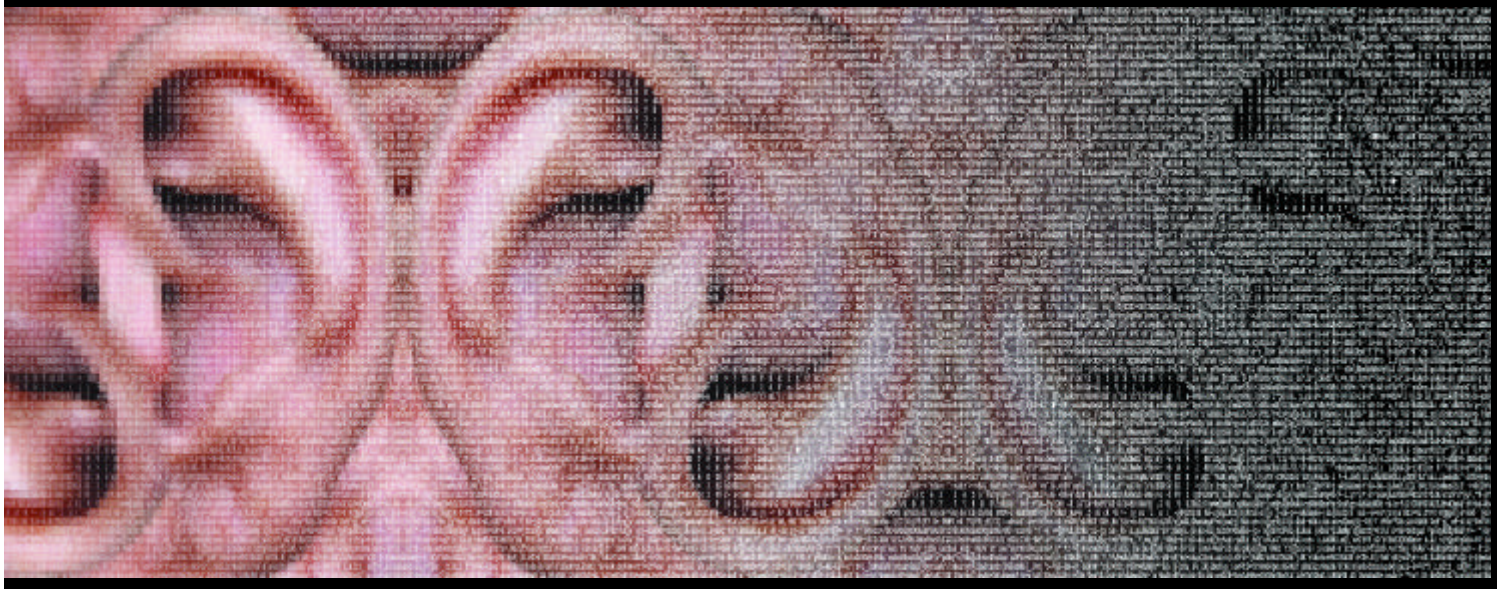
Riccardo Bianchini · Alessandro Cipriani

Il Suono Virtuale

Sintesi ed Elaborazione del Suono - Teoria e Pratica con Csound



ConTempo

www.virtual-sound.com

Questa è una copia demo ridotta del libro “Il Suono Virtuale”.

Per ordinare la versione completa, con CD Rom allegato:

www.virtual-sound.com

www.contemponet.com

o inviate una e-mail a:

info@virtual-sound.com

oppure un fax al nr.

06.35502025

Riccardo Bianchini • Alessandro Cipriani

Il Suono Virtuale

Sintesi ed Elaborazione del Suono
Teoria e Pratica con Csound

Collana *Cinema per l'Orecchio*
Direttore Editoriale: Alessandro Cipriani

BIANCHINI R. - CIPRIANI A.
Il Suono Virtuale
ISBN 88-900261-2-X

Seconda Edizione riveduta ed ampliata

Capitoli 1-9: A.Cipriani
Capitoli 10-17 e appendice: R.Bianchini

Copertina e CD Design: Alba D'Urbano e Nicolas Reichelt

Stampato in Italia
Contempo s.a.s., Roma
e-mail info@virtual-sound.com
info@contemponet.com
URL: www.virtual-sound.com
www.contemponet.com
fax +39-06.355.020.25

INDICE

PREFAZIONE

La sintesi digitale del suono è arrivata da tempo ad un altissimo livello di flessibilità e raffinatezza. Questa positiva situazione si deve in buona misura agli sforzi notevoli di Barry Vercoe, l'autore di Csound e dei suoi predecessori, MUSIC360 e MUSIC11. Ora, grazie all'universalità del linguaggio C che garantisce la facile trasferibilità tra *computer* dotati di diversi sistemi operativi, Csound è diventato lo standard dovunque vi sia *computer music*.

L'adattabilità di questo linguaggio apparirà evidente in questo libro: sia il musicista che lavora con un PC sia quello che possiede un Macintosh può studiarlo. Inoltre, un compositore che impara il Csound con un suo *computer* a casa è preparato poi ad accettare un invito a lavorare altrove, per esempio in uno studio ben attrezzato con sistemi multi-tracce di registrazione e con costosissime unità periferiche per il trattamento del suono, che saranno molto probabilmente organizzati attorno a macchine UNIX che girano la loro versione di Csound. E allora l'importanza di questo libro: per imparare Csound, soprattutto se uno deve studiare da solo, ci vuole qualcuno che spieghi i come e i perché di questo programma che offre così tante possibilità per la sintesi digitale del suono.

Lo studente si trova qui nelle mani capaci di due eccellenti insegnanti, Riccardo Bianchini e Alessandro Cipriani. Questo loro libro dimostra le notevoli esperienze didattiche da essi accumulate nell'aiutare musicisti con poca o nessuna esperienza del mondo dell'informatica a superare i primi ostacoli e le confusioni iniziali e nel condurli verso una piena comprensione e maestria della *computer music.* Tutte le tecniche di base di Csound sono spiegate in questo volume attraverso dei precisi modelli di sintesi - sintesi additiva, sintesi mediante modulazione, uso di filtri e di effetti basati sulla linea di ritardo (riverbero, eco, effetto di coro), controllo dinamico dell'evoluzione del suono, interfaccia con il mondo esterno (suoni campionati, collegamento MIDI) e altro ancora: sono le basi che permettono allo studente di costruire per conto suo ulteriori esperimenti e varianti ancora più sofisticate usando gli esempi offerti dagli autori come punti di partenza. Insomma, un'approccio pienamente "*hands-on*". Ogni capitolo offre ampi suggerimenti per gli usi più avanzati; così il libro non è valido soltanto per il principiante, ma accompagnerà il musicista attraverso le varie fasi di apprendimento del materiale, dall'inizio fino allo "*state of the art*". Bianchini e Cipriani hanno chiaramente assorbito e sperimentato una vastissima gamma di tecniche proposte da molti specialisti nel campo, e forniscono delle chiare spiegazioni e suggerimenti per i loro usi ed adattamenti, sempre in termini di efficaci esempi di programmazione Csound. Troviamo qui, per esempio, un'accurata descrizione del famoso algoritmo Karplus- Strong per la sintesi della corda pizzicata, con chiare indicazioni sul come eventualmente modificarlo. Queste modificazioni suggeriscono poi nuove possibilità di sperimentazione con altri cambiamenti ancora, tutti potenzialmente affascinanti.

Seguendo le lezioni di Bianchini e Cipriani, lo studente si troverà facilitato nella comprensione di ulteriori novità nel settore, che sono pubblicate regolarmente da riviste specializzate come “Computer Music Journal”, “Journal of New Music Research”, ecc. Un’idea originale e di gran valore nell’organizzazione del libro è quella di completare molti capitoli con alcuni pagine di “Approfondimenti”. Il musicista che è arrivato ad un buon livello di conoscenza del materiale troverà qui le informazioni tecniche che gli permetteranno di sviluppare idee originali di sintesi. Un “Approfondimento” particolarmente pertinente alla *computer music* tratta del concetto di “eventi complessi” - la possibilità di costruire uno strumento Csound per la sintesi di un insieme di più oggetti sonori nel quale la forma, la sincronizzazione e il carattere di ogni singolo oggetto sono sotto il controllo di pochi dati di partitura. Una volta costruito lo strumento per la sintesi di eventi di questo genere, il compositore può concepire la sua musica in termini di gesti complessi di larga scala anziché comporre sempre al livello di “nota per nota”. Infine, il libro conclude con una scelta di letture scritte da altri utenti Csound che hanno sviluppato applicazioni molto particolari, ma allo stesso tempo utili sia per essere utilizzate come tali, sia per dare spunto al compositore per la creazione di nuove, personali procedure.

Quindi, tanto a chi è un principiante quanto a chi ha già un bel po’ di esperienza con la sintesi digitale del suono, Bianchini e Cipriani propongono un manuale completo per l’apprendimento a vari livelli di Csound, un linguaggio in costante evoluzione grazie ai contributi di un piccolo gruppo internazionale di musicisti/programmatori che si dedicano all’aggiornamento del programma con nuovi metodi e con specifici adattamenti delle più recenti scoperte nel campo della *computer music*.

Benvenuti nell’universo Csound!

James Dashow

PREFAZIONE ALLA SECONDA EDIZIONE

Il costante aggiornamento di un libro come Il Suono Virtuale è indispensabile per mantenerlo al passo, non solo con l'evoluzione di Csound, ma anche con i nuovi sviluppi dell'audio digitale e della musica elettroacustica in generale.

Questa seconda edizione, pertanto, contiene oltre che revisioni, e correzioni, anche numerosi aggiornamenti e paragrafi interamente nuovi. Essa è stata in gran parte riscritta sulla base dell'edizione inglese (uscita nel 1999), sulle nuove esperienze degli autori, e sui preziosi suggerimenti di insegnanti, allievi e studiosi di varie nazioni che l'hanno utilizzato.

Abbiamo cercato di rendere il testo e l'esposizione ancora più chiari, compatibilmente con l'oggettiva difficoltà dell'argomento, aggiungendo nuove orchestre, partiture e figure.

Per questa seconda edizione desideriamo ringraziare, in particolare, Nyssim Lefford, che ha rivisto il testo inglese, ed Emanuele Casale, Enzo Coco, Agostino Di Scipio, Javier Leichman, Dennis Miller, Jon Christopher Nelson, Russell F. Pinkston, Antonio Saccoccio, Francesco Passarelli, Barry Truax e Alvise Vidolin per i preziosi consigli; e inoltre tutti i nostri allievi delle Scuole di Musica Elettronica di Roma e Catania.

Riccardo Bianchini e Alessandro Cipriani
Roma, settembre 2001

CHE COS'È CSOUND

Csound è un *software* per la sintesi digitale diretta del suono realizzato da Barry Vercoe allo M.I.T. (*Massachusetts Institute of Technology*). All´aggiornamento e allo sviluppo di questo *software* continuano a lavorare decine di persone in tutto il mondo: infatti si tratta di un *software* di pubblico dominio, e chiunque è libero non solo di utilizzarlo, ma anche di modificarlo e di ampliarlo. Con i più recenti processori (per PowerMac e per PC) Csound consente di svolgere la maggior parte delle operazioni in poco tempo o in tempo reale. CSound è scritto in linguaggio C, ma... attenzione! Non c'è bisogno che impariate il linguaggio C per usare Csound. É sufficiente che seguiate passo dopo passo questo manuale, che vi insegnerà a scrivere un'orchestra e una partitura, e vi troverete a creare suoni di ogni tipo con il vostro *computer*. L'importante è non spaventarsi all'inizio di fronte ai termini nuovi, poi tutto il processo di apprendimento diventerà più naturale e veloce.

LA SINTESI DIGITALE DIRETTA

Ma cosa è la sintesi digitale diretta? Se nella musica elettronica analogica, a un dato momento, servivano nove oscillatori e un filtro passabasso, occorreva acquistare o costruire nove oscillatori e un filtro passabasso. Se in una fase successiva del lavoro servivano nove filtri passabasso e un oscillatore, ebbene occorreva acquistare o costruire gli otto filtri mancanti. Nella sintesi digitale diretta, invece, è possibile programmare l'*hardware* (in questo caso il vostro *personal computer*) in modo tale che simuli nove oscillatori e un filtro, e in una fase successiva riprogrammarlo in modo tale che simuli un oscillatore e nove filtri. È evidente l'economicità e la flessibilità del processo: è infatti possibile programmare qualunque unità elementare, in modo tale da implementare qualsiasi tipo di sintesi del suono, passata, presente e futura.

CSOUND : IL MIGLIORE SINTETIZZATORE DEL MONDO

Csound è diverso da altri *software* di tipo commerciale perché, oltre ad essere gratuito, non invecchia: la sua validità è rimasta stabile perché qualunque nuovo tipo di sintesi o elaborazione del suono può essere implementato al suo interno, ed inoltre consente ogni volta al compositore di creare una macchina virtuale adatta ai suoi scopi, senza obbligarlo a quelle limitazioni cui i *software* commerciali ci hanno abituato, cioè ad avere, per esempio, 30 opzioni: efficienti e veloci, ma solo quelle 30! Imparare Csound (proprio perché è così aperto) significa, oltre a poter spaziare di più con i propri desideri sonori e musicali, anche avere le idee chiare su come funzionano i vari metodi di sintesi e di elaborazione del suono e quindi, una volta acquisite queste solide basi teoriche e pratiche, saper sfruttare al meglio anche altri tipi di *software*, compresi quelli

commerciali. Questo libro si rivolge quindi non solo a chi si occupa di ricerca musicale, ma a tutti i musicisti che vogliono guardare un passo più in là. Di che cosa avete bisogno per iniziare a leggerlo? Di una conoscenza di base della musica, dell'acustica e dell'uso pratico del *computer*. Niente di più.

QUALE COMPUTER?

Condizione necessaria perché un linguaggio per la sintesi diretta del suono abbia successo è, naturalmente, che sia disponibile sul maggior numero di macchine e sistemi operativi possibili, e che non dipenda da un particolare *hardware*. Csound è disponibile in versione per PC e Mac, per *workstation* in ambiente Unix e altre ancora. L'unico *hardware* aggiuntivo necessario (ma solo per l'ascolto del suono, non per la sintesi) è un convertitore digitale-analogico, che può benissimo essere una delle numerosissime schede audio a 16 bit oggi in commercio. Csound necessita di processori veloci, anche se l'utilizzo di processori inferiori porta agli stessi risultati, seppure ottenuti in tempi più lunghi. La scheda audio utilizzata influenza la qualità del suono riprodotto, specialmente in termini di rumore di fondo e di distorsione. Ma i *file* sonori presenti su *hard-disk* sono intrinsecamente di qualità paragonabile al CD o migliore. È anche possibile utilizzare una scheda di collegamento digitale con un DAT, e utilizzare quest'ultimo come convertitore digitale-analogico, anche se in questo modo si è legati all'uso di frequenze di campionamento standard, in pratica 32, 44.1 e 48 kHz. Csound, come tutti i linguaggi per la sintesi del suono, è nato per funzionare in tempo differito, ma con elaboratori veloci e con algoritmi di media complessità è ormai possibile, come abbiamo detto, anche la sintesi in tempo reale. Se il tempo necessario per la sintesi, per esempio, di un minuto di suono è superiore a un minuto, sarà necessario attendere il termine del processo di sintesi per l'ascolto del suono generato, e quindi si lavorerà in tempo differito. Se il tempo di sintesi è inferiore alla durata del suono, questo può essere ascoltato mentre viene generato, e quindi si lavorerà in tempo reale. Il tempo necessario per la sintesi del suono varia ovviamente a seconda della complessità del suono stesso, o meglio della complessità degli algoritmi usati per la sintesi.

PERCHÉ "IL SUONO VIRTUALE"?

Il titolo si riferisce a quella fase un po' misteriosa della composizione elettroacustica in cui abbiamo creato un suono a partire "dal nulla", cioè da idee, formule, desideri, metodi, e questo suono non ha ancora fatto il suo ingresso nel mondo fisico: è perciò un suono virtuale. Ciò è possibile solo con la sintesi digitale diretta. In questo testo troverete non solo informazioni sul programma stesso, ma anche una guida che vi porterà

attraverso la teoria e la pratica della sintesi e del trattamento del suono, nonché sui rapporti fra Csound e MIDI, e fra Csound e altri *software*. Non esistono manuali di Csound in italiano: cercheremo perciò man mano di darvi indicazioni sui termini inglesi in modo che possiate leggere, una volta finito questo testo, anche altri testi in inglese sul tema, se ne avrete voglia. Il nostro intento non è quello di esaurire tutte le informazioni su Csound ma quello di creare un ponte per chi non si è mai occupato di sintesi diretta, in modo tale che attraversare il fiume della conoscenza su questo tema non sia troppo difficoltoso. Questo libro viene da lontano: deriva dalle dispense che uno degli autori aveva realizzato nel 1977 per il corso di Musica Elettronica che a quel tempo teneva al Conservatorio "Luisa D´Annunzio" di Pescara. A tanta distanza di tempo, e visto l'enorme sviluppo che questa materia ha avuto negli ultimi anni, ben poco è rimasto del materiale originale. A quel tempo l'interfacciamento di sintetizzatori (analogici) con il calcolatore era materia di sperimentazione nei centri di calcolo universitari, e gli studi più avanzati usavano sintetizzatori MOOG ed EMS. Fare *computer music* significava recarsi in qualche grande centro di calcolo, imparare i misteri dei sistemi operativi, scrivere i programmi di sintesi, e solo al termine di una faticosa iterazione del processo scrittura/lancio del programma/correzione dai diffusori uscivano i suoni desiderati. Gli sviluppi della tecnologia tendono a nascondere la grande quantità di lavoro svolto per consentire ai musicisti di produrre facilmente musica con il proprio *personal computer*. Ma questo lavoro era guidato dal fascino e dall'entusiasmo per la ricerca: usare Csound significa potere mettere nuovamente le mani, se lo si desidera, in questa grande avventura, senza le difficoltà che le vecchie tecnologie imponevano di superare.

Desideriamo ringraziare Gabriel Maldonado, Luca Pavan, Giuseppe Emanuele Rapisarda e Fausto Sebastiani per avere letto le bozze e dato suggerimenti preziosi.

Buona lettura, e... *happy Csounding*!

1

CSOUND: COME FUNZIONA

1.1 ORCHESTRE, PARTITURE, *SOUND FILE*

Per ottenere qualsiasi tipo di suono Csound richiede la scrittura di due testi che vengono denominati rispettivamente:

1) **ORCHESTRA** (ingl. *orchestra*; l'estensione del *file* è *orc*)
2) **PARTITURA** (ingl. *score*; l'estensione del *file* è *sco*).[1]

In questi due *file* di testo scriveremo le informazioni sul tipo di "macchina virtuale" che vogliamo costruire e le operazioni che questa macchina deve compiere. Una volta scritti questi due testi chiederemo al programma Csound di *eseguire* questi dati e di creare un *file audio* (*sound file*), cioè un *file* dove sono rappresentate in codifica binaria tutte le caratteristiche del suono o dei suoni che abbiamo richiesto. A questo punto non ci rimane che chiedere alla nostra scheda di conversione di "suonare" il *sound file*. La scheda dunque legge i dati digitali scritti nel *file* di suono e li trasforma in segnale elettrico che viene inviato all'amplificatore e poi agli altoparlanti. In questo caso la scheda ha operato una conversione da digitale ad analogico, cioè ci ha consentito di ascoltare suoni le cui caratteristiche erano scritte in un *file*.

[1] A partire dalla versione 3.50 di CSound, è possibile includere questi due testi (orchestra e partitura) in un unico *file* con estensione .csd (vedi par. 1.B.1)

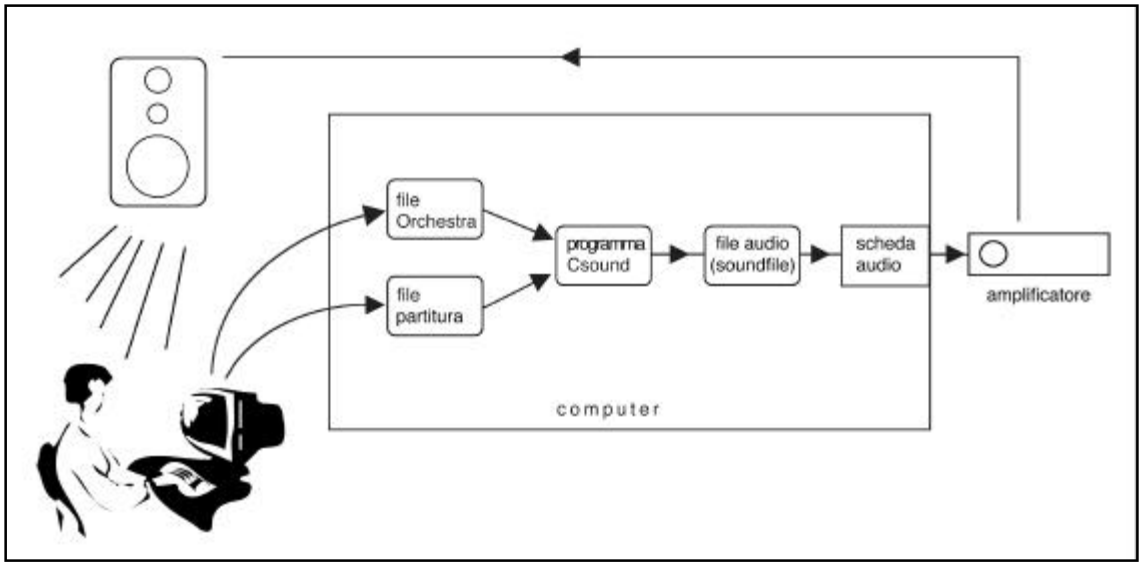


Fig. 1-1

In altri casi è possibile scrivere nei *file* orchestra e partitura (*score*) informazioni per l'elaborazione di un suono che abbiamo precedentemente campionato: per esempio, un suono di flauto può essere dapprima registrato (con un microfono collegato alla scheda

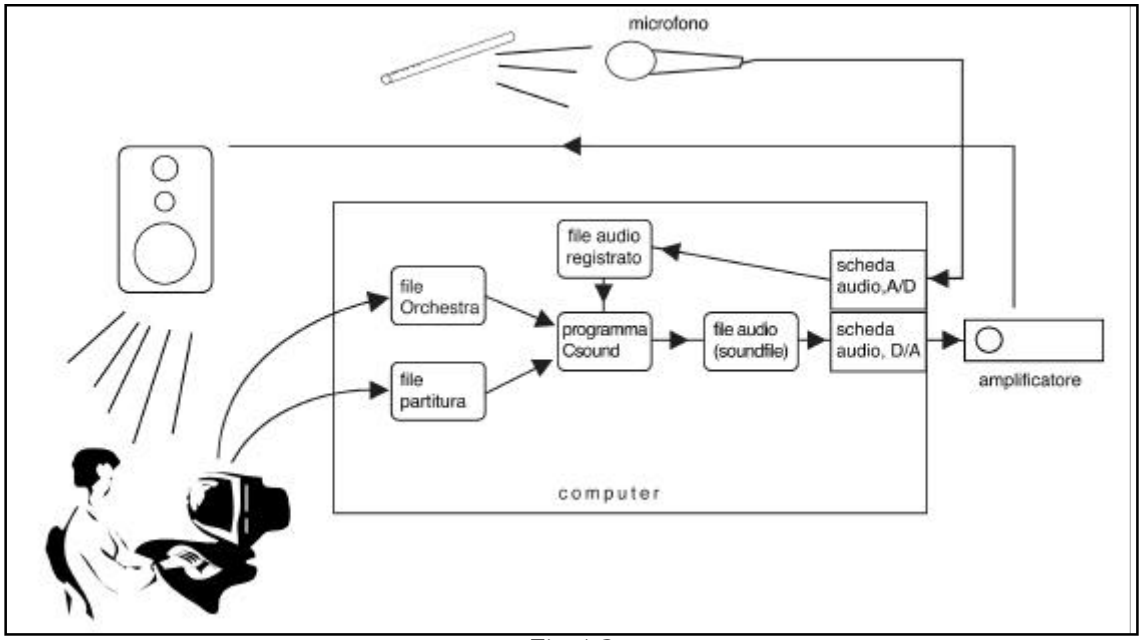


Fig. 1-2

audio che opererà una conversione analogico/digitale). Una volta che il suono è stato convertito in digitale, possiamo elaborare un'orchestra e una *score* in cui specificheremo come quel suono deve essere modificato, poi Csound eseguirà questi comandi e creerà un nuovo *sound file* che conterrà un suono di flauto, trasformato dal *computer* secondo i nostri desideri. Infine potremo ascoltare questo nuovo *sound file* operando, come prima, una conversione digitale/analogica.

1.2 COME SI OPERA CON WINDOWS/WCSHELL [2]

Come eseguire un'orchestra e una *score* già pronte e ascoltare il *file* risultante

1. Fate doppio clic sull'icona *WCShell* per entrare nel programma
2. Nella lista delle orchestre, fate clic su "oscil.orc"
3. Nella lista delle *score*, fate clic su "oscil.sco"
4. Fate clic sul pulsante *Csound* per avviare la sintesi
5. Alla fine della sintesi, chiudete Csound premendo <Invio>
6. Se tutto è andato a buon fine, fate clic sul pulsante *PLAY* per ascoltare

Come creare ed eseguire un'orchestra e una *score*

1. Fate doppio clic sull'icona *WCShell* per entrare nel programma
2. Scegliete *New Orc* dal menu *Orc*
3. Scrivete l'orchestra e salvatela con *Save as...* dal menu *File*
4. Chiudete l'*editor* di orchestra con *Exit* dal menu *File*
5. Scegliete *New Sco* dal menu *Sco*
6. Scrivete la partitura e salvatela con *Save as...* dal menu *File*
7. Chiudete l'*editor* di partitura con *Exit* dal menu *File*
8 Fate clic sui pulsanti *Update* di orchestra e *score* per aggiornare le liste, e controllate che il nome dei nuovi *file* appena creati compaiano nelle liste
9. Fate clic sul pulsante Csound per avviare la sintesi
10. Alla fine della sintesi, chiudete Csound premendo <Invio>
11. Se tutto è andato a buon fine, fate clic sul pulsante *PLAY* per ascoltare
12. Per modificare l'orchestra scegliete *Edit Orc* dal menu *Orc*.
13. Per modificare la partitura, scegliete *Edit Sco* dal menu *Sco*.

Consultate comunque il *file* "Come si installa WCshell" presente nel CDRom e l'Help in linea di WCShell, scegliendo *Help* dal menu *Help* o digitando il tasto F11.

[2] per usare Csound senza WCShell vedi paragrafo 1.A.5 a pag. 43.

1.3 COME SI OPERA CON MAC

Come eseguire un'orchestra e una *score* già pronte e ascoltare il *file* risultante

1. Entrate nella cartella dove sono i *file* del programma (*Csound* e *Perf*), fate doppio clic sull'icona di *Csound*, si aprirà una interfaccia grafica.
2. In alto vedrete due riquadri sopratitolati *Orchestra file* e *Score file*: lì vanno inseriti i nomi dell'orchestra e della *score* prescelti. Come vedete questi spazi sono ancora vuoti, fate clic su *Select* vicino all'icona dell'orchestra: apparirà una finestra di dialogo dove potete cercare la cartella in cui si trovano i nostri *file*. Fate doppio clic su "oscil.orc", cioè l'orchestra per fare il primo test. In questo modo ritrovate sull'interfaccia grafica il nome "oscil.orc" nello spazio del nome dell'orchestra e "oscil.sco" nello spazio del nome della *score*.
3. Fate clic su *Render*: apparirà un piccolo riquadro che avvisa che è stato lanciato il *Perfing*, cioè il *software* che genera il suono partendo da un'orchestra e da una *score*.
4. Quando il *perfing* è concluso, sul lato destro del piccolo riquadro apparirà la scritta "*close*" e contemporaneamente sulla finestra di testo troverete scritto "*0 errors in performance*" (zero errori in fase di esecuzione). A questo punto potete ascoltare il suono facendo clic sulla freccetta (come quella del tasto *play* dei registratori) che si trova sul lato sinistro del piccolo riquadro "*Perfing*". Potete ascoltare più volte il suono cliccando sulla freccetta, dopodiché potete chiudere la fase di ascolto facendo clic su *Close*.

Come creare ed eseguire un'orchestra e una *score*

1. Scegliete il menu *Options* e selezionate "*Auto Open File in Text Editor*" (se è selezionato quando riaprite *Options* comparirà un segno di spunta [-] accanto a questa opzione). Selezionando tale opzione compare una finestra di dialogo che vi consente di scegliere un *editor* di testo a vostro piacere con cui vanno aperti i *file* di orchestra e di partitura (ad esempio "*Simple Text*" per l'inizio può andare bene). I *file* vanno salvati con un nome che termini rispettivamente con ".orc" e ".sco".
2. Ora fate doppio clic sul nome dell'orchestra nella interfaccia grafica azzurra; si aprirà il testo contenente la vostra orchestra, che ovviamente può essere cambiato e salvato con un altro nome. Lo stesso potrete fare per la *score*.
3. Per fare una prova, fate doppio clic sul nome "oscil.sco", si aprirà una finestra con il testo della partitura: nell'ultima riga troverete scritto "i1 4 2".
4. Cancellate il numero 2 che si riferisce alla durata del suono da generare e scrivete 10, in questo modo indichiamo a Csound che il suono che desideriamo deve durare 10 secondi e non più 2 come nel suono che avete ascoltato nel test.

5. Chiudete la finestra, comparirà una finestra di dialogo in cui vi si chiede di salvare: fate clic su *Save*.
6. Fate ora clic su *generate* ed ascoltate, il suono generato dura ora 10 secondi

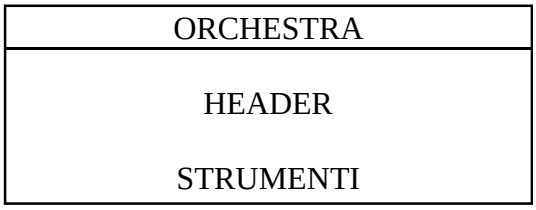
La gestione delle cartelle

Finché tutti i *file* (orchestre, *score*, *file* MIDI, *file* di analisi, programmi etc.) sono insieme nella stessa cartella non c'è bisogno di cambiare nulla. Se desiderate cambiare cartella, basta fare clic su *Default directories* nell'interfaccia azzurra: *Sound File Directory* (detta anche SFDIR) è la cartella dove vengono scritti i suoni da Csound; *Sound Sample Directory* (SSDIR) è la cartella dove devono essere posti i *sound file* che Csound deve leggere; *Analysis Directory* (SADIR) è la cartella dove verranno posti i *file* di analisi generati da Csound. Quando queste cartelle non sono definite, *perf* cerca tutti i *file* nella cartella dove esso stesso si trova.

