UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DIVISION DE ESTUDIOS DE POSGRADO

SINTESIS Y REPRODUCCION DE AMBIENTES ACUSTICOS VIRTUALES

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERIA ELECTRICA

(AREA COMUNICACIONES)

P R E S E N T A :

ANTONIO GUZMAN AVALOS

DIRECTOR: DR. FELIPE ORDUÑA BUSTAMANTE

CIUDAD UNIVERSITARIA

2000

RESUMEN

Autor Antonio Guzmán Ávalos
Título Síntesis y Reproducción de Ambientes Acústicos Virtuales

En esta tesis se describe el desarrollo del sistema de herramientas de cómputo y procesamiento de sonido llamado RA (*Ray Acoustics*), que tiene el propósito de proporcionar una plataforma para el diseño, análisis y la realización de ambientes acústicos virtuales. Este sistema consiste de tres grandes bloques: Simulación, Sintesis y Reproducción. El primero esta basado en una implantación de un método basado en acústica geométrica empleando técnicas de programación orientada a objetos, cuyo diseño se describe en detalle. En el segundo se emplean los datos generados en una simulación en conjunto con técnicas de procesamiento digital de señales para sintetizar una señal que contiene la información geométrica y acústica de espacios acústicos diversos presentados como un modelo computacional. Además, se hace que ésta incluya los efectos acústicos producidos por el torso, la cabeza y el oído de un oyente (virtual) para proveer los estímulos necesarios para producir una audición espacial. El último bloque contiene las funciones para hacer audible el campo sonoro sintetizado en el bloque anterior. La reproducción del campo sonoro se realiza en tiempo real, usando un algoritmo de convolución rápida.

PREFACIO

Esta investigación se llevó a cabo en el Laboratorio de Acústica del Centro de Instrumentos de la Universidad Nacional Autónoma de México durante el periodo 1998-2000.

Estoy profundamente agradecido con el Dr. Felipe Orduña Bustamante, mi director de tesis, por su aliento, guía, apoyo y tolerancia durante todo el tiempo que me llevo "terminar" este trabajo. También deseo agradecer al Maestro Ricardo Ruiz Boullosa por haberme introducido al área de la simulación de recintos, y con ello, haberme entusiasmado para continuar con mis estudios.

Finalmente, desde lo más profundo de mi corazón, quiero agradecer y expresar la gran deuda que tengo hacia mis padres, Sylvia Ávalos Estrada y José Antonio Guzmán Coeto, por su respaldo y paciencia; a ellos dedico este trabajo.

TABLA DE CONTENIDOS

LISTA DE ABREVIACIONES

2D	*Bidimensional*
3D	*Tridimensional*
BEM	*Boundary Element Method*
BIR	*Binaural Impulse Response*
CAD	*Computer Aided Design*
DDF	*Detected Data Format*
DSP	*Digital Signal Processing*
DWG	*Digital Waveguide*
DXF	*Drawing Interchange Format*
EDT	*Early Delay Time*
EIR	*Energy Impulse Responce*
FDTD	*Finite Difference Time Domain*
FEM	*Finite Element Method*
FFT	*Fast Fourier Transform*
FIR	*Finite Impulse Response*
GUI	*Graphical User Interface*
HRTF	*Head-Related Transfer Function*
ILD	*Interaural Level Difference*
IIR	*Infinite Impulse Respose*
ITD	*Interaural Time Difference*
OOP	*Object Oriented Programing*
RA	*Ray Acoustics*
RAF	*Ray-Acoustics Format*
RT	*Reverberation Time*
SEA	*Statistical Energy Analysis*
VBAP	*Vector Base Amplitude Panning*
WIIR	*Warped IIR*

1

INTRODUCCIÓN

Los escenarios de ambientes virtuales tienen como fin el someter a una persona a una situación en la cual sus sentidos perciban un ambiente que no corresponde a su ambiente real sino a uno diferente; uno virtual. Para alcanzar dicho objetivo, deben crearse artificialmente las señales que estimulen los sentidos para establecer el ambiente virtual deseado. Puesto que los seres humanos poseen varios modos de sensar su espacio, y su posición dentro de éste (visual, auditivo, táctil, gustativo, olfativo y coordinador), la creación de ambientes virtuales es una tarea multimodal. Sin embargo, dependiendo de la aplicación, varios modos de sensado pueden establecerse en términos de su relevancia. Por lo tanto, algunos modos pueden ser tratados con menos autenticidad o incluso ser ignorados.

El sentido del oído es de suprema importancia. Según estudios con base en la experiencia psicológica de personas que perdieron el sentido del oído, la sordera repentina implica una desconexión total entre el observador y su ambiente [1]. Este tipo de estudios llevó a distinguir tres niveles de información auditiva [2]. En el primer nivel de información están los sonidos usados para comunicarse con otras personas. Como ejemplos de este nivel simbólico están el lenguaje, la música, etc. Los sonidos puestos en el segundo nivel son los que tienen una función de advertencia, por ejemplo, el timbre del teléfono, el ruido de una sirena, etc. El tercer nivel, llamado nivel primitivo, consiste de todos los sonidos que no encajan en alguna de las primeras dos categorías. Ellos sirven como sonidos de fondo, los cuales nos rodean en nuestra vida diaria. Estos sonidos pueden ser causados por la interacción con objetos (*e.g.* escribir sobre un teclado, pisadas) o pueden consistir de sonidos incidentales de objetos en el ambiente. (*e.g.* el tictac de un reloj). Todos estos sonidos mantienen la sensación

de ser parte de un mundo vivo. De acuerdo a estos estudios, la pérdida de sonidos de nivel primitivo es la mayor causa de depresión y pérdida de la sensación de presencia reportada por gente que ha perdido el sentido del oído repentinamente.

Por todo lo anterior, la realización intencional de ambientes virtuales debe contemplar un tratamiento apropiado del modo de sensado auditivo, es decir, se debe incluir una manera de reproducir eventos sonoros con el fin de crear escenarios auditivos virtuales.

1.1 Antecedentes

Tradicionalmente, la reproducción de sonido se ha hecho monofónica o estereofónicamente. Por estos medios el sonido se percibe como si viniera de una fuente puntual o de una línea que conecta a dos fuentes puntuales, siempre y cuando se esté dentro de un ambiente carente de ecos o reverberación; un ambiente anecóico. Para crear escenarios auditivos más naturales, se requiere una técnica en la cual el sonido pueda emanar, o dé la sensación de emanar de cualquier dirección.

Para alcanzar dicho propósito, algunas técnicas han sido desarrolladas. Éstas pueden ser organizadas en dos grupos:

- Técnicas multicanal
- *Técnicas biauriculares*

Las técnicas multicanal están basadas en una adecuada simulación del campo sonoro utilizando altavoces. Estas técnicas emplean arreglos de varias bocinas, las cuales son alimentadas con diferentes señales (ver *Dolby-Surround Sound* [3]). Dentro de este grupo se encuentran también técnicas tales como la eudofonía [4], *ambisonic* [5,6,7] y paneo de amplitud basado en vectores (VBAP) [8].

Las técnicas biauriculares están basadas en el principio de que para producir un evento auditivo correspondiente a la situación sonora original, es necesaria una reproducción autentica de las señales de entrada a los dos oídos. Una onda sonora arribando desde una dirección y distancia dada, resulta en dos señales de presión sonora, una en cada oído. La transmisión está descrita en dos conjuntos de funciones de transferencia que incluyen cualquier distorsión lineal,

tales como coloración y diferencias de tiempo y espectro inter-auricular [9]. Para esta técnica se emplean, preferentemente, audífonos, o bocinas estereofónicas con filtros de cancelación de *cross-talk* dentro de una cámara anecóica [10] o un recinto ecualizado [11].

Estas técnicas, referidas como sonido espacial o tridimensional (3D), son usadas en aplicaciones que van desde juegos de computadora hasta salas de cine. El sonido 3D está bien adaptado para la reproducción de campos sonoros virtuales. Esto da la posibilidad de crear escenarios auditivos de inmersión total. Entre otras posibles áreas de aplicación para los escenarios auditivos virtuales se encuentran, por ejemplo, tele y vídeo conferencia, acústica arquitectónica y aplicaciones aeronáuticas.

1.2 Modelado de Ambientes Acústicos Virtuales

Considérese una situación auditiva común, un sonido emitido por una fuente sonora en algún lugar dentro de un espacio rodeado por paredes. Como resultado de la presencia de las paredes, las cuales representan a superficies reflectoras de sonido, surge un campo sonoro complejo. Cabe decir que dentro de este campo sonoro se encuentra codificado las propiedades físicas y geométricas del ambiente acústico. Un receptor de sonido que obtiene muestras del campo sonoro en cierta posición recibirá información no sólo sobre la señal emitida por la fuente, sino también información sobre el ambiente acústico.

Si el sensor del campo sonoro es una persona, el sonido que llega a ésta es transformada a través del oído externo (oreja, cabeza y torso) en dos señales diferentes al arribar a cada tímpano. Durante el proceso de transformación, la información espacial del campo sonoro se convierte en información temporal y espectral en las señales del tímpano. Por consiguiente, el sistema auditivo humano (considerando al cerebro parte de este sistema) puede decodificar esa información y usarla para formar una percepción auditiva. La influencia del ambiente acústico es reflejada principalmente en atributos espaciales de lo que se percibe auditivamente como dirección, distancia, impresión espacial y extensión [12].

De lo anterior, se distinguen tres partes que deben ser modeladas para crear ambientes acústicos virtuales:

- La fuente sonora
- La acústica del recinto
- El oyente

Estos puntos forman una cadena fuente-medio-receptor, la cual es típica en cualquier modelo de comunicación. Ésta es llamada también cadena de auralización*, la cual se ilustra en Fig. 1.1. Auralización se refiere al proceso de hacer audible el campo sonoro generado por una fuente dentro de un espacio, de tal forma que se produzca la sensación auditiva de estar inmerso en dicho campo [13].

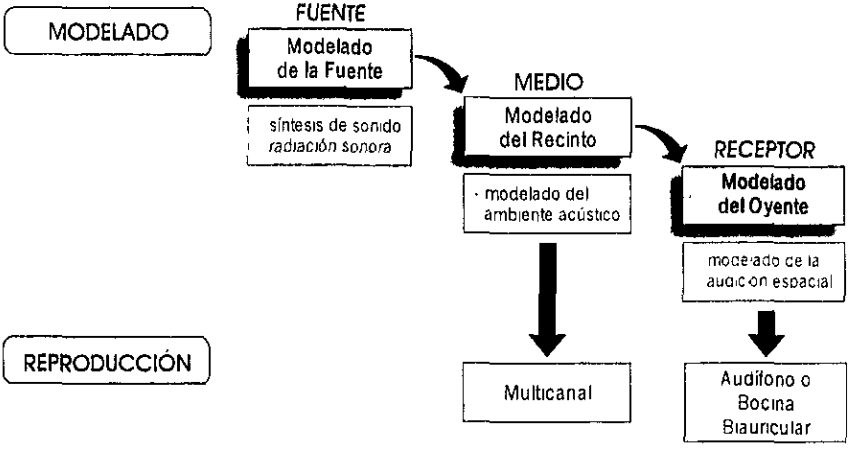

Figura. 1.1: El proceso de implantación de ambientes acústicos virtuales consiste de tres tareas de modelado separadas. Los componentes a ser modelados son la fuente sonora, el medio y el receptor [14].

El modelado de fuentes consiste de los siguientes dos puntos:

- Síntesis de sonido
- Radiación sonora

* Término tomado de la palabra inglesa *auralization*, éste es introducido para ser usado en analogía con visualización.

La síntesis de sonido ha sido ampliamente estudiada y se han dado varios progresos en este tema. Ésta se ha aplicado principalmente para sintetizar el sonido de instrumentos musicales. Una técnica interesante es el modelado físico, la cual imita el proceso de la generación del sonido [15,16]. La técnica de modelado físico comúnmente usada es el método de guía de onda digital (*digital waveguide method*, en inglés) [16-19]. La forma más simple para modelar fuentes es asumiendo que son puntuales y omnidireccionales. Típicamente, la directividad de las fuentes sonoras depende de la frecuencia, por lo que debe ser modelada para producir resultados reales. Por ejemplo, algunos instrumentos musicales radian sonido de alta frecuencia hacia el hemisferio frontal del músico y las bajas son omnidireccionales. Otro importante aspecto de la radiación sonora es la forma de la fuente. La mayoría de las fuentes pueden modelarse como fuentes puntuales, pero, por ejemplo, un piano de cola es demasiado grande como para modelarse como una sola fuente puntual. Se pueden encontrar amplios estudios sobre la física de instrumentos musicales y su radiación sonora en [20,21].

El proceso de codificar parámetros del ambiente acústico en las características de un campo sonoro puede ser modelado en una computadora usando una metodología llamada simulación de recintos [22]. En ésta se modela la propagación del sonido en un medio; generalmente aire. Aquí se toma en cuenta las trayectorias del sonido directo y las primeras reflexiones, la atenuación causada por el aire y las fronteras del recinto, y la parte difusa de la reverberación.

En una reproducción multicanal, el campo sonoro se crea usando varias bocinas alrededor de una persona dentro de una cámara anecóica. Un efecto similar es creado con reproducción biauricular, en la cual es necesario modelar al oyente (Fig. 1.1) para considerar las propiedades de la audición espacial humana. Una simple manera de proveer una sensación direccional se da modelando la diferencia del nivel interauricular (ILD) y la diferencia del tiempo interauricular (ITD). Para auralizaciones de alta calidad se necesita la función de transferencia de una cabeza (HRTF) de un sujeto o de un modelo artficial, las cuales modelan las reflexiones y filtrado de la cabeza, orejas y torso del oyente así como la dependencia en frecuencia del ILD y ITD [9,13,23]. La combinación de la simulación de recintos con las técnicas biauriculares se le conoce con el nombre de simulación biauricular de recintos [12].

1.3 Sistema RA

Las tres faces de modelado de la figura 1.1 permiten la creación de ambientes acústicos virtuales. En esta tesis se describe el desarrollo del sistema de herramientas de cómputo y procesamiento de sonido llamado RA (*Ray Acoustics*), que tiene el propósito de proporcionar una plataforma para el diseño y la realización de ambientes acústicos virtuales. Este sistema cubre las tres faces de modelado antes mencionadas, empleando técnicas basadas en acústica geométrica para el modelado de ambientes acústicos, y procesamiento digital de señales (DSP) para la producción del sonido.

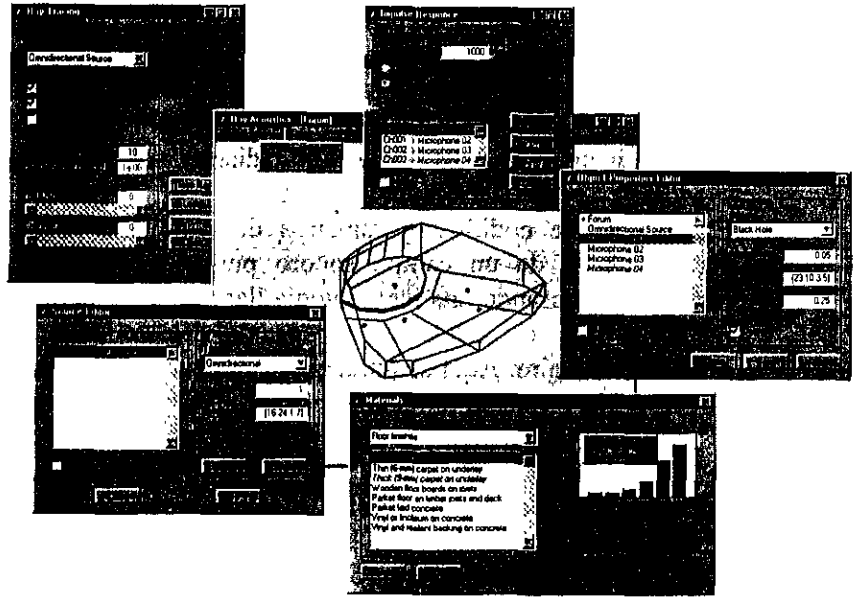

Figura 1.2. Interfaz gráfica del sistema RA. Aquí se muestra las principales herramientas del sistema usadas para modelar espacios acústicos.

El sistema RA se diseñó con la idea de producir un conjunto de herramientas de programación que pudieran evolucionar conforme a nuevos requerimientos, y a su vez sirvieran como estructura para desarrollar otras nuevas. Bajo esta visión, el sistema quedó implantado en tres bloques, clasificados de acuerdo a las tareas que en ellos se realizan (ver Fig. 1.3).

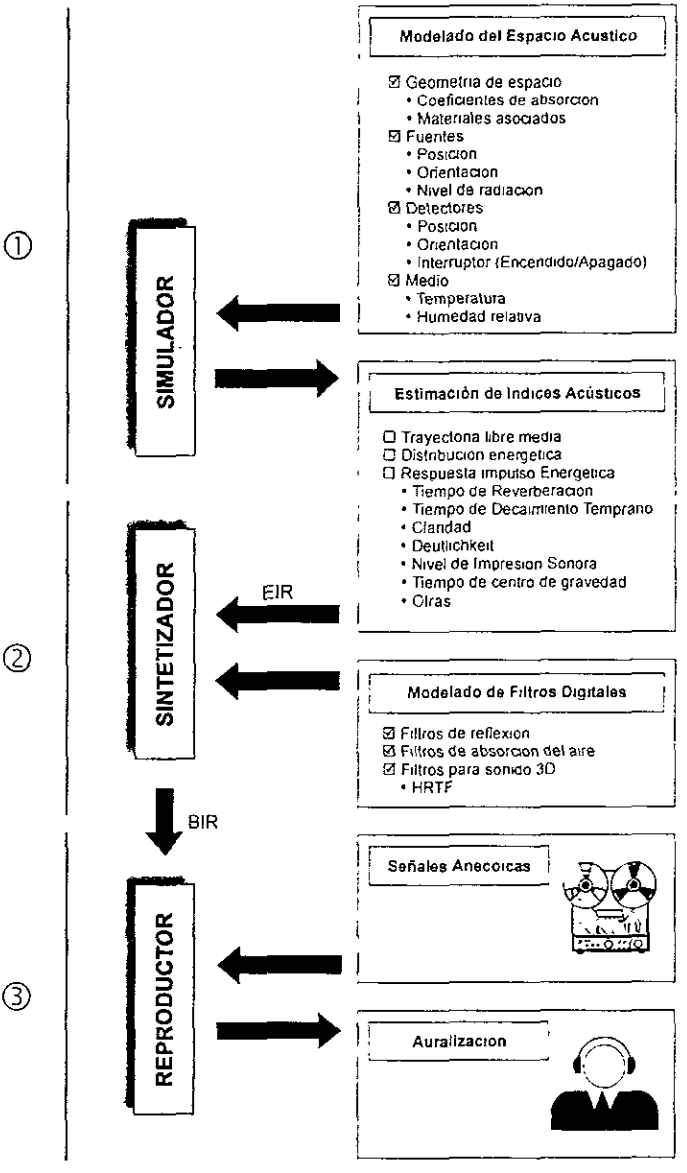

Figura. 1 3. Esquema de la estructura del sistema RA. Aquí se muestra los tres bloques que comprenden al sistema así como los módulos y operaciones que conforman cada bloque. Se presenta además la manera en como fluye y se transforma la información sobre el espacio acústico a través de los diferentes bloques para realizar auralizaciones.

En el primer bloque se llevan a cabo las operaciones de simulación. Entre éstas se encuentran el modelado de espacios acústicos*, estimación de índices acústicos, manipulación y análisis de datos. La parte medular del este bloque fue implantado empleando técnicas de programación orientada a objetos (OOP) a través del lenguaje C++. Éstas permiten modelar cada elemento y las relaciones que se definen entre ellos durante el proceso de simulación de un campo sonoro en el interior de un espacio circunscrito. La información sobre la arquitectura del espacio es extraída directamente de archivos con formato DXF, el cual es ampliamente usado en sistemas de diseño asistido por computadora (CAD). Dicha información es traducida a un nuevo formato, en el cual se presenta en forma explícita cada elemento geométrico del espacio, referencias sobre sus propiedades físicas, condiciones del medio, fuentes y detectores con sus respectivas características de radiación y sensado. Para facilitar el manejo de las operaciones de esté bloque, se hizo que el sistema trabajara a través de MATLAB (ver Fig. 1.2). La decisión de usar esta plataforma se tomó con el fin de aprovechar los recursos de visualización y de cálculo que MATLAB ofrece.

Dentro del segundo bloque se encuentran las operaciones de síntesis. Aquí se construye la respuesta impulso biauricular (BIR), a través de un conjunto de funciones que modelan la manera en que el medio, la geometría del espacio y los materiales que lo conforman, así como la oreja, cabeza y torso del oyente, van modificando espectral y temporalmente un impulso radiado a través de una fuente hasta alcanzar los tímpanos del sujeto. Todo este proceso se realiza por medio de una serie de funciones programadas en MATLAB.

El último bloque contiene las funciones para hacer audible el campo sonoro sintetizado en el bloque anterior. La reproducción del campo sonoro se realiza en tiempo real, usando un algoritmo de convolución rápida, en un sistema DSP basado en el procesador TMS320C40.

1.4 Alcances de la Tesis

En esta tesis se presenta paso a paso la implantación de cada bloque que conforma al sistema RA, poniendo especial atención al diseño del

* En este documento el término "espacio acústico" será empleado como sinónimo de recinto.

simulador, el cual es parte esencial de este sistema. En la sección de síntesis se contempla en detalle las técnicas empleadas para el diseño de los filtros que modelan la atenuación sonora por absorción del aire y la reflexión del sonido sobre diferentes materiales, y de qué manera se emplean estos, en conjunto con filtros de sonido 3D, para obtener la respuesta impulso biauricular de un modelo de recinto. Además, se plantea una forma de implementar eficientemente un algoritmo de convolución rápida.

El lector de esta tesis podrá examinar de forma analítica y auditiva la acústica de un modelo de auditorio. Esta tesis puede ser empleada como guía para introducirse al campo de la acústica virtual ya que se proporciona varias referencias sobre temas relacionados.

1.5 Contenido de la Tesis

Esta tesis está organizada como sigue: En el Capítulo 2 se muestra un panorama general de las diferentes técnicas para modelar campos sonoros. En el Capítulo 3 se presenta el diseño e implantación, empleando programación orientada a objetos, de un programa para modelar campos sonoros a través del método de trazado de rayos. En el Capítulo 4 se exhibe el uso del simulador y el tratamiento de los datos que éste genera para analizar la acústica de un recinto. En el Capítulo 5 se muestra como sintetizar un campo sonoro y la manera de hacerlo audible usando recursos de procesamiento digital de señales. El Capítulo 6 expone las conclusiones de la tesis.

2

SIMULACIÓN DE RECINTOS

La simulación de recintos es una metodología empleada para modelar campos sonoros dentro de espacios circunscritos. Ésta envuelve varios procesos para codificar parámetros del ambiente acústico en las caracteristicas de un campo sonoro. Sus orígenes se encuentran en la planeación de la arquitectura de auditorios, teatros y salas de conciertos donde la predicción de la distribución del sonido es de suma importancia. Hoy en día esta metodología se emplea no tan sólo para hacer análisis acústicos, sino también para la creación de ambientes sonoros virtuales.

2.1 Representación del Campo Sonoro

Cuando una persona se encuentra dentro de un espacio delimitado por superficies reflectoras y éste es excitado acústicamente por una fuente (ver Fig. 2.1), la persona recibirá señales sonoras directamente de la misma, pero además, debido a la existencia de superficies reflectoras, una variedad de señales de sonido reflejado llegará al oyente desde diferentes direcciones, con diferentes tiempos de arribo y diferentes formas espectrales.

Si se marcan los tiempos de arribo de varias reflexiones con líneas rectas perpendiculares sobre un eje horizontal de tiempo y se le asignan a éstas alturas proporcionales a la amplitud relativa de las reflexiones, lo que se obtiene es la respuesta impuso energética (ver Fig. 2.2). Ésta contiene toda la información significativa en la estructura temporal del campo sonoro en un cierto punto del recinto.

