



MINISTERO DELL'ISTRUZIONE, DELL'UNIVERSITA' E DELLA RICERCA

**CONSERVATORIO "Alfredo Casella"
L'AQUILA**

Diploma accademico di primo livello in Musica Elettronica

**La Tangible User Interface "METIS" come strumento per la performance
elettroacustica dal vivo**

Laureando

Balandino di Donato

Relatore

Michelangelo Lupone

Anno Accademico 2011-2012

Alla mia famiglia...

INDICE

<i>Introduzione</i>	1
<i>Capitolo 1 – Metis</i>	
1.1 Hardware.....	11
1.1.1 Computer.....	11
1.1.2 Superficie d'azione e illuminazione.....	10
1.1.3 Videocamera.....	13
1.1.4 Proiettore.....	16
1.1.5 Specchio.....	17
1.1.6 Convertitori audio.....	18
1.2 I fiducial e il processo di tracking video.....	18
1.3 Costruzione e messa in opera della Tangible User Interface Metis.....	24
1.3.1 Costruzione della struttura.....	25
1.3.2 Posizionamento della superficie d'azione e dei led infrarossi.....	25
1.3.3 Posizionamento dei dispositivi Hardware.....	26
1.3.4 Processo di calibrazione.....	29
1.4 Protocolli di comunicazione (OSC e TUOI).....	33
1.4.1 OSC.....	33
1.4.2 OSC e Max/MSP.....	38
1.4.3 OSC e Processing.....	38
1.4.4 TUOI.....	39
1.4.5 TUOI e Processing.....	42
1.5 Controllo e gesto.....	43
1.6 Elaborazione grafica.....	44
1.6.1 Processing.....	45
1.6.2 Algoritmo.....	46
<i>Capitolo 2 – Elaborazione del segnale audio</i>	
2.1 Max/MSP.....	67
2.2 Algoritmo.....	68
2.2.1 main.....	68
2.2.2 dataTest.....	69
2.2.3 signalProcessing.....	70
<i>Conclusione</i>	90
<i>Appendice</i>	
“Il Sogno nel sogno” per Metis e nastro magnetico.....	101
<i>Sitografia</i>	129

Introduzione

Nell'ambito del live electronics molti performer sentono il bisogno di sentirsi liberi di agire sui sintetizzatori audio e negli ultimi dieci anni la comunità di persone che fanno riferimento alla “live electronics performance” spingono verso la creazione di interfacce multi-touch, le quali danno la possibilità al performer di interagire mediante un gesto tattile con l'interfaccia, mediante il quale si riesce ad associare dinamicamente l'elaborazione sonora e l'elaborazione grafica. Mediante questo tipo di interfacce denominate TUI (Tangible User Interface) riusciamo non solo a garantire un controllo simultaneo di più parametri dinamici (come avviene per i dispositivi MIDI), ma ad offrire un'interazione tattile diretta con lo sfondo grafico dell'interfaccia ed una possibilità di effettuare performance multimediali dove sono eseguite insieme l'elaborazione sonora e grafica.

Una delle comunità più produttive e all'avanguardia in questo ramo di ricerca musicale è la comunità NIME (New Interfaces for Musical Expression), la quale dal 2001 organizza una conferenza annuale in cui vengono presentati i progetti più validi tra quelli presentati da ricercatori di tutto il mondo che si occupano del design delle interfacce ed interazione tra computer, essere umano e musica. Nel corso di questi anni la comunità NIME, e non solo, ha affrontato svariate tipologie di interfacce con diversi tipi di possibilità d'interazione hardware come slider, controlli rotativi, pulsanti, tastire, pad, leve, touch-screen (che potrebbero essere considerati multi-touch) mixer, strumenti MIDI, tastiere, batterie chitarre, dispositivi touch e dispositivi multi-touch con un numero massimo di tocchi rilevati simultaneamente fino a controller come *Thunder*, *Eaton and Moog's, Multiple-Touch Keyboard* e la *Continuum Fingerboard*;



Illustrazione 1: batteria elettronica prodotta dalla "Roland"



Illustrazione 2: sintetizzatore AKAI.
Francoforte, Musikmesse



Illustrazione 4: Buchla prodotto dalla Thunder



Illustrazione 3: Continuum Fingerboard
prodotto dalla Haken Audio

Ma nessuna di queste consente un interazione dinamica, multimediale, performante a qualsiasi gesto umano come su un'interfaccia multi-touch, la quale consente l'interazione mediante il tocco da un numero di dita dipendente dal tipo di sistema utilizzato dunque anche di più performer simultaneamente. Un altro vantaggio delle interfacce T.U.I. è che permettono l'interazione mediante oggetti di qualsiasi forma e colore sui quali sono applicati dei marker denominati *fiducial* (illustrazione 5), ovvero delle immagini paragonabili a codici a barre, di cui parleremo più in avanti.



Illustrazione 5: fiducial

Un'interfaccia T.U.I. può essere realizzata anche a “mano”, come è stato fatto da Marco Giordano e Maria Clara Cervelli, i quali nel 2010 hanno dato vita alla *Metis* la quale prende il nome dalla divinità greca Meti. La *Metis* è un'interfaccia che fa parte della famiglia delle T.U.I. dunque permette l'interazione combinata di gesti applicabili con le dita e fiducial su una superficie multi-touch. La progettazione della *Metis* è avvenuta in due fasi: una fase di progettazione e realizzazione dell'hardware ed una del software. L'hardware progettato e realizzato del team sopra citato è composto da una struttura in alluminio che ospita una superficie in plexiglas circoscritta da una barra di led che emano luce infrarossa; al di sotto della superficie d'azione troviamo una videocamera, un proiettore, uno specchio ed una scheda audio collegate ad un computer.



Metis

Il software utilizzato per la gestione dell'interfaccia è composto da più applicazioni collegate tra loro mediante dei protocolli di comunicazione (TUIO ed OSC): reacTIVision, Processing e Max/MSP. reacTIVision si occupa di analizzare le immagini rilevate sulla superficie d'azione mediante la videocamera ed in base ai gesti applicati proiettare uno sfondo grafico, generato con un'applicazione creata mediante l'ambiente di sviluppo "Processing" e proiettato dal videoproiettore sulla superficie d'azione ed eventualmente anche all'esterno grazie un secondo proiettore. Max/MSP invece si occupa di elaborare e generare segnali audio fruibili mediante un sistema di diffusione.

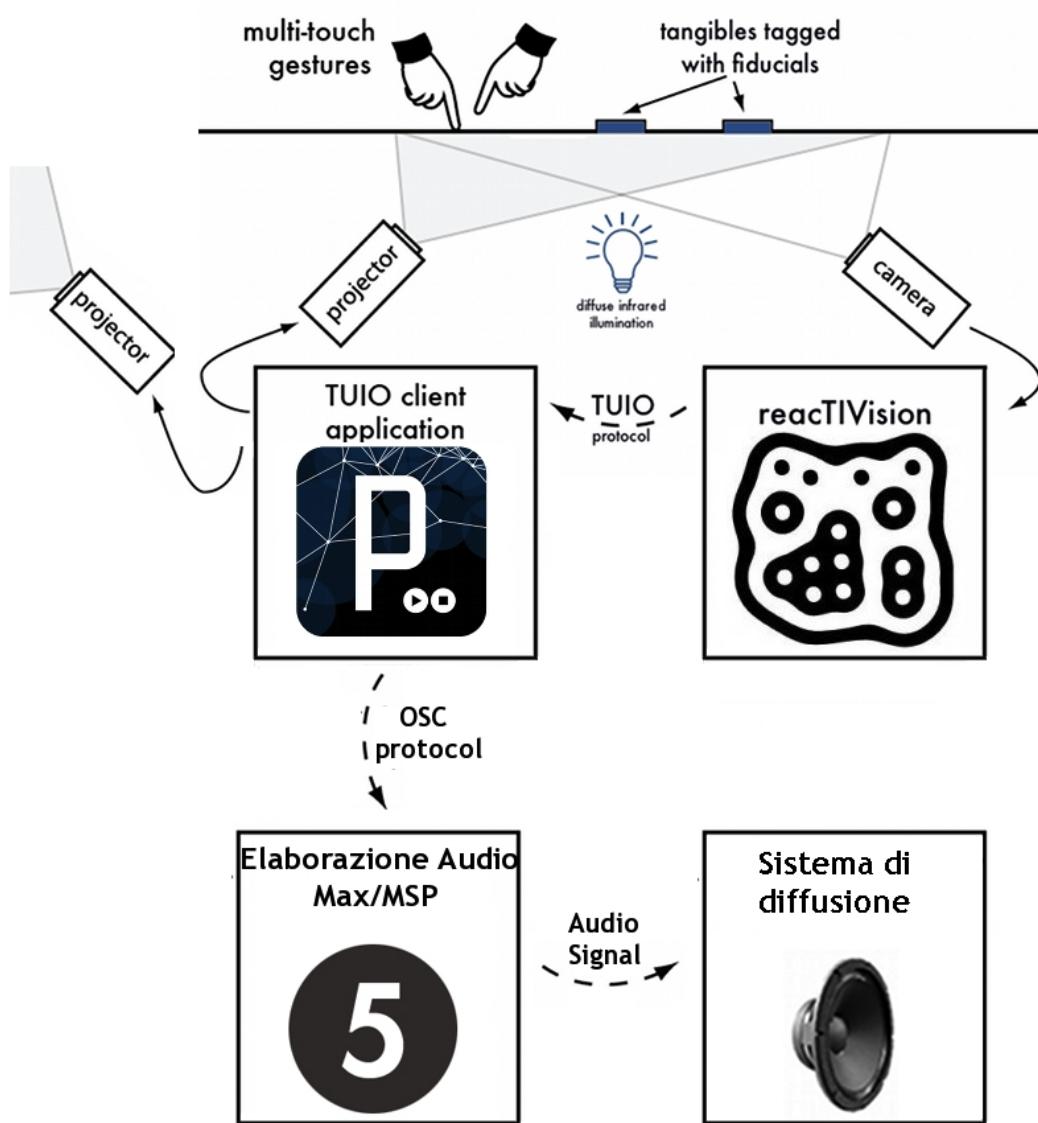


Illustrazione 6: struttura della Metis

Il mio lavoro ha avuto inizio con lo studio della Metis e delle interfacce T.U.I. e intendo le potenzialità del controller ho deciso di generare un'applicazione mediante Processing ed una mediante Max/MSP che mi permettessero di utilizzare la Metis in una performance dal vivo. Una volta analizzata l'immagine proveniente dalla telecamere, reacTIVision effettua il tracking di dita e/o fiducial, ovvero va' a rilevare i seguenti dati dall'immagine acquisita:

- aggiunta o rimozione o aggiornamento della posizione in riferimento agli assi cartesiani X,Y della Metis e/o del verso espresso in radianti di un determinato fiducial riconducibile ad numero denominato ID
- aggiunta o rimozione o aggiornamento della posizione in riferimento agli assi cartesiani X,Y della superficie della Metis di un determinato dito riconducibile ad un numero univoco denominato ID

Una volta rilevati questi dati, essi vengono organizzati in pacchetti e vengono inviati a Processing sfruttando il protocollo TUO “figlio” del protocollo OSC (Open Sound Control), entrambi i protocolli di comunicazione che sfruttano la comunicazione UDP (User Data Protocol) per trasportare pacchetti di dati da un'applicazione all'altra mediante una rete interna al computer.

I dati generati da reacTIVision verranno accolti da un'applicazione da me generata nell'ambiente di sviluppo Processing grazie alle librerie TUO, che permette di ricevere pacchetti di dati da reacTIVision (tramite le librerie aggiuntive oscP5 e netP5) che permette di inviare a sua volta pacchetti OSC a Max/MSP. L'applicazione generata è composta da tre parti principali: setup, draw, chiamata ai metodi (vedi illustrazione 7 alla pagina seguente). La prima parte quella di *setup*, si occupa di dettare le regole e le caratteristiche dell'algoritmo; il *draw* è il cuore dell'algoritmo, è il punto in cui viene fatta la scansione di tutti i dati ricevuti (processo di *polling*), ed in base ad essi vengono chiamati i *metodi* (funzioni collegate a determinati oggetti software) i quali attivano i processi di elaborazione grafica e creano dei pacchetti contenenti i dati relativi al tracking i quali vengono inviati a Max/MSP mediante il protocollo OSC.

Le elaborazioni grafiche sono di più tipi:

- cambio di sfondo grafico effettuato mediante i fiducial con ID 1, 2 e 3; ogni volta che viene aggiunto o rimosso un fiducial vi è un cambio dello sfondo grafico e ciascuno dei fiducial è legato ad un corrispettivo sfondo grafico;
- elaborazioni grafiche applicabili sul secondo sfondo grafico grazie alla variazione di posizione del fiducial con ID 4 e quella del terzo sfondo grafico grazie alla variazione della posizione del fiducial con ID 5;
- generazione di un ellisse il cui colore è stabilito mediante il suo ID, la sua luminosità mediante il valore che esprime il suo verso e la sua posizione in riferimento agli assi X,Y della Metis

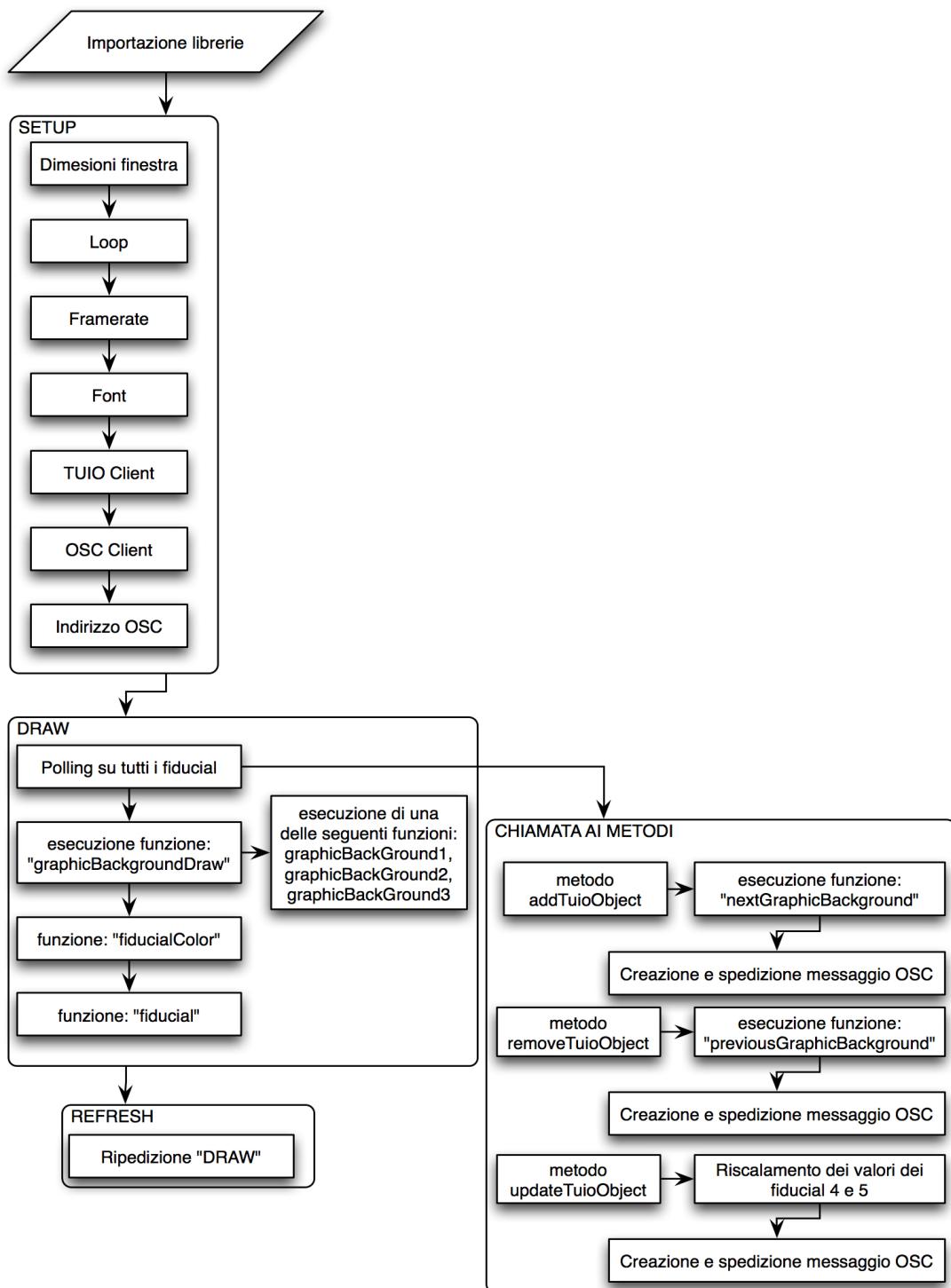


Illustrazione 7: struttura dell'applicazione generata nell'ambiente di sviluppo "Processing"

L'applicazione generata nell'ambiente di sviluppo Max/MSP è strutturata in cinque blocchi. I primi due blocchi si occupano di ricevere in ingresso rispettivamente i segnali audio ed i pacchetti di dati inviati da Processing i quali andranno a stabilire il destino dei segnali audio in ingresso (routing del segnale audio), il loro processamento e la loro spazializzazione ad eccezione del segnale all'ingresso 3 che viene inviato allo spazializzatore corrispondente e successivamente ai convertitori audio. Il routing del segnale audio, stabilito mediante il valore del verso del fiducial con ID 2 per il segnale proveniente dall'ingresso 1 e dal valore del verso del fiducial con ID 3 per il segnale proveniente dall'ingresso 2, avviene per mezzo di un'algoritmo in grado di dividere il segnale audio in ingresso in due parti ed inviarlo in due punti diversi: verso il blocco di spazializzazione o verso il blocco di processamento del segnale corrispondente, il segnale audio all'ingresso 1 al *celloProcessor* ed il segnale all'ingresso 2 al *bassSaxProcessor*.

Il *celloProcessor* ed il *bassSaxProcessor* sono il cuore del nostro algoritmo. Il segnale in ingresso al *celloProcessor* è un segnale proveniente dal violoncello il quale verrà fatto confluire interamente o in porzioni tre diversi processi di elaborazione del segnale:

- un riverbero molto lungo e corposo, in base al valore corrispondente alla posizione del fiducial con ID 4 sull'asse delle Y;
- in un banco di filtri risonanti con valori dei fattori di merito e di guadagno di ciascuno dei filtri molto alti ed un andamento delle frequenze ci centro banda con un passo pari a quello della distanza tra semitonni del temperamento equabile a partire da una frequenza di 220 Hz, in base al valore corrispondente alla posizione del fiducial con ID 4 sull'asse delle X;
- ed una FM (modulazione di frequenza) dove il segnale in ingresso sarà il segnale portante della modulazione mentre il segnale modulante e l'indice di modulazione sarà determinato da un riscalamento del segnale in ingresso, in base al valore corrispondente al verso del fiducial con ID 4;

Il segnale in ingresso al *bassSaxProcessor* invece determina:

- la generazione di un segnale formato dalla somma di un numero variabile d'impulsi audio filtrati dipendente dall'ampiezza del segnale eseguiti con una periodicità di 100 ms, il quale sarà il segnale modulante di una modulazione d'ampiezza dove il segnale modulato è il risultato di un miscelamento di due file audio letti ciclicamente dipendente dal valore corrispondente al verso del fiducial con ID 5;
- il segnale risultante dalla modulazione d'ampiezza verrà poi splittato in un banco di filtri risonanti della stessa tipologia i quello utilizzato nel *celloProcessor* ma con delle frequenze di centro banda diverse, ed una modulazione d'ampiezza dove il segnale modulante è un'onda triangolare avente una frequenza di 25 Hz ed un duty-cycle di 0.5. I due segnali appena generati vengono riscalati rispettivamente per il valore corrispondente alla posizione sugli assi x,Y del fiducial con ID 5.

Il segnale in uscita dal *celloProcessor* e dal *bassSaxProcessor* di segnale verranno sommati e convogliati verso lo spazializzatore il quale dopo aver opportunamente scalato e filtrato il segnale audio lo distribuisce verso i quattro convertitori audio, che a loro volta ci permettono la fruizione del segnale audio mediante un sistema di diffusione quadrifonico.

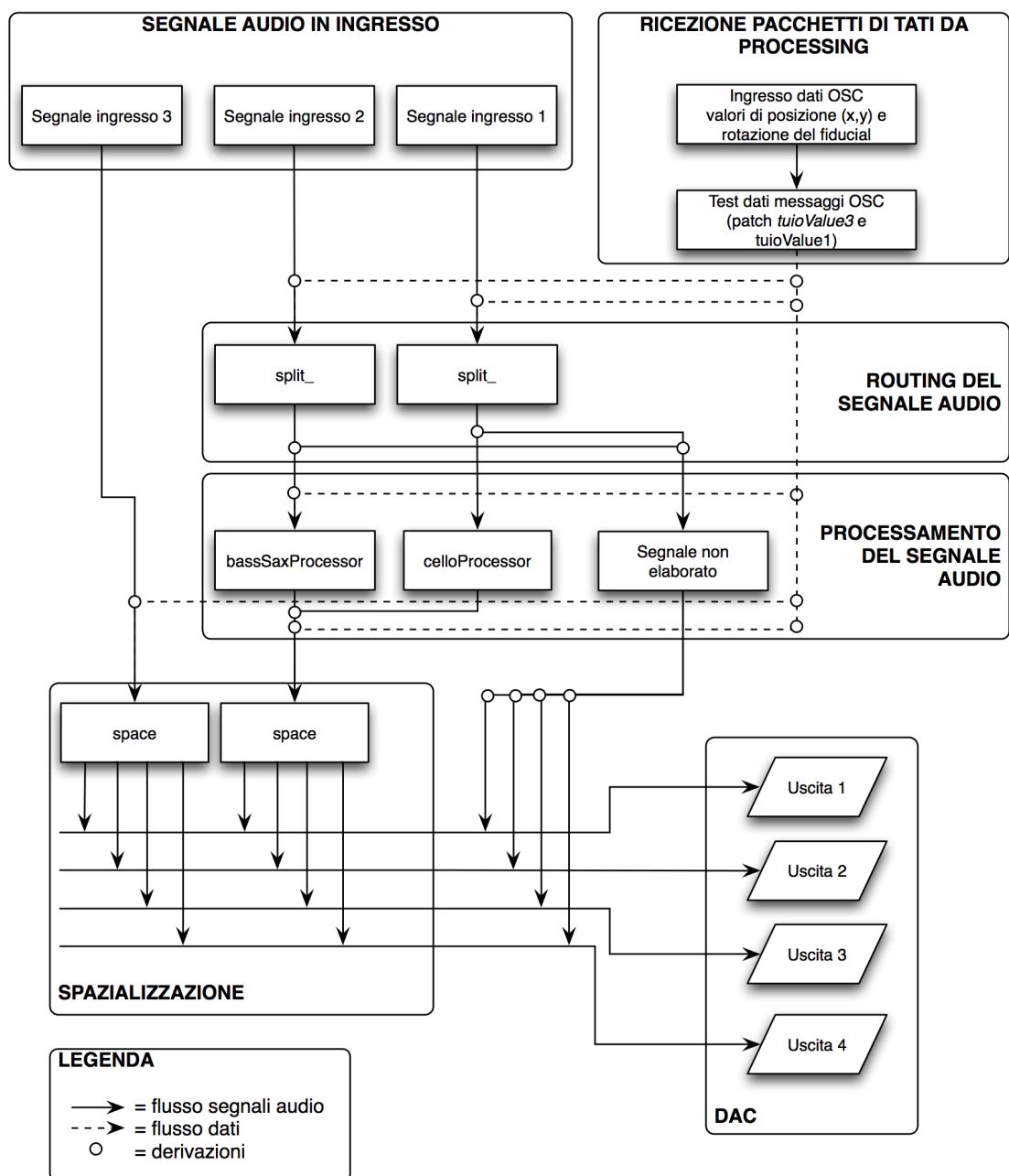


Illustrazione 8: struttura dell'applicazione generata nell'ambiente di sviluppo "Max/MSP"

Capitolo 1

Metis

1.1 Hardware

L'hardware utilizzato per realizzare la Metis è composto da una struttura di alluminio la quale contiene al suo interno un computer, una videocamera, un proiettore ed una scheda audio.

1.1.1 Computer

Il computer utilizzato è un iMac avente un processore dual core di 2.4GHz, una memoria RAM di 4 Gb con a bordo il sistema operativo OSX 10.6.8.

1.1.2 Superficie d'azione e illuminazione

Per creare un setup multi-touch all'interno del computer implementeremo un algoritmo di tracking, di cui parleremo più in avanti. Andremo a lavorare sulla superficie d'azione della Metis mediante 2 bande dello spettro della luce: l'infrarosso per il processo di tracking ed il visibile per la proiezione di uno sfondo grafico. Per questo è necessario schermare la superficie della Metis mediante l'utilizzo di LED che emanano luce infrarossa. Il loro funzionamento segue lo stesso principio dei led utilizzati per diffondere luce visibile.

Le caratteristiche da tenere in considerazione nella scelta dei led sono:

- la *potenza*: varia da 1V a 15V. Questo parametro è responsabile della quantità di luce diffusa;
- il *lobo d'irradiazione*: sta ad indicare l'apertura del fascio di luce. Esso varia dai 30° ai 90°;
- e la *lunghezza d'onda*: sta ad indicare in quale zona dello spettro della luce infrarossa opera quel tipo di led, che varia dai 3 μm ai 1000 μm .

I Led generalmente utilizzati per la creazione di uno schermo touch sono gli Osram SFH485 e Osram SFH485P i quali si differenziano solo per il lobo d'irradiazione. I primi hanno un lobo di 40° mentre i secondi di 80° e vengono scelti in base alla tecnica scelta per schermare la superficie in plexiglas di una T.U.I. di cui parleremo nel paragrafo 1.3.2 “*Posizionamento della superficie d'azione e dei led infrarossi*”

Di seguito vi sono alcuni esempi di pannelli utilizzati per la creazione di questa interfaccia:

- Polymex 50 micron double matte drafting film
- Rosco Grey
- IFOHA Digiline White
- IFOHA Digiline Contrast
- 3M Vikuiti

1.1.3 Videocamera

La Videocamera può essere utilizzata per i nostri scopi solamente dopo aver tolto il filtro a raggi infrarossi presente di fronte la lente ed aver inserito al suo posto un filtro per la luce visibile come ad esempio il negativo per foto “bruciato”, in modo tale da filtrare la luce visibile e permettere di catturare solo quella infrarossa.



Illustrazione 9: Rimozione del filtro IR;



Illustrazione 10: Applicazione del filtro per il visibile carta negativa per foto);



Illustrazione 11: Videocamera a bordo della Metis sostenuta da barre modulari e sostegni "robomac"

I parametri fondamentali da considerare nella scelta di una videocamera sono:

- a) la *risoluzione* è un parametro molto importante da considerare nella scelta di una videocamera in quanto è l'unico elemento della catena a fornirci dati utili per effettuare il tracking. Per la creazione di una superficie touch-screen di piccole dimensioni basta una Videocamera con una risoluzione di 320x240 pixel mentre per una superficie medio grande ne occorre una con una risoluzione di almeno 640x480 pixel;
- b) il *frame-rate* sta ad indicare il numero di frame captati dalla videocamera in un secondo. Più è alto questo numero e maggiore sarà il numero delle immagini da cui possiamo ricavare dati per il tracking;
- c) oggi giorno vi sono due tipi d'*interfaccia tra videocamera e computer*. Una è l'USB mentre l'altra è IEEE 1394, comunemente chiamata Fire-Wire. Quest'ultima aiuta ad avere minor latenza per merito della maggiore quantità di dati trasportati;
- d) la scelta della *lente* va fatta in base a due considerazioni: la distanza tra la superficie ed il punto d'ancoraggio della lente e le dimensioni della superficie stessa. In base a questi due valori va utilizzata una lente dotata di un appropriato angolo focale che permetta avere un fuoco il più esteso su tutta la superficie d'azione della Metis ed allo stesso tempo di avere un basso tasso di distorsione dell'immagine generata dalla geometria della lente della videocamera. Di solito vengono scelte lenti M12, C o CS con una focale di 4.3 mm la quale posta a 90 cm dalla superficie la quale ci permette di lavorare su un area di 24 x 27 pollici.

e) Di seguito una lista di telecamere utilizzabili per questo tipo di applicazioni:

<i>Casa costruttrice</i>	<i>Modello</i>	<i>Note</i>
Creative	Live Vista IM	Risoluzione 640x480 @ 15FPS, USB 2.0, Ubuntu
Creme de la Crap		Risoluzione 640 x 480 30 f/s; 1280 x 960 10 -15 f/s USB 2.0
Crypto	Compact II	Risoluzione 640 x 480 20 fps USB 2.0 con 2.10mm grand'angolo e filtro IR
Intel	YC76	
Logitech	Quickcam Chat	Risoluzione: 7 fps @352x188 compatibile anche con Linux, USB 1.1
	Quickcam Communicate STX	Risoluzione: 30fps @ 640x480.
	QuickCam Pro 9000	SMD-LED Compatibile con Linux
Microsoft	NX-6000	Risoluzione: 640x480@30fps
	VX-3000	Risoluzione: 640x480@30fps
	LifeCam VX-700	Risoluzione 640x480@30fps,
	xbox 360 cam	Risoluzione: 320x240@60fps
	VX-6000	Risoluzione: 30fps @640x480, USB 2.0.
Phillips	SPC900-NC	-
Point Grey	Dragonfly2	-
	Firefly MV	-
Samsung	VP-D101 FW	-
Sony	MiniDV DCR-HC40	Risoluzione: 720 x 480 @ 60f/s. Perfetta per il rilevamento di blobs.
Sunplus	CA533A	320x240 @ 30FPS, no IR, experimenting with ambient diffuse light.
Sweex	WC001	-
Trust	Spacecam 360	Risoluzione: 30fps 640x480.
Unibrain	Fire-i board	Risoluzione: 640x480@30fps lenti intercambiabili, FireWire
	Fire-i B/W	Con lente 2.10mm grand'angolo
Frontech	E-Cam (JIL - 2225)	Risoluzione: 30fps@640x480

1.1.4 Proiettore

L'uso dei proiettori è uno dei modi più comuni per avere una rappresentazione di uno sfondo grafico sulla superficie d'azione di un sistema T.U.I come la Metis. I proiettori si dividono in due grandi famiglie quelli con display LCD e quelli con display DLP.

I proiettori con display LCD (Liquid Cristal Displays) sono fatti proprio come i normalissimi schermi LCD di un computer o di un televisore. Essi sono costituiti da una matrice di migliaia di LED RGB i quali vengono gestiti in base all'immagine da proiettare. I proiettori DLP (Digital Light Processing) invece creano le immagini per mezzo di migliaia di specchi dinamici aventi di fronte un dispositivo rotativo contenente una serie di filtri del colore. Mediante questo sistema la luce bianca proiettata sulla matrice di specchi viene filtrata ed indirizzata verso l'esterno e la somma di tutti i fasci di luce generati dagli specchi andranno a determinare l'immagine proiettata. Come per le telecamere un elemento fondamentale nella scelta di un proiettore è il tipo di lente in relazione alla distanza tra la lente ed il pannello.

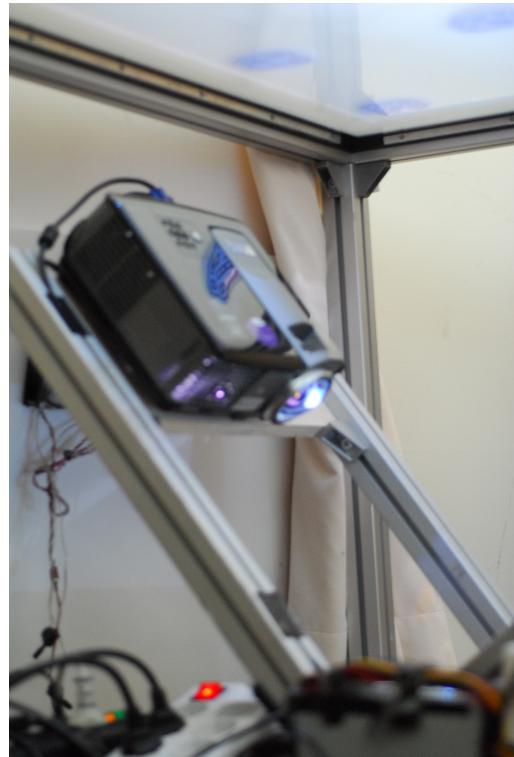


Illustrazione 12: proiettore montata a bordo della Metis sostenuta da barre modulari e sostegni "robomac"

1.1.5 Specchio

Per ottenere la proiezione di un'immagine abbastanza grande da coprire quasi interamente la superficie d'azione della Metis ci siamo dotati di uno specchio sul quale viene riflessa l'immagine proiettata dal proiettore in direzione della superficie in plexiglas in modo tale da ottenere il fascio di luce relativo alla proiezione molto più lungo che equivale a dire ottenere un'immagine maggiori dimensioni.



Illustrazione 13: specchio montata a bordo della Metis sostenuta da barre modulari e sostegni "robomac"

1.1.6 Convertitori audio

I convertitori audio devono essere tali da garantire la migliore elaborazione del segnale audio e la miglior resa sonora.