1.4 COME SI SCRIVE UN'ORCHESTRA

Attenzione! Questa parte può risultare complessa perché si tratta di entrare a contatto con nuovi termini e un modo di pensare il suono diverso dal solito. L'importante è leggere con attenzione i prossimi paragrafi di questo capitolo, superati i quali tutto sarà più semplice e lo stesso schema si ripeterà ogni volta con aggiunte e modifiche. Procediamo per ordine:

Un'orchestra è sempre composta di due sezioni: intestazione (header) e strumenti [3]



HEADER

In inglese *header* significa intestazione o testata (come quella di un giornale). Lo *header* dà quattro informazioni di base che tutti gli strumenti dell'orchestra adotteranno.

STRUMENTI

Gli strumenti costituiscono le varie "macchine virtuali" che noi vogliamo costruire. In un'orchestra si possono scrivere uno o più strumenti.

[3] in realtà lo *header* può anche mancare, e in questo caso assume i valori di *default*: *sr=44100, kr=4410, ksmps=10, nchnls=1.*

COME SI SCRIVE UNO HEADER:

Lo *header* contiene sempre 4 informazioni:

sr frequenza di campionamento dei segnali audio (*sample rate*)

kr frequenza di campionamento dei segnali di controllo (*control rate*) (vedi anche par. 1.A.1)

ksmps rapporto fra sr e kr (per esempio se sr=48000 e kr=4800 allora ksmps=10); deve essere intero

nchnls (*number of channels*) numero di canali di uscita (1=mono, 2= stereo etc.)

Gli strumenti che seguono dipendono da queste informazioni.

Per esempio, se scriviamo nello *header* che il numero di canali è 2, non potremo scrivere strumenti quadrifonici in quell'orchestra, ma solo strumenti stereofonici.

Esempio di *header*:

```
sr      =   48000
kr      =   4800
ksmps   =   10
nchnls  =   1
```

COME SI SCRIVE UNO STRUMENTO

Gli strumenti sono molto più vari, perché dipendono da ciò che desideriamo creare.

La prima cosa da scrivere è il numero di strumento, con l'istruzione *instr* seguita da un numero. L'ultima è la parola *endin* (*end of instrument*) con la quale si termina uno strumento, secondo lo schema:

```
        instr 1
…
…
…
        endin
```

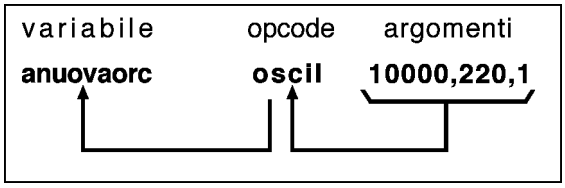
Per esempio:

```
        instr   1
anuovaorc oscil 10000, 220, 1
        out     anuovaorc
        endin
```

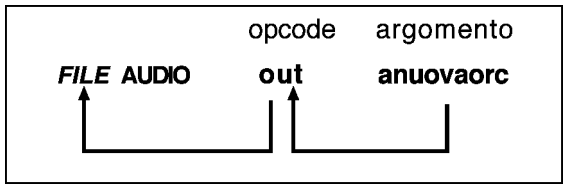
La parola *anuovaorc* indica il nome di una **variabile**.

Che cos'è una variabile?

La variabile è come un cassetto (che ha un nome, in questo caso *anuovaorc*), dove vengono depositati i risultati dell'operazione che la segue. Per esempio in questo caso abbiamo il **codice operativo** (in inglese *opcode*) *oscil* che simula un oscillatore al quale vengono assegnati alcuni valori, che in Csound vengono chiamati **argomenti**: per *oscil* il primo valore è l'ampiezza, il secondo la frequenza, il terzo il numero di funzione.



Andiamo per ordine:
come **ampiezza** nell'esempio abbiamo il valore **10000**, come **frequenza 220 Hz**, e come forma d'onda quella generata dalla **funzione numero 1** (questa funzione verrà poi definita in partitura). Tali valori vengono passati all'*opcode oscil*, il quale simula un'oscillatore che lavora con quei dati, e deposita il risultato nella variabile *anuovaorc*.
In questo caso dunque vogliamo generare suoni con ampiezza 10000 e frequenza 220 Hz. Trattandosi di suoni è bene utilizzare la frequenza di campionamento audio (*sr*), che è quella che consente una maggiore definizione.



Per fare ciò chiamiamo la variabile con un nome qualunque, purché cominci con la lettera *a* (che sta per audio).
Abbiamo così creato una variabile audio (utile appunto per i suoni) che si chiama *anuovaorc* (in altri casi possiamo chiamarla *a1, averde, aquadra, atarassia* etc.). Una volta che la variabile contiene il risultato, può a sua volta essere usata come argomento per l'*opcode out*.
out è un *opcode* che scrive nel *file* audio il risultato depositato in *anuovaorc* per farcelo ascoltare[4].
endin termina lo strumento e in questo caso anche l'orchestra.

[4] Quando usiamo CSound in tempo reale, *out* invia il risultato direttamente alla scheda audio.

Ricapitolando:

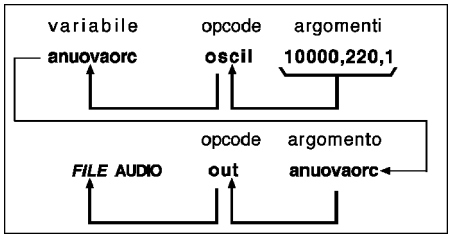
```
          instr   1
anuovaorc oscil   10000, 220, 1
          out     anuovaorc
          endin
```

anuovaorc è una **variabile audio**.

oscil è un *opcode* (codice operativo) che simula un oscillatore e richiede 3 argomenti: **ampiezza, frequenza, funzione**.

out è un *opcode* che scrive nel *file* audio il risultato depositato nell'argomento che lo segue: *anuovaorc* in questo caso diventa l'argomento dell'*opcode out.*

In pratica ogni *opcode* che esegue un'operazione richiede, per operare, alcuni dati che chiamiamo *argomenti*. Una volta eseguita l'operazione i risultati si depositano nella variabile che li precede, la quale a sua volta può diventare un argomento di un altro *opcode*, come *anuovaorc* nel caso di *out*.



Prima di *out* non abbiamo indicato alcuna variabile perché i risultati vengono inviati direttamente all'*hard disk* o alla scheda audio, quindi l'*opcode out* non genera risultati.

1.5 COME SI SCRIVE UNA PARTITURA (*SCORE*)

Anche la partitura è generalmente formata da due tipi di istruzioni: *funzioni* e *note*.

La lettera iniziale di ogni riga della partitura indica il tipo di istruzione: *f* per le funzioni, *i* per le note etc.

FUNZIONI

Le funzioni servono a creare forme d'onda di cui possiamo scegliere le caratteristiche.

Nel caso che la forma d'onda di un suono sia già definita altrove (per esempio quando vogliamo semplicemente ascoltare un suono campionato, che ha una sua forma d'onda) ci possiamo trovare a scrivere una partitura senza funzioni, cioè solo con note.

NOTE

Le note invece sono obbligatorie (beh, almeno una!). Esse non vanno pensate solo come le note di un pianoforte, bensì come eventi sonori, che possono andare da durate brevissime fino a durate di giorni interi e possono avere una altezza riconoscibile oppure no a seconda del tipo di strumento che abbiamo scritto in orchestra e del tipo di funzioni usate per la forma d'onda. L'ordine in cui vengono scritte non ha importanza, perché Csound, prima di sintetizzare il suono, esegue un riordino di tutti gli eventi.

COME SI SCRIVE UNA FUNZIONE

Le funzioni sono ovviamente molto varie. Ricordate che in orchestra avevamo stabilito che la forma d'onda era determinata dalla funzione numero 1?

```
anuovaorc   oscil 10000,220,1
```

Creiamo dunque questa funzione numero 1.

Prendiamo come primo esempio una funzione che serve a creare una forma d'onda sinusoidale:

```
f1   0    4096   10    1
```

f1 indica il **numero della funzione** (funzione 1).

0 (*creation time*) indica da che momento nella partitura viene creata la funzione: se il numero fosse 3 questa funzione verrebbe creata al terzo secondo

4096 è in questo caso il **numero di punti** che definiscono la tabella, cioè la nostra sinusoide sarà tracciata nella tabella mediante 4096 punti. Nella maggior parte dei casi il numero di punti richiesto in una funzione è una potenza di due (256, 512, 1024, 2048, 4096 etc.), in altri casi una potenza di due più uno, come vedremo. Il massimo valore possibile è 16777216 (2^{24}).

10 Questo quarto parametro è dedicato al **metodo di generazione** di forme d'onda o di funzioni, detto **GEN**. Ogni GEN ha un suo numero con cui viene identificata (GEN01, GEN02 etc.) ed utilizza un metodo diverso per la generazione di forme d'onda. Bene, in questo caso abbiamo la GEN 10, che

crea sinusoidi, perciò ogni volta che ci servirà una semplice sinusoide useremo questa GEN.

1 il fatto che ci sia solo un numero dopo il 10 significa che vogliamo solo una sinusoide. Se scrivessimo

```
f1   0   4096   10   1   1   1
```

creeremmo tre sinusoidi in rapporto armonico tra loro (fondamentale, seconda e terza armonica) e con la stessa ampiezza (1). Per ora accontentiamoci di una sola sinusoide.

Una domanda sorge a questo punto: “perché fare tutto questo lavoro per ottenere una semplice sinusoide, quando premendo un tasto su una tastiera viene fuori un suono già complesso, intonato, che dura quanto vogliamo?”. Domanda legittima. Risponderemo che se volete modificare quel suono nella tastiera e non conoscete i vari tipi di sintesi, non potrete mai farlo con cognizione di causa, e Csound è un mezzo di conoscenza straordinario che vi aiuterà a capire meglio anche la vostra tastiera o altre macchine più complesse, ma che soprattutto vi consentirà di fare cose che con il vostro campionatore o sintetizzatore non potrete mai realizzare. La risposta dunque è: avere pazienza all’inizio per acquisire strumenti di conoscenza adeguati a un compositore che voglia usare le nuove tecnologie con la consapevolezza di ciò che sta facendo.

Ricapitolando:

Numero di funzione	*Creation time* (tempo di creazione della tabella)	numero di punti o lunghezza della tabella	GEN	ampiezza della fondamentale
f1	**0**	**4096**	**10**	**1**

COME SI SCRIVONO LE NOTE

Le note sono composte da campi detti anche parametri (*parameter-fields* o p-*fields*).

Gli unici tre parametri obbligatori per ogni nota sono:

primo parametro (p1): indica quale strumento dell’orchestra deve suonare, per esempio *i1* significa lo strumento numero 1

secondo parametro (p2): rappresenta il momento di attacco della nota, per esempio 0 sarebbe all’inizio del pezzo, 3 al terzo secondo, 3.555 a tre secondi e 555 millesimi di secondo etc.

terzo parametro (p3): indica la durata della nota in secondi, per esempio 2 significa 2 secondi, oppure .5 significa mezzo secondo (lo zero prima della virgola si può omettere in Csound, ma soprattutto il separatore decimale è il punto, non la virgola, perciò 4 secondi e mezzo si scrive 4.5 e non 4,5).

Potremo inventare tanti altri parametri (p4, p5, p6...) e dare loro il significato che vogliamo a seconda di come scriveremo l'orchestra (ma questo lo vedremo più avanti).

Per ora scriviamo un esempio di partitura in cui le note abbiano solo 3 parametri, commentandone a lato il significato.

(Se vogliamo scrivere **commenti in un'orchestra o in una partitura** che Csound non valuti nei suoi calcoli, ma che siano utili per ricordarci il senso della nostra programmazione, basta mettere un punto e virgola prima di ogni commento; se si va a capo ci vuole un altro punto e virgola all'inizio della riga).

Esempio:

```
out a1                  ;questo è un commento di
                        ;cui Csound non terrà conto
```

Esempio di partitura:

```
f1   0   4096  10  1     ;num.funz. - action time - num. di punti - GEN - amp.della fondamentale
i1   0   3              ;suona  lo strumento 1, la nota parte all'inizio del pezzo, dura 3 secondi
i1   4   2              ;suona  lo strumento 1, la nota parte al quarto secondo, dura due secondi
i1   6   2              ;suona  lo strumento 1, la nota parte al sesto secondo, dura due secondi
```

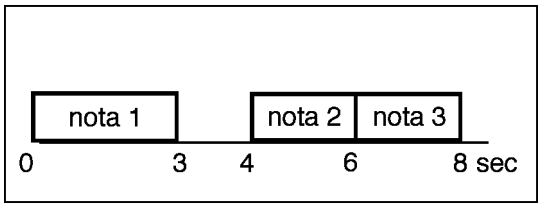
Fra la prima e la seconda nota c'è una pausa di un secondo: in Csound le pause non si scrivono, ma scaturiscono dai "vuoti" risultanti dalle durate e dai tempi d'attacco delle note.

Attenzione! Una partitura non vuole **mai** virgole, ma solo spazi o tabulazioni fra un parametro e l'altro. Un'orchestra richiede le virgole **solo** per separare gli argomenti l'uno dall'altro. Qualunque altra cosa a eccezione degli argomenti viene separata da spazi o tabulazioni anche in orchestra.

TIPS & TRICKS: è buona norma separare gli elementi di ogni riga di orchestra (variabile, opcode, argomenti) e di ogni riga di partitura (p-fields) con tabulazioni, in modo da ottenere un'orchestra ordinata e di facile lettura.

Ricapitolando:

```
; Numero strumento    Action time della nota   Durata della nota
i1                    0                        3
i1                    4                        2
i1                    6                        2
```



***ESERCIZIO**: Copiare l'orchestra e la partitura seguenti in due* file *diversi (il primo avrà estensione ".orc" e il secondo ".sco") e commentare ogni riga spiegando di cosa si tratta. Questo esercizio serve per memorizzare tutto ciò che è stato detto. Passate al prossimo paragrafo solo dopo aver ben capito e memorizzato header, strumenti, funzioni e note.*

Esempio di orchestra:

```
; oscil.orc

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

          instr       1
anuovaorc oscil       10000, 220, 1
          out         anuovaorc
          endin
```

Esempio di *score*:

```
; oscil.sco
f1   0   4096  10   1
i1   0   3
i1   4   2
```

Una volta create orchestra e *score*, eseguiamo la sintesi con Csound, seguendo le istruzioni dei paragrafi 1.2 (***Win***) o 1.3 (***Mac***). Se tutto è andato bene, nella finestra di Csound apparirà, verso la fine dei messaggi, anche la riga:

0 errors in performance

Se abbiamo commesso qualche errore, per esempio un errore di sintassi in orchestra, apparirà un messaggio simile a:

2 syntax errors in orchestra. compilation invalid

oppure

1 error in performance

In questo caso controlliamo attentamente l'orchestra o la *score*, correggiamo gli errori e rieseguiamo Csound. **A pagina xx** potete trovare una lista di errori di Csound e i rimedi per correggerli.

Altri paragrafi in questo capitolo:

APPROFONDIMENTI

1.A.1 FUNZIONAMENTO DI CSOUND

1.A.2 COSTANTI E VARIABILI

1.A.3 LA SINTASSI DI CSOUND

1.A.4 I "MATTONI" DI CSOUND

1.A.5 USARE IL COMANDO CSOUND

1.B.1 ORCHESTRA, *SCORE* E *FLAG* IN UN UNICO *FILE*: IL FORMATO *CSD*

1.C.1 I TRANSITORI DI ATTACCO E DI ESTINZIONE

1.D.1 STORIA DEI LINGUAGGI DI SINTESI

LISTA DEGLI *OPCODE*

k1	**oscil**	**ampiezza, frequenza, funzione**
a1	**oscil**	**ampiezza, frequenza, funzione**
	out	**segnale in uscita**
k1	**line**	**valore1, durata, valore2**
a1	**line**	**valore1, durata, valore2**
k1	**linseg**	**valore1, durata1, valore2, durata2, valore3 ....**
a1	**linseg**	**valore1, durata1, valore2, durata2, valore3 ....**
k1	**expon**	**valore1, durata, valore2**
a1	**expon**	**valore1, durata, valore2**
k1	**expseg**	**valore1, durata1, valore2, durata2, valore3 ....**
a1	**expseg**	**valore1, durata1, valore2, durata2, valore3 ....**
a1	**linen**	**ampiezza, attacco, durata, estinzione**

2

SINTESI ADDITIVA

2.1 SINTESI ADDITIVA A SPETTRO FISSO

La sintesi additiva è un tipo di sintesi con la quale si può creare una forma d'onda comunque complessa a partire dalla somma di forme d'onda semplici, tipicamente una somma di sinusoidi (vedi approfondimento 2.A.1 e 2.B.1).

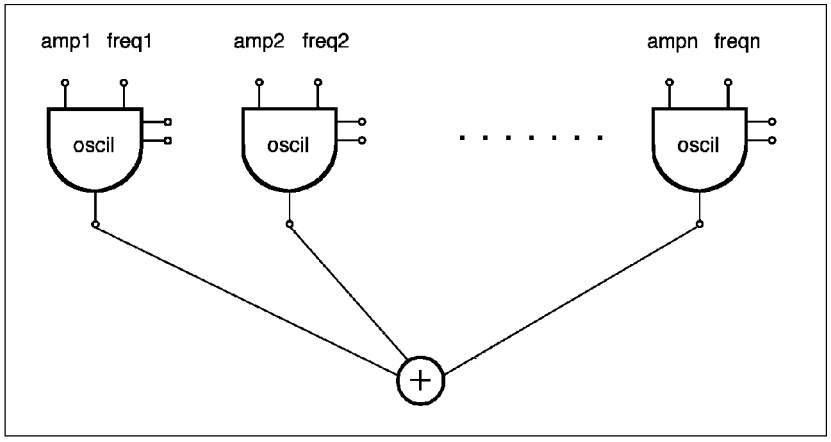


Fig. 2-1

La più semplice da realizzare è la sintesi additiva a spettro fisso con **sinusoidi in rapporto armonico fra loro**. Come abbiamo visto, questo tipo di sintesi può essere

implementata tramite l'uso della GEN10. Infatti per ottenere una forma d'onda complessa si possono determinare una serie di *componenti* (armoniche in questo caso) in relazione a una fondamentale e stabilirne le ampiezze.

Per esempio, come abbiamo accennato parlando delle funzioni, è possibile ottenere un'onda a dente di sega o un'onda quadra approssimate assegnando valori di ampiezza alle componenti armoniche, determinati in modo tale che l'ampiezza di ogni armonica sia pari a 1 / l'ordine dell'armonica stessa. La differenza fra i due esempi seguenti è che nell'onda a dente di sega sono presenti tutte le armoniche, mentre nell'onda quadra sono presenti solo quelle dispari:

Esempio: onda a dente di sega (limitata alle prime 10 armoniche)

```
f1   0   4096   10   10   5   3.3   2.5   2   1.6   1.4   1.25   1.1   1
```

Esempio: onda quadra (limitata alle prime 9 armoniche)

```
f1   0   4096   10   10   0   3.3   0     2   0     1.4   0      1.1
```

Il paragrafo 2.3 servirà ad eliminare alcune delle limitazioni, date dalla armonicità delle componenti e dalla fissità dello spettro nel tempo.

Naturalmente la sintesi additiva a spettro fisso può non essere la più interessante, vediamo perciò come si può realizzare un modello più complesso.

Altri paragrafi in questo capitolo:

2.2 SINTESI ADDITIVA A SPETTRO VARIABILE

2.3 FASE E *DC OFFSET*: GEN09 E GEN19

2.4 OSCILLATORI COMPLESSI: *BUZZ* E *GBUZZ*

APPROFONDIMENTI

2.A.1 SOMMA DI ONDE

2.A.2 TIMBRO

2.B.1 SINTESI ADDITIVA: CENNI STORICI E TEORIA

2.C.1 L'OSCILLATORE DIGITALE: COME FUNZIONA

LISTA DEGLI *OPCODE*

a1	**buzz**	**ampiezza, frequenza, numero delle armoniche, numero della funzione [, fase iniziale]**
a1	**gbuzz**	**ampiezza, frequenza, numero delle armoniche, numero d'ordine dell'armonica più bassa, moltiplicatore per le ampiezze[, fase iniziale]**
a1	**oscili**	**ampiezza, frequenza, numero della funzione [, fase iniziale]**

3

SINTESI SOTTRATTIVA

3.1 RUMORE BIANCO E FILTRI

La sintesi sottrattiva nasce dall'idea di poter creare un suono sottraendo ampiezza ad alcune componenti di un altro suono, più complesso di quello da ottenere, attraverso l'uso di filtri. Un filtro è un dispositivo che lascia passare certe frequenze meglio di altre.

Innanzitutto vediamo come ottenere facilmente un rumore bianco[1] con Csound. L'*opcode* che useremo sarà *rand*, il quale richiede solo un argomento, cioè l'ampiezza. Perché un *opcode* come *oscili* richiede tre argomenti (ampiezza, frequenza e funzione) e un generatore di rumore bianco solo l'ampiezza? Un rumore bianco in realtà nasce dalla compresenza di tutte le frequenze udibili, perciò non sarebbe possibile definirne una in particolare. Questo implica che *rand* genera forme d'onda casuali (che non contengono cioè forme d'onda cicliche); ciò significa che la forma d'onda non ha bisogno di essere definita da una funzione esterna. L'unico argomento che ci rimane da definire è dunque l'ampiezza[2].

```
;noise.orc
        sr      =   44100
        kr      =   4410
        ksmps   =   10
        nchnls  =   1
```

[1] Viene chiamato *rumore bianco* quel suono che contiene tutte le frequenze udibili, in analogia con l'ottica, in cui il colore bianco contiene tutti i colori dello spettro visibile.

[2] Si tratta in realtà di forme d'onda pseudo - casuali, come è spiegato nel par. 3.8.

```
        instr   1
a1      rand    p4
        out     a1
        endin
```

Esempio di partitura:

```
;noise.sco
i1   0   3   20000
```

Abbiamo creato in questo modo un rumore bianco di 3 secondi di ampiezza 20000.

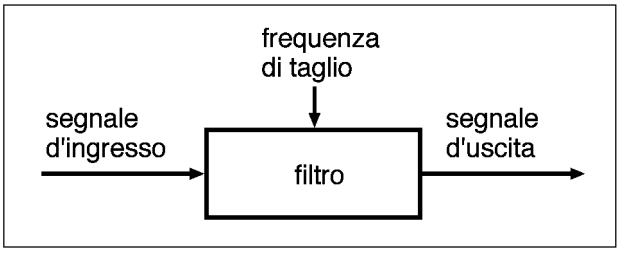
Vediamo ora come è possibile utilizzare filtri da applicare al suono che abbiamo creato.

Cominciamo con i filtri passa-basso e passa-alto. A un filtro si invia un segnale di ingresso, cioè il suono che vogliamo modificare. Possiamo definire il modo in cui questo filtro debba operare, tipicamente quali frequenze verranno eliminate o attenuate nel nostro suono. Il risultato di questa operazione, cioè i dati che descrivono il suono modificato dal filtro, verranno depositati come al solito in una variabile audio.

Innanzitutto vediamo come si creano i filtri passa-basso e passa-alto con Csound, i loro rispettivi *opcode* e la loro sintassi (argomenti):

Filtro passa-basso
a1 tone segnale d'ingresso, frequenza di taglio [3]

Filtro passa-alto
a1 atone segnale d'ingresso, frequenza di taglio



[3] Vi è, in *tone*, *atone*, *reson* e *areson*, un argomento opzionale, *istor* o *azzeramento memoria*, che determina se lo spazio di memoria interno al filtro vada azzerato a ogni nuova nota o no, a seconda che il suo valore sia 0 oppure 1. Un filtro infatti lavora non solo sul campione di suono corrente, ma anche su uno o più campioni precedenti, che vengono conservati proprio in uno spazio di memoria interno al filtro. Se questo spazio di memoria viene azzerato (*istor* = 0, il valore di *default*) a ogni nuova nota, il filtro "dimentica" i campioni di suono relativi alla nota precedente. Se invece non viene azzerato, in alcuni casi vi è, all'inizio di una nuova nota, una specie di "coda sonora" relativa alla nota precedente. Nella maggior parte dei casi questa "coda sonora" non è avvertibile, ma a volte (come nel caso di filtri passa-banda a banda molto stretta) si può avvertire un troncamento del suono precedente.

Altri paragrafi in questo capitolo:

3.2 FILTRI PASSA-BASSO DEL PRIMO ORDINE

3.3 FILTRI PASSA-ALTO DEL PRIMO ORDINE

3.4 FILTRI DI ORDINI SUPERIORI

3.5 FILTRI PASSA-BANDA

3.6 UNITÀ DI GUADAGNO: *RMS, GAIN, BALANCE*

3.7 FILTRI A PIÙ POLI E FILTRI RISONANTI

3.8 I GENERATORI DIGITALI DI RUMORE E GLI ARGOMENTI OPZIONALI DI *RAND*

APPROFONDIMENTI

3.A.1 SINTESI SOTTRATTIVA: CENNI STORICI

3.B.1 SINTESI SOTTRATTIVA: TEORIA

LISTA DEGLI *OPCODE*

k1	**rand**	**ampiezza**
a1	**rand**	**ampiezza**
a1	**tone**	**segnale d'ingresso, frequenza di taglio**
a1	**atone**	**segnale d'ingresso, frequenza di taglio**
a1	**reson**	**segnale d'ingresso, frequenza centrale, larghezza di banda**
a1	**butterhp**	**segnale d'ingresso, frequenza di taglio (passa-alto)**
a1	**butterlp**	**segnale d'ingresso, frequenza di taglio (passa-basso)**
a1	**butterbp**	**segnale d'ingresso, frequenza centrale, larghezza di banda (passa-banda)**
a1	**butterbr**	**segnale d'ingresso, frequenza centrale, larghezza di banda (elimina-banda)**
k1	**rms**	**segnale d'ingresso**
a1	**gain**	**segnale d'ingresso, ampiezza RMS**
a1	**balance**	**segnale d'ingresso, segnale di confronto**

4

DIAGRAMMI DI FLUSSO

4.1 RAPPRESENTAZIONE GRAFICA DI UN PROCESSO

Qualsiasi serie di eventi collegati fra loro (che chiameremo *processo*) può essere rappresentata in varie forme: testuali, grafiche, sonore etc. Un'orchestra Csound descrive appunto un processo, cioè una serie di eventi collegati fra loro il cui scopo è quello di produrre suoni. Ma la sua *leggibilità*, cioè la capacità di farci percepire a colpo d'occhio la funzione del processo che descrive, non è certo delle migliori. Inoltre chi vuole comprendere fino in fondo un'orchestra Csound scritta da un'altra persona è costretto a una fatica non indifferente, specialmente se si tratta di un'orchestra abbastanza complessa. Come si può risolvere questo problema?

Si può ricorrere a una rappresentazione non testuale, per esempio proprio a una rappresentazione grafica, che chiameremo *diagramma di flusso*. I diagrammi di flusso sono largamente usati in moltissimi campi dell'attività umana, appunto perché permettono la comprensione "a colpo d'occhio" di processi, così come un percorso segnato sulla mappa di una città è di comprensione molto più immediata di una rappresentazione testuale del tipo: "gira alla terza a destra, quando arrivi a una cabina telefonica gira a sinistra, prosegui dritto fino al distributore di benzina...".

4.2 ANDIAMO AL MARE

Supponiamo di trovarci di fronte a una situazione di questo tipo: un amico ci propone di andare al mare insieme domenica prossima al mattino presto, se non piove, con la

nostra auto. Quali sono le operazioni da compiere e le scelte da fare? Elenchiamole, anche quelle banali ed evidenti.

1. Mettiamo la sveglia e ci svegliamo.
2. Guardiamo fuori: piove? Se piove torniamo a letto.
3. Prendiamo quello che ci serve e usciamo.
4. Avviamo l'auto. Parte? Se non parte torniamo a letto.
5. Andiamo a prendere il nostro amico a casa sua.
6. Il nostro amico è pronto? Se non è pronto, aspettiamo.
7. Andiamo al mare.

Possiamo esprimere questo semplice processo con un diagramma di flusso? Proviamo. In Fig.4-1 possiamo vedere proprio il diagramma di flusso relativo al processo "Andiamo al mare". Notiamo come l'ordine cronologico sia rappresentato dall'alto al basso e da sinistra a destra (come nella nostra scrittura). Alcuni testi, per esempio "Mettiamo la sveglia e ci svegliamo", sono racchiusi in un rettangolo a bordi arrotondati, mentre altri, che terminano con un punto di domanda, per esempio "Guardiamo fuori. Piove?" sono racchiusi in un poligono con punte a destra e a sinistra. I primi sono *azioni*, e hanno un

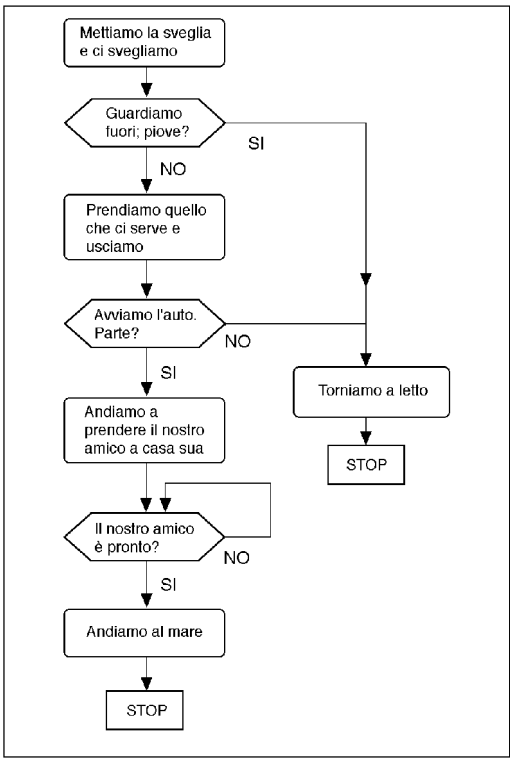


Fig. 4-1

solo punto di uscita. I secondi sono *test*, e hanno due punti di uscita, a seconda che la condizione al loro interno (cioè il risultato del test) sia vera o falsa.Quindi, se per esempio la condizione “Guardiamo fuori. Piove?” è vera, non possiamo andare al mare, quindi “Torniamo a letto”, e il processo termina con l’azione “STOP”. Se invece la condizione è falsa, cioè non piove, possiamo proseguire con il processo.

La rappresentazione grafica di processi legati alla musica è ormai un fatto comune, dal momento che molti *software* la utilizzano come interfaccia con l’utente, per esempio MAX, KYMA, Patchwork etc., e quindi ci sembra utile imparare a capire come tale rappresentazione funziona.

