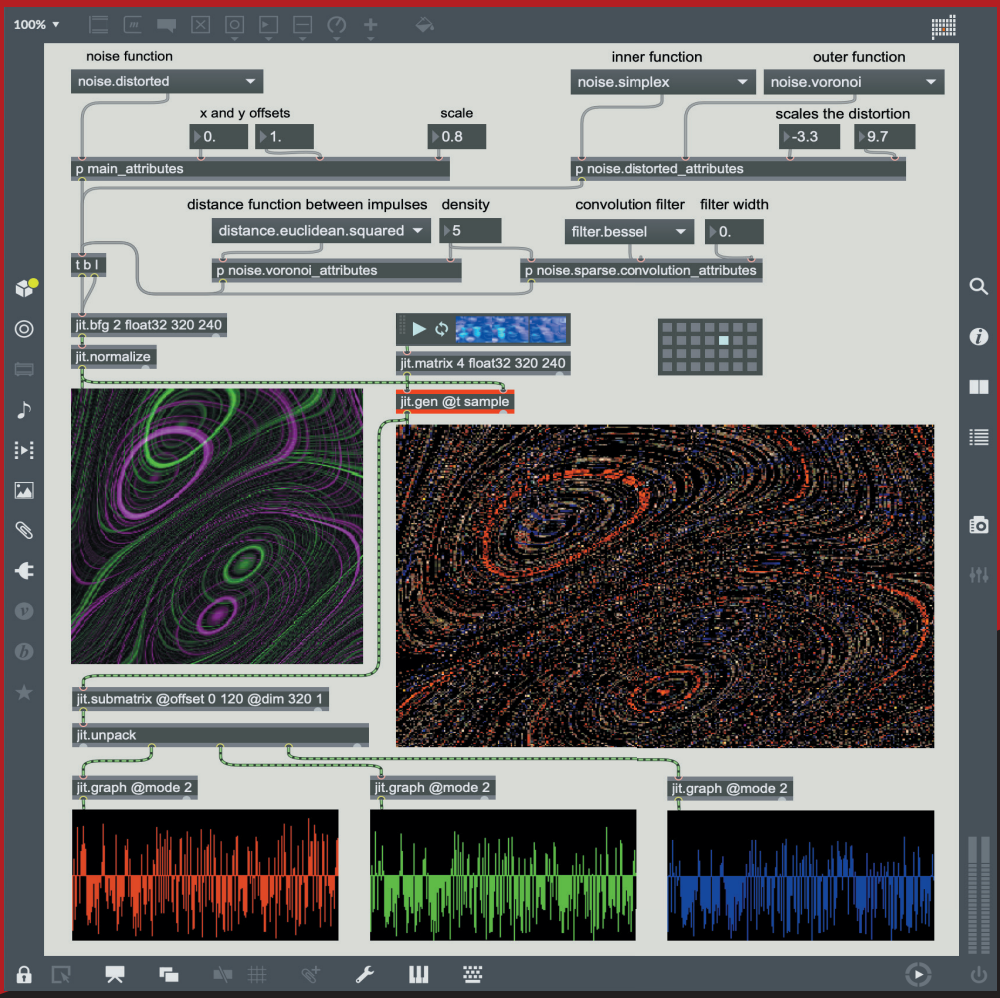
Alessandro Cipriani • Maurizio Giri



Musica Elettronica e Sound Design

Teoria e Pratica con Max 8 • volume 3

ConTempoNet

Alessandro Cipriani • Maurizio Giri

Questo è un estratto del libro:

MUSICA ELETTRONICA E SOUND DESIGN

Teoria e Pratica con Max - Volume 3

per maggiori informazioni:

www.contemponet.com

CIPRIANI A. - GIRI M.
MUSICA ELETTRONICA e SOUND DESIGN
Teoria e Pratica con Max
Vol. 3
ISBN 978-88-99212-22-3

Prima edizione 2021

Realizzazione figure: Maurizio Refice

Contemponet
e-mail posta@contemponet.com
posta@virtual-sound.com
URL: www.contemponet.com
www.virtual-sound.com

Questa pagina è intenzionalmente lasciata in bianco

INDICE

PREFAZIONE
Carmine-Emanuele Cella

Scrivere la prefazione al terzo volume di *Musica elettronica e sound design* è, in certo modo, esercizio pleonastico.

Il monumentale lavoro di Alessandro Cipriani e Maurizio Giri è, de facto, il riferimento per fare musica elettronica oggi e non ha bisogno di alcuna prefazione. Non c'è, a mia conoscenza, un corso di musica elettronica in Italia che non utilizzi i loro testi. Non solo: vari istituti all'estero ne fanno uso, in qualche forma. Io stesso, al Center for New Music and Audio Technologies (CNMAT) dell'università di Berkeley, faccio spesso ricorso ai primi due volumi per trovare esempi e strategie utili per spiegare un concetto.

Spiegare, infatti, è il disegno ultimo dei due autori. Non mancano, al mondo, libri che cercano di mostrare l'erudizione di chi li scrive. Più rari, invece, sono quei libri che si preoccupano di chi li legge, di accompagnare i lettori in un viaggio che li lascerà cambiati. I testi scritti da Cipriani e Giri fanno parte di questa rara categoria: sono testi che *spiegano*.
Con il loro stile piano, ma mai banale, gli autori accompagnano il lettore passo dopo passo, attraverso concetti complessi che vengono scomposti in piccoli pezzi, facili da capire. L'approccio adottato è enciclopedico: nell'arco di più volumi, vengono trattati tutti i concetti chiave della disciplina. Pagina dopo pagina, gli autori pongono le fondamenta di una materia eterogenea quale è la musica elettronica. Lo fanno in un modo sistematico e compassato, che a tratti riporta alla mente lo stile aristotelico.

Pleonastico, dunque, è fare una prefazione a un lavoro del genere. Mi sarà, tuttavia, utile provarci.

Anche questo terzo volume segue la collaudata formula *teoria/pratica*: dopo una presentazione dei concetti coinvolti, ne viene mostrata la realizzazione *practica* o, come si dice spesso oggi, l'implementazione. Questa parola, implementazione, è misera ed inelegante ma ha una radice importante: viene da *in-plere*, ovvero riempire dentro.
Una volta che il concetto astratto è stato presentato esso viene *riempito*, per così dire, da un esempio concreto: un programma in uno specifico linguaggio di programmazione. Il linguaggio in oggetto è chiaramente Max, ormai anch'esso stato dell'arte nel mondo della computer music. Ma non bisogna farsi ingannare: i testi di Cipriani e Giri non sono schiavi dell'implementazione e non invecchieranno per diventare oggetti impolverati sugli scaffali. Le implementazioni presentate, infatti, aggiungono nuovi dettagli ad un concetto e si uniscono ad esso. Un merito, a mio avviso.
Per tornare alla metafora aristotelica, l'endiadi formale *teoria/pratica* ricorda un po' la divisione tra *Metafisica* e *Fisica*: una divisione, in effetti, più ideale che altro. Tutto, infatti, si fonde in un *unicum* concettuale che abbraccia lo scibile dell'oggetto in indagine.

I concetti presentati in questo terzo volume sono autentici e sempre interessanti per i cultori della materia. Tra essi troviamo la spazializzazione dei segnali, la sintesi non lineare, la sintesi granulare e, particolarmente interessanti per me, le trasformazioni nel dominio della frequenza.

In un certo senso, il tema in comune tra tutti i capitoli è quello di *rappresentazione*. Questo termine ha un preciso significato, da un punto di vista matematico. Rappresentare un segnale significa *proiettarlo* in uno spazio in cui alcune specifiche proprietà sono messe in evidenza. La proiezione è creata calcolando il prodotto scalare tra il segnale originale ed un opportuno insieme di funzioni chiamato *kernel*. La rappresentazione per antonomasia è quella di Fourier, in cui il kernel è costituito da un insieme di sinusoidi e in cui l'aspetto messo in evidenza è la frequenza dei segnali (capitolo 13). È possibile, tuttavia, generalizzare questo tipo di rappresentazione usando qualsiasi tipo di kernel. Un'altra importante rappresentazione, ad esempio, è quella in cui sia il tempo che la frequenza sono esplicitati mediante un kernel fatto di sinusoidi che hanno una specifica durata. Tale rappresentazione, detta di Gabor, è alla base della sintesi granulare e dell'approccio microtemporale al trattamento del segnale (capitolo 12). Quando la proiezione, infine, è fatta con kernel che mostrano proprietà ancora più particolari, si può parlare di modulazione e di distorsione non lineare (capitolo 11). In aggiunta, il capitolo 10 si occupa della spazializzazione. Per la prima volta, si vedono finalmente raggruppate tutte le tecniche algoritmiche per la riverberazione, unificate da un unico stile di programmazione. È così possibile capire *perché* l'algoritmo di Schroeder suoni male o perché le *feedback delay network* siano difficili da controllare. Non credo che si possa fare meglio, da un punto di vista pedagogico. Come in una forma musicale, per finire, il libro è arricchito da *Interludi*. In essi, vi si trovano svelati i principi essenziali di *Gen* (il linguaggio interno a Max utile per costruire algoritmi più efficienti) e di *Jitter* (il sottoinsieme di funzioni dedicate al trattamento dei video e delle matrici).

Il libro è corredato di una serie di materiali digitali a supporto, che includono *patch* ed esempi sonori. Questi ultimi si rivelano di essenziale importanze per la comprensione di concetti presentati, in particolare per quelli più avanzati basati sulla trasformata di Fourier (capitolo 13). Con un semplice ascolto, sarà allora possibile capire la differenza tra un *bin shifting* ed uno *spectral* LFO, oppure tra uno *spectral freeze* ed uno *spectral warping*. Creare dei buoni esempi sonori è una vera arte e, anche in questo, gli autori riescono ad essere efficaci ed interessanti.

In conclusione, il terzo volume di *Musica elettronica e sound design* è un caleidoscopico catalogo di idee e applicazioni per analizzare, sintetizzare e trasformare i segnali ad ampio raggio.
Uno sforzo importante, nel panorama contemporaneo, che permette di colmare il vuoto pedagogico creatosi per mancanza di libri che *spiegano*. Chi ha insegnato musica elettronica, in qualche sua declinazione, conosce bene l'eterogeneità dell'uditorio tipico della disciplina: compositori, dee-jay, sound designer, media artist. È sempre molto difficile riuscire a comunicare

efficacemente con tutti senza perdere in spessore, quando si è in presenza di una tale diversità. Cipriani e Giri riescono a parlare a tutti senza indebolire il costrutto teorico e senza inutili specializzazioni. Un magistrale equilibrio tra comprensibilità, funzionalità ed ampiezza.

Volendo usare una espressione tipica negli Stati Uniti, direi che questo lavoro è *larger than life*: in tutta onestà, se ne sentiva proprio il bisogno.

Carmine-Emanuele Cella
Assistant professor in Music and Technology
Center for New Music and Audio Technology (CNMAT)
University of California, Berkeley

INTRODUZIONE

Questo è il terzo di una serie di volumi sulla sintesi e l'elaborazione digitale del suono. Il piano dell'opera prevede anche:

- un primo volume che tratta diversi temi fra cui la sintesi additiva, generatori di rumore, filtri, sintesi sottrattiva e segnali di controllo;
- un secondo volume che affronta temi fra cui l'audio digitale, i processori di dinamica, le linee di ritardo, il protocollo MIDI e il tempo reale, Max for Live e un capitolo dedicato all'arte dell'organizzazione del suono.

LIVELLO RICHIESTO

Tutti i volumi alternano parti teoriche a sezioni di pratica al computer, che vanno studiate in stretta connessione. Questo terzo volume può essere utilizzato da utenti esperti e che abbiano ben chiari i concetti e la pratica su Max delineati nel primo e nel secondo volume. Per questo motivo nel testo si indica spesso il paragrafo di riferimento in cui sono stati trattati gli argomenti la cui conoscenza è data per acquisita in questo volume.
Il percorso di questo libro può essere svolto in auto-apprendimento oppure sotto la guida di un insegnante.

GLI ESEMPI SONORI

Il percorso della parte teorica è accompagnato da molti esempi sonori reperibili sul sito nella pagina di supporto. Utilizzando questi esempi, si può fare esperienza immediata del suono e della sua creazione ed elaborazione senza aver ancora affrontato alcun lavoro pratico di programmazione. In questo modo lo studio della teoria è sempre in connessione con la percezione del suono e delle sue possibili modificazioni.

MAX

La parte pratica del libro è stata realizzata sul software Max 8, reperibile sul sito www.cycling74.com. Nella pagina web di supporto si possono trovare le *patch*, i *sound file*, le estensioni di libreria (le *Virtual Sound Macros*) e altri materiali di supporto alle attività pratiche.

IL PROGETTO DIDATTICO

Musica Elettronica e Sound Design non è semplicemente un libro, ma un sistema didattico complesso, non lineare, che prevede l'interazione fra conoscenza, percezione, abilità, esperienza, auto-valutazione e creatività.

In particolare il sistema si basa su proposte concrete per superare i seguenti limiti:

a) la separazione tra teoria e pratica
b) la separazione tra l'apprendimento tecnico e l'arte dell'organizzazione dei suoni
c) la separazione tra teoria e percezione
d) l'assenza del coinvolgimento attivo dello studente nel processo di apprendimento.

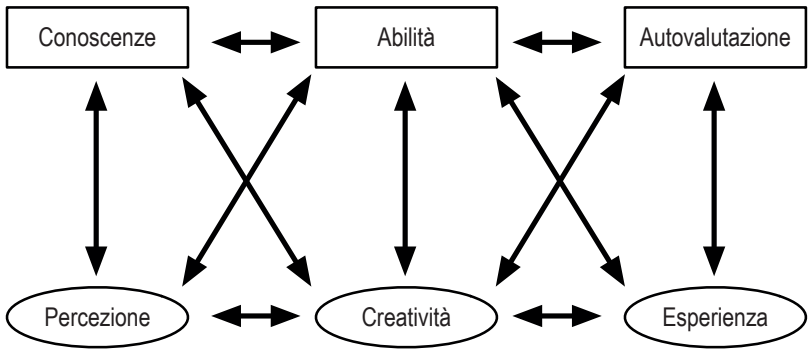


fig. I – Schema del sistema didattico

L'utente si trova quindi a interagire con una struttura a più dimensioni in cui il software, la teoria, gli esempi sonori e gli algoritmi forniti formano un tutt'uno.

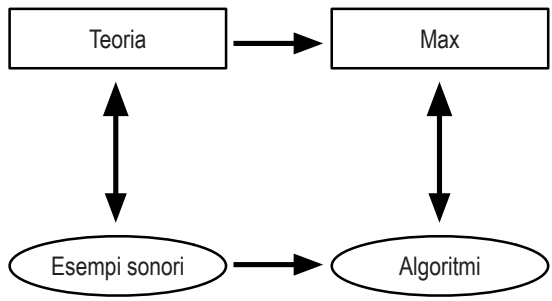


fig. II - Struttura non-lineare del sistema di apprendimento

Per questo, nell'affrontare questa terza parte del percorso l'utente potrà mettere in connessione le conoscenze e le abilità già acquisite nel campo della programmazione e dell'analisi all'ascolto con la propria creatività. A questo punto dell'itinerario, infatti, acquisito un buon livello di competenze, l'utente potrà man mano inventare lui stesso nuove possibilità di interazione fra le tecniche descritte, e l'elemento della creatività sarà molto importante per sviluppare un proprio terreno di ricerca e di invenzione. All'inizio, è stato necessario fornire regole prive di contesto a cui il lettore potesse aderire, al fine di raggiungere obiettivi immediati e rafforzare la fiducia attraverso procedure fisse. D'altro canto, riteniamo che sia particolarmente importante promuovere l'esperienza basata sul contesto, sviluppare il pensiero critico e incoraggiare sempre più l'uso della percezione e della creatività individuali man mano che si acquisiscono nuove conoscenze e abilità.

CONTENUTI E INDICAZIONI PER L'APPRENDIMENTO

Anche questo tomo, come il primo e il secondo, va studiato alternando ogni capitolo di teoria a quello corrispettivo di pratica incluse le attività al computer e l'ascolto attento degli esempi sonori.
Dopo il lavoro fatto sul secondo volume, il nostro lettore tipo ha ulteriormente evoluto le sue competenze: ci rivolgiamo quindi a un utente già esperto, che conosce tutte le tecniche di cui abbiamo parlato nei tomi precedenti ed è in grado di programmarle. Di conseguenza alcuni passaggi (nella teoria e soprattutto nella pratica) che consideriamo acquisiti non vengono dettagliati come nel primo e secondo volume. Spesso, a questo scopo, sono stati inseriti riferimenti ai volumi precedenti per rinfrescare la memoria su alcuni concetti o tecniche.
Dal punto di vista della programmazione, in questo volume ci sono due novità, che vengono trattate nei due interludi *F* e *G*. Il primo riguarda l'ambiente **Gen**. Gen è un'estensione dell'ambiente di sviluppo di Max che consente di realizzare algoritmi molto più efficienti rispetto a quelli programmabili con il solo Max. L'ambiente *Gen* infatti consente di creare *patch* che vengono immediatamente compilate in codice macchina eseguibile che processa un campione per volta, anziché gruppi di campioni come avviene in MSP. L'interludio F è allo stesso tempo un tutorial per questo ambiente e una descrizione di tante delle nuove possibilità di cui *Gen* ci permette di avvalerci. Ricorreremo spesso, nei capitoli successivi, ad algoritmi realizzati in *Gen*, sia per avere una maggiore velocità, sia per sfruttare le caratteristiche peculiari dell'estensione.

L'interludio G tratta invece dell'uso di **Jitter** per controllare, visualizzare, elaborare e generare segnali audio.
In realtà Jitter, che è un'estensione di Max, può fare molto di più perché può gestire anche immagini e video e grafica 3D in modo complesso, ma lo scopo di questo volume, come per i precedenti, è quello di approfondire specificamente le tecniche riguardanti la sintesi e l'elaborazione del suono. Di conseguenza, dopo un'introduzione all'ambiente Jitter comprendente anche brevemente la gestione di immagini e video, l'interludio G si focalizza sulla creazione di matrici e insiemi di dati per la gestione e l'interazione con l'audio.

Dopo l'interludio F, nel capitolo 10, viene delineato un percorso teorico e pratico sulle varie tecniche di simulazione del **riverbero** mediante algoritmi realizzati con linee di ritardo, a partire dai riverberi storici come quello di Schroeder, passando per le tecniche del *Freeverb* e del riverbero di Dattorro fino ad arrivare al riverbero *Feedback Delay Network*. Nello stesso capitolo si affrontano tecniche di base della **spazializzazione stereo e multicanale**.

Il capitolo successivo è incentrato sulle varie tecniche di **sintesi non lineare**, cioè su processi di sintesi ed elaborazione del suono che consentono di ottenere in uscita molte componenti non presenti all'ingresso, causando un arricchimento dello spettro e la sua traslazione. In particolare vengono affrontate le teorie e le tecniche della modulazione d'ampiezza, ad anello e a banda laterale singola, modulazione di frequenza e di fase, *feedback* PM, distorsione di fase, distorsione non lineare e la complessa ma interessantissima *Wave Terrain Synthesis*.

Il Capitolo 12 è dedicato al **microsuono**, cioè a fenomeni sonori di brevissima durata. Le tecniche riguardanti questa sfera della computer music vanno dalla sintesi granulare sincrona e asincrona a quella per formanti. Inoltre, alcune tecniche di sintesi particellare descritte da Roads nel suo testo *Microsound* vengono implementate in Max, come ad esempio la *glisson synthesis*, la *grainlet synthesis*, la *trainlet synthesis* e la *pulsar synthesis*.
Nella seconda parte di questo capitolo vengono affrontate anche tecniche di elaborazione del suono come la granulazione dei suoni campionati e il *multi-source brassage*.

Un capitolo molto esteso è il 13: la prima parte è dedicata alle tecniche di sintesi basate su **analisi**; la seconda parte alla **convoluzione**. In particolare sono state dettagliate tecniche diverse per la costruzione dei *vocoder*, sia quelle classiche, sia quelle basate su STFT. La parte più consistente di questo capitolo tratta la teoria della trasformata di Fourier e varie tecniche di elaborazione nel dominio della frequenza, diversissime fra loro, e molto utili per elaborazioni interessanti a fini musicali. A partire dalla teoria e la tecnica del *phase vocoder*, troviamo filtri *brickwall*, filtri multibanda e filtri random, LFO spettrale, *bin shifting*, sintesi incrociata mediante STFT, *bin feedback delay*, *freeze*, etc.
La seconda parte tratta la *cross-synthesis* fra due segnali mediante convoluzione, la convoluzione con microsuoni e i riverberi a convoluzione, con dettagli sulle differenze fra convoluzione diretta e la cosiddetta "convoluzione veloce".

Tornando all'interludio G, che chiude il libro, l'uso di Jitter consente fra l'altro di mettere in pratica alcune delle tecniche che abbiamo descritto in modo ancora più efficiente e quindi sarà utile per il lettore addentrarsi nella tecnica di questa interessantissima estensione di Max, che esiste da molti anni, ma sempre in evoluzione.

Commenti e segnalazioni
Correzioni e commenti sono benvenuti. Vi preghiamo di inviarli per e-mail a: a.cipriani@edisonstudio.it (www.edisonstudio.it/alessandro-cipriani/) oppure maurizio@giri.it (www.giri.it).

RINGRAZIAMENTI

Si ringraziano:
Maurizio Argentieri, Andrew Bentley, Daniel Biro, Emanuele Casale, Luigi Ceccarelli, Marco Cento, Sofia Cipriani (per i suoni di violino), Agostino Di Scipio, Edison Studio, Samuele Grippo, Paul Lansky, Marco Massimi, Curtis Roads, Barry Truax, Teresa Vasselli e Trevor Wishart.
Un ringraziamento speciale va a Carmine Emanuele Cella per le preziose indicazioni riguardanti, in particolare, il capitolo di analisi e risintesi. Ringraziamo inoltre Vincenzo Core per l'attenzione con cui si è dedicato alla lettura e analisi di questo volume e per le sue puntuali osservazioni.

DEDICA

Questo volume è dedicato a Massimo e Francesco Cipriani e ad Alex Giri.

Buona lettura,
Alessandro Cipriani e Maurizio Giri

LEGENDA DEI SIMBOLI UTILIZZATI

• ATTIVITÀ ED ESEMPI SONORI

• CONCETTI DI BASE

• VERIFICA

Interludio F
GEN: UN'INTRODUZIONE

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume I e II

OBIETTIVI

Conoscenze

- Conoscere le caratteristiche principali dell'ambiente Gen

Abilità

- Saper programmare e utilizzare algoritmi realizzati in ambiente Gen
- Saper utilizzare gli operatori Gen per la gestione delle linee di ritardo
- Saper creare algoritmi con utilizzo di *subpatch* e *abstraction* in Gen
- Saper riscrivere in Gen le *patch* MSP
- Saper scrivere delle piccole funzioni in Gen utilizzando il codice *GenExpr*
- Saper creare algoritmi con operatori Gen multicanale

Competenze

- Saper realizzare un breve studio basato sull'uso creativo di Gen

ATTIVITÀ

- Costruzione e modifiche di algoritmi

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista attributi per oggetti Max specifici - Lista operatori Gen Glossario

IF.1 L'AMBIENTE GEN

Prima di iniziare la studio di questo volume vi raccomandiamo di caricare e installare l'ultima versione della libreria *Virtual Sound Macros*, che troverete, insieme a tutto il materiale di supporto per il terzo volume, nella pagina di supporto del libro.

Come abbiamo anticipato nell'introduzione, il presente volume farà ampio uso di algoritmi realizzati in Gen, indispensabili per la realizzazione di diverse tecniche di sintesi ed elaborazione del suono che presenteremo nel corso del libro.
In questo interludio presentiamo le caratteristiche principali dell'ambiente Gen; nei capitoli successivi approfondiremo l'argomento via via che se ne presenterà la necessità.

Cerchiamo innanzitutto di capire quali sono i vantaggi di Gen rispetto alla normale programmazione MSP.
Gen è un ambiente di sviluppo all'interno di Max con cui è possibile creare delle *patch* che vengono immediatamente compilate in codice macchina eseguibile[1]. In pratica una *patch* realizzata in Gen è equivalente a un oggetto Max MSP nativo come ad esempio `cycle~` o `biquad~`. Ciò rende possibile la realizzazione di algoritmi che sarebbero estremamente inefficienti se programmati semplicemente in Max.
Un altro vantaggio è che il circuito audio in una *patch* Gen non viene processato in vettori definiti dal *Signal Vector Size* (come avviene in MSP) ma un campione alla volta: e ciò rende possibile, ad esempio, un *delay* con *feedback* di un solo campione. Questa caratteristica ci permette di costruire filtri con schemi diversi dal `biquad~`, ed è inoltre, come vedremo, indispensabile per la realizzazione di molti algoritmi che funzionano ad *audio rate*.
Infine, inviando all'oggetto `gen~` il messaggio "exportcode" è possibile esportare la *patch* in codice C++ che può essere utilizzato per creare applicazioni al di fuori dell'ambiente Max.

L'ambiente di *patching* è molto simile a quello che già conosciamo: ci sono oggetti grafici che vengono collegati tra loro tramite cavi virtuali, e l'aspetto grafico è assolutamente identico a quello del "normale" ambiente Max.
Una *patch* Gen può essere inserita (come *subpatch* o come *abstraction*) all'interno di cinque speciali oggetti: **`gen~`**, `gen`, `jit.gen`, `jit.pix` e `jit.gl.pix`.
Come è possibile evincere dal nome, il primo oggetto (`gen~`) permette di realizzare *patch* Gen per il trattamento dell'audio, il secondo (`gen`) è una versione *control rate* del primo (ne parleremo al paragrafo IF.8), mentre gli ultimi tre oggetti servono a creare algoritmi per l'elaborazione di matrici, immagini e *textures* in Jitter[2].

[1] Cioè un codice direttamente eseguibile dal processore.

[2] Jitter è l'estensione di Max che serve appunto ad elaborare matrici di dati, immagini e video: ce ne occuperemo, limitatamente al suo utilizzo per la gestione e l'interazione con l'audio, nell'Interludio G, dove parleremo anche dell'oggetto `jit.gen`.

In questo interludio parleremo esclusivamente di *patch* per il trattamento audio o dei segnali di controllo realizzate all'interno di `gen~` e `gen`.
Vediamo una prima, semplicissima *patch*: caricate il file **IF_01_firstGen.maxpat** (fig. IF.1).

