Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías De Telecomunicación

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

APLICACIÓN AURALIZACIÓN PARA TABLETA

Alumno: **Juan Demóstenes Asumu Elá Messe**

Tutor: Prof. D. Pedro Vera Candeas
Depto.: Ingeniería de Telecomunicación

Septiembre, 2016

ÍNDICE DE CONTENIDOS

1 RESUMEN

En esta memoria se describen los mecanismos que utiliza el sistema auditivo humano para determinar o localizar las fuentes sonoras en un espacio tridimensional. Con el objetivo de replicar estos métodos, se introducen los conceptos de audición binaural, los filtros HRTF y los métodos existentes para obtenerlos. Posteriormente se explican las distintas forman que existen para filtrar señales digitales; finalmente se describe el proceso seguido para desarrollar una aplicación que simule el efecto de la auralización.

2 INTRODUCCIÓN

Si repasamos el panorama actual, en cuanto a sistemas audiovisuales se refiere, nos encontraremos con un creciente interés en tecnologías como la realidad virtual o la realidad aumentada. Aunque estas tecnologías se centran más en la percepción visual del usuario, es necesario disponer de un sistema de reproducción capaz de mejorar la sensación de inmersión y realismo en los sistemas audiovisuales.

Con el objetivo de mejorar la calidad del sonido en los sistemas audiovisuales, se hace necesario hablar de la audición binaural. ¿Qué es la audición binaural? La audición binaural se puede definir como la capacidad de percibir el sonido por dos oídos. Esta característica de nuestro sistema auditivo es la base de la audición espacial o 3D que nos permite determinar la localización de una fuente sonora en el espacio.

En los sistemas audiovisuales *"la escucha"* hace referencia al soporte o métodos utilizados para grabar-reproducir el sonido. Según esta definición, podemos diferenciar la escucha monofónica, el sonido se graba y reproduce usando un solo canal; de la escucha estereofónica, el sonido se graba y reproduce con más de un canal.

Desde que fuera presentada en 1931, el sistema de escucha estereofónico es el más común entre los sistemas audiovisuales comerciales. Con este sistema se pretendía simular la audición binaural humana utilizando dos canales de grabación-reproducción.

Aunque los sistemas estereofónicos permiten añadir cierto carácter espacial al sonido, estos no consiguen simular fielmente la escucha binaural humana.

Existen dos alternativas a los sistemas estereofónicos convencionales:

- **Sistemas de sonido multicanal**. Se trata de sistemas estereofónicos de más de dos canales colocados alrededor del oyente, ver figura 2.1. En estos sistemas el soporte de reproducción suelen ser altavoces, a cada canal (altavoz) le suele corresponder una determinada dirección del espacio.

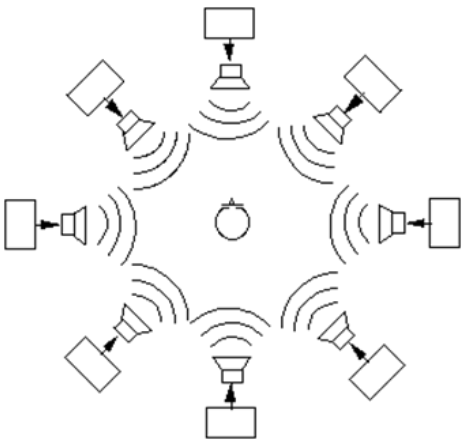

Figura 2.1 Sistema multicanal

- **Sistemas de sonido binaural.** Son sistemas estereofónicos de dos canales en los que las características espaciales se consiguen procesando las señales de audio captadas mediante métodos especiales de procesamiento de audio. En estos sistemas el soporte de reproducción suele ser simples auriculares. En la figura 2.2 se puede observar una representación de un sistema de grabación-reproducción binaural.

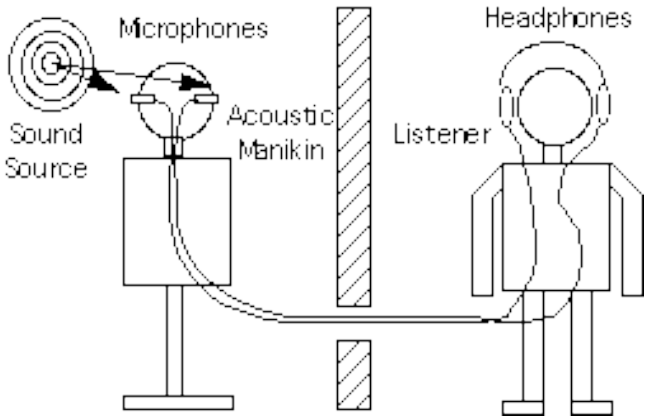

Figura 2.2 Sistema binaural

En este proyecto nos centraremos en el segundo método, a diferencia de los sistemas multicanal, los sistemas de audición binaural son más baratos y permiten reproducir sonidos con características espaciales usando simples auriculares. Su implementación se reduce al procesado de las señales de audio captadas con un tipo especial de filtros denominados HTRF, de los que se hablará más adelante.

2.1 Localización de fuentes

Para poder simular la escucha espacial humana es indispensable conocer qué métodos y técnicas utiliza el sistema auditivo humano para determinar la localización de fuentes sonoras en el espacio.

Para determinar la localización de una fuente sonora, primero debemos disponer de un sistema de coordenadas con el que describir todos los puntos del espacio tridimensional. Por su parecido, la cabeza humana puede modelarse como una esfera, por ello es conveniente usar el sistema de coordenadas esféricas como el sistema de coordenadas de referencia.

El punto de origen se situará en el centro de la cabeza; cada punto del espacio se representará mediante tres variables: el azimut (θ), la elevación (φ) y el radio (r), ver figura 2.3.

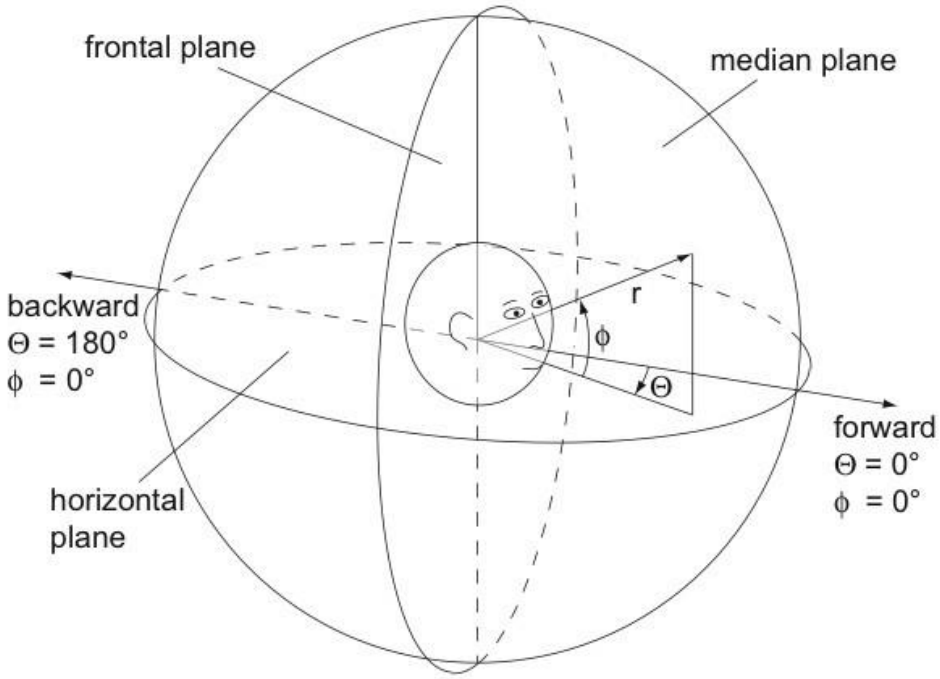

Figura 2.3 Sistema de coordenadas esféricas

El sistema auditivo humano está formado por dos canales independientes separados entre sí, el oído izquierdo y el oído derecho. Al estar separados el uno del otro, a cada oído le llega una señal diferente. Estas diferencias entre las señales que llegan a los oídos se denominan diferencias interaurales. Las diferencias interaurales junto con otros mecanismos monoaurales y cognitivos constituyen el principal mecanismo utilizado por el sistema auditivo para determinar la localización de fuentes sonoras en el espacio.

En 1907 John Structt, conocido como el Barón Rayleigh, desarrollo una teoría sobre la audición espacial denominada teoría dúplex. La teoría dúplex establece que existen dos sistemas fundamentales para determinar la localización de una fuente sonora, las **diferencias interaurales de tiempo (ITD)** y las **diferencias interaurales de nivel (ILD)**. Más tarde se observó que las diferencias interaurales de tiempo (ITD) y las diferencias interaurales de nivel (ILD), solo permitían determinar la posición de una determinada fuente sonora en el plano azimutal o plano horizontal.

Estimar completamente la posición de una fuente sonora requiere de varios parámetros como: el ITD, el ILD, la respuesta del pabellón auditivo, el movimiento de la cabeza y el movimiento de la fuente, las características de la distancia a la fuente, la reverberación y el eco. Generalmente se le da una mayor importancia a la ITD y la ILD porque fisiológicamente a la hora de localizar fuentes, el ser humano es más hábil estimando el azimut, algo hábil estimando la elevación, y nada hábil determinando la distancia a la fuente.

2.1.1 Determinar el ángulo lateral

Como se ha mencionado en el apartado 2.1, las diferencias interaurales de tiempo y las diferencias interaurales de intensidad constituyen el principal mecanismo del sistema auditivo para determinar la localización de fuentes en el plano horizontal. En un espacio tridimensional, utilizando el sistema de coordenadas esféricas, esto se traduce en determinar a qué azimut (θ) se encuentra la fuente respecto al oyente.

Diferencias Interaurales de Tiempo (ITD). Las señales de audio procedentes de una determinada fuente sonora recorren diferentes caminos para llegar a cada uno de los oídos, ver figura 2.4. Estas diferencias en el trayecto recorrido, suponen una diferencia entre los tiempos de llegada a cada uno de los oídos. El sistema auditivo humano utiliza esta diferencia en los tiempos de llegada para determinar la dirección de procedencia del sonido.

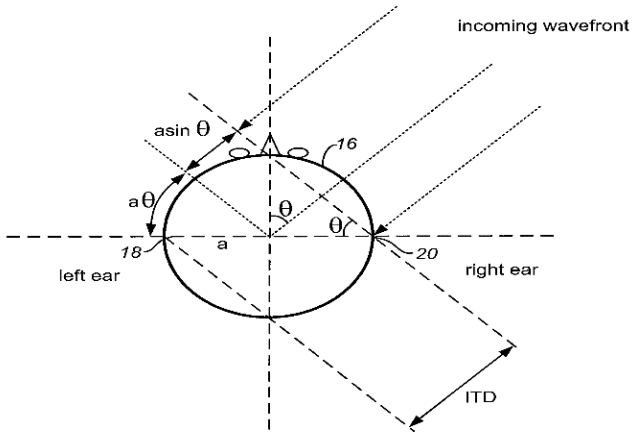

Figura 2.4 Representación ILD

Este mecanismo solo es válido para señales con una longitud de onda superior al doble de la distancia que separa ambos oídos. Esto se debe a que a menor longitud de onda, las ondas se reflejan más fácilmente produciendo que la mayor parte de ondas lleguen en fase. El diámetro medio de una cabeza humana puede aproximarse en unos 20cm; por encima de los 1000Hz, el diámetro medio de la cabeza es mayor que la longitud de onda de las señales; por lo tanto, la ITD no es útil para frecuencias mayores de 1000Hz.

Por su parte, **las Diferencias Interaurales de Nivel (ILD)** utilizan las diferencias de intensidad entre las señales que llegan a los oídos para determinar la dirección de procedencia; el sonido sonará más fuerte en el oído más cercano, ver figura 2.5. Este mecanismo se basa en el efecto sombra que ejerce nuestra propia cabeza sobre los sonidos.

Las señales de alta frecuencia tienen une longitud de onda menor que el diámetro medio de una cabeza; esto produce que las señales no se difracten y se produzcan sombras acústicas; sin embargo, a bajas frecuencias las longitudes de onda son mayores, lo que permite que las señales se difracten fácilmente y no se produzcan sombras acústicas; razón por la cual el ILD solo funciona para frecuencias superiores a los 1000Hz.

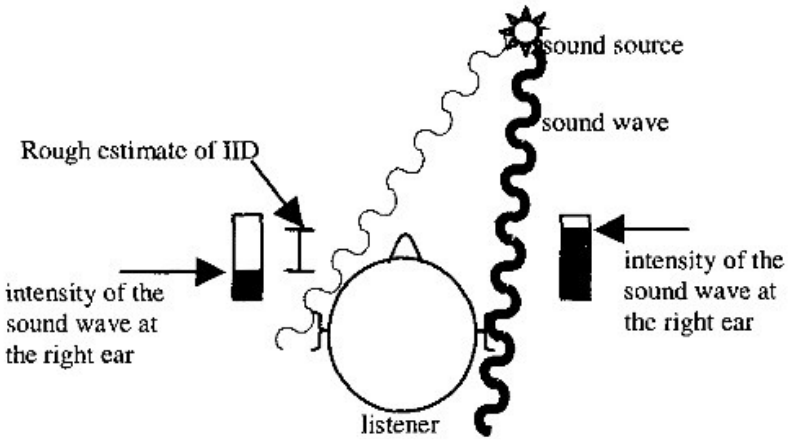

Figura 2.5 Representación diferencias interaurales de nivel

De la descripción de los sistemas interaurales de tiempo y nivel, podemos observar que ambos mecanismos se complementan. En las bajas frecuencias predominaran las diferencias interaurales de tiempo; mientras que en altas frecuencias lo harán las diferencias interaurales de nivel.

2.1.2 El cono de confusión

En el sistema de coordenadas esféricas se requieren tres variables para determinar un determinado punto del espacio: el azimut (θ), la elevación (φ) y el radio (r). Como se ha visto en el apartado anterior, los mecanismos ITD e ILD solo proporcionan un parámetro de los tres necesarios, el azimut (θ).

Para un determinado valor del azimut (θ), existen una infinidad de puntos con diferentes valores de elevación (φ) y radio (r). A este conjunto de puntos se les denomina **cono de confusión** (Figura 2.6); “cono” porque el espacio que ocupan tiene forma de cono, con el iodo de foco; “de confusión” por la ambigüedad que se produce a la hora de determinar la dirección de origen del sonido.

