-1
gk2,ihb2      FLbutton      "Stop",1,0,21,100,30,20,60,0,1,0,0
gk4,ihb4      FLbutton      "Exit",1,0, 21,100,30,20,100,0,2,0,0
ifreq1  FLvalue "",50,38,515,25
gkfreq,ih1    FLslider      "Freq",100,20000,0,5,ifreq1,355,30,143,20
gklayer,ih2   FLcount "Layer stack",1,20,1,1,2,200,30,220,80,-1

FLpanel_end
FLrun
;valori iniziali
FLsetVal_i    2000,ih1      ;freq filter
FLsetVal_i    4,ih2      ;layer filter
;rumore bianco filtrato da low pass filter
instr 1

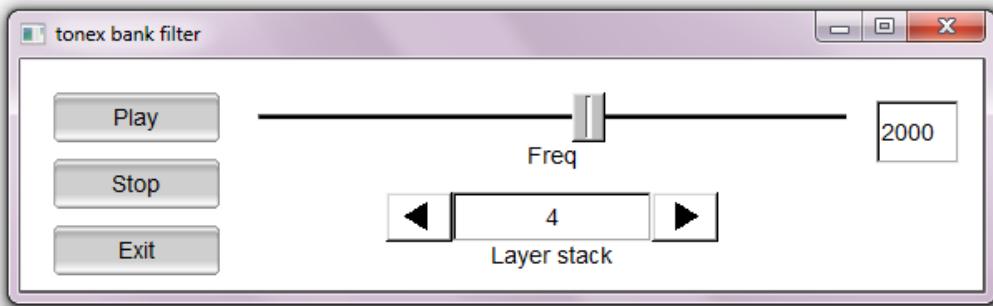
aout  rand  .6      ;rumore bianco
afilt  tonex  aout,gkfreq,i(gklayer) ;banco di filtri
out    afilt

endin

;exit
instr 2
exitnow
endin
</CsInstruments>
</CsScore>

f0      3600

</CsScore>
</CsoundSynthesizer>
```



Per costruire un rumore bianco filtrato da filtro passa alto basterà sostituire l'opcode **tonex** con **atone** (vedere l'esempio Cap.6.7_Highpass filter), anche in questo caso sarà possibile definire il numero di filtri in serie. Per un utilizzo più semplice con una sola istanza di filtro per volta, esistono gli opcode **tone** e **atone**, la sintassi e il funzionamento sono identici alle varianti con il banco in serie (manca il parametro che determina il numero di filtri in serie).

Il prossimo opcode **butterbp** è un filtro di tipo passa-banda, i filtri di *Butterworth* in Csound richiedono maggior potenza di calcolo rispetto ai filtri standard tone e atone (problema ormai trascurabile con la potenza di calcolo dei moderni processori), il vantaggio è una qualità audio e precisione superiore.

```
ares butterbp asig, kfreq, kband
```

kfreq indica la frequenza di taglio del filtro, *kband* è la larghezza di banda, con un esempio del genere :

```
ares butterbp asig, 1000, 100
```

si indica che il filtro lascerà passare solo le frequenze comprese tra i valori 950 e 1050.

Vediamo due esempi, standard e con Fltk :

Esempio Cap.6.8_Bandpass filter(nogui)

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=1
0dbfs = 1

;oscillatore buzz filtrate

instr 1

klfo  poscil      3,5,1 ;lfo
a1    buzz   1,110+klfo,100,1
a2    buzz   1,220+klfo,100,1
asum  sum   a1,a2 ;somma di due signal buzz
kfreq randomi    1000,4000,1 ;curva random per filtro
kband = p4        ;larghezza di banda
afilt butterbp    asum,kfreq,kband ;filtro passabanda
aout  balance     afilt,asum ;bilanciamento audio dei due segnali
```

```

out    aout
endin

</CsInstruments>
<CsScore>

;p4      larghezza di banda

f1    0    16384 10    1
i1    0    10    10
s
i1    0    10    100
s
i1    0    10    1000
s

</CsScore>
</CsoundSynthesizer>

```

Esempio Cap.6.8_Bandpass filter

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=1
0dbfs = 1

FLpanel      "band-pass Butterworth filter",610,190,300,300,4,-1,1

gk1,ihb1      FLbutton      "Play",1,0,21,100,30,20,0,1,0,-1
gk2,ihb2      FLbutton      "Stop",1,0,21,100,30,20,60,0,1,0,0
gk3,ihb4      FLbutton      "Exit",1,0,21,100,30,20,100,0,2,0,0

;flbox value for FLslider
ihval1        FLvalue      "",60,30,520,20
ihval2        FLvalue      "",60,30,520,70

gkf freq,ih1  FLslider      "filter freq",100,20000,-1,5,ihval1,355,30,143,20
gkband,ih2   FLslider      "bandpass",1,400,0,5,ihval2,355,30,143,70

FLpanel_end
FLrun
;valori iniziali
FLsetVal_i 4000,ih1      ;init value freq filter
FLsetVal_i 10,ih2       ;init value bandpass
;rumore bianco filtrato
instr 1

a1    rand 1
afilt butterbp    a1,gkf freq,gkband
out    afilt

endin

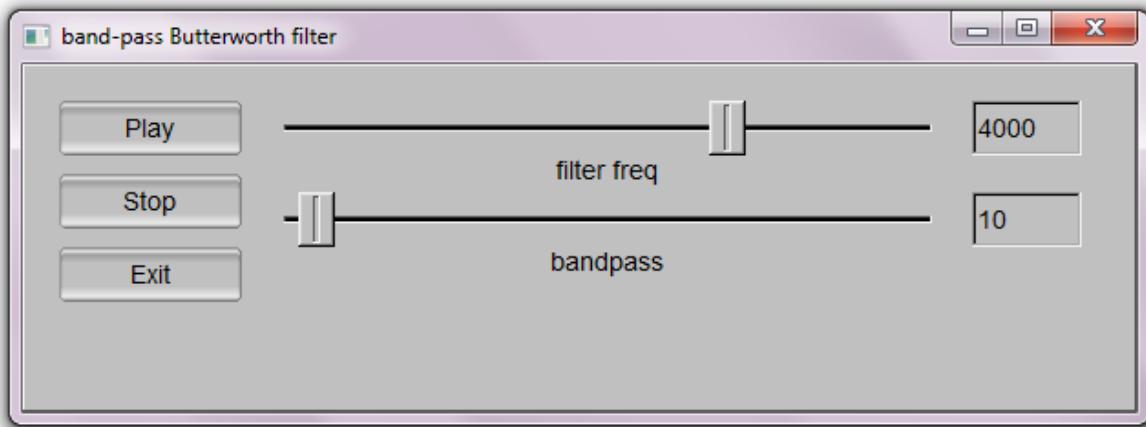
```

```

instr 2
exitnow
endin
</CsInstruments>
<CsScore>

f0      3600

</CsScore>
</CsoundSynthesizer>
```



Per costruire un banco filtri in parallelo viene in aiuto l'opcode **statevar**, si tratta di un particolare tipo di filtro con quattro uscite indipendenti : high-pass, low-pass, band-pass and band-reject, si tratta quindi di quattro filtri in uno. Oltre al parametro di frequenza centrale, **statevar** utilizza come argomento il Q del filtro (rapporto tra la frequenza di taglio e la larghezza di banda) :

```
ahp,alp,abp,abr statevar ain,kcf,kq
```

ahp,alp,abp,abr sono le quattro uscite del filtro (4 filtri diversi), seguono il segnale d'ingresso, frequenza di taglio e Q del filtro.

Vediamo un esempio in cui selezionare il tipo di segnale filtrato in uscita, oppure la somma di 2 o più filtri :

Esempio Cap.6.9_statevar filter(nogui)

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

a1    rand  1
kcutoff   randomi 100,10000,2
kQ      =     p4
iosamps  =     20 ;oversampling del processamento
```

```

        ;, aumenta la componente acuta di Q
ahp,alp,abp,abr    statevar      a1,kcutoff,kQ,iosamps

kselect      =      p5      ; sceglie il tipo di filtro

if      kselect      =      1      then
aout      =      ahp
elseif      kselect      =      2      then
aout      =      alp
elseif      kselect      =      3      then
aout      =      abp
elseif      kselect      =      4      then
aout      =      abr
elseif      kselect      =      5      then
aout      =      ahp+alp
elseif      kselect      =      6      then
aout      =      abp+abr
elseif      kselect      =      7      then
aout      =      abp+abr+ahp
elseif      kselect      =      8      then
aout      =      abp+abr+ahp+alp
endif

aclip clip  aout,0,.9   ; funzione di limiter

out    aclip

endin

</CsInstruments>
<CsScore>

i1    0    10    .1    1    ;hp
s
i1    0    10    .2    2    ;lp
s
i1    0    10    .3    3    ;bp
s
i1    0    10    .4    4    ;br
s
i1    0    10    .5    5    ;hp+lp
s
i1    0    10    .6    6    ;bp+br
s
i1    0    10    .7    7    ;bp+br+hp
s
i1    0    10    .8    8    ;4 filtri
s

</CsScore>
</CsoundSynthesizer>
```

Il prossimo esempio crea delle dissolvenze incrociate tra le quattro uscite, il controllo di questo processo avviene per mezzo di una variabile di controllo creata con *poscil*.

Esempio Cap.6.9_statevar-dissolvenze

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1      rand   1

kcutoff      =      1000
kQ      =      .9
iosamps      =      20      ;oversampling del processamento
                           ;, aumenta la componente acuta di Q
ahp,alp,abp,abr  statevar    a1,kcutoff,kQ,iosamps

;dissolvenze
k1      line  0,p3,1
k2      linseg     1,p3/2,0,p3/2,1

aout1 =      ahp*k1      ;uscita moltiplicata per 0 e 1
aout2 =      alp*k2
aout3 =      abp*(1-k1)
aout4 =      abr*(1-k2)

aright      sum    aout1,aout2
aleft sum    aout3,aout4

aclip1      clip  aright,0,.9
aclip2      clip  aleft,0,.9

outs  aclip1,aclip2      ;uscita stereo

endin

</CsInstruments>
<CsScore>

f1      0      4096  10      1

i1      0      20

</CsScore>
</CsoundSynthesizer>

```

Un semplice esempio finale per il tempo reale :

Esempio Cap.6.9_statevar filter

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10

```

```

nchnls = 1
0dbfs = 1

FLpanel      "Statevar multifilter",630,350,100,100,1,-1,-1

ival1 FLvalue      "",60,30,540,20
ival2 FLvalue      "",60,30,540,70
ival3 FLvalue      "",60,30,540,120
ival4 FLvalue      "",60,30,540,170
ival5 FLvalue      "",60,30,540,220
ival6 FLvalue      "",60,30,540,270

imin   = 1
imax   = 100
iexp   = 0
iheight = 30
iwidth  = 500
ix     = 20
iy     = 20

gk1,ih1    FLslider  "cutoff",20,15000,iexp,21,ival1,iwidth,iheight,ix,iy
gk2,ih2    FLslider  "Q",0.1,0.9,iexp,21,ival2,iwidth,iheight,ix,iy+50
gk3,ih3    FLslider  "high-pass",0,1,iexp,25,ival3,iwidth,iheight,ix,iy+100
gk4,ih4    FLslider  "low-pass",0,1,iexp,25,ival4,iwidth,iheight,ix,iy+150
gk5,ih5    FLslider  "band-pass",0,1,iexp,25,ival5,iwidth,iheight,ix,iy+200
gk6,ih6    FLslider  "band-reject",0,1,iexp,25,ival6,iwidth,iheight,ix,iy+250

FLpanelEnd

FLrun

;valori iniziali
FLsetVal_i 10000,ih1 ;init value freq filter
FLsetVal_i .5,ih2 ;init value Q
FLsetVal_i 0,ih3 ;init value amplitude
FLsetVal_i 0,ih4 ;init value freq filter
FLsetVal_i 0,ih5 ;init value resonance filter
FLsetVal_i 0,ih6 ;init value amplitude

instr 1

a1      rand  1

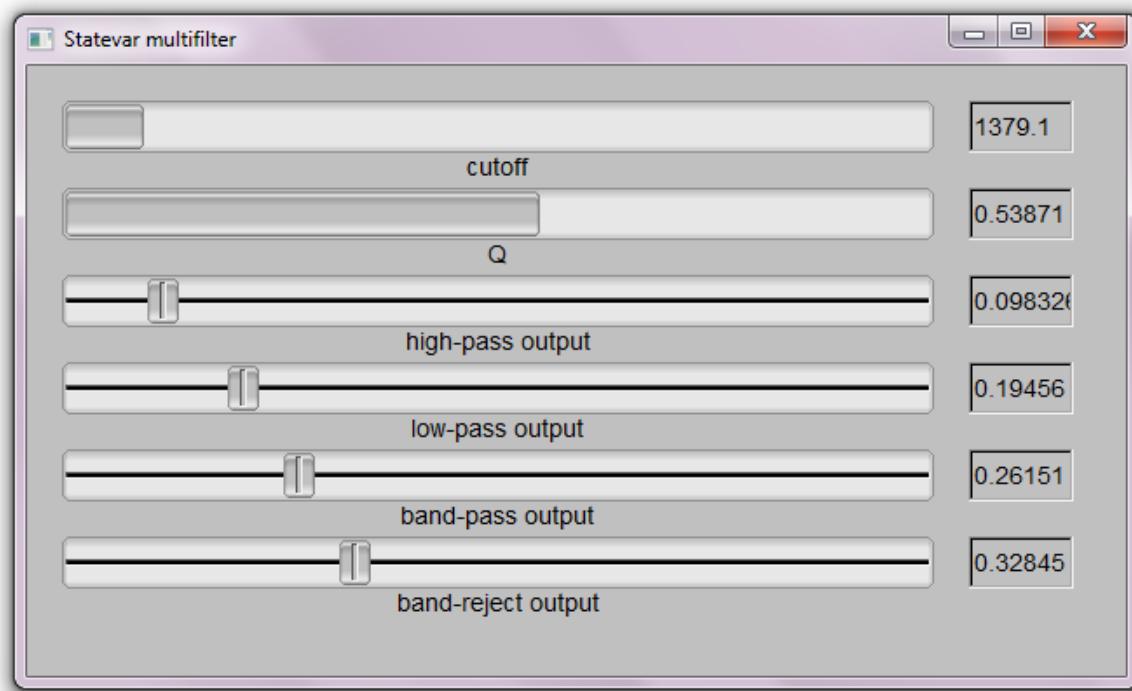
ahp,alp,abp,abr  statevar    a1,gk1,gk2,10
aout  sum    ahp*gk3,alp*gk4,abp*gk5,abr*gk6
out    aout

endin

</CsInstruments>
<CsScore>

i1      0      3600

</CsScore>
</CsoundSynthesizer>
```



Passiamo adesso all'emulazione digitale di un classico della sintesi analogica, il glorioso lowpass filter Moog. Esistono due opcode che ricreano questi tipi di filtri analogi, si tratta di **moogladder** e **moogvcf**, il prossimo esempio utilizza **moogladder** (questo opcode è disponibile solo nelle nuove versioni di Csound5, prima di questo nuovo opcode veniva utilizzato Moogvcf), si tratta di un filtro passa basso con risonanza (range 0 e 1), il modello originale dei synth Moog era costituito da quattro blocchi di filtro in cascata con un taglio di 6 db.

Esempio Cap.6.10_Moogladder(nogui)

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
sr=44100
kr=4410
ksmps=10
nchnls=1
0dbfs = 1

ga1    init  0

turnon      1      ;attiva gli strumenti per un tempo indefinito
turnon      2

;step sequencer
instr 1

ifunc =      2      ;tabella con note

krate =      .5
kstep poscil     8,krate,1;genera valori per indice della tabella
kpitch      table abs(kstep),ifunc

;(abs restituisce il valore assoluto di kstep per evitare i float number)

a1      vco2  .6,cpspch(kpitch),0;oscillatore Vco sawtooth
```

```

vincr gal,al      ; manda il segnale al filtro attivo
endin

instr 2

kfreq randomi     60,10000,2
kreso randomi     .1,.9,2

; riceve la variabile globale gal (VCO)
afilt moogladder gal,kfreq,kreso ; filtro moog

out    afilt

clear gal   ; evita effetti di accumulo

endin

</CsInstruments>
<CsScore>

f1      0      4096 10      1
f2 0 8 -2 6.00 6.03 6.05 6.06 6.07 6.11 7.00 7.03 ; note per step sequencer

f0      3600

</CsScore>
</CsoundSynthesizer>

```

Esempio Cap.6.10_moogladder

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=1
0dbfs = 1

FLpanel      "Moogladder",610,190,300,300,4,-1,1

gk1,ihb1    FLbutton    "Play",1,0,21,100,30,20,20,0,1,0,-1
gk2,ihb2    FLbutton    "Stop",1,0,21,100,30,20,60,0,1,0,0
gk3,ihb4    FLbutton    "Exit",1,0,21,100,30,20,100,0,2,0,0

; flbox value for FLslider
ihvall     FLvalue      "",60,30,520,20
ihval2     FLvalue      "",60,30,520,70
ihval3     FLvalue      "",60,30,520,120

gkfreq,ih1  FLslider    "Moog filter freq",20,20000,0,5,ihvall,355,30,143,20
gkreso,ih2  FLslider    "Resonance",.1,.95,0,5,ihval2,355,30,143,70
gkamp,ih3   FLslider    "Amplitude",0,1,0,5,ihval3,355,30,143,120
FLpanel_end
FLrun

FLsetVal_i  10000,ih1   ; init value freq filter

```

```

FLsetVal_i .5,ih2      ;init value resonance filter
FLsetVal_i .1,ih3      ;init value amplitude

instr 1

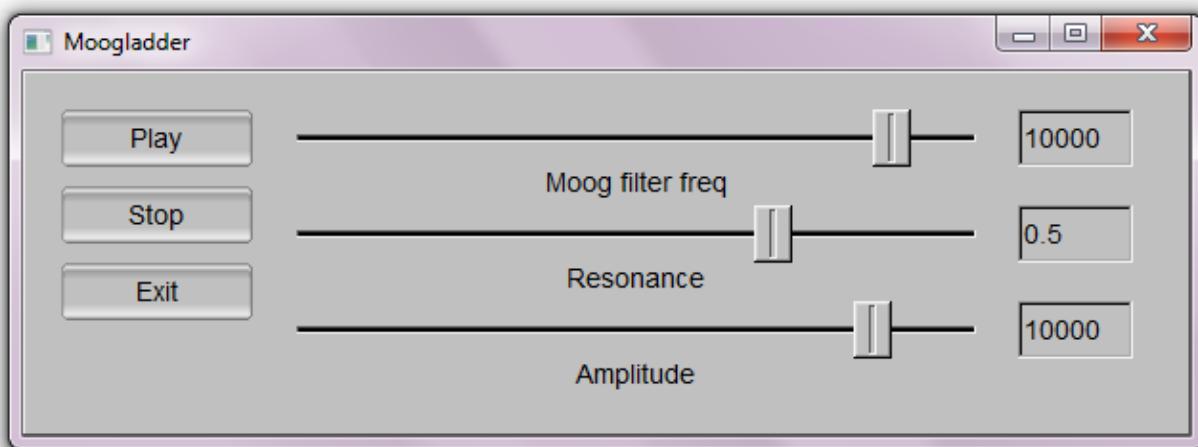
k1    randomh   60,1000,6 ;genera melodia random
a1    vco2  gkamp,k1,0 ;oscillatore Vco sawtooth
afilt moogladder a1,gkfreq,gkreso ;filtro moog
out   afilt

endin

instr 2
exitnow
endin

</CsInstruments>
<CsScore>
f0    3600
</CsScore>
</CsoundSynthesizer>

```



Le possibilità timbriche offerte da Csound con le tecniche di filtraggio sono molto ampie, si rimanda alla lettura del manuale ufficiale nelle sezioni Standard Filters e Specialized Filters.

L'opcode **fofilter** fa parte di quest'ultima categoria, il nome deriva da Fof (generatore di formanti spettrali, si tratta di un particolare tipo di sintesi granulare che verrà discussa nei capitoli successivi, la sintesi con Fof viene molto utilizzata per la simulazione del tratto vocale). Gli argomenti di **fofilter** sono la frequenza centrale della formante (Hz), attacco e decadimento dei grani generati, la sintassi di fofilter :

```
asig fofilter ain,kcf,kris,kdec
```

Esempio Cap.6.11_fofilter(nogui).csd

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=1
0dbfs = 1

```

```

instr 1
;additiva di parziali cosinusoidali

kcps = 110 ;frequenza
knh = 80 ;numero di armoniche
klh = 1 ;armonica più bassa
kmul randomi 0,1,10 ;incrementa l'ampiezza delle parziali

kvib lfo 2,5

asig gbuzz .6,kcps+kvib,knh,klh,kmul,1

;fofilter
kfreq randomi 60,2000,1
kris expon 0.001,p3,0.007
kdec expon 0.001,p3,0.04
afilt fofilter asig,kfreq,kris,kdec

aout clip afilt,0,.9 ;limiter

out aout

endin

</CsInstruments>
<CsScore>

; a cosine wave
f 1 0 16384 11 1

i1 0 10

</CsScore>
</CsoundSynthesizer>

```

L'esempio seguente utilizza questo particolare filtro su un campione audio in lettura, per leggere il campione verrà utilizzato **flooper2** in grado di creare loop con funzione di crossfade.

Esempio Cap.6.11_fofilter.csd

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=1
0dbfs = 1

FLpanel "Fofilter",610,190,300,300,4,-1,1

gk1,ihb1 FLbutton "Play",1,0,21,100,30,20,20,0,1,0,-1
gk2,ihb2 FLbutton "Stop",1,0,21,100,30,20,60,0,1,0,0
gk3,ihb4 FLbutton "Exit",1,0,21,100,30,20,100,0,2,0,0

;flbox value for FLslider

```

```

ihval1      FLvalue    "",60,30,520,20
ihval2      FLvalue    "",60,30,520,70
ihval3      FLvalue    "",60,30,520,120

gkfreq,ih1  FLslider   "filter freq",100,5000,0,5,ihval1,355,30,143,20
gkris,ih2   FLslider   "attack",0,.01,0,5,ihval2,355,30,143,70
gkdec,ih3   FLslider   "decay",0.01,.05,0,5,ihval3,355,30,143,120
FLpanel_end
FLrun
;*****FLsetVal_i 1000,ih1 ;init value freq filter
FLsetVal_i 0,ih2 ;init value attack
FLsetVal_i 0.1,ih3 ;init value decay

instr 1

a1      flooper2     .1,1,0,2,0.005,1
afilt fofilter       a1,gkfreq,gkris,gkdec
out    afilt

endin

instr 2
exitnow
endin

</CsInstruments>
<CsScore>

f1      0      0      1      "loop2.wav"  0      0
f0      3600

</CsScore>
</CsoundSynthesizer>

```



Ora una semplice gui per la sintesi sottrattiva con controllo midi, è un modello ispirato ai sintetizzatori analogici degli anni settanta, un oscillatore VCO (*voltage controlled oscillator*) filtrato da **moogladder**, il sistema ha due inviluppi dedicati al volume in uscita ed al controllo del filtro, l'oscillatore è settato per generare una semplice forma d'onda *saw* con *detune* regolabile.

Esempio Cap.6.12_Simple_VCO_Moogfilter

```

<CsoundSynthesizer>
<CsOptions>
-+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>

```

```

sr=44100
kr=4410
ksmps=10
nchnls=2
0dbfs = 1

FLpanel      "Simple VCO Moogfilter",610,550,300,100,4,-1,1
FLvkeybd    "keyboard.map",560,90,20,430
gkonoff,ihb2      FLbutton "On/Off",0,1,2,100,30,20,20,0,1,0,0
gk3,ihb4      FLbutton "Exit",1,0,21,100,30,20,70,0,2,0,0

;display per FLslider
ihval1      FLvalue "",60,30,520,20
ihval2      FLvalue "",60,30,520,70
ihval3      FLvalue "",60,30,520,120
ihval4      FLvalue "",60,30,520,120+80
ihval5      FLvalue "",60,30,520,120+130
ihval6      FLvalue "",60,30,520,120+180
ihval7      FLvalue "",60,30,520,120+230

;kout,ihandle FLslider "",imin,imax, iexp, itype, idisp, iwidth,iheight, ix, iy
gkfreq,gih1 FLslider      "Cutoff",1,(sr)/2,0,25,ihval1,355,25,143,20
gkreso,gih2 FLslider      "Resonance",0.1,.95,0,25,ihval2,355,25,143,70
gklfo,gih3 FLslider      "LFO-level",0,4,0,25,ihval3,155,25,143,120
gkrate,gih3b     FLslider      "LFO-rate",.1,6,0,25,ihval3,155,25,343,120

gkdet,gihdet     FLknob      "detune",1,1.2,0,1,-1,60,40,120

;adsr volume
gkatt,gih4  FLslider      "attack(volume)",0.01,1,0,5,ihval4,155,30,143,200
gkdec,gih5  FLslider      "decay",0,1,0,5,ihval5,155,30,143,250
gksus,gih6  FLslider      "sustain",0,1,0,5,ihval6,155,30,143,300
gkrel,gih7  FLslider      "release",0,1,0,5,ihval7,155,30,143,350
;adsr filtro moog
gkatt2,gih8 FLslider      "attack(filter)",0.01,1,0,5,ihval4,155,30,343,200
gkdec2,gih9 FLslider      "decay",0,1,0,5,ihval5,155,30,343,250
gksus2,gih10 FLslider      "sustain",0,1,0,5,ihval6,155,30,343,300
gkrel2,gih11 FLslider      "release",0,1,0,5,ihval7,155,30,343,350

FLpanel_end
FLrun
;valori iniziali
FLsetVal_i 12000,gih1 ;init value filter
FLsetVal_i .2,gih2
FLsetVal_i 2,gih3
FLsetVal_i 0.5,gih4 ;init value a
FLsetVal_i 0.3,gih5 ;init value d
FLsetVal_i .3,gih6 ;init value s
FLsetVal_i .5,gih7 ;init value r
FLsetVal_i 0.5,gih8 ;init value a
FLsetVal_i 0.3,gih9 ;init value d
FLsetVal_i .3,gih10 ;init value s
FLsetVal_i .5,gih11 ;init value r
FLsetVal_i .1,gihdet ;detune

instr 1

if      gkonoff      =      1      then
turnoff
endif

iamp  ampmidi      .6

```

```

ifreq cpsmidi
klfo oscili gklfo,gkrate,1
avcol vco2 iamp,ifreq+klfo,0
avco2 vco2 iamp,(ifreq*gkdet)+klfo,0
asum sum avcol,avco2
kadsrfilt mxadsr i(gkatt2),i(gkdec2),i(gksus2),i(gkrel2),0,1
afilt moogladder asum,gkfreq*kadsrfilt,gkres0

kadsr mxadsr i(gkatt),i(gkdec),i(gksus),i(gkrel),0,1
outs afilt*kadsr,afilt*kadsr

endin

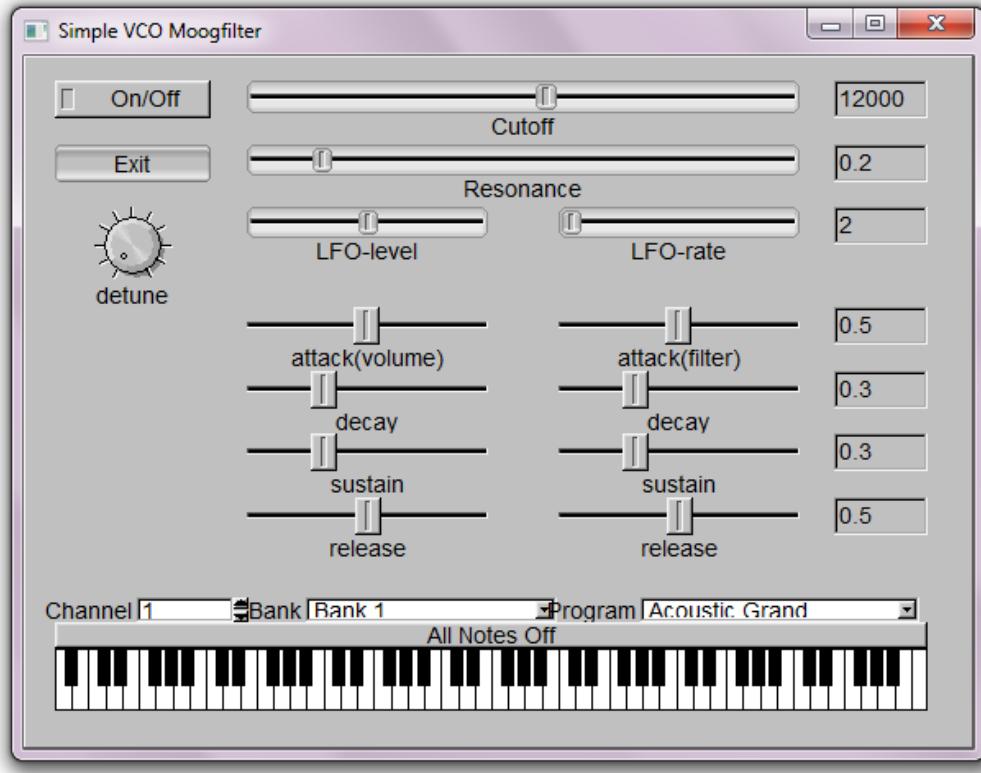
instr 2
exitnow
endin

</CsInstruments>
<CsScore>

f1 0 16384 10 1
f0 3600

</CsScore>
</CsoundSynthesizer>

```



Concludiamo questa sezione introducendo uno speciale filtro adatto a costruire equalizzatori parametrici, si tratta di un particolare filtro respingi banda in cui viene definita una frequenza di taglio ed una larghezza di banda con gain. Csound offre tre tipi di opcode adatti a questo scopo :

pareq
rbjeq

eqfil

Esempio Cap.6.13_eqfil

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
;Esempio Cap.6.13_eqfil
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1      ;suono originale
a1,a2 diskin2      "loop2.wav",1,0,1
outs a1*.7,a1*.7

endin

instr 2      ;suono equalizzato
a1,a2 diskin2      "loop2.wav",1,0,1

kcf      =      p4      ;frequenza centrale
kbw      =      p5      ;notch banda
kgain =      p6      ;notch gain

aeq      eqfil a1,kcf,kbw,kgain
outs aeq*.7,aeq*.7

endin
</CsInstruments>
<CsScore>

i1    0      4      ;suono originale
s
i2    0      4      200    200    3
s
i2    0      4      600    100    6
s
i2    0      4      1200   2000   3
s
i2    0      4      100    50     4
s

</CsScore>
</CsoundSynthesizer>
```

una variante in cui costruiamo un equalizzatore a tre bande :

Esempio Cap.6.14_eq_3band

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
```

```

<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1      ;suono equalizzato
a1,a2 diskin2      "loop2.wav",1,0,1

;prima banda eq
kcf    =      p4      ;frequenza centrale
kbw    =      p5      ;notch banda
kgain =      p6      ;notch gain

;seconda banda eq
kcf2   =      p7
kbw2   =      p8
kgain2 =      p9

;terza banda eq
kcf3   =      p10
kbw3   =      p11
kgain3 =      p12

aeq1  eqfil a1,kcf,kbw,kgain
aeq2  eqfil a1,kcf2,kbw2,kgain2
aeq3  eqfil a1,kcf3,kbw3,kgain3

aeq  sum  aeq1,aeq2,aeq3      ;tre filtri in parallelo
outs  aeq*.2,aeq*.2

endin

</CsInstruments>
<CsScore>

i1    0      4      3200  1000  3      100    1000  3      1400  1000  3
s
i1    0      4      400   1000  3      1000   100   5      200    1000  4
s
i1    0      4      2400  1000  3      1000   2000  6      100    1000  4
s

</CsScore>
</CsoundSynthesizer>

```

Visto l'elevato numero di filtri contenuti in Csound, non è qui possibile poter approfondire e illustrare le potenzialità di ogni tipo, concludiamo con un eccellente multi filtro (low-pass e high-pass) a più poli, si tratta di **clfilt**.

```
ares clfilt asig, kfreq, itype, inpol [, ikind] [, ipbr] [, isba] [, iskip]
```

itype : il valore 0 = low-pass, 1 = high-pass

inpol : permette di indicare il numero di poli (valore compreso tra 2 e 80, naturalmente con valori alti aumenta la mole di lavoro richiesta alla cpu)

ikind (opzionale) : permette di scegliere tra tre tipologie di filtro :

- 0 = Butterworth
- 1 = Chebyshev Type I
- 2 = Chebyshev Type II

(avvertenze : il manuale ufficiale indica un quarto tipo (valore = 3) di filtro Elliptical,in realtà non è implementato e viene generato un messaggio di errore dal compilatore).

Esempio Cap.6.15_clfilt

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1,a2 diskin2      "loop2.wav",1,0,1

kfreq =      p4      ;frequenza di taglio
itype =      p5      ;0 = lowpass,1 = highpass
inpol =      p6      ;numero di poli (2 a 80)

;0 for Butterworth
;1 for Chebyshev Type I
;2 for Chebyshev Type II

ikind =      p7      ;tipo di filtro

afilt1      clfilt      a1,kfreq,itype,inpol,ikind
afilt2      clfilt      a2,kfreq,itype,inpol,ikind

outs  afilt1,afilt2

endin

</CsInstruments>
<CsScore>

;lowpass Butterworth (10 poli)
i1    0      4      2000   0      10      0
s
;lowpass Chebyshev Type I
i1    0      4      2000   0      10      1
s
;lowpass Chebyshev Type II
i1    0      4      2000   0      10      2
s
;highpass Butterworth (10 poli)
```

```
i1      0      4      1000   1      10      0
s
;highpass Chebyshev Type I
i1      0      4      1000   1      10      1
s
;highpass Chebyshev Type II
i1      0      4      1000   1      10      2
s
```

```
</CsScore>
</CsoundSynthesizer>
```

Cap.7 Tecniche di modulazione

7.1 Ring modulator

7.3 Sintesi Fm,Pm

7.4 Waveshape

Ring modulator

La ring modulator (modulazione ad anello) è un'altra delle tecniche di sintesi storiche, un esempio su tutti è la composizione *Mantra (1969-70) per due pianoforti e modulatori ad anello* di *Karlheinz Stockhausen*. Si tratta di una tecnica in cui due segnali bipolarì vengono moltiplicati tra loro, il risultato sarà dato dalla somma più la differenza delle frequenze dei due segnali, mentre le frequenze originali vengono soppresse. Se chiamiamo le due sorgenti P (portante) e M (modulante) possiamo avere la seguente formula :

$$\text{Modulazione ad anello} = (\text{Pf} + \text{Mf}) + (\text{Pf} - \text{Mf})$$

In Csound possiamo realizzare una semplice modulazione ad anello nel seguente modo :

```
amodulante oscili amp, freq1, 1  
aportante oscili amodulante, freq2, 1  
  
out aportante
```

Come possiamo osservare, la sorgente amodulante (in questo caso una semplice sinusoide) entra nell'ingresso dell'oscillatore portante che va all'uscita audio; nell'esempio seguente usiamo un altro strumento Csound in cui un terzo segnale moltiplica due sorgenti sonore con spettro diverso (in questo caso una simulazione di corda pizzicata ed una sinusoide), come potete vedere vengono utilizzati valori di ampiezza molto bassi per evitare che lo strumento esploda.