La respuesta impulso se puede descomponer en dos partes, las cuales tienen diferentes efectos subjetivos. La primera consiste del sonido

directo y la respuesta temprana (primeras reflexiones). Su estructura temporal y direccional es de crucial importancia para la sensación de sonoridad, claridad, extensión y otras [24]. Es también responsable de las variaciones de lo que se percibe auditivamente en diferentes partes del recinto. Esta parte consiste principalmente de picos agudos provocados por reflexiones especulares desde las superficies o porciones de ellas, pero frecuentemente viene acompañada de un fondo difuso causado por dispersiones sonoras ocasionadas por las irregularidades de las paredes, esquinas, asientos, etc.

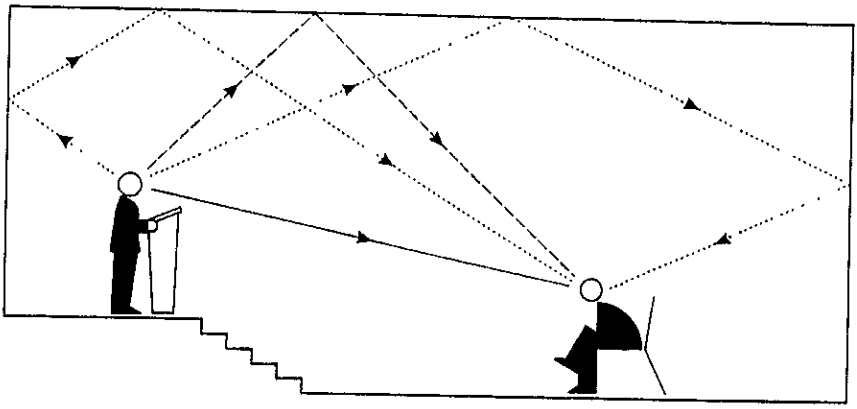

Figura. 2.1. Recinto de geometría simple y visualización del sonido directo (línea sólida) y reflexiones de primer (línea discontinua) y segundo orden (línea punteada).

La parte restante, la respuesta tardía, consiste de numerosas reflexiones superpuestas. Esto es lo que se conoce como reverberación. Ésta es también importante para la impresión auditiva, sólo que aquí se toma en cuenta el grueso de sus características espectrales y temporales en vez de su estructura detallada. Por lo tanto, la reverberación no contiene mucha información sobre la estructura direccional del campo sonoro. Generalmente, la distribución direccional del sonido temprano es muy irregular, pero conforme se incrementa el tiempo de retardo, la distribución direccional se vuelve cada vez más uniforme (difusa) debido al efecto de las reflexiones múltiples en las paredes del recinto, el cuál puede ser modelado como un proceso aleatorio. El campo sonoro dentro de un recinto promedio puede ser considerado difuso después de 100 a 150 ms [25]. Por tanto, el tiempo característico t_c

que separa la respuesta temprana de la tardía debería estar dentro de este rango.

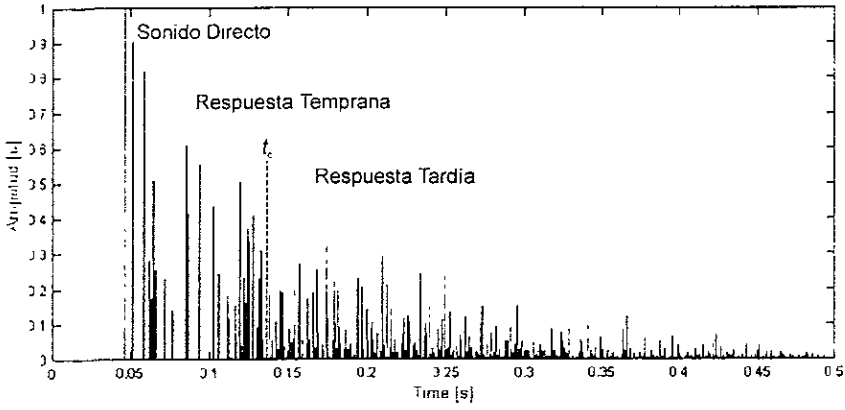

Figura 2.2. Respuesta impulso energética de un recinto (modelada con el método de trazado de rayos). En la respuesta impulso se distinguen tres partes: El sonido directo, la respuesta temprana y la respuesta tardía; ésta última empieza a partir de un tiempo característico t_c.

El objetivo de cualquier técnica para modelar la acústica de un recinto es el de obtener la respuesta impulso, ya que en ésta se encuentran codificadas las propiedades acústicas y geométricas.

Las herramientas de modelado utilizadas en la simulación de recintos pueden ser clasificadas en dos grupos (ver la Fig. 2.3):

- Métodos con modelos a escala
- Métodos por computadora

Los modelos a escala son ampliamente usados en el diseño de grandes salas de conciertos, puesto que es una técnica bien establecida para la medición de atributos acústicos. Estos son apropiados también para auralizaciones en puntos de medición predefinidos, empleando técnicas de auralización directa e indirecta [13]. El empleo de computadoras en el campo de la simulación de recintos se ha dado por más de 40 años [26-29]. Hoy en día, el uso de métodos computacionales combinados con modelos a escala, es una práctica relativamente común en el diseño de salas de concierto.

Matemáticamente, la propagación del sonido es descrita por la ecuación de onda, también conocida como la ecuación de Helmholtz. La respuesta impulso de una fuente a un punto de detección se obtiene en principio resolviendo la ecuación de onda, pero ésta raras veces se lleva a cabo en forma analítica debido a la dificultad para formular las condiciones de frontera. Por lo tanto, la solución debe darse en forma aproximada.

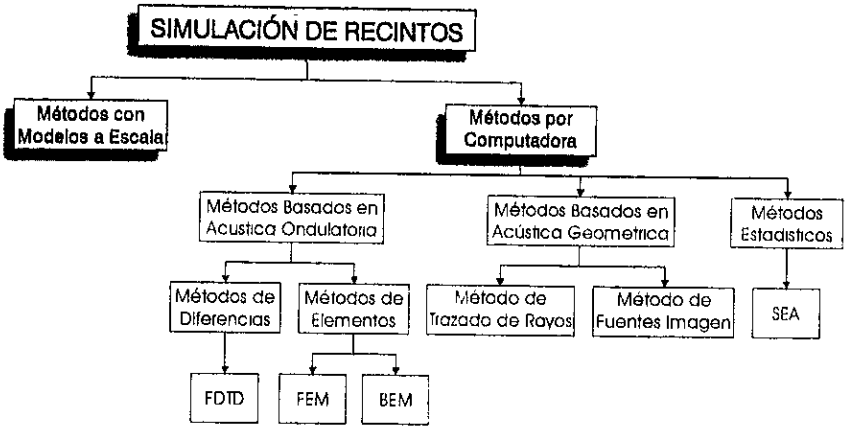

Figura 2.3. Los métodos empleados en la simulación de recintos están basados en mediciones de alta frecuencia en modelos a escala, soluciones de la ecuación de onda (acústica ondulatoria), en rayos sonoros (acústica geométrica) o en alguna técnica estadística. Diferentes métodos pueden ser empleados conjuntamente para formar un modelo híbrido.

Existen tres diferentes procedimientos en los métodos por computadora para modelar campos sonoros (ver Fig.2.3).

- Métodos basados en acústica ondulatoria
- Métodos basados en acústica geométrica
- Métodos estadísticos

Los métodos basados en acústica geométrica son las técnicas de modelado empleadas con mayor frecuencia. Recientemente técnicas basadas en acústica ondulatoria, tales como el método de elementos finitos (FEM), el método de elementos de frontera (BEM) y métodos de diferencias finitas en el dominio del tiempo (FDTD) han ganado mayor interés. Estas técnicas son adecuadas sólo para simulación a

bajas frecuencias, en contraste con las técnicas basadas en acústica geométrica que son más aptos para modelar el comportamiento acústico a altas frecuencias.

Los métodos de modelado estadístico, tales como el análisis de energía estadística (SEA) [30], son principalmente aplicados en la predicción de niveles de ruido en sistemas acoplados, en los cuales la transmisión de sonido por estructuras es un factor importante. Estos métodos no son apropiados para propósitos de auralización ya que ellos no modelan el comportamiento temporal del campo sonoro.

Los métodos con modelos a escala tienen una gran ventaja sobre los métodos anteriores. Estos son capaces de modelar correctamente fenómenos físicos complicados tales como dispersión y difracción, al contrario de los métodos por computadora que usan modelos aproximados, algunos de los cuales ignoran estos fenómenos.

2.2 Métodos con Modelos a Escala

Los primeros intentos para modelar los campos sonoros que ocurren en recintos de geometría compleja se hicieron empleando modelos a escala. Estos se usaron primeramente para observar y estudiar la propagación del sonido y evaluar ciertos atributos acústicos en diseños de salas de concierto [24]. Posteriormente, fueron empleados para llevar a cabo los primeros experimentos en la creación de espacios sonoros virtuales [31].

En un modelo a escala de un recinto se pueden medir y grabar las variaciones de presión sonora en diferentes puntos dentro de éste. La longitud de onda de la señal radiada dentro del modelo debe guardar una relación con la longitud de onda original, así como las dimensiones del modelo con las del recinto. A esta técnica se le llama escalamiento de frecuencia [32,33]. Por lo tanto el campo sonoro resultante es un modelo a escala del original. La aseveración anterior es cierta siempre y cuando las dimensiones escaladas del modelo acústico correspondan en forma precisa con el verdadero, y que se puedan reproducir las propiedades físicas del medio y de los materiales para altas frecuencias.

Existen dos formas de hacer audible al campo sonoro empleando modelos a escala. Una es de forma directa [34], aquí se radia directamente señales de audio al recinto, las cuales son recogidas por un micrófono y transmitidas a un dispositivo de almacenamiento. La reproducción se realiza escalando la distribución frecuencial de las señales grabadas. Este proceso puede hacerse en tiempo real o fuera de línea. Otra manera es en forma indirecta [31,35], esto se lleva a cabo midiendo la respuesta impulso del modelo; existen varias técnicas para este propósito [36,37]. La reproducción se realiza escalando en tiempo la respuesta impulso y convolucinando la respuesta resultante con señales anecóicas.

En años recientes, la técnica de modelos a escala ha madurado, e incluso es posible escuchar biauricularmente dentro de tales modelos, empleando sensores con forma de cabeza a escala [31,34,35]. Sin embargo, debido al considerable esfuerzo para construir los modelos, estos son empleados principalmente para trabajar en problemas de acústica de recintos de alguna relevancia comercial, tales como el diseño de salas de concierto. Para la creación de ambientes sonoros virtuales no se disponen generalmente de ellos.

2.3 Métodos Basados en Ondas

Cuando la geometría del recinto es simple (rectangular con paredes rígidas), el modelado del campo sonoro dentro de un recinto se realiza analíticamente de forma directa [38]. Para una geometría compleja, el modelado del campo sonoro puede hacerse resolviendo la ecuación de onda para condiciones de frontera complejas dadas por las superficies circundantes. Sin embargo, una solución analítica general es difícil de encontrar, ya que la formulación de las condiciones de frontera es extremadamente complicada (ver [39]). Por lo tanto, deben ser aplicados métodos numéricos basados en la ecuación de onda. Los métodos de elementos, tales como FEM y BEM, son apropiados sólo para recintos pequeños a bajas frecuencias debido a los grandes requerimientos de cómputo [13,40]. La principal diferencia entre estos dos métodos radica en la estructura del elemento. En FEM el espacio completo tiene que ser discretizado con elementos, mientras que en BEM sólo las fronteras del espacio son discretizadas. En la práctica esto implica que las matrices usadas por una solución FEM son grandes y ralas (poco densas), mientras que una matriz BEM son

pequeñas y muy densas. Particularmente, los cálculos en el BEM consumen mucho tiempo [41].

Los métodos en diferencias finitas en el dominio del tiempo (FDTD) proveen otra posible técnica para la simulación de recintos [42,43]. Esta técnica está basada esencialmente en una aproximación en diferencias finitas para las derivadas en tiempo y espacio de la ecuación de onda [44]. Los métodos FDTD producen respuestas impulso mejor adaptadas para auralizaciones, en contraste con los métodos de elementos, los cuales calculan respuestas en el dominio de la frecuencia [13].

El método de guía de ondas digitales es una variante de los métodos FDTD, con la diferencia que su origen se encuentra en el campo del procesamiento digital de señales (DSP) [45,46]. Una guía de onda digital (DWG) es una línea de retardo digital bidireccional. El estado del sistema se obtiene sumando las dos componentes de onda que viajan en direcciones opuestas. Puede demostrarse que una DWG es una solución exacta de la ecuación de onda arriba del límite de Nyquist [47]. Una malla de guías de onda digitales es un arreglo regular discretamente espaciado de DWG, acomodadas a lo largo de cada dimensión y conectadas en sus intersecciones. Cada unión es un punto sin pérdidas de igual impedancia donde la suma de las entradas es igual a la suma de las salidas, esto es, el flujo debe sumar cero, y las señales en cada cruce son iguales en el punto de unión, debido a que tienen la misma impedancia [48]. Con base en estas condiciones se puede derivar una ecuación en diferencias para los nodos de una malla rectangular 3D, la cual es equivalente a la ecuación de Helmholtz discretizada en el tiempo y espacio [49]. Este método es computacionalmente más eficiente que FEM, a pesar de que se requiere una malla más densa para obtener resultados más exactos [43]. Otro método similar puede obtenerse empleando filtros digitales de ondas multidimensionales [50] los cuales han sido usados, por ejemplo, para resolver ecuaciones diferenciales parciales en general.

El principal beneficio al usar el método de elementos sobre métodos FDTD es que uno puede crear una estructura de malla más densa donde se requiera, como para posiciones cercanas a esquinas u otros lugares acústicamente desafiantes (difíciles de modelar). Otra ventaja de los métodos de elementos es la facilidad que ofrecen en la

construcción de modelos acoplados, en los cuales varios medios de propagación de ondas son conectados unos con otros.

En todo método basado en acústica ondulatoria, la parte más difícil es la definición de las condiciones de frontera. Típicamente una impedancia compleja es requerida, pero es difícil encontrar esos datos en la literatura existente.

Aunque recientemente se han hecho progresos al usar el método de elemento finito de tiempo interactivo, estos métodos están restringidos para recintos pequeños a bajas frecuencias, debido al poder computacional requerido.

2.4 Métodos Basados en Acústica Geométrica

La acústica geométrica se basa en una solución especial de la ecuación de onda, la cual sólo es válida si la longitud de onda del sonido es pequeña comparada con las dimensiones geométricas de las superficies circundantes del recinto, y grande comparada con las irregularidades de éstas (esto se observa a partir de una frecuencia media de 1000 Hz, que corresponde a una longitud de onda de 34 cm). Si esta condición se cumple, las ondas de presión pueden ser tratadas como rayos sonoros.

Un rayo sonoro se puede interpretar como una porción de una onda esférica que se origina en cierto punto y cuya apertura tiende a cero. Éste tiene una dirección de propagación bien definida y está sujeta a las mismas leyes que un rayo de luz (aunque con diferente velocidad de propagación). De estas leyes, sólo es de importancia la ley de reflexión. La transición a otro medio y la refracción asociada no se considera en la acústica geométrica de recintos; tampoco se curvan los rayos, ya que el medio se considera homogéneo*. Por otro lado, la velocidad de propagación finita debe considerarse bajo toda circunstancia, dado que es la responsable de muchos efectos tales como la reverberación, ecos, etc. El fenómeno de difracción se omite dado que la propagación en línea recta es su principal postulado. Del mismo modo, no se considera la interferencia, es decir, si varios componentes de campos sonoros son superpuestos, sus relaciones de

* Estas condiciones son propias de recintos de tamaño promedio tales como salas de conciertos, teatros, auditorios, cines, salas de conferencias y cuartos de televisión.

fase mutua no se toman en cuenta; en lugar de eso, simplemente se suman sus densidades de energía o sus intensidades.

Es evidente que el modelado por medio de acústica geométrica no puede incluir efectos de propagación de onda tales como reflexiones difusas o difracción. Particularmente, las reflexiones difusas juegan un importante papel en la acústica del recinto. Sin embargo, como se verá más adelante, pueden aplicarse algunas técnicas para imitar, en cierta medida, tales efectos.

Básicamente existen dos diferentes algoritmos para modelar campos sonoros usando acústica geométrica: El método de imágenes [51,52] y el método de trazado de rayos [27,53]. Estos están restringidos para recintos formados por superficies planas; pero en la práctica, si se encuentran superficies curvas, éstas se aproximan con una serie de superficies planas. Ambos difieren en sus métodos de cálculo, pero si se les permiten cálculos ilimitados llegan al mismo resultado [54].

Método de Fuentes Imagen

Este método es inherentemente determinístico y permite un cálculo exacto de respuestas impulso cuya longitud esta limitada por la potencia de cómputo disponible.

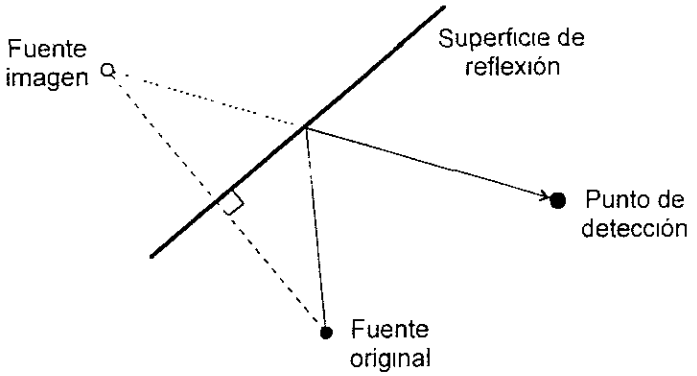

Figura 2.4. Construcción de una fuente imagen. Se puede observar que la fuente imagen y la original mantienen la misma distancia respecto a la superficie de reflexión.

La idea fundamental del método de fuentes imagen es la siguiente: Se coloca una fuente puntual frente de una superficie rígida, el campo

sonoro en cualquier punto frente a la superficie puede ser calculado sumando el campo sonoro causado por la fuente original en ese punto y el campo sonoro causado por una fuente secundaria. La fuente secundaria, referida en la literatura como fuente imagen, se obtiene reflejando la fuente original en la superficie (ver Fig. 2.4). Cabe hacer notar que la distancia recorrida a través de la trayectoria de reflexión hasta el punto de detección o de sensado, es la misma que se recorre desde la fuente imagen a dicho punto. Cuando se presenta más de una superficie ocurren varias reflexiones, lo cual generan fuentes imagen de orden múltiple.

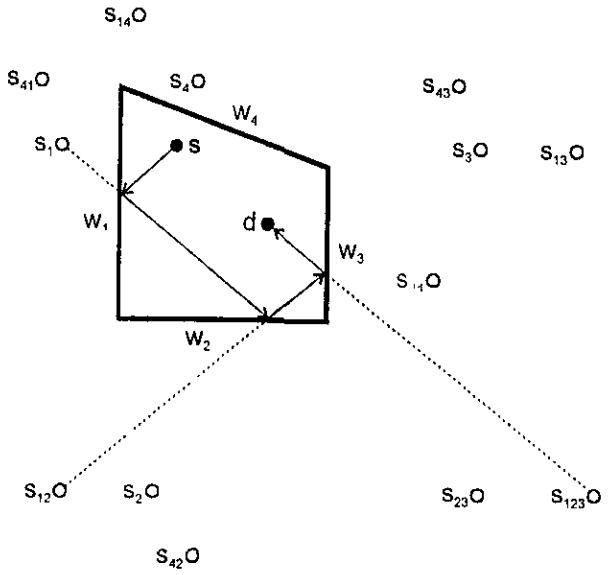

Figura 2.5. Construcción de algunas fuentes imagen de primer, segundo y tercer orden. La fuente imagen s_{34} no es válida puesto que W_4 no es visible desde s_3. A partir de una fuente imagen de orden tres, se trazó la trayectoria de un rayo desde la fuente hasta un punto de detección.

Para encontrar la trayectoria de un rayo sonoro desde la fuente origen hasta un cierto punto de detección dentro de un recinto, se aplica el siguiente proceso: Se empieza construyendo fuentes imagen de la original respecto a todas las paredes (superficies rígidas planas) que rodean al recinto, procediéndose de la misma forma con cada una de las fuentes secundarias generadas, dejando aparte aquella pared respecto a la cual se creó la última. Este proceso se sigue hasta un

orden predeterminado de fuentes imagen. Por lo tanto, una fuente imagen de quinto orden representa un rayo que sufrió cinco reflexiones. Una forma sencilla de manejar las fuentes es enumerándolas con índices de las paredes que estuvieron involucradas en el proceso de reflexión; el número de índices es igual al orden de la fuente imagen (ver Fig. 2.5). Cada fuente es el ultimo eslabón de una cadena de fuentes imagen sucesivas y con sus índices se puede leer cual pared fue golpeada a través de la trayectoria asociada a un rayo.

Para obtener la respuesta impulso energética se debe calcular primeramente el retardo de cada rayo sonoro que arriba a un punto de detección dado. Para esto solamente se divide la distancia entre cada una de las fuentes imagen, incluyendo la fuente origen, y el punto de detección, entre la velocidad de propagación. Una vez encontrados los tiempos de arribo, se deben determinar las amplitudes de cada rayo. Normalmente, para el método de imágenes se considera dos factores para la atenuación: La atenuación inversamente proporcional a la distancia a la que se encuentra la fuente imagen de punto de detección (en la suposición de que el sonido es propagado esféricamente por una fuente puntual), y la absorción de energía sonora en cada reflexión en las paredes del recinto.

Se debe tener en cuenta que cada fuente imagen sólo "ilumina" dentro de un cierto elemento de ángulo sólido, es decir, debe considerarse que tiene cierta directividad, determinada por la extensión de la pared a la que corresponde la reflexión (ver Fig. 2.6). A este ángulo sólido se le conoce con el nombre de cono (o pirámide) de visibilidad.

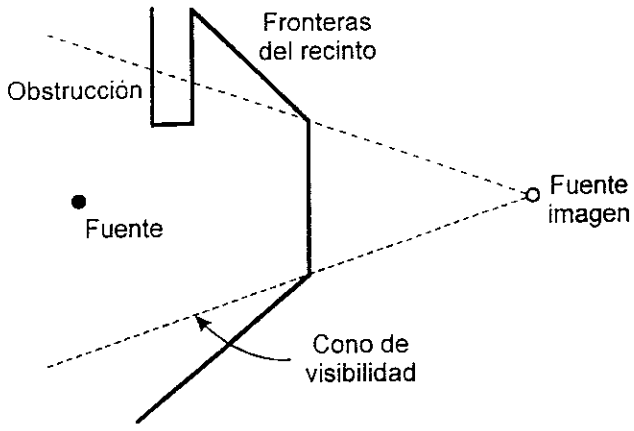

Figura 2.6. Para detectar el campo sonoro generado por la fuente imagen se debe estar dentro del cono de visibilidad y fuera de una obstrucción.

Para recintos rectangulares la implantación del método es relativamente sencilla [55,51], pero se complica cuando el recinto presenta una geometría arbitraria [52,56]. Esto se debe, entre otras cosas, a que se tienen que realizar pruebas de visibilidad y obstrucción para cada punto de detección [57]. La taza de fuentes imagen a validar se incrementa exponencialmente con el orden de reflexión y muchas de esas fuentes no son visibles en la mayoría de los casos, lo cual vuelve al algoritmo muy ineficiente para simulaciones de recintos que incluyen reflexiones de orden muy alto. Debido a lo anterior, se han desarrollado métodos híbridos para pruebas rápidas de visibilidad y obstrucción [58-60,14]. Para reducir el número de fuentes imagen potenciales, sólo las superficies que son al menos parcialmente visibles a la fuente sonora son examinadas. Este mismo principio es aplicado a las fuentes imagen (ver Fig. 2.5).

Un aspecto interesante de la aplicación de este método a recintos rectangulares, donde es regularmente usado, es que no todas las fuentes imagen son visibles ya que el número de fuentes ocultas iguala al número de visibles para fuentes de segundo orden, y la diferencia aumenta para fuentes de orden superior. Sin embargo, puede demostrarse que de un subconjunto de fuentes imagen localizadas exactamente en la misma posición sólo una es visible. Ésta se determina a partir de la posición del detector dentro del recinto. De esta manera las pruebas de visibilidad pueden omitirse para estos casos. Para recintos convexos, dentro de los cuales se encuentran los recintos rectangulares, las pruebas de obstrucción no son necesarias.

