PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

ESCUELA DE SISTEMAS

DISERTACIÓN PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE SISTEMAS Y COMPUTACIÓN

"DISEÑO E IMPLEMENTACION DE UN PROTIPO FUNCIONAL DE UN MOTOR DE AURALIZACIÓN PARA REALIDAD VIRTUAL ACÚSTICA"

ALEX ARMENDÁRIZ

DIRECTOR: DR. JOSÉ FRANCISCO LUCIO NARANJO

QUITO, 2017

DEDICATORIA

Dedico esta tesis a mi familia, a pesar de nuestra problemática convivencia, me han guiado en la vida.

AGRADECIMIENTO

Quiero agradecer, en primer lugar, a mi familia por el apoyo incondicional que he recibido de ellos. A mi madre por el cariño que me ha dado, y por enseñarme a ser mejor persona cada día. A mi padre le debo toda mi curiosidad, por haberme facilitado libros y herramientas por más inusuales que fueran. A mi hermana por todo el cariño y apoyo que me brinda diariamente. Gracias a mi tío Galo Armendáriz por su sincero apoyo desinteresado.

A mi director de tesis José Francisco Lucio Naranjo, por compartir de manera paciente y desinteresada todo su conocimiento. Ya que sin el este tema de tesis no hubiera sido posible de realizar.

A mis excompañeros de investigación, Israel Proaño y Andrés Asimbaya por ayudarme en la implementación y pruebas de este proyecto.

A todos mis amigos artistas por su ayuda y por cambiar drásticamente mi forma de pensar, en especial a Diego Carvajal por el diseño del ícono de la aplicación.

A todos esos piratas y activistas del conocimiento como Alexandra Elbakyan, que mediante la liberación de artículos científicos promueven el libre conocimiento y hacen posible la realización de trabajos de investigación de manera equitativa para todos.

Finalmente, gracias a todos los activistas software libre y open source, que mediante su código fuente, foros, blogs, etc. me transmitieron conocimiento más expresivo que cualquier texto científico.

¨Por qué alistarte en la Marina si puedes ser un pirata¨

Steve Jobs

RESUMEN

En este trabajo se presenta un módulo de software de aurilización en tiempo real que será utilizado para medir la calidad y precisión de los efectos de inmersión acústica generada por la misma. Dicho módulo cumple la función de generar, en una señal de audio cualquiera, un efecto de posicionamiento tridimensional que permite al oyente determinar la ubicación de una fuente de sonido dentro de un ambiente virtual. Este efecto se logra usando una técnica de procesamiento de señales digitales llamada convolución segmentada y varias funciones contenidas en una base de datos de HRIRs (del inglés, Head Related Impulse Responses). El módulo fue implementado apoyándose de un ambiente tridimensional virtual interactivo, en el cual el usuario puede modificar su posición mediante periféricos tales como el ratón y el teclado. De esta forma, los efectos fueron registrados, analizados, discutidos y finalmente validados.

Palabras Clave: Aurilización, convolución segmentada, ambientes 3D interactivos, ETA, realidad virtual.

ABSTRACT

The present work presents a real time auralization software module used to measure the quality and precision of the effects of acoustic immersion generated by itself. This software module has the ability to insert, in any audio signal, the effect of tridimensional positioning that allows the listener to determine the whereabouts of a sound source in a tridimensional virtual environment. This effect is achieved by using a signal processing technique called segmented convolution and several functions contained in a HRIR (Head Related Impulse Responses) database. The module was implemented with the aid of an interactive virtual three-dimensional environment, in which the user can modify their position using peripheral devices such as the mouse and the keyboard. In this way, the effects were recorded, analyzed, discussed and finally validated.

Keywords: Auralization, segmented convolution, interactive 3D environments, ETA, virtual reality.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS Y ALGORITMOS

INTRODUCCIÓN

Las técnicas de Realidad Virtual (RV) han ganado relevancia gracias al avance tecnológico computacional. Para tener una idea de su relevancia, es importante notar que en el año 2016 se registraron ventas aproximadas a mil millones de dólares en productos de realidad virtual (LEE & DUNCAN, 2016). En ese sentido, la RV está trascendiendo a estímulos meramente visuales, abriendo la puerta a otras técnicas enfocadas a sentidos tales como la audición.

Según Vorländer (2008), la palabra “aurilización” (relativa a auricular) es análoga a “visualización”, donde se “hace visible” un objeto proveniente de distintas fuentes (reales o virtuales). La aurilización ocurre cuando un efecto acústico (causado por un ambiente y las características del receptor) es procesado en un resultado audible.

La aurilización en simuladores computacionales utiliza respuestas impulsivas que se producen mediante la aplicación de algoritmos de procesamiento de señales digitales y algoritmos de simulación de propagación de la onda acústica. Los algoritmos de procesamiento de señales digitales utilizan productos de convolución entre una señal de audio digital arbitraria con una de las respuestas impulsivas biauriculares de una sala (VORLÄNDER, 2008).

Según la reseña histórica de Vorländer (2008), en 1929 Spandöck y sus colegas realizaron los primeros modelos matemáticos con los que trataron de procesar señales en Múnich. Debido a las barreras tecnológicas, no fue hasta 1949 que Spandöck, gracias a la aparición de las cintas magnéticas logró crear los primeros modelos audibles. En 1965 con Cooley y Tukey aparece el algoritmo de transformada rápida de Fourier (FFT), aunque Gauss ya había descrito los pasos críticos necesarios para el algoritmo en 1805 (BERGLAND, 1969). El algoritmo FFT redujo considerablemente los cálculos necesarios para una transformada de Fourier, lo que permitiría realizar simulaciones acústicas con la capacidad de procesamiento disponible en la época (WEISSTEIN, 2017).

En 1968 con la aparición de las computadoras, se crea el primer software de simulación acústica desarrollado por Krokstad. Más tarde en la década de los 90's con el aumento de la capacidad de procesamiento y almacenamiento de los computadores personales, las simulaciones numéricas acústicas se tornaron mucho más asequibles. Además, en esta década, Kleiner (1993) introdujo el término aurilización, y desde entonces se han realizado

importantes avances, en los algoritmos de modelado acústico, aurilización biauricular y las técnicas de reproducción de los mismos.

La meta actual por alcanzar para los investigadores es lograr construir simuladores de aurilización precisos que funcionen en tiempo real, con los cuáles se logre una completa inmersión acústica del usuario en la escena virtual. Uno de los retos más grandes ha sido unir estos simuladores con la tecnología actual de realidad virtual, que permiten que el usuario interactúe con estos ambientes virtuales (lo cual tiene un costo computacional alto) (VORLÄNDER, 2008).

La aurilización, al ser una técnica de realidad virtual, resulta sumamente útil en el análisis subjetivo acústico de recintos (tanto los existentes como los que se encuentran en fase de proyecto). También tiene aplicaciones en dispositivos electrónicos de apoyo a personas con capacidades especiales (ETA, del inglés Electronic Travel Aid), vídeo juegos, cine, entre otros.

Los dispositivos ETA tienen como objetivo detectar obstáculos y de alguna forma informar al usuario de la posición de los mismos. Una alternativa en ese sentido es generar sonidos sintetizados utilizando técnicas de aurilización, que recreen la impresión de la presencia de una fuente sonora mediante auriculares estéreo en la posición del obstáculo.

Estudios cuantitativos previos realizados en la Universidad Politécnica de Valencia para un dispositivo ETA (LENGUA & DUNAI , 2013) miden la influencia que tienen las HRIRs (del inglés, Head Related Impulse Responses) en sonidos sintetizados, en lo que se refiere a la ubicación de obstáculos por parte de personas no videntes. A diferencia de esos estudios, el presente proyecto pretende medir cuantitativa y cualitativamente la precisión de un sistema de alertas de sonido 3D con sujetos que no tengan entrenamiento previo. Para dicho objetivo se desarrolló software interactivo que recrea un ambiente donde existen diversas fuentes sonoras, formando así un entorno de pruebas.

Esta disertación se encuentra estructurada de la siguiente forma; en el capítulo I se abordan todas las técnicas matemáticas, algoritmos y técnicas de procesamiento de señales utilizadas. En el capítulo II se aborda el análisis, diseño, la implementación y pruebas necesarias para el desarrollo de la aplicación. A continuación, en el capítulo III se presentan los resultados obtenidos y finalmente en la sección IV se discuten los principales hallazgos de la investigación.

CAPÍTULO I: FUNDAMENTOS TEÓRICOS

En el presente capítulo se presentarán los fundamentos teóricos necesarios para la ejecución del proyecto. A lo largo de este capítulo se presentarán las siguientes secciones:

- Consideraciones Teóricas Acústicas
- Consideraciones Teóricas Gráficas y Físicas

1.1. Consideraciones Teóricas Acústicas

En la presente sección se presentan las consideraciones teóricas acústicas necesarias para la generación de sonido, así como sus descripciones matemáticas.

1.1.1 Sistemas Lineales Invariante en el Tiempo

Una señal es la descripción de cómo un parámetro varía dependiendo de otro parámetro, como por ejemplo el voltaje que cambia con el tiempo en los artefactos electrónicos.

Un sistema es cualquier proceso que produce una señal de salida en respuesta a una señal de entrada, estos procesos pueden ser continuos o discretos (SMITH, Chapter 5: Linear Systems, 1998). Para que sea considerado sistema lineal debe cumplir con las siguientes propiedades (PHISHDADIAN, 2016):

1. Si tenemos un sistema lineal tiene como entradas x_1 y x_2 y como resultado y_1 y y_2 respectivamente. Al multiplicar estas entradas por un factor a y b respectivamente, sumarlas y tomarlas nuevamente como entradas al sistema. Obtendremos las salidas anteriores sumadas entre sí y multiplicadas por su factor respectivo.

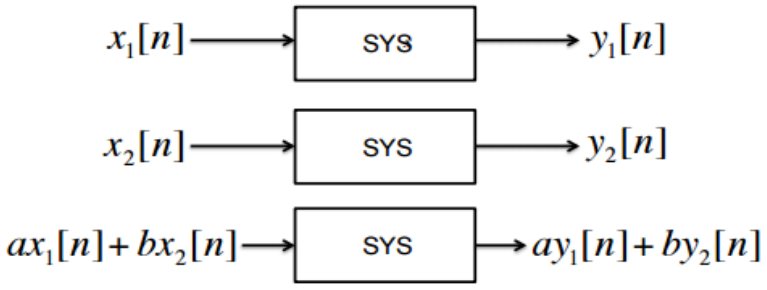


Figura 1: Principio de Superposición (PHISHDADIAN, 2016)

2. La respuesta de un sistema lineal debe ser la misma en cualquier instante del tiempo.

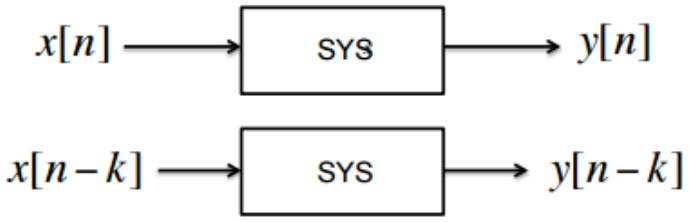


Figura 2: Invariancia en el Tiempo (PHISHDADIAN, 2016)

Si excitamos un sistema lineal invariante en el tiempo con un impulso, vamos a obtener una salida que se conoce como respuesta impulsiva. La respuesta impulsiva lleva la información acerca de la dinámica del sistema. Por lo tanto, es posible determinar la información completa acerca del sistema excitándolo con un impulso y midiendo su respuesta (MARTIN, 2011). Un impulso no es más que una herramienta matemática que no existe en el mundo físico real, y representa una magnitud infinitamente grande aplicada en un tiempo infinitesimal.

Un impulso se representa en el caso de sistemas discretos de audio de punto flotante (que sus muestras varían entre 1 y -1) por una muestra de valor 1 en la posición 0 como se muestra en la Ec. 1 y Fig. 3 (PHISHDADIAN, 2016).

$$\delta[\mathrm{n}] = \begin{cases} 1 \text{ si n } = 0 \\ 0 \text{ otros} \end{cases} \tag{1}$$

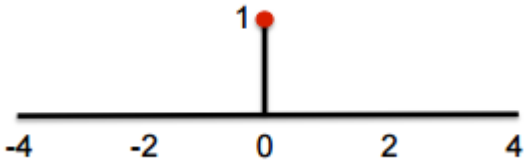


Figura 3: Impulso Discreto (PHISHDADIAN, 2016)

Y en el caso de un sistema continuo el Delta de Dirac es una función que vale cero en todo su domino, excepto en cero en el cual tiene un valor de infinito (Ec. 2). Una de las características importantes del Delta de Dirac se describe en la Ec. 3. Además, el espectro del Delta de Dirac es constante, esto quiere decir que tiene la misma respuesta para todas las frecuencias (FLETCHER, 2012).

$$\delta(t) = \begin{cases} \infty \text{ si } t = 0 \\ 0 \text{ otros} \end{cases} \tag{2}$$

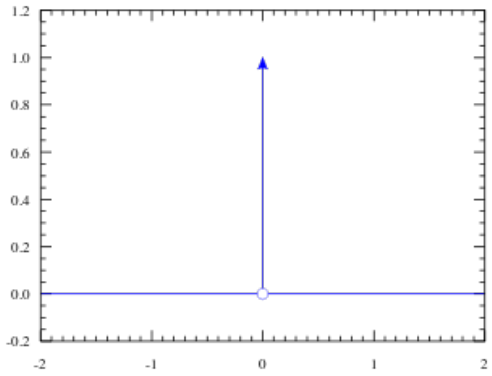


Figura 4: Delta de Dirac

$$\int_{-\infty}^{\infty} \delta(t)\, dt = 1 \tag{3}$$

El comportamiento acústico de una sala puede ser considerado un sistema lineal invariante en el tiempo. Esto sucede debido a que, para niveles de presión acústica inferiores a 130 dB, el oído medio funciona como un sistema lineal (AERTS & DIRCKX, 2009). En este sentido, dos puntos fijos cualesquiera dentro de un ambiente pueden ser considerados como los terminales de entrada y salida de un sistema lineal de transmisión acústica. La aurilización utiliza este principio de los sistemas acústicos lineales invariantes en el tiempo. Esto quiere decir que, la respuesta impulsiva caracteriza completamente el sistema de transmisión acústico desde una ubicación específica de la fuente sonora hasta la posición del receptor (ALONSO, 2012).

Para entender como reproducir los efectos en un sistema lineal invariante en el tiempo se deben entender las siguientes propiedades (PHISHDADIAN, 2016):

• Una señal arbitraria x[n] se puede expresar como la suma de las respuestas impulsivas escaladas y desplazadas (Ec. 4).

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\partial[n-k] \tag{4}$$

• Usando la Ecuación 4 como entrada a un sistema lineal invariante en el tiempo, obtendremos como salida la Ec.5 que es conocida como producto de convolución.

$$y[n] = \sum_{k=-\infty}^{\infty} x[n]h[n-k] \tag{5}$$

Por lo que, los efectos de un sistema lineal pueden ser insertados en una señal de audio arbitraria utilizando un producto de convolución (Ec.5) (PHISHDADIAN, 2016). Por esta razón, un sistema lineal acústico invariable en el tiempo para realidad virtual está definido por su respuesta impulsiva biauricular y sus efectos pueden ser insertados en una señal de audio arbitraria utilizando un producto de convolución. El producto de convolución se realiza con una respuesta impulsiva específica, obteniendo de esta forma el efecto de posicionamiento de la fuente sonora en el espacio.

1.1.2 Producto de Convolución

El producto de convolución es una función que, de forma lineal y continua, transforma una señal de entrada en una nueva señal de salida (GONZALES & WINTZ, 1977).

La Ec. 6 representa el producto de convolución en el dominio continuo.

$$(f * g)(t) = \int_0^t f(t-\tau)g(\tau)d\tau \tag{6}$$

Ya que se utilizarán simulaciones computacionales, es necesario utilizar la forma discreta para realizar el proceso de aurilización. El producto de convolución continuo debe distribuido en un peine de deltas de Dirac (secuencia de deltas de Dirac), obteniendo (la Ec. 7) la misma función que la Ec. 5.

$$f[m] * g[m] = \sum_{k=-\infty}^{\infty} f[n]g[m-n] \tag{7}$$

En un producto convolución cada muestra de la señal discreta es multiplicada por las muestras de la respuesta impulsiva, y se suman las superposiciones que existan entre los diferentes productos. La Fig. 5 muestra la representación gráfica de un producto de convolución.

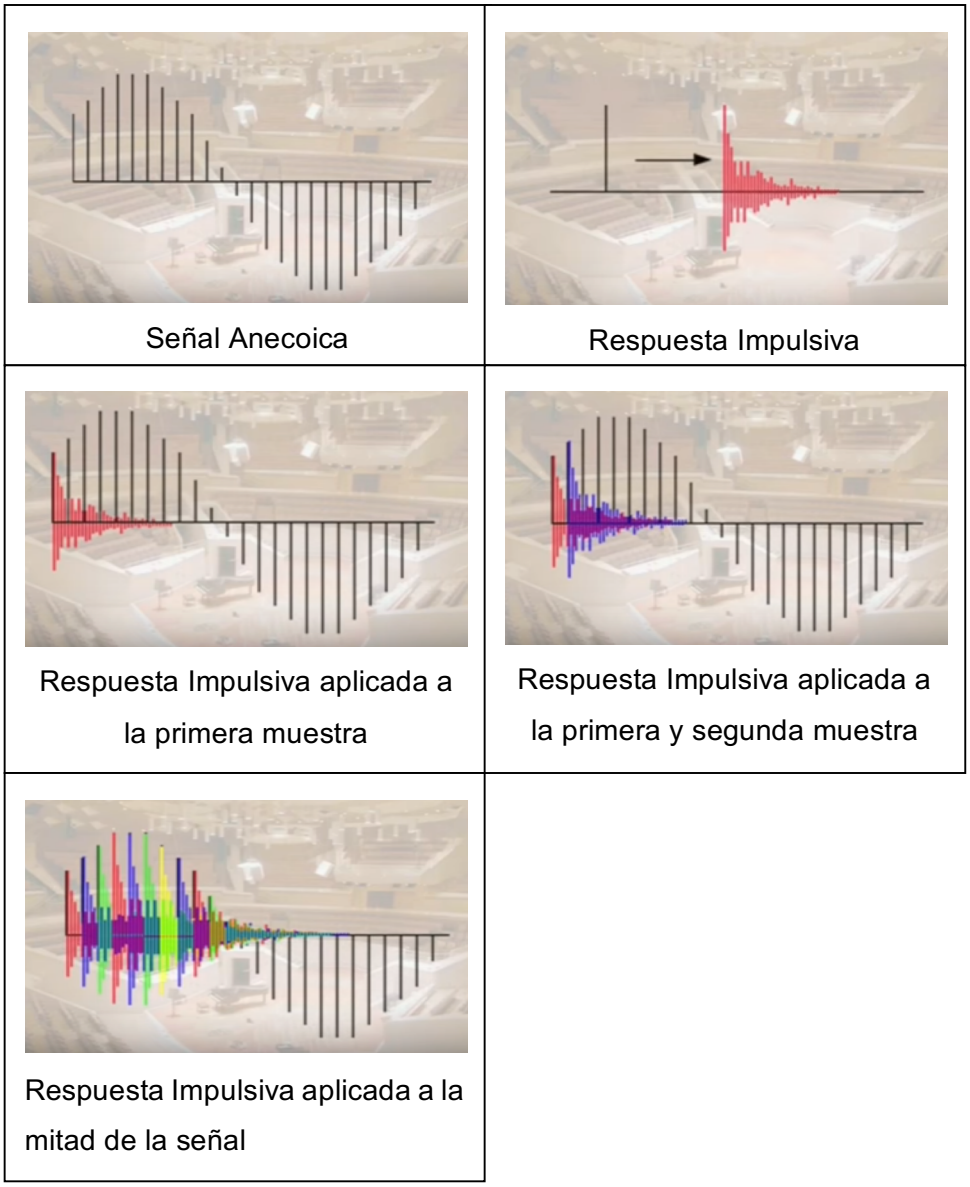


Figura 5: Descripción Gráfica del Proceso de Convolución (AUDIOEASE, 2011)

1.1.3 Respuesta Impulsiva Asociadas a la Cabeza (HRIR)

El ser humano puede localizar sonido debido en gran parte a la audición biauricular. El sonido puede llegar a una persona desde cualquier punto del espacio, este sonido va ser distorsionado debido a las reflexiones (Fig. 6 (a), cambio de dirección que cumple la Ley de Snell[1]) y difracciones (Fig. 6 (b), cambio de dirección del rayo azul) con la cabeza y el torso. Estas distorsiones son dependientes de la dirección y distancia, por lo que el sonido llega a los dos oídos con diferencias de tiempo y amplitud. Por ejemplo, el sonido de una fuente ubicada al lado derecho de la cabeza tardará más en llegar al oído izquierdo, además el sonido sufrirá un cambio espectral debido a las distorsiones anteriormente mencionadas. Estos efectos son fácilmente percibidos por una persona y son llamadas diferencias interauriculares de tiempo (ITD del inglés, interaural time differences) y diferencias interauriculares de nivel (ILD del inglés, interaural level differences). Las HRIRs (del inglés, Head Related Impulse Responses), son un par de funciones que describen de manera completa las distorsiones lineales causadas por la cabeza y torso de una persona (VORLÄNDER, 2008).