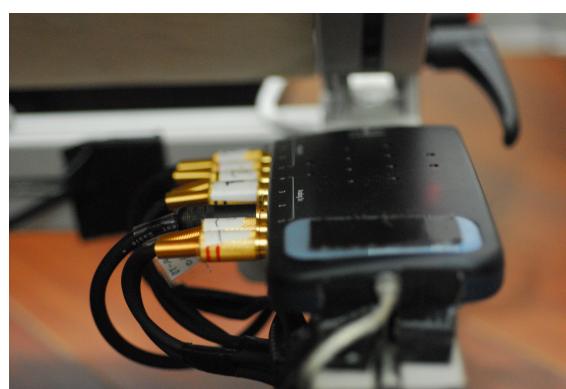


Illustrazione 14: scheda audio

1.2 I fiducial e il processo di tracking video

Il software *reactIVision* impiega una ricognizione topologica dei fiducial introdotta da Costanza e Robinson nel sistema *d-touch*. In questo tipo di approccio, da un'immagine binaria (fiducial) viene extrapolato uno schema ad albero delle regioni che formano il fiducial attraverso un processo di "segmentazione". Lo schema viene effettuato per extrapolare la gerarchia dell'immagine, dove possiamo trovare regioni in nero contenute all'interno di regioni bianche e viceversa. Le regioni dei fiducial che non contengono altre regioni appaiono come nodi dello schema ad albero.

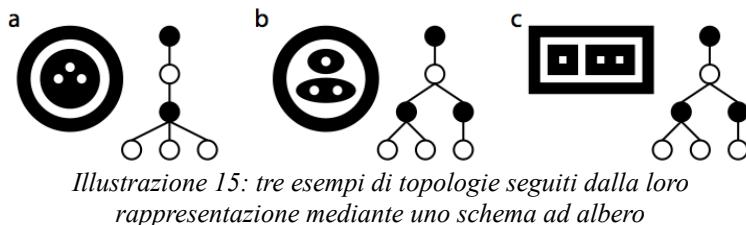


Illustrazione 15: tre esempi di topologie seguiti dalla loro rappresentazione mediante uno schema ad albero

Si può osservare nell'illustrazione 17 come il grafico dei fiducial b e c sono uguali nonostante la geometria dei fiducial corrispondenti è differente. Un'importante proprietà degli schemi ad albero derivati dai fiducial è che appartengono ad una stessa classe di grafici ramificati ad albero dunque il sistema d-touch impiega una singola topologia per tutti i fiducial nel set.

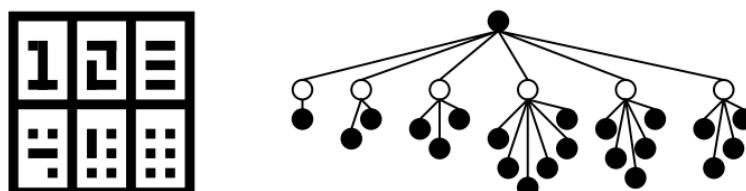


Illustrazione 16: esempio di topologia seguita dalla rappresentazione mediante uno schema ad albero

L'illustrazione 16 ci mostra un fiducial d-touch ed il suo grafico corrispondente. I fiducial appartengono ad un set formato da una griglia contenente sei regioni bianche ed ognuna di esse contiene diverse regioni nere, da una a sei. Il set contiene 120 fiducial unici i quali sono differenziati mediante un codice permutabile in uno schema ad albero grazie al numero delle regioni nere contenute all'interno di un set ordinato spazialmente di regioni bianche. Per esempio la lettura in senso orario della figura sopra (1, 2, 3, 6, 5, 4).

Abbiamo identificato un numero di aspetti dell'approccio del sistema d-touch che ha offerto idee per il miglioramento nel nostro contesto: per prima cosa il sistema d-touch si lega all'estrazione del codice permutabile in uno specifico set di fiducial. In secondo luogo la pubblicazione originale del d-touch non prescrive un metodo specifico per calcolare la sua localizzazione ed il suo orientamento. Infine la geometria semplice dei fiducial *d-touch*, realizzabile anche a mano, non è progettata per ridurre le dimensioni del fiducial.

Seguendo la loro evoluzione ed una nuova implementazione del riconoscimento dei fiducial d-touch si è deciso di produrre dei fiducial più piccoli ed eventualmente arrivare ad una tecnica scalabile che permette di variare la dimensione dei fiducial in base al loro numero. Un'altro obiettivo è stato quello di esplorare un metodo con il quale si evitasse di ricorrere ad una tecnica di processamento delle immagini come un riconoscimento ad incrocio delle immagini denominato *matching*. Dato che lo schema ad albero delle regioni è già esistente ed era relativamente costoso da definire, in termini di potenza di calcolo, è sembrato saggio farne il massimo utilizzo. I fiducial reacTIVision, da noi utilizzati, sono geometricamente indefinibili grazie alla loro struttura irregolare. Ogni fiducial in un set ha un'unica topologia. Oltre ad esprimere la giusta topologia, la geometria di ogni fiducial è obbligata dal metodo usato per calcolare la sua posizione ed il suo orientamento. Il metodo per calcolare la localizzazione e l'orientamento è influenzato significativamente dal progetto dell'algoritmo di segmentazione che conserva solamente tutte le regioni allineate con gli assi dei margini del fiducial. Il centro del fiducial viene calcolato facendo un'approssimazione del centro di tutte le regioni se le regioni hanno una geometria quadrata, circolare o relativamente piccola.

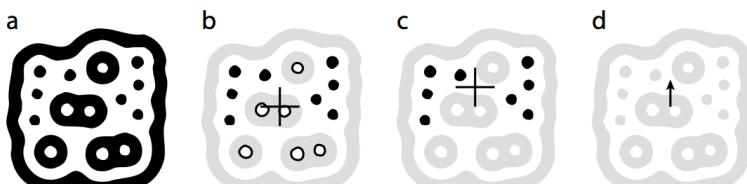


Illustrazione 17: (a) fiducial reacTIVision (b) regioni nere e bianche ed il loro centroide (c) le regioni nere ed il loro centroide e (d) il vettore che calcola il verso del fiducial

Dalle illustrazioni 17, 18 e 19 possiamo intuire come le regioni nere e bianche del fiducial sono le parti che andranno a definire la struttura dello schema ad albero che lo rappresentano, allo stesso tempo la loro media “spaziale” stabilisce il centroide del fiducial con il quale si arriverà a determinarne la posizione esatta. Nell'illustrazione 18 b notiamo come viene calcolato il centroide del fiducial applicando il calcolo della media spaziale tra le regioni bianche e nere, invece il centroide risultante delle sole regioni nere ci dà il vertice del vettore che definisce il verso del fiducial (Illustrazione 17 c), rappresentato nell'illustrazione 17 d. Ogni regione è opportunamente disegnata in modo tale da generare un determinato tipo di grafico di struttura ad albero, ed è stato applicato questo metodo per via del fatto che può essere applicato per ogni tipo di fiducial avente almeno una regione bianca ed una nera oltre a dare la possibilità di cambiare la struttura topologica del fiducial senza cambiare metodo di rilevazione. In pratica invece abbiamo constatato che per effettuare un corretto e stabile tracking si devono generare fiducial aventi almeno 4 regioni bianche e 4 regioni nere.

Prima di generare un fiducial è opportuno creare un set di schemi ad albero in modo tale anche da generare un set di fiducial tale da garantire almeno le regioni minime per effettuare correttamente il tracking. Il margine d'errore nel tracking dei fiducial aventi un elevato numero di regioni è minore rispetto ad un fiducial avente un basso numero di regioni in quanto lo schema ad albero che ne risulterebbe è più complesso quindi unico e difficilmente confondibile con un diverso fiducial avente uno schema ad albero più o meno simile. Dunque quando stiamo creando un fiducial corrispondente ad un determinato numero noi generiamo randomicamente una serie di fiducial e solo dopo decidiamo personalmente qual è il fiducial che più soddisfa le nostre esigenze di tracking, ovvero i fiducial aventi il maggior numero di regioni di cui la loro rappresentazione ad albero arrivi il più in profondità possibile.

La prima cosa di cui tenere conto nel processo di creazione di un fiducial è quella di disegnare un fiducial avente il suo centroide sempre allo stesso posto e che corrisponda con il centro del fiducial fisico.

Il fiducial può essere disegnato in un numero quasi infinito di modi, l'importante è che rispetti le caratteristiche appena elencate. Il team che ha sviluppato reacTIVision ha scelto di creare la tipologia di fiducial (da noi utilizzati) mediante un algoritmo che tiene conto degli aspetti fondamentali nel processo di tracking come: area del fiducial, simmetria, centroide e centroide del vertice del vettore. Dopo una serie di sperimentazioni su questo metodo si è scelto di posizionare le varie regioni in maniera circolare partendo da un punto fisso, con una rotazione angolare fissa e mantenendo una distanza tra le regioni costante. La lista di questi angoli andrà a determinare la lista dei genotipi dei fiducial per l'algoritmo genetico. Dunque una volta stabilito la struttura ad albero del fiducial è semplice distribuire arbitrariamente le regioni bianche e nere in maniera da soddisfare la localizzazione e l'orientamento di un fiducial. Alla base di queste regole hanno implementato un sistema computazionale in grado di calcolare differenti versioni di fiducial in un tempo al limite dell'accettabile. Quando si produce un set di 128 fiducial che formano uno schema ad albero avente 19 nodi ed un massimo di profondità di ramificazione a 3 livelli s'impiegano 12 ore di computazione mediante 11 computer collegati tra loro con a bordo un processore da 1 GHz Pentium, i quali producono 20 gruppi di fiducial di cui ogni gruppo è composto da 500 fiducial e successivamente ne vengono scelti i 128 che rappresentano il miglior risultato generato.

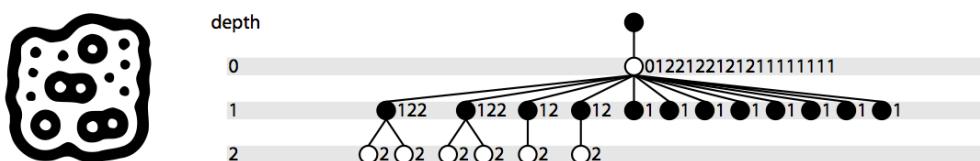


Illustrazione 18: a sinistra un fiducial e a destra la sua rappresentazione mediante uno schema ad albero

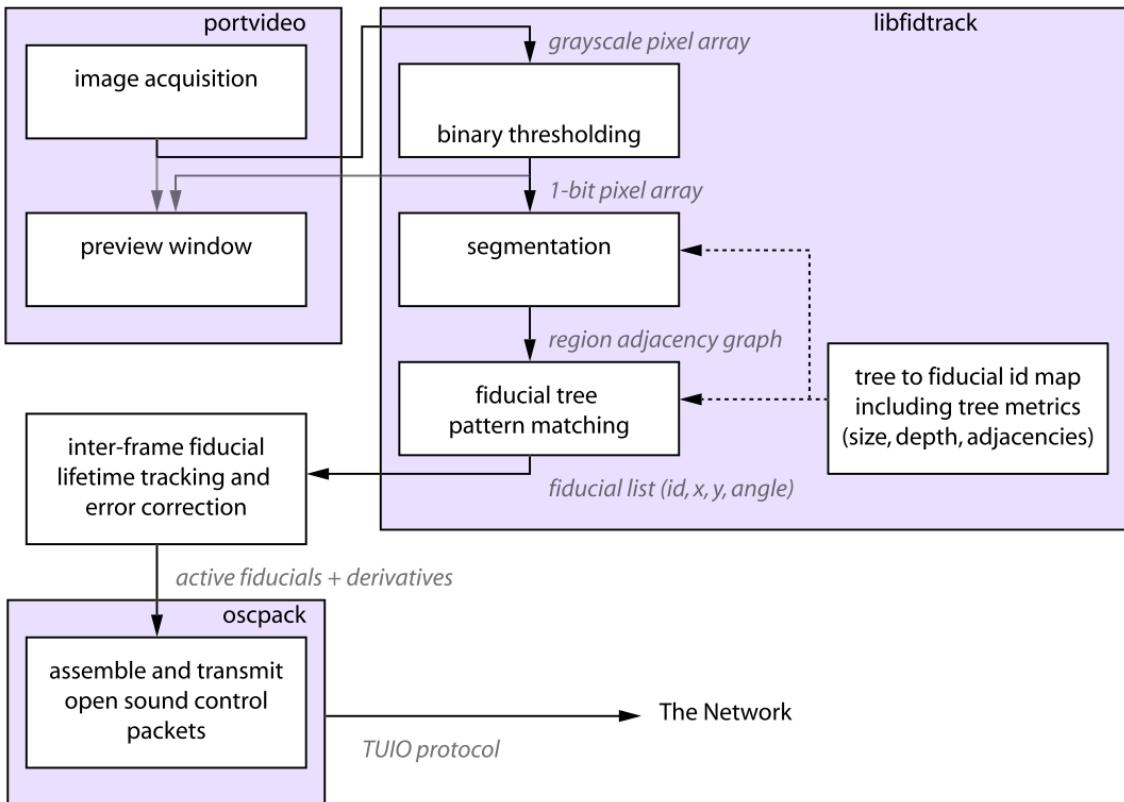
Per identificare ogni tipo di grafico gli viene assegnato una sequenza canonica di numeri la quale è in grado di descrivere anche la sua organizzazione dall'origine fino all'ultima delle ramificazioni. La sequenza della profondità è dunque definita come la sequenza dei valori indicanti la profondità dei nodi. Dove la ramificazione ha profondità 0 la profondità di qualsiasi altro nodo è il numero di bordi tra esso e la radice. La sequenza di profondità può essere utilizzata unicamente per descrivere l'ordine di profondità dello schema ad albero, tuttavia uno schema ad albero non ordinato può essere tradotto mediante più di una sequenza di valori. Per esempio la sequenza

0,1,2,3,3,2,3 e la sequenza 0,1,2,3, 2,3,3 sono entrambi valide per il fiducial dell'illustrazione 17 c. Per risolvere questa ambiguità possiamo ordinare la sequenza di numeri tenendo conto delle ramificazioni dalla più profonda alla meno profonda. Dunque la parte sinistra della sequenza che descrive la struttura alberata di un fiducial riporta i nodi più profondi della struttura e man mano che si scorrono le cifre verso destra si arriva ai nodi più superficiali. L'illustrazione 28 riporta lo schema ad albero delle regioni di un fiducial e si può notare come i nodi che indicano le regioni più profonde sono distribuite verso sinistra e man mano che si va verso sinistra si arriva a quelle superficiali; il nodo superiore non corrisponde a nessun livello dello schema ad albero in quanto serve ad indicare la zona limite che contiene tutte le regioni del fiducial.

Tracking

Lo schema 1 (nella pagina seguente) riporta la struttura del sistema reacTIVision la quale aggrega tre diverse librerie: *PortVideo*, *libfidtrack* e *oscpack*. PortVideo è un'applicazione d'acquisizione video in tempo reale multi-piattaforma la quale fornisce una finestra di visione dell'immagine catturata; libfidtrack è una libreria che implementa l'algoritmo per la riconoscione dei fiducial; oscpack è un applicazione che permette di creare pacchetti di messaggi OSC e trasmetterli (vedi paragrafo: 3.4.1 OSC).

Il sistema di comunicazione tra le varie applicazioni è il protocollo TUIO (vedi paragrafo: 3.4.2 TUIO). Inoltre vi è un codice, un algoritmo, il quale si occupa di fare da collante tra le varie applicazioni e di correggere eventuali errori generati dall'algoritmo.



Schema 1: sistema reactTIVision

PortVideo acquisisce i frame video in scala di grigi dal sistema operativo e li invia a libfidtrack. Un algoritmo “di soglia”, che filtra l’immagine captandone solo la componente utile per il tracking, produce un’immagine binaria la quale sarà utilizzata da un algoritmo di segmentazione che segmenterà il fiducial e lo ricondurrà al suo schema ad albero. Il grafico verrà poi mandato ad un algoritmo di “matching” che comparerà la struttura ad albero con un set di strutture ad albero pre-caricate e ne ricava il suo corrispondente. La cognizione del fiducial ed i suoi identificatori con localizzazione ed orientamento vengono passati ad un algoritmo il quale tiene traccia dei dati mano a mano che diventano visibili e che scompaiono. Lo stesso modulo mediante questi dati è in grado di calcolare un secondo ordine di informazioni come la velocità e l’accelerazione del vettore. Infine i dati vengono formattati seguendo il modello del protocollo Open Sound Control commutati per essere trasmessi dal protocollo TUOI e successivamente spediti mediante la rete.

1.3 Costruzione e messa in opera della Tangible User Interface Metis

La differenza sostanziale di una superficie interattiva come la Metis rispetto a tutte le altre superfici multi-touch come quelle dei tablet, cellulari e molti altri dispositivi dotati di touch-screen è che una T.U.I. come la Metis ha la possibilità di captare un alto numero di oggetti come fiducial e dita, dipendente dalle dimensioni della superficie d'azione, mentre tutti gli altri dispositivi sono vincolati da un numero massimo di punti di tracking.

Prima di iniziare a costruire una superficie interattiva come questa è bene creare un progetto della struttura e della superficie d'azione in quanto, come abbiamo visto nel paragrafo precedente, le caratteristiche hardware di una T.U.I. dipendono interamente dal tipo di struttura che si vuole implementare.

Una volta progettata la struttura e procurati tutti i dispositivi elencati nel precedente paragrafo si può andare avanti con la seguente procedura:

- costruzione della struttura
- posizionamento dei led infrarossi e della superficie d'azione
- posizionamento dei dispositivi hardware
- cablaggio
- test e verifica del funzionamento dell'hardware
- calibrazione
- test

1.3.1 Costruzione della struttura

La struttura è stata costruita mediante barre di alluminio modulari “RoboMac”, che hanno permesso di creare una struttura mobile leggera, che poteva soddisfare tutte le esigenze come quella di un supporto mobile per sostenere il proiettore e lo specchio, aiutare il posizionamento dei dispositivi hardware, assicurarne la stabilità ed il corretto ed ordinato cablaggio.

1.3.2 Posizionamento della superficie d'azione e dei led infrarossi

La superficie d'azione viene posizionata in maniera parallela alla base della Metis e fissata e sorretta mediante una cornice in alluminio creata ad hoc (illustrazione 19).

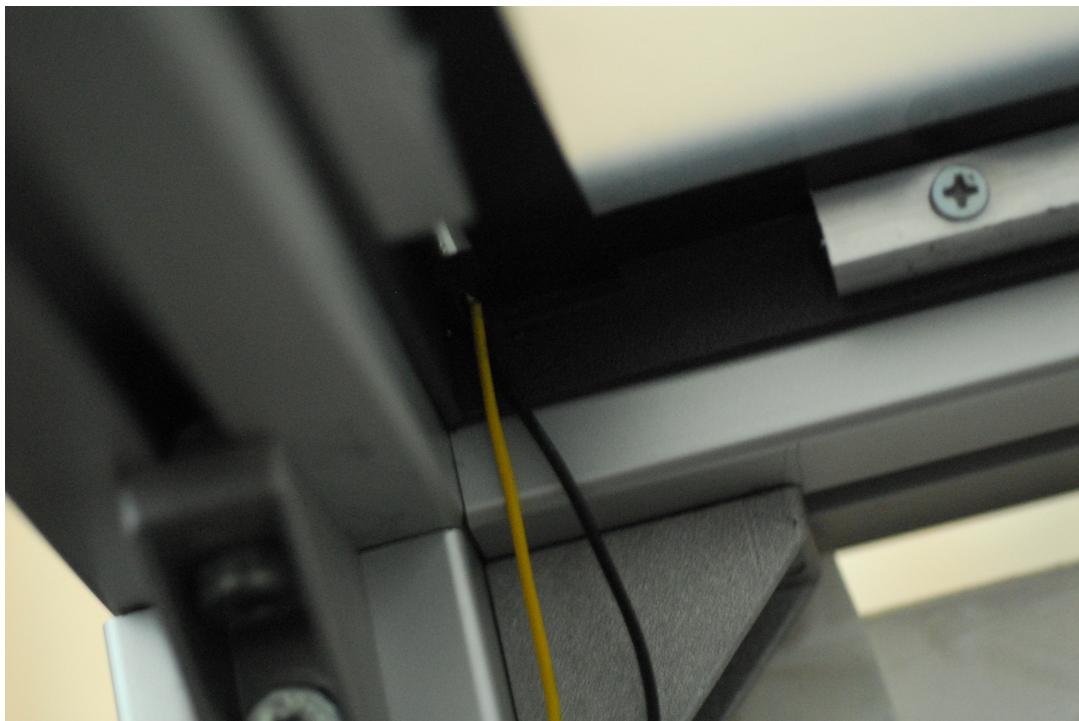


Illustrazione 19: abлагgio dei LED infrarossi che circoscrivono la superficie d'azione della Metis sorretta da una cornice in alluminio

L'illuminazione della superficie mediante l'utilizzo di LED infrarossi può essere messe in pratica sfruttando diverse tecniche:

- Frustrated Total Internal Reflection (FTIR)
- Diffused Illumination (DI)
- Laser Light Plane (LLP)
- Diffused Surface Illumination (DSI)
- Near Touch Illumination (NTI)

Quella utilizzata per la realizzazione della Metis è la Diffused Surface Illumination (DSI) (illustrazione 20).

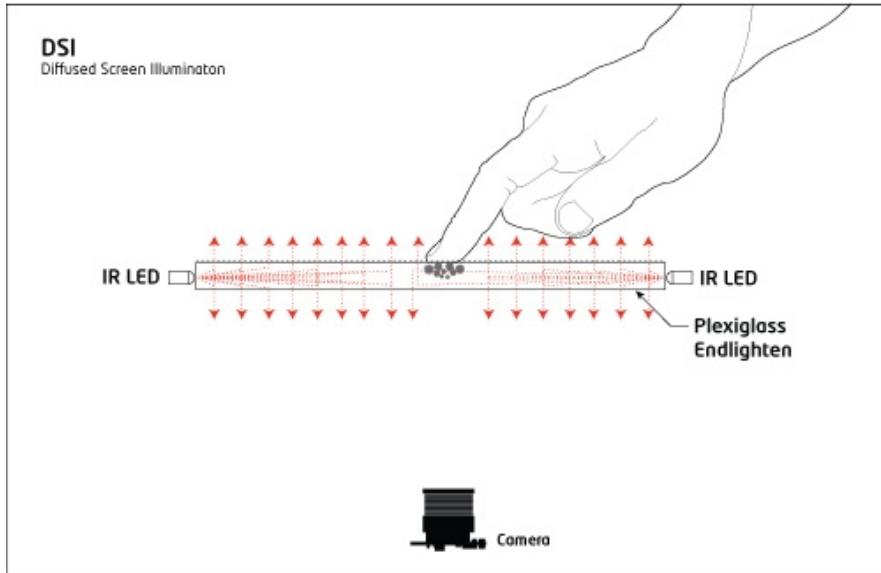


Illustrazione 20: Diffused Screen Illumination;

Questo tipo di diffusione viene può essere applicata illuminando dai bordi della superficie in plexiglas dall'esterno verso l'interno ponendo i LED in maniera perpendicolare intorno al suo bordo. La luce infrarossa emanata viene intrappolata all'interno della superficie in plexiglas mediante la sua struttura interna formata da migliaia si “specchietti” i quali riflettono la luce all'interno del materiale. Lo spessore e trasparenza del plexiglas devono essere tali da poter intrappolare la luce infrarossa, essere resistenti a pressioni fatte con le dita e a sostenere un buon numero di oggetti senza flettersi verso il basso ed permettere una buona proiezione.

1.3.3 Posizionamento dei dispositivi hardware

Posizionamento del proiettore.

Ci sono due tecniche basilari per l'istallazione del proiettore all'interno di una T.U.I.: in-roof e below-roof. La prima consiste nel disporre il proiettore sul fondo della T.U.I. in maniera perpendicolare alla superficie d'azione e con la lente rivolta verso di essa. La seconda tecnica prevede che il proiettore sia rivolto verso uno specchio il quale è rivolto a sua volta verso la superficie d'azione, in modo tale da incrementare la

lunghezza del fascio luminoso del proiettore ed ottenere un'immagine proiettata molto più grande sulla superficie d'azione. La seconda opzione è stata quella scelta per costruzione della Metis, in quanto permette di ottenere una proiezione dello sfondo grafico di dimensioni accettabili (in previsione di una performace live e dell'interazione di più performer) anche con una struttura ed una superficie d'azione di modeste dimensioni come quelle della Metis. Se si fosse scelta la prima opzione la dimensione dello sfondo grafico proiettato sarebbe stata di gran lunga inferiore.

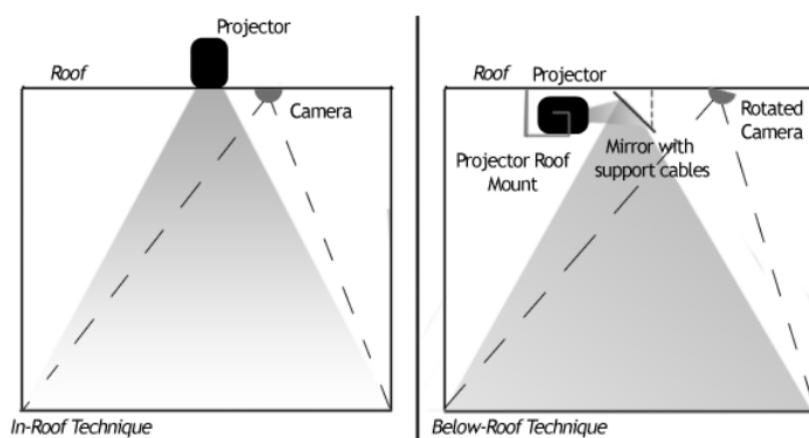


Illustrazione 21: tecniche di posizionamento dello specchio

È molto importante inoltre che il proiettore come tutti gli altri componenti siano disposti su una struttura che permetta di bloccarne la posizione, in modo tale che essa non vari nel tempo. Venire meno a questa indicazione porterebbe una serie di problemi legati alla calibrazione della videocamera (che affronteremo più in avanti) e di conseguenza otterremmo errori sia dal punto di vista sonoro che grafico.



Illustrazione 22: posizionamento del proiettore e dello specchio mediante il sistema di barre modulari in alluminio, connessioni e sistemi di bloccaggio "robomac"

Posizionamento della Videocamera

La Videocamera dev'essere posizionata in maniera tale da avere il fuoco al centro alla zona dove hanno luogo la maggior parte delle interazioni con superficie d'azione della Metis, in modo tale da non dover interagire con zone di ripresa dove la distorsione dell'immagine catturata è maggiore.



*Illustrazione 23:
Videocamera a bordo
della Metis sostenuta da
barre modulari e sostegni
"robomac"*

Computer, scheda audio, alimentatore, punti di corrente e cablaggio

La posizione del computer, della scheda audio, dell'alimentatore e dei punti di corrente non è di particolare rilevanza, l'importante è che vengono disposti in maniera tale da garantire un buon dissipamento del calore di tutti i dispositivi e che favoriscano un cablaggio il più possibile ordinato in maniera tale da avere una chiarezza nella risoluzione degli eventuali problemi e di aumentare la mobilità della T.U.I., opportunamente dotata di ruote come la Metis.

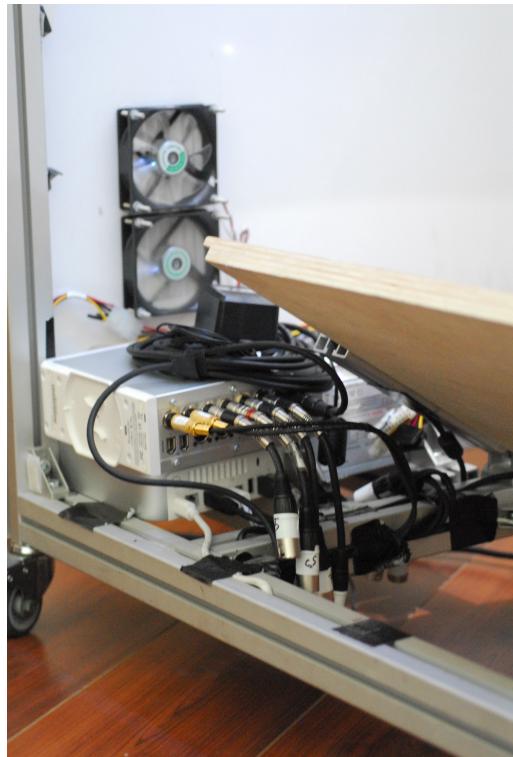


Illustrazione 24: computer, seconda scheda audio e ventole di raffreddamento

Test e verifica del funzionamento dell'hardware

Una volta terminate le fasi di montaggio dell'hardware possiamo procedere con l'accensione di tutti i dispositivi e fare un test del funzionamento e della corretta comunicazione tra tutti i dispositivi.

1.3.4 Processo di calibrazione

Questa è una delle fasi più importanti della messa in opera di una T.U.I. in quanto da qui dipenderanno tutti i valori in uscita dal processo di tracking che poi andranno a pilotare gli algoritmi di elaborazione grafica e audio. I software disponibili per effettuare questa operazione sono: *CCV*, *Touchlib* o *recTIVision*.

Molte delle interfacce create utilizzano telecamere con grandangolo per aumentare il campo di ripresa, viste le distanze ridotte tra la lente e la superficie d'azione. Queste lenti sfortunatamente distorcono l'immagine e reacTIVision è in grado di correggere questa distorsione mediante una mappa contenuta al suo interno i cui punti di riferimento sono variabili e dopo aver proiettato una mappa avente le stesse caratteristiche sulla superficie d'azione si cerca di far corrispondere tutti i punti cardine della mappa contenuta in reacTIVision con quella proiettata sulla superficie d'azione, in modo tale da ottenere una corrispondenza tra l'immagine acquisita da reacTIVision e quella reale. Per queste ragioni è stato scelto reaTIVision.

Per la calibrazione vengo proiettate mappe create ad hoc per tavoli con una superficie rettangolare o quadrata.