Otro problema es que el método de imágenes no puede ser extendido para incluir otros efectos físicos [54], sin embargo, en combinación con otras técnicas, es posible simular en cierta medida reflexiones difusas [12,61]. El método de imágenes es útil para los siguientes casos: para respuestas impulso muy cortas con alta resolución en el tiempo, para recintos formados por pocas superficies (preferentemente convexos) y para recintos rectangulares.

Método de Trazado de Rayos

Este método consiste en enviar un número finito de rayos desde un origen común localizado en el interior del recinto, lo cual representa una radiación de "partículas de energía" desde una fuente puntual. Al propagarse, los rayos se reflejan sobre las superficies que forman al recinto, las cuales absorben una fracción de su energía en cada reflexión. Por lo tanto, este proceso se detiene cuando la energía de los rayos disminuye significativamente o desaparece. De esta manera se genera un campo de energía que varía con el tiempo, diferente en cada punto en el interior del recinto, el cual puede ser detectado sensando la energía de los rayos que inciden en una o varias regiones durante un periodo de tiempo.

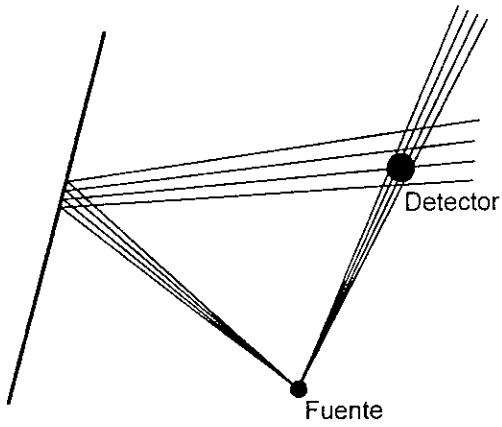

Figura 2.7. Múltiple detección del sonido directo por un detector de tamaño fijo. La trayectoria de reflexión se detecta sólo una vez debido a longitud del rayo. La diferencia angular de los rayos es constante.

En esté método, la reflexión de los rayos sonoros se modela típicamente de acuerdo con la ley de reflexión de Snell. Es posible, sin embargo, simular reflexiones total o parcialmente difusas. La reflexión totalmente difusa se da si la distribución de energía reflejada no depende del ángulo de incidencia del rayo. El suponer este tipo de reflexión se acerca más a las propiedades reflexivas de paredes reales que la reflexión especular, particularmente si interesan varias reflexiones sucesivas del rayo desde diferentes paredes [38]. Las reflexiones parcialmente difusas se llevan a cabo de acuerdo con la ley de cosenos de Lambert [24].

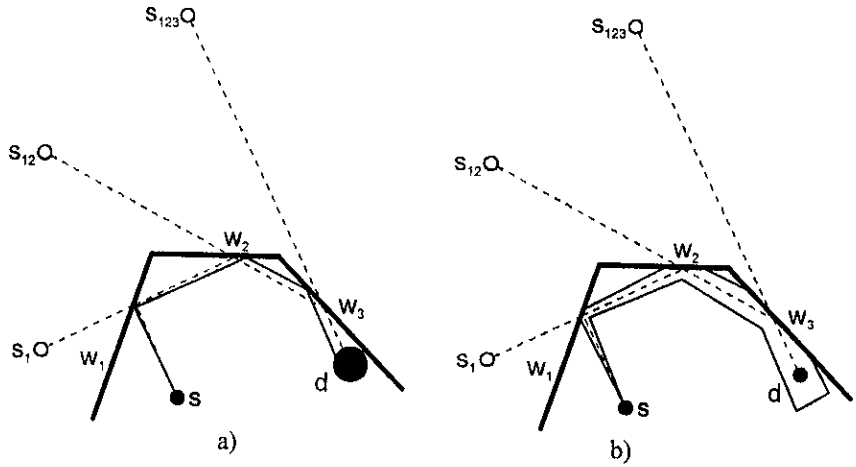

Figura 2.8. Ejemplo de una corrección de trayectoria. Un rayo, a), y un haz, b), son reflejados en las superficies W_1, W_2 y W_3 y después detectados. La fuente imagen de tercer orden S_{123} se obtiene construyendo fuentes imagen en el correspondiente orden. La línea punteada muestra la trayectoria reconstruida.

Un problema fundamental en el método de trazado de rayos radica en el tamaño del detector. Idealmente, éste debería ser puntual pero la probabilidad de que un rayo, cuyo grosor también es infinitamente pequeño, incida en un punto determinado es extremadamente baja. Para arreglar este problema se le asignan dimensiones al detector y/o al rayo que se va a trazar. Han sido propuestas diferentes soluciones para manejar este problema, lo más común es aplicar un volumen finito al detector, por ejemplo, una esfera, un cubo o un disco [53,62-64]. De todas ellas sólo la esfera exhibe un patrón de sensibilidad omnidireccional, las otras formas producen un patrón de respuesta anisótropico. Cuando se fija el tamaño del detector se presenta un efecto de desplazamiento, esto es como si un detector puntual "cazara" rayos desplazándose sobre la superficie del elemento de detección. Por lo tanto, la respuesta impulso resultante será una superposición de varias respuestas impulso en diferentes puntos [22]. Además, mientras mayor sea la distancia que recorre el rayo la probabilidad de ser detectado decrece (ver Fig. 2.7). Una manera de aumentar la probabilidad de detección es variando el tamaño del detector de acuerdo a la longitud del rayo [65] o haciendo puntual al detector y reduciendo la colimación del rayo (ver Fig. 2.8), es decir, transformar el rayo en un haz, el cual puede ser modelado como un

cono o una pirámide [66,67]. En ambas soluciones se debe aplicar una corrección de trayectorias [65].

La ventaja que tiene el método de trazados de rayos, radica en su adaptabilidad para incorporar efectos físicos tales como dispersión [68]. La dispersión en superficies se puede deber a efectos de difracción o difusión. Además, se presta para realizar compromisos entre exactitud, longitud de la respuesta impulso, resolución espacial y tiempo de cálculo [54].

3

IMPLANTACIÓN DEL MÉTODO DE TRAZADO DE RAYOS USANDO PROGRAMACIÓN ORIENTADA A OBJETOS

La parte central del sistema RA, desarrollado en esta tesis para crear espacios acústicos virtuales, es el programa que genera información sobre el campo sonoro. Este programa considera los parámetros que definen el ambiente acústico a modelar (los datos geométricos y acústicos de las superficies reflectoras de sonido y las posiciones, orientaciones y características direccionales de la fuente sonora y el receptor). Debido a su importancia, se debe poner especial atención al elegir el método de simulación y al decidir como se va a implantar.

En cuanto al método de simulación, se decidió emplear un método basado en acústica geométrica; pues este tipo de modelo, aún cuando constituye una aproximación (inferior en todo caso a un modelo ondulatorio), ofrece un mejor rendimiento computacional, al mismo tiempo que permite obtener un descripción subjetivamente relevante del campo sonoro. Por otro lado, considerando la facilidad para seleccionar entre exactitud, longitud de la respuesta impulso, resolución espacial y tiempo de cálculo, se optó específicamente por el método de trazado de rayos.

El diseño e implantación de dicho programa se hizo a través de técnicas OOP empleando el lenguaje de programación C++*. Las características de este lenguaje, tales como verificación de tipos, abstracción de datos, encapsulamiento, herencia (polimorfismo y ligamiento dinámico), y sobrecarga de operadores, permiten reducir

* En este capítulo se asume que el lector está familiarizado con conceptos básicos sobre programación orientada a objetos, en caso contrario se sugiere apoyarse en los siguientes textos [69-71]

substancialmente el esfuerzo en el desarrollo y mantenimiento de grandes paquetes de programas; por ello, y entre otras cosas, la decisión de usar dicho lenguaje.

3.1 Definición de Objetos y Clases

La clave de un buen diseño es modelar directamente algún aspecto de la realidad, es decir, capturar los conceptos de una aplicación como clases, representar las interrelaciones de éstas en formas bien definidas como la herencia y hacer esto una y otra vez en diferentes niveles de abstracción.

Para tener una idea más clara de lo que se quiere programar, se debe definir primero el problema por resolver. En cuanto al problema que nos concierne, podemos usar el siguiente planteamiento:

Un recinto está formado por paredes, techo y piso, además, su interior puede contener objetos, y cada uno de ellos podría estar compuesto de diferentes materiales con diferentes propiedades. Si se quiere estimar la respuesta impulso energética de ese recinto, se necesita contar con representaciones computacionales de los siguientes elementos: una *fuente sonora (impulsiva)*, *un sensor (micrófono)*, y *un dispositivo* de registro (o grabación); cada uno con características propias de emisión, recepción, y método de registro. El proceso para estimar la respuesta impulso energética usando el método de trazado de rayos se puede describir en términos de la interacción entre todos estos elementos: El frente de onda generado por la fuente será modificado por las superficies del recinto, el cual, al alcanzar el micrófono, será codificado en cierta forma por éste y enviado hacia el dispositivo de grabación.

Las consideraciones anteriores sugieren que el proceso de estimación de la respuesta impulso se puede describir en términos de la realización de una medición simulada. Para ello es necesario definir objetos que modelen los siguientes procesos:

- Radiación
- Sensado
- Construcción de Espacios

Para diseñar la estructura general del programa se siguen los siguientes pasos [69]:

1) Identificar las clases (objetos) y sus interrelaciones más fundamentales.
2) Depurar las clases especificando los conjuntos de operaciones que es posible realizar con ellas.
3) Depurar las clases especificando las relaciones entre ellas.
4) Especificar las interfaces de programación de las clases.

En la figura 3.1 se presenta el esquema del diseño del conjunto de clases resultantes del proceso de diseño. En las siguientes secciones se presenta la implantación de estas clases junto con el razonamiento empleado en su concepción.

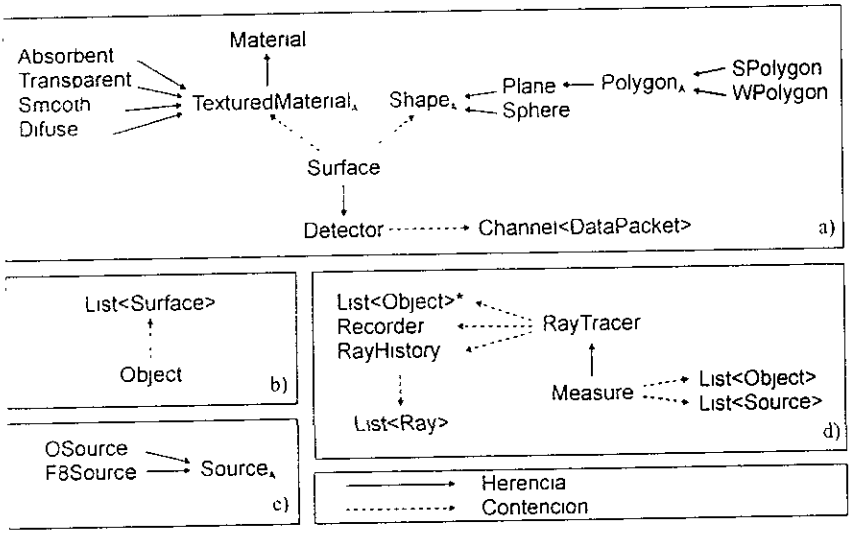

Figura 3.1. Jerarquías de clases para implantar el método de trazado de rayos. El subíndice "A" es empleado para denotar a las clases abstractas. Los tres puntos indica que se pueden derivar nuevas clases [72].

Para la implantación de las clases se desarrollaron, adicionalmente, tres librerías de clases auxiliares: Una para realizar operaciones con vectores, 3DVector, y otra para puntos, Point. La última clase, List<T>, es una clase tipo recipiente que define la funcionalidad de una lista simple, programada en C++ mediante patrones (*templates*, en inglés),

la cual puede ser manipulada por medio de un conjunto de iteradores (ver apéndice 1)

Cabe hacer notar que en lo que sigue sólo se pretende describir el diseño y las cualidades del programa empleando técnicas OOP, y no la estructura más detallada de los algoritmos de cálculo, manejo de memoria, etc.

3.2 Radiación

En el proceso de radiación se distinguen dos entidades: El objeto que radia, la fuente (impulsiva), y el objeto radiado, el rayo. Un rayo se puede conceptuar como una muestra espacial (en una dirección particular) de la energía del frente de onda que resulta al generar un impulso con una fuente puntual. En este sentido, el modelado de una fuente de sonido como una fuente de rayos puede interpretarse como una discretización direccional de su patrón de radiación en el campo lejano.

Rayo Sonoro

Un rayo sonoro se representa por medio de una línea recta, que tiene un origen y una dirección, sobre la cual se transporta un paquete de energía. Considerando lo anterior, la clase Ray queda implantada como sigue:

```
class Ray {
 private:
   Point loc;                                          // location
   Vector3D dir;                                       // direction
   float amp;                                          // amplitude of energy
 public:
   Ray(const Point& l, const Vector3D& d, float a = 1)  // constructor
     : loc(l), amp(a) { dir = Normalise(d); }
   // ...
   Point Extrap(const double& dst) const { return loc+dir*dst; }
};
```

La representación del concepto de rayo queda encapsulada en la función Ray::Extrap(), la cual proporciona el punto que el rayo alcanza después de recorrer cierta distancia.

Puesto que en el proceso de reflexión el rayo sufre cambios en su dirección de propagación y en su amplitud de energía, es necesario definir una clase que pueda almacenar la historia del rayo a lo largo de este proceso, y además proporcione una interfaz con el estado actual del rayo.

```
class RayHistory {
 private:
  Ray c_ray;                                          // current ray
  List<Ray> rays;                    // stores history of changes suffered by ray
  // ...
 public:
  // ..
  void NewRay(const Vector3D&, const float&);          // define new state of ray
  const Ray& operator()() const { return c_ray; }      // get access to current ray
};
```

La clase RayHistory está diseñada para almacenar los cambios sufridos por el rayo utilizando una clase recipiente, en este caso List<Ray>. En esta clase se pueden incluir funciones que manejen la estadística del rayo, por ejemplo, saber qué distancia a recorrido, sobre qué superficies se ha reflejado, qué tanta energía a perdido en el proceso de propagación, y otras más (ver apéndice 2). La forma de diseño de este objeto hace que éste pueda ser tratado como un tipo de rayo con memoria, ya que se tiene acceso a través de operator()() directamente a todas las propiedades de un rayo.

Fuente de Rayos

Una fuente genera rayos que parten de un punto común, siguiendo un cierto patrón direccional de radiación; ese patrón obedece la forma de radiación de la fuente puntual a modelar. Dado que pueden existir varios tipos de fuentes, se recurre a una clase base abstracta* para construir la interfaz entre varias formas de radiación:

```
class Source {
 private:
  Point location;
  float level;
 public:
  Source(const Point& p, float i = 1) : location(p), level(i) { }      // constructor
```

* Una clase que tiene al menos una función virtual pura se le nombra abstracta; y no es posible crear objetos de esa clase.

```
    // ...
    const Point& Location() const { return location; }        // read-only access by reference
    const float& Level() const { return level; }              // read-only access by reference
    virtual Ray GenRay() const = 0;                           // pure virtual function
};
```

Desde esta clase se puede derivar un sinnúmero de tipos de fuentes (ver Fig. 3.1c) definiendo, a través de la función virtual pura Source::GenRay(), la manera en que se generan los rayos, siendo responsabilidad del usuario de las clases proporcionar los datos suficientes para construirlas. Por ejemplo, para definir una fuente omnidireccional se procede del siguiente modo:

```
class OSource : public Source {
 public:
  OSource(const Point& p, float i = 1) : Source(p, i) { randomize(); }        // constructor
  // ...
  Ray GenRay() const;
};
```

Para este tipo de fuente no se necesita conocer su orientación, sólo es necesario saber su posición y el nivel de energía con el que radiará. Con estos datos la definición de la función virtual queda como:

```
Ray OSource::GenRay() const
{
 // random definition for azimuth and elevation
 Vector3D dir(random(36001)/100, random(18001)/100);
 return Ray(Location(), dir, Level());
}
```

Esta función se puede programar de forma tan complicada como complicado sea el patrón de radiación a modelar. Lo que hace especial a una función virtual, es que cuando se accede a ella a través de un apuntador de una clase base que apunta a un objeto de una clase derivada, C++ determina, al tiempo de ejecución, cual función llamar basándose en el tipo del objeto apuntado. Cabe mencionar que una función virtual pura es aquella que se declara en una clase base y no tiene definición relativa a la base.

3.3 Sensado y Registro

Para cualquier tarea de medición se debe contar siempre con un dispositivo capaz de reaccionar, generando algún tipo de información, en presencia del fenómeno físico que caracteriza al evento de interés, y otro que reciba y almacene la información generada por el primero. Por lo anterior, si se quiere contar con las funciones de sensado y registro, es necesario modelar objetos que tengan las cualidades mencionadas.

Detector

En nuestro caso, un detector es una entidad que tiene la propiedad de extraer cierta información de un rayo, empaquetarla y enviarla por un canal de transmisión, el cual a su vez puede canalizar esta información para ser procesada posteriormente (por ejemplo, para que se almacene en un archivo). El canal de transmisión, puede pensarse como una banda de transporte bidireccional, donde un paquete colocado en un extremo puede ser llevado al otro extremo y viceversa. Esto puede ser programado de la siguiente forma:

```
template<class T> class Channel {
 private:
  T msg;                                                    // message
 public:
  Channel() : msg(T()) { }                                  // constructor
  T Rx() { T tmp = msg; msg = T(); return tmp; }            // receive a message
  void Tx(const T& tx_msg) { msg = tx_msg; }                // transmit a message
  int IsThereMsg() const { if(msg == T()) return 0; return 1; }
};
```

Aquí se hace uso de un patrón (*template*) puesto que el paquete de datos a ser transportado, el mensaje, puede contener cualquier cosa. El paquete de datos puede ser alguna estructura que tenga como campos, por ejemplo, la amplitud de la energía del rayo y la distancia recorrida por éste.

Para implementar la clase Detector es necesario tener un apuntador a un objeto de tipo Channel para habilitar un canal de transmisión, y una función para extraer la información del rayo, empaquetarla y enviarla por dicho canal. Se usa un apuntador a Channel porque Detector tiene que ser conectado al canal proporcionado por el usuario de la clase para recibir la información.

```
class Detector {
 private:
  Channel<DataPacket>* comm_line;
  int interrupter;
  int connected;
 public:
  Detector(const int& i = 0)
   : interrupter(i), connected(0), comm_line(NULL) { }                    // constructor
  // ...
  void On() { interrupter = 1; }
  void Off() { interrupter = 0; }
  void Connect(Channel<DataPacket>* cl) { comm_line = cl; connected = 1; }
  int Detect(const RayHistory&);
};
```

La tarea de Detector::Connect() puede ser explicada fácilmente haciendo una analogía: Si se visualiza el canal de transmisión como un cable, esta función sería el punto donde el cable debe ser conectado. La función Detector::Detect() se encarga de extraer información de un rayo, colocarla en los campos de una estructura tipo DataPacket y mandar el paquete de datos a través del canal de transmisión. Su implantación queda de la siguiente forma:

```
int Detector::Detect(const RayHistory& ray)
{
 if(interrupter) {                                                  // check if it is on
  float d = ray.LengthPath();                                  // get information from ray
  float a = ray().Amp();
  // ...
  comm_line->Tx(DataPacket(d, a, ...));                      // data are packed and sent
  return 1;                                                        // successful operation
 }
 return 0;                                                        // unsuccessful operation
}
```

En la clase Detector se integraron algunas banderas para habilitar la detección y para saber si éste ha sido conectado a una entidad capaz de recibir el paquete de datos, por ejemplo un grabador.

Grabador

La tarea de recepción y almacenamiento de datos es conferida a una clase llamada Recorder. Siguiendo con las analogías, este objeto fue diseñado tomando la idea de una consola multicanal de grabación, con la salvedad de que en éste los canales no pueden ser mezclados;

aunque, si sirviera de algo, bien podría programarse. Además, cada canal tiene asignado un dispositivo de almacenamiento. La declaración de esta entidad tiene el siguiente aspecto:

```
class Recorder {
 private:
  int ready;
  List<Channel<DataPacket>*> channels;
  OutFileList tape;
  // ...
 public:
  Recorder() : ready(0) { }                                    // constructor
  ~Recorder() { UnPlug(); }                                    // destructor
  // ...
  void Prepare();                                  // prepare tapes for recording
  Channel<DataPacket>* Channel();
  void Rec();                                    // search for a message to record
  void Stop();                                    // stop record and save all data
  void UnPlug(int i = -1);                  // unplug either all or a specified channel
};
```

A través de la función Recorder::Channel() se entregan accesos a los canales del grabador, cuyo número está limitado sólo por cantidad de memoria disponible; por cada llamada se dispone de un nuevo canal. Antes de empezar a grabar datos, se debe tener un dispositivo de almacenamiento etiquetado apropiadamente en cada canal, esto se realiza invocando la función Recorder::Prepare(). El modelado de dicho dispositivo de almacenamiento se hace a partir de un objeto de clase OutFileList, el cual, a grandes rasgos, es una lista de archivos de salida (*stream*) con unas cuantas funciones de control (ver apéndice 2).

```
void Recorder::Rec()
{
 if(ready) {                                                  // check if it's ready to record
  for(ListItr<Channel<DataPacket>*> line(channels); line.More(); line++)
   if(line()->IsThereMsg()) {                          // check for messages on each channel
    tape[line.Index()] << line()->Rx() << "\n";                // record information on tape
    break;
   }
 }
}
```

La función Recorder::Rec(), mostrada arriba, se encarga de revisar cada canal disponible en busca de un mensaje, al encontrarlo es guardado

en el archivo que corresponde a dicho canal. Al detener la grabación con la función Recorder::Stop(), los archivos son cerrados.

3.4 Construcción de Espacios

Un espacio está formado por un conjunto objetos, los cuales pueden estar constituidos por una o varias superficies, cada superficie debe tener una cierta forma geométrica y estar compuesta de algún material, el cual posee ciertas propiedades. Siguiendo esta definición aparecen las entidades a ser modelados para construir un espacio. Primeramente, se debe concretar una clase que refleje las propiedades físicas (en nuestro caso, acústicas) de un material, para posteriormente moldearlo en una superficie con alguna forma geométrica. Una vez teniendo una clase capaz de manipular superficies en forma genérica, basta especificar una clase que maneje un conjunto de superficies para modelar objetos más complejos. Por último, es suficiente contar con un arreglo de objetos para definir un espacio.

Material

Lo que aquí se pretende modelar es cuanta energía es absorbida, reflejada o transmitida por un material cuando un rayo sonoro incide sobre él. Estas características están sujetas al tipo de material y a la forma en que éste es irradiado (en función de la dirección del rayo incidente, por ejemplo), lo cual sugiere plantear el diseño de la siguiente forma:

```
class Material {
 private:
  float abs_coef;                                          // absorbtion coeficient
  // ...
 public:
  Material(float a) : abs_coef(a) { }                      // constructor
  // ...
  virtual float Absorption(const Vector3D& i, const Vector3D& n);
};
```

Aquí sólo se presenta la parte de código que permite conocer qué tanta energía es absorbida por el material. Para construir un objeto de clase Material es necesario proporcionar el coeficiente de absorción promedio de algún tipo de material. Si la forma de absorción del material no depende de cómo éste es irradiado, la función Material::Absorption() simplemente regresará el coeficiente de absorción promedio. En caso

contrario, se necesita conocer desde qué dirección se le está radiando y una referencia a la orientación de la superficie (la normal) asociada al material para conocer el ángulo de radiación. Por ejemplo, se podría considerar que si el vector de incidencia y la normal son paralelos, la absorción es alta, y conforme el ángulo entre ellos se incrementa, la absorción disminuye; usando como referencia al coeficiente de absorción promedio.