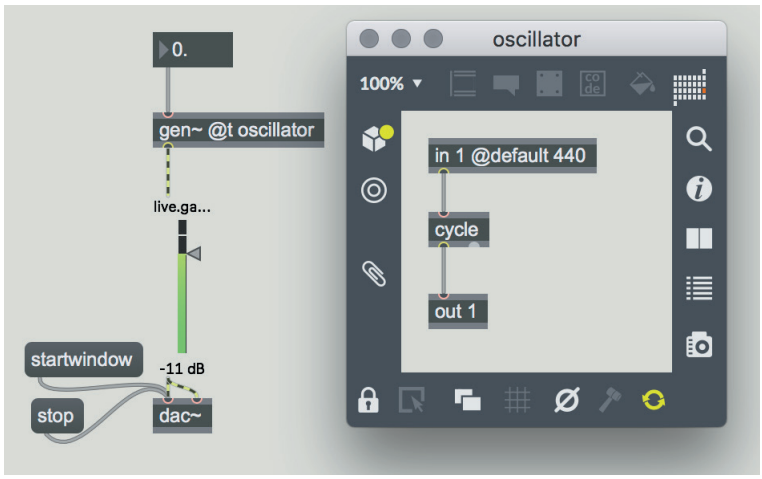


fig. IF.1: file **IF_01_firstGen.maxpat**

Facendo doppio clic sull'oggetto `gen~` visibile nella parte sinistra dell'immagine apparirà la *subpatch* di destra. Vediamo innanzitutto che l'oggetto `gen~` ha un attributo, @t (abbreviazione di @title), che specifica il titolo della *subpatch*.
Attivate ora la *patch* facendo clic sul messaggio *startwindow*: se non avete modificato il valore del *number box* collegato a `gen~`, si dovrebbe sentire un suono sinusoidale a 440 Hz.
All'interno della *subpatch* "oscillator" abbiamo un *inlet* [in 1 @default 440], molto simile agli ingressi dell'oggetto `poly~`, poi un oscillatore sinusoidale `cycle`, e infine una uscita [`out` 1], anche questa simile alle uscite di un oggetto `poly~`.
Come si può vedere, per chi conosce MSP questa semplice *patch* Gen è perfettamente comprensibile: le uniche differenze, rispetto ad un analogo programma MSP, sono che il nome degli "oggetti" è privo di tilde (`cycle` invece di `cycle~`) e i cavi sono grigi invece che giallo-neri.
Queste differenze sono dovute al fatto che all'interno di una *patch* `gen~` tutto è segnale, non ci sono messaggi Max asincroni, né tantomeno liste o stringhe; non è quindi necessario distinguere i segnali dagli altri messaggi tramite suffissi o colorazioni particolari dei cavi. Un altro motivo è che in questo modo è possibile distinguere gli operatori `gen~` dai corrispondenti oggetti MSP che hanno lo stesso nome (come vedremo ce ne sono diversi).
Notate inoltre che gli "oggetti" Gen, nella maggior parte dei casi, vengono chiamati in realtà *operatori*: possiamo così distinguere l'*operatore* Gen `phasor` dall'analogo oggetto MSP `phasor~`.
Il fatto che all'interno di una *patch* `gen~` circolino solo segnali comporta una serie di conseguenze, non tutte ovvie. Ad esempio se diamo un argomento ad un operatore, l'*inlet* corrispondente a quell'argomento sparisce. Provate

ad esempio ad aggiungere l'argomento 440 all'operatore `cycle` nella *patch* che avete caricato: il suo ingresso scomparirà. Questo avviene perché l'eventuale collegamento di un segnale a `cycle` annullerebbe immediatamente l'argomento, sostituendolo con il valore trasmesso dal segnale e rendendolo di fatto inutile.
Se non c'è alcun argomento nell'operatore `cycle` in figura, perché abbiamo sentito una sinusoide a 440 Hz quando abbiamo avviato il motore DSP? Perché abbiamo assegnato l'attributo "@default 440" all'ingresso `in`, che, in mancanza di un valore dall'esterno, genera un segnale costante il cui valore è 440.
Se ora provate a modificare il valore del *number box* nella *patch* principale, la frequenza dell'oscillatore cambierà di conseguenza.
Agli ingressi di un oggetto `gen~` possono essere collegati dei segnali o dei generatori di valori numerici Max, che vengono convertiti in segnale all'interno della *patch* `gen~`. Mentre un segnale audio viene passato all'oggetto `gen~` un numero di volte al secondo pari al *sample rate*, un valore numerico Max viene passato una volta ogni *Signal Vector* (quindi ad esempio ogni 32 o 64 cicli di DSP, vedi il paragrafo 5.1P del secondo volume): questo ci permette un piccolo risparmio di CPU nei casi in cui non sia necessario ottenere un nuovo valore ad ogni ciclo.
Inoltre, se siamo sicuri che un determinato parametro per una *patch* `gen~` verrà solo trasmesso tramite messaggi Max, possiamo utilizzare l'operatore **`param`** che ci permette di passare un parametro esterno tramite un nome (anche i parametri vengono passati una volta ogni *Signal Vector*). Vediamo un esempio: aprite la *patch* **IF_02_param.maxpat** (fig. IF.2).

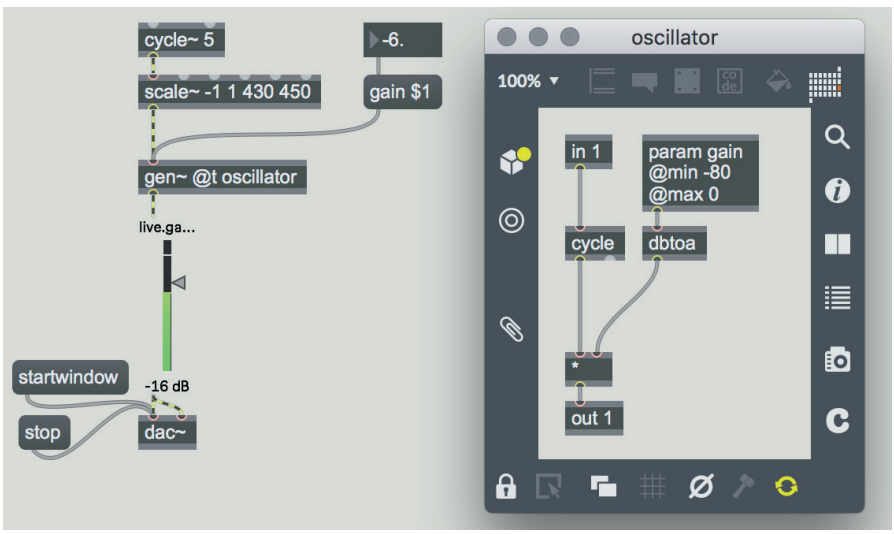


fig. IF.2: file **IF_02_param.maxpat**

Nella *subpatch* `gen~` abbiamo aggiunto l'operatore `param` a cui abbiamo dato nome "gain": se ora inviamo all'oggetto `gen~` un messaggio composto dalla stringa "gain" seguita da un valore numerico, quel valore verrà passato al param "gain" (vedi la *patch* principale sul lato sinistro della figura).

In questa seconda *patch* abbiamo aggiunto un vibrato al valore di frequenza trasmesso all'oscillatore interno per dimostrare che attraverso un ingresso di **gen~** è possibile mandare sia segnali, sia messaggi per gli operatori **param**.
Il valore di "gain" è espresso in dB (limitato tra -80 e 0 dai due attributi che abbiamo aggiunto a **param**)[3] e convertito in ampiezza dall'operatore **dbtoa**.
Un altro vantaggio di **param** è che i parametri che definisce diventano di fatto attributi di **gen~**, ed è possibile impostare il valore iniziale di un parametro attraverso la nota sintassi @nome_attributo. Provate ad esempio a modificare l'oggetto **gen~** di fig. IF.2 in questo modo:
[**gen~** @t oscillator @gain -30]
Ora l'ampiezza iniziale dell'oscillatore sarà pari a -30 dB.

Avrete forse notato che le *Toolbar* che circondano la *patcher window* di una *subpatch* Gen presentano qualche differenza rispetto a quelle che già conosciamo; vediamo alcune di queste differenze.
Nella *Toolbar* superiore sono sparite le *palette*: non possiamo infatti inserire oggetti interfaccia all'interno di una *patch* Gen, ma solo operatori. Le uniche eccezioni, presenti nella *Toolbar*, sono i *comment box* e i *panel* che, come forse ricorderete, servono a creare delle zone colorate utili per raggruppare gli oggetti e rendere più chiara una *patch*.
Inoltre è presente un nuovo oggetto, **codebox**, che ci permette di scrivere codice *GenExpr*. Di cosa si tratta? Quando assembliamo una *patch* in Gen, il sistema crea un codice testuale in linguaggio *GenExpr* (un linguaggio creato appositamente per Gen, che assomiglia a una versione molto semplificata del linguaggio C); questo codice testuale viene poi compilato in codice macchina nativo. È possibile vedere il codice *GenExpr* che viene generato da una *patch* facendo clic sull'icona raffigurante una C nella *Toolbar* di destra.
Ad esempio, il codice generato dalla *patch* Gen di figura IF.2 è questo:

```
Param gain(0, max=0, min=-80);
dbtoa_1 = dbtoa(gain);
cycle_2, cycleindex_3 = cycle(in1);
mul_4 = cycle_2 * dbtoa_1;
out1 = mul_4;
```

È possibile scrivere direttamente in *GenExpr*? Sì, utilizzando l'oggetto **codebox** che può essere richiamato tramite la *Toolbar* superiore. Una discussione di *GenExpr*, comunque, esula dagli scopi di questo interludio[4].

[3] Per conoscere gli attributi di un operatore è sufficiente digitare il carattere "@" all'interno dell'operatore stesso: apparirà il menù di *autocompletion* con tutti gli attributi disponibili. È inoltre possibile richiamare l'*help* e la *Reference Page* tramite il tasto destro del mouse in modalità *edit*, esattamente come per gli oggetti Max. A differenza di questi ultimi, però, l'*help* di un operatore non apre una *patch* eseguibile, ma un piccolo "fumetto" contenente una descrizione dell'operatore.
[4] Faremo comunque alcuni accenni alla sintassi di *GenExpr* al paragrafo IF.9, quando parleremo dell'attributo @expr.

Nella *Toolbar* inferiore abbiamo tre nuove icone, visibili in figura IF.3.

fig. IF.3: icone Gen

L'icona di destra (raffigurante due frecce che formano un cerchio) attiva e disattiva la funzione di auto-compilazione. Quando è attiva, ogni volta che si modifica la *patch* viene immediatamente generato il nuovo codice *GenExpr*. Di *default* questa opzione è attiva, e può essere utile disattivarla quando si sta lavorando ad una *patch* molto grande, per evitare che la continua ricompilazione del codice rallenti il lavoro di programmazione.
L'icona centrale serve ad avviare manualmente la compilazione del programma quando l'auto-compilazione è disattivata.
L'icona di sinistra ripristina i valori di *default*: se ad esempio nella *patch* di figura IF.1 modificate la frequenza dell'oscillatore tramite il *number box*, con un clic su questa icona potete riportare la frequenza a 440 Hz.

Come abbiamo detto, le *patch* che carichiamo in `gen~` vengono eseguite ad *audio rate*: questo rende naturalmente più complessa la programmazione rispetto all'ambiente Max dove è possibile lavorare contemporaneamente con messaggi asincroni (ad esempio quelli generati dagli oggetti interfaccia), messaggi di controllo temporizzati (ad esempio il flusso di *bang* prodotto dall'oggetto `metro`) e segnali.
D'altra parte, però, l'ambiente Gen ha degli aspetti, assenti in Max, che facilitano e rendono più compatte le *patch*. Vediamone alcuni.

Ci sono diverse variabili e costanti predefinite che facilitano i calcoli, come ad esempio *pi* (pi greco), *twopi* (due volte pi greco), *samplerate* (la frequenza di campionamento corrente) ed altre che vedremo nel corso dei prossimi capitoli. La cosa interessante è che questi valori possono essere utilizzati direttamente come argomento: se vogliamo moltiplicare un valore per pi greco ci basterà usare l'operatore [* pi].
Ad esempio, per creare un oscillatore sinusoidale utilizzando l'operatore `sin` che genera la funzione seno e ha bisogno di ricevere in *input* la fase in radianti, possiamo creare una *patch* Gen come quella di figura IF.4.

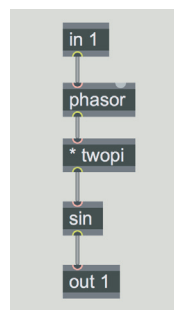


fig. IF.4: generatore di sinusoidi

Possiamo inoltre usare delle espressioni come argomento, utilizzando gli operatori e le funzioni matematiche disponibili per creare una nuova costante[5]. Ad esempio, per calcolare la radice cubica di un valore in ingresso, in Gen possiamo usare [`pow` 1/3], mentre in Max dovremmo scrivere [`pow` 0.333333], con probabile perdita di precisione.
È anche possibile includere il nome di un ingresso (in1, in2 e così via) nella definizione di un argomento. Provate ad esempio a creare una *patch* con un oggetto `gen~` che contenga la *subpatch* mostrata in figura IF.5

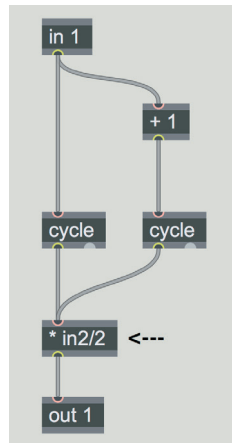


fig. IF.5: uso di un ingresso come argomento

Questa *patch* genera due sinusoidi la cui frequenza differisce di 1 Hz (causando quindi un battimento al secondo), le due sinusoidi vengono sommate tra loro e moltiplicate per il valore di ampiezza proveniente dal secondo ingresso. Il valore di ampiezza è inoltre dimezzato, per evitare che la somma delle due sinusoidi ecceda i limiti -1/1 del convertitore DA. Notate che non c'è nemmeno bisogno di creare un oggetto [`in` 2]; basta la sua presenza come argomento del moltiplicatore a creare un secondo *inlet* nell'oggetto `gen~`.
Anche i parametri passati attraverso l'operatore `param` possono essere utilizzati come argomento, vedi figura IF.6.

In questo caso è necessario creare un operatore `param` con il nome del parametro (visibile in alto a destra in figura). E naturalmente nella *patch* principale, bisogna passare il valore di ampiezza preceduto dal nome del parametro 'amp'.

[5] Le funzioni e gli operatori matematici in Gen sono per la maggior parte identici a quelli che troviamo in Max e in MSP: per una lista di funzioni e operatori disponibili vedi la guida "Gen Common Operators" nella documentazione (raggiungibile tramite il menù Help/Reference).

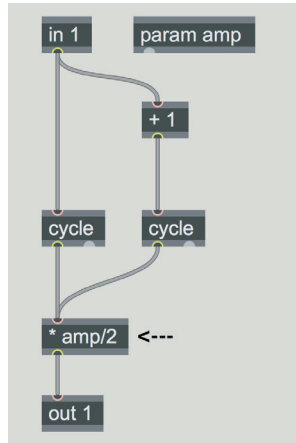


fig. IF.6: uso di un parametro come argomento

(...)

il capitolo prosegue con:

IF.2 LINEE DI RITARDO CON GEN

IF.3 SUBPATCH E ABSTRACTION IN GEN

IF.4 MEMORIZZAZIONE E GESTIONE DI DATI IN GEN

IF.5 SAMPLE AND HOLD

IF.6 RISCRIVERE IN GEN LE PATCH MSP

IF.7 GLI OPERATORI BOOLEANI

IF.8 L'OGGETTO GEN (SENZA TILDE)

IF.9 L'ATTRIBUTO @EXPR

IF.10 GEN E IL SISTEMA MULTICANALE

ATTIVITÀ
• Analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi, correzione di algoritmi

VERIFICHE
• Compiti unitari di reverse engineering

SUSSIDI DIDATTICI
• Lista oggetti Max - Lista attributi per oggetti Max specifici - Lista operatori Gen - Glossario

10T
RIVERBERO E SPAZIALIZZAZIONE

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume I e II

OBIETTIVI

Conoscenze

- Conoscere i fondamenti della teoria del riverbero
- Conoscere le basi della tecnica del riverbero di Schroeder e Schroeder-Moorer
- Conoscere le basi della tecnica del riverbero *Freeverb*
- Conoscere le basi della tecnica del riverbero di Dattorro
- Conoscere le basi della tecnica del riverbero FDN
- Conoscere i concetti base nel campo della spazializzazione stereo e multicanale

Abilità

- Saper individuare all'ascolto i mutamenti di alcuni parametri del riverbero: *decay time*, volume di una stanza, equilibrio *dry/wet*
- Saper individuare all'ascolto l'effetto *doppler*
- Saper individuare all'ascolto da un sistema quadrifonico o surround 5.1 gli effetti di rotazione, alternanza dei canali, localizzazione della distanza e della direzione del suono, differenze fra riverbero globale e riverbero locale, localizzazione di distanze del suono artificiali oltre gli altoparlanti

CONTENUTI

- Teoria di base del riverbero in campo acustico e nel dominio digitale
- Teoria di base dei riverberi di Schroeder, Schroeder-Moorer e Freeverb
- Teoria di Base del riverbero di Dattorro e FDN
- Teoria di Base della spazializzazione stereo e multicanale

ATTIVITÀ

- Esempi sonori

SUSSIDI DIDATTICI

- Concetti di base - Glossario

10.1 IL RIVERBERO

In questo capitolo introdurremo un tema complesso, quello del riverbero, o meglio della simulazione del riverbero mediante algoritmi. In particolare ci soffermeremo sugli algoritmi di riverbero realizzati con linee di ritardo. Come abbiamo accennato nel cap. 6.6, infatti, i filtri *comb* in connessione con filtri *allpass* possono essere utilizzati per la realizzazione di riverberi. Questa è solo una delle tecniche possibili. Si possono creare, infatti, anche algoritmi di riverbero mediante convoluzione, ma questi verranno trattati a parte nel capitolo 13.
Il riverbero naturale nasce, esattamente come l'eco, da riflessioni del suono sulle superfici dell'ambiente in cui esso viene prodotto. Come abbiamo spiegato nel par. 6.2, l'effetto di eco è percepibile in quanto tale se la distanza fra la sorgente sonora e l'ostacolo riflettente produce un ritardo superiore alla cosiddetta *zona di Haas* (25-35 ms, con alcune differenze a seconda del timbro e della frequenza di suono). Al di sotto di questo tempo di ritardo il suono diretto e le repliche successive tendono, percettivamente, a fondersi tra loro. Se gli ostacoli sono più di uno, come è il caso di uno spazio a forma di parallelepipedo, si può creare il fenomeno del riverbero, che comporta la produzione di un grande numero di riflessioni ravvicinate con tempi di ritardo brevi ma diversi tra loro. In questo caso l'ascoltatore tenderà a percepire le repliche come un unico evento sonoro a causa della quantità e dei tempi delle sovrapposizioni delle riflessioni con il suono stesso. Ovviamente il tipo di riverbero sarà soggetto a una serie di fattori come il materiale di cui sono fatti gli ostacoli, la forma e il numero di ostacoli su cui si riflette il suono, la forma e la grandezza dello spazio, la distanza della sorgente del suono dai vari ostacoli e, dal punto di vista percettivo, anche la distanza dell'ascoltatore dalla sorgente sonora e dagli ostacoli etc.

Il fenomeno descritto può essere simulato algoritmicamente mediante l'uso di linee di ritardo con *feedback* e filtri di varia natura. Nel tempo sono stati proposti e utilizzati algoritmi diversi per la simulazione del riverbero e qui ne analizzeremo i principali. Vediamo innanzitutto le varie fasi di un generico effetto di riverbero. Il tempo che intercorre tra il suono diretto e il primo suono riflesso si chiama ***pre-delay***; si tratta di un parametro che in alcuni algoritmi è impostabile separatamente e si riferisce, nel campo digitale, al tempo che intercorre fra il suono diretto e il primo suono riflesso, senza riferimenti alla percezione.[1] Trascorso questo tempo, all'ascoltatore arrivano le **prime riflessioni** (***early reflections***) che, compiendo un percorso più lungo rispetto a quello del suono diretto, arrivano più tardi. Le prime riflessioni, come si evince dal nome, sono i primi echi del suono riflesso che arrivano all'ascoltatore e che sono ancora distinguibili tra loro (specialmente se il suono sorgente è di tipo impulsivo o percussivo). In un algoritmo queste riflessioni vengono prodotte ripetendo il segnale d'ingresso molteplici volte, creando in queste ripetizioni

[1] In uno spazio acustico reale, invece, va considerato anche un tempo di latenza tra la generazione del suono e l'arrivo all'ascoltatore.

del suono adeguati mutamenti timbrici che simulino la risposta al segnale di un dato ambiente virtuale. Infatti, in uno spazio acustico reale, più volte un suono viene riflesso e più è debole, perché ad ogni riflessione cede una parte di energia alla superficie dell'ostacolo mutando progressivamente il timbro e diminuendo l'intensità dopo ogni riflessione. A seconda delle qualità e della forma dei materiali degli ostacoli, infatti, verrà assorbita più energia su alcune frequenze rispetto ad altre, in genere in modo simile a un filtraggio passa-basso.
A seconda del tempo di *pre-delay* e del tempo che intercorre fra le *early reflections* stesse, si ha una sensazione di uno spazio chiuso più grande o più piccolo. Ognuna delle riflessioni è una copia filtrata e ritardata del suono-sorgente. Quindi possiamo dire che le prime riflessioni ci forniscono la gran parte delle informazioni che riguardano lo spazio e le sue caratteristiche (fig. 10.1).

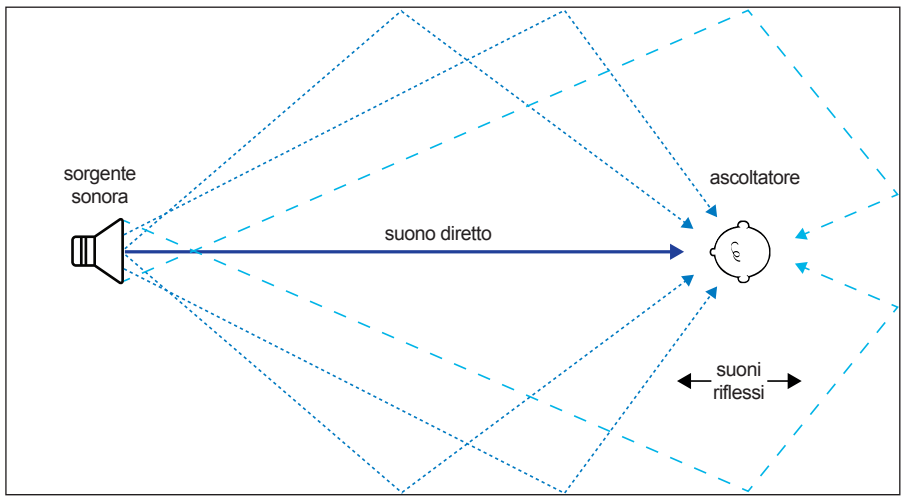


fig. 10.1: suono diretto e suoni riflessi

Successivamente arrivano all'ascoltatore le ulteriori riflessioni, che possono essere migliaia. Queste ultime si fondono insieme creando un accumulo progressivo di riflessioni mentre l'ampiezza complessiva aumenta all'inizio e poi decade in un determinato tempo, creando la sensazione di un allungamento e un cambiamento di timbro del suono stesso; chiameremo questa parte ***late reverberation***[2].

[2] Data la difficoltà di traduzione del termine, in italiano ci riferiremo alla *late reverberation* anche con termini più semplici come "coda" o "coda di riverbero".

Quando termina il suono, il riverbero continuerà per un certo tempo prima di estinguersi. In generale si definisce **tempo di riverberazione** (o **RT60**) il tempo che il segnale riverberato impiega per decrescere fino a 1/1000 della sua ampiezza iniziale, cioè il tempo che impiega ad attenuarsi di 60 dB[3].
In generale questo tempo di *decay* è inversamente proporzionale al coefficiente di assorbimento degli ostacoli presenti nella stanza (in modo particolare il soffitto, le pareti e il pavimento), ed è allo stesso tempo proporzionale alla grandezza dell'ambiente. Le caratteristiche degli ostacoli e le dimensioni dell'ambiente sono due elementi molto importanti per effettuare dei calcoli adeguati.

Il riverbero dà informazioni anche sulla distanza di un suono dall'ascoltatore. Infatti, in un ambiente più o meno riverberante, la percezione della distanza della sorgente sonora dall'ascoltatore è data dal **rapporto R/D**, cioè il rapporto fra suono riverberato e suono diretto. Il suono diretto prevale in ampiezza per suoni vicini, mentre per suoni lontani l'ampiezza del suono riverberato è maggiore di quella del suono diretto. Il suono diretto decresce in ampiezza più velocemente del suono riflesso quanto più il rapporto R/D è alto. Vedremo in maniera più dettagliata gli aspetti riguardanti la localizzazione uditiva nel par. 10.6.

ESEMPIO SONORO 10.1 • suono diretto, prime riflessioni e coda di riverbero

a) Suono diretto (*dry*)
b) Prime riflessioni (*early reflections*)
c) Suono diretto + prime riflessioni
d) Coda di riverbero (*late reverberation*)
e) Suono diretto + coda di riverbero senza prime riflessioni
f) Prime riflessioni + coda di riverbero senza suono *dry*
g) Suono diretto + prime riflessioni + coda di riverbero

Notiamo che nell'esempio e), non essendoci le prime riflessioni, si ha un riverbero un po' innaturale, come se la sorgente del suono fosse al di fuori di una caverna a distanza di qualche metro dall'entrata e il microfono fosse vicino

[3] "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. Il *decay* delle basse frequenze è più lento, quello delle alte frequenze è più rapido, perché gli ostacoli assorbono tali frequenze più di quelle gravi. 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." (Bianchini e Cipriani, 2003).

alla sorgente. A volte, quando si desidera la massima definizione del suono *dry* e contemporaneamente si vuole un riverbero, togliere le prime riflessioni può essere utile per distanziare nel tempo il segnale *dry* dalla *late reverberation* e quindi mantenere la presenza del suono originale senza rinunciare al riverbero. Ricapitolando: fra il suono diretto e la prima riflessione c'è un tempo di *pre-delay.* Alle *early reflections* segue la *late reverberation* in cui si ha un aumento e poi una diminuzione dell'ampiezza del riverbero, e in cui si fondono tutte le riflessioni, fino ad arrivare, al tempo RT60, ad un livello così basso che il riverbero si confonde con il rumore d'ambiente, per poi svanire. (fig. 10.2)

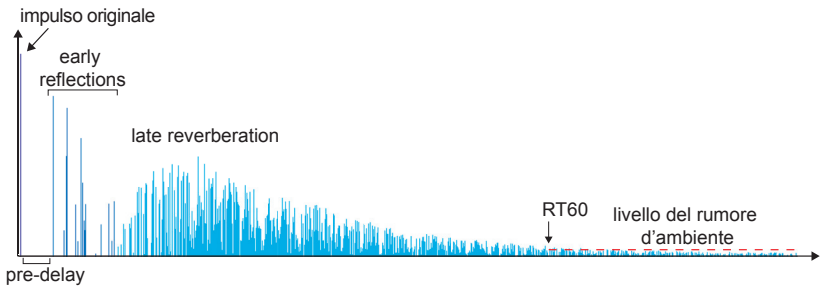


fig. 10.2: risposta all'impulso e fasi del riverbero

ESEMPIO SONORO 10.2 • riverberi diversi a seconda della posizione dell'ascoltatore in relazione alla sorgente sonora

a) Suono diretto (*dry*)
b) ascolto nella stessa stanza – distanza maggiore fra ascoltatore e sorgente sonora
c) sorgente sonora al di fuori della stanza dell'ascoltatore
d) sorgente sonora al piano superiore rispetto all'ascoltatore

La densità del riverbero dipende dalla quantità delle onde riflesse udibili e dalla loro vicinanza nel tempo. Ci saranno delle riflessioni con ampiezza maggiore ed altre con ampiezza minore. Se le riflessioni con maggiore energia sono disposte nel tempo in modo regolare si avrà una sensazione più "metallica" in quanto esse verranno a formare dei picchi su alcune frequenze. Nella realizzazione di algoritmi di riverbero questo può essere un effetto voluto, ma spesso è desiderabile avere distanze temporali irregolari fra le riflessioni più intense, per evitare di aggiungere frequenze spurie rispetto a quelle del suono originale e per avere un effetto più simile a quello naturale[4].