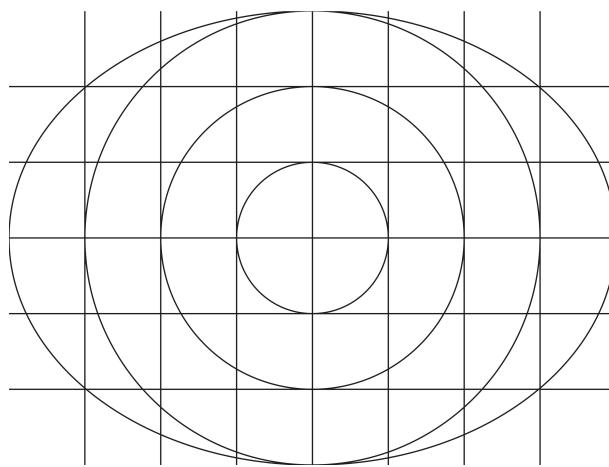


Illustrazione 25: mappa per il processo di calibrazione per T.U.I con una superficie d'azione rettangolare

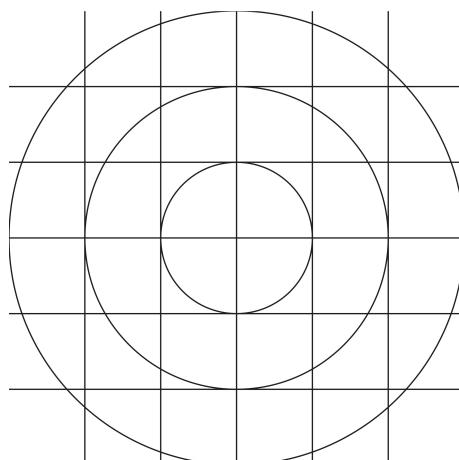


Illustrazione 26: mappa per il processo di calibrazione per T.U.I con una superficie d'azione quadrata

Di seguito verrà elencato il procedimento per effettuare una corretta calibrazione della Metis:

- apertura dell'applicazione reacTIVision ed apertura del menu di calibrazione mediante il tasto “h”

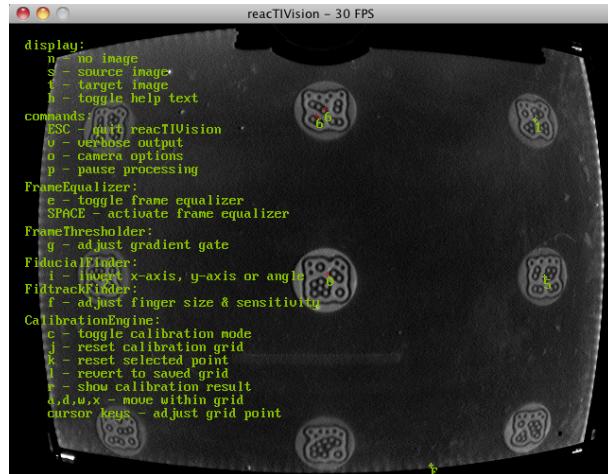


Illustrazione 27: menu di reactIVision

- calibrazione del fuoco, contrasto, gain, luminosità, nitidezza ed esposizione della Videocamera;

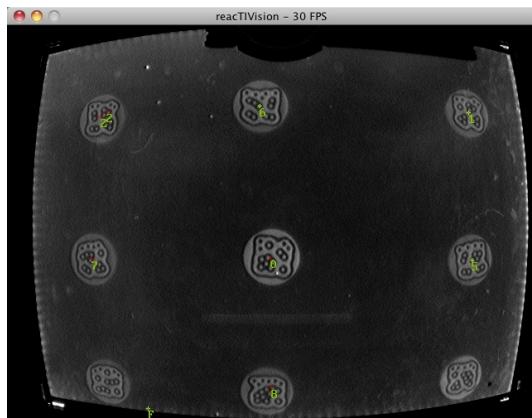


Illustrazione 28: immagine sorgente prima della calibrazione



Illustrazione 29: immagine sorgente dopo la calibrazione appena descritta

- calibrazione del tracking delle dita;
- proiezione della mappa sulla superficie d'azione;

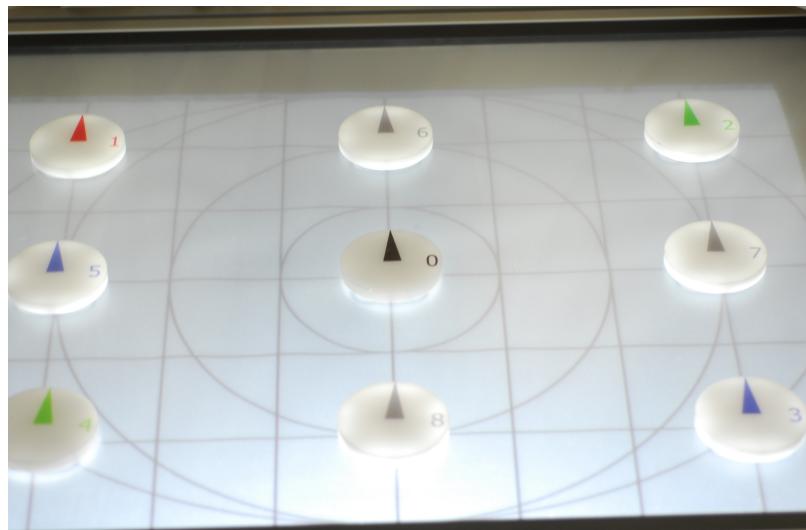


Illustrazione 30: mappa di calibrazione proiettata sulla superficie d'azione

- accesso alla modalità calibrazione mediante il tasto “c”;
- allineamento di ogni punto della mappa di *reactIVision* con la mappa proiettata. Mediante i tasti “a”, “d”, “w”, “x” si può scegliere qual è il punto della mappa di *reactIVision* da allineare alla mappa proiettata;

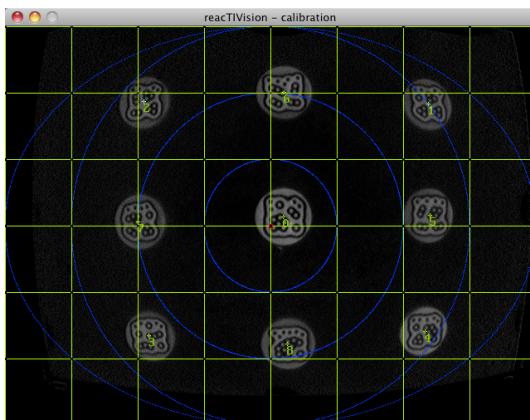


Illustrazione 31: immagine sorgente prima della calibrazione della mappa

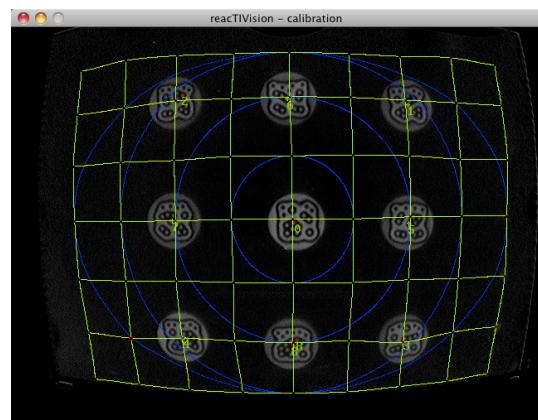


Illustrazione 32: immagine sorgente prima della calibrazione della mappa

- verifica che l'allineamento dell'immagine sia corretto premendo il tasto “r”. Questa funzione permette di visualizzare l'immagine captata dopo aver subito l'effetto della calibrazione
- posizionamento di un fiducial e di dita su più punto della superficie d'azione e verifica della corretta corrispondenza tra posizione reale e valori in uscita da *reactIVision*.

Questo processo come già detto è molto importante per far sì che le immagini acquisite dalla Videocamera siano il più possibili corrispondenti a quello che accade sulla superficie d'azione e mediante questo processo di calibrazione della “qualità” dell'immagine e di correzione della distorsione dell'immagine dovuta alla geometria semisferica della lente riusciamo ad ottenere dei buoni risultati per effettuare un corretto tracking.

1.4 Protocolli di comunicazione (OSC e TUO)

Un protocollo di comunicazione è un sistema basato su un determinato formato da messaggi e relative regole di scambio tra due sistemi di calcolo e/o comunicazione. I protocolli da noi utilizzati per la comunicazione tra le varie applicazioni implementate sono il protocollo OSC Open Sound Control ed un suo derivato, il protocollo TUO.

1.4.1 OSC

OSC è l'acronimo di Open Sound Control, un network sviluppato dal CNMAT, UC Berkeley. Open Sound Control è un protocollo per la comunicazione tra computer, sintetizzatori e altri dispositivi multimediali ottimizzati per le tecnologie dei moderni network ed è utilizzato in molti campi. Esso permette il controllo in tempo reale di processi multimediali esterni alla macchina che si sta utilizzando. Esso è senza limiti definiti, dinamico, produce dati con un'alta risoluzione, i riceventi del messaggio possono essere più di uno, i messaggi vengono inviati con un'alta risoluzione temporale ed i vari pacchetti di messaggi possono essere modificabili simultaneamente tra loro. L'unità di dati trasmessi mediante il protocollo OSC si chiama *OSC Packet* (pacchetto). Ogni applicazione che invia pacchetti OSC è un *Client*, ed ogni applicazione che li riceve è un *server*. Un pacchetto consiste in un blocco continuo di dati la cui dimensione può essere di 8 bit o suoi multipli. Il responsabile della consegna e/o

ricezione del messaggio è la rete a cui si fa riferimento. In un protocollo basato su un flusso di dati come quello TCP, il flusso può iniziare con un int32 che dà le dimensioni del primo pacchetto, seguito dal contenuto del primo pacchetto e a sua volta seguito dalle dimensioni del secondo pacchetto. Il contenuto di un pacchetto deve essere un messaggio o un *OSC Bundle*. Il primo byte del pacchetto deve dare le giuste informazioni per poter distinguere se è un messaggio o un *Bundle*. Un messaggio OSC consiste in un *Address Pattern* (tipologia di messaggio) seguito da un *Type Tag String* ed infine seguito da uno o più argomenti. Un *Address Pattern* è una stringa di dati che inizia per “/”. Un *Type Tag String* è una stringa di dati che inizia con il carattere “,” seguito da una sequenza di caratteri che corrispondono esattamente alla sequenza di *OSC Arguments* (argomenti) in un determinato messaggio. Ogni carattere dopo la virgola è chiamato *Type Tag*.

OSC Type Tag	Tipologia di argomento corrispondente
i	int32
f	float32
s	OSC-string
b	OSC-blob

Tabella 1: corrispondenza tra ogni OSC Type Tag ed il corrispondente OSC Argument:

Molte applicazioni comunicano tra istanze con l'aggiunta di argomenti non standard oltre a quelli elencati nella tabella sopra. Alle applicazioni OSC non è richiesto di riconoscere queste tipologie di dati, infatti scarta tutti i dati di cui non riconosce l'*OSC Type Tag*. Un'applicazione che fa uso di elementi aggiuntivi deve riconoscere anche i *Type Tag* contenuti nella tabella appena sotto.

OSC Type Tag	Tipologia di argomento corrispondente
h	64 bit big-endian two's complement integer
t	OSC-timetag
d	64 bit ("double") IEEE 754 numero floating point
S	Un modo alternativo per rappresentare un OSC-string (per esempio per differenziare "simboli" da "stringhe")
c	Carattere ASCII, mandato come 32 bits
r	colore RGBA 32 bit
m	Messaggio MIDI di 4 bytes. Bytes da MSB a LSB sono: id porta, status byte, dato1, dato2
T	Vero. Nessun byte allocato nell'argomento del dato.
F	Falso. Nessun byte allocato nell'argomento del dato.
N	Nil. Nessun byte allocato nell'argomento del dato.
I	Infinitum. Nessun byte allocato nell'argomento del dato.
[Indica l'inizio di un array. I tags seguenti sono dati interni all'array fino a quando una parentesi non viene chiusa.
]	Indica la fine dell'array.

Tabella 2: OSC Type Tags che devono essere utilizzati per alcuni argomenti non standard

Ogni server ha un lista di metodi i quali sono potenziali destinazioni del messaggio OSC ricevuto dal server e corrisponde a ciascuno dei punti di controllo che l'applicazione rende disponibili. Invocare un metodo equivale alla procedura di chiamare, ovvero fornire il metodo con gli argomenti e causando l'effetto della sostituzione del metodo.

I metodi di un server sono organizzati in una struttura ad albero chiamata *OSC Address Space*. Le foglie di quest'albero sono i metodi e le braccia sono chiamate *OSC Containers* (contenitore). L'*OSC Address Space* (spazio d'indirizzo) di un server può essere dinamico, ovvero il suo contenuto e la sua forma può variare nel tempo. Ogni metodo ed ogni contenitore deve avere un nome con simboli diversi da quelli della tabella sottostante:

Carattere	Nome	Codice Ascii (decimale)
''	Spazio	32
#	Number sign	35
*	Asterisco	42
,	Virgola	44
/	Barra	47
?	Punto di domanda	63
[Parentesi quadra aperta	91
]	Parentesi quadra chiusa	93
{	Parentesi graffa aperta	123
}	Parentesi graffa chiusa	125

Tabella 3: caratteri non permessi nei metodi OSC e/o contenitori OSC

L'indirizzo di un metodo è un nome simbolico che dà il percorso completo ad un metodo in uno spazio d'indirizzo, partendo dalla radice dell'albero. Un indirizzo del metodo inizia con il carattere “/” seguito dal nome di tutti i contenitori, in ordine, il percorso dall'origine del messaggio al metodo, separato da una barra segue il nome del metodo. La sintassi di un indirizzo è stata scelta per corrispondere alla sintassi URLs. Quando un server riceve un messaggio, esso può invocare il proprio spazio d'indirizzo del suo metodo, basato sulla tipologia del messaggio, nel messaggio OSC. Questo processo è chiamato *dispatching* (dispaccio) del messaggio nel metodo che corrisponde alla sua tipologia d'indirizzo. Tutti i metodi corrispondenti sono invocati con gli stessi argomenti, vale a dire, l'argomento contenuto nel messaggio. La sintassi di un indirizzo o di un tipologia d'indirizzo è racchiusa tra due barre “/” e l'ultima parte del messaggio non è seguita da nessuna barra. Un messaggio ricevuto può essere distribuito a tutti i metodi nei correnti spazi d'indirizzo il cui indirizzo corrisponde alla stessa tipologia d'indirizzo del messaggio.

Una tipologia del messaggio OSC corrisponde all'indirizzo OSC se:

- l'indirizzo del messaggio e la tipologia d'indirizzo contengono lo stesso numero di parti

- ogni parte della tipologia d'indirizzo corrisponde alla relativa parte dell'indirizzo OSC

Una tipologia d'indirizzo di un messaggio corrisponde all'indirizzo del messaggio se tutti i caratteri che compongono la tipologia d'indirizzo del messaggio corrispondono a tutti i caratteri che compongono l'indirizzo del messaggio. Un server deve avere accesso alla corretta rappresentazione del tempo assoluto corrente in quanto il protocollo OSC non provvede a meccanismi di sincronismo o di produzione di un clock. Quando si riceve un pacchetto contenente solo il messaggio, il server può invocare immediatamente il metodo corrispondente o appena è possibile dopo la ricezione del pacchetto. Se il tempo rappresentato dal *Time Tag* è precedente o uguale al tempo corrente, il server può invocare il metodo immediatamente (salvo che l'utente abbia istruito il server a scartare i messaggi che arrivano troppo tardi). Altrimenti il *Time Tag* rappresenta un tempo futuro ed il server deve salvare il *bundle* fino a quando è specificato, quindi invocare l'appropriato metodo. I *Time Tag* sono rappresentati da un numero a 64 bit a virgola fissa. I primi 32 bit specificano la parte frazionaria di un secondo ad una precisione di 200 picosecondi. Questa è la rappresentazione usata da Internet NTP. Se il valore del *Time Tag* consiste in 63 bit a 0 seguiti da un ultimo bit significativo alla fine, è un caso speciale di *Time Tag* e vuole dire “immediatamente”. Quando una tipologia d'indirizzo è spedita a molti metodi, l'ordine con il quale la corrispondenza dei metodi viene invocata non è specificata; invece quando un *bundle* contiene un messaggio multiplo, l'insieme di metodi corrispondenti al messaggio devono essere invocati nello stesso ordine di quello dei messaggi nel pacchetto.

1.4.2 OSC e Max/MSP

Il protocollo OSC e la comunicazione UDP sono stati implementati nell'ambiente di sviluppo Max/MSP dal CNMAT. Le implementazioni più sicure della comunicazione UDP sono possibili mediante gli external object *udpsend* e *udpreceive* creati dal CNMAT. Successivamente sono stati implementati altri due external object: *opensoundcontrol* e *OSC-route*. L'external Object *opensoundcontrol* prende in ingresso i messaggi in arrivo mediante la comunicazione UDP ne traduce il messaggio OSC e restituisce in uscita i dati del messaggio OSC seguendo il di liste di dati utilizzate in Max/MSP, *OSC-route* invece è molto simile all'oggetto di *route* di Max/MSP, ovvero esegue un processo di matching dei dati e tira fuori in uscita solo i dati desiderati.

1.4.3 OSC e Processing

L'ambiente di programmazione Processing struttura il protocollo OSC mediante la libreria *oscP5* sviluppata da Andreas Schlegel. La libreria *oscP5* supporta le connessioni TCP, UDP e Multicast. Le operazioni del network sono gestite dal pacchetto *netP5* che può anche essere usato da solo per operazioni di network senza il bisogno d'implementare il protocollo OSC. Per installare la libreria *oscP5* è possibile utilizzare la stessa procedura utilizzata per l'installazione della libreria *TUIO*, ovvero copiare tutto il contenuto della cartella all'interno della cartella dello sketch di lavoro ed istanziandola all'interno dello sketch (questa la denominazione di un programma nel gergo di Processing) mediante i comandi:

```
import oscP5.*;
import netP5.*;
OscP5 oscP5;
NetAddress myRemoteLocation;
```

Proprio come per la libreria TUIO dopo aver copiato ed importato la libreria nello sketch di lavoro, dobbiamo creare un'istanza del oscP5, creando appunto un costruttore per utilizzare i suoi argomenti.

```
oscP5 = new OscP5(this,12000);  
myRemoteLocation = new NetAddress("127.0.0.1",12000);
```

La variabile oscP5 stabilisce su quale porta deve mettersi in ascolto mentre la variabile myRemoteLocation stabilisce l'IP e la porta alla quale deve mandare i messaggi OSC.

I metodi utili per sfruttare questa libreria sono i seguenti:

- *OscMessage myMessage = new OscMessage("/test")* crea un messaggio con l'intestazione chiamato "test"
- *myMessage.add(123)* aggiunge al messaggio "test" i numeri 1, 2 e 3. È possibile associare anche delle variabili
- *oscP5.send(myMessage, myRemoteLocation)* Manda il messaggio appena creato con all'interno tutto il suo contenuto.

I messaggi in arrivo possono essere sfruttati mediante il metodo:

- *theOscMessage.addrPattern()* per ottenere il pattern del messaggio
- *theOscMessage.typeTag()* per ottenere il contenuto del messaggio

1.4.4 TUIO

Il protocollo TUIO è il protocollo utilizzato per le Table-Top Tangible User Interface. Esso è stato sviluppato partendo dal protocollo OSC (Open Sound Control) che favorisce l'implementazione in vari ambienti di sviluppo come Java, C++, PureData, Max/MSP, SuperCollider, Flash e Processing.

Il suo scopo è quello di interfacciare il tavolo e tradurre posizione, velocità ed orientamento tipo, e quale oggetto è sul tavolo.

Dettagli implementativi

Il protocollo TUIO è definito da due classi principali di messaggi: i messaggi *set* e quelli *alive*. I messaggi *set* comunicano le informazioni relative alla posizione, l'orientamento e lo stato di ogni singolo oggetto, mentre i messaggi *alive* comunicano la lista di oggetti presenti sul tavolo e quando sono stati utilizzati. Tutti questi messaggi sono riconducibili ad un unico oggetto mediante un ID che è possibile assegnare ad ognuno di loro.

Per riassumere possiamo dire che:

- i parametri riferiti agli oggetti vengono spediti dopo aver cambiato il loro stato usando i messaggi *set*;
- quando un oggetto viene aggiunto o rimosso viene mandato un messaggio *alive*;
- il client deduce se un oggetto è stato aggiunto o rimosso mediante i messaggi *alive*;
- i messaggi *fseq* (frame sequence) associano ad ogni ID tutti i parametri mediante i messaggi *set* e *alive*;

Efficienza ed affidabilità

Per ottenere una bassa latenza il protocollo TUIO invia dati mediante il trasporto UDP (User Data Protocol). Malgrado l'utilizzo del protocollo TUIO potrebbe causare la perdita di pacchetti di dati durante la trasmissione e proprio per questo che al suo interno troviamo delle informazioni ridondanti. In alternativa è possibile utilizzare una connessione TCP (Transmission Control Protocol) a discapito di una maggiore latenza. Dunque, ragioni d'efficienza è stato scelto di trasferire i dati mediante il trasporto UDP includendo una ridondanza dei messaggi *alive* per correggere eventuali perdite di pacchetti di dati.

Le informazioni ridondanti, inviate a bassa frequenza, includono un subset di messaggi di oggetti che non variano il proprio stato dopo un aggiornamento del sistema. Quest'ultimo viene aggiornato ciclicamente ed ogni volta vengono utilizzati dei valori che s'incrementano con il tempo e questo valore è lo stesso per tutto il pacchetto di dati da inviare nello stesso momento. Questo permette di comunicare al *client* l'ordine dei pacchetti di dati da inviare.

Formato del Protocollo

Il protocollo TUO è implementato mediante il protocollo OSC (Open Sound Controll) e segue una sintassi generale ed è composto dai seguenti messaggi:

/tuio/[profileName] set sessionID [parameterList]
/tuio/[profileName] alive [list of archive sessionIDs]
/tuio/[profileName] fseq (int32)

Parametri

I parametri di cui tiene conto il protocollo TUO sono ID, posizione e angolo che sono direttamente calcolati da recTIVision (vedi paragrafo 3.2 I fiducial e il processo di tracking video), altri come la velocità e l'accelerazione sono il risultato dell'elaborazione dei primi. Il calcolo di questi due parametri a basso livello permette una maggiore efficienza di tutto il sistema computazionale che segue.

Ogni oggetto deve avere un ID unico. Ad ogni frame viene aggiornata la *fseq* e se allo stesso frame vengono rilevati due o più nuovi oggetti essi vengono associati alla stessa classID.

s	sessionID, temporany object, int32
i	classID, fiducial ID number, int32
x, y, z	Posizione, range 0 1, float32
a, b, c	Angolo, float32, range 0 2PI
X, Y, Z	Movimento del vettore (velocità e direzione), float32
A, B, C	Rotazione del vettore (velocità di rotazione & direzione, float32
m	Accelerazione, float32
r	Accelerazione di rotazione, float32
P	Parametro libero, definito dall'hader del pacchetto di dati OSC

Tabella 4: semantica dei messaggi set inviati;

Profili

Come possiamo ben immaginare per ogni tipo d'interfaccia la lista di dati trasmessa è una lista ben precisa.

- Superfici interattive 2D:

/tuio/2Dobj set s i x y a X Y A m r

/tuio/2Dcur set s x y X Y m

Superfici interattive 2.5D

/tuio/25Dobj set s i x y z a X Y Z A m r

/tuio/25Dcur set s x y z X Y Z m

Superfici interattive 3D

/tuio/3Dobj set s i x y z a b c X Y Z A B C m r

/tuio/3Dcur set s x y z X Y Z m

Profilo Raw

/tuio/raw_[profileName]

/tuio/raw_dtouch set i x y a

Profilo Custom

/tuio/_[formatString]

/tuio/_sixyP set s i x y 0.57

1.4.5 TUO e Processing

Processing frutta la libreria TUO Processing sviluppata dagli stessi sviluppatori di reacTIVision e scaricabile gratuitamente sul sito:

<http://reactivision.sourceforge.net/#files>

Per installare la libreria TUO in Processing bisogna copiare l'intero contenuto della cartella scaricata in una cartella rinominata TUO in /libraries/sketchbook. Per utilizzare la libreria si dovrà inserire all'interno dello sketch di lavoro l'istruzione "import TUO.*;". Dopo aver copiato ed importato la libreria nello sketch di lavoro, dobbiamo creare un'istanza del TuioProcessing client, creando appunto un costruttore per utilizzare i suoi argomenti. Il TuioProcessing client si mette immediatamente in ascolto dei messaggi TUO in arrivo e genera degli eventi ad alto livello basati sui movimenti degli oggetti e dei cursori. L'istruzione è:

TuioProcessing TuioClient = new TuioProcessing(this);

Per ricevere e sfruttare messaggi TUO dobbiamo richiamare i seguenti metodi:

- *addTuioObject(TuioObject tobj)* quando un oggetto viene poggiato sulla T.U.I.
- *removeTuioObject(TuioObject tobj)* quando un oggetto viene rimosso dalla T.U.I.
- *updateTuioObject(TuioObject tobj)* quando un oggetto viene mosso sulla T.U.I.
- *addTuioCursor(TuioCursor tcur)* quando si poggia un cursore sulla T.U.I.
- *removeTuioCursor(TuioCursor tcur)* quando un cursore viene rimosso dalla T.U.I.
- *updateTuioCursor(TuioCursor tcur)* quando un cursore viene mosso sulla T.U.I.
- *refresh(TuioTime bundleTime)* aggiorna tutti i valori di tutti i fiducial sulla T.U.I.

Ogni *TuioObject* (fiducial) o *TuioCursor* (dita) vengono definiti da un unico *SessionID* che viene mantenuto per tutta la sua vita sul tavolo (da quando viene aggiunto alla sua rimozione, successivamente viene azzerato). Il *SymbolID* invece corrisponde al “nome” definitivo del fiducial dato mediante un numero. Al contrario il *CursorID* del *TuioCursor* viene assegnato in base alla sequenza d'appoggio degli stessi sulla T.U.I.. È possibile richiamare l'ID mediante i metodi *getSessionID()*, *getSymbolID()* o *getCursorID()*. Tutte le caratteristiche dei *TuioObject* e dei *TuioCursor* vengono aggiornate automaticamente ad ogni aggiornamento di ciascun *TuioObject* o *TuioCursor* i valori corrispondono sempre alla stessa istanza per tutta la loro vita sul tavolo.

Tutti gli attributi dei *TuioObject* e dei *TuioCursor* sono impacchettati all'interno del messaggio TUO e si può accedere a loro con dei metodi come *getX()* (per ottenere la coordinata X), *getY()* (per ottenere la coordinata Y) e *getAngle()* (per ottenere l'angolo). Esistono anche metodi per rilevare il valore della velocità e dello spostamento. I *TuioObject* e i *TuioCursor* hanno anche molti altri metodi per calcolare le distanze e gli angoli tra oggetti.

In alternativa la classe *TuioProcessing* contiene molti metodi per il polling dello stato di ogni singolo oggetto e cursore. Ci sono anche metodi che danno come risultato la lista di tutti i dati che si riferiscono all'oggetto o cursore:

- *getTuioObjects()* restituisce un vettore di tutti i *TuioObject* presenti sulla T.U.I.

- `getTuioCursors()` restituisce un vettore di tutti i *TuioCursor* presenti sulla T.U.I.
- `getTuioObject(long s_id)` restituisce un *TuioObject* o nulla in base alla presenza sulla T.U.I.
- `getTuioCursor(long s_id)` restituisce un *TuioCursor* o nulla in base alla presenza sulla T.U.I.

Questa è libreria è gratuita e può essere distribuito e/o modificata sotto i termini della GNU General Public License ed è stato pubblicato dalla Free Software Foundation. La libreria è basata sul reference TUIO Java ed include la libreria OpenSound Control.

1.5 Controllo e Gesto

Il controllo della Metis consiste nel gesto da applicare su di essa è stato oggetto di studio per stabilire un “catalogo di gesti” ed una sua connessione chiara ed esplicita con l'elaborazione audio corrispondente mediante il supporto dell'elaborazione di uno sfondo grafico.

Aggiunta o rimozione di un fiducial

Nel mondo reale l'introduzione o la rimozione di un elemento all'interno di un sistema di qualsiasi natura è un momento significativo, in quanto comporta la variazione del sistema all'interno del quale si viene immesso un elemento. Allo stesso modo l'aggiunta o la rimozione di un fiducial nella superficie d'azione della Metis comporta una sua variazione rappresentata da una variazione dello sfondo grafico.

Spostamento e rotazione di un fiducial

Lo spostamento e la rotazione di un fiducial sono ulteriori gesti molto importanti, e in questo caso ancor più stretta dev'essere la relazione tra il gesto e l'elaborazione sonora che ne segue. Il verso verso del fiducial nella maggior parte dei casi va a determinare un livello d'intensità sonora in uscita dal nostro sistema di diffusione e o una “quantità” d'elaborazione sonora. Il progressivo aumento d'intensità o di elaborazione verrà accompagnato da un amento della luminosità del fiducial, più alta

sarà l'intensità della diffusione sonora e/o la quantità di elaborazione del segnale e più luminoso sarà il fiducial. Riprendendo l'analogia con un qualsiasi sistema appartenente al mondo reale una variazione della posizione di un elemento al suo interno è altrettanto significativa ed in grado di variare lo stato del sistema stesso. Dal punto di vista grafico la sua variazione di posizione varierà lo sfondo grafico elaborandolo, allo stesso modo sarà responsabile di una variazione dell'elaborazione applicata sul segnale audio sia dal punto nella disposizione spaziale che nel timbro.

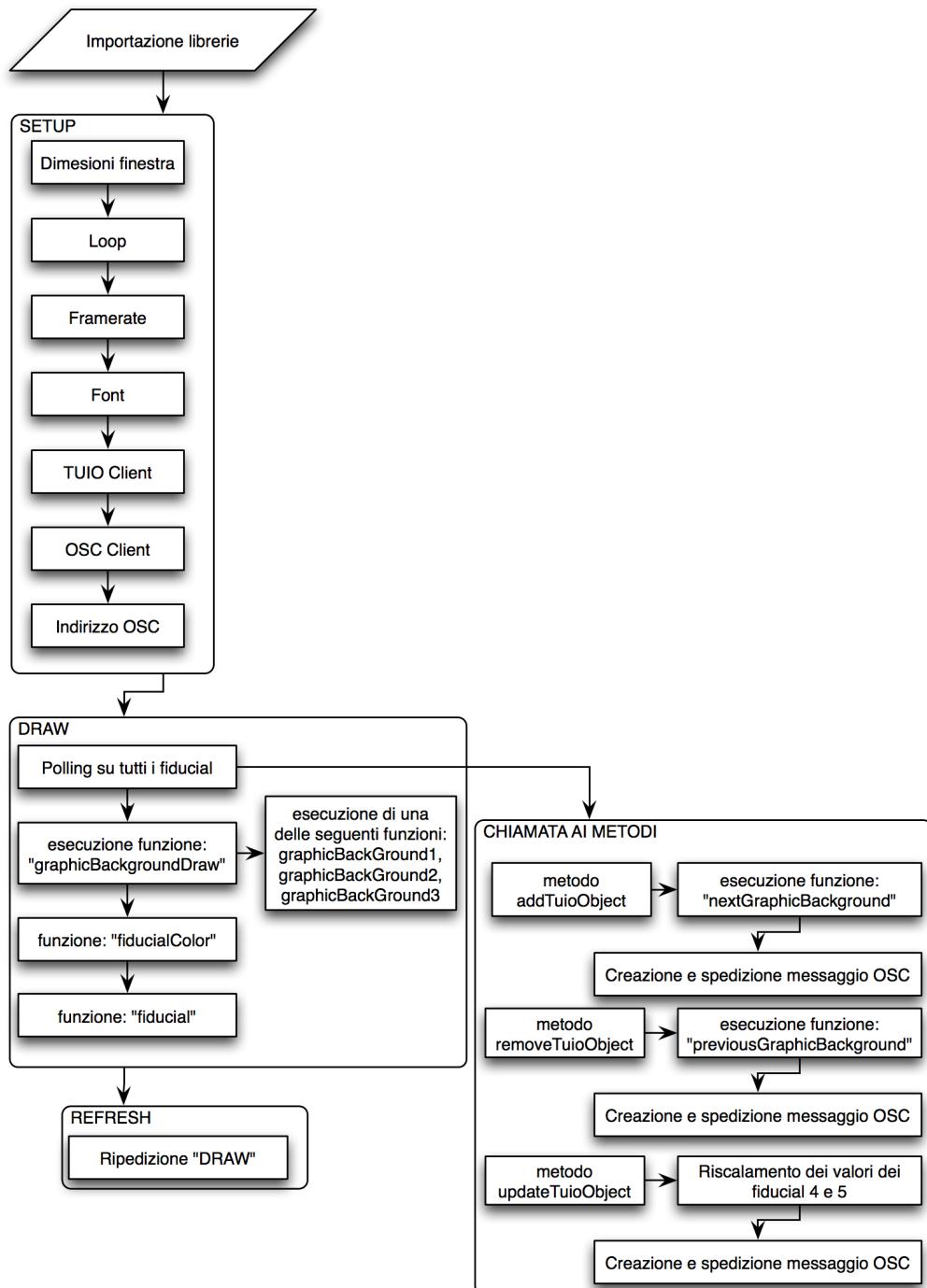
1.6 Elaborazione grafica

1.6.1 Processing

Processing è un ambiente di sviluppo open source che sfrutta un linguaggio ad oggetti per sviluppare elaborazioni grafiche. Inizialmente è stato sviluppato per creare una collezione di software e per insegnare i fondamentali della programmazione all'interno di un contesto visivo, successivamente si è evoluto anche come strumento per generare dei lavori di tipo professionale. Oggi ci sono decine di migliaia di studenti, artisti, designer, ricercatori e appassionati che usano Processing per imparare, progettare e produrre. Per la corretta comprensione del funzionamento dell'algoritmo è bene tener presente che Processing genera un'applicazione formata da più programmi (sketch) i quali possono contenere esecuzioni di algoritmi descritti in maniera sequenziale o sotto forma di funzioni e classi. L'applicazione che verrà generata e successivamente "lanciata" produce un loop di se stessa, il quale rende ciclici tutti i processi contenuti nell'applicazione.

1.6.2 Algoritmo

L'algoritmo generato mediante processing ha una struttura sequenziale da cui possiamo estrapolare due grandi blocchi principali: il *setup* e *draw*, ed altri due blocchi che aprono e chiudono lo sketch: l'*importazione delle librerie* e la funzione di *refresh*.



Schema 2: Struttura algoritmo

Il nostro sketch principale inizia con l'importazione delle librerie la quale ha luogo mediante l'esecuzione delle seguenti righe di codice:

```
import TUIO.*; // Importazione libreria "TUIO"
import oscP5.*; // Importazione libreria "oscP5"
import netP5.*; // Importazione libreria "netP5"
OscP5 oscP5; // Inizializzazione libreria "oscP5"
NetAddress myRemoteLocation; // Inizializzazione indirizzo del messaggio OSC
TuioProcessing tuioClient; // Inizializzazione TUIO Client
Sketch 1: importazione librerie
```

La seconda parte dello sketch è occupato dalla funzione di *setup* dove sono contenute tutte le regole principali dell'algoritmo, come le dimensioni della finestra, il framerate (clock interno, ossia la frequenza con cui si ripete il loop), la generazione del ciclo di loop, il font da adottare per i caratteri visibili all'interno della finestra, i dati per la creazione di un TUIO e OSC Client e per la creazione di una porta OSC a cui inviare messaggi.

```
void setup()
{
    size(1400,1050); // Dimensioni finestra
    loop(); // Ciclo di loop
    frameRate(30); // Impostazione frame rate a 30 frame al secondo
    hint(ENABLE_NATIVE_FONTS); // Abilitazione utilizzo del font
    font = createFont("Arial", 18); // Scelta del font dei caratteri
    scale_factor = height/table_size; // Fattore di riscalamento
    tuioClient = new TuioProcessing(this); // impostazione del TUIO Client
    oscP5 = new OscP5(this,7400); // Impostazione della porta a cui mandare messaggi OSC
    myRemoteLocation = new NetAddress("127.0.0.1",7400); // Impostazione della porta a cui mandare messaggi OSC
}
```

Sketch 2: setup

Nella terza parte dell'algoritmo, il *draw*, troviamo l'implementazione di tutti i processi che porteranno alla generazione degli sfondi grafici da proiettare sulla Metis. Il *draw* è la funzione dell'algoritmo che viene messa in loop e ripetuta ciclicamente. Tutti i processi contenuti al suo interno saranno ripetuti ciclicamente con un ordine scandito dalla sua organizzazione interna. Dunque tutte le funzioni e classi contenute al suo interno verrano eseguite ciclicamente, scandite dall'azione dei metodi richiamati di volta in volta a seguito della nostra interazione sulla Metis mediante un fiducial.

```

void draw()
{
    background(1); // Sfondo nero
    graphicBackgroundDraw(); // Funzione "graphicBackgroundDraw"
    Vector tuioObjectList = tuioClient.getTuioObjects(); // Polling su tutti i fiducial
    for (int i=0;i<tuioObjectList.size();i++) {
        TuioObject tobj = (TuioObject)tuioObjectList.elementAt(i);
        fiducialColor(tobj.getSymbolID(), tobj.getAngle()); // Funzione "fiducialColor"
        fiducial(tobj.getSymbolID(), tobj.getScreenX(width), tobj.getScreenY(height), tobj.getAngle(), fiducialColorR, fiducialColorG, fiducialColorB); // Funzione "fiducial"
    }
}

```

Sketch 3: draw

In ultimo, non per importanza, vi è la funzione *refresh* la quale da il comando di eseguire di nuovo la funzione *draw* per un ulteriore aggiornamento di tutti i dati.

```

void refresh(TuioTime bundleTime) {
    redraw(); // Disegna ogni qual volta che arriva un pacchetto di messaggi OSC
}

```

Sketch 4: refresh

Di seguito andremo ad analizzare tutti gli sketch inglobati dalla funzione *draw*. All'inizio del *draw* troviamo l'istruzione la quale esegue il polling su tutti i fiducial ovvero una verifica, un monitoraggio ciclico dei sottoelencati dati provenienti dal processo di tracking

- stato del fiducial: se è stato appena appoggiato, se è stato rimosso o se si sta muovendo sulla superficie. Da questa verifica ne dipendono la chiamata dei metodi (*addTuioObject*, *removeTuioObject* e *updateTuioObject*);
- posizione del fiducial espressa in coordinate cartesiane in riferimento all'asse X e Y della Metis
- il verso del fiducial calcolato in angoli radianti, il quale può variare da 0 a 2π

Una volta effettuato il *polling* su tutti i fiducial ed estratti i dati dei fiducial vengono chiamati i metodi coinvolti e di seguito le funzioni da loro inglobate.

