

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA



Università degli Studi di Roma "Tor Vergata"

Master in **Sonic Arts**

Università degli Studi di Roma "Tor Vergata"

Master in *Sonic Arts*

A. A. 2019-2020

Tesi finale

Synaesthetik - Sistema integrato per generazione
di suono e immagine in tempo reale in ambiente
GNU/Linux

Relatore:

Prof. Riccardo Santoboni

Candidato:

Francesco Casanova

Indice

1.	Introduzione.....	pag.1
1.1.	Sinestesia: da Newton all'era digitale.....	pag.1
1.2.	Integrare e condividere.....	pag.4
2.	Stato dell'arte.....	pag.6
2.1.	Dispositivi integrati per il suono digitale.....	pag.8
2.2.	Dispositivi integrati per l'immagine digitale.....	pag.10
3.	Ricerca e sviluppo dell'hardware.....	pag.13
3.1.	Scelta dei componenti.....	pag.13
3.2.	Assemblaggio.....	pag.14
4.	Ricerca e sviluppo del software.....	pag.15
4.1.	Scelta del linguaggio per il DSP e per la UI.....	pag.16
4.2.	Architettura software per l'audio.....	pag.18
4.2.1.	Generazione e sequenza di eventi sonori.....	pag.19
4.2.2.	Logica di funzionamento.....	pag.31
4.2.3.	Interfaccia per l'utente.....	pag.33
4.3.	Architettura software per il video.....	pag.39
4.3.1.	Ofelia e OpenGL in Pure Data.....	pag.39
4.3.2.	Descrittori audio e audio-reattività.....	pag.43
5.	Sviluppi futuri.....	pag.46
6.	Bibliografia.....	pag.46

1. Introduzione

1.1. Sinestesia: da Newton all'era digitale

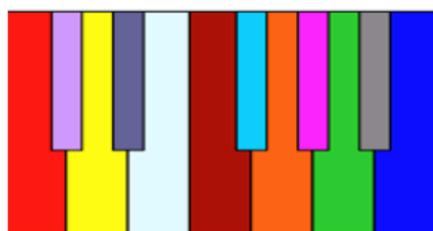
Nel 1704 Isaac Newton nel trattato Opticks, a seguito delle celebri osservazioni del 1666 inerenti alla scomposizione della luce bianca in colori attraverso un prisma triangolare, dedusse che i colori fossero sette in analogia con le sette note musicali. Nell'arcobaleno colorato che fuoriusciva dal prisma, che Newton chiamò “spettro”, furono distinti: rosso, arancio, giallo, verde, blu, indaco e violetto. Newton ipotizzò anche che ci fosse una stretta connessione tra note e colori nelle loro caratteristiche strutturali comuni. Per esempio, proprio come accade per gli intervalli musicali, Newton evidenziò l'esistenza di una relazione armonica tra i colori, a seconda della distanza che ricoprono nello spettro visivo.

Anche il matematico Louis Bertrand Castel elaborò una teoria sulla connessione tra suono e colore nel suo testo Optique des couleurs del 1740 e per primo progettò uno strumento che potesse illustrare questa analogia. Secondo Castel suono e colore hanno caratteristiche simili condividendo la loro stessa natura ondulatoria e vibrazionale, il loro propagarsi in maniera rettilinea e l'essere suscettibili alla deviazione e al cambio di direzione a seguito dei fenomeni di rifrazione e riflessione. La ricerca di un'armonia tra suoni e colori lo indusse a progettare uno strumento che potesse dimostrare concretamente le sue teorie, riproducendo simultaneamente suoni e colori: il clavecin oculaire. Questo strumento inaugurò la realizzazione di strumenti similari progettati secondo l'assunto che ci fosse una qualche stretta analogia tra note e colori e che questi apparecchi sarebbero stati in grado di rendere “visibile”

la musica". Non solo, la stretta relazione tra strumento e performer celebra anche l'inizio del connubio tra arte e tecnologia nella performance audiovisiva che da questi si protrarrà nei secoli sino alle contemporanee esperienze di realtà virtuale.¹ Oggi sappiamo che i comportamenti fisici del suono e della luce hanno tanto in comune ma la correlazione tra i due fenomeni avviene a livello cerebrale col fenomeno della sinestesia. La sinestesia è un fenomeno sensoriale-percettivo in cui determinati stimoli evocano sensazioni di natura diversa da quella normalmente sperimentata, indica una " fusione" delle percezioni di sensi distinti in un'unica sfera sensoriale. Quest'esperienza deriva dall'insorgenza di una sensazione (uditiva, visiva, olfattiva, tattile ecc.) in concomitanza ad una percezione di natura sensoriale diversa. Durante tale fenomeno, si verifica, quindi, un sincronismo funzionale di due o più organi di senso o facoltà cognitive. Ad esempio, un'immagine visiva può essere evocata in seguito ad uno stimolo acustico.

Aleksandr Skrjabin, noto compositore russo del primo novecento, fece tesoro delle precedenti ricerche condotte sul fenomeno e progettò una delle prime opere basate sullo studio della sinestesia il Prometeo.

Skrjabin nel Prometeo utilizza una corrispondenza tra le altezze del sistema temperato e i colori dello spettro visivo, corrispondenza che viene realizzata con un organo di un'ottava (Luce) in grado di generare fasci luminosi. La parte per Luce è annotata sulla parte superiore della



codifica della rappresentazione colore-altezza in Skrjabin

¹ Emanuele Ponzio - **Immagine in tempo reale** - Storie, pratiche, teorie per una introduzione alla performance audiovisiva - Mimesis edizioni - 2019

partitura, con scrittura a due voci: una cambia con l'armonia con riferimento alla nota fondamentale dell'armonia prevalente producendo il colore che Skrjabin ha associato a ciascun tasto; l'altra è costituita da note molto lunghe; i cambiamenti di colore si verificano molto lentamente. Skrjabin anticipa così il concetto di opera multimediale composta da strumenti tecnici in grado di fondere esperienze sensoriali in favore di una macro-opera d'arte più complessa.

Data l'origine storica delle sperimentazioni artistiche attorno al fenomeno della sinestesia, con l'avvento della strumentazione elettronica analogica nel corso del XX secolo, si sono divulgati in maniera capillare la progettazione e l'utilizzo di tecnologie sinestetiche sia in ambito accademico che in quello dell'emergente cultura Pop (Clavilux, liquid e light show, ecc.), fino a creare veri e propri settori d'interesse culturale (Happening Fluxus, video art). Con l'avvento del computer e dell'era digitale i costi e la performatività delle strumentazioni sono avanzati in maniera inversamente proporzionale fino ad arrivare ai moderni strumenti di programmazione audiovisiva nel mondo del sound design, dei VFX e del VideoMapping accessibili da un qualsiasi PC di fascia media. Lo studio sulla correlazione tra suono e immagine si è espanso come sono avanzati gli studi sui fenomeni sensoriali (Gestalt, psicoacustica) e non si è limitato alla correlazione tra intervallo tonale e colore, ma ha incluso tutti gli strumenti di analisi del suono e dell'immagine digitale come sorgente di dati per la sonificazione e la video-reattività. La visione si è allontanata da un utilizzo passivo del mezzo tecnologico, il medium viene usato per precise finalità comunicative e non si ferma ad una pura documentazione della realtà o a una sua riproduzione astratta. La sua capacità di intervenire sul reale e sulla sua percezione si traduce nella messa in

discussione della posizione dello spettatore. Questo avviene in particolare nelle opere interattive della computer art in cui il dato (captato da sensori) diviene il principale esecutore dell'opera.

In questo contesto di sperimentazione audiovisiva si inserisce il progetto di questa tesi: lo sviluppo di uno strumento per la generazione in tempo reale di suono e immagini ispirato all'estetica della computer music e della computer art. Oggigiorno gli strumenti per realizzare opere di arte digitale necessitano di un complesso lavoro di programmazione e di performance informatiche non di poco conto. Lo scopo di questo lavoro è quello di creare uno strumento portatile a basso consumo energetico in grado di generare e programmare suoni e immagini in movimento sincronizzati tra loro.

1.2. Integrare e condividere

Le tecnologie definite integrate o “embedded” oggi circondano la vita di ogni individuo, si espandono a macchia d’olio in ogni dispositivo che circonda la sfera dell’umano. “Smartphone”, “smart-house”, “smart-car”, ogni congegno elettronico viene espanso e le sue mansioni aumentate, grazie all’enorme utilizzo di micro-controllori e micro-processori programmabili. Mentre l’economia digitale spinge l’utente a ricercare prestazioni sempre più elevate da prodotti commerciali, l’idea che ha mosso questa ricerca è stata quella di cercare di sfruttare al massimo le potenzialità di hardware con caratteristiche limitate cercando di ottenere risultati stabili e performativi sposando la politica del Do It Yourself.

La sfida che ho lanciato per la realizzazione di questo lavoro è nata da un’indagine personale sul mondo dello sviluppo di hardware. L’economia

del silicio ha sposato l'ottica consumistica mettendo sul mercato prodotti in competizione tra loro che ostacolano la formazione mantenendo la conoscenza ad un livello superficiale e riducendola a proprietà intellettuale privata, privandola, dunque, della sua naturale essenza, il suo utilizzo comune. Questa ricerca è in netto contrasto con questa visione. L'idea è quella di costruire uno strumento creativo che sia alla portata di tutti e di cui le componenti fondanti, come la struttura hardware e software, sia aperta per comprenderne la costruzione, per essere manomessa e adeguata alle proprie esigenze. Tutto ciò non sarebbe stato possibile senza l'utilizzo di strumenti pensati in un ottica di ricerca e condivisione.

“Essere protagonisti attivi, agire e non subire il cambiamento; usare la tecnologia per soddisfare i propri bisogni e i propri desideri; porsi in un continuo dialogo con il flusso di informazioni delle reti, informatiche e umane.”

(Collettivo Ippolita) ²

² Ippolita - OPEN NON E' FREE - Comunità digitali tra etica hacker e mercato globale - Eleuthera – 2005

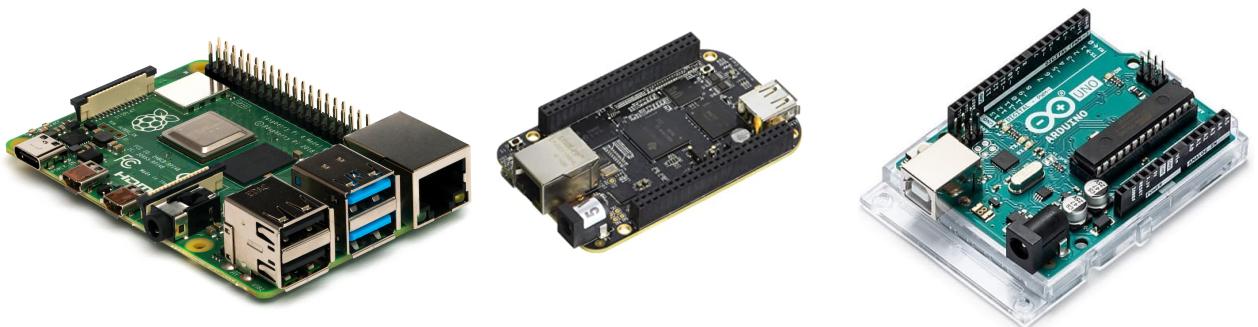
2. Stato dell'arte

Come abbiamo detto, un sistema integrato può essere definito come un componente elettronico che opera come dispositivo responsabile della gestione di una o più funzioni. L'informatica integrata può essere classificata come una sottoclasse dei sistemi integrati, in cui il componente hardware principale ha un'architettura simile a un computer. Come in altri tipi di sistemi embedded, il computer interagisce con vari componenti hardware, con la differenza che può eseguire programmi specifici dell'utente. Questa definizione più ampia consente l'inclusione di microcontrollori o processori che non eseguono necessariamente un sistema operativo ma che possono essere programmati arbitrariamente per eseguire algoritmi complessi. In alcune applicazioni informatiche integrate può essere difficile dire se il dispositivo è o meno un computer. Il miglior esempio sono gli smartphone. Sono telefoni con computer incorporato o solo computer con una funzione diversa?

All'interno di questa macro-area prenderemo in esempio alcuni hardware in grado di essere programmati per avere comportamenti sonori, visuali e interattivi diversi. Di questi vedremo quelli caratterizzati dall'uso della tecnologia embedded con processori ARM con distribuzioni GNU/Linux. Avendo un sistema operativo installato, ciò permette l'esecuzione delle stesse identiche attività che possono essere eseguite dal desktop o dal computer portatile. Questa fusione tra lo strumento e i componenti elettronici rende lo strumento completamente autonomo e libero dalla necessità di altre periferiche esterne. In altre parole, il computer desktop o laptop esterno scomparirebbe dall'equazione

Il PC desktop convenzionale è composto da diversi circuiti integrati, come la CPU, la memoria ad accesso casuale (RAM) o l'unità di elaborazione grafica (GPU), che interagiscono tramite bus di comunicazione su un grande PCB, comunemente chiamato scheda madre. I system on a chip (SoC) integrano tutti questi componenti in un unico chip. Funzionano in modo molto simile a un computer, eseguendo un sistema operativo che controlla diversi processi simultanei che condividono risorse come la CPU o la memoria. Oggi, molti SoC utilizzano il sistema operativo Linux, che consente di eseguire gran parte del software di computer desktop e laptop, senza la necessità di un refactoring significativo.

Il Raspberry Pi e il BeagleBone sono due esempi di SoC basati su ARM che sono diventati molto popolari sia in ambito accademico che hobbistico allo stesso modo di Arduino.



I microcomputer con processori ARM utilizzano una forma più tradizionale di elaborazione seriale: sono in grado di eseguire complesse sintesi audio ma a costo di una maggiore latenza. Al Centre for Digital Music, School of EECS dell'Università Queen Mary di Londra hanno eseguito dei test rigorosi che consistevano nella "misurazione del ritardo tra il suono prodotto premendo un pulsante sulla tastiera e un'uscita audio sinusoidale

attivata sul computer premendo il pulsante stesso³, per diverse configurazioni di buffer e dimensioni del periodo a 16 bit e una frequenza di campionamento di 44,1 kHz. I risultati mostrano latenze che vanno da 2,5 a 12 ms nelle varie configurazioni, con jitter non superiore a 1 ms, che è ben all'interno delle grandezze di latenza generalmente accettate per le prestazioni in tempo reale.

2.1 Dispositivi integrati per il suono digitale

Qu-bit Electronix - Nebulae

Nebulae è un campionatore granulare e una piattaforma DSP basato su Raspberry Pi. Il suo motore audio fornisce come standard un potente strumento di sintesi granulare. Può utilizzare un live input, un buffer registrato o un file audio come materiale sorgente. Il buffer del live input può registrare fino a cinque minuti di audio stereo a 48kHz, 24 bit.