Altri paragrafi in questo capitolo:

4.3 SIMBOLOGIA

4.4 DIAGRAMMI COMPLESSI

5

STEREO E SEGNALI DI CONTROLLO, VIBRATO, TREMOLO, SUONO IN 3D

5.1 ORCHESTRE STEREOFONICHE

Finora abbiamo trattato orchestre il cui *header* conteneva l'istruzione *nchnls=1*, e quindi ogni strumento doveva necessariamente essere monofonico. Introduciamo ora la possibilità di scegliere fra uno strumento stereofonico e uno monofonico variando lo *header*.

Invece di usare l'*opcode out,* useremo per l'uscita del suono l'*opcode outs.* La sua sintassi prevede due argomenti:

outs segnale in uscita sul canale sinistro, segnale in uscita sul canale destro

Vediamo un esempio di orchestra:

```
;stereo.orc
          sr      =   44100
          kr      =   4410
          ksmps   =   10
          nchnls  =   2               ;notiamo che nchnls=2
          instr       1
```

```
asine     oscil   10000, 1000, 1
aquadra   oscil   10000, 220, 2
          outs    asine, aquadra  ;asine sul canale sinistro, aquadra sul canale destro
          endin
;
          instr   2
awn       rand    10000
          outs    awn, awn        ;awn sia sul canale sinistro che su quello destro
          endin
```

con la *score*:

```
;stereo.sco
f1   0   4096   10   1
f2   0   4096   10   1   0   .33   0   .2   0   .14   0   .11
i1   0   5
i2   6   5
```

Abbiamo così creato un'orchestra stereofonica (*nchnls=2*) con due strumenti:

instr1: si ascolta *asine* a sinistra e *aquadra* a destra
instr2: si ascolta *awn* sia a sinistra che a destra, ottenendo un effetto monofonico

Vediamo ora un'altra orchestra che ci consenta di posizionare tre note nel fronte stereo, per esempio la prima nota a destra, la seconda al centro e la terza a sinistra.

```
;stereo1.orc
          sr      =   44100
          kr      =   4410
          ksmps   =   10
          nchnls  =   2
          instr       1
ast       oscili      p4, p5, 1
          outs        ast*(1-p6), ast*p6
          endin
```

outs prevede due argomenti:
il primo, cioè il segnale in uscita sul canale sinistro, sarà *ast* moltiplicato per 1 meno il valore che diamo a p6 (il segno di moltiplicazione in Csound è l'asterisco *);

il secondo, cioè il segnale in uscita sul canale di destra, sarà *ast* moltiplicato per il valore che diamo a p6.

Ed ecco un esempio di *score*:

```
;stereo1.sco
f1   0    4096   10    9     8     7     6      5     4     3     2     1
i1   0    1            10000             200                 1           ;suono a destra
i1   1.1  1            10000             200                 .5          ;centro
i1   2.2  1            10000             200                 0           ;sinistra
```

Vediamo come mai la prima nota uscirà a destra: abbiamo assegnato al sesto parametro il valore 1 (p6=1). In base a ciò che abbiamo scritto in orchestra, in uscita sul canale sinistro avremo i valori di *ast* moltiplicati per (1-p6) cioè in questo caso dato che p6=1, otterremo *ast**(1-1). 1 meno 1 è uguale a 0, perciò sul canale sinistro tutti i valori di *ast* verranno annullati perché li moltiplichiamo per 0. Sul canale destro abbiamo *ast*p6*, perciò se p6=1, i valori d'ampiezza istantanea che definiscono *ast* verranno tutti moltiplicati per 1, quindi rimarranno invariati (qualsiasi numero moltiplicato per 1 rimane invariato). Conseguenza di ciò è che tutto il suono sarà a destra.

Vediamo anche altre possibilità; lo schema generale è:

```
outs [variabile audio]*(1-p6), [variabile audio]*p6
```

Perciò se **p6=1**

outs [variabile audio]*(1-1) = **0**, [variabile audio]*1 = **variabile audio**

0 a sinistra, tutto il segnale a destra

se **p6=0**

outs [variabile audio]*(1-0) = **variabile audio**, [variabile audio]*0 = **0**

tutto il segnale a sinistra, 0 a destra.

se **p6=.5**

outs [variabile audio]*(1-.5) = **variabile audio / 2,** [variabile audio]*.5 = **variabile audio / 2**

metà del segnale a sinistra, metà a destra, suono al centro

se **p6=.75**

outs [variabile audio]*(1-.75) = **variabile audio* 1/4**, [variabile audio]*.75 = **variabile audio * 3/4**

un quarto del segnale a sinistra, tre quarti a destra

In questo modo possiamo assegnare a ogni nota una posizione fissa nel fronte stereo.

Se vogliamo invece che nel corso della nota il suono si sposti, per esempio, da sinistra a destra e poi ritorni a sinistra, possiamo usare una variabile di controllo i cui valori vadano da 0 a 1 e poi tornino a 0.

Esempio di orchestra:

```
;spostamento fissato in orchestra
            instr   1
kstereo     linseg  0, p3/2, 1, p3/2, 0
ast         oscili  p4, p5, 1
            outs    ast*(1-kstereo), ast*kstereo
            endin

;spostamento stabilito dalla score

            instr   2
kstereo     linseg  p6, p3/2, p7, p3/2, p8
ast         oscili  p4, p5, 1
            outs    ast*(1-kstereo), ast*kstereo
            endin
```

Nel secondo strumento abbiamo previsto la possibilità di avere tre posizioni, iniziale, intermedia e finale per ogni nota e queste posizioni verranno specificate nei parametri p6, p7 e p8 della *score* con valori fra 0 e 1.

Per esempio:

```
f1    0   4096   10      1
i2    0   5      20000   500   .5   1    0     ;dal centro a destra , poi a sinistra
i2    6   5      20000   500   0    .5   .5    ;da sinistra al centro, poi rimane al centro
i2    12  4      20000   500   0    .5   1     ;da sinistra a destra
```

Il metodo appena descritto è il più semplice, ma non dà risultati del tutto soddisfacenti. Infatti ascoltando con attenzione la terza nota (quella che si sposta da sinistra a destra) si può notare come il segnale, quando si trova al centro, sia di ampiezza più bassa di quando si trova a sinistra o a destra. Questo perché l'intensità percepita è proporzionale alla potenza del segnale, la quale a sua volta è proporzionale al quadrato dell'ampiezza.

Perciò se abbiamo un'unica sorgente sonora (come nel caso che il segnale provenga solo dal canale sinistro o solo dal destro), detta P la potenza e A l'ampiezza sarà:

$$\mathbf{P = A^2}$$

Quindi nel caso A=1 si avrà

$$\mathbf{P=1^2 = 1}$$

Mentre se le sorgenti sonore sono due, e per ciascuna sorgente si ha A = .5, si ottiene:

$$\mathbf{P_{sinistra} = A_{sinistra^2} = .52 = .25}$$

$$\mathbf{P_{destra} = A_{destra^2} = .52 = .25}$$

e la potenza totale sarà quindi

$$\mathbf{P_{totale} = P_{sinistra} + P_{destra} = .25 + .25 = .5}$$

pari perciò alla metà della potenza che avevamo quando il segnale proveniva da un solo canale.

Vi sono diverse possibili soluzioni a questo problema, ma una delle più semplici, dovuta a Charles Dodge, è di definire i fattori di moltiplicazione per i due canali, sinistro e destro (quelli che nell'orchestra appena vista abbiamo chiamato *1-kstereo* e *kstereo*) come la *radice quadrata* del segnale di controllo per lo stereo. Per il calcolo della radice quadrata creeremo la funzione matematica **sqrt** (*square root*) di Csound. Una possibile orchestra sarà perciò:

```
; stereo con radice quadrata

          instr   1
kstereo   linseg  0, p3/2, 1, p3/2, 0
ast       oscili  p4, p5, 1
```

```
kleft     =     sqrt(1-kstereo)          ; radice quadrata di 1-kstereo
kright    =     sqrt(kstereo)            ; radice quadrata di kstereo
          outs  ast*kleft, ast*kright
          endin
```

Oltre all'*opcode outs* ne esistono altri due, *outs1* e *outs2* per mandare in uscita rispettivamente solo il canale sinistro o solo il canale destro. Esiste inoltre l'*opcode outq* per orchestre quadrifoniche, e i rispettivi *opcode* per i canali singoli, *outq1*, *outq2*, *outq3*, *outq4* (per ascoltare la quadrifonia occorre naturalmente una scheda audio quadrifonica). Ricapitoliamo la sintassi di tutti questi *opcode* (*asig* è il segnale in uscita):

```
out       asig                     ; canale unico (mono)
outs      asig1, asig2             ; canale sinistro, canale destro
outs1     asig                     ; solo il canale sinistro
outs2     asig                     ; solo il canale destro
outq      asig1, asig2, asig3, asig4 ; I canale, II canale, III canale, IV canale
                                   ; (quadrifonia)
outq1     asig                     ; I canale
outq2     asig                     ; II canale
outq3     asig                     ; III canale
outq4     asig                     ; IV canale
```

Altri paragrafi in questo capitolo:

6

AUDIO DIGITALE

6.1 IL SUONO DIGITALE

Quando ascoltiamo musica riprodotta che non provenga da DVD (*Digital Versatile Disc*), CD (*Compact Disc*), DAT (*Digital Audio Tape*), MiniDisc o da *computer*, ma da disco in vinile (LP), nastro analogico, audiocassetta, radio o televisione, il segnale elettrico (che viene trasformato in segnale acustico dai diffusori) è di tipo *analogico*, cioè viene rappresentato da una variazione di tensione elettrica che descrive esattamente l'andamento del suono.

Nel caso di un segnale numerico (come quello che proviene da *Compact Disc*), invece, il segnale è di tipo *digitale*, viene cioè rappresentato da una serie di *bit*, o unità minime di informazione digitale.

Dal momento che, come si sa, un suono comunque complesso viene completamente definito una volta che ne siano note le *ampiezze istantanee* [1], per riprodurre correttamente un suono mediante un segnale digitale bisognerà che la sorgente (DVD, CD, DAT, *computer* etc.) invii al sistema di riproduzione una serie di numeri, ciascuno dei quali rappresenta un valore di ampiezza istantanea.

Poiché sul supporto di memorizzazione (sia esso quello meccanico del CD, sia quello magnetico del DAT) i dati vengono scritti e letti come una serie di 1 (uno) e 0 (zero), e vengono applicate particolari correzioni degli errori di lettura, anche un moderato difetto della superficie del CD (sporcizia, graffi) o una moderata

[1] L'ampiezza istantanea è il valore di ampiezza dell'onda sonora istante per istante.

diminuzione del livello di magnetizzazione del nastro del DAT, non portano di solito a errori di lettura. Inoltre ogni copia è virtualmente identica all'originale, al contrario di quanto accade per i nastri magnetici analogici, in cui a ogni processo di copia corrisponde un peggioramento della qualità[2].

Anche il rapporto segnale/rumore del CD e del DAT (grosso modo equivalenti) è nettamente migliore di quello dei registratori analogici: è infatti teoricamente di circa 96 dB, contro i 60...70 dB dei registratori analogici senza riduttori di rumore.

6.2 CONVERSIONE ANALOGICO/DIGITALE E DIGITALE/ANALOGICA

Nell'audio digitale c'è bisogno di apparecchiature particolari, che siano in grado di tradurre i segnali da analogico a numerico, e da numerico ad analogico. Queste apparecchiature sono, rispettivamente, il *convertitore analogico/digitale*, o ADC (*Analog to Digital Converter*), e il *convertitore digitale/analogico*, o DAC (*Digital to Analog Converter*).

Esaminiamo brevemente le principali caratteristiche di funzionamento del processo di *conversione analogico/digitale*, cioè la trasformazione di un segnale da analogico a digitale. Tale conversione consiste nel trasformare una tensione elettrica in un segnale numerico, che esprima istante per istante il valore della tensione stessa.

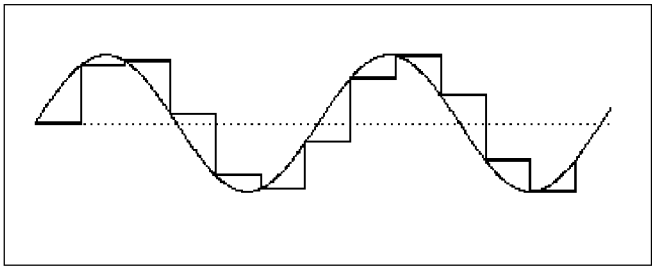


Fig. 6-1

Con riferimento alla fig. 6-1, la linea continua rappresenta il segnale analogico, cioè l'andamento della tensione elettrica. Suddividiamo ora il segnale in un certo numero di intervalli di tempo, e in ogni momento "congeliamo" il valore istantaneo del segnale stesso: otterremo un *segnale a gradini*, che assomiglia grosso modo a quello originale.

Naturalmente, più piccoli saranno gli intervalli di tempo, e più il segnale a gradini sarà simile all'originale. Al limite, per intervalli di tempo infinitamente piccoli, il segnale a gradini e quello originale coincideranno.

L'intervallo di tempo di cui abbiamo appena detto si dice *periodo di campionamento*, e il suo inverso, cioè 1/periodo, si dice *frequenza di campionamento*

[2] In realtà questo non è del tutto vero, dal momento che eventuali errori di lettura vengono corretti da sofisticati sistemi di correzione dell'errore.

o *sr* (*sample rate*). Per una corretta conversione è necessario che la frequenza di campionamento sia almeno doppia della massima componente frequenziale contenuta nel segnale da convertire. Se cioè desideriamo convertire un segnale che contenga frequenze fino a 20000 Hz, dovremo utilizzare una frequenza di campionamento di almeno 40000 Hz.

Se si tenta di convertire in digitale un segnale con una *sr* troppo bassa, si ha il fenomeno del *foldover* (ripiegamento): le componenti frequenziali che superano la metà della *sr* vengono *riflesse* al di sotto di questa. Per esempio, una componente frequenziale di 11000 Hz, convertita con una *sr* di 20000 Hz, darà luogo a una componente di *foldover* di 9000 Hz, non presente nel segnale analogico originale (vedi 6.B.1)

Un altro elemento che influisce sulla qualità della conversione è il numero di valori di ampiezza differenti che possono essere prodotti dal convertitore. Esso non potrà certamente produrre un numero infinito di cifre: tipicamente sarà espresso da un numero intero binario a 16 bit, che può esprimere solo numeri interi compresi nella gamma di valori che va da -32768 a +32767, quindi per un totale di 65535 valori di ampiezza differenti. Se usassimo invece numeri a 7 bit potremmo disporre solo di 127 valori di ampiezza differenti, con una pessima qualità audio (per maggiori particolari, vedi il par. 6.A.1). Per una buona qualità audio sono necessari almeno numeri interi binari a 16 bit. Esistono *standard* di qualità ancora migliore: per esempio il DVD (*digital versatile disk*) prevede codifiche a 16, 20 e 24 bit e frequenze di campionamento da 44100 Hz a 192000 Hz. Con 24 bit la gamma di valori va da -8388608 a +8388607 con 16777216 valori di ampiezza differenti.

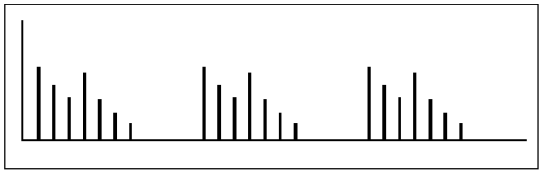


Fig. 6-2

Per quanto riguarda il processo inverso, la conversione digitale/analogica, valgono più o meno le medesime considerazioni. Dobbiamo però tenere presente che il segnale digitale convertito in analogico non è un segnale continuo, ma a gradini, che contiene più immagini dello spettro fondamentale, dette *alias*, e che sono ripetizioni dello spettro a frequenze più alte dovute alla distorsione armonica introdotta dai *gradini* (vedi fig. 6-2). Per evitare disturbi è necessario rimuovere queste immagini con un filtro analogico, che viene perciò detto *filtro anti-aliasing*. Si tratta di un filtro passabasso con frequenza di taglio pari alla massima frequenza audio che interessa, quindi, in genere

fc = sr / 2 (dove *fc* sta per *cutoff frequency*, o frequenza di taglio)

Poiché un filtro analogico non può avere una curva di taglio ideale (cioè lasciar passare immutate le frequenze desiderate e cancellare completamente le altre), e inoltre più è ripido e più introduce irregolarità nella risposta in frequenza (*ripple*) e distorsioni di fase, si preferisce oggi aumentare la frequenza di campionamento nel processo di conversione D/A, almeno quadruplicandola (*oversampling*), in modo tale da spostare a frequenze più alte le immagini degli spettri indesiderati, e in modo quindi da potere utilizzare filtri meno ripidi, che introducono pochissimo *ripple* e pochissima distorsione di fase (vedi fig. 6-3).

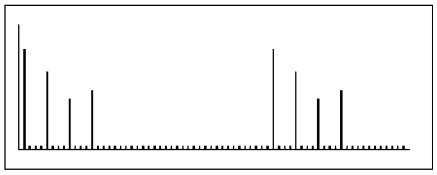


Fig. 6-3

Altri paragrafi in questo capitolo:

6.3 SCHEDE AUDIO E FORMATO DEI *FILE* AUDIO

6.4 AUDIO PER APPLICAZIONI MULTIMEDIALI (CENNI)

APPROFONDIMENTI

6.A.1 CONVERSIONE DIGITALE/ANALOGICA E ANALOGICO/DIGITALE

6.B.1 *FOLDOVER*

7

I SUONI CAMPIONATI E LA LORO ELABORAZIONE

7.1 I SUONI CAMPIONATI E GLI *OPCODE SOUNDIN* E *DISKIN*

Per ascoltare un suono campionato possiamo usare l'*opcode soundin*, il quale non consente elaborazioni particolari ma è semplice da usare. L'*opcode soundin* è un generatore audio che prende i propri dati direttamente da un *file* audio preesistente, perciò nella partitura non avremo bisogno di scrivere una funzione, in quanto la forma d'onda è già quella del *file* audio. D'altra parte come argomenti non vengono indicati ampiezza e frequenza perché queste proverranno direttamente dal campionamento stesso.

Gli argomenti di *soundin* sono gli stessi per *file* audio monofonici, stereofonici o quadrifonici:

variabile audio	*opcode*	nome del *file* audio	porzione iniziale del *file* da non leggere, in secondi [opzionale]	formato del *file* [opzionale]	commento
a1	**soundin**	**ifilcod**	**[, iskptim]**	**[, iformat]**	**; mono**
a1, a2	**soundin**	**ifilcod**	**[, iskptim]**	**[, iformat]**	**; stereo**
a1,..., a4	**soundin**	**ifilcod**	**[, iskptim]**	**[, iformat]**	**; quad**

Il primo argomento, l'unico obbligatorio, conterrà il nome del *file* da leggere:

```
a1      soundin "tappo.wav"          ;il nome del file va scritto fra virgolette
```

Un'orchestra e una *score* semplici possono essere:

```
;soundin.orc
        sr      = 44100
        kr      = 4410
        ksmps   = 10
        nchnls  = 1
        instr   1
a1      soundin     "voce.wav"
        out a1
        endin

;soundin.sco
i1      0   5
```

Il *file* viene cercato dapprima nella cartella corrente, cioè quella in cui stiamo lavorando, poi nella SSDIR e poi nella SFDIR (se sono state specificate) (vedi par. 1.17). Si può anche indicare il percorso se il *file* audio è in una cartella diversa.

```
a1      soundin "c:/corks/tappo.wav"        ;percorso e nome del file vanno scritti fra virgolette
```

In ***Win*** a partire dalla versione 3.50 in poi, viene usata, come divisione nell'indicazione del percorso, la barra normale o *slash* (/) invece della barra inversa o *backslash* (\). Quest'ultimo simbolo ha infatti assunto il significato di *continuazione di linea*. Nel caso di linee molto lunghe, sia nell'orchestra sia nella *score*, è possibile spezzarle nell'*editor* ma farle considerare a Csound come linea unica. Per esempio la riga:

```
f1   0   4096   10   1   2   3   4   5   6   7   8   9   10
```

può essere scritta come:

```
f1   0   4096   10   1   2  3   \
4   5   6   7   8   9   10
```

Un *file* audio può anche essere indicato mediante un numero intero; in ***Win*** il numero sarà dunque l'estensione del nome *soundin*.

```
a1      soundin     12              ; questa istruzione legge il file soundin.12
```

Consideriamo ora l'argomento *skiptime* (opzionale). Supponiamo che un nostro *file* audio contenga una voce che dice la parola "tarlo". Se vogliamo leggerlo per intero indicheremo solo il nome del *file*, senza aggiungere altro, dato che il valore di *default* per l'argomento *skiptime* è 0, che significa "leggi il *file* dall'inizio". Supponiamo però di volere leggere solo il frammento "arlo" escludendo la "t" . Se questa "t" dura .1 secondi possiamo scrivere

```
a1      soundin "tarlo.wav", .1      ; così viene letto solo il frammento "arlo"
```

Il terzo argomento (*iformat*) anch'esso opzionale non serve quasi mai, infatti qui si possono specificare le caratteristiche in caso di *file* senza *header*.

Possiamo scrivere quanti *opcode soundin* vogliamo all'interno di uno strumento o di un'orchestra, limitatamente alle capacità della versione di Csound utilizzata e delle impostazioni generali del sistema operativo. Tali *opcode* possono leggere *file* diversi o lo stesso *file* audio (magari con *skiptime* diversi).

Per esempio:

```
;soundin2.orc
        sr      = 44100
        kr      = 4410
        ksmps   = 10
        nchnls  = 1
        instr   1
a1      soundin     "luna.wav"
a2      soundin     "voce.wav"
aout    =   (a1 + a2) * 0.5             ;si moltiplica per 0.5 per attenuare l'ampiezza
                                        ;totale data dalla somma di due segnali, per evitare
                                        ;un'ampiezza di uscita eccessiva
        out aout
        endin
        instr   2
a1      soundin     "voce.wav"
a2      soundin     "voce.wav", 1
a3      soundin     "voce.wav", 2
kenv    linen   1, .1, p3, .1           ;l'inviluppo serve per eliminare eventuali
                                        ;"click" dovuti alla lettura del file quando
                                        ;il tempo di inizio lettura (skiptime)
                                        ;coincide con un punto in cui non c'e' silenzio
aout    =   (a1 + a2 + a3) * 0.33 * kenv    ;si moltiplica per 0.33 per attenuare l'ampiezza
```

```
                    ;totale data dalla somma di due segnali, per
                    ;evitare un'ampiezza di uscita eccessiva
        out aout
        endin

;soundin2.sco
i1      0   5
i2      5.5 5
```

Si noti, nello strumento 2, l'uso dell'inviluppo *kenv*: i *file* audio che leggiamo con *soundin* hanno già una loro ampiezza. Se vogliamo variarne l'ampiezza (e l'inviluppo è una forma di variazione dinamica dell'ampiezza) dovremo moltiplicare l'ampiezza di questi segnali per un certo fattore, che può essere una costante o una variabile (in questo caso *kenv*, variabile di controllo). Se la costante per la quale moltiplichiamo è pari a 1 (uno), non vi sarà variazione di ampiezza; se è minore di 1, l'ampiezza verrà ridotta; se è maggiore di 1, l'ampiezza verrà aumentata. Nel caso di *kenv,* il suo valore massimo è appunto pari a 1, e quindi l'ampiezza massima dei suoni letti da *soundin* non viene modificata: viene semplicemente applicato un inviluppo con tempi di attacco e di estinzione di 0.1 sec.

Attenzione, *soundin* non può essere reinizializzato (vedi 17.3).

ESERCIZIO 1: *Creare un filtro passabanda che utilizzi come segnale d'ingresso un* file *audio.*

ESERCIZIO 2*: Creato uno strumento che legga un* file *audio tramite* soundin*, assegnare lo* skiptime *al p4 della score e creare una score in cui per ogni nota il* file *sia letto da un punto diverso. Provare successivamente, avendo misurato la durata del* file *audio, a creare note con durata più breve di quella del* file *stesso, in modo da troncare il suono prima che esso sia terminato.*

L'*opcode diskin* è simile a *soundin,* ma consente di leggere il *file* a velocità diversa, per moto retrogrado e di realizzare *loop* (ripetizioni cicliche) semplici. La sua sintassi è la seguente:

a1[,a2[,a3,a4]] diskin ifilcod, kpitch[,iskiptim][, iwraparound] [,iformat]

ifilcod	nome del *file*: funziona esattamente come per *soundin*
kpitch	si tratta di un rapporto fra la frequenza desiderata e quella del *file* audio:
con *kpitch* = 1	il *file* sarà letto senza variazioni

con kpitch = 2 il *file* verrà letto al doppio della velocità ed una ottava sopra
con kpitch = .5 il *file* verrà letto alla metà della velocità ed una ottava sotto
se a *kpitch* viene dato un numero negativo il *file* verrà letto dalla fine all'inizio (dall'estinzione all'attacco)
con kpitch = 3 il *file* verrà letto una dodicesima sopra
con *kpitch*= -2 il *file* verrà letto al doppio della velocità ed una ottava sopra e dall'ultimo campione verso il primo

Per gli altri rapporti può essere utile consultare la tabella di corrispondenza fra semitoni e rapporti di frequenza in appendice al Cap. 8.
Si possono anche realizzare glissandi con *kpitch* variabile.

iskiptim (opzionale) funziona esattamente come per *soundin*
iwraparound (opzionale) è utile per realizzare un *loop* semplice, quando ha finito di leggere il *file*; se la nota è più lunga del *file* audio, la lettura ricomincia da capo (se *kpitch* era negativo appena arriva all'inizio del *file* ricomincia dalla fine). Questo argomento accetta solo i valori 1=acceso oppure 0=spento. In questo secondo caso non ci sarà *loop*, come nell'*opcode soundin.*
iformat (opzionale) funziona esattamente come per *soundin*

Alcuni esempi di uso di *diskin*:

```
;diskin.orc
        sr      = 44100
        kr      = 4410
        ksmps   = 10
        nchnls  = 1
        instr   1
iskip   =   0                   ;inizia a leggere il file dal principio
iwrap   =   1                   ;lettura in loop
kpitch  line 1, p3, 2           ;aumento della velocita' di lettura
                                ;con glissando dall'originale a ottava sopra
a1      diskin  "voce.wav", kpitch, iskip, iwrap
        out a1
        endin
        instr   2
iskip   =   0                   ;inizia a leggere il file dal principio
iwrap   =   1                   ;lettura in loop
```

```
kpitch   line p4, p3, p5          ;p4: velocita' di lettura a inizio nota
                                  ;p5: velocita' di lettura a fine nota
                                  ;kpitch: variabile di controllo che definisce
                                  ;la velocita' di lettura
a1       diskin  "archi.wav", kpitch, iskip, iwrap
         out a1
         endin

         instr   3
a1       diskin  "archi.wav", .5
kenv     linen   1, .01, p3, .1
         out a1 * kenv
         endin

;diskin.sco
i1       0   10.5                 ;lettura ciclica di "voce.wav" con
                                  ;aumento della velocita' di lettura
                                  ;glissando dall'originale a ottava sopra
s                                 ;fine sezione (vedi par.1.15)
;strum   act  durata   vel. vel.
;                inizio   fine
;                nota     nota
i2       0   3.11     1   1       ;lettura normale
i2       4   12.4     1   .5      ;glissando da normale a ottava sotto (3 ripetizioni)
i2       17  3.1      -1  -1      ;lettura retrograda
s
;strum   act  durata
i3       0   1.48                 ;legge i primi 0.79 sec del file a velocità dimezzata...
i3       +   .                    ;...per quattro volte
i3       +   .
i3       +   .
```

diskin, come *soundin*, non può essere reinizializzato (vedi 17.3).

Altri paragrafi in questo capitolo:

7.2 TRASFERIMENTO DEI DATI DI UN *FILE* AUDIO IN UNA TABELLA: GEN 01

7.3 LETTURA DI UN *FILE* AUDIO IMPORTATO IN UNA TABELLA: *LOSCIL*

7.4 *RELEASE* DOPO IL TERMINE DELLE NOTE E *RELEASE LOOP*

7.5 L'*OPCODE FOLLOW*

7.6 *LIMIT* E *ILIMIT*

LISTA DEGLI *OPCODE*

a1[,a2][,a3,a4]	**soundin**	**nomefile [,tempo_di_inizio_lettura] [, formato]**
a1[,a2][,a3,a4]	**diskin**	**nomefile, frequenza, [,tempo_di_inizio_lettura] [, lettura_circolare][, formato]**
a1	**loscil**	**amp, freq, tabella [, frequenza_di_base][modo_di_sustain_loop, inizio_loop, fine_loop] [modo_di_decay_loop, inizio_loop, fine_loop]**
k1	**linenr**	**ampiezza, attacco, decay, forma_decay**
a1	**linenr**	**ampiezza, attacco, decay, forma_decay**
a1	**follow**	**segnale, periodo_di_misura**
a1	**limit**	**segnale, limite_inferiore, limite_superiore**
k1	**limit**	**ksegnale, limite_inferiore, limite_superiore**
i1	**ilimit**	**segnale, limite_inferiore, limite_superiore**

8

ANALISI E RISINTESI

8.1 INTRODUZIONE

Uno dei metodi più interessanti per l'elaborazione di un suono è quello dell'analisi e risintesi.

Vediamo a grandi linee come si svolge il processo:

1) Un *sound file* viene analizzato tramite un programma per l'analisi. I risultati vengono posti in un *file* di analisi.

2) Il *file* di analisi può essere modificato in modo che alcune caratteristiche del suono non corrispondano più a quelle originali.

3) Questo *file* di analisi viene ora utilizzato come base per risintetizzare il suono, cioè in questa fase ritorniamo da un *file* di analisi ad un *sound file*. Il nuovo suono conterrà anche le eventuali modifiche che abbiamo apportato nel *file* di analisi.

Questo tipo di elaborazione consente per esempio di allungare o abbreviare la durata di un suono senza modificarne la frequenza, o di cambiare la frequenza del suono senza modificarne la durata. Le elaborazioni possono avvenire in modo dinamico (con accelerazioni, glissandi etc.). A seconda del tipo di metodo si può agire in modi diversi sul *file* di analisi per modificarlo e si possono ottenere elaborazioni del suono molto varie e interessanti.

I metodi di analisi e risintesi sono numerosi, ma noi ci concentreremo solo su questi tre, perché sono quelli possibili con Csound:

tipo di analisi	programma di analisi	tipo di *file* di analisi	metodo di risintesi	*opcode* Csound per la risintesi
phase vocoder (basata su FFT)	*pvanal*	fft (***Win***) o pv (***Mac***)	phase vocoder (basato su FFT inversa)	*pvoc*
analisi con filtro a eterodina	*hetro*	het	a banco di oscillatori o risintesi additiva (sinusoidale)	*adsyn*
LPC (Predizione Lineare)	*lpanal*	lpc	a banco di filtri	*lpread/lpreson*

Il *phase vocoder* è ormai usato in modo diffuso come metodo di analisi e risintesi anche in vari programmi commerciali per l'elaborazione del suono. In questi programmi troverete solo finestre in cui vi si chiede di fissare il valore dei vari parametri senza spiegarvi l'effetto di tali scelte. Nel prossimo paragrafo spiegheremo la teoria che vi sarà utile per il *phase vocoder* di Csound ma anche per capire meglio gli altri *phase vocoder*

Altri paragrafi in questo capitolo:.