Per prima analizziamo le funzioni che vengono attivate mediante qualsiasi tipo d'interazione del fiducial con la Metis: *fiducial* e *fiducialColor*.

La funzione *fiducial* genera un'ellisse con posizione e dimensioni di circa 2 millimetri più grande delle dimensioni del fiducial “fisico” con un colore variabile mediante il numero dell'ID del fiducial ed il suo verso.

```

float object_size = 118; // Dimensione Fiducial
float table_size = 760; // Dimensioni sfondo grafico
float scale_factor = 1; // Fattore di riscalamento
PFont font; // Font del testo del fiducial

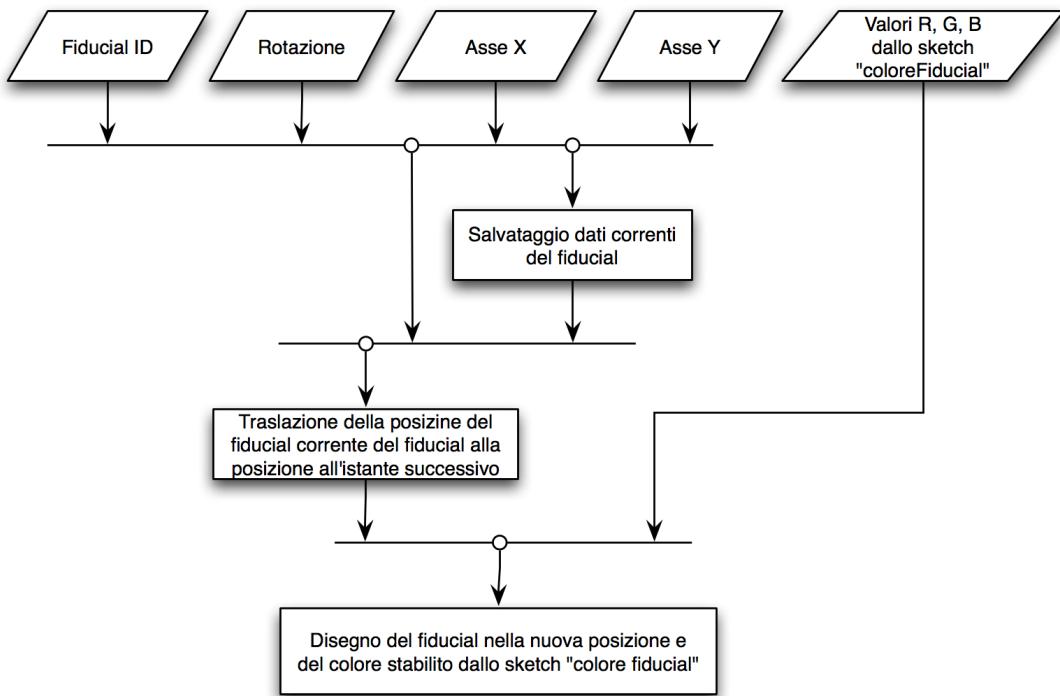
void fiducial(int ID, float X, float Y, float rotation, int R, int G,int B){ // Valori in ingresso alla funzione

    textFont(font,18*scale_factor); // Dimensioni del font
    float obj_size = object_size*scale_factor; // Dimensioni del fiducial
    pushMatrix(); // Salvataggio dei dati del al frame corrente
    translate(X,Y); // Traslazione del fiducial
    rotate(rotation); // Rotazione del fiducial
    stroke(R, G, B); // Scelta del colore del margine del fiducial
    fill(R, G, B); // Scelta del colore del fiducial
    ellipse(0,0,obj_size,obj_size); // Generazione ellisse
    strokeWeight(3); // Dimensioni della freccia che indica il verso
    stroke(255*rotation); // Colore del margine della freccia che indica il verso
    fill(255*rotation); // Colore della freccia che indica il verso
    line(0, -obj_size/2, 0, 0); // Generazione freccia che indica il verso
    popMatrix(); // Nuove coordinate e verso del fiducial
    fill(255); // Colore del font
    text(""+ID, X+50, Y); // Scrittura dell'ID sul fiducial
}

```

Sketch 5: fiducial

Per prima cosa vengono captati tutti i dati del fiducial e se al frame corrente si trova in uno stato di addTuioObject viene disegnata un'ellisse appena sotto mentre se il suo stato equivale ad un updateTuioObject, ovvero si sta aggiornando la sua posizione e/o verso i dati del fiducial vengono momentaneamente salvati mediante la funzione *pushMatrix* l'ellisse viene traslato e ruotato in base alla posizione del fiducial e/o verso al frame successivo e successivamente viene proiettato sulla superficie della Metis in base alla nuova posizione e/o verso. Nelle righe di codice di questa operazione troviamo anche tutte le istruzioni che si occupano di estrarre l'ID del fiducial e di “scriverlo” all'interno dell'ellisse.



Schema 3: fiducial

Il colore del fiducial viene stabilito mediante la funzione *fiducialColor*. Questa funzione esegue un test sull'ID del fiducial ed in base al suo ID assegna un colore diverso all'ellisse da disegnare al di sotto del fiducial in valori R, G, B e la sua intensità sarà regolata dal verso del fiducial. In base all'ID del fiducial assegnamo un valore alle componenti di colore rosso, verde e blu il quale è il risultato di una costante o del prodotto di una costante per una variabile che dipende verso del fiducial. Essendo il verso del fiducial espresso in radianti mentre noi necessitiamo di un valore che varia tra 0 ed 1, verrà prima scalato per un valore che varia tra 0 ed un quando il fiducial viene mosso tra 0 e π e tra 1 e 0 quando il valore del verso del fiducial oscilla tra π e 2π . In modo tale da ottenere un'intensità del colore dell'ellisse che illumina il fiducial dipendente dal verso.

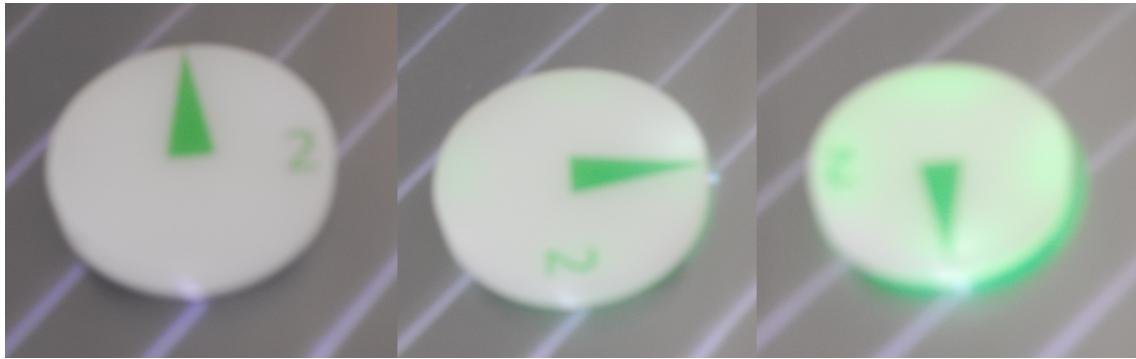


Illustrazione 33: fiducial nelle tre posizioni: $0/2\pi$, $\pi/2$, π ;

```

int fiducialColorR, fiducialColorG, fiducialColorB = 0;

void fiducialColor (int ID, float rotation){ // Dati in ingresso alla funzione

    if(rotation<PI) rotation=map(rotation, 0., PI, 0., 1.); // Valore riscalato per controllare l'intensità del colore del fiducial
    else rotation=map(rotation, PI, 2*PI, 1., 0.); // Valore riscalato per controllare l'intensità del colore del fiducial

    if( ID == 0){ // Test sul numero dell'ID del fiducial
        fiducialColorR = 0; // Componente rossa del colore del fiducial
        fiducialColorG = int(255*rotation); // Componente verde del colore del fiducial
        fiducialColorB = int(255*rotation); // Componente blu del colore del fiducial
    }
    else if(ID == 1){ // Test sul numero dell'ID del fiducial
        fiducialColorR = int(200*rotation); // Componente rossa del colore del fiducial
        fiducialColorG = 0; // Componente verde del colore del fiducial
        fiducialColorB = 0; // Componente blu del colore del fiducial
    }
    else if( ID == 2){ // Test sul numero dell'ID del fiducial
        fiducialColorR = 0; // Componente rossa del colore del fiducial
        fiducialColorG = int(200*rotation); // Componente verde del colore del fiducial
        fiducialColorB = 0; // Componente blu del colore del fiducial
    }
    else if(ID == 3){ // Test sul numero dell'ID del fiducial
        fiducialColorR = 0; // Componente rossa del colore del fiducial
        fiducialColorG = 0; // Componente verde del colore del fiducial
        fiducialColorB = int(200*rotation); // Componente blu del colore del fiducial
    }
    else if( ID == 4){ // Test sul numero dell'ID del fiducial
        fiducialColorR = 0; // Componente rossa del colore del fiducial
        fiducialColorG = int(255); // Componente verde del colore del fiducial
        fiducialColorB = 0; // Componente blu del colore del fiducial
    }
    else if( ID == 5){ // Test sul numero dell'ID del fiducial
        fiducialColorR = 0; // Componente rossa del colore del fiducial
        fiducialColorG = 0; // Componente verde del colore del fiducial
        fiducialColorB = int(255); // Componente blu del colore del fiducial
    }
    else if( ID > 6 && ID <10){ // Test sul numero dell'ID del fiducial
        fiducialColorR = int(50*rotation); // Componente rossa del colore del fiducial
        fiducialColorG = int(50*rotation); // Componente verde del colore del fiducial
        fiducialColorB = int(50*rotation); // Componente blu del colore del fiducial
    }
}

```

Sketch 6: fiducialColor

Come abbiamo già detto ogni fiducial può o meno chiamare un metodo che a sua volta richiama una funzione incaricata di attivare il processo di creazione uno sfondo grafico ed inviare un messaggio OSC contenente tutti i dati relativi allo stato del fiducial a Max/MSP mediante il protocollo OSC. Se un fiducial viene aggiunto sul tavolo verrà

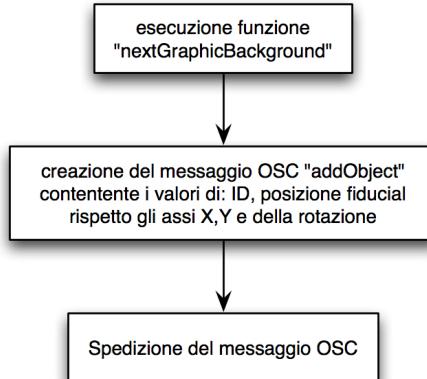
chiamato il metodo *addTuioObject* che chiama la funzione *nextGraphicBackGround* la quale prende in ingresso l'ID del fiducial appena aggiunto sul tavolo e successivamente manda il messaggio OSC a Max/MSP.

```
void addTuioObject(TuioObject tobj) { // Metodo richiamato quando viene aggiunto un fiducial

    nextGraphicBackground(tobj.getSymbolID());

    OscMessage addObject = new OscMessage("addObject"); // Creazione del messaggio OSC
    addObject.add(tobj.getSymbolID()); // ID
    addObject.add(tobj.getX()); // Coordinate asse X
    addObject.add(tobj.getY()); // Coordinate asse Y
    addObject.add(tobj.getVerso()); // Verso
    oscP5.send(addObject, myRemoteLocation); // Spedizione del messaggio OSC
}
```

Sketch 7: addTuioObject



Schema 4: addTuioObject

In base a quale fiducial ha chiamato il metodo *addTuioObject* verrà richiamata la funzione *nextGraphicBackGround* la quale varia lo stato delle variabili *graphicBackGround1*, *graphicBackGround2* o *graphicBackGround3*.

```
// Scelta degli sfondi grafici in caso di updateTuioObject
void nextGraphicBackground(int ID){

    // Attivazione sfondo grafico n.1
    if (ID == 1) {graphicBackGround1 = 1;
                  graphicBackGround2 = 0;
                  graphicBackGround3 = 0;}
    // Attivazione sfondo grafico n.2
    if (ID == 2) {graphicBackGround1 = 0;
                  graphicBackGround2 = 1;
                  graphicBackGround3 = 0;}
    // Attivazione sfondo grafico n.3
    if (ID == 3) {graphicBackGround1 = 0;
                  graphicBackGround2 = 0;
                  graphicBackGround3 = 1;}
}
```

Sketch 8: nextGraphicBackGround

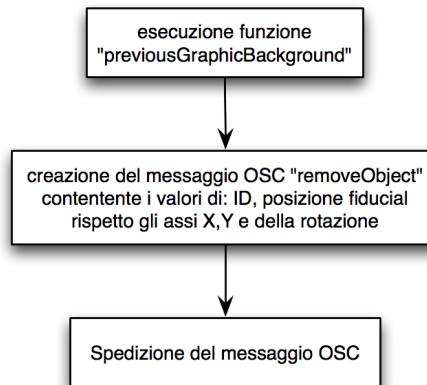
Se un fiducial viene rimosso il metodo richiamato è quello *removeTuioObject* che chiama la funzione *previousGraphicBackGround* la quale prende in ingresso l'ID del fiducial appena rimosso dal tavolo e successivamente manda il messaggio OSC relativo a Max/MSP.

```
void removeTuioObject(TuioObject tobj) { // Metodo richiamato quando viene rimosso un fiducial
    previousGraphicBackground(tobj.getSymbolID());

    OscMessage removeObject = new OscMessage("removeObject"); // Creazione del messaggio OSC
    removeObject.add(tobj.getSymbolID()); // ID
    removeObject.add(tobj.getX()); // Coordinate asse X
    removeObject.add(tobj.getY()); // Coordinate asse Y
    removeObject.add(tobj.getVerso()); // Verso
    oscP5.send(removeObject, myRemoteLocation); // Spedizione del messaggio OSC
}
```

Sketch 9: removeTuioObject

La funzione *removeTuioObject* come quella *addTuioObject* oltre ad inviare messaggi OSC verso Max/MSP richiama una funzione, in questo caso *previousGraphicBackground*, la quale è della stessa tipologia di quella *nextGraphicBackground*, ovvero provvede a modificare il valore delle variabili implicate nel processo di contenuto nella funzione *graphicBackDraw*.



Schema 5: removeTuioObject

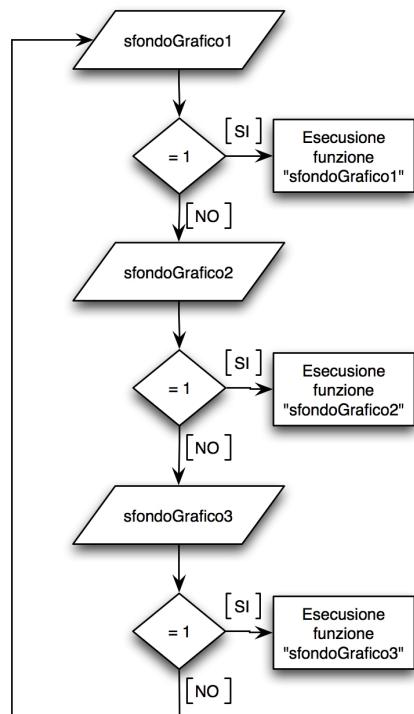
La funzione *graphicBackDraw* a cui sono direttamente collegate le funzioni *nextGraphicBackground* e *previousGraphicBackground* consiste in un test sul valore delle variabili *graphicBackGround1*, *graphicBackGround2* o *graphicBackGround3*. Quando una delle variabili appena citate è equivalente ad 1 allora verrà chiamata l'omonima funzione altrimenti si effettua di nuovo il test.

```

// Chiamata di una delle funzioni
void graphicBackGroundDraw(){
    if(graphicBackGround1 == 1) graphicBackGround1(); // Chiamata della funzione "graphicBackGround1"
    else if(graphicBackGround2 == 1) graphicBackGround2(); // Chiamata della funzione "graphicBackGround2"
    else if(graphicBackGround3 == 1) graphicBackGround3(); // Chiamata della funzione "graphicBackGround3"
}

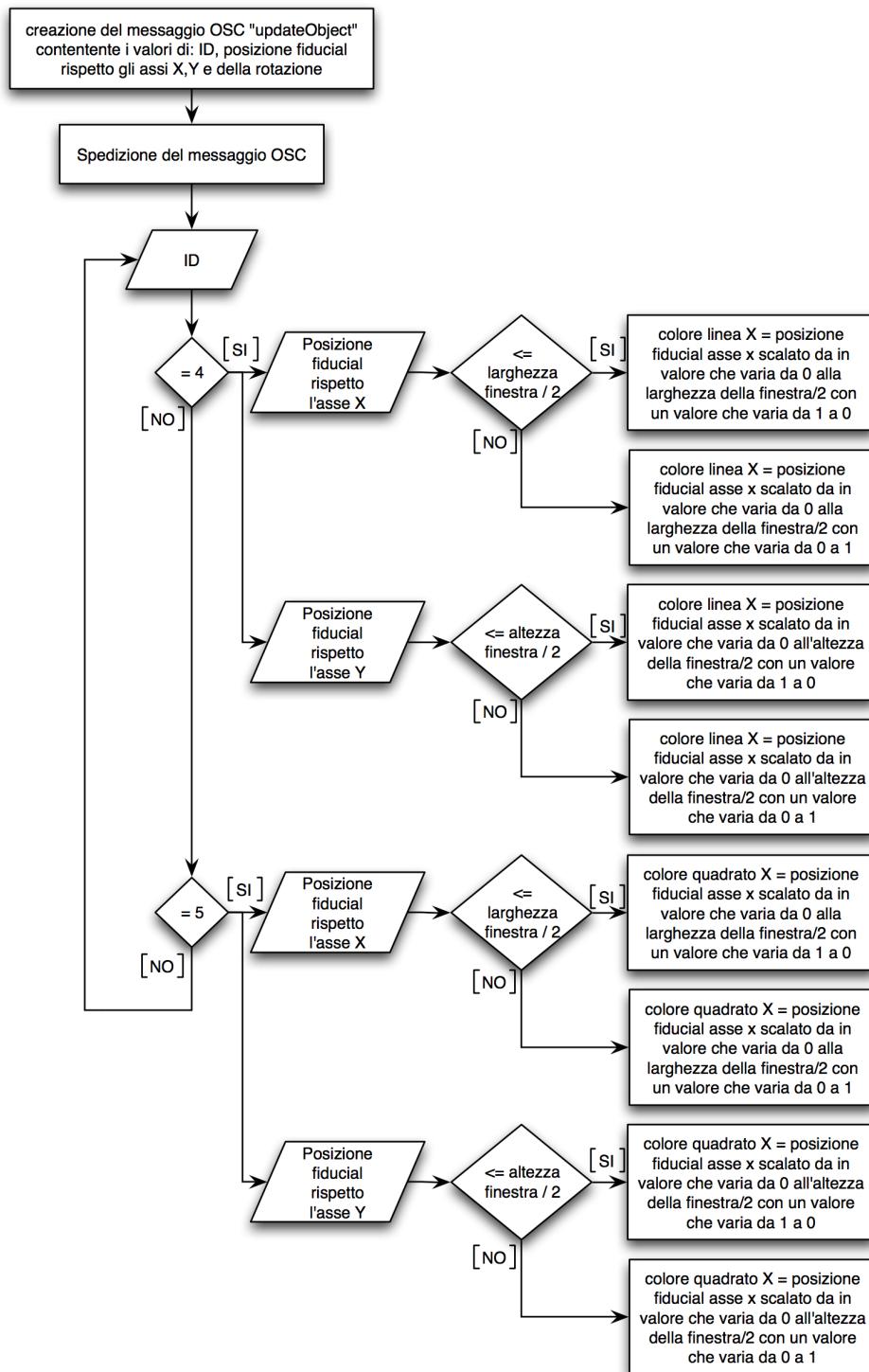
```

Sketch 10: graphicBackGroundDraw



Schema 6: graphicBackGroundDraw

Quando un fiducial viene mosso sul tavolo invece viene chiamato il metodo *updateTuioObject* il quale oltre a mandare il messaggio OSC a Max/MSP con di tutti i dati relativi al fiducial aggiornati, dopo un test sull'ID del fiducial si occupa di scalare i valori delle coordinate sugli assi X,Y relativi alla posizione del fiducial con ID pari a 4 e del fiducial con ID 5 per elaborare relativamente gli sfondi grafici 2 e 3.



Schema 7: updateTuioObject

```

void updateTuioObject (TuioObject tobj) { // Metodo richiamato quando viene aggiunto un fiducia
    if (tobj.getSymbolID()==4) { if (tobj.getScreenX(width)>=width/2.) linesColorX = map(tobj.getScreenX(width), 0, width/2, 1, 0); // Scalamento del colore delle linee sull'asse X dello sfondo grafico n. 2
        else if (tobj.getScreenX(width)<=width/2.) linesColorX = map(tobj.getScreenX(width), width/2, width, 0, 1); // Scalamento del colore delle linee sull'asse X dello sfondo grafico n. 2
        if (tobj.getScreenY(height)>=height/2.) linesColorY = map(tobj.getScreenY(height), 0, height/2, 1, 0); // Scalamento del colore delle linee sull'asse Y dello sfondo grafico n. 2
        else if (tobj.getScreenY(height)<=height/2.) linesColorY = map(tobj.getScreenY(height), height/2, height, 0, 1); } // Scalamento del colore delle linee sull'asse Y dello sfondo grafico n. 2
    if (tobj.getSymbolID()==5) { if (tobj.getScreenX(width)>=width/2.) squareColorX = map(tobj.getScreenX(width), 0, width/2, 1, 0); // Scalamento del colore delle linee sull'asse X dello sfondo grafico n. 3
        else if (tobj.getScreenX(width)<=width/2.) squareColorX = map(tobj.getScreenX(width), width/2, width, 0, 1); // Scalamento del colore delle linee sull'asse X dello sfondo grafico n. 3
        if (tobj.getScreenY(height)>=height/2.) squareColorY = map(tobj.getScreenY(height), 0, height/2, 1, 0); // Scalamento del colore delle linee sull'asse Y dello sfondo grafico n. 3
        else if (tobj.getScreenY(height)<=height/2.) squareColorY = map(tobj.getScreenY(height), height/2, height, 0, 1); } // Scalamento del colore delle linee sull'asse Y dello sfondo grafico n. 3
    }

OscMessage updateObject = new OscMessage("updateObject"); // Creazione del messaggio OSC
updateObject.addObject(tobj.getSymbolID()); // ID
updateObject.addObject(tobj.getId()); // Coordinate asse X
updateObject.addObject(tobj.getX()); // Coordinate asse Y
updateObject.addObject(tobj.getY()); // Verso
oscP5.send(updateObject, myremoteLocation); // Spedizione del messaggio OSC
}

```

Sketch 12: updateTuioObject

Il cuore dell'algoritmo è costituito dalle tre funzioni *graphicBackGround1*, *graphicBackGround2* e *graphicBackGround3*. Le tre funzioni permettono, mediante la sintesi grafica, la riproduzione di illusioni ottiche e la loro eventuale elaborazione in tempo reale. Il primo sfondo grafico *graphicBackGround1* riproduce l'illusione ottica generata dall'immagine appena sotto.

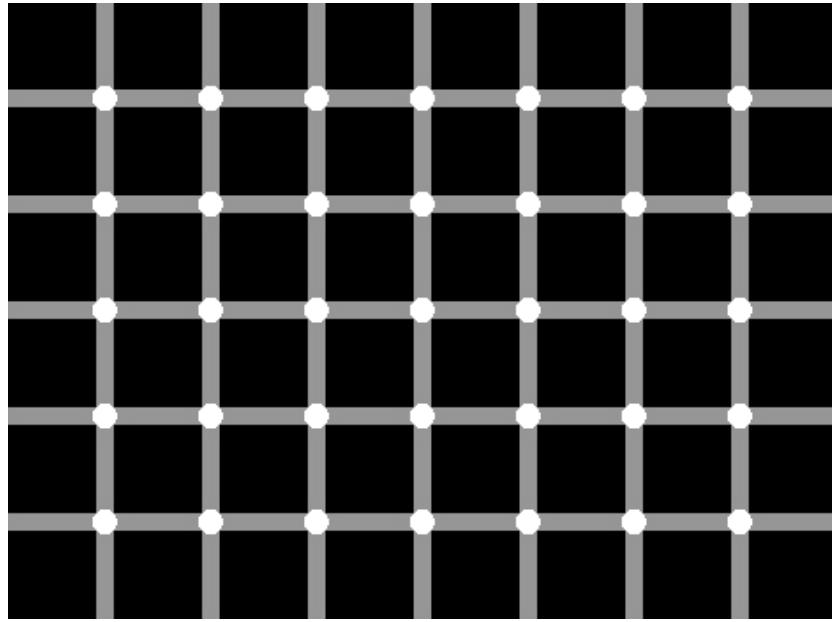


Illustrazione 34: sfondo grafico generato dalla funzione "graphicBackGround1"

Se si muovono gli occhi lungo il disegno le sfere vicine a quelle prese in considerazione sembrano nere. L'illusione è dovuta a come la luce colpisce la parte più sensibile e centrale della retina: la fovea. Quando lo sguardo comprende tutto il disegno gli aloni si vedono, se ti concentrvi su un singolo incrocio bianco la macchia scura non si vede più.

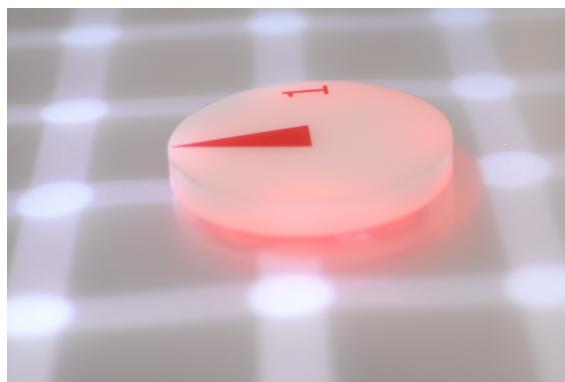
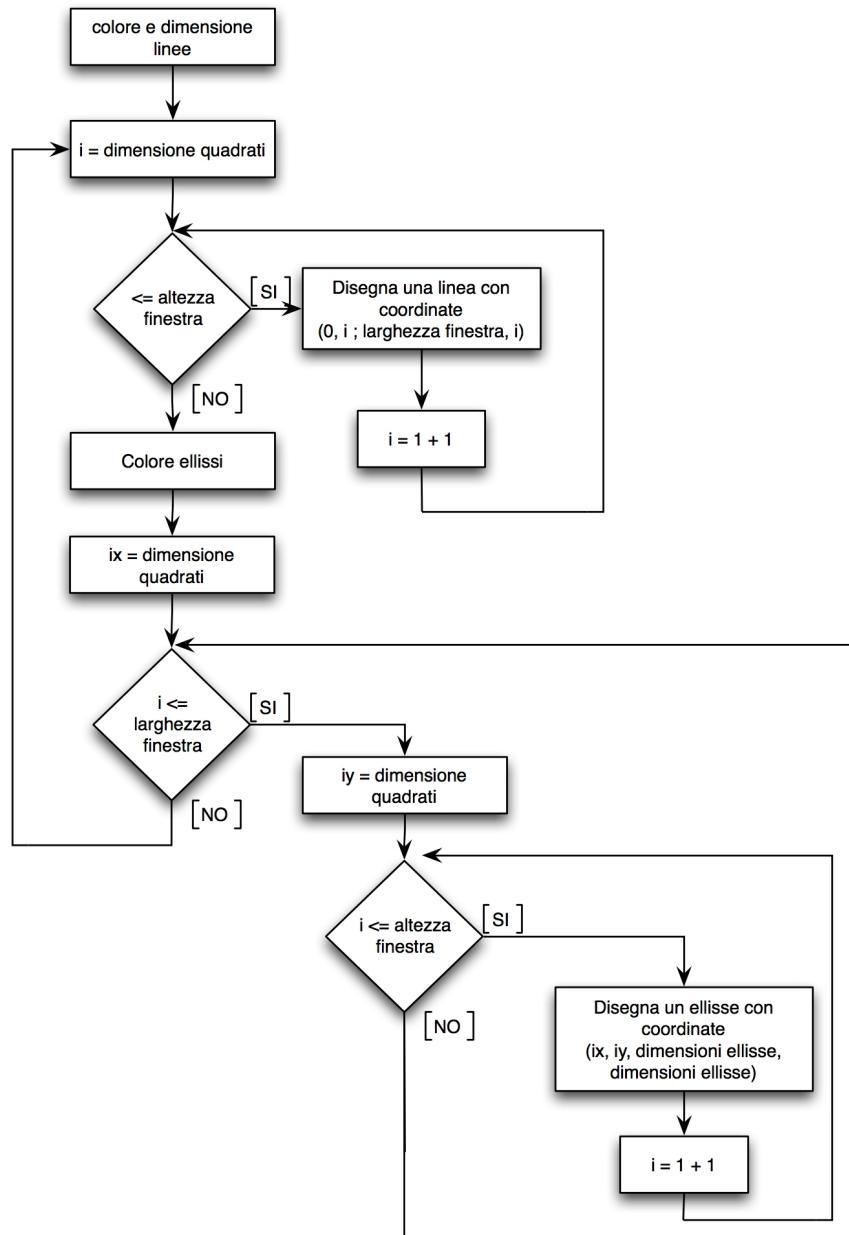


Illustrazione 35: fiducial con ID 1, il quale attiva lo sfondo grafico 1

L'algoritmo che costituisce la funzione `graphicBackGround1` è il seguente:



Schema 8: `graphicBackGround1`

che tradotto in linguaggio Processing equivale a dire:

```
int sizeEll = 30; // Dimensioni ellissi
int sizeSquare2 = 125; // Dimensioni quadrati
void graphicBackGround1(){

    // Processo di disegno delle linee
    stroke(153); // Colore linee
    strokeWeight(sizeEll); // Dimensione linee
    for(i = sizeSquare2; i<height; i=i+sizeSquare2) line(0, i, width, i); // Generazione linee orizzontali
    for(i = sizeSquare2; i<width; i=i+sizeSquare2) line(i, 0, i, height); // Generazione linee verticali

    // Processo di disegno degli ellissi
    strokeWeight(10); // Dimensione ellissi
    fill(255); // Colore ellissi (bianco)
    stroke(255); // Colore margine ellissi (bianco)
    for(int ix = sizeSquare2; ix<width; ix=ix+sizeSquare2){ // Generazione coordinate ellissi sull'asse X
        for(int iy = sizeSquare2; iy<height; iy=iy+sizeSquare2){ // Generazione coordinate ellissi sull'asse Y
            ellipse(ix, iy, sizeEll, sizeEll); // Generazione ellissi
        }
    }
}
```

Sketch 13: *graphicBackGround1*

L'algoritmo può essere diviso in due parti molto simili tra loro: una prima parte dedita a generare le linee mentre una seconda a generare le ellissi. Per ambedue le generazioni di elementi grafici sfruttiamo le potenzialità dei cicli for per determinare le coordinate delle linee e delle ellissi. Questo criterio verrà utilizzato anche per la generazione degli elementi costitutivi degli altri sfondi grafici.

Il secondo sfondo grafico generato mediante la funzione *graphicBackGround2* riproduce l'illusione ottica di Zöllner (illustrazione 36) la quale fa notare l'apparente convergenza e divergenza delle linee verticali.