Inoltre, è possibile caricare simultaneamente fino a 75 MB di audio stereo tramite la porta USB.

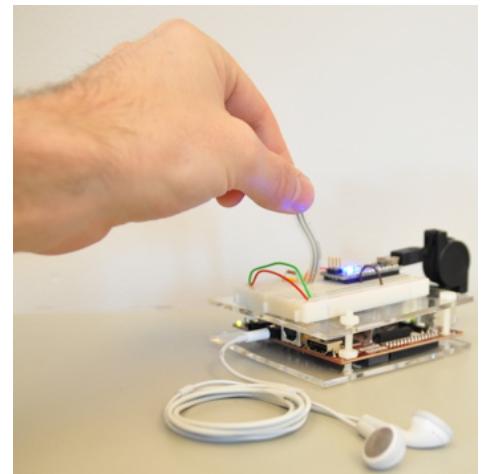


Nebulae è considerato anche come piattaforma DSP open source con un'ampia libreria di strumenti alternativi disponibili. È possibile scrivere firmware alternativi con Pure Data, Csound, SuperCollider e C++.

³ James William TOPLISS - Victor ZAPPI - Andrew McPHERSON- [Latency Performance for Real-Time Audio on BeagleBone Black](#)

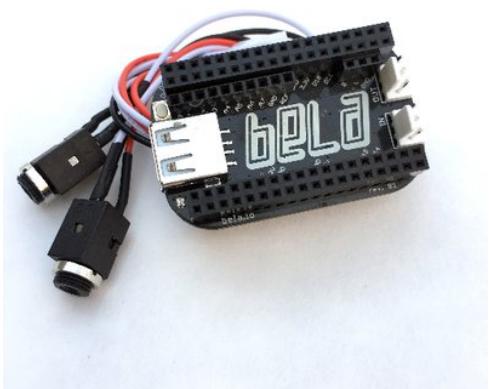
Satellite CCRMA⁴

Satellite CCRMA è una distribuzione Linux gestita dal Center for Computer Research in Music and Acoustics (CCRMA) della Stanford University. È una distribuzione che contiene miglioramenti a livello di sistema e applicazioni pre-compilate, rendendo i sistemi audio per PC Linux più accessibili agli utenti inesperti. Satellite CCRMA segue gli stessi principi generali ma è stato adattato per computer a scheda singola, come BeagleBoard-xM e RaspberryPi. Rappresenta un buon punto di partenza per sistemi musicali integrati, ma richiede comunque una competenza relativamente elevata con gli strumenti della linea di comando Linux. Senza alcun aggiornamento da diversi anni, sembra che Satellite CCRMA non sia attualmente manutenuto.



Bela⁵

Bela è una piattaforma per la creazione di strumenti musicali integrati, sviluppata dall'Augmented Instruments Laboratory della Queen Mary University di Londra. Questo sistema utilizza BeagleBoneBlack accoppiato con elettronica personalizzata che fornisce un'uscita stereo, 8 Input/Output analogici e 16 Input/Output



⁴ <https://ccrma.stanford.edu/~eberdahl/Satellite/>

⁵ <https://bela.io/>

digitali, con latenze fino a 100 μ s. L'uscita audio può essere espansa a 6 canali a 44,1 kHz o 10 canali a 22,05 kHz. Bela è programmato utilizzando un IDE online e può compilare programmi C/C ++ direttamente su BeagleBone. Può anche eseguire patch PureData, CSound e SuperCollider, anche se con alcune limitazioni. I componenti hardware e software sono open-source e le schede dei circuiti pre-assemblate sono vendute da Augmented Instruments Ltd.

2.2 Dispositivi integrati per l'immagine digitale

A differenza di piattaforme già ben standardizzate per l'implementazione di sistemi audio integrati, esperienze riguardo sistemi di generazione di immagini animate in tempo reale con processori ARM sono varie, alcuni con un ottica commerciale, altre costruite attorno a frameworks aperti come Processing e OpenFrameworks tra le comunità di maker.

MinIMad⁶

Nasce come espansione hardware del rinomato software di video-mapping “MadMapper”. Basato su Raspberry Pi 3b+. Permette all'utente di esportare sequenze video dal PC pre-prodotte per poi essere inserite nella scheda SD del Raspberry Pi ed eseguite dall'uscita HDMI mediante un software di playback video proprietario. Ha la possibilità di interconnettersi con altri MinIMad per aumentare la superficie di proiezione suddividendola in più schermi. Ha la possibilità di ricevere



⁶ <https://madmapper.com/minimad/>

messaggi OSC per essere controllato in remoto da un PC. Non utilizza motori di rendering video complessi poichè svolge la funzione di videoplayback anche con parti audio già pre-prodotte e pre-sincronizzate sul software PC.

LZX Industries⁷

LZX Industries è nata nel 2008 dalla scena DIY tra l'Australia e gli USA, progettano strumenti video creativi ispirati all'estetica della video-sintesi, attraverso l'utilizzo di microcontrollori e microprocessori assemblati perlopiù in formato eurorack.

Andor 1 Media Player , come nel caso di MiniMad è un lettore multimediale autonomo basato su Raspberry Pi 3b+ per video-arte, produzione scenica, pareti video, segnaletica e altre applicazioni. Riproduce loop video con audio senza interruzioni o mostra immagini fisse pre-caricate. Possono essere collegati tra loro più esemplari per suddividere la proiezione in più schermi. Lo caratterizzano funzionalità interessanti come un editor di playlist integrato che supporta playlist interattive, modalità video wall con

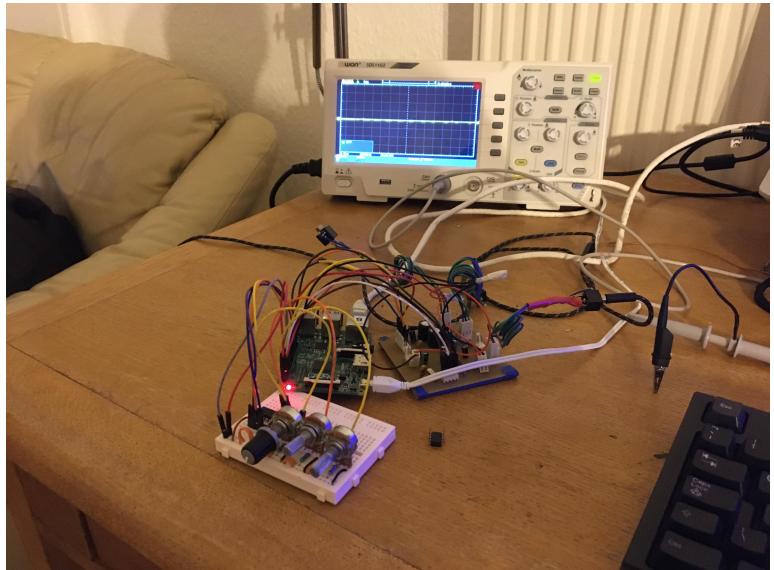


⁷ <https://lzxindustries.net/>

compensazione della cornice, controllo MIDI, opzioni di rotazione video e modelli di test del colore integrati.

bzzzbz⁸

bzzzbz è un sintetizzatore video basato su Raspberry Pi realizzato all'Università di Glasgow. L'hardware reagisce dinamicamente a un ingresso audio per generare immagini e visuals che possono essere manipolate in tempo reale dando vita a performance audiovisive con dispositivi integrati. Questo prodotto è pensato per essere un'alternativa più semplice ed economica ai sintetizzatori video analogici commerciali, fornendo comunque un'interfaccia flessibile. Essendo un progetto in fase di sviluppo non ha ancora trovato una forma definitiva ma è ancora in fase prototipale.



⁸ https://twitter.com/bzzzbz_video

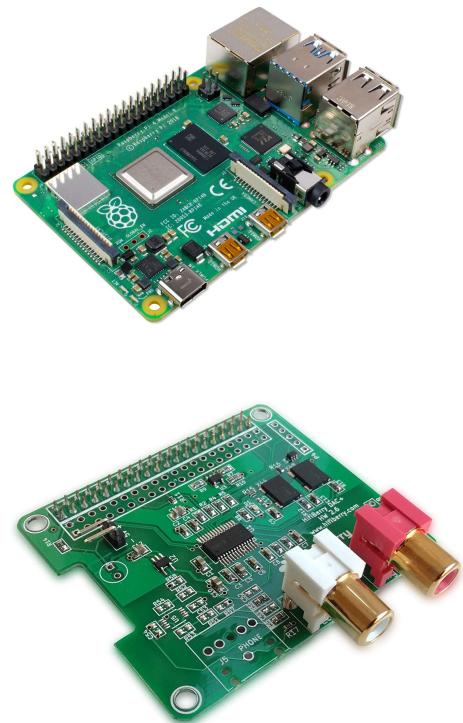
3. Ricerca e sviluppo dell'hardware

Per quanto riguarda lo sviluppo hardware di Synaesthetik, dopo aver analizzato alcuni degli esempi precedentemente citati, mi sono orientato verso la costruzione di una macchina in grado di svolgere sia le mansioni di generazione audio sia generazione video in tempo reale e sincronizzati tra loro in un unico SoC. Ho fatto una scelta dei componenti che potessero soddisfare l'idea di creare una macchina interfacciabile con cui l'utente può interagire e che sia al contempo manipolabile senza conoscere linguaggi di programmazione complessi come C++. A causa della mia ridotta conoscenza del mondo dell'elettronica, intesa come saldatura di componenti in una PCB progettata ex-novo, mi sono orientato verso l'utilizzo di hardware standardizzato.

3.1 Scelta dei componenti

Come cuore pulsante del progetto ho scelto di utilizzare il modello 4 del già noto Raspberry Pi, equipaggiato con un processore quad-core ARM-Cortex-A72, 4 GB di RAM, quattro USB, Bluetooth, Scheda di rete LAN e Wi-fi, due uscite video micro-HDMI, ed una innovativa GPU integrata con prestazioni migliorate rispetto ai precedenti modelli con supporto della risoluzione a 4K.

Data la limitata performatività del convertitore audio integrato nella scheda, ho equipaggiato il prototipo con un DAC più performante: HiFiBerry DAC+ Pro, equipaggiato con due uscite stereo parallele con convertitori fino a 192khz/24bit. Essendo l'uscita in

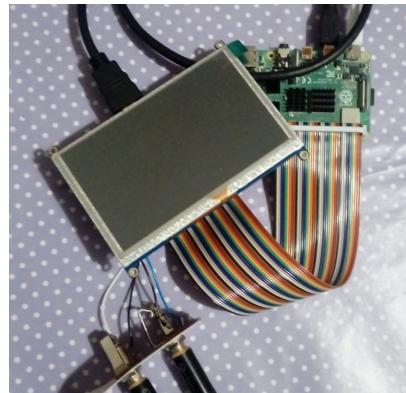
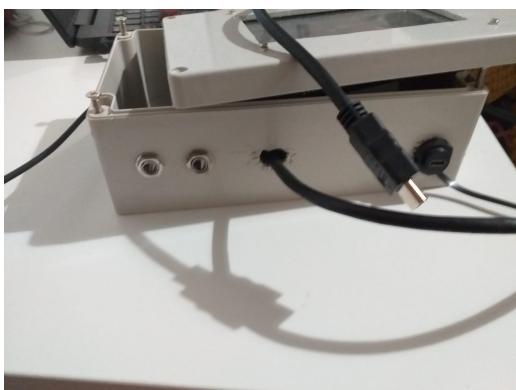


formato RCA ho aggiunto anche due Jack TS femmina per un'ulteriore output più standard. Infine ho scelto di sfruttare la doppia uscita video per utilizzare due schermi: uno dove avviene la renderizzazione della parte visuale, e l'altro con l'interfaccia grafica (UI) interattiva. Ho aggiunto dunque al prototipo uno schermo da cinque pollici con touch-screen capacitivo con una risoluzione di 800x480 pixels.

3.2 Assemblaggio

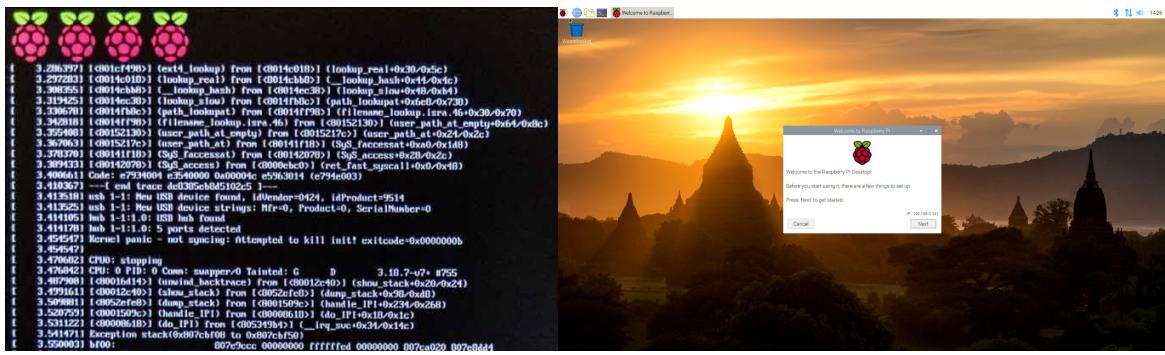
Una volta completata la lista dei componenti ho iniziato la fase di assemblaggio utilizzando dei cavi ribbon e saldando i restanti componenti.

Il prototipo inserito all'interno di una custodia in plastica presenta esposti i due Jack TS per l'uscita audio, una presa USB-C per l'alimentazione a 5 volt e un cavo HDMI da collegare ad un secondo schermo o ad un proiettore.



4. Ricerca e sviluppo del software

Una volta terminato il prototipo ho installato il sistema operativo Raspberry OS basato sulla distribuzione GNU/Linux Debian Buster. Il sistema operativo nasce con una funzione desktop pre-attivata in modo da essere utilizzato come un vero e proprio computer.