[4] Vedremo, in particolare nel par. 10.5, come alcune soluzioni vengono o da un aumento degli elementi in gioco (filtri, *delay*) o da un mutamento nel tempo dei valori dei parametri di tali elementi.

Da tutto ciò capiamo che, costruendo algoritmi per il riverbero, possiamo simulare spazi acustici reali o inventare spazi virtuali, modulando posizioni diverse per i vari suoni, alcuni dei quali saranno più presenti, in primo piano, altri più lontani. Possiamo creare anche effetti di figura-sfondo per i vari suoni; le possibilità sono praticamente infinite, compresi gli avvicinamenti e allontanamenti. Tutto ciò può essere combinato anche con la posizione dei suoni nello spazio acustico reale dell'ascolto (sugli assi *x*, *y* e *z* in contesti multicanale). Parleremo di ciò nei paragrafi dedicati alla spazializzazione. Introduciamo ora una delle tecniche più conosciute per la simulazione di riverberi: la tecnica di ricircolazione di Manfred Robert Schroeder.

10.2 IL RIVERBERO DI SCHROEDER

Manfred Schroeder nel 1961 implementò per primo un algoritmo per simulare l'effetto di riverbero in campo digitale[5]. La tecnica di ricircolazione di Schroeder utilizza due tipi di filtri come unità fondamentali[6] dei suoi algoritmi di riverbero: i filtri *comb* ricorsivi IIR e i filtri *allpass*, di cui abbiamo parlato nei paragrafi 6.6 e 6.7 del secondo volume. Ognuna di queste unità viene interconnessa alle altre a formare un algoritmo. Nella configurazione base il segnale viene mandato a 4 filtri *comb* in parallelo con *delay time* (o *loop time*) diversi in modo da non creare effetti di coincidenza degli echi e quindi di ripetitività[7]; ciò è importante per avere effetti di riverbero il più possibile "naturali". Mediante questi filtri si genera una serie di echi con ampiezze decrescenti. L'uscita dei 4 filtri *comb* viene sommata e inviata a 2 filtri *allpass* in cascata che hanno il compito di intensificare la densità delle "riflessioni" (fig. 10.3). Come vedremo, a questa configurazione verrà aggiunto un *multitap delay* (vedi par. 6.2) per simulare le prime riflessioni. Notate che il diagramma illustrato in figura 10.3 si riferisce a un riverbero monofonico; in un contesto stereofonico o multicanale, per ciascun canale i valori dei vari filtri vanno decorrelati[8] in modo da ottenere diversificazioni nei segnali in uscita.

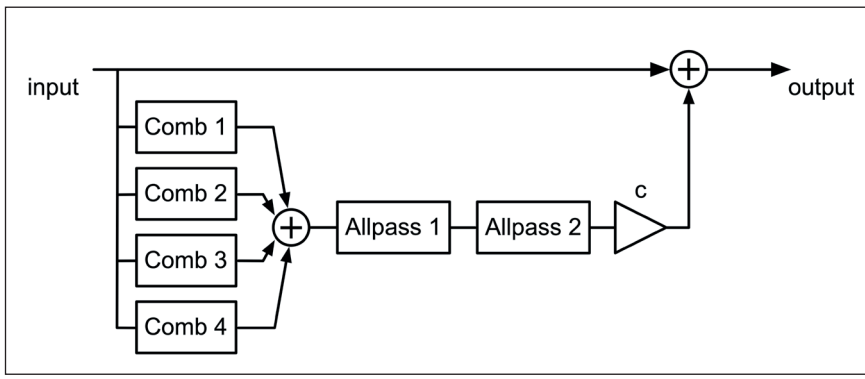


fig. 10.3: diagramma di flusso di un riverbero di Schroeder

Vediamo ora le caratteristiche particolari dei filtri *comb* e *allpass* da utilizzare nell'algoritmo di Schroeder.

5 (Schroeder, 1961).

6 Chiamate anche riverberatori "unitari" (ovvero unità elementari di riverberazione atte a formare sistemi più complessi).

7 Nella simulazione del riverbero, una delle idee per evitare ciò è quella di assegnare ai vari *delay time* valori primi rispetto all'origine, sebbene ciò non garantisca di per sé un effetto "naturale". Moorer infatti invita anche a considerare che, ad esempio, la somma o la differenza fra due numeri primi divisibile per due potrebbe causare di nuovo enfatizzazioni su alcune frequenze. (Moorer, 1979).

8 In questo caso si intende l'utilizzo di valori diversi dei tempi di *delay* dei filtri.

USO DEI FILTRI COMB NEL RIVERBERO DI SCHROEDER

Ricordiamo che nei filtri *comb* il segnale viene ritardato con un tempo di *delay d* dopodiché tale segnale, dopo essere stato moltiplicato per un fattore che chiameremo *c*, viene sommato a quello in ingresso. Si crea così un *feedback loop*, il cui scopo in questo caso è quello di simulare la ricorsività di alcune riflessioni di un suono che si susseguono l'una all'altra in uno spazio. Nel caso specifico del riverbero di Schroeder il *comb* utilizzato è sostanzialmente un *delay* con *feedback*, senza uscita del suono diretto e senza una linea di ritardo indipendente per il *feedforward*. Possiamo adottare una configurazione semplice come quella che vediamo in figura 10.4 oppure una più complessa e flessibile che spiegheremo fra poco.

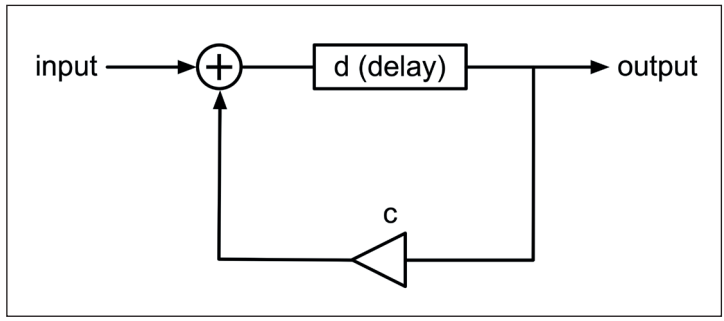


fig. 10.4: filtro *comb* per il riverbero di Schroeder

In uscita avremo un suono ritardato di un tempo *d* e poi una serie decrescente di echi. Il tempo di estinzione degli echi sarà tanto più lungo quanto maggiore è il valore di *c*, (da cui dipende l'ampiezza del segnale rimandato all'ingresso della linea di ritardo) e dalla durata del *delay d* (da cui dipende il tempo che intercorre fra i vari echi). Come sappiamo, il fattore c deve comunque essere sempre minore di 1, altrimenti l'ampiezza del suono in uscita, anziché decrescere, crescerebbe fino ad eccedere l'ampiezza massima prevista dal sistema (in digitale lo 0 dBFS). In uscita avremo quindi questo tipo di risposta all'impulso (fig. 10.5).

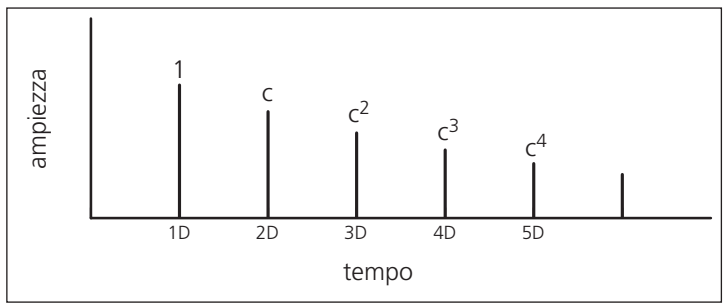


fig. 10.5: risposta all'impulso di un filtro *comb*

Il limite della configurazione in figura 10.3, seppur molto economica e assolutamente adatta al nostro scopo è che, ad esempio, non possiamo regolare separatamente l'ampiezza e il *delay* del *feedforward* rispetto a quelli

del *feedback*. Se vogliamo, possiamo anche utilizzare l'algoritmo che avevamo delineato nella figura 6.18 del secondo volume (fig. 10.6).

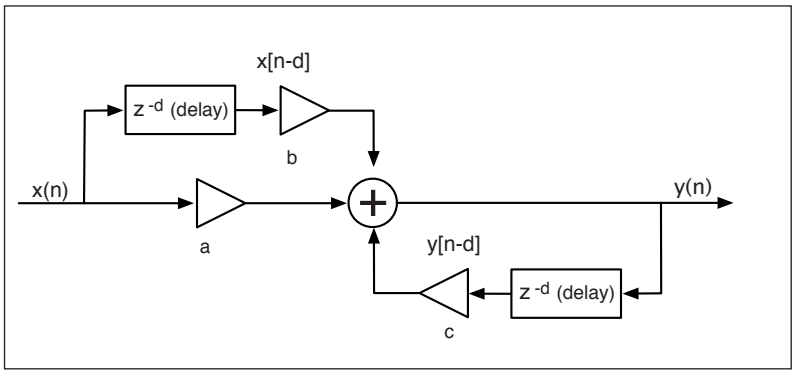


fig. 10.6: filtro *comb* IIR con *feedforward* e *feedback*

In questo caso, per l'utilizzo di questo algoritmo in funzione del riverbero di Schroeder, daremo dei valori dal risultato identico a quello della figura 10.3. In futuro questo algoritmo ci consentirà configurazioni dei valori più libere per altri scopi. Come adattiamo questo algoritmo al riverbero di Schroeder? Basterà assegnare ad *a* il valore 0, in modo da eliminare il suono diretto, a *b* il valore 1 in modo da mantenere in uscita dal *feedforward* la stessa ampiezza dell'entrata, e assegnare a *c* il valore desiderato, in modo simile alla figura 10.3. Questo algoritmo avrà due linee di ritardo anziché una, ma avrà caratteristiche di maggiore libertà nell'uso in altri casi.

USO DEI FILTRI ALLPASS NEL RIVERBERO DI SCHROEDER

Come sappiamo dal par. 6.7, i filtri *allpass* consentono di ritardare un segnale lasciando su tutte le componenti spettrali del suono una risposta in ampiezza inalterata, mentre la risposta di fase, cioè il modo in cui la fase varia a seconda della frequenza, subisce un'alterazione. Va tenuto presente che, mentre con i suoni tenuti l'*allpass* può avere un effetto "trasparente", con attacchi o *decay* improvvisi l'effetto dipendente dallo scostamento di fase diventa udibile[9].

[9] "Dobbiamo ricordare, tuttavia, che la natura "passa-tutto" dei filtri *allpass* è più di natura teorica che percettiva. Non dovremmo supporre, solo perché la risposta in frequenza è assolutamente uniforme, che il filtro sia percettivamente trasparente. In realtà, la risposta di fase del filtro passa-tutto può essere piuttosto complessa. La natura *allpass* implica solo che a lungo termine, con suoni stazionari, l'equilibrio spettrale non verrà modificato. Ciò, invece, non implica nulla del genere nelle regioni transitorie a breve termine." (Moorer, 1979. p.14)
(We must remember, however, that the all-pass nature is more of a theoretical nature than a perceptual one. We should not assume, simply because the frequency response is absolutely uniform, that the filter is perceptually transparent. In fact, the phase response of the all-pass filter can be quite complex. The all-pass nature only implies that in the long run, with steady-state sounds, the spectral balance will not be changed. This implies nothing of the sort in the short-term, transient regions.) [Tutte le traduzioni in questo libro sono a cura dello scrivente].

Nel riverberatore di Schroeder abbiamo visto che i due *allpass* sono disposti in serie; ciò vuol dire che ogni eco generato dai *comb* genera a sua volta una serie di echi in uscita dal primo *allpass* e ognuno di questi ne genera altri in uscita dal secondo filtro *allpass*. Questo è il modo in cui si possono generare riverberi con densità di ripetizioni notevoli. Per il dettaglio del filtro *allpass* di Schroeder rimandiamo al paragrafo 6.7 del secondo volume dove è descritto in modo sufficientemente esaustivo.

EVOLUZIONI SUCCESSIVE DEL RIVERBERO DI SCHROEDER
Alcuni anni dopo i primi algoritmi di riverbero, Schroeder nel 1970 aggiunse al disegno originale una linea di ritardo *multitap* per simulare le *early reflections* (fig. 10.7).

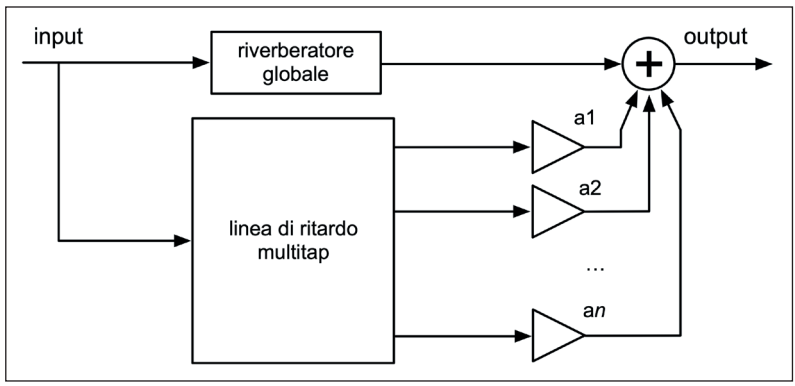


fig. 10.7: riverbero di Schroeder con *multitap delay line*

Ulteriori evoluzioni furono poi introdotte da James A. Moorer, che inserì un filtro passa-basso nei *loop* di *feedback* per modificare il tempo di riverberazione in funzione della frequenza. Vedremo nei prossimi paragrafi algoritmi più complessi ed efficaci. È importante comunque riconoscere a Schroeder di aver fondato le basi del riverbero digitale.

(...)

il capitolo prosegue con:

10.3 FREEVERB

10.4 IL RIVERBERO DI DATTORRO (PLATE REVERB)

10.5 RIVERBERO FDN (FEEDBACK DELAY NETWORK)

10.6 USI CREATIVI DEL RIVERBERO

10.7 SPAZIALIZZAZIONE DEL SUONO CON DUE CANALI
Interaural Time Difference
Interaural Intensity Difference
Head Related Transfer Function
Suoni in movimento: effetto Doppler
La tecnica Mid/Side

10.8 SPAZIALIZZAZIONE MULTICANALE DEL SUONO
Movimento del suono in un sistema multicanale
Movimento del suono "all'interno della stanza"
Sistema Surround 5.1
Tecnica M/S doppia

ATTIVITÀ
• Esempi sonori

VERIFICHE
• Test a risposte brevi

SUSSIDI DIDATTICI
• Concetti di base - Glossario

10P
RIVERBERO E SPAZIALIZZAZIONE

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume I e II, cap. 10T e interludio F

OBIETTIVI

Abilità

- Saper programmare e utilizzare algoritmi di riverbero di Schroeder, Freeverb, Dattorro e FDN
- Saper utilizzare il riverbero in modi creativi, non standard
- Saper programmare e utilizzare algoritmi di spazializzazione con due o più canali
- Saper programmare e utilizzare algoritmi di spazializzazione in surround 5.1

Competenze

- Saper realizzare un breve studio basato sull'uso creativo del riverbero e della spazializzazione

ATTIVITÀ

- Costruzione e modifiche di algoritmi

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista messaggi per oggetti Max specifici

10.1 INTRODUZIONE AGLI ALGORITMI PER IL RIVERBERO

In questo capitolo illustreremo alcuni riverberi algoritmici realizzati con diverse combinazioni di linee di ritardo. Gli elementi fondamentali di questi algoritmi sono:
– i filtri *comb* e *allpass* di Schroeder (ne abbiamo parlato nei paragrafi 6.6 e 6.7 dei capitoli di teoria e di pratica del volume 2 del libro). Utilizzeremo anche un filtro *comb* con il *lowpass* nel circuito di *feedback* simile a quello che abbiamo visto nel paragrafo IF.2 (nell'Interludio F in questo volume)
– il *multitap delay* (paragrafo 6.2 di teoria e di pratica nel secondo volume), che ci servirà per il riverbero di Dattorro
– il *Low Frequency Oscillator* (LFO: capitolo 4 del primo volume) che ci servirà per alterare nel tempo le linee di ritardo allo scopo di evitare pattern ripetitivi nel riverbero.

Si tratta quindi di processi che conosciamo bene: la novità in questo caso consiste nella quantità di linee di ritardo che entrano in gioco e nelle diverse combinazioni degli elementi.

10.2 IL RIVERBERO DI SCHROEDER

Per prima cosa caricate la *patch* **10_01_Schroeder_Rev.maxpat** (figura 10.1) e ascoltate tutti i *preset* osservando i valori dei parametri.

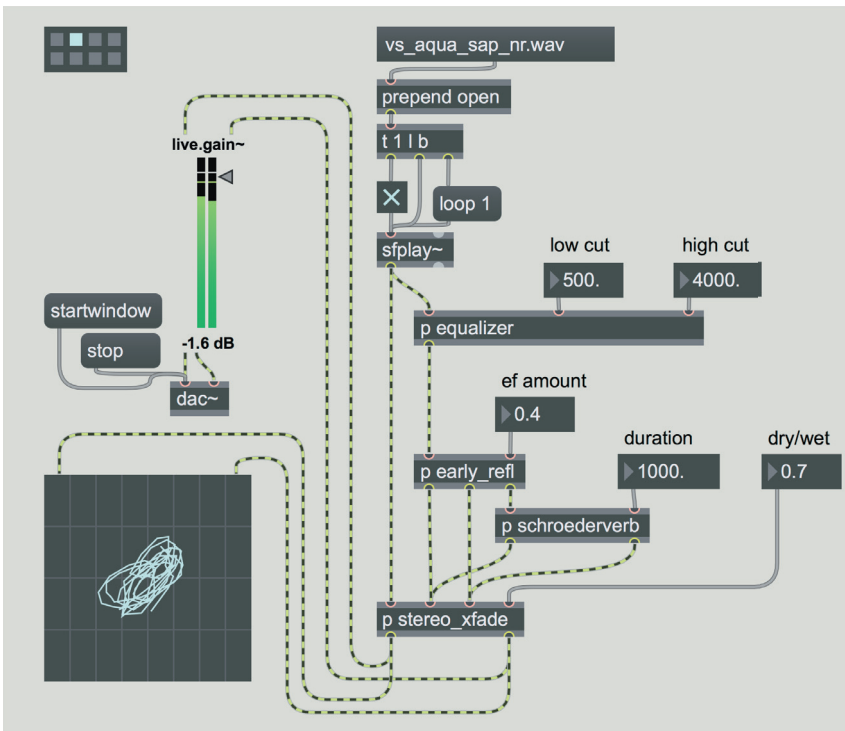


fig. 10.1: *patch* **10_01_schroeder_rev.maxpat**

L'effetto prodotto da questo algoritmo è abbastanza elementare per gli standard odierni, ed è possibile sentire echi ripetitivi e risonanze metalliche nei diversi *preset*. Tuttavia nel periodo in cui è stato proposto da Manfred Schroeder (nei primi anni '60 del secolo scorso) rappresentava un efficace sistema di simulazione digitale di ambienti riverberanti, e ancora oggi è utile proprio per ottenere questi effetti "primitivi" e metallici.
Come possiamo vedere in figura, il segnale passa attraverso quattro *subpatch*. La prima *subpatch* filtra il segnale tagliando alcune frequenze nella regione grave e nella regione acuta: questo filtraggio serve a definire (in maniera molto sommaria) le caratteristiche di assorbimento delle frequenze nell'ambiente che vogliamo simulare, ed è soprattutto utile attenuare le risonanze metalliche che si possono presentare nella zona acuta e un eccessivo rimbombo nella zona grave. L'algoritmo contenuto nella *subpatch* è piuttosto semplice e vi invitiamo ad analizzarlo autonomamente.
La seconda *subpatch* aggiunge le *early reflections* (prime riflessioni, vedi il paragrafo 10.1T): anche in questo caso l'algoritmo è molto semplice (un *delay multitap*); i valori di ritardo sono presi dall'articolo di J.A. Moorer "About This Reverberation Business" pubblicato su *Computer Music Journal* nel 1979[1].
La terza *subpatch* contiene il riverberatore vero e proprio (vedi figura 10.2).

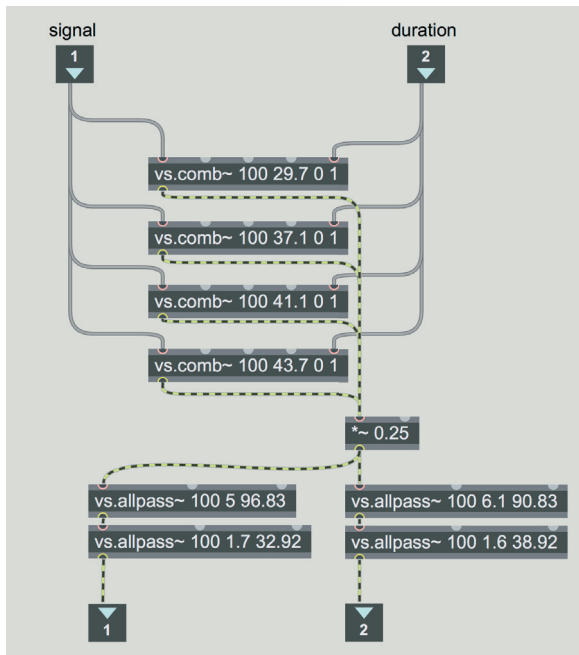


fig. 10.2: il riverbero di Schroeder

[1] (Moorer, J.A., 1979), pp. 13-28

Come già spiegato nel paragrafo 10.2T, nel riverbero di Schroeder ci sono quattro filtri *comb* in parallelo[2] che vengono inviati a due filtri *allpass* in cascata. In questa implementazione abbiamo usato due coppie di filtri *allpass* in cascata, ciascuna coppia con valori di *delay* e *decay* leggermente diversi: in questo modo abbiamo una decorrelazione tra canale sinistro e destro e otteniamo così un effetto stereofonico[3]. Il valore di durata in millisecondi (*number box* "duration" nella *patch* principale) imposta il tempo di *decay* per i quattro filtri *comb*.
Notate che abbiamo usato gli oggetti **`vs.comb~`** e **`vs.allpass~`** della libreria *Virtual Sound Macros* che ci permettono di definire il tempo di *decay* in millisecondi (vi ricordiamo che con gli oggetti standard **`comb~`** e **`allpass~`** il *decay* viene definito da un valore di *feedback*).

ATTIVITÀ

- Create nuovi *preset* con la *patch* di figura 10.1.
- Provate a modificare i valori di *delay* e *decay* dei filtri *comb* e *allpass* nella subpatch [`p schroederverb`], e provate a creare modelli di riverberazione interessanti.
- Provate a variare la quantità di filtri *comb* e *allpass* utilizzati.

(...)

[2] La configurazione di questi filtri *comb* è quella illustrata nel paragrafo 10.1 della teoria, vedi in particolare la descrizione dei parametri per la figura 10.6.

[3] I parametri utilizzati sono presi, con qualche modifica, dal libro di C. Dodge e T. A. Jerse, *Computer Music* (pag. 301).

il capitolo prosegue con:

10.3 FREEVERB

10.4 IL RIVERBERO DI DATTORRO (PLATE REVERB)

10.5 RIVERBERO FDN (FEEDBACK DELAY NETWORK)

10.6 USI CREATIVI DEL RIVERBERO

10.7 SPAZIALIZZAZIONE DEL SUONO CON DUE CANALI
Effetto Doppler
Codifica Mid/Side

10.8 SPAZIALIZZAZIONE MULTICANALE DEL SUONO
Movimento del suono in un sistema multicanale
Movimento del suono in un sistema 5.1
Tecnica M/S doppia

ATTIVITÀ
• Analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi, correzione di algoritmi

VERIFICHE
• Compiti unitari di reverse engineering

SUSSIDI DIDATTICI
• Lista oggetti Max - Lista messaggi per oggetti Max specifici - Lista operatori Gen

11T
SINTESI NON LINEARE

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti dei capp. 1-10

OBIETTIVI

Conoscenze

- Conoscere la teoria e l'uso della sintesi in modulazione d'ampiezza e ad anello
- Conoscere l'uso del DC Offset nella modulazione
- Conoscere la teoria e l'uso della modulazione a banda laterale singola
- Conoscere la teoria e l'uso della sintesi in modulazione di frequenza e di fase
- Conoscere la teoria e l'uso della tecnica di *feedback* PM
- Conoscere le nozioni di base sulle famiglie spettrali mediante FM
- Conoscere la teoria e l'uso della distorsione di fase e della distorsione non lineare (*waveshaping*)
- Conoscere la teoria e l'uso della tecnica del *waveterrain*

Abilità

- Riconoscere all'ascolto suoni generati con varie tecniche di sintesi non lineare

CONTENUTI

- Tecniche di sintesi non lineare per modulazione
- Modulazione d'ampiezza, ad anello e a banda laterale singola
- Modulazione di frequenza con modulanti e/o portanti multiple
- Famiglie spettrali
- Modulazione di fase e *feedback* PM
- Distorsione di fase
- *DNL* (distorsione non lineare) o *Waveshaping* e distorsori
- *Wave Terrain Synthesis*

ATTIVITÀ

- Esempi sonori

VERIFICHE

- Test a risposte brevi

SUSSIDI DIDATTICI

- Concetti di base - Glossario

LA SINTESI NON LINEARE

Le diverse tecniche di sintesi non lineare si distinguono dalle tecniche lineari per due caratteristiche: l'arricchimento dello spettro e/o la sua traslazione. Le tecniche lineari, come la sintesi additiva o la sintesi sottrattiva, infatti producono in uscita un segnale che non contiene componenti diverse o alterate in frequenza rispetto a quelle presenti nei segnali in ingresso, mentre tutto ciò è realizzabile con le tecniche non lineari. Un esempio tipico, come vedremo, è quello della modulazione di frequenza, mediante la quale con due soli oscillatori sinusoidali si possono ottenere suoni estremamente complessi, cosa impossibile con una tecnica lineare come la sintesi additiva.