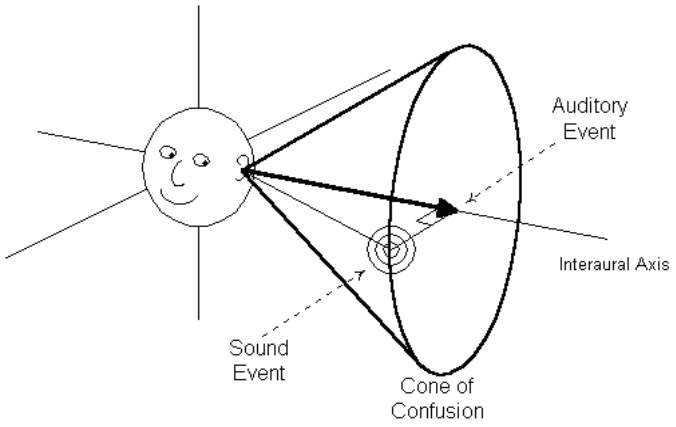

Figura 2.6 Cono de confusión

2.1.3 Determinar el ángulo de elevación y la distancia

A diferencia del ángulo lateral, la percepción de la elevación es monoaural, es decir que solo se requiere un oído para su percepción. Esta se basa en que nuestros oídos se comportan como un resonador acústico cuyas cavidades amplifican unas frecuencias y su geometría atenúan otras; los llamados indicios espectrales.

La forma de los hombros, la cabeza y el oído externo o pinna, actúan como filtros que modifican la fase y la amplitud de las ondas en función de la dirección de incidencia del sonido. En la figura 2.7 se observan ejemplos de dos medidas de la respuesta frecuencial de dos direcciones diferentes.

Por cada dirección de incidencia existen dos caminos: un camino directo y un camino más largo como consecuencia de la reflexión en el oído. A frecuencias relativamente bajas, las ondas sonoras de los respectivos caminos llegan en fase; sin embargo, en altas frecuencias, se produce un desfase entre ambas trayectorias que produce interferencias destructivas entre las ondas sonoras. La mayor interferencia se produce cuando la distancia que separa ambas trayectorias es igual a media longitud de onda.

Para valores típicos de la distancia entre trayectorias, las interferencias se producen en el rango entre 6 y 16KHz. Por eso se considera que, para una correcta detección de la elevación, es necesario que las señales contengan componentes frecuenciales entre 6 y 16KHz.

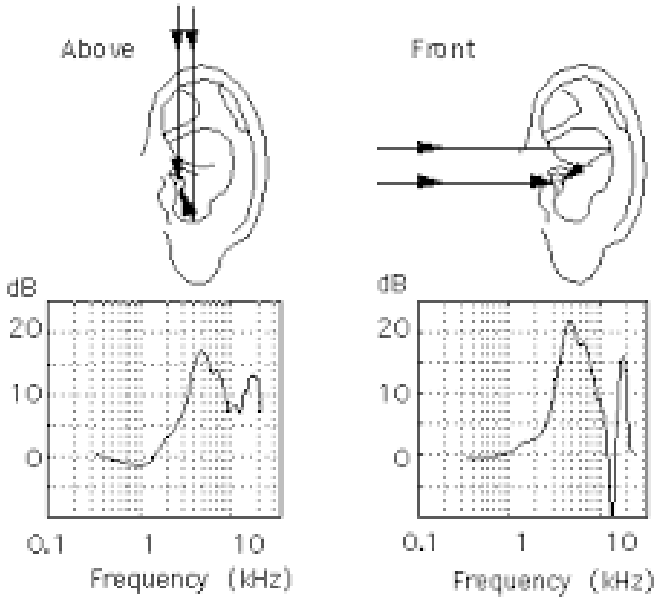

Figura 2.7 Respuesta frecuencial

El sistema auditivo humano no especialmente hábil determinando la distancia que separa un oyente de una fuente sonora. En la práctica, para simular el efecto de la distancia sobre las ondas sonoras basta con atenuar la señal dividiendo la amplitud entre la distancia a la fuente.

2.2 Head Related Transfer Fuction, HRTF

En el apartado anterior se han descrito las diferentes técnicas que utiliza el sistema auditivo humano para determinar la localización de las fuentes sonoras en el espacio. Todos estos mecanismos dependen de las características físicas de cada individuo, por lo que resulta bastante difícil establecer una relación matemáticamente entre la percepción espacial del sonido y la variación en sus componentes frecuenciales.

Una solución a este inconveniente consiste en realizar una serie de medidas directamente del oído de varios sujetos para obtener la variación del contenido espectral que depende de la dirección de incidencia. A este conjunto de medidas se las denomina "*Función de Transferencia Relativas a la Cabeza"*, HRTF.

Concretamente, la HRTF[1] se define como la respuesta en frecuencia en campo lejano de un oído especifico de cada individuo, medido desde un punto concreto del campo libre a un punto concreto del oído del individuo, normalmente para un radio fijado de distancia entre el oyente y la fuente.

[1] La definición de HTRF se ha obtenido de la tesis de master "Procesador de sonido y estudio de interpolación de la localización de fuentes basados en HRTFs para generación de sonido 3D" de la Universidad Politécnica de Valencia.

Las HRTF dependen de cuatro variables, tres coordenadas espaciales y una frecuencial. Normalmente las medidas se realizan a una distancia fija del campo lejano, reduciendo así el número de variables de la HRTF a tres: el azimut (θ), la elevación (φ) y la frecuencia (f).

La "*Respuesta al Impulso Relativa a la Cabeza"* HRIR, es el equivalente en el dominio del tiempo de la HRTF; ambas están relacionadas mediante la transformada de Fourier. La respuesta al impuso HRIR se define como la transformada inversa de Fourier de la respuesta en frecuencia HRTF. Ambas dependen del sistema hombros, cabeza y pinna de cada individuo. Dado que las características fisiológicas varían de un individuo a otro, tanto la HRIR, como la HRTF son únicas de cada sujeto.

Matemáticamente el sonido que perciben cada uno de los oído, "$x_{izq}(t)$" en el oído izquierdo y "$x_{der}(t)$" en el derecho, se expresa en el dominio del tiempo como la convolución entre el sonido emitido por la fuente $x(t)$, una respuesta al impulso conocida "$c(t)$"; principalmente producida por los propios equipos de medida, y la respuesta al impulso de cada oído; "$h_{izq,\theta,\varphi}(t)$" para oído izquierdo y "$h_{der,\theta,\varphi}(t)$" para el derecho (ver figura 2.8).

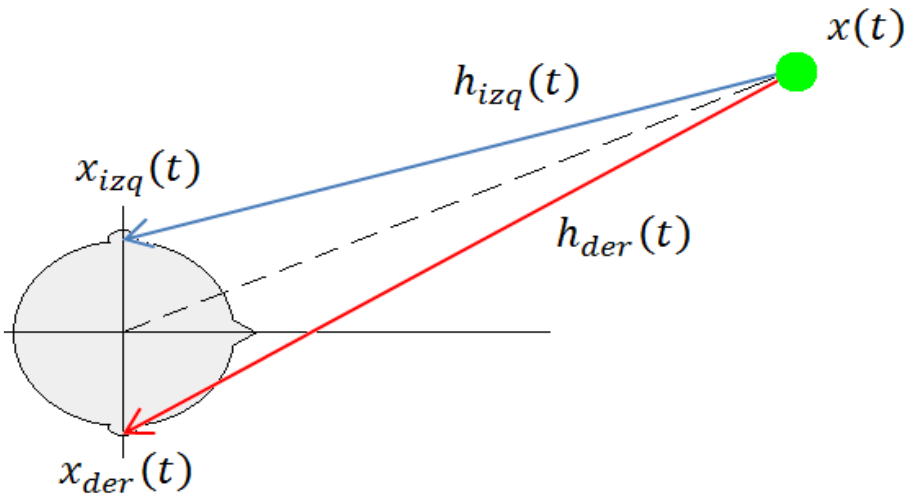

Figura 2.8 Trayectorias del sonido

$$x_{izq}(t) = x(t) * c(t) * h_{izq,\theta,\varphi}(t) \quad x_{der}(t) = x(t) * c(t) * h_{der,\theta,\varphi}(t) \qquad (1)$$

En el dominio de la frecuencia, las convoluciones se convierten en simples multiplicaciones.

$$X_{izq}(\omega) = X(\omega) \cdot C(\omega) H_{izq,\theta,\varphi}(\omega) \qquad X_{der}(\omega) = X(\omega) C(\omega) H_{der,\theta,\varphi}(\omega) \quad (2)$$

De la expresión anterior se despeja el módulo de la HRTF

$$\left|H_{izq,\theta,\varphi}(\omega)\right| = \left|\frac{X_{izq(\omega)}}{X(\omega)C(\omega)}\right| \qquad \left|H_{der,\theta,\varphi}(\omega)\right| = \left|\frac{X_{der(\omega)}}{X(\omega)C(\omega)}\right| \tag{3}$$

Y su fase

$$\angle H_{izq,\theta,\varphi}(\omega) = \angle X_{izq,\theta,\varphi}(\omega) - \angle X(\omega) - \angle C(\omega) \tag{4}$$

$$\angle H_{der,\theta,\varphi}(\omega) = \angle X_{der,\theta,\varphi}(\omega) - \angle X(\omega) - \angle C(\omega) \tag{5}$$

En la práctica las respuestas al impulso relacionado con la cabeza (HRIR) normalmente se representan como un conjunto de filtros FIR (Finite Impulse Response) donde las diferencias interaurales de tiempo (ITD) se codifican en la fase y las diferencias interaurales de nivel (ILD) en el módulo en función de cada frecuencia.

2.2.1 Medir la HRTF

En el apartado 2.2 se ha definido las HRTF como una serie de medidas para determinar la relación entre la variación espectral del sonido y su percepción espacial. Estas medidas se llevan a cabo en entornos muy silenciosos, idealmente salas anecoicas, donde no se produzcan reflexiones tempranas, reverberaciones o ecos que puedan afectar a las mediciones.

Existen dos formas de medir las HRTF:

1. **HRTF medido directamente en una persona (figura 2.9a).** Este método consiste en utilizar uno o varios voluntarios para medir el efecto que produce el sistema hombros, cabeza y pabellón auditivo en los sonidos.

   La ventaja de este método se basa en que los modelos obtenidos son mucho mejores que los que se obtendrían con cualquier otro método; su principal inconveniente radica en las diferencias fisiológicas entre el voluntario sobre el que se hicieron las medidas y el oyente final.

2. **HRTF medido en un dummy head (figura 2.9b).** Este proceso consiste en utilizar un maniquí dummy head con las características fisiológicas de un ser humano para modelar el efecto del cuerpo sobre el sonido.

Figura 2.9 Medida de la HRTF, a) en un voluntario, b) en un maniquí KEMAR

El proceso de medida consiste en generar secuencias LSS (Long Sine Sweep) o de ruido blanco, que posteriormente se capturan con un micrófono colocado en el punto donde irían las orejas en el caso de un dummy head, o en la entrada del canal auditivo si la medida se realiza en una persona.

2.2.2 Bases de datos HRTF

Medir las HRTF no es proceso especialmente sencillo y barato; existen un conjunto de bases de datos de libre distribución disponibles en internet.

- La **base de datos MIT KEMAR (Gardner & Martin, 1994)**. Formada por dos (para dos tamaños de oreja diferentes) series de medidas de la cabeza de un dummy KEMAR. Abarca 710 posiciones diferentes del espacio con valores de elevación que van de los -40 hasta los 90 grados, con una resolución máxima de azimut de 5 grados.
- LA **base de datos CIPIC**. Está formada por medidas de cuarenta y cinco sujetos diferentes. Posee veinticinco valores de azimuts diferentes y cincuenta elevaciones en incrementos de cinco grados.
- La **base de datos del IRCAM.** Esta base de datos está formada por de medidas tomadas en 51 sujetos diferentes. Por cada sujeto se realizaron 187 medidas, que consisten en 10 ángulos de elevación que van de -45 a +90 grados con incrementos de 15 grados. Los ángulos de acimut varían según el ángulo de elevación desde 24 muestras para una elevación de 0º, hasta una única muestra para elevaciones de 90º.

2.3 Filtro de señales digitales

Dada una señal discreta, x[n], y la respuesta al impulso de un filtro, h[n]; filtrar la señal x[n] con el filtro representado por h[n] se reduce a calcular la convolución discreta.

$$y[n] = x[n] * h[n] = \sum_{k=0}^{+\infty} x[k] \cdot h[n-k] \tag{6}$$

Si calculamos la transformada discreta de Fourier, DFT, de y[n], observamos que esta se puede expresar como la multiplicación de las transformadas discretas de x[n] y h[n]. Así pues, en el dominio de la frecuencia, filtrar una señal significa multiplicar el espectro de dicha señal, X[k], con la función de trasferencia del filtro, H[k].

$$Y[K] = X[K] \cdot H[K] \tag{7}$$

2.3.1 Filtrar señales en el tiempo

Como ya se ha mencionado, en el dominio del tiempo, filtrar una señal consiste en calcular la suma de convolución.

$$y[n] = \sum_{k=0}^{+\infty} x[n] \cdot h[n-k] \tag{8}$$

La convolución también se puede expresar mediante *ecuaciones lineales en diferencias*. Estas ecuaciones expresan la salida de un filtro, y[n], como la combinación de sumas, restas y multiplicaciones de muestras actuales y anteriores tanto de la señal de entrada, x[n], como de la salida, y[n].

$$y[n] = \sum_{k=0}^{Q} b_k \cdot x[n-k] - \sum_{k=0}^{P} a_k \cdot y[n-k] \tag{9}$$

En la ecuación [9], los variables b_k y a_k son los coeficientes del filtro.

Los **filtros FIR (Finite Impuse Response)** son filtros en los que la salida se obtiene como la combinan muestras actuales de la señal de entrada con muestras pasadas.

$$y[n] = \sum_{k=0}^{Q} b_k \cdot x[n-k] \tag{10}$$

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] + \cdots + b_Q \cdot x[n-Q] \tag{11}$$

En cambio, los **filtros IIR (Infinite Impulse Response)**, presentan cierta recursividad, es decir que la salida se obtiene como la combinación tanto de señales de entrada como de salida.

$$y[n] = \sum_{k=0}^{Q} b_k \cdot x[n-k] - \sum_{k=0}^{P} a_k \cdot y[n-k] \tag{12}$$

$$\begin{aligned} y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] + \cdots + b_Q \cdot x[n-Q] - a_0 \cdot y[n] \\ - a_1 \cdot y[n-1] - a_2 \cdot y[n-2] \cdots - a_p \cdot y[n-P] \end{aligned} \tag{13}$$

2.3.2 Filtrar Señales en frecuencia

En el dominio de la frecuencia la señal filtrada se obtiene como:

$$Y[K] = X[K] \cdot H[K] \tag{14}$$

Entonces y[n] vale

$$y[n] = DFT^{-1}(Y[K] = X[K] \cdot H[K]) \tag{15}$$

La Transformada discreta de Fourier de una secuencia discreta, x[n], de longitud N, se define como:

$$X[K] = \sum_{n=0}^{N-1} x[n] \cdot e^{-\frac{2\pi}{N}kn} \quad k = 0, 1, 2, \dots, N-1 \tag{16}$$

Y la trasformada inversa como:

$$x[n] = \frac{1}{N} \sum_{K=0}^{N-1} X[K] \cdot e^{\frac{2\pi}{N}kn} \quad k = 0, 1, 2, \dots, N-1 \tag{17}$$

Gracias a los algoritmos FFT (Fast Fourier Transform) el cálculo de la DFT no requiere mucha carga computacional.