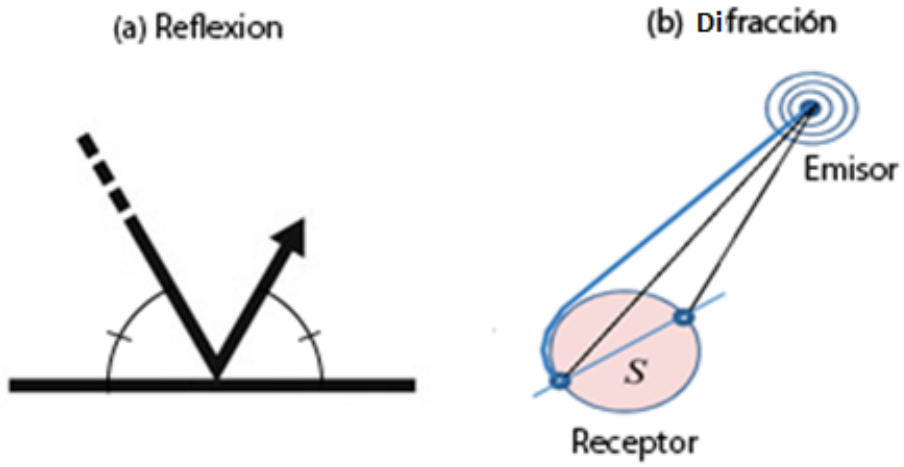


Figura 6: Reflexión y Difracción (SHIMOYAMA, 2012)

[1] **Ley de Snell**: El ángulo que forma el rayo incidente con la recta normal al plano (ángulo de incidencia) es igual al ángulo de esta normal con el rayo reflejado (ángulo de reflexión); por lo que los ángulos formados entre el rayo incidente y el plano, al igual que el rayo reflejado y el plano también son iguales. (RUIZ, 1999)

Un aspecto que merece ser mencionado es la llamada confusión frontal / dorsal. Esta se da cuando una fuente sonora se encuentra en ciertas posiciones simétricas con respecto al plano vertical (Fig. 7), en las cuales los indicios biauriculares pueden alcanzar valores muy próximos, formando conos de confusión (BENSOM & MARTENS, 2006). Este problema es naturalmente resuelto por lo seres humanos realizando pequeños movimientos rotacionales de la cabeza, transformando el problema a uno de localización lateral (OCULUS VR, LLC, 2017).

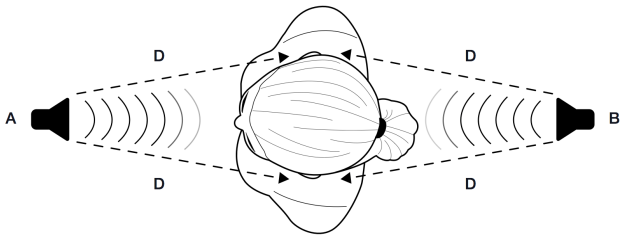


Figura 7: Ambigüedad Frontal/Dorsal (OCULUS VR, LLC, 2017)

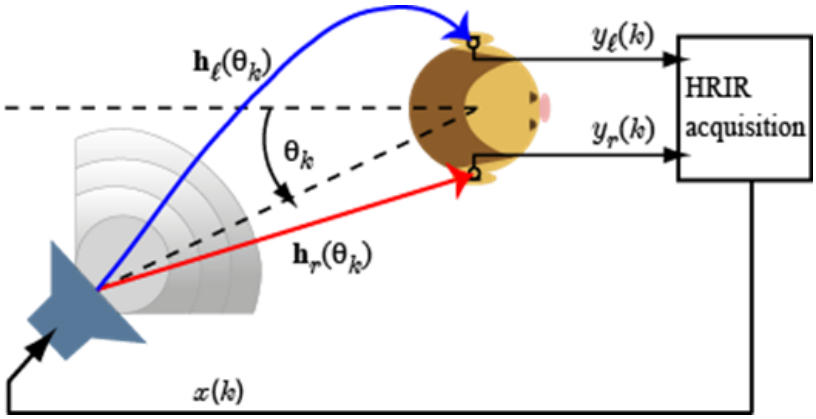


Figura 8: Mediciones Experimentales HRIRs (RWTH AACHEN UNIVERSITY, 2016)

El sistema de la Fig. 8, formado por una fuente sonora y un receptor (persona o maniquí de pruebas) se puede representar como un sistema lineal invariable en el tiempo. Por lo que se puede medir su respuesta impulsiva (HRIR) experimentalmente usando micrófonos (Fig. 9). Dichas HRIRs se miden con una frecuencia de muestreo predefinida. Una de las bases de HRIRs más conocidas es la de KEMAR del MIT que fue levantada en 1994 (GARDNER &

MARTIN, 1994). Para esto se utilizaron secuencias de ruido para obtener las respuestas impulsivas asociadas a la cabeza con una frecuencia de muestreo de 44,1 kHz. Esta base de HRIRs contiene un total de 710 posiciones diferentes (en azimut de 0 a 360 grados y en elevaciones que van desde -40 grados a +90 grados). Las mismas se realizaron en una cámara anecoica (sin eco). por lo cual las características direccionales del sonido no tienen influencia del recinto. Por lo que la aurilización por convolución está basada en el uso de un par de HRIRs convoluidas con una señal monofónica de audio anecoica, obteniendo así, una señal de dos canales de audio con características direccionales que puede ser escuchada mediante auriculares ecualizados.

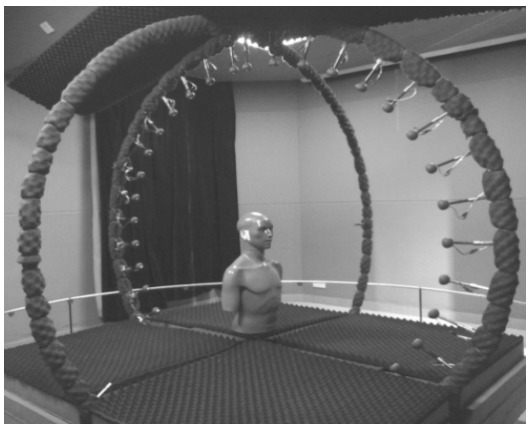


Figura 9: Mediciones de HRIRs en el Laboratorio (ZHONG, 2014)

1.1.4 Filtros Digitales

Las frecuencias bajas por naturaleza son no direccionales por lo que no es necesario añadir el efecto posicionamiento en ellas (WOZNIAK, 2015). En consecuencia, antes de realizar el proceso de convolución es necesario usar filtros digitales para dividir la señal monofónica de audio que queremos procesar en dos partes, una que contenga frecuencias bajas (de 0 Hz a 200 Hz, ya que no existen diferencias interauriculares significativas en este rango de frecuencias (VORLÄNDER, 2008)) y otra frecuencias altas (de 200 Hz a 22,05 KHz, siendo 22,05 KHz la frecuencia más alta muestreable con una frecuencia de muestreo de 44100 Hz debido al teorema Nyquist (SMITH, The Sampling Theorem, 1998)).

Un filtro se puede definir como todo procesamiento que transforma la naturaleza de una señal sonora. Un filtro digital está definido por un algoritmo computacional mediante el cual una señal digital es transformada en una segunda señal digital de salida.

1.1.4.1 Tipos de Filtros

Los filtros más conocidos son los filtros paso bajo (*Low Pass, LP*), paso alto (*High Pass,* HP), paso de banda (*Band Pass,* BP) y los filtros rechaza banda (*Band Reject, Band stop o Notch*) (GUTIÉRREZ, 2009).

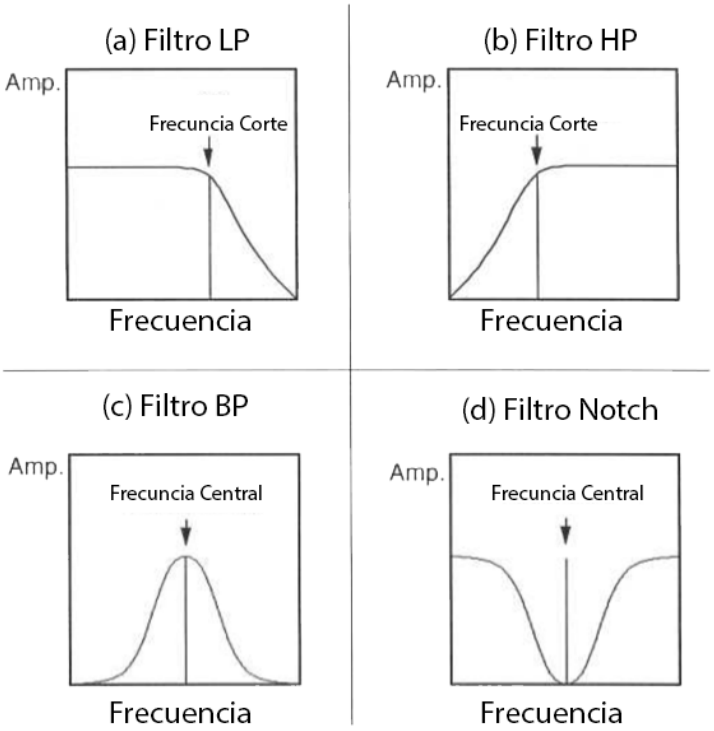


Figura 10: Tipos de Filtros Digitales (GUTIÉRREZ, 2009)

- Los filtros paso bajo (LP) (Fig. 10 (a)) atenúan las señales sobre una determinada frecuencia y dejan pasar las frecuencias que están por debajo de esa determinada frecuencia.
- Los filtros paso alto (HP) (Fig. 10 (b)) atenúan las señales bajo una determinada frecuencia y dejan pasar las frecuencias que están por encima de esa determinada frecuencia.
- Los filtros paso banda (BP) (Fig. 10 (c)) dejan pasar las frecuencias que están situadas en un determinado rango de frecuencias, es decir, en una banda determinada.
- Los filtros rechazo de banda (BR) (Fig. 10 (d)) dejan pasar todas las frecuencias excepto las que están situadas en un determinado rango de frecuencias, es decir, en una banda determinada.

1.1.4.2 Funcionamiento de Base de un Filtro Digital

Los filtros digitales tienen dos funcionamientos de base los filtros FIR (*Finite Impulse Response*) (a) y IIR (*Infinite Impulse Response*) (b), que se ilustran en la Fig. 11.

Figura 11: Funcionamiento de Base de un Filtro Digital (GUTIÉRREZ, 2009)

La salida de tanto los filtros FIR como IIR se puede definir como una combinación lineal de muestras de la entrada presentes y pasadas (Ec. 8). Los factores a_i son los coeficientes del filtro, estos coeficientes le dan las características al filtro (GUTIÉRREZ, 2009).

$$y[n] = a_0 \cdot x[n] + a_1 \cdot x[n-1] + a_2 \cdot x[n-2] + \cdots + a_N \cdot x[n-N] \quad (8)$$

Los filtros FIR (Fig. 11 (a)) funcionan combinando la señal de entrada retrasada con la señal de entrada actual (GUTIÉRREZ, 2009). Los filtros digitales basados en este funcionamiento se dice que son de respuesta impulso finita o FIR ya que su respuesta impulsiva después de una serie de muestras se transforma en cero.

Los filtros IRR (Fig. 11 (b)) funcionan retardando la señal salida, la cual combinamos con la nueva señal de entrada (GUTIÉRREZ, 2009). Estos filtros se conocen como filtros de respuesta impulsiva infinita o IIR, ya que son filtros recursivos que tienen retroalimentación (*feedback*) lo cual causa que las muestras de su respuesta impulsiva se acerquen a cero, pero nunca se transformen en cero. Se recomienda el uso de estos filtros para el uso en aplicaciones en tiempo real ya que un filtro IIR requiere un orden[2] mucho menor para cumplir las especificaciones de diseño (por lo que el costo computacional de procesamiento es menor) (MATÍNEZ, 2010).

[2] **Orden de un Filtro**: Cantidad de muestras de retraso que requiere para realizar el procesamiento de la señal (Martínez, 2010).

1.1.5 Convolución Segmentada

Al utilizar un producto de convolución, expresado en la Ec.7, puede darse que una señal sea más larga que la otra, impidiendo el procesamiento de la señal por bloques (lo que introduce retardos en el procesamiento) (SPORS, 2015).

Por lo que la Ec.7 no es apta para la generación de audio en tiempo real, ya que el procesamiento de un bloque debe tardar menos de 11,61 milisegundos (por los requerimientos que serán expuestos en el Capítulo II). Por tal motivo, es necesario usar un algoritmo de convolución segmentada, para que sea posible el uso de señales de diferentes tamaños y reducir así el costo computacional de la misma (SPORS, 2015). El procesamiento por bloques basa su funcionamiento en la Ec. 9.

$$x_L[k] = \sum_{p=0}^{L/P-1} x_p[k - p\,P] \tag{9}$$

Este algoritmo, conocido como “sumas de superposiciones” (SPORS, 2015), separa la señal $x_L[k]$ (de tamaño L) en segmentos de $x_p[k]$ (de tamaño P) que están definidos como:

$$x_p[k] = \begin{cases} x_L[k - p\,P]\,, para\ k = 0,1,\dots,P-1 \\ 0, otros \end{cases} \tag{10}$$

La longitud P del bloque depende del tamaño del buffer de salida de audio. No obstante, el tamaño de cada elemento procesado será de $P + N - 1$ (donde N es el tamaño de la respuesta impulsiva $h_N[k]$). Este problema se soluciona sumando las muestras que sobrepasen la longitud P al siguiente bloque, tal como ilustra la Fig.12.

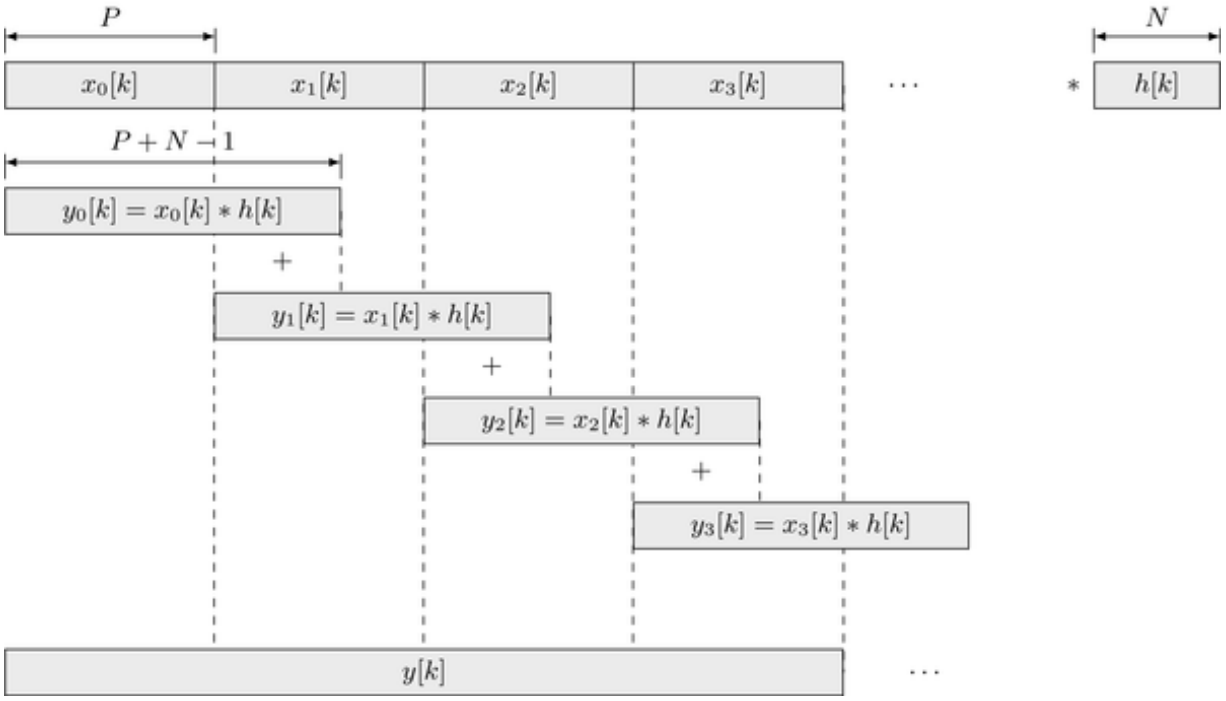


Figura 12: Convolución Segmentada (SPORS, 2015)

Por lo anterior, el resultado de una convolución $X_L[k] * h_n[k]$ se puede expresar como la suma de superposiciones de una serie de convoluciones $X_p[k] * h_N[k]$, cada una desplazada en múltiplos de P (SPORS, 2015).

1.1.6 Reducción de Ruido Convolución Segmentada

Una vez realizado el producto de convolución, al cambiar las HRIRs rápidamente se creará ruido por lo que es necesario realizar interpolaciones lineales (Ec.11) entre cada una de las muestras de la HRIR para reducir el ruido (WOZNIAK, 2015). Para este caso usaremos una variación de las interpolaciones lineales (Ec.12), donde $HRIR_c[i]$ es la i-ésima muestra de la HRTF de partida, $HRIR_t[i]$ es la i-ésima muestra de la HRTF de destino, $HRIR_f[i]$ es la i-ésima muestra de la HRTF final y $Rate$ es la tasa de transformación (la misma debe ser constante para el ruido se reduzca*)*. La Ec. 12 debe ser aplicada para cada uno de los canales de las HRIRs.

$$f(x|x_2; x_1) = f(x) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}\,(x_2 - x_1) \tag{11}$$

$$HRIR_f[i] = HRIR_c[i] + (HRIR_t[i] - HRIR_c[i]) \cdot Rate \tag{12}$$

La Ec.12 debe ser aplicada para obtener las N muestras de una HRIR de temporal, hasta que la diferencia acumulada Ec.13 sea significativamente pequeña (un valor menor a 1 es aceptable).

$$Diferecia\ Acumulada = \sum_{i=0}^{N-1} |HRIR_t[i] - HRIR_c[i]| \tag{13}$$

Este procedimiento transformará de forma gradual la HRTF de partida en la HRTF de destino, reduciendo así el ruido.

1.1.7 Análisis de Fourier

Es posible realizar una optimización final al algoritmo de convolución segmentada llamado convolución rápida, pero para ello es necesario pasar la señal al dominio de la frecuencia utilizando transformadas de Fourier. El análisis de Fourier es una técnica matemática, ampliamente utilizada en el procesamiento de señales digitales. El mismo nos permite la descomposición de una señal en sinusoides, o en términos de procesamiento de señales permite la transformación de una señal en el domino del tiempo al dominio de la frecuencia (SMITH, Chapter 8: The Discrete Fourier Transform, 1998).

1.1.7.1 Transformada Discreta de Fourier

La transformada discreta de Fourier (DFT) nos permite procesar señales discretas, que contienen un número de muestras finitas en un espacio de tiempo y nos permite obtener señales complejas en el dominio de la frecuencia (SMITH, Chapter 8: The Discrete Fourier Transform, 1998).

La transformada discreta de Fourier está definida matemáticamente por la Ec. 14.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \dots, N-1 \tag{14}$$

Si aplicamos la identidad de números complejos de Euler podemos representar la parte real e imaginaria de la Ec. 14, en la Ec. 15 y 16. La señal $x[i]$ tanto en la Ec. 15 y 16, es la señal en el dominio del tiempo que está siendo analizada. $ReX[k]$ y $ImX[k]$ son las señales complejas transformadas al dominio de la frecuencia. El índice i puede tomar valores de 0 a $N-1$, mientras que el índice k puede tomar valores de 0 a $N/2$.

$$ReX[k] = \sum_{i=0}^{N-1} x[i] \cos(2\pi ki/N) \tag{15}$$

$$ImX[k] = -\sum_{i=0}^{N-1} x[i] \operatorname{sen}(2\pi ki/N) \tag{16}$$

1.1.7.2 Transformada Inversa de Fourier Discreta

La transformada de Fourier inversa (IDFT) nos permite sintetizar una señal al pasarla del dominio de la frecuencia al dominio del tiempo.

La transformada discreta inversa de Fourier está definida matemáticamente por la Ec. 17.

$$x[i] = \sum_{k=0}^{N/2} Re\bar{X}\,[k] \cos(2\pi ki/N) + \sum_{k=0}^{N/2} Im\bar{X}\,[k] \operatorname{sen}(2\pi ki/N) \tag{17}$$

Está ecuación también es conocida como ecuación de síntesis. La señal $x[i]$ es la señal que se está transformando del dominio de la frecuencia al dominio del tiempo, el índice i puede

tomar valores de 0 a $N-1$. $Re\bar{X}$ y $Im\bar{X}$ representan las amplitudes de los cosenos y senos, y su índice k puede tomar valores de 0 a $N/2$.

Aplicando la identidad de números complejos de Euler en la Ec. 17, podemos representarla en la Ec. 18.

$$x_n = \frac{1}{N}\sum_{n=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn} \qquad n = 0, \ldots, N-1 \tag{18}$$

1.1.7.3 Obtención de Frecuencias de una DFT

Una vez realizada una transformación de un bloque de una señal en el dominio del tiempo al dominio de la frecuencia, obtenemos un conjunto de números complejos. Donde el número de datos es igual a la mitad del tamaño del bloque por lo estipulado por el Teorema de Nyquist[3] (SMITH, Chapter 8: The Discrete Fourier Transform, 1998).

La resolución de la frecuencia (Hz) se obtienen con la Ec. 19.

$$Resolución\ de\ la\ Frecuencia = \frac{Frecuencia\ Nyquist}{Tamaño\ del\ Bloque} \tag{19}$$

Para obtener la frecuencia de un determinado índice del bloque, se debe multiplicar los índices del bloque (que corresponden a un número complejo) por el tamaño de la resolución de la frecuencia. Y la amplitud de cada frecuencia es obtenida calculando el módulo de cada número complejo (DIATKINE, 2011).