In Raspberry OS i principali linguaggi di programmazione open source per scrivere il DSP possono essere riassunti in:

- **Pure Data** ⁹ (linguaggio di programmazione visuale per multimedia sviluppato da Miller Puckette)
- **CSound** ¹⁰ (sistema di elaborazione del suono e della musica originariamente sviluppato da Barry Vercoe nel 1985 al MIT Media Lab)
- **SuperCollider** ¹¹ (una piattaforma per la sintesi audio e la composizione algoritmica, utilizzata da musicisti, artisti e ricercatori che lavorano con il suono)
- **C++** ¹² (linguaggio di programmazione general purpose sviluppato in origine da Bjarne Stroustrup nei Bell Labs nel 1983 come evoluzione del linguaggio C inserendo la programmazione object-oriented)

⁹ <https://puredata.info/>

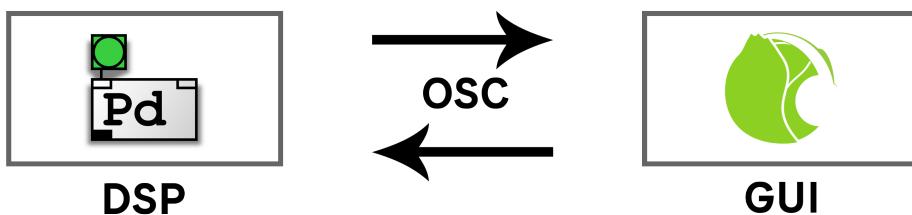
¹⁰ <https://csound.com/>

¹¹<https://supercollider.github.io/>

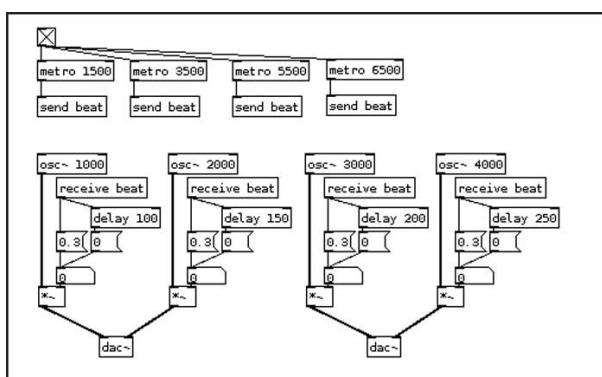
¹² <https://isocpp.org/>

4.1 Scelta del linguaggio per il DSP e per la UI

Data la mia precedente esperienza con CSound e Max/MSP, ho deciso di intraprendere lo studio di PureData (molto simile a Max/MSP nella sua caratteristica di linguaggio visuale), per la generazione ed elaborazione del suono, e Cabbage come IDE per CSound per la creazione dell'interfaccia grafica (GUI). Per far comunicare il DSP con la GUI utilizzerò il protocollo OSC.



Pure Data



Come già premesso, PD consente di creare software graficamente senza scrivere righe di codice. PD può essere utilizzato per elaborare e generare suoni, video, grafica 2D / 3D, utilizzare dati di sensori, dispositivi di input, MIDI e può

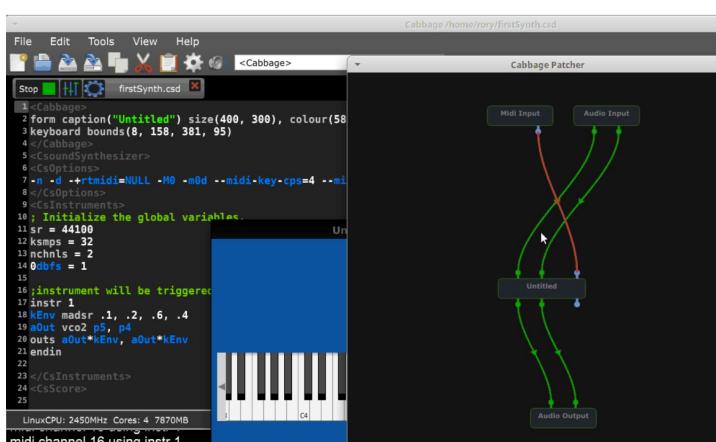
lavorare agilmente anche su reti locali e remote. Ideato da Miller Puckette (che ha partecipato alla creazione anche di Max/MSP) con tre fondamenti cardine:

- la possibilità di integrare dal vivo, in un unico ambiente, l'elaborazione dei segnali audio con il video e la grafica in genere
- la possibilità di poter controllare in tempo reale i parametri musicali attraverso la semplice gestualità

- avere l'opportunità di comporre musica realizzando delle partiture grafiche, in maniera simile al modello del sistema Upic di Xenakis

Puckette scrive che PD sta anche per Public Domain¹³, a sottolineare la natura aperta e libera di questo ambiente di lavoro, ulteriore differenza con Max/MSP. Il software gode ancora di ottima salute, sostenuto da una larga comunità di riferimento che aggiorna ed espande le funzionalità continuamente con la creazione di librerie di externals e abstractions.

Cabbage (Csound)¹⁴



Cabbage è un framework per lo sviluppo di software audio. Usando un semplice editor per generare la parte grafica e il linguaggio di sintesi audio Csound, gli utenti possono creare applicazioni o Plugins per

Windows, OSX, Linux e Android con il medesimo codice sorgente. Nel mio caso, ispirandomi alla struttura dell'interfaccia di PureData e SuperCollider - dove il core audio e l'interfaccia sono collegati tramite connessioni TCP - ho pensato di usare le possibilità grafiche di Cabbage unite alle funzionalità di networking di Csound e PureData mediante connessione UDP, per creare un'interfaccia grafica che dialogasse con il core audio mediante il protocollo OSC.

¹³ Miller Puckette, Pure Data: Another Integrated Computer Music Environment, Proceedings of the Second Intercollege Computer Music Concerts, Tachikawa 1997

¹⁴ <https://cabbageaudio.com/>

Open Sound Control (OSC)

Nel 1997 Adrian Freed e Matt Wright al CNMAT dell'Università di Berkley in California hanno introdotto il protocollo Open Sound Control come:

“un nuovo protocollo per la comunicazione tra computer, sintetizzatori di suoni e altri dispositivi multimediali ottimizzato per la moderna tecnologia di rete. Le entità all'interno di un sistema vengono indirizzate individualmente da uno schema di denominazione simbolico in stile URL aperto che include un potente linguaggio di corrispondenza dei modelli per specificare più destinatari di un singolo messaggio. Forniamo tag temporali ad alta risoluzione e un meccanismo per specificare gruppi di messaggi i cui effetti devono verificarsi simultaneamente”¹⁵

Di fatto tramite OSC è possibile far comunicare in tempo reale informazioni di controllo attraverso la rete (sia locale che remota), dando la possibilità di far dialogare, in maniera flessibile e accurata, più parametri funzionali all'esecuzione e alla programmazione del suono.

4.2 Architettura software per l'audio

Una volta scelto il linguaggio con cui programmare il DSP, mi sono interrogato su quali elementi di generazione e modulazione audio utilizzare. La volontà era quella di creare un hardware che potesse essere ritenuto una workstation a sé stante e quindi il mio scopo è stato quello di dare all'utente più strumenti possibili. Mi sono approcciato agli elementi costitutivi della generazione sonora non solo ispirandomi alle sonorità elettroacustiche ma anche a quelle del filone di musica elettronica considerata popolare, basata sulla sequenzialità del ritmo con un approccio alla sintesi di tipo sottrattivo.

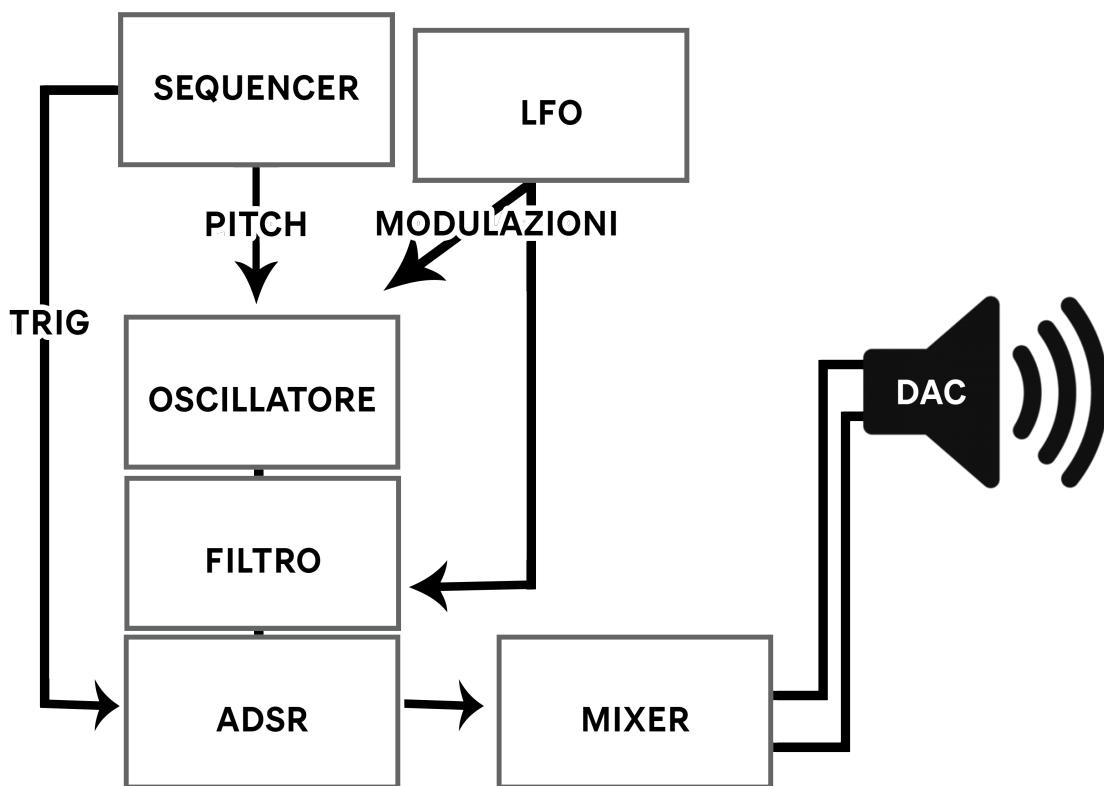
¹⁵ Wright, M., Dannenberg, R., Pope, S., Rodet, X., Serra, X. and Wessel, D. Panel: Standards from the Computer Music Community International Computer Music Conference, International Computer Music Association, Miami, FL, 2004

4.2.1 Generazione, modulazione e sequenza di eventi sonori

I componenti della catena per la generazione di suono, ritmica e modulazione scelti sono:

- Oscillatore
- Filtro
- Inviluppo (ADSR)
- Sequencer
- LFO
- Canale di mix bus

Organizzati come segue:



Oscillatore

Per quanto riguarda la scelta di un componente base come l'oscillatore ho avviato diverse ricerche per implementare sonorità interessanti utilizzando le principali forme d'onda (sinusoida, onda quadra, onda triangolare, onda a dente di sega) e tecniche di sintesi (Karplus strong,

FM, RM). Il risultato sonoro era interessante ma costoso a livello computazionale, per questo la ricerca si è focalizzata su uno o più external che potessero implementare più tecniche di sintesi e forme d'onda con consumi computazionali limitati. Il miglior compromesso tra performatività e qualità sonora è stato l'utilizzo dell'external brds~ che è basato sugli algoritmi del modulo eurorack Braids della Mutable Instruments basato sul SoC STM32 ARM Cortex e distribuito con licenza open su GitHub¹⁶. Il Braids viene definito “Un atlante delle tecniche di generazione di forme d'onda” racchiudendo all'interno di un singolo algoritmo decine di tecniche di sintesi così elencate sul sito ufficiale:

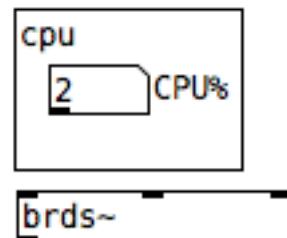
- Dente di sega stile CS-80 con filtro notch
- Onda in continuo morphing tra onda triangolare, a dente di sega, quadra, pulse, con controllo di carattere
- Onda quadra/a dente di sega con PWM (Pulse width modulation)
- Onda che fa morphing tra triangolare e sinusoide con wavefolder
- Treno d'impulsi a banda limitata con wavefolder
- Doppia onda quadra o a dente di sega con hard sync
- Tripla onda a dente di sega, quadra, triangolare o sinusoidale
- Serie di tre onde sinusoidali modulate ad anello
- Sciami di sette onde a dente di sega
- Onda a dente di sega con filtro comb
- Onda a dente di sega con downsampling e bitcrushing
- Forme d'onda filtrate in stile Casio CZ



¹⁶ <https://github.com/pichenettes/eurorack> - <https://github.com/TheTechnobear/Mi4Pd>

- Sintesi vocale o per formanti in versione lo-fi o hi-fi
- Oscillatore armonico
- Sintesi FM con vari algoritmi di feedback
- Corda pizzicata (Karplus strong)
- Corda archedgiata
- Canna e flauto
- Campana o membrana metallica
- Grancassa, piatti, e rullante della Roland 808
- Rumore elaborato da un filtro multimodale intonato
- Rumore elaborato da un doppio filtro passa-banda
- Rumore digitale con clock
- Nuvola di grani sinusoidali
- Sintesi particellare

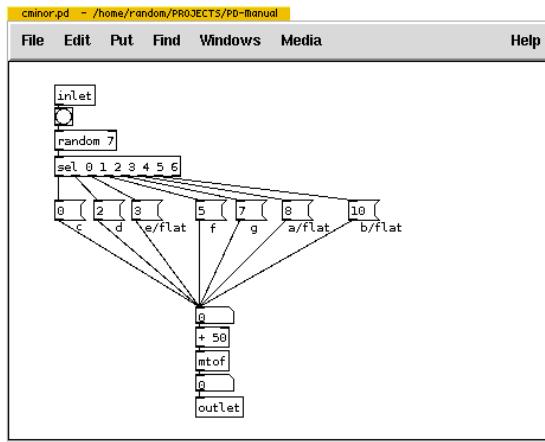
Tutto questo con un utilizzo estremamente limitato di risorse computazionali. Le possibilità date da questo external sono il primo mattone del progetto Synaesthetik.