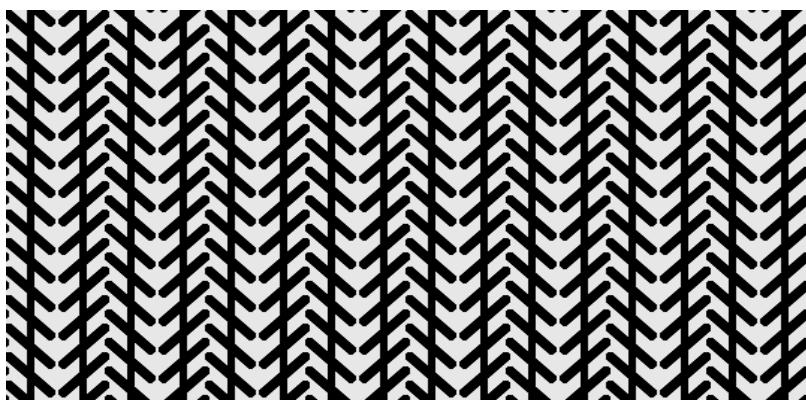
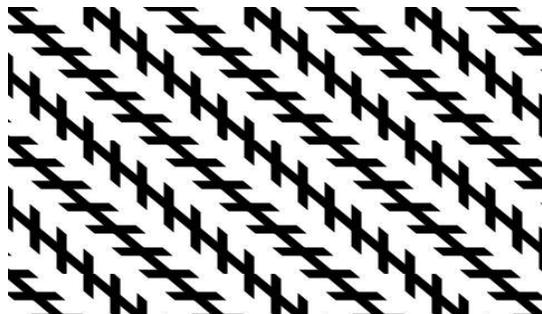


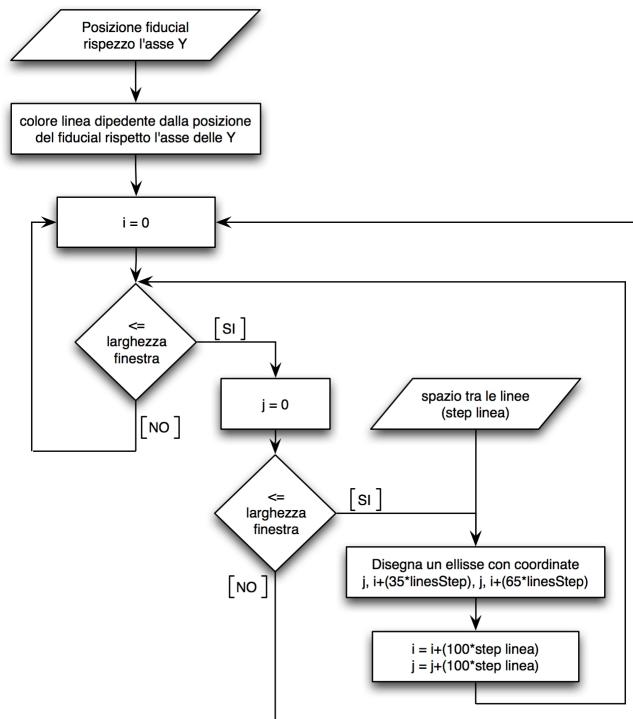
Illustrazione 36: *illusione ottica di Zöllner*

Se si ruota l'immagine di 45°, l'illusione aumenta. Il parallelismo, che in realtà sussiste tra le linee verticali, diventa evidente se si guarda l'immagine portandola all'altezza degli occhi e piatta.



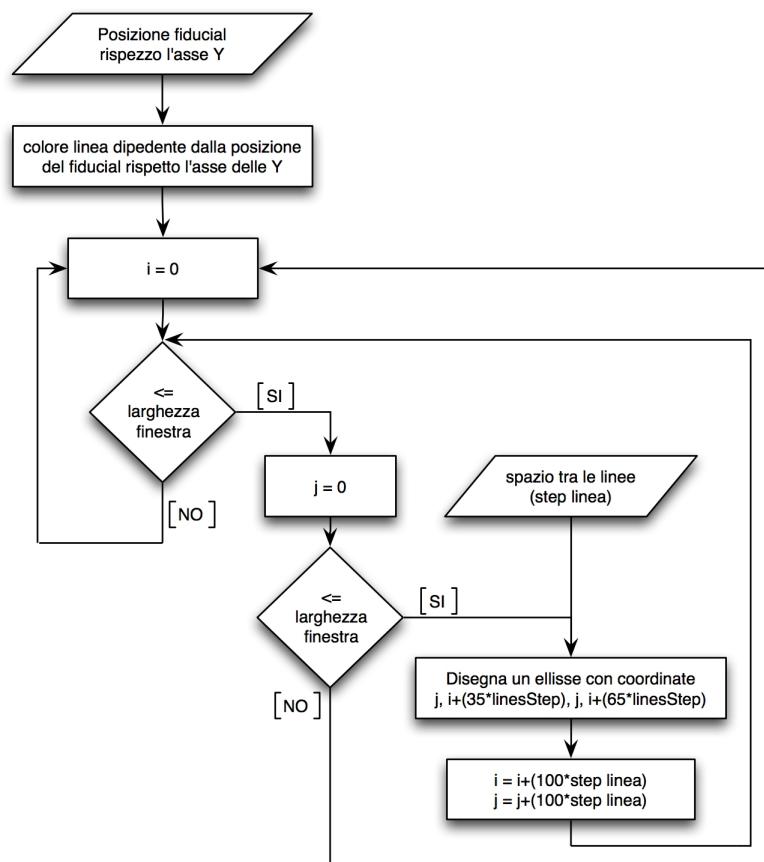
*Illustrazione 37: illusione ottica di Zöllner
ruotata di 45°*

Per il motivo appena citato nella nostra interfaccia le linee verticali saranno ruotate di 45° mentre quelle oblique saranno orizzontali e verticali. Le linee orizzontali e verticali compaiono e scompaiono in base alla posizione del fiducial. Il movimento del fiducial sull'asse delle X determina il colore delle linee orizzontali mentre il movimento del fiducial sull'asse delle Y determina il colore delle linee verticali. L'algoritmo come per il precedente si basa sullo stesso principio, ovvero di sfruttare la due gruppi di cicli for per generare le coordinate delle linee da “disegnare”. Il primo gruppo è così organizzato:



Schema 9: generazione delle linee orizzontali del secondo sfondo grafico

mentre il secondo che si occupa di disegnare le linee verticali:



Schema 10: generazione delle linee verticali del secondo sfondo grafico



Illustrazione 38: fiducial con ID 2 e 4 che elaborano il secondo sfondo grafico

```

float linesColorX, linesColorY; // Colore delle linee sull'asse X ed Y
int linesStep = 2; // Passo tra le linee
int i,j,jj, k; // Variabili cili for

void graphicBackGround2(){
    strokeWeight(2); // Dimensione linee

    // Generazione delle linee orizzontali
    stroke(linesColorX*153); // Colore delle linee in base alla posizione del fiducial sull'asse delle X
    for( i = 0; i<=width; i = i+(100*linesStep)){ // Generazione coordinate asse X delle linee
        for( j = 0; j<=width; j = j+(100*linesStep)){ // Generazione coordinate asse Y delle linee
            line(i+(35*linesStep), j, i+(65*linesStep), j); // Generazione linee
        }
    }
    for( i = 0; i<=width; i = i+(100*linesStep)){ // Generazione coordinate asse X delle linee
        for( j = 75*linesStep; j<=width; j = j+(100*linesStep)){ // Generazione coordinate asse Y delle linee
            line(i+(60*linesStep), j, i+(90*linesStep), j); // Generazione linee
        }
    }
    for( i = 0; i<=width; i = i+(100*linesStep)){ // Generazione coordinate asse X delle linee
        for( j = 25*linesStep; j<=width; j = j+(100*linesStep)){ // Generazione coordinate asse Y delle linee
            line(i+(10*linesStep), j, i+(40*linesStep), j); // Generazione linee
        }
    }
    for( i = 0; i<=width; i = i+(100*linesStep)){ // Generazione coordinate asse X delle linee
        for( j = 50*linesStep; j<=width; j = j+(100*linesStep)){ // Generazione coordinate asse Y delle linee
            line(i+(85*linesStep), j, i+(115*linesStep), j); // Generazione linee
        }
    }

    // Generazione delle linee verticali
    stroke(linesColorY*153); // Colore delle linee in base alla posizione del fiducial sull'asse delle Y
    for( i = 0; i<=width; i = i+(100*linesStep)){ // Generazione coordinate asse Y delle linee
        for( j = 125*linesStep; j<=width; j = j+(100*linesStep)){ // Generazione coordinate asse X delle linee
            line(j, i+(60*linesStep), j, i+(90*linesStep)); // Generazione linee
        }
    }
    for( i = 0; i<=width; i = i+(100*linesStep)){ // Generazione coordinate asse Y delle linee
        for( j = 75*linesStep; j<=width; j = j+(100*linesStep)){ // Generazione coordinate asse X delle linee
            line(j, i+(10*linesStep), j, i+(40*linesStep)); // Generazione linee
        }
    }
    for( i = 0; i<=width; i = i+(100*linesStep)){ // Generazione coordinate asse Y delle linee
        for( j = 50*linesStep; j<=width; j = j+(100*linesStep)){ // Generazione coordinate asse X delle linee
            line(j, i+(35*linesStep), j, i+(65*linesStep)); // Generazione linee
        }
    }
    for( i = 0; i<=width; i = i+(100*linesStep)){ // Generazione coordinate asse Y delle linee
        for( j = 100*linesStep; j<=width; j = j+(100*linesStep)){ // Generazione coordinate asse X delle linee
            line(j, i+(85*linesStep), j, i+(115*linesStep)); // Generazione linee
        }
    }

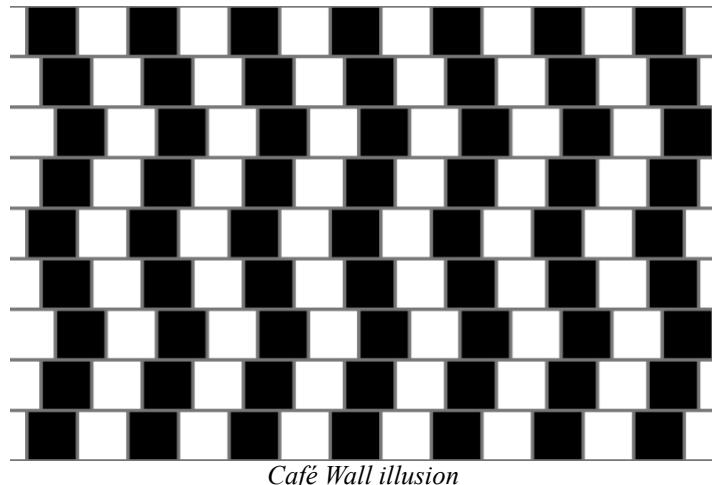
    // Generazione delle linee diagonali
    stroke (153); // Colore linee diagonali
    for( i = (50*linesStep); i<=width+(50*linesStep); i = i+(50*linesStep)) line(i, 0, 0, i); // Generazione linee diagonali
    for( i = (50*linesStep); i<=height+(50*linesStep); i = i+(50*linesStep)) line(i, width, width, i); // Generazione linee diagonali
}

```

Sketch 14: graphicBackGround2

Dallo sketch appena sopra si evince che l'algoritmo si basa su delle repliche degli algoritmi rappresentati mediante gli schemi 9 e 10 e tra un gruppo di ciclo for e l'altro; l'unica differenza sta nell'organizzazione del valore corrente della variabile utilizzata all'interno del ciclo for, la quale incrementa il suo valore di volta in volta di una costante (vedi linee di codice commentate con la dicitura “Generazione linee” dello sketch 13: *graphicBackGround2*). Scorrendo le linee di codice troveremo l'istruzione *fill(linesColor*153)* la quale decide il colore delle linee come un colore che varia dal nero al grigio chiaro mediante il prodotto della variabile *lineeColor* che può variare da 0 ad 1 e la costante 153.

Il terzo sfondo grafico propone l'illusione ottica *Café Wall illusion* la quale è composta da linee orizzontali parallele separate da quadrati bianchi e neri sfalsati verticalmente i quali rompono il parallelismo delle linee.



Questa illusione ottica inizialmente è stata descritta come *Kindergarten illusion* e riscoperta nel 1973 da Richard Gregory, noto psicologo inglese. Secondo Gregory questa illusione è stata notata da un membro del suo studio Steve Simpson, osservando una parete all'esterno del Café Wall situato ali piedi di St Michael's Hill a Bristol. Da qui prende il nome di *Café Wall illusion*.



Il professor Richard Gregory di fronte al Café Wall - St Michael's Hill, Bristol nel Febraio 2010;

Come per gran parte delle illusioni, l'illusione del *Coffe Wall* è stata attribuita in gran parte all'irraggiamento, la diffusione della luce dal buio a zone luminose dell'immagine retinica. In effetti l'immagine scompare quando i quadrati bianchi e neri sono sostituiti da quadrati di colori diversi e stessa luminosità.

```

int sizeSquare = 100; // Dimensione quadrati
float linesColor; // Colore linee
float squareColorX, squareColorY; // CColore quadrati
int x,y; // Cicli for

void graphicBackGround3(){
    strokeWeight(0); // Dimensione linee

    // Generazione primo gruppo di quadrati
    for(y=0; y<height; y=y+sizeSquare*2){ // Generazione coordinate asse Y dei quadrati
        for(x=sizeSquare/2; x<width; x=x+sizeSquare*2){ // Generazione coordinate asse X dei quadrati
            strokeWeight(0); // Dimensione margini quadrati
            fill(squareColorX*153); // Colore quadrati
            rect(x, y, sizeSquare, sizeSquare); // Coordinate quadrati
            strokeWeight(0); // Dimensione linee tra i quadrato
            stroke(153); // Colore linee orizzontali tra i quadrati
            line (0, y, width, y); // Generazione linee tra i quadrati
        }
    }

    // first group of squares
    for(y=sizeSquare; y<height; y=y+sizeSquare*2){ // Generazione coordinate asse Y dei quadrati
        for(x=sizeSquare; x<width; x=x+sizeSquare*2){ // Generazione coordinate asse X dei quadrati
            strokeWeight(0); //Dimensione margini quadrati
            fill(squareColorY*153); // Colore quadrati
            rect(x, y, sizeSquare, sizeSquare); // square
            strokeWeight(0); // Dimensione linee tra i quadrato
            stroke(153); // Colore linee orizzontali tra i quadrati
            line (0, y, width, y); // Generazione linee tra i quadrati
        }
    }
}

```

Sketch 15: graphicBackGround3

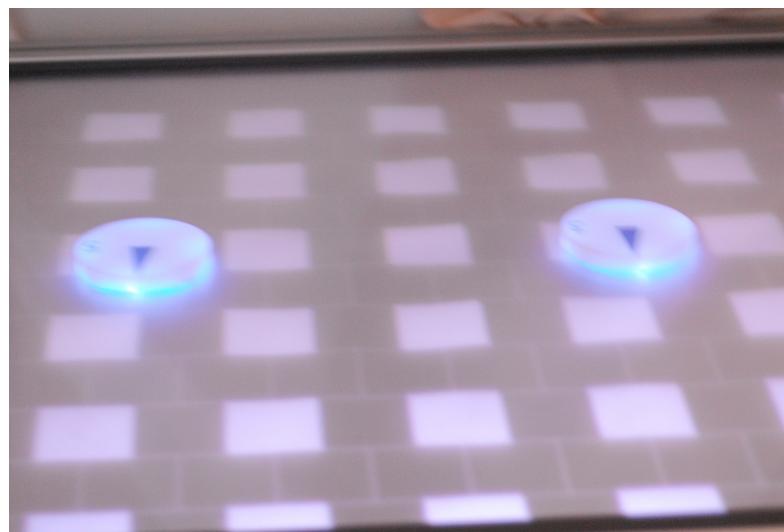
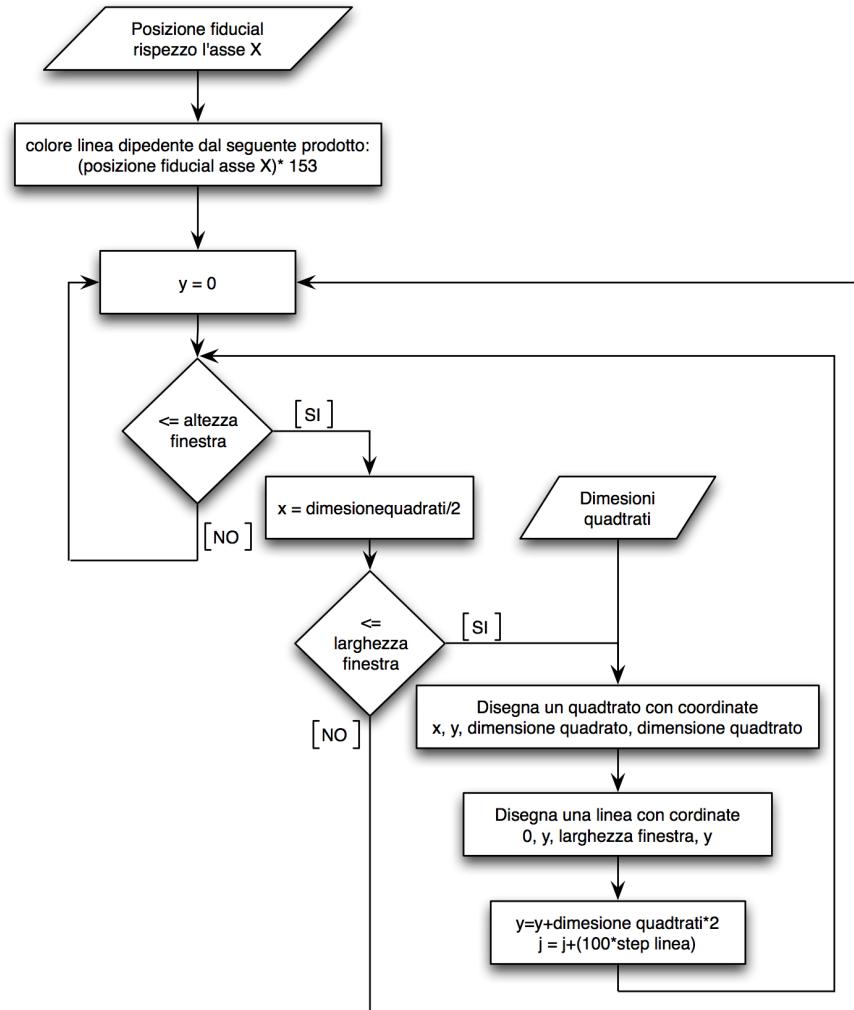


Illustrazione 39: fiducial con ID 3 e 5 che elaborano il terzo sfondo grafico



Schema 11: graphicBackGround3

Come per il primo ed il secondo sfondo grafico (v. Sketch 15 qui sopra) il terzo si può dividere in due gruppi di cicli for i quali generano le coordinate dei relativi quadrati ed il loro colore dipende dalla posizione in coordinate cartesiane rispetto gli assi X ed Y del fiducial. Il colore del primo gruppo di quadrati viene stabilito mediante la posizione del fiducial rispetto l'asse delle X, mentre il colore del fiducial del secondo gruppo di colori viene stabilito in base alla posizione del fiducial rispetto l'asse delle Y. Per stabilire il colore dei quadrati viene adottato lo stesso metodo adottato per stabilire il colore delle linee del secondo sfondo grafico, l'istruzione `fill(squareColor*153)`, la quale decide il colore dei quadrati come un colore che varia dal nero al grigio chiaro mediante il prodotto della variabile `squareColor` che può variare da 0 ad 1 e la costante

153. Ambedue i gruppi di quadrati sono generati mediante l'algoritmo descritto nello schema appena sopra, con l'unica differenza, che si può notare nello sketch nel secondo ciclo for nel primo gruppo di quadrati il punto di partenza del ciclo corrisponde alla dimensione del quadrato (*sizeSquare*), mentre nel secondo la sua metà, in modo tale che la seconda fila di quadrati verrà “disegnata” in maniera sfalsata di una distanza pari alla metà del lato di un quadrato.

Capitolo 2

Elaborazione del segnale audio

2.1 Max/MSP

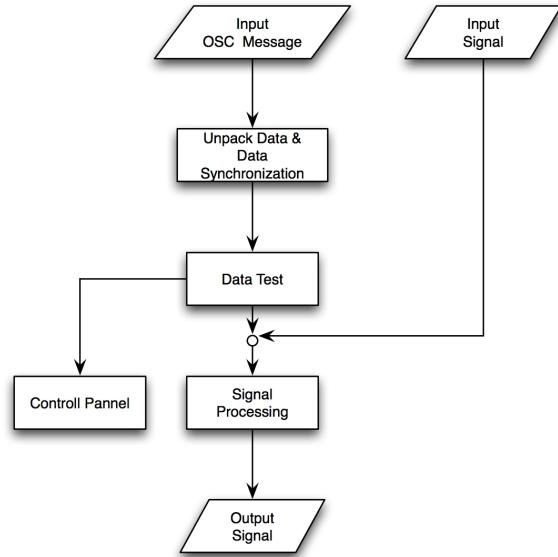
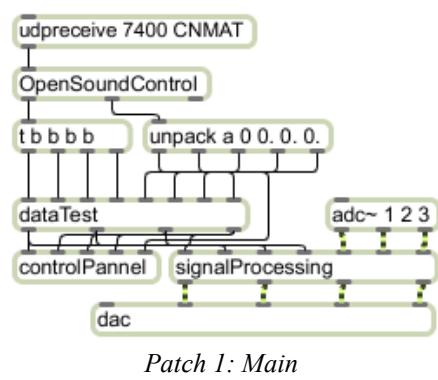
Max è un ambiente di sviluppo grafico per la musica e la multimedialità ideato ed aggiornato dall'azienda di software Cycling '74, con base a San Francisco, California. È utilizzato da oltre quindici anni da compositori, esecutori, progettisti software, ricercatori e artisti interessati a creare software interattivo. Lo stesso applicativo Max è altamente modulare: la maggior parte delle sue routine esiste in forma di libreria condivisa. Una API permette a terze parti lo sviluppo di nuove routines (chiamate external objects, oggetti esterni e patch). Come risultato, Max possiede un largo bacino d'utenza costituito da programmatore - non ricollegabili a Cycling '74 - che ne potenziano il software con estensioni (commerciali e non) al programma. Grazie appunto al suo progetto estensibile e alla sua interfaccia grafica, Max è comunemente considerato una sorta di lingua franca per lo sviluppo di software inerente alla musica interattiva.

2.2 Algoritmo

2.2.1 main

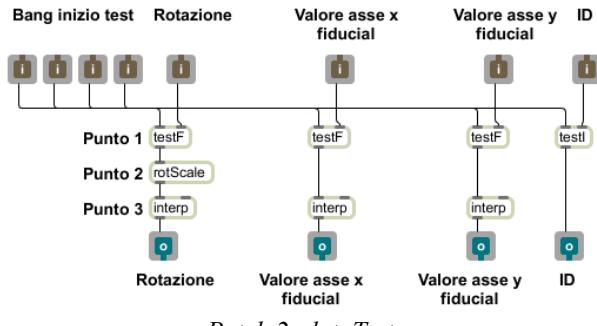
La patch principale è un contenitore di tutto l'algoritmo creato mediante Max/MSP il quale si occupa di elaborare il segnale audio in ingresso.

Il messaggio OSC in ingresso viene aperto mediante il modulo *unpack* il quale estrapola tutti gli argomenti contenuti all'interno del messaggio OSC e mediante l'utilizzo di un external object, *triggerbang*, le variabili vengono inviate come una sequenza ordinata di variabili al successivo external objects, *dataTest*. In questo livello dell'algoritmo avviene un filtraggio dei dati per eliminare tutti i dati in eccesso che potrebbero causare il malfunzionamento dell'algoritmo. I dati in uscita sono pronti per controllare gli algoritmi di processamento del segnale audio in ingresso contenuti nella patch *signalProcessing*. I segnali in uscita vengono inviati ai convertitori audio. *controlPannel* è una patch il cui scopo è solo di controllo dei dati filtrati.



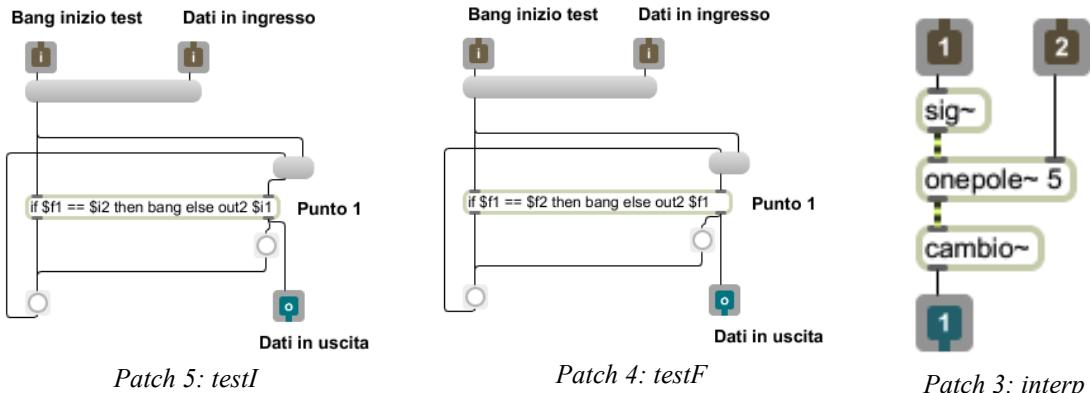
Schema 12: Patch Main

2.2.2 *dataTest*



Patch 2: *dataTest*

La patch *dataTest* effettua un filtraggio dei dati in ingresso in modo tale che solo i dati coerenti con il gesto effettuato possano interagire con gli algoritmi di elaborazione del segnale audio. Il test viene effettuato dalle patch *testI* e *testF* (Patch 2: *dataTest* – Punto 1). Ambedue le patch effettuano un controllo della disuguaglianza dei dati in ingresso. Se il dato corrente in ingresso è diverso dal suo precedente allora sarà mandato in uscita altrimenti il flusso dei dati verrà interrotto (Patch 4: *testI* – Punto 1 ; Patch 3: *testF* – Punto 1).

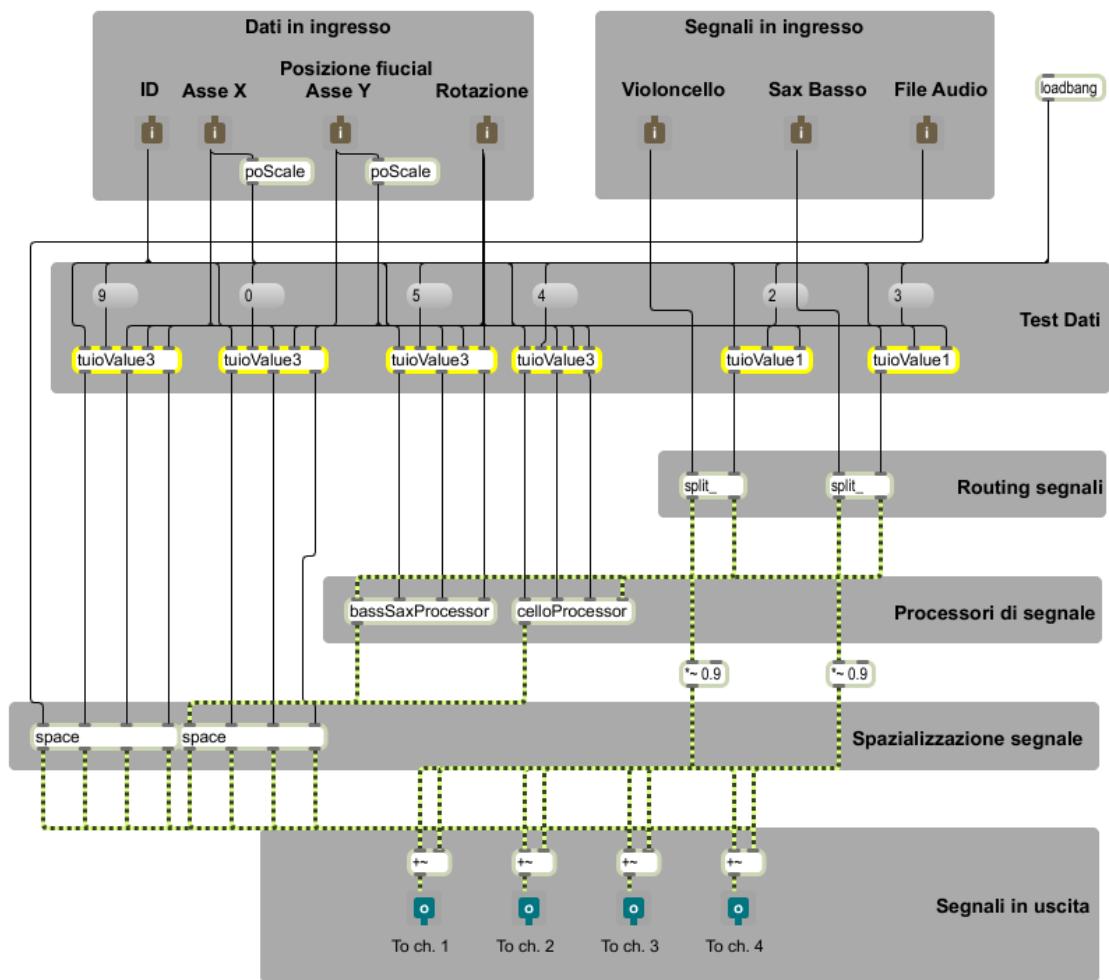


Nella patch *interp* i dati subiscono un filtraggio di tipo passa-basso (*onepole~*) a frequenza sub-audio, 5 Hz, per ottenere una migliore interpolazione dei dati. Per applicare un filtro di tipo passa-basso abbiamo dovuto prima convertire i dati in ingresso in un segnale continuo avente come valore d'ampiezza il valore del dato in ingresso mediante la patch *sig~*, successivamente filtrarlo mediante un filtro passa-basso (*onepole~*) con una frequenza di taglio di 5 Hz ed poi riconvertire il segnale continuo in un flusso di dati mediante la patch *cambio~*, la quale si occupa di estrapolare i valori d'ampiezza del segnale in ingresso.

2.2.3 signalProcessing

La patch *signalProcessing* è la patch la quale contiene e gestisce tutti i processi legati all'elaborazione del segnale ed è composta da più livelli:

- dati in ingresso
- segnali in ingresso
- test dei dati
- routing del segnale audio
- processamento del segnale audio
- spazializzazione del segnale audio
- somma ed uscita dei segnali audio



Patch 6: *signalProcessing*

a) Dati in ingresso

Nel primo livello iniziale compaiono gli ingressi che accolgono i dati OSC e due copie della patch *poScale* che si occupa di scalare opportunamente i dati in ingresso per correggere la distorsione dell'immagine provocata dalla geometria della lente della Videocamera, a causa della quale il tracking dei fiducial effettuato ai vertici della Metis si rivela molto instabile. Per via di questa imprecisione della del sistema di tracking ho deciso di escludere queste aree d'azione e dunque di ottenere dei valori massimi non ai vertici (illustrazione 42) ma a metà degli assi Y ed Y (illustrazione 41).

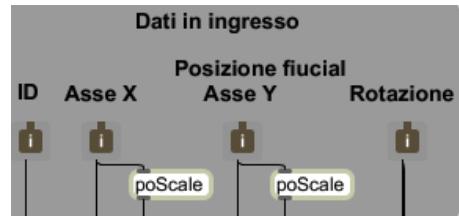


Illustrazione 40: dettaglio "Patch 6: signalProcessing"

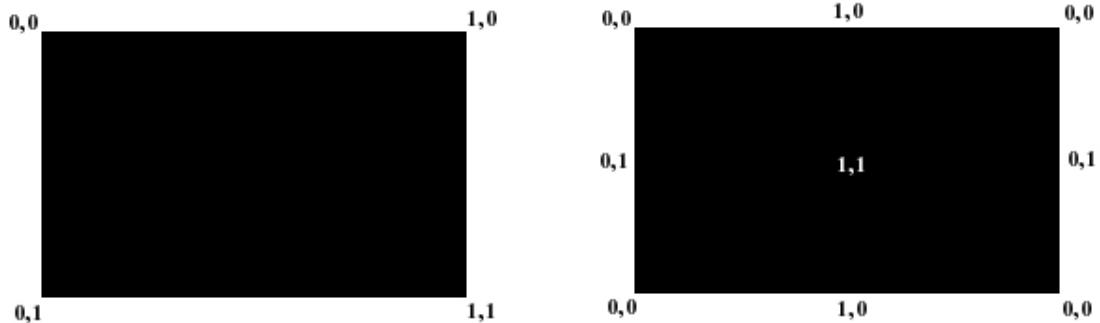
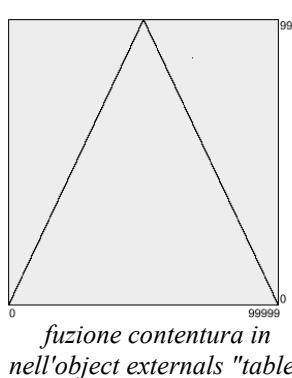


Illustrazione 42: valori originali della posizione del fiducial sugli assi Y ed y

Illustrazione 41: valori scalati della posizione del fiducial sugli assi Y ed y

L'algoritmo che provvede a scalare i valori in ingresso è contenuto nella patch *poScale*, la quale prima moltiplica i valori in ingresso per 100000 (Patch 7: *poScale* – Punto 1), che vengono poi fatti passare all'interno di una funzione triangolare (Patch 7: *poScale* – Punto 2) ed infine vengono moltiplicati per 0.00001



(Patch 7: *poScale* – Punto 3). I valori in ingresso vengono prima moltiplicati per 100000 e per 0.00001 in uscita della funzione per via del fatto che l'oggetto *table* lavora per interi e moltiplicando per queste due costanti i valori in ingresso nella patch ed in uscita da *table* otteniamo un numero avente un'accettabile interpolazione.



b) Segnali in ingresso

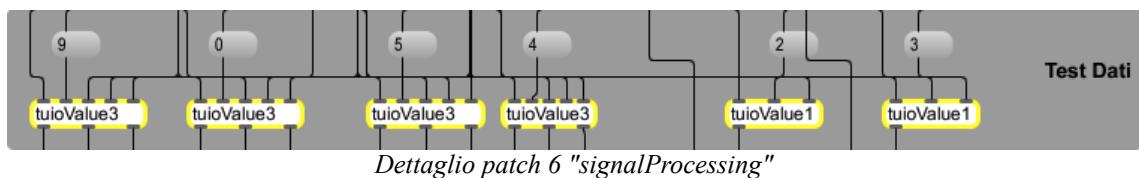
I segnali in ingresso sono i segnali provenienti da microfoni nel caso vogliamo effettuare le elaborazioni su segnali provenienti da strumenti o da un sequencer nel caso si vuole applicare le elaborazioni su segnali provenienti da un supporto digitale.