Dadas dos secuencias finitas x[n] y h[n] de longitudes L y P respectivamente, la convolución lineal entre ambas secuencias utilizando DFT se obtiene de la siguiente manera (figura 2.10):

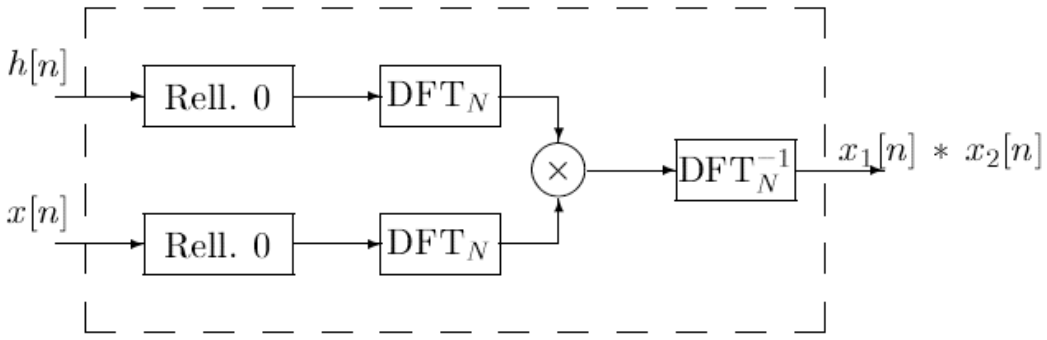

Figura 2.10 Calculo de la convolución lineal usando DFT

1. Rellenar con ceros las señales x[n] y h[n] hasta que su longitud, N, valga N≥ L + P -1.
2. Obtener la DFT de N puntos de ambas señales

$$X[K] = DFT(x[n])$$
$$H[K] = DFT(h[n]) \tag{18}$$

3. Multiplicar las DSFs de x[n] y h[n],.

$$Y[K] = X[K] \cdot H[K], \qquad con\ K = 0, 1, 2, \dots, N-1 \tag{19}$$

4. Calcular la transformada inversa de Y[K]

$$y[n] = DSF^{-1}(Y[K]) \tag{20}$$

2.3.3 Convolución por bloques

Los dos métodos de filtro de señales digitales expuestos anteriormente funcionan correctamente con señales de corta duración. Si se desea filtrar señales de longitud infinita o desconocida la mejor opción es recurrir a los métodos de convolución por bloques: El **método de solapamiento-suma (figura 2.11)** y el **método de solapamiento-almacenamiento (figura 2.12)**.

Partiendo de una señal de longitud infinita o desconocida x[n] y un filtro de longitud finita P, la convolución **mediante el método de solapamiento suma** se obtiene de esta forma:

1. Dividir x[n] en bloques,$x_{k[n]}$, de longitud L sin solapamiento ni huecos.

$$x[n] = \sum_{k=0}^{\infty} x_k[n] \tag{21}$$

2. Calcular la convolución lineal de N muestras de cada bloque de forma independiente $y_{k[n]}$, con N ≥ L + P – 1.
3. La salida total se calcula como:

$$y[n] = \sum_{k=0}^{\infty} y_k[n] \tag{22}$$

Se produce un solapamiento al recomponer la señal final debido a que los bloques $y_{k[n]}$ tienen una duración mayor que los bloques$x_{k[n]}$. Concretamente se suman las P -1 últimas muestras de un bloque con las P – 1 primeras muestras del bloque siguiente.

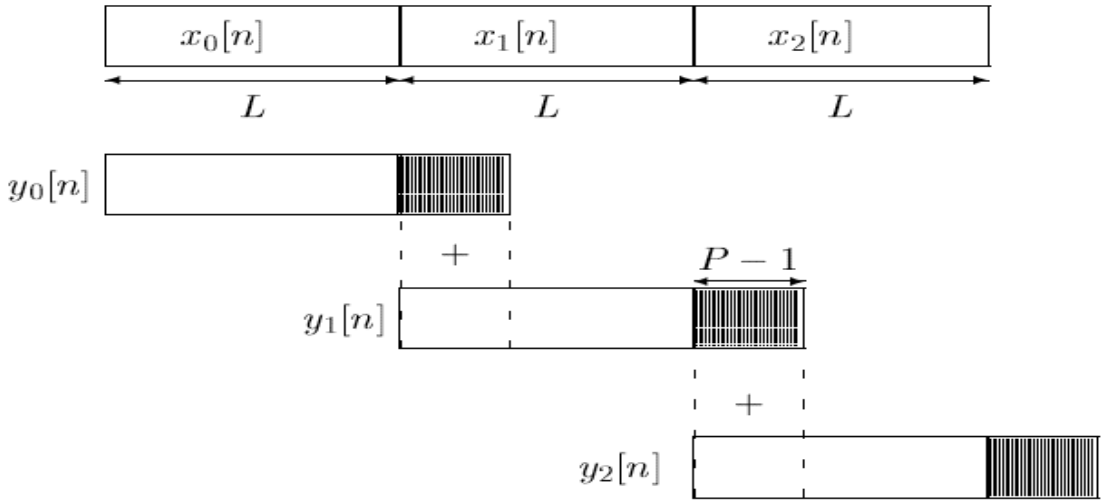

Figura 2.11 Método de solapamiento suma

Partiendo con señales similares a las del método anterior, la **convolución mediante el método de solapamiento almacenamiento** se obtiene de esta forma:

1. X[n] se divide en segmentos $x_{k[n]}$ de L (con L > P) muestras solapadas P -1 muestras.

$$x_{k[n]} = x[n + k(L - P + 1) - P + 1] \quad con\ n = 0, 1, \dots, L - 1 \tag{23}$$

2. Calcular la convolución circular de N muestras de cada segmento con el filtro h[n].

$$y_{k[n]} = x_k[n] \textcircled{N} h[n] \tag{24}$$

3. Eliminar las P – 1 primeras muestra de cada segmento $y_{k[n]}$
4. La salida total se obtiene como:

$$y[n] = \sum_k y_k[n + k(L - P + 1) - P + 1] \tag{25}$$

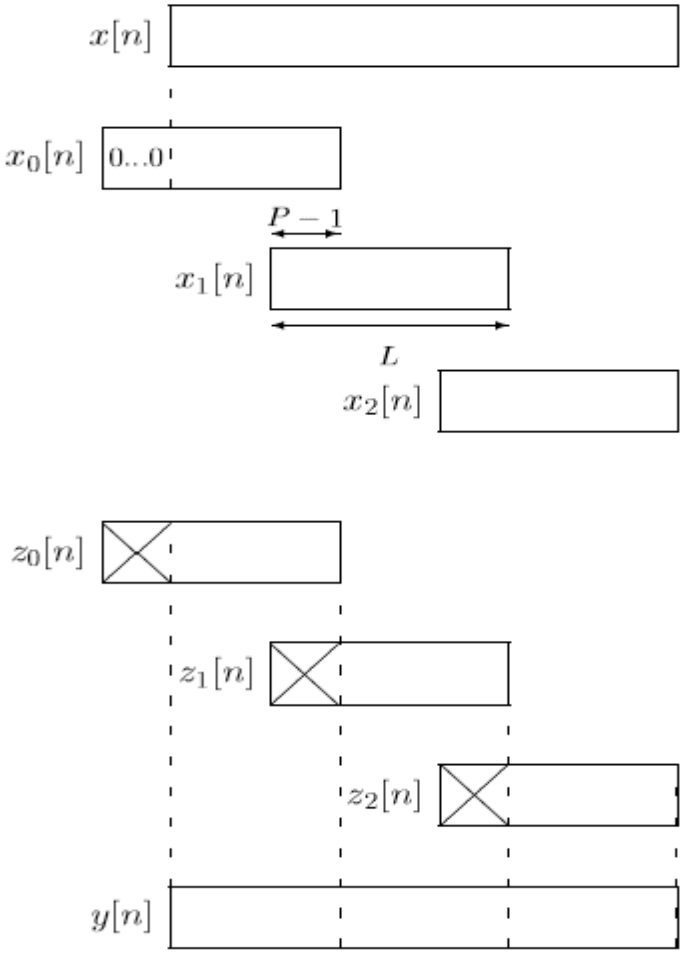

Figura 2.12 Método de solapamiento almacenamiento

3 OBJETIVOS

El objetivo general de este proyecto consiste en realizar una aplicación para tableta que, a partir de un sonido monoaural y los puntos “x” e “y” de una de una determinada trayectoria, sea capaz de añadir características espaciales al sonido que simulen el desplazamiento de una fuente sobre dicha trayectoria.

Como objetivos específicos podemos destacar:

- Conocer los mecanismos y métodos de los que hace uso el sistema auditivo humano para la localización de focos sonoros.
- Conocer y entender las HTRF.
- La síntesis de sonido binaural mediante filtros HRTF.
- Familiarizarse con el lenguaje de programación C y la herramienta de programación OpenFrameworks.
- Desarrollar una librería con la que poder sintetizar sonido binaural a partir de un sonido monoaural utilizando el lenguaje de programación C.

Para cumplir todos estos objetivos necesitaremos:

- Un PC u **ordenador portátil,** para las tareas de procesado y programación.
- Un banco de filtros HRTF, conjunto de filtros FIR con los que se añadirá las características espaciales al sonido.
- Unos **Auriculares,** para reproducir los resultados.
- Un **micrófono,** necesario para capturar audio.

4 MATERIALES Y MÉTODOS

Como su nombre indica, en este apartado explicaremos los métodos y las herramientas utilizadas para desarrollar la aplicación de auralización. Primero comenzaremos describiendo brevemente las herramientas utilizadas; a continuación se explicaran los métodos utilizados para calcular los parámetros angulares y la atenuación. Posteriormente se presentara la herramienta de programación ***OpenFrameworks*** sobre la que se ha realizado la aplicación, y finalmente se describirá detalladamente la aplicación desarrollada.

4.1 Hardware

En este apartado se realiza una breve descripción tanto de las herramientas como del hardware necesario para desarrollar y ejecutar la aplicación.

4.1.1 PC

Puede considerase un elemento importante en el desarrollo de este proyecto. Sus funciones son:

- Realizar la conversión analógico-digital o digital-analógica de señales.
- Procesar las señales (segmentar, filtrar, recomponer, reproducir, grabar…etc.)
- Compilar y ejecutar el código desarrollado.

4.1.2 Banco de Filtros HRFT

Para el proceso de síntesis de señales binaurales a partir de señales monoaurales, se requiere un banco de filtros que almacenen las características espaciales de diferentes posiciones del espacio.

Durante el desarrollo de la aplicación se ha utilizado un banco de filtros HRTF con setenta y dos filtros FIR diferentes. Cada filtro FIR representa una determinada posición de la fuente en el plano azimutal, es decir que con este banco de filtros solo se puede simular sonido binaural con un ángulo de elevación que siempre vale 0º ($\varphi = 0º$); y un azimut (θ) que va de los 0º hasta los 355º con una resolución de 5º (figura 4.1).

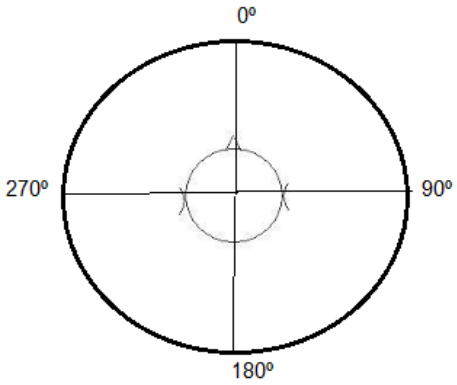

Figura 4.1 Representación plano Horizontal

4.1.3 Micrófono

Los micrófonos son transductores electroacústicos que convierten las señales acústicas en señales eléctricas para posteriormente poder almacenarlas o procesarlas. En este proyecto en concreto, el micrófono se usará para grabar la señal monoaural sobre la que posteriormente se aplicará los filtros HRTF.

4.1.4 Auriculares

En contraposición a los micrófonos, los auriculares son transductores que convierten las señales eléctricas en señales acústicas. En los sistemas de síntesis de sonido binaural mediante filtros HRTF los auriculares son el mejor soporte de escucha existente.

4.2 Síntesis de sonido espacial con HRTF, Auralización

Hasta ahora solo se ha tratado sobre los métodos que existen para caracterizar la información espacial de los sonidos. La **auralización** es el proceso por el cual se simula la escucha espacial de un recinto de forma virtual. En esta aplicación, este efecto se consigue aplicando las características espaciales de los filtros HRTF sobre grabaciones de audio mono.

Aunque existe una gran variedad de sistemas dedicados a sintetizar sonidos binaurales mediante filtros HRTF (ver esquema en la figura 4.2), el principio de funcionamiento de la mayoría es el siguiente:

- Se parte de una fuente de sonido monoaural, generalmente una grabación de audio mono.
- Se determina la posición del espacio donde se quiere situar dicha fuente respecto al oyente.

- Se procesa la señal de audio con el filtro HRTF correspondiente a esa dirección de procedencia, convolucionando la señal de entrada con el HRIR correspondiente para cada oído.
- El efecto de la distancia sobre el sonido se simula dividiendo la señal entre la distancia de la fuente al oyente, se supone que el oyente se encuentra en campo lejano respeto a la fuente.

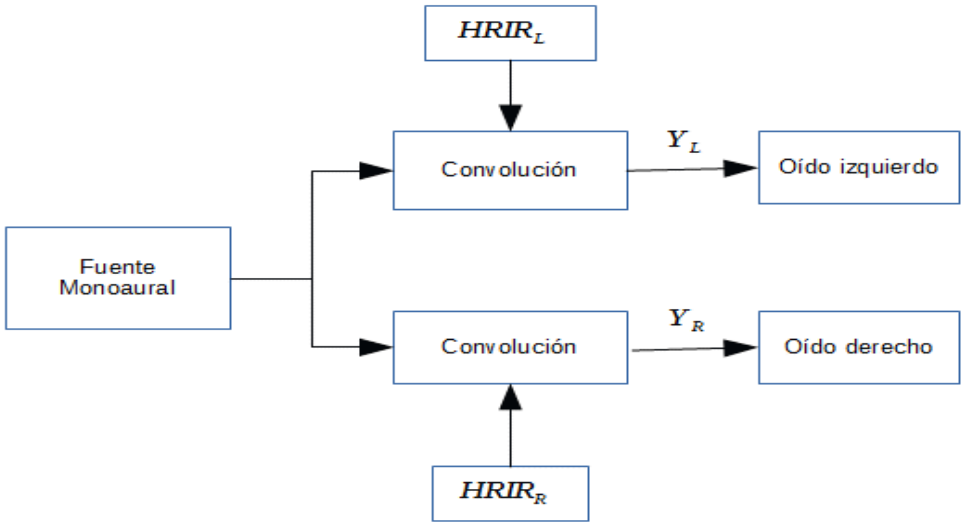

Figura 4.2 Esquema general de sistemas de síntesis de audio binaural mediante HRTF

Se puede observar que el principio de funcionamiento expuesto anteriormente solo tiene en cuenta una única posición del espacio. Ahora bien, para simular una fuente en movimiento tendremos que dividir la señal de audio en pequeños fragmentos o tramas. A cada una de las tramas se le aplicará un filtro que simule una posición diferente del espacio en función de la trayectoria que se desea recrear.

