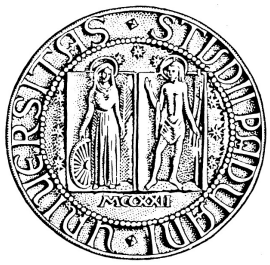


Università di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Elettronica

# Spazializzazione del suono: integrazione di un algoritmo real-time su un sistema di motion-capture

*Laureando:*
Francesco Cavazzana

*Relatore:*
Dott. Federico Avanzini

Anno Accademico 2010/2011

# Sommario

Questo documento ha lo scopo di illustrare un algoritmo *real-time*[4] ottimizzato per il *3D sound rendering* in un sistema di *motion-capture*. Di seguito una descrizione del contenuto dei capitoli:

**Primo capitolo** É un capitolo introduttivo alla spazializzazione del suono, nel quale si espongono i modelli matematici più utilizzati per modellizzare, tramite tecniche binaurali, lo spostamento di una sorgente nello spazio. Verranno esposti gli effetti causati delle parti del corpo sul suono che giunge alle orecchie e, quindi, dei parametri principali (quali ITD ed ILD) e delle funzioni che descrivono i segnali binaurali in base alla posizione della sorgente rispetto alla testa (*HRTF*). Vengono discussi, in seguito, la qualità e i limiti del modello considerato. Si parlerà brevemente di alcuni algoritmi di *3D sound rendering* e, in particolare, ci si focalizzerà sul metodo utilizzato nel progetto.

**Secondo capitolo** Descrive *PhaseSpace Impulse* (il sistema di *motion-capture* utilizzato), per permettere una miglior conoscenza dell'apparecchiatura in uso e fornire un'idea dell'ambiente su cui è stato sviluppato ed integrato l'algoritmo *real-time* in esame.

**Terzo capitolo** Il capitolo dà delle basi cognitive ed operative per l'utilizzo del software *Pure Data* (e quindi Pd-Extended): si introduce dapprima il sistema nel suo insieme, elencandone le caratteristiche principali e spiegando le funzioni dei vari blocchi, per poi andare nel dettaglio di alcune particolari funzioni.

**Quarto capitolo** In questa parte del lavoro verranno esposte in maniera dettagliata l'*external* e la *patch* realizzate. Saranno infatti analizzate tutte le funzioni implementate, eventualmente con la relativa descrizione matematica, e tutte le parti della *patch* precedentemente divise in moduli. Infine si fornirà un esempio di utilizzo della stessa con relative valutazioni sul risultato ottenuto.

# Indice

# Capitolo 1

# La spazializzazione del suono: l'audio binaurale

Il suono prodotto da una sorgente, durante il propagarsi verso l'ascoltatore, è soggetto a varie trasformazioni come ad esempio le perdite di energia isofrequenziali dovute alla distanza percorsa (*path loss*), quelle dovute alla frequenza delle onde e quelle eventualmente causate da oggetti interposti tra sorgente ed ascoltatore (*shadowing*). I flussi sonori che giungono alle orecchie dell'ascoltatore risultano quindi essere differenti: ci sarà infatti un ritardo tra i due segnali, a causa dalle lunghezze dei percorsi, e una differenza di energia, influenzati anche dal ruolo attivo dell'ascoltatore nella trasformazione del suono.

## 1.1 Il suono a livello del timpano

In questa sezione si assumerà che tutta l'informazione, utilizzata dall'ascoltatore per elaborare la localizzazione di una sorgente sonora, sia contenuta nei segnali di pressione acustica a livello dei timpani delle due orecchie: questi due segnali combinati, infatti, portano all'ascoltatore l'*informazione spaziale* del suono. L'obbiettivo sta nel comprendere e simulare come il suono sia trasformato, nel suo percorso verso i timpani, dalle parti del corpo limitrofe: la testa, la pinna, il condotto uditivo e le spalle. Tutte le considerazioni fatte in seguito fanno riferimento a sorgenti sonore posizionate nel cosiddetto *campo lontano*, cioè a distanze superiori al metro.

### 1.1.1 La testa

Le nostre orecchie non sono oggetti isolati nello spazio ma sono, invece, posizionate alla stessa altezza agli antipodi di un oggetto ad alta impedenza acustica: la testa[1]. Questa costituisce un ostacolo per la libera propagazione del suono ed ha due principali effetti:

- **ITD** (*Interaural Time Difference*), causato dal fatto che le onde sonore devono percorre una maggiore distanza per raggiungere l'orecchio più lontano. Uno studio approssimato di questo fenomeno si può effettuare facendo delle ipotesi semplificative: si suppongono la sorgente lontana e la testa sferica. La prima assunzione implica che le onde sonore che colpiscono la testa sono onde piane; la distanza ulteriore $\Delta x$, che un raggio sonoro dovrà percorrere per raggiungere l'orecchio più lontano, viene stimata grazie ad elementari considerazioni geometriche (vedi figura 1.1a):

  $$ITD \sim \frac{a}{c}(\theta + \sin\theta) \tag{1.1}$$

  Dove $c$ è la velocità del suono[1], $a$ rappresenta il raggio della testa (tipicamente $a = 8,5cm$) e $\theta$ l'angolo di *azimuth*[2], che definiscono la direzione della sorgente sonora sul piano orizzontale; la formula mostra che il minimo ITD si ha quando la sorgente è frontale ($\theta = 0$), mentre è massimo quando la sorgente è ai lati della testa ($\theta = \pi/2$)[1].

[1] Nell'aria a temperatura ambiente è circa 343 m/s.

[2] É l'angolo tra la direzione frontale dell'ascoltatore e la direzione congiungente la sorgente sonora e il centro della testa.

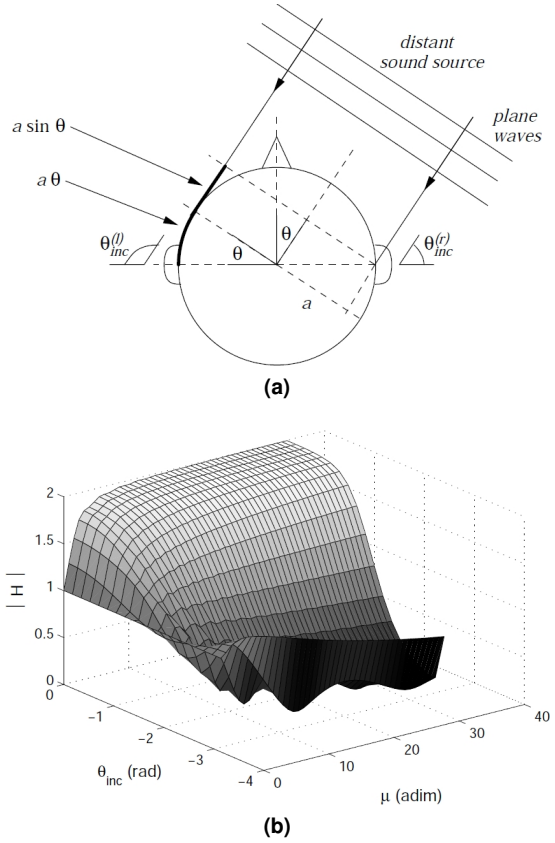



**Figura 1.1:** Stima dell'*ITD* nel caso di sorgente sonora lontana e testa sferica **(a)** e il grafico della funzione $|H_{sfera}(\rho, \theta_{inc}, \mu)|$ **(b)**

- **ILD** (*Interaural Level Difference*), poiché all'orecchio più lontano giunge un suono di minor intensità a causa della testa che introduce un'attenuazione. Mentre si può accettabilmente supporre che l'ITD non lo sia, l'ILD è fortemente legato alla frequenza: a basse frequenze (per lunghezze d'onda dell'ordine del diametro della testa) non vi è quasi alcuna differenza, mentre ad alte frequenze diventa particolarmente significativo.

  Questo parametro può essere analizzato supponendo, come già fatto i precedenza, che la testa sia perfettamente sferica (di diametro $a$) e che la sorgente sonora sia situata ad una distanza $r > a$ dal centro della sfera.

  É solito utilizzare costanti normalizzate: $\mu = \omega a/c$ (*frequenza normalizzata*) e $\rho = r/a$ (*distanza normalizzata*). Se si considera un punto sulla sfera allora, la diffrazione di un'onda acustica causata dalla sfera nel punto scelto, si può esprimere con la funzione di trasferimento:

$$H_{sfera}(\rho, \theta_{inc}, \mu) = -\frac{\rho}{\mu} \sum_{m=0}^{+\infty} (2m+1) P_m(\cos\theta_{inc}) \frac{h_m(\mu\rho)}{h'_m(\mu)}, \tag{1.2}$$

  dove $P_m$ e $h_m$ sono, rispettivamente, i polinomi di Legendre di ordine $m$-esimo e la funzione sferica di Hankel; $\theta_{inc}$ rappresenta l'angolo di incidenza.

  Dal grafico in figura 1.1b si può notare come, a basse frequenze, la funzione di trasferimento non sia dipendente dalla direzione: infatti il modulo $|H_{sfera}| \simeq 1$ per ogni $\theta_{inc}$. Quando $\mu > 1$ la dipendenza dall'angolo di incidenza diventa molto sensibile. Per $\theta_{inc} \simeq 1,74$ *rad* (circa 100 gradi) la risposta $|H_{sfera}|$ è pressoché piatta.

  Si noti, inoltre, come il minimo della risposta non si abbia per $\theta_{inc} = \pi$: questo è dovuto al fatto che tutte le onde sonore che si propagano attorno alla sfera arrivano in quel punto in fase; a frequenze elevate questo fenomeno diventa molto meno presente e la parte posteriore della sfera risulta essere in una zona di ombra sonora[1].

### 1.1.2 L'orecchio esterno

La parte esterna dell'orecchio consiste nella pinna e nel canale uditivo fino al timpano, su cui si porrà l'attenzione di questa sezione. La pinna (figura 1.2a) ha dei bassorilievi caratteristici per ogni persona ed è connessa al canale uditivo (figura 1.2b), descrivibile in prima approssimazione come un tubo di larghezza costante con pareti ad alta impedenza acustica: esso si comporta come un risonatore monodimensionale, mentre la pinna svolge la funzione di antenna acustica. Le cavità della di quest'ultima amplificano alcune frequenze, mentre la sua particolare geometria introduce interferenze che ne attenuano altre. Questo effetto di filtraggio della pinna può essere interpretato studiando le riflessioni introdotte oppure concentrandosi nel dominio della frequenza[1].

RIFLESSIONI: in figura 1.2c sono illustrate due differenti direzioni di arrivo, entrambi i casi vi sono un percorso diretto ed uno più lungo (dovuto appunto alla riflessione della pinna) dalla sorgente al canale uditivo. A frequenze moderatamente basse la pinna convoglia energia sonora e i segnali dei due percorsi arrivano in fase; a frequenze alte il ritardo tra i due segnali introduce uno sfasamento che provoca un'interferenza distruttiva. La maggiore interferenza si ottiene quando la differenza di lunghezza dei due percorsi equivale a $\lambda/2 + k\lambda$ (con $k \in \mathbb{N}$ e $k > 0$), cioè quando i due segnali sono in opposizione di fase: questo produce un *pinna notch*.

RISONANZE: tramite lo studio delle risonanze (figura 1.2d) si può effettuare una più accurata analisi attraverso misurazioni di risposte in frequenza, eseguite utilizzando un'imitazione della pinna e del canale uditivo con una terminazione in alta impedenza. Le risonanze sono dovute sia alla *cavum conchae* che al prolungamento del canale, circa del 33%, dovuto alla pinna.

In conclusione si può dire che la pinna e il canale uditivo formano una sorta di risonatore acustico, le cui frequenze di risonanza dipendono dalla direzione e dalla distanza della sorgente[1].

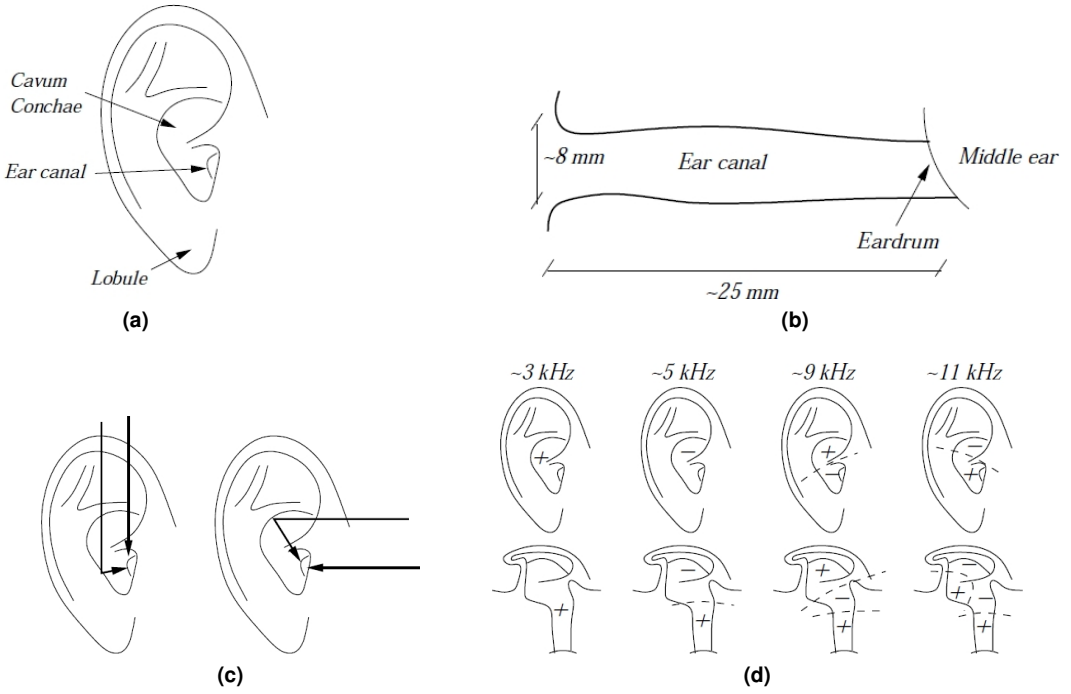



**Figura 1.2:** In alto l'orecchio esterno **(a)** e il canale uditivo **(b)** mentre, in basso, le riflessioni **(c)** e le risonanze **(d)** della pinna

### 1.1.3 Le spalle e il torso

La formazione del suono che arriva al timpano è influenzata, oltre che dall'orecchio esterno, anche dal torso. Questo, assieme alle spalle, introduce due importanti contributi: un'ulteriore riflessione sonora, che va a sommarsi al suono diretto, e un effetto di oscuramento, che attenua le onde sonore provenienti dal basso.

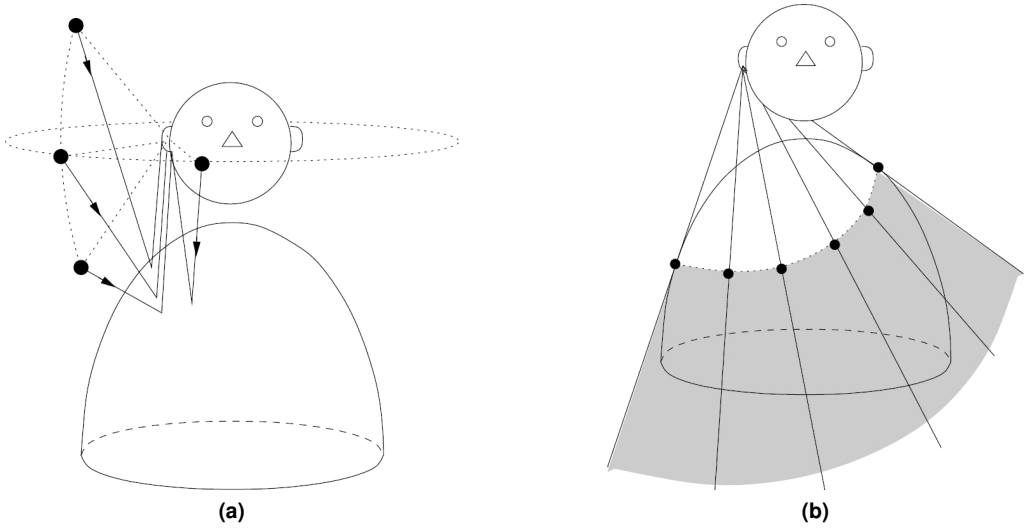



**Figura 1.3:** Torso e spalle: riflessioni **(a)** e oscuramenti **(b)**

La geometria del torso è abbastanza complicata, ma è possibile derivare una descrizione semplificata considerando un torso ellissoidale sotto una testa sferica: questo modello è anche detto *snowman model*.

RIFLESSIONI: se si misura la risposta impulsiva all'orecchio destro, ad esempio come in figura 1.3a, si può notare come l'impulso iniziale sia seguito da una serie di altri impulsi, il ritardo dei quali cambia in base all'elevazione. Questo fenomeno di riflessione è causato del torso.

Si può utilizzare la geometria semplificata dello *snowman model* per calcolare analiticamente i ritardi dei raggi riflessi, fornendo i parametri del modello e la collocazione della sorgente sonora, tenendo conto che:

- il ritardo tra il suono diretto e il riflesso non varia molto se la sorgente sonora si muove su una circonferenza nel piano orizzontale, in particolare se il raggio è grande rispetto alla testa;
- i ritardi variano sensibilmente se la sorgente sonora si muove verticalmente.

Nel dominio della frequenza le riflessioni del torso si comportano come un *comb filter* (filtro a pettine) introducendo *notch* periodici nello spettro: le frequenze dove questi si manifestano sono legate ai ritardi e perciò il *pattern* cambia in base all'elevazione della sorgente. Il *notch* più basso corrisponde al ritardo maggiore.

OSCURAMENTO: man mano che la sorgente scende raggiunge un punto al di sotto del quale le riflessioni spariscono, mentre emerge l'effetto di oscuramento. In figura 1.3b i raggi disegnati dall'orecchio ai punti di tangenza della parte superiore del torso delimitano il cono di oscuramento.

Sebbene il torso e le spalle non introducano un effetto acustico determinante, sono importanti alle basse frequenze, dove la risposta della pinna è pressoché piatta; in termini di range di frequenza gli effetti del torso sono quindi complementari a quelli della pinna[1].