Si se considera ahora que la superficie del material tiene cierta textura, esto llevaría a pensar en modelar la forma en que una partícula de energía, en nuestro caso un rayo, es reflejado sobre su superficie. Para programar la propiedad de reflexión de la textura de la superficie de un material sólo se necesita conocer la dirección de incidencia del rayo y la normal de la superficie en el punto de contacto. Todo esto sugiere usar la clase Material para derivar una nueva clase material con textura y así integrar esa nueva propiedad:

```
class TexturedMaterial : public Material {
 public:
  TexturedMaterial(float a) : Material(a) { }
  // ...
  virtual Vector3D Reflect(const Vector3D& i, const Vector3D& n) const = 0;
};
```

A partir de esta clase abstracta se pueden derivar varios materiales con diferentes texturas (ver Fig. 3.1a). Por ejemplo, para crear un material con textura lisa, el cual reflejará un rayo en forma especular (es decir, de acuerdo con la ley geométrica de reflexión), se procede de esta forma:

```
Vector3D Smooth::Reflect(const Vector3D& i, const Vector3D& n) const
{
 return n*2*DotProd(n,-i)+i;                                    // specular reflection
}
```

Así mismo, se pueden modelar reflexiones difusas, escogiendo la dirección del rayo reflejado a través de un proceso aleatorio; donde la densidad de probabilidad de la dirección respectiva puede ser determinada por la ley de Lambert, o bien adaptada a las propiedades de difusión espacial de alguna superficie; de manera parecida a lo que se sugirió para modelar una fuente.

Formas Geométricas

Una forma geométrica se modela a partir de una clase base abstracta Shape, la cual proporciona una interfaz a sus clases derivadas que permiten conocer su posición y orientación, y un método para determinar si un rayo dado es capaz de incidir en ella o no. Además, puesto que objetos de clases derivadas van a ser manipulados y eliminados a través de un apuntador a esta clase, se incluye un destructor virtual.

```
class Shape {
 public:
  Shape() { }                                                    // constructor
  virtual ~Shape() { }                                           // destructor
  // ...
  virtual Vector3D Normal(const Point&) const = 0;
  virtual float NearestDistance(const Ray&) const = 0;
};
```

La función virtual pura Shape::NearestDistance() tiene el papel de proporcionar la distancia más corta que debe cubrir un rayo para alcanzar dicha forma geométrica; con esto se puede determinar si el rayo incide en ella o no [73]. La posición y orientación de la superficie en el punto de contacto con el rayo (si lo hay), se puede determinar a partir del vector unitario normal a la superficie en dicho punto. Para mostrar cómo se pueden derivar un sinnúmero de formas geométricas a partir de la clase Shape, se presenta el modelado de una forma plana.

```
class Plane : public Shape {
 protected:
  Vector3D normal;
  float d;
 public:
  Plane() : Shape() { }                                          // constructor
  Plane(const Point&, const Point&, const Point&);               // constructor
  // ...
  Vector3D Normal(const Point&) const { return normal; }
  float NearestDistance(const Ray&) const;
};
```

Un plano puede ser construido especificando la normal y la ordenada al origen, las cuales a su vez pueden calcularse a partir de tres puntos dados sobre la superficie:

```
Plane::Plane(const Point& p1, const Point& p2, const Point& p3) : Shape()
{
 Vector3D u = p2-p1, v = p3-p1;
 normal = Normalise(CrossProd(u, v));
 d = DotProd(normal, p1);
}
```

Puesto que la normal es parte de la definición de este objeto, la función Plane::Normal() queda implantada en forma directa. Para la clase Plane, la sobrecarga de la función para calcular la distancia más cercana queda como:

```
float Plane::NearestDistance(const Ray& ray) const
{
 float denom = DotProd(normal, ray.Dir());
 // check if plane and ray are parallel
 if(denom == 0) return -1;
 float num = d - DotProd(normal, ray.Loc());
 // check if the location of the ray is either on or near to the plane
 if(fabs(num) < TOLERANCE) return -1;
 // check if the plane is directed towards the ray
 if((denom<0 && num<0) || (denom>0 && num>0)) return num/denom;
 return -1;
}
```

Aquí se hacen varias pruebas para verificar si el plano está en la trayectoria del rayo. La declaración de esta función será diferente para cada tipo de forma geométrica que se defina; de ahí la razón para definirla inicialmente como una función virtual.

Puesto que las superficies planas que forman un recinto no se extienden infinitamente en el espacio, sino que más bien están acotadas, se crea una nueva clase llamada Polygon, la cual hereda todas las propiedades de un plano. Para este caso, la sobrecarga de la función virtual queda como:

```
float Polygon::NearestDistance(const Ray& ray) const
{
 float d = Plane::NearestDistance(ray);        // use the plane described by the polygon
 if(d > 0) {                                                  // if it is a valid path
  Point p = ray.Extrap(d);                                    // get the hit point
  if(IsInside(p)) return d;                   // check if the point is inside of the polygon
  if(IsOnbound(p)) return d;                  // check if the point is on bound of the polygon
 }
```

```
    return -1;                                    // the ray doesn't hit the polygon
}
```

En el diseño de la clase Polygon se consideraron dos casos de manejo de la memoria para hacer eficientes las operaciones con polígonos: Cuando un polígono existe en forma independiente, sPolygon, y cuando se encuentra unido a otros para definir un objeto geométrico complejo, wPolygon (ver Fig. 3.1a). A grandes rasgos, el manejo de memoria para el primer caso consiste en usar una lista de vértices, para el segundo, una lista de vértices y una de índices. Para sPolygon el acceso a los vértices se lleva a cabo de forma directa, es decir, para tomar el vértice número tres se toma el elemento tres de la lista. Por otro lado, a cada objeto de clase wPolygon se le proporciona una referencia a una lista común de vértices y una lista con los índices de los vértices de los que está constituido; esto con el fin de evitar información redundante.

Para manejar ambos casos se diseño la clase Polygon como abstracta (ver Fig. 3.1a), añadiéndole dos funciones virtuales puras para el manejo de vértices:

```
class Polygon : public Plane {
 // ...
 public:
  // ...
  Vector3D Normal(const Point& p) const { return Plane::Normal(p); }
  float NearestDistance(const Ray&) const;
  virtual const Point& Vertex(unsigned) const = 0;          // read-only access
  virtual Point& Vertex(unsigned) = 0;                      // read-and-write access
  int IsInside(const Point&) const;
  int IsOnbound(const Point&) const;
// ...
};
```

Con esto, cualquier función dentro de la clase Polygon que requiera para su implantación acceso a los vértices no necesita saber la forma en como estos son almacenados. Por tanto, es responsabilidad de sus clases derivadas definir las funciones Polygon::Vertex() correctamente para así poder usar las funciones para operar polígonos. La programación de las clases sPolygon y wPolygon, así como para Polygon, conlleva otras complicaciones, tales como verificación de vértices repetidos, liberación de memoria cuando se destruye o se copia un polígono, la adhesión de un polígono aislado en un conjunto de polígonos unidos,

etc. La solución a estas complicaciones no se presenta en este documento, puesto que son detalles de implantación que no conviene exponer aquí, pues lo que se desea es mostrar con claridad el diseño aplicando técnicas OOP.

Superficie Reflectora

Como se muestra en la figura 3.1a, una superficie reflectora se conceptúa como una entidad que tiene alguna forma geométrica hecha de un material con cierta textura, y a su vez puede ser usada como detector. Para hacer que una superficie pueda ser construida con cualquier tipo de material y moldeada en cualquier forma, se hace uso de apuntadores declarados como apuntadores a las clases base Shape y TexturedMaterial. Cuando estos se hacen que apunten a un objeto de alguna clase derivada, se accede a través de funciones virtuales a los métodos relativos de cada objeto apuntado; con lo cual se produce lo que se le llama un polimorfismo en tiempo de ejecución (*runtime polymorphism*). Ahora bien, como la clase Detector por si misma es solamente una propiedad, resulta evidente la necesidad de asignarle un área de sensado. La forma más simple de hacer eso, es derivando desde la clase Detector una nueva clase Surface, con lo cual ésta última hereda la propiedad de detección, proporcionando al mismo tiempo un área de sensado. Por lo tanto, la declaración de dicha clase presenta el siguiente aspecto:

```
class Surface : public Detector {
 private:
   Shape* shape;
   TexturedMaterial* material;
   float d;                                          // auxiliar storing variable
   // ...
 public:
   Surface(Shape* s, TexturedMaterial* m)                          // constructor
      Detector(), shape(s), material(m), d(0) { }
   ~Suface { delete (void*)shape; delete (void*)material; }        // destructor
   // ...
   const float& StoredDistance() const { return d; }               // read-only accesss
   void Reflect(RayHistory&) const;
   float NearestDistance(const Ray& ray) { return d = shape->NearestDistance(ray); }
};
```

La función Surface::NearestDistance() tan sólo es un medio de acceso a la función homónima de la forma geométrica asignada a la superficie, el cual no es el caso para la siguiente función:

```
void Suface::Reflect(RayHistory& ray) const
{
 Point p = ray.HitPoint();
 Vector3D dir = material->Reflect(ray().Dir(),shape->Normal(p));
 float absorption = material->Absorption(ray().Dir(),shape->Normal(p));
 ray.NewRay(dir, absorption);
}
```

Ésta es una versión más completa para modelar la reflexión de un rayo sobre la superficie; ya que además de calcular la dirección de reflexión, obtiene el coeficiente de absorción del material para poder modificar la amplitud del paquete de energía transportado por el rayo. Con el fin de registrar los cambios sufridos por el rayo, se emplea como argumento un objeto de clase RayHistory.

Objeto Reflector

Un objeto reflector, por definición, es tan sólo un conjunto de superficies reflectoras, por lo tanto, su implementación debe contar con una lista de tipos de clase Surface (ver Fig. 3.1b) y contar con una serie de funciones para hacer que el conjunto de propiedades de cada superficie se integren como parte de las propiedades del objeto reflector. Con base en esto, la nueva clase Object queda implantada de la siguiente forma:

```
class Object {
 private:
  List<Surface*> surfaces;
  unsigned nofd;                    // number of surfaces that are enabled to detect
  // ...
 public:
  Object() : nofd(0) { }                                          // constructor
  // ...
  void AddSurface(Surface*);
  Surface* NearestHitSurface(const Ray&) const;
  void Connect(Channel<DataPacket>*);
};
```

Para ir construyendo un objeto a partir de superficies definidas, se cuenta con una función de nombre Object::AddSurface(), la cual, además de ir colocando cada superficie en una lista, lleva un registro de cuantas están habilitadas para detectar; esto con el fin de conocer cuantas lineas de transmisión se requieren para ser conectadas a través de la función Object::Connect(). La siguiente función permite saber si un

rayo en propagación llega a golpear el objeto, y de ser así sobre qué superficie incidió:

```
Surface* Object::NearestHitSurface(const Ray& ray) const
{
 List<float> distances;
 for(ListItr<Surface*> surface(surfaces); surface.More(); surface++) {
  float dst = surface()->NearestDistance(ray);                    // compute distances
  distances.Append(dst);
 }
 SearchListItr<float> distance(distances);
 if(distance.Min(0))                              // find the shortest distant greater than 0
  return surfaces[distance.Index()];    // return the surface corresponding to that distance
 return NULL;
}
```

En ésta función se calcula la distancia desde la posición del rayo, sobre su línea de propagación, hasta cada superficie que forma parte del objeto. Para obtener la superficie donde el rayo choca, basta sólo con tomar la superficie asociada a la distancia más corta mayor que cero (por convención, las superficies no visibles se indican con una distancia negativa).

3.5 Implantación del Método de Trazado de Rayos

Teniendo definido cada elemento que interviene en la generación y registro de un campo de rayos sonoros se debe diseñar una clase donde estos trabajen en conjunto, describiendo así un proceso en el que se puedan realizar mediciones simuladas dentro del campo sonoro que se modela. La clase que implanta este proceso debe hacer que un rayo dado se propague y se atenúe al reflejarse sobre las superficies que conforman al espacio dentro del cual se propaga. Además, cuando el rayo haya sufrido un número de reflexiones o su energía inicial haya disminuido significativamente, el proceso debe ser detenido. En el caso de incidir el rayo en una superficie habilitada como detector, la clase debe hacer la indicación de que el proceso de registro se lleve a cabo.

Para esto, la clase que realiza el proceso de trazado de rayos debe contener una referencia a una lista de objetos reflectores, un tipo para registrar la historia del rayo y un tipo que proporcione canales de transmisión para conectar las superficies que funcionen como detector

y además grabe la información enviada a través de dicho canal (ver Fig. 2d). La entidad resultante de este planteamiento tiene el siguiente aspecto:

```
class RayTracer {
 private:
  List<Object*>* objects;                          // a reference to a list of objects
  RayHistory ray;
  float thr;                                       // attenuation threshold of ray
  Surface* GetHitSurface(const Ray&) const;
 protected:
  Recorder recorder;
 public:
  RayTracer(List<Object*>* optr) : thr(1e-06), objects(optr) { }     // constructors
  ~RayTracer() { objects = NULL; }                                    // destructor
  // ...
  void TraceRays(const Ray& o_ray = Ray());                           // till decay
  void TraceRays(const int& n, const Ray& o_ray = Ray());             // till n reflections
  void RecOn() { recorder.Prepare(); }
  void RecOff() { recorder.Stop(); }
};
```

En esta clase se oculta una función llamada RayTracer::GetHitSurface() que es capaz de obtener una referencia a la superficie en donde el rayo hizo contacto a través de toda la lista de objetos. Cuenta además con dos funciones que realizan el trazado de rayos considerando los casos mencionados anteriormente:

```
void RayTracer::TraceRays(const Ray& o_ray) {
 ray = o_ray;                                      // initialize ray's history
 while(!ray.Decay(thr)) {                          // energy decay loop
  Surface* surface = GetHitSurface(ray());
  ray.TravelOver(surface->StoredDistance());       // throw ray towards surface
  surface->Detect(ray);
  recorder.Rec();
  surface->Reflect(ray);
 }
}
```

Aquí el rayo se propaga hasta que su energía inicial disminuye alcanzando un cierto umbral. Ahora bien, si se quiere que el rayo sufra un cierto número de reflexiones, basta tan sólo con cambiar el ciclo monitor de energía por un ciclo de conteo de reflexiones. Si se quiere registrar los datos generados por las superficies habilitadas como detectores, antes de emplear alguna de las versiones de

RayTracer.TraceRays() se debe usar la función RayTracer::RecOn(). Para guardar los datos registrados se aplica RayTracer::RecOff(); un ejemplo del del uso de estas funciones se dará más adelante.

RayTracer se usa como cimiento para la implantación de la clase que simula diferentes mediciones. La clase Measure, como lo sugiere la figura 3.1d, se debe construir a partir de una lista de fuentes y una lista de objetos:

```
class Measure . private RayTracer {
 protected:
  List<Object*> objs;
  List<Source*> source;
 public:
  Measure() : RayTracer() { }                                    // constructors
  // .
  void impulseResponse(unsigned n, int);
};
```

Por ejemplo, para realizar la medición de la respuesta impulso de un modelo de espacio dado, lo único que resta por hacer es lo siguiente:

```
void Measure::ImpulseResponse(unsigned n) {
 RecOn();                                                        // turn on the recorder
 for(ListItr<Source*> src(sources); src.More(); src++)           // iterate over all sources
  for(int i=0; i<n; i++) {                                       // loop to generate n rays per source
   Ray o_ray = src()->GenRay();                                  // generate ray
   TraceRays(o_ray),
 }
 RecOff();                                  // turn off the recorder and save received data
}
```

Aquí se vuelve evidente la ventaja de usar la programación orientada a objetos, ya que para programar un proceso tan complicado como lo es el estimar la respuesta impulso de un recinto, las líneas de código utilizadas se reducen significativamente, además de que la estructura del código guarda una relación más directa con la interpretación conceptual del proceso que realiza.

4

APLICACIONES A LA SIMULACIÓN Y ANÁLISIS DE RECINTOS

El programa de simulación de recintos desarrollado en esta tesis, fue diseñado de tal manera que pudiera facilitar cualquier información relacionada con la historia de un rayo durante el proceso de propagación. La información generada por el simulador en conjunto con los datos que definen al modelo, es suficiente, como se demostrará a continuación, para hacer un análisis de la acústica de espacios con arquitecturas muy diversas (dentro de la aproximación de la acústica geométrica).

4.1 Modelado del Espacio Acústico

Un espacio acústico está definido por su geometría, las características acústicas de los materiales presentes en él, el medio, la posición de las fuentes y la manera en como éstas radian, y las zonas de sensado o detección. Para que el simulador contemple todos estos datos en su conjunto como características de un campo sonoro, se debe presentar estos en forma codificada.

Para el sistema RA se creó un tipo de archivo de extensión RAF cuyo formato permite presentar cada objeto geométrico que constituye al espacio, referencias sobre sus propiedades físicas, condiciones del medio, fuentes y detectores con sus respectivas características de radiación y sensado. Así mismo, el sistema RA está provisto de un programa que extrae la información sobre la arquitectura del espacio directamente de archivos con formato DXF, la cual es traducida al nuevo formato. Para modelar el espacio acústico de manera sencilla se desarrolló una serie de herramientas con interfaz gráfica bajo el

ambiente de MATLAB. En la figura 4.1 se muestra un recinto modelado por medio de esas herramientas.

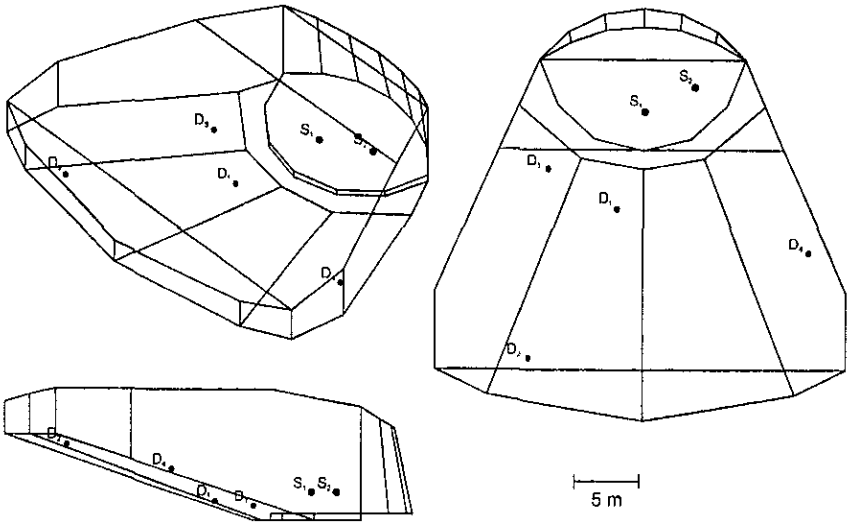

Figura 4.1. Modelo del foro 1 del auditorio Kármán en Aachen Alemania [63]. El foro cuenta con 775 lugares, un volumen de 3750 m^3 y una superficie de 1810 m^2. El foro se modelo con 30 superficies. Las fuentes y los detectores están etiquetados con S y D respectivamente.

4.2 Datos Generados en una Simulación

En una simulación, cuando un rayo incide en una zona de detección, se recolecta información significativa sobre la historia de la trayectoria de un rayo (de la fuente al detector), la cual es almacenada en un archivo con extensión DDF (*Detected Data Format*) que contiene una tabla con los siguientes campos:

- Distancia recorrida
- Amplitud de la energía transportada
- Punto de incidencia
- Dirección de incidencia
- Lista de superficies visitadas
- Puntos de que definen la trayectoria

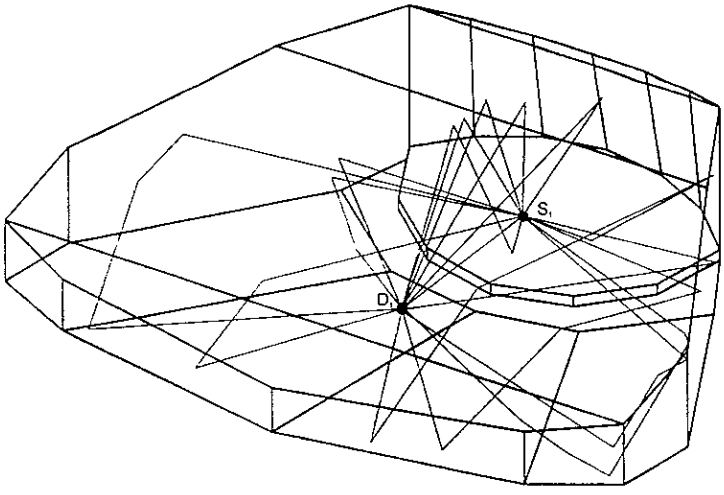

Figura 4.2 Sonido directo y trayectorias de reflexión de primer, segundo y tercer orden calculadas empleando del método de trazado de rayos. La fuente y el detector están etiquetados con S_1 y D_1 respectivamente.

Con esta serie de datos asociada a cada trayectoria es posible realizar estimaciones sobre la acústica del modelo y sintetizar la manera en como éste afecta el sonido. De hecho, una de las aplicaciones dadas al simulador se encuentra en el análisis de características acústicas de recintos. Por ejemplo, una de las pruebas más comunes empleadas en el diseño de una sala de conciertos consta en verificar, a través de diagramas de reflexión, como el sonido se radia y propaga en su interior, verificando así como las paredes intervienen en ese proceso (ver Fig. 4.2).

4.3 Análisis de Algunas Características Acústicas

Las pruebas con diagramas de reflexión proporcionan información muy especifica de la manera en como las superficies que componen a un recinto redireccionan el sonido, sin embargo, estos no muestran como se distribuye la energía a través de todo el recinto, lo cual sería útil para conocer si existen zonas con ecos, sombras o con mejor acústica.

Para obtener información sobre la distribución temporal de la energía sobre el área de la audiencia se habilitó ésta como detector, posterior-

mente se hizo que la fuente radiara 10000 rayos en todas direcciones, lo cual simula la radiación de un impulso por una fuente omnidireccional. Al graficar cada punto de incidencia con su respectivo nivel de energía para diferentes periodos de tiempo se obtiene una serie imágenes como las mostradas en la figura 4.3.

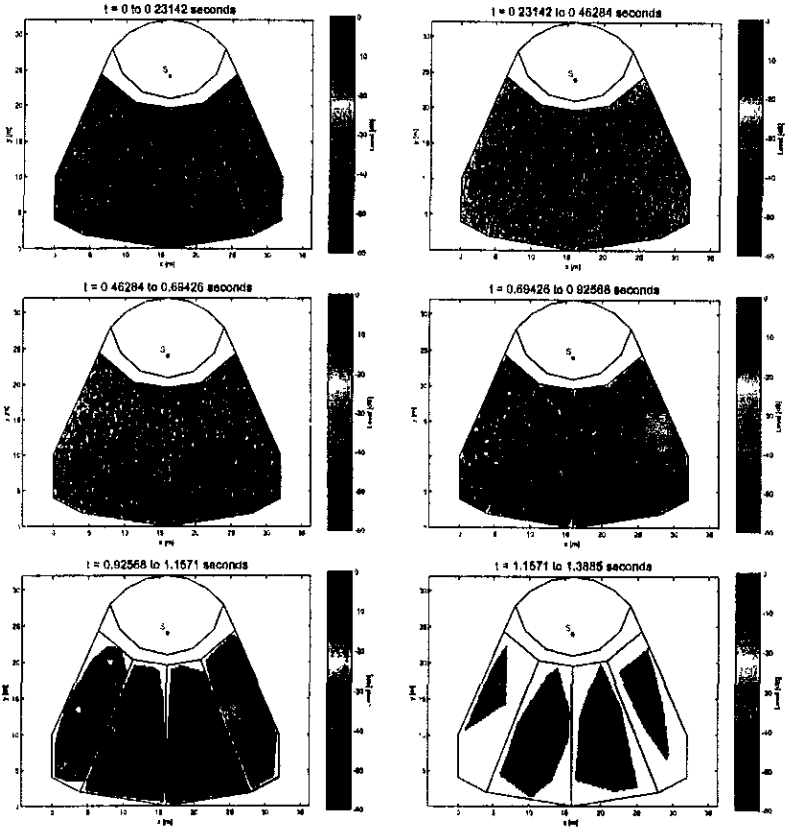

Figura 4.3. Representación gráfica de la distribución del nivel de energía sonora sobre el área del público en periodos de tiempo de 0.23 segundos, considerando una velocidad de propagación de 343.4 m/s (20°C).