Filtro

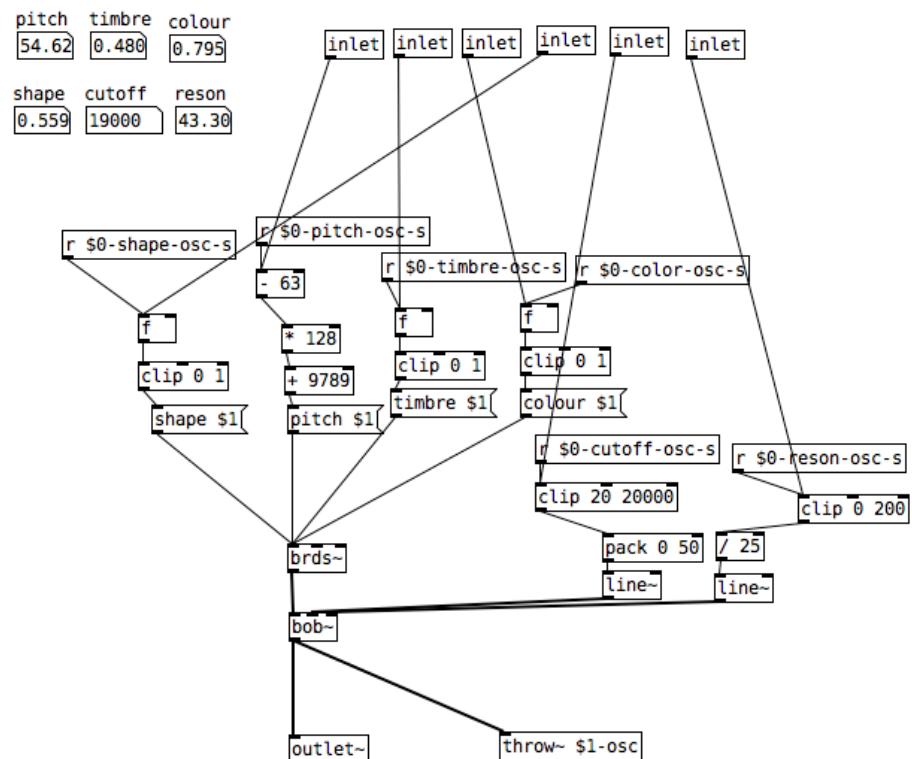
Il segnale generato dal Braids viene poi passato in un external base di Pure Data: bob~, ovvero un filtro passa-basso con risonanza di tipo ladder che imita il comportamento del filtro montato sui sintetizzatori Moog¹⁷. La combinazione tra l'oscillatore e il filtro è stata poi racchiusa in un abstraction. Le abstractions sono piccole patch, astrazioni di alcuni algoritmi che svolgono uno specifico task. Con un codice sempre più complesso, le patch possono diventare grandi e difficili da gestire, con le

¹⁷ Analysis of the Moog Transistor Ladder and Derivative Filters -Timothy E. Stinchcombe



abstractions è possibile richiamare, anche in più istanze, una singola operazione algoritmica complessa.

Nell'immagine sopra possiamo osservare una situazione in cui un algoritmo triggerato da un bang genera una nota casuale su scala di DO minore e la converte in frequenza. Inserendo inlet ed outlet questa operazione algoritmica può essere richiamata più volte. Gli externals sono molto simili concettualmente alle abstractions con la differenza che sono oggetti utilizzabili in PD ma sono scritti con sofisticati algoritmi in linguaggio C. L'abstraction del nostro oscillatore con filtro risulta dunque così:



Inviluppo

Per quanto riguarda l'utilizzo di un inviluppo per modulare la durata ed evoluzione del materiale sonoro ho utilizzato un classico calcolo lineare di tipo ADSR cioè con Attacco, Decadimento, Sostegno e Rilascio. Ricevuto un segnale di trigger viene generata la rampa di attacco, quindi una rampa di decadimento fino al punto di sostegno per poi avviarsi verso il valore 0 per la durata del tempo di rilascio, tutti i parametri sono selezionati dall'utente in millisecondi.

Sequencer

Come precedentemente descritto, l'inviluppo per essere azionato necessita di una funzione di triggering, per questo ho scelto di equipaggiare Synaesthetik con un sequencer con particolari funzionalità. Con il sequencer l'utente può arbitrariamente decidere quale e quando un'informazione di Pitch assieme al trigger può essere inviata ai Braids in un loop composto fino a un massimo di 32 step. Per ampliare le caratteristiche del sequencer ho utilizzato un algoritmo per calcolare la quantità e la distribuzione dei trigger tra gli steps denominato algoritmo Euclideo.

Ritmo euclideo

Analizzato da Godfried Toussaint nel 2005¹⁸, l'algoritmo per generare ritmi euclidei è uno degli algoritmi più antichi conosciuti, il calcolo si basa sul massimo comune divisore tra due numeri interi. L'idea è molto semplice. Il numero più piccolo viene ripetutamente sottratto dal maggiore fino a quando il maggiore è zero o diventa minore del numero

¹⁸ The Euclidean Algorithm Generates Traditional Musical Rhythms - Godfried Toussaint - School of Computer Science, McGill University - Montreal, Quebec-Canada

inferiore, in qual caso viene denominato resto. Il resto viene quindi sottratto ripetutamente dal numero più piccolo per ottenere un nuovo resto. Questo processo viene continuato fino a quando il resto è zero.

Considerando come esempio i numeri 5 e 8 . 5 si divide in 8 una sola volta con resto di 3. Quindi 3 si divide in 5 una volta con resto di 2. Quindi 2 si divide in 3 una volta con resto di 1. Infine, 2 si divide in 2 una volta con un resto di 0. Il massimo comune divisore è quindi 1. Sebbene l'algoritmo originale di Euclide usasse la sottrazione ripetuta in questo modo, la divisione intera funziona nel medesimo modo. La divisione intera (in informatica booleana: mod o %), dati due valori, divide il primo con il secondo e riconsegna in output il resto. Considerando m e k gli integer in input con $m > k$:

```
EUCLID(m, k)
if k = 0
then return m
else return EUCLID(k, m mod k)
```

Eseguendo questo algoritmo con $m = 8$ e $k = 5$ otteniamo: $\text{EUCLID}(8,5) = \text{EUCLID}(5,3) = \text{EUCLID}(3,2) = \text{EUCLID}(2,1) = \text{EUCLID}(1,0) = 1$

È chiaro dalla descrizione dell'algoritmo euclideo che se m e k sono uguali al numero di zero e uno, rispettivamente, in una sequenza binaria (con $n = m + k$) allora la struttura dell'algoritmo, può generare un gran numero di ritmi

Se consideriamo 8 step come una stringa di 8 bit e di questi 2 hanno valore 1 rispetto agli altri, possiamo ipotizzare una distribuzione lineare in questo modo:

11000000

Ora ipotizziamo di distribuire gli 1 in intervalli più uniformi possibile spostando gli zeri.

11 000000 → 1010 0000 → 100100 00

Una volta distribuiti gli zeri in maniera uniforme avremo dunque una sequenza come segue:

10001000

Se consideriamo i valori 1 come trigger di eventi in un sequencer possiamo visualizzare la sequenza anche così:

$E(2,8)$

X | | | X | | |

Vediamo un ulteriore caso dove la distribuzione uniforme degli intervalli (0) può essere più complicata per esempio calcolando $E(5,8)$, ovvero su 8 steps, 5 devono inviare un trigger.

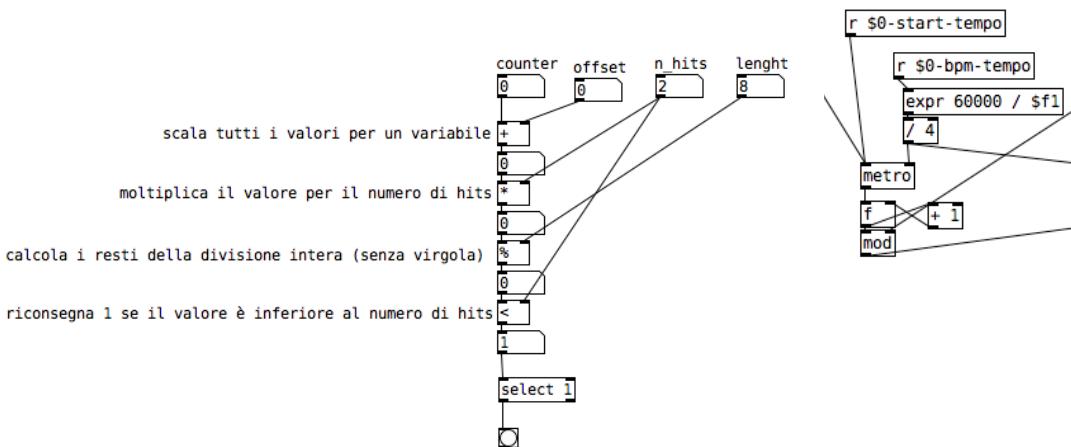
11111000 → 11111 000 → 101010 11 → 101010 11

Come si può notare vengono distribuiti uniformemente anche gli ultimi 1 di scarto generando la sequenza seguente:

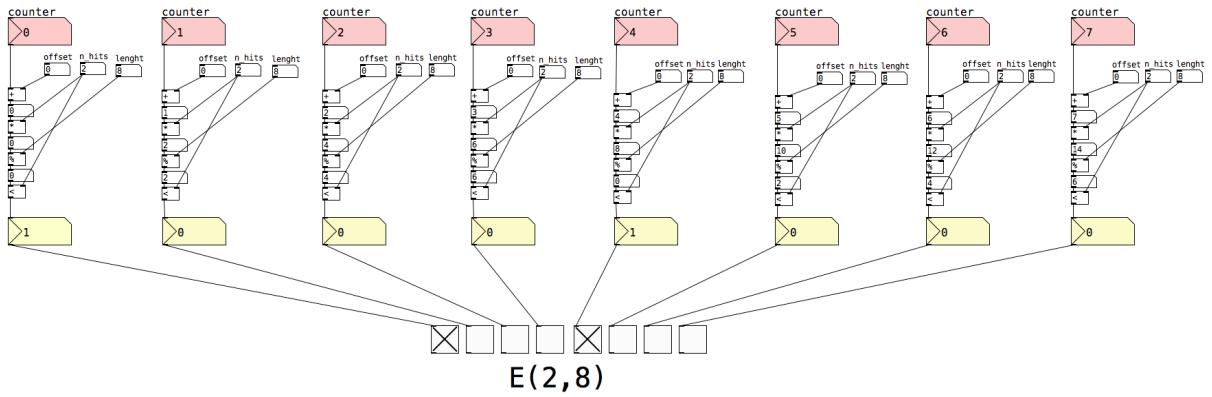
10110110
X | X X | X X |

In particolare questa sequenza si può identificare con il ritmo cubano chiamato Cinquillo. Se il calcolo fosse stato E(3,8) sarebbe stato l'equivalente del Tresillo sempre di tradizione cubana, a conferma che questo algoritmo per generare sequenze ritmiche, permette di comporre agilmente ritmi tradizionali della world music.

Ogni sequenza può essere anche traslata di un numero deciso dall'utente (offset) dando vita a innumerevoli combinazioni poliritmiche. L'implementazione con PureData è stata scritta seguendo le regole matematiche che stanno alla base del calcolo euclideo utilizzando la logica booleana.



Nell'immagine a sinistra possiamo vedere l'implementazione con operatori logici booleani. In particolare al primo valore (counter) viene collegato un contatore temporizzato che riconsegna un valore integer per ogni pulsazione (calcolata in pulsazioni al minuto come nell'immagine a destra). Per esempio se la lunghezza della sequenza è 8 avremo valori da 0 a 7 in loop a uno specifico intervallo di tempo. A questo valore viene sommato un eventuale valore di offset, viene dunque moltiplicato per la quantità di trigger per loop. L'operatore % compie una divisione intera tra il numero che arriva in inlet e la lunghezza totale della sequenza e manda in output il resto che deve, infine, essere minore al numero di trigger per loop. Se l'ultima condizione è vera verrà mandato 1 come output.



Calcolo del pitch

Nel paragrafo precedente ho illustrato il metodo di generazione dei trigger, ora vediamo come avviene il calcolo del valore di pitch da inviare al macro-oscillatore. Dato un array di un massimo di 32 valori (steps), in ogni casella viene inserita un'informazione di frequenza che identifica il pitch dello step richiamato. L'intervallo tra le frequenze può essere deciso arbitrariamente dall'utente - ad esempio 1°step(C4) - 2°step(A5) - oppure può essere calcolato utilizzando la formula del temperamento equabile.

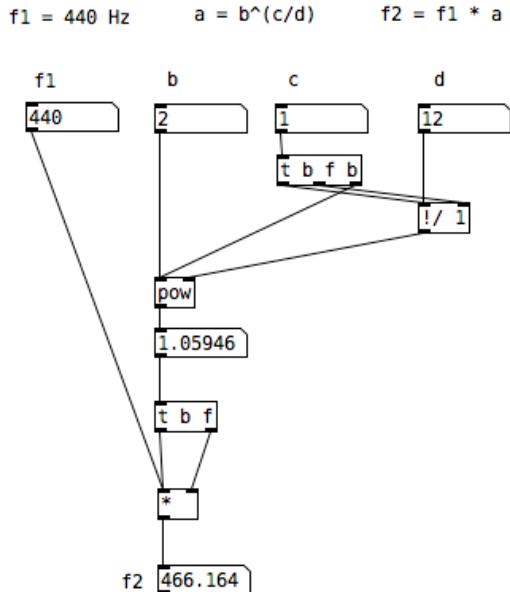
$$f1 = 440Hz = LA4 \quad f2 = f1 * a \quad a = 2^{\frac{1}{12}} = 1,05946$$

$$f2 = 466,164Hz = LA\#4$$

Nel caso esposto sopra abbiamo applicato la formula del temperamento equabile scomponendo l'intervallo ($f1 * 2$) in 12 semitonni, seguendo quindi la scala cromatica (12-TET). L'equazione può essere generalizzata in questo modo:

$$f2 = f1 * a \quad a = b^{\frac{c}{d}}$$

Grazie all'implementazione della formula generalizzata l'utente può esplorare diversi tipi di temperamenti: da quelli storicamente occidentali a quelli dell'estremo oriente o xenarmonici in genere.

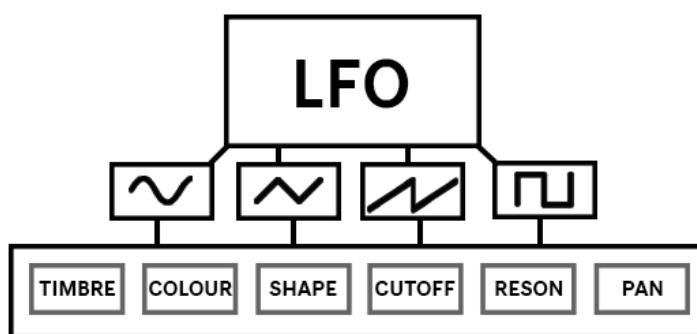


Name	Exact value in 12-TET	Decimal value in 12-TET	Cents
Unison (C)	$2^{\frac{0}{12}} = 1$	1	0
Minor second (C♯/D♭)	$2^{\frac{1}{12}} = \sqrt[12]{2}$	1.059463	100
Major second (D)	$2^{\frac{2}{12}} = \sqrt[6]{2}$	1.122462	200
Minor third (D♯/E♭)	$2^{\frac{3}{12}} = \sqrt[4]{2}$	1.189207	300
Major third (E)	$2^{\frac{4}{12}} = \sqrt[3]{2}$	1.259921	400
Perfect fourth (F)	$2^{\frac{5}{12}} = \sqrt[12]{32}$	1.334840	500
Tritone (F♯/G♭)	$2^{\frac{6}{12}} = \sqrt[2]{2}$	1.414214	600
Perfect fifth (G)	$2^{\frac{7}{12}} = \sqrt[12]{128}$	1.498307	700
Minor sixth (G♯/A♭)	$2^{\frac{8}{12}} = \sqrt[3]{4}$	1.587401	800
Major sixth (A)	$2^{\frac{9}{12}} = \sqrt[4]{8}$	1.681793	900
Minor seventh (A♯/B♭)	$2^{\frac{10}{12}} = \sqrt[12]{32}$	1.781797	1000
Major seventh (B)	$2^{\frac{11}{12}} = \sqrt[12]{2048}$	1.887749	1100
Octave (C)	$2^{\frac{12}{12}} = 2$	2	1200

Le scale xenarmoniche sono quelle che utilizzano un sistema di intervallo tonale che non si conforma al comune temperamento equabile a 12 toni. Il termine xenarmonico fu coniato da Ivor Darreg¹⁹, da *xenia* (greco ξενία), "ospitale" e *xenos* (greco ξένος) "straniero", questo implica una maggior inclusività dei temperamenti tradizionali di ogni parte del mondo. L'utente può decidere, tramite questo algoritmo, le distanze frequenziali di ogni nota suonata per step, oppure può scegliere di utilizzare il sequencer in maniera arbitraria selezionando personalmente quali steps suonare e quali valori di pitch inviare.