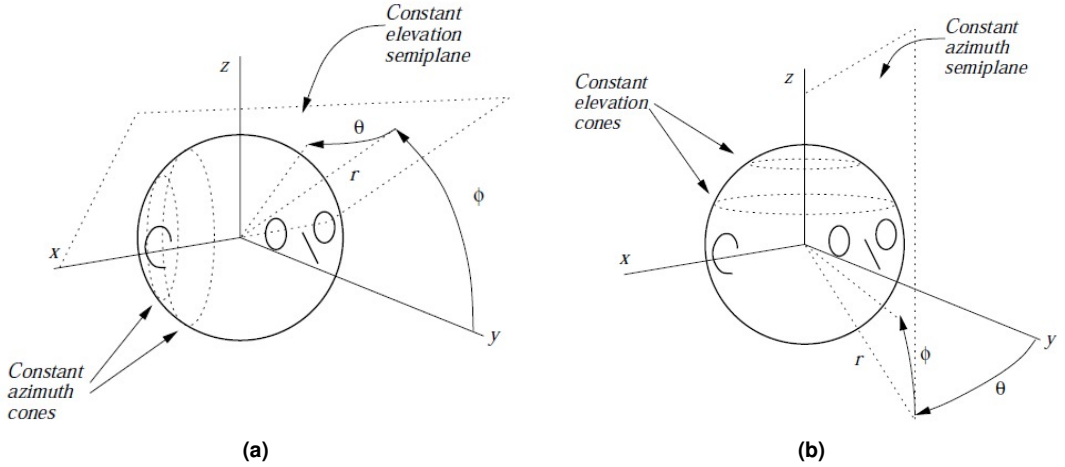



**Figura 1.4:** Coordinate polari-interaurali **(a)** e coordinate polari-verticali **(b)**

### 1.1.4 *Head Related Transfer Function (HRTF)*

Tutte le considerazioni fatte in precedenza sono da considerarsi lineari: possono quindi essere descritte da funzioni di trasferimento ed i loro effetti sommati. Tuttavia è anche possibile determinare univocamente la pressione sonora al timpano grazie alla risposta impulsiva dalla sorgente al timpano: questa procedura è detta *Head-Related Impulse Response (HRIR)* e, la sua trasformata di Laplace, *Head-Related Transfer Function (HRTF)*.

Le *HRTF* sono quindi influenzate, oltre che dal mezzo in cui si propagano, dalle varie zone del corpo dell'ascoltatore, in particolare quelle analizzate in precedenza; in sintesi:

- la *testa* ha un'effetto schermante sulle onde ad alta frequenza;

- l'*orecchio esterno* possiede delle particolari zone di risonanza che attenuano o amplificano particolari frequenze e riflette le onde con lunghezza d'onda abbastanza limitata a seconda della loro direzione;

- le *spalle* e il *torso* attenuano le onde sonore provenienti da sotto l'ascoltatore, mentre riflettono quelle provenienti dall'alto[2].

É solito utilizzare le coordinate sferiche, dove gli angoli di *azimuth* e di *elevazione*[3] sono indicati rispettivamente con $\theta$ e $\phi$, mentre la coordinata radiale è indicata con il simbolo $r$. I due sistemi di coordinate sferiche utilizzati in letteratura sono:

- *polare-interaurale* (figura 1.4a), nel quale l'*elevazione* è misurata come angolo tra la proiezione ortogonale della sorgente sul piano $yz$ e la coordinata $y$ mentre, l'angolo di *azimuth*, è l'angolo compreso tra la sorgente e la relativa proiezione sul piano $yz$;

- *polare-verticale* (figura 1.4b), dove l'*azimuth* è misurato tra la proiezione ortogonale sul piano $xy$ della sorgente e l'asse $y$, mentre l*elevazione* è calcolata tra la sorgente e la relativa proiezione sull'asse $xy$.

In alcuni casi si indicano le *HRTF* come $H^{(l),(r)}(r,\theta,\phi,\omega)$, dove $(l)$ ed $(r)$ si riferiscono alle *HRFT* ai due orecchi. Quando $r \to \infty$ (che in pratica significa $r > 1$ m) la sorgente e detta essere nel *campo lontano*: in questo caso la notazione utilizzata è $H^{(l),(r)}(\theta,\phi,\omega)$. Infine, nell'ipotesi di una simmetria perfetta, si scriverà semplicemente $H(\theta,\phi,\omega)$, con:

$$H^{(r)}(\theta,\phi,\omega) = H(\theta,\phi,\omega)$$

$$H^{(l)}(\theta,\phi,\omega) = H(-\theta,\phi,\omega)$$

[3] É l'angolo tra il segmento congiungente la sorgente sonora e il centro della testa e il piano orizzontale.

## 1.2 Percezione della sorgente sonora

In questa sezione si tratteranno brevemente le interferenze acustiche che possono influenzare la localizzazione della sorgente sonora. É importante essere consapevoli delle limitazioni introdotte dalle ipotesi semplificative.

### 1.2.1 Percezione dell'azimuth

La localizzazione sul piano orizzontale massimizza le differenze di eventi sonori che si verificano intorno all'ascoltatore, grazie ai parametri ITD ed ILD (e la relativa *Duplex Theory*), considerati i parametri chiave per la percezione dell'*azimuth*.

Si consideri un'onda sinusoidale che raggiunge entrambi gli orecchi. Il parametro ITD:

- alle *basse frequenze* è facilmente riconoscibile (vedi figura 1.5a), poiché sposta la forma d'onda di una frazione di periodo. Qualitativamente si può dire che se la metà della lunghezza d'onda del segnale è maggiore della grandezza della testa, allora è possibile identificare univocamente la fase;
- alle *alte frequenze*, invece, è ambiguo poiché possono esserci diversi cicli di sfasamento (vedi figura 1.5b). La frequenza critica, al di là della quale può manifestarsi questo effetto indesiderato, è di circa $1.5kHz$.

Per quanto concerne il parametro ILD la situazione è opposta: alle basse frequenze la risposta è sostanzialmente piatta mentre alle alte frequenze il suo effetto si fa sempre più marcato. Si può affermare, in prima approssimazione, che i parametri ITD e ILD siano complementari[1].

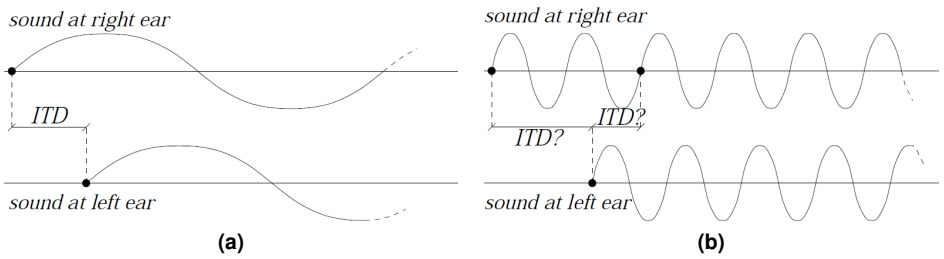



**Figura 1.5:** Differenza temporale alle orecchie: ITD non ambiguo **(a)**, ITD ambiguo **(b)**

Le informazioni fornite da ITD ed ILD possono essere ambigue: se assumiamo infatti la testa di forma sferica, una sorgente sonora collocata davanti all'ascoltatore ad un certo $\theta$ e una seconda collocata posteriormente ad un angolo $\pi - \theta$ producono i medesimi ITD ed ILD. In realtà i veri valori non sono uguali poiché: la testa umana non è sferica, vi sono asimmetrie ed altre caratteristiche facciali ed infine le orecchie non sono esattamente posizionate in corrispondenza dell'asse $x$. L'ambiguità di questi due parametri provoca nell'ascoltatore la cosiddetta *front/back confusion*, cioè un riconoscimento erroneo dell'angolo di *azimuth*, a confondere appunto l'anteriore col posteriore, e viceversa[1].

### 1.2.2 Percezione dell'elevazione

La localizzazione di una sorgente sonora, per elevazioni non nulle, risulta più complessa rispetto a quella sul piano orizzontale. In figura 1.6 vengono mostrate delle sorgenti sonore collocate su una superficie conica, detta *cono di confusione*, che estende il concetto di *front/back confusion*. Questa situazione è puramente teorica: in realtà i parametri ITD ed ILD dei punti sul cono non saranno mai esattamente identici a causa, come già discusso, delle caratteristiche facciali e delle asimmetrie della testa.

Gli effetti direzionali introdotti dalla pinna possono risolvere questi casi di confusione, in particolare per la localizzazione verticale. Il ruolo della pinna nel miglioramento della localizzazione verticale può essere verificato sperimentalmente, ad esempio, comparando giudizi in normali condizioni rispetto ad altri dove entrambe le pinne sono bypassate o coperte. Infatti la localizzazione verticale può essere eseguita anche quando un orecchio è completamente occluso: il risultato di questa prova supporta l'idea che le caratteristiche spettrali della pinna possono funzionare anche mononauralmente[1].

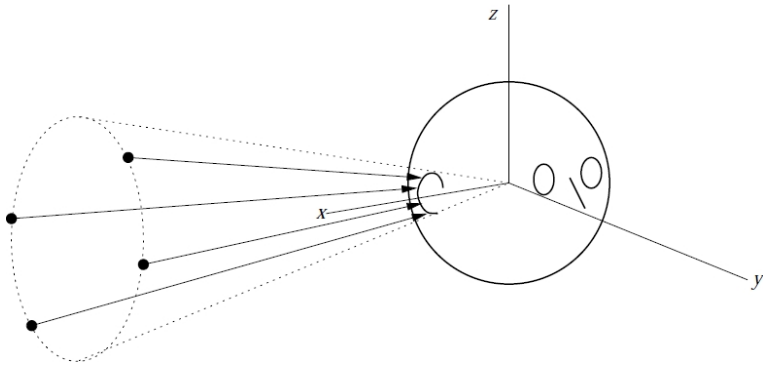



**Figura 1.6:** Cono di confusione

### 1.2.3 Percezione della distanza

Stimare la distanza di una sorgente sonora risulta ancora più difficile rispetto alla stima del suo angolo di elevazione. In assenza di altre informazioni l'*intensità* del suono è l'indizio principale utilizzato dall'ascoltatore per la collocazione spaziale della sorgente, che con l'esperienza associa il posizionamento di sorgenti sonore in base all'aumento o alla diminuzione dell'intensità. In una camera anecoica[4] la riduzione di intensità sonora è inversamente proporzionale al quadrato della distanza, il guadagno d'intensità sonora sarà quindi:

$$g = \frac{1}{d^2}, \tag{1.3}$$

dove $d$ rappresenta la distanza (modello qualitativo). Altre informazioni possono essere contenute in altri fenomeni, quali il riverbero e l'assorbimento dell'aria.

### 1.2.4 Lateralizzazione ed esternalizzazione

Nel caso di un sistema binaurale, e quindi basato sull'utilizzo di cuffie, si possono presentare due tipi di localizzazione:

LATERALIZZAZIONE: questo termine è tipicamente utilizzato per indicare uno speciale caso di localizzazione, cioè quando la sorgente sonora è percepita all'interno della testa, principalmente lungo l'asse interaurale (vedi asse $x$ in figura 1.4); si può indurre questa percezione manipolando opportunamente ITD ed ILD in un sistema binaurale. Questo fenomeno illustra un esempio fondamentale di posizionamento virtuale della sorgente nello spazio; infatti se vengono trasmessi due segnali monoaurali perfettamente identici la sensazione sarà che la sorgente sia collocata al centro della testa. Variando ITD ed ILD, inoltre, si avrà la sensazione che la sorgente si sposti lungo l'asse interaurale da un'orecchio all'altro sempre all'interno della testa: questo fenomeno è anche chiamato *inside-the-head localization* (IHL)[1].

ESTERNALIZZAZIONE: è un tipo di localizzazione molto difficile da riprodurre in un sistema binaurale basato su cuffie, dove risulta molto più semplice l'IHL; infatti non è del tutto chiaro quali siano i parametri supplementari più efficaci per rendere possibile questo tipo di localizzazione, tuttavia si è notato come l'aggiunta di un riverbero, naturale o artificiale, possa migliorarla sensibilmente[1].

## 1.3 Algoritmi per il *3D sound rendering*

Prima di analizzare gli algoritmi per il *3D sound rendering* dobbiamo capire che questi cambiano in funzione del sistema che si va ad utilizzare:

[4] É un ambiente strutturato in modo da ridurre il più possibile la riflessione sulle pareti. Il termine, dal greco, significa infatti "*privo di eco*", ed è particolarmente utile per studi che comportano la necessità di ricreare, in un ambiente chiuso, condizioni simulate di spazio aperto di dimensione infinita, come conseguenza dell'assenza di riflessioni[5].

- *sistema stereo*, è il più semplice dei sistemi che supportano un suono spazializzato. La percezione della sorgente sonora può essere spostata solamente all'interno del segmento che congiunge i due altoparlanti.

- *sistema multicanale*, è il passo successivo da un punto di vista di complessità. L'idea è di avere canali separati per ogni direzione desiderata, inclusi sopra e sotto: sistemi home-theater si basano appunto su questo concetto. In un qualsiasi ambiente si possono sfruttare i limiti della nostra percezione e posizionare piccoli altoparlanti ovunque e uno più grande, detto *subwoofer*, avente la funzione di diffondere le frequenze basse non-direzionali[1].

- *sistema binaurale*, basato sull'utilizzo di cuffie. Queste hanno una serie di svantaggi rispetto ai sistemi ad altoparlanti: sono invasive e scomode da indossare per lunghi periodi di tempo, non hanno una risposta in frequenza piatta che può compromettere la spazializzazione e rendono infine l'idea di sorgenti sonore molto "chiuse" e non compensano il movimento dell'ascoltatore se non con l'utilizzo di un sistema di *motion-tracking*. Vi sono però due importanti vantaggi: innanzitutto eliminano i riverberi e, in seguito, semplificano di molto le tecniche di *3D sound rendering*. Al contrario i sistemi ad altoparlanti soffrono del fenomeno di "cross-talk": il suono emesso da un altoparlante giungerà sempre ad entrambe le orecchie dell'ascoltatore. Se si ignora l'effetto prodotto dall'ambiente di ascolto si possono utilizzare, in prima approssimazione, tecniche di rimozione del "cross-talk" per adattare i segnali per due altoparlanti stereo alle cuffie[1].

In seguito ci si focalizzerà su tecniche basate su sistemi binaurali a cuffie.

### 1.3.1 Rendering basato sulle HRTF

L'idea di base sta nell'utilizzare *HRIR* ed *HRTF* già misurate: dato un segnale anecoico e la direzione virtuale della sorgente tramite gli angoli $(\theta, \phi)$, i segnali destro e sinistro sono sintetizzati tramite l'introduzione dell'adeguato ITD e in seguito convolvendo i due segnali con le rispettive risposte impulsive $h^{(l)}$ e $h^{(r)}$.

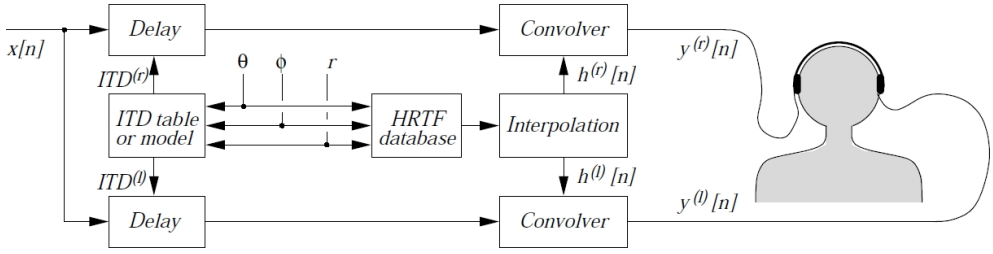



**Figura 1.7:** Schema a blocchi per il *3D sound rendering* su sistema binaurale basato su *HRTF*

#### Misurazione delle HRTF e delle ITD

La situazione tipica per la misurazione delle *HRTF* è la seguente: una camera anecoica e un insieme di altoparlanti collocati ad intervalli regolari di $\theta$ e $\phi$ sulla superficie di una sfera di raggio $r > 1$ m per evitare effetti del *campo vicino*. L'ascoltatore, ovvero la testa di un manichino, è posizionato al centro della sfera con un microfono ad entrambi gli orecchi.

Le risposte impulsive sono misurate eseguendo un segnale analogico e registrando le risposte alle orecchie per ogni posizione virtuale desiderata. Ascoltatore ed altoparlanti non hanno bisogno di essere spostati, facilitando di molto le misurazioni. Tuttavia il posizionamento del microfono costituisce un problema: può essere applicato all'ingresso del canale uditivo oppure alla fine dello stesso, per tener conto degli effetti da esso introdotti.

Le *HRTF* misurate possono essere analizzate al fine di stimare i valori di ITD e raccoglierli in una tabella che verrà utilizzata nel *rendering* (vedi primo blocco in figura 1.7). La stima degli ITD può avvenire tramite diverse metodi, quali *cross-correlation* o *leading-edge*. Alcuni approcci permettono di ricavare l'ITD in funzione della frequenza e dell'elevazione mentre il modello ricavato nell'equazione 1.1 risulta essere molto più semplificato[1].

In molte applicazioni di *3D sound* tipicamente si utilizza una singola *HRTF* per ogni utente. Un altro approccio sta nel personalizzare queste funzioni in base ad alcune caratteristiche o, alternativamente, di costruire matematicamente delle *generalized HRTF* in base a caratteristiche comuni ad un certo numero di individui.