[3] **Teorema de Nyquist**: El teorema muestreo de Nyquist indica que una señal continua puede muestrearse adecuadamente, sólo si contiene componentes de frecuencia por debajo de la mitad de la frecuencia de muestreo. Por lo que la Frecuencia más alta muestreable es igual a la mitad de la frecuencia de muestreo y es conocida como Frecuencia de Nyquist (SMITH, The Sampling Theorem, 1998).

1.1.7.4 Transformada Rápida de Fourier

La transformada rápida de Fourier (FFT) es un algoritmo para calcular la transformada discreta de Fourier que reduce el número de cálculos necesarios de $2N^2$ a $2N \log_2 N$ (WEISSTEIN, 2017). Las primeras discusiones de las FFTs aparecen en 1965 con Cooley y Tukey, aunque Gauss ya había descrito los pasos críticos necesarios para la factorización en 1805 (BERGLAND, 1969).

La FFT o transformada rápida de Fourier es un algoritmo que utiliza números complejos para reducir el número de multiplicaciones necesarias. La FFT descompone recursivamente N veces una señal que se encuentra en el dominio del tiempo, en una parte par y otra impar. Obteniendo N señales de un punto cada una (Fig. 13). Para que este algoritmo funcione es preciso procesar señales que su tamaño pueda ser representada por una potencia de dos (SMITH, Chapter 12: The Fast Fourier Transform, 1998).

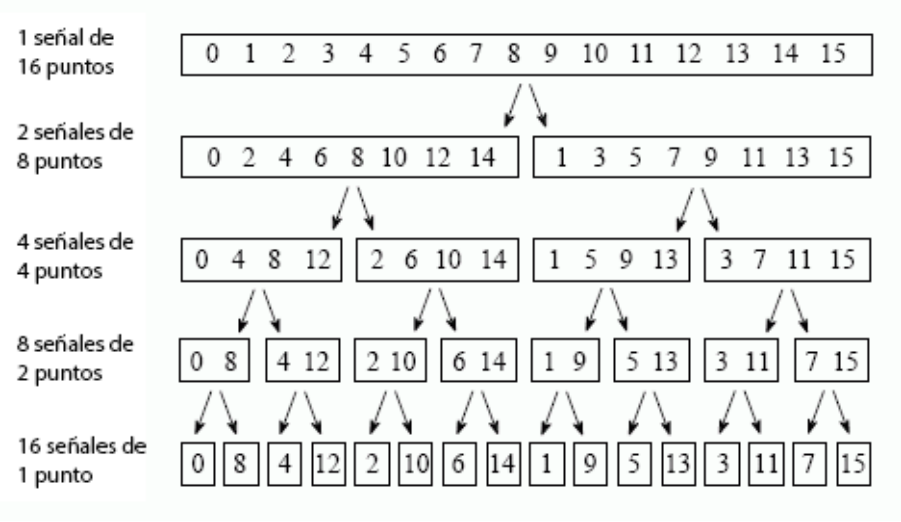


Figura 13: Descomposición de una FFT (SMITH, Chapter 12: The Fast Fourier Transform, 1998)

Una vez que la señal se encuentra descompuesta hay que encontrar el contenido espectral de las señales formadas por un solo punto, lo que es muy sencillo ya que el contenido espectral es el mismo punto solo que ahora es representado por un número complejo.

Ahora hay que combinar el contenido espectral de las N frecuencias en el mismo orden que se descompusieron. Tomando en cuenta la Fig. 13 los espectros de las 16 frecuencias de un

punto son sintetizadas en 8 frecuencias de dos puntos, y en la siguiente etapa serán sintetizadas en 4 frecuencias de cuatro puntos cada una; y así sucesivamente hasta obtener un espectro de frecuencias de 16 puntos.

La Figura 14 muestra cómo se combinan dos espectros de frecuencia en uno solo de 8 puntos, el operador (x S) indica que se multiplica el punto por una sinusoide. La Fig. 15 muestra el algoritmo básico de una FFT conocido como “mariposa” que toma dos puntos complejos de entrada y produce dos puntos complejos de salida.

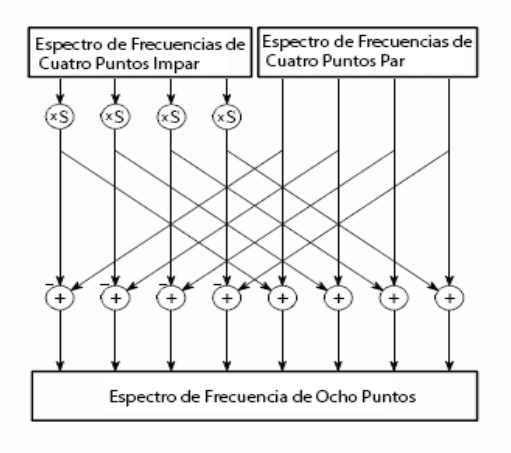


Figura 14: Diagrama de Síntesis de una FFT (SMITH, Chapter 12: The Fast Fourier Transform, 1998)

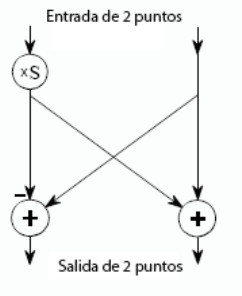


Figura 15: Diagrama de Mariposa de una FFT (SMITH, Chapter 12: The Fast Fourier Transform, 1998).

1.1.8 Convolución Rápida

La convolución rápida utiliza el teorema de la convolución (Ec.20), que plantea que la multiplicación en el dominio de la frecuencia corresponde a la convolución en el dominio del tiempo (SMITH, Chapter 18: FFT Convolution, 1998).

$$x[k] * h[k] = X[w]H[w] \tag{20}$$

Para usar este teorema la señal de entrada es transforma al dominio de frecuencia usando transformadas de Fourier, luego es multiplicada por la respuesta de frecuencia del filtro, y es transformada nuevamente al dominio del tiempo utilizando la transformada de Fourier inversa (Fig. 16).

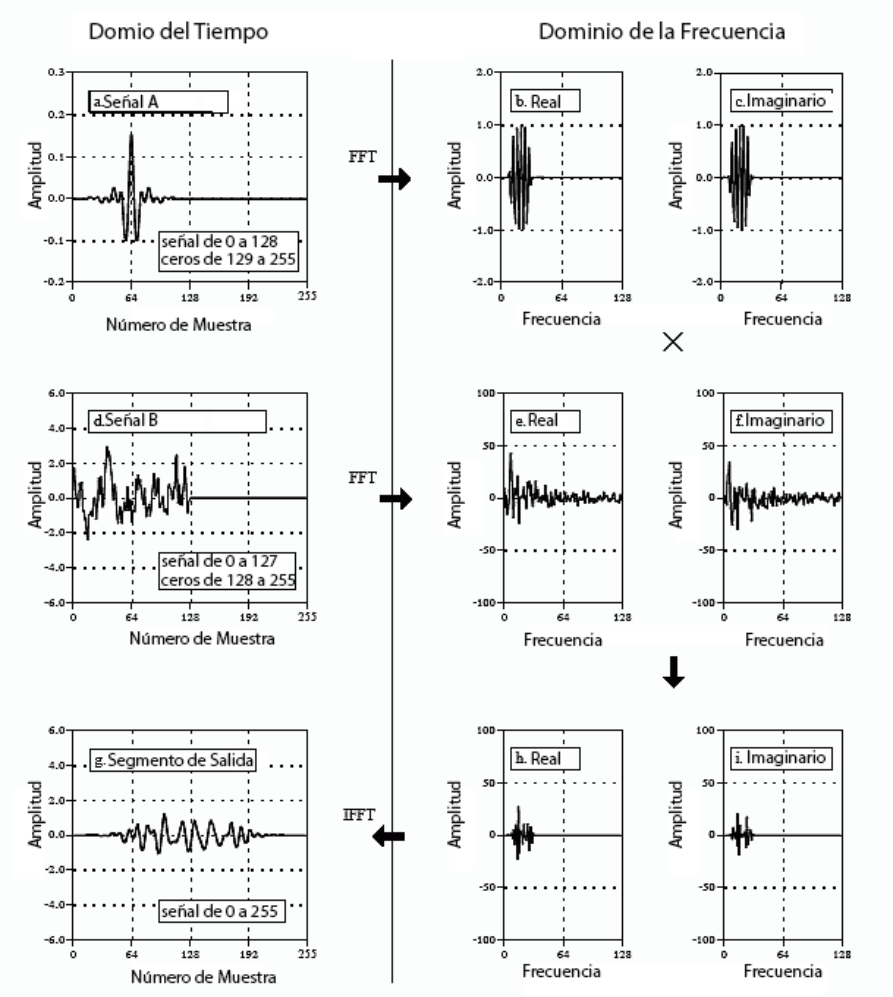


Figura 16: Convolución Rápida (SMITH, Chapter 18: FFT Convolution, 1998)

1.2. Consideraciones Teóricas Gráficas y Físicas

El ambiente 3D permite la generación y simulación del ambiente virtual para la interacción del usuario con las fuentes sonoras. El sistema permite la navegación del usuario dentro del ambiente virtual mediante una cámara en primera persona (perspectiva en primera persona) utilizando el mouse y teclado. Por otro lado, el sistema de interacción físico nos permite determinar la dirección relativa de la fuente sonora con respecto al receptor biauricular.

1.2.1 Representación de las Fuentes y Receptores

Para el cálculo del azimut y elevación del ambiente virtual (que son necesarios para el motor de aurilización) es necesaria una representación con algún tipo de dato o clase para el receptor, emisor y los obstáculos. Los emisores fueron modelados como puntos en el espacio y el receptor es representado por una esfera que tiene una unidad espacial virtual de radio. Finalmente, los obstáculos fueron modelados como conjuntos de triángulos (Fig. 17 (b)), lo que permite usar algoritmos eficientes de colisión para revisar si existe oclusión entre el rayo sonoro con algún obstáculo.

1.2.2 Sistema de Interacción Físico

El sistema de interacción físico determina un rayo que parte desde el centro del receptor hacia el centro del emisor. Luego se revisa si existe colisión entre el rayo y algún triangulo componente de algún obstáculo de la escena utilizando el algoritmo de Moller–Trumbore (1997).

Si existe colisión con algún triángulo, y la distancia de colisión es menor que la distancia del emisor al receptor, se considera que existe oclusión por lo que no se podría escuchar la fuente. Caso contrario se activa el generador de audio 3D, tal como lo indica la Fig. 17 (a).

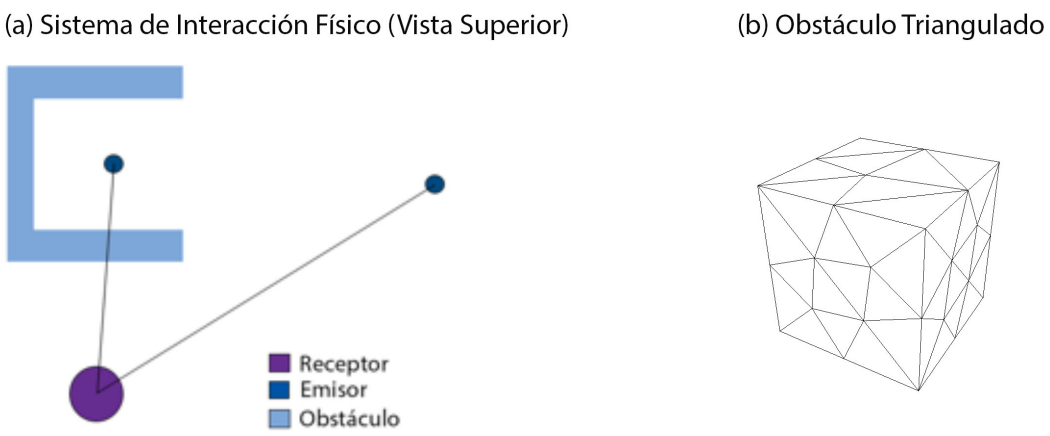


Figura 17: Sistema de Interacción Físico (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017)

Al no existir oclusión, se transforma la dirección de llegada del rayo al sistema de coordenadas del receptor (dependiente del movimiento de traslación y rotación provocado por el usuario usando el mouse y teclado). Para esto se utiliza matrices de rotación (Ec. 21 y 22) alrededor de los ejes *x* y *y* (LUCIO NARANJO, 2014). Hay que tomar en cuenta que las rotaciones se realizan en los ejes utilizando el estándar de OpenGL que se muestra en la Fig. 18.

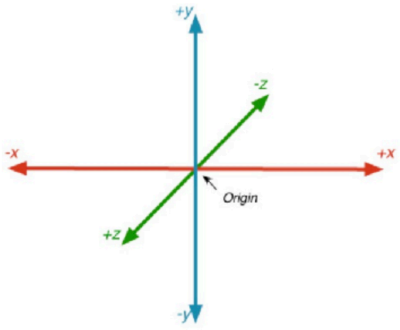


Figura 18: Ejes Estándar OpenGL (KHOLODOV, 2017)

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\text{sen}(\theta) \\ 0 & \text{sen}(\theta) & \cos(\theta) \end{bmatrix} \tag{21}$$

$$R_x(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \text{sen}(\theta) \\ 0 & 1 & 0 \\ -\text{sen}(\theta) & \text{sen}(\theta) & \cos(\theta) \end{bmatrix} \tag{22}$$

A continuación, se transforman las coordenadas cartesianas a polares utilizando las Ec. 23 y 24 (como se ilustra en la Fig. 19) para que esta sea compatible con la base de datos de HRIRs utilizada (GARDNER & MARTIN, 1994). Con esta información, se realiza la búsqueda del par de HRIRs en la base de datos más cercanos a la dirección de incidencia del rayo sobre el receptor. Para terminar el procesamiento previo al llamado del motor de aurilización, se calcula una ganancia, la cual servirá para simular la disipación del sonido dependiente de la distancia con la fuente sonora. Esta ganancia es inversamente proporcional al volumen de una esfera cuyo radio es el módulo del vector que va desde la fuente hacia el receptor. Este procedimiento se realiza cada vez que un nuevo cuadro de vıdeo en la escena se renderiza (como se ilustra en el Algoritmo 1).

$$\theta = \text{arcsen}\left(\frac{z}{\sqrt[2]{x^2 + y^2 + z^2}}\right)\frac{180}{\pi} \tag{23}$$

$$\phi = \arctan\left(\frac{x}{\text{y}}\right)\frac{180}{\pi} \tag{24}$$

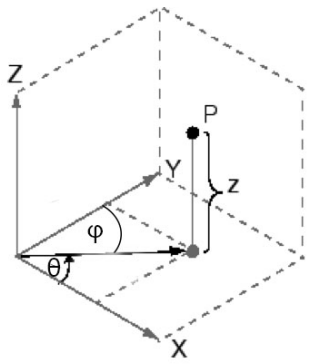


Figura 19: Transformación Coordenadas Cartesianas a Polares (MATHWORKS, 2017)

```
while no finalice el programa do
    Determinar un rayo desde el emisor hasta el centro
     del receptor;
    if no existen colisiones or módulo del rayo < que la
     distancia emisor-triángulo intersecado then
        Trasladar rayo al sistema de referencia del
         receptor;
        Transformar a coordenadas polares;
        Buscar HRIR en la base de datos;
        Calcular ganancia dependiendo de la distancia del
         emisor al receptor;
    else if Existe oclusión then
        Bloquear sonido de la fuente;
end
```

Algoritmo 1: Sistema de Interacción Físico (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017)

1.2.3 Intersección Rayo-Triángulo

Para el cálculo de las intersecciones se utilizó el algoritmo de intersección de Möller-Trumbore (1997). Este es considerado como un algoritmo rápido por lo que es usado para comparar el desempeño de otros métodos de intersección. El mismo hace extenso uso del algebra lineal y geometría analítica (PRUNIER, 2016).

Adicionalmente aprovecha la propiedad del baricentro P, que afirma que ninguna transformación (rotación, escala o traslación) realizada al triangulo afectará el punto de intersección expresado en coordenadas baricéntricas (lo que no sucedería expresando el punto de intersección en sistema cartesiano de referencia). El algoritmo hace uso de la ecuación paramétrica de las coordenadas baricéntricas (Ec. 25), que permiten representar cualquier punto dentro del triángulo como se muestran la Fig. 20.

$$\vec{P} = w\vec{A} + u\vec{B} + v\vec{C} \tag{25}$$

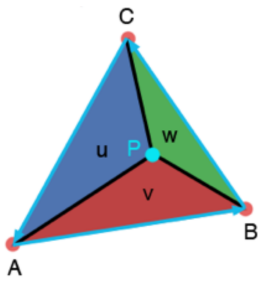


Figura 20: Coordenadas Baricéntricas (PRUNIER, 2016)

Usando la propiedad del baricentro Ec. 26, se puede remplazar el término *w* de la Ec. 25 para obtener la Ec. 27 (PRUNIER, 2016).

$$1 = w + u + v \tag{26}$$

$$\vec{P} = \vec{A} + u\left(\vec{B} - \vec{A}\right) + v\left(\vec{C} - \vec{A}\right) \tag{27}$$

Más tarde se puede utilizar la ecuación paramétrica del rayo (Ec.28) para igualar la Ec. 27 y 28, obteniendo así la Ec. 29.

$$\vec{P} = \vec{O} + t\vec{D} \tag{28}$$

$$\vec{O} - \vec{A} = -t\vec{D} + u\left(\vec{B} - \vec{A}\right) + v\left(\vec{C} - \vec{A}\right) \tag{29}$$

La Ec. 29 se puede representar de la forma matricial de la Ec. 30, y la misma es resuelta usando el método de Crammer[4] para encontrar los valores de t, u y v.

$$\left[-\vec{D} \quad (\vec{B} - \vec{A}) \quad (\vec{C} - \vec{A})\right] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \vec{O} - \vec{A} \tag{30}$$

Finalmente para revisar si existe intersección se revisa que los valores de t, u y v, no sean negativos y cumplan con lo estipulado por la Ec. 26. Encontrando así el punto de intersección.

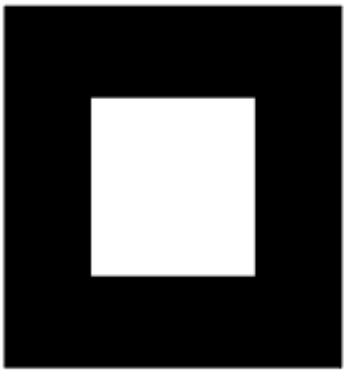
1.2.4 OpenGL

OpenGL fue el API[5] escogido para la generación de gráficos tridimensionales, el mismo facilita la comunicación con la tarjeta gráfica. El API consiste en más de 700 funciones que se usan para especificar los objetos y operaciones necesarias para producir aplicaciones con gráficos tridimensionales interactivos (SHEREINER, 2010).

OpenGL está diseñado para ser independiente del hardware y del sistema operativo. Por lo que no contiene funciones para la creación de ventanas ni el uso de dispositivos de entrada, por lo que hay que para esto hay que utilizar funciones dependientes de cada sistema operativo. Igualmente, OpenGL no provee funciones de alto nivel para describir modelos

[4] **Regla de Crammer**: Dado un sistema de ecuaciones lineales, la regla de Crammer nos permite encontrar la solución a una sola variable sin tener que resolver el sistema completo (STAPEL, 2012).
[5] **API**: "(Application Programming Interface) es un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas" (MERINO, 2014).

tridimensionales, todo debe ser creado usando un conjunto de primitivas geométricas como puntos líneas y polígonos (SHEREINER, 2010) . OpenGL por lo general se utiliza como API del lenguaje de programación C aunque existen *bindings*[6] para otros lenguajes como Java. La Fig. 21 ilustra un ejemplo sencillo de OpenGL, y contiene el código necesario para crear el cuadrado con blanco y fondo negro.



```
main() {
   InitializeAWindowPlease();

   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f(0.25, 0.25, 0.0);
      glVertex3f(0.75, 0.25, 0.0);
      glVertex3f(0.75, 0.75, 0.0);
      glVertex3f(0.25, 0.75, 0.0);
   glEnd();
   glFlush();
   UpdateTheWindowAndCheckForEvents();
}
```

Figura 21: Salida del programa de OpenGL y el código respectivo (SHEREINER, 2010).

1.2.5 Tipos de Primitivas Geométricas de OpenGL

OpenGL soporta tres clases de primitivas geométricas: puntos, segmentos de línea y polígonos cerrados. Todas estas primitivas geométricas se especifican mediante vértices, cada vértice tiene asociado atributos como: posición, color, vector normal y textura (HOCK-CHUAN, 2012). OpenGL provee en total 10 primitivas como se puede observar en la Fig. 22.

[6] **Bindings**: Se refiere a la asociación entre dos o más lenguajes de programación (TECHTARGET, 2005).

Para renderizar primitivas geométricas debemos utilizar la función *glBegin*. La función debe llevar como argumento el tipo de primitiva geométrica con la que queremos renderizar, seguido de la lista de vértices que queremos añadir (OOSTEN, 2012). Para finalizar la definición de una de estas primitivas debemos llamar a la función *glEnd*. Por facilidad los modelos (OBJ) de los obstáculos y fuentes del ambiente tridimensional se renderizaron utilizando GL_TRIANGLES.