LFO

Per dare all'utente alcuni elementi per modulare i parametri, ho inserito all'interno del motore di generazione sonora e ritmica, un LFO con quattro differenti forme d'onda inviabili a sei destinazioni differenti.

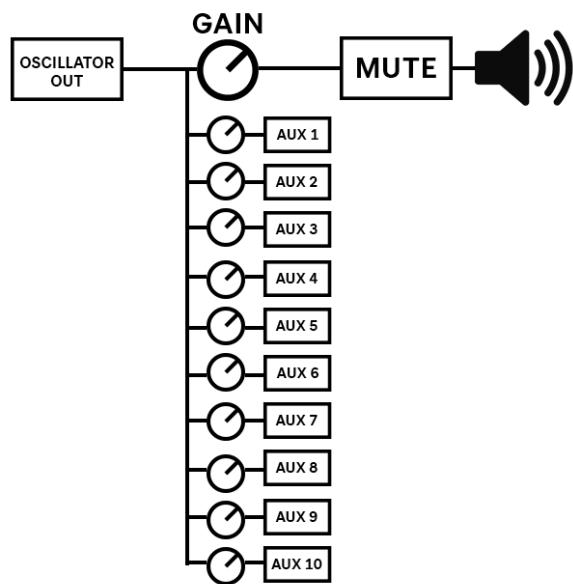


¹⁹ <http://www.afn.org/~sejic/ivor.html>

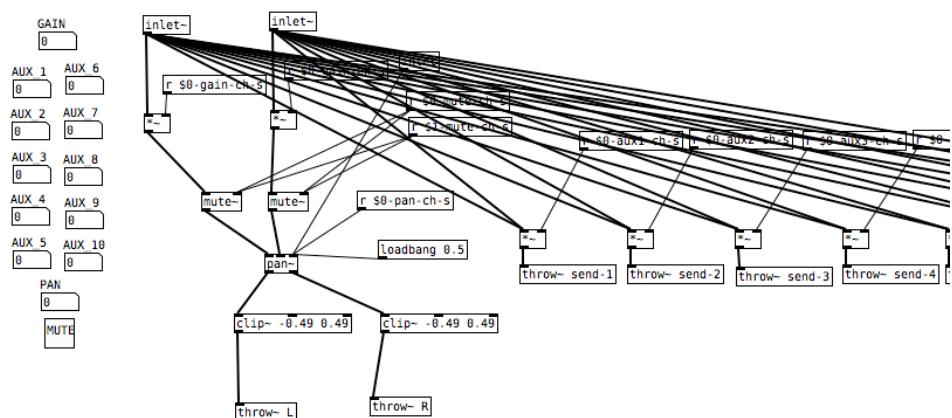
Le quattro forme d'onda - sinusoide, triangolo, a dente di sega e onda quadra - oscillano alla medesima frequenza, l'utente può selezionare il range di oscillazione e può modulare sei parametri differenti dell'oscillatore: timbre, colour, shape, cutoff, risonanza e pan.

Canale di mixbus

Una volta completata la catena di generazione sonora e ritmica ho organizzato, ispirandomi ai mixer analogici, una channel strip per mandare il suono in uscita compresa di bus per le mandate.



Come si può notare dal diagramma a blocchi le uscite ausiliari sono indipendenti dalla funzione di muting del canale e non sono affette dal Gain principale, ogni mandata ha il suo Gain indipendente dagli altri.



L'oggetto cardine dell'implementazione di questa abstraction è throw~ incorporato in PD. throw~ permette di inviare un segnale audio “wireless” sommandolo ad un altro imitando proprio un mixbus. Il segnale viene dunque “splittato” in altri dieci canali audio.

Effetti

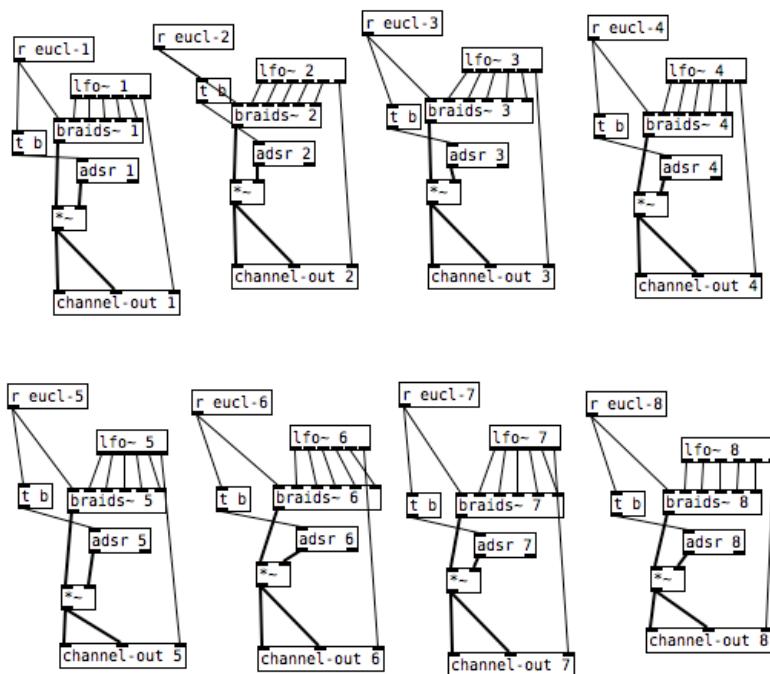
I dieci mixbus elencati poc'anzi possono inviare il segnale di una singola Channel strip ad altrettanti dieci effetti di modulazione sonora in contemporanea. Ogni modulo ha una sua channel strip che può inviare l'audio all'uscita del DAC oppure può a sua volta re-inviare il segnale tramite bus ad altri effetti in modo da creare una catena mediante l'utilizzo dei mute. Gli effetti inseriti dentro Synaesthetik fanno tutti parte di librerie esterne con licenze gratuite, in parte della collezione di oggetti della Mutable Instruments, in parte scritti da Alexandre Torres Porres (ELSE)²⁰ e sono:

- Hall reverb (Riverbero del modulo Rings della Mutable Instruments)
- Plate reverb (ELSE)
- Delay con feedback (ELSE)
- Ping Pong Delay (ELSE)
- Chorus (ELSE)
- Phaser (ELSE)
- Flanger (ELSE)
- Bitcrusher (ELSE)
- Downsampling (ELSE)
- Granulatore (Clone del modulo Clouds della Mutable Instruments)

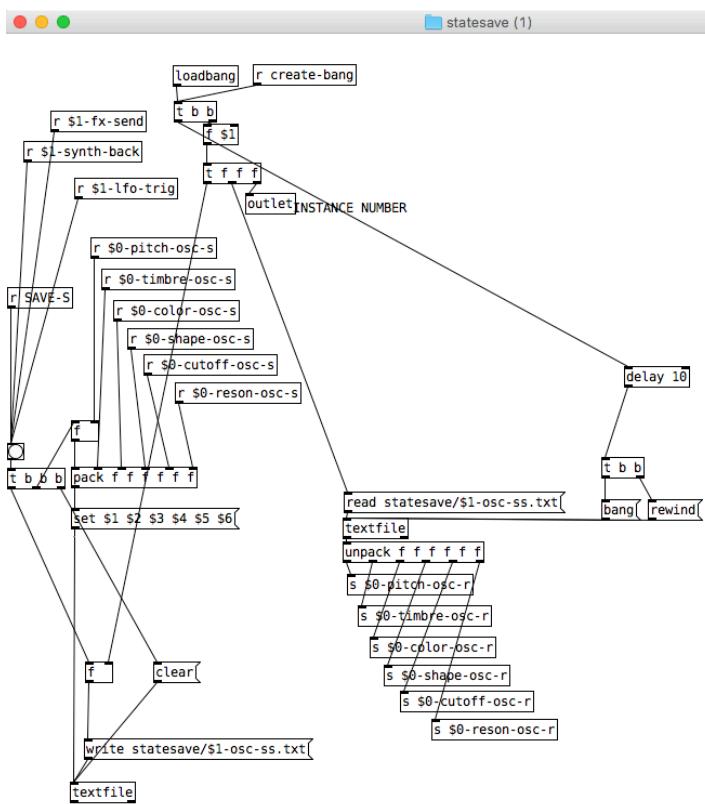
²⁰ ELSE - EL Locus Solus' Externals - <https://github.com/porres/pd-else>

4.2.2 Logica di funzionamento

L'incorporazione dei componenti appena elencati in un unico blocco può essere considerata come una componente a sé stante che chiameremo traccia. In Synaesthetik è possibile comporre con otto tracce differenti comprese di tutti gli elementi di generazione, modulazione e sequenza del materiale sonoro, il tutto racchiuso in una manciata di abstractions.

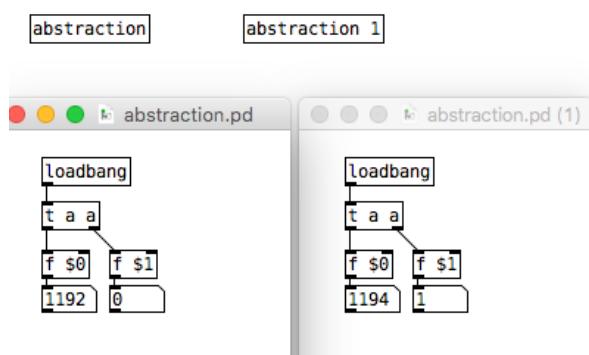


Ogni abstraction ha al suo interno un meccanismo di salvataggio di tutti i parametri in modo che l'utente possa ritrovare il proprio lavoro compositivo una volta riaccesso l'hardware. Il meccanismo funziona tramite specifiche variabili che contrassegnano una posizione in cui successivamente si inserirà un valore. In PD questo viene svolto utilizzando il simbolo del dollaro (\$). Si potrebbero anche chiamare "variabili sostitutive": cioè che segnano il punto in cui si desidera sostituirle con un valore effettivo. \$0 è una variabile sostituita internamente da un numero univoco di quattro cifre per istanza di abstraction. In altre parole, PD si prende cura che a ogni istanza di abstraction o di una patch venga assegnato questo numero univoco nella variabile \$0. L'utilità di questa variabile è immediatamente evidente nel seguente esempio:



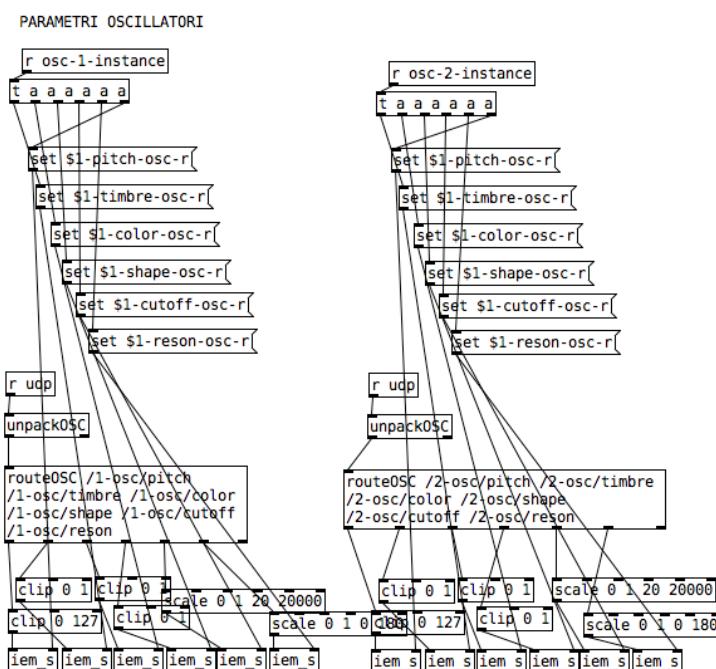
Questa patch illustra il meccanismo di salvataggio dei parametri del nostro macro-oscillatore. Da una parte i receive (`r $0-pitch-osc-s`), che ricevono le informazioni dai comandi principali della patch, di cui i valori vengono poi scritti all'interno di un file di testo (`write statesave/$1-osc-ss.txt`) e dall'altra i send (`s $0-pitch-osc-r`) i cui valori vengono letti dal medesimo file di testo (`read statesave/$1-osc-ss.txt`).

In questa patch vengono usate due tipologie di variabili sostitutive: `$0` e `$1`. Nel caso di `$0`, come abbiamo visto, si salvano i parametri dell'istanza singola dell'abstraction, replicando più volte la medesima patch ognuna invierà e riceverà parametri senza interferire con le altre. `$1` invece viene utilizzata per assegnare un indice di catalogazione per identificare quale numero di traccia stiamo trattando. Questo numero viene determinato in fase di dichiarazione dell'abstraction, di seguito si può notare come avviene l'assegnazione di `$0` e `$1`.



4.2.3 Interfaccia per l'utente

Come specificato nel paragrafo 4.1, per l'interfaccia utente l'intenzione iniziale del progetto era quella di utilizzare Cabbage, un potente software per la realizzazione di applicazioni musicali basato su CSound. Una volta completato il motore audio ho dunque implementato il protocollo OSC e categorizzato ogni parametro per la comunicazione con la GUI.