```
instr 1  
  
a1 pluck 100, 440, 220, 0, 1 ;corda pizzicata  
kfreq linseg 60, p3/2, 440, p3/2, 60 ;glissando  
a2 oscil 100, kfreq, 1 ;oscillatore con glissando di frequenze  
aring = a1*a2 ;I due segnali vengono moltiplicati  
out aring  
  
endin
```

Vediamo adesso un esempio completo con gui per poter modificare i parametri in tempo reale :

Esempio Cap.7.1_Ring modulation

```
<CsoundSynthesizer>  
<CsOptions>  
</CsOptions>  
<CsInstruments>  
  
sr=44100  
kr=4410  
ksmps=10  
nchnls=1  
0dbfs = 1  
  
FLpanel "Ring modulation", 610, 190, 300, 300, 4, -1, 1  
  
gk1,ihb1 FLbutton "Play", 1, 0, 21, 100, 30, 20, 20, 0, 1, 0, -1  
gk2,ihb2 FLbutton "Stop", 1, 0, 21, 100, 30, 20, 60, 0, 1, 0, 0  
gk3,ihb4 FLbutton "Exit", 1, 0, 21, 100, 30, 20, 100, 0, 2, 0, 0  
  
;flbox value for FLslider  
ihval1 FLvalue "", 60, 30, 520, 20  
ihval2 FLvalue "", 60, 30, 520, 70  
ihval3 FLvalue "", 60, 30, 520, 120  
  
gkamp,ih1 FLslider "Amplitude", 0, .6, 0, 5, ihval1, 355, 30, 143, 20
```

```

gkfreq1,ih2 FLslider      "freq modulation",1,1000,0,5,ihval2,355,30,143,70
gkfreq2,ih3 FLslider      "freq port",20,2000,0,5,ihval3,355,30,143,120
FLpanel_end
FLrun

FLsetVal_i .1,ih1          ;init value Amplitude
FLsetVal_i 10,ih2          ;init value freq modulation
FLsetVal_i 440,ih3         ;init value freq port

instr 1

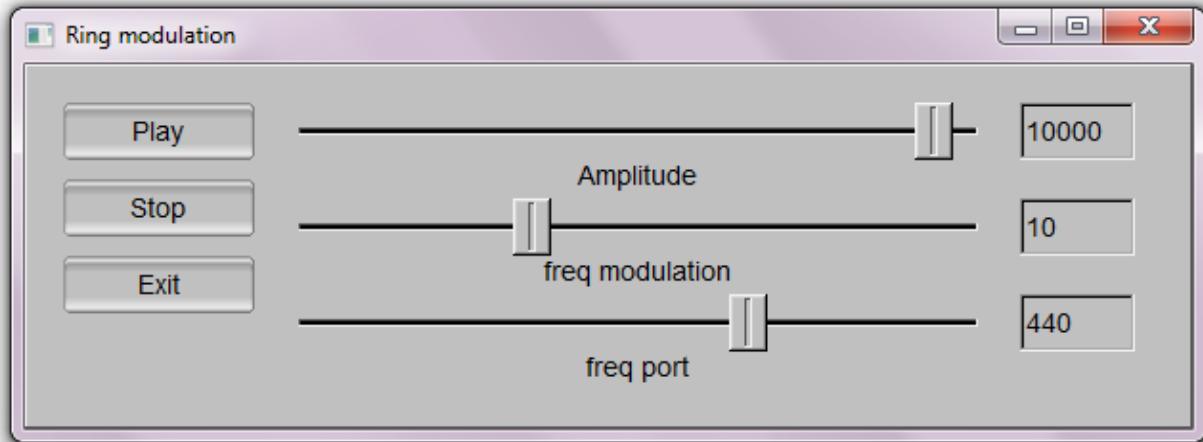
kamp  portk gkamp,0.01   ;elimina rumore durante la transizione
amod  oscili      kamp,gkfreq1,1
aport oscili      amod,gkfreq2,1
out    aport

endin

instr 2
exitnow
endin

</CsInstruments>
<CsScore>
f1    0    16384 10      1
f0    3600
</CsScore>
</CsoundSynthesizer>

```



Un'altra tecnica di modulazione molto simile è la modulazione d'ampiezza (AM), l'idea è la stessa del concetto di tremolo (segnale di controllo che modula l'ampiezza di un oscillatore), per realizzare un esempio di sintesi AM bisogna modularre l'ampiezza della portante utilizzando un oscillatore audio. Questo tipo di sintesi, a differenza della ring modulator, utilizza segnali unipolari (cioè che non oscillano tra segnali positivi e negativi), in Csound è possibile questo sommando una componente positiva alla frequenza modulante (questo valore aggiunto prende il nome di DC offset o componente di corrente continua).

un semplice esempio di modulazione AM :

```

a1 oscil 10000,440,1
amod = a1 + 1 ;viene creato un segnale unipolare aggiungendo un valore positivo

a2 oscil amod,220,1

out a2

```

una variante più interessante è quella di moltiplicare i due segnali utilizzando due funzioni d'onda diverse :

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1

a1 oscili 100,220,2
a2 oscili 100,440,1

amod = (a1 + 100) * a2

out amod

endin
</CsInstruments>
<CsScore>
f1 0 4096 10 1
f2 0 4096 10 1 .9 .8 .7 .6 .5 .4 .3 .2 .1

i1 0 4
</CsScore>
</CsoundSynthesizer>
```

Sintesi FM (modulazione di frequenza)

Il primo algoritmo di modulazione per frequenza venne introdotto da John Chowning nel 1967, per l'epoca si trattava di una svolta decisiva per la creazione di timbri complessi utilizzando due soli oscillatori, per questo ragione venne da subito preferita alla sintesi additiva, che richiedeva invece enormi quantità di dati. La Yamaha fu la prima casa produttrice ad introdurre questa tecnica di sintesi su macchine commerciali, lo storico sintetizzatore si chiamava Dx7 nel 1983 (a cui seguirono numerose varianti come Dx21, Dx100, Tx81z, fino ai modelli ibridi che integravano anche la sintesi pcm come i modelli della serie SY.).

Il concetto base di questa tecnica è quello di utilizzare un oscillatore per modulare la frequenza di un altro oscillatore detto "portante".

Prendiamo come esempio un semplice algoritmo del vibrato, in questo caso l'oscillatore modulante è un Lfo (oscillatore a bassa frequenza, in genere sotto i 16 hz) con valori di ampiezza molto bassi, nella sintesi Fm l'oscillatore modulante è di tipo audio, quindi con valori di ampiezza e frequenza simili all'oscillatore portante, il risultato è un alterazione complessa del timbro.

In un sintetizzatore come lo storico Yamaha Dx7 (basato sulla modulazione di fase che vedremo in seguito), gli oscillatori venivano chiamati operatori (tutti di tipo sinusoidale, della serie Dx solamente il modello Tx81z disponeva di operatori con altri tipi di forme d'onda), questo modello ne aveva in dotazione sei e configurabili in vari modi (alternando la funzione come portante o modulante).

La configurazione dei vari operatori viene chiamata Algoritmo, in cui viene descritto il routing e la funzione dell'operatore (modulante o portante), è anche possibile assegnare ad un operatore portante diverse modulanti con valori di frequenza diversi creando spettri molto complessi, questo grazie anche ai vari inviluppi di ogni operatore.

Una delle applicazioni più tipiche di questa tecnica di sintesi è l'emulazione dei timbri acustici (tecnica che tuttavia oggi è stata abbandonata in favore della più efficace e versatile sintesi per modelli fisici).

La sintesi Fm può essere realizzata in Csound utilizzando un opcode specifico per questa tecnica chiamato **foscil** (di cui esiste una variante interpolata chiamata **foscili**); questo opcode contiene in un'unica riga di codice i due operatori portante e modulante, la sua sintassi :

```
ares foscil xamp, kcps, xcar, xmod, kndx, ifn
```

- xamp : ampiezza del segnale
- kcps : un comune denominatore per le frequenze di car e mod
- xcar : frequenza dell'oscillatore portante moltiplicato per kcps
- xmod : frequenza dell'oscillatore modulante moltiplicato per kcps
- kndx : indice di modulazione
- ifn : funzione per forma d'onda

Una delle caratteristiche legate alla modulazione del segnale portante è il verificarsi di bande laterali (serie di frequenze somma e di frequenze differenza sui lati della portante), l'indice di modulazione serve a rendere udibili queste bande laterali, ha quindi funzione di ampiezza e influisce sulla componente spettrale della portante.

Come vengono calcolate queste bande laterali? supponiamo di avere :

Carrier = 400hz

Mod = 10hz

Con la seguente formula:

Frequenza portante ± numeri interi (1,2,3,4,5,6,7,...) Frequenza modulante

otteniamo :

Somma	differenza
C+M = 410	C-M = 390
C+2M = 420	C-2M = 380
C+3M = 430	C-3M = 370
C+4M = 440	C-4M = 360

Ecc.....

Dal punto di vista teorico la serie di frequenze-somma e frequenze-differenza è infinita ma il livello di udibilità di queste bande laterali sarà dato dall'indice di modulazione.

Prima di utilizzare foscil costruiremo un piccolo sintetizzatore Fm utilizzando **poscil** (variante di oscil ad alta precisione), l'oscillatore modulante verrà sommato alla frequenza dell'oscillatore portante, l'indice di modulazione controllerà l'intensità dell'effetto di modulazione per mezzo di un semplice inviluppo :

Esempio Cap.7.2_Simple Fm(oscil)

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
```

```
sr = 44100
```

```

kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

icps = p5 ;frequenza osc portante
kindx linseg 0,p3/2,p7,p3/2,0 ;inviluppo indice di modulazione
amodulator poscil kindx*p6,p6,1 ;oscillatore modulante
acar poscil p4,icps+amodulator,1 ;oscillatore portante modulato
aenv adsr 0.5,0,1,0.5 ;inviluppo per il segnale in uscita
out acar*aenv

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 1
;p4 = amp
;p5 = freq car
;p6 = freq mod
;p7 = indice di modulazione

i1 0 4 .6 140 200 1
i1 5 4 .6 140 200 10
i1 10 4 .6 140 200 20
i1 15 4 .6 140 200 40
i1 20 4 .6 140 200 80
i1 25 4 .6 140 200 120

</CsScore>
</CsoundSynthesizer>

```

Il seguente esempio di score permette di capire in che modo possiamo creare spettri armonici e inarmonici :

```

<CsScore>
f1 0 4096 10 1
;p4 = amp car
;p5 = freq car
;p6 = freq mod
;p7 = indice di modulazione

;spettro armonico (numeri interi per C e M)

i1 0 4 .6 200 200 5 ; C:M=1:1
s
i1 0 4 .6 200 400 5 ; C:M=1:2
s
i1 0 4 .6 200 600 5 ; C:M=1:3
s
i1 0 4 .6 200 800 5 ; C:M=1:4
s

;spettro inarmonico (nessun rapporto intero)

i1 0 4 .6 200 44.7 10
s
i1 0 4 .6 44.7 200 10
s

```

```

i1      0      4      .6     200    92     10
s
</CsScore>

```

La prossima gui illustra un semplice esempio di sintesi per modulazione di frequenza utilizzando **foscili**, il comune denominatore (primo parametro di foscili) sarà controllato in pitch tramite la tastiera midi, l'ampiezza del segnale in uscita sarà controllato da un inviluppo di tipo Adsr.

Esempio Cap.7.3_Simple FM(gui)

```

<CsoundSynthesizer>
<CsOptions>
-+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=1
0dbfs = 1

FLcolor 255,255,255
FLpanel "Simple FM",600,500,300,100,4,-1,1
FLvkeybd "keyboard.map",560,90,20,400

gk1,ihb1   FLbutton "On/Off",0,1,2,100,30,20,20,0,1,0,0
FLsetColor 155,255,35,ihb1
gk3,ihb4   FLbutton "Exit",1,0,21,100,30,20,60,0,2,0,0

;flbox value for FLslider
ihval1     FLvalue "",60,30,520,20
ihval2     FLvalue "",60,30,520,70
ihval3     FLvalue "",60,30,520,120
ihval4     FLvalue "",60,30,520,120+60
ihval5     FLvalue "",60,30,520,120+110
ihval6     FLvalue "",60,30,520,120+160
ihval7     FLvalue "",60,30,520,120+210

;kout,ihandle FLslider "",imin,imax,iexp,ittype,idisp,iwidth,iheight,ix,iy

gkcar,gih1  FLslider   "car",1,2,0,25,ihval1,355,30,143,20
gkmod,gih2  FLslider   "mod",1,2,0,25,ihval2,355,30,143,70
gkindx,gih3 FLslider   "index",0,20,0,25,ihval3,355,30,143,120
gkport,giport FLknob    "portamento",0,1,0,1,-1,60,40,150

gkatt,gih4  FLslider   "attack",.01,1,0,5,ihval4,355,30, 143,180
gkdec,gih5  FLslider   "decay",0,1,0,5,ihval5,355,30, 143,230
gksus,gih6  FLslider   "sustain",.01,1,0,5,ihval6,355,30, 143,280
gkrel,gih7  FLslider   "release",.01,1,0,5,ihval7,355,30, 143,330

FLpanel_end
FLrun
;valori iniziali
FLsetVal_i 1,gih1      ;init value carrier
FLsetVal_i 1,gih2      ;init value modulation
FLsetVal_i 2,gih3      ;init value index
FLsetVal_i 0.5,gih4    ;init value a
FLsetVal_i 0.3,gih5    ;init value d
FLsetVal_i .3,gih6     ;init value s
FLsetVal_i .5,gih7     ;init value r

```

```

instr 1

if      gk1      =      1      then
turnoff
endif

iamp  ampmidi      .6
ifreq  cpsmidi

kenv  madsr i(gkatt),i(gkdec),i(gksus),i(gkrel)

kportcar    portk gkcar,gkport
kportmod    portk gkmod,gkport
kportindx   portk gkindx,gkport

a1      foscili     iamp*kenv,ifreq,kportcar,kportmod,kportindx,1
out     a1

endin

instr 2
exitnow
endin

</CsInstruments>
<CsScore>
f1      0      16384 10      1
f0      3600
</CsScore>
</CsoundSynthesizer>

```

Provate a sperimentare timbri complessi costruendo uno strumento Fm con multiple modulanti e differenti inviluppi o indici di modulazione,un possibile esempio :

```

a1 foscili  kamp,k1,gkcar,gkmod1,gkindx1,1
a2 foscili  kamp,k1,gkcar,gkmod2,gkindx2,1
a3 foscili  kamp,k1,gkcar,gkmod3,gkindx3,1
a4 foscili  kamp,k1,gkcar,gkmod4,gkindx4,1

afm = a1+a2+a3+a4

```

la prossima gui è basata sull'istruzione :

```
afm  foscili     iamp,1,icps,icps*freqmod1,index1*kenvindx,1
```

in pratica la frequenza della portante è controllata dalla tastiera midi,mentre la modulante è data dalla moltiplicazione della portante per un valore variabile in real time,abbiamo una portante con quattro modulanti e inviluppi indipendenti per gli indici di modulazione.

Esempio Cap.7.4_FMsynth4Mod

```

<CsoundSynthesizer>
<CsOptions>
-+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>

```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
Odbfs = 1

FLpanel "FMsynth4Mod", 705, 480, 100, 100, 1, -1, -1
FLvkeybd "keyboard.map", 660, 90, 20, 370

imin = 0.01
imax = 1
iexp = 0
iheight = 10
iwidth = 150
ix = 20
iy = 20

;kout,ihandle FLslider "label",imin,imax,iexp,itype,idisp,iwidth,iheight,ix,iy
gk1,gih1 FLslider "A",imin,imax,iexp,1,-1,iwidth,iheight,ix,iy
gk2,gih2 FLslider "D",imin,imax,iexp,1,-1,iwidth,iheight,ix,iy+30
gk3,gih3 FLslider "S",imin,imax,iexp,1,-1,iwidth,iheight,ix,iy+60
gk4,gih4 FLslider "R",imin,imax,iexp,1,-1,iwidth,iheight,ix,iy+90

gk5,gih5 FLslider "A",imin,imax,iexp,1,-1,iwidth,iheight,ix+170,iy
gk6,gih6 FLslider "D",imin,imax,iexp,1,-1,iwidth,iheight,ix+170,iy+30
gk7,gih7 FLslider "S",imin,imax,iexp,1,-1,iwidth,iheight,ix+170,iy+60
gk8,gih8 FLslider "R",imin,imax,iexp,1,-1,iwidth,iheight,ix+170,iy+90

gk9,gih9 FLslider "A",imin,imax,iexp,1,-1,iwidth,iheight,ix+340,iy
gk10,gih10 FLslider "D",imin,imax,iexp,1,-1,iwidth,iheight,ix+340,iy+30
gk11,gih11 FLslider "S",imin,imax,iexp,1,-1,iwidth,iheight,ix+340,iy+60
gk12,gih12 FLslider "R",imin,imax,iexp,1,-1,iwidth,iheight,ix+340,iy+90

gk13,gih13 FLslider "A",imin,imax,iexp,1,-1,iwidth,iheight,ix+510,iy
gk14,gih14 FLslider "D",imin,imax,iexp,1,-1,iwidth,iheight,ix+510,iy+30
gk15,gih15 FLslider "S",imin,imax,iexp,1,-1,iwidth,iheight,ix+510,iy+60
gk16,gih16 FLslider "R",imin,imax,iexp,1,-1,iwidth,iheight,ix+510,iy+90

ihval1 FLvalue "", 60, 30, 65, 215
ihval2 FLvalue "", 60, 30, 65+170, 215
ihval3 FLvalue "", 60, 30, 65+340, 215
ihval4 FLvalue "", 60, 30, 65+510, 215

gkmod1,gimod1 FLslider "kmod1", 1, 20, 0, 21, ihval1, 150, 10+12, ix, iy+150
gkmod2,gimod2 FLslider "kmod2", 1, 20, 0, 21, ihval2, 150, 10+12, ix+170, iy+150
gkmod3,gimod3 FLslider "kmod3", 1, 20, 0, 21, ihval3, 150, 10+12, ix+340, iy+150
gkmod4,gimod4 FLslider "kmod4", 1, 20, 0, 21, ihval4, 150, 10+12, ix+510, iy+150

ihval5 FLvalue "", 60, 30, 405, 260
ihval6 FLvalue "", 60, 30, 405, 310

gkindx1,giindx1 FLslider "Index1", 0, 4, 0, 5, ihval5, 150+210, 10, ix, iy+250
gkindx2,giindx2 FLslider "Index2", 0, 4, 0, 5, ihval6, 150+210, 10, ix, iy+300

gksn1,gisn1 FLbutton "random adsr", 1, 0, 21, 80, 35, 565, 260, 0, 3, 0, -1
gksn2,gisn2 FLbutton "random mod", 1, 0, 21, 80, 35, 565, 310, 0, 4, 0, -1

FLpanelEnd
FLrun

FLsetVal_i 0.5,gih1 ;init value a
FLsetVal_i 0.3,gih1 ;init value d
FLsetVal_i .3,gih2 ;init value s

```

```

FLsetVal_i .5,gih3 ;init value r
FLsetVal_i 0.001,gih5 ;init value a
FLsetVal_i 0.1,gih6 ;init value d
FLsetVal_i .7,gih7 ;init value s
FLsetVal_i .1,gih8 ;init value r
FLsetVal_i 1,gih9 ;init value a
FLsetVal_i 0.3,gih10 ;init value d
FLsetVal_i .1,gih11 ;init value s
FLsetVal_i .5,gih12 ;init value r
FLsetVal_i 0.001,gih13 ;init value a
FLsetVal_i 0.2,gih14 ;init value d
FLsetVal_i .03,gih15 ;init value s
FLsetVal_i .05,gih16 ;init value r

;*****
instr 1

iamp    ampmidi      0.1      ;valore basso (0dbfs = 1)
ifreq    cpsmidi

kfreqmod1   =   gkmod1      ;freqmod associato a slider
kfreqmod2   =   gkmod2
kfreqmod3   =   gkmod3
kfreqmod4   =   gkmod4
kindex1     =   gkindx1
kindex2     =   gkindx2

kenvindx1  mxadsr      i(gk1),i(gk2),i(gk3),i(gk4)  ;inviluppo indice mod.
kenvindx2  mxadsr      i(gk5),i(gk6),i(gk7),i(gk8)
kenvindx3  mxadsr      i(gk9),i(gk10),i(gk11),i(gk12)
kenvindx4  mxadsr      i(gk13),i(gk14),i(gk15),i(gk16)

icps = ifreq ;frequenza osc portante
afm1  foscili    iamp,1,icps,icps*kfreqmod1,kindex1*kenvindx1,1
afm2  foscili    iamp,1,icps,icps*kfreqmod2,kindex2*kenvindx2,1
afm3  foscili    iamp,1,icps,icps*kfreqmod3,kindex1*kenvindx3,1
afm4  foscili    iamp,1,icps,icps*kfreqmod4,kindex2*kenvindx4,1

kdeclick  linsegr    0,.1,1,.3,.5,.2,0

aout  sum    afm1,afm2,afm3,afm4

outs  aout/2*kdeclick,aout/2*kdeclick

endin

instr 3      ;random adsr per indice mod

if      gksn1 = 1      then

iat    random      0.01,1
idec   random      0.01,1
isus   random      0.01,1
irel   random      0.01,1
iat1   random      0.01,1
idec1  random      0.01,1
isus1  random      0.01,1
irel1  random      0.01,1
iat2   random      0.01,1
idec2  random      0.01,1
isus2  random      0.01,1
irel2  random      0.01,1

```

```

iat3 random      0.01,1
idec3 random     0.01,1
isus3 random     0.01,1
irel3 random     0.01,1

FLsetVal_i iat ,gih1
FLsetVal_i idec,gih2
FLsetVal_i isus,gih3
FLsetVal_i irel,gih4
FLsetVal_i iat1 ,gih5
FLsetVal_i idec1,gih6
FLsetVal_i isus1,gih7
FLsetVal_i irel1,gih8
FLsetVal_i iat2 ,gih9
FLsetVal_i idec2,gih10
FLsetVal_i isus2,gih11
FLsetVal_i irel2,gih12
FLsetVal_i iat3 ,gih13
FLsetVal_i idec3,gih14
FLsetVal_i isus3,gih15
FLsetVal_i irel3,gih16
endif

endin

instr 4      ;random indice,freqmod

if      gksn2 =      1      then
icar  random      .5,4
imod1 random     1,20
imod2 random     1,20
imod3 random     1,20
imod4 random     1,20
indx1 random      0,4
indx2 random      0,4

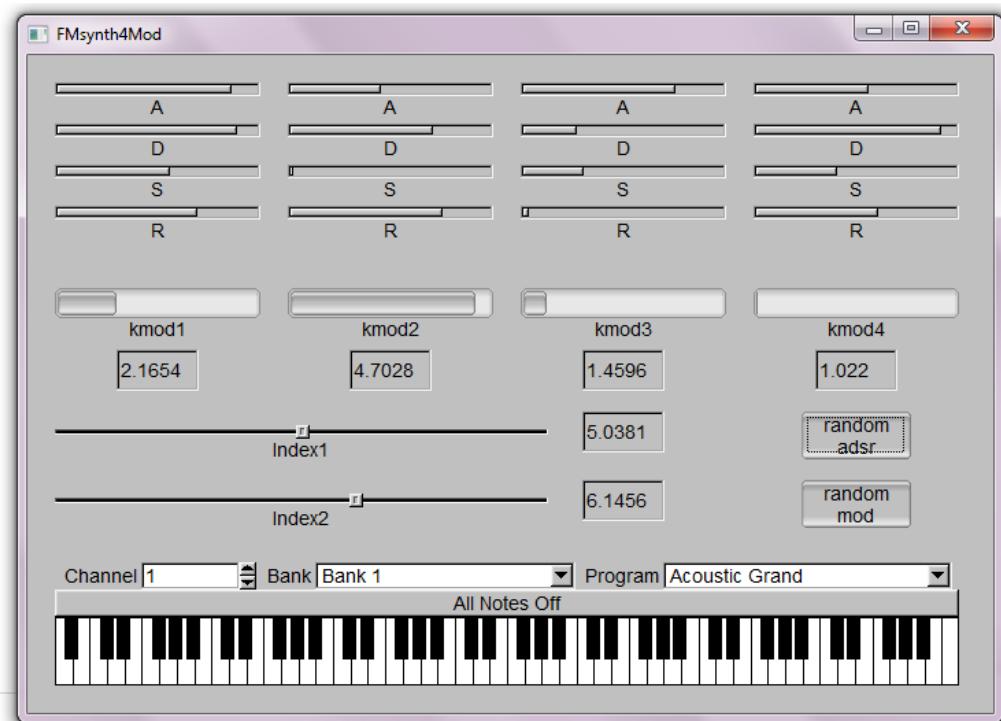
FLsetVal_i imod1,gimod1
FLsetVal_i imod2,gimod2
FLsetVal_i imod3,gimod3
FLsetVal_i imod4,gimod4
FLsetVal_i indx1 ,giindx1
FLsetVal_i indx2 ,giindx2
endif

endin

</CsInstruments>
<CsScore>

f1      0      65536 10      1
f0      3600

</CsScore>
</CsoundSynthesizer>
```



Phase modulation

La modulazione di fase è una tecnica storicamente più recente della sintesi Fm e ha trovato applicazione nel celebre sintetizzatore Yamaha Dx7, mentre nella sintesi Fm il modulatore modula la frequenza di un oscillatore, nella sintesi Pm il modulatore modula la fase di un carrier (la fase è il momento in cui la forma d'onda viene emessa, si parla di fase quando la sua durata è inferiore a quella del periodo). La phase modulation viene preferita alla sintesi Fm per una maggior precisione nell'intonazione.

Negli esempi dei capitoli precedenti abbiamo utilizzato l'opcode **table** per accedere ad un insieme di dati contenuti in una tabella, utilizzando la coppia **phasor-table** si ottiene un vero e proprio oscillatore, l'indice di table sarà determinato da **phasor** (in grado di effettuare un movimento normalizzato di fase), per questo tipo di operazione verrà utilizzata la variante **tablei** (versione interpolata), l'esempio seguente genera una sinusoide di 440 Hz :

```

iamp = 1
ifreq = 440
aphase phasor ifreq
aosc tablei aphase,1,1
out osc*iamp

<CsScore>

f1 0 16384 10 1 ;Sine wave per aosc
i1 0 4

</CsScore>

```

Aggiungendo un oscillatore modulante al parametro aphase otteniamo un semplice esempio di modulazione di fase, una delle tecniche più tipiche di questo tipo di sintesi è quello di realizzare un circuito circolare con feedback, in pratica il carrier viene usato anche come modulatore, un esempio possibile di phase modulation potrebbe essere la seguente orchestra :

Esempio Cap.7.5_phase_modulation (basato sull'esempio di *Alex Hofmann in Csound Floss manual*)

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

icar = p5 ;freq carrier
iamp = p4 ;ampiezza
kmodfreq linseg p6/2,p3/2,p6,p3/2,p6/2 ;freq modulazione
kindex linseg 0,p3/2,p7,p3/2,0 ;indice di modulazione
kfeedback = p8 ;feedback
amod oscili kindex,kmodfreq,1 ;oscillatore modulante
aphase phasor icar ;generatore di frequenza per tablei
acarrier init 0 ;inizializzo il carrier per il feedback
acarrier tablei aphase+amod+(acarrier*kfeedback),1,1,0,1 ;car+mod+feebk
kenv adsr 0.01,.9,1,0.6 ;inviluppo d'ampiezza
out (acarrier*iamp)*kenv ;uscita audio

endin

</CsInstruments>
<CsScore>

f1 0 16384 10 1

;p4 = ampiezza
;p5 = carrier
;p6 = modulation
;p7 = indice di modulazione
;p8 = feedback

i1 0 4 .6 440 100 2 0
s
i1 0 4 .6 440 440 6 .1
s

</CsScore>
</CsoundSynthesizer>
```

Vediamo un esempio ispirato ai sintetizzatori Yamaha, un gruppo di quattro operatori Pm (come il modello Tx81z, a differenza di quest'ultimo, il nostro esempio è basato su forme d'onda unicamente sinusoidali) collegati tra loro per mezzo di un semplice algoritmo che ne controlla il routing.

Esempio Cap.7.6_4phasemod

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1 ;Phase modulation instrument with 4 operators

icarfreq = p5
iamp = p4

acar init 0 ;init car1 per feedback
acar2 init 0 ;init car2 per feedback
acar3 init 0 ;init car3 per feedback
acar4 init 0 ;init car4 per feedback

kfeed randomi 0,p6,2 ;feedback

;modulanti
kmodfreq = p7 ;mod. freq.
kmodfreq2 = p8
kmodfreq3 = p9
kmodfreq4 = p10

;indice di modulazione
kindex = p11
kindex2 = p12
kindex3 = p13
kindex4 = p14

;4 operatori

;OP1
kenvindex adsr 1,.2,.7,.2
amodulator oscili kindex*kenvindex,kmodfreq*icarfreq,1
aphase phasor icarfreq
acar tablei aphase+amodulator+(acar*kfeed),1,1,0,1

;OP2
kenvindex2 adsr .01,.4,.7,.6
amodulator2 oscili kindex2*kenvindex2,kmodfreq2*icarfreq,1
aphase phasor icarfreq
acar2 tablei aphase+amodulator2+(acar2*kfeed),1,1,0,1

;OP3
kenvindex3 adsr 1,.2,.9,1
amodulator3 oscili kindex3*kenvindex3,kmodfreq3*icarfreq,1
aphase phasor icarfreq
acar3 tablei aphase+amodulator3+(acar3*kfeed),1,1,0,1

;OP4
kenvindex4 adsr 0.4,.2,1,1
amodulator4 oscili kindex4*kenvindex4,kmodfreq4*icarfreq,1
aphase phasor icarfreq
acar4 tablei aphase+amodulator4+(acar4*kfeed),1,1,0,1

kalgo = p15 ;sceglie l'algoritmo

;Algorithm1 (OP1)
if kalgo = 1 then ;se kalgo = 1 seleziona una combinazione di
operatori

```

```

aout = acar
;Algorithm2 (OP1+OP2)
elseif kalgo = 2 then
aout = (acar+acar2)/2
;Algorithm3 (OP1+OP3)
elseif kalgo = 3 then
aout = (acar+acar3)/2
;Algorithm4 (OP1+OP4)
elseif kalgo = 4 then
aout = (acar+acar4)/2
;Algorithm5 (OP1+OP2+OP3)
elseif kalgo = 5 then
aout = (acar+acar2+acar3)/3
;Algorithm6 (OP2+OP3+OP4)
elseif kalgo = 6 then
aout = (acar2+acar3+acar4)/3
;Algorithm7 (OP1+OP2+OP3+OP4)
elseif kalgo = 7 then
aout = (acar+acar2+acar3+acar4)/4
endif
;random pan
krndpan randomi 0,1,4
aR,aL pan2 aout*iamp,krndpan
kenvamp adsr 1,1,.2,1
outs aR*kenvamp,aL*kenvamp
endin

```

```

</CsInstruments>
<CsScore>
;p4 = amp
;p5 = freq
;p6 = feedback
;p7,p8,p9,p10 = modulante
;p11,p12,p13,p14 = indice di modulazione
;p15 = sceglie l'algoritmo

```

```
f1 0 16384 10 1
```

```
i1 0 5 .8 110 .09 1 2 .1 .2 .1 .2 .3 .4 1
s
```

```
i1 0 5 .8 110 .09 1 2 .1 .2 1.1 1.2 1.3 1.4 2
s
```

```
i1 0 5 .8 110 .09 2 4 6 8 1.1 1.2 1.3 1.4 3
s
```

```
i1 0 5 .8 110 .09 9 8 7 6 2 2 2 2 7
s
```

```
</CsScore>
</CsoundSynthesizer>
```

Esempio Cap.7.6_4phasemod(gui)

```
<CsoundSynthesizer>
```

```

<CsOptions>
-+rtmidi=virtual -M0
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

FLcolor 255,255,255
FLpanel "Phase modulation instrument with 4 operators",650,300,50,50,4,-1,1
FLvkeybd "keyboard.map",620,90,15,200
gkalgo,gihalgo FLcount "Algorithms",1,7,1,7,0,200,40,20,20,-1

gkmod1,gimod1    FLslider   "mod1",.1,10,-1,5,-1,355,20,243,20
gkmod2,gimod2    FLslider   "mod2",.1,10,-1,5,-1,355,20,243,60
gkmod3,gimod3    FLslider   "mod3",.1,10,-1,5,-1,355,20,243,100
gkmod4,gimod4    FLslider   "mod4",.1,10,-1,5,-1,355,20,243,140

gkfeed,gifeed    FLknob     "feedback",0,.1,0,1,-1,60,20,80

gkrnd,girnd FLbutton   "random",1,0,21,80,35,110,90,0,2,0,-1

FLpanelEnd

FLpanel "",320,370,720,50,4,-1,1

gkindx1,giindx1  FLslider   "index1",.1,2,0,5,-1,290,20,12,20
gkindx2,giindx2  FLslider   "index2",.1,2,0,5,-1,290,20,12,60
gkindx3,giindx3  FLslider   "index3",.1,2,0,5,-1,290,20,12,100
gkindx4,giindx4  FLslider   "index4",.1,2,0,5,-1,290,20,12,140

gkatt,giatt FLslider   "attack",0.01,1,0,21,-1,290,20,12,200
gkdec,gidec FLslider   "decay",0.01,1,0,21,-1,290,20,12,240
gksus,gisus FLslider   "sustain",0.01,1,0,21,-1,290,20,12,280
gkrel,girel FLslider   "release",0.01,1,0,21,-1,290,20,12,320

FLpanelEnd
FLrun

instr 1 ;Phase modulation instrument with 4 operators

icarfreq  cpsmidi      ;midi control
iamp  ampmidi     .6

acar init 0      ;init car1 per feedback
acar2 init 0     ;init car2 per feedback
acar3 init 0     ;init car3 per feedback
acar4 init 0     ;init car4 per feedback

kfeed      =      gkfeed      ;feedback

;OP1
kmodfreq   =      gkmod1      ;mod freq
kindex      =      gkindx1      ;indice modulazione
kenvindex   mxadsr     1,.2,.7,.2
amodulator oscili     kindex*kenvindex,kmodfreq*icarfreq,1
aphase      phasor     icarfreq
acar  tablei    aphase+amodulator+(acar*kfeed),1,1,0,1

;OP2

```

```

kmodfreq2 = gkmod2 ;mod freq
kindex2 = gkindex2 ;indice modulazione
kenvindex2 mxadsr .1,.4,.7,.6
amodulator2 oscili kindex2*kenvindex2,kmodfreq2*icarfreq,1
aphase phasor icarfreq
acar2 tablei aphase+amodulator2+(acar2*kfeed),1,1,0,1

;OP3
kmodfreq3 = gkmod3 ;mod freq
kindex3 = gkindex3 ;indice modulazione
kenvindex3 mxadsr 1,.2,.9,1
amodulator3 oscili kindex3*kenvindex3,kmodfreq3*icarfreq,1
aphase phasor icarfreq
acar3 tablei aphase+amodulator3+(acar3*kfeed),1,1,0,1

;OP4
kmodfreq4 = gkmod4 ;mod freq
kindex4 = gkindex4 ;indice modulazione
kenvindex4 mxadsr 0.4,.2,1,1
amodulator4 oscili kindex4*kenvindex4,kmodfreq4*icarfreq,1
aphase phasor icarfreq
acar4 tablei aphase+amodulator4+(acar4*kfeed),1,1,0,1

kalgo = gkalgo ;sceglie l'algoritmo

;Algorithm1 (OP1)
if kalgo = 1 then ;se kalgo = 1 seleziona una combinazione di
operatori
aout = acar
;Algorithm2 (OP1+OP2)
elseif kalgo = 2 then
aout = (acar+acar2)/2
;Algorithm3 (OP1+OP3)
elseif kalgo = 3 then
aout = (acar+acar3)/2
;Algorithm4 (OP1+OP4)
elseif kalgo = 4 then
aout = (acar+acar4)/2
;Algorithm5 (OP1+OP2+OP3)
elseif kalgo = 5 then
aout = (acar+acar2+acar3)/3
;Algorithm6 (OP2+OP3+OP4)
elseif kalgo = 6 then
aout = (acar2+acar3+acar4)/3
;Algorithm7 (OP1+OP2+OP3+OP4)
elseif kalgo = 7 then
aout = (acar+acar2+acar3+acar4)/4
endif
;random pan
krndpan randomi 0,1,4
aR,aL pan2 aout*iamp,krndpan
kenvamp mxadsr i(gkatt),i(gkdec),i(gksus),i(gkrel)
outs aR*kenvamp,aL*kenvamp
endin

instr 2 ;random

if gkrnd = 1 then
;index
indx1 random .1,2
indx2 random .1,2
indx3 random .1,2
indx4 random .1,2

```

```

FLsetVal_i  indx1,giindx1
FLsetVal_i  indx2,giindx2
FLsetVal_i  indx3,giindx3
FLsetVal_i  indx4,giindx4
;mod
imod1 random      .1,10
imod2 random      .1,10
imod3 random      .1,10
imod4 random      .1,10
FLsetVal_i  imod1,gimod1
FLsetVal_i  imod2,gimod2
FLsetVal_i  imod3,gimod3
FLsetVal_i  imod4,gimod4
;adsr
iatt  random      .01,1
idec  random      .01,1
isus  random      .01,1
irel  random      .01,1
FLsetVal_i  iatt,giatt
FLsetVal_i  idec,gidec
FLsetVal_i  isus,gisus
FLsetVal_i  irel,girel
endif

endin

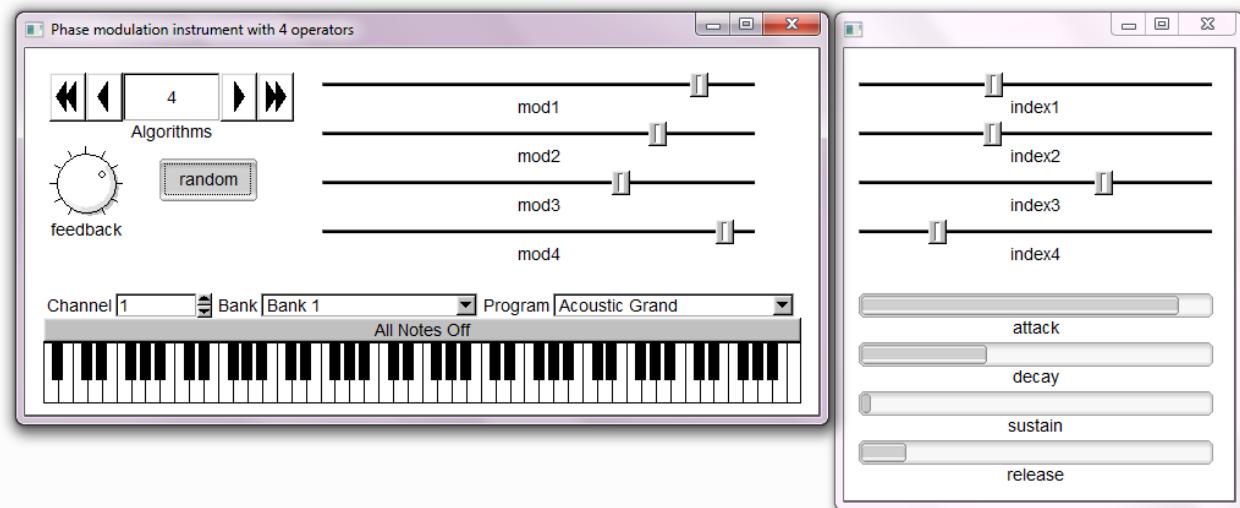
</CsInstruments>
<CsScore>

f1      0      16384 10      1
i1      0      3600

</CsScore>

</CsoundSynthesizer>

```



Waveshape

La waveshape o sintesi per distorsione non lineare è una tecnica in cui un segnale viene alterato dal punto di vista della sua componente armonica, una forma di distorsione non lineare è anche il clipping che si verifica quando un segnale ha dei picchi che superano il massimo volume permesso dal segnale, il risultato è una distorsione del segnale che causa disturbo.

La waveshape è invece un tipo di distorsione in cui si agisce sul contenuto armonico del segnale alterandone il timbro, la waveshape interviene sull'ampiezza di un suono cambiandone la forma d'onda.