Simular el sonido de un foco sonoro en movimiento aplicando filtros diferentes en la señal de audio presenta un inconveniente, la aparición de un sonido adicional, un pequeño *"click"*, como consecuencia de las discontinuidades que se producen por los cambios de filtros durante toda la señal de audio. La manera más sencilla de eliminar este problema es mediante el enventanado. El proceso consiste en, antes de calcular la convolución, multiplicar cada trama de audio con una ventana; en esta aplicación en concreto se ha usado la ventana Hanning. En la figura 4.3 se puede ver la diferencia entre antes y después de aplicar el enventanado.

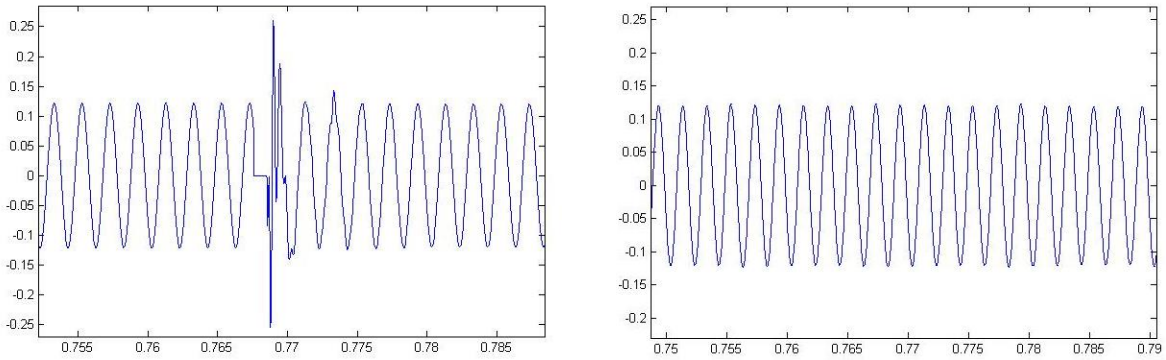

Figura 4.3 Señal filtrada a) sin enventanar, b) después con enventanado

4.3 Cálculo del ángulo lateral y la atenuación

Antes de procesar las señales, primero se debe indicar qué trayecto recorrerá el foco virtual que se desea simular. En la aplicación, estos trayectos se indicaran como una serie de puntos sobre los que deberá desplazarse el foco virtual. En la figura 4.5 se puede ver un ejemplo del recorrido de una fuente; en este caso consta de tres puntos A, B, y C.

A las rectas que forman el conjunto de puntos que constituyen el recorrido virtual de la fuente se denominan trayectorias. Así pues, el trayecto de la figura 4.5 está formado por dos trayectorias, las rectas $\overline{AB}$ y $\overline{BC}$.

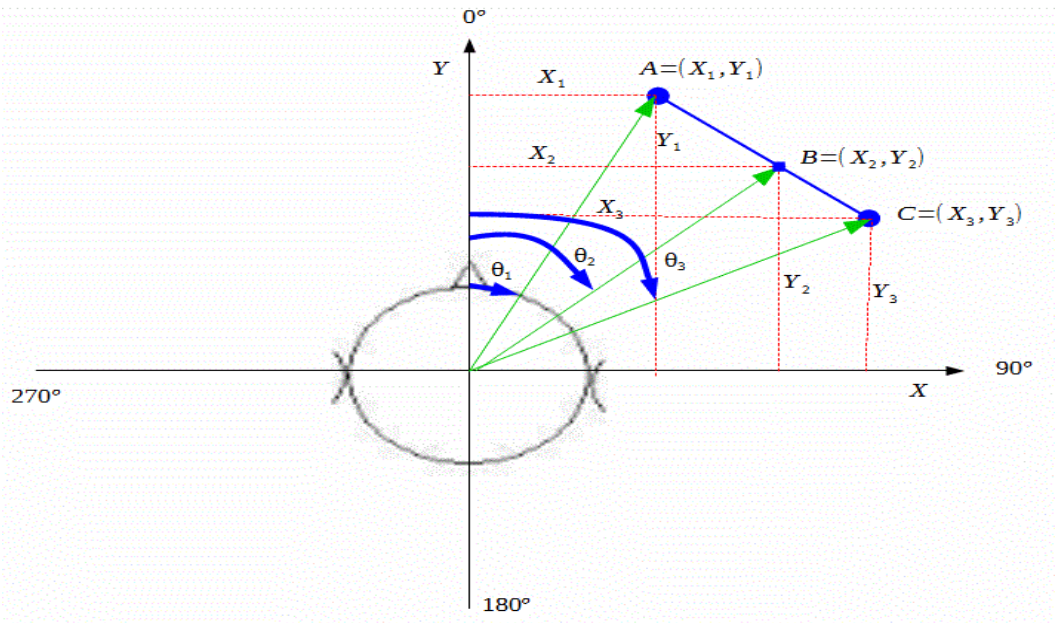

Figura 4.4 Representación de trayectorias del sonido

4.3.1 Interpolación

Simular el movimiento de una fuente sobre una trayectoria requiere más de dos puntos, por ello se recure a la interpolación para calcular los puntos intermedios entre el punto inicial y la final de cada una de las trayectorias.

Dado el punto inicial y final de una recta, $A = (X_1, Y_1)$ y $B = (X_2, Y_2)$ respectivamente (ver figura 4.5), los puntos intermedios entre ambos se calculan mediante el método de interpolación lineal:

- Primero calcular la pendiente de la recta

$$m = \frac{Y_2 - Y_1}{X_2 - X_1} \tag{26}$$

- A continuación calcular las coordenadas "X_i" de cada punto con la ecuación siguiente:

$$X_i = \begin{cases} X_1 + \left(\frac{X_2 - X_1}{N-1}\right) & si\ X_1 < X_2 \\ X_1 & si\ X_1 = X_2 \\ X_1 - \left(\frac{X_1 - X_2}{N-1}\right) & si\ X_1 > X_2 \end{cases} \qquad \text{Con i = 1,2,…N-1} \tag{27}$$

En esta ecuación "N" es el número de puntos que se desea interpolar entre los puntos A y B. A mayor valor de "N", más filtros HRTF se utilizan en una única trayectoria, por lo se obtienen mejores resultado.

- Los pesos de la trayectoria, Y_i, se calculan mediante la ecuación de la recta

$$Y_i = Y_1 + m \cdot X_i - X_1 \qquad \text{Con i = 1,2,…N-1} \tag{28}$$

Cuando X_1 es igual a X_2, la pendiente de la recta vale $m = \infty$, en ese caso los valores de las coordenadas Y_i se calculan mediante la ecuación

$$Y_i = \begin{cases} Y_1 + \left(\frac{Y_2 - Y_1}{N-1}\right) & si\ Y_1 < Y_2 \\ Y_1 & si\ Y_1 = Y_2 \\ Y_1 - \left(\frac{Y_1 - Y_2}{N-1}\right) & si\ Y_1 > Y_2 \end{cases} \qquad \text{Con i = 1,2,…N-1} \tag{29}$$

4.3.2 *Determinar el valor del azimut (ϑ)*

Como ya se mencionó anteriormente, la base de datos HRTF que ha utilizado en el desarrollo de la aplicación, consta de 72 medidas de diferentes posiciones del plano horizontal, es decir 72 ángulos de azimut diferentes.

Si nos fijamos en la figura 4.5, el azimut de cada posición se determina como:

$$\theta = \tan^{-1}\left(\frac{X_i}{Y_i}\right) \quad con\ i = 0,1,\cdots N-1 \tag{30}$$

4.3.3 *Calcular la atenuación*

La distancia que separa la fuente virtual del oyente, se modela como la atenuación de la señal. Según la figura 4.5, esta se obtiene como la hipotenusa de un triángulo.

$$Atn = \sqrt{{X_i}^2 + {Y_i}^2} \quad con\ i = 0,1 \ldots, N-1 \tag{31}$$

4.4 OpenFrameworks

OpenFrameworks es una herramienta de programación de código abierto sobre el lenguaje C/C++. Lanzada inicialmente en 2005 por Zachary Lieberman; actualmente la mantienen Zachary Lieberman, Theodore Watson, y Arturo Castro con la ayuda de la comunidad OpenFrameworks.

El principal objetivo de OpenFrameworks es proporcionar a los usuarios un acceso fácil a la multimedia, la visión artificial, redes y otras capacidades en C ++ agrupando en un solo paquete las librerías y bibliotecas más comunes tales como:

- OpenGL, GLEW, GLUT, libtess2 y Cairo para graficas
- POrtAudio, OpenAL, KissFFT o FMOD para operar con sonido
- FReeImage, para guardar y cargar imágenes
- Quicktime, Gstreamer y videoInput para reproducir y grabar videos
- Assimp, para el modelado 3D

OpenFrameworks es multiplataforma y compatible con los sistemas operativos Windows, Mac, Linux, iOS y Android. Esto significa que si usted desarrolla un proyecto para una de las plataformas, puede copiar los archivos de código fuente y compilar el proyecto para cualquier otra plataforma de la lista.

OpenFrameworks es especialmente apropiado en:

- El desarrollo de aplicaciones multimedia que trabajan en tiempo real
- Proyectos que llevan a cabo el análisis de grandes volúmenes de datos, por ejemplo, el análisis de datos de cámaras de profundidad.

En cambio, no es recomendable utilizar OpenFrameworks en proyectos que se los controles visuales como botones, casillas de verificación, listas y controles deslizantes. En este caso, la mejor opción es en el uso de plataformas de desarrollo como QT, cacao, o .Net.

4.4.1 Estructura de un proyecto OpenFrameworks

4.4.1.1 Estructura de archivos de un proyecto

Al crear un proyecto OpenFrameworks se generan una serie de carpetas. De este conjunto de carpetas, a efectos de implementación de código, las más importantes son:

- La “bin”, en ella se encontraran los archivos ejecutables de todo lo que se ha implementado. En él también se encuentra otra carpeta, “data”, donde se debe colocar todos los archivos, fotos, videos, figuras o cualquier tipo archivo externo con el que se va a trabajar.
- Otra carpeta importante es la carpeta “src”, contiene todo el código fuente del proyecto.

4.4.1.2 Estructura del código de un proyecto

Todo el código (clases, archivos de cabecera…etc.) que se genera en un proyecto OpenFrameworks se almacena en la carpeta “src”. Como mínimo esta carpeta siempre contendrá tres archivos: main.cpp, testApp.cpp y testApp.h.

El archivo main.cpp se encarga de iniciar la aplicación, solo posee un par de lineas de código que generalmente no es necesario retocar ni modificar:

El archivo testApp.h es el archivo de cabecera del testApp.cpp, en el se declaran todas las funciones y variables globales que posteriormente se definiran en el testApp.cpp. Por defecto en estos archivos se declaran una serie de funciones necesarias para ejecutar las aplicaciones, las más importantes son:

- Setup(), funcion donde se inicializan las variables globales
- Update(), en esta función se colocan todas las variables o funciones que se actualizan constantemente. Esta funcion es importante en el desarrollo de aplicaciones de tiempo real.
- Draw(), es la plicacion donde se colocaran todas aquellas instruciones cuya finalidad es representar o dibujar objetos, mostras imágenes.

Estas tres funciones se ejecutan de forma ciclica siguiendo el esquema de la figura 4.6. El resto de funciones por predefinidas, como por ejemplo keyPressed(int key), se ejecutan cuando ocurre un detrminado evento, en este caso cuando se pulsa una tecla.

Figura 4.5 orden de ejecución funciones OpenFrameworks

4.5 Desarrollo de la aplicación

El desarrollo de la aplicación se ha dividido en dos fases:

- La primera fase desarrollada en el lenguaje de programación Matlab; El objetivo era obtener una primera aproximación sobre del funcionamiento de los sistemas de síntesis de sonido binaural mediante filtros HRFT.
- La segunda fase desarrollada sobre el lenguaje de programación C, básicamente consiste en replicar los resultados obtenidos en la primera fase.

El funcionamiento de la aplicación se puede resumir en los tres puntos siguientes

1. Primero capturar sonido monoaural mediante la función **audioIn()** y almacenarla en dos vectores left y rigth, estos vectores harán de canal izquierdo y derecho respectivamente.
2. Procesar el sonido capturado con las funciones de la librería auralizacion.cpp, que como su nombre indica se ha desarrollado para auralizar sonidos.
3. Reproducir el resultado obtenido mediante la función **audioOut().**

4.5.1 Diseño de la Interfaz gráfica

Para implementar la interfaz gráfica se ha hecho uso de las funciones predefinidas de OpenFrameworks draw(). En la figura 4.7 se muestra un diagrama de flujo simplificado sobre su funcionamiento, el código se puede ver en el Anexo II: Documentacion de ofApp.