Ho preso come esempio la ricezione dei parametri dalla GUI instradati verso gli oscillatori. In questo caso vediamo che ogni parametro $\$0$ viene ricevuto da `r_osc-1-instance`, posto all'inizio del nome del parametro (`$1-pitch-osc-r`) e settato come destinazione d'invio dell'oggetto `iem_s` (che si differenzia dal classico `send` grazie al fatto che può cambiare destinazione

agilmente con il prefisso `set`). I dati OSC vengono ricevuti dall'oggetto `udpreceive` che fa parte della libreria di networking “mrpeach”²¹. I pacchetti UDP vengono interpretati, poi, da `unpackOSC` e instradati ai rispettivi `send` tramite `routeOSC`. Nel protocollo OSC i dati vengono inviati attraverso dei “canali” denominati in maniera affine agli URL dei siti web, infatti, come si può notare, tutti i parametri passano per il macro-canale `1-osc` per poi instradarsi verso il preciso parametro di destinazione (pitch, timbre, ecc.). Il protocollo è organizzato in due vie di comunicazione differenti attraverso la rete locale: mediante la porta 9000 (GUI → AudioCore) e la porta 8000 (AudioCore → GUI). L'implementazione in Cabbage è stata fatta mediante l'utilizzo di opcode per la trasmissione di pacchetti OSC su Csound: `OSCListen` e `OSCSend`. Con gli opcode `chnset` e `chnget`, che servono per far comunicare il codice CSound con

²¹ <https://puredata.info/downloads/mrpeach>

```

opcode synth,k,iSS
ihandle, Sinstance, Schan xin
kdummy init 0

#define IPADDRESS # "localhost" #
#define PORT      # 9000 # ; PORTA D'INVIO OSC

; RICEZIONE DATI OSC DA PURE DATA
kshape OSClisten ihandle, sprintf("%s-osc/shape", Sinstance), "f", kshaper
ktimbre OSClisten ihandle, sprintf("%s-osc/timbre", Sinstance), "f", ktimbrer
kcolor OSClisten ihandle, sprintf("%s-osc/color", Sinstance), "f", kcolorr

;INVIO DEI VALORI RICEVUTI ALLA GUI
if changed:k(kshape)==1 then
chnset kshaper, sprintf("%sSHAPE", Schan)
endif

if changed:k(ktimbre)==1 then
chnset ktimbrer, sprintf("%sTIMBRE", Schan)
endif

if changed:k(kcolor)==1 then
chnset kcolorr, sprintf("%sCOLOR", Schan)
endif

;RICEZIONE DEI VALORI DELLA GUI
kshapes chnget sprintf("%sSHAPE", Schan)
ktimbres chnget sprintf("%sTIMBRE", Schan)
kcolors chnget sprintf("%sCOLOR", Schan)

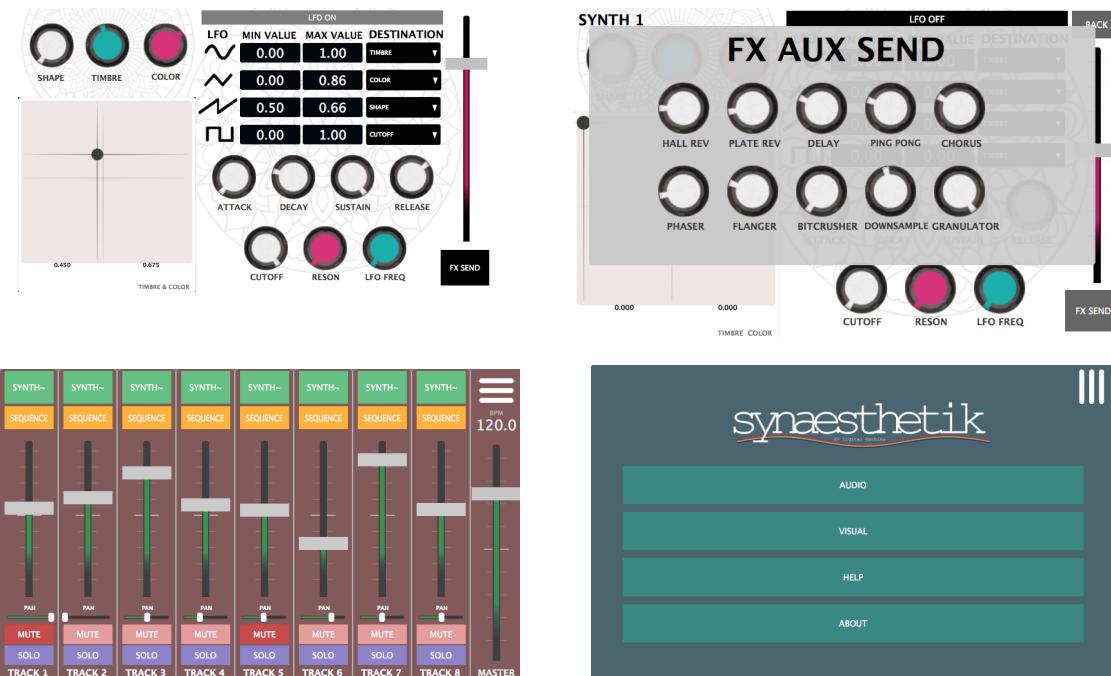
;INVIO DEI VALORI DELLA GUI VIA OSC A PURE DATA
if changed:k(kshapes)==1 then
OSCsend kshapes, $IPADDRESS, $PORT, sprintf("%s-osc/shape", Sinstance), "f", kshapes
endif

if changed:k(ktimbres)==1 then
OSCsend ktimbres, $IPADDRESS, $PORT, sprintf("%s-osc/timbre", Sinstance), "f", ktimbres
endif

if changed:k(kcolors)==1 then
OSCsend kcolors, $IPADDRESS, $PORT, sprintf("%s-osc/color", Sinstance), "f", kcolors
endif

```

l’interfaccia grafica, il protocollo di comunicazione a due vie può ritenersi completo. Come risultato abbiamo un’agile comunicazione tra i due software e la possibilità di creare un interfaccia quasi scheumorfica.

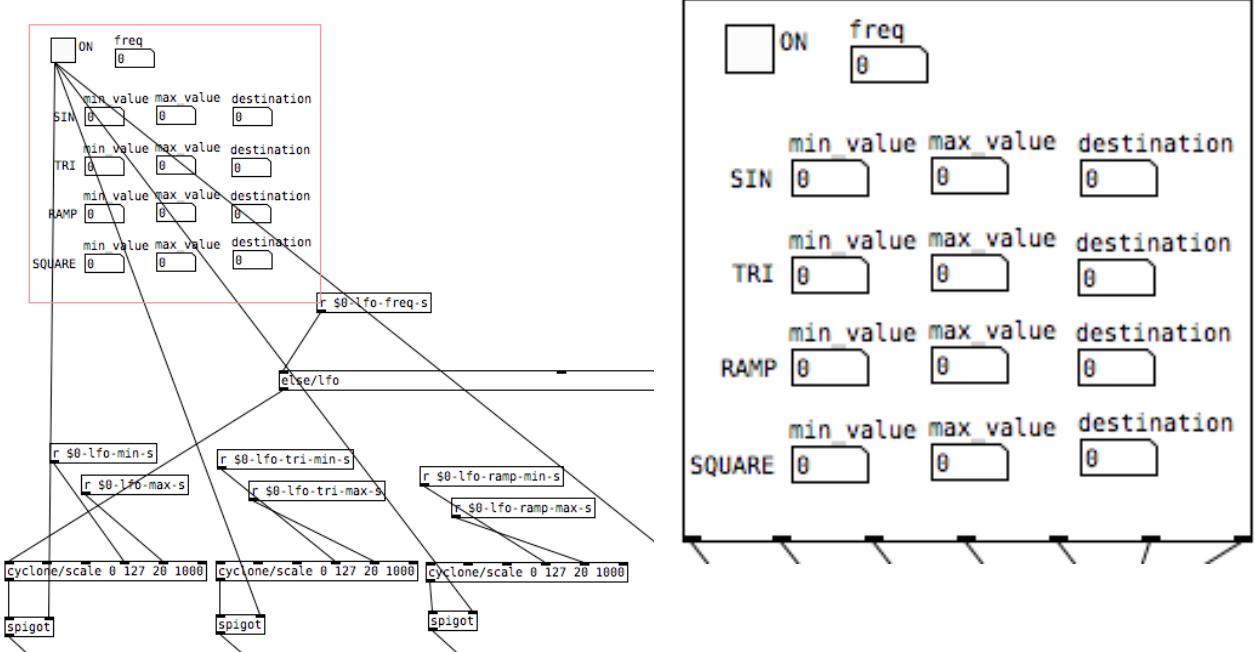


Questo sistema di comunicazione, testato con una sola traccia, è stato in grado di reggere performativamente bene all'interno del prototipo di Synaesthetik. Il risultato presentava latenze minime (< 40 ms) e senza errori di campionamento o quantizzazione dovuti allo sforzo del processore. I primi problemi sono sorti nel momento di replica del protocollo e degli elementi grafici per tutti i parametri delle otto tracce.

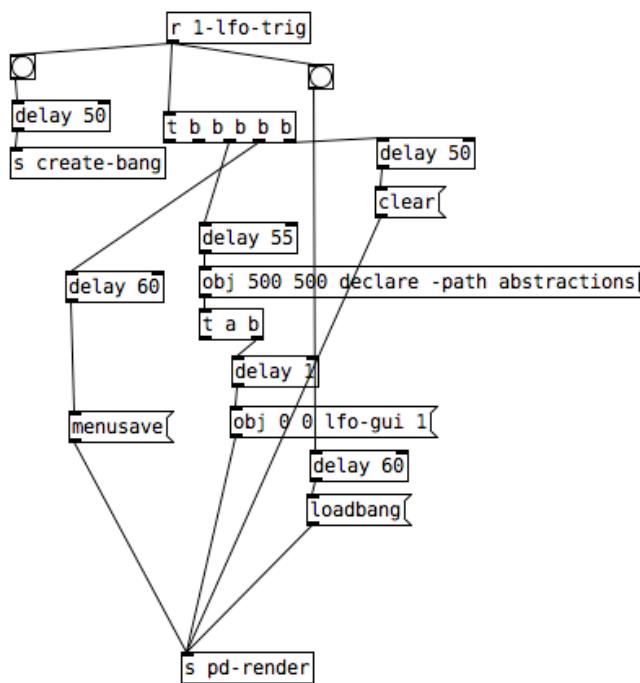
Cabbage e Csound compilano il codice nel momento di avvio dell'applicativo, questo ha evidenziato una serie di problemi oggettivi nella stesura del codice dove gli elementi grafici (più di 300) e di networking (più di 500) venivano istanziati tutti nello stesso momento causando il crash dell'applicativo per il troppo utilizzo di risorse. Per qualche settimana ho cercato di svolgere un debug inserendo tempi di Delay per permettere alle istanze di caricarsi in maniera dilazionata nel tempo, ma un altro problema è poi emerso. Per poter utilizzare Cabbage nei sistemi Linux l'utente deve scaricare il codice sorgente del software con le sue dipendenze e costruire l'applicativo mediante gli strumenti del terminale. Per via delle licenze commerciali presenti all'interno di alcune dipendenze di Cabbage il procedimento è risultato più difficile del previsto. Cabbage è un software libero e open source, ma secondo le leggi del copyright non può rilasciare il suo codice sorgente intero perché significherebbe integrare parti di proprietà intellettuale altrui, ciò ha recato oggettivamente un danno a questa ricerca e mi ha costretto ad abbandonare l'utilizzo di questo software per il progetto Synaesthetik.

L'alternativa

Arrivati a questo punto dello sviluppo, dove il motore audio era ultimato, ho dovuto ripiegare su un'idea di interfaccia più limitata graficamente ma performativamente integrabile nel sistema utilizzando PureData stesso. PD ha una classe particolare di messaggi per svolgere azioni di scripting del linguaggio, in altre parole permette di creare, eliminare, collegare o scollegare oggetti inviando messaggi all'istanza della patch stessa. Le abstractions, oltre alle caratteristiche elencate in precedenza, hanno anche una peculiarità interessante: possono essere integrate in una patch di livello macro visualizzandosi come bi-patcher.



Sfruttando queste capacità di scripting e di bi-patching ho ricostruito una GUI per ogni elemento delle otto tracce con risultati performanti anche con l'aggiunta della parte di visual. Il fatto di dividere l'architettura software in due gruppi (Audio-Core e GUI) ha permesso inoltre di non caricare tutte le risorse grafiche all'avvio del software ma man mano che l'utente clicca sulle finestre desiderate.



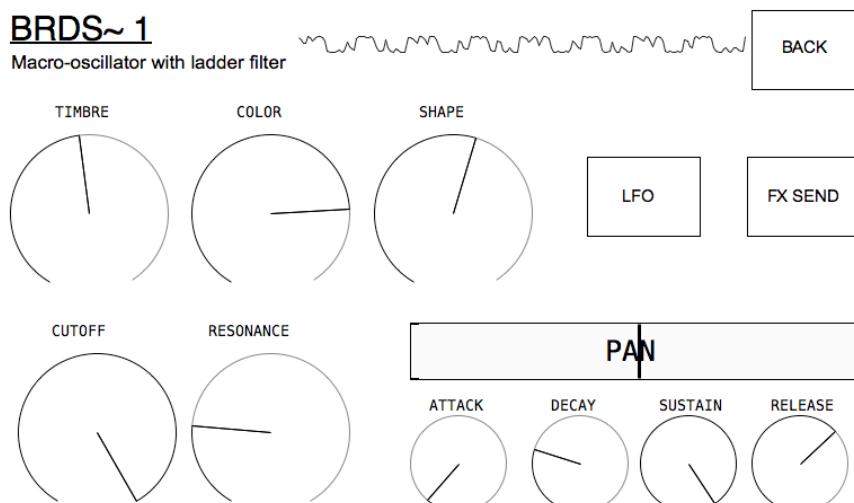
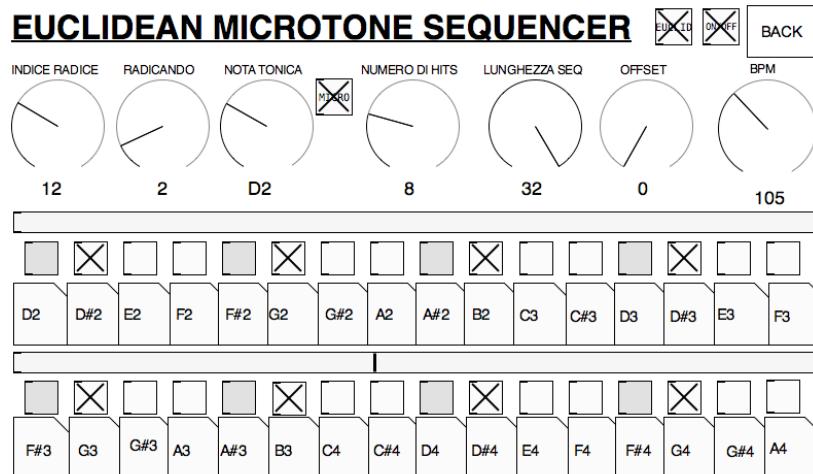
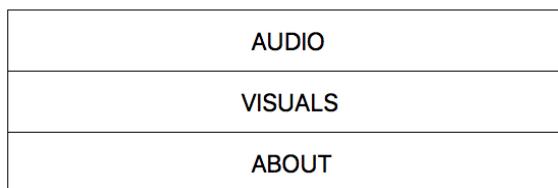
nell'esempio a fianco vediamo lo script per visualizzare la GUI per l'LFO del primo oscillatore. Si può notare il messaggio `obj` che è colui che permette di generare un oggetto all'interno della finestra di riferimento. In questo caso il software apre una finestra apposita denominata `Render` dove vengono creati ed eliminati gli elementi grafici dell'interfaccia. Con il messaggio `clear` si eliminano tutti gli elementi della