Il prossimo strumento utilizza l'opcode table per leggere una forma d'onda distorcere, un segnale in ingresso verrà usato come indice per la tabella :

Esempio Cap.7.7_waveshape

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

gidist ftgen 0,0,257,9,.5,1,270,1.5,.33,90, \
2.5,.2,270,3.5,.143,90,4.5,.111,270

instr 1

a1    pluck 10000,220,440,0,1 ;segnale da distorcere
adist table a1,gidist,1 ;tabella con funzione gidist
out   adist*10000 ;uscita audio

endin

</CsInstruments>
<CsScore>

i1    0      20

</CsScore>
</CsoundSynthesizer>
```

Csound offre alcuni opcode nati per questo utilizzo specifico, un semplice esempio di distorsione non lineare si ottiene con l'opcode **powershape** :

```
aout powershape ain,kShapeAmount
```

ain prende un segnale in ingresso, kShapeAmount è il parametro che indica la quantità di saturazione (con range preferibile tra 0 e 20).

Esempio Cap.7.8_powershape

```
<CsoundSynthesizer>
<CsOptions>
```

```

</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1

kshape      linseg      p5,p3/2,p6,p3/2,p5      ;inviluppo della distorsione
aosc  oscili      10000,cpspch(p4),1      ;oscillatore
aout powershape aosc,kshape ;distorsione del segnale
out   aout

endin

</CsInstruments>
<CsScore>

f1    0    4096  10    1
;il segnale viene distorto nell'unità di tempo passando da 0.1 a 2.3
i1    0    22    6.00  0.1    2.3

</CsScore>
</CsoundSynthesizer>

```

Il prossimo esempio utilizza l'opcode **distort** per simulare un tipo di distorsione per chitarra elettrica, al segnale in ingresso è stato aggiunto un generatore di rumore bianco per ricreare il rumore tipico degli amplificatori valvolari o a transistor.

Esempio Cap.7.9_distort

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 1
nchnls = 1

gifn ftgen 0,0,257,9,.5,1,270,1.5,.33,90,2.5,.2,270,3.5,.143,90,4.5,.111,270
instr 1

asig  pluck 10000,80,220,0,1 ;corda pizzicato
awsh  distort      asig,p4,gifn      ;distorsione
anoise      rand  100    ;rumore bianco
adeclick  linseg      0,0.02,1,p3-0.05,1,0.02,0,0.01,0;inviluppo declick
aout  =      (awsh+anoise)*adeclick ;suono distort sommato a rumore
out   aout

endin

```

```
</CsInstruments>
<CsScore>
```

```
;p4 = quantità di distorsione
```

i1	0	1	0
i1	2	1	0.2
i1	4	1	0.4
i1	6	1	0.6
i1	8	1	0.8
i1	10	1	1

```
</CsScore>
</CsoundSynthesizer>
```

Cap.8 Sample Playback

8.1 Lettura con loscil,diskin2

8.2 Loop player

8.3 soundfont

Lettura con loscil,diskin2

I moduli per leggere ed elaborare un campione audio in Csound sono principalmente tre :

loscil : legge un campione da una tabella (Gen01) ,non permette la lettura di file superiori alla durata di un minuto,questo opcode memorizza il campione audio in lettura nella *RAM*.

diskin2 : permette diverse operazioni sul file in lettura,adatto anche per file di grandi dimensioni,diskin2 apre un campione per mezzo di una lettura diretta dall'hard disk.

soundin: molto semplice,non permette operazioni di editing

Sintassi di **loscil** (di cui esiste anche una variante **loscil3** interpolata)

```
ar1 [,ar2] loscil xamp, kcps, ifn (seguono parametri opzionali)
```

questo modulo può leggere file mono e stereo (è necessario indicare un solo ingresso “ar1” per un file mono). I suoi parametri sono nell’ordine : ampiezza,velocità di lettura e numero di tabella contenente il campione,seguono vari parametri riguardanti la modalità loop,la frequenza base iniziale e molto altro,si rimanda alla documentazione ufficiale per ulteriori approfondimenti.

Vediamo un esempio completo di **loscil** con velocità di lettura variabile,filtraggio,vibrato (un nuovo opcode simile ad un oscillatore a bassa frequenza ma che produce delle continue micro-variazioni random,adatto per “naturalizzare” la freddezza e staticità di un suono elettronico).

Esempio Cap.8.0_loscil

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

turnon      2
gasig1      init  0      ;inizializzo le variabili globali
gasig2      init  0

instr 1

iamp =      p4
kpitch     oscil      p5,p6,5 ;varia la velocità di lettura
ifn =       1
;vibrato
kaverageamp =      4
kaveragefreq =      kpitch
kvib vibr   kaverageamp,kaveragefreq,5
;lettura samples
a1,a2 loscil    iamp,kpitch+kvib,ifn,p7

;bandpass filter
kcutoff     randomi   100,14000,10      ;curva random per filtro
kband =     1000 ;larghezza di banda
afilt1     butterbp   a1,kcutoff,kband ;filtro passabanda
afilt2     butterbp   a2,kcutoff,kband
```

```

aout1 balance      afilt1,a1    ;bilanciamento audio dei due segnali
aout2 balance      afilt2,a2
;uscita
kenv   linseg     0, 0.02, 1, p3 - 0.05, 1, 0.02, 0, 0.01, 0
outs  aout1*kenv,aout1*kenv
vincr gasig1,aout1*kenv ;assegno la variabile al alla variabile globale
vincr gasig2,aout1*kenv

endin

instr 2      ;reverb
;prende in ingresso le variabili globali
aL,aR freverb   gasig1,gasig2,.9,0.35,sr,0
outs  aL,aR
clear gasig1,gasig2      ;azzera la memoria per evitare accumulo

endin

</CsInstruments>
<CsScore>
f1    0      0      1      "loop1.wav"  0      0      0
f2    0      0      1      "loop2.wav"  0      0      0

f5    0      4096   10     1

i1    0      1      1      .1      1      1
s
i1    0      2      1      .2      .5      2
s
i1    0      1      1      .3      .4      1
s
i1    0      2      1      .4      .3      2
s

</CsScore>
</CsoundSynthesizer>
```

Con le ultime release di Csound è anche possibile importare in tabella,sample audio in formato mp3 (mono o stereo),per questa operazione usiamo una funzione specifica chiamata *Gen49* :

f# time size 49 filcod skiptime format

un esempio pratico :

```
f1    0      [131072*8]  49      "test.mp3"  0      1
```

il valore contenuto tra le parentesi quadre indica la grandezza della tabella,tuttavia,anche con valori così alti,
loscil non è in grado di leggere file audio di elevata durata. L'ultimo parametro della *Gen49* indica il numero
di canali del campione audio mp3 :

1 - Mono file	3 - First channel (left)
2 - Stereo file	4 - Second channel (right)

Esempio Cap.8.0_loscil_mp3

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
```

```

<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

;versione mono

instr 1

iamp = p4
kpitch = p5

a1    loscil3    iamp,kpitch,1,1

outs a1,a1

endin

</CsInstruments>
<CsScore>

f1    0      [131072*8]  49      "test.mp3"  0      1

i1    0      10     1      1

</CsScore>
</CsoundSynthesizer>
```

Introduciamo adesso uno degli storici opcode di Csound per la lettura di file audio :

Sintassi di **diskin2** :

```
a1[, a2[, ... a24]] diskin2 ifilcod,kpitch[,iskiptim,iwrap....]
```

- ifilcod : nome del file in lettura
- kpitch : velocità di lettura, con valori negativi il file viene letto al contrario
- iskiptim : punto di inizio lettura del file (di default = 0)
- iwrap : con valori positivi il file viene messo in loop

Esempio Cap.8.1_diskin2

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1
```

```

instr 1
a1    diskin2 "beats.wav",-1,0,1
out   a1
endin

instr 2
kspeed      linseg      .1,p3/2,10,p3/2,.1
a1    diskin2    "beats.wav",kspeed,0,1
out   a1
endin

</CsInstruments>
<CsScore>

i1    0      4
s
i2    0      4
s

</CsScore>
</CsoundSynthesizer>
```

Prendiamo in esame alcune tecniche di utilizzo di **diskin2** molto pratiche, proviamo ad utilizzarlo come un piccolo campionatore controllando l'intonazione di un campione da una tastiera midi :

Esempio Cap.8.2_sample player

```

<CsoundSynthesizer>
<CsOptions>
-+rtmidi=virtual -M1
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

ifrq  cpsmidi      ;attiva controllo midi
a1,a2 diskin2    "strings.wav",ifrq/127
aenv  linsegr     0.0 ,0.0015,1.0,p3,1.0,0.002,0.0
outs  a1*aenv,a2*aenv

endin

</CsInstruments>
<CsScore>

f0    3600

</CsScore>
</CsoundSynthesizer>
```

come abbiamo potuto notare l'esempio presenta un problema legato al tipo di opcode, il pitch midi controlla anche la velocità di lettura e quindi il file viene rallentato nel registro basso e accelerato nel range acuto, una possibile soluzione può essere quella di assegnare un campione specifico ad un preciso tasto della tastiera (immaginiamo di disporre del campionamento completo di un violino, con singoli file audio per ogni

nota, come nelle moderne librerie orchestrali). Il prossimo esempio utilizza **notnum** che restituisce il valore del tasto midi premuto.

Esempio Cap.8.3_sample player2

```
<CsSoundSynthesizer>
<CsOptions>
-M1 --rtmidi=virtual
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

inote notnum      ; invia il numero di nota midi (da 0 a 127)

if    inote =      60    then ; se inote = 60 attiva "808CLAP.WAV"
asig  diskin2      "samples/808CLAP.WAV",1
elseif   inote =      61    then
asig  diskin2 "samples/LIVE.WAV",1
elseif   inote =      62    then
asig  diskin2 "samples/LONGTRIANGLE.WAV",1
elseif   inote =      63    then
asig  diskin2 "samples/PANNER.WAV",1
elseif   inote =      64    then
asig  diskin2 "samples/RINGER.WAV",1
elseif   inote =      65    then
asig  diskin2 "samples/SLIDE.WAV",1
elseif   inote =      66    then
asig  diskin2 "samples/STEREOCONGA.WAV",1
elseif   inote =      67    then
asig  diskin2 "samples/TOMBASH.WAV",1
endif

; mette in mute lo strumento al di fuori del range (60-67) definito
; per eliminare un fastidioso problema audio
if    inote <      60    then
turnoff
elseif   inote >      67    then
turnoff
endif

outs  asig, asig

endin

</CsInstruments>
<CsScore>

f0    3600

</CsScore>
</CsSoundSynthesizer>
```

Loop player

diskin2 è la versione aggiornata di uno degli storici opcode di Csound (nelle versioni precedenti esisteva l'opcode **diskin**); dalla versione di Csound5 ad oggi sono stati introdotti numerosi nuovi opcode per il sampling, **flooper2** è simile a diskin2 ma permette di definire segmenti del file in lettura con assoluta precisione, è inoltre più adatto per la lettura dei file in loop grazie alla modalità di crossfade :

```
asig flooper2 kamp,kpitch,kloopstart,kloopend,kcrossfade,ifn \
[, istart,imode,ifenv,iskip]
```

Come potete osservare non vi è nessun parametro relativo al file in lettura, questo perché la lettura del file avviene per mezzo di una tabella del tipo :

```
f1      0      0      1      "beats.wav"  0      0      0
```

Esempio Cap.8.4_flooper2

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

kamp = .6 ;ampiezza
kpitch = 1 ;velocità lettura
kstart = 0 ;punto di inizio lettura file
kend = 1 ;punto finale lettura file
kcross = 0.005 ;crossfade per il loop
ifn = 1 ;tabella con il file da aprire
imode = p4 ;con imode = 1 il file viene letto al contrario

aout flooper2 kamp,kpitch,kstart,kend,kcross,ifn,0,imode

out aout
endin

</CsInstruments>
<CsScore>

f1      0      0      1      "beats.wav"  0      0      0

i1      0      4      0
s
i1      0      4      1
s

</CsScore>
</CsoundSynthesizer>
```

una semplice variante con i suoi parametri random :

Esempio Cap.8.4B_flooper2

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1
kamp = .6 ;ampiezza
kpitch randomi .5,2,1 ;velocità lettura
kstart randomh 0,1,4 ;punto iniziale
kend randomh 1,2,4 ;punto finale
kcross = 0.005 ;crossfade per il loop
ifn = 1 ;tabella con il file da aprire

aout1 flooper2 kamp,kpitch,kstart,kend,kcross,ifn,0,0 ;lettura normale
aout2 flooper2 kamp,kpitch,kstart,kend,kcross,ifn,0,1 ;lettura contrario

outs aout1,aout2

endin

</CsInstruments>
<CsScore>

f1 0 0 1 "beats.wav" 0 0
i1 0 20 0
s

</CsScore>
</CsoundSynthesizer>
```

Adesso una gui per poter editare il file in tempo reale :

Esempio Cap.8.5_flooper2(gui)

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2
0dbfs = 1

FLpanel "Playback soundfiles",550,190,300,300,4,-1,1

gk1,ihb1 FLbutton "@+4>",1,0,21,100,30,20,20,0,1,0,-1
gk2,ihb2 FLbutton "@+1square",1,0,21,100,30,20,60,0,1,0,0
gk3,ihb3 FLbutton "Backward",1,0,3,100,30,20,100,0,1,0,-1
gk4,ihb4 FLbutton "Exit",1,0,21,100,30,20,140,0,2,0,0

gkpitch,ih1 FLslider "Pitch",.1,3,0,5,-1,355,30,143,20
```

```

gkstart,ih2 FLslider      "Loop-start",.01,5,0,5,-1,355,30,143,70
gkend,ih3   FLslider      "Loop-end",.01,5,0,5,-1,355,30,143,120

FLpanel_end
FLrun

FLsetVal_i 2,ih1
FLsetVal_i 0,ih2
FLsetVal_i 2,ih3

instr 1

;asig flooper2 kamp,kpitch,kloopstart,kloopend,kcrossfade,ifn
aout  flooper2    .6,gkpitch,gkstart,gkend,.0005,1,0,i(gk3)
outs  aout,aout

endin

instr 2

exitnow

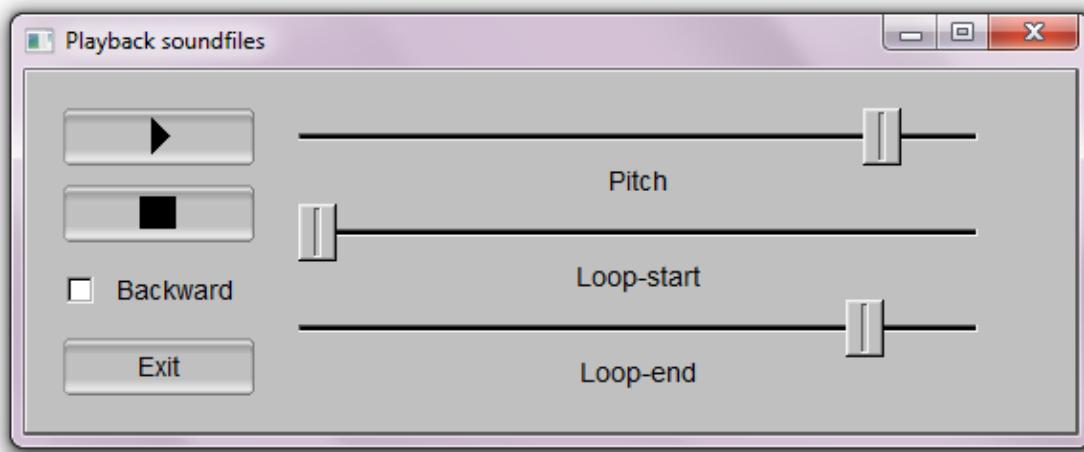
endin

</CsInstruments>
<CsScore>

f1      0      0      1      "drums_loop.aif"  0      0      0
f0      3600

</CsScore>
</CsoundSynthesizer>

```



Il prossimo opcode permette la creazione di loop assai più complessi e ritmici rispetto a flooper2, si tratta di **BBCutm** (esiste in versione mono e stereo), questo particolare opcode permette la creazione di ritmi molto complessi simili a stili elettronici come la drum'n bass.

```
a1 BBCutm asource,ibps,isubdiv,ibarlength,iphrasebars,inumrepeats
```

- asource : segnale da processare
- ibps : tempo da tagliare in battiti per secondo
- isubdiv : suddivisioni per battuta (8 corrisponde ad 1/8 di una battuta di 4/4)
- ibarlength : numero di battiti per battuta
- iphrasebars : lunghezza della frase ottenuta dai tagli

- inumrepeats : numero di ripetizioni della frase

l'esempio seguente utilizza come sorgente per **BBcutm** un sample playback creato con **flooper2** :

Esempio Cap.8.6_BBcut

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2
0dbfs = 1

FLpanel "BBCut mono",620,300,300,300,4,-1,1

gk1,ihb1      FLbutton      "@+4>",1,0,21,100,30,20,20,0,1,0,-1
gk2,ihb2      FLbutton      "@+1square",1,0,21,100,30,20,60,0,1,0,0
gk3,ihb3      FLbutton      "Backward",1,0,3,100,30,20,100,0,1,0,-1
gk4,ihb4      FLbutton      "Exit",1,0,21,100,30,20,140,0,2,0,0

gk5,ihb5      FLcount "ibps",1,120,1,10,11,100,40,40,220,0,1,0,-1
gk6,ihb6      FLcount "isubdiv",1,120,1,10,11,100,40,150,220,0,1,0,-1
gk7,ihb7      FLcount "ibarlength",1,120,1,10,11,100,40,260,220,0,1,0,-1
gk8,ihb8      FLcount "iphrasebars",1,120,1,10,11,100,40,370,220,0,1,0,-1
gk9,ihb9      FLcount "inumrepeats",1,120,1,10,11,100,40,480,220,0,1,0,-1

gkpitch,ih1   FLslider      "Pitch",.1,5,-1,5,-1,455,30,143,20
gkstart,ih2   FLslider      "Loop-start",.01,5,-1,5,-1,455,30,143,70
gkend,ih3    FLslider      "Loop-end",.01,5,-1,5,-1,455,30,143,120

FLpanel_end
FLrun

FLsetVal_i     2,ih1
FLsetVal_i     0,ih2
FLsetVal_i     5,ih3

FLsetVal_i     4,ihb5
FLsetVal_i     16,ihb6
FLsetVal_i     4,ihb7
FLsetVal_i     4,ihb8
FLsetVal_i     4,ihb9
FLsetVal_i     1,ihb2

instr 1

;asig flooper2 kamp, kpitch, kloopstart, kloopend, kcrossover, ifn
asource flooper2      .6,gkpitch,gkstart,gkend,.0005,1,0,i(gk3)

;a1 BBCUTM asource, ibps, isubdiv, ibarlength, iphrasebars, inumrepeats
aout    BBCUTM asource,i(gk5),i(gk6),i(gk7),i(gk8),i(gk9)
outs    aout,aout

endin
instr 2
exitnow
```

```

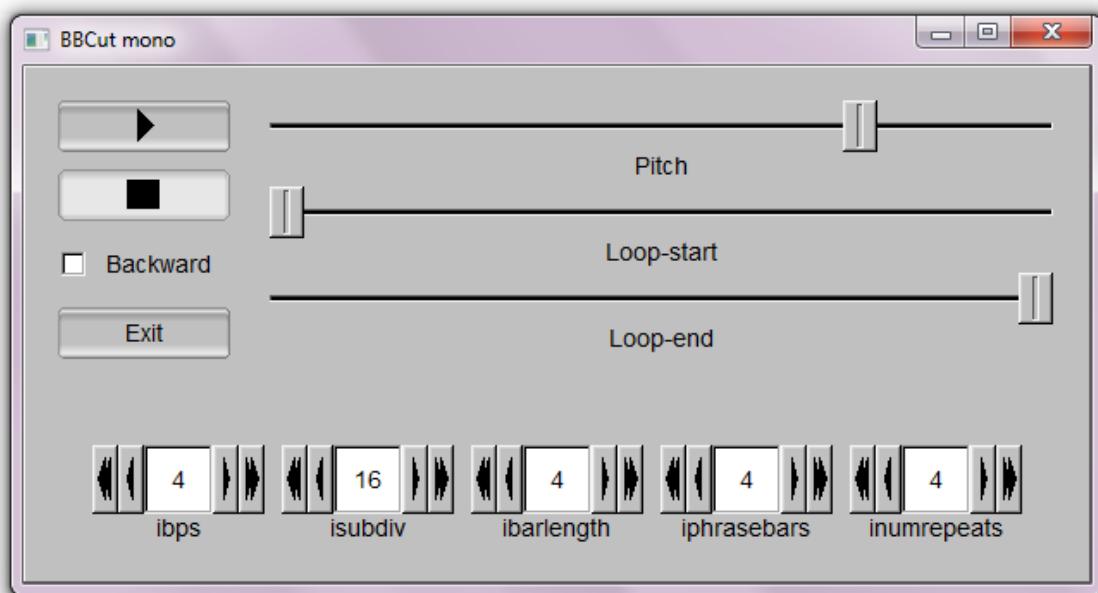
endin

</CsInstruments>
<CsScore>

f1      0      0      1      "A strum 92 bpm.wav"  0      0      0
f0      3600

</CsScore>
</CsoundSynthesizer>

```



Lettura SoundFont

I SoundFont sono un formato proprietario di E-mu Systems riconducibile alla Sample-based synthesis, la tecnica è quella di campionare strutture di sintesi come additiva e sottrattiva (per esempio una saw filtrata con un lowpass filter) o strumenti acustici. Non ci addentreremo in quelli che sono gli utilizzi di questo formato nell'industria del software, ma affronteremo i vantaggi dell'integrazione di questo formato all'interno di Csound, tra tutti quello di poter disporre di una immensa libreria di SoundFont free disponibile online che va dal campionamento di sintetizzatori fino agli strumenti acustici di vario tipo; dal punto di vista creativo è interessante poter costruire uno strumento Csound con un generatore sonoro complesso (un SoundFont di synth lead, bass o pad) da utilizzare nelle proprie composizioni Dsp-based, sottoponendolo a trasformazioni timbriche.

Il motore in grado di leggere SoundFont è fluidsynth, per aprire un'istanza di questo motore utilizziamo l'opcode **fluidEngine**.

Vediamo come suonare un SoundFont in tempo reale tramite un controller midi, l'istanza di fluidsynth va dichiarata in modo globale in orchestra prima della scrittura degli strumenti.

Esempio Cap.8.8_SoundFont

```

<CsoundSynthesizer>
<CsOptions>
-M1 -+rtmidi=virtual
</CsOptions>
<CsInstruments>

```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
Odbfs = 1

giengine fluidEngine ;apre il motore di fluidsynth
isfnum fluidLoad "HS Strings.sf2",giengine,1 ;legge il SoundFont
fluidProgramSelect giengine,1,isfnum,0,0 ;seleziona un preset dal
SoundFont aperto

instr 1

midinoteonkey p4,p5 ;opcode con doppia funzione di controllo midi e score
;fluidNote giengine,1,ikeyivel
fluidNote giengine,1,p4,p5 ;suona una nota sul canale 1 del fluidsynth

endin

instr 2

ivol init 1 ;volume globale
asig1,asig2 fluidOut giengine ;uscita audio di fluidsynth
outs asig1*ivol,asig2*ivol

endin

</CsInstruments>
<CsScore>

i1 0 3600
i2 0 3600

</CsScore>
</CsoundSynthesizer>

```

Una variazione dell'esempio appena creato permette la creazione di loop all'interno di un buffer in memoria,**sndloop** è un opcode specifico per la registrazione e il sample playback del campione registrato :

```
aout,krec sndloop ain,kpitch,ktrig,idur,ifad
```

- ain : suono in ingresso
- krec : segnale di recording on
- kpitch : velocità di lettura del buffer registrato
- ktrig : con valore = 1 attiva la registrazione
- idur : durata del buffer
- ifad : dissolvenza per il file creato in loop
-

Esempio Cap.8.9_sndloop

```
<CsoundSynthesizer>
<CsOptions>
-M1 --rtmidi=virtual
</CsOptions>
<CsInstruments>
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

giengine fluidEngine ;apre il motore di fluidsynth
isfnum fluidLoad "HS Strings.sf2",giengine,1 ;legge il SoundFont
fluidProgramSelect giengine,1,isfnum,0,0 ;seleziona un preset dal
SoundFont aperto
;panel
FLpanel "Sndloop",580,240,300,300,4,-1,1

FLvkeybd "keyboard.map",520,100,35,120

gk2,ihb2 FLbutton "REC On/Off",1,0,2,100,30,20,20,0,4,0,-1
gk4,ihb4 FLbutton "Exit",1,0,21,100,30,20,60,0,3,0,0

ifreq1 FLvalue "",50,38,515,25
gkfreq,ih1 FLslider "pitch",.1,10,-1,5,ifreq1,355,30,143,20

FLpanel_end
FLrun

FLsetVal_i 1,ih1
;end panel

garec init 0 ;inizializza variabile loop in registrazione

instr 1

p4 init 0
p5 init 0
midinoteonkey p4,p5 ;opcode con doppia funzione di controllo midi e score

;fluidNote giengine,1,ikeyivel
fluidNote giengine,1,p4,p5 ;suona una nota sul canale 1 del fluidsynth

endin

instr 2

ivol init 1 ;volume globale
asig1,asig2 fluidOut giengine ;uscita audio di fluidsynth
aout1 = asig1*ivol
aout2 = asig1*ivol
outs aout1,aout2
garec = aout1+aout2+garec

endin

instr 4

idur = 4 ;durata della sequenza in registrazione
icrossfade = 0.05
kpitch = gkfreq ;velocità di lettura del campione registrato
ktrig = gk2 ;attiva la registrazione
arec,krec sndloop garec,kpitch,ktrig,idur,icrossfade

;segna che indica la modalità "rec on" visualizzata nella console

printk 1,krec

```

```

outs  arec,arec

garec =      0      ;azzera variabile rec per evitare accumulo

endin

instr 3

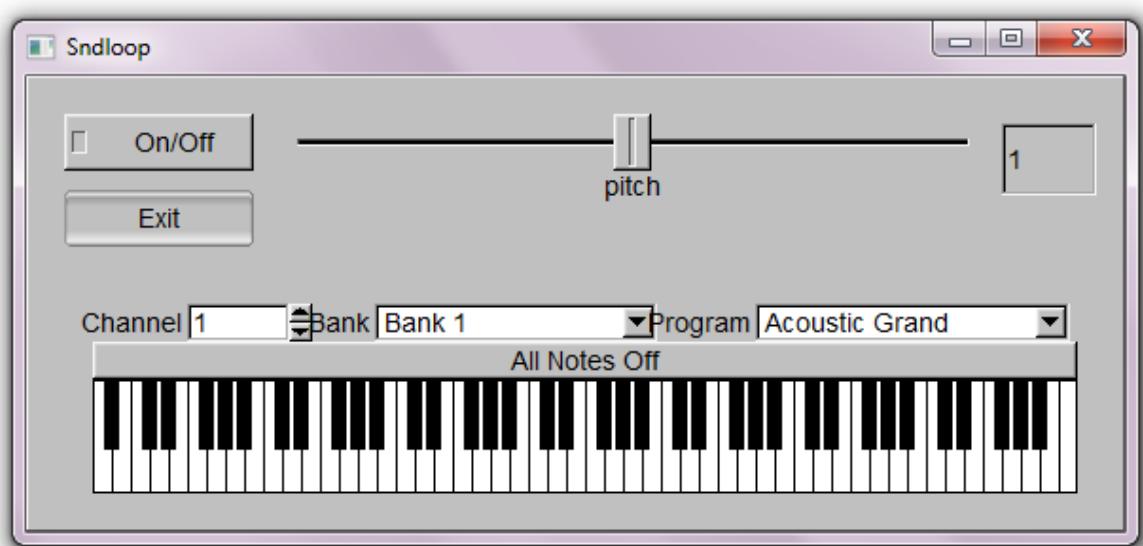
exitnow

endin

</CsInstruments>
<CsScore>

i1    0    3600
i2    0    3600
i4    0    3600

</CsScore>
</CsoundSynthesizer>
```



Questa semplice gui permette di registrare un frammento della performance midi della durata di 4 secondi, il suono registrato viene messo in play ed è possibile controllare la velocità di lettura, il pulsante On/Off attiva e disabilità la modalità di recording/sample playback.

Cap.9 Sintesi granulare

9.1 Basic grain

9.2 syncgrain, syncloop

9.3 partikkel granular synthesizer

9.4 vocal synthesis

Basic grain

La sintesi granulare è una tecnica molto complessa basata sul concetto di “grano”,ossia di una piccola entità della durata compresa in genere tra i 5 ms e i 100 ms,con un suo inviluppo d’ampiezza,frequenza,posizione nello spazio.La forma d’onda del singolo grano può essere di vari tipi e le sue caratteristiche vengono controllate da una serie di parametri.

La densità dei singoli grani può essere rarefatta (in cui percepiamo l’intervallo di tempo tra un grano e il successivo,intervallo che può essere costante o variabile anche in modo random) ,con strutture ad alta densità i grani possono anche essere sovrapposti creando una sorta di nuvola granulare.Un suono complesso viene quindi generato utilizzando enormi quantità di unità minuscole.Tra i primi teorici della sintesi granulare vanno ricordati Dennis Gabor e Iannis Xenakis,in seguito un decisivo lavoro di ricerca fu svolto da Curtis Roads e Barry Truax,quest’ultimo fu il primo a realizzare un esempio di sintesi granulare in tempo reale utilizzando un Dsp collegato a un microcomputer dell’epoca (PDP).

Le tecniche per ottenere delle texture granulari sono basate sul processamento di suoni di sintesi o di campioni audio,come possiamo immaginare le difficoltà iniziali di questa sintesi erano legate alla potenza di calcolo dei computer dell’epoca,oggi grazie alla velocità delle moderne cpu l’unico limite è legato solo alla propria fantasia(!?) ed è possibile realizzare complesse texture granulari anche in tempo reale.Ancora oggi i linguaggi di sintesi rappresentano lo strumento ideale per la creazione di strumenti di sintesi granulare,non essendo ancora diventata una tecnica di diffusione commerciale,tuttavia esistono strumenti granulari commerciali come il modulo Malström di Reason e l’ottimo Reaktor della Native Instruments.

I parametri legati alla sintesi granulare sono generalmente :

- intervallo di tempo tra la generazione di un grano con il seguente,determinerà la densità granulare,quando l’intervallo di tempo è costante si definisce sintesi granulare sincrona
- durata dei singoli grani (grain size)
- forma d’onda dei grani
- forma d’onda per l’inviluppo dei grani
- variazioni random della frequenza dei grani
- time stretching
- numero di sovrapposizione dei grani (overlap)
- numero massimo dei grani prodotti per unità di tempo
- segnale di modulazione per la frequenza dei grani (ad esempio con la sintesi Fm)
- trasposizione delle frequenze

Dal punto di vista timbrico possiamo ottenere una texture molto “pulviscolare” e granulosa con alta densità e durata dei singoli grani su valori molto bassi vicini ai 5 ms.Nel caso della granulazione di un suono reale preregistrato,possiamo avvicinarsi al suono quasi inalterato lavorando invece con valori di durata del grano molto alti (vicino ai 100 ms).

La complessità di questa tecnica risiede quindi nella costruzione di metodi per generare eventi sonori piccolissimi e la loro disposizione temporale,per costruire uno strumento granulare in Csound per prima cosa ci servirà un generatore sonoro,una semplice sinusoida con inviluppo d’ampiezza e frequenza definita,ancora più interessante è utilizzare un generatore di frequenze random ,ad esempio :

```
instr 2
k1      =      rnd(1000)    ;generatore numeri casuali
a1      poscil     1,60+k1,1  ;oscillatore con frequenza random
```

```

aenv  linseg      0,0.02,1,p3-0.04,1,0.02,0,0.01,0    ;inviluppo
outs  a1*aenv,a1*aenv
endin

```

l'opcode **rnd** genera numeri casuali partendo da un range dato, il numero generato sarà diverso ad ogni istanza dello strumento (ed è proprio quello che ci servirà adesso), uno dei metodi più semplici per generare dei microeventi sonori è quello di costruire un secondo strumento che controllerà la sinusoide di instr 2, uno dei più adatti a questo scopo è **schedwhen** :

schedwhen ktrigger,kinsnum,kwhen,kdur

- ktrigger : il valore = 1 attiva lo strumento
- kinsnum : numero dello strumento da controllare
- kwhen : corrisponde a p2 nella score
- kdur : durata dell'evento

schedwhen controllerà quindi la durata di ogni singolo evento del generatore sonoro, per controllare invece gli intervalli di tempo? uno dei metodi più semplici è quello di reiterare ogni istanza di **schedwhen** per mezzo di **timeout** e **reinit**, il seguente frammento :

```

igrainrate = p4 ;intervallo di tempo tra i singoli grani

play:

timeout 0,igrainrate,continua

reinit play

continua:

```

Esempio Cap.9.1 simple grain.csd

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

igrainrate =      p4      ;intervallo di tempo tra i singoli grani

play:
timeout      0,igrainrate,continua
reinit      play
continua:

ktrigger    =      1      ;zero bypassa, 1 attiva
kinsnum     =      2      ;suona lo strumento numero 2
kwhen =      0      ;p2,inizio dell'evento
kgrainsize  =      p5      ;durata dei grani
schedwhen   ktrigger,kinsnum,kwhen,kgrainsize
endin

```

```

instr 2
k1      =      rnd(1000)    ;generatore numeri casuali
a1      poscil     .4,60+k1,1 ;oscillatore con frequenza random
aenv    linseg     0,0.02,1,p3-0.04,1,0.02,0,0.01,0 ;inviluppo

;random pan
kpan   =      birnd(1)
al,ar pan2  al,kpan,2

outs  al*aenv,ar*aenv

endin

</CsInstruments>
<CsScore>

f1      0      16384 10      1
;p4 = rate
;p5 = grain size

i1      0      5      .9      .1
s
i1      0      5      .8      .09
s
i1      0      5      .7      .08
s
i1      0      5      .6      .07
s
i1      0      5      .2      .06
s
i1      0      5      .1      .05
s
i1      0      5      .09     .04
s
i1      0      5      .03     .04
s
i1      0      5      .02     .04
s

</CsScore>
</CsoundSynthesizer>

```

Abbiamo visto un primo rudimentale approccio alla sintesi granulare, provate a sperimentare con vari settaggi di durata dei grani e velocità degli eventi, oppure utilizzando altre forme d'onda per il segnale granulato.

Csound offre una serie molto ampia di opcode specifici per la sintesi granulare, dai più antichi come **grain** (con le varianti più aggiornate di **grain2** e **grain3**) e **granule** (quest'ultimo è nato per granularizzare campioni audio) fino al recente **partikkel**, quest'ultimo realizza numerose tecniche di sintesi granulare ed è basato sui concetti espressi da *Curtis Roads in Microsound(2001)*.

Introduciamo un opcode relativamente semplice da usare con cui ottenere texture simili all'esempio precedente, **grain2** è un semplice generatore in cui oltre ai parametri di grain-rate e grain-size richiede :

- frequenza e tabella della forma d'onda da granularizzare
- una seconda forma d'onda per l'inviluppo dei singoli grani
- variazione random della frequenza

-numero dei grani sovrapposti

```
ares grain2 kcps,kfmd,kgdur,iovrlp,kfn,iwfn
```

Esempio Cap.9.2_grain2.csd

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

gisine      ftgen 1,0,8192,10,1,.6,.3
ihanning    ftgen 2,0,8192,-20,2      ;Hanning
ibarlett    ftgen 3,0,8192,-20,3      ;Bartlett (Triangle)

instr 1
kcps = p5      ;grain freq
kfmd = p6      ;variazione random in Hz.
kgdur = p7      ;durata dei grani in secondi
iovrlp = 10    ;numero di grani sovrapposti
kfn = 1         ;grain waveform
iwnf1 = 2       ;inviluppo a1
iwnf2 = 3       ;inviluppo a2

a1   grain2      kcps,kfmd,kgdur,iovrlp,kfn,iwnf1
a2   grain2      kcps*2,kfmd,kgdur,iovrlp,kfn,iwnf2

adeclick linseg 0,0.01,1,p3-0.05,1,0.04,0,1,0
outs (a1*adeclick)*p4, (a2*adeclick)*p4

endin

</CsInstruments>
<CsScore>

i1 0 5 .1 220 1 1
s
i1 0 5 .1 220 20 .5
s
i1 0 5 .1 220 40 .1
s
i1 0 5 .1 220 60 .08
s

</CsScore>
</CsoundSynthesizer>
```

Syncgrain

Negli esempi precedenti abbiamo generato texture granulari con suoni di sintesi, un altro degli utilizzi più tipici di questa tecnica è quello di utilizzare campioni di suoni veri come generatore audio per la sintesi granulare, il nuovo opcode **syncgrain** (sviluppato da *Victor Lazzarini*) ha un numero relativamente basso di parametri e risulta molto pratico per questo scopo :

```
asig syncgrain kamp,kfreq,kpitch,kgrsize,kprate,ifun1,ifun2,iolaps
```

il parametro ifun1 legge la tabella contenente il campione audio,**syncgrain** realizza un tipo di sintesi granulare di tipo sincrono,oltre ai parametri di base per il controllo della texture include il time stretching con cui variare la velocità di lettura del file senza alterare l'intonazione.

