IMPLEMENTACIÓN EN TIEMPO REAL DE MODELOS DE SONIDO BINAURAL DENTRO DE UN SISTEMA TRIDIMENSIONAL INTEGRADO DE IMAGEN Y AUDIO

HENRY FABIÁN GIORGI BARRERA
JAVIER RICARDO MORENO RODRÍGUEZ

BOGOTÁ D.C.
PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA ELECTRÓNICA
2005

IMPLEMENTACIÓN EN TIEMPO REAL DE MODELOS DE SONIDO BINAURAL DENTRO DE UN SISTEMA TRIDIMENSIONAL INTEGRADO DE IMAGEN Y AUDIO

HENRY FABIÁN GIORGI BARRERA
JAVIER RICARDO MORENO RODRÍGUEZ

Trabajo de grado para optar al título de
Ingeniero Electrónico

Director
JAIRO ALBERTO HURTADO LONDOÑO M.Sc.
Ingeniero Electrónico

BOGOTÁ D.C.
PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA ELECTRÓNICA
2005

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA ELECTRÓNICA

RECTOR MAGNÍFICO:	**R.P. GERARDO REMOLINA VARGAS S.J.**
DECANO ACADÉMICO:	**Ing. FRANCISCO JAVIER REBOLLEDO MUÑOZ**
DECANO DEL MEDIO UNIVERSITARIO:	**R.P. ANTONIO JOSÉ SARMIENTO NOVA S.J.**
DIRECTOR DE CARRERA:	**Ing. JUAN CARLOS GIRALDO CARVAJAL**
DIRECTOR DEL PROYECTO:	**Ing. JAIRO ALBERTO HURTADO LONDOÑO**

ARTÍCULO 23 DE LA RESOLUCIÓN No. 13 DE JUNIO DE 1946

“La universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado.

Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque los trabajos no contengan ataques o polémicas puramente personales. Antes bien, que se vea en ellos el anhelo de buscar la verdad y la justicia”

AGRADECIMIENTOS

Agradecemos muy especialmente a las personas que hicieron posible la exitosa culminación del presente trabajo de grado, en especial a los ingenieros Jairo Alberto Hurtado Londoño y César Julio Bustacara quienes estuvieron siempre al tanto del desarrollo y prestos a brindar una asesoría efectiva.

De igual forma agradecemos muy especialmente a nuestros padres por su apoyo desinteresado y constante, a ellos debemos el haber podido finalizar con éxito el exigente proceso que constituyó todo el curso de la carrera. Esperamos haber llenado sus expectativas.

CONTENIDO

pág

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

Pág.

LISTA DE ANEXOS

pág.

INTRODUCCIÓN

La percepción del mundo para un ser humano se obtiene a través de los sentidos. En especial la vista, el oído y el tacto permiten que las personas obtengan una información tridimensional del entorno en que se desenvuelven. Con el pasar del tiempo, el cerebro aprende a interpretar la información recibida permitiendo que la persona logre ubicar las fuentes de sonido dentro de un marco tridimensional para de esta forma interactuar con el ambiente que la rodea.

Recientemente se ha experimentado un creciente interés en la reproducción virtual de ambientes tridimensionales buscando alcanzar altos niveles de realismo para los usuarios que los experimentan e interactúan con ellos. Para lograr tales niveles de realismo, es importante que las interfaces visual y auditiva se presenten de la forma más aproximada posible a como las encontraría el usuario en un escenario real.

Los avances en la parte visual han tenido especial desarrollo en los últimos 20 años, actualmente es posible crear ambientes fácilmente asimilados como reales por parte del usuario. Sin embargo el desarrollo en la parte auditiva se ha visto limitado debido a la gran variedad de sistemas para la reproducción de sonido con que cuenta el público en general, complicando de esta manera la implementación de un único modelo para generar realismo auditivo. Es así como ha surgido una variedad de sistemas cuyo nivel de calidad varía en proporción a su precio.

En el presente trabajo de grado se implementaron dos modelos de sonido binaural que permitieron generar sonido tridimensional para luego ser reproducido a través audífonos convencionales. De igual forma se creó un ambiente gráfico tridimensional dentro del cual el usuario puede cambiar de

posición en dos dimensiones, esta interacción con el sistema determina tanto la ubicación en la interfaz gráfica como el sonido a reproducir.

El sistema se desarrolló bajo la plataforma de programación C++ con la ayuda de librerías para el manejo gráfico de OpenGL, la API (Application Programming Interface) DirectSound que se encuentra contenida en DirectX, el programa de modelado 3D Max Studio y el programa para el manejo de archivos 3D Exploration. Esta selección de programas se hizo con base en criterios de velocidad de procesamiento y disponibilidad de herramientas para facilitar el desarrollo del proyecto.