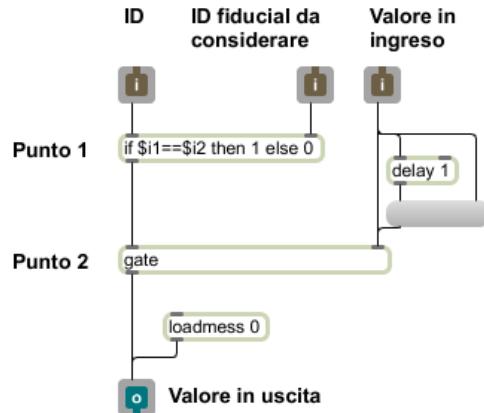
Illustrazione 43: dettaglio "Patch 6: signalProcessing"

c) Test dei dati

In questo livello ha luogo un secondo test il quale regola il flusso dei dati (posizione sugli assi X,Y e rotazione del fiducial) in base all'ID del fiducial. Se l'ID del fiducial corrisponde ad un determinato ID (Patch 8: tuioValue1 – Punto 1) allora il flusso dei dati può avere luogo altrimenti verrà interrotto (Patch 8: tuioValue1 – Punto 2). A determinare questa legge è la sono le patch *tuioValue1* e *tuioValue3*.



Le patch *tuioValue1* e *tuioValue3* si differenziano solo dal numero di dati su cui effettuare il test ma il principio è lo stesso. La patch *tuioValue1* ha un ingresso mentre la patch *tuioValue3* ne ha 3.

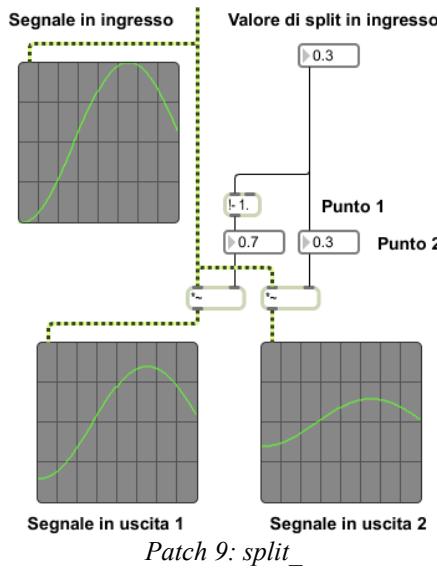


Patch 8: tuioValue1

Come possiamo vedere dalla patch 8, il flusso dei dati viene o meno interrotto in base al valore dell'ID del fiducial su cui si sta agendo (ingresso 1); se corrisponde ad un determinato valore (ingresso 2) il flusso dei dati proveniente dall'ingresso 3 verrà mandato in uscita altrimenti verrà interrotto.

c) Routing del segnale

Il routing del segnale è gestito principalmente dalla patch *split_*, la quale si occupa di deviare il flusso dei segnali su una e/o un'altra uscita.

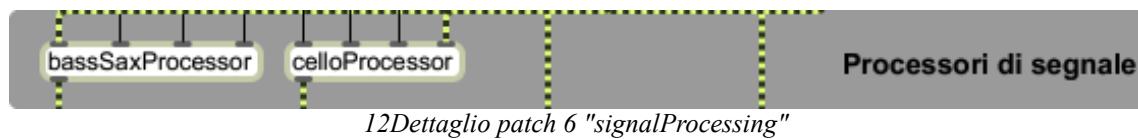


Patch 9: split_

Il segnale in ingresso viene spartito in due, uno viene scalato per un variabile che dipende dal valore della rotazione del fiducial (Patch 9: *split_* - Punto 2) mentre l'altro per il complementare di quest'ultimo (Patch 9: *split_* - Punto 1).

d) Processamento del segnale audio

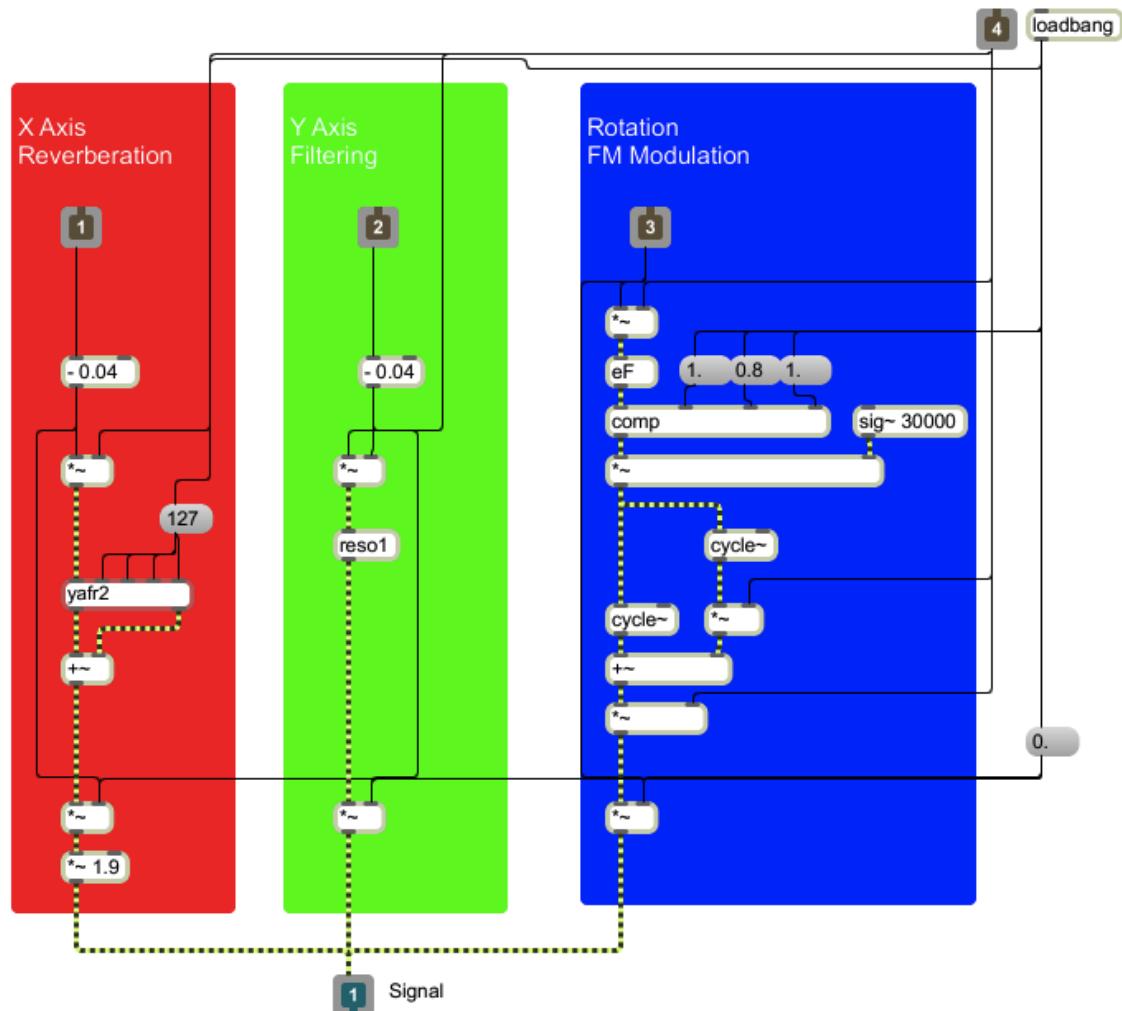
Questa sezione può essere definita come il cuore dell'algoritmo. Essa si occupa di elaborare il segnale audio in ingresso.



I “processori” del segnale sono due: un primo che si occupa di elaborare il segnale di un violoncello (*celloProcessor*) ed un secondo che si occupa di processare il segnale di proveniente da un Sax Basso (*bassSaxProcessor*).

celloProcessor

Sul violoncello sono applicate tre tipologie di processamento del segnale audio: un riverbero, un filtraggio mediante un banco di filtri risonanti ed una modulazione di frequenza.



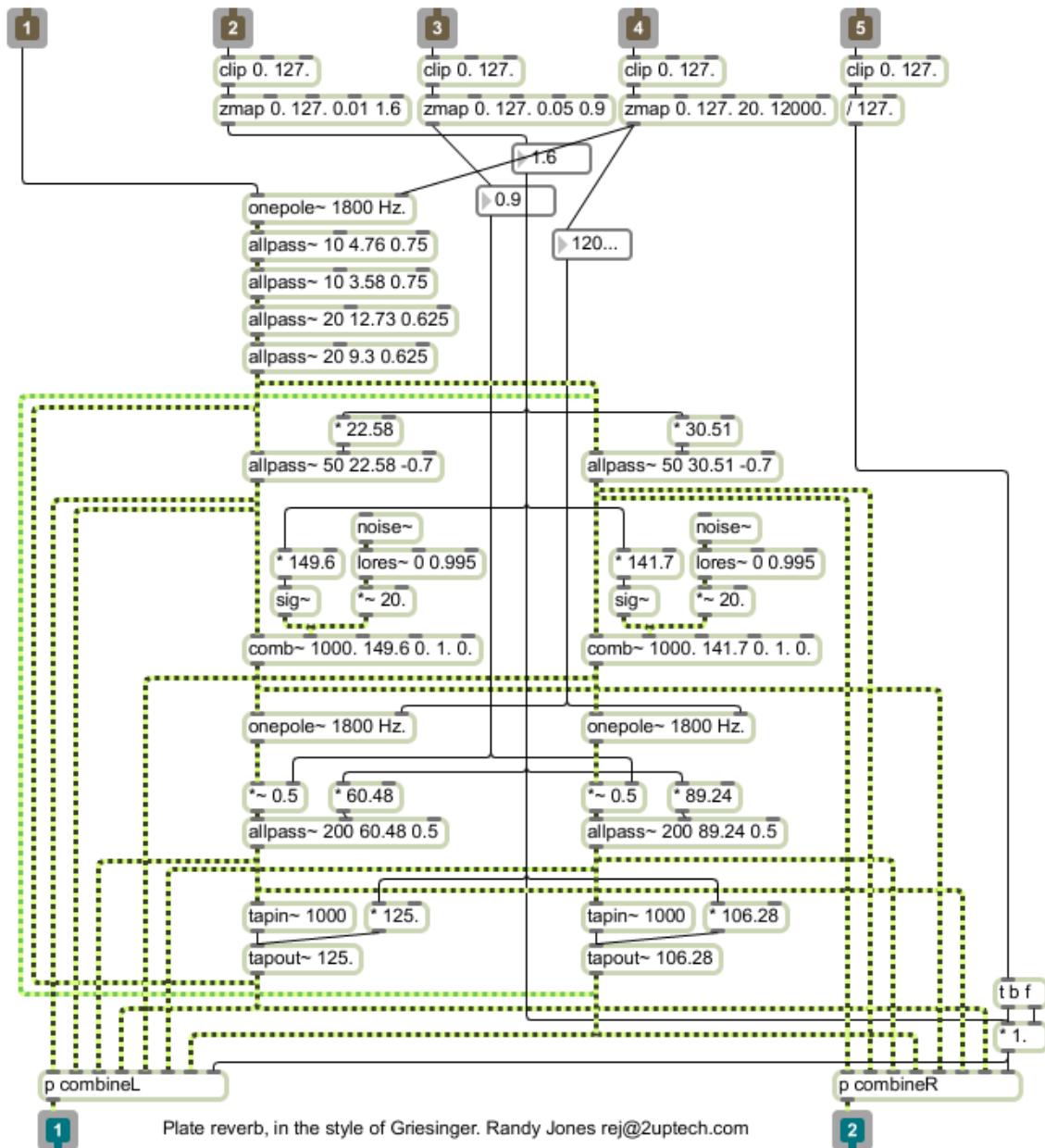
Patch 10: *celloProcessor*

Il segnale in ingresso viene spartito in tre parti e mandato in tutto e tre gli elaboratori ed il segnale risultante verrà scalato in base al valore delle coordinate sull'asse delle X, al valore delle coordinate sull'asse delle Y o della rotazione. Dunque noi non controlliamo i parametri diretti delle elaborazioni ma bensì la quantità di segnale in ingresso ed in uscita dai singoli processori di segnale.

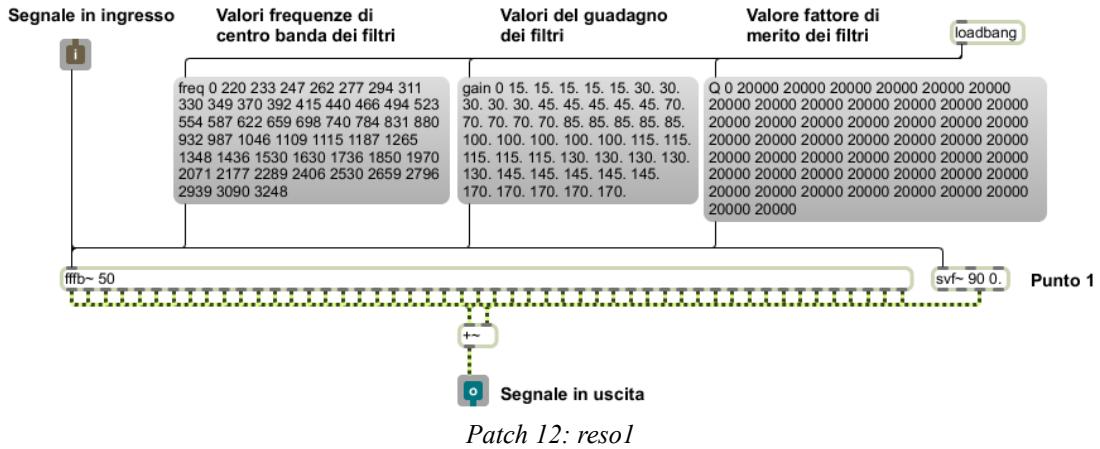
Riverbero

Il riverbero applicato sul segnale è un riverbero proprietario di una libreria di Max/MSP denominato *yaf2*.

Il riverbero *yaf2* è un riverbero basato nella combinazione di filtri passa-tutto (*allpass*), filtri passa-basso (*onepole~*), filtri-comb (*comb~*) e delay (*tapin* e *tapout*).



Resol



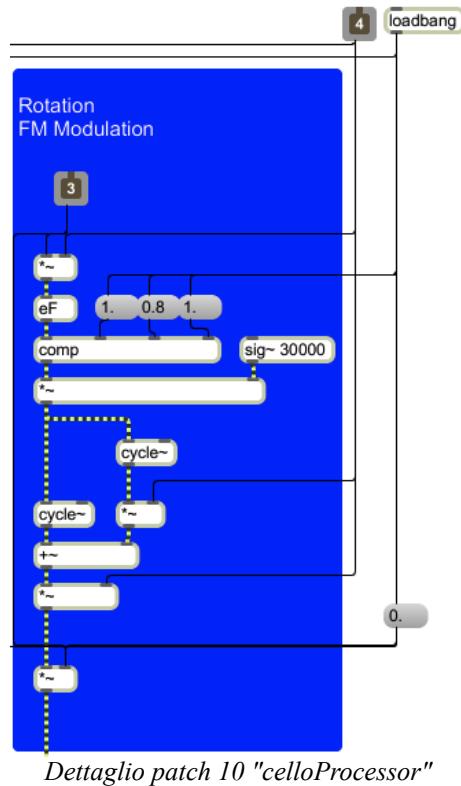
La patch *reso1* implementa un banco di filtri risonanti (patch *ffff~*) applicato su di un segnale proveniente da un violoncello. I valori delle frequenze di centro banda, il loro fattore di merito ed il loro guadagno sono opportunamente definiti in modo tale da esaltare le parziali, specialmente le ultime, del segnale in ingresso e garantirsi un decadimento temporalmente lungo. Appena sotto la tabella che riporta tutti i valori di: frequenza di centro banda, fattore di merito e guadagno del banco di filtri risonanti.

Filtro	Frequenza di centro banda (Hz)	Fattore di merito (dB)	Guadagno (dB)
1	220	20000	15
2	233	20000	15
3	247	20000	15
4	262	20000	15
5	277	20000	15
6	294	20000	30
7	311	20000	30
8	330	20000	30
9	349	20000	30
10	370	20000	30
11	392	20000	45
12	415	20000	45
13	440	20000	45
14	466	20000	45
15	494	20000	45

16	523	20000	70
17	554	20000	70
18	587	20000	70
19	622	20000	70
20	659	20000	70
21	698	20000	85
22	740	20000	85
23	784	20000	85
24	831	20000	85
25	880	20000	85
26	932	20000	100
27	987	20000	100
28	1046	20000	100
29	1109	20000	100
30	1115	20000	100
31	1187	20000	115
32	1265	20000	115
33	1348	20000	115
34	1436	20000	115
35	1530	20000	115
36	1630	20000	130
37	1736	20000	130
38	1850	20000	130
39	1970	20000	130
40	2071	20000	130
41	2177	20000	145
42	2289	20000	145
43	2406	20000	145
44	2530	20000	145
45	2659	20000	145
46	2796	20000	170
47	2939	20000	170
48	3090	20000	170
49	3248	20000	170
50	3415	20000	170

FM

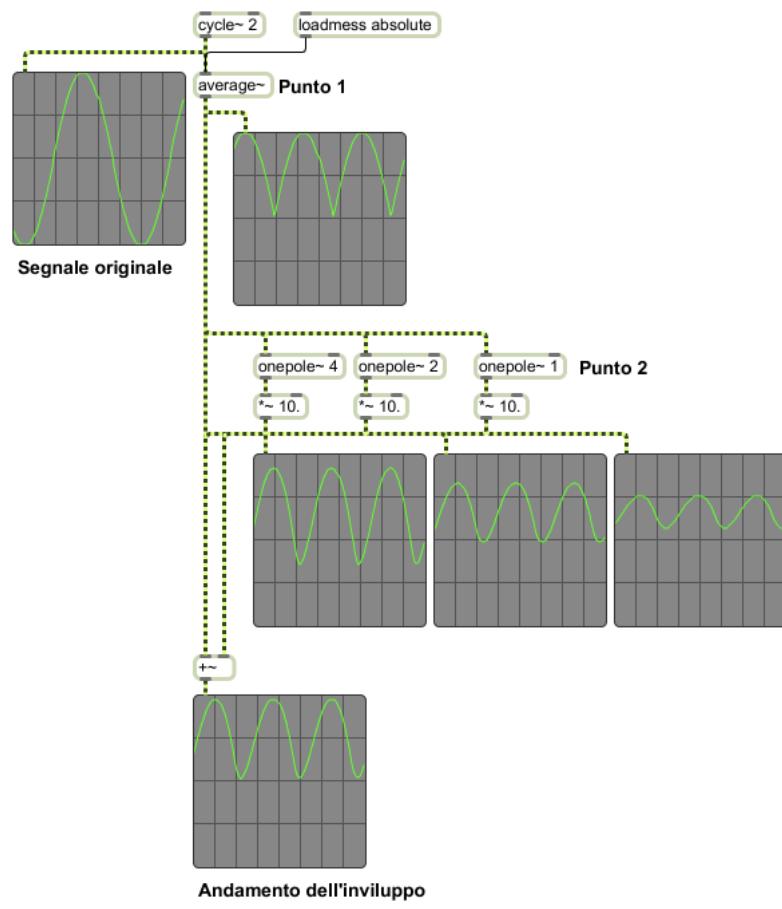
Quest'ultima parte della patch sfrutta il segnale in ingresso sia come segnale da elaborare che come segnale di controllo per variare i parametri della modulazione di frequenza.



Prima di procedere nella descrizione dell'algoritmo dovremo introdurre due nuove patch: *eF* e *comp*.

eF

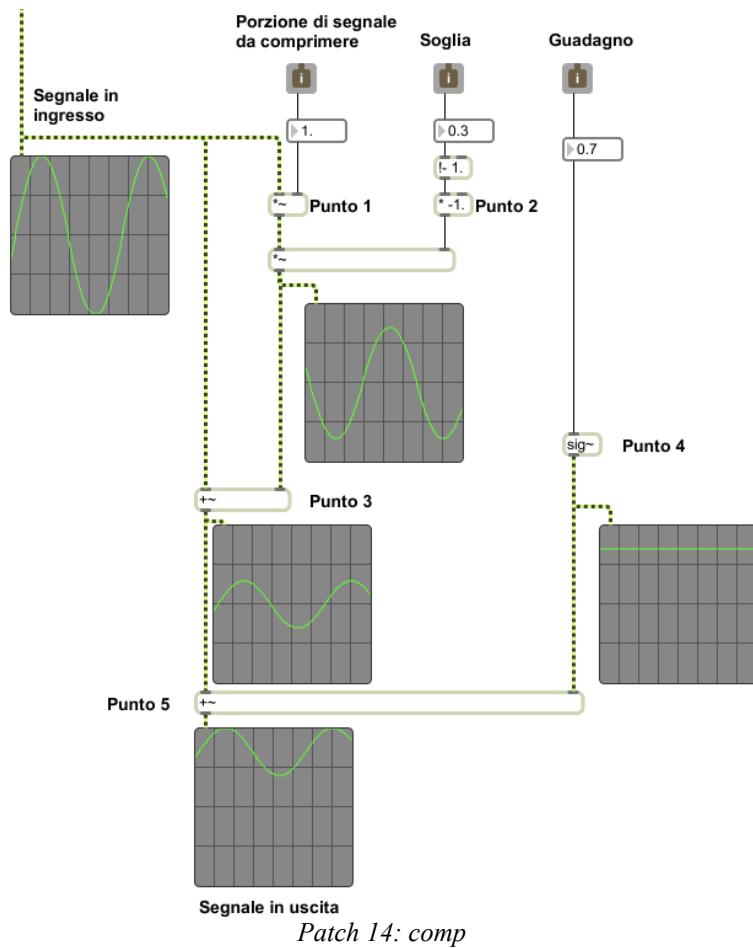
eF è un inseguitore d'inviluppo (Patch 13: *eF*; acronimo di envelope follower) prende in esame il segnale in ingresso né fa il valore assoluto (external object *average~*) (Patch 13: *eF* – Punto 1) ; successivamente applica tre filtri di tipo passa-basso (external object *onepole~*) con frequenze di taglio rispettivamente di 1, 3 e 5 Hz (Patch 13: *eF* – Punto 2) per ottenere in uscita un segnale “smussato”. Infine vengono sommati tutti i segnali in uscita dai filtri e dall'external object *average~*.



Patch 13: *eF*

comp

comp è un compressore che effettua una compressione di tipo tradizionale per applicare una riduzione della dinamica del segnale audio e innalzare o diminuire l'intensità complessiva del segnale.

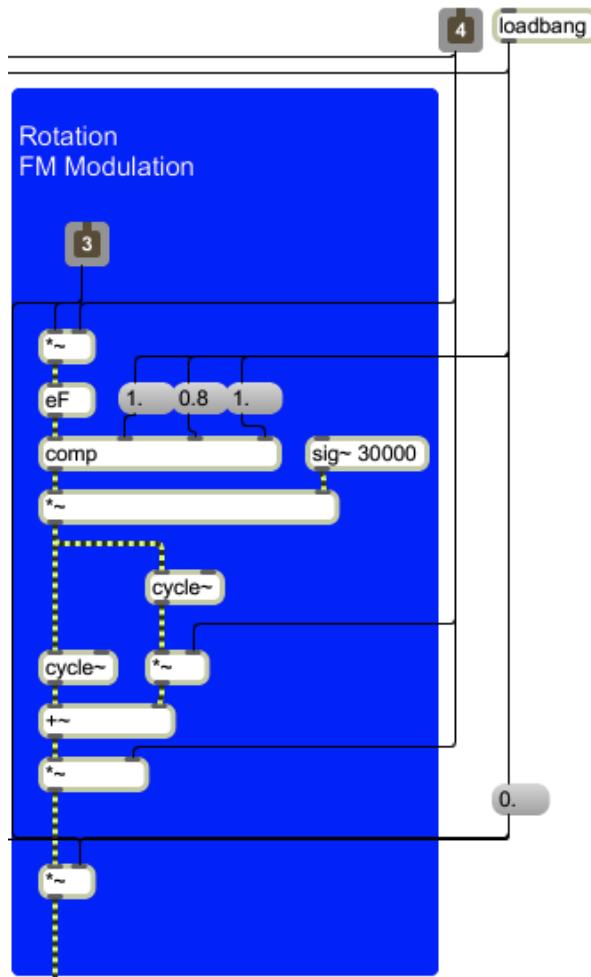


Il segnale in ingresso viene prima spartito in due vie. Una delle parti del segnale, verrà scalata per una variabile che andrà a determinare la porzione di segnale da comprimere (Patch 10: *comp* - Punto 1), successivamente verrà riscalato di nuovo ed invertito di fase (Patch 10: *comp* - Punto 2). La successiva somma (Patch 10: *comp* - Punto 3) del segnale originale con il segnale scalato ed invertito di fase consente una riduzione della dinamica del segnale. Al segnale risultante viene sommato un segnale continuo generato mediante la patch *sig~* (Patch 10: *comp* - Punto 4) in modo tale da riportare l'intensità del segnale corrente alla stessa del segnale originale (Patch 10: *comp* - Punto 5).

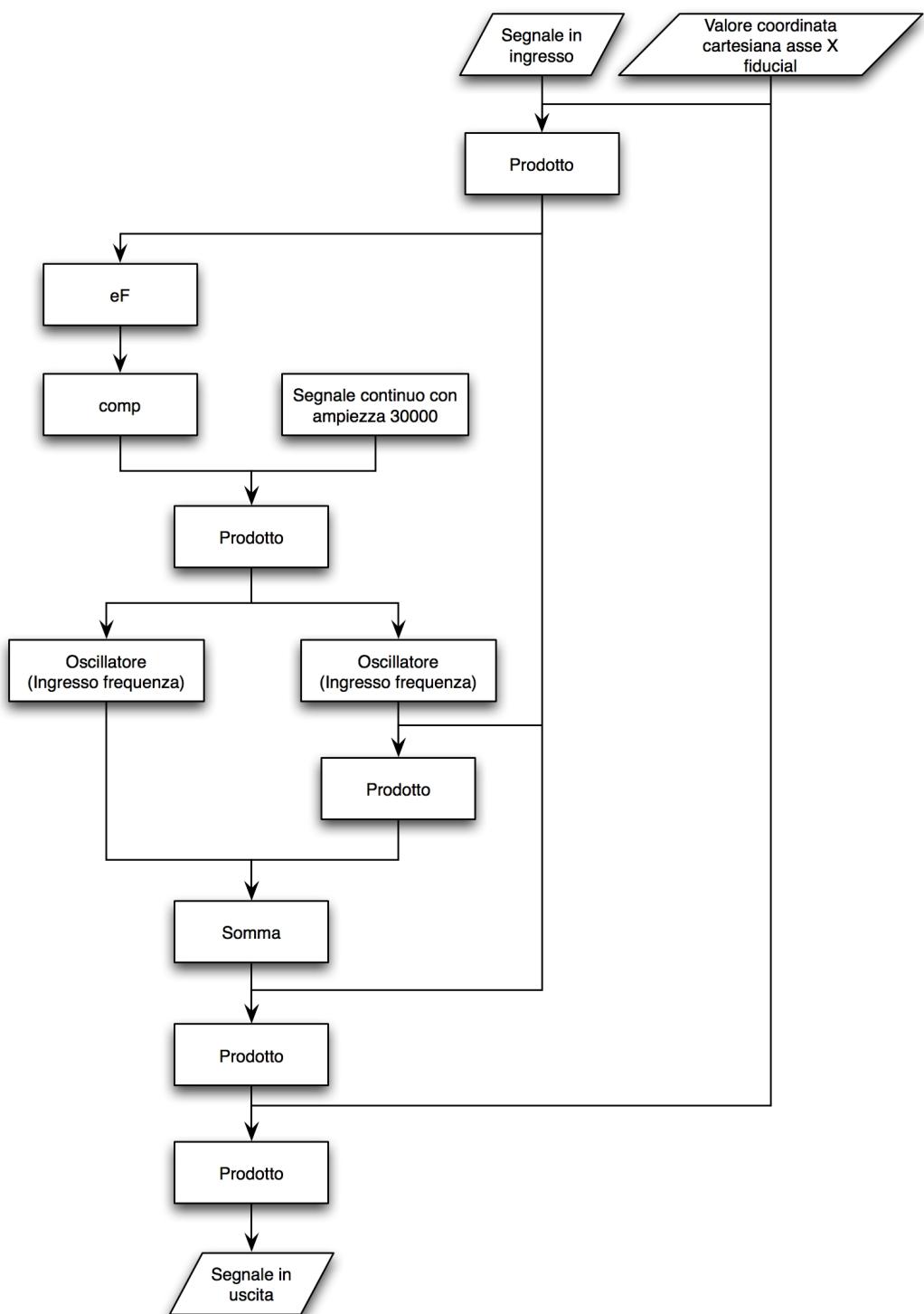
Ora che conosciamo il funzionamento delle patch *eF* e *comp* possiamo continuare con la descrizione della terza parte dell'algoritmo che si occupa di elaborare il violoncello.

Il segnale in ingresso viene prima riscalato per una variabile dipendente dal valore della rotazione del fiducial il quale stabilisce la porzione di segnale che dev'essere coinvolto nella modulazione di frequenza, che può variare tra 0 ed 1 (se la freccia è rivolta verso l'alto il valore sarà 0 mentre se la freccia è rivolta verso il basso il valore in uscita sarà 1). Il segnale appena rascalato viene preso in esame da un'inseguitore d'inviluppo (*eF*) il quale ne calcola l'andamento dell'ampiezza. Il segnale in uscita viene compresso mediante un compressore (*comp*) e successivamente moltiplicato per un segnale continuo con ampiezza 30000.

Il processo appena descritto permette mediante il segnale in ingresso di stabilire il valore della frequenza dei due segnali sinusoidali che saranno i segnali di controllo del segnale modulante e dell'indice di modulazione della modulazione di frequenza, mentre il segnale modulato sarà il segnale in ingresso. Considerando che il segnale in ingresso è un pizzicato Bartok, ovvero un segnale di tipo quasi impulsivo, otteniamo in uscita un segnale che all'istante temporale iniziale è uguale a quello originale mentre con lo scorrere del tempo la distanza (in termini di Hz) tra una parziale e l'altra aumenta con l'aumentare dell'ampiezza del segnale in ingresso.



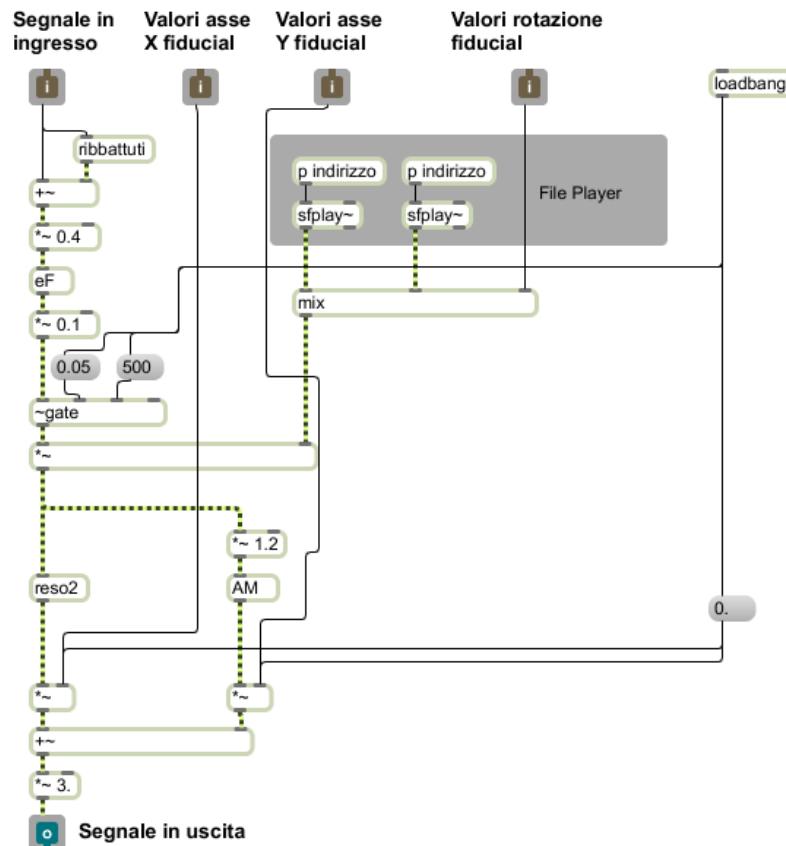
dettaglio patch 10 "celloProcessor"



Schema 13: modulazione di frequenza effettuata nella patch 10 - "celloProcessor"

bassSaxProcessor

Appena sotto possiamo notare l'algoritmo costruito mediante Max/MSP il quale si occupa di processare il segnale proveniente da un Sax Basso. Prima di addentrarci nella descrizione della struttura dell'algoritmo dell'algoritmo *bassSaxProcessor* è bene parlare di ulteriori patch ancora non incontrate prima come: *indirizzo*, *ribattuti*, *mix*, *~gate*, *reso2* e *AM*.