8.2 IL *PHASE VOCODER*: L'ANALISI

8.3 IL *PHASE VOCODER*: LA RISINTESI

8.4 ANALISI A ETERODINA (*HETRO*)

8.5 LA RISINTESI CON *ADSYN*

8.6 SIMULAZIONE DEL TRATTO VOCALE: L'ANALISI CON *LPANAL*

8.7 SIMULAZIONE DEL TRATTO VOCALE: LA RISINTESI CON *LPREAD/LPRESON*

APPROFONDIMENTI

8.A.1 *FAST FOURIER TRANSFORM (FFT)*

Appendice - Tabella di corrispondenza fra semitoni e rapporti di frequenza

LISTA DEGLI *OPCODE*

a1 pvoc puntatore lettura file di analisi, fattore di moltiplicazione per trasposizione della frequenza, nome del file di analisi[, conservazione dell'inviluppo spettrale]

a1 adsyn fattore molt. ampiezza, fattore molt. frequenza, fattore molt. velocità lettura file, nome del file di analisi

krmsr, krmso, kerr, kcps lpread puntatore lettura file di analisi, nome del file di analisi

krmsr ampiezza efficace dei residui dell'analisi
krmso ampiezza efficace del segnale
kerr errore commesso nel calcolo della frequenza fondamentale
kcps frequenza fondamentale

a1 lpreson segnale di eccitazione

9

USO DI *FILE* MIDI

9.1 GLI STANDARD MIDI *FILE*

Nel 1982 i maggiori costruttori di strumenti musicali elettronici si misero d'accordo per stabilire uno *standard* di comunicazione fra strumenti, il cosiddetto *standard* MIDI (*Musical Instruments Digital Interface*, Interfaccia digitale per strumenti musicali), e nel 1993 si è giunti alla definizione di *Standard MIDI File*, o *SMF*, un formato per memorizzare sequenze MIDI, riconosciuto da tutti i programmi musicali (*sequencer*, programmi per la stampa della musica etc.). Uno *Standard MIDI File*, che in ambiente *Win* è caratterizzato dall'estensione MID, contiene tutte le informazioni necessarie per l'esecuzione.

Quando si parla di esecuzione nascono spesso perplessità in chi inizia a fare musica con il computer. Infatti eseguendo un *file* audio o un *file* MIDI si ottiene, apparentemente, lo stesso risultato: quello di produrre suono. Ma nel primo caso (*file* audio) abbiamo semplicemente convertito da digitale ad analogico un *file* che contiene la forma d'onda completa del suono, mentre nel secondo caso (*file* MIDI) abbiamo inviato alla scheda audio solo le istruzioni per l'esecuzione, come un direttore dà alla sua orchestra istruzioni, ma non produce suono. Nella maggior parte delle schede audio, oltre ai convertitori digitale/analogico e analogico/digitale, è presente una sezione in grado di produrre suoni in base a determinate istruzioni, che indicano alla scheda quali note produrre, quando e con quale timbro e dinamica; ma nel *file* MIDI *non è contenuto alcun suono.*

Esistono due tipi di *SMF*, il Tipo 0 e il Tipo 1, che differiscono fra loro perché il Tipo 0 contiene una sola traccia, mentre il Tipo 1 può contenere fino a 256 tracce.

Naturalmente la traccia dello *SMF* Tipo 0 può contenere messaggi MIDI indirizzati a tutti e 16 i canali. Uno *SMF* di Tipo 0 è costituito dai seguenti elementi:

Intestazione generale (*general header*), che contiene informazioni relative alla identificazione del *file*, alla divisione temporale, al tempo metronomico, al tempo musicale, alla chiave etc.
Intestazione di traccia (*track header*), che contiene dati relativi alla traccia.
Traccia, che contiene i messaggi MIDI eseguibili (*Note ON, Note OFF, Program change* etc.), separati da informazioni di temporizzazione.

Uno *SMF* di Tipo 1 è costituito dagli stessi elementi, ma da più tracce, quindi, per esempio:

Intestazione generale
Intestazione di traccia 1
Traccia 1
Intestazione di traccia 2
Traccia 2
Intestazione di traccia 3
Traccia 3
...

Intestazione di traccia 19
Traccia 19
etc.

Quasi tutti i programmi musicali sono in grado di leggere e salvare *file*, oltre che nel formato proprietario, anche in formato *Standard MIDI File*.

Altri paragrafi in questo capitolo:

9.2 CONTROLLARE CSOUND CON UNO STANDARD MIDI *FILE*

9.3 ASSEGNAZIONE AGLI STRUMENTI

9.4 I CONVERTITORI MIDI

9.5 CONVERSIONE DA STANDARD MIDI *FILE* A PARTITURA E VICEVERSA

APPROFONDIMENTI

9.A.1 LO STANDARD MIDI

9.A.2 LE CARATTERISTICHE DI UNO STRUMENTO MIDI

9.A.3 I NUMERI DI MIDI

9.A.4 IL PROTOCOLLO MIDI

LISTA DEGLI *OPCODE*

k1	**linen**	**ampiezza, tempo di attacco, tempo di estinzione, fattore di attenuazione della curva di estinzione**
a1	**linenr**	**ampiezza, tempo di attacco, tempo di estinzione, fattore di attenuazione della curva di estinzione**
kval	**midictrlsc**	**numero del controller [,valore massimo] [, valore minimo] [,valore iniziale]**

Vedi par. 9.4 per altri *opcode*

10

CONTROLLI MIDI E TEMPO REALE

10.1 USARE CSOUND IN TEMPO REALE

Con l'aumento della potenza di calcolo dei *personal computer,* è possibile sintetizzare suono in tempo reale. La complessità del suono che è possibile produrre dipende naturalmente, oltre che dalla complessità dell'orchestra e dal numero di voci contemporanee (polifonia), anche e soprattutto dalla potenza del calcolatore a disposizione.[1]

La possibilità di sintesi in tempo reale apre naturalmente il vasto campo dell'interazione con interpreti vocali o strumentali, o comunque con eventi che per la loro imprevedibilità non consentono una sincronizzazione con il nastro magnetico: permette di iniziare un evento sonoro in sincrono con un evento esterno, permette di adattare la velocità di esecuzione a quella di uno o più strumentisti. Ora è la macchina che segue l'uomo, e non viceversa.

Si apre anche la possibilità di elaborazione del suono in tempo reale, finora riservata solo a *hardware* specializzato o a calcolatori molto costosi.

Allo stato attuale, per il controllo in tempo reale di Csound l'unico mezzo è quello di inviare messaggi MIDI, che possono provenire da uno strumento, da una MIDI *Master Keyboard*, da uno dei cosiddetti *Mixer MIDI* (cioè da un apparecchio dotato di potenziometri che generano messaggi MIDI di tipo *controller*), da un secondo

[1] Per potenza, in questo particolare caso, si intende la velocità delle operazioni in *floating point*, su cui si basa principalmente l'elaborazione in Csound.

calcolatore (o dallo stesso) che, contemporaneamente a Csound, esegua un programma che invia messaggi MIDI. Il limite è solo la fantasia del musicista. È necessario però, per sfruttare adeguatamente tutta la potenza di calcolo, una certa cura nella scrittura delle orchestre, evitando istruzioni inutili (specialmente quelle che coinvolgono variabili di tipo audio) e abbassando, per quanto è possibile, la frequenza dei segnali di controllo (*kr*).

Per esempio, questa riga di orchestra:

```
a2      =       a1/2
```

può essere più efficientemente riscritta come:

```
a2      =       a1*.5
```

dal momento che una moltiplicazione viene eseguita più velocemente di una divisione. Oppure il frammento:

```
aout    =    a1*kvol/4+a2*kvol/4+a3*kvol/4
```

sarà riscritto come:

```
k1      =       kvol*.25
aout    =       (a1+a2+a3)*k1
```

sostituendo a tre divisioni e tre moltiplicazioni una assegnazione e due moltiplicazioni.

Per usare Csound in tempo reale è necessario sostituire al nome del *file* audio di uscita i nomi riservati *devaudio* oppure *dac* (a seconda della piattaforma impiegata e della versione di Csound utilizzata), quindi:

csound -W -omiofile.wav miaorc.orc miascore.sco

genera il *file* audio *miofile.wav*, mentre

csound -odevaudio miaorc.orc miascore.sco

genera il suono in tempo reale.

***Win* Come attivare il tempo reale**

Nella finestra di dialogo della sintesi, nel riquadro di testo *Wave* compare il nome del *file* da generare. Accanto c'è il pulsante *Realtime Out*: facendo clic su di esso, il nome del *file* audio cambia in *devaudio*, e la scritta sul pulsante cambia in *Audio File*. Così facendo nuovamente clic sul pulsante si torna alla generazione di un *file* audio e così via. Per modificare i valori del *buffer* per il tempo reale, bisogna agire sul *flag -b*, cambiandone opportunamente il valore (vedi par. 10.3)

***Mac* Come attivare il tempo reale**

Se viene attivata la casella *Audio out*, il programma invia i campioni generati direttamente al *Sound Manager* per suonarli invece che scriverli in un *file*. Per modificare i valori del *buffer* per il tempo reale potete aprire *Set Buffer* nel menu *Preferences*. Vedi anche il par. 10.3.

Altri paragrafi in questo capitolo:

10.2 PARTITURA E ORCHESTRA PER IL TEMPO REALE

10.3 QUALCHE ACCORGIMENTO PER USARE CSOUND IN TEMPO REALE

11

MODULAZIONE D'AMPIEZZA E MODULAZIONE AD ANELLO

11.1 INTRODUZIONE ALLA MODULAZIONE D'AMPIEZZA E AD ANELLO

"Una modulazione è l'alterazione dell'ampiezza, della frequenza o della fase di un oscillatore provocata da un altro segnale".[1] L'oscillatore che viene modulato viene detto portante (*carrier*), l'oscillatore che modula viene detto modulante (*modulator*).

Ricordate il modo in cui abbiamo prodotto il tremolo?

In quel caso abbiamo realizzato una leggera **modulazione dell'ampiezza** (tremolo) e ci siamo serviti di un segnale modulante (variabile di controllo) che faceva variare l'ampiezza del segnale portante. La caratteristica dell'oscillazione del segnale modulante era di avere una frequenza molto bassa, al di sotto della banda audio, e un'ampiezza minima: perciò l'unico effetto che l'oscillatore modulante provocava era quello di variare di poco l'ampiezza del segnale portante.

Ma che cosa succede se la frequenza dell'oscillatore modulante è più alta?

In questo caso abbiamo un fenomeno diverso, cioè la comparsa di frequenze nuove che si aggiungono allo spettro della portante. Tali frequenze vengono dette laterali, poiché appaiono in modo simmetrico sopra e sotto la frequenza della portante, come vedremo. Da qui nascono sia la tecnica della modulazione d'ampiezza (AM = *Amplitude*

[1] Charles Dodge e Thomas A. Jerse, *Computer Music*, Schirmer, New York, NY, 1985 p.80.

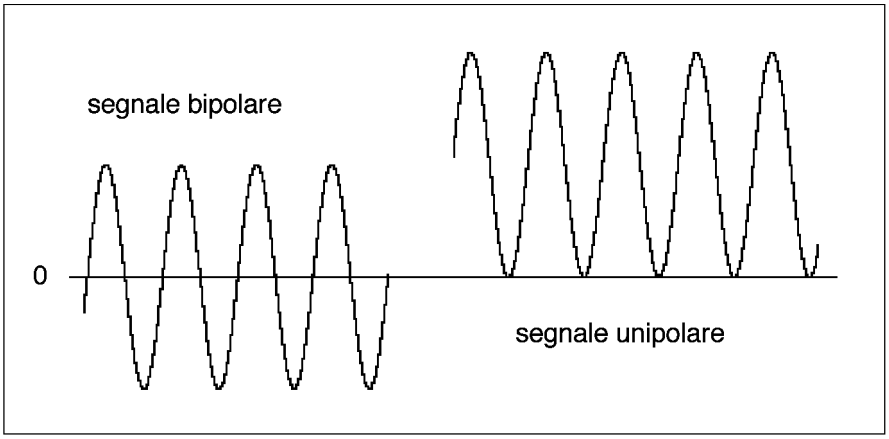


Fig. 11-1

Modulation), che quella della modulazione ad anello (RM = *Ring Modulation*). La differenza fra questi due tipi di modulazione sta nel fatto che mentre la modulazione d'ampiezza utilizza un segnale modulante unipolare, la modulazione ad anello utilizza un segnale modulante bipolare (vedi Fig.11-1).

I segnali **bipolari** sono quelli che oscillano fra valori positivi e valori negativi d'ampiezza, per esempio:

```
a1 oscili 10000, 220, 1          ;ampiezza oscillante fra 10000 e -10000
```

I segnali **unipolari** sono quelli che oscillano solo nel campo positivo (o solo nel campo negativo). Per crearli c'è bisogno di aggiungere una costante, cioè un segnale che non varia, a un'oscillazione bipolare. Tale costante viene chiamata *DC Offset* (*Direct Current Offset*, o componente di corrente continua, vedi par.2.7).

Facciamo un esempio di segnale unipolare nel campo positivo, utile per la modulazione d'ampiezza:

```
abimod     oscili 1 , 220, 1     ;ampiezza oscillante fra 1 e - 1
aunimod  = abimod+1              ;aunimod oscillerà fra 0 e 2, cioè solo nel campo positivo
```

Abbiamo dunque aggiunto una componente continua (*DC Offset*) di 1 alle oscillazioni di *abimod*, spostandole tutte nel campo positivo.

Nel Cap.5 abbiamo usato sia i segnali modulanti bipolari sia quelli unipolari, ma in quei casi tali segnali non erano nella banda audio.

11.2 MODULAZIONE D'AMPIEZZA (AM)

11.3 MODULAZIONE AD ANELLO

APPROFONDIMENTI

11.A.1 LE FORMULE DELLA MODULAZIONE DI AMPIEZZA E DELLA MODULAZIONE AD ANELLO

11.B.1 CENNI STORICI SULLA MODULAZIONE AD ANELLO

12

MODULAZIONE DI FREQUENZA (FM)

12.1 TEORIA DI BASE

Come la sintesi in AM, anche quella in FM, nella sua forma semplice, si basa su un oscillatore modulante (*modulator*) e uno portante (*carrier*). In questo caso però il segnale modulante modula la frequenza e non l'ampiezza dell'oscillatore portante. Uno degli schemi più semplici per la modulazione di frequenza è quello illustrato in Fig.12-1, analogo a quello che abbiamo usato per il vibrato nel par. 5.3.

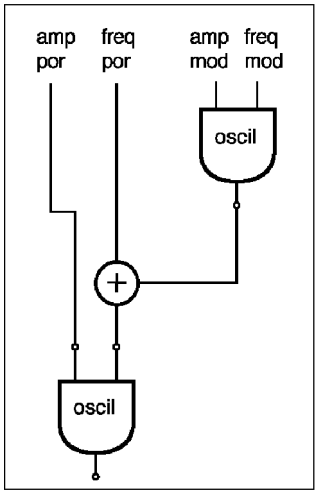


Fig. 12-1

Abbiamo dunque due oscillatori sinusoidali, uno chiamato *portante* e l'altro *modulante*. Se l'ampiezza del segnale modulante è 0, non abbiamo alcuna modulazione e ciò che rimane è solo l'oscillazione sinusoidale dell'onda portante. Se aumentiamo l'ampiezza del segnale modulante, ha luogo la modulazione di frequenza e il segnale modulante devia verso l'acuto e verso il grave la frequenza dell'oscillatore portante. Più aumentiamo l'ampiezza del segnale modulante, maggiore è la deviazione di frequenza del segnale portante. Nel momento in cui l'uscita dell'oscillatore modulante è positiva, la frequenza dell'oscillatore portante è più alta della sua frequenza di base. Nel momento in cui invece l'uscita dall'oscillatore modulante è negativa, la frequenza dell'oscillatore è più bassa della sua frequenza di base. Il massimo mutamento di frequenza che l'oscillatore portante subisce viene chiamato deviazione (*deviation*), oppure deviazione di picco (*peak frequency deviation*), ed è espresso in Hertz.

Mentre con l'AM (nel caso che portante e modulante siano due sinusoidi) otteniamo una sola coppia di bande laterali, nella FM abbiamo una serie (teoricamente infinita) di coppie di bande laterali. Il numero delle bande laterali udibili dipende dalla deviazione: più ampia è la deviazione, maggiore è il numero di bande laterali che hanno ampiezza sufficiente per poter essere udite.

Se C=1000 Hz, e M=3 Hz, otterremo dunque una serie di frequenze somma e di frequenze differenza (bande laterali):

somma	differenza
1003 (C+M)	997 (C-M)
1006 (C+2*M)	994 (C-2*M)
1009 (C+3*M)	991 (C-3*M)
1012 (C+4*M)	988 (C-4*M)
1015 (C+5*M)	985 (C-5*M)
.....	.....

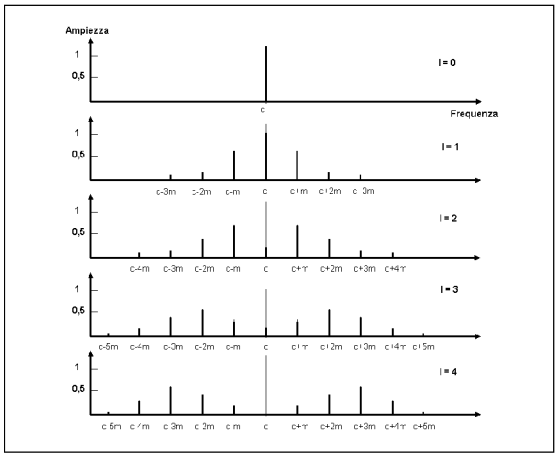
Teoricamente sono *sempre* presenti tutte le frequenze somma e le frequenze differenza (C+M, C-M, C+2M, C-2M, etc....) fino all'infinito, ma per essere percepite tali componenti laterali devono avere un'ampiezza sufficiente. Il numero delle bande laterali udibili dipende dall'indice di modulazione. L'indice di modulazione (**I**) è uguale alla deviazione di picco (**D**) diviso la frequenza della modulante (**M**). Ad esempio

se D = 100 Hz e M = 100 Hz allora I = 1
se D = 200 Hz e M = 100 Hz allora I = 2

In pratica, per indici di modulazione e frequenze modulanti basse, le frequenze laterali (somma e differenza) di ordine elevato non sono udibili perché non hanno una ampiezza

sufficiente. Infatti la deviazione (**D**) è uguale all'indice di modulazione (**I**) per la frequenza della modulante:

D = I * M



In realtà più aumenta l'indice di modulazione, più aumenta il numero delle bande udibili, ma, come notiamo nella figura, man mano che aumenta il valore dell'indice, l'energia viene "rubata" alla portante e alle frequenze laterali vicine ad essa e distribuita alle bande laterali più estreme. Per chi usa la modulazione di frequenza e vuole controllare questo meccanismo la faccenda si fa complicata: come si può sapere la larghezza di banda del suono ottenuto tramite FM? Ci vengono in aiuto due regole approssimative ma semplici:

a) il numero delle coppie che hanno più di un centesimo dell'ampiezza della portante è approssimativamente I+1 (De Poli, 1983). In pratica, se la portante ha ampiezza 0 dB, l'ampiezza di queste coppie sarà compresa fra 0 e –40 dB.

b) la larghezza di banda totale è approssimativamente uguale a due volte la somma della frequenza di deviazione più la frequenza della modulante (Chowning, 1973)

Larghezza_di_banda = ~ 2 * (D+M)

Una delle ricchezze di questa tecnica di sintesi è dovuta al fatto che si possono produrre componenti laterali che cadono nella parte negativa della banda audio. Tali componenti negative vengono "riflesse" intorno alla frequenza 0 Hz e si miscelano con le componenti appartenenti alla parte positiva, come spiegheremo poco più avanti. Consideriamo la seguente partitura (che potremo, volendo, sperimentare con l'orchestra *fm.orc* descritta più oltre):

```
;fm.sco
f1    0     4096    10     1
;       act       dur       amp. por     frq. por     frq. mod     indice di modulazione
i1      0         2.9       10000        1000         3            10
i1      3         .         10000        1000         3            30
i1      6.        .         10000        1000         3            50
i1      9.        .         10000        1000         3            1000
;
i1      12        2.9       10000        1000         100          10
i1      15        .         10000        1000         100          30
i1      18        .         10000        1000         100          50
i1      21        .         10000        1000         100          1000
```

Nelle prime tre note dell'esempio si percepisce un glissando ascendente/discendente, perché le frequenze laterali udibili sono poche, molto vicine a 1000, quindi comprese nella stessa banda critica[1], e non influenzano la percezione timbrica (percepiamo infatti una sinusoide glissata).

Nella quarta nota la situazione si fa diversa: dato che, a parità di frequenza della modulante, più è alto l'indice di modulazione e maggiore sarà la deviazione, e dunque maggiore sarà l'ampiezza delle componenti laterali estreme, in questo caso percepiremo qualcosa di diverso dalla sinusoide che glissa.

Calcoliamo la deviazione di questa ultima nota ricordando che

D = I * M:

D = 3*1000 = 3000 Hz.

La gamma di frequenza delle bande laterali più significative sarà compresa fra la frequenza della portante meno la deviazione (C-D=1000-3000) e la frequenza della portante più la deviazione (C+D=1000+3000), quindi fra -2000 Hz e 4000 Hz. Qui le cose sembrano complicarsi, perché abbiamo frequenze negative: cosa succede quando si hanno frequenze negative?

Tutte le frequenze sotto lo zero ricompaiono in controfase e a specchio nel campo positivo (-2000 diventa 2000 in controfase) e quindi il suono diventa più complesso ogniqualvolta la sua gamma di frequenza o anche solo le sue componenti laterali udibili si riflettono nel campo audio e dunque si sommano algebricamente con le altre. In Fig.12-2 è mostrato il caso di una portante di 2000 Hz e una modulante di 3000 Hz: Nello spettro risultante le frequenze negative -1000 Hz e -4000 Hz si riflettono nel campo positivo e in controfase, producendo così una diminuzione di ampiezza delle componenti di 1000 Hz e 4000 Hz già presenti.

[1] In un suono complesso la banda critica corrisponde alla più piccola differenza di frequenza fra due componenti che consenta la loro percezione come suoni separati, piuttosto che come un solo suono.

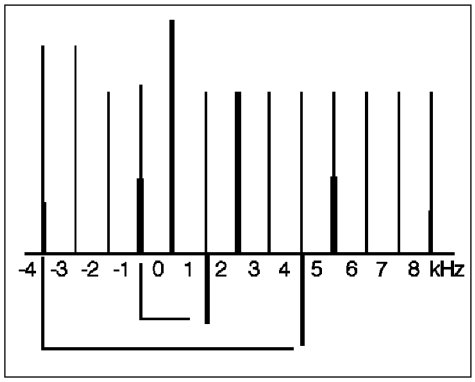


Fig. 12-2

Lo stesso effetto di riflessione nel campo udibile ha luogo nel caso che la gamma di frequenza o le componenti laterali superino la frequenza di Nyquist (*sr/2*). Per esempio, se abbiamo una frequenza di campionamento di 22050 Hz con frequenza di Nyquist pari a 11025 Hz e abbiamo una portante di 5000 Hz con frequenza modulante di 5000 Hz e indice di modulazione 3, la frequenza delle bande laterali più significative varierà fra -10000 e +20000 Hz, quindi avremo la riflessione sia delle frequenze negative, sia di quelle superiori alla frequenza di Nyquist (11025 Hz in questo caso). In Fig.12-3 è mostrato questo caso: la componente a 15000 Hz (linea tratteggiata) viene "riflessa" per effetto del *foldover*, e diventa di 15000 - 11025 = 3975 Hz.

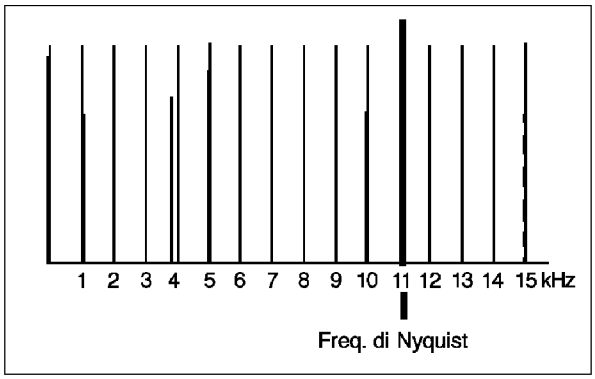


Fig. 12-3

Altri paragrafi in questo capitolo:

12.2 ORCHESTRE PER LA FM SEMPLICE

12.3 FAMIGLIE SPETTRALI

12.4 FM CON PORTANTI MULTIPLE

12.5 FM CON MODULANTI MULTIPLE

APPROFONDIMENTI

12.A.1 LE FORMULE DELLA FM

12.A.2 SIMULAZIONE DI SUONI STRUMENTALI

12.B.1 CENNI STORICI

LISTA DEGLI *OPCODE*

ar foscil ampiezza, frequenza_di_base, portante, modulante, indice, tabella[, fase]
ar foscili ampiezza, frequenza_di_base, portante, modulante, indice, tabella[, fase]

13

VARIABILI GLOBALI, ECO, RIVERBERO, *CHORUS, FLANGER, PHASER*, CONVOLUZIONE

13.1 ECO E RIVERBERO

Eco e riverbero sono due dei cosiddetti “effetti” che si usano nella sintesi e nel trattamento del suono. L’eco simula l’effetto del suono che viene riflesso da un ostacolo, ed è percepibile se la distanza fra la sorgente sonora e l’ostacolo riflettente è tale che il segnale di eco giunge all’ascoltatore con un ritardo di almeno 1/20 di secondo (Fig. 13-1 in alto). Se gli ostacoli sono più di uno, come è il caso di una stanza a forma di parallelepipedo, vi possono essere il fenomeno di eco multipla (Fig. 13-1 in basso), o quello di riverbero, in cui le eco multiple si fondono insieme (Fig. 13-2).

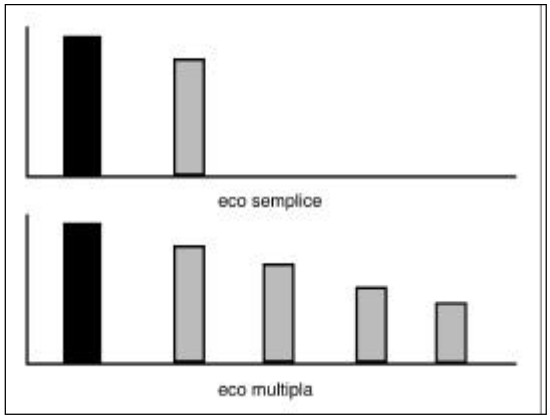


Fig. 13-1

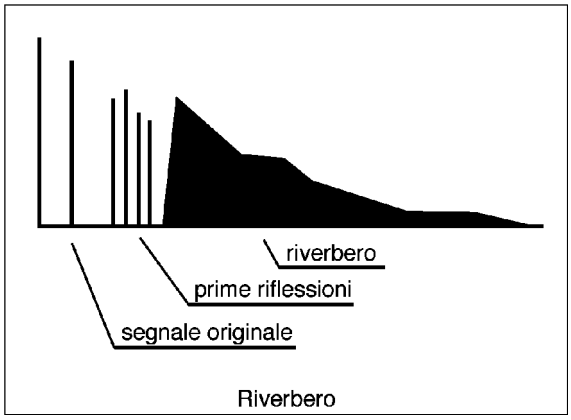


Fig. 13-2

Qualche millisecondo dopo il segnale originale si odono le prime riflessioni (*early reflections*), poi le ulteriori eco, che possono essere migliaia, si fondono insieme, e l'ampiezza complessiva di queste ultime decade in un determinato tempo, fino a scomparire. Ma perché in un ambiente si ha il fenomeno del riverbero? Vediamo in Fig.13-3 la rappresentazione in pianta di un ambiente rettangolare. All'ascoltatore arriva prima il suono diretto emesso dalla sorgente sonora, poi via via tutti i suoni riflessi, che, dovendo compiere un percorso più lungo del suono diretto, giungono più tardi. In particolare, arriveranno prima i suoni riflessi solo una volta, poi i suoni riflessi due volte, tre volte e così via. Naturalmente più volte un suono viene riflesso, e più è debole, perché a ogni riflessione cede una parte di energia alla parete.

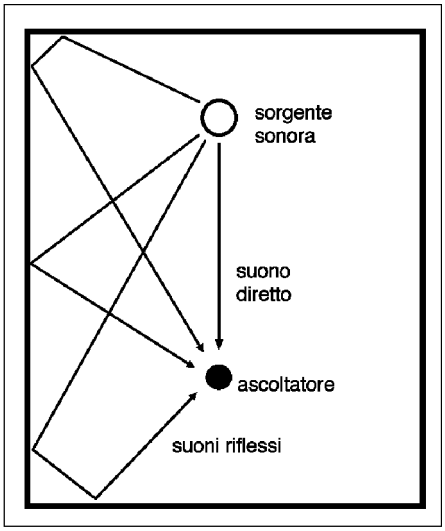


Fig. 13-3

Si definisce *tempo di riverberazione* il tempo che il segnale riverberato impiega per decrescere di 60 dB. Il tempo di riverberazione è uno dei parametri che definisce le caratteristiche acustiche di un ambiente. In realtà, poiché il fattore di assorbimento dei materiali non è costante con la frequenza, per studiare a fondo le caratteristiche di riverberazione di un ambiente bisogna effettuare le misure a varie frequenze.

Ma quali sono le influenze del tempo di riverberazione sull'ascolto? Il suo valore ottimale varia a seconda del genere di suono e del volume dell'ambiente. Più questo è grande, maggiore sarà il tempo di riverberazione ottimale, per il quale è comunque possibile dare alcuni valori indicativi: per il parlato, da 0.5 a 1 secondo; per la musica da camera, circa 1.5 secondi; per la musica sinfonica da 2 a 4 secondi; per la musica organistica 5 secondi e più.