En ésta se puede observar que en los primeros segundos después de haber radiado un impulso a través de la fuente S_1 que la distribución de la energía es uniforme en toda el área de sensado. Conforme pasa el tiempo, el nivel de energía disminuye casi con el mismo ritmo en todo el recinto. Al alcanzar los 0.9 segundos se empieza a distinguir zonas

más reverberantes que otras, así mismo, aparecen zonas en blanco donde no se tiene información para ese periodo de tiempo, pero se puede asumir que ahí el sonido ha desaparecido. En el periodo de 1.15 a 1.38 segundos, mientras la energía ya se extinguió en la mayor parte del recinto, aparece una zona en la parte posterior donde la energía persiste. Esto implica que en esa zona se generan ondas estacionarias por lo que el sonido se vuelve más reverberante.

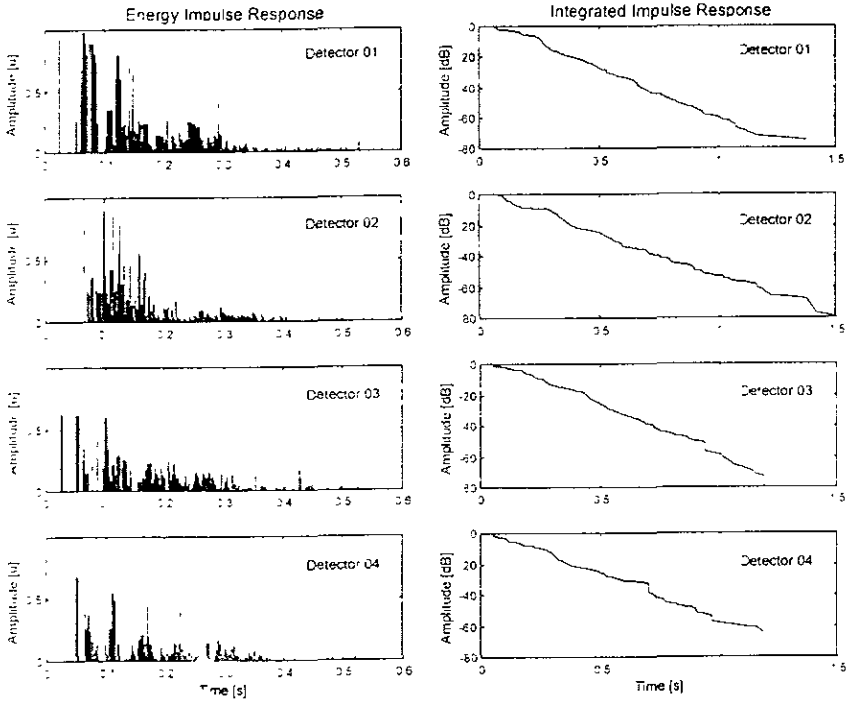

Figura 4.4. Primeros 0.6 segundos de la respuesta impulso energetica para detectores distribuidos en diferentes puntos dentro del recinto (columna izquierda), con sus correspondientes curvas de decaimiento (columna derecha).

Para hacer un análisis más puntual de las variaciones del campo sonoro se distribuyeron cuatro detectores esféricos de 20 cm de diámetro en el interior del recinto (ver Fig. 4.1). La posición de estos se eligieron con base en la información de la figura 4.3 para asegurar que los datos recopilados en la simulación sean representativos del campo sonoro total; por ejemplo, el detector D_2 se colocó dentro de la

zona donde se generan ondas estacionarias y D_1 se puso en las primeras filas frente al escenario.

En la figura 4.4 se presentan los resultados de la simulación de la respuesta impulso energética para los cuatro detectores, los cuales se obtuvieron a radiar 10000 rayos a través de la fuente S_1. En éstas se puede apreciar el retardo debido a la distancia a la que se encuentra cada detector de la fuente, y las variaciones de las distribuciones de las espigas. Sin embargo, para hacer un análisis más formal de cada respuesta éstas deben ser integradas aplicando la siguiente ecuación [74]:

$$y(t) = \int^{\infty} h^2(\tau)\, d\tau \qquad (4.1)$$

Con las curvas de decaimiento resultantes (ver Fig. 4.4) se puede calcular el tiempo de reverberación de cada respuesta. El tiempo de reverberación (RT_{60}) es el intervalo de tiempo en el cual el nivel de presión sonora baja 60 dB. El mayor RT_{60} corresponde al detector D_2 (1.169 s), lo cual se esperaba debido a su posición. Por otro lado, el menor RT_{60} es para el detector D_1 (1.002 s), lo cual no era evidente al observar las respuestas impulso energéticas. Como D_1 se localiza en una zona abierta, las contribuciones de reflexiones de orden superior son muy pequeñas; de ahí la razón de éste resultado.

Un examen más completo sobre la acústica del recinto debe incluir un análisis por frecuencia de sus atributos acústicos. Debido a que en la literatura sólo se puede conseguir valores de coeficientes de absorción de materiales por bandas de octava, el análisis está restringido para esas seis bandas. En la tabla 4.1 se presenta los resultados de este tipo de análisis.

Antes de comentar éstos resultados es preciso introducir algunos conceptos. El tiempo de decaimiento temprano (EDT) es el tiempo que se requeriría para que el nivel de presión sonora inicial decrezca 60 dB a una tasa de decrecimiento observada en los primeros 10 dB de la curva de decaimiento. Este parámetro es empleado en paralelo con RT_{60}, y al igual que RT_{60} se calcula a partir de la respuesta impulso integrada empleando la relación $EDT = 6(t_{15} - t_5)$, en donde $y(t_{15}) = -15\,\text{dB}$ y $y(t_5) = -5\,\text{dB}$.

	125 Hz	250 Hz	500 Hz	1000 Hz	2000 Hz	4000 Hz
RT_{60} [s]	2.157	1.679	0.827	0.711	0.584	0.518
EDT [s]	2.078	1.723	1.005	0.709	0.452	0.394
C_{80} [dB]	-2.614	-3.124	0.697	4.097	4.031	6.519
t_s [s]	0.1666	0.1463	0.0858	0.0766	0.0733	0.0711

Tabla 4.1. Atributos acústicos del modelo de auditorio. Todos los valores fueron promediados de cuatro respuestas impulso medidas por cuatro detectores excitados por dos fuentes omnidireccionales (ver Fig. 4.1).

La claridad C_{80} es la relación entre la energía temprana, la cual se considera estar dentro de los primeros 80 ms, y la tardía; en otras palabras, es la relación entre la contribución energética de las primeras reflexiones y la cola reverberante. La ecuación para calcularla es la siguiente [75]:

$$C_{80} = 10\log_{10}\left(\frac{\int_0^{0.08} h^2(t)\,dt}{\int_{0.08}^{\infty} h^2(t)\,dt}\right) \tag{4.2}$$

El tiempo de centro de gravedad t_s está caracterizado por el primer momento de la respuesta impulso energética [76]:

$$t_s = \frac{\int_0^{\infty} t\,h^2(t)\,dt}{\int_0^{\infty} h^2(t)\,dt} \tag{4.3}$$

este parámetro indica el instante en que la energía se encuentra en equilibrio (la energía de $h(t)$ para $t < t_s$ es igual a la energía posterior a t_s), además señala, bajo el criterio de la energía, el punto en donde se separa el sonido temprano ($t < t_s$) del reverberante ($t \geq t_s$). Una interpretación más formal de esta cantidad consiste en pensar que en t_s se concentra la energía promedio de la señal.

De acuerdo a la tabla 4.1, considerando el tiempo de reverberación y el EDT, las componentes de baja frecuencia de una señal radiada en el interior del recinto son más persistentes que las de alta frecuencia. Además, las altas frecuencias aparecen más claras que las bajas. Esto se debe, de acuerdo a t_s, a que la energía para altas frecuencias se

concentra antes de los primeros 80 ms y conforme la frecuencia baja el tiempo de centro de gravedad se incrementa. Lo que se puede decir del modelo de auditorio es que sus parámetros acústicos están dentro de los estándares contemplados en la literatura [76], aunque cabe hacer notar que estos valores son sólo validos para los materiales asignados a los objetos que conforman al modelo. Además, los parámetros calculados para frecuencias debajo de 1000 Hz deben ser tomados con ciertas reservas ya que sólo consideran información parcial sobre el campo sonoro (desde el ámbito de la acústica *geométrica*).

Con los diferentes tipos de análisis mostrados aquí se puede apoyar el diseño de una sala de conciertos, teatros o auditorios para que cumplan especificaciones muy precisas. Por ejemplo, basado en el análisis anterior, si se quiere que en el área frente al escenario el tiempo de reverberación sea mayor podrían colocarse algunos difusores sobre esa área para generar algunas ondas estacionarias. Para verificar su orientación se puede hacer uso de diagramas de reflexión y para observar como estos modifican la distribución de energía en la totalidad del recinto se aplica una prueba de distribución temporal de energía apoyado con cálculos de algunos parámetros acústicos.

4.4 Corrección de Trayectorias

Un análisis definitivo de la acústica de un recinto es el que uno puede hacer auditivamente, lo cual es perfectamente posible aplicando un tratamiento extra a los datos generados por el simulador.

Para empezar, se debe corregir los problemas intrínsecos del método de trazado de rayos que están asociados al tamaño finito del detector. Cuando el detector no es puntual, un conjunto de rayos que comparten la misma trayectoria puede incidir en el detector (ver Fig. 4.5), lo cual no puede ser tolerado si lo que se quiere es encontrar las trayectorias asociadas a un solo punto en el interior del espacio de interés.

Para un análisis de parámetros acústicos el uso de datos producidos por múltiples detecciones no es un problema ya que lo que se intenta obtener es una estimación numérica de la acústica de toda una zona o incluso de todo el recinto. Por otro lado, un evento auditivo se asocia a una posición fija del oyente, por lo tanto, las contribuciones de cada

trayectoria debe ser dada por un solo rayo que incida en el punto donde se quiere producir dicho evento auditivo.

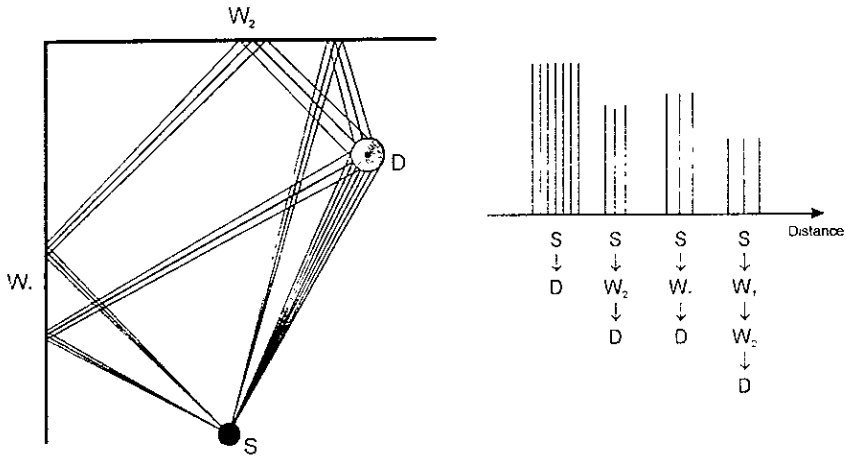

Figura 4.5. Incidencia múltiple de rayos sobre un detector esférico de tamaño fijo a través de cuatro trayectorias (izquierda) y la representación de cada rayo detectado considerando la distancia recorrida y la energía transportada (derecha). Las distancias más cortas asociadas a cada trayectoria se marcan con líneas más obscuras.

Un método para corregir las trayectorias* parte de calcular la respuesta impulso empleando detectores esféricos. Cuando se trazan rayos desde un punto sobre la superficie de la esfera (visible desde dicho punto) se nota que el rayo que recorre la menor distancia, apunta directamente al centro de la esfera (ver Fig. 4.5). Entonces, basta sólo con tomar el rayo con menor distancia recorrida del todo conjunto de rayos que comparten la misma trayectoria y sumarle el radio del detector para completar su recorrido hasta el centro de la esfera. Con este procedimiento se puede aproximar los resultados del método de trazado de rayos a los que se obtendrían aplicando el método de fuentes imagen. En la figura 4.6 se presenta las fuentes imagen resultantes de aplicar el método de corrección de trayectoria a la respuesta impulso energética del detector D_1.

* La corrección de trayectoria para aplicaciones auditivas es importante sólo para las primeras reflexiones, puesto que la cola reverberante (reflexiones de orden superior) se considera difusa, lo cual se obtiene al emplear un detector de tamaño finito.

El próximo paso para hacer audible el campo sonoro consiste en sintetizar una respuesta impulso empleando técnicas de procesamiento digital de señales, tomando como base los datos corregidos de la etapa de simulación, para finalmente reproducirla a través de audífonos o bocinas. Todo este proceso se tratará en detalle en el siguiente capítulo.

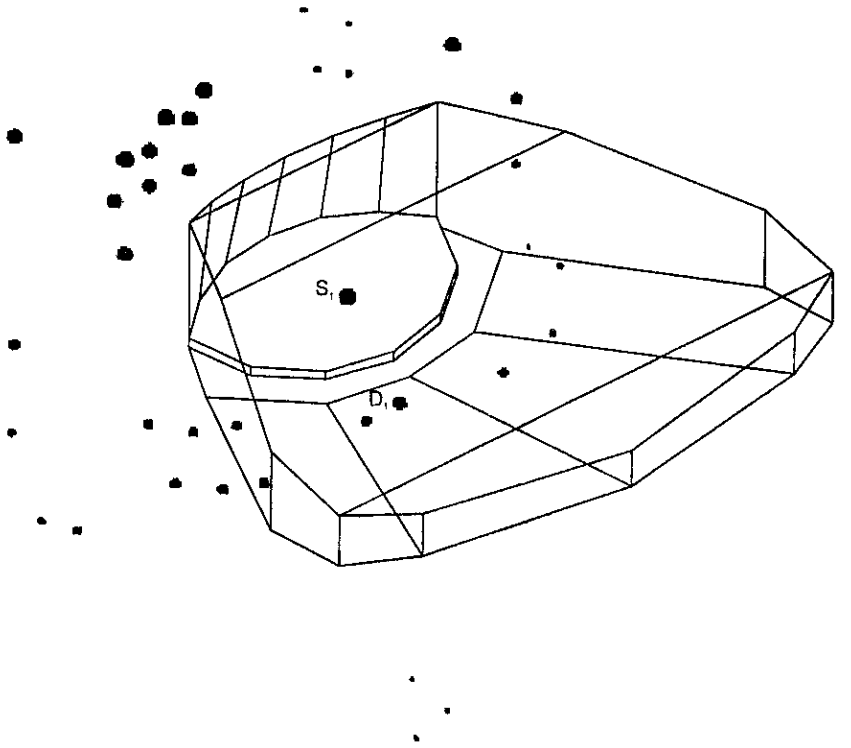

Figura 4.6. Distribución de las primeras fuentes imagen considerando la distancia a la que se encuentran del detector. El diámetro de las fuentes imagen es proporcional a la energía de radiación. La fuente y el detector se denotan por S_1 y D_1 respectivamente.

5

AURALIZACIÓN

El término *Auralización* se refiere al proceso de hacer audible una representación sintética del campo sonoro generado por una fuente dentro de un espacio, de tal forma que se produzca la sensación auditiva de estar inmerso en dicho campo. La representación del campo sonoro puede ser obtenida por modelado numérico, por modelado acústico a escala, o por otros medios.

El objetivo de la auralización es preservar las características acústicas que son más importantes para recrear la impresión auditiva de las característica acústicas del espacio modelado. La precisión y la calidad de la ilusión auditiva final estarán limitadas por los recursos de computo disponibles (o permisibles) para la realización del modelo.

5.2 Simulación de la Atenuación Sonora por Absorción del Aire

La atenuación del sonido en el medio de transmisión (normalmente aire) depende principalmente de la temperatura, humedad y de la distancia recorrida. Existen varios factores que participan en la absorción del sonido en el aire [38]. En un ambiente típico el más importante es la relajación térmica.

El aire, a grandes rasgos, actúa como un filtro paso bajas cuya frecuencia de corte depende de la distancia recorrida y de la humedad [77]. A altas frecuencias, el aire crea un efecto de dispersión, aunque este efecto puede ser ignorado debido a que las diferencias relativas entre los valores máximos y mínimos de la velocidad del sonido es sólo en promedio de 4.5×10^{-4} en el rango audible de frecuencias [78].

La atenuación sobre una trayectoria de longitud r, en metros, debida sólo a la absorción del aire puede ser expresada por

$$a(r) = \frac{p(r)}{p(0)} = \exp(-mr) \tag{5.1}$$

donde $p(r)$ es la presión sonora después de viajar la distancia r, $p(0)$ es la presión sonora inicial en $r = 0$ y m es el coeficiente de absorción del aire en Nepers por metro. Los valores de m pueden encontrarse en tablas [38] o calcularse por medio de expresiones analíticas [79,80], aunque lo más común es encontrar directamente los valores para la atenuación por absorción del aire como función de la temperatura, humedad y distancia [81,82]. Otro factor a considerar en la atenuación del sonido se debe a que conforme la onda de presión diverge, su amplitud decrece inversamente proporcional a la distancia recorrida [24].

Modelado de Filtros de Absorción del Aire

Se parte de suponer que la propagación del sonido en el aire se comporta linealmente y que el sistema es lineal e invariable en el tiempo. Con esta aproximación, el problema se concentra en como simular la absorción del aire en una forma eficiente, exacta y físicamente plausible.

Con base en ecuaciones estandarizadas para el cálculo de la atenuación del sonido por absorción atmosférica [82] es posible obtener la magnitud de la función de transferencia de la atenuación por absorción del aire como función de la frecuencia. Sin embargo, estos modelos de absorción normalmente no proveen información sobre la fase del espectro de atenuación. A fin de determinar una respuesta en el dominio del tiempo (real y causal) que sea compatible con la magnitud del espectro de atenuación resultante, es necesario determinar de alguna manera la fase del espectro de atenuación.

Una posibilidad es calcular la respuesta real y causal de fase mínima, para lo cual se prosigue del siguiente modo: Siendo $a(f_i)$ la magnitud del espectro de atenuación por absorción del aire en un rango de frecuencias f_i, con $i = 1,2,3,...,N$, para una determinada distancia, se debe asegurar primero la simetría en el dominio de la frecuencia para

así garantizar que el filtro sea real. Puesto que $a(f_i)$ no es simétrica, se emplea la relación

$$A(f_k) = \begin{cases} a(f_k) & \text{para } k \le N \\ a(f_{2N-k}) & \text{para } k > N \end{cases}, \tag{5.2}$$

donde $k = 1,2,3...,2N-2$. Aplicando la transformada de Hilbert al logaritmo natural de $A(f_k)$ se produce una distribución de fase mínima $\phi(f_k)$ para la magnitud de la función dada. Para agregar la componente de fase basta evaluar la ecuación

$$A(f_k) = A(f_k)e^{-j\{\ln[\phi(f_k)]\}} \tag{5.3}$$

con lo cual se obtiene la función de transferencia de un filtro FIR de fase mínima; lo cual garantiza la causalidad [83].

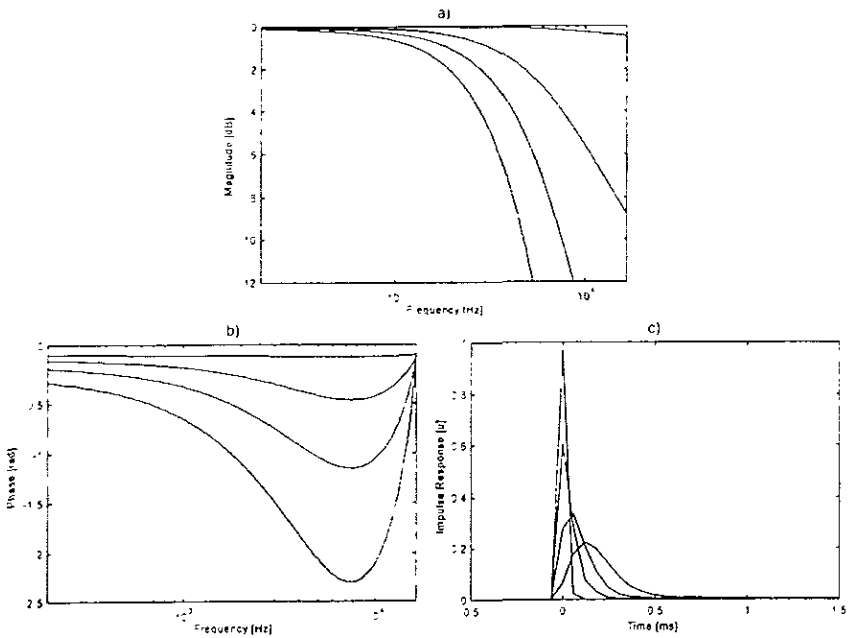

Figura 5.1 Respuestas de magnitud y fase de filtros de atenuación por absorción del aire para distancias correspondientes a 1, 20, 50 y 100 metros, bajo condiciones de temperatura y humedad relativa de 20°C y 20% respectivamente.

La figura 5.1 ilustra el modelado de la magnitud de la respuesta de filtros de atenuación por absorción del aire para cuatro distancias desde la fuente al receptor utilizando este método. En la figura 5.2 se muestra el efecto adicional de la atenuación por la distancia (de acuerdo a la ley $1/r$) en los filtros de las funciones de transferencia de la absorción del aire.

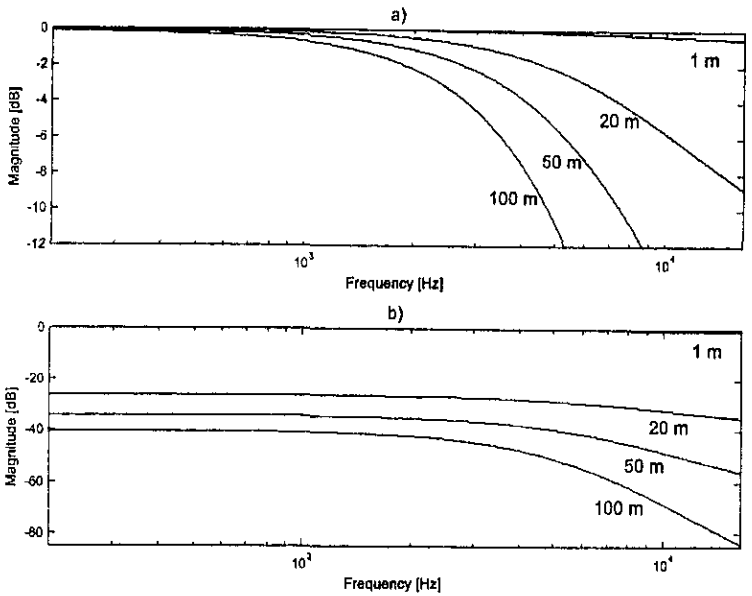

Figura 5.2. a) Magnitud de la atenuación por absorción del aire como función de la distancia y frecuencia. b) Magnitud combinada de la atenuación por absorción del aire y atenuación por la distancia. Para ambas gráficas se usaron condiciones de 20% de humedad y 20°C de temperatura.

5.3 Simulación de las Propiedades Acústica de los Materiales

El problema de modelar la reflexión de ondas sonoras sobre la superficie de materiales es complejo. Debido al comportamiento temporal y espectral del sonido reflejado como función del ángulo incidente, el fenómeno de dispersión y difracción, etcétera, dificulta el desarrollo de modelos numéricos que sean precisos en todos los aspectos. La forma más común de caracterización de materiales de superficies acústicas está basada en coeficientes de absorción medidos en bandas de octava de 125, 250, 500, 1000, 2000 y 4000 Hz. El coeficiente de absorción es la razón entre la energía absorbida y la

energía incidente. Por otra parte, la relación entre el coeficiente de absorción $\alpha(\omega)$ y el factor complejo de reflexión o reflectancia $R(j\omega)$ está dada por

$$\alpha(\omega) = 1 - R(j\omega)^2 \tag{5.4}$$

donde $R(j\omega) = \sqrt{1-\alpha(\omega)}$ puede ser usada para obtener la magnitud de los factores de reflexión cuando se conocen los coeficientes de absorción. A partir de esta información, es posible determinar una función de reflexión de fase mínima utilizando el mismo método descrito anteriormente. De esta manera, se puede obtener el espectro de reflexión del material (magnitud y fase) $R(j\omega)$, o bien la función de reflexión correspondiente en el dominio del tiempo $r(t)$, donde $r(t)$ es la transformada inversa de Fourier de $R(j\omega)$.