11.1 TECNICHE DI MODULAZIONE DELL'AMPIEZZA: AM, RM E SSB

Una modulazione è l'alterazione dell'ampiezza istantanea, della frequenza istantanea o della fase di un segnale provocata da un altro segnale. La modulazione può avere luogo fra due oscillatori (nel caso più semplice) o può essere utilizzata in algoritmi molto complessi; è inoltre possibile modulare anche suoni campionati. Come accennato nel par. 4.4T del primo volume, se abbiamo due oscillatori in un algoritmo di modulazione, chiameremo **portante** (*carrier* in inglese) l'oscillatore che viene modulato, e modulante (*modulator*) l'oscillatore che modula.

Nel par. 4.5 abbiamo già visto un esempio di tremolo realizzato mediante un algoritmo di modulazione dell'ampiezza. In quel caso il segnale modulante proveniva da un LFO (quindi con una frequenza al di sotto della banda audio) e presentava un'ampiezza limitata rispetto a quella della portante. L'effetto provocato dall'oscillatore modulante era quello di variare leggermente in modo ciclico l'ampiezza del segnale portante. Tale effetto veniva percepito nel dominio del tempo.
Consideriamo ora il caso in cui la frequenza dell'oscillatore modulante sia più alta e la sua ampiezza maggiore: avremo un fenomeno diverso dal punto di vista percettivo, cioè la comparsa di frequenze nuove (dette **bande laterali**) che si aggiungono allo spettro della portante. Esse appaiono in modo simmetrico sopra (**USB**, *Upper Sideband*) e sotto (**LSB**, *Lower Sideband*) la frequenza della portante, come vedremo nel prossimo paragrafo. L'effetto in questo caso verrà percepito nel dominio della frequenza.
A partire da un tremolo (quindi da un LFO), aumentiamo la frequenza della modulante: quale sarà il confine in cui si perde la percezione del tremolo e cominciano a percepirsi frequenze laterali distinte? Sappiamo che normalmente, se la frequenza della modulante è sotto i 10 Hz, il nostro apparato uditivo riesce a distinguere le singole variazioni d'ampiezza (un tremolo, appunto), ma non le frequenze laterali come distinte da quella centrale; ciò avviene perché tali frequenze laterali ricadono all'interno della banda critica, cioè esistono, ma sono troppo vicine a quella centrale per essere percepite come separate.

Proviamo ad aumentare la frequenza della modulante oltre i 10 Hz: portando questa frequenza oltre la metà della banda critica[1], inizieremo a percepire le bande laterali e a perdere la percezione delle singole variazioni d'ampiezza. Si crea dunque una zona grigia, simile a quella che percepivamo con i battimenti all'aumentare della distanza fra le due frequenze (vedi par. 2.2).

ESEMPIO SONORO 11.1
Da suono non modulato a tremolo fino a bande laterali distinte

Possiamo considerare 3 tecniche in cui viene modulata l'ampiezza istantanea:

– **Ring Modulation (RM)** (**modulazione ad anello**)
– **Amplitude Modulation (AM)** (**modulazione d'ampiezza**)
– **Single Sideband modulation (SSB)** (**modulazione a banda laterale singola**)

La differenza fra l'AM e la RM sta nel fatto che mentre la modulazione d'ampiezza utilizza un segnale modulante unipolare, la modulazione ad anello utilizza un segnale modulante bipolare. Come sappiamo fin dal par.1.1, i segnali **bipolari** oscillano fra valori positivi e valori negativi d'ampiezza, mentre i segnali **unipolari** oscillano solo nel campo positivo (o solo nel campo negativo).
Nel cap. 4 abbiamo utilizzato sia segnali modulanti bipolari, sia quelli unipolari, ma in quei casi tali segnali non erano mai nella banda audio. In questo capitolo i segnali modulanti non saranno più limitati alla banda sub-audio.
Nella SSB, utilizzata in particolare per realizzare *frequency shifter* in tempo reale, viene utilizzata solo una delle due bande laterali della modulazione ad anello.

MODULAZIONE AD ANELLO (RM)

Il nome modulazione ad anello (o ring modulation) proviene dalla sua originaria implementazione analogica in cui si usava un "anello" (in inglese ring) di diodi per moltiplicare il segnale di input con una onda quadra bipolare.[2]

L'algoritmo per la **modulazione ad anello** più comune nella programmazione prevede la moltiplicazione nel dominio del tempo[3] di due segnali audio bipolari,

[1] Possiamo percepire come distinti due suoni solo quando essi coinvolgono terminazioni nervose sufficientemente distanti, e quindi ricadono in due bande critiche distinte. Sotto i 200 Hz la larghezza delle bande critiche è più o meno costante, ma da 200 Hz in su circa, la banda critica aumenta la sua larghezza all'aumentare della frequenza. Per il calcolo dei valori di banda critica vedi par. 2.2T.
[2] La *RM*, come l'*AM*, la *SSB* e la *FM*, veniva utilizzata prevalentemente nella tecnica delle trasmissioni radio. Nello studio della WDR di Colonia nacque l'idea di utilizzare la modulazione ad anello per la modifica del suono. Karlheinz Stockhausen la utilizzò in *Momente*, *Mixtur*, *Mantra* e *Mikrophonie I*.
[3] Vedremo nel Cap. 13T come la moltiplicazione di due segnali nel dominio della frequenza (cioè della rappresentazione spettrale dei segnali) equivale invece a una convoluzione.

che solo per comodità chiameremo portante e modulante, perché in realtà, essendo la moltiplicazione un'operazione commutativa, i due segnali svolgono lo stesso ruolo qualunque sia l'ordine in cui li moltiplichiamo. In fig. 11.1 abbiamo l'esempio di moltiplicazione di due segnali sinusoidali, ma si possono scegliere segnali più complessi, come vedremo.

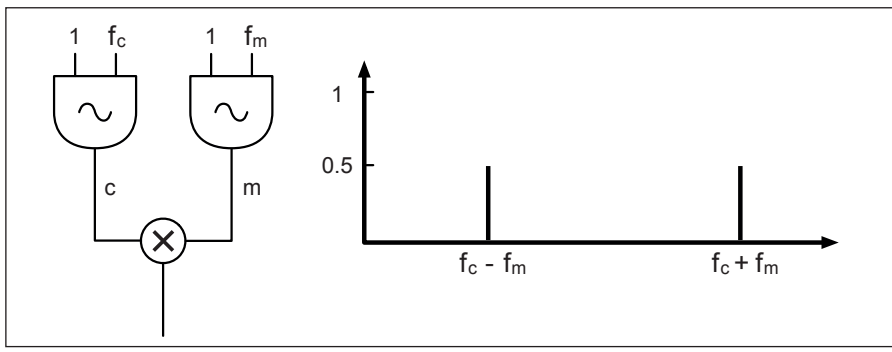


fig. 11.1: modulazione ad anello mediante moltiplicazione di segnali bipolari sinusoidali

Nella modulazione ad anello con forme d'onda sinusoidali, se f_C è 700 Hz e f_M è 200 Hz, in uscita avremo due bande laterali, composte da due sinusoidi le cui frequenze sono la somma e la differenza delle frequenze in ingresso[4]:

$f_C - f_M$	$f_C + f_M$
700 - 200 = 500 Hz	**700 + 200 = 900 Hz**

Per spiegare questo fenomeno possiamo rifarci alla seconda formula di Werner[5]

$$\cos\alpha\cos\beta = \frac{1}{2}\left[\cos(\alpha+\beta)+\cos(\alpha-\beta)\right]$$

Moltiplicando due coseni, rispettivamente di angolo α e β, il risultato sarà uguale al coseno della somma degli angoli ($\alpha + \beta$, corrispondente, nel nostro caso, alla banda superiore) più il coseno della differenza degli angoli ($\alpha - \beta$, corrispondente alla banda inferiore) il tutto diviso 2.

[4] Infatti il nome utilizzato nel campo della comunicazione è *double-sideband suppressed-carrier modulation* (in italiano *modulazione a doppia banda laterale e portante soppressa*).

[5] Johann Werner (1468-1522) è stato un matematico e cartografo tedesco. Le formule di Werner, nate nel campo della trigonometria, sono utilizzate, fra l'altro, per descrivere la formazione di bande laterali nell'AM anche nel campo della radiotecnica.

Come accennato, possiamo modulare un suono campionato, o un suono in tempo reale utilizzato come portante. Possiamo anche moltiplicare tra loro due segnali complessi di sintesi, o anche campionati, senza uso di oscillatori. In tutti questi casi avremo in uscita un suono contenente frequenze somma e differenza di tutte le componenti spettrali dei due suoni.

Facciamo l'esempio in cui la portante ***c*** contenga 3 componenti e la modulante ***m*** contenga 2 componenti.

c	***m***
800, 1600, 2400 Hz	**250, 500 Hz**

Mediante la modulazione ad anello otterremo in uscita 12 componenti. Per conoscere il numero di componenti in uscita, infatti, basta moltiplicare il numero di componenti di *c* per il numero di componenti di *m* per 2 (in questo caso 3*2*2). È possibile, però, che in alcuni casi, se ci sono componenti in rapporto armonico fra loro, vengano generate frequenze uguali che in questo caso sommeranno algebricamente le loro ampiezze e quindi il numero delle componenti sarà minore.

Nel caso descritto sopra avremo in uscita le seguenti frequenze (vedi spettro in fig. 11.2):

***c+m* = 1050, 1300, 1850, 2100, 2650, 2900 Hz**
***c-m* = 300, 550, 1100, 1350, 1900, 2150 Hz**

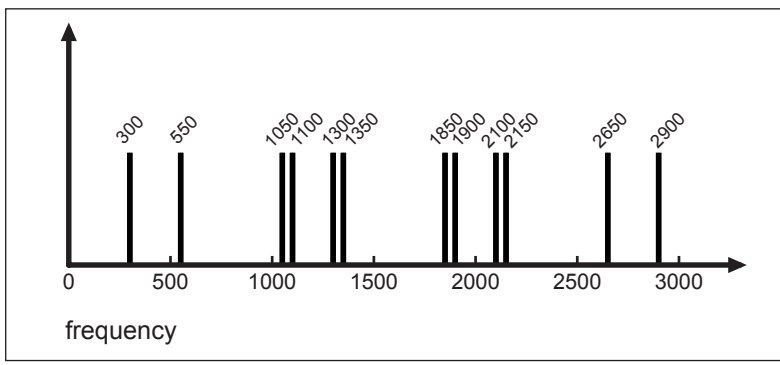


fig. 11.2: spettro in uscita dall'algoritmo di modulazione con *m* e *c* complessi

Per quanto concerne le ampiezze, nel caso di due sinusoidi, ogni componente in uscita avrà un'ampiezza che è il risultato dell'ampiezza di *c*m/2*. Nel caso di suoni complessi, ogni componente in uscita avrà ampiezza uguale a quella delle componenti di *c* e *m* che l'hanno generata moltiplicate fra loro e poi divise per 2. Se l'ampiezza di *m* (o di *c*) è 0 non avremo alcun segnale in uscita.

Si possono ottenere anche frequenze con valore negativo, nel caso in cui la frequenza della modulante sia maggiore della frequenza della portante.

Ad esempio se la f_C è 700 Hz, e diamo a f_M il valore 1000, otterremo questi valori di frequenza in uscita:

$f_C - f_M$	$f_C + f_M$
-300	**1700**

Come spiegato nel par. 5.1 del secondo volume, le frequenze sotto lo zero ricompaiono con segno invertito e a specchio nel campo positivo (-200 Hz diventa 200 Hz, -300 Hz diventa 300 Hz etc.). Ciò avviene anche per eventuali frequenze laterali che superino la frequenza di Nyquist.

MODULAZIONE D'AMPIEZZA (AM)

Iniziamo da un esempio base di AM: a questo scopo utilizziamo un segnale sinusoidale *unipolare* in banda audio (ad es. 300 Hz) per modulare l'ampiezza di un oscillatore portante sinusoidale (ad es. 700 Hz). Osserviamo prima il comportamento del segnale dal punto di vista della frequenza: a differenza della modulazione ad anello, in questo caso otterremo in uscita anche la frequenza della portante, 700 Hz oltre alle due frequenze laterali. Come nella modulazione ad anello, la prima avrà un valore pari a quello della frequenza della portante più quello della frequenza della modulante (in questo caso 700+300 = 1000 Hz); la seconda avrà un valore pari a quello della frequenza della portante meno quello della frequenza della modulante (in questo caso 700-300 = 400 Hz).
Chiamiamo f_C (*carrier frequency*) la frequenza della portante, e f_M (*modulator frequency*) la frequenza della modulante; la larghezza di banda del suono in uscita, cioè la differenza tra la componente più grave e quella più acuta, sarà uguale a 2* f_M.
Nel segnale in *output* avremo tre frequenze, quella centrale della portante e le due laterali date dalla modulazione:

$f_C - f_M$	f_C	$f_C + f_M$
400	**700**	**1000**

Se la frequenza della portante e quella della modulante sono in rapporto armonico fra loro, otterremo un suono armonico, altrimenti avremo in uscita un suono inarmonico. Se f_M = 200 Hz e f_C = 400 Hz otterremo i seguenti valori di frequenza in uscita (cioè un suono armonico con fondamentale 200 Hz con seconda e terza armonica):

$f_C - f_M$	f_C	$f_C + f_M$
200	**400**	**600**

Consideriamo ora le ampiezze: innanzitutto notiamo che per ottenere un segnale unipolare (cioè il segnale usato come modulante) dobbiamo sommare un segnale bipolare a un DC Offset.[6]

[6] Come sappiamo, il *DC Offset* è una componente fissa a 0 Hz, perciò non crea bande laterali.

Nello schema classico dell'AM il valore di questo *DC Offset* è uguale a quello dell'ampiezza della modulante sinusoidale. Ad esempio, avendo l'ampiezza della modulante uguale a 1, che quindi oscilla fra 1 e -1, aggiungendo un *DC Offset* che ha lo stesso valore (1) otterremo un segnale unipolare che oscilla fra 0 e 2.
In uscita, dopo la modulazione, otterremo un'ampiezza della componente centrale pari a quella del *DC Offset* (1) e due frequenze laterali le cui ampiezze saranno pari alla metà di quella centrale (.5). Dato che sommando le ampiezze delle bande laterali con quella della portante possiamo superare l'ampiezza massima, dovremo riscalare l'ampiezza in uscita. La figura 11.3 riassume la situazione.

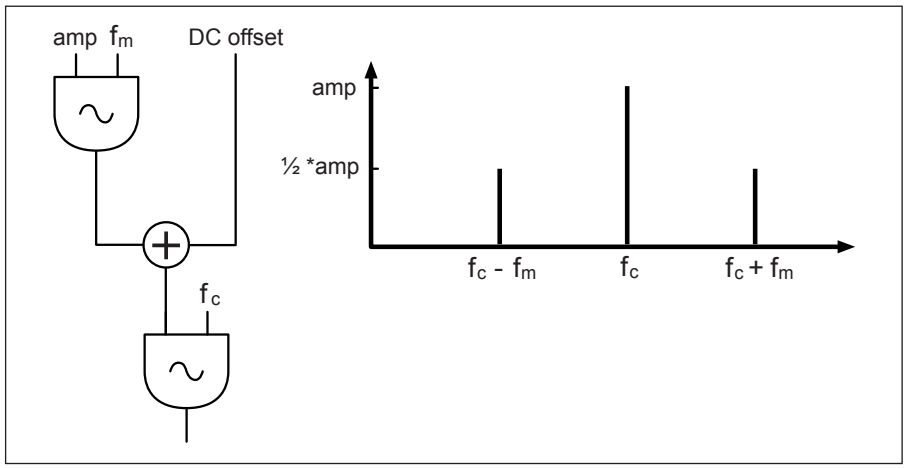


fig. 11.3: diagramma di flusso e spettro risultanti da AM in configurazione classica

In sostanza, come accennato, il risultato è simile a quello della RM per le frequenze laterali, ma in uscita otterremo anche il segnale alla frequenza della portante perché nella modulante è presente un *DC Offset*, il quale determina l'ampiezza della componente della portante in uscita. Come sappiamo il *DC offset* può essere considerato una componente a 0 Hz; abbiamo quindi due segnali che modulano la portante: la modulante e il *DC Offset*. Possiamo quindi dire che la "modulazione" tra quest'ultimo e la portante genera in uscita la frequenza della portante stessa:

$$f_C \pm 0 = f_C$$

Questo spiega perché nella AM compare la portante in uscita e nella RM no.[7]

[7] In molti casi si utilizza anche un parametro chiamato *indice di modulazione*: si tratta di un moltiplicatore dell'ampiezza della modulante, il quale consente di variare il rapporto fra l'ampiezza della modulante e il *DC Offset*. Se l'indice di modulazione è uguale a 1, l'ampiezza della modulante sarà uguale al valore del *DC Offset*. A parità di *DC Offset* e variando l'indice di modulazione, modificheremo il rapporto tra le ampiezze delle bande laterali e l'ampiezza della portante.

Partendo da quest'ultima considerazione possiamo rendere indipendenti l'ampiezza del *DC Offset* e quella della modulante e ottenere un algoritmo che permetta di variare indipendentemente l'ampiezza in uscita delle bande laterali e della portante, anche tramite inviluppi, e ottenere una variazione continua dello spettro tra RM e AM. Vediamo il modello in figura 11.4.

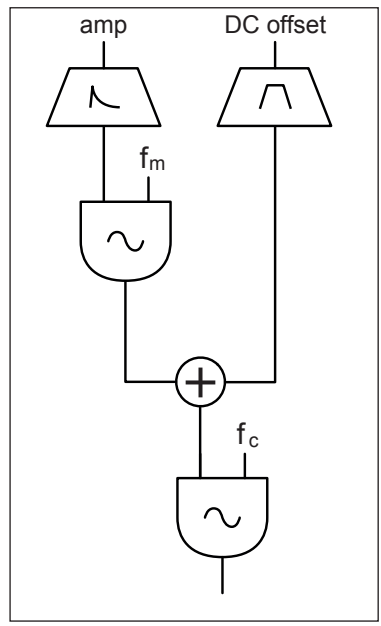


fig. 11.4: algoritmo AM con *DC Offset* indipendente dall'ampiezza della modulante

ESEMPIO SONORO 11.2
Passaggio continuo da AM a RM e viceversa

(...)

il capitolo prosegue con:

Modulazione a banda laterale singola (ssb) e frequency shifting

11.2 MODULAZIONE DI FREQUENZA E DI FASE: FM, PM E FEEDBACK PM
FM: teoria di base
Famiglie spettrali
Famiglie di rapporti c:m
FM: modulazioni complesse
Uso di regole matematiche nel brano *Stria* di John Chowning
Modulazione di fase (PM)
Feedback PM

11.3 DISTORSIONE DI FASE

11.4 DISTORSIONE NON LINEARE (DNL) O WAVESHAPING
Distorsione non lineare di sinusoidi
Uso dei polinomi di Chebyshev per la DNL
Distorsione di segnali complessi
Altri tipi di distorsione non lineare

11.5 WAVE TERRAIN SYNTHESIS (WTS)

11.6 SPLIT SYNTHESIS

ATTIVITÀ
• Esempi sonori

VERIFICHE
• Test a risposte brevi

SUSSIDI DIDATTICI
• Concetti di base - Glossario

11P
SINTESI NON LINEARE

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume I e II, interludio F, Cap. 10 e Cap.11

OBIETTIVI

Abilità

- Saper programmare e utilizzare algoritmi di AM, RM, PM, FM
- Saper programmare e utilizzare algoritmi di distorsione di fase, waveshaping, waveterrain, synthesis e distorsori

Competenze

- Saper realizzare un breve studio basato sull'uso creativo delle sintesi non lineari

ATTIVITÀ

- Costruzione e modifiche di algoritmi

SUSSIDI DIDATTICI

- Lista di oggetti Max - Lista attributi e messaggi Max specifici - Lista operatori gen

11.1 TECNICHE DI MODULAZIONE DELL'AMPIEZZA: AM, RM E SSB

RING MODULATION

Realizzare una modulazione ad anello (*ring modulation* o RM) tra due suoni sinusoidali è molto semplice, è sufficiente moltiplicare tra loro le uscite di due oscillatori (figura 11.1).

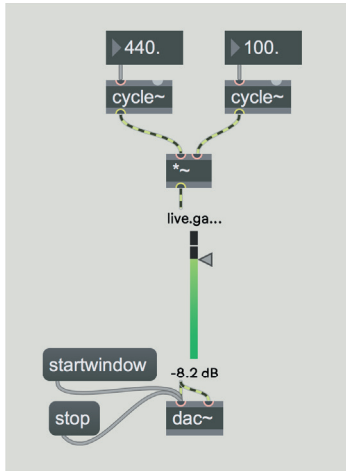


fig. 11.1: *ring modulation* tra due oscillatori sinusoidali

Ricostruendo la *patch* in figura, e impostando i valori di frequenza di 440 e 100 Hz per i due oscillatori, otteniamo in uscita due suoni sinusoidali di frequenza 340 (cioè 440 - 100) e 540 (cioè 440 + 100) Hz. Provate ad incrementare o diminuire gradualmente la frequenza del secondo oscillatore e udrete i due suoni risultanti allontanarsi o avvicinarsi tra loro.
Il fenomeno della *ring modulation* sembrerebbe applicarsi solo a frequenze che si trovano nel *range* audio (20-20.000 Hz). Provate infatti a portare la frequenza del secondo oscillatore a 1 Hz, trasformandolo quindi in un LFO: il suono risultante è una sinusoide di 440 Hz la cui ampiezza segue l'andamento dell'LFO. In questo caso, apparentemente, non abbiamo più in uscita la somma e la differenza di due frequenze, ma solo una singola frequenza la cui ampiezza varia nel tempo.
In realtà questa distinzione è dovuta solo ai limiti della nostra percezione, perché naturalmente le formule trigonometriche (formule di Werner) che governano la RM sono valide indipendentemente dalle frequenze utilizzate. Se applichiamo queste formule al caso descritto, infatti, abbiamo come frequenze risultanti 439 (cioè 440 - 1) e 441 (440 + 1) Hz; essendo due frequenze molto vicine danno luogo al fenomeno dei battimenti, il cui risultato percettivo in questo caso è un suono a 440 Hz che oscilla 2 volte al secondo. La differenza tra 439 e 441 è infatti 2, mentre il valore medio tra le due frequenze è 440 (cioè (439+441)/2, vedi paragrafo 2.2 del primo volume).

Una RM tra due sinusoidi di 440 e 1 Hz produce quindi lo stesso effetto che si ha sommando due sinusoidi di 439 e 441 Hz rispettivamente: ricostruite la *patch* di figura 11.2 e verificate che il suono prodotto dall'RM e dalla somma delle sinusoidi sia identico.

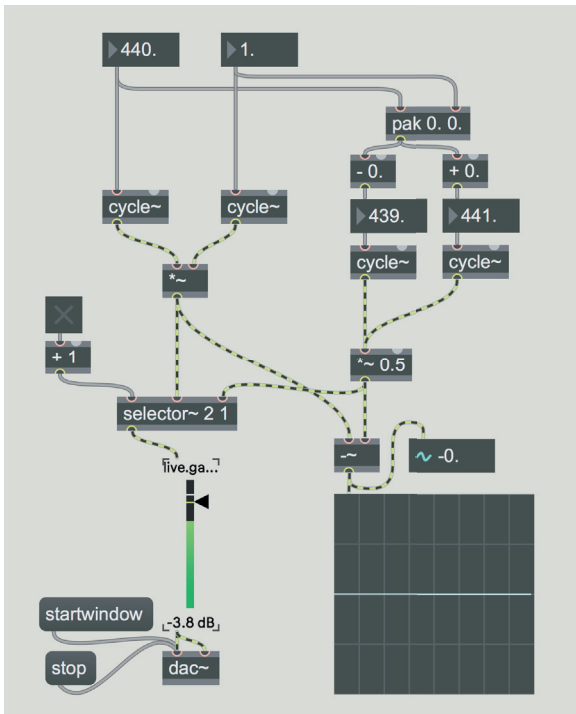


fig. 11.2: RM e battimenti

Notate che abbiamo dimezzato la somma delle due sinusoidi, in modo che il segnale risultante vari tra -1 e 1, come nel caso della moltiplicazione, e non tra -2 e 2. Alternando l'ascolto tra il primo e il secondo segnale (tramite il `selector~`) non si dovrebbe sentire alcun click di discontinuità, perché i due segnali sono identici. Come si vede in figura, inoltre, possiamo verificare se due segnali sono uguali sottraendoli tra loro: se il risultato è un segnale costante pari a 0, i due segnali sono equivalenti.

Per ottenere un suono più complesso possiamo moltiplicare più sinusoidi in cascata. La frequenza di ogni nuova sinusoide viene aggiunta e sottratta a tutte le componenti già esistenti, raddoppiandone il numero. La quantità di componenti risultanti per *n* oscillatori moltiplicati tra loro è quindi pari a 2^{n-1}. Con 4 oscillatori abbiamo, ad esempio, 8 componenti (cioè 2^3): ricostruite la *patch* di figura 11.3.

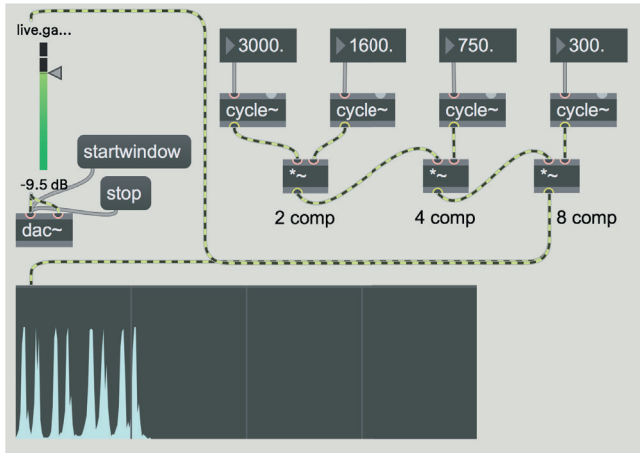


fig. 11.3: RM multipla

Come sappiamo dal capitolo di teoria è possibile effettuare la RM tra un suono concreto (ad esempio un suono strumentale) e un oscillatore sinusoidale, con risultati sicuramente più interessanti di quelli realizzabili con semplici sinusoidi. Caricate la *patch* **11_01_instr_RM.maxpat** (figura 11.4).