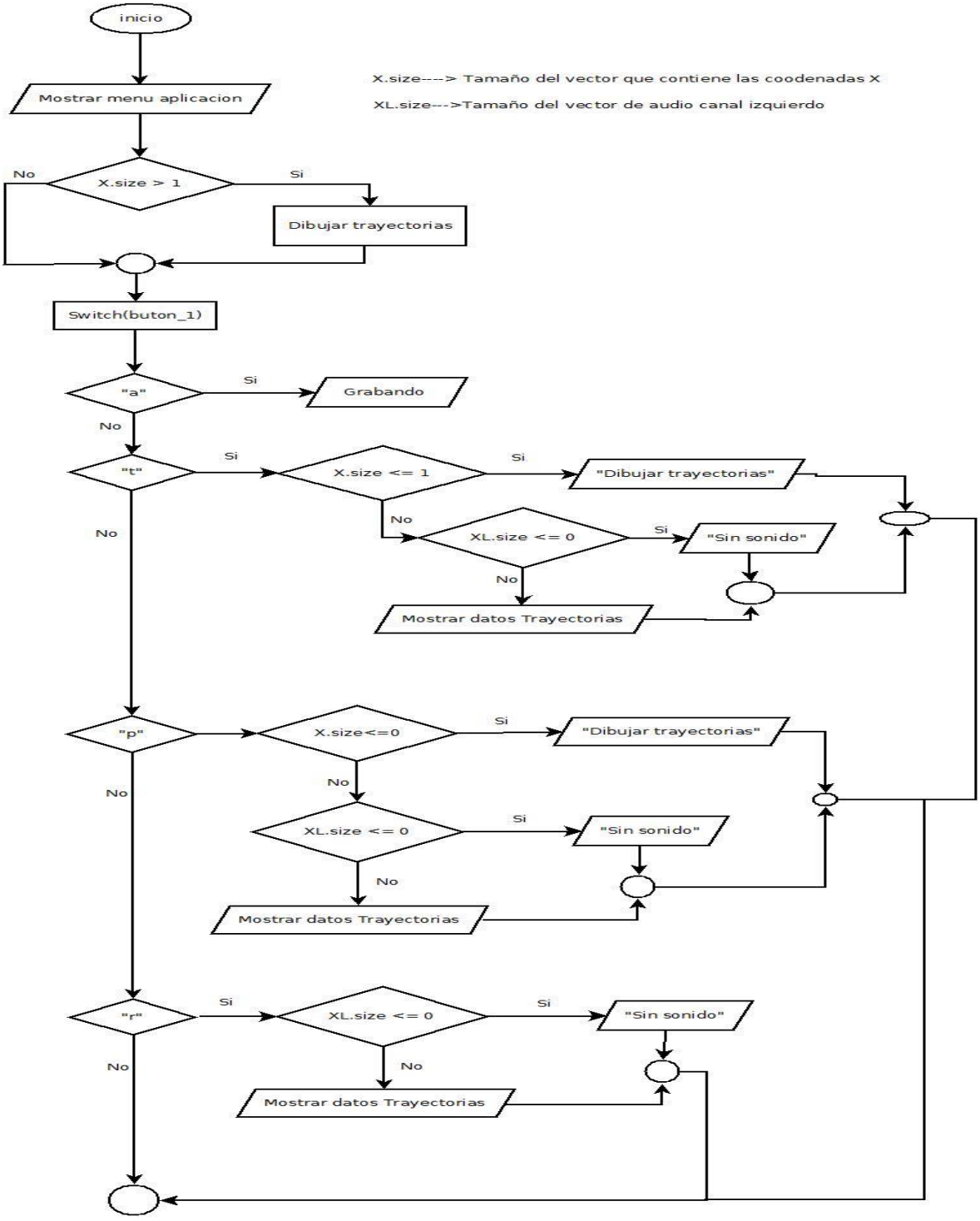

Figura 4.6 Diagrama de flujo interfaz grafica

4.5.2 Funciones de captura y reproducción de audio

Para trabajar con señales de audio se ha usado la clase ofSoundStream de OpenFrameworks. Esta clase actúa como el puente entre el programador y el hardware de audio del PC.

Atributos de la clase ofSoundStream

- **SampleRate**, frecuencia de muestreo del sonido.
- **bufferSize**, el tamaño del buffer de audio, sus valores típicos son 256 o 512.
- **nBuffers,** es el número de buffers que el sistema crea y posteriormente intercambia continuamente.
- **nOutputs**, número de canales de la señal de salida.
- **nInputs**, número de canales de la señal de entrada

Los métodos de la clase ofSoundStream

- **Setup()**, funciona como un constructor de la clase.
- **Start()**, inicia el proceso de captura o de reproducción del sonido.
- **Stop(),** detiene los procesos de captura y reproducción de sonidos.

La función que se encargara de capturar sonido es **audioIn();** sus parámetros de entrada son: el número de canales de entrada, **nInputs**; el tamaño del buffer, **nBuffers** y un puntero al buffer que contiene las muestras del sonido **inputs**. En la figura 4.8a se muestra el diagrama de flujo de esta función.

La función **audioOut()** se encargara de reproducir el sonido obtenido. Como parámetros de entrada tiene: el número de canales de salida, **noutputs**; el tamaño del buffer, **nBuffers** y un puntero al buffer de salida, **output**. En la figura 4.8b se muestra el diagrama de flujo de esta función.

Ejemplo de uso:

1. Crear un objeto de la clase ofSoundStream e inicializarla.

```
ofSoundStream sonido;
sonido.setup(this, 2, 1, 44100, 256, 4);
```

En este caso se ha creado un objeto que trabajara con sonidos con

- Frecuencia de muestreo de 44100Hz
- Un canal de entrada y dos de salida
- Cuatro buffers con una capacidad de 256 muestras cada uno.

2. Una vez inicializado el objeto, usar los métodos start(), para iniciar el stream de datos, o stop(), para detenerlo.

sonido.start();

sonido.stop();

La sentencia **sonido.start()** inicia la llamada a las funciones audioIn() y audioOut() de forma simultánea, eso no es conveniente, por ello es necesario implementar un mecanismo que alterne la llamada de uno u otro. En el *Anexo II: código del ofApp,* se puede ver la implementación de las funciones **audioIn(), audioOut()** y el uso de la clase ofSoundStream para manejar los sonidos que se producen en la aplicación .

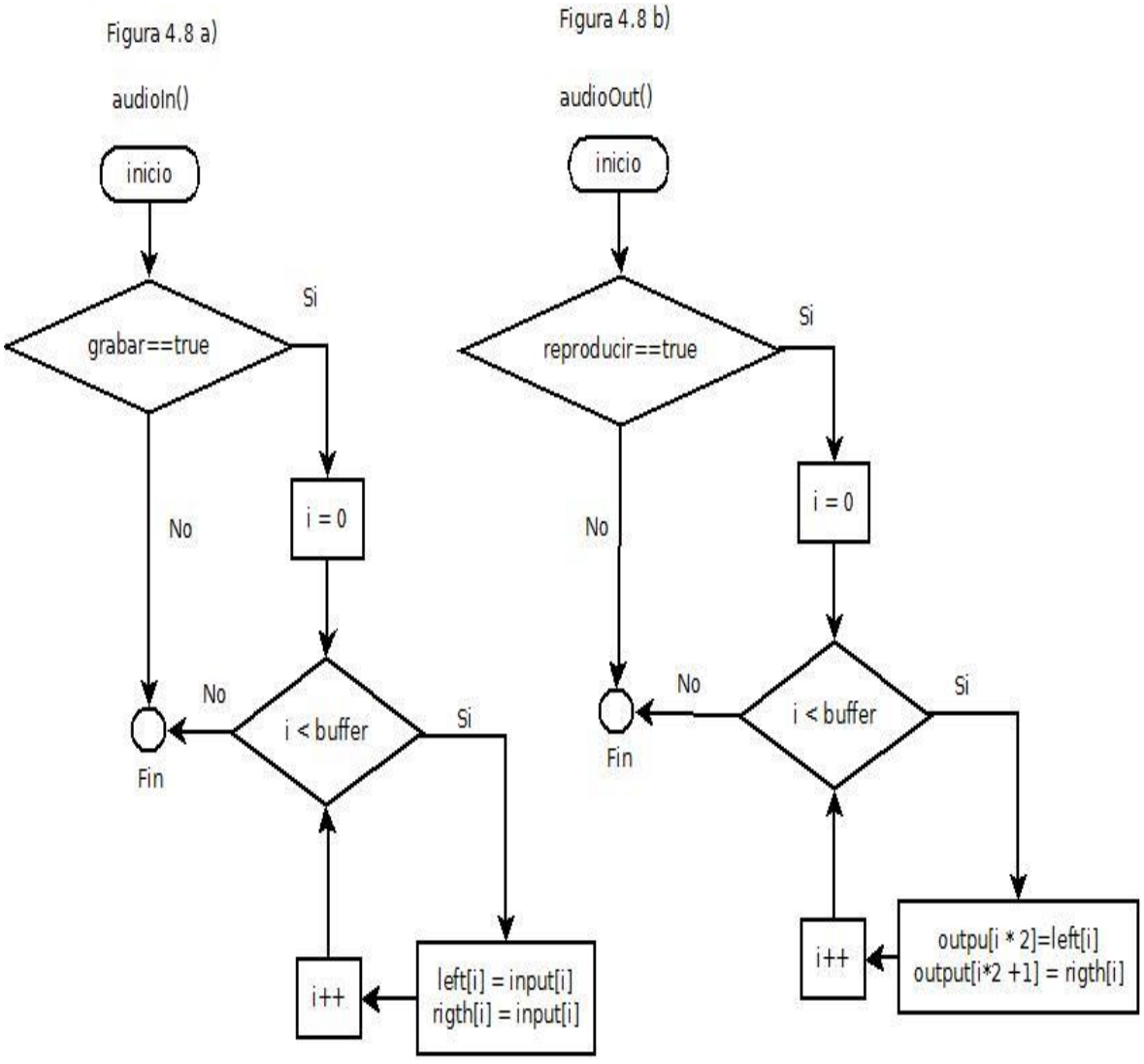

Figura 4.7 Diagrama de flujo a) audioIn(), b) audioOut()

4.5.3 Librería auralizacion.cpp

La librería auralizacion.cpp es el eje principal de la aplicación. Su función procesar las señales de audio para conseguir el efecto de la auralización. Su implementación se puede ver en el Anexo II: Documentacion de la Liberia auralizacion.pp, en él se definen de las siguientes funciones:

- interX() e interY()
- atenuaDistancia()
- angulos()
- auralizacion()
- setup_parameter()
- filter()
- Start()

Las funciones ***interX()*** e ***interY()*** calculan la interpolación de cada trayectoria siguiendo el método que se definió en el aparatado 4.3.1. Como resultada se obtienen una serie de puntos pertenecientes a la misma trayectoria. Estos puntos se almacenan en los vectores ***vector_X*** y ***vector_Y***, para las coordenadas x e y respectivamente.

La función ***atenuaDistancia()*** calcula la distancia entre el origen de coordenadas y cada uno de los puntos de los vectores ***vector_X*** y ***vector_Y*** mediante la fórmula [31] definida en el apartado 4.3.3. Los valores obtenidos se almacenan en vector, ***vector_atenuadistancia***.

El método ***angulos()*** obtiene, a partir de los vectores de coordenadas **vector_X** y **vector_Y**, el ángulo de cada posición mediante la fórmula [30] del apartado 4.3.2. Cada uno de los valores del azimut (θ) obtenidos por ***angulo()*** se aproxima al valor de azimut del banco de filtros HRTF más próximos a él. Estos angulos se almacenan en un vector ***vector_angulos***.

El método ***auralizacion()*** es la función principal de la librería. Recibe como parámetros de entrada: los vectores que contienen las señales de audio **Raudio y Laudio** correspondientes al canal derecho e izquierdo respectivamente; los vectores con las coordenadas X e Y; un vector con el tiempo de duración de cada trayectoria, y la frecuencia de muestreo. En la imagen 4.8 se muestra el diagrama de flujo de su implementación.

Auralizacion() se encarga de:

- Establecer el orden de llamada a las demás funciones de la librerías
- Dividir el trayecto a simular en un conjunto de trayectorias
- Dividir la señal de audio en segmentos en función de la duración de cada trayectoria.

- Calcular los ángulos y la atenuación que se aplicaran a cada trayectoria
- Reensamblar la señal a partir de los resultados obtenidos de las diferentes trayectorias.

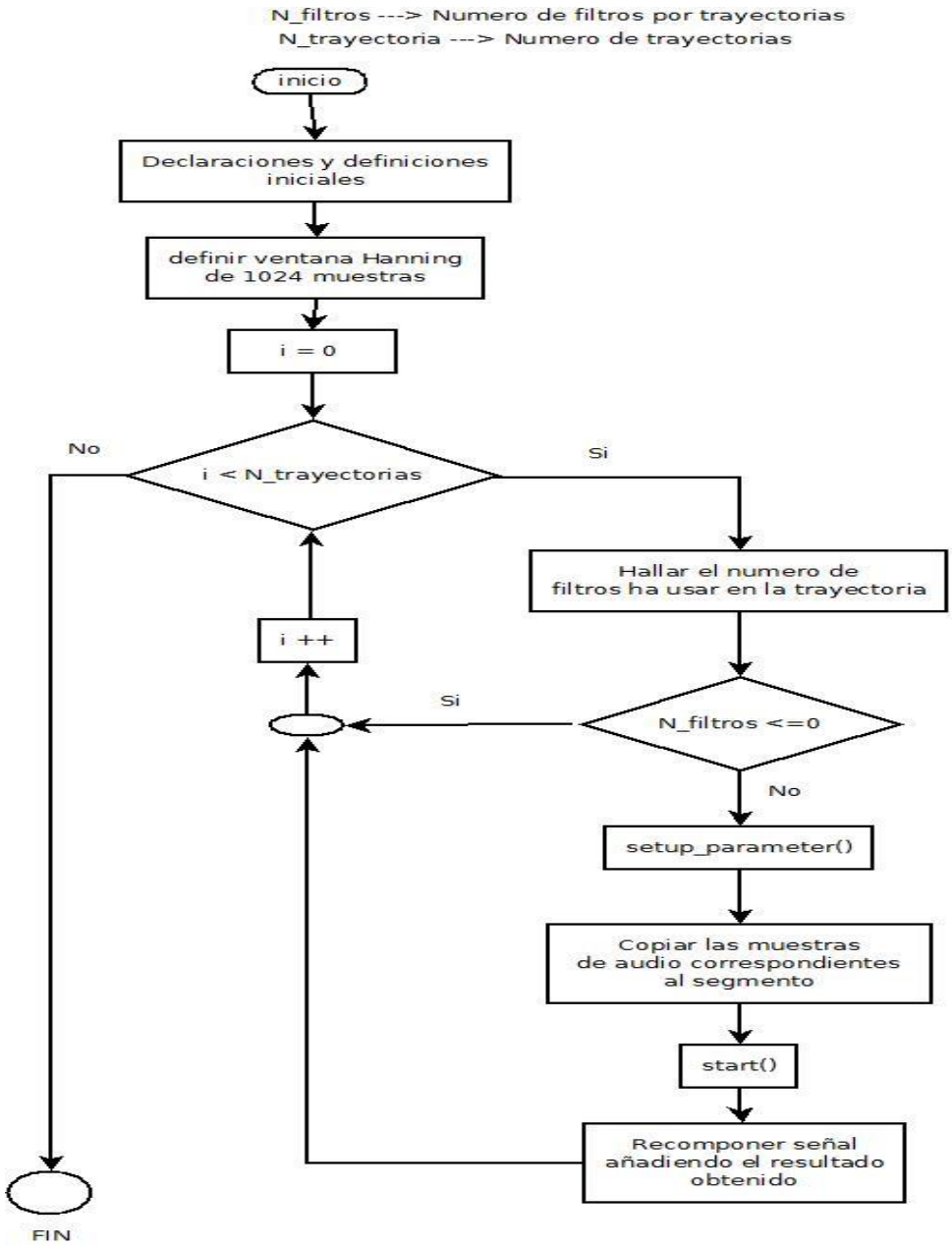

Figura 4.8 Diagrama de flujo de la función auralizacio()

La función ***Start()*** se encarga de procesar las muestras de señales de cada trayectoria. Sus parámetros de entrada son: los vectores que contienen las señales de audio de la trayectoria, un vector con los angulos de la trayectoria, el vector con las atenuaciones y la ventana Hanning de 1024 muestras. En la figura 4.8 se muestra el diagrama de flujo de la función ***start().***

En concreto esta función se encarga de:

- Tomar tramas de señales de 1024 muestras con solapamientos de 512 muestras, es decir que las 512 primeras muestras de una trama coinciden las 512 últimas muestras de las de la trama anterior.
- Cargar el filtro correspondiente a cada trama
- Multiplicar cada trama con la ventana Hanning
- Filtrar cada trama con su correspondiente filtro HRIR llamando a la función filter().