Modelado Filtros de Reflexión

Para modelar los filtros de reflexión puede aplicarse el método anterior a los valores de reflectancia calculados a partir de los coeficientes de absorción. Pero debido a que sólo se cuentan con seis valores por material, se debe realizar previamente un proceso de interpolación sobre un rango de frecuencias. Sin embargo, esto no es estrictamente correcto, ya que los coeficientes resultantes difieren de los originales. Esto se debe a que los coeficientes de absorción originales representan un valor medio sobre una cierta banda de frecuencias y no a una sola, como lo asume el método de interpolación. Paralelamente surge otro problema, no se conoce el comportamiento del material fuera del rango de 125 a 4000 Hz. Aunque pueda ser estimado, no se puede asegurar que el filtro resultante modele la respuesta del material para bajas y altas frecuencias.

Otro método consiste en usar los valores de reflectancia interpolados dentro del rango de 125 a 4000 Hz. para ser transformados en valores complejos, agregándole una componente de fase mínima; el tratamiento por fase mínima no causa efectos auditivos críticos [84]. Posteriormente se emplea un algoritmo que aplica un ajuste por mínimos cuadrados (de cero a la frecuencia de Nyquist) para modelar filtros IIR desde el dominio de la frecuencia [85]. En la figura 5.3 se

muestran algunos filtros de reflexión para diferentes materiales calculados por este método.

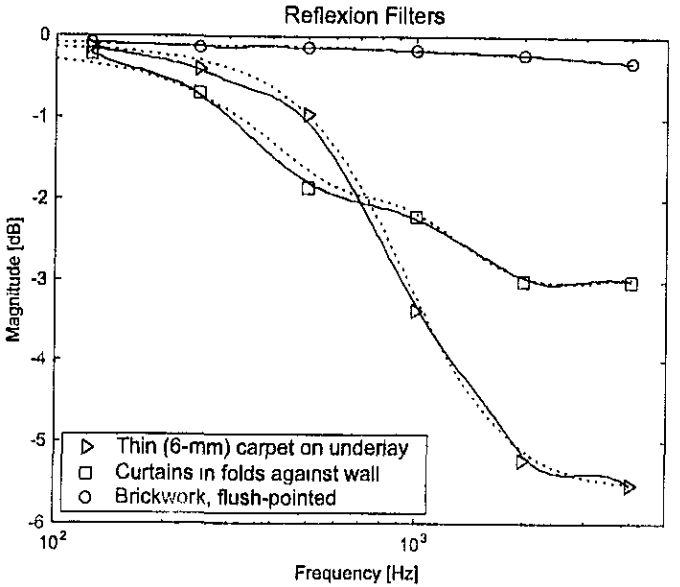

Figura 5.3. Tres diferentes filtros IIR de fase mínima diseñados para aproximarse a valores de reflectancia. La línea continua representa la respuesta objetivo y la punteada es la respuesta del filtro correspondiente.

5.4 Sonido 3D

Un elemento indispensable para la creación de auralizaciones realistas, además de un modelo adecuado de la transmisión del sonido en un recinto, es la inclusión de los efectos acústicos producidos por el torso, la cabeza y el oído del oyente (virtual), ya que estos proveen estímulos de gran relevancia en el proceso de audición espacial del ser humano. Existen modelos analíticos y numéricos que simulan la propagación y difracción de una onda alrededor de una esfera o una réplica de una cabeza humana [86-88]. Sin embargo la representación más fiel de estos efectos se da a través de las llamadas HRTFs (del inglés *Head-Related Transfer Functions*), las cuales representan la función de transferencia en el campo libre desde una fuente de sonido hasta cada uno de los canales auditivos (izquierdo y derecho) del sujeto (o, más comúnmente, de un maniquí acústico con las dimensiones del ser humano promedio).

Puesto que las HRTFs son salidas de un sistema lineal e invariable en el tiempo, sus respuestas impulso puede ser usadas directamente para realizar auralizaciones. Opcionalmente se puede considerar el diseño de filtros de arquitectura más sencilla (más eficientes, operacionalmente hablando) que aproximan la respuesta de las HRTFs. Esto es útil en auralizaciones interactivas, donde cualquier cambio en la posición de la cabeza del oyente o de la fuente dentro del espacio virtual debe ser modelado en el menor tiempo posible para asegurar continuidad en el evento auditivo.

En el sistema RA se emplearon únicamente las respuestas impulso de un conjunto de HRTFs como aproximaciones de filtros FIR. No se consideró el uso de métodos de diseño de filtros ya que la síntesis del campo sonoro se hace fuera de línea. A pesar de esto, resulta útil hacer una revisión de las diversas técnicas que se reportan en la literatura para el diseño de filtros de sonido 3D; esta revisión se presenta en seguida.

Diseño de Filtros para Sonido 3D

La manera más directa para aproximar HRTFs es usar el diseño de filtros FIR por ventaneo [89,90]. Un filtro de cualquier orden se obtiene aplicando una ventana rectangular a la respuesta impulso original. El uso de una ventana rectangular se justifica en que ésta exhibe una óptima aproximación a la respuesta en frecuencia original en el sentido de mínimos cuadrados [91]. Otra manera de aproximar mediciones de HRTF se basa en el uso de técnicas de estimación espectral para modelado de filtros IIR [92-95].

De acuerdo a estudios comparativos basados en criterios auditivos, los filtros FIR tienen un mejor desempeño que los IIR, aunque para los primeros se necesita un número mayor de coeficientes [91]. Sin embargo, el uso de filtros IIR alabeados (WIIR) mejora considerablemente su desempeño, además de disminuir el número de coeficientes significativamente [96]. En este tipo de filtros se considera el comportamiento del oído humano al momento de su diseño. Alabeo (*warping*) se refiere a un remuestreo del espectro de una función de transferencia a una escala alabeada de frecuencia de acuerdo a una resolución no lineal (como la escala de Bart [97] o la escala de ancho de banda equivalente [98]).

El método de diseño de filtros para sonido 3D depende principalmente de la aplicación y de que tanto realismo se quiere imprimir a la auralización. Para mayor información sobre técnicas de diseño de este tipo de filtros revisar [99,100].

5.5 Síntesis

La síntesis de la respuesta acústica se obtiene a través de la combinación de las respuestas impulso de todos los elementos descritos anteriormente, incluyendo la transmisión de sonido a través del medio, la reflexión de sonido en las superficies que limitan el espacio y las HRTFs del oyente. La forma en la que se combinan estos elementos está determinada por la información que contiene la historia de cada rayo trazado en la simulación de la respuesta impulso del recinto. La información que es relevante para la síntesis de la respuesta acústica total es la siguiente: la distancia recorrida por cada rayo, la lista de superficies visitadas y la dirección de incidencia sobre el detector. Con la distancia r se calcula el retardo τ de cada rayo y el filtro de absorción del aire, la lista de superficies se emplea para saber qué filtros de reflexión aplicar para cada trayectoria, y la dirección de incidencia señala qué filtro HRTF usar.

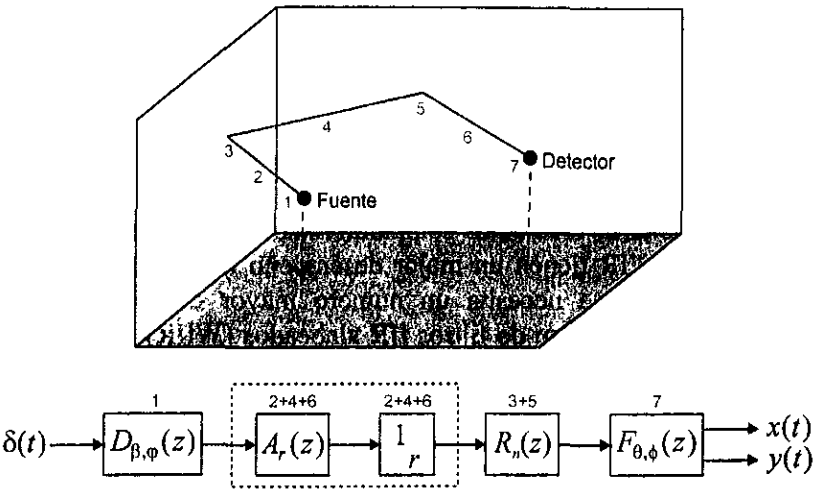

Figura 5.4. Ejemplo de la síntesis de una trayectoria de reflexión de segundo orden. Cada bloque representa a un filtro que depende de parámetros de entrada. El cuadro de línea punteada sugiere la aplicación de un solo filtro.

La figura 5.4 ilustra una trayectoria de reflexión de segundo orden y su representación como un sistema de filtros, el cual al ser excitado con un impulso $\delta(t)$ entrega dos señales de salida; $y(t)$ y $x(t)$. (que corresponden a la transmisión de sonido hasta cada uno de los dos oídos). Todos los filtros del sistema son lineales, por lo que pueden ser aplicados en un orden arbitrario. Los filtros para sintetizar la trayectoria del rayo de la figura 5.3 se describen a continuación:

- Directividad de la fuente 1, $D_{\beta,\varphi}(z)$.
- Atenuación por absorción del aire $A_r(z)$ para la distancia acumulada 2+4+6.
- Ley 1 r de atenuación por la distancia acumulada 2+4+6.
- Filtros de reflexión $R_n(z)$ de las superficies $n = 3, 5$.
- Filtro HRTF $F_{\theta,\phi}(z)$, 7, para reproducción en dos canales.

El filtro de directividad de la fuente se toma a partir de la dirección de salida del rayo, denotada aquí por β y φ. Este filtro puede omitirse si se considera una fuente omnidireccional.

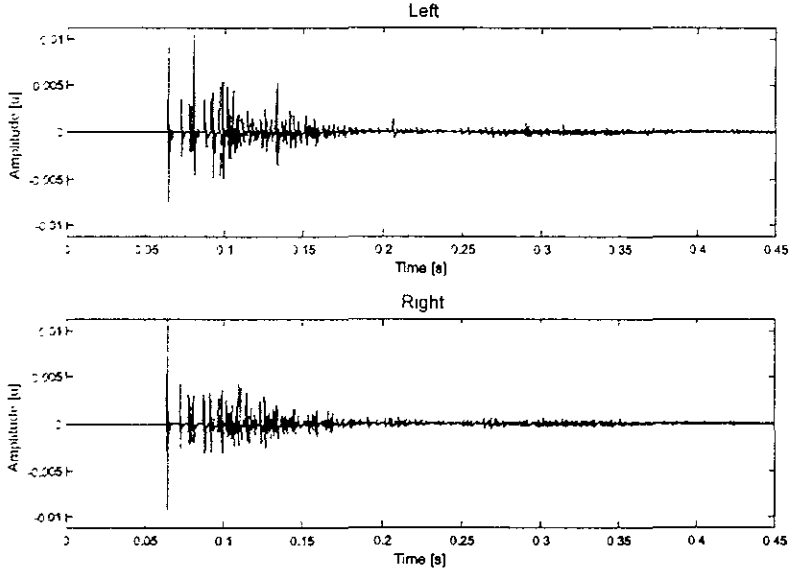

Figura 5.5. Primeros 0.4 sec. de una respuesta impulso biauricular sintetizada, donde la cabeza del sujeto está orientada directamente hacia la fuente.

El proceso de síntesis debe aplicarse a cada trayectoria *i* que conforma a la respuesta impulso. Posteriormente, las señales resultantes deben ser ensambladas sumando cada una de ellas considerando sus retardos asociados τ_i:

$$h_{left}(t) = \sum_{i=1}^{N} x_i(t-\tau_i) \quad \text{y} \quad h_{right}(t) = \sum_{i=1}^{N} y_i(t-\tau_i) \tag{5.5}$$

con esta última operación se obtiene la respuesta impulso biauricular del espacio modelado. En la figura 5.5 se muestra un ejemplo de una BIR sintetizada para el modelo presentado en la figura 4.1. Este tipo de síntesis modela tan sólo las contribuciones de reflexiones especulares al campo sonoro. Para modelar la parte difusa del campo, se puede emplear algunas de las técnicas propuestas en [61,67].

Esta técnica para sintetizar respuestas impulso biauriculares está sujeta a controversias, debido a que sólo intenta generar respuestas impulso combinando la mayor parte de la información acústica presente en el fenómeno de propagación de sonido en el interior de un recinto y no a través del modelado de dicho fenómeno físico. De hecho, la técnica considera al fenómeno tan sólo como un sistema.

5.5 Reproducción

La respuesta impulso biauricular representa un filtro FIR, y la generación de una señal de salida en términos de una señal de entrada se puede calcular mediante la convolución (discreta) de la señal de entrada con cada una de las dos respuestas a impulso del filtro biauricular. Sin embargo la longitud de la respuesta es grande (del orden de varios miles de coeficientes), por lo que la realización en tiempo real del proceso de convolución requiere un uso intenso de recursos de cómputo. En este caso, resulta imperativa la elección de un algoritmo adecuado de convolución rápida [7].

La reproducción de campos sonoros virtuales usando técnicas biauriculares puede realizarse a partir de grabaciones acústicamente secas (carente de ecos y reverberación), que pueden obtenerse realizando grabaciones en el interior de una cámara anecoica. El proceso de reproducción consiste entonces en la convolución de la

respuesta impulso biauricular con las señales anecoicas. Opcionalmente, se puede realizar procesamiento adicional a fin de ecualizar el sistema empleado en la reproducción de sonido (la cuál puede realizarse a través de audífonos, por ejemplo).

Convolución en Tiempo Real

Una convolución puede realizarse en dos formas: En el dominio del tiempo y en el dominio de la frecuencia. La selección entre una u otra depende de la longitud de las señales a convolucionar. Cuando la longitud del filtro es extremadamente grande, se recurre a variaciones de algoritmos de convolución en el dominio de la frecuencia, los cuales requieren del orden de $\log_2(N)$ operaciones aritméticas por cada muestra de salida de un filtro de N coeficientes, en vez de las N que se requieren en una implementación en el dominio del tiempo. Sin embargo, el algoritmo produce un retardo en la salida de aproximadamente N muestras. La implantación de un algoritmo de convolución en el dominio de la frecuencia se basa en la siguiente propiedad de la transformada discreta de Fourier (DFT): La DFT inversa del producto de dos DTFs corresponde a una convolución circular [7].

La convolución circular claramente no es igual a la convolución lineal, de hecho sólo algunos términos corresponden con esta última. Por ejemplo, para las secuencias $x(m)$ y $h(n)$, donde $1 \le m \le M$, $1 \le n \le N$, con $N < M$, y $h(n) = 0$ para $n > N$, la convolución circular se puede expresar como

$$y(m) = \sum_{n=1}^{N} h(n)x(m-n) \qquad (5.6)$$

De las muestras resultantes de la convolución solamente las últimas $M - N + 1$ corresponden a una convolución lineal. Ahora bien, si la secuencia $x(m)$ es infinita, es necesario calcular la convolución circular en bloques de entrada sucesivos de longitud M, y para asegurar un flujo continuo de datos de salida, se debe repetir el cálculo a una tasa de $R \le M - N + 1$ muestras por bloque, lo cual implica que cada bloque de entrada contendrá R muestras nuevas y $M - R \ge N - 1$ muestras estarán traslapadas. Estas últimas deben ser guardadas y reubicadas entre bloques de entrada sucesivos, de ahí el nombre de

esta técnica: Método de "traslapa y guarda" [83] (*overlap-save*, en inglés).

Esta técnica puede implantarse eficientemente usando la trasformada rápida de Fourier (FFT), para lo cual las longitudes de los bloques comúnmente son potencias de dos. A continuación se presenta una forma de implementar el método de "traslapa y guarda" para aplicaciones en tiempo real:

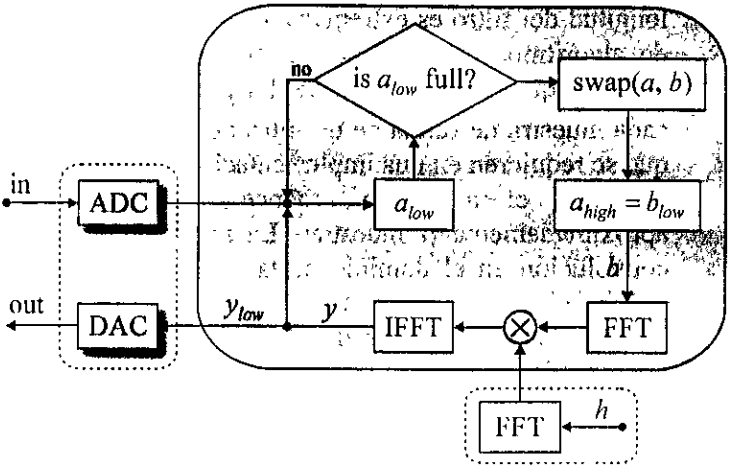

Figura 5.6. Esquema descriptivo para implantar el método de convolución rápida de "traslapa y guarda". Los recuadros punteados sugieren el uso de un solo bloque.

Se tiene una secuencia de datos $h(n)$ con $1 \le n \le N$ el cual debe ser convolucionado con una secuencia de longitud infinita, en otras palabras, una secuencia generada por un convertidor analógico digital. Primero el vector h debe ser extendido a una potencia de dos mayor o igual que el doble de su longitud original. Esto se hace agregando $M - N$ ceros al final de h, donde

$$M = 2^{\text{ceil}\left(\frac{\log(2N)}{\log(2)}\right)} \tag{5.7}$$

Se debe contar con tres vectores adicionales: $y(m)$, $a(m)$ y $b(m)$, donde $1 \le m \le M$. El primero es el vector de convolución y los dos últimos funcionan como bloques de almacenamiento temporal. Estos vectores deben ser tratados como si estuvieran compuestos de dos

vectores de longitud $L = M\ 2$, el vector superior y el vector inferior. Por ejemplo, para el vector a:

$$a_{high} = \begin{bmatrix} a(1) \\ a(2) \\ \vdots \\ a(L) \end{bmatrix} \quad \text{y} \quad a_{low} = \begin{bmatrix} a(1+L) \\ a(2+L) \\ \vdots \\ a(2L) \end{bmatrix} \tag{5.7}$$

Definido lo anterior, se procede a aplicar el proceso ilustrado en la figura 5.6. Para una convolución en tiempo real, se aplica una rutina extra de llenado y vaciado que opera en paralelo con el algoritmo. Esta rutina es la responsable de acumular en a_{low} una nueva muestra de entrada y producir la salida leyendo un dato de y_{low} para cada intervalo de muestreo.

Una vez lleno a_{low} (lo cual implica que todos los datos de y_{low} fueron leídos) se permuta el contenido de los bloques a y b, esto con el fin de liberar el bloque a para recibir nuevas muestras. Enseguida de la permutación, se procede a copiar en a_{high} las muestras viejas almacenadas ahora en b_{low}. La convolución circular producida al calcular la IFFT del producto de las FFTs del bloque b y el vector h, estará contenida en el vector y. Ahora sólo resta tomar la parte que corresponde a la convolución lineal almacenada en y_{low}, lo cual comienza un nuevo ciclo.

Este algoritmo fue implantado en una tarjeta TMS320C40 para operar en tiempo real y producir una convolución en dos canales.

5.6 Simulaciones y Auralizaciones Demostrativas

Con el fin de mostrar los resultados generales de este trabajo se realizaron simulaciones de la respuesta impulso en el modelo de la figura 4.1. Con los datos generados en las simulaciones, se sintetizaron varias respuestas impulso biauriculares para diferentes posiciones del oyente. En la serie de figuras 5.7 al 5.10, se presentan cuatro BIRs donde se ilustra también la situación del oyente al realizar la síntesis.

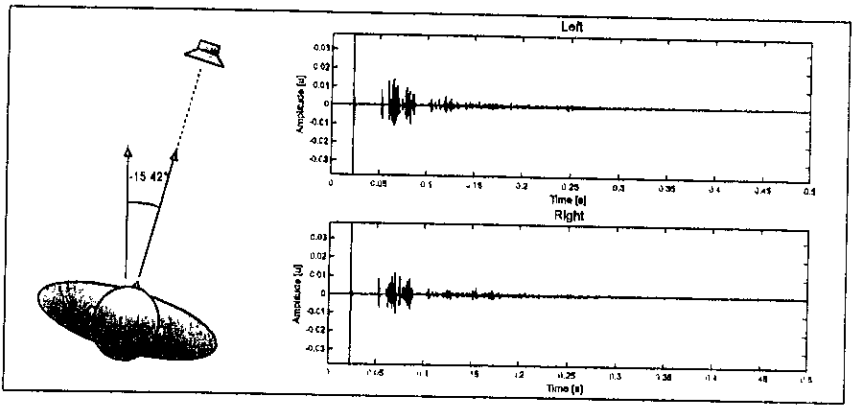

Figura 5.7. Respuesta impulso biauricular para un sujeto orientado directamente hacia la fuente, la cual está desviada 15.42° a la derecha y 0.096° hacia arriba del eje frontal. Archivo correspondiente: `fS1D1a.wav`.

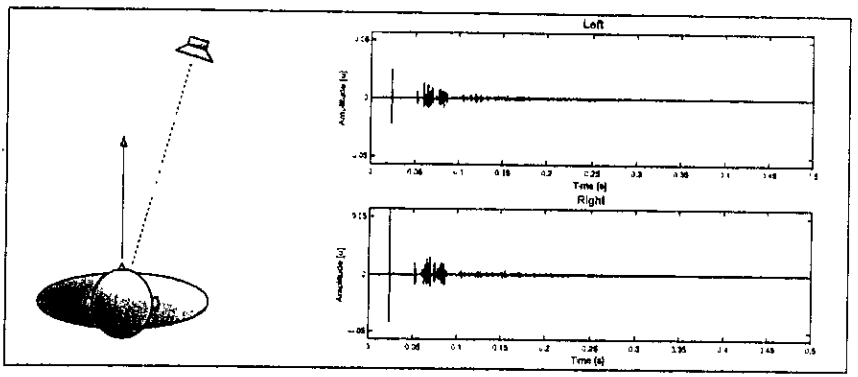

Figura 5.8. Respuesta impulso biauricular para un sujeto alineado sobre el eje frontal (rotación 0°) con la cabeza elevada ligeramente. Archivo correspondiente: `fS1D1b.wav`.

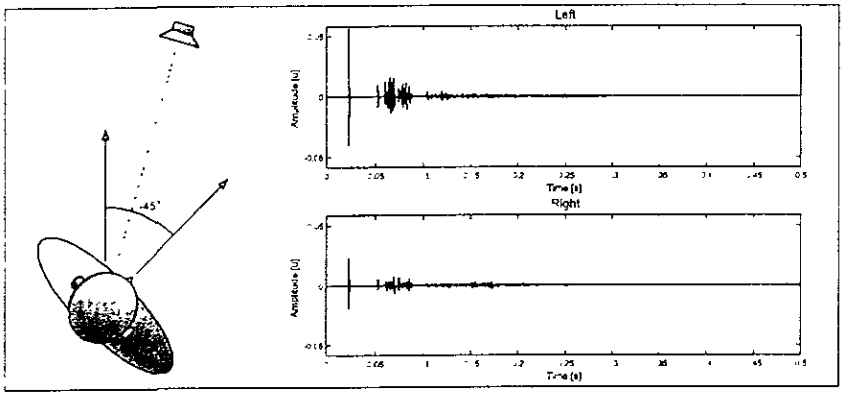

Figure 5.9. Respuesta impulso biauricular para un sujeto orientado 45° a la derecha; la cabeza está elevada ligeramente. Archivo asociado a ésta imagen: fS1D1c.wav

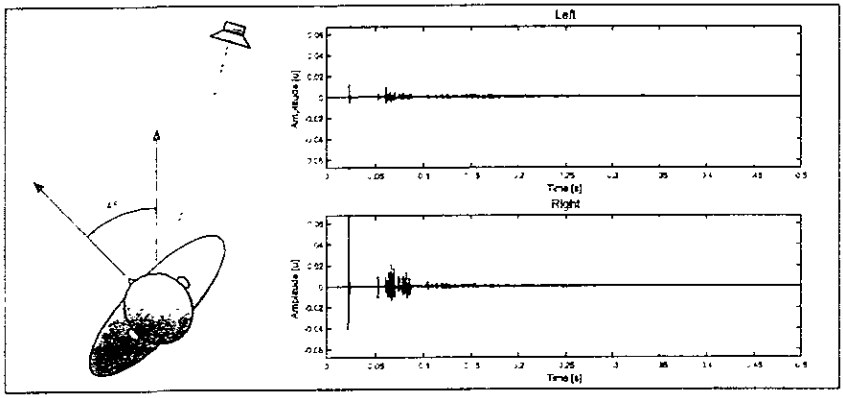

Figura 5.10. Respuesta impulso biauricular para un sujeto orientado 45° a la izquierda; la cabeza está elevada ligeramente. Archivo asociado a esta figura: fS1D1d.wav.

