

Alessandro Petrolati
SintGran - SINTESI GRANULARE A FLUSSI INDIPENDENTI

Laboratorio Elettronico per la Musica Sperimentale
Conservatorio di Musica G. Rossini, Pesaro

INTROITO

SintGran è una patchwork di Csound in grado di generare n -flussi indipendenti su fronte stereofonico all'interno di un singolo strumento, l'implementazione è basata sulla filosofia della deviazione di flusso (*if...kgoto*).

Il file orchestra è composto di due strumenti, uno per il controllo a macrolivello dei parametri di sintesi (Score o realtime-midi) e l'altro per la generazione dei flussi (voci).

In GSC4, programma scritto da Eugenio Giordani, la generazione del grano e quindi la manutenzione di phase, delay, frequenza, ecc.. avviene all'interno di un tempo che si aggiorna a velocità i -time, dopodiché il flusso del programma si interrompe e reinizializza dando origine ad un riaggiornamento delle variabili i -time. Ogni strumento implementa un flusso e quindi avremo n -flussi uguale a n -strumenti più naturalmente quello di controllo di tutte le voci.

Poiché in uno strumento che preveda la asincronicità di più flussi non è possibile utilizzare la reinizializzazione in quanto l'interruzione di un flusso interrompe anche gli altri, l'unico modo per risolvere il problema è quello di ricreare, attraverso l'uso degli operatori logico-matematici, quei moduli che già si trovano nella libreria di csound, modificandoli al proprio uso.

Come esempio: il modulo *oscili* che nel parametro di phase accetta solo valori i -time e che in GSC4 si occupa della lettura della tabella audio o forma d'onda prototipale, qui non è stato possibile utilizzarlo in quanto il parametro di phase accetta soltanto valori i -time, e dovendo invece gestire la phase a velocità kr non si poteva mantenerla dando origine ad una mera modulazione in ampiezza più che sintesi granulare.

Questa strutturazione, che utilizza principalmente operatori primitivi, offre innumerevoli possibilità, si crea "l'oggetto" facendolo su misura a seconda dell'esigenza, e soprattutto, dato che si tratta di una "forma aperta" cioè non inscatolata in un modulo csound, offre al compositore-musicista la possibilità di essere manipolato, modificato, ampliato a seconda dell'uso che se ne intenda fare al momento. Inoltre la manutenzione della sintesi avviene con solo 2 strumenti, tornando utile nel caso in cui si decida di inserire questo algoritmo all'interno di uno più complesso.

STRUTTURAZIONE GENERALE

In linea di massima la generazione sia del grano che del frammento audio da involuppare è affidata a dei phasori che si muovono all'interno di tabelle (*vedi figura1*).

I phasori sono stati creati in questo modo: $k1 = k1 + kincr$, dove $kincr$ tiene conto dell'unità di tempo di aggiornamento cioè $1/kr$, e dell'unità di tempo desiderato $1/tempodesiderato$, poiché viene esplicitato in millesimi di secondo e passato dallo strumento di controllo, sarà $kincr=1/((gkdur*.001)*(1/kr))$. Del tutto simile sono i calcoli degli altri incrementi. Questi valori vengono aggiornati ogni volta che il flusso passa per la zona *reset* e restano immutati per tutta la generazione del grano. Quando questi ($k1$ e/o $kd1$) sono arrivati ad 1, ricondizionano il flusso facendolo passare per i punti desiderati. Poiché l'incremento del grano ($k1$) e del delay ($kd1$) vengono caricati contemporaneamente, d'altronde come gli altri, ma in questo caso il delay deve essere cronologicamente dopo la generazione del grano, si è ricorsi a un'espressione (*konoff*) che tiene bloccato il valore dell'uno mentre l'altro è libero e viceversa.