Por último esta la función ***filter()***, cuya función es filtrar cada trama de audio con su correspondiente filtro, así como aplicar las atenuaciones.

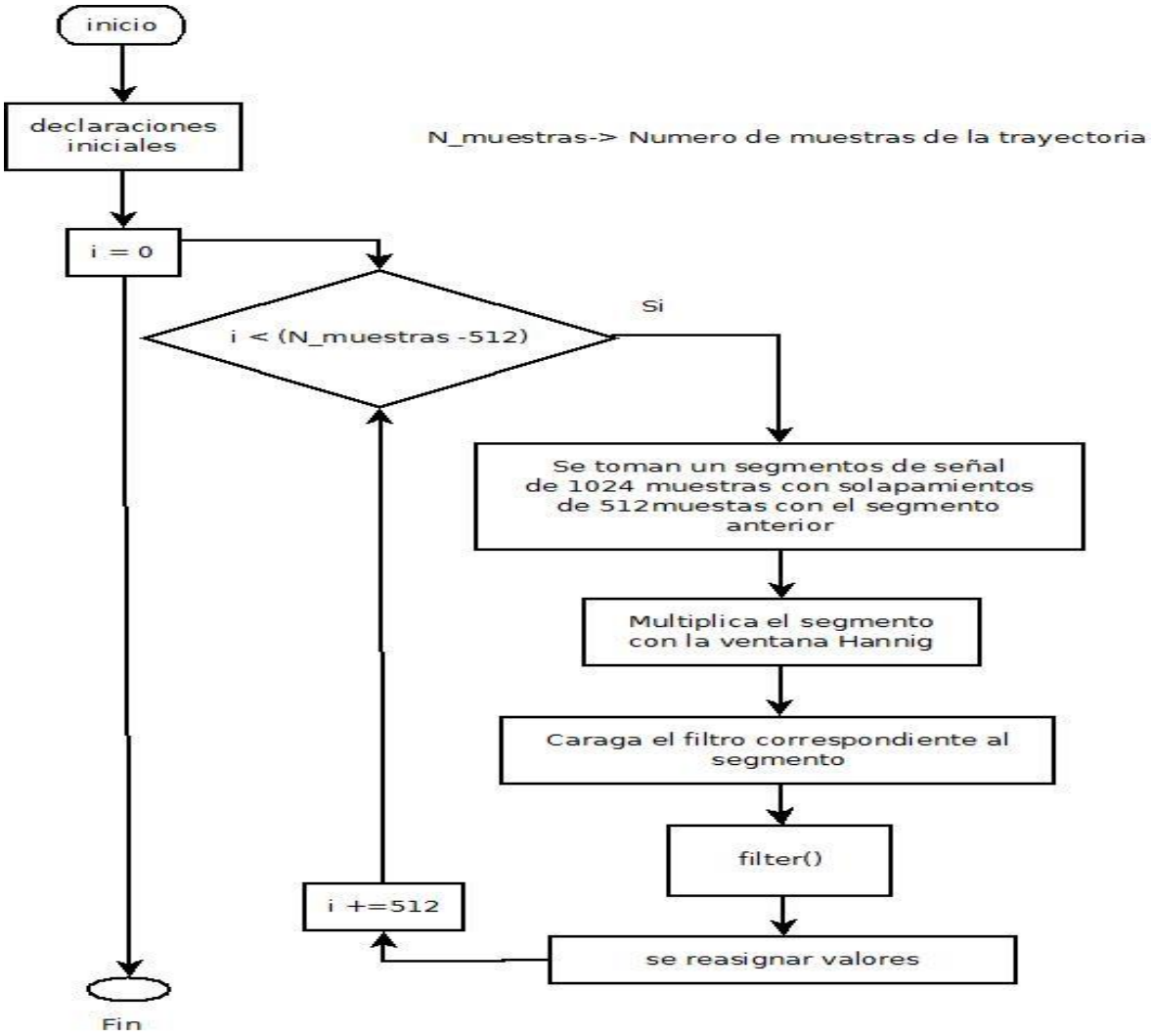

Figura 4.9 Diagrama de flujo de la función start()

5 RESULTADOS Y DISCUSIÓN

El resultado obtenido, figura 5.1, es una aplicación que permite simular un foco en movimiento. Los focos virtuales pueden desplazarse en el sentido de izquierda a derecha o viceversa.

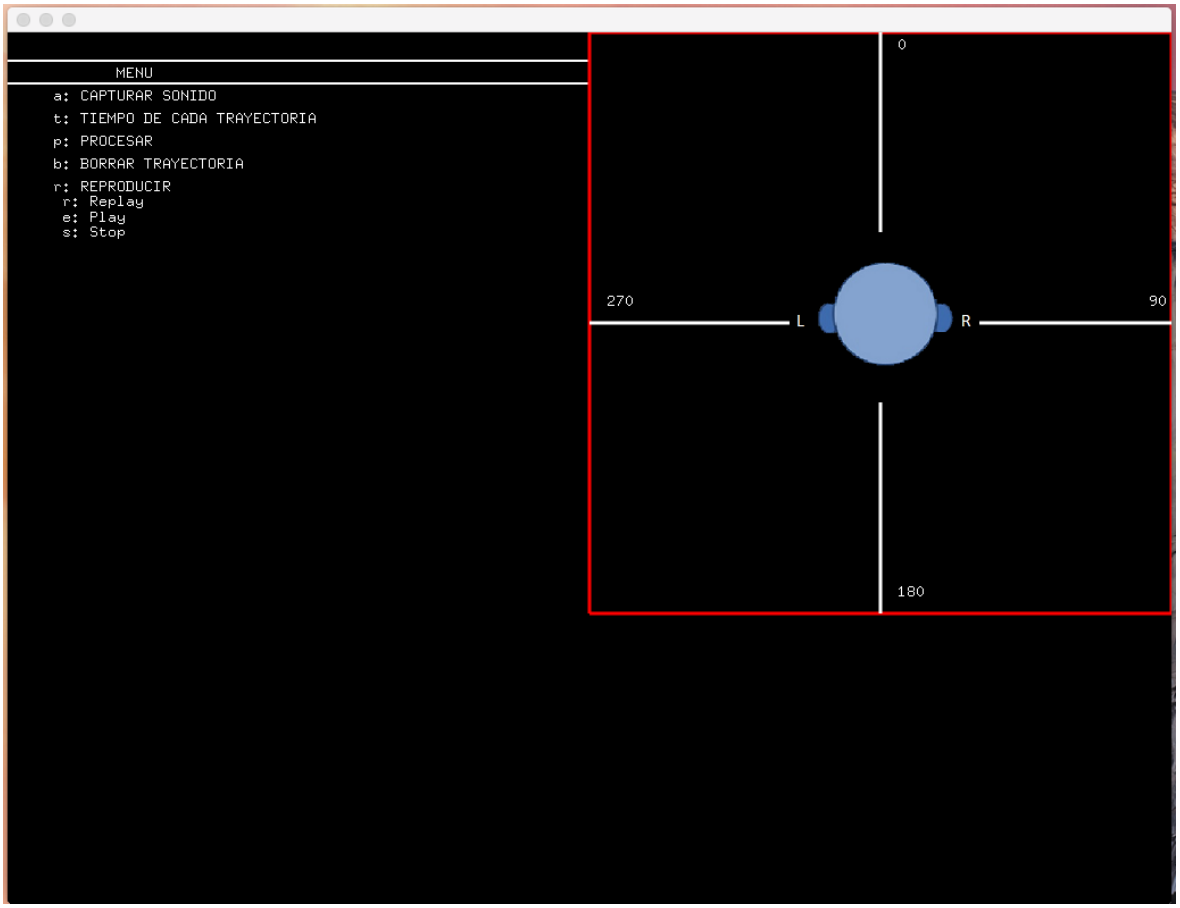

Figura 5.1 Vista de la interfaz de la aplicación

5.1 Dificultades encontradas

Durante el desarrollo de la aplicación han surgidos diferentes dificultades. La principal dificultad fue sincronizar los procesos de grabación y reproducción del sonido. Estas funciones requieren de un conocimiento profundo del funcionamiento de la clase *ofSoundStream*.

Otra dificultad que se produjo fue el tratamiento de la base de datos de filtros HRTF. La base de datos originalmente en formato .mat (formato de datos de Matlab) resulto bastante difícil de manejar en el lenguaje C/C++. Tras buscar varias librerías que permitieran manejar este tipo de datos, se optó por cambiar el formato y almacenar los filtros en archivos con extensión .txt (texto plano) mucho más fácil de manejar en C/C++.

5.2 Posibles Mejoras

Dado que se ha usado una base de datos con HRTF que solo permite simular sonido espacial en el plano horizontal, una mejora seria utilizar una base de datos que también registre datos sobre la elevación. Esto permitiría ampliar el espacio virtual a simular de un plano (plano azimutal), a todo es espacio tridimensional.

Mejorar el tiempo de procesamiento también seria esencial si se pretende usar la aplicación en sistemas de tiempo real. Para sonidos con una duración de 2 minutos se observa un retardo considerable.

5.3 Líneas de Futuro

El futuro de esta aplicación está fuertemente relacionado con el mundo del entretenimiento audiovisual. Una línea de futuro que considero bastante interesante, seria sustituir la base de datos HRTF actual por una con un mayor número de medidas, con esto se conseguiría poder simular el espacio tridimensional completo.

La librería *auralizacion.cpp* podría incluirse en entornos de desarrollo de videojuegos como UnrealEngine o Unity para mejor la sensación de inmersión de los juegos.

6 ANEXOS

6.1 ANEXO I. Manual del usuario

Antes de empezar a usar la aplicación, es necesario revisar que tenemos conectado correctamente un micrófono y un auricular en sus correspondientes puertos de entrada.

Usar la aplicación es tan sencillo como seguir los pasos que se indican en el menú:

1. Primero dibujar la trayectoria en el espacio habilitado (ver figura 7.1). Si desea modificar la trayectoria, simplemente tiene que pulsar la tecla “b” y esta se borrará.

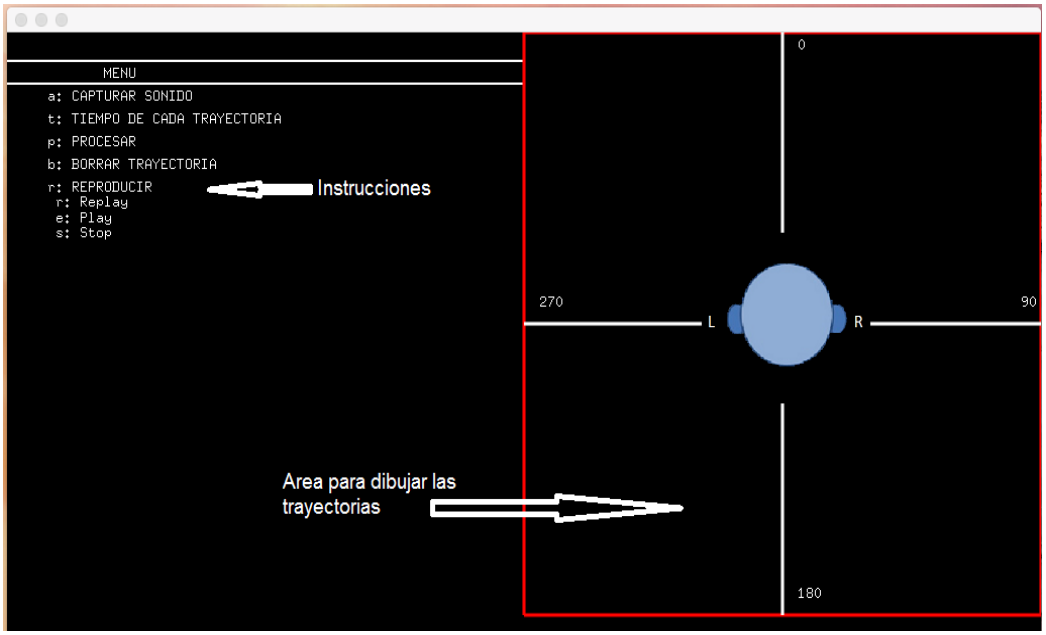

Figura 6.1 Pantalla principal de la aplicación

2. A continuación pulsar la tecla “a” para grabar la señal monoaural. Una vez pulsado la tecla “a”, se podrá observar como este cambia de color indicando que se está grabando sonido (ver figura 7.2)

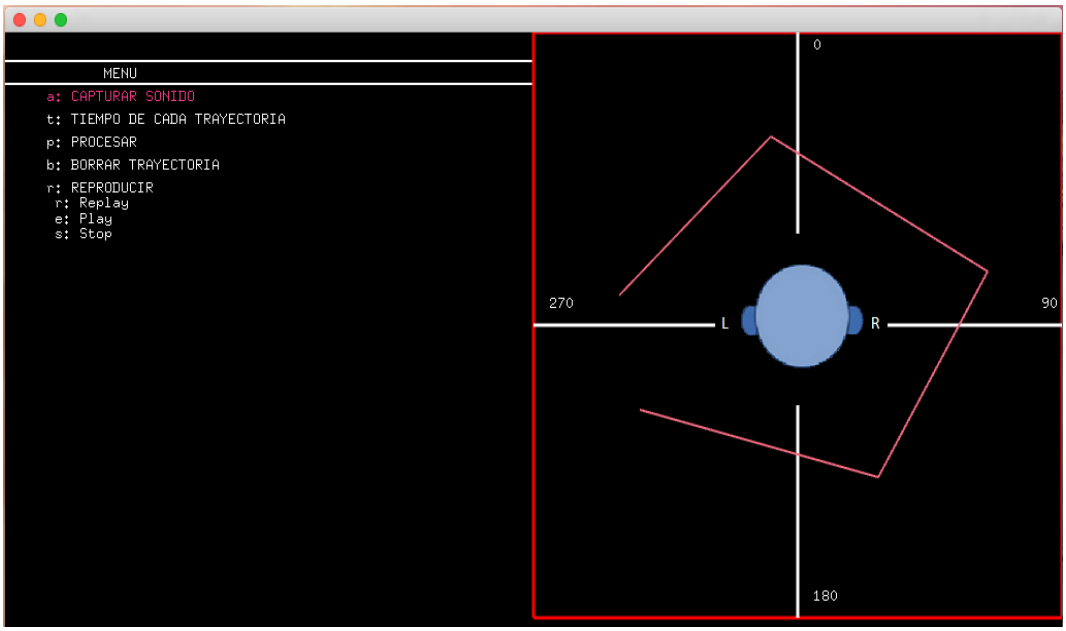

Figura 6.2 Ilustración de un ejemplo de trayectoria con la aplicación grabando

3. Una vez que se ha grabado el sonido deseado, pulsar la tecla “t” para introducir el tiempo de duración de cada trayectoria.
4. Al pulsar “t”, aparecerá la siguiente pantalla, (Figura 7.3), donde podremos seleccionar el tiempo de duración de cada trayectoria. La selección de la duración de cada trayectoria se realiza de la siguiente manera:
    - Primero con el puntero del ratón sobre la barra de tiempos, arrastrar el puntero manteniendo pulsado el botón izquierdo del ratón hasta seleccionar el tiempo de duración de la trayectoria deseado.
    - Una vez seleccionado el tiempo, pulsar la tecla “Enter” para pasar a la siguiente trayectoria.