Altri paragrafi in questo capitolo:

APPROFONDIMENTI

13.A.1 COSTRUZIONE DI RIVERBERI

LISTA DEGLI *OPCODE*

ar	**delayr**	**tempo di ritardo [, disposizione iniziale della memoria interna] [RB1]**
	delayw	**segnale di ingresso**
a1	**deltap**	**tempo di ritardo**
a1	**deltapi**	**tempo di ritardo**
ar	**delay**	**segnale di ingresso, tempo di ritardo [, disposizione iniziale della memoria interna] [RB2]**
ar	**delay1**	**segnale di ingresso [,disposizione iniziale della memoria interna]**
ar	**reverb**	**segnale di ingresso, tempo di riverberazione [,disposizione iniziale della memoria interna]**
ar	**nreverb**	**segnale di ingresso, tempo di riverberazione, riverberazione delle alte frequenze [,disposizione iniziale della memoria interna] [, numero dei filtri *comb*, numero**

		della tabella contenente i dati dei filtri *comb*] [, numero dei filtri *alpass*, numero della tabella contenente i dati dei filtri *alpass*]
ar	**vdelay**	**segnale di ingresso, tempo di ritardo, tempo massimo di ritardo [,disposizione iniziale della memoria interna]**
ar1[,...[,ar4]]]	**convolve**	**segnale di ingresso, nome del file, canale**
ar	**dconv**	**segnale di ingresso, dimensione del *buffer*, numero della tabella**
ar	**comb**	**segnale di ingresso, tempo di riverberazione, tempo di *loop* [,disposizione iniziale della memoria interna]**
ar	**alpass**	**segnale di ingresso, tempo di riverberazione, tempo di *loop* [,disposizione iniziale della memoria interna]**

[RB3]

[RB4]

14

SINTESI PER DISTORSIONE NON LINEARE (DNL), COMPRESSORI E SINTESI VETTORIALE

La maggior parte dei tipi di sintesi e trattamento del suono che vedremo in questo capitolo fa uso dell'*opcode table*. Prima di passare alle applicazioni, è quindi necessario approfondire le conoscenze su questo *opcode* e sulla generazione di tabelle con metodi diversi da quelli finora visti.

14.1 GEN02 E CONSIDERAZIONI AGGIUNTIVE SULLE FUNZIONI

La GEN02 copia semplicemente nei punti di una tabella i valori dei campi che scriviamo dopo i primi quattro parametri, cioè dopo il numero di funzione, il *creation time*, le dimensioni e il numero della GEN.

fn t s 2 v1 v2 v3 ...

n	numero della tabella
t	tempo al quale la tabella deve essere generata
s	dimensioni
2	numero della GEN; se è positivo, la tabella verrà riscalata in modo che il suo valore massimo sia 1 (uno); se negativo (-2) non viene effettuato riscalamento
v1, v2, ...	valori da inserire in tabella

Facciamo un esempio:

```
f1 0 16  2  0 1 2 3 4 5 6 7 8 9 10 9 8 7 6 5
;riscala  con valori fra 0 e 1

f1 0 16 -2  0 1 2 3 4 5 6 7 8 9 10 9 8 7 6 5
;non riscala (val. fra 0 e 10)
```

Entrambe le tabelle hanno una dimensione di 16 punti, quindi abbiamo 16 indici che identificano locazioni da riempire con altrettanti valori

Indici	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valori tabella	0	1	2	3	4	5	6	7	8	9	10	9	8	7	6	5

Nel caso della prima funzione abbiamo scritto 2 come numero di GEN, perciò tutti i valori vengono riscalati fra 0 e 1 come segue:

Indici	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Valori tabella	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	1	.9	.8	.7	.6	.5

Nel secondo caso i valori rimarranno quelli da noi specificati, perché quando si indica un numero di GEN negativo Csound non riscala i valori, cioè li mantiene come sono.

Un altro esempio sul riscalamento con la GEN 10 può essere:

```
f1 0 4096 10 2        ;riscala fra -1 e 1
f1 0 4096 -10 2       ;NON riscala (val. fra -2 e 2)
```

Sarà bene riassumere meglio il significato di alcuni termini riguardanti le funzioni, prima di passare al prossimo paragrafo.

a. Funzioni

Una **funzione** è un metodo matematico di generazione di valori di una variabile (dipendente) a seconda dei valori assunti da un'altra variabile (indipendente). Per esempio:

y = f(x)

in cui:

x = variabile indipendente (assume valori a piacere)
y = variabile dipendente (assume valori dipendenti da x)
(altri esempi: y = 2*x, y = sin(x), y = 4*x2 + 3 ...)

b. GEN

Una **GEN** è un **numero di metodo** di generazione di una tabella di valori, cioè contiene l'implementazione di una funzione.

c. Tabelle

Una **tabella** è un insieme monodimensionale di valori (vettore) accessibile con un **indice**. Per esempio:

Indice	0	1	2	3	4	5	6	7	8
Tabella	1	5	7	3	7	9	56	3	12

quindi all'**indice** 4 corrisponde il **valore** 7.

In Csound le tabelle vengono organizzate secondo un criterio particolare: esse occupano spazi *numerati* da 1 a un numero dipendente dalla particolare versione di Csound utilizzata; ciascuno spazio può contenere, in tempi diversi, tabelle diverse, cioè una tabella può sostituirne un'altra nello stesso spazio.

Per esempio:

```
f 1 0  4096 10 1                ;questa tabella 'vive' da 0 a 10 sec, perché...
f 1 10 4096 10 1 .5 .4 .3 .2 .1 ;...quest'altra tabella SOSTITUISCE la precedente al 10° secondo,
                                ;e quindi "nasce" a 10 sec, e "vive" fino a che non viene a sua volta sostituita, o fino
                                ;alla fine della partitura, o fino a una istruzione che richiede la "distruzione" della
                                ;tabella, p.es.:
f -1 20                         ;in questo modo la tabella precedente cessa di esistere a 20 secondi
```

Altri paragrafi in questo capitolo:

14.2 USO DELL'*OPCODE TABLE*

14.3 TABELLE COSTITUITE DA SPEZZATE DI RETTA, ESPONENZIALI E *SPLINE* CUBICI: GEN05, GEN07, GEN08

14.4 SINTESI PER DISTORSIONE NON LINEARE (*WAVESHAPING*)

14.5 USO DEI POLINOMI DI CHEBISHEV (GEN13)

14.6 *TABLE*: APPLICAZIONI PER COMPRESSORI ED ESPANSORI DI DINAMICA

14.7 GEN03

14.8 DISSOLVENZA INCROCIATA DI TABELLE: LA SINTESI VETTORIALE

LISTA DEGLI *OPCODE*

i1 table indice, funzione [, modo][, offset][, wrap]
k1 table indice, funzione [, modo][, offset][, wrap]
a1 table indice, funzione [, modo][, offset][, wrap]
i1 tablei indice, funzione [, modo][, offset][, wrap]
k1 tablei indice, funzione [, modo][, offset][, wrap]

15

SINTESI GRANULARE E SINTESI PER FORMANTI

15.1 CHE COSA È LA SINTESI GRANULARE

Il suono, come sappiamo, è un fenomeno dinamico, in continua evoluzione. Ma, in certi casi, lo si può suddividere in piccoli periodi nei quali i suoi parametri non variano, un po' come accade nel cinema o nella televisione, dove il susseguirsi di immagini statiche, purché di durata abbastanza piccola, danno l'impressione del movimento. La sintesi granulare attua qualcosa di analogo per quanto riguarda il suono: è cioè un metodo di sintesi nel quale il suono viene considerato come un susseguirsi di piccoli "quadri" statici.

La sintesi granulare nasce dalla possibilità di generare un'alta densità di piccoli eventi acustici, detti appunto grani, la cui durata in genere varia da 10 a 100 millisecondi. La caratteristica di un grano di suono è quella di avere un inviluppo generalmente simmetrico, che può variare da una forma a campana di tipo Gaussiano ad un inviluppo costituito da un attacco, un *sustain* e un *decay* o anche solo un attacco e un *decay* (vedi Figg.15-1 e 15-2). I grani possono essere intervallati da pause, e allora verranno percepiti come eventi singoli; oppure possono essere accostati o addirittura sovrapposti, e verranno uditi come un flusso di suono ininterrotto, all'interno del quale potremo percepire una "granularità", cioè una discontinuità, una continua variabilità. Se stabiliamo un parallelo con la visione, il suono al suo interno non è "liscio", ma piuttosto "rugoso".

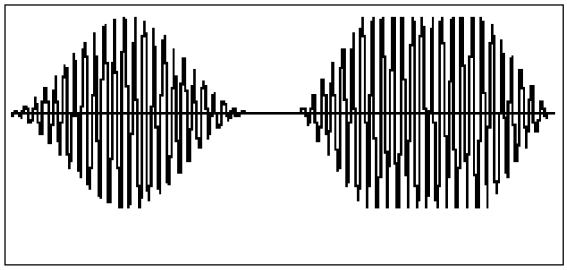


Fig. 15-1

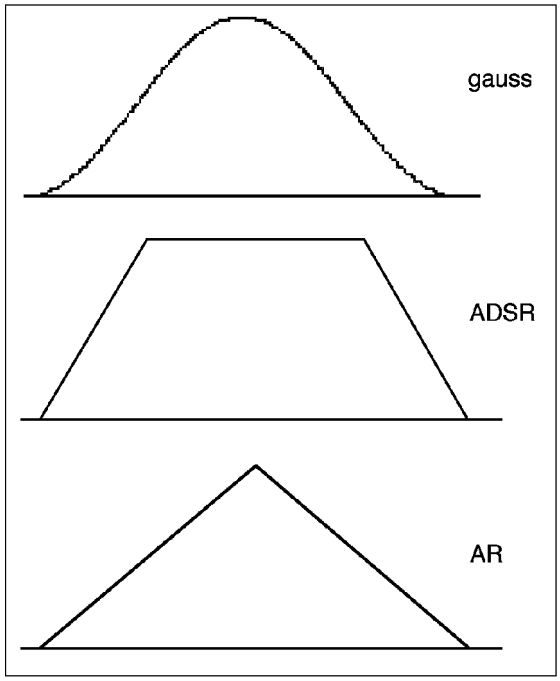


Fig. 15-2

Ascoltiamo l'esempio sonoro generato con la seguente coppia orchestra/*score* senza analizzare, per il momento, come essa sia costruita, in quanto gli *opcode timout*, *reinit* e *rireturn* verranno trattati nel par. 17.3. Basterà dire che la forma d'onda contenuta nei ...

Altri paragrafi in questo capitolo:

15.2 L'*OPCODE GRAIN*

15.3 L'*OPCODE GRANULE*

15.4 SINTESI PER FORMANTI: FOF

15.5 *STRETCHING* DEL SUONO: *SNDWARP*

APPROFONDIMENTI

15.A.1 SINTESI GRANULARE: CENNI STORICI

LISTA DEGLI *OPCODE*

ar **grain ampiezza, frequenza, densità, offset_di_ampiezza, offset_di_frequenza, funzione_di_forma_d'onda, funzione_di_inviluppo, durata_massima_grani**

ar **granule ampiezza, num_voci, velocità_di_lettura, modo, soglia_ampiezza, tabella, trasposizione, porzione_di_file_da_saltare, variazione_random_da_saltare, durata_da_usare, pausa_fra_grani, variazione_random_di_pau-sa, durata_grano, variazione_random_durata_grano, attacco, decay [,seme_random] [,altezza1] [,altezza 2] [,altezza 3] [,altezza 4] [,tabella_di_inviluppo]**

ar **fof ampiezza, fondamentale, formante, ottavizzazione, larghezza_di_banda_formante, attacco, durata, decay, sovrapposizioni, funzione_seno, tabella_di_inviluppo, durata[, fase_iniziale] [, modo_di_frequenza]**

asig[, acmp] sndwarp ampiezza, modifica_di_durata, modifica_di_frequenza, tabella_suono, inizio_tabella, dimensione_finestra, variazione_random_finestra, sovrapposizione_finestre, tabella_forma_finestra, itime_mode

16

LA SINTESI PER MODELLI FISICI

16.1 INTRODUZIONE

La sintesi per modelli fisici (*physical modeling*) è un potente mezzo per la produzione (o riproduzione) di suoni che possiedono un alto grado di similitudine con suoni reali, e, più in generale, di suoni caratterizzati da un elevato “realismo” anche quando non imitano suoni già esistenti.

A differenza di altri tipi di sintesi, che hanno come obiettivo quello di riprodurre le caratteristiche del suono, la sintesi per modelli fisici prende come punto di partenza le caratteristiche fisiche dello strumento che genera un dato suono, e ne costruisce un modello matematico.

16.2 L’ALGORITMO DI KARPLUS E STRONG

Uno dei primi algoritmi di sintesi per modelli fisici proposti è la simulazione di corda pizzicata dovuta ad Alex Karplus e Kevin Strong. Anche se, a stretto rigore, non rientra nella sintesi per modelli fisici, la si può considerare un antenato di questa.

L’idea di base fu di partire da un rumore e di operare una sorta di filtraggio ricorsivo di questo rumore fino a ottenere un suono a spettro molto più semplice, al limite una sinusoide. Poiché Karplus e Strong cercavano di mettere a punto un algoritmo che potesse essere facilmente implementato in *hardware* (in particolare sui *microcomputer* degli inizi degli anni Ottanta), dovevano limitare al massimo il numero delle operazioni.

Pensarono quindi di iniziare da una tabella, riempita con valori casuali, e di leggerla, facendo, a ogni lettura, la media fra il valore corrente e quello precedente.

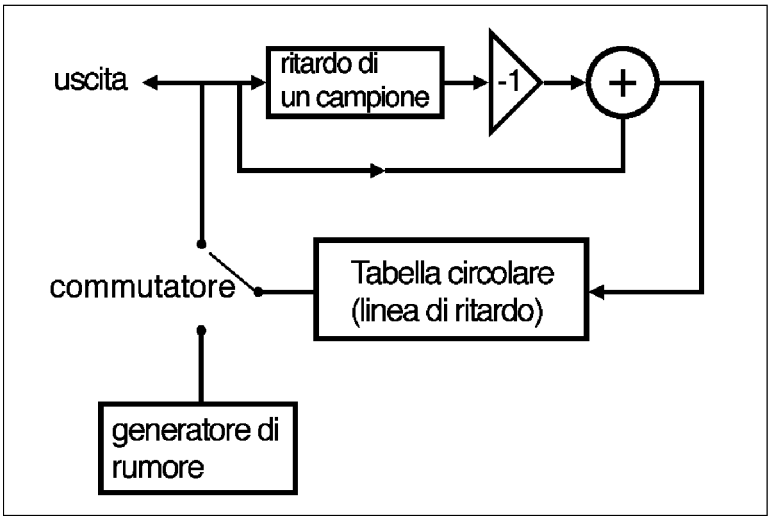


Fig. 16-1

Con riferimento alla Fig.16-1, vi è un generatore di rumore, attivo solo in fase di inizializzazione della nota, che, prima che il suono abbia inizio, riempie la tabella circolare (implementata con una semplice linea di ritardo) con una serie di valori casuali. Quindi il commutatore rappresentato in figura collega la tabella con il generatore di rumore. Appena la tabella è stata riempita, può iniziare il suono. Il commutatore scatta, e il segnale inizia a circolare. Viene prelevato dalla tabella il primo valore, il quale è poi inviato all'uscita. Ma viene anche ritardato di un campione, invertito di segno, e sommato al campione successivo, facendo così una media, il cui valore viene scritto nella tabella. Dopo un certo tempo il segnale si attenua, perdendo armoniche alte, fino a scomparire.

Più in dettaglio, il calcolo di un campione di suono avviene secondo la formula:...

Altri paragrafi in questo capitolo:

16.3 CORDA PIZZICATA

16.4 PIASTRA PERCOSSA

16.5 TUBO CON ANCIA SINGOLA

APPROFONDIMENTI

17

CSOUND COME LINGUAGGIO DI PROGRAMMAZIONE

17.1 CSOUND È UN LINGUAGGIO DI PROGRAMMAZIONE

Csound è, a tutti gli effetti, un linguaggio di programmazione. Certo, è particolarmente orientato alla sintesi del suono (è stato concepito per questo), ma in linea di principio nulla vieta di utilizzarlo, per esempio, per eseguire calcoli. Avviamo Csound con questa orchestra:

```
;calcola.orc
        sr      =   100
        kr      =   1
        ksmps   =   100
        nchnls  =   1

        instr       1
        i1      =   log(10) ;calcola il logaritmo naturale di 10 e pone il risultato in i1
        print       i1      ;l'opcode print visualizza valori, in questo caso visualizza  i1
                            ;(riguardo l'opcode print vedi par. 17.5)
        endin
```

e questa partitura:

```
;calcola.sco
i1      0   .01
```

Csound visualizzerà sullo schermo il valore 2.303, il logaritmo naturale di 10. Notiamo alcune anomalie: la frequenza di campionamento *sr* è di soli 100 Hz: ma sono anche troppi, dal momento che non ci interessa sintetizzare alcun *file* audio. La frequenza di controllo è addirittura di 1 Hz: ma nell'orchestra non c'è traccia di segnali di controllo, quindi anche questo valore (come qualsiasi altro) va benissimo. La durata della nota in partitura è di 1/100 di secondo soltanto, ma poiché non ci interessa il suono, va bene così.

Come tutti i linguaggi di programmazione, Csound ha campi in cui è molto efficace (la sintesi) e campi in cui è debole (la gestione dei dati, la strutturazione etc.). Ci interessa però sottolineare che Csound permette anche di scrivere programmi (cioè orchestre) dotate di una certa "intelligenza", orchestre in grado di reagire differentemente a seconda dei dati che ricevono. Avremmo anche potuto eseguire Csound con il *flag -n (nosound)* per non generare alcun *file* audio.

17.2 MODIFICA DEL FLUSSO DI PROGRAMMA E OPERATORI DI CONVERSIONE

Normalmente le istruzioni di un'orchestra Csound vengono eseguite sequenzialmente, dalla prima all'ultima. È possibile però modificare il flusso del programma, cioè l'ordine in cui le istruzioni vengono eseguite. Inoltre è possibile eseguire un particolare frammento di codice o un altro a seconda del verificarsi di determinate condizioni.

Le più importanti istruzioni che permettono di modificare il flusso di programma sono le *istruzioni di salto*:

igoto etichetta
kgoto etichetta
goto etichetta

igoto fa saltare[1] l'esecuzione del programma alla riga contrassegnata da *etichetta*, ma solo durante il passo di inizializzazione (vedi par.1.A.1).

kgoto fa saltare l'esecuzione del programma alla riga contrassegnata da *etichetta*, ma solo durante il passo di calcolo dei segnali di controllo.

goto fa saltare l'esecuzione del programma alla riga contrassegnata da *etichetta*, sia durante il passo di inizializzazione sia durante il passo di calcolo dei segnali di controllo.

[1] Si definisce *salto* l'azione per la quale non viene eseguita la riga successiva di un programma, ma una riga diversa indicata nell'istruzione di salto, e l'esecuzione continua da quella riga in poi.

Un'etichetta *(label)* è un identificatore alfanumerico (cioè una stringa di caratteri) seguito da ":", che deve essere il primo elemento di una riga di orchestra; per esempio

```
pippo: a1   oscil   10000, 440, 1
```

Non esiste invece alcuna istruzione di salto durante il passo di calcolo dei segnali audio.

Vi sono poi istruzioni che modificano il flusso di programma solo al verificarsi di una determinata condizione logica *COND*, i cosiddetti *salti condizionali*, e sono:

```
if    ia COND  ib    igoto     etichetta
if    ka COND  kb    kgoto     etichetta
if    ia COND  ib    goto      etichetta
```

ia e *ib* sono espressioni, e *COND* è uno degli *operatori logici* permessi in Csound. Questi sono

>	(maggiore di)
<	(minore di)
>=	(maggiore o uguale a)
<=	(minore o uguale a)
==	(uguale a)
!=	(diverso da)

Si noti che il simbolo "=" indica l'assegnazione a una variabile (per esempio *ia = 12*, mentre il simbolo "==" è un operatore di confronto (*13==13* dà VERO, *12==13* dà FALSO).

Quindi nel frammento di orchestra:

```
...
        if      i1>i2   goto salto
a1      rand    iamp
        goto    ok
salto:
a1      oscil   iamp, ifrq, 1
ok:
...
```

se *i1* è maggiore di *i2*, allora si salta all'etichetta *salto*, e e la variabile audio *a1* viene definita come uscita di un oscillatore; se *i1* è minore o uguale a *i2*, allora non vi è salto e alla variabile audio *a1* viene assegnato un rumore bianco.

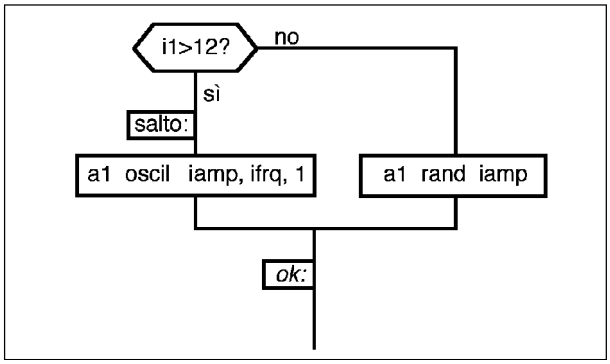


Fig. 17-1

Notiamo il salto incondizionato *goto ok,* che serve a impedire l'esecuzione della riga

```
a1      oscil  iamp,ifrq,1
```

dopo la prima assegnazione con *rand*.

In Fig.17-1 è illustrato il diagramma di flusso di questo frammento di orchestra.

Ripetiamo che, per motivi di velocità di esecuzione, non sono previsti *test* condizionali sulle variabili audio, cioè non è possibile scrivere:

```
if      a1<a2       goto  pippo
```

Se ne abbiamo proprio bisogno, anche a costo di una certa lentezza di esecuzione, come facciamo? Poniamo semplicemente la frequenza di controllo uguale a quella audio, e usiamo gli operatori che permettono di convertire un tipo di variabile in un altro, in questo modo:

```
        sr      =   44100
        kr      =   44100
        ksmps   =   1
        nchnls  =   1
        instr       1
a1      oscil       10000, 440, 1
a2      oscil       10000, 458, 1
k1      downsamp    a1
k2      downsamp    a2
        if          k1<k2   goto    pippo
        ...
```

Gli operatori che permettono di convertire un tipo di variabile in un altro sono:

```
i1    =          i(ksig)
i1    =          i(asig)
k1    downsamp   asig[, iwlen]
a1    upsamp     ksig
a1    interp     ksig[, istor]
```

i(ksig) e *i(asig*) convertono rispettivamente la variabile di controllo *ksig* e la variabile audio *asig* in una variabile di inizializzazione, "fotografandone" il valore in un dato istante e mantenendolo fisso per tutta la durata della nota.

downsamp converte un segnale audio *asig* in un segnale di controllo *k1*.

upsamp e *interp* convertono un segnale di controllo *ksig* in un segnale audio *a1*. L'*opcode upsamp* ripete semplicemente il valore di *ksig* per *ksmps* volte, mentre *interp* opera una interpolazione lineare fra valori successivi di *ksig*.

iwlen (opzionale) è la lunghezza di una finestra (in campioni) all'interno della quale si calcola una media del segnale audio. Il valore massimo è *ksmps.* I valori 0 e 1 significano "nessuna media" (infatti non è possibile calcolare la media di un solo campione); per richiedere il calcolo della media sarà necessario specificare un valore maggiore di uno. Per *default* non viene calcolata alcuna media.

Altri paragrafi in questo capitolo:

17.3 REINIZIALIZZAZIONE

17.4 PROLUNGARE LA DURATA DI UNA NOTA

17.5 *DEBUGGING*

17.6 FUNZIONI MATEMATICHE E TRIGONOMETRICHE

17.7 ASSEGNAZIONE CONDIZIONALE

APPROFONDIMENTI

17.A.1 GENERAZIONE DI EVENTI COMPLESSI

LISTA DEGLI *OPCODE*

	igoto	**etichetta**
	kgoto	**etichetta**
	goto	**etichetta**
	if	**ia COND ib igoto etichetta**
	if	**ka COND kb kgoto etichetta**
	if	**ia COND ib goto etichetta**
i1	**=**	**i(variabile di controllo)**
i1	**=**	**i(variabile audio)**
k1	**downsamp**	**variabile audio [, lunghezza della finestra]**
a1	**upsamp**	**variabile di controllo**
a1	**interp**	**variabile di controllo [, inizializzazione della memoria]**
	reinit	**etichetta**
	rireturn	
	timout	**tempo di inizio, durata, etichetta**
	ihold	
	turnoff	
	print	**variabile di inizializzazione [,variabile di inizializzazione2,...]**
	display	**variabile, periodo [, flag di attesa]**
	dispfft	**variabile, periodo, dimensione della finestra[,tipo di finestra][, unità di misura di ampiezza] [, flag di attesa]**
	printk	**periodo, variabile di controllo [, numero di spazi]**
	printk2	**variabile di controllo [, numero di spazi] printks "testo", periodo, variabile di controllo1, ..., variabile di controllo4**
ir	**pow**	**base, esponente**
kr	**pow**	**base, esponente, [fattore di normalizzazione]**
ar	**pow**	**base, esponente, [fattore di normalizzazione]**

i(kx)	**conversione in variabile i**
int(x)	**parte intera**
frac(x)	**parte decimale**
abs(x)	**valore assoluto**
sqrt(x)	**radice quadrata**
exp(x)	**esponenziale**
log(x)	**logaritmo naturale**
log10(x)	**logaritmo in base 10**
ampdb(x)	**conversione da dB a ampiezza assoluta**
dbamp(x)	**conversione da ampiezza assoluta a dB**

sin(x)	**seno**
sininv(x)	**arcoseno**
sinh(x)	**seno iperbolico**
cos(x)	**coseno**
cosinv(x)	**arcocoseno**
cosh(x)	**coseno iperbolico**
tan(x)	**tangente**
taninv(x)	**arcotangente**
tanh(x)	**tangente iperbolica**

APPENDICE

MATEMATICA E TRIGONOMETRIA

A.1.1 FREQUENZE DELLA SCALA CROMATICA TEMPERATA

ottave	0	1	2	3	4	5	6	7
DO	32.7032	65.4064	130.8128	261.6256	523.2511	1046.5023	2093.0045	4186.0090
DO#	34.6478	69.2957	138.5913	277.1826	554.3653	1108.7305	2217.4610	4434.9221
RE	36.7081	73.4162	146.8324	293.6648	587.3295	1174.6591	2349.3181	4698.6363
RE#	38.8909	77.7817	155.5635	311.1270	622.2540	1244.5079	2489.0159	4978.0317
MI	41.2034	82.4069	164.8138	329.6276	659.2551	1318.5102	2637.0205	5274.0409
FA	43.6535	87.3071	174.6141	349.2282	698.4565	1396.9129	2793.8259	5587.6517
FA#	46.2493	92.4986	184.9972	369.9944	739.9888	1479.9777	2959.9554	5919.9108
SOL	48.9994	97.9989	195.9977	391.9954	783.9909	1567.9817	3135.9635	6271.9270
SOL#	51.9131	103.8262	207.6523	415.3047	830.6094	1661.2188	3322.4376	6644.8752
LA	55.0000	110.0000	220.0000	440.0000	880.0000	1760.0000	3520.0000	7040.0000
LA#	58.2705	116.5409	233.0819	466.1638	932.3275	1864.6550	3729.3101	7458.6202
SI	61.7354	123.4708	246.9417	493.8833	987.7666	1975.5332	3951.0664	7902.1328

Altri paragrafi in questo capitolo:

LETTURE

CSOUND E GNU/LINUX

di Nicola Bernardini

1. INTRODUZIONE: CHE COS'È GNU/LINUX

GNU/Linux[1] è un sistema operativo (come Windows e MacOs, per intenderci) completamente Libero (con la "L" maiuscola), nato su piattaforme *hardware* Intel (486 e Pentium) e oggi disponibile anche su DEC Alpha, su PowerMac (PowerPC) e su Sun SPARC.

Il *Software Libero* è quel programma la cui licenza d'uso rispetti i seguenti quattro principi fondamentali:

- 1. la libertà di esecuzione del programma per qualsivoglia scopo
- 2. la libertà di studiare il funzionamento del programma e di adattarlo ai propri scopi
- 3. la libertà di ridistribuire copie del programma
- 4. la libertà di apportare miglioramenti al programma e di ridistribuire pubblicamente questi miglioramenti in modo che la comunità ne possa beneficiare

Ci sono alcune licenze d'uso studiate per favorire lo sviluppo del *Software Libero*.Tra queste le più note sono la licenza GNU/GPL (General Public License) e la licenza Free/BSD. In aggiunta alle quattro libertà fondamentali, la licenza GNU/GPL garantisce anche l'inviolabilità presente e futura delle libertà sopracitate per il *software* con essa licenziato.

Tecnicamente, questo significa anche che:

- 1. GNU/Linux è garantito contro le "incompatibilità" ingenerate da classiche strategie commerciali truffaldine (ad es.: "la nuova versione del *software xy* non gira più sul vecchio hardware", ecc.); per questo motivo GNU/Linux viene utilizzato, tra l'altro, in numerosi progetti di "ecologia informatica" (riciclaggio di *hardware* obsoleto, ecc.)
- 2. GNU/Linux è disponibile in numerosissimi siti Internet in tutte le forme (eseguibili binari per macchine Intel, Alpha, PowerPC, ecc.; *tutto* il codice in

[1] il prefisso *GNU* (un acronimo ricorsivo che sta per *GNU's Not Unix*™)è utilizzato nel software prodotto o patrocinato dalla FSF (*Free Software Foundation*), la prima e la maggiore coalizione di programmatori dedicata alla produzione di *Software Libero*.

forma sorgente; documentazione e così via) ed è disponibile inoltre in numerose distribuzioni su CD-ROM a prezzi estremamente contenuti

- 3. la disponibilità delle sorgenti di GNU/Linux rende agevole (a chi lo desideri) il controllo del funzionamento dei programmi, la loro eventuale modifica per soddisfare esigenze diverse

- 4. l'utilizzazione diffusa da parte di numerosi specialisti (dell'ordine di qualche milione di persone) consolida molto rapidamente il *software* utilizzato permettendogli di evolvere alla stessa velocità dell'*hardware*; il *software* così prodotto e verificato è, generalmente, di qualità notevolmente superiore ai corrispondenti pacchetti commerciali ed evolve a ritmi molto elevati; naturalmente, non essendoci necessità commerciali da soddisfare, è possibile rimanere indietro con le versioni senza problemi (le nuove versioni servono in generale per correggere problemi, aggiungere funzionalità e gestione di nuovo *hardware*)

- 5. la non-commerciabilità del *software* spinge GNU/Linux a comunicare con tutti i sistemi operativi in commercio (Windows, DOS, MacOs, OS/2, ecc.) attraverso l'adozione di qualsiasi standard di fatto (sia commerciale che di ricerca) reputato utile o interessate dalla comunità; GNU/Linux può quindi leggere qualsiasi *file system* e possiede numerosi emulatori per i sistemi più diffusi

Inoltre GNU/Linux è, a tutti gli effetti, un sistema completo di tipo Unix™[2] quindi si tratta di un sistema realmente *multitasking* (cioè che può eseguire numerosi programmi contemporaneamente) e multi-utente (cioè permette a numerosi utenti, attraverso diversi terminali, di lavorare sullo stesso elaboratore).