En la actualidad existen alrededor del mundo trabajos en el campo del sonido binaural, sobre todo como evolución de los ampliamente difundidos sistemas "*Surround*". El sonido binaural permite al usuario determinar la posición y dirección exacta de un sonido y por lo tanto se percibe como un gran avance para introducir mejoras significativas en el realismo de imágenes de películas, en juegos de computadora e incluso pistas de canciones comerciales.

En Colombia, y más específicamente al interior de la Universidad Javeriana, es posible encontrar un aporte importante en el trabajo con modelos de espacialización de sonido. El trabajo de investigación "Generación de Sonido Binaural a partir de una entrada monofónica"[1] exploró la generación de sonido binaural, obteniendo grabaciones que fueron presentadas a los usuarios para la posterior evaluación de los resultados en la percepción. En el presente trabajo se propuso generar el sonido en tiempo real e involucrar más sentidos, como un posible desarrollo para la mejora en la espacialización del sonido.

Se considera que se realizaron aportes en este campo mediante la evaluación de los resultados obtenidos al utilizar dos modelos de sonido binaural, en

[1] HURTADO L., Jairo A. Generación de sonido binaural a partir de una entrada monofónica. Investigación de Maestría. Pontificia Universidad Javeriana. Bogotá D.C. 2002.

combinación con el uso de una interfaz gráfica. Quedan por lo tanto abiertas nuevas posibilidades para la investigación en este campo y para la implementación de aplicaciones utilizando los modelos de sonido evaluados a lo largo del trabajo.

1. MARCO TEÓRICO

La audición es un proceso complejo que involucra principalmente el trabajo conjunto del cerebro y el oído, es gracias a esta interacción que el ser humano puede identificar y ubicar diferentes sonidos que se encuentran en su entorno y que llegan a él gracias a las propiedades del medio que lo rodea [23].

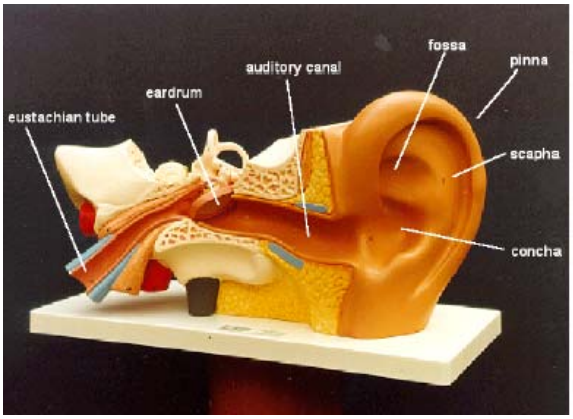


Figura 1. Modelo del sistema auditivo humano[2].

Cuando un sonido incide en el oído, una parte de la energía es reflejada mientras que otra parte es absorbida y pasa al interior del oído. El interior del oído está compuesto por múltiples cavidades que resuenan a diferentes frecuencias, es por esto que el sonido que arriba a la oreja es modificado y puesto a punto para que los órganos más internos se encarguen de traducirlo a señales que puedan ser interpretadas por el cerebro.

[2] Tomado de SIBBALD, Alastair. An introduction To Sound And Hearing. Sensaura. 3D Positional Audio.

Ha sido el interés de varios investigadores modelar el proceso auditivo y utilizarlo en diversas aplicaciones, especialmente para la creación y simulación de sonidos que puedan ser interpretados por el oyente como reales.

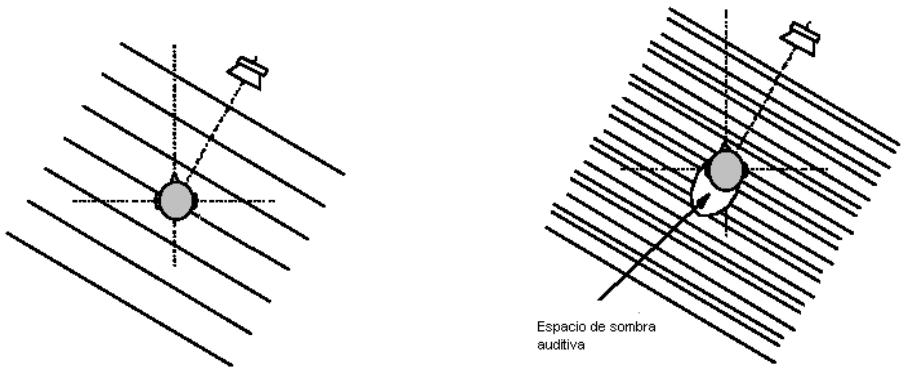
Existen señales primarias que hacen posible la habilidad para ubicar sonidos en un contexto tridimensional. Las señales de sonido primarias incluyen la diferencia de amplitud interaural (Inter-Aural Amplitud Difference IAD), el retardo de tiempo interaural (Inter-Aural Time Delay) y la modificación espectral implementada por el oído externo.