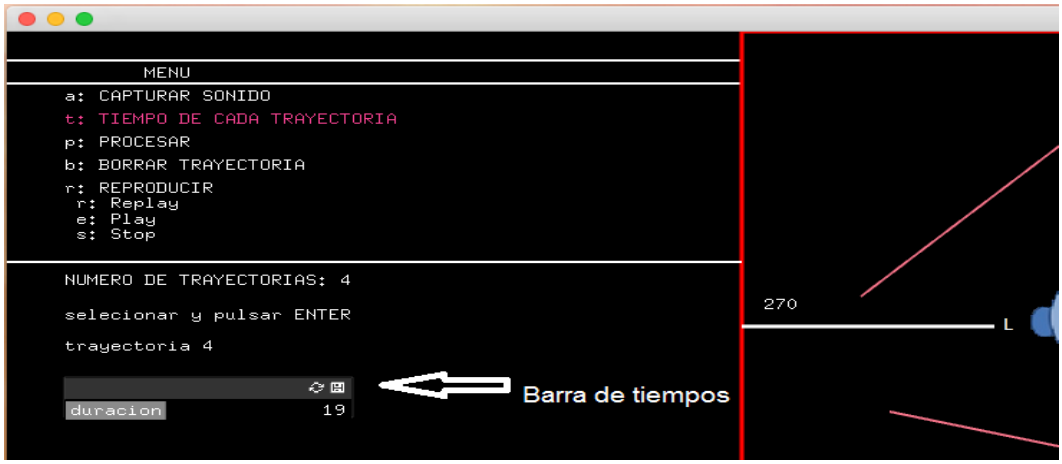

Figura 6.3 Ilustración barra de tiempos

Es importante destacar que la suma de todos los tiempos de las trayectorias debe ser igual o inferior a la duración total del sonido grabado.

5. Después de seleccionar la duración de todas las trayectorias, pulsar la tecla “p” para sintetizar el sonido binaural.
6. Una vez que la aplicación termine de procesar las señales, las instrucciones para reproducir el resultado cambiaran de color (ver figura 7.4).

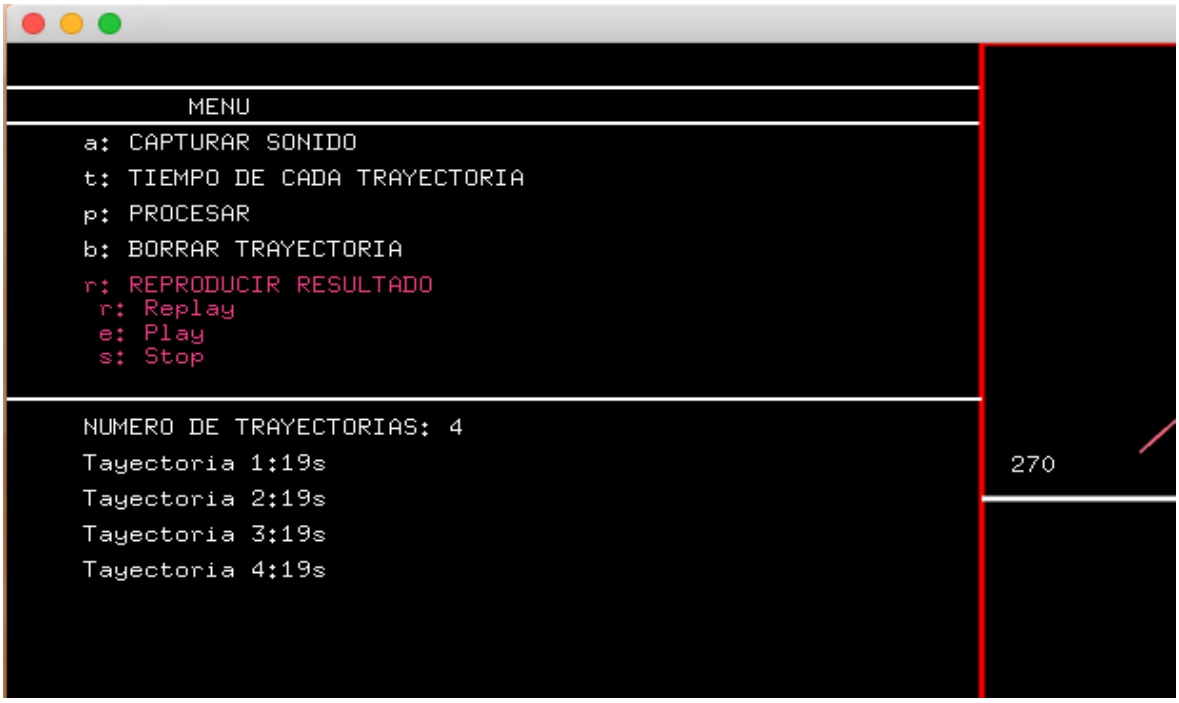

Figura 6.4 Instrucciones de para reproducir el sonido

6.2 ANEXO II. Documentación técnica

En este anexo se adjunta el código desarrollado.

6.2.1 Documentación librería Auralización.cpp

```
#include "ofMain.h"
#include "auralizacion.h"
#include <vector>
#include <fstream>
#include <string>
```

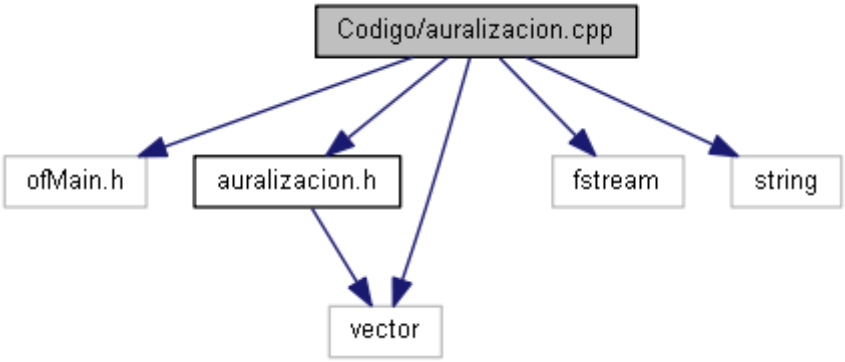

Figura 6.5 Dependencias auralizacion.cpp

Variables

- **const float _PI = 3.14159**

Funciones

- **void interX (std::vector< float > &vector_X)**

  Esta función calcula la interpolacion de las coordenadas X.

- **void interY (std::vector< float > &vector_Y, const std::vector< float > &vector_X)**

  Esta función calcula los valores de las coordenadas Y.

- **void atenuaDistancia (std::vector< float > &vector_atenuadistancia, const std::vector< float > &vector_X, const std::vector< float > &vector_Y)**

  Esta función calcula las atenuaciones que se producen en los puntos de la trayectoria.

- **void angulos (std::vector< int > &vector_angulos, const std::vector< float > &vector_X, const std::vector< float > &vector_Y)**

  Función que calcula el angulo de cada punto de la trayectoria.

- **void filter (float *auxR, float *auxL, float *filtroR, float *filtroL, float ampl)**

  Funcion que se encarga de realizar la convolucion entre las tramas de 1024 muestras de audio y sus corespondientes filtros.

- **void setup_parameter (int N_filtros, const float X1, const float X2, const float Y1, const float Y2, std::vector< int > &vector_angulos, std::vector< float > &vector_atenuadistancia)**

  Función que se encarga de gestionar las llamadas a InterX(), InterY(), **atenuaDistancia()** y **angulos()** para calcular los parámetros de la trayectoria.

- **void Start (std::vector< float > &Raux, std::vector< float > &Laux, const std::vector< float > &ampl, const std::vector< int > &angl, const std::vector< float > &Hann)**

  Función que gestiona el procesamiento de los segmentos de audio corespondientes a cada trama. Divide estos segmentos de audio en tramas de sonido de 1024 muestras con solapamiento de 512 muestras, las multiplica por una ventana Hanning de 1024 muestras. y posteriormente las filtra llamando a la funcion **filter()**.

- **void auralizacion (std::vector< float > &Raudio, std::vector< float > &Laudio, const std::vector< float > &X, const std::vector< float > &Y, const std::vector< int > &tiempo_trayectoria, int f_muestreo)**

Documentación de las variables

const float _PI = 3.14159

Parámetros:

_PI	Constante que define el valor del número pi.

Documentación de las funciones

void angulos (std::vector< int > & vector_angulos, const std::vector< float > & vector_X, const std::vector< float > & vector_Y)

Función que calcula el angulo de cada punto de la trayectoria.

Parámetros:

vector_Y	Vector que contiene las coordenadas Y de la trayectoria.
vector_X	Vector que contiene las coordenadas X de la trayectoria.
vector_angulos	Vector donde se almacenan los angulos de la trayectoria.

Devuelve:

Dado que se trabaja con los vectores originales de cada variable, esta función no devuelve ningún valor.

void atenuaDistancia (std::vector< float > & vector_atenuadistancia, const std::vector< float > & vector_X, const std::vector< float > & vector_Y)

Esta función calcula las atenuaciones que se producen en los puntos de la trayectoria.

Parámetros:

vector_Y	Vector que contiene las coordenadas Y de la trayectoria.
vector_X	Vector que contiene las coordenadas X de la trayectoria.
vector_atenuadistancia	Vector que contiene las atenuaciones de cada punto de la trayectoria.

Devuelve:

Esta función no devuelve ningún valor, trabaja directamente con los vectores originales de cada trayectoria.

void auralizacion (std::vector< float > & **Raudio**, *std::vector< float > &* **Laudio**, *const std::vector< float > &* **X**, *const std::vector< float > &* **Y**, *const std::vector< int > &* **tiempo_trayectoria**, *int* **f_muestreo**)

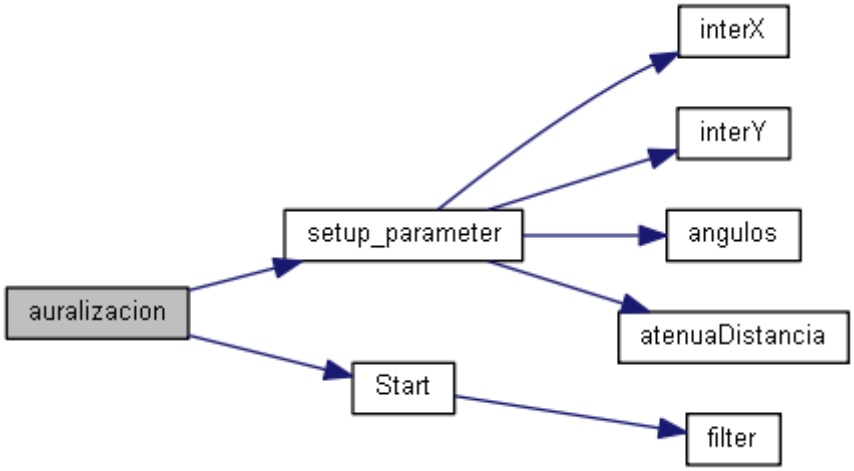

Figura 6.6 Gráfico de llamadas función auralizacio()

Función principal de la librería, gestiona todo el proceso de procesamiento de las señales de audio e indica el orden de llamada de las demás funciones de esta librería.

Parámetros:

Raudio	Vector con la señal de audio del canal derecho.
Laudio	Vector con la señal de audio del canal izquierdo.
X	vector con todas las coordenadas X que constituyen el trayecto del foco que se desea simular.
Y	vector con todas las coordenadas Y que constituyen el trayecto del foco que se desea simular.
tiempo_trayectoria	Vector que almacena el tiempo de duración de cada trayectoria.
f_muestreo	Frecuencia de muestreo de la señal de audio.

Devuelve:

Esta función no devuelve ningún valor, modifica directamente los datos almacenados en cada vector o variable.

*void filter (float * auxR, float * auxL, float * filtroR, float * filtroL, float ampl)*

Función que se encarga de realizar la convolución entre las tramas de 1024 muestras de audio y sus correspondientes filtros.

Parámetros:

auxR	Puntero que apunta al vector con la trama de audio correspondiente al canal derecho.
auxL	Puntero que apunta al vector con la trama de audio correspondiente al canal izquierdo.
filtroR	Puntero que apunta al vector que contiene los coeficientes del filtro del canal derecho.
filtroL	Puntero que apunta al vector que contiene los coeficientes del filtro del canal izquierdo.
ampl	Parámetro que contiene el valor de la atenuación a aplicar a la trama.

Devuelve:

Esta función no devuelve ningún valor, solo modifica los valores a los que apuntan los punteros auxR y auxL.

Nota:

Esta función se llama desde la función start() cada vez que se desea filtrar una trama.

Ver también:

Start()

void interX (std::vector< float > & vector_X)

Esta función calcula la interpolación de las coordenadas X.

Parámetros:

vector_X	Vector que contiene las coordenadas X de la trayectoria.

Nota:

Esta función junto con la función **interX()** realizan la interpolación de puntos entre los puntos inicial y final de una trayectoria.

Devuelve:

Esta función no devuelve ningún valor, trabaja directamente con los vectores.

Ver también:

interY()

void interY (std::vector< float > & vector_Y, const std::vector< float > & vector_X)

Esta función calcula los valores de las coordenadas Y.