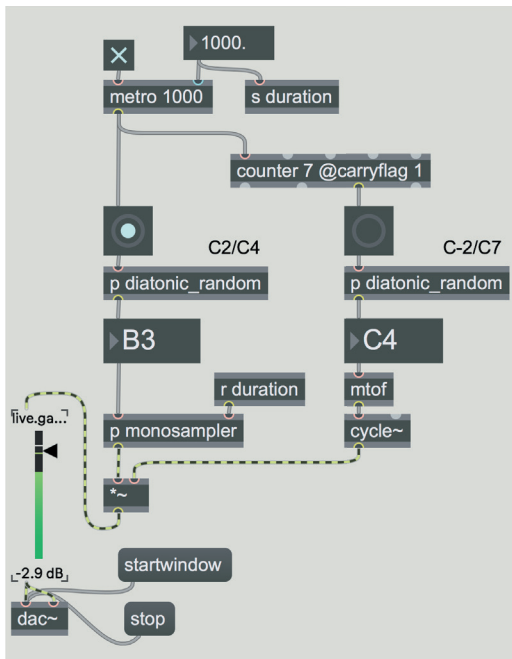


fig. 11.4: *patch* **11_01_instr_RM.maxpat**

In questa *patch* vengono generate delle note casuali diatoniche (corrispondenti ai tasti bianchi del pianoforte) ad ogni *bang* dell'oggetto `metro`: la *subpatch* [`p` monosampler] contiene un piccolo campionatore monofonico che riproduce

il suono di un vibrafono. Un altro generatore casuale di note diatoniche (che genera un nuovo valore ogni 8 *bang*) è collegato a un oscillatore sinusoidale che effettua una *ring modulation* con il suono campionato. Ascoltate attentamente la variazione timbrica prodotta dalla *patch*: notate che ogni volta che cambia la frequenza dell'oscillatore sinusoidale si ha una nuova "famiglia" di timbri simili. Vediamo ora cosa succede facendo la RM tra un generatore di rumore a frequenza variabile e un oscillatore sinusoidale: caricate la *patch* **11_02_noise_RM.maxpat** (figura 11.5).

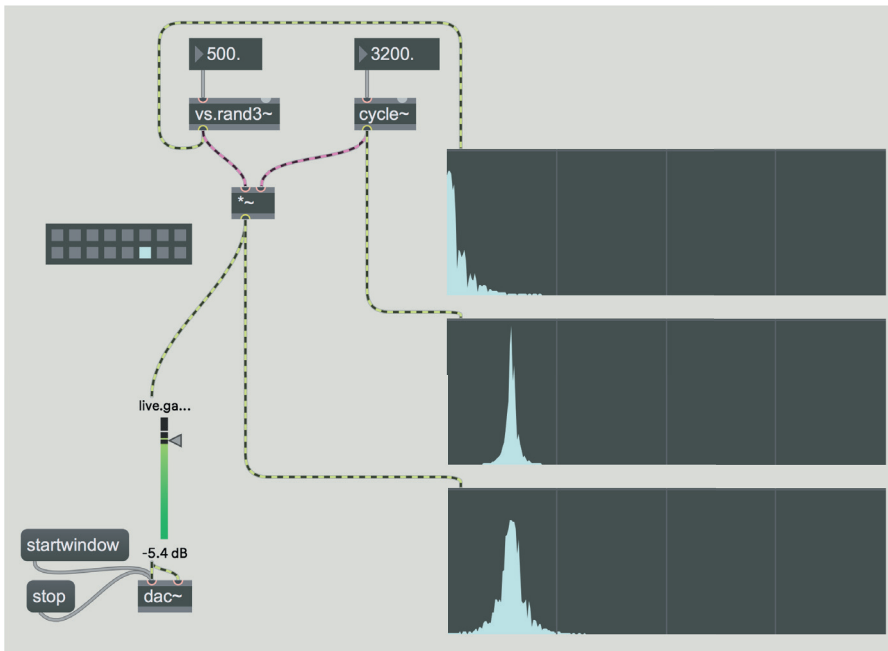


fig. 11.5: *patch* **11_02_noise_RM.maxpat**

Come sappiamo dal paragrafo 3.1 del primo volume, un generatore di rumore a frequenza variabile produce bande di frequenze la cui larghezza è proporzionale alla frequenza impostata. Se effettuiamo una RM di questo segnale con una sinusoide, otteniamo una traslazione di queste bande al di sopra, e specularmente al di sotto, della frequenza della sinusoide. Vediamo, nei tre spettroscopi di figura 11.5, in alto lo spettro del generatore di rumore, al centro lo spettro (un singolo picco) dell'oscillatore sinusoidale, e in basso lo spettro positivo e negativo del generatore di rumore traslato sulla frequenza della sinusoide.
Provate a diminuire gradualmente la frequenza del generatore di rumore: le bande dello spettro si restringono attorno alla frequenza dell'oscillatore sinusoidale, il quale diventa sempre più chiaramente percepibile, con un effetto simile a quello ottenuto dall'innalzamento del fattore Q in un filtro passa-banda che filtra un rumore bianco.

ATTIVITÀ

– Nella *patch* di figura 11.5 sostituite l'oscillatore sinusoidale con un suono campionato e osservate come il generatore di rumore altera lo spettro della sorgente sonora.
– Fate la RM tra un suono campionato e una sinusoide alla frequenza di Nyquist (ad esempio [`cycle~ 22050`] se la frequenza di campionamento è 44100 Hz). Osservate con lo spettroscopio che si ottiene uno spettro rovesciato (in pratica le frequenze più acute prendono il posto di quelle più gravi e viceversa): sapreste spiegare perché? Rileggete le sezioni su *foldover* e *aliasing* che si trovano al paragrafo 5.1 del secondo volume).

MODULAZIONE D'AMPIEZZA

Vediamo subito una *patch* che ci permette di miscelare indipendentemente la portante e la modulante in una AM: caricate il file **11_03_instr_AM.maxpat** (figura 11.6).

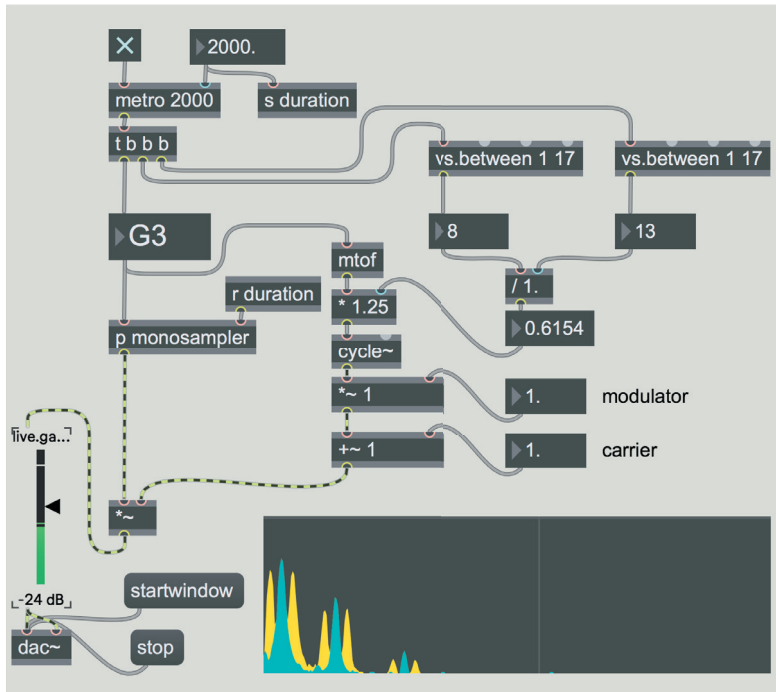


fig. 11.6: file **11_03_instr_AM.maxpat**

In questa *patch* la portante è il suono di vibrafono che abbiamo già usato in precedenza, mentre la modulante è un oscillatore sinusoidale, la cui frequenza è calcolata in rapporto alla frequenza della portante. Questo rapporto è espresso con un numero frazionario in cui sia il denominatore, sia il numeratore sono valori casuali compresi tra 1 e 16.

L'uscita della modulante è prima moltiplicata per il valore espresso dal *number box* "modulator", al segnale ottenuto viene aggiunto un valore costante (*DC Offset*) tramite il *number box* "carrier". Modificando il valore del *number box* "modulator" possiamo variare l'ampiezza delle bande laterali, mentre il *number box* "carrier" varia l'ampiezza del suono della portante.
In basso a destra è visibile lo spettro del suono risultante: le componenti della portante sono colorate in azzurro, mentre le bande laterali generate dalla modulazione sono in giallo.
Lasciamo al lettore l'analisi della *patch*: notate che se azzeriamo il valore "carrier" lasciando il "modulator" a 1, otteniamo una RM; viceversa se azzeriamo il valore "modulator" e lasciamo "carrier" a 1 otteniamo semplicemente il suono della portante. Altre combinazioni di questi valori ci permettono di ottenere timbri più o meno inarmonici.
Possiamo gestire i parametri "modulator" e "carrier" con degli inviluppi, in modo da variare dinamicamente il timbro: vediamo un esempio, caricate la *patch* **11_04_instr_AM_env.maxpat** (figura 11.7).

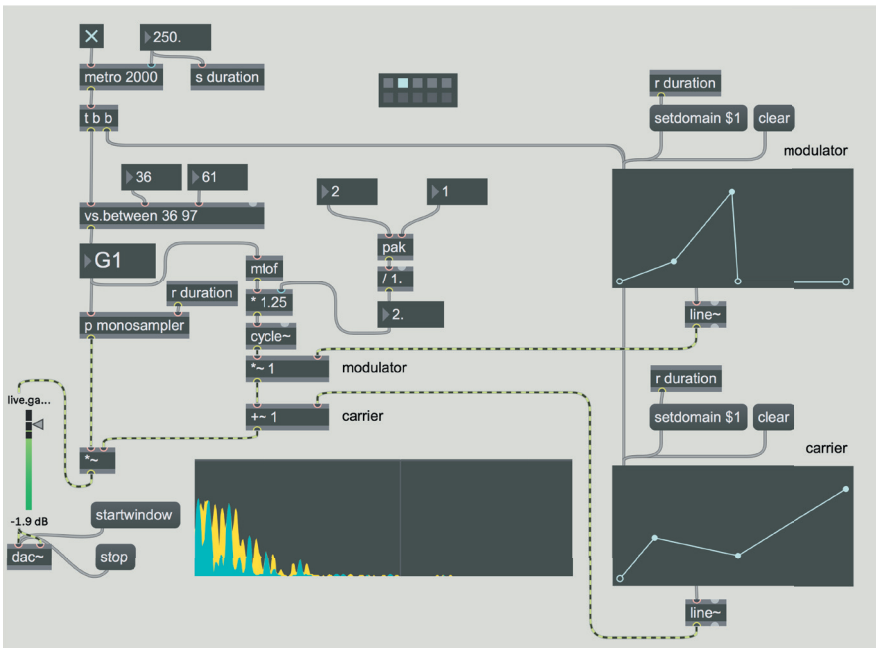


fig. 11.7: *patch* **11_04_instr_AM_env.maxpat**

Questa *patch* è una variazione della precedente ed è abbastanza facile da analizzare. Ascoltate i diversi *preset* e notate che il rapporto di frequenza tra modulante e portante non viene generato casualmente, ma è fisso per ogni *preset*, mentre la nota eseguita dalla modulante viene generata casualmente. Create altri *preset*.

(...)

il capitolo prosegue con:

Modulazione a banda laterale singola (frequency shifting)

11.2 MODULAZIONE DI FREQUENZA E DI FASE: FM, PM E FEEDBACK PM
Famiglie spettrali
Modulazione con feedback
Modulazioni complesse

11.3 DISTORSIONE DI FASE

11.4 DISTORSIONE NON LINEARE (DNL) O WAVESHAPING
Polinomi di Chebyshev
DNL senza lookup table

11.5 WAVE TERRAIN SYNTHESIS (WTS)
Curve di Lissajous
Curva rodonea
Funzioni a 2 variabili per il terrain
Orbite quasi periodiche e aperiodiche

11.6 SPLIT SYNTHESIS

ATTIVITÀ
• Analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi, correzione di algoritmi

VERIFICHE
• Compiti unitari di reverse engineering

SUSSIDI DIDATTICI
• Lista oggetti Max - Lista messaggi e attributi per oggetti Max specifici - Lista operatori Gen

Questa pagina è intenzionalmente lasciata in bianco

12T
MICROSUONO

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume I e II, capp. 10 e 11

OBIETTIVI

Conoscenze

- Conoscere i fondamenti e le varie tipologie della sintesi granulare sincrona
- Conoscere i meccanismi su cui si basano i mutamenti timbrici e di altezza nella sintesi granulare sincrona
- Conoscere le tecniche di sintesi granulare asincrona e per formanti
- Conoscere i vari metodi di granulazione dei suoni campionati e sintesi particellare
- Conoscere i concetti di flusso e nuvola e le configurazioni intermedie
- Conoscere le caratteristiche qualitative dei parametri e la loro randomizzazione
- Conoscere i concetti base di altre tecniche nel campo del microsuono: *multi-source brassage* e *micromontage*

Abilità

- Saper individuare all'ascolto i mutamenti dei vari parametri della sintesi granulare e particellare
- Saper individuare all'ascolto i mutamenti dei vari parametri della granulazione dei suoni campionati

CONTENUTI

- Sintesi granulare sincrona, asincrona e per formanti
- *Glisson synthesis, grainlet synthesis, trainlet synthesis* e *pulsar synthesis*
- Granulazione di suoni campionati
- *Multi-source brassage* e *micromontage*

ATTIVITÀ

- Esempi sonori

SUSSIDI DIDATTICI

- Concetti di base - Glossario

"Al di sotto del livello delle note troviamo il regno del microsuono, quello delle particelle di suono.
Le particelle microacustiche sono rimaste invisibili per secoli. Gli sviluppi tecnologici più recenti ci consentono di esplorare le bellezze di questo mondo che un tempo non poteva essere visto. Le tecniche microsoniche fanno dissolvere i rigidi mattoni dell'architettura musicale – le note – in un mezzo più fluido e flessibile. I suoni possono fondersi, evaporare, o trasformarsi in altri suoni. (…)

Quando le particelle si allineano in rapida successione, generano l'illusione di una continuità del suono che chiamiamo altezza. Quando le particelle vagano, fluiscono in ruscelli e rivoli. Agglomerazioni dense di particelle formano nuvole di suono roteanti le cui forme si evolvono nel tempo…" (Curtis Roads)[1]

Questo capitolo è incentrato sulla costruzione, mediante varie tecniche, di fenomeni microsonori (cioè di breve durata). Diversi autori hanno realizzato esperimenti in questo campo e pezzi di musica basati su microsuoni, fra cui il fisico Dennis Gabor[2] e i compositori Iannis Xenakis[3], Curtis Roads, Barry Truax, Horacio Vaggione, Trevor Wishart.

12.1 SINTESI GRANULARE

La **sintesi granulare** è una tecnica di creazione di segnali sonori basati sulla combinazione di particelle di suono, dette grani. Questo tipo di sintesi rappresenta un metodo molto efficace per costruire trame sonore estremamente ricche e complesse mediante la generazione, sovrapposizione e spazializzazione di una grande quantità di suoni di durata brevissima.

Un **grano** (come abbiamo già accennato nel paragrafo 5.4 del secondo volume) è un evento sonoro molto breve, in genere di durata compresa fra 1 e 100 millisecondi, alla cui forma d'onda (tipicamente prodotta da un oscillatore)

[1] *("Beneath the level of the note lies the realm of microsound, of sound particles. Microsonic particles remained invisible for centuries. Recent technological advances let us probe and explore the beauties of this formerly unseen world. Microsonic techniques dissolve the rigid bricks of music architecture - the notes - into a more fluid and supple medium. Sounds may coalesce, evaporate, or mutate into other sounds." (…) "When the particles line up in rapid succession, they induce the illusion of tone continuity that we call pitch. As the particles meander, they flow into streams and rivulets. Dense agglomerations of particles form swirling sound clouds whose shapes evolve over time.") (Roads, 2001, p.VII).*

[2] Gabor, 1947.

[3] Xenakis, 1960.

viene imposto un inviluppo per evitare fenomeni di variazione istantanea dell'ampiezza (risultante in possibili click) e per modellarne lo spettro (fig.12.1).

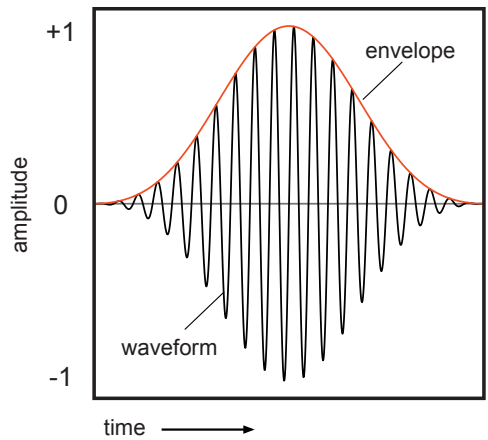


fig. 12.1: un grano con forma d'onda sinusoidale e inviluppo

I grani possono essere intervallati da pause ed essere percepiti come eventi sonori singoli, oppure possono essere sovrapposti in tale numero da essere percepiti come un flusso di suono ininterrotto, così come può essere un suono di pioggia con poche gocce fino a un suono di pioggia molto forte. Non a caso, in uno dei pezzi più significativi in sintesi granulare, "*Riverrun*", Barry Truax adottò proprio la metafora dell'acqua per la costruzione del pezzo pur utilizzando suoni sintetici. Da ciò possiamo capire che, in particolare con questo tipo di sintesi, la tradizionale divisione fra microstruttura e macrostruttura in musica si scioglie, per così dire in un continuum, come riferisce Jean Claude Risset a proposito della granulazione.[4]

4 "Colmando i divari tra le sfere tradizionalmente disconnesse come materiale e struttura, o vocabolario e grammatica, il software crea un *continuum* tra microstruttura e macrostruttura. Non è più necessario mantenere le tradizionali distinzioni tra un'area esclusiva della produzione del suono e un'altra dedicata alla manipolazione strutturale su un più ampio livello temporale. La scelta della granulazione, ovvero della frammentazione degli elementi sonori, è un modo per evitare incidenti su un *continuum* scivoloso: permette di ordinare gli elementi all'interno di una scala mentre consente di cogliere i singoli elementi. La preoccupazione formale si estende fino alla microstruttura, alloggiandosi nella grana del suono." *(By bridging gaps between traditionally disconnected spheres like material and structure, or vocabulary and grammar, software creates a continuum between microstructure and macrostructure. It is no longer necessary to maintain traditional distinctions between an area exclusive to sound production and another devoted to structural manipulation on a larger temporal level. The choice of granulation, or of the fragmenting of sound elements, is a way of avoiding mishaps on a slippery continuum: it permits the sorting of elements within a scale while it allows individual elements to be grasped. The formal concern extends right into the microstructure, lodging itself within the sound grain).* (Risset, 2005)

ESEMPIO SONORO 12.1
– Estratto da *Riverrun* di Barry Truax (da 0:00 a 5:00)[5]
– Estratto da *Riverrun* di Barry Truax (da 15:30 a 19:44)

Alcuni autori come Truax, Dodge e Jerse tendono a delimitare la durata massima dei grani nella sintesi granulare a 50 millisecondi in quanto al di sopra di questa durata la percezione di fusione fra i grani e la particolare interdipendenza fra tempo e frequenza/spettro tendono a perdersi.
Chiariamo meglio questo concetto: se ascoltiamo il suono di un oscillatore sinusoidale alla frequenza di 500 Hz per una durata di tempo relativamente breve, diciamo 100 millisecondi o più, riusciamo comunque a percepire la frequenza, e soprattutto percepiamo un suono il cui spettro contiene energia principalmente alla frequenza di 500 Hz. Se diminuiamo gradualmente la durata del suono la percezione dell'altezza diventa sempre meno chiara e la distribuzione dell'energia spettrale si allarga attorno alla frequenza centrale di 500 Hz. In altre parole la dispersione dell'energia spettrale attorno alla frequenza del suono è inversamente proporzionale alla durata del suono stesso (interdipendenza fra tempo e suono/spettro). Minore è la durata del grano, dunque, più larga sarà la banda di frequenze risultanti (fig. 12.2).

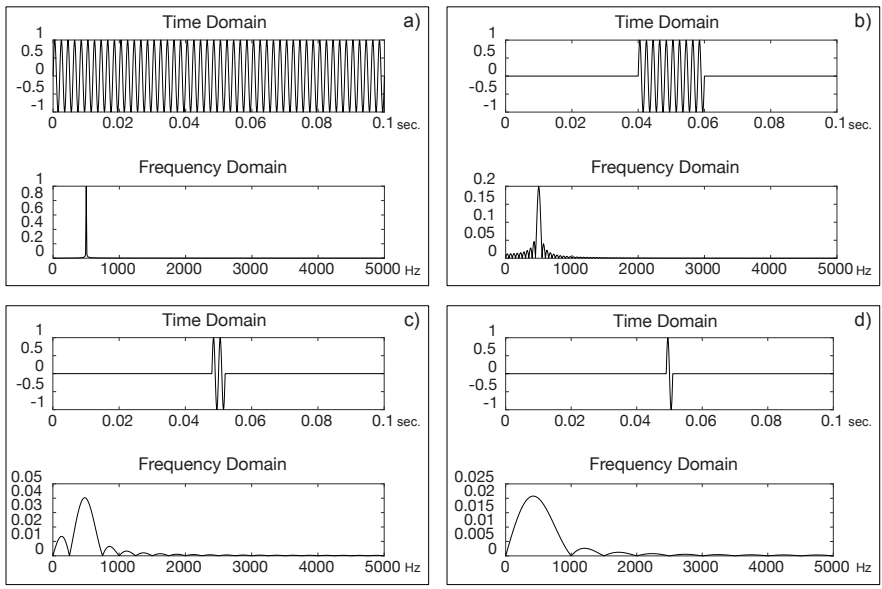


fig. 12.2: interdipendenza fra tempo e suono/spettro

[5] Versione a 24 bit stereo gentilmente fornita da Barry Truax. La versione stereo 16 bit è pubblicata su CSR-CD 8701 *Digital Soundscapes*. Cambridge Street Records. La versione originale è quadrifonica.

Il caso limite è costituito dall'impulso discreto unitario (un singolo campione di valore diverso da 0 seguito da campioni con valore 0) che, come sappiamo dal capitolo 3, ha uno spettro contenente energia a tutte le frequenze, da 0 Hz alla frequenza di Nyquist.
I grani di suono di durata molto breve, al di sotto dei 10 millisecondi, vengono quindi percepiti come impulsi le cui caratteristiche spettrali dipendono dalla durata del grano, dalla forma d'onda del suono, dalla forma dell'inviluppo e dalla frequenza del suono. Inoltre, a seconda della frequenza del suono stesso, a parità di durata, si avrà una maggiore (con suoni acuti) o minore (con suoni gravi) percezione della frequenza del suono.
Un'altra caratteristica è la modulazione d'ampiezza che può diventare percettivamente rilevante se i grani durano meno di 50 millisecondi e si susseguono regolarmente.

Come abbiamo visto nel capitolo 11 dedicato alla modulazione, se un segnale (detto modulante) modula l'ampiezza di un altro segnale (detto portante), vengono create due bande laterali le cui componenti hanno frequenze pari alla somma e alla differenza tra le componenti della portante e quelle della modulante. Nel nostro caso la portante è l'evento sonoro che viene inviluppato, e la modulante è il flusso degli inviluppi. Ad esempio, se abbiamo un flusso regolare di grani con inviluppo di forma sinusoidale unipolare (**finestra di Hann** o **hanning window**, vedi sotto) di 20 ms (50 grani al secondo, cioè 50 Hz) che modula un suono sinusoidale di 120 Hz, otterremo, oltre al suono di 120 Hz stesso, anche una componente laterale di 170 Hz (120 Hz + 50 Hz) e una di 70 Hz (120 Hz – 50 Hz): verrà quindi generato un suono inarmonico. Naturalmente più il periodo di generazione dei grani è lungo, più le componenti laterali si stringono vicino alla frequenza della portante fino al punto (sotto i 20 Hz, con grani di durata maggiore di 50 ms) da non essere più percepibili come frequenze separate. Se l'inviluppo o il suono dell'oscillatore non sono puramente sinusoidali come nel caso precedente, si avranno coppie laterali multiple derivate dalla somma e la differenza tra tutte le componenti in gioco. Torneremo sull'argomento nel prossimo paragrafo e vedremo come questo effetto di modulazione di ampiezza cambia sostanzialmente quando, tramite un meccanismo di *hard sync*, la fase del suono portante viene riportata a zero ogni volta che si produce un nuovo inviluppo.
Questa tecnica utilizza principalmente suoni di sintesi (additiva, tabellare, FM, etc.) come contenuto spettrale dei vari grani. Ogni grano ha un proprio inviluppo, una sua durata, una sua distanza nel tempo dal grano successivo, una sua forma d'onda e contenuto timbrico e una sua localizzazione nello spazio. Ognuno di questi parametri può essere fisso oppure cambiare i propri valori nel tempo. La gestione dei vari parametri di così tanti eventi richiede un controllo globale; sarebbe impossibile, infatti, definire i valori di ogni parametro per ogni singolo grano, quando i grani da generare e controllare per ogni secondo possono essere nell'ordine delle migliaia. È necessario perciò avere nell'algoritmo un procedimento automatico per la generazione dei grani e un controllo globale dei valori di tutti i parametri. Mediante questo controllo ad alto livello si possono stabilire traiettorie e anche mete da raggiungere, consentendo quindi all'utente di orientare il flusso del tempo verso una determinata direzione o punto focale, e in sostanza dando forma coerente a migliaia di piccole particelle di suono.