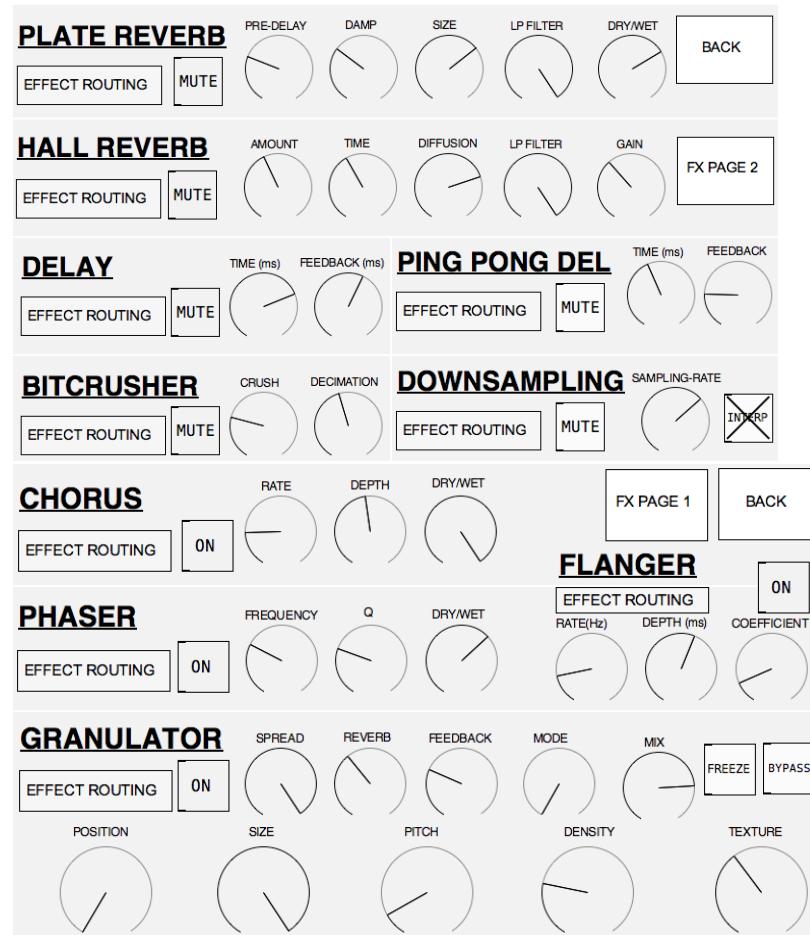
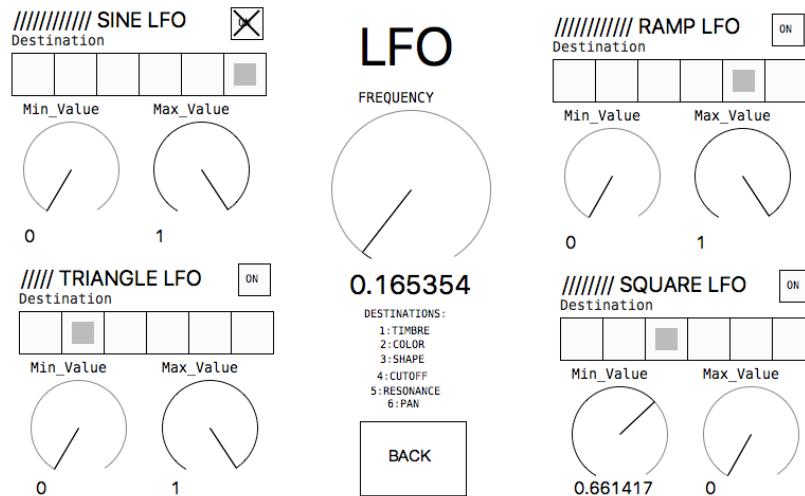
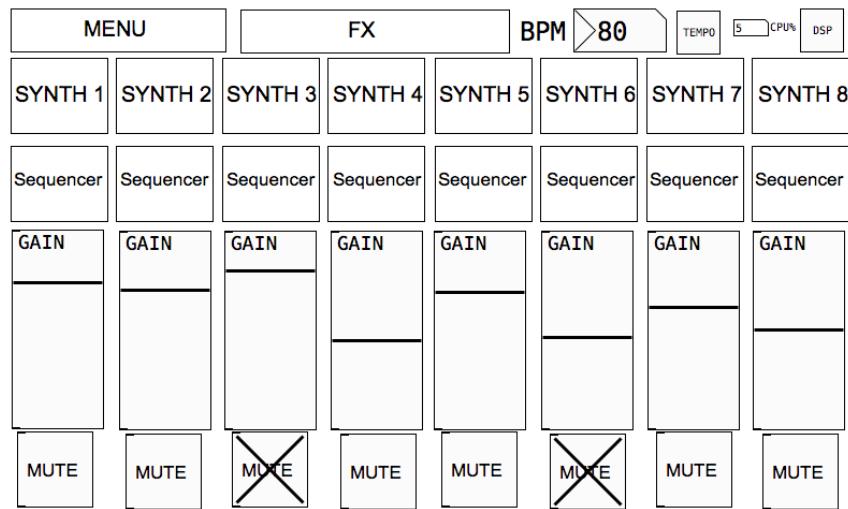
Nell'esempio a fianco vediamo lo script per visualizzare la GUI per l'LFO del primo oscillatore. Si può notare il messaggio `obj` che è colui che permette di generare un oggetto all'interno della finestra di riferimento. In questo caso il software apre una finestra apposita denominata `Render` dove vengono creati ed eliminati gli elementi grafici dell'interfaccia. Con il messaggio `clear` si eliminano tutti gli elementi della

che serve alla patch per identificare il percorso dell'abstraction e lfo-gui che è la patch contenente gli elementi grafici. Alla ricezione del loadbang la patch legge i valori all'interno dei file di testo per i salvataggi, nel momento di cambio da una finestra ad un'altra viene inviato un messaggio (create-bang) che salva i parametri nel file di testo andando a generare un meccanismo di aggiornamento continuo tra la GUI e il motore Audio.

synaesthetik

Audiovisual embedded software





4.3 Architettura software per il video

L'implementazione della parte di generazione video in tempo reale è stata realizzata sempre all'interno dell'ecosistema di PureData. Storicamente PureData è equipaggiato con una libreria per la generazione di Visual denominata GEM, ma dal 2018 è stata pubblicata Ofelia²². Ofelia è un a libreria di external che ti consente di utilizzare OpenFrameworks attraverso LUA all'interno di PD. OpenFamewoks è un toolkit open source scritto in C ++ per arte digitale legata al video. LUA è un linguaggio di scripting potente e leggero. Grazie alla funzione di scripting di LUA, è possibile scrivere codice OpenFrameworks direttamente su una patch PD o tramite un editor di testo, a differenza dei programmi compilati in C/C++, è possibile visualizzare immediatamente il risultato mentre si modifica il codice. Grazie alle abstraction per la prototipazione con Ofelia rilasciate dagli utenti della comunità²³ è stato possibile realizzare lavori complessi partendo da geometrie basilari.

4.3.1. Ofelia e OpenGL in Pure Data

OpenFrameworks (e quindi anche Ofelia) è un linguaggio che esegue funzioni di rendering 2D e 3D basato su OpenGL. OpenGL (Open Graphics Library) è una potente libreria grafica che si pone il compito di interfacciarsi con l'hardware e fornire al programma una serie di primitive, più o meno essenziali, per lo sviluppo di applicazioni nel campo del rendering tridimensionale. Queste primitive comprendono funzioni per la manipolazione dei pixel, per la proiezione di poligoni in 3D e per la gestione del movimento degli stessi, per la rappresentazione di luci e

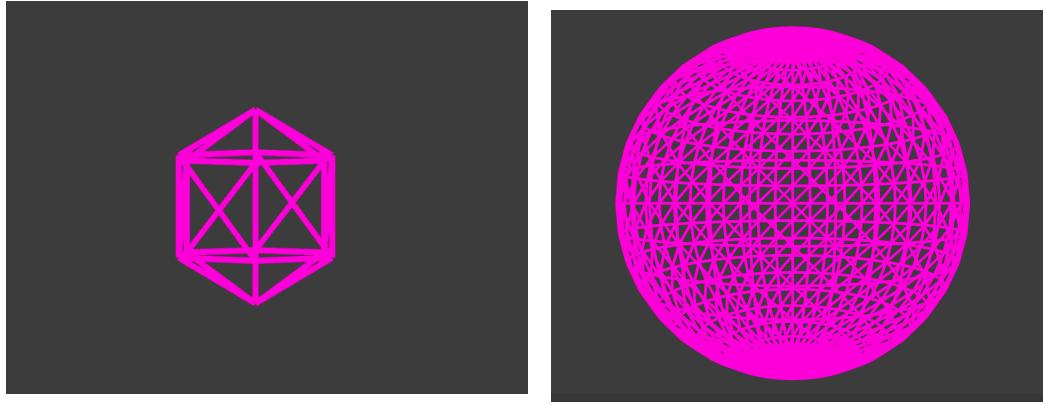
²² <https://github.com/cuinjune/Ofelia>

²³ <https://github.com/60-hz/Ofelia-Fast-Prototyping>

colori, per il texture mapping etc. Il prefisso Open sta ad indicare che OpenGL è una architettura aperta. Una delle peculiarità di OpenGL che la distingue immediatamente dalle sue rivali (ad esempio DirectX di Microsoft) è la portabilità. Infatti è disponibile su tutte le piattaforme più diffuse. Questa estrema portabilità ha fatto sì che anche i costruttori di hardware si adeguassero ad essa e ora un gran numero di schede video moderne hanno la possibilità di eseguire la maggior parte delle funzioni OpenGL direttamente in hardware con un netto vantaggio in termini di prestazioni anche in ambiente ARM.



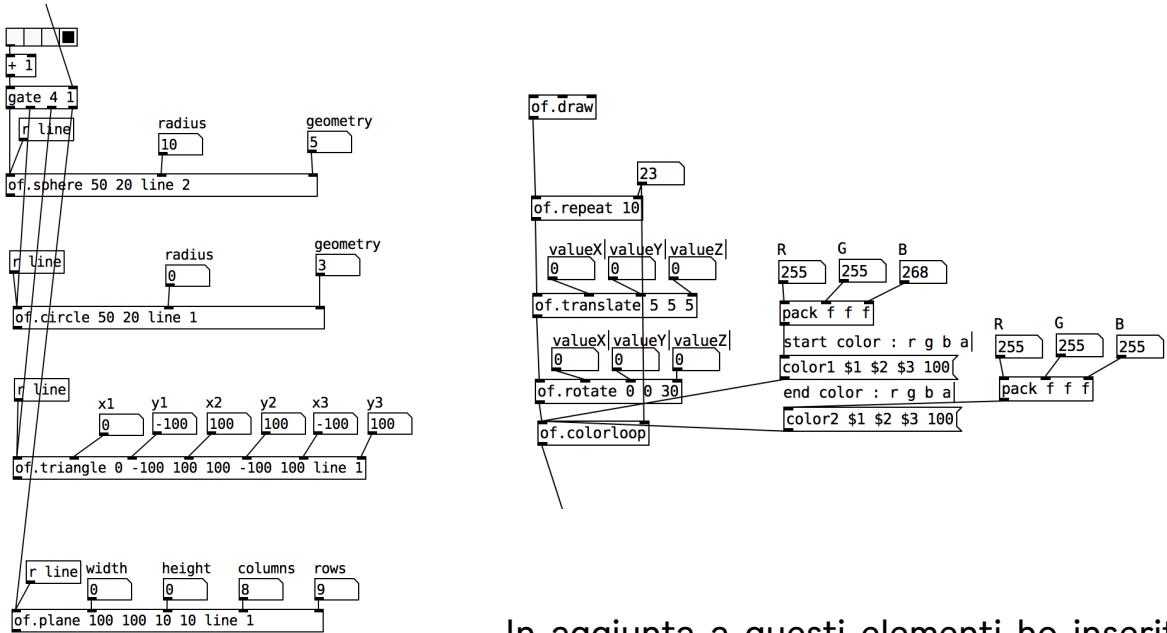
Per prima cosa, dunque, con le abstraction di Ofelia ho generato una finestra dove la GPU renderizzerà i nostri pixel tramite l'oggetto `of.window`. Una volta creata la finestra, per il rendering necessitiamo dell'oggetto `of.draw` alla cui outlet vanno collegati in serie gli elementi per il render. Come abbiamo anticipato, è possibile fare rendering 3D partendo da geometrie di base, in questo caso la sfera viene generata tramite l'utilizzo di poligoni affiancati l'un l'altro. Di seguito un esempio di sfera composta da 6 e 100 poligoni:



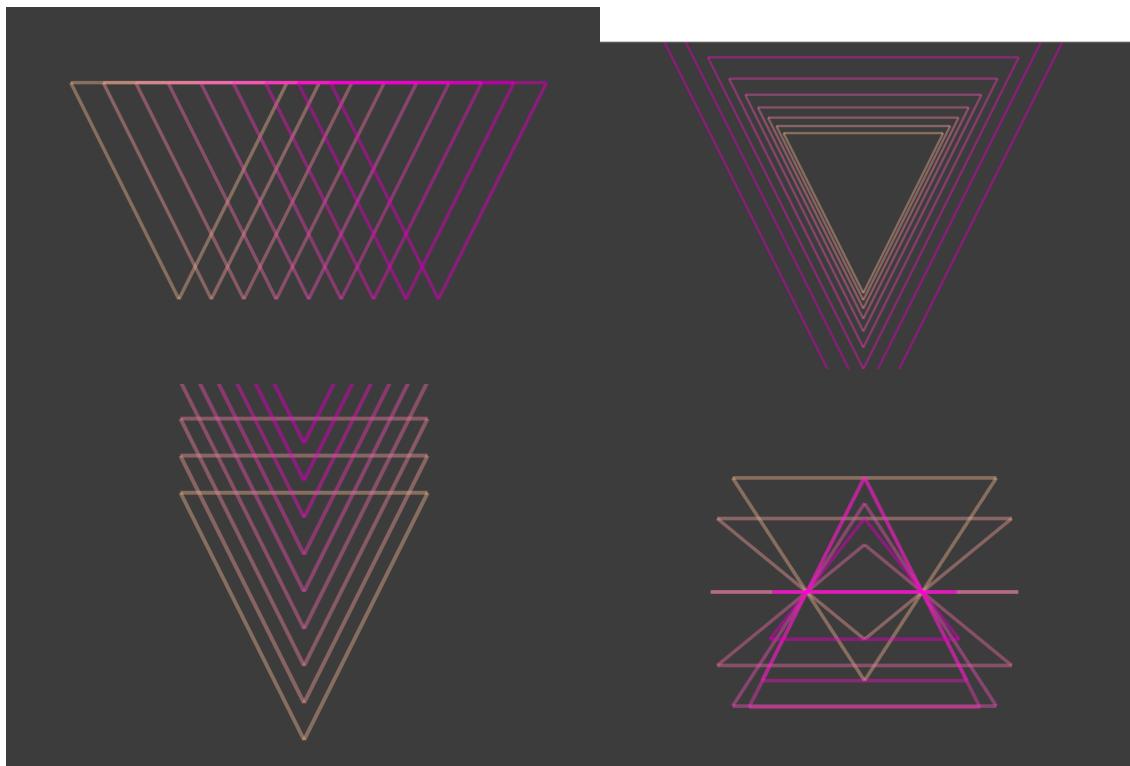
Le geometrie di base che ho utilizzato all'interno del software di generazione video sono: sfera, cerchio, triangolo e piano rettangolare. Per generare il primo livello di sfondo della nostra finestra ho generato una sfera (`of.sphere`) con raggio 500 pixel e 100 poligoni a cui viene applicata una texture dall'oggetto `of.multiimage` che legge a loop 504 fotogrammi di un video, il tutto può essere ruotato tramite l'oggetto `of.rotate` di 360° lungo gli assi XYZ.