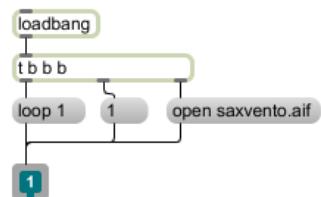


Patch 15: bassSaxProcessor

indirizzo

Le patch *indirizzo* contengono dati utili alla corretta esecuzione di campioni audio, spediti in uscita con un ordine ben preciso al player *sfplay*. Di seguito l'ordine d'invio dei messaggi:

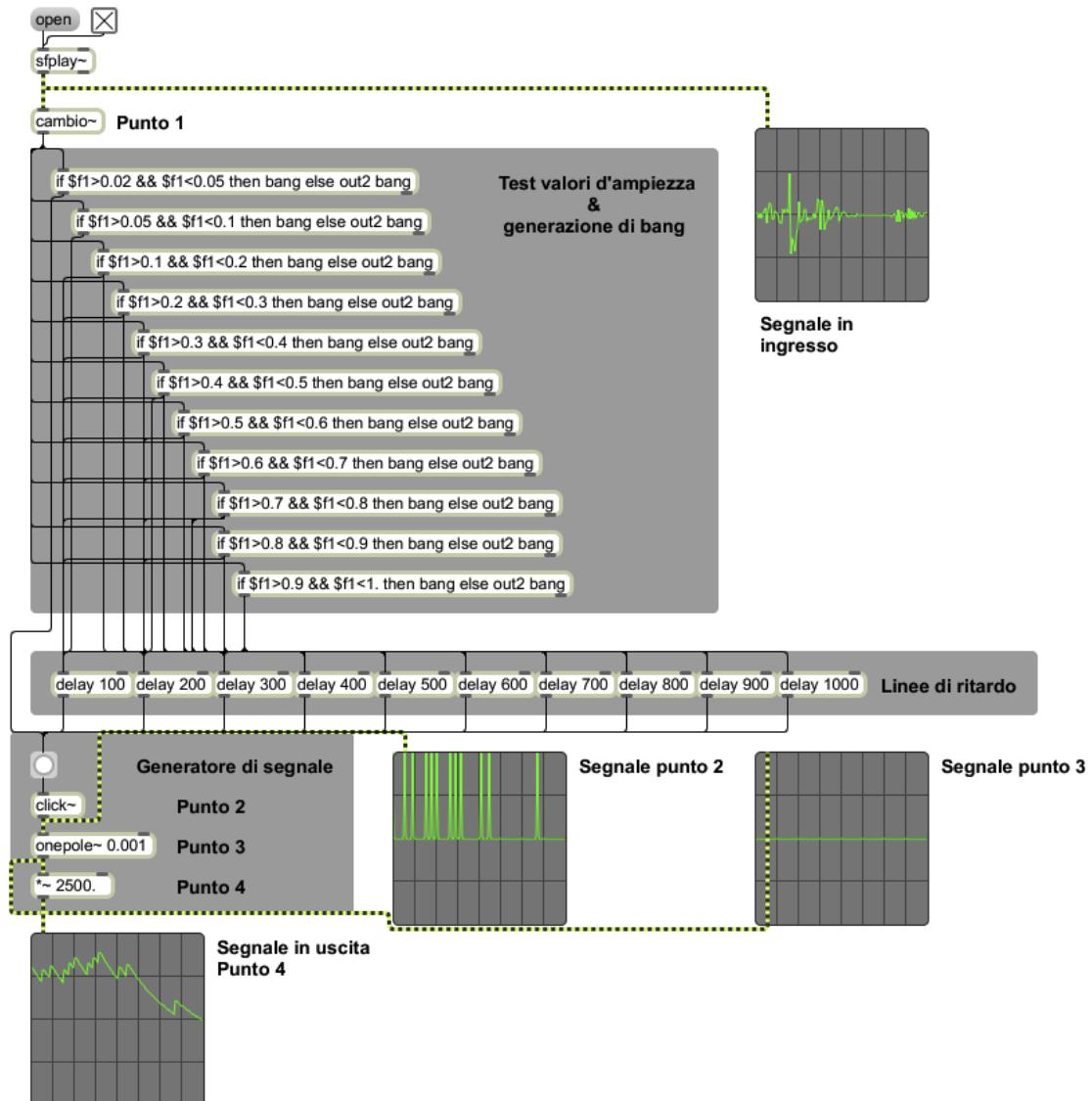
- *open*, file da leggere;
- *l*, start della lettura;
- *loop 1*, inizio del loop;



Patch 16: indirizzo

ribattuti

La patch *ribattuti* genera una sequenza di impulsi il cui numero d'impulsi dipende dall'ampiezza del segnale in ingresso. Per prima cosa vengono calcolati i valori d'ampiezza del segnale in ingresso mediante la external object *cambio~* (Patch 17: *ribattuti* – punto 1). Successivamente vengono effettuati 11 test sul flusso dei dati, dove si verifica se il valore dell'ampiezza del segnale in ingresso rientra in un certo range. Ogni qual volta che il valore d'ampiezza del segnale in ingresso si trova anche per un solo istante in uno di questi range viene generato un bang. A parte il primo tutti gli altri bang subiscono l'effetto di un delay il cui tempo di ritardo aumenta progressivamente con l'aumentare dell'ampiezza.



Patch 17: *ribattuti*

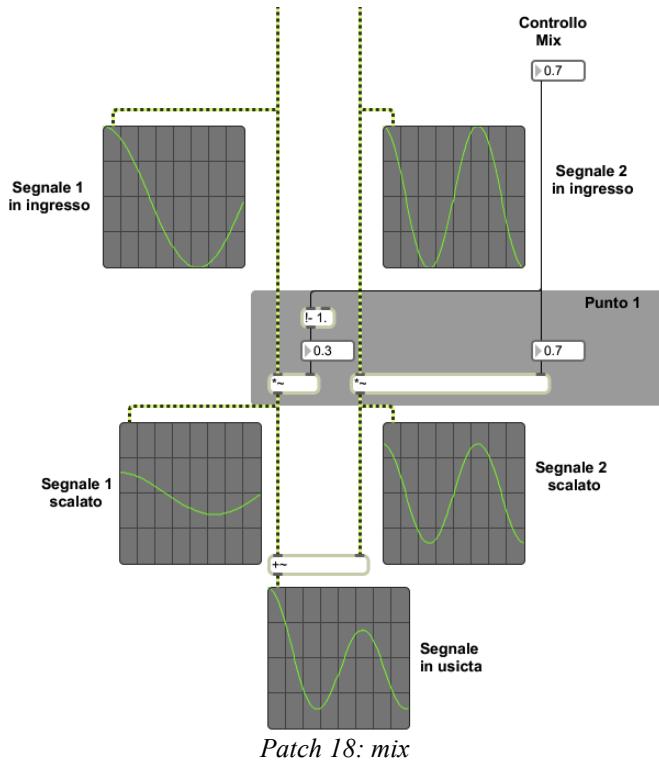
Di sotto i range d'ampiezza ed i rispettivi delay sono:

1. da 0.02 a 0.05 (delay 0 secondi)
2. da 0.05 a 0.1 (delay 100 ms)
3. da 0.1 a 0.2 (delay 200 ms)
4. da 0.2 a 0.3 (delay 300 ms)
5. da 0.3 a 0.4 (delay 400 ms)
6. da 0.4 a 0.5 (delay 500 ms)
7. da 0.5 a 0.6 (delay 600 ms)
8. da 0.6 a 0.7 (delay 700 ms)
9. da 0.7 a 0.8 (delay 800 ms)
10. da 0.8 a 0.9 (delay 900 ms)
11. da 0.9 a 0.1 (delay 1000 ms)

I bang una volta ritardati vanno a controllare un generatore d'impulsi (*click~*) (Patch 17: *ribattuti* – Punto 2). Gli impulsi generati vengono filtrati mediante un filtro passa-basso (*onepole~*) avente una frequenza di taglio di 0.001 Hz in modo tale da smussare l'inviluppo dell'ampiezza del segnale generato dal generatore d'impulsi (Patch 17: *ribattuti* – Punto 3). Il segnale appena smussato viene moltiplicato per una fattore 2500 per via del forte filtraggio appena effettuato (Patch 17: *ribattuti* – Punto 4). Il segnale ottenuto è impiegato per modulare l'ampiezza della somma dei due campioni in uscita dalla patch *mix* (vedi patch *mix*, pagina seguente).

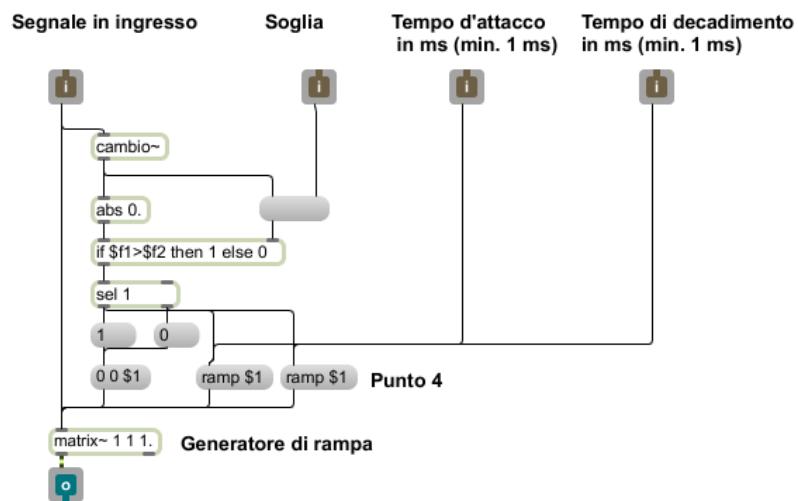
mix

La patch *mix* si occupa di fare la somma di due segnali, di cui il secondo è scalato per un fattore di riscalamento variabile tra 0 ed 1 mentre il primo per il suo complementare (patch 18 *mix* – punto 1).



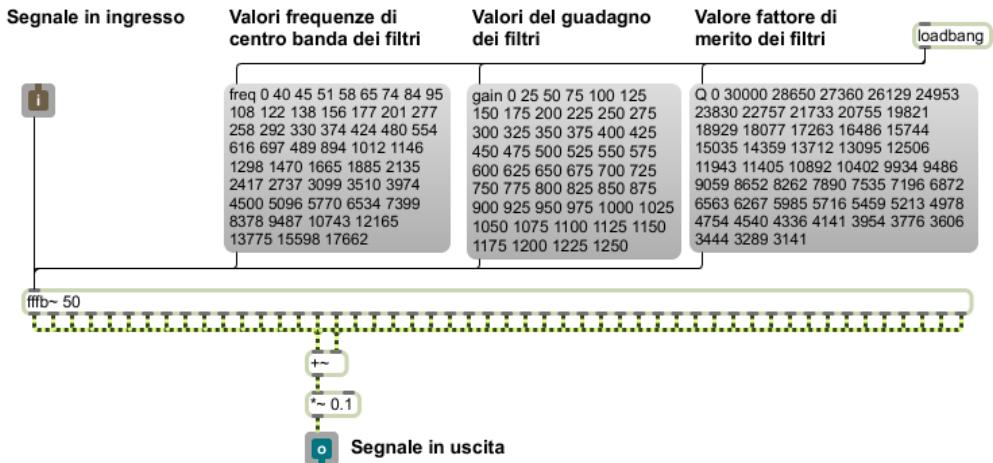
\sim gate

Come dice la traduzione letterale del termine, cancello, porta ecc., la patch \sim gate ci permette di gestire il flusso del segnale in ingresso, ovvero di stabilire se il flusso di dati può o meno raggiungere l'uscita. Il segnale in ingresso viene analizzato dal external object *cambio~* la quale estrapola i valori d'ampiezza del segnale in ingresso (Patch 19: *gate* - Punto 1). Successivamente mediante la external object *abs* viene calcolato il suo valore assoluto (Patch 19: \sim gate - Punto 2) in modo tale che nel successivo test applicato sui valori in uscita abbiamo a che fare solo con valori positivi. Il test applicato (Patch 19: \sim gate - Punto 3) verifica la superiorità dei valori variabili in ingresso con un secondo valore costante detto "soglia". Se i valori variabili sono superiori alla soglia allora il segnale in ingresso potrà raggiungere l'uscita altrimenti il suo flusso sarà interrotto. La variazione dei due stati di "on" ed "off" viene attuata da una rampa di cui se ne può stabilire il tempo di azione. Se il test risulterà positivo la rampa generata varierà da 0 ad 1 nel "tempo d'attacco" prestabilito; mentre se risulterà negativo la rampa generata varierà da 1 a 0 nel "tempo di decadimento" prestabilito (Patch 19: \sim gate - Punto 4).



Patch 19: \sim gate

reso2



Patch 20: reso2

La patch *reso2* è costituita un banco di filtri risonanti della stessa tipologia di quello utilizzato per il violoncello, la differenza sta nei valori utilizzati per stabilire la frequenza di centro banda, il fattore di merito ed il guadagno dei filtri.

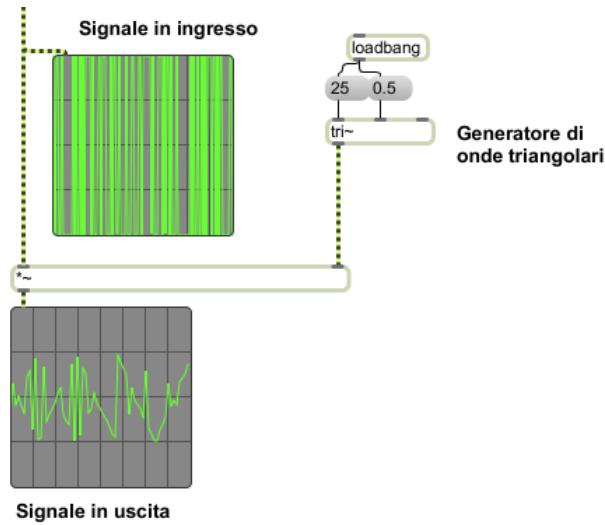
Appena sotto la tabella che riporta tutti i valori di: frequenza di centro banda, fattore di merito e guadagno del banco di filtri risonanti.

Filtro	Frequenza di centro banda (Hz)	Fattore di merito (dB)	Guadagno (dB)
1	40	30000	25
2	45	28650	50
3	51	27360	75
4	58	26129	100
5	65	24953	125
6	74	23830	150
7	84	22757	175
8	95	21733	200
9	108	20755	225
10	122	19821	250
11	138	18929	275
12	156	18077	300
13	177	17263	325
14	201	16486	350
15	277	15744	375
16	258	15035	400

17	292	14359	425
18	330	13712	450
19	374	13095	475
20	424	12506	500
21	480	11943	525
22	554	11405	550
23	616	10892	575
24	697	10402	600
25	489	9934	625
26	894	9486	650
27	1012	9059	675
28	1146	8652	700
29	1298	8262	725
30	1470	7890	750
31	1665	7535	775
32	1885	7196	800
33	2135	6872	825
34	2417	6563	850
35	2737	6267	875
36	3099	5985	900
37	3510	5716	925
38	3974	5459	950
39	4500	5213	975
40	5096	4978	1000
41	5770	4754	1025
42	6534	4540	1050
43	7399	4336	1075
44	8378	4141	1100
45	9487	3954	1125
46	10743	3776	1150
47	12165	3606	1175
48	13775	3444	1200
49	15598	3289	1225
50	17662	3141	1250

AM

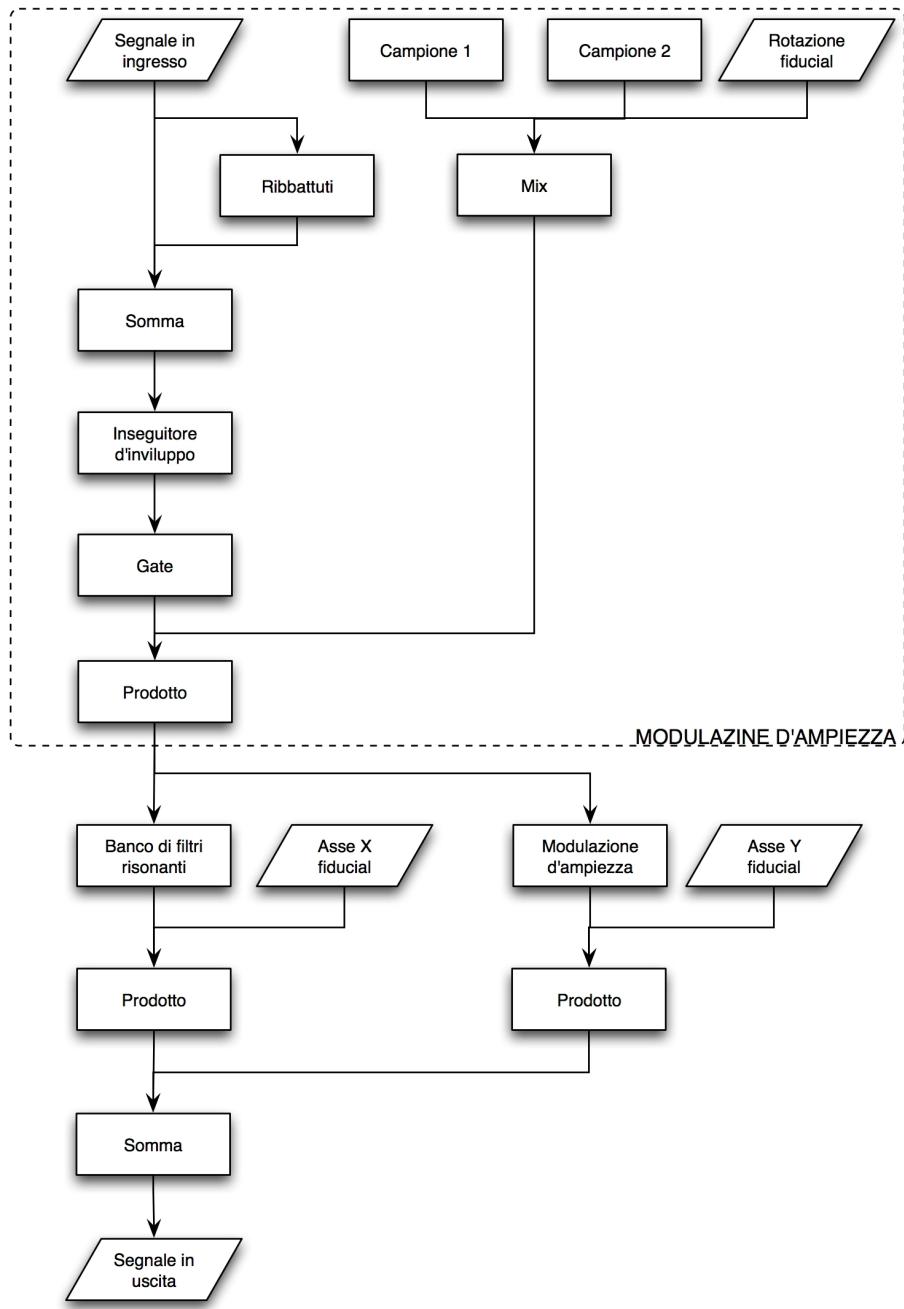
La patch *AM*, come lascia immaginare il nome stesso della patch, applica una modulazione d'ampiezza sul segnale in ingresso il quale viene modulato mediante un'onda triangolare con frequenza 25 Hz ed un duty-cycle di 0.5.



Patch 21: AM

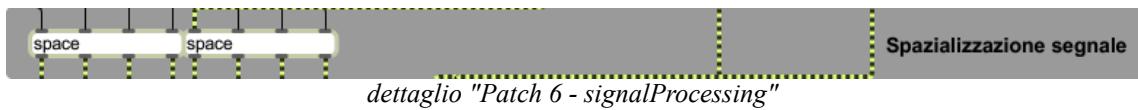
Dopo aver descritto le patch *indirizzo*, *ribattuti*, *mix*, *~gate*, *reso2* e *AM* possiamo tornare alla descrizione dell'algoritmo *bassSaxProcessor* aiutandoci con lo schema n. 3 alla pagina seguente.

Il segnale in ingresso viene utilizzato per generare una serie d'impulsi (*ribattuti*) i quali vengono poi sommati al segnale in ingresso. Il segnale risultante viene analizzato da un inseguitore d'inviluppo ed il segnale risultante verrà utilizzato come segnale modulante in una successiva modulazione d'ampiezza di cui il segnale modulato è la somma, mediante la patch *mix*, di due file audio, effettuata agendo sul controllo rotativo del fiducial. Il segnale in uscita dalla modulazione d'ampiezza a sua volta viene spartito ed inviato sia ad un banco di filtri risonanti (Patch 20: *reso2*) sia ad un'ulteriore modulazione d'ampiezza (Patch 21: *AM*), dove viene modulato mediante un'onda triangolare avente una frequenza di 25 Hz ed un duty-cycle di 0.5. I due segnali vengono poi scalati per un fattore che dipende dalla posizione sugli assi X,Y del fiducial sul tavolo.



Schema 14: bassSaxProcessor

f) Spazializzazione del segnale audio



Il segnale in uscita dalle patch *celloProcessor*, *bassSaxProcessor* ed il segnale proveniente dall'ingresso numero 3 del convertitore audio analogico-digitale vengono spazializzati nella sala d'ascolto mediante la pach *space* ed un sistema di diffusione adeguato e disposto in maniera quadrifonica.

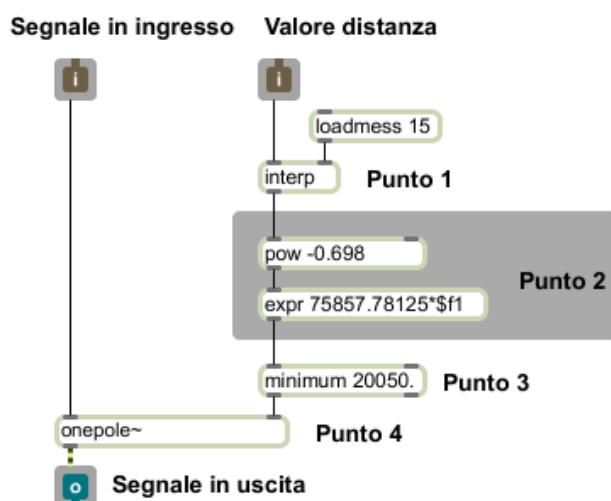
Per meglio comprendere il funzionamento dell'algoritmo che si occuperà di spazializzare il segnale all'interno di un sistema quadrifonico è doveroso introdurre le seguenti patch: *airScale* e *gainScale*.

airScale

airScale è una patch la quale simula il filtraggio di tipo passa-basso di un segnale audio dovuto dall'effetto di filtraggio dell'aria. Il filtro applicato sarà un filtro dinamico di tipo passa-basso la cui frequenza di taglio è determinata dal risultato dell'equazione 1 appena sotto, la quale calcola la frequenza di taglio in funzione della distanza.

$$f_c(t) = 10^{4.88} d^{-0.698}$$

Equazione 1: Calcolo frequenza di taglio del filtro passa-basso



Patch 22: airScale

L'implementazione di questo algoritmo appare nella patch 22. I valori della distanza vengono interpolati mediante la patch *interp* (Patch 22: *airScale* - Punto 1). Il flusso di dati appena generato andrà a soddisfare le esigenze dell'equazione 1 la quale ci darà in uscita un flusso di valori che corrispondono alla frequenza di taglio del filtro passa-basso (Patch 22: *airScale* - Punto 2) il quale agirà sul segnale in ingresso (Patch 22: *airScale* - Punto 4). Prima di raggiungere il filtro, il valore della frequenza di taglio subirà l'effetto di un "limiter" per far sì che il valore del flusso dei dati in uscita dall'equazione 1 non superi il valore 20050 che corrisponde al valore della frequenza massima udibile da un orecchio umano (Patch 22: *airScale* - Punto 3).

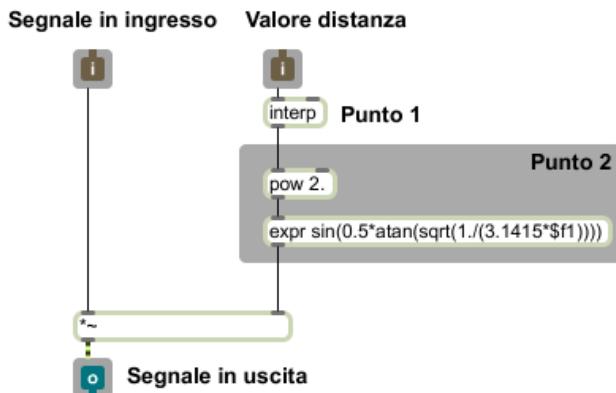
gainScale

La patch si occupa di scalare il segnale in ingresso in funzione della distanza tra uno dei vertici della Metis ed il fiducial. Il valore dell'attenuazione da applicare sul segnale viene calcolata mediante l'equazione 2 appena sotto.

$$P_R = P \operatorname{sen}^2\left(\frac{1}{2} \operatorname{arctg} \sqrt{\frac{1}{\pi R^2}}\right)$$

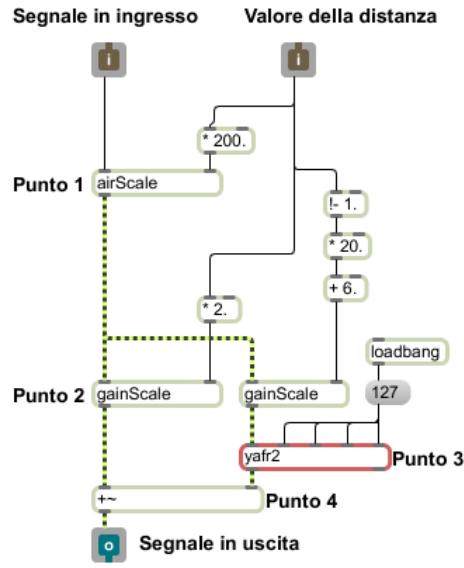
Equazione 2: Calcolo attenuazione applicata dovuta dalla distanza dalla sorgente al punto d'ascolto

Come possiamo vedere dalla patch 23: *gainScale*, l'algoritmo generato è molto simile all'algoritmo della patch 22, *airScale*. Il valore della distanza viene prima interpolato mediante la patch *interp* (Patch 23: *gainScale* - Punto 1) e successivamente va a risolvere l'equazione 2 (Patch 23: *gainScale* - Punto 2).



Patch 23: *gainScale*

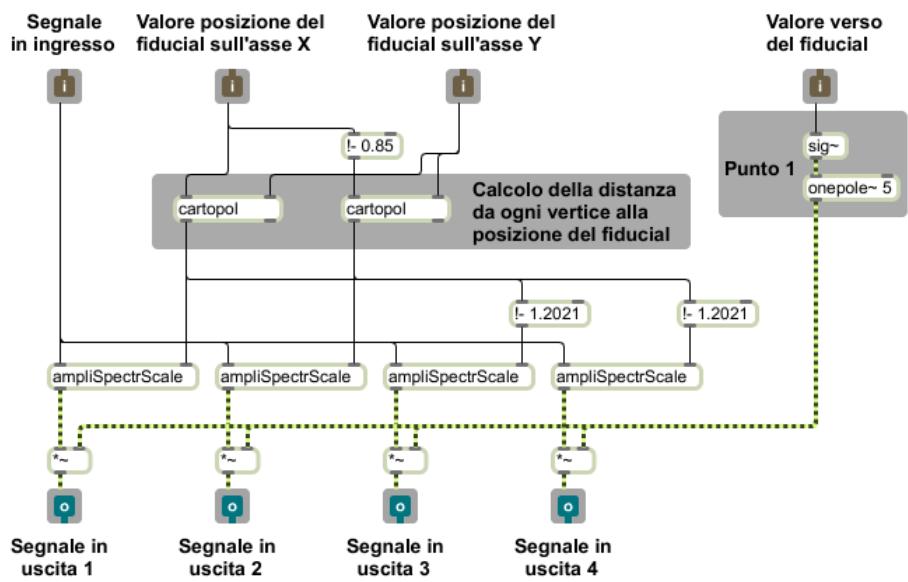
Le patch *airScale* e *gainScale* insieme all'external object *yarf2* (esaminata precedentemente), sono gli elementi costitutivi della patch *ampliSpectralScale*. Il segnale in ingresso attraversa per prima la patch *airScale* subendo il filtraggio di tipo passa-basso (Patch 24: *ampliSpectralScale* – Punto 1). Il segnale in uscita dalla patch *airScale* viene spartito in due ed ambedue i segnali convergono verso due copie della patch *gainScale* (Patch 24: *ampliSpectralScale* – Punto 2) la quale attenua l'intensità del segnale in ingresso. Il segnale che fuoriesce dalla seconda copia della patch *gainScale* passa all'interno dell'external object *yarf2* la quale applica un riverbero sul segnale (Patch 24: *ampliSpectralScale* – Punto 3). Infine il segnale in uscita dalla prima copia della patch *gainScale* ed il segnale in uscita dall'external object *yarf2* vengono sommati e mandati in uscita (Patch 24: *ampliSpectralScale* – Punto 4).



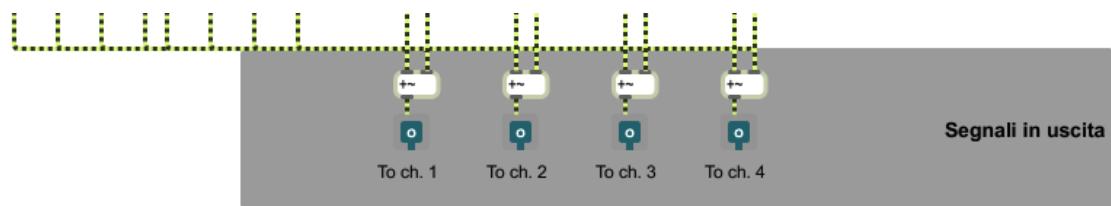
Patch 24: *ampliSpectralScale*

Nella patch appena sotto (Patch 25: *space*) si nota che il segnale in ingresso viene spartito e mandato in 4 copie della patch *ampliSpectralScale*. In ogni copia delle patch vie è richiesto anche il valore della distanza tra il fiducial ed il centro della Metis sugli assi X,Y; questo calcolo viene effettuato mediante la patch *cartopol*.

I quattro segnali in uscita dalle 4 copie delle patch *ampliSpectralScale* vengono scalati a loro volta dal valore dipendente dal verso del fiducial, il quale viene interpolato mediante la conversione del flusso dei dati in un segnale continuo avente un'ampiezza pari al valore corrente in ingresso (Patch 25: *space* - Punto 1).