INVILUPPO DI UN GRANO

Di solito gli **inviluppi dei grani** sono di tipo simmetrico. La loro forma può essere basata sulla finestra di Hann (realizzabile tramite un ciclo di coseno unipolare rovesciato), oppure può essere triangolare, trapezoidale, gaussiana, quasi-gaussiana etc. (vedi sotto) ma si possono usare anche inviluppi non simmetrici a due fasi (ad esempio con attacco immediato e decadimento esponenziale, o viceversa, denominati da Roads rispettivamente ***expodec*** e ***rexpodec***). Tali forme sono dette anche funzioni finestra, e il grano di suono viene realizzato moltiplicando l'inviluppo per la forma d'onda desiderata.
Questa moltiplicazione crea la modulazione d'ampiezza di cui abbiamo parlato sopra, e dato che ogni tipo di inviluppo ha un diverso contenuto spettrale, la scelta dell'inviluppo è determinante per definire lo spettro del grano risultante.
In figura 12.3 vediamo le finestre di:

a) Hann
b) Gaussiana
c) Quasi-Gaussiana (inviluppo gaussiano con un sustain nella parte centrale)
d) Triangolare
e) Trapezoidale
f) Expodec - attacco immediato, meno di 10 ms, e decay esponenziale
g) Rexpodec - attacco esponenziale e decay immediato, meno di 10 ms
h) Impulso limitato in banda (**sinc function**, cioè sin(*x*)/*x*) - crea un effetto di forte modulazione
i) Blackman
j) Welch - realizzato tramite una singola sezione parabolica

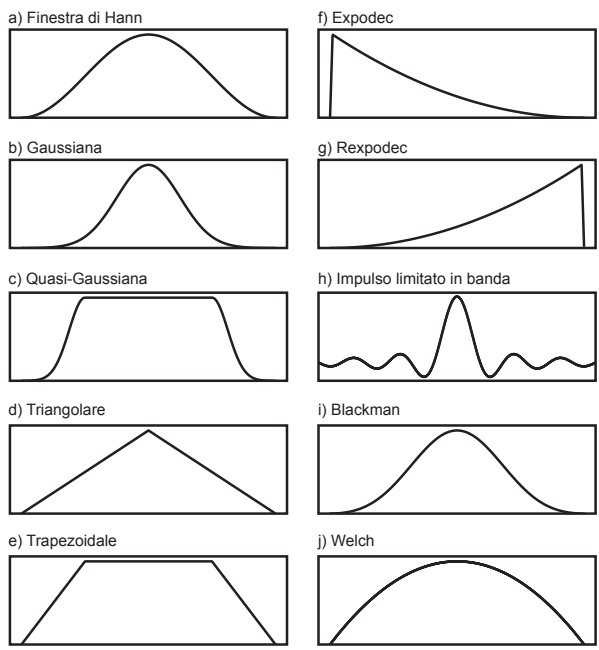


fig. 12.3: alcuni esempi di funzioni finestra (inviluppi)

ESEMPIO SONORO 12.2 • Esempi con inviluppi diversi
- Finestra di Hann
- Gaussiano
- Quasi-Gaussiano
- Triangolare
- Trapezoidale
- Expodec
- Rexpodec
- Impulso limitato in banda (sinc function)
- Blackman
- Welch

FORME D'ONDA PER LA SINTESI GRANULARE

La forma d'onda del suono che vogliamo granulare può variare nel tempo, oppure può essere fissa o può variare da grano a grano. Può essere periodica o aperiodica, si può utilizzare ad esempio anche un generatore di rumore. Inoltre, nonostante siamo nel campo della sintesi granulare e non della granulazione di suoni campionati (che tratteremo nel paragrafo 12.5), possiamo considerare sintesi granulare anche quella in cui la tabella dell'oscillatore sia riempita con un frammento di un suono campionato. Tratteremo più in dettaglio queste differenze nel par. 12.5.

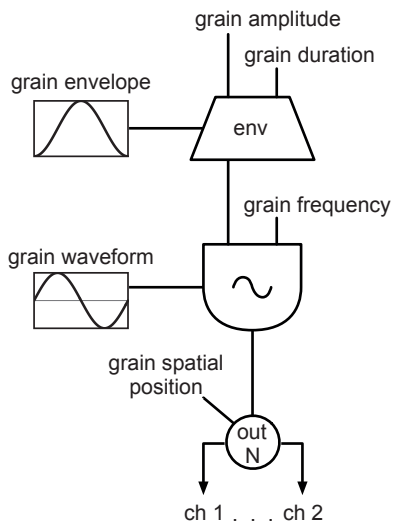


fig. 12.4: algoritmo di sintesi granulare semplice

LA NUVOLA E IL FLUSSO

Abbiamo detto che la sintesi granulare si costruisce mediante la generazione di suoni sintetici che vengono granulati. Riguardo la distribuzione dei grani generati nel tempo si possono avere due modalità generali:

sintesi granulare sincrona (a cui ci riferiremo con il concetto di "**Flusso**").
Nella sintesi granulare sincrona ogni grano segue il precedente, i grani sono distribuiti ad intervalli che possono essere più o meno regolari e il flusso lineare è determinato da un *grain rate* (numero di grani al secondo). Questo *rate* dei grani può essere fisso o variabile (ad esempio con un glissando o con un *range* limitato di variazione random). I grani che si succedono l'uno all'altro possono sovrapporsi (anche parzialmente tramite un *crossfade*) o prevedere una pausa fra un grano e l'altro.

sintesi granulare asincrona (a cui ci riferiremo con il concetto di "**Nuvola**").
I grani in questo caso sono distribuiti in modo irregolare, costituendosi come agglomerazioni di particelle la cui forma si evolve nel tempo, con un'evoluzione statistica, di cui una delle proprietà è la densità degli eventi. Dato che la generazione dei grani non è soggetta a un flusso lineare, è più corretto, in questo caso, usare il termine *grain density* (densità dei grani) anziché *grain rate*.
A una densità bassa corrisponderà la generazione di pochi e isolati eventi sparsi nel tempo e nello spazio, a una densità maggiore si otterranno nuvole di grani dense.
Nei prossimi paragrafi entreremo nel dettaglio di entrambe le modalità.

ESEMPIO SONORO 12.3 • Flusso e nuvola
a) Flusso: rate fisso
b) Flusso: rate variabile
c) Nuvola: densità bassa
d) Nuvola: densità alta

(...)

il capitolo prosegue con:

12.2 SINTESI GRANULARE SINCRONA E SINTESI PER FORMANTI
Le 4 tipologie di sintesi granulare sincrona
L'utilizzo di segnali aperiodici
Configurazioni probabilistiche del flusso dei grani
Parametri principali della sintesi granulare sincrona
Variazione casuale dei parametri
FOF (fonction d'onde formantique)

12.3 SINTESI GRANULARE ASINCRONA
Uso di porzioni di suoni campionati caricati in tabella

12.4 SINTESI PARTICELLARE (PARTICLE SYNTHESIS)
Glisson synthesis
Wavelet/grainlet synthesis
Trainlet synthesis
Pulsar synthesis

12.5 GRANULAZIONE E SEGMENTAZIONE DI SUONI CAMPIONATI
Puntatore e time stretching granulare
Variazioni della frequenza
Variazioni del grain rate
Variazioni del duty cycle
Variazioni random del grain rate e del duty cycle
Variazione combinata di più parametri
Flussi granulari spazializzati (voci)
Granulazione selettiva
Multi-source brassage e micromontage

ATTIVITÀ
• Esempi sonori

VERIFICHE
• Test a risposte brevi

SUSSIDI DIDATTICI
• Concetti di base - Glossario

12P
MICROSUONO

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume I e II, capp. 10 e 11 (teoria e pratica), interludio F e cap.12T

OBIETTIVI

Abilità

- Saper programmare e utilizzare algoritmi di sintesi granulare e particellare sincrona e asincrona
- Saper programmare e utilizzare algoritmi di sintesi per formanti e di granulazione di suoni campionati
- Saper applicare processi probabilistici e variazioni randomiche dei vari parametri della sintesi granulare, particellare, FOF e granulazione
- Saper programmare e utilizzare algoritmi di multi-source brassage

Competenze

- Saper realizzare un breve studio basato sull'uso della sintesi granulare e della granulazione dei suoni campionati e di microsuoni in generale

ATTIVITÀ

- Costruzione e modifiche di algoritmi

SUSSIDI DIDATTICI

- Lista di oggetti Max - Lista attributi e messaggi per oggetti Max specifici - Lista operatori gen

12.1 SINTESI GRANULARE

Come abbiamo visto nel capitolo di teoria gli elementi costitutivi di un grano sono l'inviluppo e la forma d'onda, o meglio l'evento sonoro che viene "racchiuso" nell'inviluppo. Vediamo come possiamo implementare in Max questi elementi.

INVILUPPO DI UN GRANO

Alcuni degli inviluppi che useremo in questo capitolo possono essere realizzati direttamente dall'oggetto `buffer~`: questo oggetto infatti può generare o modificare il proprio contenuto in base ai messaggi che riceve.
Con il messaggio *fill* (riempi) possiamo chiedere all'oggetto `buffer~` di riempire la propria memoria con determinati valori. Il messaggio [fill 1] fa sì che tutti i campioni del *buffer* assumano il valore 1, [fill -0.5] riempirà il *buffer* di campioni con valore -0.5 e così via.
Il messaggio *apply* invece impone una funzione finestra (*windowing function*) al contenuto del *buffer*. Ad esempio il messaggio [apply hanning] realizza una finestra di Hann, [apply triangle] una finestra triangolare. Ci sono anche altre funzioni come ad esempio *hamming*, *blackman* e *welch*.[1] Come si vede non tutte le finestre di cui abbiamo parlato nella teoria sono disponibili: affronteremo la questione tra poco.
Diamo un'occhiata alla figura 12.1: è importante notare che una funzione finestra va applicata al contenuto già presente nel *buffer*; un *buffer* "vuoto" (cioè contenente solo campioni con valore 0) non subirebbe alcuna modifica.
Con il messaggio *fill* riempiamo dunque il *buffer* con campioni di valore costante, e con il messaggio *apply* imponiamo una funzione finestra.

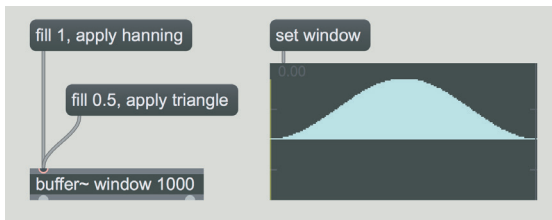


fig. 12.1: funzioni finestra con `buffer~`

I due messaggi della *patch* in figura (che vi invitiamo a ricostruire) servono a realizzare una finestra di Hann e una finestra triangolare (quest'ultima con ampiezza dimezzata).
Aggiungete alla *patch* di figura 12.1 dei *message box* con cui creare le finestre *hamming*, *blackman* e *welch*; non dimenticate di riempire il *buffer* con un valore costante tramite il messaggio *fill*! Osservate la differente forma delle varie finestre.

[1] Ci sono altre caratteristiche relative ai messaggi *fill* e *apply*, vi invitiamo ad approfondirle aprendo la *patch* di help dell'oggetto `buffer~` e il reference manual.

Passiamo ora alle funzioni che non è possibile realizzare con il messaggio *apply*; abbiamo già visto nel paragrafo IB.9 del primo volume come sia possibile memorizzare degli inviluppi arbitrari in un *buffer* (e se non vi ricordate come si fa, vi invitiamo a rileggere il paragrafo in questione).
Vediamo ad esempio come possiamo realizzare una finestra trapezoidale: per semplificarci il compito partiamo da una finestra triangolare che modificheremo successivamente. Ricreate la *patch* di figura 12.2.

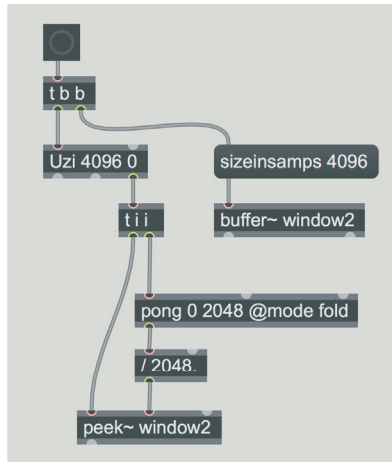


fig. 12.2: finestra triangolare

In questa figura vediamo un oggetto, **pong**, che è la versione Max dell'oggetto MSP `pong~` di cui abbiamo parlato al paragrafo 11.4P. Come l'omologo MSP, anche questo oggetto può lavorare in tre modalità: *clip*, *wrap* e *fold* (per la descrizione vi rimandiamo al già citato paragrafo 11.4P). La differenza con `pong~` è che in questo caso le modalità vanno impostate, in formato testuale, tramite l'attributo @mode, e non come argomento numerico.

L'oggetto `uzi` genera una serie di valori da 0 a 4095; questi valori vengono inviati all'oggetto [`pong` 0 2048 @mode fold] che fa "ripiegare all'indietro" tutti i valori maggiori di 2048, generando quindi una serie che va da 0 a 2048 e poi ritorna indietro. Dividendo i valori in uscita per 2048 otteniamo una finestra triangolare che va da 0 a 1.

Con due semplici operazioni possiamo trasformare la finestra triangolare in trapezoidale. Modificate la *patch* come da figura 12.3 (aggiungete gli oggetti `*` e `clip`).

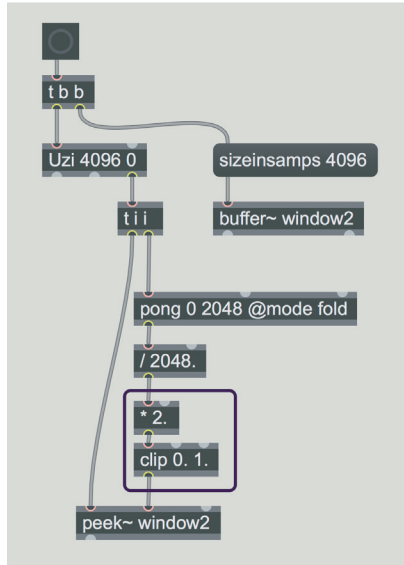


fig. 12.3: finestra trapezoidale

In questa *patch* l'altezza del triangolo viene raddoppiata, successivamente l'oggetto `clip` "taglia" la parte superiore del triangolo trasformandolo in un trapezio.

ATTIVITÀ

- Come sappiamo dalla teoria, la finestra di Hann è realizzata tramite un coseno unipolare rovesciato. Modificate la *patch* di figura 12.2 in modo che ricrei la finestra di Hann (senza naturalmente usare i messaggi *fill* e *apply*!)
- La base minore (superiore) del trapezio generato dalla *patch* di figura 12.3 è lunga la metà (cioè il 50%) della base maggiore. Se moltiplichiamo l'altezza del triangolo per 3 invece che per 2, la base minore del trapezio ricavato sarebbe 2/3 (cioè il 66.666%) della maggiore, moltiplicando per 4 avremmo una base minore pari a 3/4 (75%) della maggiore. Aggiungete alla *patch* un piccolo algoritmo che permetta di impostare la base superiore come percentuale della base inferiore: in altre parole il valore 50 deve generare il moltiplicatore 2, il valore 66.666 deve generare il moltiplicatore 3 e il valore 75 deve generare il moltiplicatore 4 (e naturalmente gli altri valori devono generare gli opportuni moltiplicatori).

E ora una buona notizia; la libreria *Virtual Sound Macros* contiene un oggetto che ci permette di creare tutte le funzioni finestra che ci serviranno per la sintesi granulare: `vs.winfunc`. Aprite il file **12_01_winfunc.maxpat** (fig. 12.4).

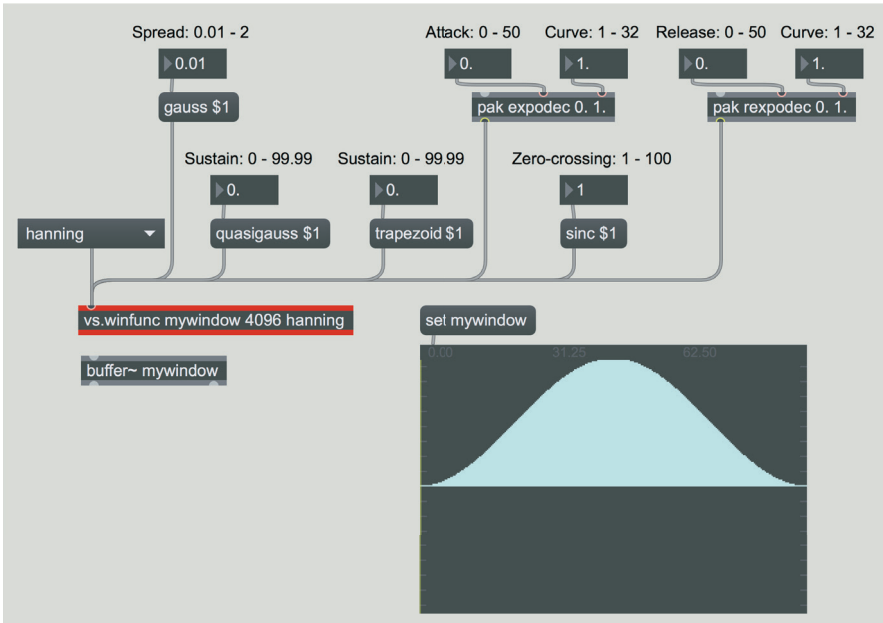


fig. 12.4: file **12_01_winfunc.maxpat**

Questo oggetto può avere i seguenti argomenti (opzionali):
- nome del *buffer*
- dimensione del *buffer* in campioni
- nome della funzione finestra seguita da eventuali parametri.

È anche possibile inviare il nome della funzione e i relativi parametri all'ingresso dell'oggetto. Ecco le funzioni disponibili:
– *hanning* (finestra di Hann), nessun parametro
– *gauss*, parametro *spread* (ovvero la larghezza della campana) da 0.01 a 2.
– *quasigauss*, parametro *sustain* da 0% a 99.99%
– *triangle*, nessun parametro
– *trapezoid*, parametro *sustain* da 0% a 99.99%
– *expodec*, parametri *attack*, da 0% a 50%, e curve (fattore esponenziale) da 1 a 32
– *rexpodec*, parametri *release*, da 0% a 50%, e *curve* da 1 a 32
– *sinc*, parametro *zero-crossing* (numero di volte in cui viene attraversato lo zero) da 1 a 100
– *blackman*, nessun parametro
– *welch*, nessun parametro

Provate tutte le funzioni contenute nell'**umenu** a sinistra. Poi osservate come cambiano gli inviluppi delle funzioni che accettano i parametri modificando i valori dei *number box* che trovate nella *patch*.

ATTIVITÀ

Aprite con un doppio click l'oggetto **vs.winfunc** (che come tutti gli oggetti della libreria *Virtual Sound Macros* è un'*abstraction*) e confrontate l'algoritmo che genera la funzione trapezoidale con l'analogo algoritmo che abbiamo visto in fig. 12.3. Poi analizzate il funzionamento degli algoritmi che generano le funzioni *expodec* e *rexpodec*.

FORME D'ONDA PER LA SINTESI GRANULARE

Praticamente quasi tutti i generatori di suono che abbiamo incontrato nei capitoli precedenti sono utilizzabili per la sintesi granulare: dal semplice oscillatore sinusoidale agli oscillatori limitati in banda, dalla sintesi vettoriale ai generatori di rumore, dalle tabelle create in sintesi additiva o riempite con frammenti di suoni campionati alle risonanze prodotte dagli impulsi filtrati, e naturalmente tutti i suoni prodotti con le diverse tecniche di sintesi non lineare. Vedremo nel corso del capitolo come e quando usare le varie sorgenti.

(...)

il capitolo prosegue con:

12.2 SINTESI GRANULARE SINCRONA E SINTESI PER FORMANTI
Le 4 tipologie di sintesi granulare sincrona
L'utilizzo di segnali aperiodici
Configurazioni probabilistiche del flusso dei grani
Modulazione dei parametri tramite inviluppi
Altri parametri della sintesi granulare sincrona
Duty cycle
Durata Del Grano
Glissando
Variazione casuale dei parametri
Range di collocazione spaziale dei grani: pan e riverbero
Utilizzo della polifonia: le voci
Sovrapposizione di grani in un unico flusso
Sintesi per formanti

12.3 SINTESI GRANULARE ASINCRONA
Uso di porzioni di suoni campionati caricati in tabella

12.4 SINTESI PARTICELLARE (PARTICLE SYNTHESIS)
Glisson synthesis
Wavelet/grainlet synthesis
Trainlet synthesis
Pulsar synthesis

12.5 GRANULAZIONE E SEGMENTAZIONE DI SUONI CAMPIONATI
Granulazione sincrona
Brassage, multi-source brassage ed elaborazione del suono in tempo differito
Granulazione di un segnale in tempo reale
Buffer circolare
Doppio buffer

ATTIVITÀ
• Analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi, correzione di algoritmi

VERIFICHE
• Compiti unitari di reverse engineering

SUSSIDI DIDATTICI
• Lista oggetti Max - Lista messaggi e attibuti per oggetti Max specifici - Lista operatori Gen

13T
ANALISI, RISINTESI E CONVOLUZIONE

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume I e II, capp. 10, 11 e 12

OBIETTIVI

Conoscenze

- Conoscere le basi della tecnica del vocoder
- Conoscere i fondamenti della teoria della trasformata di Fourier
- Conoscere le basi dell'elaborazione del suono mediante manipolazione dei *BIN*
- Conoscere le basi della tecnica dell'analisi e risintesi, in particolare del phase vocoder
- Conoscere le basi della tecnica della convoluzione
- Conoscere le basi della sintesi incrociata mediante convoluzione
- Conoscere le basi della tecnica dei riverberi a convoluzione
- Conoscere i concetti base nel campo della convoluzione con microsuoni

Abilità

- Saper individuare all'ascolto diversi effetti realizzati con phase vocoder
- Saper individuare all'ascolto l'effetto del vocoder sui suoni di voce
- Saper distinguere all'ascolto la differenza fra un effetto di vocoder e uno di harmonizer
- Saper individuare all'ascolto diversi effetti realizzati mediante cross-synthesis

CONTENUTI

- Teoria di base del vocoder
- Teoria di base della trasformata di Fourier
- Teoria di base dell'analisi e risintesi mediante phase vocoder
- Teoria di base della convoluzione
- Teoria di base della sintesi incrociata mediante convoluzione
- Teoria di base della convoluzione con microsuoni
- Teoria di base dei riverberi a convoluzione

ATTIVITÀ

- Esempi sonori

SUSSIDI DIDATTICI

- Concetti di base - Glossario

LE TECNICHE DI SINTESI BASATE SU DATI DI ANALISI

Uno dei metodi più interessanti per l'elaborazione di un suono è quello dell'**analisi e risintesi** (o **sintesi basata su analisi**, *synthesis by analysis*). Si tratta di una tecnica per la sintesi basata su dati di analisi precedentemente raccolti.
Vediamo a grandi linee come si svolge il processo di base che può essere in tempo differito o in tempo reale:

1) Un *sound file*, o un suono generato in tempo reale, viene analizzato tramite un algoritmo per l'analisi. I risultati vengono posti in un file di analisi o inviati in tempo reale all'algoritmo di risintesi.
2) I dati del file di analisi o generati in tempo reale possono essere modificati in modo che alcune caratteristiche del suono non corrispondano più a quelle originali.
3) I dati trasformati vengono ora utilizzati come base per risintetizzare il suono, cioè in questa fase ritorniamo dai dati di analisi a un suono. Il nuovo segnale conterrà anche le eventuali modifiche che abbiamo apportato nei dati di analisi.

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 e interazioni fra più file molto varie e interessanti.
Questo tipo di elaborazioni dinamiche consente per esempio di allungare o abbreviare la durata di un suono e contemporaneamente modificarne la frequenza in modo autonomo, di realizzare *stretching spettrale*, modificare singole componenti e i loro inviluppi, creare effetti di *spectral morphing* fra un suono e un altro etc.
Esistono numerose tecniche per l'analisi del suono, e non ne esiste una ideale[1]. Alcune di queste sono basate sull'analisi di Fourier, una tecnica molto importante che approfondiremo nel secondo paragrafo.
La scelta della tecnica da utilizzare dipende dalle caratteristiche spettrali del tipo di suono che deve essere analizzato e dal suono che si vuole ottenere. Già dagli anni '80 molti compositori si sono dedicati a queste tecniche con risultati significativi. In particolare segnaliamo quattro pezzi basati su sistemi di risintesi della voce umana e altri suoni: *Vox 5* di Trevor Wishart (1986) e *Study in White* di Joji Yuasa (1987), realizzati con la tecnica del *phase vocoder*; *Idle Chatter* di Paul Lansky (1990) in cui la tecnica di analisi e risintesi esplorata fu la *Linear Predictive Coding* (*LPC*);[2] *Mortuos Plango Vivos Voco* di Jonathan Harvey (1980)

[1] Ad esempio: analisi e risintesi con *phase vocoder* (basata su FFT); analisi e risintesi con filtro a eterodina; analisi e risintesi sottrattiva e *LPC* (*Linear Predictive Coding*, Predizione Lineare); analisi e risintesi con tecnica mista (deterministica e stocastica, vedi Serra, X. 1989); SMS (Sintesi per Modellizzazione Spettrale) e ATS (Analisi-Trasformazione-Sintesi), *wavelet synthesis*, etc.

[2] La codifica per predizione lineare è una tecnica elaborata per analizzare il suono della voce. Questa tecnica consente, a partire da tale analisi, di produrre i coefficienti di un filtro *all-pole*. Mediante tale filtro si può risintetizzare il suono originario a partire da suoni complessi armonici e inarmonici. Ovviamente i dati del filtro possono essere elaborati per ottenere poi, in uscita dal filtro stesso, un suono diverso da quello originario. I primi esperimenti furono proposti già nel 1966 in Giappone da Shuzo Saito e Fumitada Itakura. Per approfondimenti vedi Makhoul, 1975.

basata sull'analisi e risintesi mediante FFT di suoni di campana e su elaborazioni di registrazioni della voce di un bambino mediante il sistema CHANT[3]. Un pezzo più recente che però utilizza un diverso concetto di analisi e risintesi è *Speakings* di Jonathan Harvey (2007-2008)[4].

ESEMPIO SONORO 13.1

a) Estratto dal pezzo *Idle Chatter* di Paul Lansky[5] (da 0:00 a 3:29)
b) Estratto dal pezzo *Vox 5* di Trevor Wishart[6]
c) Estratto da *Inferno*[7] di Edison Studio (da 15:46 a 16:15 e da 16:51 a 17:40)
d) Estratto dal pezzo *Mortuos plango, vivos voco* di Jonathan Harvey (da 1:42 a 4:42)[8]

[3] Il sistema CHANT per la sintesi vocale è stato sviluppato da Gerald Bennett e Xavier Rodet.

[4] *Speakings* è un pezzo per *live electronics* e orchestra scritto da Jonathan Harvey "con il proposito artistico di far parlare un'orchestra mediante processi di computer music" (*with the artistic aim of making an orchestra speak through computer music processes*). (Nouno, G., et al., 2009). Il *tool* principale per l'orchestrazione assistita utilizzato fu *Orchidée*. L'evoluzione attuale dei sistemi *Orchid** dell'Ircam di Parigi, nati da un'idea originale di Ian Maresz e dell'Ircam Orchestration Workgroup, si chiama *Orchidea*, un *framework* per l'orchestrazione assistita statica e dinamica sviluppato da Carmine Emanuele Cella all'interno di un progetto congiunto dell'Ircam (Music Representation Team), HEM e UC Berkeley (vedi Gillick, J. et al., 2019, Cella, C.E., 2020 e 2021).