Altri paragrafi in questa lettura:

[2] In effetti, GNU/Linux si chiama così per evitare problemi di *copyright* sul marchio Unix™ e anche perché l'iniziatore del progetto si chiama Linus Torvalds).

6. STRUMENTI AUSILIARI GENERICI

7. INDIRIZZARIO INTERNET PER CSOUND E GNU/LINUX

GENERAZIONE E MODIFICA DI PARTITURE CON ALTRI LINGUAGGI

di Riccardo Bianchini

1. SCELTA DEL LINGUAGGIO DI PROGRAMMAZIONE

Se vogliamo generare o modificare una partitura con un linguaggio di programmazione di uso generale, la prima scelta da operare sarà, ovviamente, quale linguaggio usare. Vi sono linguaggi più adatti e linguaggi meno adatti; linguaggi facili da apprendere e linguaggi che prima di essere usati hanno bisogno di un periodo di studio abbastanza lungo; linguaggi, infine, che sono disponibili su una determinata piattaforma e linguaggi che non lo sono.

Si ottengono in genere risultati interessanti usando i cosiddetti linguaggi per l'intelligenza artificiale (LISP, PROLOG etc.), perché permettono di definire regole di generazione di una partitura in forma complessa e abbastanza vicina al linguaggio umano.

In questa lettura useremo invece il linguaggio BASIC, perché è di facile apprendimento ed è disponibile praticamente su tutti i *personal computer*. Il particolare dialetto che useremo è il Microsoft™ Quick Basic. Se vogliamo sostituirlo con Visual Basic, basterà scrivere i programmi che troverete in questa lettura all'interno di *Form Load*, sostituendo alle righe contenenti le istruzioni *PRINT USING* le righe commentate che le seguono.

2. CHE COSA CI SERVE?

Un linguaggio di programmazione come il Microsoft™ Quick Basic è molto ricco di istruzioni. Quali caratteristiche del linguaggio ci serviranno? Ben poche: la capacità di leggere e scrivere un file di testo, la capacità di formattare l'*output*, alcune strutture di controllo (*FOR...NEXT*, *WHILE...WEND* e *DO...LOOP*) e la capacità di eseguire calcoli semplici.

Utile, ma non indispensabile, è la possibilità di richiamare *subroutine* con argomenti.

Altri paragrafi in questa lettura:

SINTESI ADDITIVA CONTROLLATA DA DIADI

di James Dashow
(traduzione di Riccardo Bianchini)

LA SINTESI DEL SUONO MEDIANTE ITERAZIONE DI FUNZIONI NON LINEARI

di Agostino Di Scipio

1. INTRODUZIONE

L'iterazione di funzioni non lineari è parte della teoria matematica del caos deterministico (*chaos theory*). Nel 1991 iniziai a studiare la possibilità di utilizzare questo tipo di procedimenti per la sintesi del suono. Mi accorsi allora che questo approccio era particolarmente adatto a sintetizzare suoni di proprietà acustiche e percettive dinamiche assai particolari, talvolta uditi come tessiture sonore ricche di turbolenze e transienti di rumore (aperiodicità), talvolta percepite come immagini sonore di spettro quasi armonico (periodicità).

Questa lettura si limita a fornire alcuni esempi di realizzazione con Csound (per approfondimenti scientifici e teorici si rimanda alle indicazioni incluse in bibliografia). Si tratta di esempi molto semplici, facilmente modificabili ed estendibili da parte del lettore. Si deve tener presente che il modo più costruttivo di affrontare questo tipo di sintesi è di carattere esplorativo e sperimentale, dal momento che piccoli mutamenti nei valori dei parametri causano cambiamenti anche assai radicali nei risultati ottenuti, in maniera spesso imprevedibile.

Infine bisogna sottolineare che stiamo qui trattando di una tecnica di sintesi "non-standard", ovvero di un processo che non ha alla base alcun modello propriamente acustico ma un modello procedurale arbitrario, un processo di composizione del suono capace di dar luogo a sequenze numeriche che, se considerate come campioni sonori, possono avere qualità interessanti a fini musicali e di *sound-design*.

2. DESCRIZIONE GENERALE

Iterare una funzione significa applicare una trasformazione *f* ad un valore iniziale, *x(0)*, e nel ripetere poi l'applicazione della stessa trasformazione al risultato ottenuto, e così di seguito per n volte:

x(n) = f (x(n-1))

Indicheremo *x(n*) come la n-esima "iterata" ottenuta dall'applicazione di *f* a *x(0)*. Se *f* è nonlineare (per esempio una sinusoide, una spezzata, un polinomio di grado elevato, etc.), l'andamento del processo varia sensibilmente al variare di *x(0)* e degli eventuali parametri di *f* e in alcuni casi rende il risultato dell'iterazione del tutto imprevedibile.

Per utilizzare questo processo come tecnica di sintesi del suono, possiamo ricorrere al seguente schema di istruzioni:

a - inizializza *x(0*) e i parametri di *f*'
b - calcola la n-esima iterata, *x(n)*, ottenendo così il campione in uscita
c - aggiorna *x(0)* e i parametri di *f*
d - ripeti da *b*

In altre parole la successione dei campioni sarà la successione delle n-esime iterate della funzione *f* al variare di *f* e *x(0)*. Se *i* è l'indice di successione dei campioni, la sequenza sarà formulata sinteticamente come:

x(n,i) = f(i) (x(n-1,i))

Questo metodo è assolutamente generale, e definisce in effetti una classe di possibili forme di sintesi del suono. Il principio costante sta nel fatto che la forma d'onda del suono che viene generato è la successione delle n-esime iterate di una determinata funzione. Per effettuare una implementazione del processo, si tratta allora di scegliere la particolare funzione nonlineare.

Si pensi ad esempio alla tecnica nota come *waveshaping* o distorsione non lineare (DNL): abbiamo appunto una funzione distorcente (ad esempio un polinomio di Chebishev, o una sinusoide) che trasforma un segnale in ingresso (una successione di valori). Se il risultato lo sottoponiamo di nuovo alla trasformazione dalla quale è stato ottenuto, avremo allora una forma particolare di iterazione di funzione nonlineare.

Naturalmente la scelta di *f* è decisiva e determina il tipo di risultati sonori raggiunti. Tuttavia in ambito scientifico è stato sottolineato che le successioni numeriche ottenute con questo tipo di processo dipendono nelle loro caratteristiche morfologiche tempo-varianti dal semplice fatto che si tratta di un processo iterativo: è l'operazione di iterazione che permette di fare emergere forme coerenti, siano esse ordinate e regolari, siano esse caotiche e apparentemente prive di regolarità.

Altri paragrafi in questa lettura:

3. IMPLEMENTAZIONE

RIFERIMENTI BIBLIOGRAFICI

GSC4: SINTESI GRANULARE PER CSOUND

di Eugenio Giordani

1. INTRODUZIONE

L'idea di scrivere un programma per la sintesi granulare in ambiente Csound è nata quasi come una sfida verso una tipica idiosincrasia dei linguaggi *Music-n* (Music V, Music 11, Cmusic, Csound etc.) relativa alla loro impostazione basata sul dualismo orchestra-partitura. In generale questa filosofia di fondo riproduce, sebbene con evidenti differenze, un concetto di costruzione musicale legato alle note (*note-oriented*) e per questo lineare. Nei linguaggi *Music-n*, sebbene una nota possa sottendere algoritmi molto elaborati, viene impiegata generalmente una relazione paritetica tra orchestra e *score*: una riga di partitura (nota) produce un oggetto sonoro, complesso a piacere, ma un solo oggetto. Se l'algoritmo è molto complesso, le linee di partitura possono essere poche e molto rarefatte, anche se ciò non rappresenta di certo una regola..

Nella sintesi granulare, che come è noto necessita di una grandissima quantità di frammenti sonori (grani o *quanta acustici*) per unità di tempo, sarebbe impensabile demandare alla *score* la generazione dei singoli grani, sia per evidenti questioni di praticità sia perché non sarebbe agevole governare i parametri della sintesi ad alto livello. L'obiettivo è stato quindi quello di automatizzare il più possibile la generazione dei singoli grani, e di lasciare quindi al musicista la possibilità di controllare globalmente gli andamenti di tutti i parametri della sintesi. Per fare questo è stato necessario pensare ad una struttura algoritmica dell'orchestra in grado di generare all'interno di una singola nota, un numero molto grande di "sub-note", intendendo con questo termine microeventi acustici di durata generalmente molto limitata. Lasciando al lettore la facoltà di approfondire l'organizzazione progettuale e i dettagli di programmazione relativi all'orchestra, viene data di seguito una descrizione della sola utilizzazione del programma in termini di programmazione della *score* così che sia possibile la sperimentazione immediata dell'algoritmo.

2. STRUTTURA GENERALE DELL'ALGORITMO E DESCRIZIONE DEI PARAMETRI DELLA SINTESI

GSC4 è in grado di generare 4 flussi indipendenti (voci) di grani sonori su un fronte stereofonico, ma è possibile per l'utente l'estensione dell'algoritmo ad un numero maggiore di voci e di canali. La ragione per cui la versione base è a 4 voci dipende dal fatto che questa configurazione consente alla *score* di essere eseguita in tempo reale con le versioni recenti di Csound.

Seguendo questa configurazione di base, l'orchestra è costituita da 4 strumenti di generazione, uno strumento per le funzioni di controllo (come a tutti gli strumenti di generazione) e infine uno strumento per il missaggio e l'uscita e precisamente:

instr 1, 2, 3, 4 : strumenti di generazione
instr 11 : strumento per il controllo
instr 21 : strumento per il missaggio e l'uscita

Dato questo schema, per generare un evento completo occorre quindi attivare contemporaneamente 6 strumenti ed attribuire ad essi lo stesso *action-time* (p2) e la stessa durata (p3).

Il maggior numero di parametri (fino a p13) è concentrato nello strumento per il controllo, mentre gli altri strumenti contengono solo 4 parametri (da p1 a p4). Il controllo del processo di granulazione avviene attraverso la specificazione da parte dell'utente di un certo numero di funzioni che descrivono l'andamento nel tempo dei parametri di sintesi così definiti ed associati allo strumento 11 (controllo):

1)	durata media dei grani in millisecondi	p4
2)	variazione *random* della durata dei grani in millisecondi	p5
3)	*delay* medio dei grani in millisecondi	p6
4)	variazione *random* del *delay* dei grani in millisecondi	p7
5)	proporzione di rampa in unità adimensionali	p8
6)	frequenza centrale della forma d'onda audio in Hz	p9
7)	variazione random della frequenza della forma d'onda in Hz	p10
8)	fase centrale della forma d'onda (normalizzata)	p11
9)	variazione *random* della fase della forma d'onda (normalizzata)	p12
10)	ampiezza complessiva (normalizzata)	p13

Ciascuno di questi valori indica il numero di funzione assegnata per il controllo del parametro associato per cui, durante l'attivazione dello strumento, devono esistere all'interno della *score* 10 funzioni tabulate con uno dei metodi di generazione disponibili (GEN). Se ci riferiamo al listato relativo alla *score* riportato in Appendice, possiamo osservare le funzioni di controllo dei vari parametri della sintesi.

f11 :
la durata media dei grani è definita da una funzione lineare (GEN07) con valore iniziale di 10 millisecondi che dopo 256/512 di p3 (durata complessiva dell'evento) raggiunge il valore di 20 millisecondi, rimane costante per 128/512 e tende al valore finale di 16 millisecondi dopo 128/512.

f12:
il valore massimo della variazione *random* della durata dei grani è definito da una funzione lineare (GEN07) con valore iniziale di 4 millisecondi che dopo 256/512 di p3 si riduce a 1 millisecondo e dopo 256/512 tende al valore finale di 0 millisecondi (nessuna variazione *random*).

f13:
il *delay* dei grani è definito da una funzione lineare (GEN07) con valore iniziale di 10 millisecondi che dopo 256/512 di p3 aumenta a 20 millisecondi e dopo 256/512 tende al valore finale di 5 millisecondi.

f14:
il valore massimo della variazione *random* del *delay* dei grani è definito da una funzione lineare (GEN07) con valore iniziale di 0 millisecondi (nessuna variazione *random*) che per 128/512 di p3 rimane costante a 0 millisecondi; dopo 256/512 sale a valore a 2 millisecondi e dopo 128/512 tende al valore finale 0 millisecondi (nessuna variazione *random*).

f15:
la proporzione di rampa dei grani è definita da una funzione lineare (GEN07) con valore iniziale di 2 che dopo 256/512 di p3 aumenta a 4 e dopo 256/512 tende al valore finale di 2. In pratica, la forma dell'inviluppo dei grani si trasforma gradualmente da un triangolo iniziale verso un trapezio e di nuovo un triangolo. Nel triangolo, la rampa di salita è esattamente pari alla metà della durata dell'inviluppo mentre la seconda metà dell'inviluppo è pari alla rampa di discesa (la fase di *sustain* è nulla). Nel trapezio si arriva ad una rampa di salita (e discesa) pari ad 1/4 della durata dell'inviluppo (la fase di *sustain* non è nulla e vale 2/4 la durata dell'inviluppo).

f16:
la frequenza della funzione audio è definita da una funzione lineare (GEN07) con valore iniziale di 220 Hz che rimane costante per l'intero evento. Nella *score* è riportata una linea (commentata) relativa all'andamento di valori di controllo per la frequenza (da 1.345 a 3.345) quando venga impiegato come materiale audio il *file* esterno *sample.wav* (vedi *f1*) di lunghezza 32768 campioni. In tal caso il valore iniziale 1.345 deriva proprio dal rapporto 44100/32768 e rappresenta la frequenza originale del campione.

f17:
il valore massimo della variazione *random* della frequenza della funzione audio è definito da una funzione lineare (GEN07) con valore iniziale 0 Hz (nessuna variazione) e che tende dopo p3 al valore finale 110 Hz (equivale al 50 % di modulazione)

f18:
la fase della funzione audio è definita da una funzione lineare (GEN07) con valore iniziale di 0 che rimane costante per tutto l'intero evento.

f19:
il valore massimo della variazione *random* della fase della funzione audio è definito da una funzione lineare (GEN07) con valore iniziale 0 (nessuna variazione) e che rimane costante per tutto l'intero evento.

f20:
l'ampiezza complessiva è definita da una funzione lineare (GEN07) con valore iniziale di 0 che dopo 128/512 di p3 sale a 1, rimane costante per 256/512 e tende a 0 dopo 128/512.

f1:
la forma d'onda audio è definita da una funzione somma di seni (GEN10) con un contenuto spettrale definito dai coefficienti di ampiezza delle singole componenti

È opportuno notare che, ad eccezione delle funzioni *f1* ed *f20*, il parametro p4 è sempre negativo (-7) poiché i *breakpoints* devono esprimere i valori dei vari parametri in una scala assoluta.

Per i quattro strumenti di generazione a cui si riferiscono *i1*, *i2*, *i3*, *i4*, è sufficiente specificare oltre ai 3 parametri obbligatori p1, p2 e p3, il numero della funzione audio (*f1* in questo esempio) e per lo strumento 21 il fattore di scala all'uscita.

La funzione audio può essere indifferentemente un prototipo di forma d'onda periodica o un segnale campionato importato con la funzione GEN01. Nel primo caso il valore del parametro p4 dello strumento 21 deve contenere l'ampiezza massima all'uscita (da 0 a 32767), mentre nel secondo il valore è definito nell'ambito da 0 a 1. Ciò dipende dal fatto che generalmente il segnale audio campionato non viene rinormalizzato in sede di lettura (GEN-1).

Come già accennato, è opportuno evidenziare che il segnale audio può essere sia un ciclo di una funzione periodica sia un vero e proprio segnale campionato. Benché non vi sia nessuna differenza funzionale nell'uno o nell'altro caso, occorre fare attenzione nella specifica degli ambiti frequenziali.

Nel primo caso il valore della frequenza e le eventuali variazioni *random* sono esplicitamente quelli desiderati. Nel secondo caso il valore nominale della frequenza naturale dell'oscillatore (ovvero quel valore per cui il segnale audio viene riprodotto alla sua frequenza originale) si può calcolare dal rapporto tra la frequenza di campionamento (*sr*) e la lunghezza della tabella che contiene la forma d'onda (*Fn = sr* / lunghezza della tabella).

Ad esempio se il segnale audio campionato a 44.1 kHz ha una lunghezza di 65536 campioni (circa 1.486 secondi), la frequenza naturale sarà pari a 44100 / 65536 = 0.672 Hz.

Dal momento che quasi mai si verifica che la durata del segnale audio corrisponde esattamente alla lunghezza di tabella potenza di 2 è sufficiente prevedere una lunghezza di tabella che approssima per eccesso tale valore. Rispetto all'esempio precedente, se la durata del segnale audio a 44.1 kHz fosse di 1.2 secondi l'effettiva lunghezza di tabella dovrebbe essere 44.1 x 1.2 = 52920 campioni e quindi approssimabile per eccesso con una lunghezza di 65536. La frequenza naturale sarebbe comunque sempre 0.672 Hz poiché gli oscillatori di Csound sono pensati per indirizzare aree di memoria che abbiano come dimensione una potenza di due. La differenza sostanziale è che l'indice di fase in questo caso deve essere compreso tra 0 e 0.807, valore quest'ultimo che si ottiene dal rapporto 52920/65536.

Anche le variazioni *random* di frequenza dovranno essere di ordine di grandezza congruente con i valori deterministici impostati. Sempre riferendoci all'esempio precedente, con un valore di frequenza naturale pari a 0.672 Hz, una variazione pari al 10 % vale circa 0.06 Hz.

Dal momento che la variazione è assoluta e non percentuale, l'ambito di variabilità non è simmetrico rispetto al valore centrale (o medio), e quindi per grandi variazioni sarebbe più opportuno che i valori fossero espressi percentualmente o convertiti in unità *octave*.

Scendendo nei dettagli pratici, e immaginando che l'uso più frequente della Sintesi Granulare sia quello di applicare tale processo a forme d'onda campionate, le considerazioni seguenti dovrebbero chiarire il concetto di frequenza naturale e di come i relativi controlli si relazionano con essa.

Dal punto di vista dell'utente, come si è accennato sopra, le cose assumono una prospettiva diversa a seconda che la forma d'onda da granulare sia un solo ciclo di segnale o un intero *soundfile*. Vale la pena di ricordare che, in generale, un oscillatore "non ha modo di sapere" quando si verifica un caso piuttosto che l'altro: la generazione del segnale da parte di un oscillatore è legata meramente ad un processo di lettura ciclica dei campioni con un determinato passo di incremento (intero o frazionario) all'interno della tabella che li contiene; quando l'incremento è unitario allora il segnale viene riprodotto alla frequenza naturale. In ogni caso vale sempre la relazione :

Fn = I x SR/ L

dove :

Fn = frequenza naturale dell'oscillatore
SR = frequenza di campionamento
I = passo di lettura in tabella
L = lunghezza della tabella

Riferendoci all'esempio già citato, se si vuole granulare un *file* di durata 1.2 s campionato a 44.1 kHz e lo si vuole riprodurre alla sua frequenza naturale, occorrerà specificare una funzione per il controllo della frequenza del tipo :

```
f16   0   512  -7   0.672   512   0.672
```

dove 0.672 è il valore in Hz della frequenza naturale del segnale che si ottiene dal rapporto *sr*/lunghezza tabella ovvero 44100/65536 (essendo 65536 la lunghezza che approssima per eccesso il numero di campioni corrispondenti ad 1.2 s di suono, cioè 44100 x 1.2 = 52920). Quando si prevede di utilizzare sempre e solo la granulazione dei segnali campionati , il valore della variabile *ifreq* va modificato nell'orchestra e cioè sempre moltiplicato per l'espressione *sr/ftlen(ifun)*. In questo modo la linea di *score* che controlla la frequenza si trasforma nel modo seguente:

```
f16   0   512  -7   1   512   1
```

In questo modo la frequenza viene gestita come rapporto rispetto alla frequenza naturale, evitando così di dover calcolare volta per volta i valori reali della frequenza. Se ad esempio si vuole eseguire un glissato continuo del segnale granulato dalla frequenza naturale ad una quinta naturale sopra (rapporto intervallare 3:2=1.5) la linea di controllo diviene:

```
f16   0   512  -7   1   512   1.5
```

Un discorso analogo può essere fatto per la variazione randomica.

Se la funzione audio è un prototipo di funzione periodica, il valore della fase dell'oscillatore non ha praticamente influenza sul risultato acustico finale mentre diventa di vitale importanza se la funzione audio è un segnale campionato. In questo caso infatti, la fase dell'oscillatore assume la funzione di puntatore alla tabella e permette di "granulare" punti ben precisi all'interno del segnale audio.

L'impiego più semplice ed immediato di questo parametro si realizza attraverso la definizione di una funzione lineare (GEN07) con valore iniziale 0 e valore finale 0.999 :

```
f18   0   512  -7   0   512   0.999
```

In questo caso la forma d'onda viene letta mantenendo la direzione temporale originale. Se i valori 0 e 0.999 vengono scambiati di posto, la forma d'onda viene letta in modo retrogrado. Seguendo questo principio si possono ottenere risultati molto diversi ed interessanti.

Per esempio la seguente funzione realizza nella prima metà dell'evento la retrogradazione della prima metà della forma d'onda, mentre nella seconda metà realizza la compressione relativa temporale dell'intera forma d'onda, questa volta ripercorsa in modo diretto:

```
f18  0  512  -7  0.5  256  0  256  0.999
```

Naturalmente la durata dell'evento (p3) può essere mantenuta identica alla durata reale della forma d'onda o viceversa aumentata o ridotta. In questi due ultimi casi si avrà rispettivamente una dilatazione o una compressione assoluta del tempo.

Altre possibilità si aprono impiegando funzioni di controllo non lineari e di forma più complessa, eventualmente aggiungendo una variazione *random* della fase attraverso la funzione f19

3. DESCRIZIONE DELL'ALGORITMO DI SINTESI

L'algoritmo implementato per la sintesi granulare segue l'approccio indicato da B.Truax [2] e benché quest'ultimo sia stato realizzato in tempo reale attraverso il processore DMX-1000 controllato da un *host computer* (DEC PDP-11) si è tentato di riprodurne la sua filosofia di fondo. In sostanza si trattava di realizzare un banco di generatori di inviluppi controllati da un insieme di parametri che nel caso originale venivano aggiornati ad un *rate* di 1 kHz (1 millisecondo) dall'*host* (fig.1).

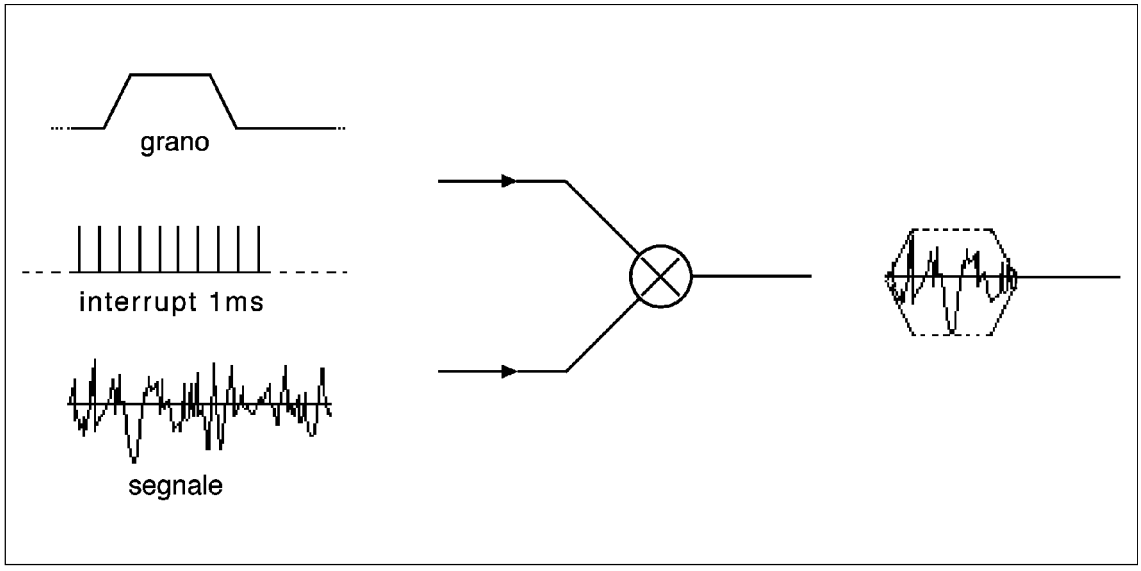


Fig. 1

Nell'implementazione di Truax questa condizione veniva raggiunta attraverso due programmi concorrenti, uno sul DSP e l'altro sullo *host*. Mentre il primo aveva il compito di generare gli inviluppi, le relative letture del materiale audio (registrato o attraverso sintesi diretta FM o a *table look-up*) e le moltiplicazioni e riscalamenti necessari al processo, il secondo doveva provvedere all'aggiornamento dei parametri della sintesi con un meccanismo, probabilmente basato su una procedura a *interrupt*.

La genesi di questa implementazione Csound risale al 1989 e quindi in un periodo in cui la sintesi in tempo reale era esclusivo appannaggio dei DSP. D'altra parte, essendo i programmi della famiglia *Music-n* (Music V, Music 11, Cmusic, Csound etc.) lo strumento più diffuso e più economico impiegato dai compositori, poteva essere utile realizzare uno strumento di sintesi granulare con tali mezzi, anche se in tempo differito.

Data la necessità di generare una grande quantità di micro-eventi per unità di tempo, non era ragionevole seguire un approccio un evento/una nota. In tal modo il problema sarebbe stato spostato a un eventuale programma di *front-end* (interfaccia di utilizzazione) per la preparazione della partitura indipendente dalla sintassi di Csound. A tale scopo sarebbe stato sufficiente utilizzare Cscore o altri pre-processori.

Altri paragrafi in questa lettura:

DA CSOUND A MAX - GENERAZIONE DI PARTITURE E SINTESI IN TEMPO REALE CON MACINTOSH

di Maurizio Giri

1. CHE COS'È MAX

MAX, progettato originariamente all'IRCAM[1] da Miller Puckette per il controllo del sistema 4x, è un ambiente grafico interattivo completamente configurabile che gestisce ed elabora flussi di dati per il controllo di strumenti elettronici o di altri programmi.

La comunicazione tra computer e macchine da controllare avviene tramite il protocollo MIDI.

MAX dispone di un centinaio di funzioni primitive visualizzabili sullo schermo del computer come oggetti grafici muniti di ingressi e di uscite.

Questi oggetti sono collegabili tra loro ed i dati passano da un oggetto all'altro attraverso i collegamenti. Ogni oggetto esegue una qualche operazione sui dati che riceve, e passa il risultato dell'elaborazione agli oggetti a cui è collegato. Un insieme di oggetti collegati che svolge una determinata funzione si chiama "*patch*" (che significa, più o meno, "collegamento provvisorio", con chiaro riferimento ai vecchi sintetizzatori analogici modulari che venivano programmati con connessioni fisiche effettuate al volo tramite *patch cords*)

MAX si potrebbe quindi definire un linguaggio grafico di programmazione: si tratta infatti di un sistema completamente aperto che permette, se adeguatamente programmato, di risolvere la maggior parte dei problemi inerenti le esecuzioni *live-electronics*.

MAX è inoltre espandibile: un *patch* si può trasformare in un oggetto (*subpatch*) con ingressi e uscite e può essere utilizzato nello stesso modo di una funzione interna, oppure si può scrivere una funzione esterna in C e trasformarla in un oggetto MAX.

Proprio per questo la libreria degli oggetti continua ad espandersi grazie ai contributi degli utilizzatori del programma.

In questo breve tutorial esamineremo le principali funzioni di MAX e vedremo come sia possibile usarlo per creare partiture per Csound. Parleremo anche di MSP, una estensione di MAX che permette di fare sintesi ed elaborazione del suono in tempo reale. Verranno infine realizzate alcune traduzioni di orchestre Csound in *patch* Max/MSP.

Attenzione: per una migliore comprensione del testo è necessario lanciare il programma "Esempi Max Csound" ed aprire gli oggetti (le "scatolette") contenuti nella finestra "*Max-Csound mainpatch*" con un doppio click. Chi ha una copia di Max, o la versione runtime (MaxPlay) può usare i *file* che si trovano nella cartella "Esempi Max Csound Folder". Il contenuto verrà spiegato nel corso del tutorial.

---

[1] Attualmente Max è prodotto e distribuito da David Zicarelli e dalla sua società Cycling '74 (www.cycling74.com) per la piattaforma Macintosh. Esistono programmi simili a Max per altre piattaforme, a cui accenneremo alla fine di questa lettura.

Altri paragrafi in questa lettura:

DIRECTCSOUND E VMCI:
IL PARADIGMA DELL' INTERATTIVITÀ

di Gabriel Maldonado

1. INTRODUZIONE

La filosofia con cui venne inizialmente progettato Csound era quella dei linguaggi di sintesi in tempo differito, come MUSIC V; filosofia che è rimasta sostanzialmente immutata fin dagli inizi degli anni '60.

Nonostante questo paradigma continui a soddisfare buona parte delle necessità compositive di oggi, e, sebbene questo approccio conservi ancora parecchi vantaggi, non si può negare che il tempo differito in qualche modo ostacola i compositori che richiedono un contatto diretto e immediato col materiale sonoro.

Inizialmente Csound non poteva essere usato nelle performance dal vivo, perché i limiti di velocità dei computer di quell'epoca non consentivano il tempo reale. Di conseguenza non includeva nessuna funzionalità orientata al tempo reale. All'inizio degli anni '90, Barry Vercoe (l'inventore di Csound) aggiunse alcuni *opcode* (cioè i moduli di Csound) per il supporto MIDI da usarsi nel tempo reale. A quell'epoca le macchine in grado di permettere il tempo reale con Csound erano soltanto le Silicon Graphics e qualche altra costosa *workstation* UNIX.

Oggi qualunque PC è sufficientemente veloce da permettere a Csound il tempo reale. DirectCsound, una versione di Csound orientata in modo specifico al tempo reale, colma tutte le lacune riguardanti sia il controllo dal vivo dei parametri di sintesi che la mancanza di interattività di cui Csound soffriva inizialmente. In questa versione sono state implementate molte novità, come il supporto degli ingressi e delle uscite MIDI (che permettono a Csound la comunicazione col mondo esterno), e la riduzione del ritardo dovuto alla latenza; ponendo fine a problemi che apparivano irrisolvibili nelle prime versioni di Csound.

Le nuove funzionalità permettono un controllo totale dei parametri, che possono essere definiti dall'utente in modo estremamente flessibile. Questa flessibilità, unita alla potenza di sintesi, rendono DirectCsound superiore a qualunque sintetizzatore MIDI, in tutti i campi di impiego possibili. Solo qualche anno fa, avere una *workstation* di tale potenza a casa propria, era un sogno. Quella potenza era dominio esclusivo di macchine del costo di centinaia di migliaia di dollari. Essendo DirectCsound gratuito, è sufficiente disporre di un economico PC dotato di scheda audio per essere in grado sia di comporre musica interattivamente, sia di fare performance dal vivo. E' oggi possibile pensare a Csound come ad uno strumento musicale universale.