Un'alternativa molto utilizzata sta nell'utilizzare le cosiddette "dummy heads" (teste di manichino), costruite mediando un consistente numero di misure antropometriche, ad ottenere delle misure standard per testa, pinne e torso. Il modello di manichino più utilizzato è probabilmente *KEMAR* (*Knowles Electronic Manikin for Auditory Research*) che facilita la misurazione delle *HRIR*; inoltre i microfoni posizionati all'interno della *dummy head* hanno una risposta alle basse frequenze migliore dei microfoni sonda. Gli altoparlanti utilizzati nella misurazione sono stati posizionati ad una distanza di $1,4$ m per evitare, come già detto in precedenza, gli effetti del *campo vicino*; la frequenza di campionamento è di $44,1$ kHz[6]. Il numero complessivo di misurazioni è 710, distribuite per elevazioni da $-40^\circ$ a $90^\circ$ come descritto nella seguente tabella:

| Elevazione | Numero di misure | Incremento di azimuth |
|---|---|---|
| -40 | 56 | 6,43 |
| -30 | 60 | 6,00 |
| -20 | 72 | 5,00 |
| -10 | 72 | 5,00 |
| 0 | 72 | 5,00 |
| 10 | 72 | 5,00 |
| 20 | 72 | 5,00 |
| 30 | 60 | 6,00 |
| 40 | 56 | 6,43 |
| 50 | 45 | 8,00 |
| 60 | 36 | 10,00 |
| 70 | 24 | 15,00 |
| 80 | 12 | 30,00 |
| 90 | 1 | / |

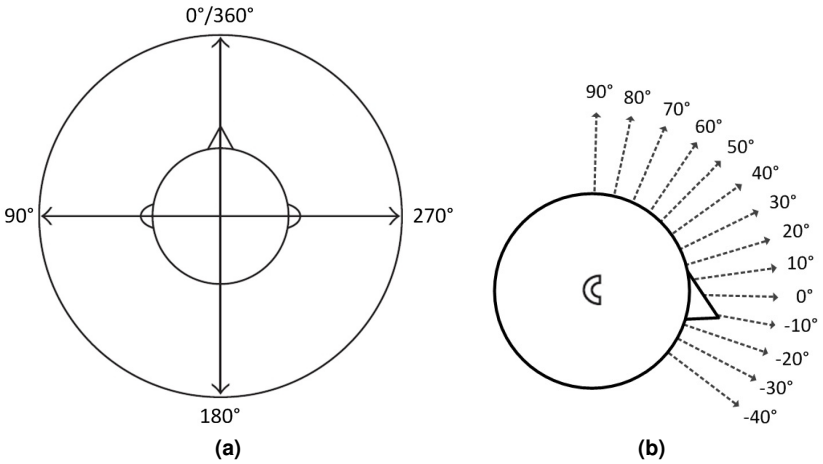



**Figura 1.8:** Angoli di azimuth **(a)** ed elevazione **(b)** coperti nella misurazione

Se il *KEMAR* è perfettamente simmetrico basterà effettuare solamente metà delle misurazioni previste: le *HRTF* sarebbero a loro volta simmetriche. Nella misurazione effettuata le due orecchie, invece, non sono uguali e, coprendo tutta la superficie della sfera, si potranno ottenere due funzioni per simmetria[6].

### 1.3.2 La fase di post-elaborazione

Le misurazioni vengono in seguito rielaborate. Tipicamente si effettua una post-equalizzazione per eliminare potenziali non-linearità spettrali degli altoparlanti, dei microfoni e delle cuffie utilizzate per la riproduzione; ad esempio i microfoni sonda sono inefficienti per frequenze inferiori ai 400 Hz ed un *"bass boosting"* (potenziamento dei bassi) risulta, quindi, una procedura comune. L'effetto del canale uditivo, se questo non faceva parte delle misurazioni, può essere introdotto grazie ad equalizzazioni standard, anche se solitamente nell'ascolto in cuffia non è richiesta quest'elaborazione, per evitare "doppie risonanze"[1].

Spesso è applicata un'ulteriore elaborazione per ridurre la ridondanza dei dati delle *HRTF*: infatti alcune caratteristiche spettrali, indipendenti dalla posizione, non hanno bisogno di essere salvate in ogni *HRTF*. Pertanto viene stimata una funzione, calcolando una media logaritmica del modulo di diverse misurazioni nello spazio, detta *CTF* (*Common Transfer Function*). Questa particolare funzione include le caratteristiche spettrali condivise da ogni *HRTF* come, ad esempio, il canale uditivo. Durante la fase di post-elaborazione la componente *CTF* può essere rimossa, ad ottenere la cosiddetta *DTF* (*Directional Transfer Function*). La *DTF* è funzione di $\theta$ e $\phi$ ed è la quantità che raccoglie le caratteristiche spettrali addette alla spazializzazione. Sia $C(\omega)$ la *CTF* e siano $D^{(l),(r)}(\theta, \phi, \omega)$ le *DTF*, rispettivamente sinistra e destra, sconosciute. Sono stimabili dalle *HRTF* grazie alla seguente relazione:

$$H^{(l),(r)}(\theta, \phi, \omega) = C(\omega) D^{(l),(r)}(\theta, \phi, \omega) \tag{1.4}$$

L'importanza di questa divisione di funzioni sta nel poter modificare un numero minore di parametri per approssimare gli effetti della spazializzazione, rendendo minore la complessità computazionale[1].

### 1.3.3 Interpolazione

La misurazione delle risposte impulsive può essere eseguita solamente per un numero finito di posizioni e, nel caso in cui si stia considerando una posizione intermedia, si rivela necessaria l'*interpolazione* dei dati. Nel caso in cui si sfrutti l'approccio *nearest neighbor* verranno prodotti rumori quando la sorgente è in movimento.

Un metodo semplice per lavorare direttamente sulle *HRIR* sta nell'utilizzo dell'interpolazione bilineare, che consiste semplicemente nel calcolare la risposta ad un dato punto $(\theta, \phi)$ tramite la media pesata dei quattro campioni più prossimi. Nel dettaglio, se una *HRIR* è stata misurata con incrementi $\theta_{grid}$ e $\phi_{grid}$, si può stimare il valore $\hat{h}$ in un punto arbitrario

$$\hat{h}[n] = (1 - c_\theta)(1 - c_\phi) h_1[n] + c_\theta (1 - c_\phi) h_2[n] + c_\theta c_\phi h_3[n] + (1 - c_\theta) c_\phi h_4[n] \tag{1.5}$$

dove $h_k[n]$ $(k = 1, ..., 4)$ sono le risposte impulsive associate ai quattro punti più vicini alla posizione desiderata. Inoltre:

$$c_\theta = \frac{\theta \bmod \theta_{grid}}{\theta_{grid}}, \quad c_\phi = \frac{\phi \bmod \phi_{grid}}{\phi_{grid}} \tag{1.6}$$

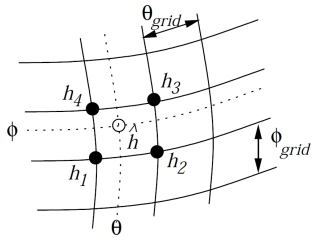



**Figura 1.9:** Interpolazione delle *HRIR*

Per qualsiasi punto nello spazio si può, quindi, dedurre la risposta impulsiva corrispondente a partire da una serie di punti campionati; maggiore sarà il numero di punti campionati, migliore sarà il risultato dell'interpolazione. Possono essere utilizzati anche altri tipi di interpolazione, ma quella appena esposta si rivela efficace data la bassa complessità computazionale, che in un ambito *real-time* risulta essere fondamentale, e i buoni risultati che si ottengono[1].

# Capitolo 2

# *PhaseSpace*: un sistema di *motion-capture*

Il sitema *PhaseSpace Impulse* cattura movimenti complessi in modalità *real-time* grazie ad avanzate tecnologie hardware e software. La cattura del movimento all'interno di un determinato volume è ottenibile muovendo corpi aventi specifici LED attaccati, previo posizionamento delle telecamere del PhaseSpace attorno a tale volume.

Le telecamere individuano le posizioni dei LED e trasmettono le informazioni ad un computer centrale che le processa e calcola le posizioni attuali, disponibili per ulteriori elaborazioni all'interno di un apposito server.

Il sistema PhaseSpace consiste di:

- un *HUB/server* che gestisce i dispositivi (sezione 2.1);
- un apparecchio di calibrazione (sezione 2.2);
- *LED System* (sezione 2.3);
- telecamere (sezione 2.4);
- un software specifico (sezione 2.5).

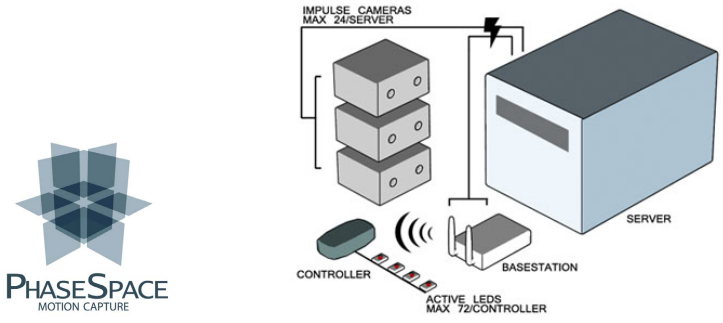



**Figura 2.1:** Il simbolo **(a)** e uno schema del sistema **(b)**

## 2.1 Il *server* e l'HUB

L'HUB (letteralmente *fulcro*, *mozzo*, *elemento centrale*) rappresenta un dispositivo di rete che funge da nodo di smistamento: nel sistema *PhaseSpace Impulse* questo oggetto è incorporato nel computer. All'HUB vengono connesse le telecamere e la *LED Base Station*: la funzione di questo apparecchio risulta perciò quella di raccogliere le informazioni inviate dalle telecamere per poi inoltrarle al *server*. Quest'ultimo monta un apposito software che permette, previa calibrazione e settaggio dei vari dispositivi, la visualizzazione *real-time* del movimento dei *sensori LED* all'interno dell'area coperta dalla telecamere.


**Figura 2.2:** *Computer/server* **(a)** e l'HUB integrato **(b)**

## 2.2 Dispositivo *calibration wand*

La calibrazione del sistema viene effettuata tramite lo strumento *calibration wand* (figura 2.3); questo dispositivo permette una calibrazione accurata in pochi minuti, grazie all'ausilio di otto LED distribuiti sulla lunghezza dello stesso.

Innanzitutto prima di iniziare la procedura di calibrazione è necessario che il sistema abbia identificato tutti i dispositivi e, inoltre, che tutti i LED utilizzati siano stati riconosciuti; il programma di calibrazione può essere successivamente eseguito dal *server* o da una macchina remota. Nella fase di calibrazione questo dispositivo viene impiegato per:

- il puntamento delle telecamere, cioè la procedura preliminare alle operazioni di allineamento;

- la scelta dell'origine, infatti nella parte iniziale della fase di calibrazione si chiede di posizionare la *calibration wand* nel punto in cui si desidera impostare l'origine del sistema di riferimento;

- la determinazione degli assi di riferimento, si devono infatti fornire al sistema due ulteriori punti, grazie ai quali verranno fissati gli assi del sistema; la determinazione del terzo asse, quello verticale, non necessita di alcuna procedura.

**Figura 2.3:** *Calibration wand*

## 2.3 Dispositivi LED

I LED (*Light Emitting Diode*) del sistema PhaseSpace Impulse si possono considerare come i dispositivi che generano informazione. I componenti di questa categoria sono descritti in questa sezione.

### 2.3.1 LED Base Station


**Figura 2.4:** La *LED Base Station* **(a)** e le interfacce di cui dispone **(b)**

| | |
|---|---|
| **Frequenza del segnale** | 2.4 GHz |
| **Dimensioni** | 127 mm x 76 mm x 30 mm |
| **Peso** | 136 g |

**Figura 2.5:** Dati tecnici *LED Base Station*

Le interfacce disponibili sono:

- una porta RJ11 (6-pin)
- una porta RJ45 (8-pin Ethernet)
- porte per *LED strings*
- antenna RF esterna

La funzione primaria della *LED Base Station* (vedi figura 2.4) è quella di collegare l'HUB al sistema LED; è connessa all'HUB tramite una cavo Ethernet. Ha come funzioni:

- trasmissione di un segnale di temporizzazione alla *LED Driver Unit*
- programmazione dei LED
- gestione di *LED strings* in maniera separata

### 2.3.2 LED Driver Unit

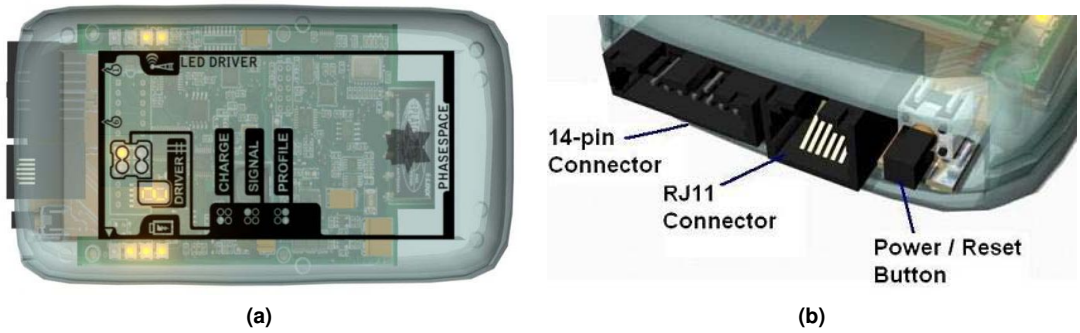



**Figura 2.6:** La *LED Driver Unit* **(a)** e le interfacce **(b)**

| | |
|---|---|
| **Capacità di gestione** | 72 LED |
| **Durata batteria** | 4 ore |
| **Dimensioni** | 126 mm x 70 mm x 25 mm |
| **Peso** | 90 g |

**Figura 2.7:** Dati tecnici della *LED driver unit*

La *LED Driver Unit* è la strumento che comanda i *LED Modules*; consiste in una batteria e un ricevitore in RF, il quale capta il segnale si temporizzazione inviato dalla *LED Base Station*. Le interfacce di cui dispone questo dispositivo sono le seguenti (vedi figura 2.6):

- Tasto power/reset
- Una porta RJ11 (a 6-pin)
- Una porta a 14-pin per *LED string*

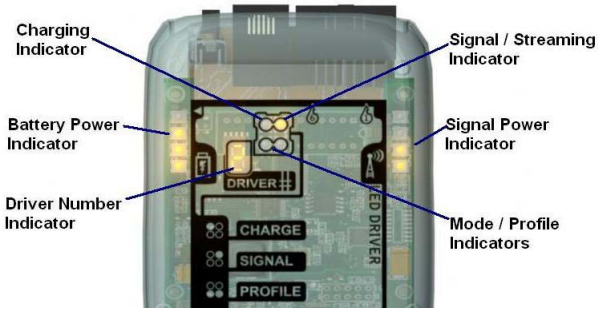



**Figura 2.8:** Indicatori *LED Driver Unit*

Il dispositivo è dotato di indicatori per permettere all'operatore di verificare lo stato dell'unità (vedi figura 2.8):

- **Battery Power** - carica della batteria
- **Signal Power** - nitidezza del segnale inviato dalla *LED Base Station*
- **Driver Number** - mostra il numero del *driver* assegnato all'unità (durante la calibrazione verrà visualizzata la lettera $C$)
- **Mode/Profile** - i due LED inferiori mostrano la modalità utilizzata dal sistema
- **Signal-Streaming** - il LED destro superiore è acceso quando il sistema sta inviando dati e l'unità li sta ricevendo
- **Charging** - il LED sinistro superiore è acceso quando l'unità è in carica

### 2.3.3 LED Strings

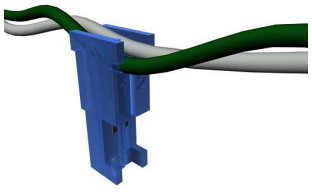
Le *LED Strings* consistono in un *LED cable* e in *LED cable connectors*.

I *LED cables* consistono in due cavi, per ognuno dei quali ogni connettore ha un incavo; alla *LED Driver Unit* si possono collegare fino a sei *LED strings*, le quali possono disporre di massimo 12 moduli, cioè 12 sensori.

Generalmente le *LED Strings* non sono assemblate, nel senso che i connettori non sono attaccati al cavo; questo permette il posizionamento degli stessi dipendentemente dalla tipologia di *motion-capture* che si intende fare.

### 2.3.4 LED Modules

I *LED Modules* (in figura 2.9c) contengono le sorgenti luminose che sono identificate dalle telecamere, comandate da un microprocessore che controlla la durata e l'ampiezza dell'impulso luminoso. É possibile modificarne la luminosità dei LED dal software fornito. Ciascuno dei 12 LED ha una denominazione (da $A$ ad $L$) per renderne unica l'identificazione; questi moduli vanno attaccati alle *LED String* (vedi figura 2.9b).



**(a)** *Connettore attaccato al cavo.*

**(b)** *LED Module connesso alla LED String.*


**(c)** *LED Modules*

**Figura 2.9:** *LED Strings* e *LED Modules*

## 2.4 Le telecamere


(a) (b)

**Figura 2.10:** Le telecamere: i sensori frontali **(a)** e le porte *Ethernet* **(b)**

| | |
|---|---|
| **Optical Resolution** | 3600 x 3600 (12,4 Megapixel) |
| **Subpixel Resolution** | 30000 x 30000 (a 480 Hz) |
| **Dimensioni** | 108 mm x 83 mm x 57 mm |
| **Peso** | 227 g |

**Figura 2.11:** Dati tecnici delle telecamere

Le telecamere del sistema PhaseSpace Impulse sono utilizzate per triangolare in tempo reale le posizioni dei marcatori LED utilizzati; utilizzano sensori CCD (*Charge-Coupled Device*) lineari ad alta velocità e risoluzione. Ogni telecamera è dotata di due aperture sulla parte frontale, all'interno delle quali sono montate una lente semicilindrica e un sensore CCD lineare, posizionato alla distanza focale della lente e perpendicolare all'asse della stessa.

Nella parte laterale di ogni dispositivo si possono trovare due porte Ethernet, infatti è necessario un collegamento di questi dispositivi all'HUB; accanto ad ogni porta, inoltre, è presente un indicatore per segnalare la presenza di connessione.

Le telecamere sono facilmente posizionabili tramite comuni supporti per videocamere o fissabili in maniera permanente a muri o altre strutture.

### 2.4.1 La connessione

La connessione con l'HUB/Server del sistema viene effettuata tramite normali cavi Ethernet. É possibile connettere catene di massimo sei telecamere per ogni porta dell'HUB come osservabile in figura 2.12: per connettere una telecamera si dovrà, quindi, collegare un cavo dalla porta superiore (*Next Camera*) dell'ultimo dispositivo della catena alla porta inferiore (*Previous Camera or HUB*) del nuovo apparecchio; la corretta connessione di HUB e telecamere è segnalata dagli appositi LED indicatori.