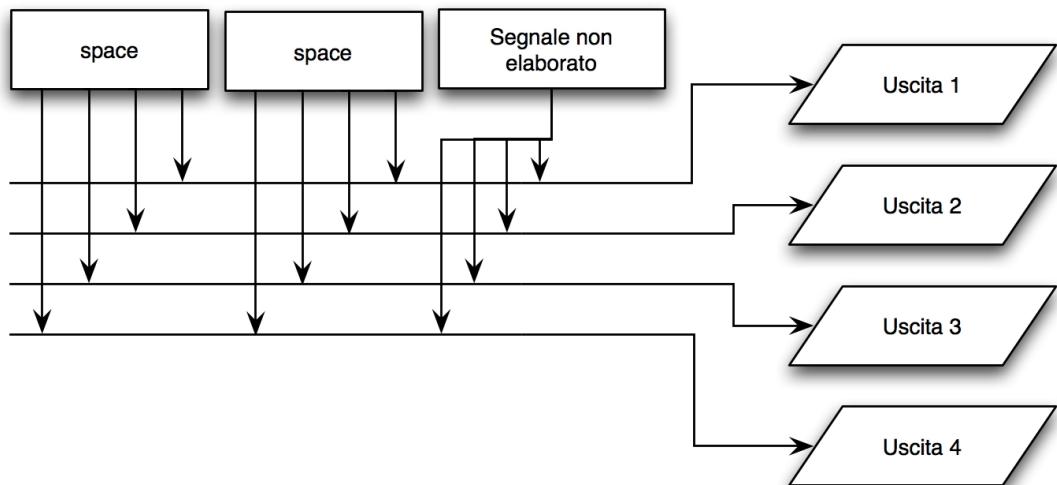


g) Somma dei segnali audio



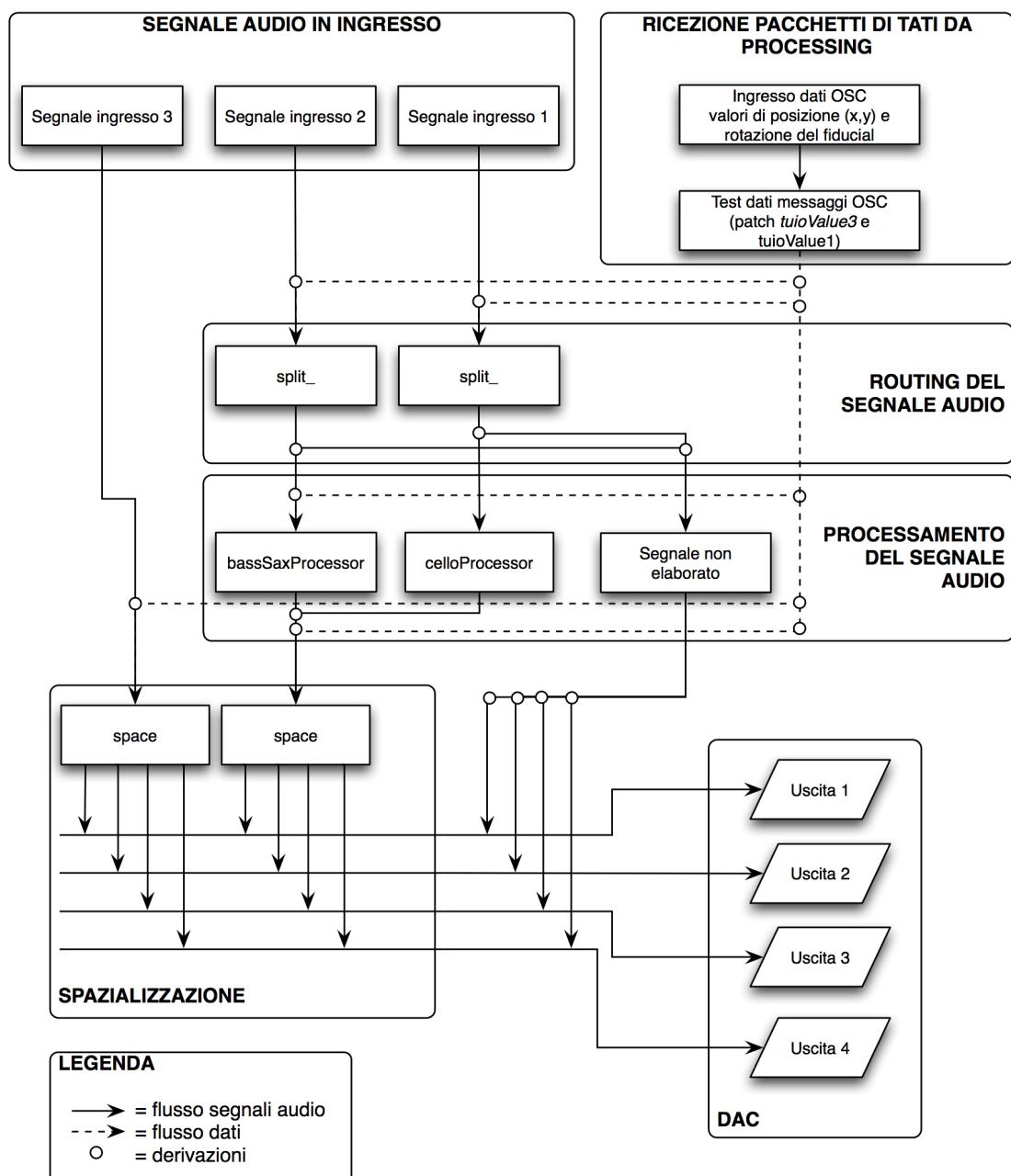
dettaglio "Patch 6: signalProcessing"

Tutti i segnali in uscita, spazializzati e non, vengono sommati e mandati alle relative uscite secondo lo schema 15 (appena sotto).



Schema 15: Somma segnali audio

Lo schema appena sotto ci riassume tutti i percorsi che effettuano sia i segnali sia i dati nella patch *signalProcessing*.



Schema 16: patch "signalProcessing"

Conclusione

Il percorso effettuato nello studio delle nuove interfacce come le T.U.I. ed in particolare della Metis, è stato lungo faticoso ed altrettanto interessante nella ricerca di metodi di comunicazione tra applicazioni, la ricerca di gesti utili all'interazione con l'interfaccia e ben fruibili mediante l'associazione del gesto con elaborazioni grafiche e sonore, lo sviluppo di applicazioni grafiche e sonore ed infine ricercare di possibilità di utilizzo nel live electronics della Metis. Questa ricerca ha ampliato di gran lunga i miei orizzonti e le mie idee di live electronics a tal punto d'iniziare ad impiegare la T.U.I. per progetti futuri come il live electronics, installazioni interattive ed applicazioni commerciali e professionali nel campo del sound design ed del sound reinforcement.

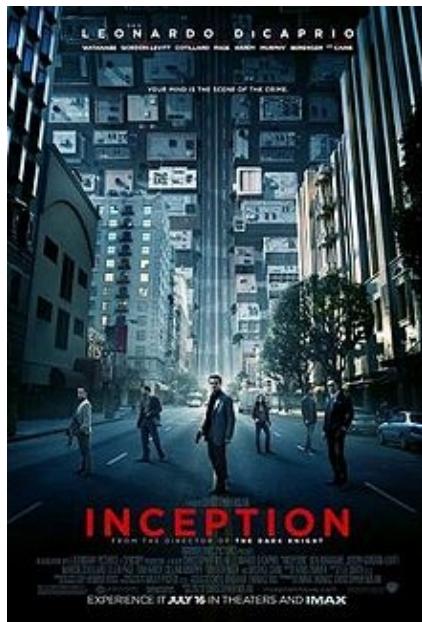
Appendice

Il Sogno nel sogno

I Ideazione. Dalla veglia al sogno multiplo

L'ideazione del brano “Il sogno nel sogno” è in parte riconducibile alla visione del film “Inception” di Christopher Nolan (2010). Nel film il protagonista ed il suo socio sono incaricati di estrarre o immettere nelle menti delle persone idee e pensieri, e lo fanno entrando nei loro sogni. Ma per impiantare un idea all'interno di una persona bisogna scendere ad un livello più profondo dell'inconscio, quindi s'inizia a ipotizzare di intervenire nei sogni all'interno dei sogni.

Affascinato da questo spunto narrativo ho cercato di creare nel mio brano una simile evoluzione, seguendo idealmente un individuo in stato di veglia che si addormenta ed inizia a sognare, e successivamente sogna di sognare, fino a che non torna alla stato di veglia.



*Illustrazione 44: locandina del film
"Inception"*

Da questo spunto ideativo sono scaturiti i ragionamenti, le riflessioni e scelte sul linguaggio musicale da adottare nella rappresentazione musicale di questa idea. La mia scelta è stata quella di utilizzare un linguaggio in parte concettuale (cioè un linguaggio dove i materiali musicali sono scelti per la loro capacità di simboleggiare determinate idee), in parte più strettamente legato alla tradizione elettronica. Un linguaggio musicale simbolico e descrittivo, ma anche musicalmente autonomo.

2. *Organico strumentale*

L'esamle strumentale è stato definito in maniera tale da soddisfare le esigenze compositive sia strettamente musicali che concettuali e semantiche. Ho scelto di utilizzare il pianoforte per rappresentare lo stato di veglia del sognatore. Essendo la veglia lo stato cosciente e razionale della nostra vita è stata associata ad un pianoforte, uno strumento completo e promotore di sonorità razionali e classiche. Il sogno è narrato dal violoncello uno strumento che permette, grazie anche all'elaborazione elettronica, di utilizzare sonorità più oniriche rispetto a quelle possibili sul pianoforte, ma ancora molto legate alle abitudini degli ascoltatori. In ultimo, il Sax basso, che racconta il sogno nel sogno, è stato scelto proprio per raccontare questo mondo mentale estremamente irreale per la sua singolare sonorità e le sue possibilità timbriche.

Tutto il controllo, sia delle operazioni di avvio e interruzione dei processi, sia di variazione dei parametri di elaborazione del suono, avviene tramite la *Metis*.

La Metis, si occupa di controllare e gestire la successione dei file audio necessari all'esecuzione e delle elaborazioni sonore provenienti da un supporto digitale nel caso si scelga l'ipotesi di un'esecuzione senza strumenti dal vivo, o dai microfoni, in caso in cui si scelga l'esecuzione con strumentisti. Il controllo come vedremo nel paragrafo successivo viene gestito mediante dei *fiducial* (controlli mobili, cfr. § 2.2) ai quali sono stati assegnati colori, numeri di identificazione e successivamente sono stati classificati per tipologia funzionale.

- Fiducial
 - n. 3 fiducial *Input*: gestiscono il segnale in ingresso (dai microfoni o da un supporto digitale)
 - n. 3 fiducial *Audio*: gestiscono la lettura di file audio
 - n. 2 fiducial *Elaboratori*: dedicati al controllo delle elaborazioni del segnale
 - n. 1 *Main*: spazializzazione e controllo del guadagno in uscita dalle patch di elaborazione.

Numero	Tipologia	Strumento	Colore
0	Main	Elaboratori	[Color Box]
1	Input	Pianoforte	[Color Box]
2		Violoncello	[Color Box]
3		Sax basso	[Color Box]
4	Elaborator e	Violoncello	[Color Box]
5		Sax Basso	[Color Box]
6	Player	Caduta 1	[Color Box]
7		Caduta 2	[Color Box]
8		Soffio	[Color Box]
9		Si b Elaborato	[Color Box]

3. Il materiale sonoro

I materiali sonori utilizzati per le parti strumentali, per i campioni ma anche il risultato sonoro delle elaborazioni delle parti strumentale sono il frutto di un'associazione concettuale tra l'ideazione del brano con la sua realizzazione musicale. Le elaborazioni del segnale effettuate sulle parti strumentali, oltre ad essere definite in base a criteri concettuali sono state stabilite in base alle caratteristiche morfologiche e spettrali dei materiali strumentali ed in vista del risultato sonoro voluto.

3.1 Parti strumentali

Veglia

Il sognatore suona una semplice melodia al pianoforte fino a quando non inizia man mano a sognare. Il tema della veglia è costituito da una semplice melodia suonata da un pianista (illustrazione 45).

Illustrazione 45: Tema del pianoforte

Alla settima battuta il protagonista del “racconto” inizia ad addormentarsi fino a quando non entra nella fase che ho denominato di *caduta*. Questo, musicalmente, vuol dire condurre questa melodia verso quello che sarà la *caduta*, mediante una variante del tema ottenuta alterando ritmicamente la melodia delle due mani.

Illustrazione 46: variante del tema del pianoforte e introduzione alla Caduta



Figura 4: Scena dal film Inception

Caduta.

La prima caduta vuole descrivere una discesa infinita preannunciando quello che sarà il continuo evolversi del brano, ovvero la discesa verso il sogno nel sogno. Questo concetto viene espresso musicalmente mediante la lettura in loop di un file audio contenente quattro scale discendenti ed editate in modo tale da emulare il comportamento dello Schepard Tone, di cui è evidente l'analogia percettiva con la cosiddetta Scala di Penrose (figura 5) . L'attivazione della lettura avviene mediante il fiducial *caduta*.

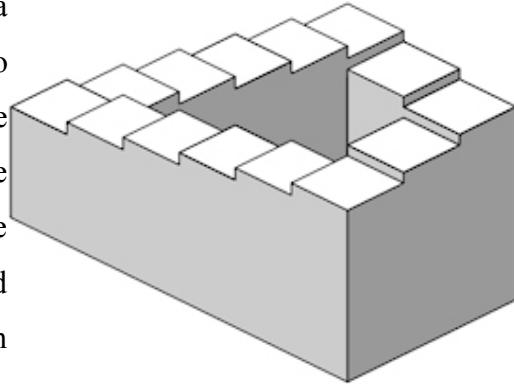


Figura 5: Scala di Penrose



Illustrazione 47: realizzazione della scala di Shepard mediante note di pianoforte campionate.



Illustrazione 48: scala per terze minore

Poco dopo l'attivazione del file audio il pianoforte esegue degli armonici al pianoforte eseguiti suonando una nota con il pedale della cordiera alzato e contemporaneamente appoggiando un dito sulla base della corda del pianoforte (sull'attaccatura della corda alla cordiera). Le note suonate sono successioni di scale composte da 4 note con un intervallo di terza minore (figura 7). L'intervallo scelto tra le altezze è dovuto alla volontà di non manifestare una tonalità ben precisa. Alla fine di ogni scala riprende un'altra scala di cui la prima nota è una seconda minore al di sotto della prima nota della scala precedente. Le illustrazioni 10 e 11 sono un esempio di scale consecutive, nella 10 vediamo che la scala inizia in MIb, mentre nella 11 in RE.

55

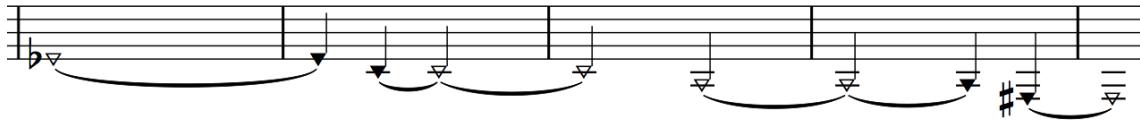


Illustrazione 49: scala per terze minori

60

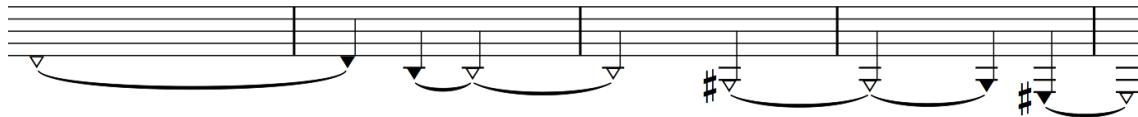


Illustrazione 50: scala per terze minori

Queste scale verranno eseguite fino a quando non si arrivi di nuovo allo stato di veglia.

Il sogno

Il nostro soggetto dormiente, ormai divenuto sognatore, apre una porta e scende una scala per ben 3 volte. Ogni volta che apre una porta trova delle scale sempre diverse. Questa sequenza di porte è tradotta musicalmente con le scale che vengono eseguite dal violoncello con delle tecniche d'esecuzione sempre diverse e anche tecniche di elaborazione del segnale sempre diverse. Ad ogni apertura della porta, musicalmente rappresentata da un'arcata strofinata su tutte le corde tra il ponte e la cordiera, vi è un incremento di una semiminima dagli armonici eseguiti al pianoforte (Illustrazione 51).

25



Illustrazione 51: i primi due pentagrammi sono riferiti al pianoforte mentre il terzo al violoncello.

Il primo episodio è composto da tre scale composte di 4 note con una durata di una minima per ogni nota. Queste tre scale vengono eseguite con l'arco e in crescendo da *mezzo-piano a forte*.

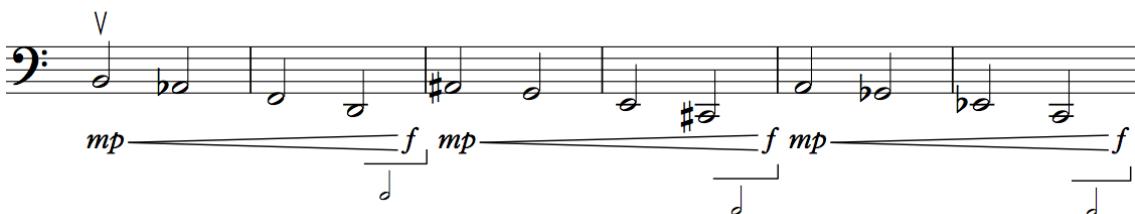


Illustrazione 52: episodio 1.

Il suono del violoncello a sua volta viene elaborato mediante un banco di filtri risonanti con un fattore Q molto elevato in modo tale da trattenere a lungo le note suonate ed i suoi armonici.

Il secondo episodio è costituito da 3 *ribattuti con legno* glissati con diverse densità di rimbalzi nel tempo. Il primo poco denso, il secondo di media densità ed il terzo molto denso di rimbalzi. L'intervallo scelto per il glissando è quello di una sesta maggiore, dalla prima e l'ultima nota di una scala per terze minori all'interno di un'ottava. Il segnale proveniente dal violoncello viene elaborato mediante un riverbero tenta di simulare un ambiente molto grande e riverberante.



Illustrazione 53: episodio 2

Il terzo episodio sempre costituito da scale di terze minori ma questa volta eseguite con la tecnica del pizzicato Bartok. I pizzicati vengono elaborati applicando sul segnale una modulazione di frequenza (F.M. – Frequency Modulation).

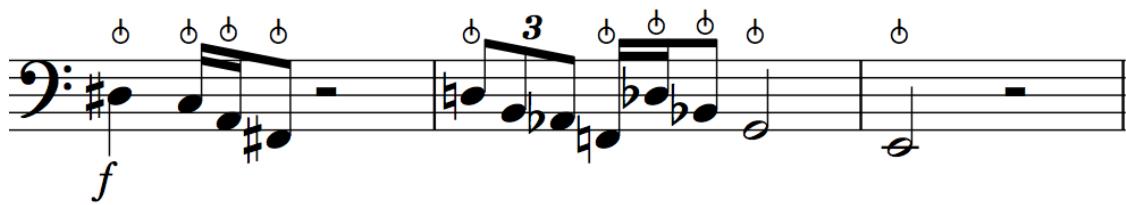


Illustrazione 54: episodio 3;

Il passaggio alla successiva “caduta” è formato da tre ripetizioni di un frammento sonoro così composto:

- un armonico artificiale in mezzo-forte, sfiorando alla quinta della durata di una semiminima a 100 di metronomo;
- un glissando di ottava discendente, in crescendo (da mezzo-piano a mezzo-forte), con durata di una minima con il punto; la sua altezza di partenza è la nota risultante dell'armonico appena eseguito;
- ed infine un armonico come il primo con la differenza che l'altezza a cui dev'essere sfiorato è l'altezza d'arrivo del glissando;

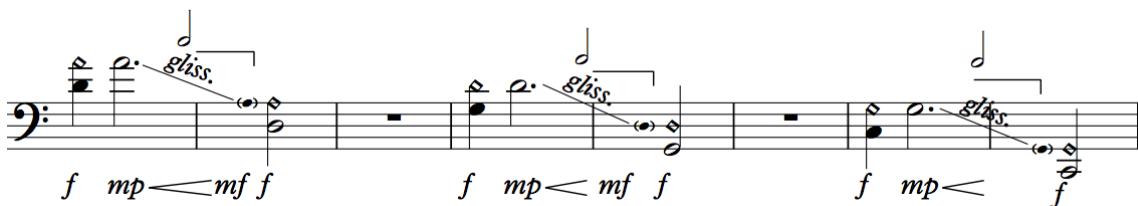


Illustrazione 55: modulazione alla caduta

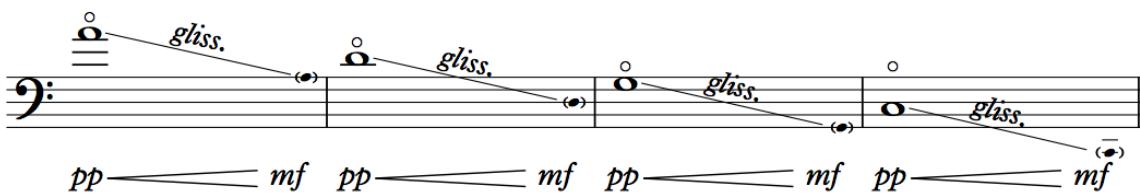


Illustrazione 56: caduta

Caduta 2

La seconda caduta, come la prima, è formata da un frammento pre-registrato letto in loop, composto da 4 armonici naturali glissati di ottava discendente, ognuno su una corda vuota. Nel contempo il violoncellista esegue le stesse note eseguite dal pianoforte, stessa altezza e durata ma in crescendo. Le note suonate hanno il compito di creare continuità con la note suonate dal pianoforte. Mentre la nota proveniente dal pianoforte decade, viene rinforzata da quella proveniente dal violoncello.

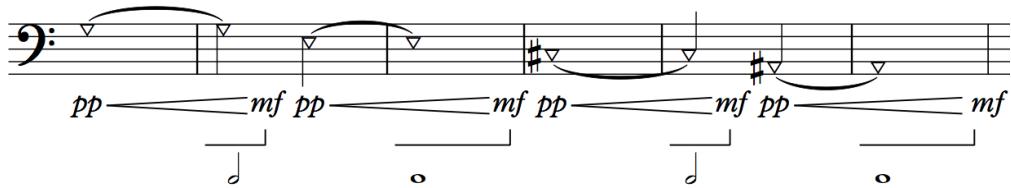


Illustrazione 57: *caduta eseguita dal violoncellista*

Il Sogno nel sogno. Il “Sogno nel sogno” è rappresentato dal sax basso. La “trama” del sogno racconta un'avventura su una nave da crociera in navigazione durante una giornata prima soleggiata e con il mare calmo, successivamente piovosa e con il mare in tempesta. Come il primo sogno, possiamo suddividere anche questo in tre episodi.

Il primo episodio descrive la prima parte della giornata. Il sassofonista esegue dei soffiati per simulare delle onde in mare in una giornata tranquilla e soleggiata (Illustrazione 58).

Illustrazione 58: *onde*

Il secondo episodio ha inizio con l'introduzione sul tavolo di un fiducial che mette in loop il frammento dell'episodio 1 (Illustrazione 40). Poco dopo vi è la presentazione della nave, mediante 3 esecuzioni del SIb basso in forte, che stanno a simulare il suono della tromba di una nave. La terza esecuzione viene elaborata mediante un banco di filtri risonanti e il suono in uscita dai filtri viene modulato in ampiezza mediante l'inviluppo di un campione di nota tenuta di cui è stato fatto il time stretching. Quest'ultima nota ha anche la funzione di preparazione al terzo episodio, i cui protagonisti sono una sequenza di colpi di chiave elaborati con 3 diverse tecniche d'elaborazione del segnale. I colpi di chiave sono distribuiti per registri, uno grave uno medio ed uno acuto e per densità di colpi, poco denso, medio denso e molto denso. I 3 tipi di elaborazione sono: filtraggio mediante un banco di filtri risonanti, granulazione e modulazione d'ampiezza con la tecnica dell'envelope follower; le tecniche d'elaborazione si possono anche combinare tra loro. Da battuta 99 alla battuta 102 possiamo avvertire una modulazione alla *risalita* (la risalita è il passaggio da un livello profondo ad un livello più superficiale) che tenta di esprimere la "quiete dopo la tempesta" ovvero un leggero venticello che, musicalmente, è un fischiato tra i denti con piccole variazioni di dinamica e piccoli glissati.

Illustrazione 59: *fischiatto tra i denti*

Risalita La risalita consiste nella graduale rimozione dei fiducial dalla T.U.I. in ordine contrario con cui sono stati inseriti, mentre le parti strumentali vanno man mano spegnendosi fino a far rimanere solo il pianoforte, simbolo del ritrovato stato di veglia.

Veglia Nello stato di veglia come all'inizio vi è la ripresa delle battute 2, 3 e 4.

3.2 Elaborazione elettronica

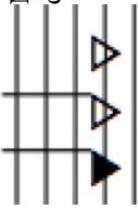
Le elaborazioni elettroniche effettuate sul materiale sonoro sono state scelte in maniera puntuale per sposarsi con esso e farne fuoriuscire le sue proprietà oniriche. Le elaborazioni si basano sull'utilizzo di filtri risonanti, il riverbero, la modulazione d'ampiezza e quella di frequenza.

I filtri risonanti vengono utilizzati sia nel “sogno” che nel “sogno nel sogno” per far brillare e rendere onirici (cioè alonati di riverbero, arricchiti di risonanze) suoni come: arcate tenute eseguite dal violoncello, suoni di soffiato e note tenute eseguite sax basso. Il riverbero è utilizzato sui ribattuti con legno eseguiti dal violoncello per creare un nuovo scenario sonoro chiuso ma dalle dimensioni infinite; la modulazione d'ampiezza è impiegata per mutare un suono come quello dei colpi di chiave del sax basso un soffiato o note tenute sfruttando l'inviluppo del suono dei colpi di chiave. La modulazione di frequenza è ottenuta sfruttando il segnale proveniente dal violoncello, il quale esegue dei pizzicati Bartok, come segnale di controllo sia del segnale portante, modulante che dell'indice di modulazione. Ultimo ma non meno importante, il time-stretch effettuato su alcuni campioni di violoncello e di sax basso. Come possiamo notare tutti i suoni tendono ad essere mutati, trasformati, snaturati proprio per concettualizzare l'idea di base che è quella del “sogno nel sogno”.

Balandino Di Donato

Il sogno nel sogno

Pianoforte



Nota suonata con il pedale abbassato ed con un dito dell'altra mano che stoppa la corda della nota eccitata appena dopo all'attaccatura della corda alla cordiera.

*Illustrazione 1:
armonico;*

Violoncello



*Illustrazione 2: sfregato tra ponte e
cordiera;*

La prima nota indica la durata complessiva del glissando e la seconda il punto d'arrivo.
Quando la durata del glissando dura più di una battuta e/o termina nella battuta successiva, la durata del glissando nella battuta successiva viene indicata al di sopra del pentagramma.

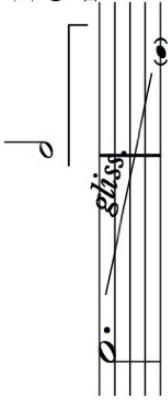
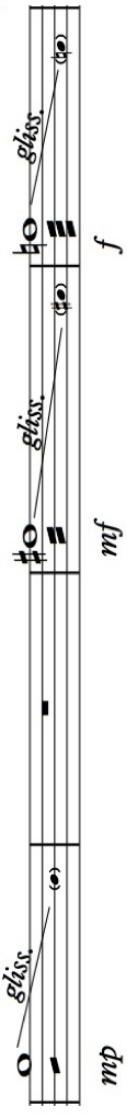


Illustrazione 3: glissato;

Ribattuto con legno



Le linee al di sotto della testa della nota che si riferiscono al ribattuto con legno indicano la densità di colpi ovvero la quantità di colpi all'interno della durata del Jeté glissato. Una linea bassa densità, due linee media densità, tre linee alta densità.

Sax



*Illustrazione
4: soffiatto
espirato;*



Illustrazione 5: soffiatto espirato;

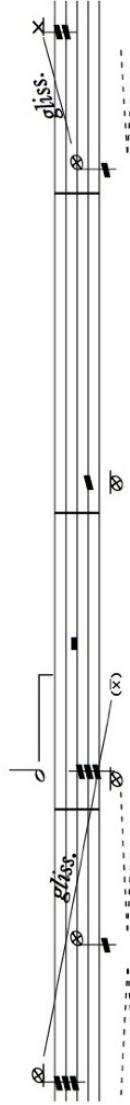


Illustrazione 6: colpi di chiave;

L'illustrazione 7, appena sopra indica l'esecuzione di colpi di chiave. L'altezza della nota non indica la diteggiatura esatta del colpo di chiave am pensi il registro entro cui deve avvenire una sequenza di colpi di chiavi che ha durata quanto la durata della nota ed una densità di colpi pari al numero delle linee sul gamba della nota. Una linea bassa densità, due linee media densità, tre linee alta densità. Il crescendo tratteggiato indica un crescendo di densità di colpi di chiave.

Elettronica



I trighi per l'elettronica sono 3. Sul primo troviamo la notazione riguardante il verso del fiducial, sul secondo la posizione sul tavolo e sul terzo il numero del fiducial.

La rotazione è indicata per mezzo di un cerchio che indica il fiducial ed una freccia che attraversa il suo diametro indicandone il suo verso. Con un punto all'interno di una quadrato è indicata la posizione del fiducial sulla Metis, mentre il numero indica l'ID del fiducial.

Per facilitare la corrispondenza e la riconoscibilità dei fiducial sono stati attribuiti anche dei colori ai fiducial. Ad i 3 fiducial input rosso per il numero 1, verde al 2 e blu al 3. Per i fiducial elaboratori: verde acceso al 4 e blu acceso al 5, in quanto sono i fiducial elaboratori dei fiducial 2 e 3. Il 6, 7, 8 e 0 neri. I primi 3 sono fiducial Audio mentre l'ultimo è il fiducial main dei fiducial elaboratori (4 e 5). Per ulteriori approfondimenti visualizzare il capitolo sui fiducial. Se racchiusa in un rettangolo. Se racchiusa in un rettangolo quando quel fiducial dev'essere inserito sul tavolo.

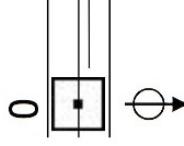


La "X" al posto del numero sta ad indicare che quel fiducial dev'essere rimosso.

*Illustration
e 7:
fiducial*



Il glissando viene utilizzato per indicare una variazione improvvisata della posizione del fiducial ma di durata pari a quella del glissando.
Mentre quando le battute sono completamente vuote, senza nemmeno le pause vuol dire che la posizione, rotazione e durata dei gesti sono del tutto improvvisati.



*Illustration 9: variazione di
posizione del fiducial di tipo
improvvisato*

Il sogno nel sogno

J = 100

Piano

The musical score consists of two staves. The top staff is for the right hand (treble clef) and the bottom staff is for the left hand (bass clef). Measure numbers 0" through 33" are indicated above the staves. Various dynamics and performance instructions are included, such as *mf*, *mf*, *p*, *pp*, *rall.*, and grace notes. Measure 36" is attributed to "Balandino Di Donato". A bracket labeled "Fiducial" connects the two staves. Below the staves, a track labeled "Fiducial" shows a sequence of numbered boxes (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16) with a circled "⊕" symbol above them, and the instruction "HAND UP" above the first box.

2 39" 42" 44" 48" 52" 56" 100" 104" 108" 112" 116" 120" 124"

Pno.

Fiducial 6

Piano

Fid.

1'28" 1'32" 1'36" 1'40" 1'44" 1'48" 1'52" 1'56" 2'00" 2'04" 2'08" 2'12" 2'16" 2'20" 3
 30 31 32 33 34 35 36 37 38 39 40 41 42 43

Vlc.
 Pno.

2 0 4 0

1'28" 1'32" 1'36" 1'40" 1'44" 1'48" 1'52" 1'56" 2'00" 2'04" 2'08" 2'12" 2'16" 2'20" 3
 30 31 32 33 34 35 36 37 38 39 40 41 42 43
 Vlc.
 Pno.

2 0 4 0

312^a 316^a 320^a 324^a 328^a 332^a 336^a 340^a 344^a 348^a 352^a 356^a 400^a
 56 57 58 59 60 61 62 63 64 65 66 67 68

Pno.
 Vlc.
 Cello
 Fig. 6
 Pno.

6

VII

100

A musical score for Sax Basso. The staff begins with a treble clef, followed by a dynamic marking of *pp*, a tempo marking of *mf*, and a key signature of three sharps. The score consists of two measures. Measure 3 starts with a half note followed by a whole note. Measure 7 starts with a half note followed by a whole note. Both measures end with a fermata over the last note.

6

122

500" 504" 508" 512" 516" 520" 524" 528" 532" 536" 540" 544" 548" 552" 7

83 84 85 86 87 88 89 90 91 92 93 94 95 96

Vlc. Sax Basso

Fid. 6 Fid. 7 Fid. 8

11c.

Coda di chiave

Viola: Measures 11-12. Dynamics: *mf pp*, *mf pp*, *mf pp*, *mf pp*. Articulation: *pizz.*

Sax Basso: Measures 11-12. Dynamics: *p*, *p*, *p*, *p*.

Fid. 7
Vic.

10 7'40" 7'44" 7'48" 7'52" 7'56" 8'00" 8'04" 8'08" 8'12" 8'16" 8'20" 8'24" 8'28" 8'35

Pno. 123 124 125 126 127 128 129 130 131 132 133 134

Vcl. *Ribattuto con legno* *sfz* *sfz* *sfz* *sfz* *f*

Sax Basso *mf* *mf* *f*

Fid. *mf* *f*

Fid. 6 *mf* *f*

Pno. *pp* *mf*

Vcl. *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp*

Sax Basso *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp*

Fid. 7 *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp*

Vcl. *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf*

Sax Basso *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp*

Fid. 8 *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp*

Vcl. *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf*

Sax Basso *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp* *mf* *pp*

8'32" 8'36" 8'40" 8'44" 8'48" 8'52" 8'56" 9'00" 9'04" 9'08" 9'12" 9'16" 9'20" 11

136 137 138 139 140 141 142 143 144 145 146 147 148

Pno.

Vlc.

Sax Basso

fischiatto (tra i denti)

pfp — *mf* > *mp* < *mf* — *f* — *mf* < *f* > *pfp*

Fid.

Fid. 6

Pno.

Fid. 7

Vlc.

pfp — *mf* > *mp* — *mf* *pfp* — *mf*

pfp — *mf* *pfp* — *mf* *pfp* — *mf* *pfp* — *mf* *pfp* — *mf* *pfp* — *mf* *pfp* — *mf* *pfp* — *mf*

Fid. 8

Sax Basso

pfp — *mf* > *mp* — *mf* *pfp* — *mf*

12

921" 924" 928" 930" 933" 935" 937" 940" 943"

Pno.

Frid.

Fig. 6 Pno.

Fig. 8 Sax Basso

Sitografia

Wikipedia - <http://en.wikipedia.org/>

Processing - <http://www.processing.org/>

Tuio - <http://www.tuio.org/>

reactIVision - <http://reactivision.sourceforge.net/>

OSC – Open Sound Control - <http://opensoundcontrol.org/>

Cycling 74 - <http://cycling74.com/>

NUI Group - <http://nuigroup.com/>

NIME - <http://www.nime.org/>