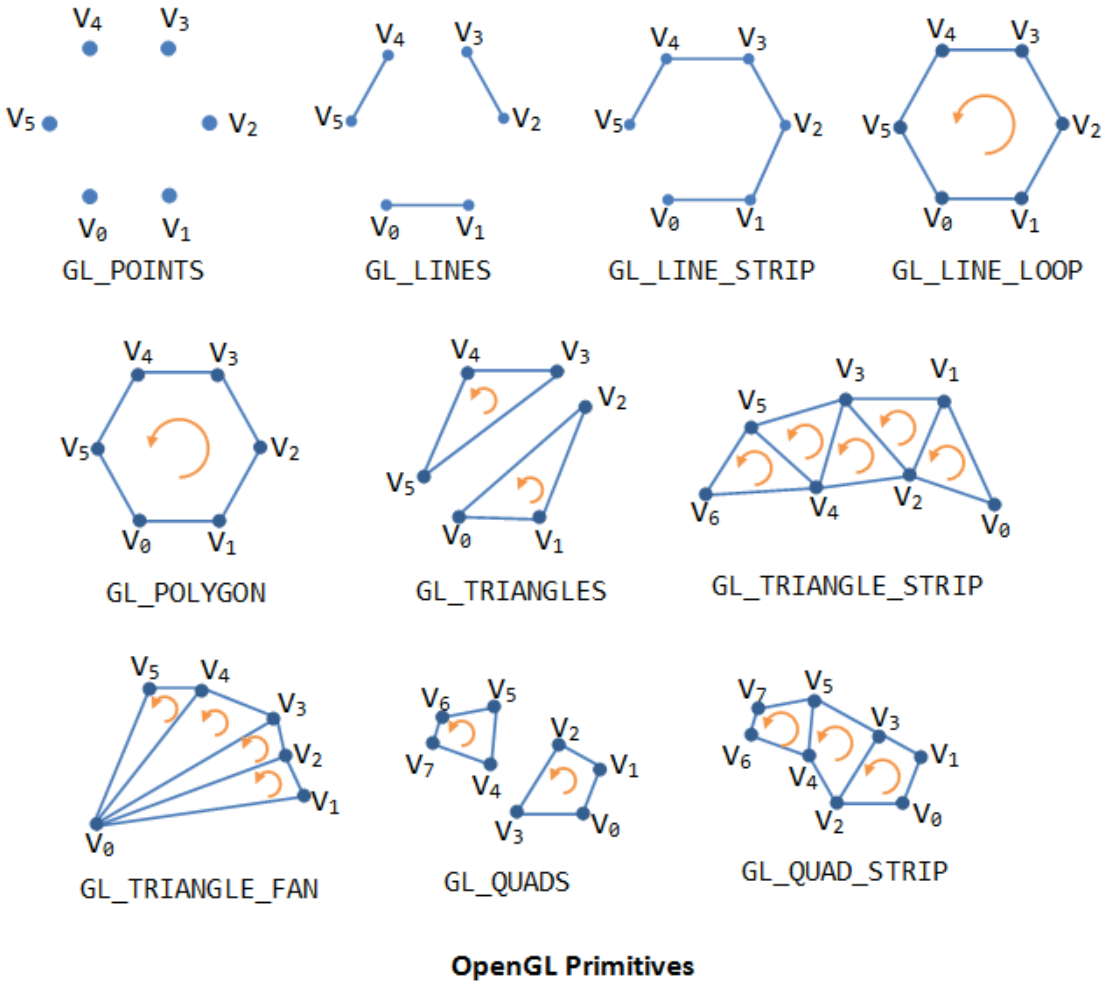


Figura 22: Primitivas Geométricas OpenGL (HOCK-CHUAN, 2012)

1.2.6 Modelos Tridimensionales OBJ

Para la representación de los obstáculos y fuentes se utilizó el formato OBJ los cuales son archivos de texto simples que se componen de triángulos (Fig. 23). Los archivos OBJ definen la geometría y otras propiedades de un modelo tridimensional creado por Wavefront Technologies (SCULPTELO, 2017).

La geometría poligonal de estos archivos utiliza puntos, líneas y caras para definir un objeto mientras que la geometría de forma libre usa curvas y superficies (BOURKE, 2012). Un archivo OBJ viene acompañado de un archivo MTL el cual posee la información de textura, color y material (SCULPTELO, 2017)

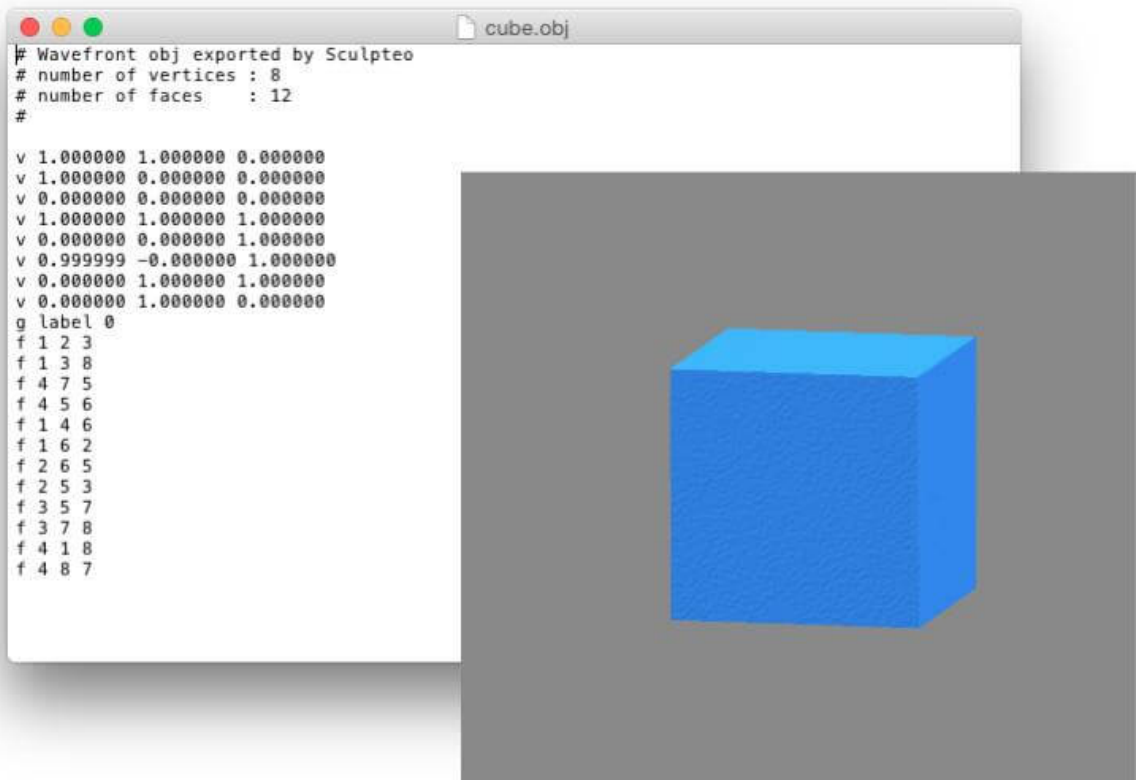


Figura 23: Archivo OBJ y su representación 3D (SCULPTELO, 2017)

1.2.7 Sistema de Coordenadas de OpenGL

Dentro de OpenGl se utilizan varios sistemas de coordenadas. La ventaja que nos brinda esto es que varios cálculos son mucho más fáciles de realizar en un sistema de coordenadas específico (DE VRIES, Coordinate Systems, 2016). Existen un total de cinco sistemas de coordenadas diferentes (Fig. 24):

- Espacio Local (*Local Space* o *Object Space*)
- Espacio del Mundo Virtual (*World Space*)
- Espacio de Visión (*View Space* o *Eye Space*)
- Espacio de Recorte (*Clip Space*)
- Espacio de la Pantalla (*Screen Space*)

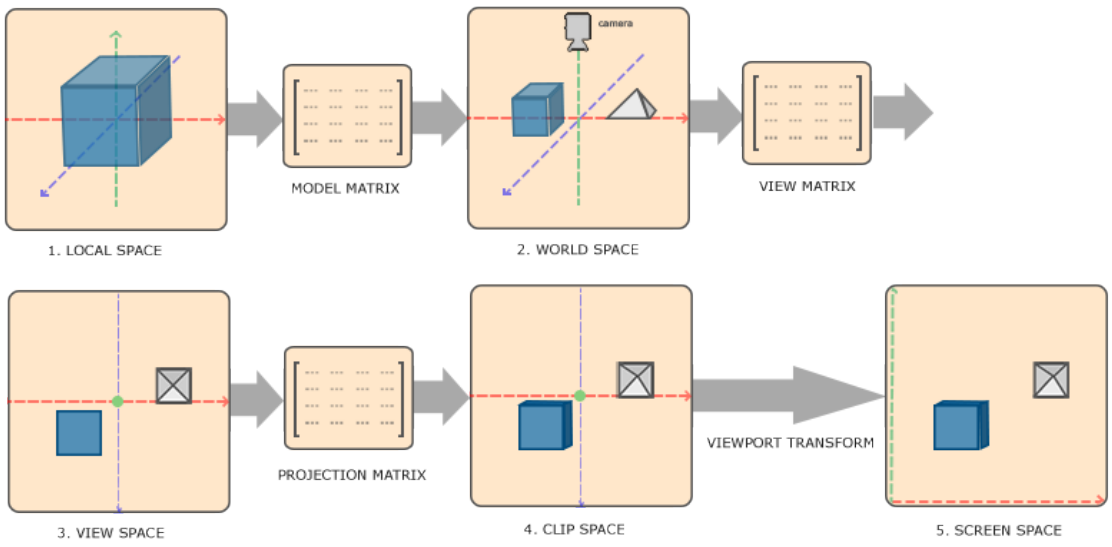


Figura 24: Transformaciones de Sistemas de Coordenadas OpenGL (DE VRIES, Coordinate Systems, 2016)

Para transformar las coordenadas de un sistema a otro e deben utilizar varias matrices de transformación. Las matrices más importantes son: de modelo, de vista y de proyección. El proceso de transformación según (DE VRIES, Coordinate Systems, 2016) se describe a continuación:

- Se definen las coordenadas locales que son relativas al origen del objeto, son las coordenadas en las cuales el objeto empieza.
- El siguiente sistema de coordenadas a las que se debe transformar es relativo a las coordenadas locales del mundo. Este sistema de coordenadas es relativo al origen global del mundo.
- La siguiente transformación se debe realizar en el sistema de coordenadas de visión (*view-space*) de esta manera obtenemos las coordenadas como si las estuviéramos observando desde el sistema de visión de la cámara.
- Después se procesan las coordenadas de visión en un rango de -1 a 1 y se determina que vértices van a ser visibles en la pantalla.
- Finalmente, las coordenadas de recorte se transforman a las coordenadas de la ventana de OpenGl y se envían al rasterizador[7] para obtener la imagen plana resultante.

1.2.8 Cámara de OpenGL

La cámara depende de 4 parámetros que son los siguientes y los podemos observar en la Fig. 25:

- **Posición de la cámara (*Position*):** La posición de la cámara es el vector en el mundo virtual que apunta a la posición donde se encuentra la cámara.
- **Dirección de la cámara (*Direction*):** El siguiente vector que se requiere es la dirección de la cámara. Este vector indica a donde apunta la cámara. Por ejemplo, en la Fig. 25 la cámara apunta hacia el origen de la escena. Para obtener el vector de dirección basta con restar dos vectores, por ejemplo, en el caso de la Fig. 25 se debe restar el vector de la posición del origen de la escena.

[7] **Rasterizador**: "El rasterizador determina cómo representar datos 3D tales como posición, color y textura en una superficie 2D" (MICROSOFT, ¿Qué es el estado de rasterizador?, 2017).

- **Eje derecho (*Right*):** El siguiente vector que necesitamos es el que representa el eje positivo x del espacio vectorial de la cámara. Para obtener el eje derecho necesitamos especificar un vector normal paralelo al eje *y* del mundo virtual. Después debemos realizar un producto cruz entre la dirección de la cámara y el vector del paso anterior. Una vez realizado esto tenemos un vector perpendicular al eje y de la escena y al vector de dirección que apunta en la dirección del eje x positivo del espacio vectorial de la cámara.
- **Eje Normal (*Up Axis*):** Conociendo el vector x y z del espacio vectorial de la cámara ahora podemos obtener el eje normal (y) del espacio vectorial de la cámara. Para esto solo debemos realizar un producto cruz entre x y z.

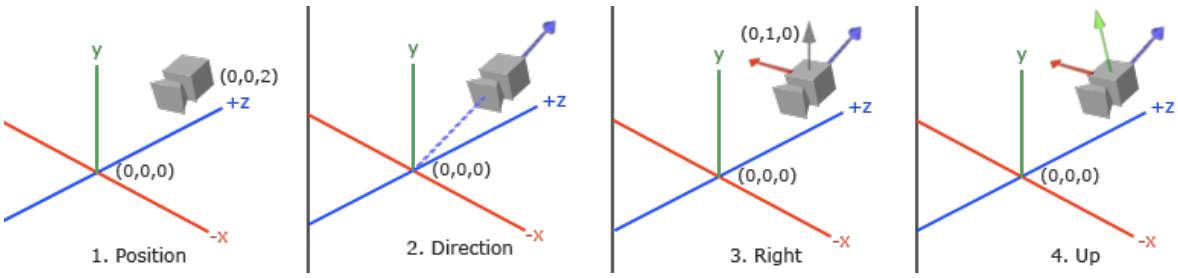


Figura 25: Parámetros Necesarios para la Cámara (DE VRIES, Camera, 2016)

1.2.9 Transformación LookAt

La transformación *LookAt* es fundamental para la simulación de la cámara, como se describirá la sección Movimiento de la Cámara. Si tenemos definido un sistema de coordenadas por 3 ejes perpendiculares. Podemos crear una matriz (Ec. 31) (donde R es el vector *right*, U es el vector *up*, D es el vector *direction* y, P es el vector *position* de la Fig. 25) que contenga los 3 ejes, la cual podremos transformar a cualquier sistema de coordenadas espaciales multiplicándolo por un vector de traslación (DE VRIES, Camera, 2016).

$$LookAt = \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (31)$$

El vector de posición de la matriz de traslación tiene sus coordenadas negativas, ya que se requiere mover el mundo virtual en la dirección contraria a la que nos estamos moviendo. La función *LookAt* en pocas palabras crea una matriz de vista que mira a un determinado objetivo.

1.2.10 Movimiento de la Cámara

Dentro de OpenGL no existe un concepto de cámara, pero se puede simular el efecto moviendo los objetos de la escena en dirección contraria (DE VRIES, Camera, 2016). Generando así el efecto de que nos movemos, para esto se deben realizar operaciones con los vectores de la Fig. 25, representados por los vectores de las Ec. 32, 33 y 34.

$$\overrightarrow{Pos} = 2\vec{k} \quad (32)$$

$$\overrightarrow{Front} = -\vec{k} \quad (33)$$

$$\overrightarrow{Up} = \vec{j} \quad (34)$$

La función *lookAt* de openGL toma como argumentos tres vectores: la posición de la cámara (*position*), posición a la que mira (*center,* que se obtiene con la Ec. 35) y el vector normal (*up*). Para calcular la matriz LookAt (Ec. 31), la función internamente calcula el vector $\overrightarrow{Direction}$ con la Ec. 36 y el vector $\overrightarrow{Right}$ usando la Ec. 37 (KRHONOS GROUP, 2011).

$$\overrightarrow{Center} = \overrightarrow{Pos} + \overrightarrow{Front} \quad (35)$$

$$\overrightarrow{Direction} = \overrightarrow{Center} - \overrightarrow{Pos} \tag{36}$$

$$\overrightarrow{Right} = \overrightarrow{Direction} \times \overrightarrow{Up} \tag{37}$$

Si se define una velocidad para la cámara se la puede multiplicarla por el vector $\overrightarrow{Front}$ y sumarla al vector inicial $\overrightarrow{Pos_o}$, obtendremos un nuevo vector $\overrightarrow{Pos_f}$ (Ec. 38) que dará el efecto de movimiento de la cámara hacia delante al usarlo con la matriz *LookAt*. Para recrear el efecto de movimiento hacia atrás basta con cambiar el procedimiento anterior por una resta (Ec. 39).

En cambio, para realizar un movimiento a la izquierda o a la derecha, debemos encontrar el vector perpendicular al vector $\overrightarrow{Front}$ y $\overrightarrow{Up}$ multiplicarlo por la velocidad de la cámara. Luego sumarlo o restarlo dependiendo del movimiento que se requiera realizar (Ec. 40 y 41, tomando en cuenta que el operador $\widehat{\ }$ es el vector unitario).

$$\overrightarrow{\mathrm{Pos_f}} = \overrightarrow{\mathrm{Pos_o}} + \left(\overrightarrow{\mathrm{Front}} \cdot \mathrm{Speed}\right) \tag{38}$$

$$\overrightarrow{\mathrm{Pos_f}} = \overrightarrow{\mathrm{Pos_o}} - \left(\overrightarrow{\mathrm{Front}} \cdot \mathrm{Speed}\right) \tag{39}$$

$$\overrightarrow{\mathrm{Pos_f}} = \overrightarrow{\mathrm{Pos_o}} + \left(\widehat{\overrightarrow{\mathrm{Front}} \times \overrightarrow{\mathrm{Up}}}\right) \mathrm{Speed} \tag{40}$$

$$\overrightarrow{\mathrm{Pos_f}} = \overrightarrow{\mathrm{Pos_o}} - \left(\widehat{\overrightarrow{\mathrm{Front}} \times \overrightarrow{\mathrm{Up}}}\right) \mathrm{Speed}\ (41) \tag{41}$$

1.2.11 Movimientos Rotacionales de la Cámara

Para una experiencia más parecida a la que se tiene en el mundo real no basta con mover la cámara para adelante, atrás y los lados. Sino que también se deben implementar las rotaciones de la cámara. Para realizar rotaciones se debe cambiar el vector *cameraFront* dependiendo del movimiento del mouse, para esto se debe tomar en cuenta los tres ángulos de Euler (*pitch*, *raw* y *roll,* ver Fig. 26) (DE VRIES, Camera, 2016).

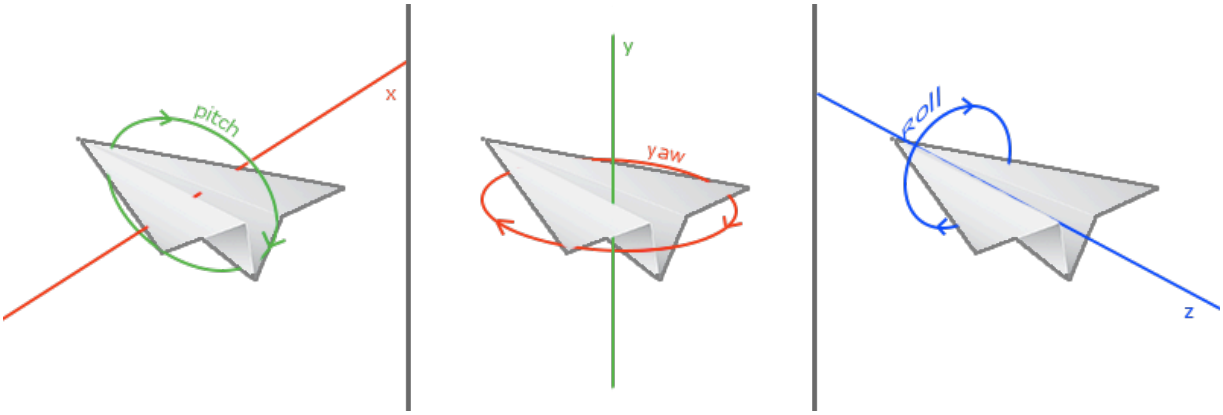


Figura 26: Ángulos de Euler (DE VRIES, Camera, 2016)

Para la cámara en primera persona solo son necesarios los valores de *pitch* y *yaw*. Para realizar las transformaciones de rotación se deben utilizar funciones trigonométricas. Ya que el ángulo *pitch* se encuentra en el plano (*x/z*)-*y* (como se puede observar el la Fig. 27) usando trigonometría básica podemos encontrar los nuevos valores del vector dirección ($\overrightarrow{Direction}$) rotados de *x*, *y* y *z* usando las Ec. 42, 43 y 44.

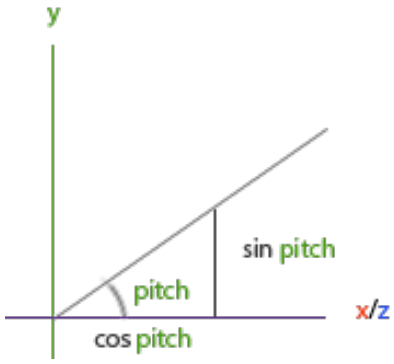


Figura 27: Plano Y- X/Z (DE VRIES, Camera, 2016)

$$\mathrm{x} = \cos(\mathrm{pitch}) \tag{42}$$

$$\mathrm{y} = \mathrm{sen}(\mathrm{pitch}) \tag{43}$$

$$\mathrm{z} = \cos(\mathrm{pitch}) \tag{44}$$

Finalmente se puede observar en la Fig. 28 que el eje *x* y *z* son alterados por la rotación del ángulo *yaw*. Por lo que al resultado de la transformación del ángulo pitch se debe multiplicar por las transformaciones del ángulo *yaw*. Utilizando trigonometría podemos obtener las nuevas coordenadas finales del vector dirección ($\overrightarrow{Direction}$) ya transfomada por el ángulo *pitch* y *yaw* (Ec. 45, 46 y 47).