[5] CD BCD 9050. Courtesy of Paul Lansky.

[6] Courtesy of Trevor Wishart.

[7] *Inferno* è il primo lungometraggio italiano (1911) di F.Bertolini, A.Padovan e G. De Liguoro. È un film muto sulla prima cantica de *La Divina Commedia* di Dante. Il collettivo di compositori *Edison Studio* (Mauro Cardi, Luigi Ceccarelli, Fabio Cifariello Ciardi e Alessandro Cipriani) ha composto una colonna sonora elettroacustica per questo film in cui si fondono e si scambiano di funzione voci, ambienti sonori e musica. Il film con la colonna sonora in surround è stato pubblicato nel 2011 nel Libro-DVD CR/10 Cineteca di Bologna, Collana *Il Cinema Ritrovato* (vedi Cipriani et al., 2004 e Gazzano, ed. 2014).

[8] CD Sargasso, SCD 28029. Versione stereo. Courtesy of Sargasso. Harvey ha realizzato anche versioni in quadrifonia e in ottofonia.

13.1 IL VOCODER (O CHANNEL VOCODER)

"**Vocoder**" è una parola composta; è formata infatti da due termini in lingua inglese: *voice* ed *encoder* (o *coder*) cioè "voce" e "codificatore". Questo tipo di elaborazione, noto ai più come effetto di "orchestra parlante" o "voce robotica" fin dagli anni '70, consente di modificare un suono ad ampio spettro mediante l'inviluppo spettrale tempo-variante di un segnale di voce parlata. Il primo prototipo di *vocoder* fu inventato da Homer Dudley nel 1928, per fini non musicali, bensì come parte di una ricerca sulla codifica e la crittografia della voce per la trasmissione telefonica[9]. Solo nel 1948 Werner Meyer-Eppler (più tardi conosciuto come uno dei fondatori dello studio WDR di Colonia) pubblicò una tesi in cui applicò le stesse tecniche a un uso musicale. Il primo strumento musicale contenente un *vocoder* fu il *Siemens Synthesizer* alla fine degli anni '50, seguito poi dai primi *vocoder* realizzati da Robert Moog alla fine degli anni '60.
Il *vocoder* oggetto di questo paragrafo, che viene realizzato nel dominio del tempo, viene anche chiamato *channel vocoder* e va distinto dal *phase vocoder* che viene realizzato nel dominio della frequenza e di cui ci occuperemo nei paragrafi successivi.

Possiamo suddividere genericamente la struttura del *vocoder* in due parti, l'*encoder* e il *decoder*. Prima di entrare nell'*encoder*, il segnale della voce viene suddiviso in vocali e consonanti. Queste ultime vengono inviate direttamente all'uscita, mentre le vocali costituiranno il segnale in ingresso nell'***encoder*** il quale contiene un banco di filtri passa-banda in parallelo. Questa separazione viene attuata per una migliore comprensibilità della voce. Le consonanti, infatti, se vengono filtrate, diventano meno comprensibili e si può generare un "ronzio" (che in genere è indesiderabile) in corrispondenza del loro passaggio attraverso i filtri. Per effettuare questa separazione possiamo utilizzare uno ***zero crossing detector***, il quale valuta quante volte la forma d'onda attraversa lo zero. I suoni consonantici tendono ad avere attraversamenti dello zero più frequenti, in quanto contengono una quantità di componenti acute maggiori rispetto alle vocali. Nello *zero crossing detector* impostiamo un valore di soglia relativa alla quantità di attraversamenti del valore zero[10] che l'energia spettrale del segnale deve superare per classificare un determinato suono come consonantico.

[9] Nel 1939, nella *New York World Fair*, fu mostrato il *voder*, che era uno strumento per sintetizzare la voce, non venivano quindi usati microfoni come nel *vocoder*, ma era presente solo la parte del *decoder*. Venivano riprodotti diversi suoni e intonazioni della voce attraverso treni di impulsi, rumori filtrati e una pura manipolazione meccanica per far "parlare" la macchina controllandola con una serie di tasti e controlli fisici. Il *vocoder* invece è uno strumento per elaborare la voce che richiede quindi una voce reale (mediante microfono, un suono registrato o campionato) e una serie di controlli.

[10] Questa valutazione viene effettuata dal *detector* facendo il confronto fra il campione attuale e quello precedente per controllare se il segnale è passato dal campo positivo a quello negativo o viceversa.

Questo valore di soglia va determinato mediante l'ascolto (per prova ed errore) sulla base del suono specifico della voce. Sulla base di questa impostazione, se il valore in uscita supera la soglia il suono viene classificato come consonante, in caso contrario sarà considerato dal *detector* come una vocale. Quando questa soglia viene superata il segnale può essere inviato direttamente all'uscita, oppure a un generatore di rumore, mentre se la soglia non viene superata il segnale viene mandato all'*encoder*. Il suono della parte della voce contenente le vocali (cioè quello sotto la soglia) viene quindi ulteriormente "suddiviso" nell'*encoder* in un certo numero di bande frequenziali.

Il numero e la diversa larghezza delle bande in relazione alle zone frequenziali influiscono sul buon risultato (vedremo nella parte pratica come operare). Per ogni banda di frequenza viene misurato il livello d'ampiezza nel tempo mediante una serie di *envelope follower* posti in uscita da ogni filtro. In questo modo otteniamo una rappresentazione dell'energia spettrale nel tempo del suono della voce. Gli inviluppi in uscita dagli *envelope follower* vengono utilizzati nel ***decoder*** come segnali di controllo per un secondo banco di filtri passa-banda (contenente a sua volta lo stesso numero e tipologia di filtri dell'*encoder*).
In questo secondo banco di filtri (la cui ampiezza è controllata dai suddetti inviluppi) viene mandato in input un secondo suono dallo spettro ricco (può essere un rumore, un suono di orchestra, etc.). Tale suono viene dunque filtrato sulla base dell'evoluzione dell'energia spettrale della voce nel tempo.

L'effetto particolare che rende la voce "sintetica", per così dire, è dato anche dal fatto che l'informazione delle frequenze fondamentali della voce non viene calcolata. Ad esempio, se il secondo segnale è quello di un accordo suonato da un'orchestra, le frequenze fondamentali della voce, una volta elaborata, saranno quelle di quell'accordo e non quelle della voce originaria. Ovviamente però è possibile mixare il suono in uscita dal *vocoder* con il suono della voce *dry*, a seconda del nostro scopo. Possiamo utilizzare il *vocoder* anche con altri suoni oltre la voce, ad esempio una batteria (esempio sonoro 13.2K) o altri tipi di suoni monodici. Si può anche utilizzare, per il *decoder*, uno strumento virtuale con codifica MIDI, ad esempio un pianoforte campionato. L'*encoder* questa volta conterrà un numero di filtri corrispondente ai tasti del pianoforte, intonati a distanza di semitono temperato l'uno dall'altro. I valori di ampiezza ottenuti in uscita dall'*encoder* verranno convertiti in valori di *velocity*. Questi valori verranno associati alle note MIDI corrispondenti. Suonando le varie note si potrà così "armonizzare", mediante il pianoforte virtuale, il suono in ingresso (esempio sonoro 13.2L con variazione dei parametri).
È anche possibile utilizzare, come secondo segnale, non un suono campionato, ma un suono complesso realizzato in sintesi additiva (addizionando sinusoidi o suoni già complessi). Con questa tecnica si può controllare il suono in uscita in modo più vario e plasmabile dall'utente (esempio sonoro 13.2M in cui sentiamo prima il flauto *dry* e poi con *vocoder* e variazioni del *decay* e del *gain*).

In figura 13.1 vediamo uno schema che rappresenta il flusso dei segnali in un *vocoder*.

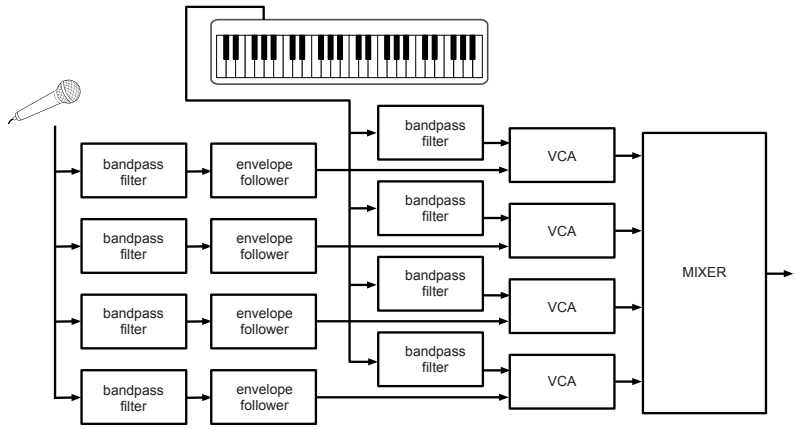


fig. 13.1 schema di un *vocoder*

Una tecnica mista interessante[11], inoltre, è quella di mixare il *vocoder* con la tecnica dell'*harmonizer* (abbiamo trattato quest'ultimo nel par. 8.3T e 6.9P). Ricordiamo che l'*harmonizer* consente di creare *pitch* doppi, tripli, etc. che possono essere intonati su intervalli determinati dall'utente in relazione al suono originale (e quindi alla fondamentale) della voce. La differenza con il *vocoder* è che questi intervalli nell'*harmonizer* non sono in relazione a un secondo segnale, bensì sono dati da fattori di trasposizione del suono e quindi le frequenze delle copie seguono continuamente quella della fondamentale, alla distanza determinata dall'intervallo fra l'originale e ognuna delle copie. Facciamo un esempio: la voce canta glissando da DO a FA e poi a SOL e viene inviata contemporaneamente sia all'*harmonizer*, sia al *vocoder*. Se nell'*harmonizer* impostiamo una trasposizione di quarta, quinta e nona, e contemporaneamente mandiamo nel *decoder* del *vocoder* un suono contenente un accordo di settima DO-MI-SOL-SIb, avremo che le stesse parole cantate produrranno contemporaneamente accordi fissi dati dal *vocoder* e accordi in continuo mutamento creati dall'*harmonizer* in relazione alle fondamentali cangianti del suono originale. Questo è efficace anche come metodo per rendere una voce parlata (che quindi ha continui cambiamenti istantanei della fondamentale) come fonte di armonie non tradizionali ma percepite all'interno di un contesto armonico determinato dal *vocoder*.

[11] Abbiamo descritto una tecnica analoga nel par. 8.4 in cui veniva combinato l'*harmonizer* con filtri risonanti. Queste tecniche miste di elaborazione della voce sono state utilizzate da Alessandro Cipriani nel pezzo *Il Pensiero Magmatico* (Cipriani-Taglietti, 1996) presso l'Edison Studio utilizzando il sistema digitale per la sintesi in tempo reale MARS (Musical Audio Research Station). La MARS fu creata nel 1991 da un team di numerosi ricercatori, fra cui Giuseppe Di Giugno, Emmanuel Favreau, Eugenio Guarino, Andrea Paladin e Sylviane Sapir, presso l'IRIS di Paliano (FR) (vedi Cavaliere et al. 1992, Palmieri et al. 1992 e Andrenacci et al., 1997).

ESEMPIO SONORO 13.2

a) Suono di voce parlata dry
b) Suono di orchestra granulata
c) *Vocoder* che utilizza a) e b)
d) Suono c) mixato con il suono a)
e) Voce cantata
f) Voce cantata e) con *vocoder*
g) Voce cantata e) con *harmonizer*
h) Voce cantata con *harmonizer* e *vocoder*
i) Voce parlata con *harmonizer* e *vocoder*
j) Estratto dalla parte elettronica de *Il Pensiero Magmatico* di A.Cipriani e S.Taglietti; tecnica mista *vocoder+harmonizer*[12]
k) Batteria+*vocoder* con suono b: variazioni del *decay* e del fattore Q
l) MIDI *vocoder*: *electronics*+piano MIDI; variazioni del *gain* e del fattore Q
m) Flauto *dry* e poi con *vocoder* con variazioni del *decay* e del *gain*.

(...)

[12] Dalla traccia 11 *Nel magma incandescente* (da 2:30 a 5:14). La versione completa per pianoforte, percussioni, coro misto ed elettronica è pubblicata su PAN CD 3059. Courtesy of EDI-PAN.

il capitolo prosegue con:

ATTIVITÀ
• Esempi sonori

VERIFICHE
• Test a risposte brevi

SUSSIDI DIDATTICI
• Concetti di base - Glossario

Questa pagina è intenzionalmente lasciata in bianco

13P
ANALISI, RISINTESI E CONVOLUZIONE

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume I e II, interludio F, capp. 10, 11, 12 (teoria e pratica) e cap. 13T

OBIETTIVI

Abilità

- Saper programmare e utilizzare algoritmi basati sulla tecnica del vocoder
- Saper programmare e utilizzare algoritmi di elaborazione del suono basati sulla trasformata di Fourier mediante manipolazione dei bin
- Saper programmare e utilizzare algoritmi di elaborazione del suono basati sulla tecnica dell'analisi e risintesi, in particolare del phase vocoder
- Saper programmare e utilizzare algoritmi di elaborazione del suono mediante convoluzione, effettuando sintesi incrociata, riverberi a convoluzione e convoluzione con microsuoni

Competenze

- Saper realizzare un breve studio basato sull'uso della convoluzione e del phase vocoder

ATTIVITÀ

- Costruzione e modifiche di algoritmi

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista messaggi, argomenti e attributi per oggetti Max specifici - Lista operatori Gen

13.1 IL VOCODER

Vediamo innanzitutto come si può realizzare un *encoder*: abbiamo bisogno per prima cosa di un banco di filtri, ovvero dell'oggetto **`fffb~`** (*fast fixed filter bank*) di cui abbiamo parlato nel paragrafo 3.7P del primo volume. Dal momento che normalmente questo oggetto presenta tante uscite quanti sono i filtri del banco, per praticità utilizzeremo la versione multicanale **`mcs.fffb~`** che riunisce tutti i filtri in un'unica uscita multicanale (vedi figura 13.1).

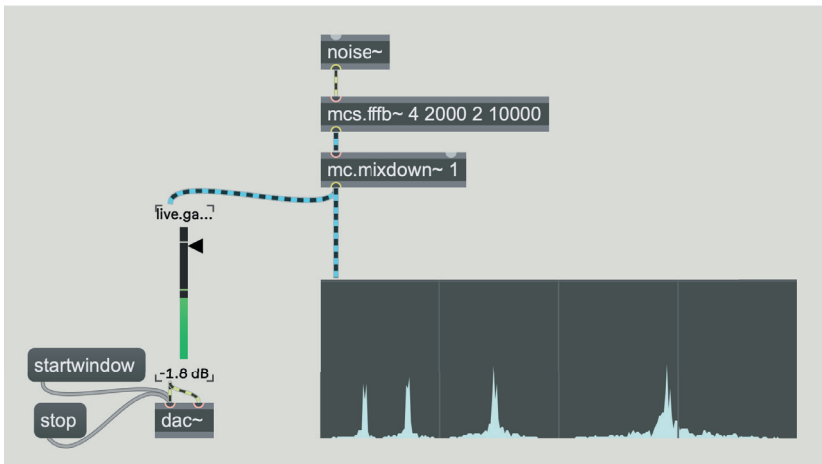


fig. 13.1: l'oggetto **`mcs.fffb~`**

Ricordiamo gli argomenti dell'oggetto: il primo indica il numero di filtri presenti nel banco, il secondo la frequenza del primo filtro, il terzo un fattore di moltiplicazione per i filtri successivi (nel caso in figura il fattore di moltiplicazione è 2, questo significa che ogni filtro ha il doppio della frequenza del precedente, in altre parole i filtri procedono per ottave) e il quarto argomento rappresenta un fattore Q che viene applicato a tutti i filtri del banco.
Come sappiamo dal paragrafo 13.1T, per ciascun filtro abbiamo bisogno di conoscere il livello di ampiezza in uscita, e possiamo farlo tramite l'oggetto **`avg~`**. Questo oggetto riceve un segnale e una serie di *bang*. Ad ogni *bang* ricevuto l'oggetto produce (sotto forma di numero Max) il valore medio (assoluto) del segnale in ingresso calcolato dal *bang* precedente (vedi figura 13.2).

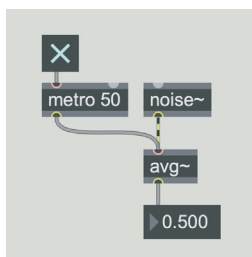


fig. 13.2: oggetto **`avg~`**

Nella *patch* **13_01_encoder.maxpat** (figura 13.3) utilizziamo questi oggetti per creare un *encoder*.

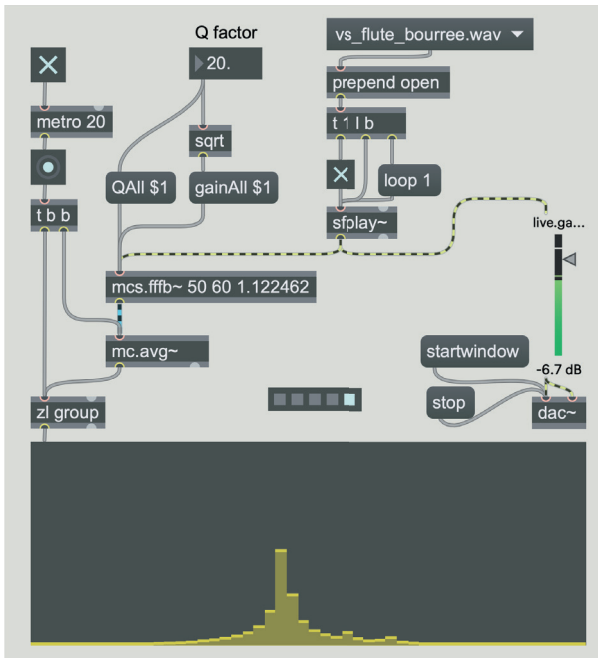


fig. 13.3: *patch* **13_01_encoder.maxpat**

Un segnale audio viene inviato all'oggetto `mcs.fffb~`: questo oggetto contiene un banco di 50 filtri, la frequenza del primo filtro è 60 Hz e il rapporto tra i filtri è 1.122462, cioè $\sqrt[6]{2}$, pari a un incremento di un tono tra un filtro e il successivo. Abbiamo in altre parole un banco di filtri a sesti d'ottava (ogni ottava è divisa cioè in 6 parti). Il cinquantesimo e ultimo filtro avrà una frequenza pari a $60 * (\sqrt[6]{2})^{49} = 17241$ Hz circa.
Notate che la potenza a cui eleviamo il rapporto di tono è 49 e non 50 perché la serie di moltiplicazioni avviene a partire dal secondo filtro; potremmo dire che il primo filtro ha una frequenza di $60 *(\sqrt[6]{2})^{0} = 60$ Hz.
Tramite il *number box* in alto regoliamo il fattore Q di tutti i filtri (messaggio "QAll") e il *gain* (messaggio "gainAll") che facciamo corrispondere alla radice quadrata del fattore Q, per compensare la perdita di energia che un Q alto comporta[1].
Il segnale multicanale generato da `mcs.fffb~` viene inviato all'oggetto `mc.avg~` il quale riceve un *bang* ogni 20 millisecondi da un `metro`: ad ogni *bang* `mc.avg~` genera i 50 valori corrispondenti ai 50 canali; questi valori vengono inviati a un oggetto [`zl` group] che li raggruppa in una lista e la passa all'oggetto `multislider` in basso. Quello che viene visualizzato dal `multislider` è in pratica il profilo spettrale del suono inviato al banco di filtri.

[1] Vedi il paragrafo 3.3P del primo volume.

Notate che quando il fattore Q è molto alto i filtri passa-banda di `mcs.fffb~` tendono a produrre delle risonanze che si smorzano lentamente, come sappiamo, e questo rallenta i movimenti del profilo spettrale che vediamo nel `multislider`. Possiamo quindi utilizzare il fattore Q per regolare la velocità di cambiamento del profilo spettrale: confrontate a questo proposito il primo *preset*, che imposta il fattore Q a 20, con il secondo, che lo imposta a 1000.

Tramite un *decoder* possiamo utilizzare il profilo spettrale ottenuto per modellare un secondo suono ad ampio spettro. Vediamo un esempio completo nella *patch* **13_02_vocoder.maxpat** (figura 13.4).

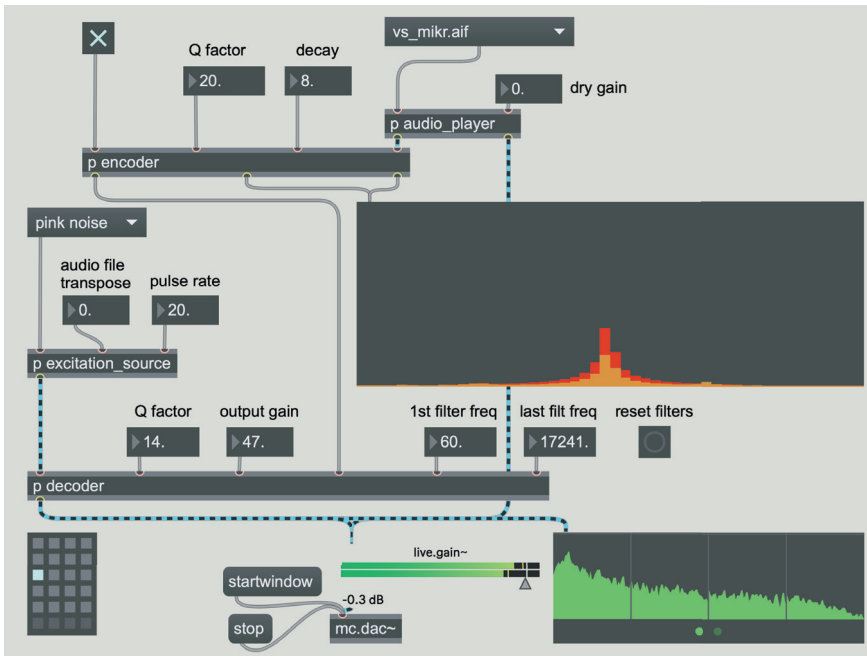


fig. 13.4: *patch* **13_02_vocoder.maxpat**

Prima di analizzare il contenuto delle *subpatch* vediamo la funzione dei parametri visibili in figura.
In alto abbiamo la *subpatch* "encoder" che contiene una versione elaborata della *patch* precedente; i due parametri numerici sono il fattore Q e il *decay* degli *envelope follower* (uno per ogni filtro). Il `toggle` controlla il `metro` interno e se viene disattivato lo spettro rimane "congelato". Tramite un `umenu`, collegato alla *subpatch* "audio_player", è possibile scegliere un file audio da inviare all'*encoder*.
Nella parte sinistra, a metà della figura, vediamo la *subpatch* "excitation_source" con la quale è possibile selezionare una sorgente di eccitazione per il banco di filtri ricevente; sono disponibili un suono orchestrale granulato, un generatore di rumore rosa, e un generatore di impulsi.
In basso infine abbiamo la *subpatch* "decoder" all'interno della quale si trova il banco di filtri che riceve, oltre al segnale della sorgente di eccitazione, il profilo

spettrale dalla *subpatch* "encoder" in alto. Possiamo modificare il fattore Q e il *gain* di questo banco di filtri, e inoltre possiamo decidere la frequenza del primo e dell'ultimo filtro (modificandole quindi rispetto a quelle del banco di filtri dell'*encoder*): all'interno della *subpatch*, come vedremo, c'è un algoritmo che stabilisce il relativo fattore di moltiplicazione per i filtri del banco. Tramite il *bang button* "reset filters" visibile in basso a destra, i parametri delle frequenze dei filtri tornano ad essere identici a quelli del banco di filtri dell'*encoder*.
Ascoltate tutti i *preset* e osservate come sono impostati i parametri. Provate a disattivare e attivare il `toggle` in alto a sinistra per "congelare" il profilo spettrale prodotto dall'encoder. Modificate i valori dei parametri per realizzare nuovi *preset*.
Vediamo ora il contenuto della *subpatch* [`p` encoder] che, come abbiamo detto, è derivato dalla *patch* **13_01_encoder.maxpat**. Aprite la *subpatch* con un doppio clic (figura 13.5).

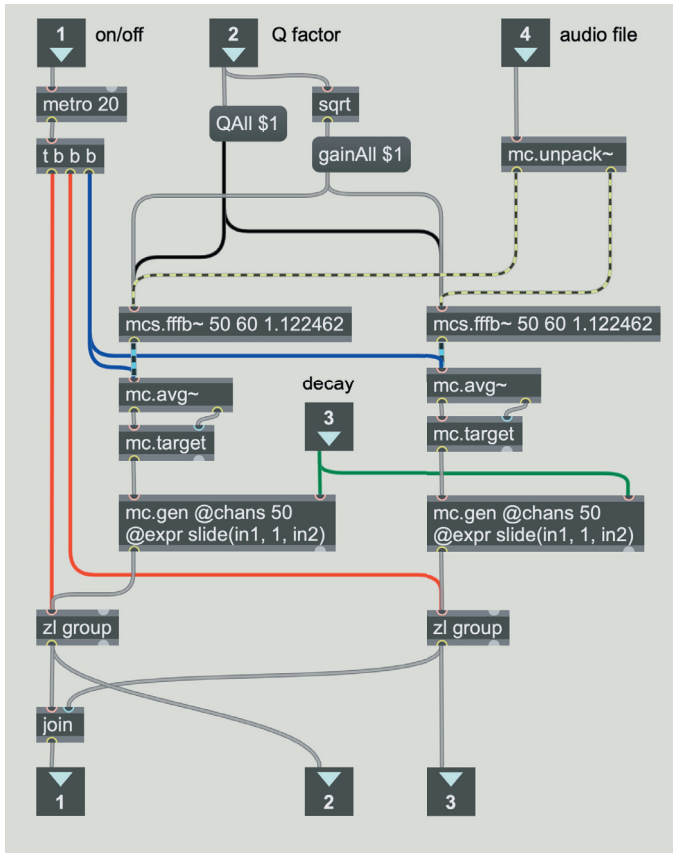


fig. 13.5: *subpatch* [`p` encode]

Il file audio da analizzare, che arriva al quarto `inlet` visibile in alto a destra, è stereofonico e i due canali vengono inviati a due banchi di filtri identici: come nella *patch* precedente ogni banco contiene 50 filtri a distanza di tono l'uno dall'altro e la frequenza del primo filtro è 60 Hz.