Esempio Cap.9.3_syncgrain.csd

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 2
0dbfs = 1

instr 1

kamp = p4 ;ampiezza

kfreq randomi .1,.50,1 ;density grain (1 a 100)
kpitch randomi .2,.2,1 ;grain pitch(negative =backwards)
kgrsize randomi .01,.1,.1 ;grain size

kprate1 transeg 0.1, p3*0.25,p5, 1,p3*0.75,p6,0.1
kprate2 transeg 0.1, p3*0.25,p6, 1,p3*0.75,p5,0.1

ifun1 = 1 ;file audio in tabella
ifun2 = 3 ;inviluppo grano
iolaps = 2 ;numero di sovrapposizioni

al syncgrain kamp,kfreq,kpitch,kgrsize,kprate1,ifun1,ifun2,iolaps
ar syncgrain kamp,kfreq,kpitch,kgrsize,kprate2,ifun1,ifun2,iolaps

outs al,ar

endin

</CsInstruments>
<CsScore>

f1 0 0 1 "vocal.aiff" 0 0 0
f3 0 8192 -20 2 ;Hanning

i1 0 40 1 .1 4

</CsScore>
</CsoundSynthesizer>
```

una variante di questo opcode è **syncloop**,il quale introduce la possibilità di indicare il punto di inizio di lettura file,e il segmento del campione audio da mettere in loop.Questa coppia di opcode può essere utilizzata per simulare la sintesi per formanti(Fof),per ottenere un simile effetto si consiglia di operare con valori di grain size = 0.04;la frequenza della formante sarà contenuta nel parametro kpitch.

Esempio Cap.9.4_syncloop.csd

```
<CsoundSynthesizer>
```

```

<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 16
nchnls = 2
0dbfs = 1

turnon      2      ;reverb sempre ON

gasig1      init 0      ;inizializzo le variabili globali
gasig2      init 0

instr 1

kamp = p4      ;ampiezza

kfreq randomi .001,30,1 ;density grain (1 a 100)
kpitch    randomi .2,2,1 ;grain pitch(negative =backwards)
kgrsize   randomi .01,.1,.1 ;grain size

kprate1    transeg 0.1, p3*0.25,p5, 1,p3*0.75,p6,0.1
kprate2    transeg 0.1, p3*0.25,p6, 1,p3*0.75,p5,0.1

ifun1 = 1      ;file audio in tabella
ifun2 = 3      ;inviluppo grano
iolaps = 2     ;numero di sovrapposizioni

kstart = 1
kend = 2

al syncloop kamp,kfreq,kpitch,kgrsize,kprate1,kstart,kend,ifun1,ifun2,iolaps
ar syncloop kamp,kfreq,kpitch,kgrsize,kprate2,kstart,kend,ifun1,ifun2,iolaps

aclip1 clip al,0,.9
aclip2 clip ar,0,.9

outs aclip1,aclip2

vincr gasig1,aclip1
vincr gasig2,aclip2

endin

instr 2      ;reverb
;prende in ingresso le variabili globali
aL,aR reverbsc gasig1,gasig2,.9,12000,sr,0
outs aL,aR
clear gasig1,gasig2      ;azzera la memoria per evitare accumulo

endin

</CsInstruments>
<CsScore>

f1 0 0 1 "vocal.aiff" 0 0 0
f3 0 8192 -20 2 ;Hanning

i1 0 40 1 .1 4

```

```
</CsScore>
</CsoundSynthesizer>
```

partikkel granular synthesizer

Partikkel è uno dei più entusiasmanti e complessi opcode contenuti in Csound, è stato introdotto nel 2007 con la versione 5.07 ed è stato sviluppato e ideato da *Thom Johansen, Torgeir Strand Henriksen e Øyvind Brandtsegg*.

L'idea di **partikkel** nasce dalla lettura di *Microsound* di *Curtis Roads* nel quale vengono descritte alcune tecniche granulari di nuova generazione (Glisson, Pulsar, Grainlet e Trainlet synthesis, quest'ultima inclusa in questo opcode), spesso la sintesi granulare viene definita come sintesi per particelle da cui deriva il nome. Si tratta di un sistema in grado di generare con un unico opcode tutti i vari tipi di tecniche di sintesi granulare (sincrona, asincrona, lettura di campioni audio, modulazione fm, ecc) ed è composto da un numero assai elevato di parametri da renderlo forse l'opcode più complesso da gestire in Csound.

Partikkel può avere fino a quattro sorgenti in ingresso (sintetiche o campionate) le quali possono essere mixate tra loro o trasposte individualmente, dispone di un input per la modulazione fm che può essere disabilitato e le uscite audio possibili possono arrivare fino ad 8.

Con **partikkel** è anche possibile sincronizzare il processo granulare con il timing di altri processi esterni. (Si consiglia la lettura di *Microsound* per la descrizione della trainlet synthesis inclusa in **partikkel**).

```
a1 [, a2, a3, a4, a5, a6, a7, a8] partikkel agrainfreq, \
    kdistribution, idisttab, async, kenv2amt, ienv2tab, ienv_attack, \
    ienv_decay, ksustain_amount, ka_d_ratio, kduration, kamp, igainmasks, \
    kwavfreq, ksweepshape, iwavfreqstarttab, iwavfreqendtab, awavfm, \
    ifmamptab, kfmenv, icosine, ktraincps, knumpartials, kchroma, \
    ichannelmasks, krandommask, kwaveform1, kwaveform2, kwaveform3, \
    kwaveform4, iwaveamptab, asamplepos1, asamplepos2, asamplepos3, \
    asamplepos4, kwavekey1, kwavekey2, kwavekey3, kwavekey4, imax_grains \
    [, iopcode_id]
```

Un esempio di partikkel nella struttura classica orchestra/score, poniamo molta attenzione al commento di ogni parametro :

Esempio Cap.9.5_partikkel

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

giSquare    ftgen 1,0,4097,7,1,2048,1,0,-1,2048,-1
giSaw ftgen 2,0,8193,7,1,8291,-1
giSine      ftgen 3,0,65537,10,1
giTri ftgen 4,0,8193,7,0,2048,1,4096,-1,2048,0
giCosine    ftgen 5,0,8193,9,1,1,90
giattack    ftgen 6,0,8193,19,0.5,1,270,1
gidecay     ftgen 7,0,8193,19,0.5,1,90,1
```

```

;partikkel instrument

instr 1

kamp = p4
kgrainfreq = p5*.1 ;rate (.1 a 200)
kdistr = 0 ;distribuzione periodica dei grani
idisttab = -1 ;(default)distribuzione dei grani(-1 si annulla)
async = 0 ;no sync input
kenv2amt = 0 ;nessun inviluppo secondario
ienv2tab = -1 ;default inviluppo secondario
iattack = giattack ;inviluppo attack
idecay = gidecay ;inviluppo decay
ksus = 0.5 ;sustain per il singolo grano
kratio = 0.5 ;bilanciamento tra attack and decay time
kdur = p6 ;grain size (.1 a 20)
igain = -1 ;(default) no gain per grano
kwavfreq = p5 ;frequenza dei grani (valore costante)
ksweep = 0 ;frequenza sweep (0=no sweep)
istarttab = -1 ;default frequency sweep start
ifreqendtab = -1 ;default frequency sweep end
afminput poscil p7,p8,3
awavfm = afminput ;oscil FM input
ifmamptab = -1 ;default FM scaling (=1)
kfmenv = -1 ;default FM envelope (flat)
icosine = giCosine ;cosine tabella
kTrainCps = kgrainfreq ;trainlet cps uguale a grain rate
kpartials = 3 ;numero di parziali in trainlet
kchroma = 1 ;bilanciamento di parziali in trainlet
ichmasks = -1 ;(default)all grains to output 1
krndmask = p9 ;componente random di grainrate (.1 a .9)
kwave1 = p10 ;sorgente in ingresso
kwave2 = p10
kwave3 = p10
kwave4 = p10
iwaveamp = -1 ;(default)quattro sorgenti mixate
apos1 = 0 ;fase di ogni sorgente
apos2 =
apos3 =
apos4 =
kkey1 = 1 ;trasposizione di ogni sorgente
kkey2 = 1
kkey3 = 1
kkey4 = 1
imax_grains = 100 ;massimo numero di grani

a1 partikkel kgrainfreq,kdistr,idisttab,async,kenv2amt,ienv2tab, \
iattack,idecay,ksus,kratio,kdur, kamp,igain, \
kwavfreq,ksweep,istarttab,ifreqendtab,awavfm, \
ifmamptab,kfmenv,icosine,kTrainCps,kpartials, \
kchroma,ichmasks,krndmask,kwave1,kwave2,kwave3,kwave4, \
iwaveamp,apos1,apos2,apos3,apos4, \
kkey1,kkey2,kkey3,kkey4,imax_grains

aout clip a1,.9
outs aout,aout
endin

</CsInstruments>

```

```

<CsScore>

;p4 = amp
;p5 = ;rate
;p6 = ;grain size
;p7,p8 = fm amp/freq
;p9;componente random di grainrate (.1 a .9)
;p10 = waveform

i1    0     10    1     200   20    0     0     0     1
s
i1    0     10    1     100   1     0     0     .5    4
s
i1    0     10    1     400   8     2     3     .9    3
s
i1    0     10    1     1000  2     0     0     .3    3
s
i1    0     10    1     2000  8     1     3     .9    3
s

</CsScore>
</CsoundSynthesizer>

```

Vediamo adesso un esempio di sintesi granulare per il controllo in tempo reale,i parametri principali sono :

- ampiezza
- grain rate
- grain size
- oscillatore in input per la modulazione Fm (con ampiezza e frequenza controllabili)
- joystick per trasposizioni della sorgente sonora
- possibilità di scegliere una forma d'onda diversa per i due generatori granulari (da scegliere tra quattro funzioni diverse)
- componente di variazione random per il grain rate

Esempio Cap.9.5_partikkel-gui

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

gkwave1    init  3
gkwave2    init  3

```

```

giSquare    ftgen 1,0,4097,7,1,2048,1,0,-1,2048,-1
giSaw      ftgen 2,0,8193,7,1,8291,-1
giSine     ftgen 3,0,65537,10,1
giTri      ftgen 4,0,8193,7,0,2048,1,4096,-1,2048,0
giCosine   ftgen 5,0,8193,9,1,1,90
giattack   ftgen 6,0,8193,19,0.5,1,270,1
gidecay    ftgen 7,0,8193,19,0.5,1,90,1

FLpanel     "partikkel",600,380,150,100,1,-1,-1
FLscroll    600,350,0,0

gkOnOff,ihOnOff  FLbutton    "On/Off",1,0,22,90,25,10,320,0,1,0,-1

;ihandle FLvalue "label",iwidth,iheight,ix,iy
ival7 FLvalue    "",50,30,10,270
ival8 FLvalue    "",50,30,80,270
ival9 FLvalue    "",50,30,150,270
ival10    FLvalue "",50,30,220,270
ival11    FLvalue "",50,30,290,270
ival12    FLvalue "",50,30,360,270

imin = 100
imax = 1
iexp = 0
iheight = 30
iwidth = 500
ix = 20
iy = 20
iwidth2 = 30

;kout,ihandle FLslider "label",imin,imax,iexp,itype,idisp,iwidth,iheight,ix,iy
gk1,ih1    FLslider    "amp",0,1,iexp,24,ival7,iwidth2,iheight+200,ix,iy
gk2,ih2    FLslider    "rate",.1,200,-1,24,ival8,iwidth2,iheight+200,ix+70,iy
gk3,ih3    FLslider    "size",.1,20,iexp,24,ival9,iwidth2,iheight+200,ix+140,iy
gk4,ih4    FLslider    "Fm amp",.01,10,iexp,24,ival10,iwidth2,iheight+200,ix+210,iy
gk5,ih5    FLslider    "Fm freq",.1,16,iexp,24,ival11,iwidth2,iheight+200,ix+280,iy
gk6,ih6    FLslider    "rndmask",.1,.9,iexp,24,ival12,iwidth2,iheight+200,ix+350,iy
gkx,gky,gihx,gihy FLjoy "transposition",.1,50,.1,50,-1,-1,-1,140,140,430,20
gkwave1,ih7 FLcount   "Waveform1",1,4,1,1,1,140,30,430,212,-1,1,0,1
gkwave2,ih8 FLcount   "Waveform2",1,4,1,1,1,140,30,430,282,-1,1,0,1

FLscrollEnd
FLpanelEnd

FLrun

FLsetVal_i 0,ih1
FLsetVal_i 10,ih2
FLsetVal_i 10,ih3
FLsetVal_i 0,ih4
FLsetVal_i 0,ih5
FLsetVal_i .5,ih6
FLsetVal_i 3,ih7
FLsetVal_i 3,ih8
FLsetVal_i 1,gihx
FLsetVal_i 1,gihy
;partikkel instrument
instr 1

if      gkOnOff      =      0      then ;on/off strumento
turnoff
endif

```

```

kamp      =      gk1
kgrainfreq =      gk2    ;rate
kdistr     =      0      ;distribuzione periodica dei grani
idisttab   =      -1    ;(default)distribuzione dei grani(-1 si annulla)
async      =      0      ;no sync input
kenv2amt   =      0      ;nessun inviluppo secondario
ienv2tab   =      -1    ;default inviluppo secondario
iattack    =      giattack ;inviluppo attack
idecay     =      gidecay  ;inviluppo decay
ksus       =      0.5   ;sustain per il singolo grano
kratio     =      0.5   ;bilanciamento tra attack and decay time
kdur       =      gk3    ;grain size
igain      =      -1    ;(default) no gain per grano
kwavfreq   randomh 60,2000,gk2 ;frequenza dei grani
ksweep     =      0      ;frequenza sweep (0=no sweep)
istarttab  =      -1    ;default frequency sweep start
ifreqendtab=      -1    ;default frequency sweep end
afminput   oscili   gk4,gk5,3
awavfm     =      afminput ;oscil FM input
ifmamptab =      -1    ;default FM scaling (=1)
kfmenv     =      -1    ;default FM envelope (flat)
icosine    =      giCosine ;cosine tabella
kTrainCps  =      kgrainfreq ;trainlet cps uguale a grain rate
kpartials  =      3      ;numero di parziali in trainlet
kchroma   =      1      ;bilanciamento di parziali in trainlet
ichmasks   =      -1    ;(default)all grains to output 1
krndmask   =      gk6    ;componente random di grainrate
kwavel     =      gkwavel ;sorgente in ingresso
kwave2    =      gkwavel
kwave3    =      gkwavel
kwave4    =      gkwavel
iwaveamp   =      -1    ;(default)quattro sorgenti mixate
apos1     =      0      ;fase di ogni sorgente
apos2     =      0
apos3     =      0
apos4     =      0
kkey1     =      gkx    ;trasposizione di ogni sorgente
kkey2     =      gky
kkey3     =      gkx
kkey4     =      gky
imax_grains =      100   ;massimo numero di grani

a1      partikkel kgrainfreq,kdistr,idisttab,async,kenv2amt,ienv2tab,\ 
        iattack,idecay,ksus,kratio,kdur, kamp,igain,\ 
        kwavfreq,ksweep,istarttab,ifreqendtab,awavfm,\ 
        ifmamptab,kfmenv,icosine,kTrainCps,kpartials,\ 
        kchroma,ichmasks,krndmask,kwavel,kwave2,kwave3,kwave4,\ 
        iwaveamp,apos1,apos2,apos3,apos4,\ 
        kkey1,kkey2,kkey3,kkey4,imax_grains

;i valori di trasposizione per a1 vengono raddoppiati per a2
kkey5     =      gkx*2
kkey6     =      gky*2
kkey7     =      gkx*2
kkey8     =      gky*2

a2      partikkel kgrainfreq,kdistr,idisttab,async,kenv2amt,ienv2tab,\ 
        iattack,idecay,ksus,kratio,kdur, kamp,igain,\ 
        kwavfreq,ksweep,istarttab,ifreqendtab,awavfm,\ 
        ifmamptab,kfmenv,icosine,kTrainCps,kpartials,\ 
        kchroma,ichmasks,krndmask,kwavel,kwave2,kwave3,kwave4,\ 
        iwaveamp,apos1,apos2,apos3,apos4,\ 

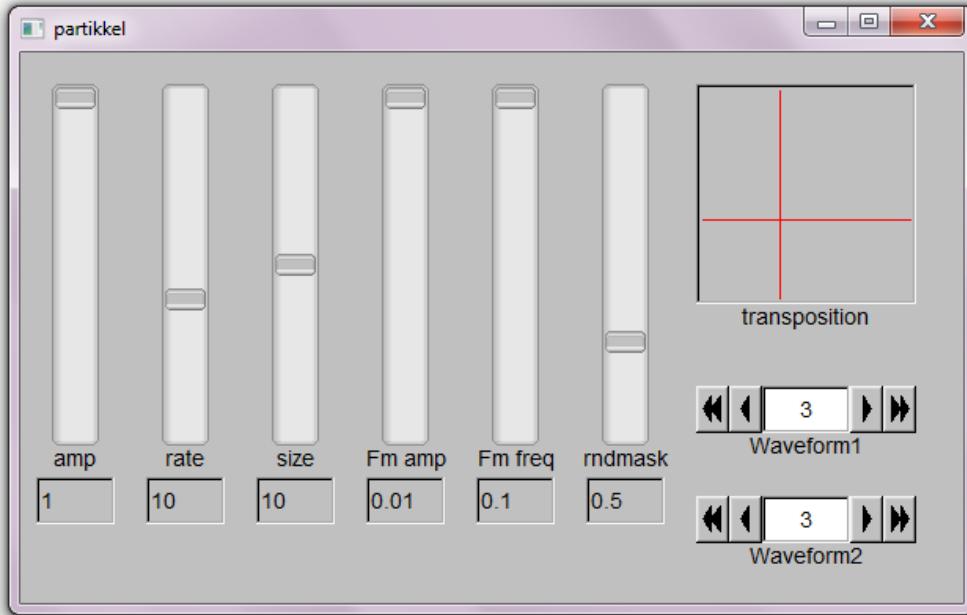
```

```

kkey5,kkey6,kkey7,kkey8,imax_grains

outs a1,a2

endin
</CsInstruments>
<CsScore>
i1 0 3600
</CsScore>
</CsoundSynthesizer>
```



Nel prossimo esempio utilizziamo una tecnica di sottoprogramma (simile alle subpatchs di Max) che racchiude gli innumerevoli parametri di partikkel, lo scopo è quello di gestire in orchestra una versione semplificata dell'opcode in cui modifichiamo solo i parametri fondamentali. In Csound questo tipo di operazione si chiama U.d.o (user defined opcodes) e l'utilizzo è quello di un qualunque opcode di Csound. Vediamo adesso un esempio che integra il controllo midi sui parametri di partikkel, l'esempio seguente utilizza due istanze di sintesi granulare in cui è possibile controllare lo sfasamento tra i due canali left e right.

Esempio Cap.9.6_partikkel-softsync

```

<CsoundSynthesizer>
<CsOptions>
-M1 --rtmidi=virtual
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

garev1      init  0
garev2      init  0

#include    "table.txt"
#include    "partikkel_softsync.txt"

instr 1
;numeri controller midi
```

```

;1 = rate
;2 = size
;3 = sincronizzazione due istanze partikkel
;4 = deviazione rate seconda istanza partikkel
;5 = cutoff filtro
;6 = resonance filtro
;7 = illevel reverb
iamp          ampmidi .6
igrainFreq   cpsmidi
kgrainrate   ctrl7 1,1,.1,50 ;grain rate
kgrainsize   ctrl7 1,2,.1,50 ;grain size
iosc2Dev     ctrl7 1,3,.1,5 ;deviazione rate seconda istanza partikkel
iMaxSync    ctrl7 1,4,.1,5 ;sincronizzazione due istanze partikkel
iwave1       =      giwave1
iwave2       =      giwave2
iwave3       =      giwave3
iwave4       =      giwave4
iattack      =      giSigmoRise
idecay       =      giSigmoFall
iCosine      =      giCosine

a1,a2 partikkel_softsync      iamp,kgrainrate,kgrainsize,igrainFreq,iosc2Dev,\ 
iMaxSync,iwave1,iwave2,iwave3,iwave4,iattack,idecay,iCosine

kfiltfreq   ctrl7 1,5,100,20000 ;cutoff filtro
kfiltreso   ctrl7 1,6,.1,.9    ;resonance filtro

afiltL       moogvcf2 a1,kfiltfreq,kfiltreso
afiltR       moogvcf2 a2,kfiltfreq,kfiltreso

aenv linsegr  0,.1,1,.3,.5,.2,0

outs afiltL*aenv,afiltR*aenv

garev1      =      afiltL+garev1
garev2      =      afiltR+garev2

endin

instr 2
klevel      ctrl7 1,7,0,.9 ;reverb
a1,a2 reverbsc  garev1,garev2,klevel,12000, sr, 0.5, 1

outs a1,a2

garev1      =      0
garev2      =      0
endin

</CsInstruments>
<CsScore>
i1    0    3600
i2    0    3600
</CsScore>
</CsoundSynthesizer>

```

Vediamo adesso l'utilizzo di partikkel sulla granulazione di un file audio contenuto in una tabella, non ci sono limiti per la durata del file contenuto in tabella. Anche il prossimo esempio utilizza una versione personalizzata e dal numero di parametri ridotto, la prossima gui è scritta per QuteCsound e pertanto non può essere letta con altri editor.

Esempio Cap.9.7_partikkel-sample(qutecsoud)

```

<CsSoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2

giCosine    ftgen 0,0,8193,9,1,1,90
giDisttab   ftgen 0,0,32768,7,0,32768,1
giWin ftgen 0,0,4096,20,9,1
giFile      ftgen 0,0,0,1,"vocal.aiff",0,0,0 ;soundfile for source waveform

#include "Partikkkel_sample.txt"

instr 1

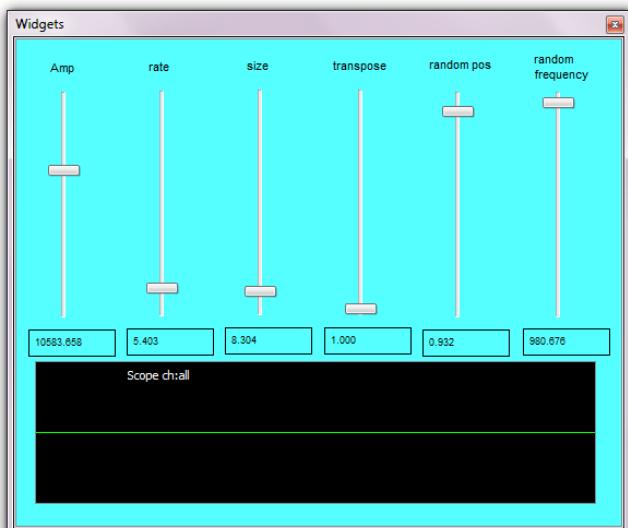
ifiltab = giFile      ;tabella campione audio
kgrainamp invalue "amp" ;ampiezza
kgrainrate invalue "rate" ;grain rate
kgrainsize invalue "size" ;grain size
kcen invalue "transpose" ;trasposizione
kposrand invalue "randompos" ;random grainrate
kcentrand invalue "random" ;freq random
icosintab = giCosine    ;tabella cosine waveform
idisttab = giDisttab
iwin = giWin ;function table with window shape
iSync = .4 ;strength of synchronization
iDev = 2 ;fattore grains/secondi for second partikkkel instance

a1,a2 Partikkkel_sample ifiltab,kgrainamp,kgrainrate,kgrainsize,\ 
kcen,kposrand,kcentrand,icosintab,idisttab,iwin,iSync,iDev
outs a1,a2

endin

</CsInstruments>
<CsScore>
i1 0 3600
</CsScore>
</CsSoundSynthesizer>

```



Gli esempi visti in questa sezione su partikkel rappresentano solo la più piccola parte delle reali potenzialità di questo opcode, questi esempi devono essere interpretati come punto di partenza per ulteriori esplorazioni tecniche e timbriche, per approfondimenti si rimanda alla documentazione ufficiale.

vocal synthesis

Un particolare tipo di sintesi granulare è la sintesi per formanti, si tratta di una tecnica basata sul concetto di formante (ossia una particolare zona di risonanza che si comporta come un filtro passabanda), in Csound questa tecnica si chiama **FOF** (Fonction d'onde formantique) e fu sviluppata all'Ircam nel progetto *Chant* di *Xavier Rodet*.

In Csound l'opcode specifico prende il nome di **fof** (di cui esiste una variante **fof2** e **fof3**, quest'ultima implementata al momento solo in CsoundAV di *G.Maldonado*), l'utilizzo di questo opcode è molto legato alle ricerche sulla sintesi vocale, con **fof** è possibile costruire una simulazione del tratto vocale utilizzando le formanti vocali, in pratica un generatore fof produce un treno di impulsi filtrati da filtri passabanda.

```
ares fof xamp,xfund,xform,koct,kband,kris,kdur,kdec,iolaps,ifna,ifnb,itotdur
```

- xamp : ampiezza
- xfund : frequenza fondamentale dell'impulso di sinusoidi
- xform : frequenza del formante
- koct : indice di ottavizzazione, in genere uguale a zero
- kband : larghezza di banda del formante (-6 db)
- kris,kdur,kdec : inviluppo degli impulsi di sinusoidi
- iolaps : numero di sovrapposizione degli impulsi
- ifna,ifnb : la prima funzione è per le sinusoidi, la seconda per l'inviluppo
- itotdur : normalmente uguale a p3

Il prossimo esempio è un semplice tentativo di imitazione vocale utilizzando le formanti di un tenore (con vocale “a”), nel manuale ufficiale troviamo i valori delle formanti e kband da utilizzare con fof (e non solo) :

Table D.21. tenor “a”

freq (Hz)	650	1080	2650	2900	3250
bw (Hz)	80	90	120	130	140

utilizzando cinque istanze di fof, ognuno con una frequenza diversa per la formante, si riesce ad ottenere una discreta imitazione della vocale “a”, in questo esempio la frequenza fondamentale sarà modulata da un leggero vibrato con componenti random per simulare il vibrato naturale della voce cantata.

Esempio Cap.9.8_vocal synthesis

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1
```

```

kamp = p4 ;ampiezza

;formanti tenore "a"

kform1 = 650
kform2 = 1080
kform3 = 2650
kform4 = 2900
kform5 = 3250

;larghezza di banda delle formanti tenore

kbn1 = 80
kbn2 = 90
kbn3 = 120
kbn4 = 130
kbn5 = 140

kfund linseg p5,p3/2,p6,p3/2,p5 ;frequenza fondamentale

koct = 0

;0.003,0.02,0.007 valori consigliati per la sintesi vocale
kris = 0.003
kdur = 0.02
kdc = 0.007
iol = 100 ;sovraposizioni
ifna = 1 ;funzione sinusoida
ifnb = 2 ;funzione attack,decay
idur = p3 ;durata di fof

;vibrato con componente random

kjitter jitter 8,1,3
kvib lfo 2,kjitter

k1 madsr 1,0,1,0.01;inviluppo d'ampiezza

a1 fof kamp*k1,kfund+kvib,kform1,koct,kbn1,kris,kdur,kdc,iol,ifna,ifnb,idur
a2 fof kamp*k1,kfund+kvib,kform2,koct,kbn2,kris,kdur,kdc,iol,ifna,ifnb,idur
a3 fof kamp*k1,kfund+kvib,kform3,koct,kbn3,kris,kdur,kdc,iol,ifna,ifnb,idur
a4 fof kamp*k1,kfund+kvib,kform4,koct,kbn4,kris,kdur,kdc,iol,ifna,ifnb,idur
a5 fof kamp*k1,kfund+kvib,kform5,koct,kbn5,kris,kdur,kdc,iol,ifna,ifnb,idur

aout = a1+a2+a3+a4+a5

kenv linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0;declick

outs (aout)*kenv,(aout)*kenv

endin

</CsInstruments>
<CsScore>

f1 0 8192 10 1
f2 0 8192 19 0.5 0.5 270 0.5

i1 0 .5 .3 100 300
s
i1 0 .6 .3 300 200

```

s
i1 0 .7 .3 300 80
s
i1 0 .9 .3 80 200
s
i1 0 1.5 .3 200 200
s
i1 0 3 .3 100 100
s

</CsScore>
</CsoundSynthesizer>

Cap.11 Modelli fisici

10.1 waveguide

10.2 Costruzione di risonatori

10.3 Scanned synthesis

Le tecniche di sintesi per l'emulazione del timbro degli strumenti acustici sono di due tipi :

- **sintesi Pcm(pulse code modulation)** : tecnica basata sul campionamento digitale di strumenti per costruire una banca dati di timbri (memorizzata su memorie "Rom") ,i waveform vengono poi elaborati ad esempio con tecniche di sintesi sottrattiva.Celebre la workstation Korg M1 (a cui seguono Trinity,Triton,Karma,Oasys).Pertanto non va considerata come una vera e propria tecnica di sintesi in quanto il suono prodotto è in parte preregistrato,la qualità della simulazione acustica è migliorata nel corso degli anni grazie alle Rom di dimensioni maggiori,quindi con la possibilità di immagazzinare campioni con frequenze di campionamento più alte.
- **sintesi per modelli fisici** : processo di analisi e ricostruzione virtuale delle singole componenti di uno strumento musicale come i materiali,le dimensioni,i processi di eccitazione e vibrazione.Vista l'alta mole di calcoli richiesti,questo tipo di tecnica trova ancora oggi poche applicazioni negli strumenti hardware commerciali.

In Csound esiste una vera e propria libreria di modelli fisici (strumenti percussivi,corde pizzicate,strumenti a fiato),oltre a unità più elementari come impulsi,risuonatori di frequenze modali dei materiali e modelli a guida d'onda.Nella versione per windows è anche integrata la libreria Stk di *Perry Cook* con numerosi modelli di strumenti acustici.

Waveguide

La sintesi a guida d'onda o *waveguide* studia la traiettoria di un'onda sonora in un corpo risonante (come una piastra percossa,una membrana,un tubo percorso da aria).

Elementi fondamentali di un modello a guida d'onda sono gli eccitatori (immaginiamo un archetto di uno strumento ad arco,il martelletto di un pianoforte, il plettro che pizzica una corda ,ecc),linee di ritardo (per simulare la traiettoria dell'onda) e filtri.

L'approccio iniziale con questo tipo di sintesi può avvenire per mezzo di specifici opcode modellati con questa tecnica,tuttavia è anche possibile ricostruire in Csound un modello fisico partendo da unità generatrici più elementari (con la stessa modalità già incontrata nel capitolo sulla sintesi granulare).

Alcuni opcode di modelli a guida d'onda sono : **wgbow,wgflute,wgclar,wgbowedbar,pluck**.Sempre in questa categoria troviamo alcuni strumenti che si comportano come un risonatore : **wguide1, wguide2,streson**.

Alcuni esempi con wgflute e wgpluck,rispettivamente un modello fisico di flauto e di corda pizzicata :

```
ares  wgflute kamp,kfreq,kjet,iatt,idetk,kngain,kvibf,kvamp,ifn  
ares  wgpluck icps,iamp,kpick,iplk,idamp,ifilt,axcite
```

Esempio Cap.10.1_wgflute.csd

```
<CsoundSynthesizer>  
<CsOptions>  
-odac -M0 --rtmidi=virtual  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
kr = 4410  
ksmps = 10  
nchnls = 2  
  
FLpanel      "Wgflute",620,290,300,300,8,1,1  
FLvkeybd "keyboard.map",580,100,15,180  
  
ih1    FLvalue      "",70,25,542,12
```

```

gk1,ikjit    FLslider    "jet",0.1,0.9,0,5,ih1,530,20,5,10
ih2    FLvalue     "",70,25,542,72
gk2,igain   FLslider    "gain",0.1,0.5,0,5,ih2,530,20,5,70

ih3    FLvalue     "",70,25,542,132
gk3,ivib    FLslider    "Vibrato",.1,10,0,5,ih3,530,20,5,130

;valori iniziali
FLsetVal_i 0.02,ikjit
FLsetVal_i 0.1,igain
FLsetVal_i 1,ivib

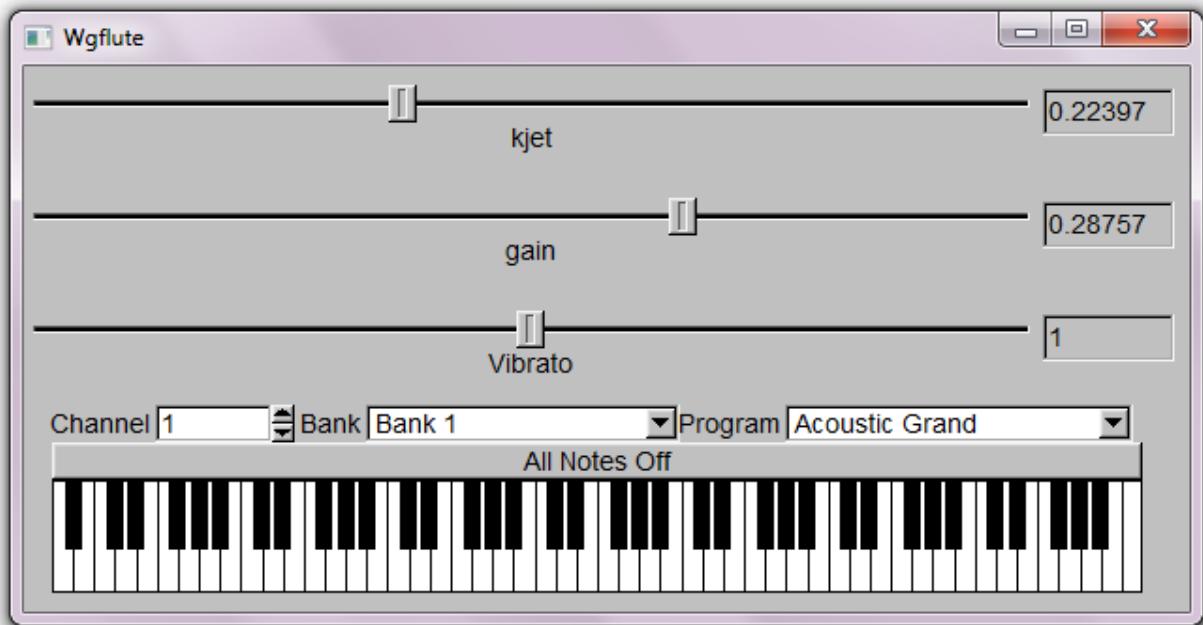
FLpanelEnd
FLrun

instr 1

ifrq  cpsmidi      ;controllo midi
iamp  ampmidi      10000
a1    wgflute       iamp,ifrq,gk1,0.1,0.1,gk2,gk3,0.05,1
kenv  linsegr      0,2,1,.2,.5,.2,0 ;inviluppo d'ampiezza
outs  a1*kenv,a1*kenv

endin

</CsInstruments>
<CsScore>
f1    0      16384 10      1
f0    36000
</CsScore>
</CsoundSynthesizer>
```



Le caratteristiche principali di wgflute riguardano le variabili jet (con cui controllare l'intensità del getto d'aria) e gain che controlla la quantità di rumore del segnale (serve a simulare il soffio prodotto dall'esecutore).

Nel prossimo esempio abbiamo una simulazione di corda pizzicata con il controllo sul punto della corda in cui avviene l'eccitazione, il parametro kpick influenza invece il timbro in uscita.

Esempio Cap.10.2_wgpluck.csd

```
<CsoundSynthesizer>
<CsOptions>
-odac1 -M0 -+rtmidi=virtual
</CsOptions>
<CsInstruments>
:Esempio Cap.10.2_wgpluck.csd
sr      = 44100
kr      = 441
ksmps   = 100
nchnls = 2

FLpanel      "wgpluck", 620, 400, 300, 300, 4, -1, 1

FLvkeybd "keyboard.map", 580, 100, 15, 260

ih1  FLvalue "", 70, 25, 542, 12
gk1, ipick  FLslider "kpick", .1, 2, 0, 5, ih1, 530, 20, 5, 10

ih2  FLvalue "", 70, 25, 542, 72
gk2, iplk   FLslider "iplk(point-plucked)", 0.1, 1, 0, 5, ih2, 530, 20, 5, 70

ih3  FLvalue "", 70, 25, 542, 132
gk3, idamp  FLslider "damping", 1, 100, 0, 5, ih3, 530, 20, 5, 130

ih4  FLvalue "", 70, 25, 542, 192
gk4, ifilt   FLslider "Filter", 1, 10000, 0, 5, ih4, 530, 20, 5, 190

;settings init
FLsetVal_i .5,    ipick
FLsetVal_i 1,     iplk
FLsetVal_i 10,    idamp
FLsetVal_i 1000,  ifilt

FLpanelEnd

FLrun

instr 1

ifrq  cpsmidi ;controllo midi
iamp  ampmidi 12000
kenv  linsegr  1,.1,1,.3,.5,.2,0 ;inviluppo d'ampiezza
axcite oscil 1,1,1 ;segnale eccitatore
apluck wgpluck ifrq,iamp,gk1,i(gk2),i(gk3),i(gk4),axcite
outs (apluck)*kenv, (apluck)*kenv

endin

</CsInstruments>
<CsScore>
f1      0      4096  10      1
f0      36000
</CsScore>
</CsoundSynthesizer>
```

I risonatori sono dei moduli in grado di simulare il comportamento e le caratteristiche timbriche di un particolare corpo risonante, immaginiamo di simulare il risonatore di una piastra percossa eccitata dall'archetto di un violino. La funzione di un risonatore è quella di un filtro in cui le frequenze sono date dalle frequenze modalì dei materiali.

- wguide1 : modello di corda pizzicata costituito da una linea di ritardo e da un filtro passa basso del primo ordine.
- wguide2 : modello di piastra percossa con algoritmo simile a wguide1 ma in parallelo (due linee di ritardo e due filtri)
- streson : modello di corda

per eccitare un risonatore è possibile utilizzare uno speciale opcode in grado di generare un impulso a intervalli prestabiliti ,la sintassi di **mpulse** :

aecc **mpulse** kamp, kintvl

- kamp : ampiezza dell'impulso
- kintvl : intervallo di tempo in secondi tra un impulso e il successivo,con valore = zero si ha un solo impulso
-

la complessità degli eventi generati nel prossimo esempio è generata dal controllo random di kintvl,il risultato è un flusso di impulsi a velocità variabile nel tempo.