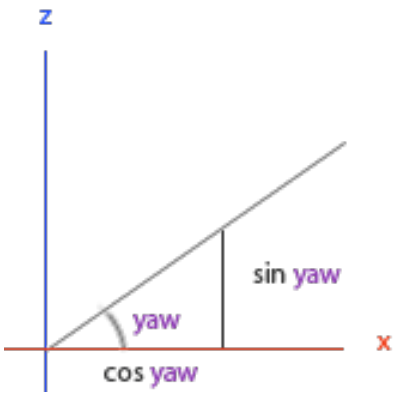


Figura 28: Plano Z- X (DE VRIES, Camera, 2016)

$$x = \cos(pitch)\cos(yaw) \tag{45}$$

$$y = \mathrm{sen}(pitch) \tag{46}$$

$$z = \cos(pitch)\,\mathrm{sen}(yaw) \tag{47}$$

CAPÍTULO II: METODOLOGÍA

En el presente capítulo se van a detallar todo lo requerido para la implementación de la aplicación. Por lo que es fundamental definir inicialmente una metodología de desarrollo como se detalla en la sección Análisis, la misma servirá de guía para el resto de fases del proyecto. Cada una de estas fases serán descritas en sus secciones respectivas:

- Análisis
- Diseño
- Implementación
- Pruebas

2.1 Análisis

En la presente sección se va a realizar el análisis previo a la implementación de la aplicación. Para esto se va definir los requerimientos funcionales del software además de la metodología, herramientas de desarrollo, diagramas de diseño y normas a seguir. Igualmente se presentarán los entornos de desarrollo que se van a usar además del lenguaje de programación y *framework* que se utilizaron. Finalmente, esta sección terminará con el diseño de la interfaz de usuario con el que se construyó la aplicación.

2.1.1 Metodología de Desarrollo Incremental

Una metodología de desarrollo es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas informáticos. Al ser un proyecto de investigación donde existía una gran complejidad en los requerimientos del proyecto se optó por escoger la metodología de desarrollo incremental. Esta trata de bajar los riesgos en el desarrollo de un proyecto, dividiendo el desarrollo del mismo en mini-cascadas. Cada mini-cascada completa una parte pequeña del proyecto antes de seguir con el siguiente incremento del mismo. En la metodología incremental se define el concepto inicial de software, requerimientos, análisis y arquitectura son definidos utilizando una metodología de cascada seguidas de un proceso iterativo de prototipado que termina con la culminación del proyecto (CMS INFORMATION TECHNOLOGY, 2008).

2.1.1.1 Fases de la Metodología de Mini-Cascada

En una metodología de cascada el ciclo de vida abarca las siguientes fases (Fig. 29) (BLE, 2013):

Análisis del Sistema: se establecen los requerimientos de todos los elementos del sistema.

Diseño: el diseño del software se enfoca en cuatro aspectos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. Este proceso traduce los requisitos en una representación del software antes de que comience la implementación del mismo.

Implementación: el diseño se traduce en código, que se transforma más tarde en un ejecutable.

Pruebas: una realizada la implementación se realizan las pruebas del programa. La prueba se centra en la lógica interna del software y en las funciones externas. Las pruebas aseguran que una entrada definida produce los resultados que realmente se requieren.

Mantenimiento: el software sufrirá cambios después de que se entrega al cliente, debido a diferentes factores por lo que es indispensable realizar un seguimiento del mismo para afrontar los cambios sin impactos negativos en el software.

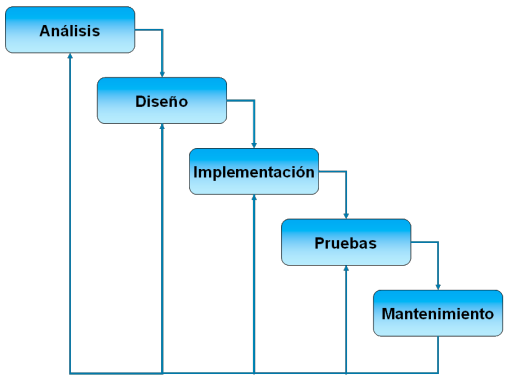


Figura 29: Metodología Cascada (GONZÁLEZ, 2006)

2.1.2 Análisis de Requerimientos Funcionales

El objetivo principal del desarrollo de este proyecto es implementar un módulo de aurilización en tiempo real, el mismo cumple la función de dar posicionamiento al sonido de una fuente virtual mediante el uso de una base de datos de funciones HRIR. Además, se creará un ambiente virtual, que mediante un mouse y un teclado permitirá al usuario interactuar con las fuentes virtuales del módulo de aurilización. La programación en tiempo real de audio debe seguir buenas prácticas y normas de desarrollo, para que funcione de manera correcta como se describen más tarde en este capítulo.

2.1.3 Licencia y Nombre del Proyecto

Este proyecto se liberará bajo una licencia de Software Libre, en un repositorio público de Github (una plataforma de desarrollo colectivo) bajo el nombre de Auris (proviene del latín y significa oreja). El ícono (Fig. 30) fue diseñado por el artista gráfico Diego Carvajal ((CARVAJAL, 2017)) del colectivo artístico KuskinaMat (KUSKINA MAT, 2017).

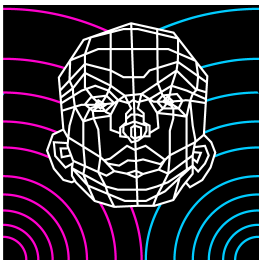


Figura 30: Ícono de Auris

2.1.4 Iteraciones Realizadas

Cada una de las siguientes iteraciones incrementales fueron realizadas, en el orden indicado abajo. Cada iteración incrementa la funcionalidad del proyecto, hasta completarlo con la última iteración.

- **Convolución Segmentada:** se integró FFTConvolver un proyecto de software libre al motor de audio de JUCE.
- **Convolución Segmentada HRIRs:** se añadió la posibilidad de cambiar las respuestas impulsivas en tiempo real en el proyecto FFTConvolver, para usarlas con las HRIRs obtenidas de la Librería en C de HRTFs Kemar.
- **CrossFader-Filtros IRR:** se implementó el reductor de ruido, además se añadieron filtros para no procesar las señales no direccionales.
- **Ambiente 3D:** se implementó un ambiente 3D basado en el tutorial de DigiBen de GameTutorials, y se retiraron todas las dependencias del API de Windows cambiándolas por funciones de JUCE multiplataforma.
- **Sistema Interacción Física:** se implementaron todos los cálculos necesarios para obtener la posición de la fuente relativa al receptor, además del control de volumen que cambia con la distancia. Se usaron las matrices de rotación del proyecto Ray Tracer de Song Ho Ahn / Shipeng Xu.
- **Sincronización de Hilos:** se utilizaron variables atómicas con el azimut y elevación para poder compartir los datos entre hilos de manera segura.
- **Algoritmos de Oclusión (Intersección Möller -Trumbore):** se utilizó la librería Tiny Obj, para cargar los obstáculos y fuentes en el ambiente virtual. Además de la implementación del algoritmo de Möller –Trumbore para la simulación de oclusión.

2.1.5 Herramientas Necesarias

Para el desarrollo de la aplicación fueron necesarias varias herramientas como el lenguaje de programación, el framework y entornos de desarrollo. A continuación, se presentan las herramientas seleccionadas.

2.1.5.1 Lenguaje de Programación C++

El lenguaje escogido fue C++, el mismo fue creado por Bjarne Stroustrup a mediados de los años 80's. C++ debido a su rapidez y poder es una estándar en la industria de los gráficos computacionales y audio. Fue diseñado para proveer las facilidades que brindaba Simula con la eficiencia y flexibilidad del lenguaje de programación C. Actualmente la última versión estable de C++ es la 11, y es usado en casi cualquier lugar incluyendo computadoras, teléfonos, autos y cámaras. C++ es un lenguaje de programación de nivel medio de abstracción por lo que por lo general es usado en funciones muy cercanas al "kernel"[8] del sistema operativo donde la mayoría de usuarios y programadores no llegan (STROUSTRUP, 2013).

C++ es un lenguaje compilado. Para que un programa se ejecute, su código fuente debe ser procesado por un compilador que produce archivos de tipo objeto, que son combinados por el "linker"[9] para crear un programa ejecutable (como se puede observar en el Fig. 31) (STROUSTRUP, 2013).

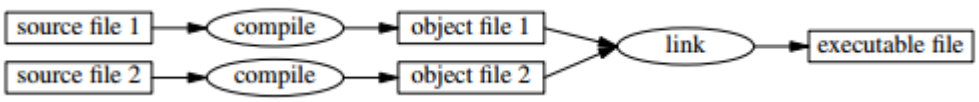


Figura 31: Proceso de generación de un ejecutable en C++ (STROUSTRUP, 2013)

[8] **Kernel:** "Se puede definir como el corazón del sistema operativo" (DE LUCAS, 2004).
[9] **Linker**: En un compilador es el encargado de unir múltiples archivos objeto. (STROUSTRUP, 2013).

2.1.5.2 JUCE Framework

JUCE (Jules Utility Class Extension) es el framework[10] escogido de C++ para el desarrollo. Fue creado por Julian Storer y derivó de la creación de varias aplicaciones de audio como Traktion (Fig. 32), por lo que cuenta con gran soporte para aplicaciones de audio y gráficos 2D y 3D. El código de la librería fue abierto en 2003 y en 2005 empezó a ser comercializada por Raw Material Software Ltd., donde Jules era el dueño y único desarrollador. En 2014 ROLI Ltd. adquirió Raw Material Software y JUCE (ROLI Ltd., 2017).

JUCE es una librería multiplataforma soporta Windows, Mac OS, Linux, iOS y Android, posee una herramienta para administrar proyectos por lo que facilita la compilación de los mismos. Permite crear interfaces de usuario versátiles y permite la integración de las mismas con OpenGL. JUCE además tienen una gran cantidad de objetos y funciones para la programación de aplicaciones de audio y plugins (ROLI Ltd., 2017).

Figura 32: Traktion 7 (TRACKTION COORPORATION, 2017)

[10] **Framework:** "un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación" (PÉREZ, 2015)

2.1.5.3 Entornos de Desarrollo

A continuación, se detallan los entonos de desarrollo utilizados, la aplicación fue desarrollada principalmente en OsX debido a la baja latencia del sistema operativo. Para su desarrollo se utilizó Xcode, además la aplicación requirió una versión para Windows debido a falta de del suficiente hardware Apple para las pruebas de percepción. Para el desarrollo en Windows se utilizó Visual Studio. A continuación, se presenta un análisis de estos entornos de desarrollo.

2.1.5.3.1 Xcode

Xcode es un entorno de desarrollo integrado (IDE, en sus siglas en inglés) (ver Fig. 33) para OsX que se compone de un conjunto de herramientas creadas por Apple para facilitar el desarrollo de aplicaciones para OsX, iOS, watchOS y tvO. En su versión 9 usa como compilador clang, y puede compilar código de C, C++, Swift, Objective-C, Objective-C++ y AppleScript. Actualmente Xcode se puede conseguir de manera gratuita del Mac App Store. Xcode además posee herramientas como simuladores de dispositivos, depurador y rediseño de código (APPLE COMPUTER INC., 2017).

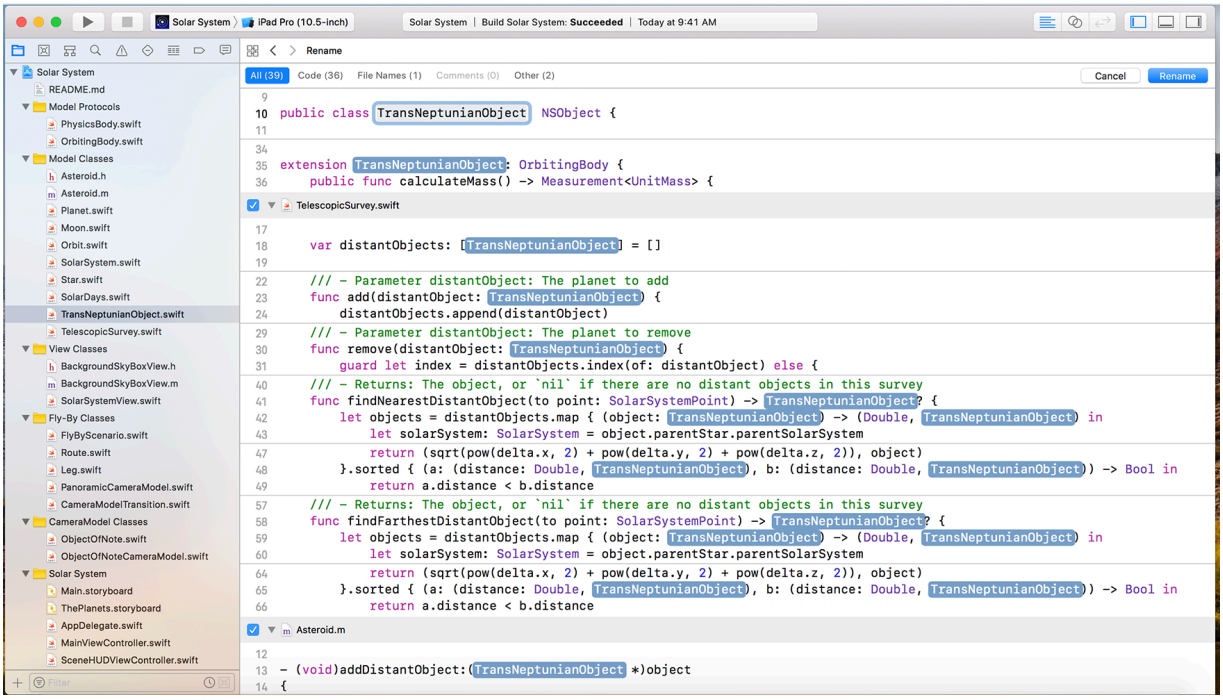


Figura 33: Xcode 9 (APPLE COMPUTER INC., 2017)

2.1.5.3.2 Microsoft Visual Studio

Microsoft Visual Studio (Fig. 34) es un entorno de desarrollo integrado para sistemas operativos Windows y OsX (que actualmente se encuentra en fase de pruebas). Permite el desarrollo de aplicaciones en varios lenguajes de programación, tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP. Además, permite el desarrollo de aplicaciones web gracias a ASP.NET MVC, Django, etc. (MICROSOFT, Novedades de Visual Studio, 2017).

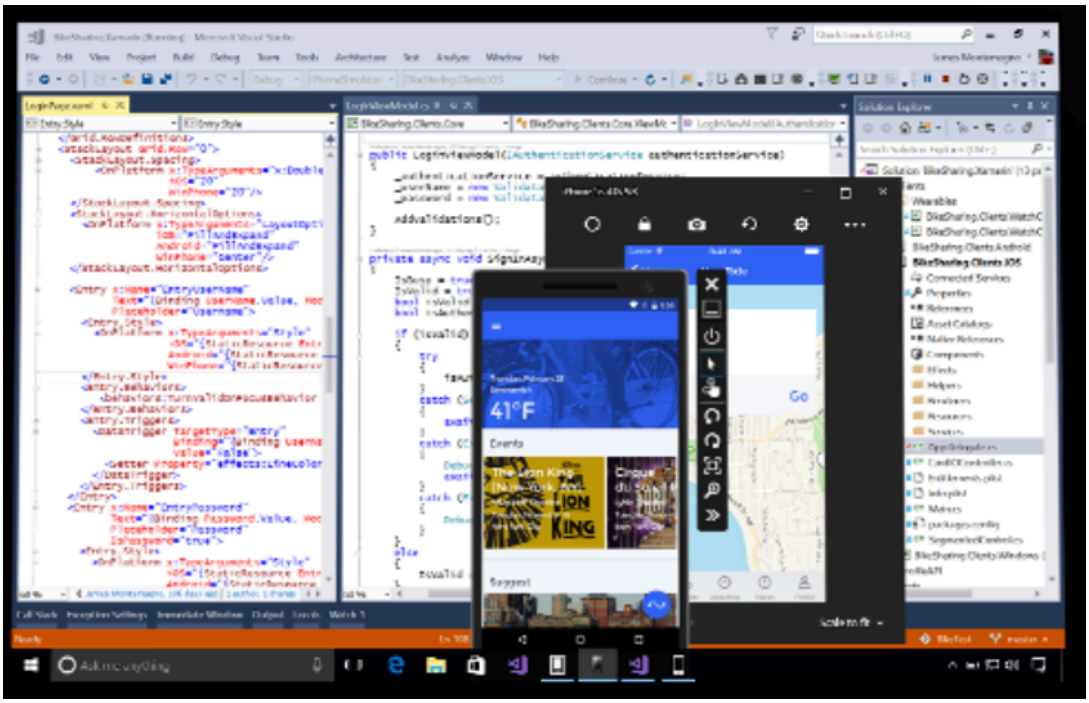


Figura 34: Visual Studio 2017 (MICROSOFT, Novedades de Visual Studio, 2017)

2.1.6 Buenas Prácticas y Normas en Aplicaciones de Audio

Las aplicaciones de audio en tiempo real deben seguir algunos principios que las aplicaciones "normales" no suelen seguir. Estos principios no tienen relación con la plataforma en la que estemos programando, hay dos principales consideraciones que debemos tomar en cuenta según Bencina (2011):

1. El audio de una aplicación no debe contener ruido (*glitchs*[11])
2. Y las aplicaciones en tiempo real no esperan por nada

En un sistema operativo las muestras de audio no son recibidas por la tarjeta de audio de muestra a muestra, sino se proveen en bloques (*buffers*) al driver de la tarjeta o a alguna capa intermedia del sistema operativo, como se puede observar en la Fig 35. Por lo general el *API* de audio realiza peticiones de un nuevo buffer mediante una función llamadas (*callbacks*), si tenemos una tasa de muestreo de 44100 muestras por segundo y un buffer de 256 muestras tendremos llamadas al buffer cada 5.8 milisegundos (que se obtiene dividiendo las muestras entre la tasa de muestreo). Por esta razón no importa lo que suceda en la aplicación, la misma debe proveer muestras cada 5.8 milisegundos sin excepciones. Si se llegara a perder un *buffer* o este tardaría mucho en procesarse escucharíamos ruido causado por la pérdida de información de audio (BENCINA, 2011).

[11] **Glitch**: es una súbita irregularidad en un sistema electrónico por ejemplo el sonido de un CDs rayado.

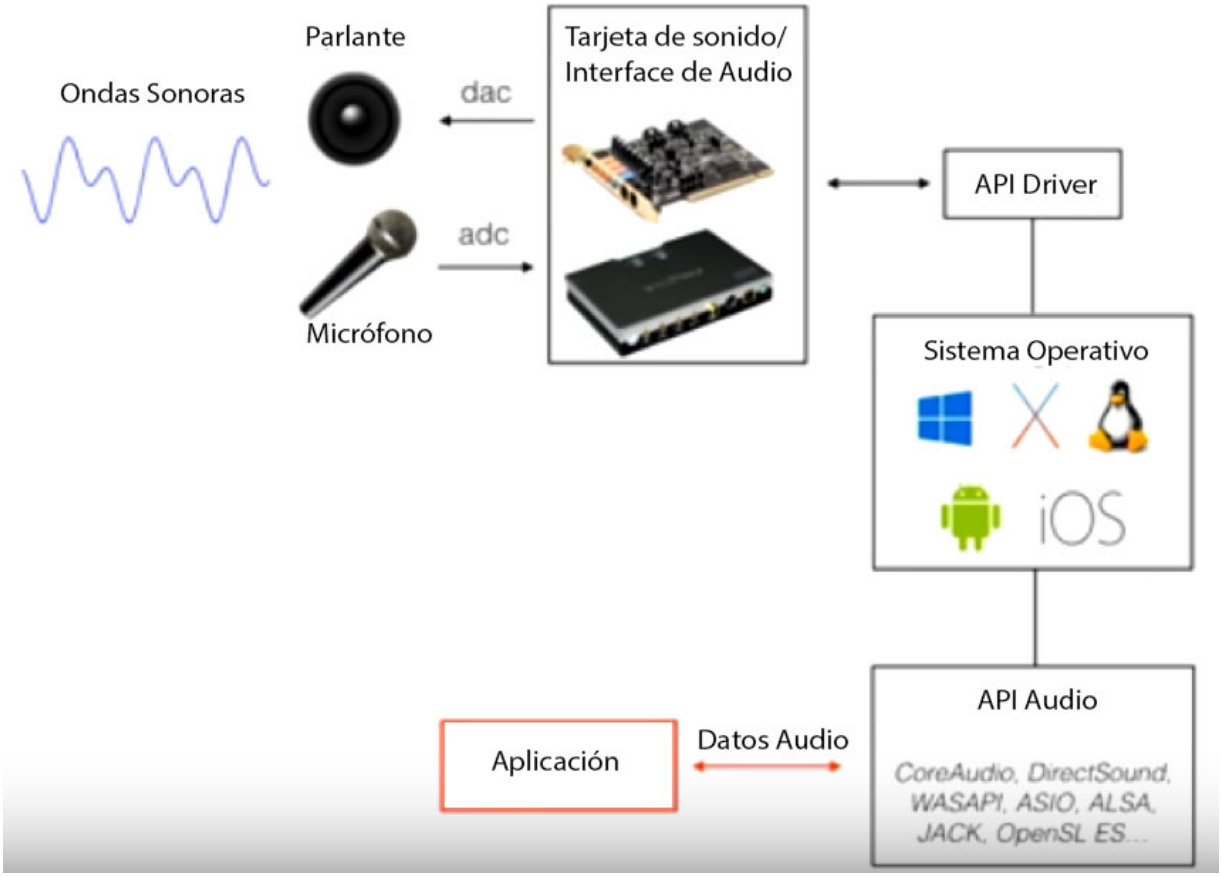


Figura 35: Funcionamiento de una Aplicación de Audio (DOUMLER, 2017)

2.1.6.1 Asignación de Memoria en Aplicaciones de Audio en Tiempo Real

Dentro del hilo de audio de una aplicación de audio en tiempo real no se debe realizar asignaciones de memoria en un hilo de audio, por ejemplo no se deben llamar las funciones *malloc*, *free*, new o *delete* de C/C++. Las funciones de asignación de memoria suelen proteger los datos usando bloqueos por lo que llamarlas no es seguro (por lo ya discutido en la sección de Fundamentos Teoricos). Además las funciones de asignación de memoria podrían estar usando algoritmos que tomen tiempos impredecibles por lo que no se deben llamar en los hilos de procesamiento de audio. Tomando en cuenta todas estas reestriciones las asignaciones de memoria siempre se deben realizar al iniciar la aplicación (BENCINA, 2011).