**Figura 2.12:** Connessione di una catena di telecamere all'HUB/Server

É necessario prestare attenzione a non connettere o disconnettere dispositivi all'HUB mentre il *server* è acceso, perciò le telecamere e la *LED Base Station* devono essere collegate in modalità *off-line*. Non è inoltre consentito l'utilizzo di cavi Ethernet *crossover*, ma solamente di normali cavi Ethernet possibilmente schermati per soddisfare i limiti di emissioni.

### 2.4.2 Il posizionamento

I posizionamenti delle telecamere dipendono dalle applicazioni e dall'area che si desidera coprire:

- per il *full-body tracking*, è consigliato posizionare le telecamere in maniera circolare per garantire una copertura più efficiente dell'ambiente tracciato (vedi figura 2.13);
- per l'*head tracking*, sarà necessario posizionare le telecamere a semicerchio diretto verso la parte frontale della testa.

**Figura 2.13:** Esempio di posizionamento delle telecamere per il *full-body tracking*

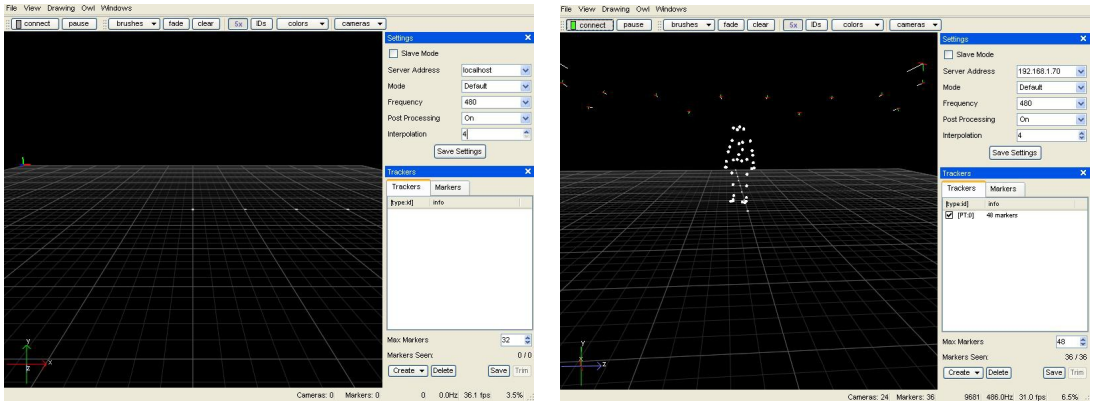
## 2.5 Il software

Il sistema *PhaseSpace Impulse* include il software *PhaseSpace* già installato nell'HUB/*server*; questo è necessario per il setup e la calibrazione del sistema, nonché per iniziare o visualizzare una sessione di *motion-capture*.

Si analizzano di seguito i passi per configurare il sistema *PhaseSpace Impulse*:

1. Posizionare le telecamere
2. Connettere fino ad un massimo di 6 telecamere in configurazione a catena per ogni porta
3. Connettere la *LED Base Station* ad una porta dell'HUB
4. Accendere il *server*, l'utente ha la possibilità di eseguire il software anche da un calcolatore remoto. se il software è eseguito esclusivamente da una macchina remota, non è richiesto alcun *login* al *server*. Nel caso sia richiesto si dovranno effettuare le operazioni **(a)** e **(b)**:
   (a) Effettuare il login con i seguenti dati:
   **LOGIN:** *demo*
   **PASSWORD:** *demo*
   (b) Eseguire il comando *"startx"* da prompt per avviare le X-WINDOWS
5. Aprire il *PhaseSpace Configuration Manager*, ovvero il software per configurare i LED, dal browser ed effettuare il login:
   **USER NAME:** *admin*
   **PASSWORD:** *phasespace*
6. Accendere la *LED Driver Unit* (eventualmente connetterla alla *LED Base Station*)
7. Calibrare il sistema utilizzando la *calibration wand* e il programma *calib*

8. Configurare le posizioni dei LED
9. Avviare un client
10. Inizio cattura del *PhaseSpace*




**(a)** **(b)**

**Figura 2.14:** Schermata principale **(a)** e utilizzo del software in un'applicazione *full-body tracking* **(b)**

# Capitolo 3

# L'ambiente *Pure Data*

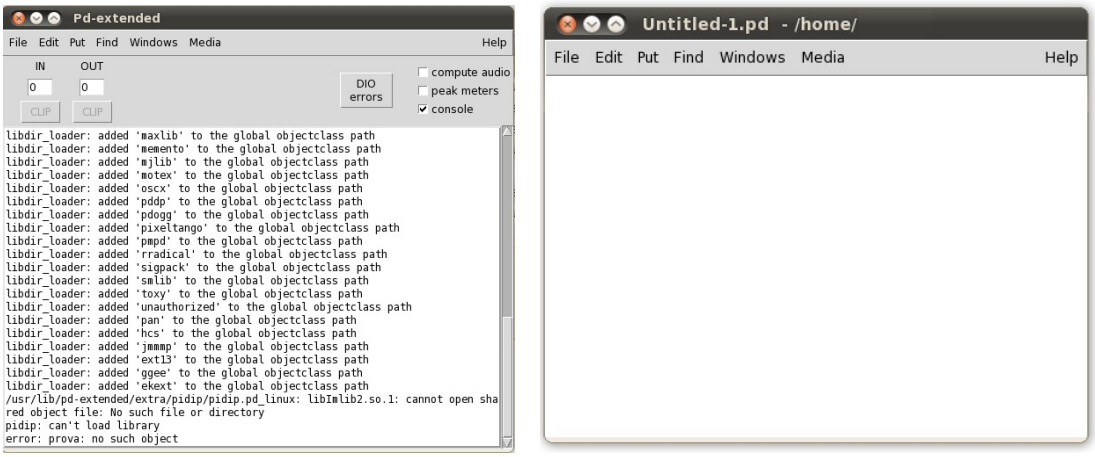

É un ambiente di programmazione grafica *real-time*, scritto nel 1996 dall'ingegnere del software Miller Puckette, già sviluppatore di *Max/Msp*, per la creazione di musica dal computer e per lavori multimediali.

**É un linguaggio di programmazione** a tutti gli effetti perché consente di realizzare algoritmi più o meno complessi come gli altri linguaggi.

**L'interfaccia è grafica** poiché non è necessario scrivere codice da un editor di testo, ma si realizzano delle *patch* combinando fra loro vari tipi di oggetti grafici: gli algoritmi vengono creati selezionando una serie di entità grafiche collegate tra loro all'interno della cosiddetta *patch window*, creando una sorta di *flow chart*. La presenza dei collegamenti tra gli oggetti definisce questo come un linguaggio di programmazione orientato al flusso di dati (*datastream-oriented programming language*).

**É in modalità *real-time*** visto che gli algoritmi sono interattivi e i parametri sono modificabili anche durante l'esecuzione, nel corso della quale sono ammessi anche inserimenti di nuovi oggetti e collegamenti tra gli stessi. Il comportamento non è uguali ai classici linguaggi di programmazione dove il codice deve essere processato prima di essere utilizzato, ma si comporta come uno strumento musicale classico: l'esecutore sente istantaneamente la modifica del suono. Questa caratteristica rende il software utilizzabile anche per performance live.

**É un software multipiattaforma, versatile ed *open-source*** essendo possibile consultarne i sorgenti, modificarlo, redistribuirlo e, soprattutto, creare nuove applicazioni e librerie per aumentarne le funzionalità. Da quest'ultima opportunità offerta dall'ambiente nasce la versione Pd-Extended, la quale include ulteriori librerie di molti programmatori, musicisti, ingegneri acustici e compositori che si sono uniti allo sviluppo di *Pure Data*. Il software è *open source*: non è sviluppato da alcuna azienda e non è in vendita. Uno svantaggio di questa politica risiede nel fatto che l'accessibilità del programma è ristretta a pochi utenti. Nonostante ciò, grazie alla disponibilità immediata del programma garantita da internet e all'avanzamento del progetto, *Pure Data* è ottimizzato per l'utilizzo a livelli professionali.




**(a)** **(b)**

**Figura 3.1:** Pd *window* **(a)** e *patch window* **(b)**

## 3.1 Panoramica dell'ambiente

Il programma dispone di due finestre fondamentali, delle quali ora si andranno ad analizzare le funzionalità:

**Pd *window*** (ovvero la finestra principale, vedi Fig 3.1a) avente tre funzioni fondamentali: mostrare i messaggi tramite il terminale integrato nella finestra, definire il percorso delle librerie da caricare e consentire la configurazione delle impostazioni di audio e MIDI.

I messaggi iniziali si riferiscono al caricamento delle librerie esterne, eventuali errori; l'utente stesso può stampare messaggi su questa finestra tramite appositi oggetti.

Dal menu File è possibile:

- creare, aprire, chiudere o salvare le *patch* (New, Open, Close, Save)
- impostare il Path delle librerie esterne
- decidere quali librerie caricare allo Startup di Pd-Extended
- inviare messaggi alla console (Message)

Dalla voce di menù Media è possibile:

- impostare il MIDI (MIDI settings)
- impostare l'audio (Audio settings)
- avviare il motore DSP (Audio ON/OFF)
- testare il corretto funzionamento di Audio e MIDI (Test audio and MIDI)

Si sottolinea come Pd-Extended imposti automaticamente Path e Startup in modo da caricare le principali librerie, il tutto per garantire una completa funzionalità dell'ambiente. Per creare una nuova *patch* basterà ora eseguire il comando New (Ctrl+n) dal menu File .

***Patch window*** che rappresenta l'ambiente di lavoro vero e proprio (vedi Fig 3.1b); ha un menu molto simile a quello della precedente finestra ma, ovviamente, maggiormente orientato all'editing delle *patch*. Queste possono trovarsi in due differenti stati funzionali (si può agilmente passare dall'uno all'altro tramite la combinazione Ctrl+E):

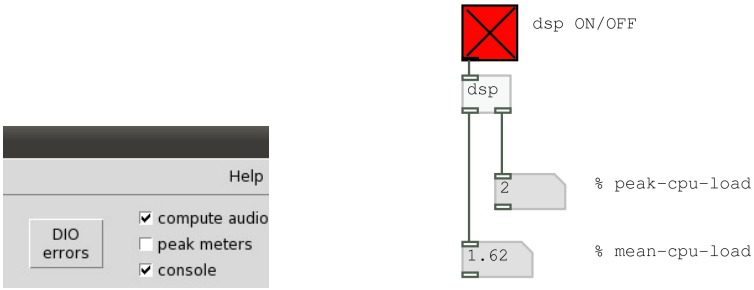
EDIT MODE: è possibile inserire e spostare oggetti;

RUN MODE: serve per gestire la *patch* quando questa è in azione, ovvero per permettere di modificare gli oggetti interattivi.

## 3.2 Il motore DSP

Quando gli algoritmi processano esclusivamente dati, Pd-Extended è completamente attivo sin dall'apertura in elaborazione *real-time*; per quando riguarda invece i file audio si rivela necessario avviare un particolare processore, detto DSP (acronimo di *Digital Signal Processor*), per la computazione di questi segnali.

Il DSP si occupa di elaborare in tempi molto rapidi il segnale audio, nonché di effettuare trasformazioni *analogico-digitali* e *digitali-analogiche* caratteristiche di questi particolari segnali[10]; è attivabile dalla Pd *window* spuntando la voce compute audio (Fig 3.2a), oppure dalla *patch window* tramite il posizionamento degli appositi blocchi, in edit mode, e poi spuntare l'oggetto GUI (*graphic user interfaces*) in run mode (Fig 3.2b).




(a) (b)

**Figura 3.2:** Il DSP attivato nella Pd *window* **(a)** e nella *patch window* **(b)**

## 3.3 Le *box* in PD

Dalla *patch window* è possibile creare gli algoritmi tramite entità dette *box*, suddivisibili in quattro categorie:

- oggetti (*object box*)
- messaggi (*message box*)
- GUI (*graphical user interface*)
- commenti

Gli elementi fondamentali per la programmazione in PD sono le *object box*, ovvero gli oggetti, di forma rettangolare e con un determinato numero di *inlets* (ingressi, posti nella parte superiore) ed *outlets* (uscite, poste nella parte inferiore). Esistono due tipi di cavi che si possono utilizzare per collegare i riquadri:

- i più sottili trasportano informazioni riguardanti il controllo degli oggetti;
- i più spessi trasmettono il segnale vero e proprio.

Gli oggetti che gestiscono file audio hanno una tilde ~ dopo il nome, mentre gli altri oggetti ne sono sprovvisti. Nei riquadri dei messaggi (*message box*) il testo riportato sarà il messaggio inviato in output.

## 3.4 Lo *scheduling*

*Pure Data* utilizza numeri a 64-bit in virgola mobile per rappresentare il tempo, garantendo un'elevata precisione e pressoché l'assenza di *overflow*. Il tempo è visualizzato dall'utente in millisecondi[7].

### 3.4.1 Audio e messaggi

Le elaborazioni dell'audio e dei messaggi sono eseguite in maniera interlacciata. Negli intervalli tra un'elaborazione e l'altra i ritardi potrebbero finire, oppure potrebbero verificarsi delle particolari condizioni esterne. Questi eventi possono causare una cascata di messaggi in modalità *depth-first*; ognuna di queste cascate termina prima che il messaggio successivo sia calcolato o si verifichi un tick del DSP. I messaggi non vengono mai passati agli oggetti durante i tick del DSP[7].

### 3.4.2 Il carico computazionale

Lo *scheduler* mantiene una particolare procedura, scelta dall'utente, nelle sue computazioni; si cerca di rimanere al passo del tempo reale computando un piccolo numero di campioni, in modo da assorbire imprevisti aumenti del tempo di calcolo. Questo *audio buffer* può essere impostato dall'utente.

Se *Pure Data* non riesce a rimanere al passo, le lacune (occasionali o frequenti) appariranno negli stream di ingresso ed uscita. Le computazioni *real-time* del programma competono con l'interfaccia grafica per il tempo di CPU; una gestione di tale problema deve ancora essere scritta, in ogni caso è consigliabile non utilizzare troppi oggetti GUI *real-time* per evitare problemi nell'emissione dell'audio. Se una finestra è chiusa i dati relativi ad essa non vengono inviati, permettendo di risparmiare risorse computazionali.

### 3.4.3 Determinismo

Tutte le cascate di messaggi sono gestite dallo *scheduler* in modo che si concludano prima di un tick del DSP; è da sottolineare come nessun calcolo sia rischedulato per mantenere il passo con il *real-time*: ogni evento si verificherà come stabilito dallo *scheduler*. Inoltre se due cascate di messaggi sono previste per il medesimo tempo logico, esse verranno eseguite nell'ordine in cui sono state pianificate. Queste considerazioni sono necessarie per rendere deterministico il sistema[7].

In un sistema *real-time* il tempo logico, corrispondente al prossimo campione audio da modificare, è leggermente in ritardo rispetto all'istante reale, visto che il segnale deve essere sottoposto a computazioni più o meno dispendiose in termini di tempo. Le computazioni di controllo ed audio sono eseguite alternativamente, secondo l'ordine logico temporale.

Quando si determina l'uscita di un segnale audio, o si sta valutando qualche ingresso di controllo, è necessario che il tempo logico sia lo stesso per tutta la durata delle istruzioni, come se queste fossero eseguite nel medesimo istante. In questo modo, le computazioni dei segnali audio, se eseguite correttamente, saranno deterministiche: l'esecuzione delle stesse operazioni audio in un sistema *real-time* e non, dovrebbe produrre gli stessi risultati.

## 3.5 Elaborazione dati

Poiché *Pure Data* è un programma *real-time*, si rivela necessario un livello diverso da quello audio per la gestione degli oggetti interni alla *patch*; la computazione di tali segnali deve poter essere eseguita ad intervalli irregolari di tempo: ad ogni computazione viene, perciò, associata una corrispondenza ad un tempo logico specifico.

Si rende indispensabile l'utilizzo di un tempo logico e non di un tempo reale perché, in tal modo, si possono tener separati i calcoli eseguiti dal computer: questi, infatti, possono variare per molte ragioni, anche per istruzioni apparentemente identiche[3]. Il tempo logico indica quale sarà il primo campione audio di uscita che rispecchierà il risultato della computazione, mentre in sistemi non *real-time* questo implica che il tempo logico inizi da zero e la sua durata sia pari alla lunghezza del segnale da elaborare. Ad ogni azione di controllo è associato un istante logico, al raggiungimento del quale viene eseguita l'elaborazione, con conseguente cambiamento dell'output. Per poter garantire una corretta computazione, i calcoli dei segnali di controllo e dei segnali audio vengono eseguiti a turno in ordine temporale crescente[3]. Lo scopo principale del livello di controllo è la gestione dei blocchi audio (vedi prossima sezione).

## 3.6 Elaborazione audio

In *Pure Data* i segnali audio vengono rappresentati tramite numeri a 32-bit in virgola mobile, in modo da avere tutta la gamma dinamica desiderabile. Tuttavia, a seconda dell'hardware audio, input ed output possono essere limitati a 16 o 24 bit. Agli ingressi sono associati valori compresi tra -1 e 1 e, di conseguenza, anche i valori in output apparterranno a tale range. *Pure Data* ha di default una frequenza di campionamento di 44100 Hz, a meno che non vengano modificate impostazioni audio dall'utente.

Possono essere letti o scritti file di campioni sia a 16-bit o 24-bit in virgola fissa o in 32-bit in virgola mobile, nei formati WAV, AIFF, oppure nel formato AU, tramite gli oggetti soundfiler, writesf~ e readsf~[7].

La maggior parte dei software di *computer music* computa l'audio in blocchi per aumentare la qualità della resa. Ogni oggetto audio incorre in un aumento del carico di computazione, ogni qual volta esso viene chiamato, pari circa a venti volte il costo medio per calcolare un unico campione: è necessario quindi costituire blocchi di dimensioni non troppo piccole, per diminuire il numero di chiamate. Ad esempio se il blocco è composto da 64 campioni (come di default), l'*overhead*[1] introdotto è del 30%, mentre se il blocco fosse di un unico campione l'*overhead* sarebbe addirittura del 2000%.