Il video viene quindi renderizzato in tempo reale nella finestra. In aggiunta al livello di background ho pensato di introdurre un secondo livello dove ricada l'attenzione del fruttore. In questo ho inserito una delle geometrie sopra elencate. Ogni geometria possiede numerose funzioni di parametrizzazione: della sfera e del cerchio si possono selezionare raggio e poligoni, del triangolo le coordinate di ognuno dei suoi tre punti e del piano la larghezza, l'altezza e il numero di colonne e righe della matrice.

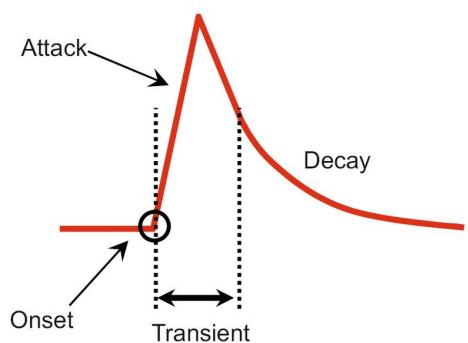


In aggiunta a questi elementi ho inserito ulteriori quattro oggetti per la modulazione delle sorgenti: traslazione, rotazione, ripetizione e informazioni di colore a loop partendo da un colore fino ad arrivare ad un altro. Tramite l'utilizzo di questi effetti, l'utente è in grado di generare dei pattern generativi che, se legati sinesteticamente, possono dar vita a performance audiovisive molto suggestive.



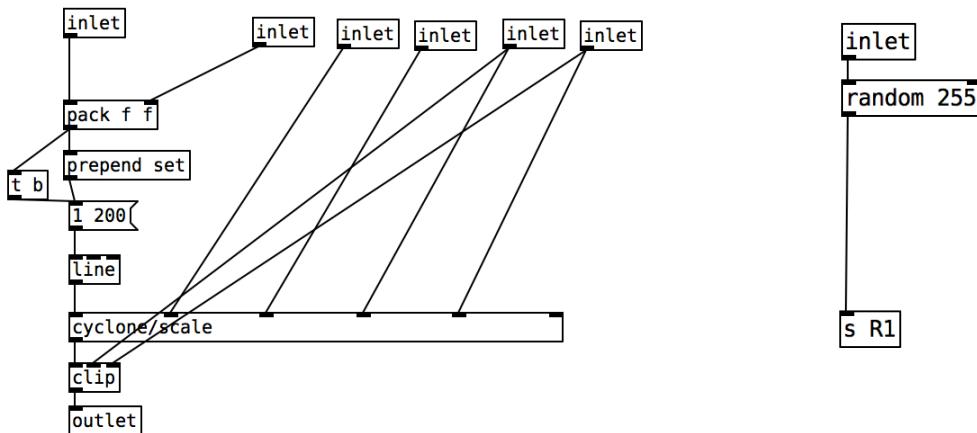
4.3.2 Descrittori audio e audio-reattività

Come ho scritto nel paragrafo introduttivo di questa tesi, l'obiettivo è quello di costruire un hardware sinestetico, in grado di produrre suoni e immagini legate in qualche modo tra loro. Il fenomeno sinestetico si basa soprattutto sulla sincronizzazione nel tempo di stimoli sensoriali differenti, per questo ho implementato un algoritmo che permetta di analizzare il suono e inviare in tempo reale dati numerici ad un ricevitore. Per l'analisi ho utilizzato la libreria di external di PD "timbreID"²⁴ questa potentissima raccolta implementa strumenti di analisi molto efficaci e precisi. La libreria può essere utilizzata per una varietà di applicazioni in tempo reale e non, inclusa la classificazione del suono, la ricerca di un preciso timbro in un suono complesso, la visualizzazione del suono, la segmentazione automatica, l'ordinamento dei suoni in base al timbro, la stima della chiave e del tempo. Nel mio caso mi sono focalizzato sulla ricerca degli onset del suono. L'onset può essere considerato il primo punto di attacco di un inviluppo, tra gli oggetti di TimbreID c'è l'oggetto bark~ che riconsegna in outlet un bang ogni volta che nel buffer di analisi (nel mio caso di 2048 campioni) si presenta un inviluppo d'attacco. Nel campo della musica elettronica pop (basata sul beat ciclico) vengono spesso utilizzati come sorgente di elementi sinestetici le informazioni di inviluppo in modo da creare loop audio-reactivi che permettono di visualizzare il suono o più precisamente il ritmo. Nel mio



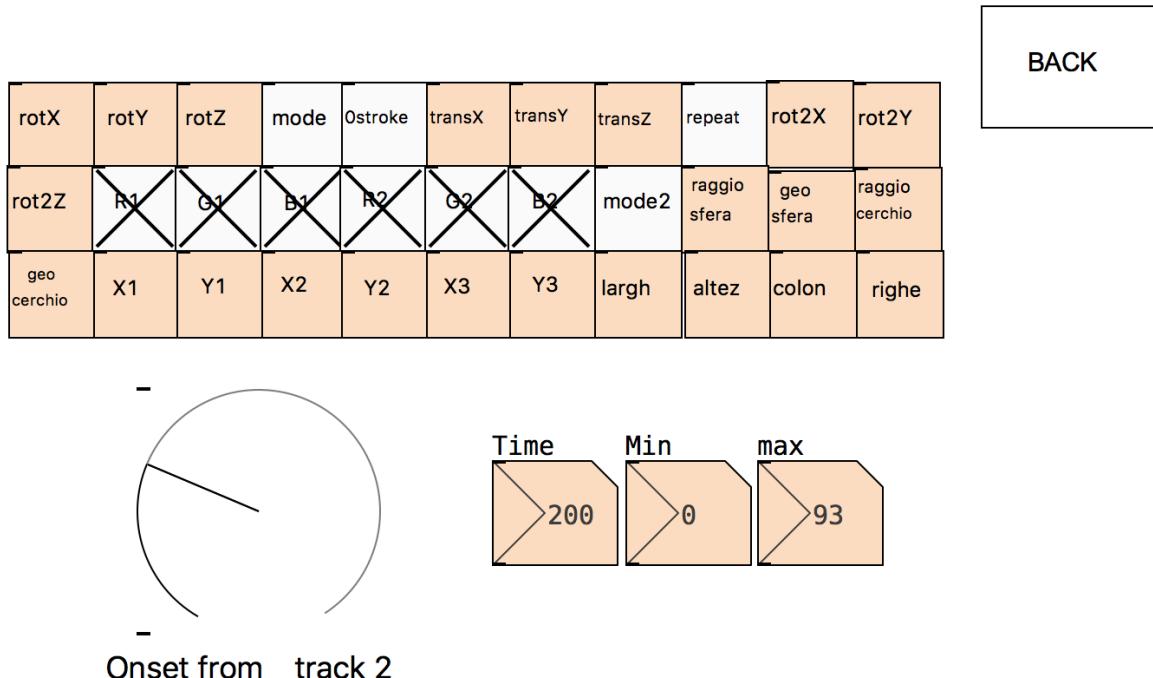
²⁴ A TIMBRE ANALYSIS AND CLASSIFICATION TOOLKIT FOR PURE DATA- William Brent - University of California, San Diego - Center for Research in Computing and the Arts - <http://williambrent.conflations.com/papers/timbreID.pdf>

caso vengono analizzati gli onset di tutte le otto tracce e i dati vengono inviati tramite una matrice di send ai parametri delle geometrie e delle modulazioni che vengono rappresentate nello schermo. Ovviamente ogni parametro ha uno specifico intervallo di valori su cui lavora, per esempio i modulatori per ruotare la figura utilizzano un intervallo da 0° a 360° mentre i valori di larghezza e altezza utilizzano la quantità di pixel. Per questo motivo ho creato due piccoli algoritmi in grado di ri-parametrizzare secondo gli intervalli giusti: uno di matrice randomica e uno con interpolazione lineare e scelta dei valori di ri-scalatura.



Le destinazioni di questi valori sono trentatré suddivise per tutto l'algoritmo di Ofelia: Rotazione asse X dello sfondo, rotazione asse Y dello sfondo, rotazione asse Z dello sfondo, modalità di rendering (linee, piano e punti), larghezza della linea, trasposizione sull'asse X, trasposizione sull'asse Y, trasposizione sull'asse Z, ripetizione, rotazione sull'asse X, rotazione sull'asse Y, rotazione sull'asse Z, valori RGB della prima forma, valori RGB dell'ultima forma, scelta della forma, raggio della sfera, lati del poligono della sfera, raggio del cerchio, geometrie del cerchio, valori XYZ per tutti i vertici del triangolo, larghezza e altezza del piano, numero di righe e colonne del piano.

L'interfaccia utente è organizzata in maniera tale che l'utente può selezionare quale degli otto onset ricevere e, tramite una matrice di bottoni, può scegliere dove inviarla e, in caso, scalare i valori adattandoli all'intervallo desiderato con un tempo di interpolazione.



Il solo utilizzo degli onset come sorgente di analisi penalizza l'audio-reattività quando si tratta di suoni più statici o morfologicamente lenti nei cambiamenti e sarà sicuramente uno degli elementi primari da prendere in considerazione per gli sviluppi futuri di Synaesthetik. Tramite l'utilizzo di questi algoritmi i risultati ottenuti, comunque, possono variare tantissimo e riconsegnare all'utente anche risultati inaspettatamente performanti. Il progetto del primo prototipo di Synaesthetik è quindi concluso.

5. Sviluppi futuri

Questo lavoro mi ha entusiasmato tantissimo, addentrarmi dentro le dinamiche dello sviluppo software osservandole a livello microscopico mi ha permesso di rendermi conto dell'enorme complessità dei sistemi di elaborazione computazionale in cui siamo immersi. Ho scoperto e colmato molte lacune che davo per assodate e mi ha insegnato come ragionare seguendo la logica computazionale dei sistemi di elaborazione. Il prototipo elaborato per questa tesi ha molti limiti e probabilmente anche qualche bug ma spero sia l'inizio di un percorso di ricerca e sviluppo che non si esaurisca con il conseguimento di questo titolo ma continui e , chissà, trovi anche un suo pubblico spazio d'interesse nelle comunità di musicisti e makers. Sicuramente il prossimo passo sarà suddividere l'elaborazione su due SoC: uno per l'elaborazione audio ed uno per l'elaborazione video, in modo da liberare risorse e non richiedere il massimo della potenza ad un singolo sistema integrato.

6. Bibliografia e Sitografia

Emanuele Ponzio - Immagine in tempo reale - Storie, pratiche, teorie per una introduzione alla performance audiovisiva - **Mimemis edizioni** - 2019

Ippolita - OPEN NON E' FREE - Comunità digitali tra etica hacker e mercato globale - **Eleuthera** – 2005

James William TOPLISS - **Victor ZAPPI** - **Andrew McPHERSON**- Latency Performance for Real-Time Audio on BeagleBone Black - <https://www.eecs.qmul.ac.uk/~andrewm/lac2014.pdf>

Miller Puckette, Pure Data: Another Integrated Computer Music Environment, Proceedings of the Second Intercollege Computer Music Concerts, Tachikawa 1997

Wright, M., Dannenberg, R., Pope, S., Rodet, X., Serra, X. and Wessel, D. Panel: Standards from the Computer Music Community at the International Computer Music Conference, International Computer Music Association, Miami, FL, 2004

Timothy E. Stinchcombe - Analysis of the Moog Transistor Ladder and Derivative Filters

Godfried Toussaint - The Euclidean Algorithm Generates Traditional Musical Rhythms - **School of Computer Science, McGill University - Montreal, Quebec-Canada** - <http://cgm.cs.mcgill.ca/~godfried/publications/banff.pdf>

Satellite by CCRMA - <https://ccrma.stanford.edu/~eberdahl/Satellite/>

Bela a platform for embedded audio - <https://bela.io/>

MiniMad - <https://madmapper.com/minimad/>

LZX Industries - <https://lzxindustries.net/>

Raspberry Pi Foundation - <https://www.raspberrypi.org/>

Bzzzbz - https://twitter.com/bzzzbz_video

Pure Data - <https://puredata.info/>

Csound - <https://csound.com/>

Cabbage - <https://cabbageaudio.com/>

Mutable Instruments Eurorack code repository - <https://github.com/pichenettes/eurorack>

Mutable Instruments ported to PD - <https://github.com/TheTechnobear/Mi4Pd>

Ivor Darreg - Biography - <http://www.afn.org/~sejic/ivor.html>

ELSE - EL Locus Solus' Externals - <https://github.com/porres/pd-else>

MrPeach Networking Library - <https://puredata.info/downloads/mrpch>

Ofelia - <https://github.com/cuinjune/Ofelia>

Ofelia fast prototyping - <https://github.com/cuinjune/Ofelia>