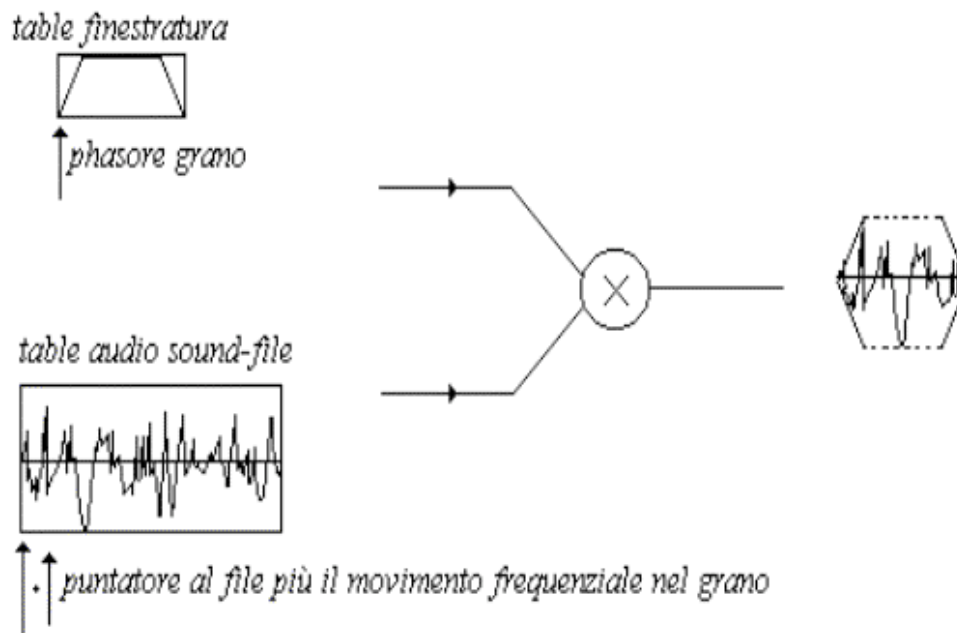


figura 1

L'idea è stata quella di creare una zona "franca" (*reset*) la quale viene saltata dal flusso del programma fin quando una tale condizione-x non sarà verificata. Ogni flusso è strutturato in tre zone contrassegnate da tre etichette, le interruzioni di questo sono a velocità kr essendo questa anche la velocità di aggiornamento dei phasori.

reset:

....

.... zona "franca", il flusso passa qui ogni volta che la generazione del grano

.... precedente sia avvenuta

grano:

....

.... il phasore si muove da 0 a 1 e legge i punti della tabella che contiene la finestratuta

....

delay:

....

.... un altro phasore si muove da 0 a 1 e indica il tempo di durata delay

....

Come intestazione vengono messe delle condizioni utilizzando la filosofia degli

if kgoto label

L'ALGORITMO

Quando il programma viene inizializzato, si caricano in memoria da subito certi valori che sono:

irate init 1/*kr* ;velocità di aggiornamento dei phasori

iwin = *p4* ;numero funzione per la table che identifica il ;tipo di finestrata

ifun = *p5* ;numero funzione per la table che identifica il ;tipo di sound-file

isample = *p6* ;valore di setup che identifica il numero di ;campioni del sound-file

isamprate = *p7*/*sr* ;valore di setup (*p7*) che identifica la ;frequenza di campionamento originale diviso la ;frequenza di campionamento dell'orchestra, ;serve come riscalatura per l'incremento di ;frequenza al fine di decorrelarlo dalla ;velocità audio dell'orchestra.

isize = *ftlen*(*ifun*);numero campioni della funzione che contiene ;il sound-file

idir = *p8* ;valore che indica la direzione di lettura dei ;campioni all'interno del grano (o +1 o -1)

Di seguito viene riportata l'implementazione per la generazione di un flusso; proviamo a seguire un paio di giri del programma:

instr 2

k1 init 0

kd1 init 0

k2 init 0

kd2 init 0

iwin = *p4*

ifun = *p5*

isample = *p6*

isamprate = *p7*/*sr*

irate init 1/*kr*

isize = *ftlen*(*ifun*)

idir = p8

if kd1 != 0 kgoto delay1

if k1 == 0 kgoto reset1

if k1 < 1 kgoto grano1

reset1:

k1 = 0

kd1 = 0

kpt1 = 0

kdur1 = 1/(gkdur1*.001)

kdelta1 = kdur1*irate

kdel1 = 1/((gkdel1*.001)+0.0001)

kdelay1 = kdel1*irate

kph1 = gkph1/(isize/isample)

kincr1 = ((ksmps/isize)*gkfreq)* idir

grano1:

k1 = k1 + kdelta1

konoff1 = (k1 >= 1 ? 0 : 1)

kgrano1 = k1*konoff1

delay1:

kd1 = kd1+(kdelay1*(1-konoff1))

if kd1 >= 1 kgoto reset1

kamp1 tablei kgrano1,iwin, 0, 1

kpt1 = kpt1 + (kincr1*isamprate)

akpnt1 interp kph1+kpt1

```
asig1 tablei akpunt1, ifun,1, 0, 1
```

```
out asig1*kamp1
```

```
endin
```