Nella musica elettronica vengono utilizzati altoparlanti per generare il suono: questi strumenti sono dotati di membrane che, vibrando, producono il suono. Queste vibrazioni sono controllate dal computer e, nel caso specifico di *Pure Data*, questo processo è gestito dall'oggetto dac~ (*digital audio converter*). Il suo compito è di trasformare numeri in virgola mobile a 32 bit in un segnale elettrico che, una volta amplificato, farà produrre una vibrazione alla membrana dell'altoparlante (la posizione più convessa che la membrana può assumere è associata al numero 1, mentre la più concava è associata al numero -1)[3]. Il *rate* con cui vengono inviate le informazioni audio è, per default, di 44100 Hz (quindi viene riprodotto un campione ogni $1,45$ ms). É necessario comprendere come le informazioni fluiscono dal livello di controllo a quello audio.

**Il *flusso di controllo*** è un insieme di numeri determinati in base alle computazioni di controllo. Può coinvolgere sezioni temporali posizionate ad intervalli regolari od irregolari. L'esempio più semplice è composto da una serie di informazioni indicanti gli intervalli temporali $(\ldots, t(0), t(1), \ldots)$ di campioni audio, che saranno disposti in ordine non decrescente. Il *flusso di controllo* può essere concepito come un controllore MIDI che cambia ad intervalli irregolari. In questo caso si indica solo l'istante temporale in cui accade un evento[3].

**Il *flusso di controllo numerico*** è rappresentato da una coppia di valori che associa a qualche attimo temporale il valore che la funzione di uscita assume in quell'istante: $\ldots, (t(0), x(0)), (t(1), x(1)), \ldots$
Il *flusso di controllo numerico* può essere visto come un segnale audio nel quale, per ogni istante temporale, viene associato un determinato valore. Tuttavia, a differenza dei segnali audio veri e propri, in cui il *rate* è costante, questo particolare flusso non ha una frequenza precisa: si possono utilizzare tre tecniche per effettuare la conversione:

1 - nella prima si cerca di effettuare la conversione il più velocemente possibile: ogni campione del segnale di output è pari al più recente valore del segnale di controllo. Questa tipologia di conversione di rivela molto efficace per flussi di controllo che non cambiano molto frequentemente rispetto alla lunghezza del blocco: il vantaggio sta nella semplicità di computazione e nella massima risposta ai cambiamenti. Se gli aggiornamenti del flusso sono troppo veloci rispetto alla lunghezza del blocco, la frequenza di "campionamento" potrebbe diventare minore della frequenza di Nyquist, provocando un effetto di *aliasing*.

2 - nella seconda conversione ogni nuovo valore del flusso di controllo al tempo $t$ incide sui campioni successivi a tale istante. É molto simile alla precedente con una lunghezza di blocco pari ad 1 campione, con una frequenza di conversione opportunamente alta: risulta essere la soluzione migliore nel caso in cui il flusso di controllo cambi rapidamente.

[1] In informatica, la parola inglese *overhead* (letteralmente in alto, che sta di sopra) serve per definire le risorse accessorie, richieste in sovrappiù rispetto a quelle strettamente necessarie, per ottenere un determinato scopo in seguito all'introduzione di un metodo o di un processo più evoluto o più generale[5].

3 - la terza conversione si utilizza un'interpolazione di due punti per ottenere una maggior accuratezza. Dati i due vettori:

$$(n, x), \quad (n + f, y)$$

siano $x$ il valore del segnale all'istante $n$ (con $n \in \mathbb{N}$), e $y$ il valore del segnale all'istante $n + f$ (con $0 \leq f < 1$, ovvero $f$ rappresenta una frazione di unità temporale). Allora il campione $(n + 1, z)$ avrà valore:

$$z = fx + (1 - f)y$$

cioè una media pesata, in base all'istante $f$, tra il valore del campione precedente e del campione intermedio rilevato.
Questa tecnica fornisce una migliore rappresentazione del *flusso di controllo*, a discapito di un maggior carico computazionale e quindi di un ritardo.
I *flussi di controllo numerici* possono essere convertiti in segnali audio utilizzando funzioni rampa per addolcire le discontinuità[3].

### 3.6.1 Un esempio: il blocco delay~

Di seguito il blocco delay~ in una configurazione che permette di ritardare i campioni del file audio caricato:

- con il blocco più in alto si carica il file audio desiderato;
- con il messaggio 1 il file viene avviato, con 0 viene fermato;
- l'oggetto readsf~ 2 legge il file audio e restituisce in uscita i due canali stereo, mentre l'*outlet* più a destra invia un *bang* nel momento in cui la lettura del file è terminata;
- il blocco +~ unisce i due canali stereo in un unico segnale;
- l'oggetto delay~ 44100 ritarda l'uscita di un campione audio del numero di campioni inviati dal messaggio numero di campioni in base al *rate* indicato, cioè 44100 Hz;
- dall'oggetto precedente escono i due canali con i ritardi, che vengono inviati al blocco dac~ per essere riprodotti.

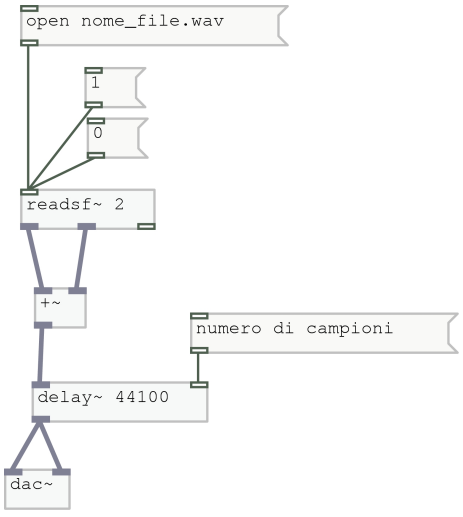



**Figura 3.3:** Un esempio di utilizzo di un oggetto delay~

### 3.6.2 L'*object box* earplug~

Questa *object box* ricopre un ruolo importantissimo nel progetto: esegue infatti le interpolazioni (vedi sezione 1.3.3) delle risposte impulsive, per determinate angolazioni in ingresso, ed effettua la convoluzione delle stesse con i campioni del file audio in esame. Il blocco è quindi un *filtro binaurale a convoluzione*, che utilizza le misurazioni delle *HRTF* del KEMAR in una versione più compatta; perché questo funzioni è necessaria anche la presenza di un ulteriore file (di nome earplug_data.txt) che contiene le risposte impulsive da convolvere.

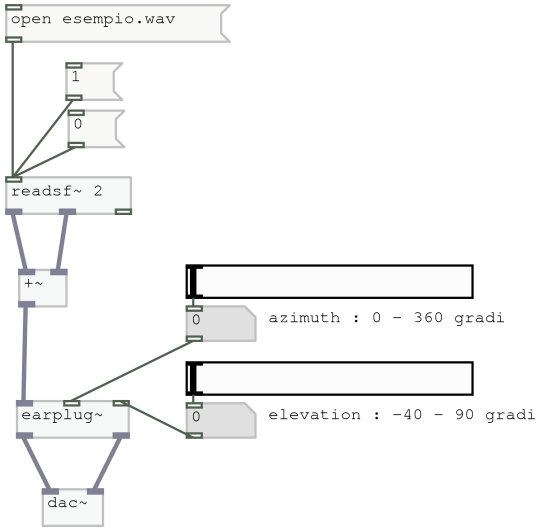



**Figura 3.4:** Un esempio di utilizzo di un oggetto earplug~

In ingresso si hanno quindi:

- i campioni del file audio, letti dall'oggetto readsf~ 2, nei due canali stereo;
- l'angolo di azimuth limitato nel range 0°-360°;
- l'angolo di elevazione per il range -40°-90°.

In uscita si hanno i due canali stereo che, nel caso illustrato in figura 3.4, sono spediti all'oggetto dac~ per essere riprodotti.

# Capitolo 4

# Il progetto

Il progetto consiste nell'integrare ed ottimizzare una prima versione del programma[4] per la spazializzazione del suono, in modalità *real-time*, all'interno di un'area delimitata dalle telecamere del sistema *PhaseSpace Impulse*. Quest'area consiste di un parallelepipedo (di dimensioni 300 mm x 350 mm x 250 mm, vedi figura 4.2) ed, essendo sufficiente un *head tracking* (vedi sezione 2.4.2), risulta sufficiente una copertura di 3 dei 4 lati attorno all'ascoltatore con le 8 telecamere a disposizione.

All'interno di questo volume può liberamente muoversi una persona che indossa delle particolari cuffie, munite dei necessari *LED modules*, posizionati in maniera da riuscire a capire, in ogni istante, la posizione della testa dell'ascoltatore e la direzione verso cui essa è rivolta. Per avere queste due importanti informazioni è sufficiente conoscere la posizione di soli tre punti: l'orecchio destro (detto RIGHT), l'orecchio sinistro (detto LEFT) e la parte superiore della testa (detta UP).

Per incrementare la certezza che questi punti siano acquisiti con continuità dal sistema, si posizionerà una coppia di sensori per ognuno di questi tre punti: nel caso in cui vengano riconosciuti entrambi allora il programma dovrà calcolare il punto medio tra i due, altrimenti verrà considerato solamente il sensore non oscurato.


**Figura 4.1:** Cuffie con sensori connesse alla *LED Driver Unit* **(a)** e il guanto con il sensore associato alla sorgente **(b)**

Le sorgenti possono essere sia posizionate e mosse virtualmente da programma, che associate ad un nuovo sensore; la parte principale del programma sta proprio nel calcolare, istante per istante, la posizione della sorgente relativamente al sistema della testa, questo per garantire, appunto, che si possa simulare la provenienza del suono da tale punto nello spazio.

Il lavoro svolto si può suddividere in due parti principali:

- creazione dell'*external* "*phasespace.pd_linux*"
- creazione della *patch* "*phasespace_patch.pd*"

Nella prima si implementerà un algoritmo la cui funzionalità è di calcolare le coordinate relative della sorgente rispetto all'ascoltatore, mentre nella seconda si costruirà, in PD-EXTENDED, l'intero sistema per l'elaborazione *real-time* dei dati e dei file audio.

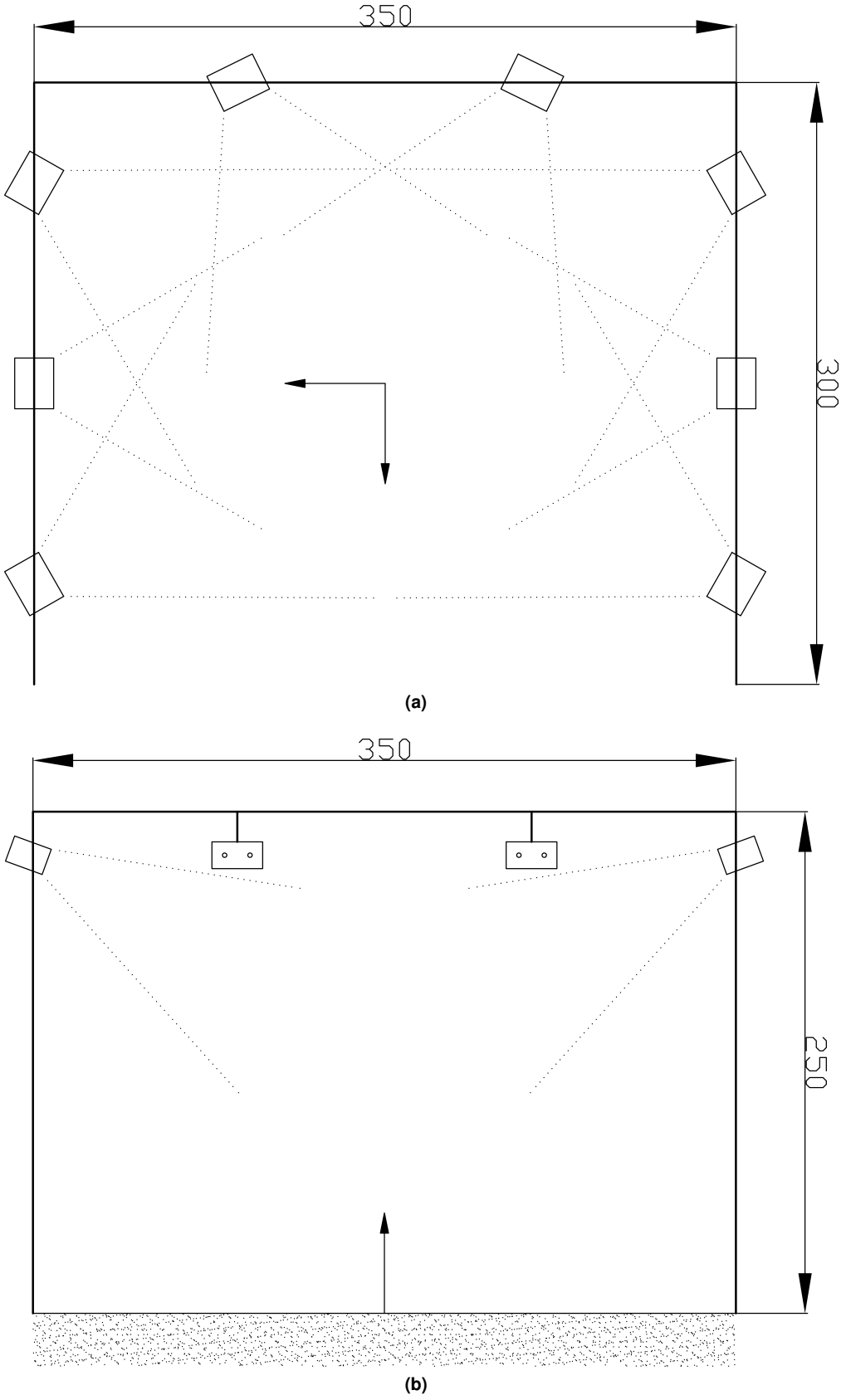



**Figura 4.2:** Vista da sopra **(a)** (con gli assi $x$, verso sinistra, e $z$, verso il basso) e frontale **(b)** (con l'asse $y$)

## 4.1 L'external "*phasespace.pd_linux*"

Quest'external, come anticipato, si occupa di calcolare le coordinate relative della sorgente rispetto alla testa ed è stata scritta in linguaggio C.

Si analizzeranno di seguito tutte le parti del programma in modo da comprenderne il funzionamento.

**Direttive di inclusione** Nella primissima parte del codice (vedi listato 4.1) è necessario includere gli *header-files* contenenti le funzioni utilizzate:

**Listing 4.1: Direttive di inclusione**

```
1 #include <stdio.h>
  #include <math.h>
3 #include "m_pd.h"
```

- `<stdio.h>` : è la libreria standard del linguaggio C
- `<math.h>` : è la libreria matematica, che ha al suo interno, ad esempio, le funzioni trigonometriche
- "`m_pd.h`" : è una libreria particolare, al cui interno si trovano molteplici direttive di definizione, per interfacciare il programma con l'ambiente Pd-Extended

**Definizione strutture** Qui si definiscono il puntatore ad una classe statica `*phasespace_class` e tre strutture fondamentali per il programma:

**Listing 4.2: Strutture**

```
1 static t_class *phasespace_class;

3 typedef struct _coords {
    t_float x, y, z;
5 } t_coords;

7 typedef struct _plane {
    t_float A, B, C, D;
9 } t_plane;

11 typedef struct _phasespace {
    t_object x_obj;
13   t_coords L, R, U, B, F, D, O, P;
    t_plane Ho, Me, Fr;
15   t_float set, angle, angle_prod;
    t_outlet *out1, *out2, *out3, *out4, *out5, *out6;
17 } t_phasespace;
```

- `t_coords` : descrive un punto in $\mathbb{R}^3$ tramite tre coordinate di tipo `float`
- `t_plane` : descrive un piano in $\mathbb{R}^3$ con quattro costanti `float`
- `t_phasespace` : descrive il sistema *PhaseSpace*
  - `x_obj` è un parametro richiesto da Pd-Extended;
  - `L, R, U, B, F, D, O, P` corrispondono ai vari punti della testa e alla sorgente (rispettivamente: left, right, up, back, front, down, origin, point source); è da sottolineare come solamente i primi tre *markers* della testa siano realmente acquisiti, mentre i rimanenti sono calcolati in funzione dei precedenti;
  - `Ho, Me, Fr` sono, rispettivamente, i piani orizzontale, mediano e frontale;
  - `set, angle, angle_prod` sono tre parametri interni all'*external*; i primi due avranno come informazione il tipo di coordinate in uscita e l'unità di misura degli angoli, mentre il terzo è funzione del precedente;
  - `*out1, *out2, *out3, *out4, *out5, *out6` sono i puntatori alle uscite.

**Lettura delle coordinate in ingresso** In ingresso all'*external* vengono passate le coordinate dei tre sensori della testa e del sensore associato alla sorgente.

**Listing 4.3: Lettura delle coordinate in ingresso**

```
void take_observer_coords( t_phasespace *f,
                           t_floatarg f1, t_floatarg f2, t_floatarg f3,
                           t_floatarg f4, t_floatarg f5, t_floatarg f6,
                           t_floatarg f7, t_floatarg f8, t_floatarg f9,
                           t_floatarg f10,t_floatarg f11,t_floatarg f12) {
    //marker L
    f->L.x = f1;
    f->L.y = f2;
    f->L.z = f3;

    //marker R
    f->R.x = f4;
    f->R.y = f5;
    f->R.z = f6;

    //marker U
    f->U.x = f7;
    f->U.y = f8;
    f->U.z = f9;

    //marker sorgente
    f->P.x = f10;
    f->P.y = f11;
    f->P.z = f12;
}
```

Come parametri di ingresso abbiamo, oltre al puntatore alla struttura `t_phasespace *f`, 12 numeri di tipo `float` a rappresentare le coordinate dei 4 sensori, che vengono assegnate ai corrispondenti *markers*.

**Calcolo dei *markers* mancanti** A questo punto si hanno solamente le coordinate dei *markers* acquisiti nel listato 4.3: sarà ora necessario calcolare anche gli altri punti della testa (vedi figura 4.3) per avere i dati sufficienti ad eseguire le successive elaborazioni.