2.1.6.2 Otras Fuentes de Ruido en el Audio

Normalmente cuando usamos lenguajes de programación de alto nivel que utilizan colectores de basura (GC o *Garbage Collectors*), y si estos se activan en el hilo de procesamiento de audio pueden generar ruido (también conocido en inglés como *glitchs*). Ya que los GC puede utilizar bloqueos internamente, por lo que si se va a utilizar uno de estos lenguajes es pertinente revisar si existen colectores de basura que funcionen de manera segura en tiempo real (BENCINA, 2011).

Otra fuente de ruido es el sistema de paginación del sistema operativo, por lo que si el sistema operativo pagina la memoria que estamos usando y la ubica los datos en la memoria virtual. Lo que haría esperar al hilo de audio hasta que el sistema operativo vuelva a cargar los datos necesarios desde la memoria virtual a la RAM. Por lo que al manejar grandes cantidades de RAM debemos utilizar mecanismos que mantengan estos datos en la memoria RAM, como son *lock* y *munlock* en OS X y Linux, *virtualLock* y *virtualUnlock* en Windows (BENCINA, 2011).

2.2 Diseño

En la presente sección se presentan todas las consideraciones hechas y diagramas de diseño realizados antes de la implementación de la aplicación.

2.2.1 Modelo Conceptual

El modelo de clases de la Fig. 36, contiene todas las clases necesarias para el desarrollo de este proyecto. Uno de los aspectos más importantes del diseño lo podemos observar en la clase *SoundSource* que es la encargada de conectar el módulo de aurilización con el ambiente 3D, y es la que tiene la responsabilidad de la sincronización de ambos hilos. Otro aspecto importante en el diseño es que la clase *AudioDeviceManager* está contenida dentro de la clase *MainComponent*, y es referenciada por la clase *SoundSource*. Lo que nos permite el procesamiento de varias fuentes de manera simultánea.

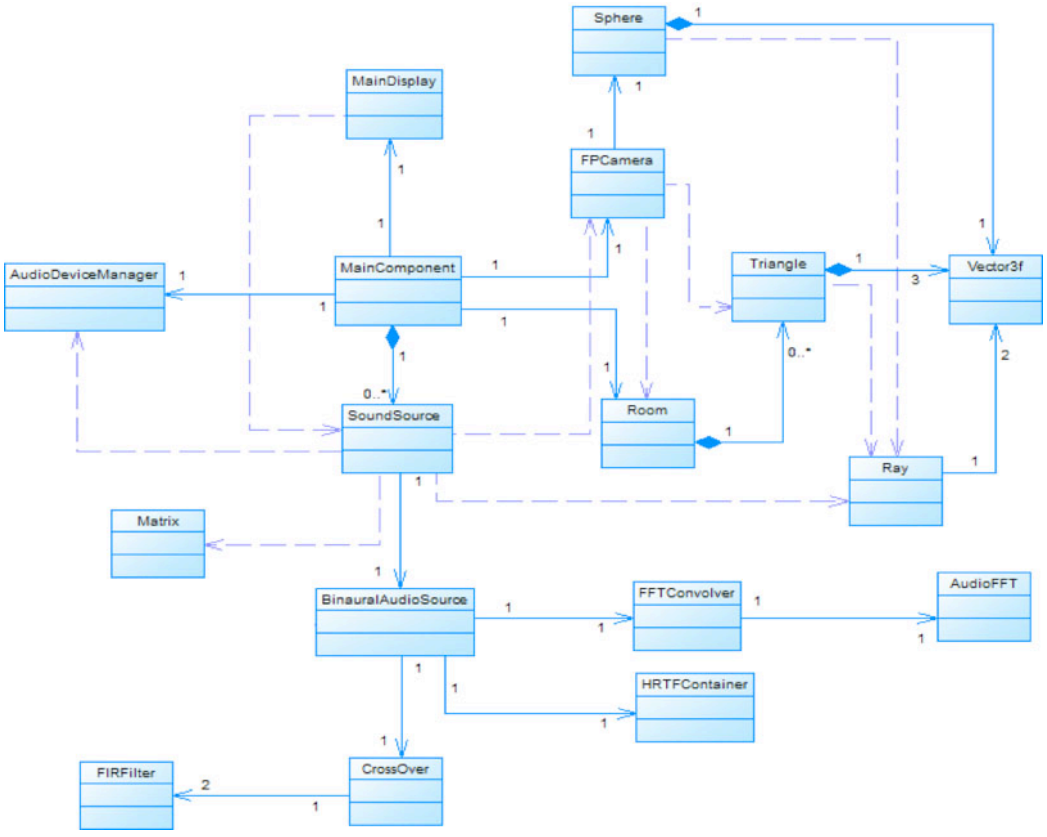


Figura 36: Modelo Conceptual de Clases

La clase responsable del sistema de interacción físico es la clase *SoundSource* la misma utiliza las clases *Sphere*, *Triangle*, *Vector3f*, *Ray* y *Room* para realizar las simulaciones. La clase *Room* representa el obstáculo (en este caso un modelo 3D de un laberinto) que está compuesto de objetos de la clase *Triangle*. Al realizar las simulaciones, la clase *Triangle* es la encargada de determinar si existen colisiones con algún objeto de la clase *Ray*. Finalmente, la clase *SoundSource* es la encargada de realizar la translación del objeto *Ray* recibido (utilizando objetos de la clase *Matrix*) al sistema de coordenadas del receptor.

La clase responsable del procesamiento de señales es la clase *BinauralAudioSource* en la misma se utiliza un objeto *FFTConvolver* que realiza un producto de convolución entre la señal de audio con una HRTF obtenida del objeto *HRTFContainer* (la clase HRTFContainer contiene la base de HRTFs de Kemar del MIT). Además, la clase *BinauralAudioSource* hace uso del objeto *CrossOver* que contiene filtros que separan la señal en bajas y altas frecuencias (de 0 a 200 hz, y de 200 hz a 22,05 khz) para reducir el ruido del producto de convolución en las frecuencias altas.

La clase que tiene la responsabilidad principal de los gráficos es la clase *MainComponent* que es la encargada de manejar todo el contexto de OpenGL. La misma contiene un objeto de la clase *FPCamera*, que es la clase encargada de brindarnos la perspectiva en primera persona. Igualmente, la clase *MainComponent* contiene objetos de la clase *SoundSource* y *Room*, estas dos clases poseen métodos que son utilizados para ser graficados usando el contexto de OpenGL. Finalmente, la clase *MainComponent* contiene una clase *MainDisplay* que es la encargada de mostrar la posición de las fuentes (que son obtenidas de los objetos *SoundSource*) en relación con la cabeza virtual que se muestra en la esquina superior izquierda de la aplicación.

2.2.2 Arquitectura de Hilos

Para el funcionamiento en tiempo real, la aplicación requiere que sus principales procesos se distribuyan en dos hilos (que se procesan en paralelo). Para el caso del audio, se ejecuta un hilo de alta prioridad, mientras que para los gráficos se utiliza un hilo de prioridad normal. Ambos hilos se ejecutan de manera simultánea por lo que es necesaria la sincronización de ambos hilos de procesamiento. En la Fig. 37 se muestra la arquitectura de los hilos de una típica aplicación de audio.

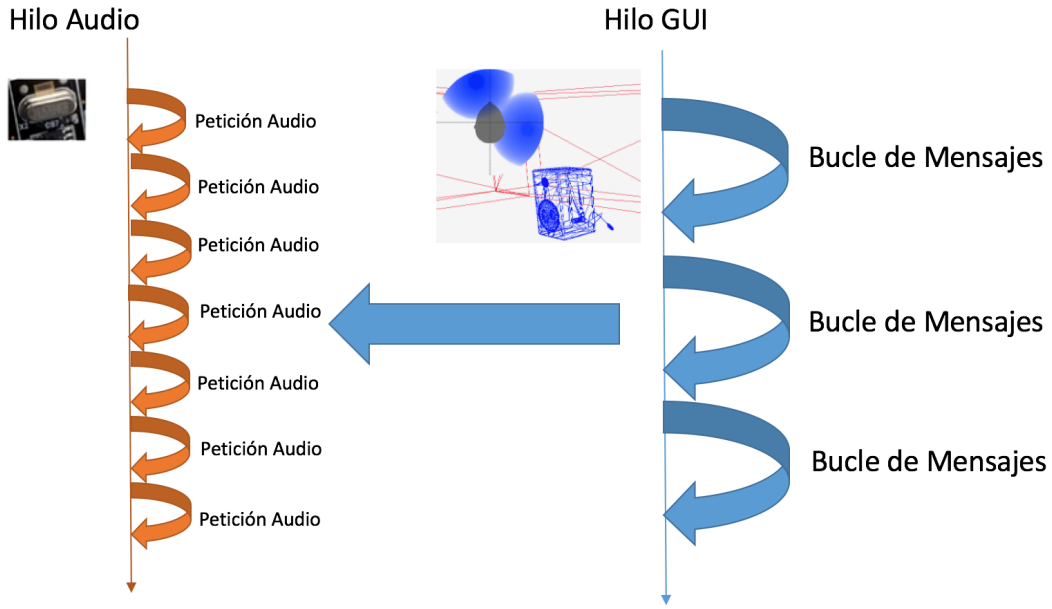


Figura 37: Arquitectura de Hilos de la Aplicación (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017)

Es importante señalar la sincronización de los hilos debe realizarse libre de bloqueos (BENCINA, 2011). Se entiende como bloqueo a cualquier procedimiento que requiera esperar por un recurso (esperas causadas por el procesamiento de otros hilos o lectura de datos del disco duro, presencia de *mutex*[12] o *socket*[13], entre otras). Cuando la información es procesada

[12] **Mutex**: Un mutex es un objeto bloqueante que previene que otros hilos accedan a ese espacio de memoria (TCR NETWORK, Reference: mutex, 2017).
[13] **Socket**: Un socket es un punto final de un enlace de comunicación bidireccional entre dos programas que se ejecutan en la red (ORACLE, 2015).

en tiempo real, esperar por un recurso causa la pérdida del buffer de información de audio, lo cual se traduce en breves interrupciones en la información generando ruido.

Para realizar una sincronización libre de bloqueos en este caso es aconsejable el uso de variables atómicas para garantizar su seguridad, evitando de esta forma que existan *data races* (que se explicarán más adelante). Para este caso se utilizaron como variables atómicas de tipo *float* (de la librería standard de C++ (TCR NETWORK, Reference: atomic, 2017)) al azimut θ y la elevación φ, las cuales se obtienen al transformar la dirección de incidencia del rayo proveniente de la fuente de sonido, de coordenadas cartesianas a coordenadas polares (WILLIAMS, 2012).

2.2.2.1 Variables Atómicas

En una aplicación de C++ todas las variables tienen una dirección de memoria. Si dos hilos acceden a direcciones de memoria diferentes no hay problema, pero si dos hilos tratan de acceder a la misma dirección de memoria pueden presentarse dificultades. Si los hilos que acceden a la vez solo leen los datos de la dirección de memoria no hay problemas, pero si ambos tratan de modificar los datos de una dirección de memoria tenemos lo que se conoce como *data races*. Un *data race* tiene como resultado un comportamiento no predecible de la aplicación, lo que genera errores en tiempo de ejecución. Para evitar *data races* se debe modificar el orden de acceso a las direcciones de memoria. Una variable atómica va a forzar que los accesos a la memoria de las operaciones de escritura se ejecuten antes que las de lectura (WILLIAMS, 2012).

Las variables atómicas nos dan la facilidad de realizar un ordenamiento forzado de los accesos a la memoria entre ambos hilos. Las mismas fuerzan los accesos entre ambos hilos utilizando operaciones atómicas. Las operaciones atómicas son operaciones de bajo nivel indivisibles que no pueden quedar a mitad de ejecución, simplemente tienen dos posibilidades haber sido ejecutadas o están sin ejecutar (WILLIAMS, 2012). De esta manera las variables atómicas permiten el acceso seguro a un espacio de memoria desde dos hilos de manera simultánea.

2.2.3 Interfaz de Usuario

La interfaz de usuario es bastante simple, se trata de una vista en primera persona (semejante a la de un juego FPS[14]) que se controla usando el mouse (para cambiar el azimut y elevación de la misma) (similar a los controles de DOOM que se muestra en la Fig. 38) y el teclado para cambiar la posición del punto del observador dentro del ambiente virtual.

Figura 38: Doom Juego FPS (DOS GAMES ARCHIVES, 2017)

[14] **FPS**: First Person Shooter o videojuegos de disparos en primera persona, son un género de videojuegos y subgénero de los videojuegos de disparos en los que el jugador observa el mundo desde la perspectiva del personaje protagonista.

El ambiente tridimensional y las fuentes sonoras son renderizados como mallas de triángulos que son cargados desde archivos OBJ (Fig. 39), y se visualizan en ¨pantalla completa¨. Para las fuentes se utiliza un modelo tridimensional de un parlante y para el obstáculo se utiliza un modelo de laberinto simple. Además, se incluyó un indicador graficado en la esquina superior izquierda que muestran donde están ubicadas las fuentes sonoras en relación con la cabeza virtual.

Utilizando auriculares, el usuario puede interactuar con las fuentes sonoras cambiando su posición utilizando los periféricos antes mencionados. Este cambio de posición generará en la señal de audio el efecto sonoro tridimensional correspondiente a la nueva posición, tanto de dirección como de distancia. Cada fuente reproduce una canción diferente cargada desde un archivo mp3.

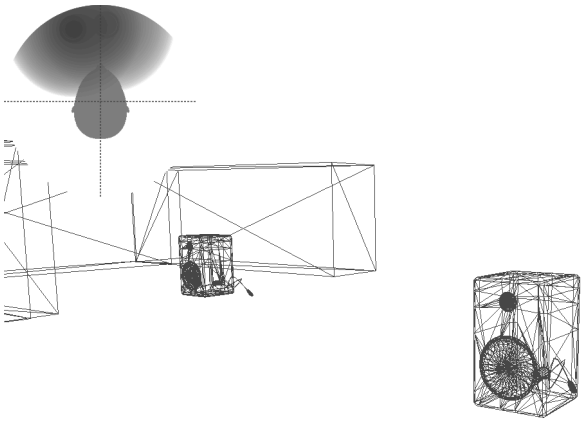


Figura 39: Interfaz Gráfica Diseñada

2.2.4 Plan de Pruebas del Sistema

La Tabla 1 detalla el plan de pruebas del sistema, que se elaboró tomando en cuenta los requerimientos funcionales de la aplicación. Y se usó para validar el correcto funcionamiento de los módulos implementados.

ID	Caso de prueba	Entradas	Resultado Esperado
C1	Ingreso a la aplicación	El usuario abre la aplicación	Se despliega toda la interface de usuario, mostrándose la escena, fuentes y visor de posicionamiento de la fuente.
C2	Movimiento Azimutal Cámara/Audio 3D	El usuario rota el azimut usando el mouse	La cámara se actualiza dependiendo del movimiento azimutal, el visor de posición de la fuente sonora muestra donde se escucha actualmente la fuente y el motor de aurilización produce el efecto de posicionamiento del sonido.
C3	Movimiento de Elevación Cámara/Audio 3D	El usuario cambia la elevación usando el mouse	La cámara se actualiza dependiendo del movimiento de elevación, el visor de posición de la fuente sonora muestra donde se escucha actualmente la fuente y el motor de aurilización produce el efecto de posicionamiento del sonido.
C4	Acercamiento a la fuente Cámara/Audio 3D	El usuario se acerca a la fuente desde la posición inicial utilizando el teclado	Cuando presione las flechas del teclado debe cambiar de posición la cámara, y mientras más cerca se encuentre el usuario a la fuente el volumen debe incrementarse.
C5	Posicionamiento en medio de las dos fuentes Cámara/Audio 3D	El usuario se ubica en medio de las dos fuentes	Se deben visualizar las dos fuentes en el visor, y además se debe escuchar el efecto de posicionamiento de cada una.
C6	Movimiento Azimutal Cámara/Audio 3D en medio de las dos fuentes	El usuario se ubica en medio de las dos fuentes y cambia el azimut	La cámara se actualiza dependiendo del movimiento de azimut. El visor de posición de las fuentes sonoras muestra donde se escuchan actualmente las fuentes.
C7	Movimiento Elevación Cámara/Audio 3D en medio de las dos fuentes	El usuario se posiciona detrás de una pared cubriéndose de la fuente	La cámara se actualiza dependiendo del movimiento de elevación. El visor de posición de las fuentes sonoras muestra donde se escuchan actualmente las fuentes.
C8	Efecto de Oclusión de Audio (Posicionamiento detrás de las paredes que bloquee las fuentes)	El usuario se posiciona detrás de una pared cubriéndose de la fuente	La fuente debe dejarse de escuchar por completo si no existe sonido directo hacia el usuario y el visualizador no debe mostrar ninguna fuente.

Tabla 1: Plan de Pruebas del Sistema

2.3 Implementación

En la presente sección se documentará el desarrollo del prototipo funcional de la aplicación. Para esto se van a establecer los estándares de desarrollo que se van a utilizar y las funcionalidades implementadas en la aplicación. Además, se presentarán los proyectos se Software Open Source y Libres que se utilizaron para el desarrollo de esta aplicación.

2.3.1 Estándares de Desarrollo

Los estándares usados son los mismos que los usados en Juce (STORER, 2017) (y solo se aplica para el código nuevo generado y no los proyectos libres que se usaron) y se describen a continuación.

- Usar el Indentado Allman en los bucles de control y funciones escribir las siguientes líneas de código utilizando tabs.
- Usar *Camel-Case* en el nombrado de variables y clases ej. *MyClassName*, *myVariableName*
- Los nombres de las clases deben empezar con mayúscula ej. MyClassName
- Los nombres de las variables miembro deben empezar con minúscula ej. myVariableName
- Todos los nombres de las variables deben estar en inglés
- No usar guiones bajos en el nombrado de variables
- Si se sobrescribe un método de una clase padre se debe utilizar la palabra clave override
- En lo posible no manejar la memoria manualmente usando “malloc/free” o “new/delete”, en vez usar vectores o punteros inteligentes
- En lo posible no usar librerías de c en vez utilizar la librería estándar de c++
- No definir funciones dentro de un macro
- Las variables que tienen vida solo dentro del bucle se deben declarar en el mismo ej. "for (int i = 0; i < 10; ++i)"

- Usar siempre variables de acceso privado dentro de una clase excepto cuando el uso de variables públicas simplifique el desarrollo y no sean un problema en la seguridad del programa.

2.3.2 Librerias y Proyectos Open Source Usados

Dentro del desarrollo del proyecto se basó e integró varios proyectos se Software Open Source y Libres, debido a la complejidad y corto tiempo de desarrollo del proyecto. Los mismos se enumeran y describen a continuación:

FFTConvolver

Para el algoritmo de convolución rápida se utilizó el proyecto FFTConvolver, en el cual se modificó para poder cambiar las repuestas impulsivas asociadas a la cabeza en tiempo real.

- **Autor:** HiFi-LoFi
- **Licencia:** GPL v3
- **Link:** https://github.com/HiFi-LoFi/FFTConvolver

OouraFFT

Para el algoritmo de transformadas rápidas de Fourier se utilizó las transformadas de la librería de Ooura, en la cual se utiliza dentro del proceso de convolución rápida.

- **Autor:** Takuya Ooura
- **Licencia:** BSD
- **Link:** http://www.kurims.kyoto-u.ac.jp/~ooura/

Kemar HRTF Librería en C

Para poder usar fácilmente las respuestas impulsivas relacionadas con la cabeza se utilizó está librería que contiene una base de datos de HRIRs completamente programada en C. Esto facilitó bastante el desarrollo ya que las mediciones originales de Bill Gardner and Keith Martin se encontraban en varios archivos WAV y realizar las búsquedas de la HRIR correcta resultaba un proceso bastante complejo.

- **Autor:** Aristotel Digenis
- **Licencia:** MIT
- **Link:** https://github.com/greekgoddj/mit-hrtf-lib

BinAural VST

Este proyecto consta plugin de audio (VST[15]), que posiciona el sonido en un espacio tridimensional usando HRTFs. De este proyecto se estudió y adaptó el *cross-fader* que se utilizó para quitar el ruido debido al rápido cambio de las HRTFs cuando se mueve la cabeza. Además, se adaptó el componente que permite visualizar la posición de la fuente.

- **Autor:** Tomasz Woźniak
- **Licencia:** GPL
- **Link:** https://github.com/tmwoz/binaural-vst

Cámara Primera Persona

Este proyecto es uno de una serie de tutoriales para la creación de juegos. El mismo contiene una cámara con visión en primera persona, que nos permite navegar por un mundo tridimensional usando el mouse y teclado. El mismo proyecto está completamente programando con el API de Windows32 por lo que se eliminaron todas las dependencias del mismo usando JUCE para conseguir código portable multiplataforma.

- **Autor:** DigiBen (gametutorials.com)
- **Licencia:** MIT
- **Link:** https://github.com/gametutorials/tutorials

[15] **VST**: “es una interfaz estándar desarrollada por Steinberg para conectar sintetizadores de audio y plugins de efectos a editores de audio y sistemas de grabación” (ÁLVAREZ, 2010).

Tiny Obj

Este proyecto consta de un lector de archivos OBJ, el mismo se utilizó para cargar los modelos y obstáculos en la aplicación.