Esempio Cap.10.3_wguide1,wguide2,streson.csd

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1 ;(ampiezza massima = 1 db)

;impulso con risonatore di piastra percossa
Instr 1
krnd randomi .009,.1,2000
asig mpulse 3,krnd
kfreq1 randomi 60,500,.1
kfreq2 randomi 60,500,.2
kcut1 randomi 1000,20000,.3
kcut2 randomi 1000,20000,.4
kfeed1 = .2 ;la somma dei feedback non deve superare il valore = .5
kfeed2 = .2
awg wguide2 asig,kfreq1,kfreq2,kcut1,kcut2,kfeed1,kfeed2
outs awg,awg

endin

;rumore bianco con risonatore di piastra percossa
instr 2
asig rand .1
kfreq1 randomi 60,10000,1
kfreq2 randomi 60,10000,2
kcut1 randomi 60,20000,3
kcut2 randomi 60,20000,4
kfeed1 = .2
kfeed2 = .2
awg wguide2 asig,kfreq1,kfreq2,kcut1,kcut2,kfeed1,kfeed2
outs awg,awg

endin

;impulso con waveguide composto da delay line e filtro passa-basso
```

```

instr 3
krnd randomi .001,.1,2000
asig mpulse 3,krnd
kfreq randomi 60,400,3
kcut randomi 1000,10000,4
kfeed = .5
awg wguidel asig,kfreq,kcut,kfeed
outs awg,awg

endin

;rumore bianco con waveguide composto da delay line e filtro passa-basso
instr 4
asig rand .1
kfreq randomi 100,1000,3
kcut randomi 1000,10000,4
kfeed = .5
awg wguidel asig,kfreq,kcut,kfeed
outs awg,awg

endin

;impulso con risonatore di corda pizzicata
instr 5

krnd randomi .009,.1,1000
asig mpulse 3,krnd

;pitch del risonatore generato da una tabella con scala definita
;kmelody legge la tabella(f2) in modo casuale generando una melodia
iscale = 2
kmelody randomi 1,22,1
kpitch table kmelody,iscale ;tabella con scala diatonica

ifdbgain = .9
astr1 streson asig,cpspch(kpitch),ifdbgain
astr2 streson asig,cpspch(kpitch)*1.4,ifdbgain
aenv linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0
outs astr1*aenv,astr2*aenv

endin

;file audio con risonatore di corda pizzicata
;il risultato è simile a sonorità ottenute con l'effetto flanger o un filtro
comb
instr 6

iscale = 2
kmelody randomi 1,22,3
kpitch table kmelody,iscale
asig1,asig2 diskin2 "drums_loop.aif",1,0,1
ifdbgain = .9
astr1 streson asig1,cpspch(kpitch)*.9,ifdbgain
astr2 streson asig2,cpspch(kpitch)*2,ifdbgain
aenv linseg 0,0.02,1,p3-0.05,1,0.02,0,0.01,0
outs astr1*aenv,astr2*aenv

endin

</CsInstruments>
<CsScore>
f2 0 32 -2 6.07 6.09 6.11 7.00 7.02 7.04 7.06
    7.07 7.09 7.11 8.00 8.02 8.04 8.06

```

```

8.07 8.09 8.11 9.00 9.02 9.04 9.06
10.07 10.09 10.11 11.00 11.02 11.04 11.06

i1    0      10
s
i2    0      10
s
i3    0      10
s
i4    0      10
s
i5    0      10
s
i6    0      10
s

</CsScore>
</CsoundSynthesizer>
```

Nella distribuzione di Csound per windows troviamo un'altra libreria per modelli fisici molto ricca di modelli strumentali, si tratta di un estratto di una più completa libreria chiamata STK (*Synthesis ToolKit in C++*) di *Perry Cook* e *Gary P. Scavone*. La libreria offre emulazioni di strumenti acustici e offre un utilizzo relativamente semplice (escludendo i numerosi parametri opzionali che riguardano, ad esempio, la posizione del plettro su una corda, l'intensità del soffio in uno strumento a fiato, ecc), un esempio di sintassi STK :

```
asignal STKClarinet ifrequency, iamp
```

si raccomanda l'utilizzo di questi opcode nella seguente modalità :

```
asignal STKClarinet ifrequency, iamp
```

```
out     asignal*10000
```

oppure utilizzando 0dbfs = 1 nell'intestazione

esempio :

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 2

instr 1

ifrq    =        440
iamp    =        1
asig    STKClarinet    ifrq, iamp
        outs asig*10000, asig*10000
endin

</CsInstruments>
<CsScore>
i1    0      4
</CsScore>
</CsoundSynthesizer>
```

L'elenco dei modelli fisici integrati dalla STK :

STKBandedWG, STKBeeThree, STKBlowBotl, STKBlowHole, STKBowed, STKBrass, STKClarinet, STKDrummer, STKFlute, STKFMVoices, STKHevyMetl, STKMandolin, STKModalBar, STKMoog, STKpercFlut, STKPlucked, STKResonate, STKRhodey, STKSaxofony, STKShakers, STKSimple, STKSitar, STKStifKarp, STKTubeBell, STKVoicForm, STKWhistle, STKWurley.

Costruzione di risonatori

Nel paragrafo precedente abbiamo visto l'utilizzo di alcuni risonatori di corda e piastra percossa, per costruire dei risonatori personalizzati di materiali specifici è possibile utilizzare l'opcode **mode** (basato sul modello fisico massa-molla-smorzatore), la sua funzione è quella di filtrare un segnale in ingresso con una frequenza di risonanza e un fattore di smorzamento (damping).

Per creare un risonatore è possibile usare diverse istanze di mode utilizzando i rapporti delle frequenze modali dei materiali (vedere la sezione Modal Frequency Ratios dal csound manual per le tabelle di frequenze modali di oggetti e strumenti, contributo di *Scott Lindroth*).

```
aout mode ain,kfreq,kQ
```

- ain : segnale in ingresso
- kfreq : frequenza di risonanza
- kQ : damping (fattore di smorzamento)
-

il prossimo esempio utilizza due istanze di **mode** sommate e con frequenza variabile random, lo strumento eccitatore è un generatore di rapidi impulsi come negli esempi con wguide1 e 2.

Esempio Cap.10.4_mode.csd

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>
;Esempio Cap.10.4_mode.csd
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1
;eccitatore ad impulsi
krnd  randomi    .01,.2,1000
aecc  mpulse     3,krnd

;risonatore
;p5    damping
imode = 3      ;inizia la fase di inizializzazione con un valore casuale
kfreq1  randomh   500,8000,2,imode
kfreq2  randomh   100,7000,3,imode
amode1  mode aecc,kfreq1,p5
amode2  mode aecc,kfreq2,p5
amode = amode1+amode2

;uscita
aout  balance    amode,aecc
aout = aout*p4

;random pan
kpan  randi 1,1,-1
al,ar pan2 aout,kpan,2
```

```

outs al,ar
endin
</CsInstruments>
<CsScore>
i1    0      20     15000 1000
s
</CsScore>
</CsoundSynthesizer>
```

Vediamo adesso un esempio di risonatore partendo dalle frequenze modali contenute nel *canonical reference manual*,prendiamo la tabella dello strumento Dahina tabla :

[1, 2.89, 4.95, 6.99, 8.01, 9.02]

abbiamo bisogno in questo caso di sei istanze di **mode**,una frequenza base comune al risonatore sarà moltiplicata per ognuno dei rapporti di frequenze modali relative a Dahina tabla,infine verranno sommate le sei istanze per l'uscita audio creando un modello di risonatore basato sulle tabla.

;Esempio Cap.10.5_Dahina tabla.csd

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1
;input con file audio
aecc,aecc2 diskin2      "drums_loop.aif",1,0,1

;risonatore
;Dahina tabla modal ratios [1, 2.89, 4.95, 6.99, 8.01, 9.02]
;p5 = pitch base
;p6 = damping
amode1 mode aecc,cpspch(p5)*1,p6
amode2 mode aecc,cpspch(p5)*2.89,p6
amode3 mode aecc,cpspch(p5)*4.95,p6
amode4 mode aecc,cpspch(p5)*6.99,p6
amode5 mode aecc,cpspch(p5)*8.01,p6
amode6 mode aecc,cpspch(p5)*9.02,p6

aresonator sum amode1,amode2,amode3,amode4,amode5,amode6

aout balance aresonator,aecc

outs aout*p4,aout*p4

endin

</CsInstruments>
<CsScore>
i1    0      2      1      8.01   100
s
i1    0      2      1      8.04   100
s
```

```

i1      0      4      1      9.04   100
s
</CsScore>
</CsoundSynthesizer>

```

ora un esempio in cui **mode** viene utilizzato anche come eccitatore :

Esempio Cap.10.6_wood.csd

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1
;eccitatore Red Cedar wood plate
;[1, 1.47, 2.09, 2.56]
kpitch      =      cpspch(p5)/2
kq          =      100
apulse      mpulse    1,2

amode1      mode  apulse,kpitch*1,kq
amode2      mode  apulse,kpitch*1.47,kq
amode3      mode  apulse,kpitch*2.09,kq
amode4      mode  apulse,kpitch*2.56,kq
aecc  sum    amodel1,amode2,amode3,amode4

;risonatore membrana
;p5      pitch base
;p6      damping
a1  mode  aecc,cpspch(p5)*1,p6
a2  mode  aecc,cpspch(p5)*1.59334,p6
a3  mode  aecc,cpspch(p5)*2.13555,p6
a4  mode  aecc,cpspch(p5)*2.65307,p6
a5  mode  aecc,cpspch(p5)*2.29542,p6
a6  mode  aecc,cpspch(p5)*2.9173,p6
a7  mode  aecc,cpspch(p5)*3.50015,p6
a8  mode  aecc,cpspch(p5)*4.05893,p6
a9  mode  aecc,cpspch(p5)*3.59848,p6
a10 mode  aecc,cpspch(p5)*4.23044,p6
a11 mode  aecc,cpspch(p5)*4.83189,p6
a12 mode  aecc,cpspch(p5)*5.41212,p6
a13 mode  aecc,cpspch(p5)*4.90328,p6
a14 mode  aecc,cpspch(p5)*5.5404,p6
a15 mode  aecc,cpspch(p5)*6.15261,p6
a16 mode  aecc,cpspch(p5)*6.74621,p6

aresonator =      (a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+a11+a12+a13+a14+a15+a16)/16
aout  balance  aresonator,aecc

outs  (aout+aecc)*p4, (aout+aecc)*p4

endin

</CsInstruments>
<CsScore>
i1      0      4      10000  6.05   1000

```

```

s
i1    0      4      10000 7.02  1000
s
i1    0      4      10000 7.07  1000
s
i1    0      4      10000 8.05  1000
s
i1    0      4      10000 8.01  1000
s
</CsScore>
</CsoundSynthesizer>
```

Adesso un semplice processo generativo per la costruzione di risonatori, si tratta della stessa tecnica vista nel capitolo sulla sintesi additiva, l'idea quindi di un processo di controllo per generare un risonatore partendo da una tabella con i rapporti modali, introduciamo l'opcode (versione UDO) **Modalsynth** :

```
Modalsynth ifreq,idamp,instrument,inumber,itab
```

- ifreq : modal ratios
- idamp : Q del risonatore
- instrument : numero dello strumento da controllare
- inumber : numero di frequenze modali da generare
- itab : tabella con modal ratios

Esempio Cap.10.7 _Modalsynth

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

gitibetan_bowl    ftgen 9,0,-7,- \
2,1,2.77828,5.18099,8.16289,11.66063,15.63801,19.99

#include   "Modalsynth.h"
;Modalsynth      ifreq,idamp,instrument,inumber,itab

instr 1

ifreq =      p4      ;modal ratios
idamp =      p5      ;Q del risonatore
instrument = 2       ;controlla lo strumento 2
inumber =     p6      ;numero di frequenze modali da generare
itab =        p7      ;tabella con modal ratios
Modalsynth ifreq,idamp,instrument,inumber,itab

endin

instr 2      ;generatore di risonatore

itable      table p6,p7 ;tabella con modal ratios
a1,a2 diskin2  "drums_loop.aif",1,0,1 ;segnaile in ingresso
areso mode a1,p4*itable,p5  ;risonatore
aout balance areso,a1    ;controlla il livello tra i due segnali
```

```

outs  aout*.2,aout*.2

endin

</CsInstruments>
<CsScore>
;p4    =      frequenza base
;p5    =      Q
;p6    =      numero di modal ratios
;p7    =      tabella

i1    0      4      250    100    7      9
s
i1    0      4      450    10     7      9
s
i1    0      4      150    100    7      9
s
i1    0      4      650    50     7      9
s

</CsScore>
</CsoundSynthesizer>
```

Scanned synthesis

La sintesi Scanned rappresenta una variante della sintesi per modelli fisici e si tratta di uno dei metodi sintetici più recenti. Sviluppato da Bill Verplank, Rob Shaw e Max Mathews tra il 1998 e il 1999 all' Interval Research. La scanned synthesis è basata sul modello di una particolare corda vibrante costituita da una serie di masse e molle collegate tra loro e messa in vibrazione con varie modalità (anche questa tecnica è basata sul principio massa-molla-smorzatore).

Immaginiamo un modello di masse collegate tra loro da molle messe in vibrazione da frequenze al di sotto dei 15 hz, quindi in un campo non udibile, per renderle udibili il sistema viene scansionato periodicamente e la forma del modello viene trasformata in una forma d'onda. Il pitch della forma d'onda è determinato dalla velocità di scansione, quindi la generazione sonora e il controllo del pitch avvengono su due processi differenti.

Il risultato è una lenta forma d'onda dinamica vibrante con il controllo determinato da numerosi parametri. La scansione del modello di masse-molle avviene seguendo una traiettoria definita, è possibile scansionare un modello con varie scansioni simultanee. In Csound la rappresentazione del modello è rappresentata da una matrice in cui è possibile definire l'ordine in cui le masse sono collegate tra loro, un esempio di matrice o griglia che definisce le connessioni :

		1		2		3		4	
<hr/>									
1		0		1		0		0	
<hr/>									
2		0		0		1		0	
<hr/>									
3		0		0		0		1	
<hr/>									
4		0		0		0		0	
<hr/>									

Viene utilizzato il valore di zero quando le due masse non sono connesse, nell'esempio grafico abbiamo un sistema con i seguenti collegamenti : (1,2) (2,3) (3,4) .

Csound offre diversi opcode per la sintesi scanned, i principali sono due :

- **scanu** : definisce il modello massa-molla da scansionare generando una waveform
- **scans** : segue la traiettoria intorno al modello e genera audio in uscita

```
scantu init,irate,ifnvel,ifnmass,ifnstif,ifnctr,ifndamp,kmass, \
kstif,kcentr,kdamp,ileft,iright,kpos,kstrngth,ain,idisp,id
```

- init : posizione iniziale della massa
- irate : intervallo di tempo tra i successivi aggiornamenti della massa, con valori grandi la corda verrà aggiornata con velocità bassa e piccole variazioni timbriche.
- ifnvel : tabella contenente la velocità iniziale della massa, potrebbe avere lo stesso valore del numero di masse
- ifnmass : tabella che contiene la massa di ogni massa del modello rappresentato
- ifnstif : tabella che contiene la rigidezza delle molle per ogni connessione
- ifnctr : tabella con la forza centrale di ogni massa
- ifndamp : tabella con fattore di smorzamento per ogni massa, potrebbe avere lo stesso valore del numero di masse
- kmass : ridimensiona le masse
- kstif : ridimensiona la rigidezza delle molle
- kcentr : ridimensiona la forza centrale
- kdamp : ridimensiona lo smorzamento
- ileft ,iright,kpos : posizione del martello sulla corda
- kstrngth : quantità di forza del martello
- ain : audio input alla velocità delle masse

```
ares scans kamp,kfreq,ifn,id,iorder
```

- kamp : ampiezza
- kfreq : frequenza
- ifn : tabella per la traiettoria della scansione sul modello
- id : numero identificativo della forma d'onda prodotta da scanu, i due id devono coincidere
- iorder(opzionale) : valore di interpolazione interno, range tra 1 e 4.

il prossimo strumento introduce la sintesi scanned con il controllo midi del pitch, il modello di sintesi viene costruito utilizzando **scantu** con un solo processo di scansione realizzato da **scans**, la matrice utilizzata dalle molle è contenuta nel file "string-128".

Esempio Cap.10.8_scanned

```
<CsoundSynthesizer>
<CsOptions>
-odac1 -M1 -+rtmidi=virtual
</CsOptions>
<CsInstruments>
```

```
sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
```

```
instr 1
```

```
iamp    ampmidi    12000
ifrq    cpsmidi
irate =     .01
ifnvel   =      6
ifnmass   =      2
ifnstif   =      3
ifnctr   =      4
```

```

ifndamp      =      5
kmass =      2
kstif =      .1
kcentr      =      .1
kdamp =     -.01
ileft =      .5
iright =      .5
kpos =       0
ky =        0
ain =       0
idisp =      1
id =        2

scanu 1,irate,ifnvel,ifnmass,ifnstif,ifnctr,ifndamp,kmass,kstif, \
kcentr,kdamp,ileft,iright,kpos,ky,ain,idisp,id

a1      scans iamp,ifrq,7,id

outs  a1,a1

endin

</CsInstruments>
<CsScore>
f1 0 128 10 1
; Initial condition
;f1 0 128 7 0 64 1 64 0
; Masses
f2 0 128 -7 1 128 1
; Spring matrices
f3 0 16384 -23 "string-128"
; Centering force
f4 0 128 -7 0 128 2
; Damping
f5 0 128 -7 1 128 1
; Initial velocity
f6 0 128 -7 0 128 0
; Trajectories
;f7 0 128 -5 .001 128 128
f7 0 128 -23 "spiral-8,16,128,2,1over2"
f0 3600
</CsScore>
</CsoundSynthesizer>
```

pur essendo una tecnica di sintesi relativamente recente,vi erano fin da subito notevoli difficoltà per il controllo in tempo reale a causa della complessità dell'algoritmo usato,il problema è del tutto risolto con le moderne cpu attuali in cui possiamo programmare anche multiple istanze di scansione per ogni modello creato.

Adesso un esempio leggermente più complesso in cui il modello viene scansionato da quattro istanze di **scans**,ognuna con tabelle differenti per la traiettoria di scansione e trasposizione delle frequenze.Il controllo midi non si limiterà unicamente alla frequenza ma controllerà la massa,la rigidezza delle molle,il rate e il fattore di smorzamento.

Esempio Cap.10.9_quadra scans

```
<CsoundSynthesizer>
<CsOptions>
-odac1 -M1 -+rtmidi=virtual
</CsOptions>
<CsInstruments>
```

```

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2

instr 1

iamp    ampmidi      12000
ifrq    cpsmidi
irate   ctrl7 1,1,.01,.2 ;rate
kmass   ctrl7 1,2,5,20; mass
kstif   ctrl7 1,3,.01,1; stiffness
kcentr  ctrl7 1,4,0,1; centering
kdamp   ctrl7 1,5,-1,.01; damping

ifnvel    =     6
ifnmass   =     2
ifnstif   =     3
ifnctr    =     4
ifndamp   =     5
ileft     =     .5
iright    =     .5
kpos      =     0
ky        =     0
ain       =     0
idisp     =     1
id        =     2

scanu 1,irate,ifnvel,ifnmass,ifnstif,ifnctr,ifndamp,kmass, \
kstif,kcentr,kdamp,ileft,iright,kpos,ky,ain,idisp,id

a1    scans iamp,ifrq,7,id    ;quattro istanze di scans con trasposizioni
a2    scans iamp,ifrq*2,8,id
a3    scans iamp,ifrq,9,id
a4    scans iamp,ifrq*1.5,10,id

asum1 sum  a2,a3 ;left e right con scansioni differenti
asum2 sum  a4,a1

kadsr mxadsr      1,1,.2,1    ;inviluppo
outs  asum1*kadsr,asum2*kadsr

endin

</CsInstruments>
<CsScore>
f1 0 128 10 1
; Initial condition
;f1 0 128 7 0 64 1 64 0
; Masses
f2 0 128 -7 1 128 1
; Spring matrices
f3 0 16384 -23 "cylinder-128,8"
; Centering force
f4 0 128 -7 0 128 2
; Damping
f5 0 128 -7 1 128 1
; Initial velocity
f6 0 128 -7 0 128 0
; Trajectories
f7 0 128 -5 .001 128 128
f8 0 128 -7 0 128 128
f9 0 128 -5 1 32 127 32 64 32 127 32 1

```

```
f10 0 128 -7 0 64 127 64 0
```

```
f0 3600  
</CsScore>  
</CsoundSynthesizer>
```

La prossima interfaccia grafica integra la possibilità di scegliere le matrici di connessione tramite l'oggetto FLcount, si tratta di un esempio di sintesi scanned con filtraggio (l'oggetto FLjoy controllerà frequenza e risonanza di un filtro moog), riverbero (di cui tratteremo l'utilizzo in modo approfondito nei prossimi capitoli) e componente di variazione random del pitch con **jspline**. Pur essendo una tecnica di sintesi corposa e ricca, una via interessante di sperimentazione è proprio quella delle tecniche di filtraggio e processamento del segnale, il risultato da vita a texture dinamiche di grande suggestione.

Esempio Cap.10.10_Scanned synthesizer (gui)

```
<CsoundSynthesizer>  
<CsOptions>  
-odac1 -M1 -+rtmidi=virtual  
</CsOptions>  
<CsInstruments>  
  
sr = 44100  
kr = 4410  
ksmps = 10  
nchnls = 2  
  
garev1      init  0  
garev2      init  0  
  
FLpanel      "Scanned synthesizer",600,450,150,100,1,-1,-1  
FLvkeybd     "keyboard.map",560,100,15,340  
gkOnOff,ihOnOff  FLbutton "On/Off",1,0,22,90,25,10,320,0,1,0,-1  
  
;ihandle FLvalue "label",iwidth,iheight,ix,iy  
ival7 FLvalue  "",50,30,10,270  
ival8 FLvalue  "",50,30,80,270  
ival9 FLvalue  "",50,30,150,270  
ival10    FLvalue "",50,30,220,270  
ival11    FLvalue "",50,30,290,270  
ival12    FLvalue "",50,30,360,270  
imin = 100  
imax = 1  
iexp = 0  
iheight = 30  
iwidth = 500  
ix = 20  
iy = 20  
iwidth2 = 30  
  
;kout,ihandle FLslider "label",imin,imax,iexp,itype,idisp,iwidth,iheight,ix,iy  
gk1,ih1 FLslider "rate",.01,2,iexp,24,ival7,iwidth2,iheight+200,ix,iy  
gk2,ih2 FLslider "mass",1,10,iexp,24,ival8,iwidth2,iheight+200,ix+70,iy  
gk3,ih3 FLslider "stiffness",.01,.5,iexp,24,ival9,iwidth2,iheight+200,ix+140,iy  
gk4,ih4 FLslider "centering",0.1,1,iexp,24,ival10,iwidth2,iheight+200,ix+210,iy  
gk5,ih5 FLslider "jitter",.1,20,iexp,24,ival11,iwidth2,iheight+200,ix+280,iy  
gk6,ih6 FLslider "reverb",.1,.95,iexp,24,ival12,iwidth2,iheight+200,ix+350,iy  
gkx,gky,gihx,gihy FLjoy "Moog filter",100,20000,.1,.9,-1,-1,-1,-1,140,140,430,20  
gkspring,ih7 FLcount    "spring matrix",11,15,1,1,1,140,30,430,212,-1,1,0,1  
  
FLpanelEnd  
FLrun  
;valori iniziali oggetti grafici
```

```

FLsetVal_i 0.1,ih1
FLsetVal_i 5,ih2
FLsetVal_i .02,ih3
FLsetVal_i 0.5,ih4
FLsetVal_i .1,ih5
FLsetVal_i .1,ih6
FLsetVal_i 11,ih7
FLsetVal_i 6000,gihx
FLsetVal_i .1,gihy

;scanned instrument
instr 1
iamp ampmidi      10000
ifrq cpsmidi
irate =      i(gk1) ;rate
kmass =      gk2; mass
kstif =      gk3; stiffness
kcentr =     gk4; centering
kdamp =     -.01; damping
ifnvel =     6
ifnmass =    2
ifnstif =    i(gkspring)
ifnctr =     4
ifndamp =    5
ileft =      .5
iright =     .5
kpos =       0
ky =        0
ain =       0
idisp =     1
id =        2

scanu 1,irate,ifnvel,ifnmass,ifnstif,ifnctr,ifndamp,kmass, \
kstif,kcentr,kdamp,ileft,iright,kpos,ky,ain,idisp,id

kjit jspline      gk5,1,20      ;variazione pitch random

a1   scans iamp,(ifrq*1.1)+kjit,7,id      ;quattro scans con trasposizioni
a2   scans iamp,ifrq+kjit,8,id
a3   scans iamp,(ifrq*.9)+kjit,9,id
a4   scans iamp,ifrq+kjit,10,id

afilt1   moogvcf2   a1+a4,gkx,gky      ;moog filter
afilt2   moogvcf2   a3+a2,gkx,gky

kdeclick linsegr    0,.1,1,.3,.5,.2,0
outs afilt1*kdeclick,afilt2*kdeclick

garev1 =      afilt1+garev1
garev2 =      afilt2+garev2

endin

instr 2      ;riverbero

a1,a2 reverbsc  garev1,garev2,gk6,12000,sr,0.5,1

outs a1,a2
garev1 =      0
garev2 =      0

endin

```

```

;attiva/disattiva lo strumento
instr 100

if      gkOnOff      = 0      then ;on/off strumento
turnoff2 1,8,0      ;instr 1
endif

endin

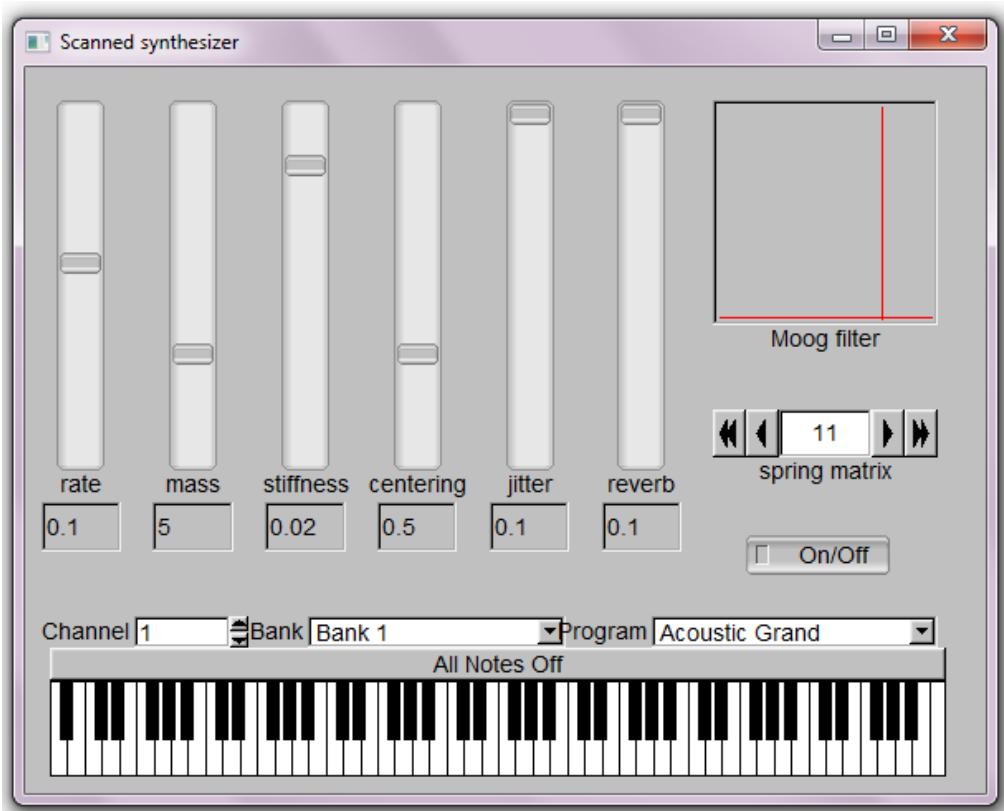
</CsInstruments>
<CsScore>

; Initial condition
f1    0      128    10     1
; Masses
f2    0      128    -7     1      128    1
; Spring matrices
f11   0      16384  -23   "circularstring-128"
f12   0      16384  -23   "grid-128,8"
f13   0      16384  -23   "cylinder-128,8"
f14   0      16384  -23   "torus-128,8"
f15   0      16384  -23   "string-128"
; Centering force
f4    0      128    -7     0      128    2
; Damping
f5    0      128    -7     1      128    1
; Initial velocity
f6    0      128    -7     0      128    0
; Trajectories
f7    0      128    -5     .001   128    128
f8    0      128    -7     0      128    128
f9    0      128    -5     1      32     127    32     64     32
f10   0      128    -7     0      64     127    64     0      127    32     1

f0 3600
i2 0 3600
i100 0 3600

</CsScore>
</CsoundSynthesizer>

```



Cap.11 Spectral Processing

11.1 phase vocoder

11.2 pitch shifting,time stretching,morphing

11.3 harmonizer,spectral blur,spectral filter,spectral delay

Phase vocoder

Il phase vocoder è una delle tecniche di analisi e risintesi più usate nella storia della computer music, il suo funzionamento è basato sul principio di *Fourier* secondo cui un segnale periodico può essere rappresentato dalla somma di onde di tipo sinusoidale dette armoniche, un segnale rappresentato da una serie di armoniche viene chiamato spettro, ognuna di queste armoniche ha una propria frequenza, ampiezza e fase. In pratica viene analizzato un campione audio, i dati ottenuti vengono utilizzati per ri-sintetizzare il suono, il vantaggio è quello di poter manipolare le frequenze o ampiezze delle sinusoidi ottenute dalla scomposizione, creando nuove sonorità.

Nello specifico, la tecnica usata nel phase vocoder viene chiamata STFT (*short time Fourier transform*), il principio è quello di realizzare l'analisi dividendo il segnale in piccole finestre temporali (windowing) di pochi millisecondi, ogni finestra ha un suo inviluppo che può essere di vari tipi standard (rettangolare, triangolare, Hanning, Gauss, ecc.). Il risultato di questa analisi a brevi finestre viene analizzato e i dati ottenuti (**frame**) forniscono la descrizione dei mutamenti temporali del suono analizzato, la grandezza del frame influenzera la ri-sintesi finale. Il processo finale di risintesi viene chiamato FFT (*fast Fourier transform*).

In pratica il frame rappresenta il numero di campioni con cui il segnale viene rappresentato, valori di frame alti (deve sempre essere una potenza di due, ad esempio 4096) permetteranno di ottenere una ri-sintesi molto fedele al suono originale analizzato (si consiglia di utilizzare sempre valori di frame superiori o uguali alla frequenza di campionamento del segnale divisa per 100 :

$$\text{frame size} = \text{sample rate}/100$$

(nel caso di una frequenza pari a 44100 Hz la grandezza del frame dovrebbe essere superiore a 441).

Nelle storiche versioni di Csound, il processo di analisi e risintesi richiedeva enormi quantità di calcolo, per questo motivo i primi opcode (pvoc, pvanal, ecc) realizzavano il processo dividendo in due fasi distinte l'analisi e la risintesi. In pratica viene scritta un'orchestra in grado di analizzare un file audio (generalmente mono) ottenendo come risultato un nuovo file con i dati dell'analisi, una seconda orchestra si occupa di leggere il file ottenuto dall'analisi e ri-sintetizzarlo.

In questo capitolo ci occuperemo invece delle moderne tecniche di analisi e risintesi in tempo reale, la versione di Csound5 introduce una serie nuova di opcode (realizzati dal musicista-ricercatore *Victor Lazzarini*) chiamati *pvs opcodes* che permettono di abbattere i limiti imposti dal lavoro in tempo differito dei vecchi opcode.

Riassumiamo in modo schematico un processo di analisi e risintesi :

- prima fase : un campione audio viene analizzato
- i dati ottenuti vengono lasciati inalterati o processati
- i dati finali passano ad un processo di risintesi.

Che tipo di processamento del segnale possiamo ottenere con il phase vocoder? tra i tanti, possiamo variare la durata del suono analizzato senza modificare l'intonazione, oppure modificare l'intonazione senza variare la durata (tutti procedimenti impossibili con i più semplici opcode come diskin2, ecc).

Introduciamo adesso due opcode per il phase vocoder, si tratta di **pvsanal** (*Richard Dobson 2001*) e il più recente **pvstanal** (*Victor Lazzarini 2010*), quest'ultimo è in grado di modificare pitch e tempo di lettura in tempo reale. La sintassi di **pvanal** :

```
fsig pvsanal ain, ifftsize, ioverlap, iwinsize, iwintype
```

I parametri riguardano la grandezza della FFT in campioni (sempre una potenza di due), il numero di sovrapposizione delle finestre, la grandezza di una finestra di analisi e il suo tipo di inviluppo (0 = Hamming window, 1 = von Hann window).

Come avviene la risintesi in tempo reale? per mezzo di **pvsynth**, in grado di ri-sintetizzare usando la FFT.

Esempio Cap.11.1_pvsanal

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1

instr 1

asample diskin2 "guitar.wav",1,0,1

ifftsize      =      1024 ;FFT size in campioni
ioverlap      =      256 ;numero di sovrapposizioni delle finestre
iwinsize      =      1024 ;size della finestra di analisi
iwintype      =      0    ;inviluppo Hamming

fftin pvsanal      asample,ifftsize,ioverlap,iwinsize,iwintype      ;phase vocoder
aout  pvsynth      fftin ;risintesi
out   aout
endin

</CsInstruments>
<CsScore>
i1 0 10
</CsScore>
</CsoundSynthesizer>
```

Come possiamo ascoltare, il risultato della risintesi è molto vicino alla qualità sonora del campione analizzato, il prossimo esempio utilizza **pvstanal** con cui ottenere le tecniche di time stretching e pitch shifting, in questo caso il file audio da analizzare sarà contenuto in una tabella :

```
fsig pvstanal ktimescal,kamp,kpitch,ktab,[kdetect,kwrap,ioffset,ifftsize,etc..])
```

in questa versione di phase vocoder il valore di di FFT-size viene dato come argomento opzionale(di default 2048), tuttavia è possibile indicare un valore diverso purchè sia sempre una potenza di due.

Esempio Cap.11.2_pvstanal(gui)

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2

FLpanel      "pvstanal",550,250,300,300,4,-1,1

gkOnOff,ihOnOff  FLbutton "On/Off",1,0,22,100,30,140,180,0,1,0,-1
gk4,ihb4  FLbutton "Exit",1,0,21,100,30,20,180,0,2,0,0
gk1,gih1  FLslider "amplitude",0.1,20000,0, 5,-1, 500,30, 10, 20
```

```

gk2,gih2 FLslider "speed",.1,20,-1, 5,-1, 500,30, 10, 70
gk3,gih3 FLslider "pitch",.1,20,-1, 5,-1, 500,30, 10, 120

FLpanel_end
FLrun
;valori iniziali
FLsetVal_i 1000,gih1
FLsetVal_i 2,gih2
FLsetVal_i 2,gih3

instr 1

if      gkOnOff      = 0      then ;on/off strumento
turnoff
endif

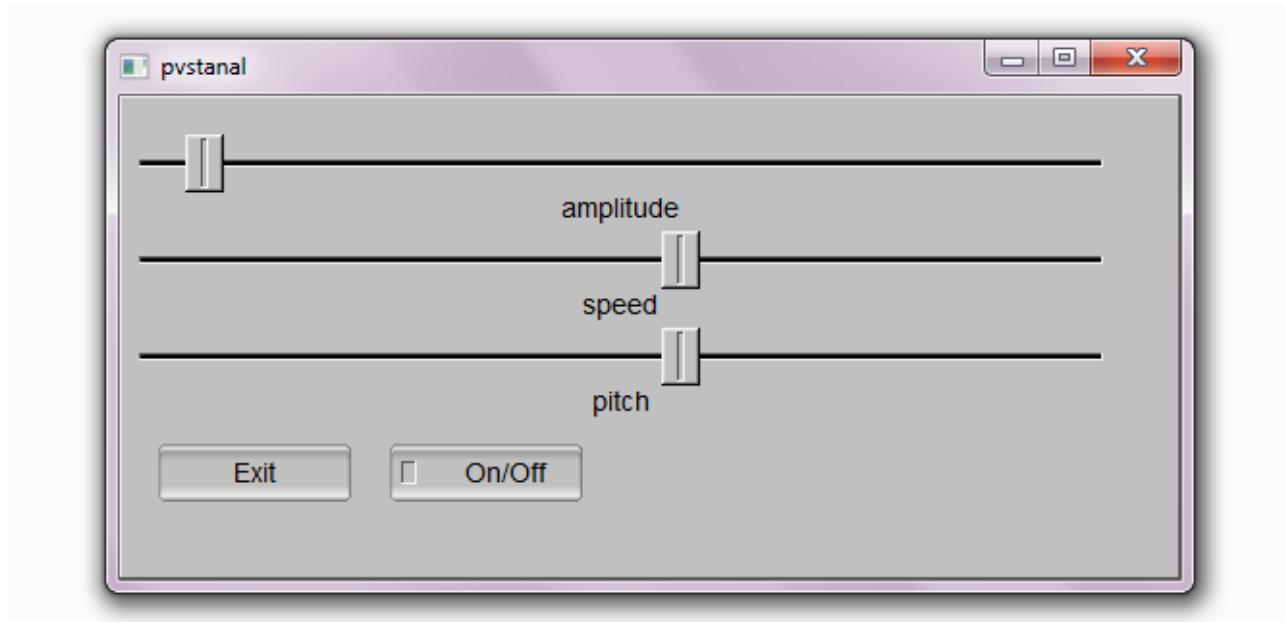
kamp   =      gk1    ;amppiezza
kspeed   =      gk2    ;velocità del file
kpitch   =      gk3    ;intonazione
ifile   =      1      ;tabella con file audio
fsig   pvstanal   kspeed,kamp,kpitch,ifile
aout   pvsynth   fsig   ;risintesi
outs   aout,aout

endin

instr 2
exitnow
endin

</CsInstruments>
<CsScore>
f1      0      0      1      "guitar.wav"      0      0      0
i1      0      3600
</CsScore>
</CsoundSynthesizer>

```



Una delle tecniche più interessanti di processamento spettrale è l'interpolazione tra due campioni analizzati, il morphing si realizza processando i dati di analisi prima della risintesi, **pvsmorph** prende come argomenti di ingresso i due file di analisi :

```
fsig pvsmorph fftin1,fftin2,kamp,kfreq
```

L'interpolazione tra i valori di ampiezza e frequenza dei due campioni deve essere in un range compreso tra 0 e 1; l'esempio seguente effettua il morphing passando dal campione n°1 al campione n°2 per ritornare al primo campione nel tempo p3 della score.