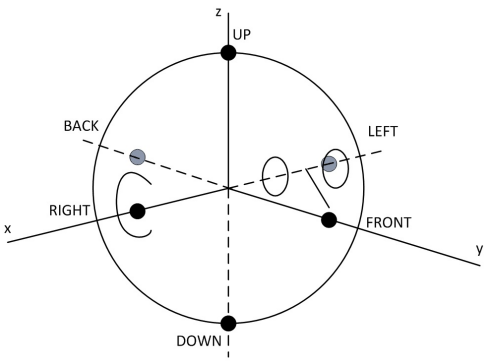



**Figura 4.3:** Disposizione dei *markers* sulla testa

**Listing 4.4: Calcolo dei *markers* mancanti**

```
void calcola_markers(t_phasespace *f) {

    // ---- ORIGIN ----
    f->O.x = (f->L.x + f->R.x)/2;
    f->O.y = (f->L.y + f->R.y)/2;
    f->O.z = (f->L.z + f->R.z)/2;        // punto medio tra  i markers L e R

    // ----- BACK -----
    t_coords v_1, v_2, v_3;

    v_1.x = f->L.x - f->R.x;
    v_1.y = f->L.y - f->R.y;
    v_1.z = f->L.z - f->R.z;             // vettore RL

    v_2.x = f->L.x - f->U.x;
    v_2.y = f->L.y - f->U.y;
    v_2.z = f->L.z - f->U.z;             // vettore UL
```

```
19
    v_3.x = v_1.y*v_2.z - v_1.z*v_2.y;
21  v_3.y = v_1.z*v_2.x - v_1.x*v_2.z;
    v_3.z = v_1.x*v_2.y - v_1.y*v_2.x;   // prodotto vettoriale tra v1 e v2
23
    t_float n_v1 = sqrt(pow(v_1.x, 2) + pow(v_1.y, 2) + pow(v_1.z, 2));
25  t_float n_v2 = sqrt(pow(v_2.x, 2) + pow(v_2.y, 2) + pow(v_2.z, 2));
    t_float n_v3 = sqrt(pow(v_3.x, 2) + pow(v_3.y, 2) + pow(v_3.z, 2));
27
    t_float sin_alpha = n_v3 / (n_v1 * n_v2);
29  t_float resize = 0.5 / (n_v2 * sin_alpha);

31  f->B.x = f->O.x + resize * v_3.x;
    f->B.y = f->O.y + resize * v_3.y;
33  f->B.z = f->O.z + resize * v_3.z;

35
    // ----- FRONT -----
37  f->F.x = f->O.z - resize * v_3.x;
    f->F.y = f->O.z - resize * v_3.y;
39  f->F.z = f->O.z - resize * v_3.z;

41
    // ----- DOWN -----
43  f->D.x = 2*(f->O.x) - f->U.x;
    f->D.y = 2*(f->O.y) - f->U.y;
45  f->D.z = 2*(f->O.z) - f->U.z;
}
```

- ORIGIN: l'origine del sistema riferito all'ascoltatore è situata nel punto medio tra i *markers* LEFT e RIGHT.
- BACK: per la determinazione delle coordinate di questo punto è necessario, innanzitutto, calcolare $\overrightarrow{RL}$ e $\overrightarrow{UL}$ per poi eseguirne il prodotto vettoriale (vedi nell'appendice la sezione A.1):

  $$\overrightarrow{V_1} = \overrightarrow{RL} \times \overrightarrow{UL} \qquad\qquad \overrightarrow{V_1} \perp \overrightarrow{RL},\ \overrightarrow{V_1} \perp \overrightarrow{UL}$$

  Questi due vettori giacciono entrambi sul piano mediano perciò il risultato del prodotto vettoriale, precedentemente eseguito, sarà un vettore perpendicolare a tale piano; la norma (vedi nell'appendice la sezione A.3) del vettore $V_1$:

  $$\|\overrightarrow{V_1}\| = \|\overrightarrow{RL}\|\|\overrightarrow{UL}\| \sin\alpha,$$

  sarà quindi necessario un ridimensionamento: si vuole infatti che la distanza tra l'origine dell'ascoltatore e il BACK sia uguale alla distanza dei *markers* LEFT e RIGHT. Il fattore di ridimensionamento avrà valore:

  $$k = \frac{1}{2\|\overrightarrow{UL}\| \sin\alpha},$$

  per ottenere infine:

  $$\overrightarrow{V_1}' = k\overrightarrow{V_1} = \frac{\overrightarrow{RL} \times \overrightarrow{UL}}{2\|\overrightarrow{UL}\| \sin\alpha}$$

  il quale possiede la corretta norma e la corretta direzione. L'ultimo passaggio consiste nel sommare questo vettore al vettore origine del sistema dell'ascoltatore:

  $$\overrightarrow{B} = \overrightarrow{O} + \overrightarrow{V_1}' = \overrightarrow{O} + k\overrightarrow{V_1}$$

- FRONT: è sufficiente sottrarre vettorialmente a $\overrightarrow{O}$ il vettore $\overrightarrow{V_1}'$ precedentemente calcolato:

  $$\overrightarrow{F} = \overrightarrow{O} - \overrightarrow{V_1}' = \overrightarrow{O} - k\overrightarrow{V_1}$$

- DOWN: basta sottrarre al vettore $\overrightarrow{O}$ il vettore $\overrightarrow{OU}$:

  $$\overrightarrow{D} = \overrightarrow{O} - \overrightarrow{OU} = \overrightarrow{O} - (\overrightarrow{U} - \overrightarrow{O}) = 2\overrightarrow{O} - \overrightarrow{U}$$

**Calcolo dei piani** Questa funzione calcola le costanti dell'*equazione implicita* del piano per i piani orizzontale, mediano e frontale.

**Listing 4.5: Calcolo dei piani**

```
void calcola_piani(t_phasespace *f) {

    // piano orizzontale   (markers B L R)
    f->Ho.A = (f->L.y-f->B.y)*(f->R.z-f->B.z) - (f->L.z-f->B.z)*(f->R.y-f->B.y);
    f->Ho.B = (f->L.z-f->B.z)*(f->R.x-f->B.x) - (f->L.x-f->B.x)*(f->R.z-f->B.z);
    f->Ho.C = (f->L.x-f->B.x)*(f->R.y-f->B.y) - (f->L.y-f->B.y)*(f->R.x-f->B.x);
    f->Ho.D = - (f->B.x*f->Ho.A) - (f->B.y*f->Ho.B) - (f->B.z*f->Ho.C);

    // piano mediano       (markers O U B)
    f->Me.A = (f->U.y-f->O.y)*(f->B.z-f->O.z) - (f->U.z-f->O.z)*(f->B.y-f->O.y);
    f->Me.B = (f->U.z-f->O.z)*(f->B.x-f->O.x) - (f->U.x-f->O.x)*(f->B.z-f->O.z);
    f->Me.C = (f->U.x-f->O.x)*(f->B.y-f->O.y) - (f->U.y-f->O.y)*(f->B.x-f->O.x);
    f->Me.D = - (f->O.x*f->Me.A) - (f->O.y*f->Me.B) - (f->O.z*f->Me.C);

    // piano frontale      (markers U L R)
    f->Fr.A = (f->L.y-f->U.y)*(f->R.z-f->U.z) - (f->L.z-f->U.z)*(f->R.y-f->U.y);
    f->Fr.B = (f->L.z-f->U.z)*(f->R.x-f->U.x) - (f->L.x-f->U.x)*(f->R.z-f->U.z);
    f->Fr.C = (f->L.x-f->U.x)*(f->R.y-f->U.y) - (f->L.y-f->U.y)*(f->R.x-f->U.x);
    f->Fr.D = - (f->U.x*f->Fr.A) - (f->U.y*f->Fr.B) - (f->U.z*f->Fr.C);
}
```

Per determinare le costanti di identificazione del piano basterà imporre la condizione A.3, ovvero che il determinante di tale matrice sia nullo. Eseguendo i calcoli ed esplicitando i coefficienti di $x$, $y$ e $z$ e il termine noto, si ottengono i risultati implementati nel listato 4.5:

$$a = (y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_1)$$
$$b = (z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_1)$$
$$c = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$
$$d = -x_1 a - y_1 b - z_1 c$$

I tre punti utilizzati per calcolare i piani sono i *markers* della testa, dei quali solamente LEFT, RIGHT e UP sono acquisiti dal sistema, mentre i rimanenti sono calcolati dalla funzione nel listato 4.4; è importante sottolineare come l'ordine dei punti considerati sia ininfluente ai fini del calcolo delle costanti.

**Calcolo delle coordinate cartesiane della sorgente** Si procede con il calcolo delle coordinate cartesiane del *marker* della sorgente rispetto all'ascoltatore:

**Listing 4.6: Coordinate della sorgente**

```
t_float new_x(t_phasespace *f) {

   t_float d_Me_P, dist_R, dist_L;

   //calcolo la distanza del punto P dal piano mediano
   d_Me_P = fabs(f->Me.A*f->P.x+f->Me.B*f->P.y+f->Me.C*f->P.z+f->Me.D) /
            sqrt(pow(f->Me.A,2)+pow(f->Me.B,2)+pow(f->Me.C,2));

   //verifico se P e' nel semispazio destro o sinistro
   dist_R = sqrt(pow(f->P.x-f->R.x,2) + pow(f->P.y-f->R.y,2) + pow(f->P.z-f->R.z,2));
   dist_L = sqrt(pow(f->P.x-f->L.x,2) + pow(f->P.y-f->L.y,2) + pow(f->P.z-f->L.z,2));
   if (dist_R <= dist_L)
      return d_Me_P;
   else
      return -d_Me_P;
}

//-------------------------------------------------------------------------------

t_float new_y(t_phasespace *f) {
   t_float d_Fr_P, dist_F, dist_B;

   //calcolo la distanza del punto P dal piano frontale
   d_Fr_P = fabs(f->Fr.A*f->P.x+f->Fr.B*f->P.y+f->Fr.C*f->P.z+f->Fr.D) /
            sqrt(pow(f->Fr.A,2)+pow(f->Fr.B,2)+pow(f->Fr.C,2));

   //verifico se P e' nel semispazio anteriore o posteriore
   dist_F = sqrt(pow(f->P.x-f->F.x,2) + pow(f->P.y-f->F.y,2) + pow(f->P.z-f->F.z,2));
   dist_B = sqrt(pow(f->P.x-f->B.x,2) + pow(f->P.y-f->B.y,2) + pow(f->P.z-f->B.z,2));
```

```
    if (dist_F <= dist_B)
31     return d_Fr_P;
    else
33     return -d_Fr_P;
}
35
//-------------------------------------------------------------------------------
37
t_float new_z(t_phasespace *f) {
39   t_float d_Ho_P, dist_U, dist_D;

41   //calcolo la distanza del punto P dal piano orizzontale
    d_Ho_P = fabs(f->Ho.A*f->P.x+f->Ho.B*f->P.y+f->Ho.C*f->P.z+f->Ho.D) /
43           sqrt(pow(f->Ho.A,2)+pow(f->Ho.B,2)+pow(f->Ho.C,2));

45   //verifico se P e' nel semispazio superiore o inferiore
    dist_U = sqrt(pow(f->P.x-f->U.x,2) + pow(f->P.y-f->U.y,2) + pow(f->P.z-f->U.z,2));
47  dist_D = sqrt(pow(f->P.x-f->D.x,2) + pow(f->P.y-f->D.y,2) + pow(f->P.z-f->D.z,2));
    if (dist_U <= dist_D)
49     return d_Ho_P;
    else
51     return -d_Ho_P;
}
```

Si utilizza la formula di distanza di un punto da un piano:

$$d(P,\pi) = \frac{|ax_P + by_P + cz_P + d|}{\sqrt{a^2+b^2+c^2}} \tag{4.1}$$

e poi si calcolano le distanze di tale punto da due opportuni *markers*, uno opposto all'altro rispetto all'origine, per determinare il segno corretto della coordinata da restituire.

**Calcolo della distanza e degli angoli di azimuth ed elevazione** Sono di seguito implementate delle funzioni per il calcolo la distanza e gli angoli di azimuth ed elevazione:

**Listing 4.7: Distanza e angoli della sorgente**

```
t_float distanza_sorgente(t_phasespace *f) {
2   return sqrt(pow(new_x(f),2) + pow(new_y(f),2) + pow(new_z(f),2));
}
4
//-------------------------------------------------------------------------------
6
t_float azimuth(t_phasespace *f) {
8
    t_float x = new_x(f);
10  t_float y = new_y(f);

12  t_float d_xy = sqrt(pow(x,2) + pow(y,2));

14  if (x == 0) {
       if (y >= 0)
16        return 0;
       else
18        return M_PI;
    }
20
    if (x > 0)
22     return -acos(y / d_xy);
    else
24     return acos(y / d_xy);
}
26
//-------------------------------------------------------------------------------
28
t_float elevation(t_phasespace *f) {
30
    t_float x = new_x(f);
32  t_float y = new_y(f);
    t_float z = new_z(f);
34
    t_float d_xyz = sqrt(pow(x, 2) + pow(y, 2) + pow(z, 2));
36  t_float d_xy  = sqrt(pow(x, 2) + pow(y, 2));

38  if (z == 0)
       return 0;
40  if (z > 0)
       return  acos(d_xy / d_xyz);
42  else
       return -acos(d_xy / d_xyz);
44 }
```

Si procede innanzitutto con il calcolo della norma del vettore $\overrightarrow{OP}$, cioè la distanza tra l'ascoltatore e la sorgente sonora:

$$r = \|\overrightarrow{OP}\| = \sqrt{d(P,Me)^2 + d(P,Fr)^2 + d(P,Ho)^2} = \sqrt{x_P^2 + y_P^2 + z_P^2}$$

L'angolo di azimuth $\theta$ viene calcolato grazie al legame che intercorre tra il modulo della coordinata $y_P$ e la norma della proiezione sul piano XY del vettore $\overrightarrow{OP}$

$$|y_P| = |cos\theta| \|\overrightarrow{OP_{XY}}\|$$

dove

$$\|\overrightarrow{OP_{XY}}\| = \sqrt{x_P^2 + y_P^2}$$

e, considerando gli opportuni casi, si ricava che:

$$\theta := \begin{cases} 0, & \text{se } x = 0 \text{ e } y \geq 0, \\ \pi, & \text{se } x = 0 \text{ e } y < 0, \\ -\arccos \frac{y_P}{\|\overrightarrow{OP_{XY}}\|}, & \text{se } x > 0, \\ \arccos \frac{y_P}{\|\overrightarrow{OP_{XY}}\|}, & \text{se } x < 0. \end{cases}$$

Manca infine l'angolo di elevazione, calcolato sfruttando la relazione:

$$\|\overrightarrow{OP_{XY}}\| = |\cos\phi| \|\overrightarrow{OP}\|$$

dove

$$\|\overrightarrow{OP}\| = \sqrt{x_P^2 + y_P^2 + z_P^2}$$

e, come già fatto in precedenza, se ne considerano le casistiche:

$$\phi := \begin{cases} 0, & \text{se } z = 0, \\ \arccos \frac{\|\overrightarrow{OP_{XY}}\|}{\|\overrightarrow{OP}\|}, & \text{se } z > 0, \\ -\arccos \frac{\|\overrightarrow{OP_{XY}}\|}{\|\overrightarrow{OP}\|}, & \text{se } z < 0. \end{cases}$$

L'importanza della differenziazione di queste casistiche nell'algoritmo sta, principalmente, nel fatto di evitare errori in esecuzione e, in secondo luogo, per renderlo più performante.

**Impostazione del sistema di coordinate** La funzione si occupa di acquisire il parametro `t_floatarg arg`, tramite il quale si imposta il tipo di rappresentazione che dovranno avere tre delle sei coordinate in uscita all'*external*, e di comunicare la scelta attraverso la Pd *window* (vedi sezione 3.1 ).

**Listing 4.8: Impostazione del sistema di coordinate**

```
void phasespace_system(t_phasespace *f, t_floatarg arg)
{
    f->set = arg;

    if (arg == 0)
       post(" COORDINATE CARTESIANE :  (x, y, z)");
    else if (arg == 1)
       post("COORDINATE INTERAURALI :  r  azimuth[0,360]  elevation[-90,90]");
    else if (arg == 2)
       post("  COORDINATE VERTICALI :  r  elevation[-90,90]  azimuth[0,360] ANTIORARIO");
    else if (arg == 3)
       post("  COORDINATE VERTICALI :  r  elevation[-90,90]  azimuth[0,360] ORARIO");
}
```

Le scelte possibili sono:

- COORDINATE CARTESIANE (`arg = 0`): rappresentazione di un punto tramite il vettore $(x, y, z)$.
- COORDINATE POLARI INTERAURALI (`arg = 1`): punto nello spazio descritto dal vettore $(r, \theta, \phi)$, cioè dal modulo, dall'angolo di azimuth e dall'angolo di elevazione (vedi figura 1.4a).

- COORDINATE POLARI VERTICALI (`arg = 2`): punto nello spazio descritto dal vettore $(r, \phi, \theta)$, cioè dal modulo, dall'angolo di elevazione e dall'angolo di azimuth (vedi figura 1.4b).
- COORDINATE POLARI VERTICALI (`arg = 3`): punto nello spazio descritto come al punto precedente, ma con l'angolo di azimuth crescente in senso orario.

**Impostazione dell'unità di misura degli angoli** Il parametro imposta uno specifico valore di conversione, per ottenere in uscita gli angoli nell'unità di misura desiderata; nel caso della scelta di coordinate cartesiane questo parametro risulta, vista l'assenza di angoli, ininfluente.

**Listing 4.9: Impostazione dell'unità di misura degli angoli**

```
1 void phasespace_angle(t_phasespace *f, t_floatarg arg)
  {
3
    f->angle = arg;                 //scelta la rappresentazione in gradi o radianti
5
    if (arg == 0) {
7      post("Angoli in radianti");
       f->angle_prod = M_PI / 180;  //imposto il valore di conversione
9   }
    else if (arg == 1) {
11     post("Angoli in gradi");
       f->angle_prod = 1;           //imposto il valore di conversione
13  }
  }
```

**Coordinate in uscita** Nella funzione, dopo aver acquisito ed elaborato i dati, si mandano in uscita le coordinate: le prime tre sempre in rappresentazione polare interaurale con gli angoli in gradi limitati (azimuth $\theta \in [0°, 360°]$ ed elevazione $\phi \in [-40°, 90°]$), le rimanenti tre uscite restituiscono, invece, le coordinate nella rappresentazione scelta dall'utente, grazie all'impostazione dei parametri trattati nei due precedenti paragrafi.