Parámetros:

vector_Y	Vector que contiene las coordenadas Y de la trayectoria.
vector_X	Vector que contiene las coordenadas X de la trayectoria.

Devuelve:

Esta función no devuelve ningún valor, trabaja directamente con los vectores originales.

Nota:

Esta función junto con la función **interX()** realizan la interpolación de puntos entre los puntos inicial y final de una trayectoria.

Ver también:

interX()

void setup_parameter (int N_filtros, const float X1, const float X2, const float Y1, const float Y2, std::vector< int > & vector_angulos, std::vector< float > & vector_atenuadistancia)

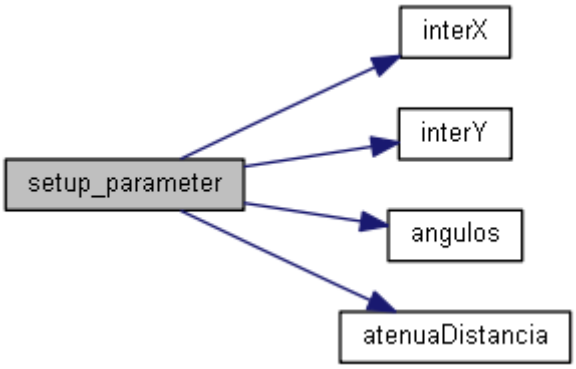

Figura 6.7 Gráfico de llamadas función setup_parameter()

Función que se encarga de gestionar las llamadas a InterX(), InterY(), atenuaDistancia() y angulos() para calcular los parámetros de la trayectoria.

Parámetros:

N_filtros	Numero de filtros que se aplicaran a la trayectoria.
X1	Valor de la coordenada X del punto inicial de la trayectoria.
X2	Valor de la coordenada X del punto final de la trayectoria.
Y1	Valor de la coordenada Y del punto inicial de la trayectoria.
Y2	Valor de la coordenada Y del punto final de la trayectoria.
vector_angulos	Vector donde se almacenan los angulos de cada punto de la trayectoria.
vector_atenuadistancia	Vector que contiene las atenuaciones de cada punto de la trayectoria

Devuelve:

Esta función no devuelve ningún valor, solo modifica los valores de los vectores.

void Start (std::vector< float > & Raux, std::vector< float > & Laux, const std::vector< float > & ampl, const std::vector< int > & angl, const std::vector< float > & Hann)

Figura 6.8 Gráfico de llamadas para esta función Start()

Función que gestiona el procesamiento de los segmentos de audio corespondientes a cada trama. Divide estos segmentos de audio en tramas de sonido de 1024 muestras con solapamiento de 512 muestras, las multiplica por una ventana Hanning de 1024 muestras, y posteriormente las filtra llamando a la función **filter()**.

Parámetros:

Raux	Vector que contiene las muestras de audio de la trayectoria correspondientes al canal derecho
Laux	Vector que contiene las muestras de audio de la trayectoria correspondientes al canal izquierdo
ampl	Vector que contiene las atenuaciones de los puntos que forman la trayectoria.
angl	Vector que contiene los angulos de posición de los puntos que forman la trayectoria.
Hann	Ventana Hanning de 1024 muestras

Devuelve:

Esta función no devuelve ningún valor, modifica los valores de los vectores.

Ver también:

filter()

6.2.2 *Documentación del main.cpp*

```
#include "ofMain.h"
#include "ofApp.h"
#include <math.h>
```

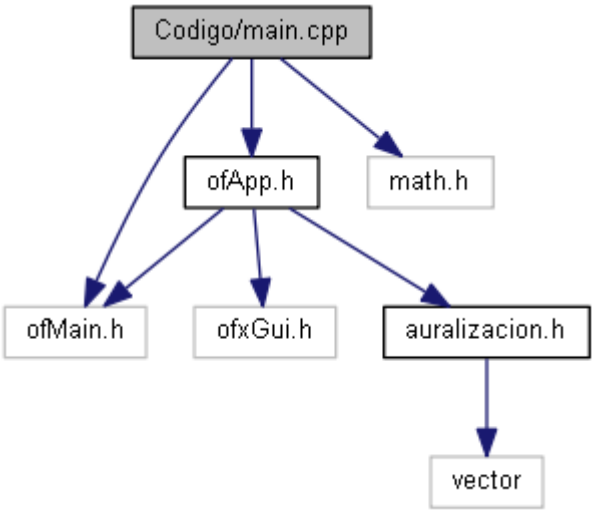

Figura 6.9 Dependencias main.cpp

Funciones

- **int main ()**

Documentación de las funciones

int main ()

Función principal de la aplicación

6.2.3 Documentación del ofApp.cpp

```
#include "ofMain.h"
#include "ofxGui.h"
#include "auralizacion.h"
```

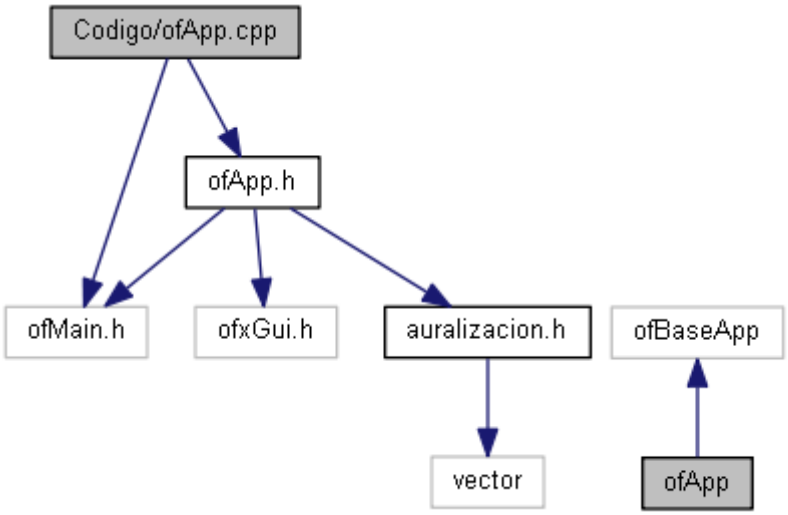

Figura 6.10 Dependencia ofApp.cpp

Métodos públicos

- **void setup ()**

  Función donde se inicializan todas las variables globales de la aplicación.

- **void update ()**

  Función donde se actualizan los valores de las variables de la aplicación.

- **void draw ()**

- **void keyPressed (int key)**

  Función que gestiona los controles de la aplicacion que dependen de las teclas. Esta funcion se ejecuta cada vez que se presiona una tecla.

- **void mousePressed (int x, int y, int button)**

  Funcion que controla las teclas del raton.

- **void audioIn (float *input, int bufferSize, int nChannels)**

  Funcion que gestiona la captura de sonido de sonido.

- **void audioOut (float *output, int bufferSize, int nChannels)**

  Funcion que gestiona la reproduccion del sonido de sonido.

Atributos públicos

- **ofSoundStream sonido_in**
- **ofSoundStream sonido_out**
- **vector< float > left**

  Vector donde se almacenan la señal de audio corespondiente al canal izquierdo.
- **vector< float > right**

  Vector donde se almacenan la señal de audio corespondiente al canal derecho.
- **int reproducir**
- **int grabar**
- **int playpos**
- **int N**

  Variable que contiene la longitud actual del vector de left.
- **int fs**

  Variable global con la frecuencia de muestreo del sonido.
- **vector< int > trayectoria_duracion**
- **vector< float > X**
- **vector< float > Y**
- **ofxPanel mipanel**
- **ofParameter< int > duraciones**
- **int max**
- **int gui**
- **int button_1**
- **int button_2**
- **string notas**
- **ofImage imagen**

Documentación de las funciones miembro

void ofApp::audioIn (float * input, int bufferSize, int nChannels)

Función que gestiona la captura de sonido de sonido.

Parámetros:

input	Datos de entrada.
bufferSize	Tamaño del buffer de audio.
nChannels	Número de canales del audio

void ofApp::audioOut (float * output, int bufferSize, int nChannels)

Función que gestiona la reproducción del sonido de sonido.

Parámetros:

output	Datos de salida
bufferSize	Tamaño del buffer de audio.
nChannels	Número de canales del audio

void ofApp::draw ()

Función que gestiona la interfaz grafica de la aplicación

void ofApp::keyPressed (int key)

Función que gestiona los controles de la aplicación que dependen de las teclas. Esta función se ejecuta cada vez que se presiona una tecla.

Parámetros:

key	Variable que almacena la tecla presionada.

Gráfico de llamadas para esta función:

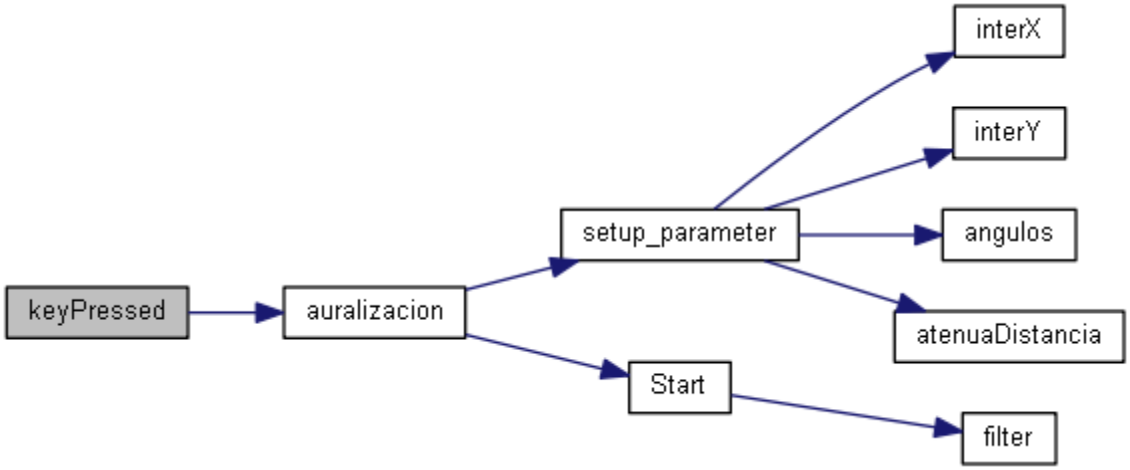

Figura 6.11 Gráfico de llamadas función KeyPressed()

void ofApp::mousePressed (int x, int y, int button)

Función que controla las teclas del ratón.

Parámetros:

x	Almacena la posición X donde se pulso una tecla del ratón
y	Almacena la posición Y donde se pulso una tecla del ratón
button	Indica que tecla del ratón se presionó. Esta función junto con la función **keyPressed()** constituyen el mecanismo de la aplicación para recibir indicaciones del usuario.

void ofApp::setup ()

Función donde se inicializan todas las variables globales de la aplicación.

void ofApp::update ()

Función donde se actualizan los valores de las variables de la aplicación.

Documentación de las Variables miembro

int ofApp::button_1

Parámetros:

button_1	Variable principal para controlar el menú de la aplicación.

int ofApp::button_2

Parámetros:

button_2	Variable adicional para controlar el menú de la aplicación.

ofParameter<int> ofApp::duraciones

Parámetros:

duraciones	Variable adicional necesaria en la gestión de mipanel

int ofApp::fs

Variable global con la frecuencia de muestreo del sonido.

int ofApp::grabar

Parámetros:

grabar	Variable que controla el proceso de grabación de audio **audioIn()**.

int ofApp::gui

Parámetros:

gui	Variable adicional necesaria en la gestión de mipanel

ofImage ofApp::imagen

Parámetros:

imagen	Objeto de la clase ofImage utilizado para cargar las imágenes de la interfaz grafica

vector<float> ofApp::left

Vector donde se almacenan la señal de audio correspondiente al canal izquierdo.

int ofApp::max

Parámetros:

max	Contiene el tiempo de duración máximo disponible

ofxPanel ofApp::mipanel

Parámetros:

mipanel	Objeto de la clase ofxPanel con el que gestiona la entrada de los tempos de duración de cada trayectoria

int ofApp::N

Variable que contiene la longitud actual del vector de left.

string ofApp::notas

Parámetros:

notas	Las variables notas se utilizan para gestionar los mensajes que se imprimen en pantalla

int ofApp::playpos

Parámetros:

playpos	Variable que contiene la pasión actual que se está reproduciendo.

int ofApp::reproducir

Parámetros:

reproducir	Variable que controla la función **audioOut()**.

vector<float> ofApp::right

Vector donde se almacenan la señal de audio correspondiente al canal derecho.

ofSoundStream ofApp::sonido_in

Parámetros:

sonido_in	Objeto de la clase ofSoundStream para gestionar la entrada de sonido.

ofSoundStream ofApp::sonido_out

Parámetros:

sonido_out	Objeto de la clase ofSoundStream para gestionar la salida sonido.

vector<int> ofApp::trayectoria_duracion

Parámetros:

trayectoria_duracion	Vector con las duraciones de cada trayectoria

vector<float> ofApp::X

Parámetros:

X	Vector donde se guardan las coordenadas X que constituyen el trayecto del foco que se desea simular.

vector<float> ofApp::Y

Parámetros:

Y	Vector donde se guardan las coordenadas Y que constituyen el trayecto del foco que se desea simular.

6.3 ANEXO III. Índice de figuras

7 REFERENCIAS BIBLIOGRÁFICAS

7.1 Referencias bibliográficas sobre localización de fuentes y HRTF

[1] Ramón Rodríguez Mariño. Técnicas de sonido binaural en la postproducción audiovisual
https://riunet.upv.es/handle/10251/14732

[2] Acústica de recintos y Auralización
http://www.profesores.frc.utn.edu.ar/electronica/fundamentosdeacusticayelectroacustica/pub/file/FAyE0811E1-Quero-Costantino-Acosta.pdf

[3] Álvaro Lahoz Xaus. Sonido 3D: La percepción acústica egocéntrica y principios de sistemas espaciales
https://alvarblog.files.wordpress.com/2009/11/ra_lahoz_303.pdf

[4] Introduction to head-related transfer functions (hrtf's): representations of hrtf's in time, frequency, and space.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.442.5500&rep=rep1&type=pdf

[5] Karmelo Usategui de la Peña. "Procesador de sonido y estudio de métodos de interpolación de la localización de fuentes basados en HRTFs para la generación de audio 3D".
https://riunet.upv.es/handle/10251/9527?show=full

[6] Jorge Andrés Torres Viveros. Aplicación de técnica de grabación y mezcla binaural para audio comercial y/o publicitario

7.2 Referencias bibliográficas sobre C/C++ y OpenFrameworks

[7] Bruce Eckel. Pensar en C++, Volumen 1

[8] Denis Perevalov. Mastering openFrameworks: Creative Coding Demystified. Liverly Place. Packt Publishing Ltd. Septiembre 2013

[9] Miloš Ljumović. C++ Multithreading Cookbook. Liverly Place. Packt Publishing Ltd. Agosto 2014.