Esempio Cap.11.3_morphing

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 1

instr 1

asnd1 diskin2      "debussy.wav",1,0,1
asnd2 diskin2      "guitar.wav",1,0,1

fftin1      pvsanal      asnd1,1024,256,1024,1; fft-analysis of file
fftin2      pvsanal      asnd2,1024,256,1024,1; fft-analysis of file

kfreq linseg      0,p3/2,1,p3/2,0
kamp  linseg      0,p3/2,1,p3/2,0

fmorph      pvsmorph      fftin1,fftin2,kamp,kfreq      ;interpolazione

aout  pvsynth      fmorph      ;risintesi

out   aout
endin

</CsInstruments>
<CsScore>
i1    0    20
</CsScore>
</CsoundSynthesizer>
```

Adesso un esempio interattivo con stretching indipendente per ogni campione :

Esempio Cap.11.4_morphing(gui)

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2

FLpanel      "morphing",560,320,300,300,4,-1,1
```

```

gkOnOff,ihOnOff    FLbutton    "On/Off",1,0,22,100,30,140,260,0,1,0,-1
gkrnd,ihrnd FLbutton    "random",1,0,22,100,30,260,260,0,3,0,-1
gkexit,ihb4 FLbutton    "Exit",1,0,21,100,30,20,260,0,2,0,0

;sample1
gkamp1,ih1  FLslider    "amplitude",0.1,20000,0, 5,-1, 220,30, 10, 20
gkspeed1,gih2   FLslider    "speed",.1,20,-1, 5,-1, 220,30, 10, 70
gkpitch1,gih3   FLslider    "pitch",.1,20,-1, 5,-1, 220,30, 10, 120

;sample2
gkamp2,ih4  FLslider    "amplitude",0.1,20000,0, 5,-1, 220,30, 320, 20
gkspeed2,gih5   FLslider    "speed",.1,20,-1, 5,-1, 220,30, 320, 70
gkpitch2,gih6   FLslider    "pitch",.1,20,-1, 5,-1, 220,30, 320, 120

gkmorph,gihmorph  FLslider    "morphing",0,1,0, 5,-1, 530,30, 10, 180

FLpanel_end
FLrun
;valori iniziali
FLsetVal_i 1000,ih1
FLsetVal_i 2,gih2
FLsetVal_i 2,gih3
FLsetVal_i 1000,ih4
FLsetVal_i 2,gih5
FLsetVal_i 2,gih6

instr 1

if      gkOnOff      = 0      then ;on/off strumento
turnoff
endif

fsig1 pvstanal     gkspeed1,gkamp1,gkpitch1,1      ;fft sample1
fsig2 pvstanal     gkspeed2,gkamp2,gkpitch2,2      ;fft sample2

fmorph      pvsmorph     fsig1,fsig2,gkmorph,gkmorph  ;morphing

aout  pvsynth      fmorph      ;risintesi

outs  aout,aout

endin

instr 2
exitnow
endin

instr 3      ;random speed,pitch

if      (gkrnd == 1)      kgoto CONTINUE
if      (gkrnd == 0)      kgoto STOP

CONTINUE:

krand randomi     .1,2,2
krand2     randomi .1,2,1
krndmorph  randomi 0,1,1
FLsetVal  1,krand,gih2 ;random speed
FLsetVal  1,krand2,gih3 ;random pitch
FLsetVal  1,krand,gih5 ;random speed
FLsetVal  1,krand2,gih6 ;random pitch

```

```

FLsetVal    1,krndmorph,gihmorph ;morphing

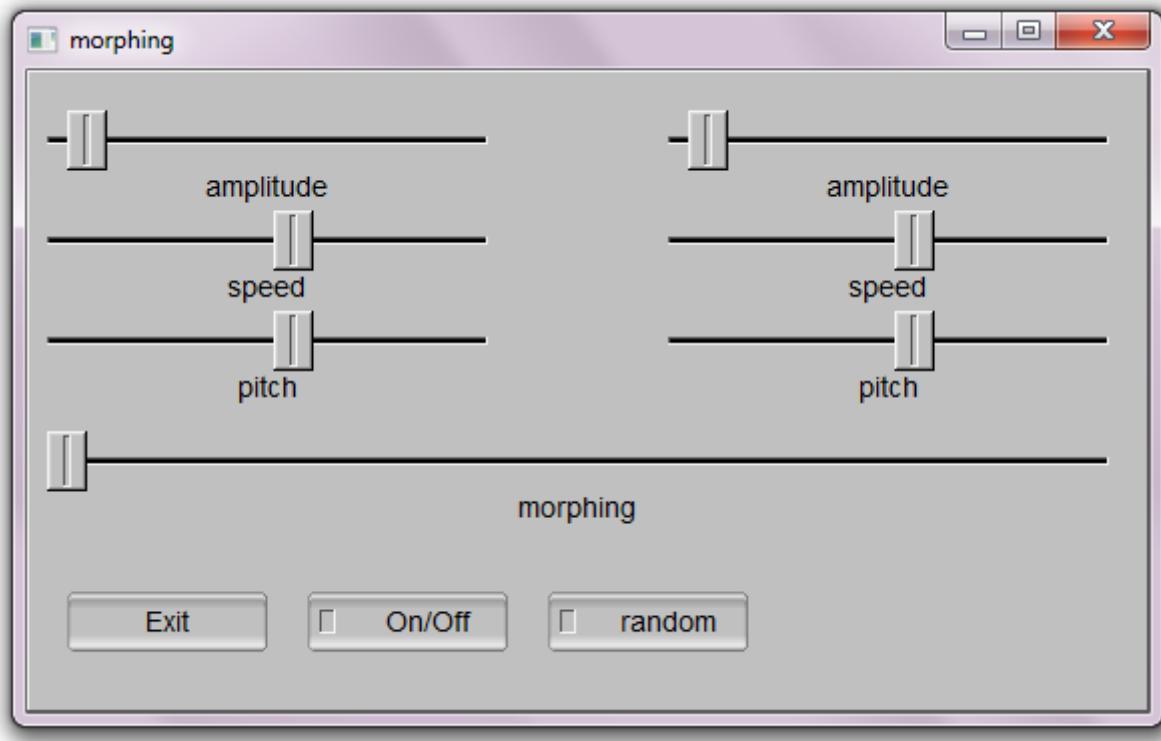
STOP:

endin

</CsInstruments>
<CsScore>
f1    0      0      1      "guitar.wav"      0      0      0
f2    0      0      1      "debussy.wav"     0      0      0

i1    0      3600
</CsScore>
</CsoundSynthesizer>

```



(nella versione di Csound5 per Osx si verifica un piccolo bug utilizzando automazioni random di un oggetto Fltk, pertanto l'esempio precedente funziona in modo completo solo su sistemi operativi windows, il pulsante "random" genera interruzioni del segnale audio variabili su Osx).

harmonizer,spectral blur,spectral filter,spectral delay

In questa sezione verranno prese in esame alcune tecniche di processing spettrale utilizzando come base le tecniche viste precedentemente, il procedimento è sempre quello di aprire un file audio in lettura, svolgere l'analisi con il phase vocoder, modificare con varie tecniche i dati ottenuti dall'analisi, infine mandare il segnale all'ingresso di **pvsynth** per la risintesi.

Una delle tecniche più semplici di processamento di dati spettrali è quella della trasposizione, la sorgente sonora di **pvsanal** viene elaborata da **pvscale** in grado di realizzare una trasposizione della frequenza. Varie istanze di trasposizione possono essere usate per realizzare un harmonizer (partendo generalmente da un segnale monofonico); per un miglior risultato si consiglia di utilizzare una finestra di analisi con inviluppo di tipo Hanning (iwintype = 0 per pvsanal).

Esempio Cap.11.5_harmonizer

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 1

instr 1

asample diskin2    "debussy.wav",1,0,1
iwintype    =      1      ;0 = Hanning window
fftin pvsanal    asample,1024,256,1024,iwintype      ;fft-analysis of file

;harmonizer
fsig1 pvscale    fftin,.5
fsig2 pvscale    fftin,1.2
fsig3 pvscale    fftin,1.6
fsig4 pvscale    fftin,3

aout1 pvsynth    fsig1 ; risintesi
aout2 pvsynth    fsig2
aout3 pvsynth    fsig3
aout4 pvsynth    fsig4

aharmon      sum    aout1,aout2,aout3,aout4

out    aharmon
endin

</CsInstruments>
<CsScore>
i1 0 20
</CsScore>
</CsoundSynthesizer>
```

Provate a modificare la sorgente d' ingresso con un segnale microfonico esterno :

```
ainput in
iwintype    =      1      ;0 = Hanning window
fftin pvsanal    ainput,1024,256,1024,iwintype ;fft-analysis of file
```

per attivare la porta audio in ingresso di Csound è necessario il flag :

```
<CsOptions>
-iadc -odac
</CsOptions>
```

iadc e odac richiedono un numero (esempio : -iadc1 -odac1) relativo ad ingressi e uscite della propria scheda audio, per conoscere il numero corretto (nel caso di macchine windows si consiglia di aprire la porta con driver Asio) basta indicare un numero di porta audio errato, ad esempio -iadc100, in questo caso Csound genererà un messaggio di errore (proprio perché la nostra scheda audio difficilmente disporrà di 100 ingressi fisici!), nella finestra di console saranno visualizzati i corretti numeri di porte audio disponibili sul proprio sistema.

Processando un segnale esterno in tempo reale potrebbero verificarsi problemi come dropouts del segnale in uscita, questo è causato dal buffer di memoria troppo piccolo per svolgere un'operazione così complessa, per ovviare a questo problema si consiglia un flag di comando simile a :

```
<CsOptions>
-iadc -odac -b2000      -B2000
</CsOptions>
```

Introduciamo un altro procedimento spettrale conosciuto con il nome di *blur*, letteralmente significa “sfocatura”, è un processo in cui si analizza il flusso audio in tempo reale e si applica una media per la frequenza e l'ampiezza di ogni canale, dal punto di vista sonoro il risultato è simile a quello di un filtro passa-basso che smussa le ampiezze e frequenze nel tempo .

```
fsig pvsblur fsigin, kblurtime, imaxdel
```

- fsigin : ingresso file analisi
- kblurtime : tempo in secondi durante il quale le finestre saranno mediate (amp/freq), range 0 e 1
- imaxdel : tempo di allocazione della memoria durante il processo spettrale.

Esempio Cap.11.6.blur

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10
nchnls=2

FLpanel      "spectral blur", 550, 250, 300, 300, 4, -1, 1
gkOnOff, ihOnOff   FLbutton      "On/Off", 1, 0, 22, 100, 30, 140, 180, 0, 1, 0, -1
gk4, ihb4      FLbutton      "Exit", 1, 0, 21, 100, 30, 20, 180, 0, 2, 0, 0
gk1, ih1       FLslider      "amplitude", 0.1, 20000, 0, 5, -1, 500, 30, 10, 20
gk2, gih2      FLslider      "speed", .1, 20, -1, 5, -1, 500, 30, 10, 70
gk3, gih3      FLslider      "pitch", .1, 20, -1, 5, -1, 500, 30, 10, 120
gkbl, ihbl     FLknob       "blur", 0, 1, 0, 1, -1, 70, 410, 150

FLpanel_end
FLrun
;valori iniziali
FLsetVal_i 1000, ih1
FLsetVal_i 2, gih2
```

```

FLsetVal_i 2,gih3

instr 1

if      gkOnOff      = 0      then ;on/off strumento
turnoff
endif

kamp   =      gk1
kspeed   =      gk2
kpitch   =      gk3
ifile =      1
fsig  pvstanal    kspeed,kamp,kpitch,ifile

fftblur     pvsblur     fsig,gkbl,1

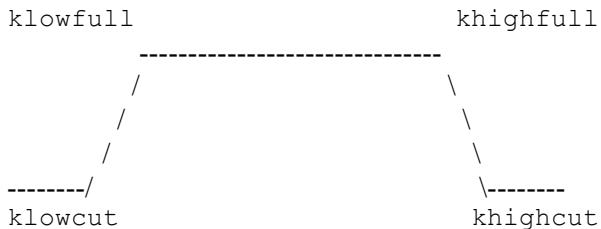
aout  pvsynth     fftblur
outs  aout,aout
endin

instr 2
exitnow
endin

</CsInstruments>
<CsScore>
f1    0      0      1      "guitar.wav"      0      0      0
i1    0      3600
</CsScore>
</CsoundSynthesizer>

```

Tra i vari processi spettrali di questa sezione vi sono anche i filtri spettrali di due tipi : passa-banda e respingi-banda (notch), si tratta di particolari filtri riservati unicamente al filtraggio di componenti di analisi con FFT, immaginiamo una banda passante delimitata da un inviluppo spettrale trapezoidale :



la sintassi dei due filtri è identica :

```

fsig pvsbandp fsigin,xlowcut,xlowfull,xhighfull,xhighcut
fsig pvsbandr fsigin,xlowcut,xlowfull,xhighfull,xhighcut

```

Esempio Cap.11.7_spectral-filter (notch)

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr=44100
kr=4410
ksmps=10

```

```

nchnls=2
0dbfs = 1

FLpanel      "pvstanal",550,250,300,300,4,-1,1
gkOnOff,ihOnOff   FLbutton "On/Off",1,0,22,100,30,140,180,0,1,0,-1

gk4,ihb4      FLbutton      "Exit",1,0,21,100,30,20,180,0,2,0,0
gk1,ih1       FLslider      "amplitude",0.1,2,0,5,-1,500,30,10,20
gk2,gih2       FLslider      "speed",.1,20,-1,5,-1,500,30,10,70
gk3,gih3       FLslider      "pitch",.1,20,-1,5,-1,500,30,10,120

FLpanel_end

FLpanel      "pvsbandr",250,250,870,300,4,-1,1
gk5,ih5       FLslider      "klowcut",1,1000,0,5,-1,220,30,10,20
gk6,ih6       FLslider      "klowfull",1,2000,0,5,-1,220,30,10,70
gk7,ih7       FLslider      "khightfull",1,2000,0,5,-1,220,30,10,120
gk8,ih8       FLslider      "khightcut",1,2000,0,5,-1,220,30,10,170

FLpanel_end

FLrun

;valori iniziali
FLsetVal_i    .1,ih1
FLsetVal_i    2,gih2
FLsetVal_i    2,gih3
FLsetVal_i   1000,ih5
FLsetVal_i   1200,ih6
FLsetVal_i   1200,ih7
FLsetVal_i   1000,ih8

instr 1

if      gkOnOff      = 0      then ;on/off strumento
turnoff
endif

kamp =      gk1
kspeed =     gk2
kpitch =    gk3
ifile =     1
fsig  pvstanal    kspeed,kamp,kpitch,ifile

klowcut =    gk5
klowfull =   gk6
khightfull = gk7
khightcut =  gk8
fbandp     pvsbandr    fsig,klowcut,klowfull,khightfull,khightcut ;notch filter

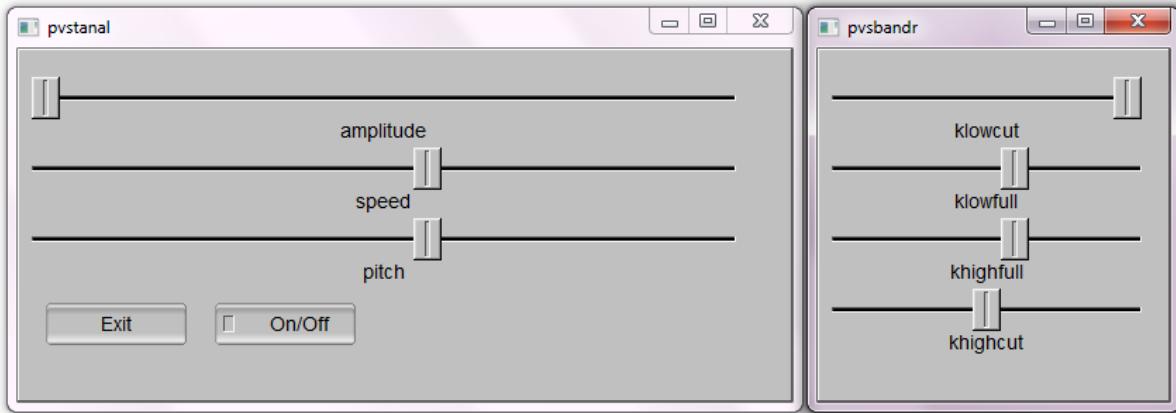
aout  pvsynth     fbandp
outs  aout,aout
endin

instr 2
exitnow
endin

</CsInstruments>
<CsScore>
f1    0      0      1      "guitar.wav"      0      0      0

```

```
i1      0      3600
</CsScore>
</CsoundSynthesizer>
```



Il prossimo esempio introduce il concetto di buffer spettrale,l'analisi fatta con il phase vocoder viene memorizzata in un buffer di memoria,uno specifico opcode leggerà porzioni di questo buffer partendo da valori temporali diversi,il risultato di questi buffer in lettura viene ri-sintetizzato.Questa tecnica permette di creare veri e propri ritardi spettrali (spectral delay),immaginiamo di avere quattro istanze di lettura del buffer,ogni lettura legge il file FFT partendo da punti temporali diversi (espressi in secondi),la sovrapposizione delle varie istanze di lettura creerà vere e proprie linee di ritardo.

```
ihandle, ktime  pvsbuffer fsig,ilen
```

- ktime : tempo di scrittura del buffer
- fsig : segnale FFT
- ilen : lunghezza del buffer in secondi

pvsbuffer viene letto da **pvsbufread** :

```
fsig pvsbufread  ktime,khandle[,ilo,ihi,iclear]
```

- ktime : prende in ingresso il ktime di pvsbuffer
- khandle : il nome del buffer in ingresso
- ilo,ihi(opzionali) : range di frequenze al quale leggere il buffer

il prossimo esempio utilizza due strumenti di lettura del buffer,ognuno con ingresso temporale diverso e range di frequenza diverso,il risultato viene risintetizzato e sommato al suono originale creando un particolare timbro di delay.

Esempio Cap.11.8_spectral delay1

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

asample diskin2 "guitar.wav",1,0,1
```

```

fftin pvsanal      asample,1024,256,1024,1; fft-analisi
ibufl,kt1    pvsbuffer   fftin,1; buffer di 1 secondo
fread1      pvsbufread  kt1-.2,ibufl,100,2000 ;lettura buffer con tempi diversi
fread2      pvsbufread  kt1-.3,ibufl,300,5000

aout1 pvsynth     fread1      ;risintesi
aout2 pvsynth     fread2

outs  aout1+asample*.5,aout2+asample*.5 ;suono iniziale + buffer
endin

</CsInstruments>
<CsScore>
i1 0 20
</CsScore>
</CsoundSynthesizer>
```

Il prossimo esempio utilizza otto istanze di lettura buffer,l'analisi viene fatta separando l'immagine stereo e con buffer di grandezza diversa,ogni strumento di lettura avrà ritardi variabili random e range di frequenze random.(attenzione ad utilizzare unicamente file stereo,per una versione mono è necessario modificare la sorgente in ingresso con **diskin2** lasciando un solo canale input).

Esempio Cap.11.9_spectral delay2

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 100
nchnls = 2

instr 1

aleft,aright diskin2 "slowharp.wav",1,0,1

fftin1      pvsanal      aleft,4092,256,4092,0; fft-analisi,left e right separati
fftin2      pvsanal      aright,4092,256,4092,0

ibufl,kt1    pvsbuffer   fftin1,4; 4-sec buffer
ibufl,kt2    pvsbuffer   fftin2,3; 3-sec buffer

;ksig randomh      kmin,kmax,rate
kdel1 randomh      .05,1,1      ;valori random per ritardi
kdel2 randomh      .1,1.2,2
kdel3 randomh      .5,1.4,3
kdel4 randomh      .1,2,4
kdel5 randomh      .1,2.2,5
kdel6 randomh      .1,2.4,6
kdel7 randomh      .1,2.5,7
kdel8 randomh      .1,2.6,8

kiminrnd      =      2000 ;valore base range
;valori random per range di frequenza massimo per pvsbufread
;valore massimo = (sample rate) / 2
```

```

khi1    randomi  kiminrnd, (sr)/2,1
khi2    randomi  kiminrnd, (sr)/2,2
khi3    randomi  kiminrnd, (sr)/2,3
khi4    randomi  kiminrnd, (sr)/2,4
khi5    randomi  kiminrnd, (sr)/2,5
khi6    randomi  kiminrnd, (sr)/2,6
khi7    randomi  kiminrnd, (sr)/2,7
khi8    randomi  kiminrnd, (sr)/2,8

fread1    pvsbufread  kt1-kdel1,ibuf1,500,khi1 ;8 punti di lettura diversi
fread2    pvsbufread  kt2-kdel2,ibuf2,500,khi2
fread3    pvsbufread  kt1-kdel3,ibuf1,500,khi3
fread4    pvsbufread  kt2-kdel4,ibuf2,500,khi4
fread5    pvsbufread  kt1-kdel5,ibuf1,500,khi5
fread6    pvsbufread  kt2-kdel6,ibuf2,500,khi6
fread7    pvsbufread  kt1-kdel7,ibuf1,500,khi7
fread8    pvsbufread  kt2-kdel8,ibuf2,500,khi8

fsall1    pvsmix     fread1,fread2      ;mixa due componenti spettrali
fsall2    pvsmix     fread3,fread4
fsall3    pvsmix     fread5,fread6
fsall4    pvsmix     fread7,fread8

aout1 pvsynth      fsall1      ;risintesi
aout2 pvsynth      fsall2
aout3 pvsynth      fsall3
aout4 pvsynth      fsall4

aL    sum   aout1,aout2
aR    sum   aout3,aout4

outs  aL*.6+aleft*.6,aR*.6+aright*.6      ;segnale iniziale + spectral delay
endin

</CsInstruments>
<CsScore>
i1 0 100
</CsScore>
</CsoundSynthesizer>
```

Cap.12 I processori di segnale

12.1 Delay

12.2 Riverbero

12.3 Chorus,Flanger,Phaser

12.4 Routing

12.5 Live electronics performance

I processori di segnale in ambito digitale prendono il nome di DSP (digital signal processor), si tratta di tutti quei processi algoritmici in grado di effettuare trasformazioni di un suono in ingresso, il procedimento si può riassumere nel seguente schema :

- sorgente analogica in ingresso (ad esempio un segnale microfonico)
- conversione analogico digitale del segnale iniziale (ADC)
- processamento (tra i tanti : chorus, flanger, phaser, riverbero, ecc)
- conversione digitale analogico (DAC)

Csound offre numerose unità elementari per la costruzione di qualunque tipo di processore di segnale, oltre ad alcuni opcode che elaborano il segnale in maniera già complessa.

Linee di ritardo

Una delle tecniche storiche di trattamento del segnale è quella di sovrapporre un segnale ad un'altra sorgente sonora che ritarda il segnale stesso, numerose istanze di linee di ritardo (delay) possono essere utilizzate per creare effetti di eco, oppure processori più complessi come l'effetto chorus.

Introduciamo gli opcode :

delayr
deltap (o deltapi, versione interpolata)
delayw

Il primo opcode **delayr** ha la funzione di catturare una porzione di suono all'interno della memoria (buffer), **deltap** leggerà questa porzione di memoria partendo da un determinato punto e infine il suono ritardato verrà scritto dall'opcode **delayw**. Nel prossimo esempio verranno costruite due linee di ritardo (left e right) con la tecnica di *feedback*, che consiste nel moltiplicare un segnale ritardato per un determinato fattore numerico, in pratica il suono verrà sommato a se stesso per un determinato numero di volte, è importante non usare valori di moltiplicazione troppo elevati (per evitare effetti di accumulo che porterebbe all'esplosione del segnale, si consiglia di usare valori di feedback compresi tra 0 e 1).

Esempio Cap.12.1_simple_delay

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1    pluck .6,440,440,0,1      ; segnale ingresso

;prima linea di ritardo
abuf1 delayr      1      ;buffer memoria
```

```

adel1      deltapi    p4      ;punto da cui prelevare il segnale
afeed1     =      p6 * adel1 + a1    ;numero di ripetizioni
delayw     afeed1      ;scrive il segnale in ritardo

;seconda linea di ritardo
abuf2 delayr    1
adel2      deltapi    p5
afeed2     =      p6 * adel2 + a1
delayw     afeed2

;uscita audio stereo
kdeclick   linseg      0,0.02,1,p3-0.05,1,0.02,0,0.01,0

outs  (a1+adel1*.5)*kdeclick,(a1+adel2*.5)*kdeclick ;suono originale+delay
endin

</CsInstruments>
<CsScore>
;p4,p5 = valori di ritardo (left,right)
;p6 = feedback (0 = no feedback)

i1  0    4    .1    .2    0
s
i1  0    4    .2    .4    .1
s
i1  0    4    .3    .5    1
s
i1  0    4    .1    .2    .9
s

</CsScore>
</CsoundSynthesizer>

```

Provate a sperimentare con valori di feedback diversi, ascoltiamo come il numero di ripetizioni diminuisce in ampiezza con *feedback* < 1, mentre il segnale con ritardo risulta invariato in ampiezza con *feedback* = 1.
Per realizzare multiple istanze di delay in modo assai più semplice viene in aiuto l'opcode **multitap**, in cui è possibile indicare un valore di gain per ogni linea di ritardo creata.

```
ares multitap asig [, itime1] [, igain1] [, itime2] [, igain2] [...]
```

Esempio Cap.12.2_multitap

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1    diskin2    "guitar.wav",1,0,1      ;segnale in ingresso

itime =     .2    ;linea di ritardo con gain
igain =     .5
itime2 =    .4

```

```

igain2      =      .5
itime3      =      .6
igain3      =      .5

amtap multitap    a1,itime,igain,itime2,igain2,itime3,igain3

kpan randomi    0,1,1 ;random pan
aL,aR pan2 amtap,kpan

outs a1+aL,a1+aR ;segnale originale+multitap delay

endin

</CsInstruments>
<CsScore>

i1    0    20

</CsScore>
</CsoundSynthesizer>
```

Riverberi

Csound implementa vari tipi di unità di riverbero fin dalle primissime release,nelle storiche versioni la scelta era limitata principalmente ad alcuni modelli (**nreverb**,**reverb** e **reverb2**) dalla qualità non eccelsa,pertanto faremo riferimento ai due modelli implementati solo nelle versioni recenti e dalla qualità audio decisamente superiore e paragonabili ai riverberi professionali che troviamo come plugin nei più comuni sequencer audio-midi.Csound offre anche la possibilità di costruire riverberi personalizzati per mezzo di unità elementari come i filtri **alpass** e **comb**.

La comunità dei musicisti legata a Csound utilizza oggi principalmente i seguenti opcode stereo :

```
aoutL, aoutR      freeverb      ainL,ainR,kRoomSize,kHFDamp
```

freeverb è un'unità basata su un sorgente in C++ di pubblico dominio,si tratta di un eccellente unità di riverberazione basata su otto filtri comb in parallelo per ogni canale,a cui seguono quattro unità di filtri allpass in parallelo.Il suo utilizzo è relativamente semplice e non richiede enormi risorse di calcolo.I suoi parametri permettono di definire il reverb size (range compreso tra 0 e 1) e il fattore di decadimento delle frequenze alte.

(freeverb contiene inoltre alcuni parametri facoltativi,fare riferimento alla documentazione ufficiale)

Un'altra unità di alto livello implementata in Csound5 è l'opcode **reverbsc** basato sul lavoro di *Sean Costello* :

```
aoutL, aoutR      reverbsc      ainL,ainR,kfblvl,kfco
```

alcuni settaggi per il parametro kfblvl (livello di feedback,range tra 0 e 1) :

- small room: kfblvl = 0.6
- small hall: kfblvl = 0.8
- large hall: kfblvl = 0.9
- infinite : kfblvl = 1
-

kfco = cutoff relative al filtro lowpass,range compreso tra 0 e israte/2,con valori bassi si ottiene un veloce decadimento delle frequenze alte.

Negli esempi seguenti l'unità di riverbero sarà scritta come uno strumento indipendente in orchestra,è pratica diffusa non utilizzare un riverbero in modalità seriale all'interno dello strumento da trattare,questo perché il

decadimento risulterebbe vincolato alla durata dello strumento stesso, creando quindi interruzioni dell'effetto di riverberazione. Pertanto verranno inizializzate delle variabili globali e lo strumento "riverbero" rimarrà in modalità "On" per tutta la durata della performance (midi o in differita), il suono risultante sarà quindi dato dal segnale in ingresso puro sommato al suono riverberato.

Esempio Cap.12.3_freeverb

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

gasig1      init  0      ;inizializzo le variabili globali
gasig2      init  0

instr 1

a1,a2 diskin2      "drums_loop.aif",1,0,1

outs  a1,a2 ;uscita del segnale non riverberato

vincr gasig1,a1    ;assegno la variabile a1 alla variabile globale
vincr gasig2,a2

endin

instr 2      ;reverb
;prende in ingresso le variabili globali
aL,aR freeverb    gasig1,gasig2,p4,0.35,sr,0

outs  aL,aR

clear gasig1,gasig2    ;azzera la memoria per evitare accumulo
endin

</CsInstruments>
<CsScore>

i1    0      4
i2    0      4      .3
s
i1    0      4
i2    0      4      .9
s

</CsScore>
</CsoundSynthesizer>
```

Esempio Cap.12.4_reverbsc

```

<CsSoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

gasig1      init  0
gasig2      init  0

instr 1

a1,a2 diskin2      "drums_loop.aif",1,0,1
outs  a1,a2

vincr gasig1,a1
vincr gasig2,a2

endin

instr 2      ;reverb

aL,aR reverbsc    gasig1,gasig2,p4,p5

outs  aL,aR

clear gasig1,gasig2

endin

</CsInstruments>
<CsScore>

;p4 = room size
;p5 = filtro frequenze alte

i1    0      4
i2    0      4      .6      12000 ;small "live" room
s
i1    0      4
i2    0      4      .8      12000 ;small hall
s
i1    0      4
i2    0      4      .9      12000 ;large hall
s
i1    0      4
i2    0      4      1      12000 ;infinite
s

</CsScore>
</CsSoundSynthesizer>

```

Considerando che i parametri room-size e filtro di reverbsc sono variabili a Krate, è possibile creare variazioni dinamiche nel tempo, il prossimo esempio controlla in modo random questi parametri, il risultato crea un interessante dialogo tra la sorgente originale e le riflessioni del suono create.

Esempio Cap.12.5_randomverb

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

gasig1      init  0
gasig2      init  0

instr 1

a1,a2 diskin2      "drums_loop.aif",1,0,1
outs a1,a2

vincr gasig1,a1
vincr gasig2,a2

endin

instr 2      ;reverb

ksize randomi    0,1,2
kfilt randomi    100,(sr)/2,10

aL,aR reverbsc   gasig1,gasig2,ksize,kfilt

kpan randomi     0,1,10      ;random pan
a1,a2 pan2  (aL+aR)*.5,kpan

outs a1,a2

clear gasig1,gasig2
endin

</CsInstruments>
<CsScore>

i1    0    100
i2    0    100

</CsScore>
</CsoundSynthesizer>
```

Adesso un altro utilizzo creativo del riverbero in cui il suono riflesso viene sottoposto ad un processamento spettrale con effetto blur.

Esempio Cap.12.6_spectralverb

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

gasig1      init  0
gasig2      init  0

instr 1

a1,a2 diskin2      "drums_loop.aif",1,0,1
outs  a1,a2

vincr gasig1,a1
vincr gasig2,a2

endin

instr 2      ;reverb

ksize randomi    0,1.5,20
kfilt randomi   100,(sr)/2,10

aL,aR reverbsc  gasig1,gasig2,ksize,kfilt

fftin1      pvsanal    aL,1024,256,1024,1      ;analisi phase vocoder
fftin2      pvsanal    aR,1024,256,1024,1

kblurdepth1 randomi   .1,.9,2
kblurdepth2 randomi   .1,.9,5

fftblur1    pvsblur    fftin1,kblurdepth1,1      ;random blur
fftblur2    pvsblur    fftin2,kblurdepth2,1

a1      pvsynth     fftblur1      ;risintesi
a2      pvsynth     fftblur2

outs  a1*.8,a2*.8

clear gasig1,gasig2

endin

</CsInstruments>
<CsScore>

i1      0      100
i2      0      100

</CsScore>
```

```
</CsoundSynthesizer>
```

Chorus,Flanger,Phaser

In questa parte del capitolo sui Dsp prenderemo in esame alcuni dei trattamenti del segnale più importanti e utilizzati.

L'effetto chiamato "chorus" è un tipo di processamento in cui il segnale viene sommato a numerose istanze del segnale stesso con leggere variazioni causate da linee di ritardo variabili e con deviazioni dell'intonazione. Per creare un effetto chorus esiste un opcode specifico in grado di realizzare un ritardo variabile nel tempo, si tratta di **vdelay**:

```
ares vdelay      asig,adel,imaxdel
```

- asig : segnale in ingresso
- adel : delay in millisecondi
- imaxdel : massimo valore del ritardo in millisecondi

un semplice esempio di chorus stereo, ci sono otto ritardi variabili (divisi tra left e right) in cui controllare la profondità dell'effetto e la velocità (simile ai pedalini chorus usati dai chitarristi elettrici):

Esempio Cap.12.6_chorus

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1    diskin2      "guitar.wav",1,0,1

idepth     =      p4      ;intensità dell'effetto
irate =      p5      ;velocità del ritardo variabile

k1    randomi      .01,idepth,irate
adel1 vdelay      a1,k1,1000

k2    randomi      .01,idepth*0.1,irate*0.1
adel2 vdelay      a1,k2,1000

k3    randomi      .01,idepth*0.2,irate*0.2
adel3 vdelay      a1,k3,1000

k4    randomi      .01,idepth*0.3,irate*0.3
adel4 vdelay      a1,k4,1000

k5    randomi      .01,idepth*0.4,irate*0.4
adel5 vdelay      a1,k5,1000

k6    randomi      .01,idepth*0.5,irate*0.5
adel6 vdelay      a1,k6,1000

k7    randomi      .01,idepth*0.6,irate*0.6
```

```

adel7 vdelay      a1,k7,1000
k8    randomi     .01,idepth*0.7,irate*0.7
adel8 vdelay      a1,k8,1000
aenv   linseg 0, 0.02, 1, p3 - 0.05, 1, 0.02, 0, 0.01, 0
outs  ((adel1+adel2+adel3+adel4)*aenv)*.3,((adel5+adel6+adel7+adel8)*aenv)*.3
endin

</CsInstruments>
<CsScore>

i1    0      4      .1      .3
s
i1    0      4      1       2
s
i1    0      4      4       4
s
i1    0      4      5       16
s
i1    0      4      18      18
s

</CsScore>
</CsoundSynthesizer>
```

La tecnica del **flanger** nasce storicamente in ambito analogico, si trattava di mettere in riproduzione due nastri dello stesso materiale registrato, uno dei due nastri in riproduzione veniva premuto in alcuni punti creando delle variazioni nella velocità di lettura, il suono del primo nastro veniva sovrapposto al secondo nastro sottoposto a questo trattamento creando questo particolare effetto.

Csound offre un'unità di effetto chiamata proprio flanger :

```

ares flanger      asig,adel,kfeedback
      - asig : segnale in ingresso
      - adel : ritardo in secondi
      - kfeedback : valore di feedback (range tra 0 e 1)
```

un esempio di **stereo flanger** in cui due linee di ritardo sono variate da un oscillatore a bassa frequenza, i parametri di controllo riguardano la densità dell'effetto, la velocità di Lfo e la quantità di feedback.

Esempio Cap.12.7_flanger

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1    vco2  .2,55,0      ;sawtooth
a2    vco2  .2,220,0
```

```

asum sum a1,a2

idepth      = p4      ;profondità
irate = p5      ;velocità LFO

adel1 poscil      idepth/2000,irate/2000,1
adel2 poscil      idepth/2000,irate/1000,1

kfeedback = p6      ;feedback

aflang1 flanger   asum,adel1,kfeedback
aflang2 flanger   asum,adel2,kfeedback

outs (aflang1+asum)*.5,(aflang2+asum)*.5;segnale originale+effetto
endin

```

</CsInstruments>
<CsScore>

f1	0	4096	10	1	
i1	0	5	20	1	.1
s					
i1	0	5	20	20	.4
s					
i1	0	5	40	40	.9
s					

</CsScore>
</CsoundSynthesizer>

Un'altra possibile variante con quattro istanze di flanger ,ognuna con valori random per il parametro delay.

Esempio Cap.12.8_flanger_quad

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

instr 1

a1,a2 diskin2      "drums_loop.aif",1,0,1

idepth      = p4
irate = p5
kfeed = p6

;valori random per delay
adel1 randh idepth/10,(irate*1.2)/10
adel2 randh idepth/10,irate/10
adel3 randh idepth/10,(irate*2)/10

```

```

adel4 randh idepth/10,irate/20

klfo oscili .01,.05,1 ;Lfo

aflang1 flanger a1,adel1+klfo,kfeed
aflang2 flanger a2,adel2+klfo,kfeed
aflang3 flanger a1,adel3+klfo,kfeed
aflang4 flanger a2,adel4+klfo,kfeed

outs (aflang1+aflang2+a1)*.5,(aflang3+aflang4+a2)*.5

endin

</CsInstruments>
<CsScore>

f1 0 4096 10 1

i1 0 4 .1 1 .1
s
i1 0 4 .2 2 .3
s
i1 0 4 .3 3 .4
s
i1 0 4 .2 4 .5
s

</CsScore>
</CsoundSynthesizer>
```

Tra gli effetti storici più importanti troviamo anche il **phaser**, si tratta di una tecnica in cui il segnale viene processato da alcuni particolari filtri chiamati **allpass** disposti in serie, si tratta di filtri in grado di modificare la fase. Costruendo una catena di filtri allpass modulati da uno o più Lfo (oscillatori a bassa frequenza), e comandando la catena di filtri al suono originale otteniamo il celebre effetto **phaser**. Anche per questo tipo di trattamento esiste un opcode dedicato **phaser2** (anche in versione semplificata chiamato **phaser1**) , il nostro esempio utilizza una versione semplificata in versione U.d.o.