Le nuove caratteristiche di questa versione di Csound saranno presentate nelle sezioni seguenti, insieme a diversi esempi. Nell'ultima sezione saranno presentate anche alcune funzionalità di VMCI (*Virtual Midi Control Interface*, un programma che emula diversi tipi di controller MIDI). VMCI è stato progettato al fine di fornire un mezzo per controllare DirectCsound in tempo reale, ma può essere usato anche con qualunque altro strumento MIDI.

Altri paragrafi in questa lettura:

LISTA DEGLI *OPCODE*

ISTRUZIONI DELLO *HEADER*

```
        sr          = iarg
        kr          = iarg
        ksmps       = iarg
        nchnls      = iarg

iafn    ftgen       ifn, itime, isize, igen, iarga[,...iargz]

        massign     ichnl, insnum
        strset      iarg, "stringtext"
```

PREPROCESSORE DI ORCHESTRA

```
        #include   "filename"
        #define NAME # replacement text #
        #define NAME(a' b ' c...) # replacement text #
        $NAME.
        $NAME.( a ' b ' c...)
        #undef NAME
```

DEFINIZIONE E ATTIVAZIONE DEGLI STRUMENTI

```
        instr       insnum [, insnum2...]
        endin

        turnon      insnum [,itime]
        schedule    insnum, iwhen, idur, ....
        schedwhen   ktrigger, kinsnum, kwhen, kdur, ....
        schedkwhen  ktrigger, kmintime, kmaxinst, kinsnum, kwhen, kdur [,kp4,...,kpN]
```

ASSEGNAZIONE A VARIABILI

```
        pset    const1, const2, const3, ...
xr      =       xarg
xr      init    iarg
ir      tival
```

REINIZIALIZZAZIONE

reinit label
rigoto label
rireturn

CONTROLLO DI DURATA

ihold
turnoff

MODIFICA DEL FLUSSO DEL PROGRAMMA - SALTI

igoto label
tigoto label
kgoto label
goto label

if ia R ib igoto label
if ka R kb kgoto label
if ia R ib goto label
(R: >, <, >=, <=, ==, =, !=, <, >)
timout istrt, idur, label

ASSEGNAZIONE CONDIZIONALE

(a R b ? v1 : v2) (R: >, <, >=, <=, ==, =, !=, <, >; a, b not a-rate)

CONTROLLO DELL'ESECUZIONE IN TEMPO REALE

inum active insnum
cpuprc insnum, ipercent
maxalloc insnum, icount
prealloc insnum, icount

TEMPO

nr timek
nr times

kr timeinstk
kr timeinsts
kr rtclock

clockon inum
clockoff inum
ival readclock inum

CONTROLLO

ktrig trigger ksig, kthreshold, kmode
ktrig seqtime ktime_unit, kstart, kloop, initndx, kfn_times
trigseq ktrig_in, kstart, kloop, initndx, kfn_values, kout1 [,kout2, kout3, ..., koutN]
kpeak peak nsig
ar follow asig, idelt
ar follow2 asig, katt, krel
ktemp tempest kin, iprd, imindur, imemdur, ihp, ithresh, ihtim, ixfdbak, istartempo, ifn [,idisprd, itweek]
koct,kamp pitch asig, iupdte, ilo, ihi, idbthresh [,ifrqs, iconf, istrt, iocts, iq, inptls, irolloff, iskip]
kcps,krms pitchamdf asig, imincps, imaxcps [,icps] [,imedian] [,idowns] [,iexcps] [,irmsmedi]
tempo ktempo, istartempo

CONTROLLI DELL'INTERFACCIA GRAFICA

ksig sensekey
kx, ky xyin iprd, ixmin, ixmax, iymin, iymax [,ixinit, iyinit]
setctrl islidernum, kvalue, itype [,"label"]
kout control kslidernum
kans buttoninum
kans checkbox inum

MIDI

massign ichnl, insnum
ival notnum
ival veloc [imin, imax]
icps cpsmidi
ncps cpsmidib [irange]
ival cpstmid ifn
ioct octmidi

noct octmidib [irange]
ipch pchmidi
npch pchmidib [irange]
iamp ampmidi iscal [,ifn]
kaft aftouch [imin, imax]
xbend pchbend [imin, imax]
nval midictrl inum [,imin, imax]
initc7/14/21 ichan, ictlno, (ictlno2, ictlno3,) ivalue
ctrlinit ichnkm, ictlno1, ival1 [,ictlno2, ival2 [,ictlno3, ival3[,..ival32]]

ndest midic7/14/21 ictlno, (ictlno2, ictlno3,) nmin, nmax [,ifn]
ndest ctrl7/14/21 ichan, ictlno, (ictlno2, ictlno3,) nmin, nmax [,ifn]
nval chanctrl ichan, ictlno [,ilow,ihigh]

k1,...,kN sliderN ichan, ictlnum1, imin1, imax1, init1, ifn1, ...,ictlnumN, iminN, imaxN, initN, ifnN
k1,...,kN sliderNf ichan, ictlnum1, imin1, imax1, init1, ifn1, icutoff1, ...,ictlnumN, iminN, imaxN, initN, ifnN, icutoffN
i1,...,iN sliderN ichan, ictlnum1, imin1, imax1, ifn1, ...,ictlnumN, iminN, imaxN, ifnN

k1,...,kN sNb14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, initvalue1, ifn1, ..., ictlno_msbN, ictlno_lsbN, iminN, imaxN, initvalueN, ifnN
i1,...,iN sNb14 ichan, ictlno_msb1, ictlno_lsb1, imin1, imax1, ifn1, ..., ictlno_msbN, ictlno_lsbN, iminN, imaxN, ifnN

kstatus, kchan, kdata1, kdata2 midiin
midiout kstatus, kchan, kdata1, kdata2

noteon ichn, inum, ivel
noteoff ichn, inum, ivel
noteondur ichn, inum, ivel, idur
noteondur2 ichn, inum, ivel, idur

moscil kchn, knum, kvel, kdur, kpause
midion kchn, knum, kvel
midion2 kchn, knum, kvel, ktrig

outic ichn, inum, ivalue, imin, imax
outkc kchn, knum, kvalue, kmin, kmax
outic14 ichn, imsb, ilsb, ivalue, imin, imax

outkc14 kchn, kmsb, klsb, kvalue, kmin, kmax

outipb ichn, ivalue, imin, imax
outkpb kchn, kvalue, kmin, kmax
outiat ichn, ivalue, imin, imax
outkat kchn, kvalue, kmin, kmax

outipc ichn, iprog, imin, imax
outkpc kchn, kprog, kmin, kmax

outipat ichn, inotenum, ivalue, imin, imax
outkpat kchn, knotenum, kvalue, kmin, kmax

nrpn kchan, kparmnum, kparmvalu
mdelay kstatus, kchan, kdata1, kdata2, kdelay

mclock ifreq
mrtmsg imsgtype

xtratim iextradur
kflag release

FUNZIONI

iafn ftgen ifn, itime, isize, igen, iarga[,...iargz]

ftlen(ifn) (init-time args only)
ftlptim(ifn) (init-time args only)
ftsr(ifn) (init-time args only)
nsamp(ifn) (init-time args only)
nr tableng nfn
tableicopy idfn, isfn
tablecopy kdfn, ksfn

tableiw isig, indx, ifn [,ixmode] [,ixoff] [,iwgmode]
tablewxsig, xndx, ifn [,ixmode] [,ixoff] [,iwgmode]
tablewkt xsig, xndx, kfn [,ixmode] [,ixoff] [,iwgmode]
tableigpw ifn
tablegpw kfn

ar tablera kfn, kstart, koff
kstart tablewa kfn, asig, koff

tablemix kdfn, kdoff, klen, ks1fn, ks1off, ks1g, ks2fn, ks2off, ks2g
tableimix idfn, idoff, ilen, is1fn, is1off, is1g, is2fn, is2off, is2g

OPERAZIONI MATEMATICHE

Operatori aritmetici e logici (&&, ||, +, -, *, /, %, ^)
i(x) (control-rate args only)
int(x) (init- or control-rate args only)
frac(x) (init- or control-rate args only)
abs(x) (no rate restriction)

sqrt(x) (no rate restriction)
exp(x) (no rate restriction)
log(x) (init- or control-rate args only)
log10(x) (init- or control-rate args only)
logbtwo(x) (no rate restriction)
powoftwo(x) (no rate restriction)
sin(x) (no rate restriction)
sininv(x) (no rate restriction)
sinh(x) (no rate restriction)
cos(x) (no rate restriction)
cosinv(x) (no rate restriction)
cosh(x) (no rate restriction)
tan(x) (no rate restriction)
taninv(x) (no rate restriction)
tanh(x) (no rate restriction)
xr taninv2 xa, xb
ampdb(x) (no rate restriction)
dbamp(x) (init- or control-rate args only)
ampdbfs(x) (no rate restriction)
dbfsamp(x) (init- or control-rate args only)

Generatori di numeri casuali
rnd(x)
birnd(x)

Opcode Equivalenti degli operatori aritmetici

xr	**add**	**xa, xb**
xr	**sub**	**xa, xb**
xr	**mul**	**xa, xb**
xr	**div**	**xa, xb**
xr	**mod**	**xa, xb**
xr	**divz**	**xa, xb, nsubst**
xr	**pow**	**xarg, xpow [,inorm]**
ar	**sum**	**a1, a2, a3, ...**
ar	**product**	**a1, a2, a3, ...**
ar	**mac**	**ksig1, asig2, ksig3, asig4, ...**
ar	**maca**	**asig1, asig2, asig3, asig4, ...**

CONVERTITORI DI ALTEZZA

octpch(pch)	**(init- or control-rate args only)**
pchoct(oct)	**(init- or control-rate args only)**
cpspch(pch)	**(init- or control-rate args only)**
octcps(cps)	**(init- or control-rate args only)**
cpsoct(oct)	**(no rate restriction)**

icps	**cps2pch**	**ipch, iequal**
icps	**cpsxpch**	**ipch, iequal, irepeat, ibase**

GENERATORI

nr	**line**	**ia, idur1, ib**
nr	**expon**	**ia, idur1, ib**
nr	**linseg**	**ia, idur1, ib [,idur2, ic[…]]**
nr	**expseg**	**ia, idur1, ib [,idur2, ic[…]]**
ar	**expsega**	**ia, idur1, ib [,idur2, ic[...]]**
nr	**transeg**	**ia, idur, itype, ib, [idur2, itype, ic[…]]**
nr	**linsegr**	**ia, idur1, ib [,idur2, ic[…]], irel, iz**
nr	**expsegr**	**ia, idur1, ib [,idur2, ic[...]], irel, iz**
nr	**adsr**	**iatt, idec, islev, irel [,idel]**
nr	**xadsr**	**iatt, idec, islev, irel [,idel]**
nr	**madsr**	**iatt, idec, islev, irel [,idel] [,ireltim]**

nr mxadsr iatt, idec, islev, irel [,idel] [,ireltim]

xr table xndx, ifn [,ixmode] [,ixoff] [,iwrap]
xr tablei xndx, ifn [,ixmode] [,ixoff] [,iwrap]
xr table3 xndx, ifn [,ixmode] [,ixoff] [,iwrap]
kr tablekt xndx, kfn [,ixmode] [,ixoff] [,iwrap]
ar tableikt xndx, kfn [,ixmode] [,ixoff] [,iwrap]

kr oscil1 idel, kamp, idur, ifn
kr oscil1i idel, kamp, idur, ifn
ar osciln kamp, ifrq, ifn, itimes

nr phasor xcps [,iphs]
nr phasorbnk xcps, kndx, icnt [, iphs]

nr oscil xamp, xcps, ifn [,iphs]
nr oscili xamp, xcps, ifn [,iphs]
nr oscil3 xamp, xcps, ifn [,iphs]

nr poscil kamp, kcps, ifn [,iphs]
nr poscil3 kamp, kcps, ifn [,iphs]

nr lfo kamp, kcps [,itype]
ar mpulse kamp, kfreq [,ioffset]

ar buzz xamp, xcps, knh, ifn [,iphs]
ar gbuzz xamp, xcps, knh, kih, kr, ifn [,iphs]

ar vco kamp, kcps [,iwave] [,kpw] [,isine] [,imaxd]

ar adsyntkamp, kcps, iwfn, ifreqfn, iampfn, icnt[, iphs]
ar hsboscil kamp, ktone, kbrite, ibasfreq, iwfn, ioctfn[, ioctcnt[, iphs]]

ar adsyn kamod, kfmod, ksmod, ifilcod

ar foscil xamp, kcps, xcar, xmod, kndx, ifn [,iphs]
ar foscili xamp, kcps, xcar, xmod, kndx, ifn [,iphs]

ar fmvoice kamp, kfreq, kvowel, ktilt, kvibamt, kvibrate, ifn1, ifn2, ifn3, ifn4, ivibfn

ar fmbell kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
ar fmrhode kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
ar fmwurlie kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn

ar fmmetal kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
ar fmb3 kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn
ar fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate, ifn1, ifn2, ifn3, ifn4, ivfn

ar1[,ar2] loscil xamp, kcps, ifn [,ibas] [,imod1, ibeg1, iend1] [,imod2, ibeg2, iend2]
ar1[,ar2] loscil3 xamp, kcps, ifn [,ibas] [,imod1, ibeg1, iend1] [,imod2, ibeg2, iend2]

ar lposcil kamp, kfreqratio, kloop, kend, ifn [,iphs]
ar lposcil3 kamp, kfreqratio, kloop, kend, ifn [,iphs]

ar fof xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur [,iphs] [,ifmode]
ar fof2 xamp, xfund, xform, koct, kband, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur, kphs, kgliss
ar fog xamp, xdens, xtrans, aphs, koct, kband, kris, kdur, kdec, iolaps, ifna, ifnb, itotdur [,iphs] [,itmode]

ar grain xamp, xcps, xdens, kampoff, kcpsoff, kgdur, igfn, iwfn, imgdur [,igrnd]

ar granule xamp, ivoice, iratio, imode, ithd, ifn, ipshift, igskip, igskip_os, ilength, kgap, igap_os, kgsize, igsize_os, iatt, idec [,iseed] [,ipitch1] [,ipitch2] [,ipitch3] [,ipitch4] [,ifnenv]

ar[,acmp]sndwarp xamp, xtimewarp, xresample, ifn1, ibeg, iwsize, irandw, ioverlap, ifn2, itimemode

ar1, ar2[, acmp1, acmp2] sndwarpst xamp, xtimewarp, xresample, ifn1, ibeg, iwsize, irandw, ioverlap, ifn2, itimemode

ar pluck kamp, kcps, icps, ifn, imeth [,iparm1, iparm2]
ar wgpluck ifreq, iamp, kpick, iplk, idamp, ifilt, axcite
ar wgpluck2 iplk, kamp, ifreq, kpick, kabsor
ar repluck iplk, kamp, ifreq, kpick, kabsor, axcite

ar wgbow kamp, kfreq, kpres, kratio, kvibf, kvamp, ifn [,iminfreq]
ar wgbrass kamp, kfreq, kliptens, idetk, kvibf, kvamp, ifn [,iminfreq]

ar wgclar kamp, kfreq, kstiff, iatt, idetk, kngain, kvibf, kvamp, ifn [,iminfreq]

ar wgflute kamp, kfreq, kjet, iatt, idetk, kngain, kvibf, kvamp, ifn [,iminfreq] [,ijetrefl] [,iendrefl]
ar wgbowedbar kamp, kfreq, kpos, kbowpres, kgain[, kconst, ktVel, ibowpos, ilow]

ar voice kamp, kfreq, kphoneme, kform, kvibf, kvamp, ifn, ivfn
ar mandol kamp, kfreq, kpluck, kdetune, kgain, ksize, ifn [,iminfreq]
ar moog kamp, kfreq, kfiltq, kfiltrate, kvibf, kvamp, iafn, iwfn, ivfn

ax,ay,az planet kmass1, kmass2, ksep, ix, iy, iz, ivx, ivy, ivz, idelta [,ifriction]
ax,ay,az lorenz ks, kr, kb, kh, ix, iy, iz, iskip

scanu init, irate, ifnvel, ifnmass, ifnstif, ifncentr, ifndamp, kmass, kstif, kcentr, kdamp, ileft, iright, kx, ky, ain, idisp, id
ar scans kamp, kfreq, ifntraj, id [, korder]

ar vibes kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec
ar marimba kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn, idec [,idoubles] [,itriples]
ar gogobel kamp, kfreq, ihrd, ipos, imp, kvibf, kvamp, ivibfn
ar shaker kamp, kfreq, kbeans, kdamp, knum, ktimes [,idecay]
ar cabasa iamp, idettack[, knum, kdamp, kmaxshake]
ar crunch iamp, idettack[, knum, kdamp, kmaxshake]

ar sekere iamp, idettack[, knum, kdamp, kmaxshake]
ar sandpaper iamp, idettack[, knum, kdamp, kmaxshake]
ar stix iamp, idettack[, knum, kdamp, kmaxshake]
ar guiro iamp, idettack[, knum, kdamp, kmaxshake, kfreq, kfreq1]
ar tambourine iamp, idettack[, knum, kdamp, kmaxshake, kfreq, kfreq1, kfreq2]
ar bamboo iamp, idettack[, knum, kdamp, kmaxshake, kfreq, kfreq1, kfreq2]

ar dripwater iamp, idettack[, knum, kdamp, kmaxshake, kfreq, kfreq1, kfreq2]
ar sleighbells iamp, idettack[, knum, kdamp, kmaxshake, kfreq, kfreq1, kfreq2]

GENERATORI RANDOM

nr rand xamp [,iseed] [,iuse31]
nr randh xamp, xcps [,iseed] [,iuse31]
nr randi xamp, xcps [,iseed] [,iuse31]

rnd(x)
birnd(x)
ar pinkish xin[, imethod, inumbands, iseed, iskip]
ar noise xamp, kbeta

xr linrand krange
xr trirand krange
xr exprand krange
xr bexprnd krange
xr cauchy kalpha
xr pcauchy kalpha
xr poisson klambda
xr gauss krange
xr weibull ksigma, ktau
xr betarand krange, kalpha, kbeta
xr unirand krange
seed ival

MODIFICATORI DI SEGNALE

SRCONV - Convert soundfile sample rate.
DNOISE - Denoise soundfiles.

kr downsamp asig [,iwlen]
ar upsamp ksig
ar interp ksig [,iskip]
nr samphold xsig, xgate [,ival, ivskip]

nr integ xsig [,iskip]
nr diff xsig [,iskip]

xr ntrpol xsig1, xsig2, npoint [,imin, imax]

ar fold asig, kincr

kr portk ksig, khtim [,isig]
kr port ksig, ihtim [,isig]

kr tonek ksig, khp [,iskip]

ar tone asig, khp [,iskip]
ar tonex asig, khp [,inumlayer, iskip]

kr atonek ksig, khp [,iskip]
ar atone asig, khp [,iskip]
ar atonexasig, khp [,inumlayer, iskip]

kr resonk ksig, kcf, kbw [,iscl, iskip]
ar reson asig, kcf, kbw [,iscl, iskip]

ar resonxasig, kcf, kbw [,inumlayer, iscl, iskip]
ar resonyasig, kbf, kbw, inumlayer, ksep [,iscl, iskip]

kr aresonk ksig, kcf, kbw [,iscl, iskip]
ar aresonasig, kcf, kbw [,iscl, iskip]

ar resonrasig, kcf, kbw [,iscl, iskip]
ar resonzasig, kcf, kbw [,iscl, iskip]

ar butterhp asig, kfreq [,iskip]
ar butterlp asig, kfreq [,iskip]
ar butterbp asig, kfreq, kband [,iskip]
ar butterbr asig, kfreq, kband [,iskip]

ar lowpass2 asig, kcutoff, kq [,iskip]
ar lowresasig, kcutoff, kresonance [,iskip]
ar lowresx asig, kcutoff, kresonance [,inumlayer, iskip]
ar vlowres asig, kcutoff, kresonance, iord, ksep

ar biquad asig, kb0, kb1, kb2, ka0, ka1, ka2 [,iskip]
ar moogvcf asig, xfco, xres [,iscale]
ar lpf18 asig, kfco, kres, kdist
ar rezzy asig, xfco, xres [,imode]

ar tbvcf asig, xfco, xres, kdist, kasym

nr filter2 nsig, iM,iN,ib0,ib1,..., ibM,ia1,ia2,...,iaN
ar zfilter2 asig, kdamp,kfreq,iM,iN,ib0,ib1,...,ibM,ia1,ia2,...,iaN

alo,ahi,abnd svfilter asig, kcf, kq[, iscl]

ar pinkish xin[, imethod, inumbands, iseed, iskip]

ar nlalp ain, klfact, knfact [, iskip]
ar nlfilt ain, ka, kb, kd, kL, kC
ar pareq asig, kc, kv, kq [,imode]

are, aim hilbert asig

ar delayr idlt [,iskip]
delayw asig
ar delay asig, idlt [,iskip]
ar delay1 asig [,iskip]

ar deltap kdlt
ar deltapi xdlt
ar deltap3 xdlt
ar deltapn xnumsamps
ar multitap asig, itime1, igain1, itime2, igain2 . . .
ar vdelay asig, xdel, imaxdel [,iskip]
ar vdelay3 asig, xdel, imaxdel [,iskip]

ar comb asig, krvt, ilpt [,iskip] [,insmps]
ar alpass asig, krvt, ilpt [,iskip] [,insmps]
ar reverb asig, krvt [,iskip]
ar reverb2 asig, ktime, khdif [,iskip]
ar nreverb asig, ktime, khdif [,iskip] [,inumCombs, ifnCombs] [,inumAlpas, ifnAlpas]
ar nestedap asig, imode, imaxdel, idel1, igain1 [,idel2, igain2 [,idel3, igain3]]
ar, al babo asig, ksrcx, ksrcy, ksrcz, irx, iry, irz [,idiff [,ifno]]

ar wguide1 asig, xfreq, kcutoff, kfeedback
ar wguide2 asig, xfreq1, xfreq2, kcutoff1, kcutoff2, kfeedback1, kfeedback2

ar streson asig, kfr, ifdbgain

ar cross2 ain1, ain2, ilen, iovl, iwin, kbias

ar phaser1 asig, kfreq, iord, kfeedback[, iskip]

ar phaser2 asig, kfreq, kq, iord, imode, ksep, kfeedback
ar flanger asig, adel, kfeedback [,imaxd]

ar harmon asig, kestfrq, kmaxvar, kgenfrq1, kgenfrq2, imode, iminfrq, iprd
ar distort1 asig [,kpregain, kpostgain, kshape1, kshape2]

AMPIEZZA E SPAZIALIZZAZIONE

kr rms asig [,ihp, iskip]
ar gain asig, krms [,ihp, iskip]
ar balance asig, acomp [,ihp, iskip]

ar dam ain, kthresh, icomp1, icomp2, irtime, iftime

ar dcblock asig [,igain]
xr limit xsig, nlow, nhigh
xr wrap xsig, nlow, nhigh
xr mirror xsig, nlow, nhigh
ar clip ain, imethod, ilimit[, iarg]

nr linen xamp, irise, idur, idec
nr linenr xamp, irise, idec, iatdec
nr envlpxxamp, irise, idur, idec, ifn, iatss, iatdec [,ixmod]
nr envlpxr xamp, irise, idur, idec, ifn, iatss, iatdec [,ixmod] [,irind]

a1,...,a4 pan asig, kx, ky, ifn [,imode] [,ioffset]

a1,a2[,a3,a4] locsig asig, kdegree, kdistance, kreverbsend

a1,a2[,a3,a4] locsend

a1,..., a4 space asig, ifn, ktime, kreverbsend [,kx, ky]

a1,..., a4 spsend

kr spdist ifn, ktime [,kx, ky]

aL, aR hrtfer asig, kAz, kElev, "HRTFcompact"

vbaplsinitidim, ils_amount, idir1, idir2,...

(N = 4, 8, or 16)
a1,...,aN vbapNasig, iazi, iele, ispread
a1,...,aN vbapNmove asig, ispread, ifld_amount, ifld1, ifld2, ...

vbapz inumchans, indx, asig, iazi, iele, ispread
vbapzmove inumchans, indx, asig, ispread, ifld_amount, ifld1, ifld2, ...

OPERAZIONI SU DATI DI ANALISI

HETRO - Fourier analysis for adsyn generator.

ar adsyn kamod, kfmod, ksmod, ifilcod

PVANAL - Fourier analysis for phase vocoder generators.
PVLOOK - Read and print out PVANAL file content info.
ar pvoc ktimpnt, kfmod, ifilcod [, ispecwp, iextractmode, ifreqlim, igatefun]
ar pvadd ktimpnt, kfmod, ifile, ifn, ibins [,ibinoffset, ibinincr, iextractmode, ifreqlim, igatefun]
kfrq,kamp pvread ktimpnt, ifile, ibin
pvbufread ktimpnt, ifile
ar pvinterp ktimpnt, kfmod, ifile, kfreqscale1, kfreqscale2, kampscale1, kampscale2, kfreqinterp, kampinterp
ar pvcross ktimpnt, kfmod, ifile, kamp1, kamp2, [ispecwp]

tableseg ifn1, idur1, ifn2 [,idur2, ifn3[...]]
tablexseg ifn1, idur1, ifn2 [,idur2, ifn3[...]]
ar vpvoc ktimpnt, kfmod, ifilcod[, ispecwp][, ifnmagctrl]

LPANAL - Linear predictive analysis for lpread/lpreson generators
krmsr, krmso, kerr, kcps lpread ktimpnt, ifilcod [,inpoles] [,ifrmrate]
ar lpreson asig
ar lpfreson asig, kfrqratio
lpslot islot
lpinterp islot1,islot2,kmix

CVANAL - Impulse response Fourier analysis for convolve operator

ar1[,...[,ar4]]] convolve ain, ifilcod [,ichan]

IL SISTEMA DI PATCHING ZAK

zakinit isizea, isizek
ir zir indx
kr zkr kndx
ar zar kndx
ar zarg kndx, kgain

ziw isig, indx
zkw ksig, kndx
zaw asig, kndx

zkcl kfirst, klast
zacl kfirst, klast

ziwm isig, indx [,imix]
zkwm ksig, kndx [,imix]
zawm asig, kndx [,imix]

kr zkmod ksig, kzkmod
ar zamodasig, kzamod

INGRESSO E USCITA

a1 in

a1, a2 ins

a1,...,a4 inq
a1,...,a6 inh
a1,...,a8 ino

a1,...,a16 inx
a1,...,a32 in32

a1 inch kchannel
inz kZA_indx

a1 soundin ifilcod [,iskptim] [,iformat]

a1, a2 soundin ifilcod [,iskptim] [,iformat]

a1,...,a4 soundin ifilcod [,iskptim] [,iformat]

a1[,a2[,a3,a4]] diskin ifilcod, kratio [,iskiptim] [,iwraparound] [,iformat]

outasig

outs	**asig1, asig2**
outs1/2	**asig**
outq	**asig1, asig2, asig3, asig4**
outq1/2/3/4	**asig**
outh	**asig1, asig2, asig3, asig4, asig5, asig6**
outo	**asig1, ..., asig8**
outx	**asig1, ..., asig16**
out32	**asig1, ..., asig32**
outc	**asig1[, asig2,....]**
outch	**kch1, asig1, kch2, asig2, ...**

outz kZA_indx

soundout asig, "soundfilename" [,iformat]

SOUNDFONTS

ifilhandle sfload "filename"

sfplist ifilhandle

sfilist	**ifilhandle**
sfpassign	**istartindex, ifilhandle**

ipreindex sfpreset iprog, ibank, ifilhandle, ipreindex

a1, a2 sfplay ivel, inotnum, xamp, xfreq, ipreindex [, iflag]
a1 sfplaym ivel, inotnum, xamp, xfreq, ipreindex [, iflag]

a1, a2 sfinstr ivel, inotnum, xamp, xfreq, instrNum, ifilhandle [, iflag]
a1 sfinstrm ivel, inotnum, xamp, xfreq, instrNum, ifilhandle [, iflag]

ACCESSO AI FILE

```
ihandle   fiopen "ifilename",imode
          fout   "ifilename", iformat, aout1 [, aout2, aout3,.... ,aoutN]
          foutk  "ifilename", iformat, kout1 [, kout2, kout3,....,koutN]
          fouti  ihandle, iformat, iflag, iout1 [, iout2, iout3,....,ioutN]
          foutir ihandle, iformat, iflag, iout1 [, iout2, iout3,....,ioutN]

       fin       "ifilename", iskipframes, iformat, ain1 [, ain2, ain3,.... ,ainN]
       fink      "ifilename", iskipframes, iformat, kin1 [, kin2, kin3,.... ,kinN]

fini   "ifilename", iskipframes, iformat, in1 [, in2, in3,.... ,inN]

       vincr     asig, aincr
       clear     avar1 [,avar2, avar3,...,avarN]

ilen   filelen   ifilcod
isr    filesr    ifilcod
ichnls filenchnls ifilcod
ipeak  filepeak  ifilcod, [ichnl]

       display   nsig, iprd [,inprds] [,iwtflg]
       dispfft   nsig, iprd, iwsiz [,iwtyp] [,idbouti] [,wtflg]

       print     iarg [,iarg,...]
       printk    itime, kval [,ispace]
       printk2   kval [,ispace]
       printks   "txtstring", itime, kval1, kval2, kval3, kval4

k1     readk     "ifilname", iformat, iprd [,interp]
k1,k2  readk2    "ifilname", iformat, iprd [,interp]

k1,k2,k3  readk3   "ifilname", iformat, iprd [,interp]
k1,…, k4  readk4   "ifilname", iformat, iprd [,interp]

       dumpk     ksig,   "ifilname", iformat, iprd
       dumpk2    ksig1, ksig2, "ifilname", iformat, iprd
       dumpk3    ksig1, ksig2, ksig3, "ifilname", iformat, iprd
       dumpk4    ksig1, ksig2, ksig3, ksig4, "ifilname", iformat, iprd
```

MESSAGGI DI ERRORE E DI AVVERTIMENTO DI CSOUND

Csound può visualizzare numerosissimi messaggi: alcuni di errore (e l'esecuzione spesso verrà interrotta), altri di avvertimento (*Warning*), altri che sono semplici comunicazioni. Non è possibile trattare tutti i casi, ma la tabella seguente può aiutare l'utilizzatore di Csound a comprendere ciò che sta accadendo e a prendere le misure opportune.