la prima condizione è falsa perché appena sopra il valore in questione $kd1$ è stato inizializzato a valore 0, quindi si passa alla seconda che invece è vera. Il flusso a questo punto salta a *reset1* dove si assegnano ai phasori della durata grano ($k1$), durata delay ($kd1$) e frequenza ($kpt1$) il valore 0, di seguito vengono calcolati gli incrementi dei phasori come sopra descritto, nel caso del puntatore al file $kph1$, viene direttamente calcolato il valore utile cioè i punti della tabella riempiti realmente dai campioni audio, il valore normalizzato della globale $gkph1$ diviso il numero che risulta dalla ulteriore divisione tra la lunghezza tabella e il numero reale dei campioni audio $gkph1/(isize/isample)$. Nel calcolo della frequenza viene moltiplicata la frequenza naturale per la velocità di aggiornamento $ifrnt*irate$, ma questa formula può essere semplificata da $(ksmps/isize)$ poiché nella prima il risultato teneva conto sia di kr che di sr e quindi il $ksmps$ funziona da ago della bilancia; il tutto riscalato per un numero che viene fornito dalla globale, che trasporta il valore di frequenza in relazione armonica (1=originale, 2=ottava sopra, 3/2=quinta ecc...) e ancora moltiplicato per un numero (che nello score si esplicita o +1 o -1) che fa cambiare di segno l'incremento, dando origine a un movimento inverso del phasore per la frequenza naturalmente il numero (*idir*) può essere anche una variabile kr , ma tenendo conto che il cambio tra -1 e +1 deve avvenire in maniera impulsiva, con lo scarto di un campione.

N.B. nella tablei che si occupa della lettura audio, deve essere abilitato con 1 l'ultimo parametro (*wraparound*).

Come seguito il flusso procede ed arriva a *grano1*

a questo punto mentre $k1$ comincia a muoversi da 0 a 1 riscalato automaticamente per i punti della tabella, essendo abilitato il parametro opzionale dopo la funzione con 1, la condizione ($konoff1 = k1 > 1 ? 0 : 1$), tiene congelato in quanto moltiplica ad ogni passaggio per 0 il valore di incremento del delay ($kdelay1*(1-konoff1)$), di seguito il phasore per la frequenza $kpt1$ si muove con velocità, data appunto dal calcolo dell'incremento in relazione alla frequenza, per i punti del sound-file, tenendo conto del puntatore al file (offset) fornitogli direttamente dalla globale e congelato a valore kr nella parte *reset*, tutto viene interpolato a velocità audio al fine di non compromettere la frequenza naturale di lettura tabella, generati i primi campioni audio ($ksmps$) il programma rivaluta le condizioni in testa, trova di nuovo la prima falsa, ma adesso anche la seconda in quanto essendo $k1$ diverso da 0, passa alla terza e trovandola verificata **salta direttamente a grano1**, qui si ritrovano i valori lasciati e si continua fino quando la terza condizione non si avvererà quindi il flusso verrà linearmente ripristinato dando origine ad una nuova lettura e calcolo degli incrementi quindi durata grano, delay, frequenza e puntatore al file (phase generale).

Facciamo conto che, riprendendo l'esempio precedente, il delay fosse stato diverso da 0 e ricominciamo dalla sezione *grano1*:

l'espressione $konoff1 = (k1 >= 1 ? 0 : 1)$ verifica la condizione prima, in ordine cronologico, rispetto al terzo *if....* posto in testa, quindi a questo punto azzerà il valore del phasore $k1$ in quanto lo moltiplica per 0, ma libera il valore di incremento del delay ($kdelay1$) in quanto lo moltiplica per $(1-konoff1)$. Adesso è il phasore del delay che gestisce il flusso del programma, questo non si muove per nessun punto della tabella in quanto non necessario dato che il valore di ampiezza del delay deve essere 0.

la condizione *if kd2 >1 kgoto reset1* riporterà il flusso a *reset*, ma fin quando questa non sarà verificata il flusso ripartirà dagli *if....* in testa e quindi a questo giro troverà vera la prima condizione che farà **saltare il flusso direttamente a delay1**.

CONCLUSIONE

I prossimi sviluppi del programma potrebbero essere:

1)Attualmente SintGran può leggere sia funzioni prototipali di forme d'onda, sia sound-file, ma si devono utilizzare coppie di program-file (*.sco *.orc) diverse, oppure modificare l'ultima riga di ogni flusso sostituendo *tablei* con un modulo *oscili* prima di fare girare il programma. Questo si poteva già ovviare inserendo entrambi i moduli in ogni flusso e condizionarlo adeguatamente indirizzandolo all'una o all'altra strada, ma non mi sembrava in linea con la filosofia del sistema oltre ad appesantirlo notevolmente.

2)Modificare lo strumento di controllo governandolo con funzioni lette esternamente da eventi midi.

3)Randomizzare la direzionalità di lettura all'interno del grano.

4)Randomizzare il numero per la funzione audio da granulare, questo è possibile con i nuovi aggiornamenti di csound 3.47 utilizzando i nuovi moduli (*tableikt ecc..*).

5)Inserire lo strumento di controllo direttamente dentro quello per la generazione voci, arrivando quindi allo stesso risultato di gestione della sintesi con solo uno strumento.