- **Autor:** Syoyo Fujita
- **Licencia:** MIT
- **Link:** https://github.com/syoyo/tinyobjloader

Ray Tracer

Este proyecto consta de un simulador acústico que utiliza trazado de rayos para las simulaciones, del cual se utilizaron las matrices de rotación.

- **Autor:** Song Ho Ahn / Shipeng Xu
- **Link:** https://github.com/billhsu/RayTracer

2.4 Pruebas del Sistema

En la presente sección se va a documentar las pruebas realizadas (funcionales y de percepción del usuario) de la aplicación, tomando en cuenta las funcionalidades desarrolladas, por lo que se usó en cuenta el plan de pruebas (Tabla 1) desarrollado en la sección Análisis.

2.4.1 Requerimiento de Hardware

Para las pruebas se utilizó un computador con las siguientes características:

- MacBook Pro (15-inch, Late 2011)
- OsX El Capitan V10.11.6
- Porcesador: 2,4 GHz Intel Core i7
- 4 GB 1333 MHz DDR3
- AMD Radeon HD 6770M

La aplicación al ser escrita con un framework multiplataforma puede ser compilado y ejecutado en Windows, OsX y Linux, sin mayores cambios al código fuente. La misma fue utilizada en Windows, pero las pruebas funcionales se realizaron bajo OsX.

Las pruebas fueron realizadas una vez que los dos módulos (aurilización y el ambiente interactivo 3D) fueron terminados e integrados. En la Fig. 40 se muestra la interfaz de la aplicación el indicador superior indica las posiciones de las fuentes, mientras las mallas de color magenta representan los obstáculos y las mallas de color cian son las fuentes sonoras. El ambiente permite la interacción del usuario con las fuentes mediante el teclado (movimientos de translación) y el mouse (movimientos de rotación).

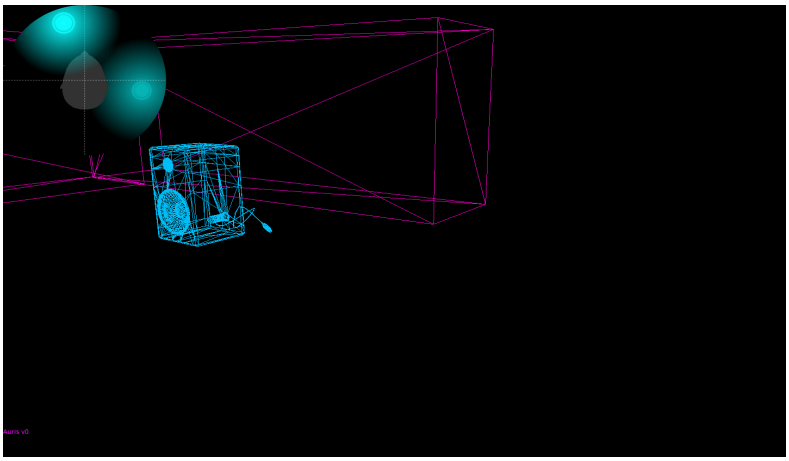


Figura 40: Captura de Pantalla de la Aplicación

2.4.2 Caso de Prueba (C1)

Descripción: Ingreso a la aplicación

Entradas: El usuario abre la aplicación.

Resultado esperado: Se despliega toda la interface de usuario, mostrándose la escena, fuentes y visor de posicionamiento de la fuente (Fig. 41).

Estatus: Aprobado

Capturas de pantalla:

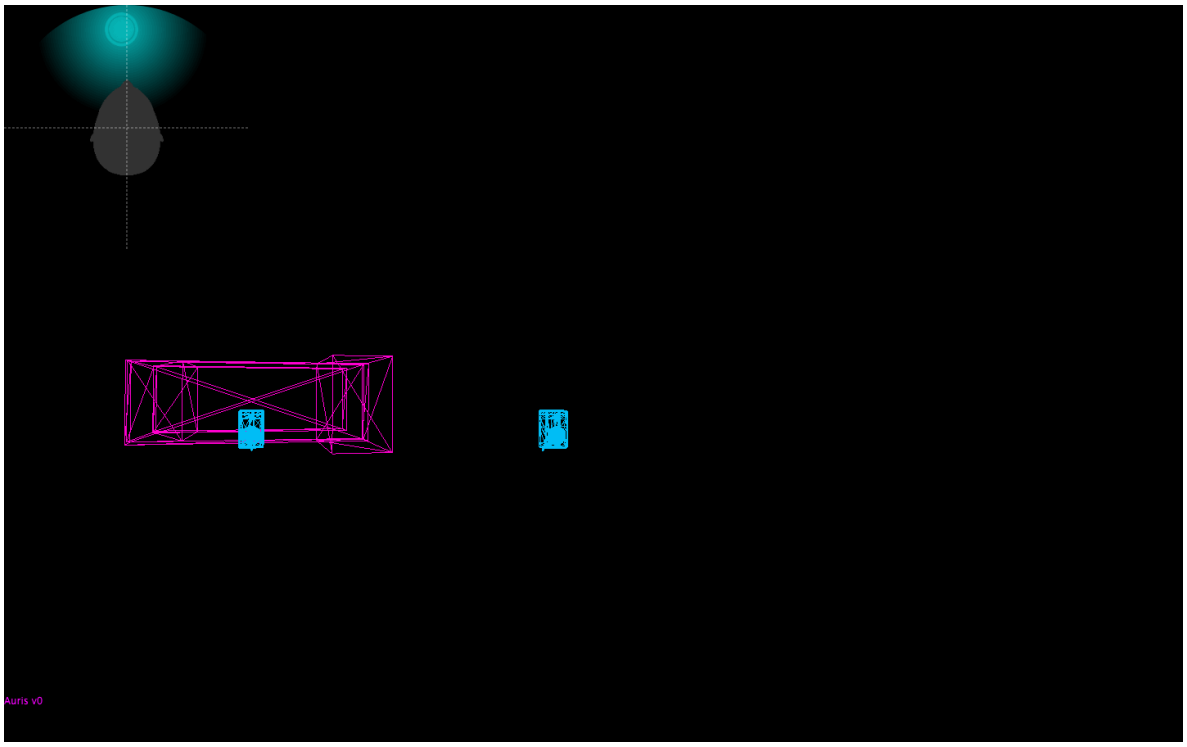


Figura 41: Caso 1 Inicio del Sistema

Comentario: La aplicación se inicia de manera exitosa, con todos los controles funcionando de manera exitosa.

2.4.3 Caso de Prueba (C2)

Descripción: Movimiento Azimutal Cámara/Audio 3D

Entradas: El usuario rota el azimut usando el mouse

Resultado esperado: La cámara se actualiza dependiendo del movimiento azimutal, el visor de posición de la fuente sonora muestra donde se escucha actualmente la fuente y el motor de aurilización produce el efecto de posicionamiento del sonido (Fig. 42).

Estatus: Aprobado

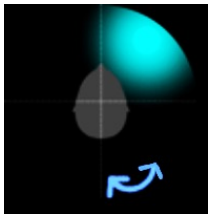


Figura 42: Movimiento Azimutal de la Cabeza

Comentario: Las rotaciones azimutales de la cámara funcionan de manera exitosa usando el mouse y el sonido generado concuerda con la posición en el mundo virtual.

2.4.4 Caso de Prueba (C3)

Descripción: Movimiento de Elevación Cámara/Audio 3D

Entradas: El usuario cambia la elevación usando el mouse

Resultado esperado: La cámara se actualiza dependiendo del movimiento de elevación, el visor de posición de la fuente sonora muestra donde se escucha actualmente la fuente y el motor de aurilización produce el efecto de posicionamiento del sonido (Fig. 43).

Estatus: Aprobado

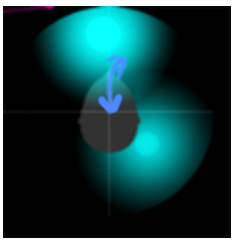


Figura 43: Movimiento de Elevación de la Cabeza

Comentario: Las rotaciones en elevación de la cámara funcionan de manera exitosa usando el mouse, el sonido generado concuerda con la posición en el mundo virtual y el visor funciona correctamente.

2.4.5 Caso de Prueba (C4)

Descripción: Acercamiento a la fuente Cámara/Audio 3D

Entradas: El usuario se acerca a la fuente desde la posición inicial utilizando el teclado

Resultado esperado: Cuando presione las flechas del teclado debe cambiar de posición la cámara, y mientras más cerca se encuentre el usuario a la fuente el volumen debe incrementarse (Fig. 44).

Estatus: Aprobado

Capturas de pantalla:

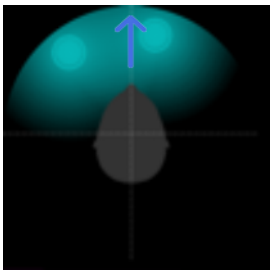


Figura 44: Acercamiento a la fuente sonora

Comentario: Los movimientos causados por el teclado funcionan correctamente, y mientras más cerca está la fuente más fuerte es el sonido generado.

2.4.6 Caso de Prueba (C5)

Descripción: Posicionamiento en medio de las dos fuentes Cámara/Audio 3D

Entradas: El usuario se ubica en medio de las dos fuentes

Resultado esperado: Se deben visualizar las dos fuentes en el visor, y además se debe escuchar el efecto de posicionamiento de cada una (Fig. 45).

Estatus: Aprobado

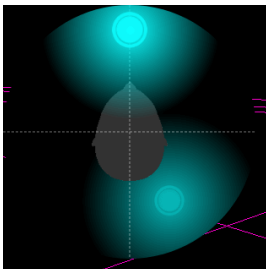


Figura 45: Posicionamiento en Medio de las Dos Fuentes

Comentario: El sonido generado por las dos fuentes concuerda con la posición en el mundo virtual y el visor funciona correctamente.

2.4.7 Caso de Prueba (C6)

Descripción: Movimiento Azimutal Cámara/Audio 3D en medio de las dos fuentes

Entradas: El usuario se ubica en medio de las dos fuentes y cambia el azimut

Resultado esperado: La cámara se actualiza dependiendo del movimiento de azimut. El visor de posición de las fuentes sonoras muestra donde se escuchan actualmente las fuentes (Fig. 46).

Estatus: Aprobado

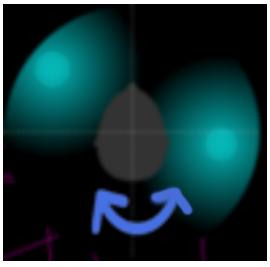


Figura 46: Movimiento Azimutal en Medio de las Dos Fuentes

Comentario: Las rotaciones en azimut de la cámara funcionan de manera exitosa usando el mouse, el sonido generado por las concuerda con la posición y rotaciones en el mundo virtual, el visor funciona correctamente.

2.4.8 Caso de Prueba (C7)

Descripción: Movimiento Elevación Cámara/Audio 3D en medio de las dos fuentes

Entradas: El usuario se posiciona detrás de una pared cubriéndose de la fuente

Resultado esperado: La cámara se actualiza dependiendo del movimiento de elevación. El visor de posición de las fuentes sonoras muestra donde se escuchan actualmente las fuentes (Fig. 47).

Estatus: Aprobado

Capturas de pantalla:

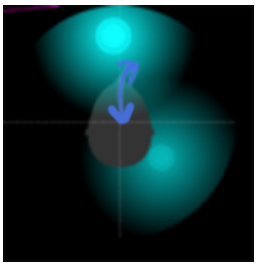


Figura 47: Movimiento de Elevación en Medio de las Dos Fuentes

Comentario: Las rotaciones en elevación de la cámara funcionan de manera exitosa usando el mouse, el sonido generado por las concuerda con la posición y rotaciones en el mundo virtual, el visor funciona correctamente.

2.4.9 Caso de Prueba (C8)

Descripción: Efecto de Oclusión Audio (Posicionamiento detrás de las paredes que bloquee la fuente)

Entradas: El usuario se posiciona detrás de una pared cubriéndose de la fuente

Resultado esperado: La fuente debe dejarse de escuchar por completo si no existe sonido directo hacia el usuario y el visualizador no debe mostrar ninguna fuente (Fig. 48).

Estatus: Aprobado

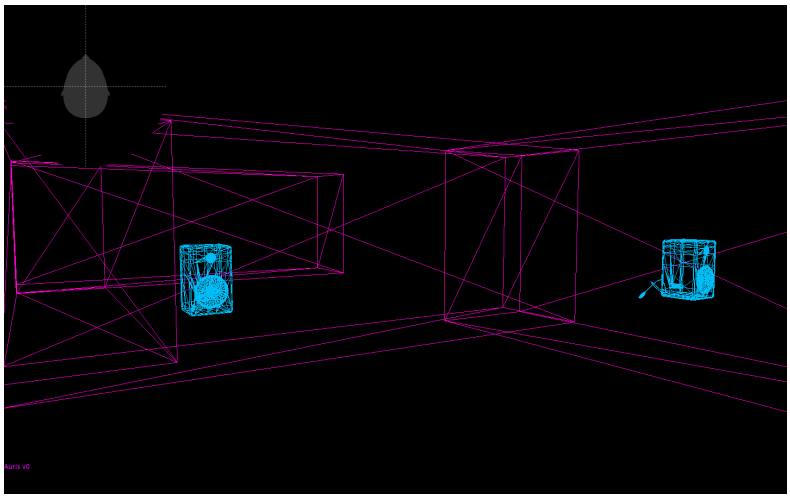


Figura 48: Posicionamiento Detrás de la las Paredes

Comentario: El sonido deja de escucharse por completo, por lo que el visor no muestra ninguna fuente activa.

CAPÍTULO III: RESULTADOS

Para probar el realismo en el procesamiento y precisión de los resultados que produce esta aplicación se desarrollaron 2 tipos de pruebas (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017):

1. Prueba de navegación ciega.

2. Prueba de valoración subjetiva.

3.1 Prueba de navegación ciega

Para la primera prueba participaron dos sujetos (sin entrenamiento previo). Esta prueba consistía en realizar una navegación dentro del ambiente virtual por medio de los periféricos (mouse y teclado), completamente a ciegas (sin ayuda del monitor). El sujeto tenía una posición inicial fija en el ambiente virtual con una elevación de 1 unidad y la fuente de sonido era ubicada alrededor del sujeto en una posición aleatoria (con un azimut variable entre 0◦ y 360◦ a la altura del piso) a 26 unidades de distancia del sujeto. El objetivo era llegar lo más cerca posible de la fuente sonora, guiándose solamente por el sonido que era escuchado por el usuario por medio de auriculares. Una vez que el sujeto creía llegar a su objetivo, este debía presionar una tecla para finalizar la prueba, la cual se repitió 50 veces.

Los resultados indican que se logró reducir la distancia inicial de separación en un 93,96% en promedio. Esto a pesar de que las pruebas fueron realizadas con las HRIRs de la cabeza artificial KEMAR y no las específicas de cada sujeto de prueba, lo cual puede generar distorsiones en la percepción (ALGAZI, 2001). Adicionalmente, la base de datos de HRIRs no cuentan con datos para elevaciones menores a −40◦, lo cual genera mayores impresiones cuando el sujeto está cerca de la fuente (GARDNER & MARTIN, 1994).

Los resultados de las distancias finales que separan a la fuente sonora del sujeto se presentan en las Tablas 2 y 3. Se utilizaron 6 intervalos para analizar la información. La segunda columna define el rango de distancia de los intervalos. Para cada intervalo se presenta la frecuencia de ocurrencia, fi (tercera columna), y la frecuencia acumulada, Fi (cuarta columna). Además, en la quinta columna se presenta el rango de error relativo obtenido.

Ni	Distancia	fi	Fi	Error
1	0,0998 a 0,8072	10	10	0 % al 3 %
2	0,8072 a 1,4146	13	23	3 % al 5 %
3	1,4146 a 2,0220	8	31	5 % al 8 %
4	2,0220 a 2,6294	9	40	8 % al 10 %
5	2,6294 a 3,2368	7	47	10 % al 12 %
6	3,2368 a 3,8441	3	50	12 % al 15 %

Tabla 2: Resultados Sujeto 1 (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017)

Ni	Distancia	fi	Fi	Error
1	0,0501 a 0,6194	8	8	0 % al 2 %
2	0,6194 a 1,1887	12	20	2 % al 5 %
3	1,1887 a 1,7580	14	34	5 % al 7 %
4	1,7580 a 2,3273	6	40	7 % al 9 %
5	2,3273 a 2,8966	7	47	9 % al 11 %
6	2,8966 a 3,4659	3	50	11 % al 13 %

Tabla 3: Resultados Sujeto 2 (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017)

Los números muestran que el 62% de los intentos el error no sobrepasa el 8% de la distancia original. Apenas el 20% de los intentos sobrepasa el 10%, sin embargo, ninguno de estos supera el 15%. El error relativo promedio fue de 6,04 %. Cabe recalcar que los gráficos de frecuencia de ocurrencia de los intervalos, registran que existe una tendencia hacia abajo de los intervalos de peor desempeño (ver Fig. 49 y 50).

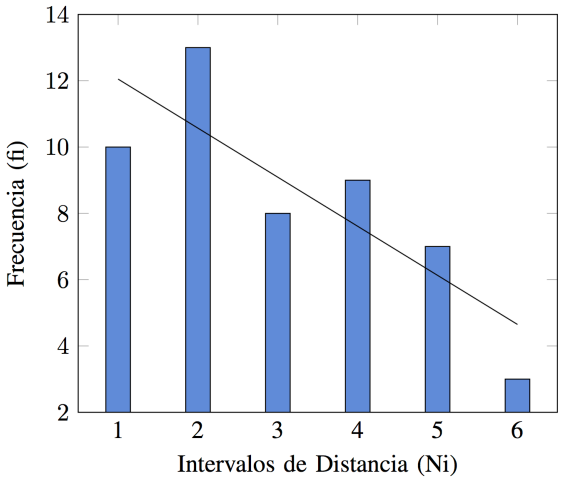


Figura 49: Histograma Sujeto 1 (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017)

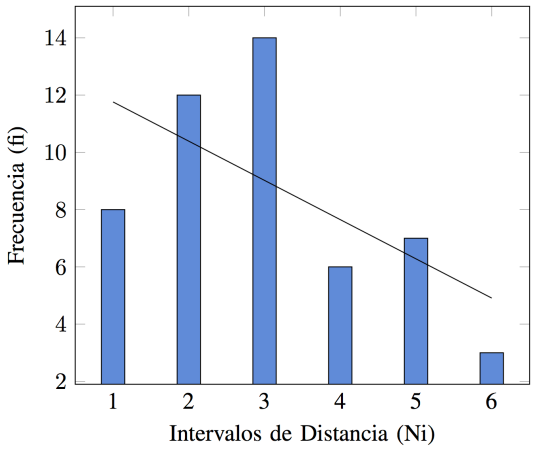


Figura 50: Histograma Sujeto 2 (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017)

Las Figuras de frecuencia de ocurrencia de los intervalos (Fig. 49 y 50), registran que existe una tendencia hacia abajo de los intervalos de peor desempeño. Lo que muestra que existe un proceso de adaptación a la aplicación.

3.2 Prueba de valoración subjetiva

La segunda prueba consistió en una valoración subjetiva realizada a 70 sujetos durante la presentación de la aplicación durante el evento “Encuentros ciencia y tecnología EPN 2016 Ciudades Sostenibles en el Siglo XXI” en el marco del Hábitat III. En esta validación se pidió a los visitantes a que naveguen libremente en un ambiente virtual tridimensional donde se encontraban varias fuentes sonoras. Después, se les solicitó que llenen una encuesta cualitativa de dos preguntas de opción múltiple sobre el realismo del sonido 3D generado por la aplicación.

La primera pregunta (ver Fig. 53) estaba relacionada a la impresión del sujeto sobre si se consiguió generar correctamente sonidos tridimensionales. El 90 % respondió validando el efecto de la aplicación.

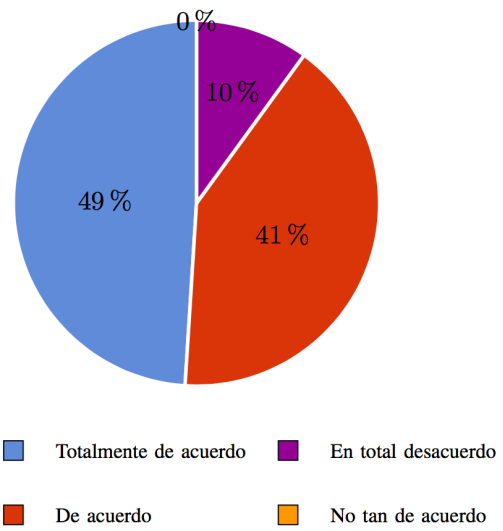


Figura 51: ¿Considera que el software simula correctamente sonido tridimensional? (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017)

La segunda pregunta (ver Fig. 52) buscaba información sobre cuán difícil fue percibir el efecto de inmersión acústica 3D en términos de usabilidad de la aplicación. El 53% de los encuestados reportó haber percibido el efecto de forma inmediata. Al siguiente 36% le costó un poco acostumbrarse, mientras que el 11% restante reportó haber tenido dificultades significativas.