**Listing 4.10: Coordinate in uscita**

```
  void phasespace_observer_coords(t_phasespace *f, t_symbol *s, int argc, t_atom *argv)
2 {
4   if (argc == 12) {          // mi accerto che il numero di dati passati sia corretto
6      t_float k[12];          // creo un vettore di 12 elementi
8      // copio i dati passati come parametri nel vettore appena creato
       int i;
10     for (i = 0; i < 12; i++)
          k[i] = atom_getfloat(argv + i);
12
       // carico i valori nella struttura phasespace
14     take_observer_coords(f,k[0],k[1],k[2],k[3],k[4],k[5],k[6],k[7],k[8],k[9],k[10],k[11]);
16     calcola_markers(f);     // calcolo i markers mancanti
       calcola_piani(f);       // calcolo i piani orizzontale, mediano e frontale
18
20     t_float az = 180 / M_PI * azimuth(f);
       t_float el = 180 / M_PI * elevation(f);      // angoli in gradi
22     t_float d  = distanza_sorgente(f);
24
       // Predispongo gli angoli in modo che siano compresi nel range di valori richiesto
26     // 0 < azimuth < 360
       while (az < 0)
28        az += 360;
       while (az > 360)
30        az -= 360;
       // -40 < elevation < 90
32     t_float el_limitato = el;
       if (el_limitato < -40)
34        el_limitato = -40;
36     // ------------------------------------------------------------------------------
       // le prime 3 uscite sono in rappresentazione INTERAURALE con i corretti range
38     outlet_float(f->out1, d);
       outlet_float(f->out2, az);
40     outlet_float(f->out3, el_limitato);
42
```

```
        // le altre 3 invece sono a scelta dell'utente
44      if (f->set == 0) {                  // CARTESIANA
           outlet_float(f->out4, new_x(f));
46         outlet_float(f->out5, new_y(f));
           outlet_float(f->out6, new_z(f));
48      }
        else {
50         if (f->set == 1) {               // INTERAURALE
              outlet_float(f->out4, d);
52            outlet_float(f->out5, az * f->angle_prod);
              outlet_float(f->out6, el * f->angle_prod);
54         }
           else if (f->set == 2) {          // VERTICALE (azimuth antiorario)
56            outlet_float(f->out4, d);
              outlet_float(f->out5, el * f->angle_prod);
58            outlet_float(f->out6, az * f->angle_prod);
           }
60         else if (f->set == 3) {          // VERTICALE (azimuth orario)
              outlet_float(f->out4, d);
62            outlet_float(f->out5, el * f->angle_prod);
              outlet_float(f->out6, (360 - az) * f->angle_prod);
64         }
        }
66   }
     else post("Numero di coordinate non corretto.");
68 }
```

**Funzioni per *PureData*** Le funzioni di seguito sono proprie delle *external* di Pd-Extended:

`phasespace_bang` è una funzione che viene chiamata quando vengono spediti dei messaggi ad un'istanza della classe; questo particolare tipo di funzioni sono interfacciati con il sistema di messaggi di Pd-Extended.

`*phasespace_new` crea una nuova classe e ritorna un puntatore al suo prototipo; il primo argomento è il nome simbolico della classe, mentre gli ultimi due definiscono il costruttore e il distruttore della classe.

Quando infatti viene creato un nuovo oggetto, il costruttore di classe `(t_phasespace *)pd_new(phasespace_class)` alloca ed inizializza lo spazio di memoria; nel caso in cui, invece, un oggetto venga distrutto, il distruttore libera la memoria dinamica riservata.

`phasespace_setup` è la funzione che permette a *Pure Data* di creare e trovare la classe.

**Listing 4.11: Funzioni per *PureData***

```
  void phasespace_bang(t_phasespace *f) {
2    post("bang received");
  }
4

6 void *phasespace_new(t_symbol *s, int argc, t_atom *argv) {
     t_phasespace *x = (t_phasespace *)pd_new(phasespace_class);
8    inlet_new(&x->x_obj, &x->x_obj.ob_pd, &s_list, gensym("observer_coords"));

10   x->out1 = outlet_new(&x->x_obj, &s_float);
     x->out2 = outlet_new(&x->x_obj, &s_float);
12   x->out3 = outlet_new(&x->x_obj, &s_float);

14   x->out4 = outlet_new(&x->x_obj, &s_float);
     x->out5 = outlet_new(&x->x_obj, &s_float);
16   x->out6 = outlet_new(&x->x_obj, &s_float);

18   return (void *)x;
  }
20

22 void phasespace_setup(void) {
     phasespace_class = class_new(gensym("phasespace"), (t_newmethod)phasespace_new,
24                      0, sizeof(t_phasespace), CLASS_DEFAULT, A_GIMME, 0);
     class_addbang(phasespace_class, phasespace_bang);
26   class_addmethod(phasespace_class, (t_method) phasespace_observer_coords,
                     gensym("observer_coords"), A_GIMME, 0);
28   class_addmethod(phasespace_class, (t_method) phasespace_system,
                     gensym("system"), A_DEFFLOAT, 0);
30   class_addmethod(phasespace_class, (t_method) phasespace_angle,
                     gensym("angle"), A_DEFFLOAT, 0);
32 }
```

**La compilazione dell'*external*** La compilazione è stata eseguita tramite un MAKEFILE appositamente creato. Risulterà necessario eseguire da bash, nella directory del file C, il seguente comando:

```
:~$make NAME=phasespace
```

ad ottenere l'*external* "*phasespace.pd_linux*".
Il comando `:~$make clean` elimina, invece, tutti i file C e le *external* compilate.

**Listing 4.12: Makefile**

```
NAME =                          #inserito da terminale
TARGET = $(NAME).pd_linux
CC = gcc


all: $(TARGET)


.SUFFIXES: .pd_linux
CFLAGS = -DPD -O2 -Wall -W -Wshadow -Wstrict-prototypes
         -Wno-unused -Wno-parentheses -Wno-switch $(OPT_CFLAGS)


.c.pd_linux:
   $(CC) -Wall -c -o $(NAME).o -fPIC $(NAME).c
   ld -export-dynamic -shared -o $(NAME).pd_linux $(NAME).o -lm
   rm -f $*.o


clean:
   rm -f *.o *.pd_* so_locations
```

## 4.2 La patch "*phasespace_patch.pd*"

**Sensori della testa** Per ogni punto della testa di cui acquisire la posizione (cioè LEFT, RIGTH, UP) vengono posizionati due *markers* (vedi figura 4.1a); questo per garantire una maggior sicurezza che la posizione venga acquisita, visto che i sensori possono essere oscurati. La serie di sensori è collegata alla *LED Driver Unit*, la quale riceve il segnale di sincronizzazione dalla *LED Base Station*.

Nello spezzone di programma *PureData*, presentato in figura 4.4, si hanno come ingressi le coordinate fornite dal *PhaseSpace* riguardo ai sensori implicati nell'*head tracking*; questi ingressi consistono di pacchetti di quattro numeri in virgola mobile, che devono essere separati tramite il comando unpack.

I primi tre numeri rappresentano le coordinate, con l'accortezza di negare il primo e di invertire l'ordine gli altri due: il tutto per garantire che il sistema di assi finale corrisponda con quello considerato nell'*external*. Con i blocchi send si spedisce il pack di tre numeri numeri in virgola mobile alla sezione di programma in figura 4.6; con il comando receive vengono ricevute le coordinate che, in seguito ad un unpack, vengono divise di un fattore 1000: infatti le misure sono espresse in millimetri, mentre si desidera convertirle in metri.

Segue il blocco change che ha la funzionalità di mandare in uscita l'ingresso solamente quando questo cambia.

Il blocco trigger (trigger bang float o equivalentemente t b f) spedisce in uscita gli ingressi da destra verso sinistra: quindi prima verrà inviato l'elemento float a pack f f e poi il bang all'inlet calda dello stesso per aggiornarne l'uscita.

Il passaggio successivo consiste nel verificare se entrambi i sensori sono stati individuati e nell'agire di conseguenza:

- se $f1$\neq 0$ e $f2$\neq 0$ allora si esegue una media delle coordinate
- se $f1$= 0$ e $f2$\neq 0$ allora si considera solamente la coordinata $f2
- se $f1$\neq 0$ e $f2$= 0$ allora si considera solamente la coordinata $f1

Il blocco sel 0 ha la funzione di filtrare tutte le coordinate di valore nullo che potrebbero susseguirsi in un momento di oscuramento del sensore rispetto alle telecamere: infatti i blocchi centrali loadbang, metro 30 e 0 hanno la funzionalità di azzerare continuamente le coordinate. Il blocco loadbang carica, appunto, un bang al blocco metro 30 che, a sua volta, invia un bang

ogni 30 millisecondi al *message box*; quest'ultimo ad ogni impulso invierà in uscita il messaggio riportato nel proprio blocco, cioè il numero 0.

I valori in uscita dai tre processi paralleli, uno per ogni coordinata, vengono convogliati nel blocco pack f f f, ovvero in un pacchetto di tre numeri in virgola mobile, per poi essere spediti ad elaborazioni successive tramite gli oggetti send right, send left e send up.

**Sensore della sorgente** Per quanto concerne la sorgente sonora (vedi figura 4.4), si utilizza un solo sensore e, perciò, non si rivela necessaria la procedura di controllo subita dalle coordinate dei sensori della testa di cui si è appena discusso. Come inlet (ingresso) si hanno anche in questo caso le coordinate fornite dal *PhaseSpace* in un pack di quattro numeri in virgola mobile: come fatto anche per i precedenti sensori si nega la prima coordinata e si invertono le due successive, mentre il quarto elemento non risulta di interesse. Il passaggio della divisione per 1000 è effettuato immediatamente prima del pack e del send.

**Elaborazione delle coordinate** In figura 4.5 le coordinate appena acquisite vengono elaborate dall'*external* precedentemente esposta. Le prime tre uscite saranno le coordinate relative espresse in rappresentazione polare interaurale, che serviranno per elaborazioni successive, mentre le rimanenti saranno espresse nella rappresentazione scelta dall'utente.

***Air Absorption*** L'assorbimento dell'aria è un fenomeno che, in funzione della distanza, attenua i suoni ad alta frequenza: il comportamento dell'aria può essere quindi paragonato ad un filtro passa-basso. Sebbene il fenomeno sia complesso, nel presente progetto si vuole fare una riproduzione qualitativa del fenomeno: si utilizza un filtro passa-basso del secondo ordine (vedi in figura 4.8) che, in base all'uscita dello spezzone di programma in figura 4.7 (determinata dalla distanza), filtra il segnale audio.

***Doppler Effect*** L'effetto Doppler è un cambiamento apparente della frequenza o della lunghezza d'onda di un'onda percepita da un osservatore che si trova in movimento rispetto alla sorgente delle onde. Per quelle onde che si trasmettono in un mezzo, come le onde sonore, la velocità dell'osservatore e dell'emettitore vanno considerate in relazione a quella del mezzo in cui sono trasmesse le onde. L'effetto Doppler totale può quindi derivare dal moto di entrambi, ed ognuno di essi è analizzato separatamente[5].
Nel presente progetto, questo cambiamento apparente di lunghezza d'onda, viene simulato qualitativamente introducendo un ritardo nell'emissione dei campioni audio.

**Attenuazione distanza** L'attenuazione della distanza (vedi sezione 1.2.3) è realizzata semplicemente applicando la relazione già descritta: è una riproduzione qualitativa che non risulta essere molto fedele alla realtà.

**Schema principale** In figura 4.8 si possono osservare i passaggi che deve subire il suono prima di essere riprodotto:

- innanzitutto il suono viene acquisito come *inlet*;
- si applica il *Doppler Effect* in base a quanto già descritto;
- viene effettuata un'operazione di filtraggio, tramite un filtro passa-basso del secondo ordine, per simulare l'assorbimento dell'aria;
- i campioni vengono attenuati in base alla distanza;
- il file audio viene spazializzato dall'oggetto earplug~, nel quale viene effettuata una convoluzione dei campioni con le risposte impulsive del KEMAR, eventualmente in seguito ad un'interpolazione;
- infine i campioni vengono mandati in uscita secondo i due canali stereo.

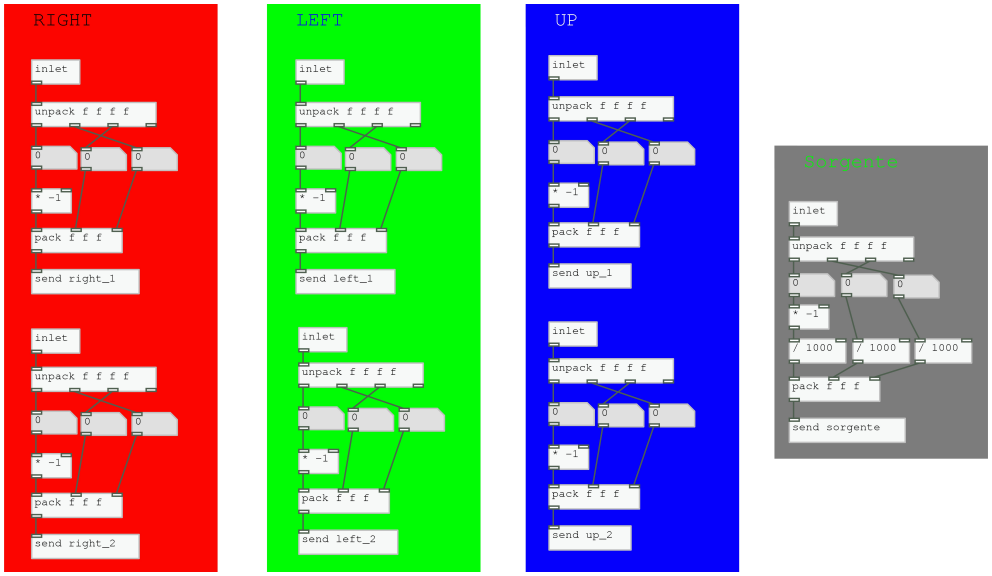



**Figura 4.4:** Acquisizione dei dati, relativi ai sensori della testa e della sorgente, in uscita dal *PhaseSpace Impulse*