Nella tabella si usano i seguenti simboli:

<NOME>, <NOME1> sono parole o frasi
<NOMEFILE> è il nome di un file che viene letto o scritto
<X>, <N>, <M> sono numeri

Messaggio	**Traduzione**	*Spiegazione e rimedi*
/dev/audio: cannot do AUDIO_GETINFO	**/dev/audio: impossibile reperire AUDIO_GETINFO**	*Problema probabilmente dovuto ai driver della scheda audio*
/dev/audio: could not write all bytes requested	**/dev/audio: impossibile scrivere tutti i byte richiesti**	*Problema probabilmente dovuto ai driver della scheda audio, o a un guasto hardware*
/dev/dsp: could not write all bytes requested	**/dev/dsp: impossibile scrivere tutti i byte richiesti**	*Problema probabilmente dovuto ai driver della scheda audio, o a un guasto hardware*
ADSYN cannot load <NOME>	**ADSYN non può caricare <NOMEFILE>**	*Controllare percorso e nome del file hetro*
adsyn: not initialised	**adsyn; non inizalizzato**	*Mancano argomenti? Controllare percorso e nome del file hetro*
adsynt: freqtable not found	**adsynt: tabella delle frequenze non trovata**	*Creare la tabella, o controllarne la validità*
adsynt: wavetable not found	**adsynt: tabella non trovata**	*Creare la tabella, o controllarne la validità*
AIFF 3-byte samples not supported	**I campioni di suono a 3 byte non sono supportati da AIFF**	*Csound non supporta audio a 24 bit per i file AIFF*
AIFF does not support <NOME> encoding	**AIFF non supporta la codifica <NOME>**	*Csound non supporta la codifica richiesta per i file AIFF*
AIFF-C 3-byte samples not supported	**I campioni di suono a 3 byte non sono supportati da AIFF-C**	*Csound non supporta audio a 24 bit per i file AIFF-C*
alaw and ulaw not implemented here	**alaw e ulaw non implementati**	*Non richiedere sistemi di compressione alaw e ulaw*

alaw audio_in not yet implemented	**ingresso audio alaw non implementato**	*Non richiedere sistemi di compressione alaw e ulaw*
alaw not yet implemented	**alaw non implementato**	*Non richiedere sistema di compressione alaw*
Application Signature not pErF	**La firma dell'applicazione non è pErF**	
-b <N> probably too large, suggest <= 2048	**Il valore <N> del buffer è probabilmente troppo grande, si suggerisce <= 2048**	*Ridurre le dimensioni del buffer software*
-B <N> probably too large, suggest 1024	**Il valore <N> del buffer è probabilmente troppo grande, si suggerisce 1024**	*Ridurre le dimensioni del buffer*
... CONTINUA ...	***... CONTINUA ...***	***... CONTINUA ...***

BIBLIOGRAFIA ESSENZIALE

AA.VV., *MIDI Specifications*, MIDI International User's Group, Los Altos, CA., USA, 1983

Backus, J. *The Acoustical Foundation of Music*. New York, New York: Norton

Balena F., De Poli, Giovanni, "Un modello semplificato del clarinetto mediante oscillatore non lineare", in *Musica e tecnologia: industria e cultura per lo sviluppo del Mezzogiorno*, Quaderni di Musica/Realtà - UNICOPLI, 1987, Milano

Berry, R.W., "Experiments in Computer Controlled Acoustical Modelling (A Step Backwards?)", in *Proceedings of the 14th Computer Music Conference*, Feedback Papers, 1988, Köln, Germania

Bianchini, R. 1987. 'Composizione automatica di strutture musicali', in *I profili del suono*. Musica Verticale-Galzerano, 1987, Salerno

Bianchini, R. 1996. 'WCShell e i suoi software tools per la generazione e la modifica di partiture in formato Csound', in *La terra fertile, Atti del Convegno*. L'Aquila 1996

Boulanger, Richard (a cura di), *The Csound Book*, 2000, M.I.T. Press, Cambridge, Massachusetts, USA

Chadabe, Joel, *Electric Sound - The Past and Promise of Electronic Music*, Prentice Hall, 1997 Up.Saddle River, New Jersey

Chamberlin, Hal, *Musical Applications of Microprocessors*, Hayden Book Company, 1980, Hasbrouck Heights, NJ, USA

Chowning, John, "La sintesi di spettri acustici complessi mediante tecniche di modulazione di frequenza", in Henri Pousseur (a cura di), *La musica elettronica*, Milano 1975

Cott, J. 1973. *Stockhausen. Conversations with the Composer.* New York, NY, 1973

De Poli, Giovanni, "Tecniche numeriche di sintesi della musica", in *Bollettino LIMB n.1*, La Biennale di Venezia, 1981, Venezia

Dodge, Charles e Jerse, Thomas A., *Computer Music*, Schirmer Books, 1985, New York, NY, USA

Forin, Alessandro, "Spettri dinamici prodotti mediante distorsione con polinomi equivalenti in un punto", in *Bollettino LIMB n.2*, La Biennale di Venezia, 1981, Venezia

Gabor, D. (1947) "Acoustical Quanta and the Theory of Hearing", in *Nature*, Vol 159, N°4044

Holman, Tomlinson *Surround Sound 5.1 - Up and Running*, Focal Press - Butterworth-Heinemann, 2000, Boston, U.S.A.

Huber, David M. e Runstein, Robert E., *Manuale della registrazione sonora*, Hoepli, 1999, Mlano

Moles, A.(1969) "The Sonic Object", in *Information Theory And Esthetic Perception*, University. of Illinois Press, Urbana

Moorer, J.A., "Signal Processing Aspects of Computer Music", in J.Strawn (a cura di), *Digital Audio Signal Processing. An Anthology,* William Kaufmann Inc., Los Altos, Ca., 1985

Morresi, Nello, *Dispense di acustica applicata*, inedito

Pousseur, Henri (a cura di), *La musica elettronica*, Feltrinelli, 1976, Milano

Prieberg, Fred K. 1963. *Musica ex machina*, , Einaudi, 1963, Torino

Risset, Jean Claude, “Tecniche digitali del suono: influenza attuale e prospettive future in campo musicale”, in *Bollettino LIMB n.2*, La Biennale di Venezia, 1981, Venezia

Risset, Jean Claude e Wessel, David, “Indagine sul timbro mediante analisi e sintesi”, in *Bollettino LIMB n.2*, La Biennale di Venezia, 1981, Venezia

Roads, Curtis, “Automated Granular Synthesis of Sound”, in *Computer Music Journal*, 2(2), pagg. 61-62, 1978, M.I.T. Press, Cambridge, Massachusetts, USA

Roads, Curtis, “Granular Synthesis of Sound”, in Roads, Curtis e Strawn, John (a cura di), *Foundations of Computer Music*, The MIT Press, 1985, Cambridge, Massachusetts, USA

Roads, Curtis, *The Computer Music Tutorial*, The MIT Press, 1995, Cambridge, Massachusetts, USA

Roads, C., and Strawn, J. (ed.). 1985. *Foundations of Computer Music*. Cambridge, Massachusetts: The MIT Press

Seto, William W., *Teoria ed applicazioni di Acustica*, ETAS Libri, 1978, Milano

Spiegel, Murray R., *Manuale di matematica*, ETAS Libri, 1974, Milano

Smith, Julius O. III, "Discrete-Time Modeling of Acoustic Systems with Applications to Sound Synthesis of Musical In-struments", in *Proceedings of the Nordic Acoustical Meeting*, Helsinki, 1996

Strawn, John (a cura di), *Digital Audio Signal Processing*, William Kaufmann Inc., 1985, Los Altos, CA, USA

Tisato, Graziano, “Sintesi dei suoni vocali e del canto in modulazione di frequenza”, in *Musica e tecnologia: industria e cultura per lo sviluppo del Mezzogiorno*, Quaderni di Musica/Realtà - UNICOPLI, 1987, Milano

Truax, Barry, *Handbook for Acoustic Ecology*, 2nd edition (CD-ROM version CSR-CDR9901), version 1.1, Cambridge Street Publishing, 1999, Burnaby, B.C., Canada

Truax, Barry, “Real-Time Granular Synthesis with a Digital Signal Processor”, in *Computer Music Journal*, Vol.12, N°2, Summer M.I.T Press, 1988, Cambridge, Mass. pp.14-26

Vercoe, Barry, Media Lab MIT & Contributors, (John ffitch, Richard Boulanger, Jean Piché, & David Boothe, a cura di), *Csound Manual (Canonical Version 4.10)*, Media Lab - MIT, 1986-1992-2001, Cambridge, Massachusetts, USA

Wiener, N.(1964) "Spatio-Temporal Continuity, Quantum Theory, and Music", in *The Concepts Of Space And Time*, Boston Studies XXII, M.Capek (Ed.), D.Riedel, 1975, (1975), Boston, Mass., USA

Xenakis, Iannis, *Formalized Music*, Indiana U.Press, 1971, Bloomington, IN, USA

SITI INTERNET

1. SITI PRINCIPALI

2. SOFTWARE

3. UNIVERSITA', CENTRI DI RICERCA E ASSOCIAZIONI

Riccardo Bianchini
Milano, 1946

e-mail: riccardobianchini@tiscalinet.it
web.tiscalinet.it/rbianchini

Riccardo Bianchini ha studiato pianoforte, composizione e musica elettronica, e ingegneria al Politecnico di Milano. Dal 1974 ha insegnato Musica Elettronica nei Conservatori di Pescara e Milano, e dal 1987 è titolare della stessa cattedra al Conservatorio "Santa Cecilia" di Roma.

Ha tradotto in italiano molti importanti testi sulla musica, fra i quali *Lo stile classico* e *Le forme sonata* di C. Rosen.

Ha collaborato con numerose riviste (Rivista IB, Perspectives of New Music etc.).

Dal 1995 tiene corsi presso diverse Università e centri di Uruguay, Argentina e Cile.

È autore di diversi *software* musicali, fra i quali *Hypermusic* e *WCShell*.

Dal 1983 al 1991 ha collaborato con RAI-Radiotre per l'organizzazione e la presentazione di concerti e programmi di musica contemporanea.

Nel 1977 ha fondato a Milano l'*ensemble edgar varèse*, un gruppo da camera il cui repertorio spaziava dal Cinquecento veneziano alla musica contemporanea, e lo ha diretto in oltre venti concerti. Nel 1993 ha fondato un gruppo analogo, *Farfensemble*, tuttora in attività.

Le sue composizioni (orchestrali, vocali, strumentali, elettroacustiche, per TV e teatro) sono pubblicate e registrate da Edipan e BMG-Ariola, e sono state eseguite e/o radiotrasmesse in Europa, USA, Cuba, Argentina, Uruguay, Cile e Australia.

BIBLIOGRAFIA

Bianchini,R. 1973. "La nuova musica in Italia", in *Storia della musica Oxford-Feltrinelli, Vol.X*. Milano: Feltrinelli
Bianchini, R. 1976. "La musica elettronica", in *Rivista IBM*, 12. Milano
Bianchini, R. 1976. "Musica e letteratura (II)", in *Enciclopedia Feltrinelli Fischer.* Milano: Feltrinelli

Bianchini, R. 1976. “La musica contemporanea”, in *Musica 1,2,3,4,5*. Milano
Bianchini, R. 1976. “La musica informatica”, in *Musica Domani, 22*. Milano
Bianchini, R. 1985. *Computer Music: manuale di informatica musicale*. Inedito
Bianchini, R. 1987. “Composizione automatica di strutture musicali”, in *I profili del suono*. Salerno: Galzerano
Bianchini, R. 1996. “WCShell e i suoi *software tools* per la generazione e la modifica di partiture in formato Csound”, in *La terra fertile, Proceedings*. L’Aquila
Bianchini, R, a cura di Cristiano, C. *I territori di Montopoli e Bocchignano*. Montopoli di Sabina.
Bianchini, R. 1999. “La musica elettronica in Italia”, in *Azzurra, 4*. Córdoba: Istituto Italiano di Cultura

TRADUZIONI

1973. *Storia della musica Oxford-Feltrinelli, Vol.X*. Milano: Feltrinelli (traduzione di *The Oxford History of Music*)
Pousseur H., a cura di. 1975. *La musica elettronica.* Milano: Feltrinelli (traduzione)
Rosen, C. 1979. *Lo stile classico*. Milano: Feltrinelli (traduzione di *The Classical Style*)
Dick, R. 1979. *L’altro flauto*. Milano: Ricordi (traduzione di *The Other Flute*)
Rosen, C. 1984. *Le forme sonata*. Milano: Feltrinelli (traduzione di *Sonata Forms*)

DISCOGRAFIA

10 storie Zen, per flauto, clarinetto, vibrafono, viola, violoncello e pianoforte.
WNC Ensemble. PAN PRC S2062
Roèn, per flauto, clarinetto, fagotto, corno, violino, viola, violoncello e pianoforte
L’Artisanat Furieux Ensemble, dir. T. Battista. PAN CDC 3010
Klimt, per flauto, oboe, clarinetto, violino, viola, violoncello, pianoforte e nastro.
Romensemble, dir. F.E.Scogna RCA CCD 3001
Machu Picchu, per flauto, oboe, clarinetto, fagotto, 2 trombe, corno, trombone e nastro
Farfensemble, dir. R.Bianchini. ED0009

COMPOSIZIONI

haiku, (1976, 10:00), pianoforte e nastro
Mirror, (1976, 05:00), flauto e pianoforte
Collettivo II, (1976, 05:00), flauto, oboe, clarinetto, fagotto, violino, violoncello e pianoforte. EDIPAN
Due racconti, (1979, 05:00), 2 flauti, 2 clarinetti, fagotto, viola e pianoforte. EDIPAN

La nave bianca (Preludio), (1980, 05:00), orchestra da camera. EDIPAN
La nave bianca (musiche di scena), (1980, 25:00), orchestra da camera e coro maschile
Roèn, (1982, 05:00), flauto, clarinetto, fagotto, corno, violino, viola, violoncello e pianoforte. EDIPAN
Riyàr, (1982, 05:00), ottavino, flauto in DO, flauto in SOL, flauto basso (1 flautista) EDIPAN
Sedrùna, (1982, 5:00), pianoforte a quattro mani. EDIPAN
Tre quadri immaginari, (1983, 10:00), arpa. EDIPAN
Quattro canti: 1. "Di più cupi sentieri" (D.Villatico), 2. "La tierra que era mía" (J.G,.Durán), 3. "I have done" (J.London), 4. "Im wunderschönen Monat Mai" (R.M.Rilke), (1980-1988, 12:00), soprano e pianoforte. EDIPAN
6 Preludi, (1980-1984, 11:00), pianoforte. EDIPAN
Due fogli d'album, (1985, 02:00), flauto e pianoforte
Foglio d'album, (1985, 1:30), violino e pianoforte
La principessa senza tempo, (1985, 14:10), flauto e nastro. EDIPAN
Alias, (1985, 5:00), 2 oboi e fagotto
Chanson d'aube, (1986, 6:00), 4 trombe, 4 corni, 4 tromboni
Rosengarten, (1986, 05:00) violino e orchestra
Our Faust, (1986, 17:00), clarinetto, trombone, contrabbasso e live electronics BMG-Ariola
Arsól, (1987, 10:00), computer in tempo reale e nastro quadrifonico
Divertimento, (1988, 8:20), 13 strumenti e nastro quadrifonico. EDIPAN
Somanón, (1989, 8:00), 11 archi. EDIPAN
Preuss (1989, 16:20), violino, violoncello e nastro. BMG-Ariola
Fànes (1989, 8:00), flauto e flauto in SOL (1 flautista)
Alberei (1990, 6:00), quartetto di saxofoni. BMG-Ariola
Saluto a Pablo (1990, 2:00), soprano, flauto e clarinetto
Cuando sonó la trompeta (1990, 7:30), soprano e nastro. BMG-Ariola
Klimt (1991,10:50), flauto, oboe, clarinetto, violino, viola, violoncello, pianoforte e nastro. BMG-Ariola
Chanson d'aube II (1991,5:00), 2 oboi, 2 clarinetti, 2 corni e 2 fagotti
Tre ricercari (1993, 5:00), 2 trombe, corno e trombone
Machu Picchu (1993, 14:00), flauto, oboe, clarinetto, fagotto, 2 trombe, corno, trombone e nastro
Poche note… per Enzo Porta, (1994, 2:00), violino
6 Preludi (II quaderno), (1994, 11:00), pianoforte
Il contrabbasso poteva non esserci, (1995, 2:30), 2 flauti, oboe, pianoforte e quartetto d'archi
Naste, (1995, 2:00), flauto e violoncello

Howl, (1995, 6:30), voce femminile o maschile e nastro
I dannati della terra, (1996, 28:00), attore, soprano, flauto, percussioni, nastro e proiezioni di immagini
Ghe Xe, (1997, 5:00), flauto in SOL e pianoforte
Aria di Albertine (da "Doppio sogno"), (1997, 4:30), soprano, flauto, oboe, clarinetto, fagotto, pianoforte e quartetto d'archi
Canciones para las estrellas, (1997, 6:30), nastro
Canciones para las estrellas, (1997, 8:00), pianoforte e nastro
Los pájaros del sueño, (1998, 9:00), clarinetto e nastro
Montevideana, (1999, 5:00), nastro (paesaggio sonoro)
How Deep the Sea, (1999, 5:30), *Jazz Band* (4 sax, 3 trombe, 2 tromboni, pianoforte, basso e batteria)
Alle Menschen werden Brüder, (1999, 7:40), recitante, violino e nastro quadrifonico
Sottovoce, (2000), nastro
Para parar las aguas del olvido, (2001, 5:30), flauto, clarinetto (anche clarinetto basso), trombone e pianoforte
L'homme armé, (2001, 20:00), coro a 8 voci, *live electronics* e nastro

TRASCRIZIONI E REVISIONI

A.Gabrieli, *Ricercar nel duodecimo tono*, (1977), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
G.Gabrieli, *Quattro canzoni per sonar a quattro*, (1977), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
A.Willaert, *Ricercar X*, (1977), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
J. da Modena, *Ricercar III*, (1977), tromba, corno e trombone
Anonimi Francesi, *Suite di danze*, (1977), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
H.Pousseur, *Icare apprenti*, (1977), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
F.Schubert, *4 Ländler*, (1993), quintetto di fiati
F.Schubert, *4 Ländler*, (1994), flauto, oboe, clarinetto, fagotto e quartetto d'archi
F.Schubert, *Deutsche Tänze*, (1994), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
W.A.Mozart, *Musiche di palcoscenico da "Don Giovanni"*, (1994), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
H.Purcell, *Suite*, (1995), tromba solista, flauto, oboe, clarinetto, fagotto, corno e trombone

J.Lennon, P.McCartney, *Eleanor Rigby*, (1995), flauto, corno inglese, clarinetto, fagotto, tromba, corno e trombone
J.Lennon, P.McCartney, *Penny Lane*, (1995), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
J.Lennon, P.McCartney, *Yesterday*, (1995), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
J.Lennon, P.McCartney, *Girl*, (1995), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
J.Lennon, P.McCartney, *Lady Madonna*, (1995), flauto, oboe, clarinetto, fagotto, tromba, corno e trombone
Bela Bartók, *Danze Rumene*, (1993), quintetto di fiati
Bela Bartók, *Danze Rumene*, (1994), flauto, oboe, clarinetto, fagotto e quartetto d'archi
K.Weill, *Songs* (1996), soprano, flauto, oboe, clarinetto, fagotto e quartetto d'archi

photo by Chris Bitten

Alessandro Cipriani
Tivoli (Rm), 1959

e-mail a.cipriani@edisonstudio.it
www.edisonstudio.it
www.cnimusic.it

Diplomato in Composizione (G.Bizzi) e in Musica elettronica (R.Bianchini) al Conservatorio S.Cecilia di Roma, ha approfondito i suoi studi con Barry Truax in Canada presso la Simon Fraser University.
Ha collaborato per dieci anni con l'artista visiva Alba D'Urbano realizzando la parte musicale di 4 video e 4 videoistallazioni sonore. Si è dedicato successivamente alla produzione di pezzi per strumenti e nastro interessandosi al rapporto fra presenza ed assenza del corpo nel rito dell'esecuzione, dal 1994 ha cominciato a lavorare su pezzi intermediali (musica, video, poesia) e dal 1999 si dedica ad un progetto su canti di tradizione orale e musica elettroacustica documentato nel suo nuovo cd per la CNI "Al Nur". Recentemente ha realizzato alcune colonne sonore per film-documentari in cui il senso viene veicolato in maniera consistente anche attraverso il suono e in cui ambienti sonori e musica, elaborati al computer, si fondono e si scambiano di ruolo .
Alcuni dei suoi lavori sono stati selezionati e menzionati nei concorsi di Bourges (Fr) e Newcomp (USA), all'International Computer Music Conference '94 e '95 e '99 (Danimarca ,Canada., Cina), Discoveries (U.K.), Colloquio di Informatica Musicale, International Symposium on Electronic Arts '95 (Quèbec, Can.) etc. Suoi pezzi sono stati diffusi da numerose radio nazionali e sono stati eseguiti in numerosi festival in Europa, Nordamerica, Sudamerica e Cina, fra cui Inventionen (Berlino), EMS Stoccolma) Synthèse (Bourges), Engine 27 (New York), Festival di Ravenna etc..
Alcuni brani sono pubblicati su 2 Compact Disk dell'ICMC '95 e '99 e dell'Edipan. Ha vinto il 'Government of Canada Award 1995-96' e il Main Prize al Concorso "Musica Nova" (Praga).
Cipriani è titolare della Cattedra di Musica Elettronica all'Istituto Musicale Pareggiato 'V.Bellini' di Catania da vari anni. Ha tenuto corsi e conferenze presso l'Accademia di Santa Cecilia a Roma e in varie Università italiane , canadesi e degli Stati Uniti.
Ha pubblicato articoli di analisi musicale e teoria sviluppando un'ipotesi sulla nascita di una tradizione elettroacustica (Musica/Realtà, Bollettino GATM, atti ICMC, Colloquio di Informatica Musicale, La Terra Fertile, Aperture, etc.). E' uno dei soci fondatori di Edison Studio (Roma).

BIBLIOGRAFIA

Cipriani, A. 1993 "Visibili..." in *Atti del X Colloquio di Informatica Musicale*, LIM-DSI Univ. degli Studi di Milano, Milano, pp.404-6

Cipriani A. 1993 *Due tesi complementari sulle due versioni di Kontakte di K.Stockhausen* - Tesi di Diploma in Musica Elettronica - Conservatorio di Musica S.Cecilia - Roma

Cipriani A. 1995 "Towards an electroacoustic tradition?" in *Proceedings of the International Computer Music Conference*, ICMA, Banff, pp. 5-8

Cipriani A. 1995 "Problems of methodology: the analysis of *Kontakte*" in *Atti del X Colloquio di Informatica Musicale*, AIMI , Bologna, pp. 41-44

Cipriani A. 1996 "Verso una tradizione elettroacustica? Appunti per una ricerca" in *Musica/Realtà* N°49 Marzo LIM Lucca pp.18-24

Cipriani A. 1996 "Tradizione orale, tradizione scritta, tradizione elettroacustica" in *Atti del II Convegno La Terra Fertile - Incontro Nazionale di Musica Elettronica* - Conservatorio di Musica "A.Casella", L'Aquila

Bianchini R.- Cipriani A. 1998 *Il Suono Virtuale*, Contempo, Roma

Cipriani A. 1998 "*Kontakte* (Elektronische Musik) di K.Stockhausen: genesi, metodi, forma" in *Bollettino G.A.T.M*. anno V, n.1 GATM Univ. Studi Bologna

Cipriani A. 1998 "Musica e Internet: arte come esperienza, arte come codice" in *Aperture* n.5, Roma

Bianchini R.- Cipriani A. 2000 *Virtual Sound*, Contempo, Roma

Cipriani A. "Energie Elettroniche" in QsQs anno III numero 16, CNI , Roma 2001

DISCOGRAFIA

A.Cipriani
QUADRO
per quartetto d'archi e nastro
Incluso nel CD
International Computer Music Conference '95 – PRCD1600

A.Cipriani-S.Taglietti
IL PENSIERO MAGMATICO
per nastro, piano, percussione e coro misto
EDIPAN – PAN CD 3059

A.Cipriani
AL NUR (La Luce)
per canto islamico, zarb, daf e nastro
Incluso nel CD
International Computer Music Conference '99 – PRCD2000

A.Cipriani
AL NUR
CD monografico
Compagnia Nuove Indye - CNDL 13172

CATALOGO DELLE OPERE

#1, #2, #3
Kreis
(1987-1991)
tre opere video di Alba D'Urbano
con musica originale di Alessandro Cipriani
(finalista al Video Festival di Locarno)
(1987-1991)

#4
Kreis: la piazza
video installazione sonora
(in collaborazione con Alba D'Urbano)
realizzata all' EASA - Berlin Kulturstadt Europas - Esplanade Berlino Ago 1988

#5
Circolo Vizioso
video installazione sonora
(in collaborazione con Alba D'Urbano)
realizzata per la prima volta all' Internationaal Audio Visueel Experimenteel Festival 1989

#6
Circoscritto
video installazione sonora
(in collaborazione con Alba D'Urbano) realizzata
al Centro di Video Arte (palazzo dei Diamanti) di Ferrara nella mostra "POLISET"
Dic.1991

#7
Luce di due soli
per pianoforte, vibrafono e nastro (25'25")
(1991)
(in collaborazione con Giovanni Bietti)
Finalista al 1991 NEWCOMP COMPUTER MUSIC COMPETITION (USA) -
Prima Esecuzione 29° Festival di Nuova Consonanza - Roma
(Pf. G.Bietti - Vib. F.Cecilia)

#8
Visibili
per due violini e nastro (8'30")
(1992)
Menzione al "21e CONCOURS INTERNATIONAL DE MUSIQUE ELECTROACOUSTIQUE" di BOURGES (Francia) 1993
Selezionato all'INTERNATIONAL COMPUTER MUSIC CONFERENCE 1994 (Aarhus - Danimarca) e al X Colloquio di Informatica Musicale - Milano
Prima Esecuzione XV Festival Musica Verticale - Roma (D.Conti- A.Scolletta)

#9
Quadro
per quartetto d'archi e nastro (10'30")
(1993)
Prima esecuzione 18° Cantiere Internazionale d'Arte di Montepulciano (Quartetto Arianna)
CD INTERNATIONAL COMPUTER MUSIC CONFERENCE 1995

#10
Terra Fluida
(1991 - 1994)
Video di Alba D'Urbano con musica originale di Alessandro Cipriani
Main Prize al Concorso Internazionale di Musica Elettroacustica "Musica Nova"
della Radio-Televisione della Repubblica Ceca (Praga)
Prima Esecuzione - Musica Nova
Praga, 19 Dic. 1996

#11
Recordare
per flauto a becco basso e contrabbasso e nastro (12'24") (1994)
Prima esecuzione 3 Nov '94 - Musica Verticale/Progetto Musica '94
Goethe Institut - Roma (flauti a becco Antonio Politano)
Finalista al 25eCONCOURS INTERNATIONAL DE MUSIQUE
ELECTROACOUSTIQUE" di BOURGES (Francia) 1997
Selezionato all' XI Colloquio di Informatica Musicale - Bologna 95

#12
L'Acqua, il Musico, Lo Specchio
dialogo scenico musicale di Alessandro
Cipriani e Giovanni Bietti per due attori, 6 musicisti, video e nastro magnetico (1'00"ca.) (1993-94)
Prima Esecuzione 3 Dicembre '94 - Progetto Musica '94 (Spazi Aperti) Acquario - Roma

#13
In Memory of a Recorder
per nastro magnetico (15' 45") (1994)
Selezionato all'International Symposium on Electronic Arts 95
(Montreal) e "Discoveries '96" (Aberdeen - Scozia)
Prima Esecuzione Festival Animato/Progetto Musica '95 Sala Uno - Roma

#14
Il Pensiero Magmatico
per nastro magnetico, pf. , percussioni e coro misto (dur. 53 minuti) (1995-96)
(in collaborazione con Stefano Taglietti, testi di Bizhan Bassiri)
Prima Esecuzione 18 Ottobre 1996 Musée FRC Le Creux de l'Enfer Centre d'Art Contemporain Thiers Francia
CD Edipan (PAN CD 3059)

#15
Still Blue
(1996)
Homage to Derek Jarman
per nastro magnetico, pianoforte, violoncello e sax
Prima Esecuzione Freon Ensemble - Progetto Dyonisos
29 Novembre '96 - Acquario Romano - Progetto Musica '96

#16
Pensiero Magmatico
(1997)
per video e nastro magnetico
Video di Bizhan Bassiri
Musica Alessandro Cipriani
Prima esecuzione Galleria Miscetti
Marzo 1997 Roma

#17
Quem quaeritis non est hic
(1997)
Istallazione sonora per nastro magnetico in quadrifonia
prima esecuzione Istallazione Alba D'Urbano "Quem quaeritis..."
Kassel Luglio 1997

#18
Still Blue
(Homage to Derek Jarman)
The Video
(1998)
Video di A.Cipriani, S. Di Domenico e G.Latini
Musica di Alessandro Cipriani
Prima esecuzione Nuova Consonanza - Rome
Selezionato a “CORTO CIRCUITO ’98 ” European Festival of Audio-Visual Communication - Napoli

TRILOGIA DEL CANTO RELIGIOSO

#19
Aqua Sapientiae / Angelus Domini
(1996-2000)
per due cantori di canto gregoriano e nastro magnetico
prima esecuzione Musica Verticale/Progetto Musica '96
9 Dicembre '96

#20
Al Nur (La Luce)
(1997-2000)
per un cantore di canto islamico, percussioni persiane nastro magnetico e video
prima esecuzione 12 Novembre 1997
Musica e Scienza /Progetto Musica '97 Goethe Institut - Roma
Pubblicato su: CD INTERNATIONAL COMPUTER MUSIC CONFERENCE 99

#21
Mimahamakim
(1999)
per canto ebraico (4 voci femminili)e nastro

#.22
Net
per nastro
(2000)

#23
Reflection of the Moon over two Springs / Into the Light
per Er-hu, Gu Zheng e nastro
(2000)

#24
Colonna sonora per il Film Documentario
Al Nur (La Luce)
Regia di S.Di Domenico,G.Latini, M.Rovetto
(Tilak Film 2000)

Selezione "Elettroshock -30 anni di video in Italia"
Edizioni musicali CNI

#.25
Colonna sonora per il Film Documentario
Il Ritorno di Tuuli
Regia di S.Di Domenico,G.Latini, M.Rovetto
(Navert-Lang-Tilak 2001)
Edizioni musicali CNI

#.26
Colonna sonora e co-sceneggiatura sonora per il Film Documentario
Lorenza - In the World of Silence
Regia di G.Latini
(Forma Digitale-Videodream 2001)
in collaborazione con il British Film Institute di Londra
Edizioni musicali CNI

(I pezzi n.19-20-21-22-23 sono pubblicati sul CD "Al Nur" - CNI CNDL13172)

INDICE ANALITICO

F

G

H

I

K

N

O

P

Q

R

S

T

U

V

W