```
a1,a2 Stereo_phaser asig,krate,kdepth,istages
```

Esempio Cap.12.9_phaser

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

#include "stereo_phaser.h"

instr 1

asig diskin2 "guitar.wav",1,0,1

krate = p4 ;velocità Lfo
kdepth = p5 ;modulazione
istages = 10 ;numero di filtri allpass in serie
```

```

a1,a2 Stereo_phaser      asig,krate,kdepth,istages
outs  a1+asig*.4,a2+asig*.4
endin

</CsInstruments>
<CsScore>

f1    0      4096  10     1
i1    0      4       .1    100
s
i1    0      4       .5    2000
s

</CsScore>
</CsoundSynthesizer>

```

Routing

In questa sezione verranno affrontate le tecniche di collegamento audio più moderne ed efficienti per definire una performance live o una composizione elettroacustica. Programmando una struttura con Csound assai complessa,ad esempio con l'utilizzo di processori multi-effetto,in cui il segnale necessita di attraversare il processo di trattamento in determinati momenti,oppure di entrare in un effetto chiamato "x" e di uscire dall'effetto "y",ecc,ecc,ci si scontra con un problema tipico dei linguaggi di sintesi,ossia il livello troppo elevato di astrazione del codice.Questo rende complessa,tra le varie cose,la pratica di integrazione futura sullo stesso sorgente (pur essendo ben commentato),alcune delle tecniche seguenti utilizzano come idea base,quella di definire con un nome specifico il tipo di trattamento o segnale,per esempio :

strumento chitarra

strumento kick

riverbero sempre acceso

output L/R di chitarra

input L/R multidelay

immaginiamo di poter raggiungere con Csound,la stessa flessibilità offerta dai sequencer multi traccia tradizionali,in cui controlliamo gli effetti sulle singole tracce,il routing nel caso di processi multi effetto,il send e return,i canali bus,ecc

Negli esempi fin qui visti sul trattamento del segnale,abbiamo incontrato (anche varie volte nei capitoli precedenti sulla sintesi) la tecnica in cui si dichiarano delle variabili globali,unite ad un processo di accumulo e azzeramento del segnale,si tratta degli opcode **vincr** e **clear**.

```

gasig1      init  0
gasig2      init  0

instr 1

a1,a2 ins  ;segnale in ingresso (microfono)

outs  a1,a2

```

```

vincr gasig1,a1
vincr gasig2,a2

endin

instr 2      ;reverb

aL,aR reverbsc    gasig1,gasig2,p4,p5

outs  aL,aR

clear gasig1,gasig2

endin

```

questa è una delle tecniche storiche di Csound,tuttavia tale scelta può rivelarsi non efficace lavorando su strumenti Csound molto complessi e che richiedono un alto numero di trattamenti del segnale.

I metodi di routing più usati possono essere sintetizzati in tre modalità :

- vincr,clear
- chnset, chnget
- connect,alwayson

I vantaggi offerti dai metodi *chnset,chnget* e *connect,alwayson* riguardano la possibilità di utilizzare stringhe di testo,in questo modo è possibile monitorare in modo meno astratto il percorso del segnale.

Esempio Cap.12.10_chnset

```

<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 1
0dbfs = 1

instr 1

a1      diskin2      "guitar.wav",1,0,1
chnset      a1,"guitar" ;uscita del segnale a1
endin

instr 2

ainput      chnget      "guitar"      ;segnale a1 entra in input
k1      randomi      100,2000,1
a1      butterbp      ainput,k1,1000
out a1

endin

```

```

</CsInstruments>
<CsScore>

i1      0      8
i2      0      8

</CsScore>
</CsoundSynthesizer>

```

Il prossimo esempio rappresenta probabilmente la scelta più moderna di routing del segnale in Csound, introduciamo gli opcode **connect**,**alwaysOn**,**outleta**,**inleta** :

connect : permette il collegamento tra un segnale iniziale e un canale bus di output utilizzando stringhe.

Esempio :

```
connect      "guitar","guitar_OUT","chorusFX","chorus-OUT"
```

alwaysOn : attiva un determinato strumento per tutta la durata definita nella score (p3).

Esempio :

```
alwayson    "reverbx"
```

outleta,inleta : canali bus di collegamento, funzionano unicamente tramite il collegamento **connect**.

Esempio :

```
connect      "noisex","noise_out","reverbx","rev_out"
```

```
alwayson    "reverbx"
```

```
instr noisex
```

```
aout  rand  .7
```

```
outlet      "noise_out",aout ;bus uscita per riverbero
```

```
endin
```

```
instr reverbx
```

```
a1      inleta      "rev_out" ; segnale ingresso da instr rumore
```

```
al,ar reverbsc  al,a1,.9,12000,sr,0
```

```
outs  al,ar
```

```
endin
```

Esempio Cap.12.11_connect_alwayson

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 2
0dbfs = 1

connect      "impulse","impulse_outL","reverbx","rev_outL" ;left
connect      "impulse","impulse_outR","reverbx","rev_outR" ;right
alwayson    "reverbx"

instr impulse

k1    randomi   .05,.2,5 ;impulsi random con risonatore
a1    mpulse     4,k1
kfreq randomi  100,1000,5
areso wguidel  a1,kfreq,kfreq*2,.9

kpan  randomi   0,1,5 ;random pan
aL,aR pan2  areso,kpan

outleta      "impulse_outL",aL ;bus uscita per riverbero
outleta      "impulse_outR",aR

outs  aL,aR ;uscita audio

endin

instr reverbx

a1    inleta     "rev_outL" ;segnaile ingresso da instr impulse
a2    inleta     "rev_outR"

al,ar reverbsc  a1,a2,.9,12000,sr,0

outs  al,ar

endin

</CsInstruments>
<CsScore>

i"impulse" 0    200

</CsScore>
</CsoundSynthesizer>
```

L'esempio finale racchiude molte delle tecniche esplorate negli ultimi capitoli, si tratta di uno strumento adatto ad una performance di live electronics in cui Csound processa in tempo reale un segnale esterno (per esempio un microfono con uno strumento acustico), per gestire il segnale esterno csound offre gli opcode **in** e **ins**.

```
ar1,ar2      ins
```

per attivare una porta audio di ingresso proveniente dal nostro convertitore audio, si utilizza il seguente flag :

-iadc

Un completa linea di comando potrebbe essere :

```
<CsOptions>
-iadc1      -odac1      -b2000      -B2000
</CsOptions>
```

L'utilizzo di un buffer size (-b,-B) può generare una certa latenza nel segnale in uscita, tuttavia diventa una prassi indispensabile lavorando con trattamenti del segnale assai complessi (ad esempio il phase vocoder in tempo reale con i pvs opcodes).

Il nostro esempio di performance live è sintetizzato nel seguente schema :

segnale in ingresso stereo (due microfoni)

trattamento del segnale : risonatore, riverbero, pitch shifter, spectral delay, spectral blur, spectral filter (notch)

caratteristiche : i processori sono collegati in parallelo, alcuni trattamenti del suono hanno l'output controllato da un autopan random, gli slider dell'interfaccia gestiscono unicamente la quantità di ogni singolo effetto, è anche possibile controllare gli slider tramite un controller midi esterno (attivare il flag -M con numero di porta midi relativo al proprio controller esterno).

Segnale in uscita : segnale originale + segnale processato, uscita stereo, ogni uscita viene mixata in un unico segnale per la scrittura di file stereo su disco.

Esempio Cap.12.12_live_electronics

```
<CsoundSynthesizer>
<CsOptions>
-odac1 -b2000
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

#define      NOMEFILE      # "recording.wav" #

seed 0

;connessioni
connect  "signal","signal_outL","reverbX","rev_inL"      ;left
connect  "signal","signal_outR","reverbX","rev_inR"      ;right
connect  "signal","signal_outL","pitch_shifter","harm_inL"
connect  "signal","signal_outR","pitch_shifter","harm_inR"
connect  "signal","signal_outL","pvsdelay","pvsdel_inL"
```

```

connect      "signal","signal_outR","pvsdelay","pvsdel_inR"
connect      "signal","signal_outL","blur","blur_inL"
connect      "signal","signal_outR","blur","blur_inR"
connect      "signal","signal_outL","pvsnotch","pvsnotch_inL"
connect      "signal","signal_outR","pvsnotch","pvsnotch_inR"
connect      "signal","signal_outL","resonator","reso_inL"
connect      "signal","signal_outR","resonator","reso_inR"
connect      "signal","vu_outL","meter","met_inL"      ;vu meter
connect      "signal","vu_outR","meter","met_inR"

;recording
connect      "signal","rec_outL","record","rec_inL"    ;recording
connect      "signal","rec_outR","record","rec_inR"
connect      "resonator","rec_outL","record","rec_inL"
connect      "resonator","rec_outR","record","rec_inR"
connect      "reverbx","rec_outL","record","rec_inL"
connect      "reverbx","rec_outR","record","rec_inR"
connect      "pitch_shifter","rec_outL","record","rec_inL"
connect      "pitch_shifter","rec_outR","record","rec_inR"
connect      "pvsdelay","rec_outL","record","rec_inL"
connect      "pvsdelay","rec_outR","record","rec_inR"
connect      "blur","rec_outL","record","rec_inL"
connect      "blur","rec_outR","record","rec_inR"
connect      "pvsnotch","rec_outL","record","rec_inL"
connect      "pvsnotch","rec_outR","record","rec_inR"

;strumenti sempre attivi
alwayson     "resonator"
alwayson     "reverbx"
alwayson     "pitch_shifter"
alwayson     "pvsdelay"
alwayson     "blur"
alwayson     "pvsnotch"
alwayson     "meter"
alwayson     "controller"
alwayson     "record"

;Gui FLTK
FLpanel      "Live electronics GUI",750,400,100,100,1,-1,-1
FLcolor      193,215,88

ival1 FLvalue  "",60,30,540,20
ival2 FLvalue  "",60,30,540,70
ival3 FLvalue  "",60,30,540,120
ival4 FLvalue  "",60,30,540,170
ival5 FLvalue  "",60,30,540,220
ival6 FLvalue  "",60,30,540,270
ival7 FLvalue  "",60,30,540,320

imin = 1
imax = 100
iexp = 0
iheight = 30
iwidth = 500
ix = 20
iy = 20

gk1,gih1   FLslider   "Input",0,1,iexp,21,ival1,iwidth,iheight,ix,iy
FLsetColor2 0,200,220,gih1
FLsetColor 193,215,88,gih1

gk2,gih2   FLslider   "resonator",0,1,iexp,25,ival2,iwidth,iheight,ix,iy+50
gk3,gih3   FLslider   "pitchshift",0,1,iexp,25,ival3,iwidth,iheight,ix,iy+100

```

```

gk4,gih4    FLslider   "pvs-filter",0,1,iexp,25,ival4,iwidth,iheight,ix,iy+150
gk5,gih5    FLslider   "pvs-delay",0,1,iexp,25,ival5,iwidth,iheight,ix,iy+200
gk6,gih6    FLslider   "blur",0,1,iexp,25,ival6,iwidth,iheight,ix,iy+250
gk7,gih7    FLslider   "reverb",0,1,iexp,25,ival7,iwidth,iheight,ix,iy+300

gk8,gih8    FLslider   "Meter",0,1,iexp,2,-1,40,250,ix+640,iy
FLsetColor2 22,20,220,gih8
FLsetColor 193,215,88,gih8

gkexit,gihexit FLbutton  "Exit",1,0,1,100,30,630,320,0,100,0,0

FLpanelEnd
FLrun

```

```

#include    "Small_Handbell.h"           ;resonator opcode

instr signal

a1,a2 ins  ;segnale esterno
;a1,a2      diskin2    "sampleaudio",1,0,1

outleta    "signal_outL",a1 ;bus uscita L/R
outleta    "signal_outR",a2

outs  a1*gk1,a2*gk1      ;segnale originale

outleta    "vu_outL",a1*gk1 ;bus uscita vu-meter
outleta    "vu_outR",a2*gk1

outleta    "rec_outL",a1*gk1 ;bus uscita recording
outleta    "rec_outR",a2*gk1

endin

instr pitch_shifter

a1      inleta     "harm_inL"  ;segnale ingresso da instr signal
a2      inleta     "harm_inR"

fftin pvsanal    a1+a2,1024,256,1024,1  ;fft-analysis of file

;harmonizer

kpitch1   randomh   .1,2,2      ;random trasposizioni
kpitch2   randomh   .1,2,3
kpitch3   randomh   .1,2,4
kpitch4   randomh   .1,2,5

fsig1 pvscale   fftin,kpitch1
fsig2 pvscale   fftin,kpitch2
fsig3 pvscale   fftin,kpitch3
fsig4 pvscale   fftin,kpitch4

aout1 pvsynth   fsig1 ; risintesi
aout2 pvsynth   fsig2
aout3 pvsynth   fsig3
aout4 pvsynth   fsig4

;delay1

```

```

abuf1 delayr      1      ;buffer memoria
adel1      deltapi     2      ;punto da cui prelevare il segnale
afeed1     =      .1 * adell + aout1      ;numero di ripetizioni
delayw     afeed1      ;scrive il segnale in ritardo

;delay2
abuf2 delayr      1      ;buffer memoria
adel2      deltapi     3      ;punto da cui prelevare il segnale
afeed2     =      .1 * adel2 + aout2      ;numero di ripetizioni
delayw     afeed2      ;scrive il segnale in ritardo

;delay3
abuf3 delayr      1      ;buffer memoria
adel3      deltapi     4      ;punto da cui prelevare il segnale
afeed3     =      .1 * adel3 + aout3      ;numero di ripetizioni
delayw     afeed3      ;scrive il segnale in ritardo

;delay4
abuf4 delayr      1      ;buffer memoria
adel4      deltapi     5      ;punto da cui prelevare il segnale
afeed4     =      .1 * adel4 + aout4      ;numero di ripetizioni
delayw     afeed4      ;scrive il segnale in ritardo

outs  (adel1+adel3)*gk3, (adel2+adel4)*gk3

outleta    "rec_outL", (adel1+adel3)*gk3 ;bus uscita recording
outleta    "rec_outR", (adel2+adel4)*gk3

endin

instr pvsdelay

a1      inleta      "pvsdel_inL"      ;segnaile ingresso da instr signal
a2      inleta      "pvsdel_inR"

fftin1    pvsanal     a1,4092,256,4092,0; fft-analisi, left e right separati
fftin2    pvsanal     a2,4092,256,4092,0

ibuf1,kt1  pvsbuffer   fftin1,4; 4-sec buffer
ibuf2,kt2  pvsbuffer   fftin2,3; 3-sec buffer

;ksig randomh      kmin,kmax,rate
kdel1 randomh      .05,1,1      ;valori random per ritardi
kdel2 randomh      .1,1.2,2
kdel3 randomh      .5,1.4,3
kdel4 randomh      .1,2,4
kdel5 randomh      .1,2.2,5
kdel6 randomh      .1,2.4,6
kdel7 randomh      .1,2.5,7
kdel8 randomh      .1,2.6,8

kiminrnd     =      200      ;valore base range
;valori random per range di frequenza massimo per pvsbufread
;valore massimo = (sample rate) / 2

khi1    randomi   kiminrnd,(sr)/2,1
khi2    randomi   kiminrnd,(sr)/2,2
khi3    randomi   kiminrnd,(sr)/2,3
khi4    randomi   kiminrnd,(sr)/2,4
khi5    randomi   kiminrnd,(sr)/2,5
khi6    randomi   kiminrnd,(sr)/2,6
khi7    randomi   kiminrnd,(sr)/2,7

```

```

khi8      randomi   kiminrnd, (sr)/2,8

fread1    pvsbufread  kt1-kdel1,ibuf1,500,khi1 ;8 punti di lettura diversi
fread2    pvsbufread  kt2-kdel2,ibuf2,500,khi2
fread3    pvsbufread  kt1-kdel3,ibuf1,500,khi3
fread4    pvsbufread  kt2-kdel4,ibuf2,500,khi4
fread5    pvsbufread  kt1-kdel5,ibuf1,500,khi5
fread6    pvsbufread  kt2-kdel6,ibuf2,500,khi6
fread7    pvsbufread  kt1-kdel7,ibuf1,500,khi7
fread8    pvsbufread  kt2-kdel8,ibuf2,500,khi8

fsall1   pvsmix     fread1,fread2      ;mixa due componenti spettrali
fsall2   pvsmix     fread3,fread4
fsall3   pvsmix     fread5,fread6
fsall4   pvsmix     fread7,fread8

aout1 pvsynth     fsall1      ;risintesi
aout2 pvsynth     fsall2
aout3 pvsynth     fsall3
aout4 pvsynth     fsall4

aL      sum       aout1,aout2
aR      sum       aout3,aout4

outs   aL*gk5,aR*gk5      ;spectral delay

outleta  "rec_outL",aL*gk5 ;bus uscita recording
outleta  "rec_outR",aR*gk5

endin

instr blur

a1      inleta      "blur_inL"  ;segnale ingresso da instr signal
a2      inleta      "blur_inR"

fftin1   pvsanal    a1+a2,1024,256,4092,0; fft-analisi

kdepth   randomi    .1,1,10

fftblur   pvsblur    fftin1,kdepth,1

aout   pvsynth     fftblur

;delay
abuf  delayr     1      ;buffer memoria
adel  deltapi    4      ;punto da cui prelevare il segnale
afeed = .2 * adel + aout ;numero di ripetizioni
delayw  afeed ;scrive il segnale in ritardo

kpan  randomi    0,1,2 ;random pan
aout1,aout2 pan2  adel,kpan

outs   aout1*gk6,aout2*gk6

outleta  "rec_outL",aout1*gk6    ;bus uscita recording
outleta  "rec_outR",aout2*gk6

endin

instr pvsnotch

```

```

a1    inleta      "pvsnotch_inL"      ; segnale ingresso da instr signal
a2    inleta      "pvsnotch_inR"

fsig  pvsanal     a1+a2,1024,256,4092,0; fft-analisi

klowcut   randomi   100,2000,2
klowfull  randomi   100,12000,4
khightfull randomi   100,2000,6
khightcut  randomi   100,12000,8

fbandp     pvsbandr   fsig,klowcut,klowfull,khightfull,khightcut ;spectral band
pass

aout  pvsynth     fbandp
;delay
abuf  delayr      1      ;buffer memoria
adel  deltapi     .1      ;punto da cui prelevare il segnale
afeed =   .1 * adel + aout ;numero di ripetizioni
delayw  afeed ;scrive il segnale in ritardo

kpan  randomi   0,1,2 ;random pan
aout1,aout2 pan2  adel,kpan

outs  aout1*gk4,aout2*gk4

outleta   "rec_outL",aout1*gk4      ;bus uscita recording
outleta   "rec_outR",aout2*gk4

endin

instr resonator

a1    inleta      "reso_inL"  ; segnale ingresso da instr signal
a2    inleta      "reso_inR"

kfreq1   randomh   500,1200,2 ;valori random per freq e Q
kfreq2   randomh   500,1200,4
kQ1     randomh   1,1000,6
kQ2     randomh   1,1000,8

aresol   Small_Handbell   a1,kfreq1,kQ1      ;risonatori
areso2   Small_Handbell   a2,kfreq2,kQ2

;delay1
abuf1 delayr      1      ;buffer memoria
adel1    deltapi     .5      ;punto da cui prelevare il segnale
afeed1 =   .2 * adel1 + aresol      ;numero di ripetizioni
delayw  afeed1 ;scrive il segnale in ritardo
;delay2
abuf2 delayr      1      ;buffer memoria
adel2    deltapi     .7      ;punto da cui prelevare il segnale
afeed2 =   .2 * adel2 + areso2      ;numero di ripetizioni
delayw  afeed2 ;scrive il segnale in ritardo

kpan1 randomi   0,1,1 ;random pan
kpan2 randomi   0,1,2
aout1,aout2 pan2  adel1,kpan1
aout3,aout4 pan2  adel2,kpan2

outs  (aout1+aout3)*gk2,(aout2+aout4)*gk2

```

```

outleta      "rec_outL", (aout1+aout3)*gk2 ;bus uscita recording
outleta      "rec_outR", (aout2+aout4)*gk2

endin

instr meter ;vu meter

a1    inleta      "met_inL"   ;segnale ingresso da instr signal
a2    inleta      "met_inR"

ktrig metro 15
kval max_k a1+a2,ktrig,0
FLsetVal ktrig,kval,gih8

endin

instr reverbx

a1    inleta      "rev_inL"   ;segnale ingresso da instr signal
a2    inleta      "rev_inR"

ksize randomi .6,.95,10 ;random size

aL,aR reverbsc a1,a2,ksize,12000,sr,0

kpan1 randomi 0,1,5 ;random pan
kpan2 randomi 0,1,2 ;random pan
aout1,aout2 pan2 aL,kpan1
aout3,aout4 pan2 aR,kpan2

outs (aout1+aout2)*gk7, (aout3+aout4)*gk7

outleta      "rec_outL",aout1*gk7 ;bus uscita recording
outleta      "rec_outR",aout2*gk7

endin

instr record      ;mix recording

a1    inleta      "rec_inL"
a2    inleta      "rec_inR"

fout $NOMEFILE,4,a1,a2

endin

instr controller ;midi slider controller

;setta il valore midi iniziale per ogni controller
ctrlinit 1,1,0,1,2,0,1,3,0,1,4,0,1,5,0,1,6,0,1,7,0

;input
k1    ctrl7 1,1,0,1 ;controller midi 1,canale midi 1,range tra 0 e 1
ktrig changed k1
FLsetVal ktrig,k1,gih1 ;ktrig invia i dati di k1 allo slider gih1
;Fx
k2    ctrl7 1,2,0,1
ktrig2     changed k2

```

```

FLsetVal      ktrig2,k2,gih2
;Fx
k3    ctrl7 1,3,0,1
ktrig3      changed      k3
FLsetVal      ktrig3,k3,gih3
;Fx
k4    ctrl7 1,4,0,1
ktrig4      changed      k4
FLsetVal      ktrig4,k4,gih4
;Fx
k5    ctrl7 1,5,0,1
ktrig5      changed      k5
FLsetVal      ktrig5,k5,gih5
;Fx
k6    ctrl7 1,6,0,1
ktrig6      changed      k6
FLsetVal      ktrig6,k6,gih6
;Fx
k7    ctrl7 1,7,0,1
ktrig7      changed      k7
FLsetVal      ktrig7,k7,gih7

```

endin

```

instr 100 ;chiusura Gui
exitnow
endin

```

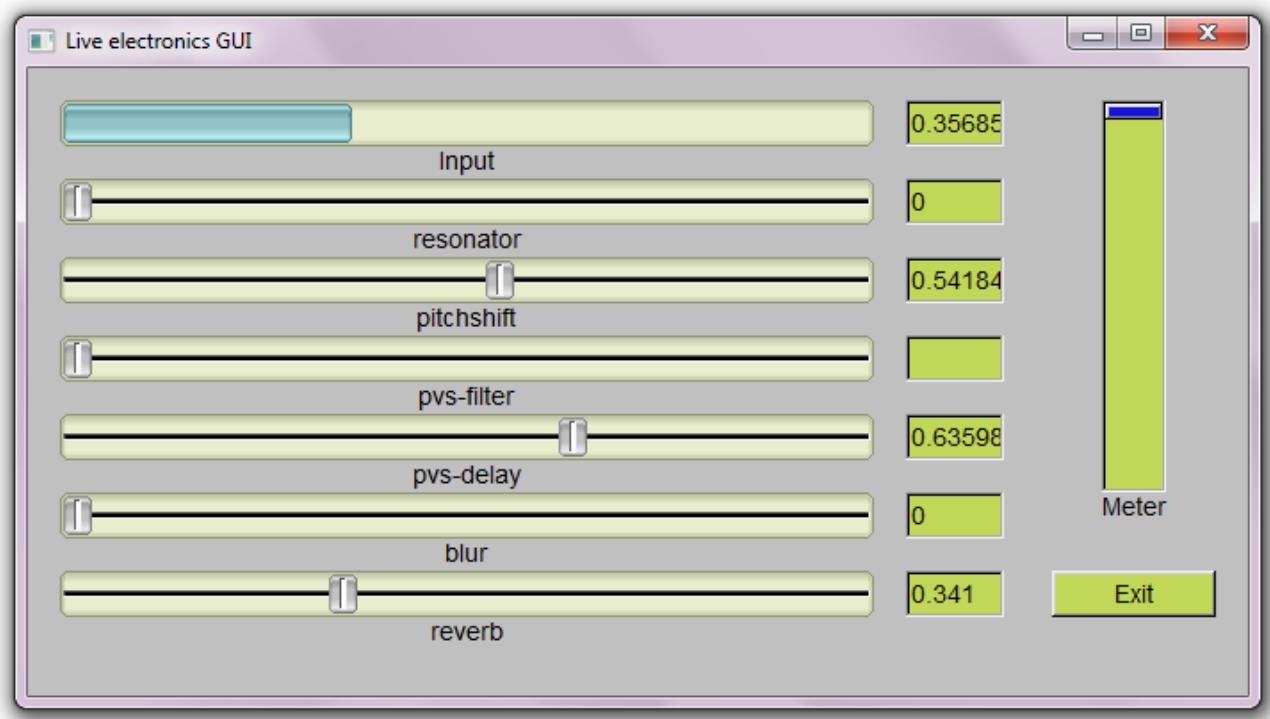
</CsInstruments>

<CsScore>

i"signal" 0 3600

</CsScore>

</CsoundSynthesizer>



Prendiamo in esame una variante dell'esempio precedente, questa volta la gui verrà costruita con quetcsound, la principale modifica riguarderà l'introduzione della spazializzazione quadrifonica in tempo reale. Per prima cosa occorrerà modificare l'header della nostra orchestra nel seguente modo :

```
sr = 44100
ksmps = 32
nchnls = 4 ;quattro canali in uscita
0dbfs = 1
```

Csound contiene numerosi opcode per la gestione e controllo multicanale, in questo esempio utilizzeremo la variante quadrifonica di **pan2** :

```
a1,a2,a3,a4      pan      asig,kx,ky,ifn
```

kx,ky : rappresentano le coordinate del suono nello spazio :

left-front at (0,1), right-front at (1,1), left-rear at the origin (0,0), and right-rear at (1,0)

ifn : indica una tabella con una sequenza memorizzata che descrive l'aumento di ampiezza di un altoparlante durante il movimento del suono, rispetto ad un altoparlante adiacente.

Infine sarà necessario modificare ogni singolo strumento per la versione quadrifonica, l'opcode **outs** verrà sostituito da **outq**.

Esempio Cap.12.13_live_electronics_quad

```
<CsoundSynthesizer>
<CsOptions>
;-odac1 -b2048 -B2048
</CsOptions>
<CsInstruments>

sr = 44100
ksmps = 32
nchnls = 4 ;quattro canali in uscita
0dbfs = 1

connect      "signal","signal_outL","reverbX","rev_inL"      ;left
connect      "signal","signal_outR","reverbX","rev_inR"      ;right
connect      "signal","signal_outL","pitch_shifter","harm_inL"  ;left
connect      "signal","signal_outR","pitch_shifter","harm_inR"  ;right
connect      "signal","signal_outL","pvsdelay","pvsdel_inL"   ;left
connect      "signal","signal_outR","pvsdelay","pvsdel_inR"   ;right
connect      "signal","signal_outL","blur","blur_inL"        ;left
connect      "signal","signal_outR","blur","blur_inR"        ;right
connect      "signal","signal_outL","pvsnotch","pvsnotch_inL"  ;left
connect      "signal","signal_outR","pvsnotch","pvsnotch_inR"  ;right
connect      "signal","signal_outL","resonator","reso_inL"    ;left
connect      "signal","signal_outR","resonator","reso_inR"    ;right

alwaysOn      "resonator"
alwaysOn      "reverbX"
alwaysOn      "pitch_shifter"
alwaysOn      "pvsdelay"
alwaysOn      "blur"
alwaysOn      "pvsnotch"
```

```

#include    "Small_Handbell.h"      ;resonator opcode

instr signal

a1,a2 ins  ;segnale esterno
;a1,a2    diskin2    "fileaudio",1,0,1

outleta   "signal_outL",a1*.5    ;bus uscita L/R
outleta   "signal_outR",a2*.5

k1      invalue    "slider0"    ;slider qutecsound gui

ksl    portk k1,.01    ;portamento per evitare rumori durante la transazione
outq  (a1*.5)*ksl,(a2*.5)*ksl,(a1*.5)*ksl,(a2*.5)*ksl;segnale originale
endin

instr pitch_shifter

a1      inleta     "harm_inL"  ;segnale ingresso da instr signal
a2      inleta     "harm_inR"

fftin pvsanal    a1+a2,1024,256,1024,1  ;fft-analysis of file

;harmonizer

kpitch1   randomh   .1,2,2      ;random trasposizioni
kpitch2   randomh   .1,2,3
kpitch3   randomh   .1,2,4
kpitch4   randomh   .1,2,5

fsig1 pvscale    fftin,kpitch1
fsig2 pvscale    fftin,kpitch2
fsig3 pvscale    fftin,kpitch3
fsig4 pvscale    fftin,kpitch4

aout1 pvsynth    fsig1 ; risintesi
aout2 pvsynth    fsig2
aout3 pvsynth    fsig3
aout4 pvsynth    fsig4

;delay1
abuf1 delayr    1      ;buffer memoria
adel1    deltapi    1      ;punto da cui prelevare il segnale
afeed1   =   .1 * adel1 + aout1    ;numero di ripetizioni
delayw   afeed1    ;scrive il segnale in ritardo

;delay2
abuf2 delayr    1      ;buffer memoria
adel2    deltapi    2      ;punto da cui prelevare il segnale
afeed2   =   .1 * adel2 + aout2    ;numero di ripetizioni
delayw   afeed2    ;scrive il segnale in ritardo

;delay3
abuf3 delayr    1      ;buffer memoria
adel3    deltapi    3      ;punto da cui prelevare il segnale
afeed3   =   .1 * adel3 + aout3    ;numero di ripetizioni
delayw   afeed3    ;scrive il segnale in ritardo

```

```

;delay4
abuf4 delayr      1      ;buffer memoria
adel4      deltapi     4      ;punto da cui prelevare il segnale
afeed4      =      .1 * adel4 + aout4      ;numero di ripetizioni
delayw      afeed4      ;scrive il segnale in ritardo

kx      randomi     0,1,.5      ;random quad pan
ky      randomi     0,1,1

aq1,aq2,aq3,aq4      pan      (adel1+adel3+adel2+adel4)*.5,kx,ky,4,1,1

k1      invalue      "slider1"

ksl      portk k1,.01

outq  aq1*ksl,aq2*ksl,aq3*ksl,aq4*ksl

endin

instr pvsdelay

a1      inleta      "pvsdel_inL"      ;segnale ingresso da instr signal
a2      inleta      "pvsdel_inR"

fftin1      pvsanal     a1,4092,256,4092,0; fft-analisi, left e right separati
fftin2      pvsanal     a2,4092,256,4092,0

ibuf1,kt1      pvsbuffer    fftin1,4; 4-sec buffer
ibuf2,kt2      pvsbuffer    fftin2,3; 3-sec buffer

;ksig randomh      kmin,kmax,rate
kdel1 randomh      .05,1,1      ;valori random per ritardi
kdel2 randomh      .1,1.2,2
kdel3 randomh      .5,1.4,3
kdel4 randomh      .1,2,4
kdel5 randomh      .1,2.2,5
kdel6 randomh      .1,2.4,6
kdel7 randomh      .1,2.5,7
kdel8 randomh      .1,2.6,8

kiminrnd      =      2000 ;valore base range
;valori random per range di frequenza massimo per pvsbufread
;valore massimo = (sample rate) / 2

khi1 randomi      kiminrnd,(sr)/2,1
khi2 randomi      kiminrnd,(sr)/2,2
khi3 randomi      kiminrnd,(sr)/2,3
khi4 randomi      kiminrnd,(sr)/2,4
khi5 randomi      kiminrnd,(sr)/2,5
khi6 randomi      kiminrnd,(sr)/2,6
khi7 randomi      kiminrnd,(sr)/2,7
khi8 randomi      kiminrnd,(sr)/2,8

fread1      pvsbufread   kt1-kdel1,ibuf1,500,khi1      ;8 punti di lettura diversi
fread2      pvsbufread   kt2-kdel2,ibuf2,500,khi2
fread3      pvsbufread   kt1-kdel3,ibuf1,500,khi3
fread4      pvsbufread   kt2-kdel4,ibuf2,500,khi4
fread5      pvsbufread   kt1-kdel5,ibuf1,500,khi5
fread6      pvsbufread   kt2-kdel6,ibuf2,500,khi6
fread7      pvsbufread   kt1-kdel7,ibuf1,500,khi7
fread8      pvsbufread   kt2-kdel8,ibuf2,500,khi8

```

```

fsall1      pvsmix      fread1,fread2      ;mixa due componenti spettrali
fsall2      pvsmix      fread3,fread4
fsall3      pvsmix      fread5,fread6
fsall4      pvsmix      fread7,fread8

aout1 pvsynth      fsall1      ;risintesi
aout2 pvsynth      fsall2
aout3 pvsynth      fsall3
aout4 pvsynth      fsall4

kx      randomi      0,1,.5      ;random quad pan
ky      randomi      0,1,1

aq1,aq2,aq3,aq4  pan      (aout1+aout2+aout3+aout4)*.5,kx,ky,4,1,1

k1      invalue      "slider2"

ksl    portk k1,.01

outq  aq1*ksl,aq2*ksl,aq3*ksl,aq4*ksl

endin

instr blur

a1      inleta      "blur_inL"  ;segnale ingresso da instr signal
a2      inleta      "blur_inR"

fftin1      pvsanal     a1+a2,1024,256,4092,0; fft-analisi

kdepth      randomi      .1,1,10

fftblur      pvsblur     fftin1,kdepth,1

aout  pvsynth      fftblur

;delay
abuf  delayr      1      ;buffer memoria
adel  deltapi     2      ;punto da cui prelevare il segnale
afeed =   .2 * adel + aout ;numero di ripetizioni
delayw  afeed ;scrive il segnale in ritardo

kx      randomi      0,1,.5      ;random quad pan
ky      randomi      0,1,1

aq1,aq2,aq3,aq4  pan      adel*.5,kx,ky,4,1,1

k1      invalue      "slider3"

ksl    portk k1,.01

outq  aq1*ksl,aq2*ksl,aq3*ksl,aq4*ksl

endin

instr pvsnotch

a1      inleta      "pvsnotch_inL"      ;segnale ingresso da instr signal
a2      inleta      "pvsnotch_inR"

```