Como se puede observar, cuando el sujeto rota su cabeza la amplitud de las señales del oído izquierdo y derecho varia en comparación a cuando ésta se sitúa directamente hacia la fuente.

Algo que no resulta tan obvio en las figuras es la diferencia de tiempo interauricular asociada a esas rotaciones, y menos aún la manera en como el campo sonoro simulado envuelve al oyente. Por ello, se decidió incluir en esta tesis un disco magnético flexible que contiene una serie de archivos de sonido con BIRs y algunas auralizaciones para que el lector pueda evaluar auditivamente todos los atributos espaciales.

El disco magnético anexo a este trabajo contiene el archivo `Examples.zip` (comprimido con el programa WinZip™) en donde se encuentra una serie de archivos de sonido en formato WAV con respuestas impulso biauriculares a una frecuencia de muestreo de 44.1 kHz, estos archivos se listan en la tabla 5.1. Los nombres de los archivos indican en forma codificada, qué fuente y qué detector se empleó en la medición de la respuesta impulso. Por ejemplo, el archivo `fS1D1a.wav` corresponde a una BIR medida de la fuente S_1 al detector D_1 para una determinada orientación 'a' del oyente y la 'f' hace referencia al nombre del recinto; *Forum*.

		Orientación del Oyente	
Archivo	Distancia	Azimut	Elevación
`fS1D1a.wav`	07.93 m	-15.42°	0.096°
`fS1D1b.wav`	07.93 m	0°	0.096°
`fS1D1c.wav`	07.93 m	-45°	0.096°
`fS1D1d.wav`	07.93 m	45°	0.096°
`fS1D2a.wav`	22.01 m	-26.8°	-0.160°
`fS1D2b.wav`	22.01 m	0°	0°
`fS1D2c.wav`	22.01 m	-45°	0°
`fS1D2d.wav`	22.01 m	45°	0°

Tabla 5.1. Listado de archivos con BIRs para diferentes posiciones y orientaciones del oyente. La segunda columna se refiere a la distancia que hay entre la posición de la fuente y el punto donde está colocado el oyente.

Las señales listadas arriba fueron normalizadas con un factor 14.85 ya que debido a la distancia a la que se encuentra la fuente (y a la

absorción de los materiales), los valores de la amplitud de las BIRs son muy bajos. Si se desea usar estas señales para realizar auralizaciones se deben dividir por el factor de normalización para recuperar sus valores originales, el no hacer esa operación se perdería valiosa información sobre las condiciones del espacio simulado.

Para hacer evaluaciones auditivas se incluye un archivo de nombre `Hola.wav` que contiene una señal de voz acústicamente seca, la cual puede ser convolucionada con las BIRs. Como ejemplos de auralizaciones se incluyeron archivos de nombre `Hola_*.wav`, las cuales dan una mejor apreciación subjetiva sobre atributos espaciales del campo sonoro.

La reproducción de los archivos de sonido debe realizarse a través de audífonos para asegurar que las señales exciten el oído correcto, además de mantener las respuestas simuladas relativamente libres de interferencias. En una reproducción con altavoces llegan contribuciones a ambos oídos de las señales designadas para el oído izquierdo y derecho, además éstas se ven afectadas por la respuesta del cuarto en donde se esté realizando la reproducción. Esto puede arreglarse hasta cierto punto usando filtros *cross-talk* y filtros inversos para eliminar la respuesta del cuarto. Sin embargo, se debe mantener la cabeza fija para no alterar el evento auditivo.

Al escuchar el grupo de auralizaciones D1, la voz se distingue con cierta claridad (mayor que en las auralizaciones D2). Esto se debe a que en este caso (D1) el punto de prueba se encuentra a 7.93 m de la fuente (relativamente cerca del escenario), y por lo tanto el sonido directo y las primeras reflexiones tienen una mayor amplitud relativa con respecto al sonido reverberante. Esto produce una sensación de estar envuelto por el campo sonoro. En el grupo de auralizaciones D2 el sonido no es tan claro, y de hecho se percibe difuso. Lo anterior se debe a que el punto de prueba se encuentra en la parte posterior del foro (a 22.1 m de la fuente) donde la composición del campo sonoro está dominada por contribuciones de sonido reverberante (correspondiente a reflexiones de orden superior). Además, debido a que el techo en esa zona es más bajo se producen ondas estacionarias, lo cual lo hace más reverberante (más difuso) el campo sonoro.

Cuando se reproducen los archivos con terminación 'a', la fuente se percibe frente a uno. Al escuchar detenidamente los archivos con terminación 'b', para los cuales el oyente está dirigido al frente del escenario, tal vez pueda distinguirse un ligero desplazamiento de la fuente a la derecha (para algunas personas esto no será muy evidente dado que la auralización se basa en HRTFs distintas a las propias). Los archivos con terminación 'c' y 'd' simulan una rotación de la cabeza a la derecha e izquierda, respectivamente; por lo que en las auralizaciones, la fuente de sonido se percibirá hacia la izquierda (terminación 'c') y hacia la derecha (terminación 'd'). En estos dos casos, las diferencias de amplitud de las señales en cada oído se percibirán un poco más claramente. Todo esto es congruente con la posición de la fuente que se modeló; es decir, situada ligeramente a la derecha de los puntos de prueba (correspondientes a la posición virtual del oyente).

6

CONCLUSIONES

6.1 Principales Resultados de la Tesis

Los principales resultados de la tesis pueden resumirse como sigue:

- El empleo de técnicas de programación orientada a objetos resulta ser una herramienta práctica para implantar sistemas en evolución para aplicaciones científicas. Esto es evidente para el programa de simulación presentado aquí, ya que la clase que aplica el método de trazado de rayos no se ve alterada por cualquier nueva definición de material con textura, fuente o forma geométrica.

- El simulador de recintos es lo suficientemente flexible y general como para considerar espacios de geometría arbitraria que puedan ser aproximados con superficies planas, así mismo, no presenta restricciones en el número de fuentes y detectores a emplear en una simulación.

- Los datos generados por el bloque simulación del sistema RA son suficientes para llevar a cabo análisis acústicos en cualquier modelo aproximado por superficies planas, considerando siempre que estos reflejaran sólo una aproximación basada en acústica geométrica.

- La síntesis de un campo sonoro a través de la combinación de respuestas impulso, incorpora la información suficiente como

para producir en el oyente la sensación de estar inmerso en un determinado escenario auditivo y lo cual permite evaluar auditivamente sus atributos acústicos. Esto puede verificarse con los archivos de sonido adjuntos a este trabajo.

6.2 Contribuciones

En esta tesis se presenta una recopilación de la literatura más representativa sobre simulación biauricular de recintos, la cual cubre diferentes áreas de investigación: Acústica, Psicoacústica, Tecnología Biauricular, Procesamiento Digital de Señales, Procesamiento Digital de Audio y una nueva área introducida en este trabajo; Programación Orientada a Objetos. Además, se hicieron las siguientes contribuciones directas al campo de la simulación biauricular de recintos:

- Una manera general, flexible y eficiente de implementar el método de trazado de rayos, útil para cualquier aplicación relacionada con radiación a altas frecuencias, por ejemplo, para la simulación de recintos acústicos, gráficas por computadora o para diseño de antenas conformadas.

- Una nueva manera de presentar datos de una simulación de trazado de rayos para reflejar la manera en como se distribuye la energía sobre un área para diferentes lapsos de tiempo, los cuales son útiles para observar las variaciones en la reverberación en diferentes zonas.

- Un nuevo procedimiento para corregir trayectorias para simulaciones que aplican el método de trazado de rayos.

- Desarrollo de un conjunto de herramientas para evaluar la acústica de un recinto, sintetizar y reproducir ambientes acústicos virtuales.

6.3 Trabajo Futuro

Para obtener resultados aún más cercanos a la acústica real de algún recinto se deben evaluar otros métodos para simular campo sonoros (por ejemplo, métodos basados en acústica ondulatoria), modelar

correctamente la manera en como un material refleja el sonido creando filtros de reflexión direccionales basándose directamente en mediciones y hacer una investigación más profunda sobre la implantación de filtros de sonido 3D.

Este trabajo proporciona la estructura general para un sistema más ambicioso en el cual se pretende crear espacios auditivos virtuales interactivos, donde el usuario pueda moverse libremente a través del modelo y experimentar la sensación real de estar inmerso en dicho espacio.

La realización de este objetivo requiere más trabajo de investigación y desarrollo, pero los resultados presentados en esta tesis prueban la viabilidad de crear espacios acústicos virtuales.

A1

CLASES SOPORTE

Vector3D Class

Constructors	
Vector3D()	*Default*
Vector3D(float, float, float)	*Define a vector via x-y-z values*
Vector3D(float)	*Define a vector with all values equal*
Vector3D(const Vector3D& v)	*Copy constructor*
Public Member Functions	
float& X()	*Read and write access by reference*
float& Y()	*Read and write access by reference*
float& Z()	*Read and write access by reference*
const float& X() const	*Read-only access by reference*
const float& Y() const	*Read-only access by reference*
const float& Z() const	*Read-only access by reference*
Vector3D Projection(const Vector3D&) const	*Vectoronal projection*
int DominantDir() const;	*Extract the dominant direction*
int Turn(const Vector3D&) const	*Detecting a left versus a right turn*
float Sum() const	*Adds the vector elements*
float Modulus() const	*Computes the modulos of the vector*
void Normalise()	*Apply a normalisation to the vector*
Overloaded Operators	
const float& operator [](unsigned) const	*Read-only access to vectors elements*
float& operator [](unsigned)	*Read and write access to vector elements*
Vector3D operator+ () const	*Compound assigment operator*
Vector3D& operator+= (const Vector3D&)	*Compound assigment operator*
Vector3D& operator*= (const Vector3D&)	*Compound assigment operator*
Vector3D& operator/= (const Vector3D&)	*Compound assigment operator*
Vector3D& operator*= (float)	*Compound assigment operator*
Vector3D& operator/= (float)	*Compound assigment operator*

Friends	
float Modulus(const Vector3D&)	*Gets the modulus of a vector*
Vector3D Normalise(const Vector3D&)	*Normalise a vector*
float DotProd(const Vector3D&, const Vector3D&)	*Applies dot product*
Vector3D CrossProd(const Vector3D&, const Vector3D&)	*Applies cross product*
int DirCmp(const Vector3D&, const Vector3D&)	*Compares directions*
Vector3D operator+ (const Vector3D&, const Vector3D&)	*Aritmetical operator*
Vector3D operator- (const Vector3D&, const Vector3D&)	*Aritmetical operator*
Vector3D operator (const Vector3D&, const Vector3D&)*	*Aritmetical operator*
Vector3D operator/ (const Vector3D&, const Vector3D&)	*Aritmetical operator*
Vector3D operator* (const Vector3D&, float)	*Aritmetical operator*
Vector3D operator* (float, const Vector3D&)	*Aritmetical operator*
Vector3D operator/ (const Vector3D&, float)	*Aritmetical operator*
Vector3D operator/ (float, const Vector3D&)	*Aritmetical operator*
int operator== (const Vector3D&, const Vector3D&)	*Relational operator*
int operator!= (const Vector3D&, const Vector3D&)	*Relational operator*
int operator< (const Vector3D&, const Vector3D&)	*Relational operator*
int operator> (const Vector3D&, const Vector3D&)	*Relational operator*
Int operator<= (const Vector3D&, const Vector3D&)	*Relational operator*
int operator>= (const Vector3D&, const Vector3D&)	*Relational operator*
ostream& operator<< (ostream&, const Vector3D&)	*Inserts a ray into output stream*
istream& operator>> (istream&, Vector3D&)	*Extracts a ray from input stream*

Point Class

Constructors	
Point()	*Default*
Point(float, float, float)	*Define a point via x-y-z values*
Point(float)	*Define a point with all values equal*
Point(const Vector3D& p)	*Converts a vector to a point*
Point(const Point& p)	*Copy constructor*
Public Member Function	
void OutFormat(int)	*Sets the output format*
Friends	
friend float Distance(const Point&, const Point&)	*Cumputes the distance between two points*
friend ostream& operator<<(ostream&, const Point&)	*Inserts a ray into output stream*
Base Class	
Vector Class	*Public*

AbsList<T> Class

Constructors	
AbsList()	*Default*
Public Member Function	
unsigned Length()	*Returns the number nodes*
Pure Virtual Functions	
int IsEmpty()	*Returns 1 if is empty, else return 0*
int IsFull()	*Returns 1 if is full, else return 0*
void Insert(const T&)	*Append a node*
void Clear()	*Removes all nodes*

List<T> Class

Constructors	
List()	*Default*
List(const List<T>&)	*Copy constructor*
Protected Member Function	
ListNode* First() const	*Returns the first node*
Public Member Function	
int IsEmpty()	*Returns 1 if is empty, else return 0*
int IsFull()	*Returns 1 if is full, else return 0*
void Insert(const T&)	*Append a node*
void Clear()	*Removes all nodes*
void Delete(unsigned)	*Removes ith node*
void Append(const T&)	*Inserts a node at end of list*
void Head(const T&)	*Inserts a node at begin of list*
void Insert(const T&, unsigned)	*Inserts a node at specified location*
const T& Node(unsigned)	*Returns the ith item*
T& Node(unsigned)	*Returns access to ith item*
Overloaded Operators	
const T& operator [](unsigned) const	*Returns the ith item*
T& operator [](unsigned)	*Returns access to ith item*
List<T>& operator= (const List<T>&)	*Replaces a list for other*
Base Class	
AbsList Class	*Public*

SortList<T> Class

Constructors	
SortList()	*Default*
SortList(const List<T>& l)	*Construction via list to sort*
Public Member Function	
void Insert(const T&)	*Insert a node in sorted order*
Base Class	
List Class	*Public*

Iterator<T> Class

Constructors	
Iterator()	*Default*
Pure Virtual Functions	
int More() const	*Returns 1 if current position is a valid position*
void Next()	*Advance a node*
void ReStart()	*Sets current position to first*
unsigned Index() const	*Returns current index*
Virtual Overloaded Operators	
const T& operator()() const	*Returns item in current position*
int operator++ ()	*Advance a node (prefix)*
int operator++ (int)	*Advance a node (postfix)*

ListItr<T> Class

Constructors	
ListItr(const List<T>&)	*Sets iteration by refence*
ListItr(const List<T>*)	*Sets iteration by using a pointer*
Public Member Functions	
int More() const	*Returns 1 if current position is a valid position*
void Next()	*Advance a node*
void ReStart()	*Sets current position to first*
unsigned Index() const	*Returns current index*
Overloaded Operators	
T& operator()()	*Returns item in current position*
const T& operator()() const	*Modifies item in current position*
int operator++ ()	*Advance a node (prefix)*

int operator++ (int)	*Advance a node (postfix)*
Base Class	
Iterator Class	*Public*

SearchListItr<T> Class

Constructors	
SearchListItr(const List<T>&)	*Sets search iteration by refence*
SearchListItr(const List<T>*)	*Sets search iteration by using a pointer*
Public Member Functions	
int Search(T)	*Sets current position to view value*
int IsFound(T)	*If value is found returns 1 else 0*
int Find(T)	*Return 1 if a value is found, seting current position, -1 if more than one value is found. seting a list of indices; else 0*
void Min()	*Sets current position at the minor item*
Int Min(T)	*Sets current position at an item value and if it is founded return 1 else 0*
List<unsigned> Indices()	*Returns a list with found item indices*
Base Class	
ListItr Class	*Public*

SortListItr<T> Class

Constructors	
SortListItr(const List<T>&)	*Sets sort iteration by refence*
SortListItr(const List<T>*)	*Sets sort iteration by using a pointer*
Public Member Functions	
void BubbleSort()	*Applies the bubblesort algorthm*
Base Class	
ListItr Class	*Public*

ioListItr<T> Class

Constructors	
ioListItr(const List<T>& l, int m = COLUMN)	*Sets iteration by refence*
ioListItr(const List<T>* l, int m = COLUMN)	*Sets iteration by using a pointer*
Public Constant	
enum { ROW, COLUMN }	*Flags to read-write mode*

Public Member Functions	
int Save(const char *) const	*White a list in disk*
int Load(const char *)	*Reads a list from disk*
void DisplayMode(int)	*Sets the display mode (0: ROW, 1: COLUMN)*
Friend	
ostream& operator<< (ostream&, ioListItr<T>&)	*Inserts a list into output stream*
Base Class	
ListItr Class	*Public*

A2

CLASES DEL SIMULADOR

Ray Class

Constructors	
Ray()	*Default*
Ray(const Point&, const Vector3D&, float i=1)	*Uses origin, direction and level*
Ray(const Ray&)	*Copy constructor*
Public Member Functions	
const Point& Loc() const	*Get ray location*
const Vector3D& Dir() const	*Get ray direction*
const float& Amp() const	*Get ray amplitude*
float Interp(const Point&) const	*Get the distance to reach, or be near, a point*
Overloaded Operators	
Ray& operator=(const Ray&)	*Copy a ray*
float operator/(const Ray&) const	*Ray amplitude division*
Friends	
ostream& operator<<(ostream&, const Ray&)	*Inserts a ray into output stream*
istream& operator>>(istream&, Ray&)	*Extracts a ray from input stream*

RayHistory Class

Constructors	
RayHistory()	*Default*
RayHistory(const Ray&)	*Begins the history of a given ray*
Public Member Functions	
const float& LengthPath() const	*Returs the distance traveled by a ray*
const Point& HitPoint() const	*Returns the point where the ray impinges*
int Decay(const float& d = 1e-06) const	*Returns 1 if the ray energy reaches d value*
void TravelOver(const float&)	*Moves the ray certain distance*

void PutHitObjCode(const string&)	*Add in a list the objeject code*
string HitObjs() const	*Get a list with the hit object code*
List<Point> PathPoints() const	*Returns the points that define a ray path*
void NewRay(const Vector3D&, const float&)	*Update the ray's history*
Overloaded Operators	
const Ray& operator()()const	*Get access to current ray*
RayHistory& operator=(const Ray&)	*Setup ray's history*

Source Class

Constructors	
Source()	*Default*
Source(const Point&, float i = 1)	*Uses source location and radiation level*
Source(const Source&)	*Copy constructor*
Public Member Functions	
const Point& Location() const	*Put location by reference*
Point& Location()	*Get location by reference*
const float& Level() const	*Put level by reference*
float& Level()	*Get level by reference*
Pure Virtual Function	
Ray GenRay()	*Generates a ray acording to derivate class*

OSource Class

Constructors	
OSource()	*Default*
OSource(const Point&, float i = 1)	*Uses location and radiation level*
Public Member Function	
Ray GenRay()	*Generates rays with the same level*
Friends	
ostream& operator<<(ostream&, const Osource&)	*Inserts a source into output stream*
istream& operator>>(istream&, OSource&)	*Extracts a source from input stream*
Base Class	
Source Class	*Public*

Shape Class

Constructor	
Shape()	*Default*
Public Member Functions	
char* Name() const	*Get shape name*
Vector3D Normal(const Point&) const	*Returns the normal at the point*
float NearestDistance(const Ray&) const	*Returns the nearest distance*

Plane Class

Constructors	
Plane()	*Default*
Plane(float, float, float, float)	*Uses plane equation parameters*
Plane(const Vector3D&, const Point&)	*Uses normal and a point*
Plane(const Point&, const Point&, const Point&)	*Uses three points*
Plane(const Point&, const Point&)	*Build a 2D plane*
Public Member Functions	
char* Name() const	*Get a string with the word plane*
Vector3D Normal(const Point&) const	*Returns the normal at the point*
float NearestDistance(const Ray&) const	*Returns the nearest distance*
int PointOnPlane(const Point&) const	*Checks if the point is on the plane*
Friends	
Vector3D Projection(const Vector3D&, const Plane&)	*Proyects a vector on the plane*
Point Projection(const Point&, const Plane&)	*Proyects a point on the plane*
ostream& operator<<(ostream&, const Plane&)	*Inserts a plane into output stream*
istream& operator>>(istream&, Plane&)	*Extracts a plane from input stream*
Base Class	
Shape Class	*Public*

Sphere Class

Constructors	
Sphere()	*Default*
Sphere(const Point&, float)	*Uses location and radius*
Public Member Functions	
char* Name() const	*Get a string with the word sphere*
Vector3D Normal(const Point&) const	*Returns the normal at the point*

float NearestDistance(const Ray&) const	*Returns the nearest distance*
Friends	
ostream& operator<<(ostream&, const Sphere&)	*Inserts a sphere into output stream*
istream& operator>>(istream&, Sphere&)	*Extracts a sphere from input stream*
Base Class	
Shape Class	*Public*

Polygon Class

Constructor	
Polygon()	*Default*
Protected Member Function	
int Precompute()	*Precompute some polygon parameters*
Public Member Functions	
char* Name() const	*Get a string with the word polygon*
Vector3D Normal(const Point&) const	*Returns the normal at the point*
float NearestDistance(const Ray&) const	*Returns the nearest distance*
unsigned NumVerts() const	*Returns the number of vertices*
int IsInside(const Point&) const	*Checks if a point is inside polygon*
float Area() const	*Compute the polygon area*
void Display() const	*Displays the information of the polygon*
Pure Virtual Functions	
Point& Vertex(unsigned)	*Read and write access by reference*
const Point& Vertex(unsigned)	*Read-only access by reference*
Overloaded Operators	
int operator==(const Polygon&) const	*Checks if two polygons are equal*
int operator!=(const Polygon&) const	*Checks if two polygons are differents*
Base Class	
Plane Class	*Public*

sPolygon Class

Constructors	
sPolygon()	*Default*
sPolygon(const List<Point>&)	*Uses a list of vertices*
sPolygon(const wPolygon&)	*Converts a wPolygon into a sPolygon*
Public Member Functions	
char* Name() const	*Get a string with the word sPolygon*

Point& Vertex(unsigned i)	*Read and write access by reference*
const Point& Vertex(unsigned i) const	*Read-only access by reference*
int Build(const List<Point>&)	*Builds a sPolygon via a list of points*
Overloaded Operators	
sPolygon& operator=(const sPolygon&)	*Copy a sPolygon object*
sPolygon& operator=(const wPolygon&)	*Copy a wPolygon in a spolygon object*
int operator==(const sPolygon&) const	*Checks if two sPolygons are equal*
int operator!=(const sPolygon&) const	*Checks if two sPolygons are differents*
int operator==(const wPolygon&) const	*Compares a sPolygon with a wpolygon*
int operator!=(const wPolygon&) const	*Compares a sPolygon with a wpolygon*
Friends	
ostream& operator <<(ostream& , const sPolygon&)	*Inserts a sPolygon into output stream*
Istream& operator >>(istream& , sPolygon&)	*Extracts a sPolygon from input stream*
Base Class	
Polygon Class	*Public*

wPolygon Class

Constructors	
wPolygon()	*Default*
wPolygon(const List<Point>&)	*Uses a reference to a list of vertices*
wPolygon(const List<Point>*)	*Uses a pointer to a list of vertices*
wPolygon(const List<unsigned>&, List<Point>&)	*Uses a index list and a list of vertices*
wPolygon(const sPolygon&, List<Point>*)	*Converts a sPolygon to a wPolygon*
Public Member Functions	
char* Name() const	*Get a string with the word wPolygon*
Point& Vertex(unsigned i)	*Read and write access by reference*
const Point& Vertex(unsigned i) const	*Read-only access by reference*
Overloaded Operators	
wPolygon& operator=(const sPolygon&)	*Copy a wPolygon object*
int operator==(const wPolygon&) const	*Checks if two wPolygons are equal*
int operator!=(const wPolygon&) const	*Checks if two wPolygons are differents*
int operator==(const sPolygon&) const	*Compares a wPolygon with a spolygon*
int operator!=(const sPolygon&) const	*Compares a wPolygon with a spolygon*
Friends	
ostream& operator <<(ostream& , const wPolygon&)	*Inserts a wPolygon into output stream*
istream& operator >>(istream& , wPolygon&)	*Extracts a wPolygon from input stream*
Base Class	