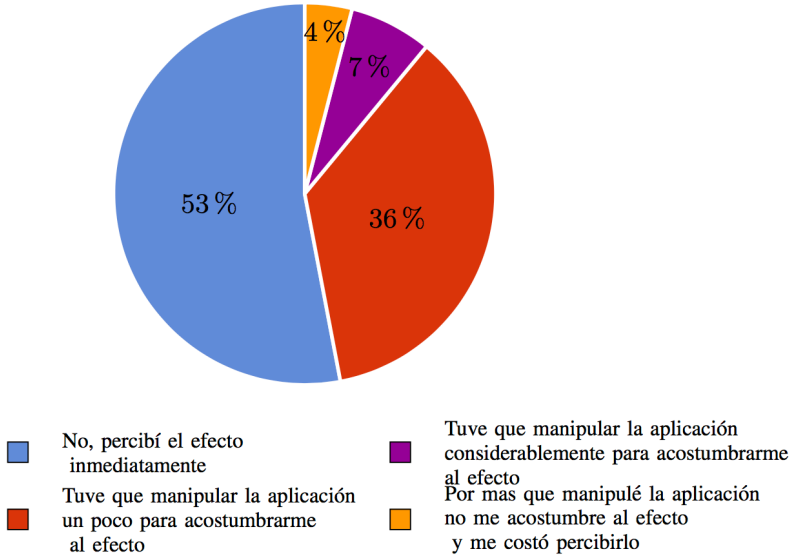


Figura 52: ¿Tuvo necesidad de acostumbrarse al sonido para percibir la sensación 3D? (ARMENDÁRIZ, LUCIO NARANJO, ASIMBAYA, & PROAÑO, 2017)

Se registró que un 90% de los sujetos, percibieron los efectos de la aplicación. Aunque a un 36% le costó un poco acostumbrarse y al 11% tuvo dificultades significativas, concordando con los resultados de la prueba de navegación a ciegas en que existe un proceso de adaptación a la aplicación.

CAPÍTULO IV: CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Se desarrolló y validó un módulo de aurilización que funciona en tiempo real dentro de un ambiente 3D en el cual el usuario puede trasladarse y rotar sobre su propio eje usando periféricos como el mouse y el teclado. Además, se realizaron pruebas con dicha herramienta, una cuantitativa y otra cualitativa para poder validar el realismo del módulo. Los resultados cuantitativos demuestran que, a pesar de utilizar las HRIRs de la cabeza artificial KEMAR (y no las específicas de cada sujeto), los sujetos son capaces de acercarse a la fuente a una distancia del 93,96% comparada con la original. El análisis estadístico de frecuencia muestra que existe una tendencia a la baja de los casos de más bajo desempeño. Por lo que medida se repita el proceso, se produce un proceso de adaptación del sujeto a los sonidos 3D generados por la aplicación.

Por otro lado, los resultados de las pruebas cualitativas muestran que en un 90% de los casos el efecto del motor de aurilización fue percibido de manera casi inmediata validan dando así positivamente la generación del efecto sonoro 3D. En el caso del 10% que no experimentó el efecto del motor de aurilización, se estima que existe un problema relacionado con falta de familiaridad con la interfaz de la aplicación, o esté relacionado a impresiones producidas por no usar las HRIRs propias del sujeto. Estos problemas pueden resolverse con un proceso de entrenamiento, tanto en el uso de la aplicación como en la utilización de HRIRs ajenas al sujeto.

El motor de aurilización fue probado exitosamente con las pruebas tanto funcionales como de percepción. Por tal motivo, este prototipo de software tiene el potencial de ser aplicado para la generación de sonidos 3D en dispositivos ETA (de manera que alerten de obstáculos a personas con discapacidad visual) como en productos de realidad virtual.

4.2 Recomendaciones

A pesar de que el proyecto se realizó utilizando frameworks multiplataforma, la compilación y el funcionamiento correcto del código no se garantiza en todas las plataformas (debido a los diferentes estándares de los compiladores). Por lo que se recomienda realizar pruebas (en cada incremento de desarrollo) en todas las plataformas a las que se quiera desplegar el ejecutable.

Igualmente, al basar un proyecto en trabajos de código de open source es recomendable realizar una adaptación (*refactor*) a los estándares de codificación propios de manera automática. Para esto, existen herramientas para realizar estas adaptaciones que pueden reducir considerablemente el tiempo y trabajo de los desarrolladores.

Para futuros trabajos es recomendable reducir aún más la complejidad computacional del motor de aurilización paralelizando el procesamiento del audio cuando se trabaja con varias fuentes sonoras al mismo tiempo. Esto permitirá lograr alertar a los usuarios de dispositivos ETA de la presencia de más obstáculos o a su vez se puede lograr generar una sensación más real en dispositivos de realidad virtual.

GLOSARIO

Anecoico: Capaz de absorber las ondas sonoras o electromagnéticas sin reflejarlas. Sin Eco (RAE).

Banda: conjunto de frecuencias sonoras delimitadas por un rango (FADGI).

Biauricular: relacionado a ambos oídos (dictionary.com LLC).

Discreto: una función, variable o sistema se considerarán discretos, en contraposición a continuos, si son divisibles un número finito de veces (RENZE & WEISTEIN).

Domino del Tiempo: es un término que hace referencia al análisis de funciones matemáticas o señales respecto al tiempo (CUFF, 2017).

Dominio de la Frecuencia: es un término que hace referencia al análisis de funciones matemáticas o señales respecto a su frecuencia (CUFF, 2017).

Entonos de Desarrollo (IDE): es una aplicación informática que proporciona herramientas integrales que facilitan el desarrollador y depuración del software (TechTarget).

Frecuencia de Muestreo: la frecuencia de muestreo determina la cantidad de muestras por unidad de tiempo que se toman de una señal continua para producir una señal discreta(señal digital) (FADGI).

Primitivas Geométricas: en OpenGL es el resultado de una secuencia de vértices al ser renderizada, las mismas pueden ser puntos, líneas, polígonos, triángulos, etc. (KHRONOS GROUP, 2017)

REFERENCIAS

MARTIN, D. (2011). *Función Transferencia y Respuesta Impulsiva.* Recuperado el 29 de mayo de 2017, de Universidad Nacional del Sur: http://lcr.uns.edu.ar/fvc/NotasDeAplicacion/FVC-Mart%C3%ADn%20Daprotis.pdf

PHISHDADIAN, F. (2016). *Filters, Reverberation & Convolution.* Obtenido de Northwestern University Computer Science: http://www.cs.northwestern.edu/~pardo/courses/eecs352/lectures/MPM16-topic9-Filtering.pdf

GONZALES, R., & WINTZ, P. (1977). *Convolución (definición).* lONDRES: Addison-Wesley Publishing Company.

VORLÄNDER, M. (2008). *Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality.* Aachen: Springer.

RWTH AACHEN UNIVERSITY. (abril de 2016). *Head Related Impulse Responses (HRTFs)*. Recuperado el 14 de junio de 2017, de RWTH AACHEN University: http://www.iks.rwth-aachen.de/en/research/speechaudio-communication/hrtfs/

WOZNIAK, T. (8 de abril de 2015). *Implementing Binaural (HRTF) Panner Node with Web Audio API.* Recuperado el 6 de junio de 2017, de Code & Sound: https://codeandsound.wordpress.com/2015/04/08/implementing-binaural-hrtf-panner-node-with-web-audio-api/

ZHONG, X.-L. (5 de marzo de 2014). *Head-Related Transfer Functions and Virtual Auditory Display*. Obtenido de InTechOpen: http://www.intechopen.com/books/soundscape-semiotics-localisation-and-categorisation/head-related-transfer-functions-and-virtual-auditory-display#F4

GUTIÉRREZ, E. (2 de nobiembre de 2009). *Introducción al filtrado digital.* Recuperado el 2017 de mayo de 2017, de Universitat Pompeau Fabra: http://www.dtic.upf.edu/~egomez/teaching/sintesi/SPS1/Tema7-FiltrosDigitales.pdf

MATÍNEZ, M. (2010). *Diseño de Filtros IRR.* Recuperado el 29 de mayo de 2017, de Universitat de València Open Course Ware: http://ocw.uv.es/ingenieria-y-arquitectura/filtros-digitales/tema_4_diseno_de_filtros_iir.pdf

SMITH, S. W. (1998). *The Sampling Theorem.* Recuperado el 6 de junio de 2017, de The Scientist and Engineer's Guide to Digital Signal Processing: http://www.dspguide.com/ch3/2.htm

DIATKINE, C. (2011). *FFT Size*. Recuperado el 2017 de junio de 2017, de Support IRCAM AudioSculpt: http://support.ircam.fr/docs/AudioSculpt/3.0/co/FFT%20Size.html

SMITH, S. W. (1998). *Chapter 12: The Fast Fourier Transform*. Recuperado el 2 de agosto de 2017, de The Scientist and Engineer's Guide to Digital Signal Processing: http://www.dspguide.com/ch12/2.htm

BERGLAND, G. D. (1969). A Guided Tour of the Fast Fourier Transform. *IEEE Spectrum , 6*, 41-52.

SMITH, S. W. (1998). *Chapter 18: FFT Convolution*. Recuperado el 26 de 11 de 2016, de The Scientist and Engineer's Guide to Digital Signal Processing: http://www.dspguide.com/ch18/2.htm

LUCIO NARANJO, J. F. (19 de mayo de 2014). Inteligencia computacional aplicada a la geración de respuestas impulsivas bi-auriculares y en aurilizacion de salas. *Ph.D. Dissertation* . Rio de Janeiro: Universidade do Estado do Rio de Janeiro.

MATHWORKS. (2017). *Documentation cart2pol*. Recuperado el 14 de junio de 2017, de MathWorks: https://www.mathworks.com/help/matlab/ref/cart2pol.html

SHEREINER, D. (2010). *OpenGL Programming Guide.* Pearson Education.

MERINO, M. (12 de julio de 2014). *¿Qué es una API y para qué sirve?* Recuperado el 2 de agosto de 2017, de TICbeat: http://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/

HOCK-CHUAN, C. (julio de 2012). *3D Graphics with OpenGL*. Recuperado el 7 de junio de 2017, de Basic Theory: https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

TECHTARGET. (abril de 2005). *What is bind*. Recuperado el 24 de julio de 2017, de Techtarget: http://whatis.techtarget.com/definition/bind

OOSTEN, J. V. (27 de enero de 2012). *Rendering Primitives with OpenGL*. Recuperado el 7 de junio de 2017, de 3D Game Engine Programming: https://www.3dgep.com/rendering-primitives-with-opengl/

SCULPTELO. (2017). *OBJ File : Color 3D Printing File Format*. Recuperado el 2 de agosto de 2017, de Sculpteo: https://www.sculpteo.com/en/glossary/obj-file-3d-printing-file-format/

DE VRIES, J. (2016). *Coordinate Systems*. Recuperado el 14 de junio de 2017, de Learn OpenGL: https://learnopengl.com/#!Getting-started/Coordinate-Systems

DE VRIES, J. (2016). *Camera*. Recuperado el 14 de junio de 2017, de Learn OpenGL: https://learnopengl.com/#!Getting-started/Camera

MICROSOFT. (2017). *¿Qué es el estado de rasterizador?* Recuperado el 14 de junio de 2017, de Developer Network: https://msdn.microsoft.com/es-es/library/ff604996(v=xnagamestudio.40).aspx

KRHONOS GROUP. (18 de mayo de 2011). *GluLookAt code*. Recuperado el 26 de julio de 2017, de OpenGL Wiki: https://www.khronos.org/opengl/wiki/GluLookAt_code

BENCINA, R. (13 de Agosto de 2011). *Real-time audio programming 101: time waits for nothing*. Obtenido de Ross Bencina: http://www.rossbencina.com/code/real-time-audio-programming-101-time-waits-for-nothing

TCR NETWORK. (2017). *Reference: atomic*. Recuperado el 14 de junio de 2017, de cplusplus.com: http://www.cplusplus.com/reference/atomic/

TCR NETWORK. (2017). *Reference: mutex*. Recuperado el 2017 de junio de 2017, de cplusplus.com: http://www.cplusplus.com/reference/mutex/mutex/

ORACLE. (2015). *What Is a Socket?* Recuperado el 14 de junio de 2017, de The Java™ Tutorials: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

WILLIAMS, A. (2012). *C++ Concurrency in Action: Practical Multithreading.* New York: Manning Publications Co.

GONZÁLEZ, A. J. (2006). *Ingeniería de Software*. Recuperado el 26 de julio de 2017, de Universidad Técnica Federico Santa María: http://profesores.elo.utfsm.cl/~agv/elo329/1s06/lectures/SoftwareEngineeringv2.pdf

BLE, C. (2013). *Capítulo 1. El agilismo*. Recuperado el 2017 de julio de 2017, de Libros Web - Diseño ágil con TDD: http://librosweb.es/libro/tdd/capitulo_1/modelo_en_cascada.html

STROUSTRUP, B. (2013). *The C++ Programing Language.* Ann Arbor: Adison-Wesley.

DE LUCAS, J. (2004). *Sistemas Operativos*. Recuperado el 27 de julio de 2017, de Linares: http://platea.pntic.mec.es/jdelucas/sistemasoperativos.htm

ROLI Ltd. (2017). *History and Development*. Obtenido de JUCE: https://www.juce.com/history-and-development

TRACKTION COORPORATION. (2017). *T7 Daw*. Obtenido de Tracktion: https://www.tracktion.com/products/t7-daw

PÉREZ, B. (2015). *Cuaderno Práctico de Linux. Sistemas Operativos Monopuesto.*

APPLE COMPUTER INC. (2017). *What's New in Xcode 9*. Recuperado el 14 de junio de 2017, de Developer Apple: https://developer.apple.com/xcode/

MICROSOFT. (2017). *Novedades de Visual Studio*. Recuperado el 14 de junio de 2017, de Visual Studio: https://www.visualstudio.com/es/?rr=https%3A%2F%2Fwww.google.com%2F

DOS GAMES ARCHIVES. (2017). *DOOM*. Recuperado el 14 de junio de 2017, de DOS Games Archives: http://www.dosgamesarchive.com/download/doom/

STORER, J. (2017). *Coding Standards*. Recuperado el 14 de junio de 2017, de JUCE: https://www.juce.com/learn/coding-standards

ÁLVAREZ, J. (07 de julio de 2010). *¿Qué son los plugins VST y cómo se instalan?* Recuperado el 14 de junio de 2017, de Future Music: http://www.futuremusic-es.com/que-son-los-plugins-vst/

ALGAZI, V. R. (24 de octubre de 2001). *Database, The CIPIC HRTF.* Recuperado el 14 de junio de 2017, de Semantic Scholar: https://pdfs.semanticscholar.org/cee9/f63da2cafe7dd7b8bd0752bea57f38d4afc5.pdf

KHOLODOV, I. (2017). *Matrices*. Recuperado el 14 de junio de 2017, de C-Jump: http://www.c-jump.com/bcc/common/Talk3/Math/Matrices/Matrices.html

DOUMLER, T. (9 de Octubre de 2017). *CppCon 2015: "C++ in the Audio Industry"*. Obtenido de Youtube CppCon: https://www.youtube.com/watch?v=boPEO2auJj4

BOURKE, P. (2012). *Object Files (.obj)*. Recuperado el 2 de agosto de 2017, de Paul Bourke: http://paulbourke.net/dataformats/obj/

AUDIOEASE. (30 de Noviembre de 2011). *Altiverb 7 guided tour*. Recuperado el 02 de agosto de 2017, de Youtube Audioease Channel: https://www.youtube.com/watch?v=EpzNgP8uThs

KLEINER, M. (1993). Auralization an overview. *J. Audio Eng. Soc. , 41*, 861.

LEE, P., & DUNCAN, S. (2016). *Virtual reality (VR): a billion dollar niche*. Recuperado el 29 de mayo de 2017, de Deloitte: https://www2.deloitte.com/global/en/pages/technology-media-and-telecommunications/articles/tmt-pred16-media-virtual-reality-billion-dollar-niche.html

LENGUA, I., & DUNAI , L. (01 de junio de 2013). *Dispositivo De Navegación Para Personas Invidentes Basado En La Tecnología Time Of Flight*. Recuperado el 01 de AGOSTO de 2017, de Scielo: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0012-73532013000300004

SMITH, S. W. (1998). *Chapter 5: Linear Systems*. Recuperado el 02 de agosto de 2017, de The Scientist and Engineer's Guide to Digital Signal Processing: http://www.dspguide.com/ch5/1.htm

AERTS, J., & DIRCKX, J. (13 de junio de 2009). Nonlinearity in eardrum vibration as a function of frequency and sound pressure. *Hearing Research 263* .

FLETCHER, T. (27 de marzo de 2012). *The Discrete Fourier Transform*. Recuperado el 2 de agosto de 2017, de CS 4640: Image Processing Basics : http://www.coe.utah.edu/~cs4640/slides/Lecture14.pdf

CARVAJAL, D. (2 de agosto de 2017). *Calavera Corp*. Recuperado el 2017, de Calavera Corp: http://calaveracorp.tumblr.com/

KUSKINA MAT. (2 de agosto de 2017). *Kuszkina Mat*. Recuperado el 2017, de Tumblr: http://kuzkinamatcrew.tumblr.com/

GARDNER, B., & MARTIN, K. (18 de mayo de 1994). *HRTF Measurements of a KEMAR DummyHead Microphone*. Recuperado el 26 de noviembre de 2016, de Bucknell University: http://www.linux.bucknell.edu/~kozick/elec32007/hrtfdoc.pdf

ARMENDÁRIZ, A., LUCIO NARANJO, J., ASIMBAYA, A., & PROAÑO, I. (2017). Módulo de Aurilización en Tiempo Real para Ambientes Virtuales Interactivos. *Revista PUCE* .

BENSOM, D., & MARTENS, W. (2006). Evaluating the principal spectral components positioning a virtual sound source on a cone of confusion. *J. Acoust. Soc. Am., v. 119, n. 5* , 3296.

OCULUS VR, LLC. (2 de agosto de 2017). *Localization and the Human Auditory System*. Recuperado el 2017, de Oculus Developer:

https://developer.oculus.com/documentation/audiosdk/latest/concepts/audio-intro-localization/#audio-intro-localization

SHIMOYAMA, R. (2012). Bio-inspired sound source localization compensated for sound diffraction by binaural head and torso. *2012 IEEE International Conference on Computational Intelligence and Cybernetics (CyberneticsCom)* .

CMS INFORMATION TECHNOLOGY. (27 de marzo de 2008). *Centers for Mediacare & Medicaid Services.* Recuperado el 28 de Marzo de 2016, de Selecting a Development Approach: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf

WEISSTEIN, E. W. (2017). *Fast Fourier Transform.* Recuperado el 6 de junio de 2017, de MathWorld--A Wolfram Web Resource: http://mathworld.wolfram.com/FastFourierTransform.html

STAPEL, E. (2012). *Cramer's Rule*. Recuperado el 03 de abril de 2017, de http://www.purplemath.com/modules/cramers.htm

SPORS, S. (2015). *Digital Signal Processing*. Recuperado el 26 de noviembre de 2016, de 6.3. Segmented Convolution: http://dsp-nbsphinx.readthedocs.io/en/nbsphinx-experiment/nonrecursive_filters/segmented_convolution.html

PRUNIER, J.-C. (2016). *Ray Tracing: Rendering a Triangle*. Recuperado el 03 de abril de 2017, de Scratchapixel: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection

ALONSO, G. (mayo de 2012). *Síntesis de respuesta impulsiva de recintos a travéz del método de trazado de rayos.* Recuperado el 29 de Mayo de 2017, de Universidad Tecnológica Nacional: http://www.profesores.frc.utn.edu.ar/electronica/fundamentosdeacusticayelectroacustica/pub/file/FAyE0912E1-Alonso-Budde-Zannier.pdf

SMITH, S. W. (1998). *Chapter 8: The Discrete Fourier Transform*. Recuperado el 26 de noviembre de 2016, de The Scientist and Engineer's Guide to Digital Signal Processing: http://www.dspguide.com/ch8.htm

MÖLLER, T., & TRUMBORE, B. (1997). Journal of Graphics Tools. *Department of Computer Science University of Virginia* .

RUIZ, F. J. (1999). *Propagación de la luz: índice de refracción y camino óptico*. Recuperado el 6 de septiembre de 2017, de INTEF: http://acacia.pntic.mec.es/~jruiz27/light/refracciones.html

RAE. (s.f.). *Diccionario Real Academia Española*. Recuperado el 6 de septiembre de 2017, de anecoico: http://dle.rae.es/srv/search?m=30&w=anecoico

FADGI. (s.f.). *Federal Agencies Digitization Guidelines Initiative Glossary*. Recuperado el 6 de septiembre de 2017, de Term: Visible spectrum: http://www.digitizationguidelines.gov/term.php?term=visiblespectrum

RENZE, J., & WEISTEIN, E. W. (s.f.). *Discrete Mathematics*. Recuperado el 6 de septiembre de 2017, de MathWorld: http://mathworld.wolfram.com/DiscreteMathematics.html

CUFF, P. (2017). *ELE 201: Information Signals*. Recuperado el 6 de septiembre de 2017, de Frequency Domain and Fourier Transforms: https://www.princeton.edu/~cuff/ele201/kulkarni_text/frequency.pdf

FADGI. (s.f.). *Federal Agencies Digitization Guidelines Initiative Glossary*. Recuperado el 6 de septiembre de 2017, de Term: Sampling rate (audio): http://www.digitizationguidelines.gov/term.php?term=samplingrateaudio

KHRONOS GROUP. (s.f.). *Primitive*. Recuperado el 6 de septiembre de 2017, de OpenGL Wiki: https://www.khronos.org/opengl/wiki/Primitive

TechTarget. (s.f.). *DEFINITION: integrated development environment (IDE)*. Recuperado el 6 de septiembre de 2017, de TechTarget: http://searchsoftwarequality.techtarget.com/definition/integrated-development-environment

dictionary.com LLC. (s.f.). *binaural*. Recuperado el 6 de septiembre de 2017, de Dictionary.com: http://www.dictionary.com/browse/binaural

KHRONOS GROUP. (2017 de septiembre de 2017). *Primitive*. Obtenido de OpenGL Wiki: https://www.khronos.org/opengl/wiki/Primitive