```

fsig pvsanal      a1+a2,1024,256,4092,0; fft-analisi

klowcut    randomi    100,2000,2
klowfull   randomi    100,12000,4
khightfull randomi    100,2000,6
khightcut  randomi    100,12000,8

fbandp     pvsbandr   fsig,klowcut,klowfull,khightfull,khightcut ;spectral band
pass

aout pvsynth      fbandp
;delay
abuf delayr      1      ;buffer memoria
adel deltapi     3      ;punto da cui prelevare il segnale
afeed = .1 * adel + aout ;numero di ripetizioni
delayw     afeed ;scrive il segnale in ritardo

kx    randomi    0,1,.5      ;random quad pan
ky    randomi    0,1,1

aq1,aq2,aq3,aq4  pan     adel*.5,kx,ky,4,1,1

k1    invalue     "slider4"

ksl   portk k1,.01

outq aq1*ksl,aq2*ksl,aq3*ksl,aq4*ksl

endin

instr resonator

a1    inleta      "reso_inL"  ;segnale ingresso da instr signal
a2    inleta      "reso_inR"

kfreq1    randomh    500,1200,2  ;valori random per freq e Q
kfreq2    randomh    500,1200,4
kQ1     randomh    1,1000,6
kQ2     randomh    1,1000,8

aresol    Small_Handbell  a1,kfreq1,kQ1
areso2    Small_Handbell  a2,kfreq2,kQ2

;delay1
abuf1 delayr      1      ;buffer memoria
adel1     deltapi     1      ;punto da cui prelevare il segnale
afeed1   = .2 * adel1 + aresol    ;numero di ripetizioni
delayw     afeed1     ;scrive il segnale in ritardo
;delay2
abuf2 delayr      1      ;buffer memoria
adel2     deltapi     1      ;punto da cui prelevare il segnale
afeed2   = .2 * adel2 + areso2    ;numero di ripetizioni
delayw     afeed2     ;scrive il segnale in ritardo

kx    randomi    0,1,.5      ;random quad pan
ky    randomi    0,1,1

aq1,aq2,aq3,aq4  pan     (adel1+adel2)*.5,kx,ky,4,1,1

k1    invalue     "slider5"

ksl   portk k1,.01

```

```

outq  aq1*ksl,aq2*ksl,aq3*ksl,aq4*ksl
endin

instr reverbx

a1    inleta      "rev_inL"    ; segnale ingresso da instr signal
a2    inleta      "rev_inR"

aL,aR reverbsc  a1,a2,.9,12000,sr,0

kx    randomi    0,1,.5        ;random quad pan
ky    randomi    0,1,1

aq1,aq2,aq3,aq4  pan     (aL+aR)*.5,kx,ky,4,1,1

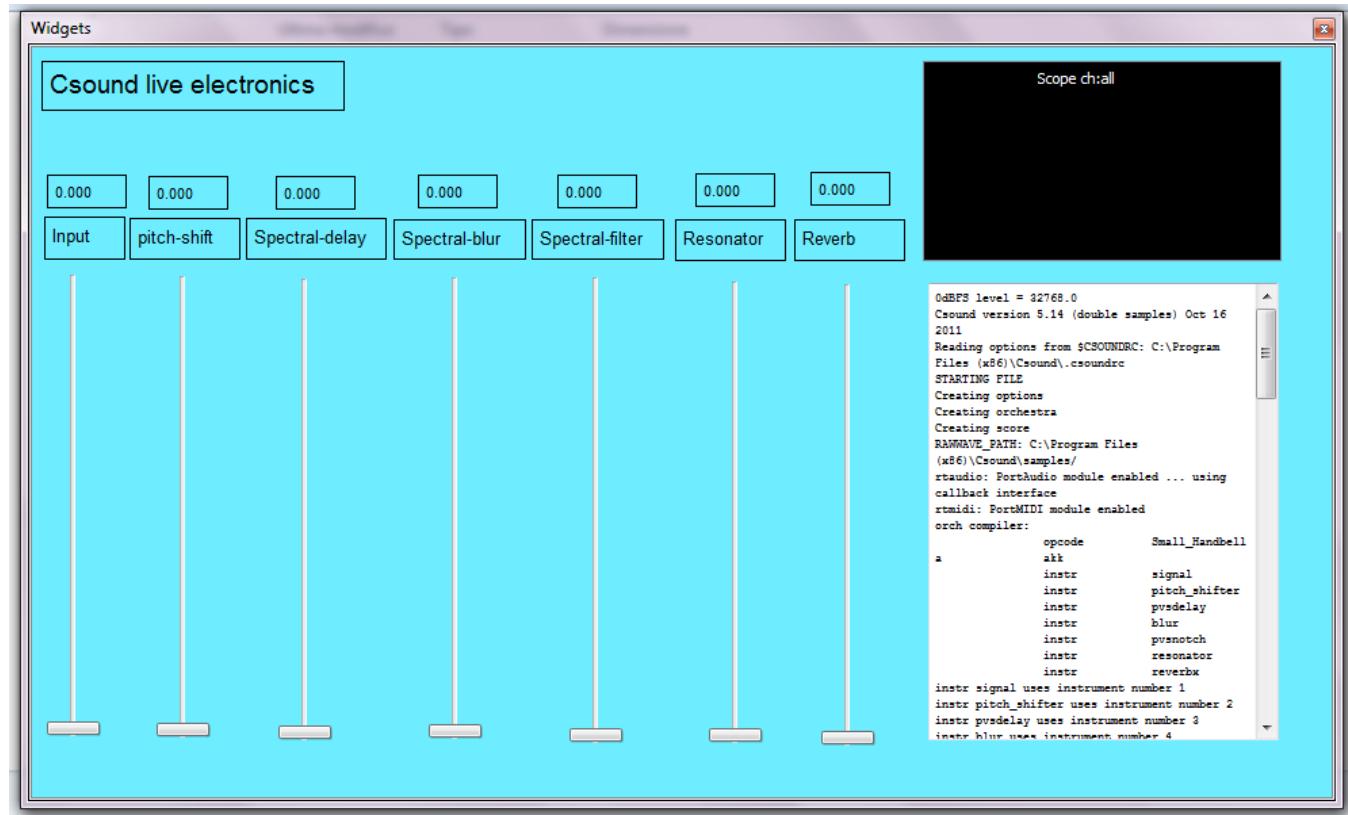
k1    invalue     "slider6"

ksl   portk k1,.01

outq  aq1*ksl,aq2*ksl,aq3*ksl,aq4*ksl
endin

</CsInstruments>
<CsScore>
f4    0      8193  9      .25    1      0      ;pan quad
i"signal"  0      3600
</CsScore>
</CsoundSynthesizer>

```



Cap.13 Approfondimenti

14.1 Pwgl

14.2 Generazione di score per Csound

14.3 U.D.O opcodes

14.4 Librerie di sintesi (Chowning,Risset)

Composizione assistita

Una delle aree più affascinanti e relativamente recenti, legate al mondo della musica elettroacustica (ma anche cameristica, sinfonica, operistica), è quella della *composizione assistita dal calcolatore*.

Si tratta di ambienti di programmazione visuale (simili a sistemi come *Max/Msp*, *Pure data*) in cui il compositore realizza algoritmi in grado di formalizzare, analizzare, elaborare del materiale musicale di tipo melodico, ritmico, armonico.

Il vantaggio di questi sistemi è quello di lavorare ad un alto livello di rappresentazione sonora, per mezzo di pratici editor di pentagramma. Il risultato finale di una *patch* elaborata con questi linguaggi, può essere un frammento di partitura a più voci, una melodia basata su processi aleatori, una successione di accordi generata da una melodia data, ecc. ecc.

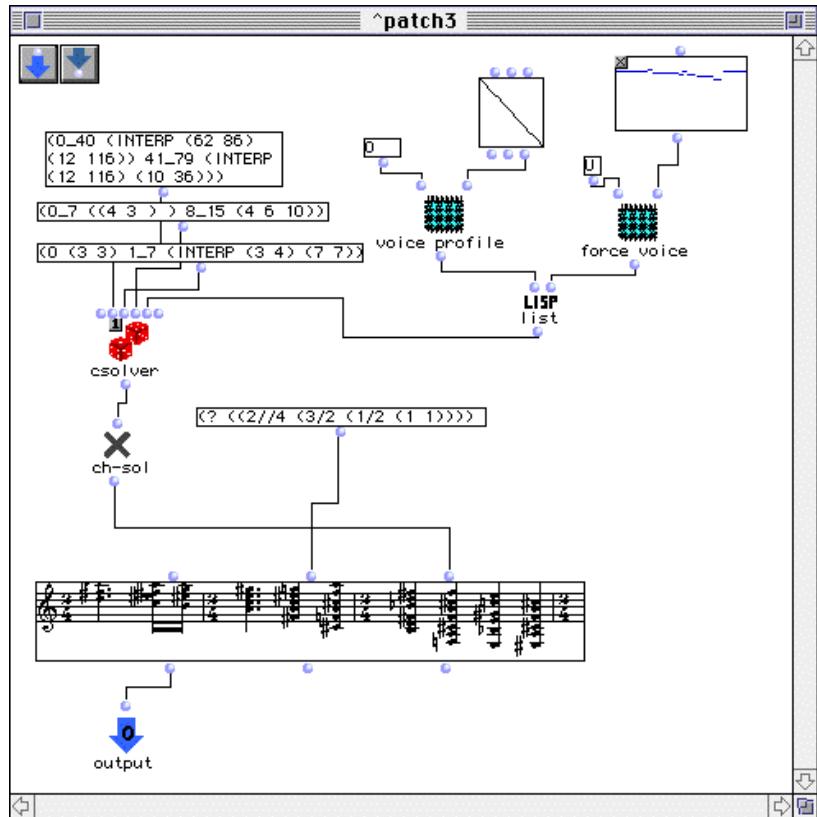
Gli ambienti di composizione assistita più diffusi sono principalmente due, *Open Music* e *Pwgl*.

OpenMusic è un discendente di *Patchwork* (*Laurson, Rueda, Duthen, Assayag, Agon*), un ambiente per la composizione assistita da computer sviluppato per la prima volta all' *IRCAM* nei primi anni '90. *OpenMusic* è attualmente disponibile per le versioni di Mac OS X (PowerPC e Intel), Windows. Il codice sorgente è stato rilasciato sotto licenza *GNU GPL*.

Si tratta di un ambiente di programmazione visuale basato su Common Lisp, offre un alto livello di rappresentazione del materiale musicale tramite editor di partitura, midi piano roll, rappresentazione di segnali audio, per questo ambiente di programmazione esistono alcune librerie specifiche adatte a lavorare con Csound :

- *omChroma* : sviluppata da *Carlos Agon* e *Marco Stroppa*, è basata sul concetto di sintetizzatore modulare, con questa libreria è possibile interfacciarsi con Csound e Diphone (un motore di sintesi sviluppato all'Ircam, oggi non più aggiornato).
- *Om2Csound* : sviluppata da *Karim Haddad*, permette la creazione di score per Csound.

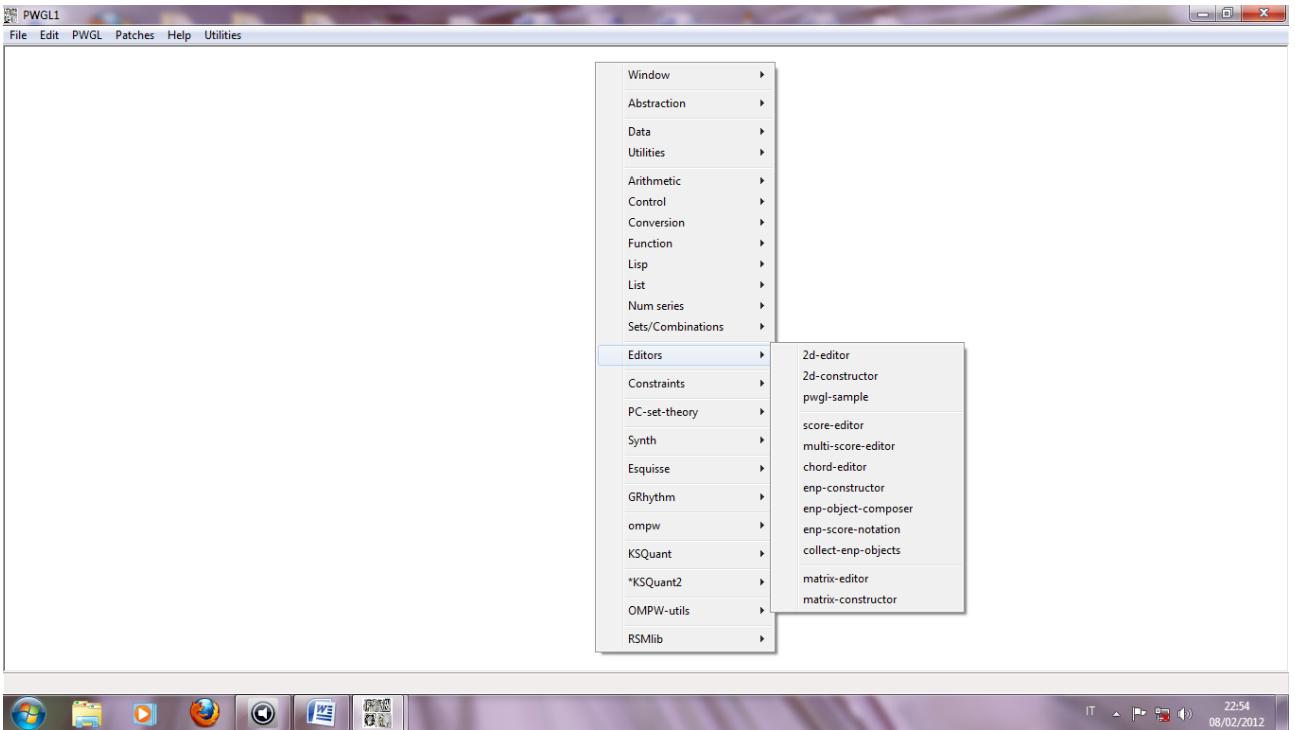
Un tipico esempio di patch creata con *OpenMusic* :



PWGL è un linguaggio visivo basato su concetti simili a Patchwork (*Laurson, Rueda, Duthen, Assayag, Agon*) , *PWGL* è stato progettato da zero e contiene numerosi miglioramenti rispetto a quest'ultimo. Per esempio, la parte grafica del sistema è stata realizzata in *OpenGL*. *OpenGL* offre diversi vantaggi, quali supporti multi-piattaforma, accelerazione hardware, Floating-point e grafica sofisticata 2D e 3D-graphics. Una finestra in *PWGL* contiene oggetti che sono basati su funzioni scritte in linguaggio Lisp. Contiene al suo interno numerosi oggetti adatti alla manipolazione di dati musicali, a differenza di OpenMusic è provvisto di un motore di sintesi interno.

Per prima cosa procuriamoci la versione di *Pwgl* per il nostro sistema operativo al seguente link :
<http://www2.siba.fi/pwgl/downloads.html>.

Apriamo il programma e andiamo su Pwgl-Audio/MIDI setup per settare la nostra scheda audio. Ora un breve sguardo al menu oggetti :

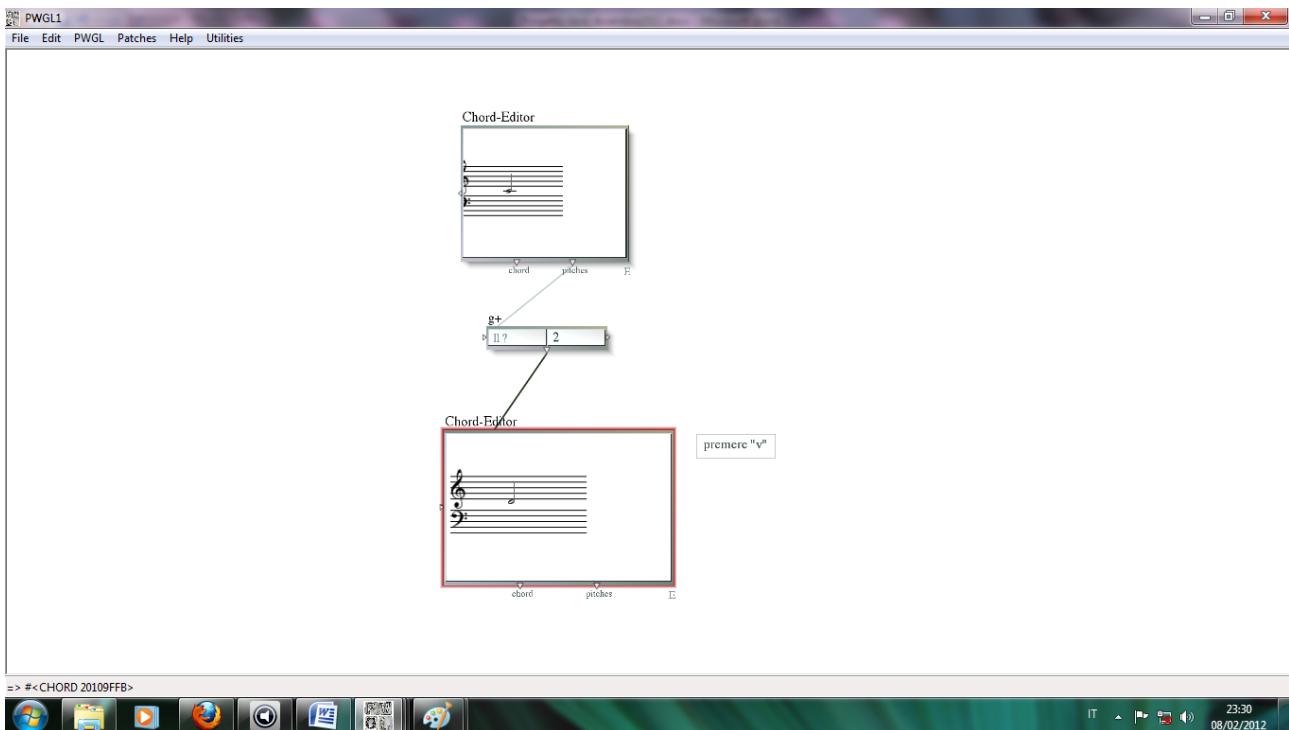


Per iniziare a muovere i primi passi in quest’ambiente, si consiglia di esplorare l’ottimo Help che trovate alla voce *Help-Pwgl help...*, si tratta di un vero e proprio tutorial strutturato in una raccolta di patches dimostrative delle funzioni e oggetti principali di *Pwgl*.

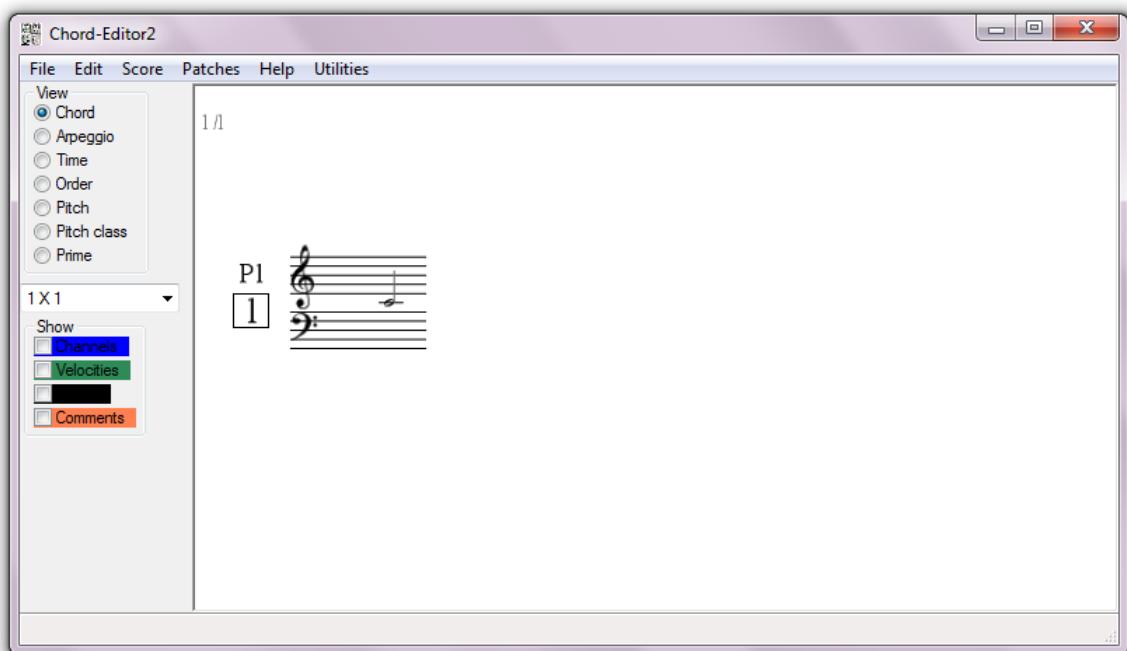


Gli esempi che seguono non vogliono essere dei veri e propri tutorial di questo linguaggio (funzione che esula dagli scopi di questo libro), tuttavia servono come introduzione alla materia trattata in questo capitolo, ossia l’utilizzo di *Pwgl* come score-generator per Csound (per approfondimenti su questo ambiente di composizione assistita si rimanda alla documentazione ufficiale, in particolare il *Pwgl book*, che potete scaricare dal sito ufficiale).

L’esempio che segue introduce uno dei principali oggetti grafici di *Pwgl*, si tratta di *Chord-Editor*. In questo semplicissimo esempio di trasposizione, l’oggetto “g +” prende come input la nota “do” di *Chord-Editor*, effettua una operazione matematica di addizione e invia il risultato al secondo *Chord-Editor* che visualizza la nota “re” (per visualizzare il risultato finale, è necessario cliccare sul secondo *Chord-Editor* e premere in seguito il tasto “v” della vostra tastiera).



L'oggetto *Chord-Editor* ha numerose funzioni di scrittura musicale (chord, arpeggio, time, order, pitch, pitch class), provate a cliccare sopra l'oggetto, troverete la seguente finestra :



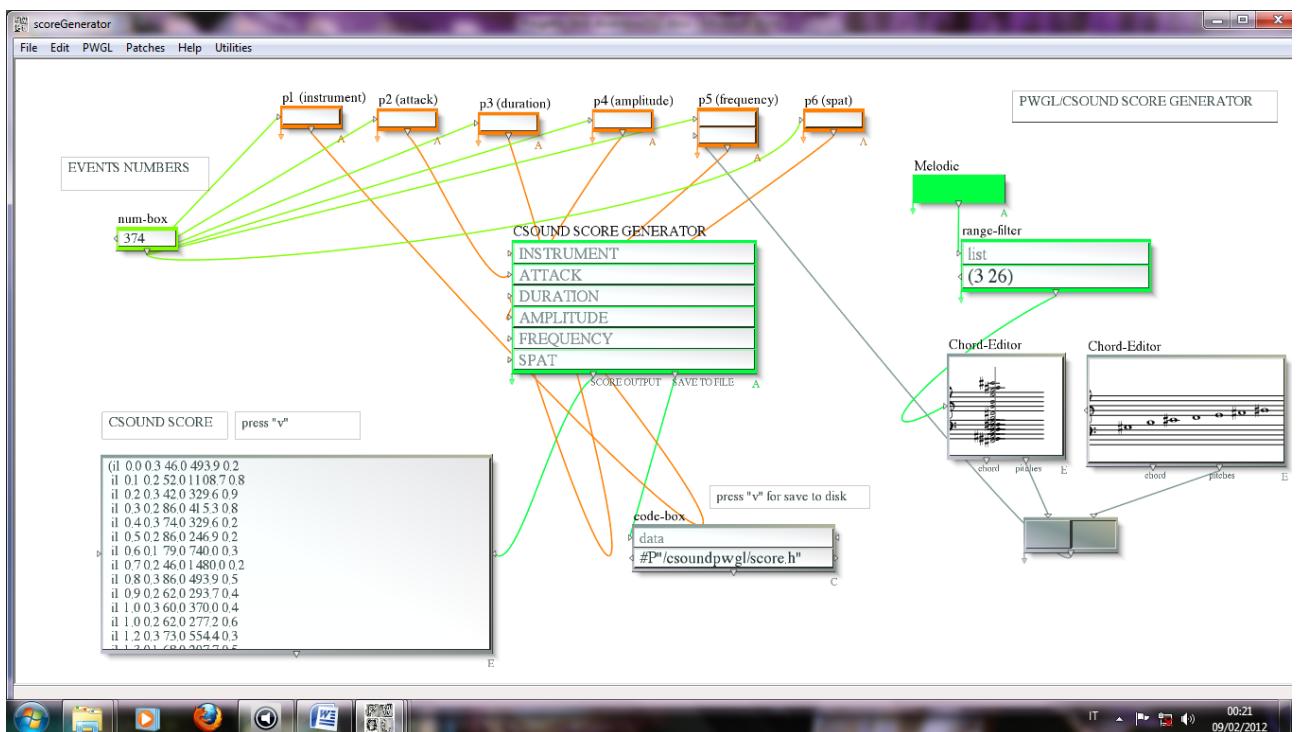
Per aggiungere note è sufficiente fare un clic sopra la nota "do", premendo in seguito il tasto "A" della vostra tastiera. Le funzioni *Chord* e *Arpeggio* permettono di visualizzare un gruppo di note sotto forma di accordo o di linea melodica.

Dopo questa brevissima panoramica introduttiva, ci occuperemo di una patch scritta per Pwgl in grado di generare delle score per Csound attraverso metodi di generazione algoritmica. Si tratta di un esempio molto semplice ma adatto a sviluppi, integrazioni, personalizzazioni future. Immaginiamo di definire delle regole generali per la composizione di un brano, tenendo conto sia di elementi musicali che timbrici, lo schema seguente riassume un esempio di *Pwgl-score generator* :

- attacco : range numerico da cui una funzione random genererà i valori di "p2" (attacco)

- durata : range numerico da cui una funzione random genererà i valori di “p3” (durata)
- ampiezza : range numerico da cui una funzione random genererà i valori di “p4” (ampiezza con conversione ampdb)
- frequenza : il materiale scritto, manipolato o generato da *Chord-Editor* verrà convertito in valori di frequenza (conversione midi-frequenza), le frequenze ottenute saranno controllate da una funzione random che genererà delle strutture melodiche.
- pan : range numerico (compreso tra 0 e 1) da cui una funzione random genererà i valori di “p6” (posizione del suono nello spazio left/right), in pratica ogni nota avrà una posizione di pan diversa.

Apriamo il file scoreGenerator.pwgl :



I parametri di controllo della nostra score sono rappresentati dagli oggetti color “arancione”, si tratta di alcune semplici *sub patch* in cui è possibile indicare il range di valori di ogni parametro. La generazione del materiale melodico è controllata dall’oggetto color “verde” *Melodic*, in cui è possibile selezionare (filtrare) l’output attraverso la funzione *range-filter*, quest’ultimo oggetto invia i dati al primo *Chord-Editor* che troviamo sulla destra.

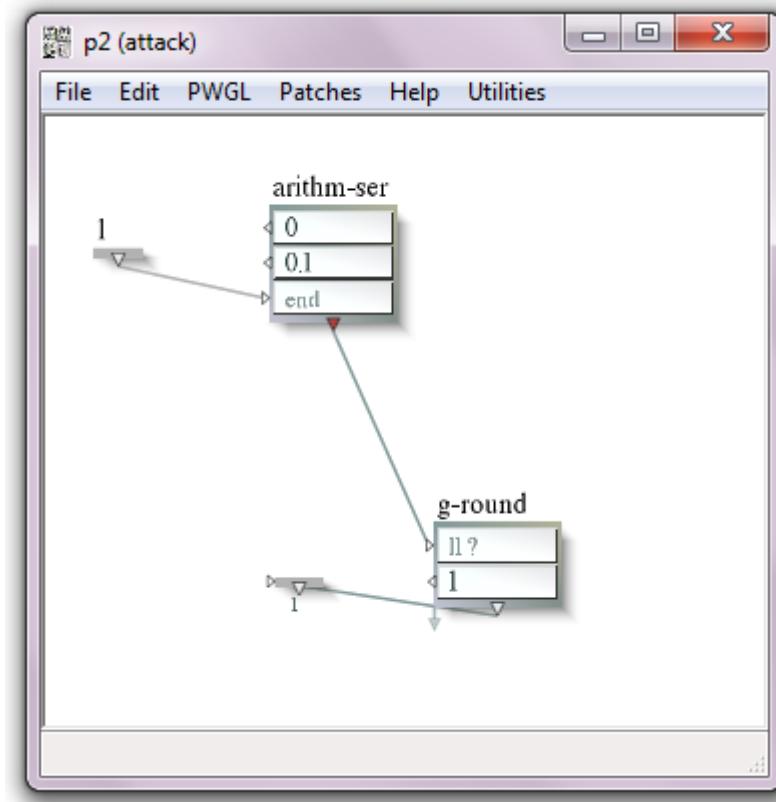
Il risultato finale viene visualizzato nella text-box situata sulla sinistra (Csound score), è possibile salvare il file finale in due modi :

- apprendo la text box e utilizzando la funzione “save”
- utilizzando la *code-box* a cui è associata una funzione lisp in grado di scrivere un file su disco, in questo caso il file “score.h” verrà scritto all’interno della cartella “csoundpwgl”. Se la cartella non è stata precedentemente creata è possibile visualizzare un messaggio di errore, si consiglia pertanto di operare nel seguente modo :

windows : creare la cartella in C:\csoundpwgl

Osx : creare la cartella in Macintosh HD \ csoundpwgl

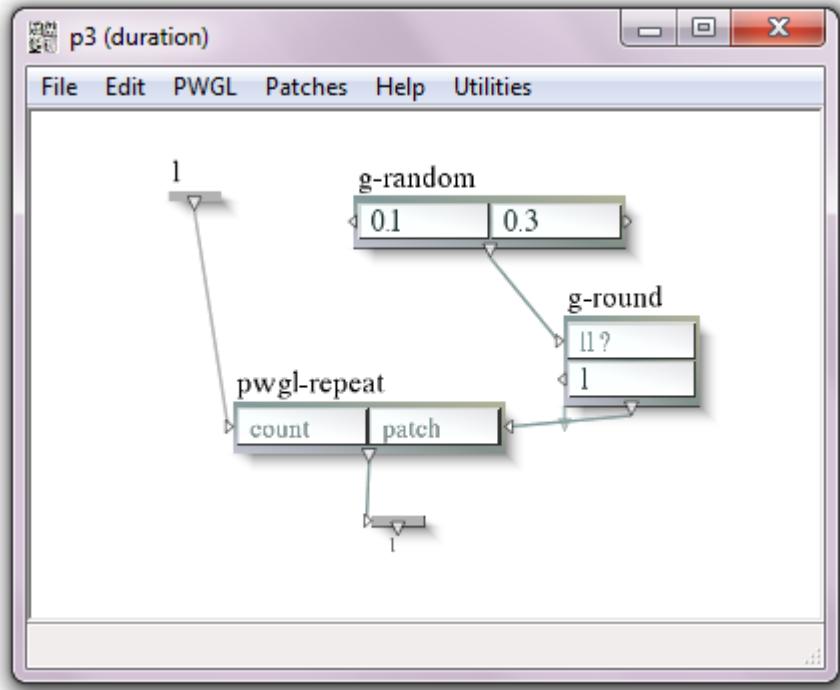
Apriamo adesso l'oggetto *p2(attack)* :



Questa *sub patch* genera una serie di valori per l'attacco (p2) in ordine crescente (con incremento di 0.1) :

```
0  
0.1  
0.2  
0.3  
0.4  
0.5  
0.6  
0.7  
0.8  
0.9  
1  
...
```

La sub patch “p3(duration)” genera invece una serie di valori per p3 utilizzando come base un range prefissato :



un possibile output potrebbe essere :

```

0.2
0.1
0.2
0.2
0.3
0.1
0.3
0.2
..
..
..
..
```

User-defined opcode

La tecnica degli *UDO opcode* permette di scrivere i propri moduli di Csound personalizzati, si tratta di opcode scritti direttamente con il linguaggio Csound (da non confondere con gli standard opcode in linguaggio C con cui è costruito Csound). L'incredibile vantaggio di questa tecnica è quella di racchiudere, incapsulare, riutilizzare in modo semplice e immediato, interi frammenti di codice. Ad esempio potremmo voler trasformare in opcode, uno strumento legato ad una particolare tecnica di sintesi, una porzione di codice in grado di svolgere un complesso calcolo matematico, un processore di segnale, ecc, ecc.

Immaginiamo una semplice orchestra come la seguente, abbiamo una forma d'onda ispirata alla celebre *supersaw* dei sintetizzatori Roland, in pratica un banco di 7 saw ognuno con indipendente e variabile *pitch detune*.

Esempio Cap.13.1_Hipersaw

```

<CsSoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
;kr = 4410
ksmps = 16
nchnls = 1
0dbfs = 1

instr 1

iamp = p4

kdetune = p6

krnd1 poscil      kdetune,.1,1
krnd2 poscil      kdetune,.2,1
krnd3 poscil      kdetune,.3,1
krnd4 poscil      kdetune,.4,1
krnd5 poscil      kdetune,.5,1

a1    vco2  iamp,p5,0
a2    vco2  iamp,p5+kdetune,0
a3    vco2  iamp,p5+(kdetune+krnd1)*.1,0
a4    vco2  iamp,p5+(kdetune+krnd2)*.1,0
a5    vco2  iamp,p5+(kdetune+krnd3)*.1,0
a6    vco2  iamp,p5+(kdetune+krnd4)*.1,0
a7    vco2  iamp,p5+(kdetune+krnd5)*.1,0

asum  sum   a1,a2,a3,a4,a5,a6,a7

kcutf = p7
kresof = p8

afilt moogladder  asum,kcutf,kresof

aout  clip  afilt,0,.9

kenv  adsr  0.1,.3,1,0

out   aout*kenv

endin

</CsInstruments>
<CsScore>

f1     0      16384 10      1

```

```
i1      0      5      .6      120      5      12000 .82
```

```
</CsScore>
</CsoundSynthesizer>
```

Quanto sarebbe più comodo avere questo strumento con un'unica riga di codice?

```
asuono      Hipersaw    iamp,ifreq,kdetune,kcut,kreso
```

Per scrivere il nostro opcode abbiamo bisogno di incapsulare questo strumento all'interno di una particolare struttura, per certi versi simile alla scrittura di un qualsiasi strumento di un'orchestra. Introduciamo la schema generale di un *UDO opcode*, costituito da tre nuovi moduli chiamati *opcode,xin e xout*.

```
opcode  name,  outtypes, intypes ; nome, uscita, tipi di variabili (a,k,i,ecc..)
xinarg1 [, xinarg2] [, xinarg3] ... [xinargN]  xin
... inseriamo il codice dello strumento...
xout  xoutarg1 [, xoutarg2] [, xoutarg3] ... [xoutargN]
endop
```

la nostra Hipersaw diventa :

```
;UDO Hipersaw
opcode      Hipersaw,a,kkkkk ;5 variabili di controllo
kamp,kcps,kdetune,kcut,kreso  xin
... inseriamo il codice dello strumento...

xout  aout      (un solo argomento = mono)
endop
```

Vediamo adesso un'applicazione completa :

Esempio Cap.13.2_Hipersaw-udo

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
;kr = 4410
ksmps = 16
nchnls = 1
0dbfs = 1

;UDO Hipersaw
```

```

opcode      Hipersaw,a,kkkk
gisine      ftgen 0,0,4096,10,1
kamp,kcps,kdetune,kcut,kreso  xin
krnd1 poscil      kdetune,.1,gisine
krnd2 poscil      kdetune,.2,gisine
krnd3 poscil      kdetune,.3,gisine
krnd4 poscil      kdetune,.4,gisine
krnd5 poscil      kdetune,.5,gisine

a1    vco2  kamp,kcps,0
a2    vco2  kamp,kcps+kdetune,0
a3    vco2  kamp,kcps+(kdetune+krnd1)*.1,0
a4    vco2  kamp,kcps+(kdetune+krnd2)*.1,0
a5    vco2  kamp,kcps+(kdetune+krnd3)*.1,0
a6    vco2  kamp,kcps+(kdetune+krnd4)*.1,0
a7    vco2  kamp,kcps+(kdetune+krnd5)*.1,0

asum  sum   a1,a2,a3,a4,a5,a6,a7
afilt moogladder  asum,kcut,kreso
aout  clip  afilt,0,.9
xout  aout          ; write output
endop
;end UDO

;applicazione Hipersaw
instr 1
iamp =      p4
ifreq =     p5
kdetune =   p6
kcutf =    p7
kreso =    p8
a1    Hipersaw    iamp,ifreq,kdetune,kcutf,kreso
kenv  adsr  0.1,.3,1,0
out   a1*kenv

endin

</CsInstruments>
<CsScore>

i1    0      5      .6      120     5      12000 .82

</CsScore>
```

```
</CsoundSynthesizer>
```

Esempio Cap.13.3_chorus

Vediamo adesso una semplice implementazione di un chorus :

```
<CsoundSynthesizer>
<CsOptions>
</CsOptions>
<CsInstruments>

sr = 44100
kr = 4410
ksmps = 10
nchnls = 2
0dbfs = 1

opcode      chorus, a,akk

asig, kdepth, krate xin

k1ch  randi kdepth/2,krate,1
ar1    vdelay      asig,kdepth/2+k1ch,10
k2ch  randi kdepth/2,krate*1.1,.2
ar2    vdelay      asig,kdepth/2+k2ch,10
k3ch  randi kdepth/2,krate*0.9,.2
ar3    vdelay      asig,kdepth/2+k3ch,10
k4ch  randi kdepth/2,krate*0.8,.1
ar4    vdelay      asig,kdepth/2+k4ch,10

aout =      (ar1+ar2+ar3+ar4)/4

xout  aout

endop

instr 1

a1      pluck .9,110,440,0,1

kdepth      =      3
krate =      5

afx  chorus      a1,kdepth,krate

outs  afx,afx

endin

</CsInstruments>
<CsScore>

i1    0      4

</CsScore>
</CsoundSynthesizer>
```

All'interno della cartella Cap.13 troverete due librerie di UDO, si tratta degli strumenti storici di *Jean Claude Risset* e *John Chowning* che ho convertito in opcode (e che ho utilizzato nel mio vecchio progetto *Vectrosynth*, uno strumento per la sintesi vettoriale che potete trovare su www.csound.com) La libreria di *Risset* pone maggiore attenzione sull'esplorazione della sintesi additiva (la classica "campana" di *Risset*, i glissandi di armonici, ecc), mentre la libreria di *Chowning* è naturalmente rivolta alla sintesi Fm. Nell'ordine troviamo :

J.C.Risset library :

RissBell, RissClar, RissDrum, RissDrum2, RissetGliss, RissFlute, RissHarmon, RissNoise, RissOctave, RissRing.

J.Chowning library :

CHOWNClar, CHOWNglass, CHOWNLoop, CHOWNPad, CHOWNPerc, CHOWNpiano, CHOWNrebell, CHOWNSoprano, CHOWNString, CHOWNTrumpet, CHOWNVibr, CHOWNwown.

Un esempio dalla libreria di *Risset* :

```
<CsoundSynthesizer>
<CsOptions>
-odac -M0 --rtmidi=virtual -b1024 -B4000
</CsOptions>
<CsInstruments>

sr      =      44100
kr      =      4410
ksmps   =      10
nchnls  =      2

opcode     RissBell, a, ik
gisine    ftgen 0,0,32768,-9,-2,1,1,1,1,1,1
iamp,kfqc  xin
idur     iamp
ifunc    gisine
ilevel   = iamp/11
kael    expon ilevel,idur,ifunc
kae2    expon (ilevel*.67),(idur*.9),ifunc
kae3    expon ilevel,(idur*.65),ifunc
kae4    expon (ilevel*1.8),(idur*.55),ifunc
kae5    expon (ilevel*2.67),(idur*.325),ifunc
kae6    expon (ilevel*1.67),(idur*.35),ifunc
kae7    expon (ilevel*1.46),(idur*.25),ifunc
kae8    expon (ilevel*1.33),(idur*.2),ifunc
kae9    expon (ilevel*1.33),(idur*.15),ifunc
kae10   expon ilevel,(idur*.1),ifunc
kae11   expon (ilevel*1.33),(idur*.075),ifunc

a1    poscil    kael,(kfqc*.56),gisine
a2    poscil    kae2,(kfqc*.56+1),gisine
a3    poscil    kae3,(kfqc*.92),gisine
a4    poscil    kae4,(kfqc*.92+1.7),gisine
a5    poscil    kae5,(kfqc*1.19),gisine
```

```
a6      poscil      kae6, (kfqc*1.7),gisine
a7      poscil      kae7, (kfqc*2),gisine
a8      poscil      kae8, (kfqc*2.74),gisine
a9      poscil      kae9, (kfqc*3),gisine
a10     poscil      kae10, (kfqc*3.76),gisine
a11     poscil      kae11, (kfqc*4.07),gisine

aout = (a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+a11)*.5
kdeclick linsegr 1,0.01,1,0.33*1,0

xout aout*kdeclick

endop

instr 1

ilevel      ampmidi      15000

ifreq cpsmidi

a1      RissBell    ilevel,ifreq

outs  a1,a1

endin

</CsInstruments>
<CsScore>

f0      3600

</CsScore>
</CsoundSynthesizer>
```