Dopo aver calcolato le ampiezze medie dei 50 filtri con **mc.avg~** ricaviamo altrettanti *envelope follower* inviando, tramite **mc.target**, i valori ottenuti all'oggetto **mc.gen** che implementa, grazie all'attributo @expr, l'operatore **slide**[2]. Utilizzando il terzo **inlet** della *subpatch* possiamo impostare un fattore di *decay* (cioè il parametro "slide-down" di *slide*) per l'*envelope follower* che, come possiamo constatare ascoltando i *preset*, dà un effetto di coda riverberante al suono in uscita dal *decoder*.
I valori corrispondenti agli *envelope follower* dei due banchi vengono raggruppati e inviati alla *subpatch* che contiene il *decoder*. I due banchi di *envelope follower* vengono anche inviati, tramite la seconda e terza uscita, ai due **multislider** sovrapposti visibili nella parte destra della *patch* principale, che mostrano il profilo spettrale dei due canali audio.
Passiamo alla *subpatch* [**p** decoder], visibile in figura 13.6.

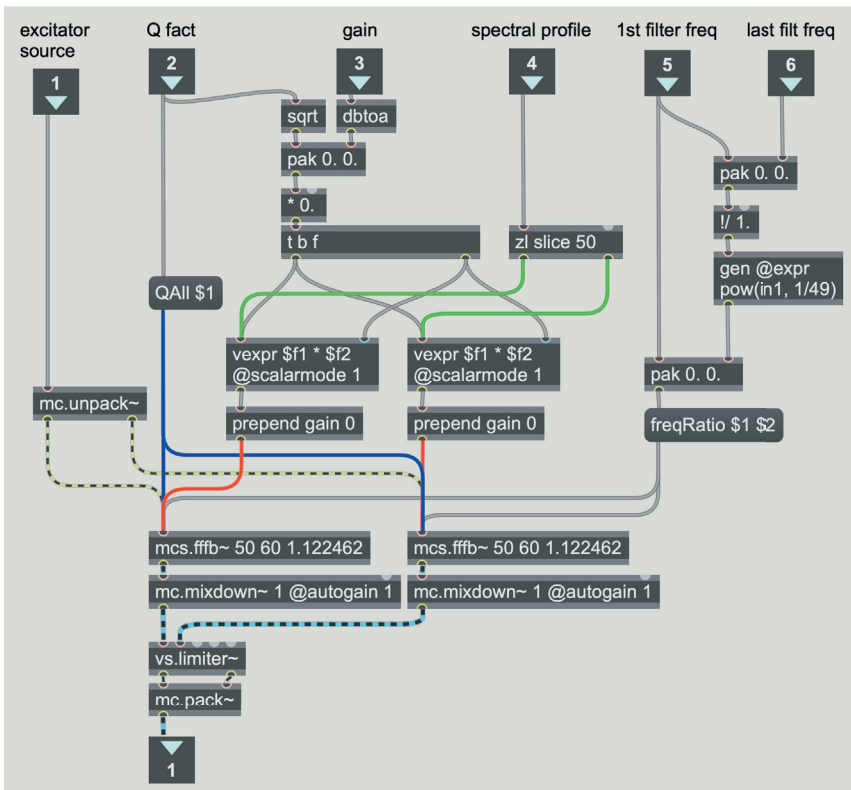


fig. 13.6: *subpatch* [**p** decoder]

Nella parte bassa vediamo i due banchi di filtri **mcs.fffb~**, uno per il canale audio sinistro e uno per il destro dell'uscita stereofonica. Questi due oggetti ricevono tutti i segnali e i messaggi in arrivo dagli **inlet**. Vediamoli a uno a

[2] Si tratta naturalmente della versione Gen dell'oggetto **slide~** che già conosciamo dal paragrafo 7.1P (dedicato agli *envelope follower*) del secondo volume.

uno: il primo `inlet` riceve la sorgente di eccitazione, il secondo riceve il fattore Q. Il terzo un fattore di *gain* che viene moltiplicato per la radice quadrata del fattore Q: come abbiamo già detto questo serve a compensare la perdita di energia dovuta alla selettività del fattore Q. Il valore ottenuto serve a riscalare i valori del profilo spettrale provenienti dall'*encoder* (quarto `inlet`).
Gli ultimi due `inlet` ricevono il valore di frequenza da assegnare al primo e all'ultimo filtro del banco. Innanzitutto viene calcolato, tramite l'oggetto [`!/` 1.], il rapporto tra la frequenza dell'ultimo filtro e quella del primo: con questa operazione otteniamo il fattore di moltiplicazione da applicare alla frequenza del primo filtro per ottenere quella dell'ultimo. Successivamente calcoliamo la radice 49ma di questo valore, e otteniamo il fattore di moltiplicazione dei filtri che, se applicato 49 volte di seguito, ci porta dalla frequenza del primo filtro a quella dell'ultimo. Utilizziamo questo fattore, unitamente alla frequenza del primo filtro, come parametro per i due `mcs.fffb~` (*message box* [freqRatio $1 $2]).

ATTIVITÀ

- Modificate il numero dei filtri contenuti nei banchi dell'*encoder* e del *decoder* (devono essere naturalmente identici), e modificate di conseguenza le *subpatch* che li utilizzano. Ascoltate attentamente i risultati ottenuti aumentando o diminuendo il numero dei filtri (non necessariamente un numero minore di filtri suona "peggio")
- Aggiungete un "gate" subito dopo gli oggetti `mc.avg~` della *subpatch* [`p` encode] (figura 13.5), in modo che i valori di ampiezza minori di una certa soglia vengano azzerati
- Simulate un "filtro brickwall" ponendo a 0, nel *decoder*, l'ampiezza dei primi filtri (*brickwall* passa-alto) e/o quella degli ultimi filtri (*brickwall* passa-basso): naturalmente saranno necessari due parametri che indichino quali sono il primo e l'ultimo filtro da utilizzare effettivamente
- Simulate un filtraggio "a pettine" azzerando un filtro su due, due filtri su tre etc.

Il *decoder* che abbiamo usato nella *patch* precedente lavora in sintesi sottrattiva: prendiamo un suono con uno spettro ricco di componenti (un rumore, un pieno orchestrale, etc.) e lo modelliamo attraverso dei filtri.
Sarebbe possibile un *decoder* in sintesi additiva? Certamente sì, è sufficiente sostituire il banco di filtri con un banco di oscillatori, ed è quello che facciamo nella *patch* **13_03_additive_vocoder.maxpat** (figura 13.7).

(...)

il capitolo prosegue con:

13.2 LA TRASFORMATA DI FOURIER

13.3 L'ELABORAZIONE NEL DOMINIO DELLA FREQUENZA: IL PHASE VOCODER
Filtri brickwall
Filtri crossover, multiband e random
LFO spettrale
Riduttore di rumore mediante noise gate spettrale
Bin shifting (frequency shifting spettrale): spostamento delle posizioni dei bin
Moltiplicazione degli indici dei bin (pitch shifting spettrale)
Stretching/shrinking spettrale
Delay dei bin con feedback (spectral delay)
Modifica della fase e phase stop
Randomizzazione delle fasi dei bin
Freeze
Sintesi incrociata tramite STFT

13.4 TIME STRETCHING E PITCH SHIFTING CON IL PHASE VOCODER

13.5 CONVOLUZIONE E CROSS-SYNTHESIS

13.6 RIVERBERI A CONVOLUZIONE

ATTIVITÀ
• Analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi, correzione di algoritmi

VERIFICHE
• Compiti unitari di reverse engineering

SUSSIDI DIDATTICI
• Lista oggetti Max - Lista messaggi, argomenti e attibuti per oggetti Max specifici - Lista operatori Gen - Glossario

Questa pagina è intenzionalmente lasciata in bianco

Interludio G
JITTER PER L'AUDIO

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume I e II, interludio F, capp. 10, 11, 12 e 13 (teoria e pratica)

OBIETTIVI

Abilità

- Saper programmare e utilizzare semplici algoritmi di gestione ed elaborazione di immagini e video con Jitter
- Saper programmare e utilizzare algoritmi di manipolazione dei suoni mediante matrici con Jitter
- Saper programmare e utilizzare algoritmi di granulazione, phase vocoder e wave terrain synthesis con Jitter
- Saper programmare e utilizzare algoritmi per la cross-synthesis fra due suoni campionati mediante la convoluzione offline realizzata con Jitter

Competenze

- Saper realizzare un breve studio basato sull'uso di Jitter

ATTIVITÀ

- Costruzione e modifiche di algoritmi con Jitter

SUSSIDI DIDATTICI

- Lista oggetti Jitter - Lista attributi e messaggi per oggetti Jitter specifici - Glossario

IG.1 INTRODUZIONE A JITTER

Jitter è un'estensione di Max che serve, tra le altre cose, ad elaborare matrici[1] di dati, immagini e video.
In questo interludio, dopo una prima introduzione alla gestione delle immagini e dei video che ci permetterà di familiarizzare con l'ambiente Jitter e con le sue caratteristiche, ci occuperemo principalmente dell'utilizzo di Jitter per creare matrici e insiemi di dati che possiamo utilizzare per il controllo, la visualizzazione, l'elaborazione e la generazione di segnali audio.
Vediamo innanzitutto cosa si intende per matrice. Se cercate la definizione in un testo di informatica troverete molto probabilmente che una matrice è un *array* a due dimensioni. Come sappiamo dal primo volume, un *array*[2] è un insieme ordinato di elementi dello stesso tipo (ad esempio valori interi, valori in virgola mobile, etc.) che possono essere identificati tramite un numero di indice.
Nel corso dei precedenti capitoli abbiamo avuto a che fare più volte con degli *array*. Gli oggetti `buffer~`, `table` e `multislider` ad esempio sono tutti contenitori di insiemi ordinati di elementi dello stesso tipo che possono essere identificati con un numero di indice, cioè sono contenitori di *array*.
Una matrice bidimensionale, invece, è un insieme ordinato di elementi dello stesso tipo rappresentabile come una tabella suddivisa in righe e colonne: in questo caso ciascun elemento è identificato da due numeri di indice, uno per la riga e una per la colonna in cui si trova l'elemento. In Jitter le matrici possono avere da 1 a 32 dimensioni, e naturalmente ciascun elemento di una matrice a n dimensioni viene identificato tramite *n* numeri di indice.
Abbiamo già avuto a che fare anche con le matrici bidimensionali, quando abbiamo usato l'oggetto che si chiama, appunto, `matrix~`.

[1] Per completezza segnaliamo che oltre alle matrici Jitter può operare anche con *textures*, cioè insiemi di dati che vengono elaborati, tramite appositi algoritmi, dal processore grafico del computer invece che dalla CPU. In questo capitolo ci occuperemo esclusivamente di matrici.
[2] Un *array* può essere chiamato anche "vettore", termine che useremo, con un significato diverso, nel paragrafo IG.7.

Tornate alla *patch* 11_18_DX4.maxpat (paragrafo 11.2P): ne vediamo un dettaglio in figura IG.1.

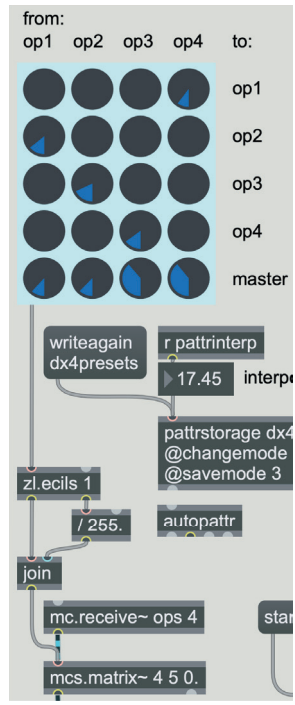


fig. IG.1: la matrice bidimensionale della *patch* 11_18_DX4.maxpat

Qui abbiamo una matrice bidimensionale con 4 colonne e 5 righe, che contiene quindi 20 (4x5) elementi: tramite i *dial* dell'oggetto interfaccia `matrixctrl` possiamo impostare il valore di ciascun elemento. Questi valori rappresentano, come sappiamo, il fattore di riscalamento delle connessioni tra i segnali in ingresso e i segnali in uscita dall'oggetto `mcs.matrix~` (versione multicanale di `matrix~`) visibile in basso nella figura.
Un altro esempio di matrice bidimensionale di cui abbiamo parlato è il *wave terrain* (vedi paragrafo 11.5).

Vediamo finalmente come funzionano le matrici in Jitter: caricate il file **IG_01_jitter_matrix.maxpat** (figura IG.2).
L'immagine contiene 5 mini *patch* che mostrano alcune delle caratteristiche di base di Jitter. Notate che ci sono diversi oggetti il cui nome comincia con il prefisso "jit": questo prefisso identifica gli oggetti Jitter (così come il prefisso "mc" identifica gli oggetti multicanale, o la tilde identifica gli oggetti MSP).
Prima di analizzare le 5 mini *patch* è importante sottolineare che le matrici Jitter, contrariamente alle matrici di cui abbiamo parlato sopra, possono contenere più valori per ogni elemento (o cella) della matrice stessa. Possiamo cioè avere, ad esempio, una matrice bidimensionale di 2x2 celle, ciascuna delle quali contiene 3 valori, per un totale di 2 * 2 * 3 = 12 valori. Il numero di valori per cella deve essere lo stesso in tutta la matrice.

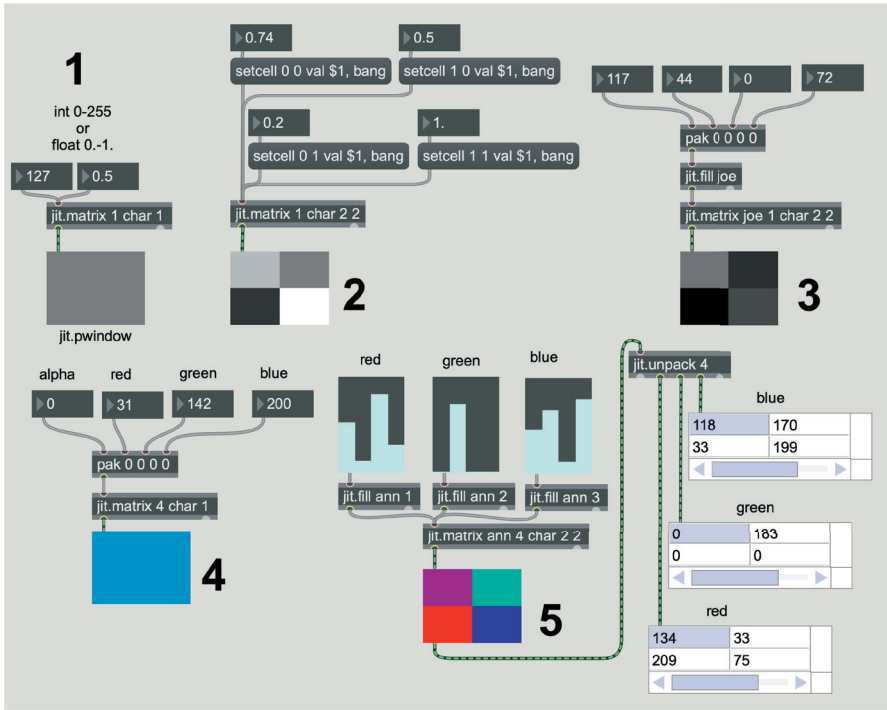


fig. IG.2: file **IG_01_jitter_matrix.maxpat**

L'oggetto **`jit.matrix`** crea una matrice, cioè uno spazio in memoria che conterrà i dati, ed è quindi l'oggetto fondamentale della libreria.
Nella prima mini *patch* questo oggetto ha tre argomenti: il primo indica il numero di valori che ci sono per ogni cella della matrice (ci si riferisce a questi valori anche come piani di una matrice); la nostra matrice ha quindi un solo valore per cella. Il secondo argomento stabilisce di che tipo sono questi valori: il tipo *char* indica un numero intero compreso tra 0 e 255 (cioè un numero rappresentabile con 8 bit, vedi anche il paragrafo 5.2T nel secondo volume). A seguire ci sono tanti argomenti quante sono le dimensioni della matrice, e ogni argomento specifica la grandezza della relativa dimensione: in questo caso c'è un solo argomento che vale 1; abbiamo quindi una matrice monodimensionale con una sola cella. Si tratta della più piccola matrice possibile, costituita da un singolo elemento.
L'oggetto grafico che si trova al di sotto di `jit.matrix` si chiama **`jit.pwindow`**, e lo abbiamo già visto in diverse *patch* del paragrafo 11.5P. Questo oggetto ci permette di vedere il contenuto di una matrice in forma grafica: per la precisione, quando la matrice contiene un solo piano (cioè un solo valore per cella) come in questo caso, i valori vengono visualizzati come sfumature di grigio.
Parliamo ora dei valori: come abbiamo detto il tipo *char* corrisponde a un numero intero a 8 bit, tra 0 e 255. Se modifichiamo il valore dell'*integer number box* in alto a sinistra, possiamo vedere che il colore del `jit.pwindow` passa gradualmente dal nero (valore 0) al bianco (valore 255). Una convenzione di

Jitter stabilisce che i valori di tipo *char* possono anche essere rappresentati come numeri in virgola mobile compresi tra 0 e 1: provate a modificare il valore del *floating point number box* collegato al **jit.matrix** e verificate che anche in questo caso il colore di **jit.pwindow** passa attraverso le diverse sfumature di grigio.
Notate che la connessione tra **jit.matrix** e **jit.pwindow** avviene attraverso un cavo verde con strisce orizzontali nere: questo nuovo *pattern* ci dice che i dati trasmessi non sono né semplici valori numerici, né segnali; quello che viene trasmesso è, infatti, il nome della matrice che è stata creata da **jit.matrix**. L'oggetto Jitter ricevente, in questo caso **jit.pwindow**, è in grado di risalire dal nome al contenuto della matrice. Se volete vedere qual è il nome della matrice collegate un oggetto **print** all'uscita di **jit.matrix** e poi inviate un nuovo valore numerico. Sulla Max Console apparirà la scritta "jit_matrix" e un numero a molte cifre preceduto da una "u": questo è il formato dei nomi assegnati automaticamente alle matrici via via che vengono create; come vedremo tra poco possiamo dare alle matrici anche nomi più significativi.

Nella seconda mini *patch* abbiamo una matrice bidimensionale di 2x2 celle; anche in questo caso c'è un solo piano (o valore per cella) e il tipo è *char*. Questa volta se vogliamo modificare i valori dobbiamo specificare la cella di destinazione: utilizziamo quindi il messaggio "setcell" che deve essere seguito dall'indice della cella (cioè dai valori delle coordinate, che partono da 0) e dall'argomento "val" seguito dal valore che vogliamo impostare. Dopo aver impostato il valore è necessario mandare un *bang* all'oggetto **jit.matrix** affinché quest'ultimo comunichi il nome della matrice agli oggetti collegati: per questo motivo, dopo il messaggio "setcell" con i relativi argomenti, il *message box* contiene un messaggio "bang".
Se osservate i valori delle coordinate all'interno dei quattro *message box* noterete che sono, nell'ordine, [0, 0] [1, 0] [0, 1] [1, 1]. La prima coordinata è il numero della colonna (cioè la coordinata *x*) mentre la seconda coordinata è il numero della riga (cioè la coordinata *y*): modificate i valori dei quattro *number box* e osservate come la cella relativa cambi di colore. Notate che la coordinata [0, 0] corrisponde all'angolo in alto a sinistra.

Possiamo anche mandare una lista di valori alla matrice, utilizzando l'oggetto **jit.fill** (mini *patch* numero 3). Questo oggetto ha bisogno di conoscere il nome della matrice da riempire. È possibile dare un nome a una matrice inserendolo come primo argomento in **jit.matrix**: notate infatti che nella terza mini *patch* la matrice si chiama "joe". Questo nome serve ad altri oggetti per lavorare su una stessa matrice, esattamente come avviene con il nome che l'oggetto **buffer~** utilizza per condividere il proprio contenuto. È anche possibile (esattamente come con **buffer~**) avere più oggetti **jit.matrix** con lo stesso nome che si riferiscono alla stessa matrice.

L'oggetto [`jit.fill` joe] riceve una lista di valori e memorizza ogni elemento in una cella della matrice, procedendo riga per riga. Quando ha finito di scrivere i valori nella matrice invia un *bang* alla sua uscita di sinistra: abbiamo collegato questa uscita con l'ingresso[3] di **`jit.matrix`** che a sua volta invia il nome della matrice a **`jit.pwindow`**.

Con la mini *patch* numero 4 vediamo come si visualizzano i colori nell'oggetto **`jit.pwindow`**: abbiamo bisogno di una matrice con quattro piani, cioè con quattro valori per ogni cella. Questi quattro valori corrispondono rispettivamente al canale alfa[4] e alle componenti rossa, verde e blu: si tratta in altre parole di una codifica ARGB (*alpha*, *red*, *green*, *blue*) in cui dosando opportunamente ciascuno dei tre colori primari possiamo ottenere circa 16 milioni di colori[5]. La matrice di questa mini *patch* è costituita da una singola cella, ed è sufficiente inviare una lista con i quattro valori dei quattro piani per visualizzare il colore relativo.
Provate a modificare i valori per generare nuovi colori.

La matrice della mini *patch* numero 5 è bidimensionale (2x2) e avendo 4 piani permette di generare celle "colorate". In questo caso utilizziamo tre **`multislider`** collegati a tre **`jit.fill`** per impostare i valori dei tre piani corrispondenti ai colori primari. Il nome della matrice è "ann", e notate che ogni **`jit.fill`** ha un secondo argomento che specifica il numero di piano da riempire: i piani sono numerati a partire da 0; abbiamo quindi 0 = alfa (che non viene impostato perché non ha effetto), 1 = rosso, 2 = verde, 3 = blu.
All'uscita di **`jit.pwindow`** abbiamo collegato un nuovo oggetto, **`jit.unpack`**, che separa i piani della matrice che riceve. Alle uscite corrispondenti ai piani 1, 2 e 3 abbiamo collegato tre **`jit.cellblock`**. Abbiamo già parlato di questo oggetto al paragrafo 2.2P del primo volume, dove lo abbiamo usato per visualizzare i valori di un segnale multicanale: qui lo utilizziamo nella sua funzione originaria che è appunto quella di visualizzare il contenuto di una matrice.
Se aprite l'*inspector* di uno dei **`multislider`** potete constatare che generano liste di valori in virgola mobile compresi tra 0 e 1; ma come si può vedere i valori ricevuti dagli oggetti **`jit.cellblock`** sono interi compresi tra 0 e 255. Questo significa che i numeri in virgola mobile vengono convertiti in interi a 8 bit prima di essere memorizzati nella matrice.

(...)

[3] Notate che il cavo che collega **`jit.fill`** a **`jit.matrix`** è un cavo grigio, perché trasmette un semplice messaggio *bang*.
[4] Il canale alfa serve a regolare la trasparenza della cella quando si miscelano più immagini: nei casi che stiamo trattando in questo paragrafo il parametro è ininfluente.
[5] Ciascuna componente può avere 256 valori, e questo ci dà un totale di 256x256x256 = 16777216 colori diversi.

il capitolo prosegue con:

ATTIVITÀ
• Analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi, correzione di algoritmi

VERIFICHE
• Compiti unitari di reverse engineering

SUSSIDI DIDATTICI
• Lista oggetti Jitter - Lista messaggi, argomenti e attibuti per oggetti Jitter specifici - Glossario

Alessandro Cipriani • Maurizio Giri

Musica Elettronica e Sound Design

Teoria e Pratica con Max 8 • volume 3

Argomenti trattati

Riverbero e usi creativi del riverbero - Spazializzazione a due o più canali – AM, RM, SSB, FM, e PM - Distorsione non lineare - Wave terrain synthesis - Split synthesis - Sintesi granulare e particellare - Granulazione e segmentazione di suoni campionati - Vocoder - Analisi e risintesi - Cross-synthesis - Convoluzione - Jitter per l'audio - Programmazione con GEN

"Non mancano, al mondo, libri che cercano di mostrare l'erudizione di chi li scrive. Più rari, invece, sono quei libri che si preoccupano di chi li legge, di accompagnare i lettori in un viaggio che li lascerà cambiati. I testi scritti da Cipriani e Giri fanno parte di questa rara categoria: sono testi che *spiegano*. (…) Il terzo volume di *Musica elettronica e sound design* è un caleidoscopico catalogo di idee e applicazioni per analizzare, sintetizzare e trasformare i segnali ad ampio raggio. (…) Cipriani e Giri riescono a parlare a tutti senza indebolire il costrutto teorico e senza inutili specializzazioni. Un magistrale equilibrio tra comprensibilità, funzionalità ed ampiezza." (dalla prefazione di **Carmine-Emanuele Cella**, Assistant Professor in Music and Technology, CNMAT - University of California, Berkeley).

Questo è il terzo tomo di un sistema didattico organico comprendente una corposa sezione online composta da centinaia di esempi sonori e interattivi o video, glossari di teoria e di pratica, test, programmi scritti in Max, una libreria di oggetti Max espressamente creata per questi volumi e numerose attività pratiche anche con Gen e Jitter.

ALESSANDRO CIPRIANI è coautore con R.Bianchini del testo *Virtual Sound* sulla programmazione in Csound. Le sue composizioni sono pubblicate da *Computer Music Journal*, *International Computer Music Conference*, CNI, etc. Ha scritto musiche per il Teatro dell'Opera di Pechino, per film e documentari in cui ambienti sonori, dialoghi e musica, elaborati al computer, si fondono ed hanno funzioni miste; fra questi, i film muti *La Corazzata Potemkin*, *Inferno*, e *Das Cabinet des Dr. Caligari*, pubblicati in DVD dalla Cineteca di Bologna con il collettivo Edison Studio. Ha tenuto seminari in numerose università (Univ. of California, Sibelius Academy Helsinki, Conservatorio Tchaikovsky di Mosca, DMU-Leicester, etc.). È titolare della Cattedra di Composizione Musicale Elettroacustica del Conservatorio di Frosinone. È membro dell'Editorial Board della rivista *Organised Sound* (Cambridge University Press).

MAURIZIO GIRI è docente di Composizione ed insegna tecniche di programmazione con Max nei Conservatori di Latina e Frosinone. Ha scritto musica strumentale ed elettroacustica. Attualmente si occupa di musica elettronica e nuove tecnologie applicate all'elaborazione digitale del suono, all'improvvisazione e alla composizione musicale. Ha scritto software per l'improvvisazione elettroacustica e il live electronics. Ha fondato *Amazing Noises*, una software house che sviluppa applicazioni musicali e plug-in per dispositivi mobili e computer, ed è *partner* di Ableton con cui ha pubblicato diversi *device* Max for Live. Ha pubblicato tutorial su Max in riviste specializzate. È stato artista residente a Parigi (*Cité Internationale des Arts*) e a Lione (GRAME). Ha collaborato con l'*Institut Nicod* alla *École Normale Supérieure* di Parigi a un progetto di filosofia del suono.

ISBN 978-88-99212-22-3

www.contemponet.com

€ 54.00 iva inclusa