Polygon Class	*Public*

Line Class

Constructors	
Line()	*Default*
Line(const Point&, const Point&)	*Uses two points to define a line*
Line(const Line&)	*Copy constructor*
Public Member Functions	
Point& From()	*Read and write access to the 1st point*
const Point& From() const	*Read-only access to the 1st point*
Point& To()	*Read and write access to the 2nd point*
const Point& To() const	*Read-only access to the 2nd point*
float Length() const	*Returns the length of the line*
int PointOnLine(const Point&) const	*Checks if a point is on the line*
int Intersection(const Line&) const	*Checks if a line intersects other*
Overloaded Operators	
Line& operator=(const Line&)	*Copy a line*
Line operator-() const	*Swap 1st point and 2nd point*
int operator==(const Line&) const	*Checks if its length is equal to other one*
int operator!=(const Line&) const	*Checks if its length is diferent to other one*
int operator<(const Line&) const	*Compares its length with other one*
Conversion Function	
operator Vector3D() const	*Converts a line into a vector*
Friends	
int operator==(const Line&, const Line&)	*Checks if both lengths are equals*
int operator!=(const Line&, const Line&)	*Checks if both lengths are differents*
int operator<(const Line&, const Line&)	*Compares lengths*
ostream& operator<<(ostream&, const Line&)	*Inserts a line into output stream*

Channel<T> Class

Constructors	
Channel()	*Default*
Public Member Functions	
T Rx()	*Receive message*
void Tx(const T&)	*Send message*
int IsThereMsg() const	*Checks if there is a message*

Detector Class

Constructors	
Detector(const int&)	*On-off contructor*
Detector(const Detector&)	*Copy constructor*
Public Member Functions	
void On()	*Turn on the detector*
void Off()	*Turn off the detector*
void Connect(Channel<PacketData>*)	*Jack to connect to the channel*
int IsConnected() const	*Checks if the detector is connected*
Virtual Functions	
int IsAbleToDetect() const	*Checks it the detector is on*
int Detect(const RayHistory&)	*Apply detection*
Friends	
istream& operator >>(istream&, Detector&)	*Extracts a Detector from input stream*
ostream& operator <<(ostream&, const Detector&)	*Inserts a Detector into output stream*

OutFileList Class

Constructors	
OutFileList()	*Default*
OutFileList(const String& , const String& , const int&)	*Uses a label, an extension and a number of files to open*
Public Member Functions	
unsigned NumFiles() const	return the number of opened files
void Label(const String&, const String&)	Setd the file names and extension
void Prepare(const int&)	Prepares n files, use before Label()
void Prepare(const String&, const String&, const int&)	Prepares n files
void Close(unsigned)	Close and delete from list the ith file
void Close()	close all files and reset stream list
Overloaded Operators	
ofstream& operator[] (unsigned) const	Get access to a file from the list

Recorder Class

Constructors	
Recorder()	*Default*
Public Member Functions	

channel<DataPacket>* Channel()	*Accessing to the channels*
void Prepare()	*Prepares tapes for recording*
int Counter(unsigned)	*Counts number of data recorded per channel*
void Rec()	*Search for a message to record*
void Stop()	*Stops recording and save all data*
void UnPlug(unsigned)	*Unplugs either all or a specified channel*

Material Class

Constructors	
Material()	*Default*
Material(float)	*Uses an absorption coeficient value*
Protected Member Functions	
float& AbsCoef()	*Read and write access by reference*
const float& AbsCoef() const	*Read-only access by reference*
Virtual Functions	
float Absorption(const Vector3D&, const Vector3D&)	*Returns the absorption coeficient acording to the incidence angle*

TexturedMaterial Class

Constructors	
TexturedMaterial()	*Default*
TexturedMaterial(float)	*Uses an absorption coeficient value*
Pure Virtual Functions	
Vector3D Reflect(const Vector3D&, const Vector3D&) const	*Returns the reflexión angle according to derivate class and considering the incident angle*
Base Class	
Material Class	*Public*

Surface Class

Constructors	
Surface()	*Default*
Surface(Shape*, TextureMaterial*)	*Define a surface by its shape and material*
Public Member Functions	
void SetUp(Shape*, TextureMaterial*)	*Setup a surface by its shape and material*
void Reflect(RayHistory&) const	*Modifies the ray's path and level*

float NearestDistance(const Ray&) const	*Returns the nearest distance via Ray*
float NearestDistance(const RayHistory&) const	*Returns the nearest distance via RayHistory*
const float& StoredDistance()	*Read-only access to a variable wich store a value*
unsigned& Parent() const	*Read and write access to the number of the group where belongs the surface*
const unsigned& IDNum() const	*Returns a identification number*
const string IDCode()	*Returns a identification number*
Base Class	
Detector Class	*Public*

Object Class

Constructors	
Object()	*Default*
Public Member Functions	
void AddSurface(Surface&)	*Adds a new surface to the object via reference*
void AddSurface(Surface*)	*Adds a new surface to the object via pointer*
List<Point>& VertexList()	*Access to a list of vertices*
Surface* NearestHitSurface(const Ray&) const	*Returns the nearest hit surface*
int IsAbleToDetect() const	*Checks if one o more surfaces can detect*
void Connect(Channel<PacketData>*)	*Connect*
unsigned NumDetectors() const	*Returns the number of surfaces which are able to detect*

RayTracer Class

Constructors	
RayTracer()	*Default*
RayTracer(List<Object*>*)	*Uses a pointer to the list of pointers to objects*
Public Member Functions	
void SetThr(float)	*Sets the threshold of atenuation*
void SetObjsPtr(List<Object*>*)	*Sets the pointer to the list of pointers to objects*
void TraceRays(const Ray&)	*Trace ray till decay*
void TraceRays(const int&, const Ray&)	*Trace rays till n reflections*
void RecOn()	*Turns on the ray-data recorder*
void RecOff()	*Turns off the ray-data recorder*
int CompletedDetections(unsigned)	*Check if at least n reflexions were made*
List<Point> PathRay() const	*Returns a list of points from the ray tracing process*

Measure Class

Constructors	
Measure()	*Default*
Measure(const string& fn)	*Needs the name of the RAF file*
Public Constant	
enum { CRAY, CDET }	*Flags to stop simulation*
Public Member Functions	
void Load(const string&)	*Loads a list of objects from a RAF-file*
void AttenuationThreshold(float)	*Sets attenuation threshold*
List<Point> RayTracing(unsigned, unsigned)	*Traces rays using the kth source*
List<Point> RayTracing(const Ray&, unsigned)	*Traces rays using a given ray*
List<Point> RayTracing(const Vector3D&, unsigned, unsigned)	*Trace rays from the kth source on a given direction*
void ImpulseResponse(unsigned n, int)	*Computes the impulse response*
Base Class	
RayTracer Class	*Public*

BIBLIOGRAFÍA

[1] Gilkey, R. H., and Weisenberger, J. M., "The sense of presence for suddenly deafened adult", Presence, **4**, 357-363 (1995)

[2] Ramsdell, R. S., *The Psychologhy of Hard-of hearing and the Deafened Adult*, 4th Edition, New York: Holt, Rinehart & Winston (1978)

[3] Julstrom, S., "A high-performance surround sound process for home video", J. Audio Eng. Soc., **35**, 536-562 (1987)

[4] Scherer, P., "Ein neues Verfahren der raumbezogenen Stereophonie mit verbesserter Übertragung der Rauminformation", Rundfunktechnische Mitteilungen, **21**, 196-204 (1977)

[5] Gerzon, M. A., "Periphony: With-height sound reproduction", J. Audio Eng. Soc., **21**, 2–10 (1973).

[6] Gerzon, M. A., "Ambisonics in multichannel broadcasting and video", J. Audio Eng. Soc., **33**, 859-871 (1985)

[7] Malham, D. And Myatt, A., "3-D sound spatialization using ambisonic techniques". Computer Music J., **19**, 58–70 (1995)

[8] Pulkki, V., "Virtual sound source positioning using vector base amplitude panning", J. Audio Eng. Soc., **45**, 456–466 (1997)

[9] Møller, H., "Fundamentals of binaural technology", Applied Acoustics, **36**, 171-218 (1992)

[10] Damaske, P., "Head related two channel sterophony with loudspeaker reproduction", J. Acoust. Soc. Am., **50**, 1109-1115 (1971)

[11] Miyoshi, M. and Koizumi, N., "NNT's research on acoustics for future telecommunication services", Applied Acoustics, **36**, 307-326 (1992).

[12] Lehnert, H. and Blauert, J., "Principles of binaural simulation", Applied Acoustics, **36**, 259-291 (1992)

[13] Kleiner, M., Dalenbäck, B. and Svensson, P., "Auralization, an overview", J. Audio Eng. Soc., **41**, 861-874 (1993)

[14] Savioja, L., Huopaniemi, J., Lokki, T. and Väänänen, R., "Creating interactive virtual acoustic environments", J. Audio Eng. Soc., **47**, 675–705, (1999)

[15] Välimäki, V. and Takala, T., "Virtual musical instruments – natural sound using physical models", Organised Sound, **1**, 75–86 (1996)

[16] Orduña, F. y Elizalde, R., "Desarrollo de un simulador de instrumentos musicales de viento", En Memorias SOMI XIV Congreso de Instrumentación, Tonantzintla, Puebla, México. 533-537 (1999)

[17] Smith, J. O., "Physical modeling using digital waveguides", Computer Music J., **16**, 74–87 (1992)

[18] Välimäki, V., Huopaniemi, J., Karjalainen, M. and Jánosy, Z., "Physical modeling of plucked string instruments with application to real-time sound synthesis", J. Audio Eng. Soc., **44**, 331–353 (1996)

[19] Smith, J. O., "Principles of digital waveguide models of musical instruments" In *Applications of Digital Signal Processing to Audio and Acoustics*, Boston: Kluwer Academic (1997)

[20] Meyer, J., *Acoustics and the Performance of Music*, Frankfurt/Main:Verlag das Musikinstrument (1978)

[21] Fletcher, N. H. and Rossing, T. D., *The Physics of Musical Instruments*, New York: Springer (1991)

[22] Guzmán Ávalos, A., *Simulación de Recintos Acústicos*, Tesis de Ingeniería. México: Universidad Autónoma de México (1998)

[23] Blauert, J., *Spatial Hearing. The Psychophysics of Human Sound Localization*, 2nd Edition, Cambridge: MIT Press (1999)

[24] Cremer, L. And Müler, Helmut A., *Principles and Aplications of Room Acoustics*, Volume 1, London: Applied Science Publishers (1982)

[25] Kuttruff, H., "Auralization of Impulse responses modeled on the basis of ray-tracing results", J. Audio Eng. Soc., **41**, 876-880 (1993)

[26] Allred, J. C. and Newhouse, A., "Applications of the montecarlo method to architectural acoustics", J. Acoust. Soc. Am., **30**, 1-4 (1958)

[27] Krockstad, A, Strøm, S. and Sørsdal, S., "Calculating the acoustical room response by the use of a ray tracing technique", J. Sound Vib., **8**, 118-125 (1968)

[28] Schroeder, M. R., "Digital simulation of sound transmission in reverberant spaces", J. Acoust. Soc. Am., **47**, 424–431 (1970)

[29] Schroeder, M. R., "Computer models for concert hall acoustics", Am. J. Physics, **41**, 461–471 (1973)

[30] Lyon, R. and DeJong, R., *Theory and Application of statistical Energy Analysis*, 2nd edition, Massachusetts: Butterworth-Heinemann (1995).

[31] Xiang, N. and Blauert, J., "Binaural scale modeling for auralization and prediction of acoustics in auditoria", Applied Acoustics, **38**, 267-290 (1993)

[32] Spandöck, F., "Die Vorausbestimmung der Akustik eines Raumes mit Hilfe von Modellversuchen", Proc. 5th Int. Conf. Acoustics, 313-320 (1965)

[33] Brebeck, D., Bucklein, R. and Spandöck, F., "Akustisch ähnliche Modelle als Hilfsmittel für der Raumakustic", Acustica, **18**, 213-226 (1967)

[34] Oguchi, K., Ikeda, S. and Nagata, M., "Application of binaural hearing to scale-model testing", J. Audio Eng. Soc., **41**, 931-937 (1993)

[35] Polack, J. D., Meynial, X. and Grillon, V., "Auralizarion in scale models: Procesing of impulse responses", J. Audio Eng. Soc., **41**, 939-944 (1993)

[36] Borish, J., and Angell, J. B., "An efficient algorithm for measuring the impulse response using pseudorandom noise". J. Audio Eng. Soc., **31**, 478-486 (1983)

[37] Rife, D. D., and Vanderkooy, J., "Transfer-function measurement with maximum-length sequences", J. Audio Eng. Soc., **37**, 419-444 (1989)

[38] Kuttruff, H., *Room Acoustics*, 3nd. Edition, London: Applied Science Publishers (1991)

[39] Cox, T. J., and Lam, Y. W., "Evaluation of methods for predicting the scattering from simple rigid panels", Applied Acoustics, **40**, 123-140 (1993)

[40] Pietrzyk, A., "Computer modeling of the sound field in small rooms", Proc. AES 15th Int. Conf. on Audio, Acoustics & Small Spaces, Copenhagen, Denmark, 24-31 (1998)

[41] Seybert, A. F. and Wu, T. W., "Acoustic modeling: Boundary element methods", In *Encyclopedia of Acoustics*, Vol. 1, New York: John Wiley & Sons (1997)

[42] Botteldooren, D., "Finite-difference time-domain simulation of low-frequency room acoustic problems", J. Acoust. Soc. Am., **98**, 3302–3308 (1995)

[43] Savioja, L., Backman, J., Järvinen, A., and Takala, T., "Waveguide mesh method for low-frequency simulation of room acoustics", In Proc. International Congress on Acoustics (ICA'95), Vol. 2, Trondheim, Norway, 637-641 (1995)

[44] Strikwerda, J., *Finite Difference Schemes and Partial Differential Equations*, Chapman & Hall, New York (1989)

[45] Van Duyne, S. and Smith, J. O., "Physical modeling with the 2-D digital waveguide mesh", In Proc. Int. Computer Music Conf. (ICMC'93), Tokyo, Japan, 40–47 (1993)

[46] Van Duyne, S. and Smith, J. O., "The 2-D digital waveguide mesh", In Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA'93), Mohonk, New Paltz, NY (1993)

[47] Savioja, L., *Modeling Techniques for Virtual Acoustics*, Doctorate Thesis, Finland: Helsinki University of Technology (1999)

[48] Savioja L., Rinne, T. and Takala, T., "Simulation of Room Acoustics with a 3-D Finite Difference Mesh", In Proc. International Computer Music Conference (ICMC'94), Aarhus, Denmark, 463-466 (1994)

[49] Savioja, L., Karjalainen, M. and Takala, T., "DSP Formulation of a Finite Difference Method for Room Acoustics Simulation". In NORSIG'96 IEEE Nordic Signal Processing Symposium, Espoo, Finland, 455-458 (1996)

[50] Schetelig, T. and Rabenstein, R., "Simulation of three-dimensional sound propagation with multidimensional wave digital filters". In Proc. Int. Conf. Acoust., Speech, Signal Processing (ICASSP'98), Vol. 6, Seattle, WA, 3537–3540 (1998)

[51] Allen, J. B. And Berkley, D. A., "Image method for efficient simulating small-room acoustics", J. Acoust. Soc. Am., **65**, 943-950 (1979)

[52] Borish, J., "Extension of the image model to arbitrary polyhedra", J. Audio Eng. Soc., **75**, 1827-1836 (1984)

[53] Kulowski, A., "Algorithmic representation of the ray tracing technique", Applied Acoustics, **18**, 449-469 (1984)

[54] Stephenson, U., "Comparison of the mirror image source method and the sound particle simulation method", Applied Acoustics, **29**, 35-75 (1990)

[55] Gibbs, B., M. and Jones, D. K., "A simple image method for calculating the distribution of sound pressure levels within an enclosure", Acustica, **26**, 24-32 (1972)

[56] Lee, H. and Lee, B.-H., "An efficient algorithm for the image model technique", Applied Acoustics, **24**, 87-115 (1988)

[57] Kulowski, A., "Optimization of a point-in-polygon algorithm for computer models of sound field in rooms", Applied Acoustics, **35** (1992) 63-74.

[58] Van Maercke, D., "Simulation of sound fields in time and frequency domain using a geometrical model", In Proc. 12th Int. Congr. Acoust. (ICA'86), Vol. 2, Toronto, Canada, E11-7 (1986)

[59] Vorländer, M., "Simulation of the trasient and steady-state sound propagation in rooms using new combined ray-tracing/image-source method", J. Acoust. Soc. Am., **86**, 172-178 (1989)

[60] Naylor, G. M., "ODEON – another hybrid room acoustical model", Applied Acoustics, **38**, 131–143, 1993.

[61] Heinz, R., "Binaural room simulation based on an image source model with addition of statistical methods to include the diffuse sound scattering of walls and predict the reverberant tail", Applied Acoustics, **38**, 145-159 (1993)

[62] Stephenson, U., "Eine Schallteilchen-Computersimulation zur Berechnung der für die Hörsamkeit in Konzersälen maßgebenden Parameter", Acustica, **59**, 1-20 (1984)

[63] Vorländer, M., "Ein Strahlverfolgungs-Verfahren zur Berechnung von Shallfeldern in Räumen", Acustica, **65**, 138-148 (1988)

[64] Ondet, A.M. and Barbry, J. L., "Modeling of sound propagatión in fitted workshops using ray tracing", J. Acoust. Soc. Am., **85**, 787-796 (1989)

[65] Lehnert, H. "Systematic error of the ray tracing algorithm", Applied Acoustics, **38**, 207-221 (1993)

[66] Vian, J. P. and Maercke, D., "Calculation of the room impulse response using a ray-tracing method", In Proc. International Congress on Acoustics (ICA'86), Vancouver, Canada, 74-78 (1986)

[67] Lewers, T., "A combined beam tracing and radiant exchange computer model of room acoustics", Applied Acoustics, **38**, 161-178 (1993)

[68] Nakagawa, K., Miyahima, T. and Tahara, Y., "An improved geometrical sound field analysis in rooms using scatered sound and audible using room acoustic simulator", Applied Acoustics, **38**, 115-129 (1993)

[69] Stroustrup, B., *The C++ Programing Languaje*, 2nd Edition, Massachusets: Addison-Wesley (1994)

[70] Sedgewick, R., *Algorithms in C++: Fundamentals, Data Structures, Sorting, Searching*, 3rd Edition, Massachusets: Addison-Wesley (1999)

[71] Eckel, B., *Thinking in C++*, Volume 1, 2nd Edition, New Jersey: Prentice Hall (2000)

[72] Guzmán, A., Orduña, F. y Ruiz, R., "Diseño e implementación de una jerarquia de clases para la simulación de espacios acústicos usando programación orientada a objetos", En Memorias SOMI XIV Congreso de Instrumentación, Tonantzintla, Puebla, México, 528-532 (1999)

[73] Hill, F. S., *Computer Graphics*, New York: Macmillian Publishing Company (1990)

[74] Schroeder, M. R., "New method of measuring reverberation time", J. Acoust. Soc. Am., , 409-412 (1965)

[75] Beranek, L., *Concert and Opera Halls How they Sound*, New York: Acoustical Society of America (1996)

[76] Kürer, R., "Einfaches Meßverfaren zur Bestimmung der 'Schwerpunktzeit' raumakustischer Impulsantworten", Proc. 7th Int. Congr. on Acoustics, Budapest, 23-A-5 (1976)

[77] Evans, E. J., and Bazley, E. N., "The absorption of sound in air at audio frequencies". Acustica, **6**, 238-45 (1956)

[78] Kneser, H. O., "Die Dispersion hochfrequenter Schallwellen in Kohlensäure". Annalen der Physik, 5 Folge, 777-784 (1931).

[79] Bass, H. and Bauer, H. J., "Atmospheric absorption of sound: Analytical expressions". J. Acoust. Soc. Am., **52**, 821–825 (1972)

[80] Bass, H. E., "Absorption of sound by the atmosphere" Phys. Acoust. **17** (1984)

[81] Sutherland, L. C. and Gilles, A. D. "Atmospheric sound propagation", in *Encyclopedia of Acustics*, Volume 3, New York: John Wiley & Sons (1997)

[82] Standard ISO 9613-1, "Calculation of the Absorption of Sound by the Atmosphere", in *Acoustics — Attenuation of Sound During Propagation Outdoors* — Geneva: International Standards Organization (1993)

[83] Oppenheim, A.V. and Schafer, R.W., *Discrete-Time Signal Processing*, New Jersey: Prentice Hall (1989)

[84] Kuttruff, H., "On the audibility of phase distortions in rooms and its significance for sound reproduction and digital simulation in room acoustics", Acustica, **74**, 3-7 (1991)

[85] Matlab, *Signal Processing Toolbox User Guide*, Massachusetts: The MathWorks Inc., (1999)

[86] Cooper, D. H. and Bauck, L., "Corrections to L. Schwarz, 'Zur Theorie der Beugung einer evenen Schallwelle auf der Kugel', Akust Z. Vol. 8, 91-117 (1943)", J. Acoust. Soc. Am.. **80**, 1793-1802 (1986)

[87] Rabinowitz, W. M., Maxwell, J. and Wei, M., "Sound localizations cues for magnified head: Implications from sound difraction about a rigid sphere", Presence, **2**, 125-129 (1993)

[88] Duda R. and Matens, W., "Range-dependece of the HRTF of spherical head", J. Acoust Soc. Am., 104, 3048-3058 (1998)

[89] Kistler, D. and Wightman, F., "A model of head-related transfer functions based on principal components analysis and minimum-phase reconstruction", J. Acoust. Soc. Am., **91**, 1637–1647 (1992)

[90] Begault, D., "Challenger to the successful implementation of 3-D sound", *J. Audio Eng. Soc.*, **39**, 864–870 (1991)

[91] Sandvad, J. and Hammershøi, D., "Binaural auralization. Comparison of FIR and IIR representation of HIRs," Presented at the 96th AES Convention, Amsterdam, The Netherlands, 3862-3870 (1994)

[92] Blommer, M. A. And Wakefield, G. H., "On the design of pole-zero approximations using a logarithmic error measure", IEEE Trans. Signal processing, **42**, 3245-3248 (1994)

[93] Asano, F., Suzuki, Y. and Sone, T., "Role of spectral cues in median plane localization", J. Acoust. Soc. Am., **88**, 159-168 (1990)

[94] Kulkarni, A. and Colburn, H. S., "Infinite-impulse-response filter models of the head-related transfer function", J. Acoust. Soc. Am., **97**, 3278 (1995)

[95] Mackenzie, J., Huopaniemi, J., Välimäki, V., and Kale, I., "Low-order modelling of head-related transfer functions using balanced model truncation", IEEE Signal Proc. Letters (1996)

[96] Houpaniemi, J.and Karjalainen, M., "Comparación of filter design methodes for 3-D sound", IEEE Nordic Signal Processing Syposium (NORSIG96) 24-27 (1996)

[97] Zwicker, E. and Fastl, H., *Psychoacoustics: Facts and Models*, Heidelberg: Springer Verlag (1990)

[98] Moore, B. and Glasberg, B., "Suggested formulae for calculating auditory-filter bandwidths and excitación patterns", J. Acoust. Soc. Am., **74**, 750-153 (1983)

[99] Houpaniemi, J. and Smith, J. O., "Spectral and Time-Domain Preprocessing and the choice of modeling error criteria for binaural digital filters", in Proc. AES 16th Int. Conf. on Spatial Sound Reproducción, Rovaniemi, Findland, 301-312 (1999)

[100] Houpaniemi, J., Zacharov N. and Karjalainen, M., "Objetive and subjetive evaluation of head related transfer function design" J. Audio. Eng. Soc., **47**, 218-239