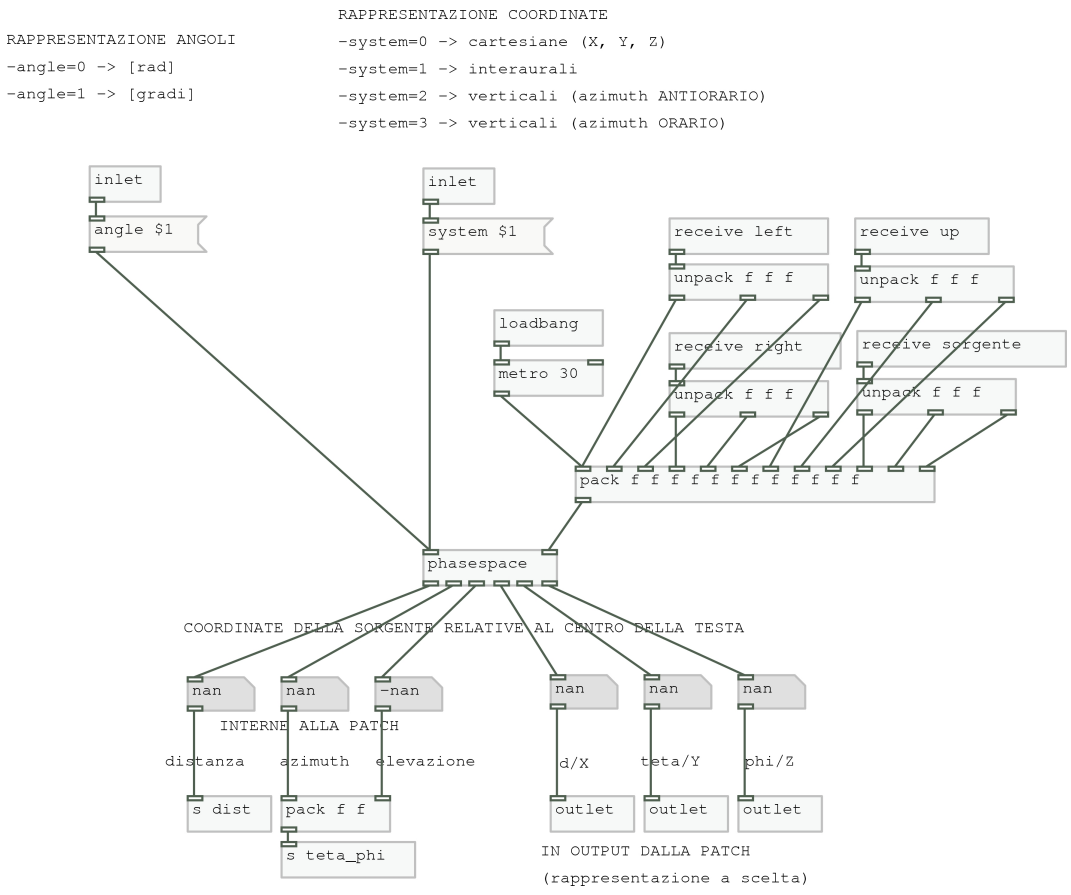



**Figura 4.5:** Elaborazione delle coordinate

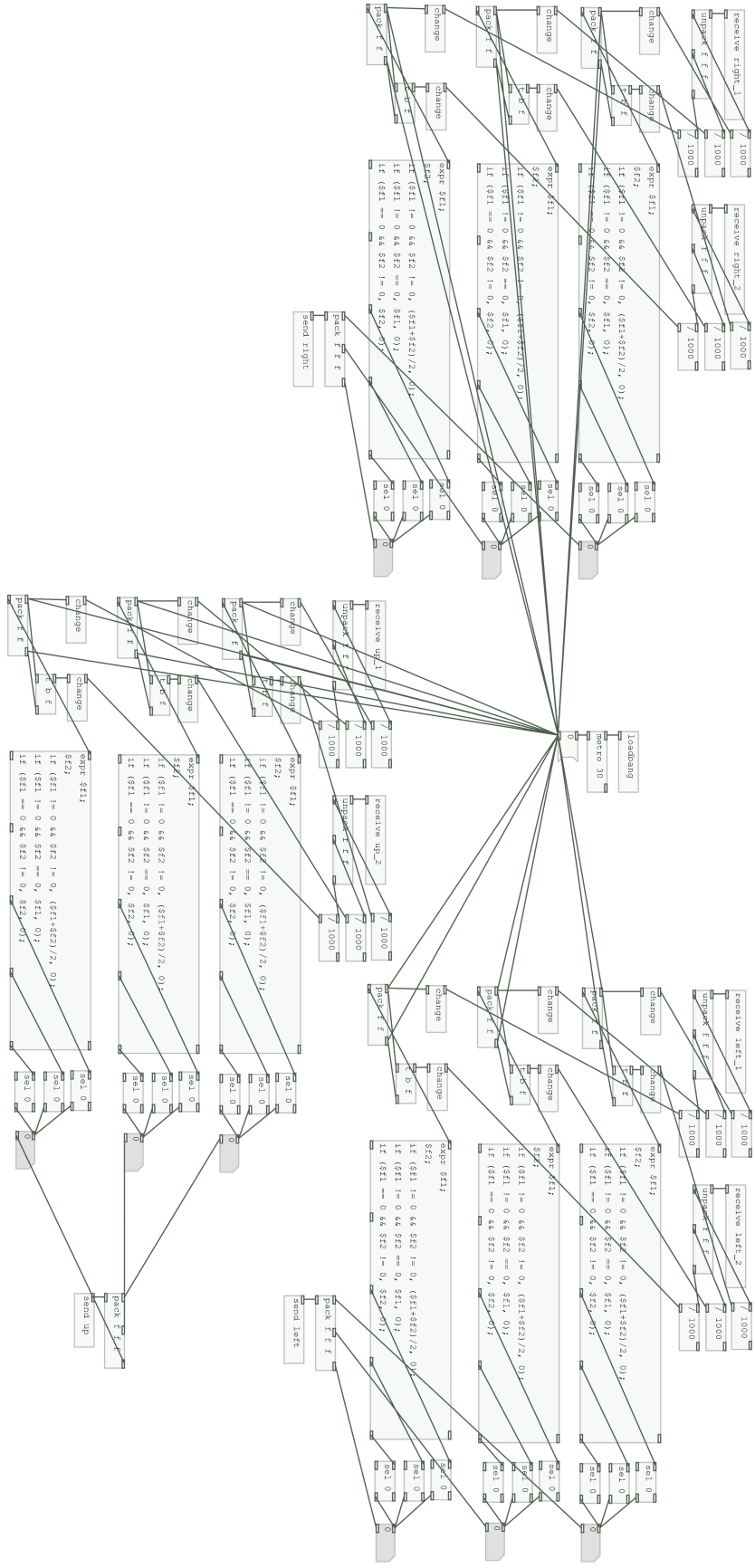


**Figura 4.6:** Verifica delle coordinate dei sensori della testa

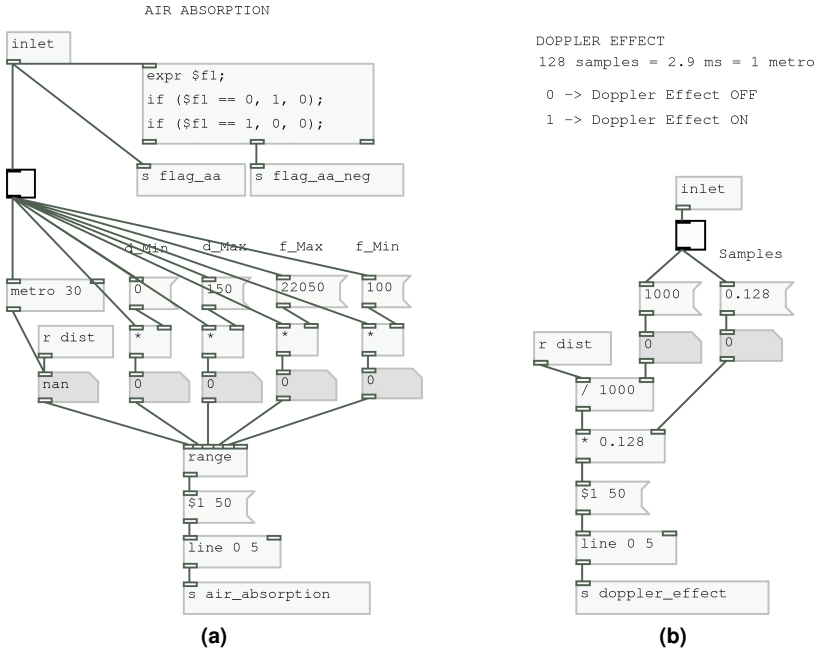



**Figura 4.7:** *Air Absorption* **(a)** e *Doppler Effect* **(b)**

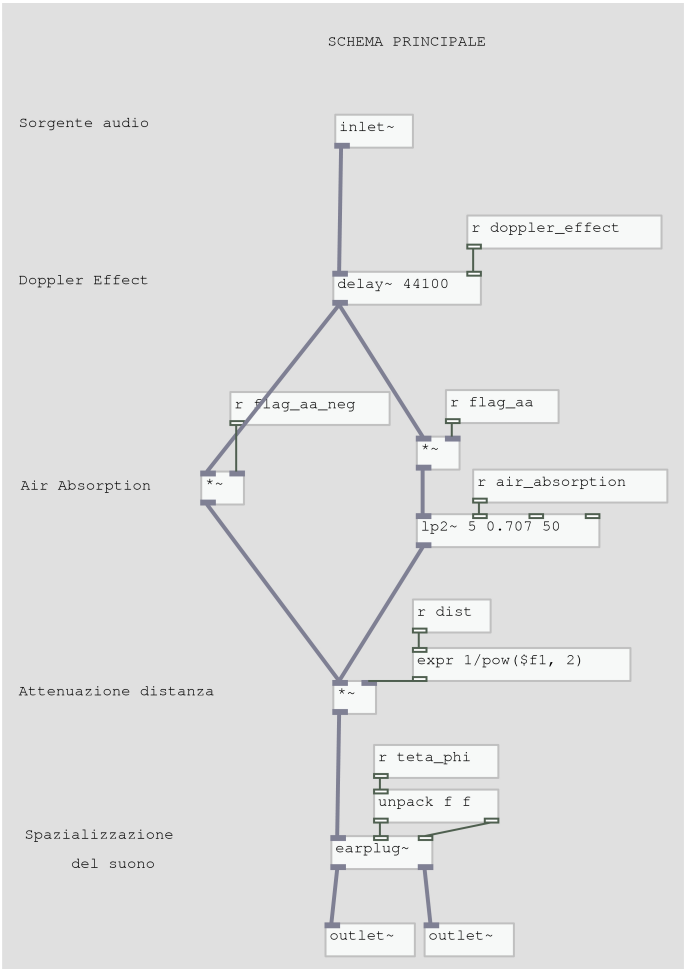



**Figura 4.8:** Schema principale

## 4.3 Esempio di utilizzo e valutazioni

L'*object box* ottenuta risulta essere molto comoda: tutta la parte di elaborazione è infatti compattata, permettendo un più rapido utilizzo del sistema, nonché un ambiente di lavoro più ordinato e modulare. Nella figura seguente se ne può osservare un semplice utilizzo.

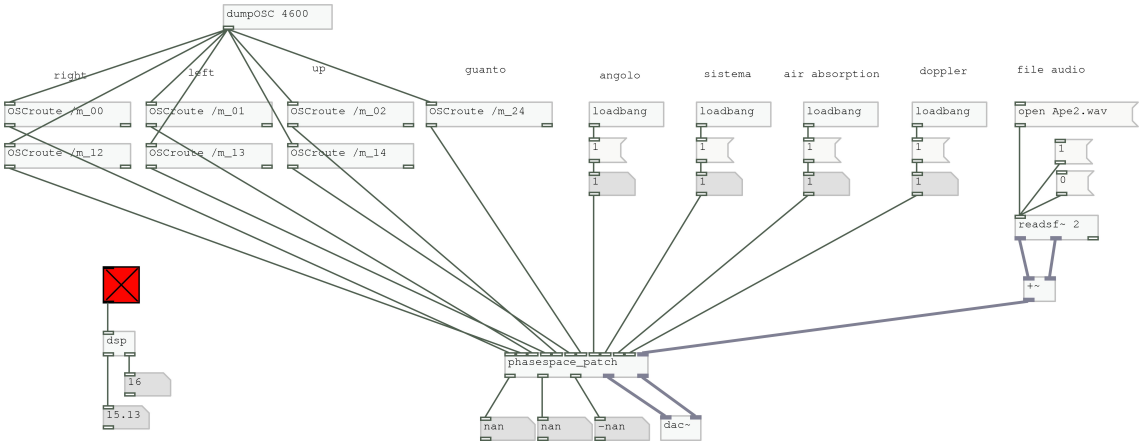



**Figura 4.9:** Esempio di utilizzo della *patch*

La *patch* realizzata non riesce a riprodurre in maniera fedele la spazializzazione per molteplici motivi:

- i sensori delle orecchie sulle cuffie, non sono stati posizionati in corrispondenza dell'orecchio ma leggermente più in su;
- le misurazioni delle risposte impulsive dell'oggetto earplug~ sono state eseguite in una camera anecoica, perciò gli effetti di riverbero e le riflessioni, ad esempio del pavimento, non sono riprodotte;
- le computazioni non sono elaborate in maniera fluida, compromettendo la pulizia del segnale audio in particolare quando si effettuano spostamenti bruschi della sorgente rispetto alla testa;
- l'effetto *Doppler* è difficilmente percettibile, a causa del problema di fluidità esposto al punto precedente;
- l'effetto di assorbimento dell'aria è difficilmente percettibile, visto che a breve distanza non si manifesta in maniera significativa;
- l'effetto di simulazione della distanza, essendo basato su un modello qualitativo riferito ad una camera anecoica, non può dirsi soddisfacente: l'intensità sonora dovrebbe infatti decrescere meno velocemente rispetto alla distanza.

In conclusione si è realizzato un modulo molto comodo per *Pure Data*, facilmente utilizzabile e modificabile anche per altre applicazioni con il *Phasespace Impulse*, che però adotta modelli qualitativi: un primo miglioramento starebbe appunto nell'adottare modelli più accurati.

# Appendice A

# Richiami matematici

## A.1 Il prodotto vettoriale

**Definizione 1.** (Prodotto vettoriale o esterno) *Dati due vettori* $\boldsymbol{u}$ *e* $\boldsymbol{v}$ *si dice loro* prodotto vettoriale o esterno *il vettore* $\boldsymbol{w}$*, che si indica con* $\boldsymbol{u} \times \boldsymbol{v}$ *(si legge* $\boldsymbol{u}$ *vettoriale* $\boldsymbol{v}$*), definito come segue[9]:*

*(i) se* $\boldsymbol{u}$ *e* $\boldsymbol{v}$ *sono paralleli* $\boldsymbol{u} \times \boldsymbol{v} = 0$

*(ii) se* $\boldsymbol{u}$ *e* $\boldsymbol{v}$ *non sono paralleli, il vettore risultante:*

- *ha modulo pari a* $\|\boldsymbol{w}\| = \|\boldsymbol{u} \times \boldsymbol{v}\| = \|\boldsymbol{u}\|\|\boldsymbol{v}\| \sin \widehat{\boldsymbol{uv}}$*;*
- *ha direzione perpendicolare sia a* $\boldsymbol{u}$ *che a* $\boldsymbol{v}$*;*
- *ha il verso di avanzamento di una vite destrorsa, che ruota nel senso in cui* $\boldsymbol{u}$ *si sovrappone a* $\boldsymbol{v}$*.*

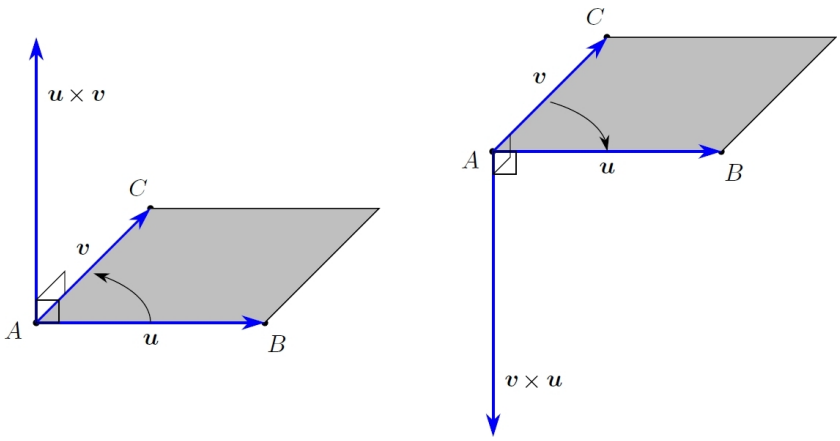



**Figura A.1:** Esempi grafici di prodotti vettoriali tra due vettori

Dati due vettori $\boldsymbol{u} = (u_x, u_y, u_z)$ e $\boldsymbol{v} = (v_x, v_y, v_z)$ si considera la seguente matrice simbolica:

$$\begin{pmatrix} \boldsymbol{x} & \boldsymbol{y} & \boldsymbol{z} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{pmatrix}$$

Il *prodotto vettoriale* dei due vettori si ottiene calcolando il determinante simbolico della matrice:

$$\begin{aligned} \boldsymbol{u} \times \boldsymbol{v} = \begin{vmatrix} \boldsymbol{x} & \boldsymbol{y} & \boldsymbol{z} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix} &= (u_y v_z - u_z v_y)\boldsymbol{x} - (u_x v_z - u_z v_x)\boldsymbol{y} + (u_x v_y - u_y v_x)\boldsymbol{z} \\ &= (u_y v_z - u_z v_y)\boldsymbol{x} + (u_z v_x - u_x v_z)\boldsymbol{y} + (u_x v_y - u_y v_x)\boldsymbol{z}. \end{aligned} \tag{A.1}$$

## A.2 Il piano nello spazio

Consideriamo l'*equazione implicita del piano* in $\mathbb{R}^3$:

$$ax + by + cz + d = 0 \tag{A.2}$$

Perché sia un'equazione di primo grado in tre incognite, come si vuole che sia, occorre che i coefficienti $a$, $b$ e $c$ non siano contemporaneamente nulli; sono equivalenti le espressioni: $a^2 + b^2 + c^2 > 0$ oppure $|a| + |b| + |c| > 0$.

Siano dati tre punti non allineati nello spazio:

$$\boldsymbol{v}_1 = (x_1, y_1, z_1) \qquad \boldsymbol{v}_2 = (x_2, y_2, z_2) \qquad \boldsymbol{v}_3 = (x_3, y_3, z_3)$$

Per determinare le costanti di identificazione del piano su cui giacciono tutti e tre i punti, basterà imporre la seguente condizione:

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0 \tag{A.3}$$

ovvero che il determinante di tale matrice sia nullo. Eseguendo i calcoli ed esplicitando i coefficienti di $x$, $y$, $z$ ed il termine noto, si ottengono i seguenti risultati:

$$\begin{aligned} a &= (y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_1) \\ b &= (z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_1) \\ c &= (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1) \\ d &= -x_1 a - y_1 b - z_1 c \end{aligned} \tag{A.4}$$

## A.3 La norma

**Definizione 2.** *Sia $V$ uno spazio vettoriale reale o complesso. Si dice* norma *una funzione con una corrispondenza $\boldsymbol{x} \mapsto \|\boldsymbol{x}\|$ da $V$ in $\mathbb{R}$ tale che:*

*(i)* $\boldsymbol{x} \neq \boldsymbol{0} \implies \|\boldsymbol{x}\| > 0$;

*(ii)* $\|a\boldsymbol{x}\| = |a|\|\boldsymbol{x}\|, \quad \forall a \in \mathbb{K}, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in V$;

*(iii)* $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in V$.

*La coppia $(V, \|\cdot\|)$ si dice* spazio vettoriale normato*, abbreviato s.v.n.[8].*

In generale, per ogni numero $p \geq 1$, si può considerare la norma:

$$\|\boldsymbol{x}\|_p := \left( \sum_{k=1}^{n} \|x_k\|^p \right)^{\frac{1}{p}}$$

in particolare se $p = 2$, la norma si dice *euclidea*:

$$\|\boldsymbol{x}\|_2 := \left( \sum_{k=1}^{n} \|x_k\|^2 \right)^{\frac{1}{2}}$$

Sia, inoltre, il campo $\mathbb{K} = \mathbb{R}^3$; allora $\boldsymbol{x} = (x_1, x_2, x_3)$ e

$$\|\boldsymbol{x}\|_2 := \left( \sum_{k=1}^{3} \|x_k\|^2 \right)^{\frac{1}{2}} = \sqrt{\|x_1\|^2 + \|x_2\|^2 + \|x_3\|^2},$$

che può essere scritta come:

$$\|\boldsymbol{x}\|_2 := \sqrt{|x_1|^2 + |x_2|^2 + |x_3|^2} \tag{A.5}$$

# Bibliografia


[1] F. Avanzini, 2009. "*Sound in space*", in Algorithms for Sound and Music Computing, for Computer Science class in Sound and Music Computing.

[2] S. Spagnol, 2007-2008. "*Rendering in tempo reale di audio spaziale nel campo vicino*".

[3] S. Mezzalira, 2010-2011. "*Realizzazione di un algoritmo real-time per il rendering spaziale binaurale del suono, basato su modelli antropomorfici*".

[4] F. Ficarra, 2009-2010. "*Motion capture, rendering binaurale e realtà virtuale*".

[5] `http://it.wikipedia.org/`.

[6] B. Gardner, K. Martin, MIT Media Lab, "*HRTF Measurement of KEMAR Dummy-Head Microphone*".

[7] M. Puckette, `http://crca.ucsd.edu/~msp/Pd_documentation/index.htm`.

[8] G.C. Barozzi, 2004. "*Matematica per l'Ingengneria dell'Informazione*", Zanichelli.

[9] Luciano Battaia, "*Algebra lineare e geometria analitica*", `http://www.batmath.it`.

[10] Francesco Bianchi, "*Inventare il suono con Pure Data*", Manuale introduttivo di musica elettronica.