Además de las señales de sonido primarias, existen factores secundarios que contribuyen al la capacidad de localización, entre ellas encontramos las reflexiones en los hombros y el torso, las reflexiones de los objetos del entorno (paredes, muebles, etc.) así como algunas consideraciones sicológicas. Es importante destacar estas últimas ya que predisponen al individuo en el proceso auditivo, por ejemplo, si se escucha un helicóptero se tiende a pensar que se encuentra volando en el aire y no en tierra, asimismo si se escucha ladrar un perro se esperaría que estuviera debajo de nosotros y no encima. Las indicaciones visuales también soportan los factores auditivos, cuando el cerebro combina las imágenes visuales con las imágenes acústicas nos provee con una percepción tridimensional de nuestros alrededores.

1.1. SEÑALES PRIMARIAS.

Las orejas están situadas simétricamente en los lados de la cabeza, la cual tiene un ancho aproximado de 16 cm. Cuando una fuente de sonido se encuentra directamente en frente del sujeto, ambas orejas están expuestas al sonido de la misma forma. Sin embargo cuando la fuente de sonido se mueve hacia un lado de

la cabeza, una oreja quedará más expuesta al sonido que la otra, esta última queda dentro del espacio de sombra que genera la cabeza; este efecto, que se ilustra en la Figura 2, genera diferencias en las amplitudes de las señales de sonido que arriban a cada oreja desde la fuente.



a. b.

Figura 2: a) Difracción para longitud de onda mayor al tamaño de la cabeza
b) Difracción para longitud de onda menor al tamaño de la cabeza[3]

Los efectos de difracción alrededor de la cabeza complican los efectos de sombra. Cuando una onda de sonido encuentra barreras, salen a relucir las propiedades de onda de la energía: el sonido se difracta alrededor del obstáculo de una forma análoga a como la luz se difracta alrededor de una rendija. Como regla general, las ondas de sonido pueden difractarse eficientemente alrededor de los objetos cuando su longitud de onda es significativamente mayor que el tamaño del objeto, mientras que cuando el tamaño del mismo es mucho mayor que la longitud de onda, los efectos de difracción son mínimos. En general los efectos de difracción son considerables principalmente en frecuencias entre 700 Hz y 8 kHz.

[3] Tomado de: SIBBALD, Alastair. Virtual Ear Technology. Sensaura. 3D Positional Audio.

1.2. ITD e IAD

El retardo interaural (ITD) se refiere a la diferencia entre el tiempo de llegada de las señales a los oídos izquierdo y derecho. A menos que la fuente de sonido esté en uno de los polos (directamente en frente, detrás, arriba o abajo) el frente de onda llegará a los oídos en tiempos diferentes. Por ejemplo si la fuente se encuentra exactamente en el lado derecho de sujeto, el sonido deberá viajar una distancia mayor para alcanzar el oído izquierdo de la que debe viajar para alcanzar el sonido derecho.

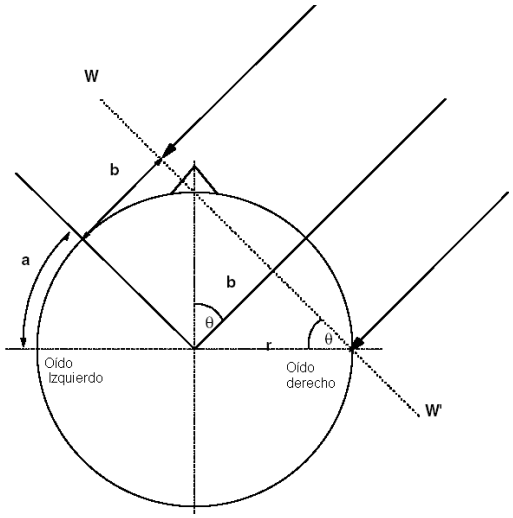


Figura 3: Diagrama de cabeza esférica con desfase interaural.

En la figura 3 se muestra una vista plana de una cabeza conceptual con los oídos izquierdo y derecho recibiendo una señal de sonido desde una fuente con determinado ángulo de azimut θ (aproximadamente 45º). Cuando el frente de onda llega al oído derecho, puede verse que existe una distancia (**a+b**) que debe recorrer antes de encontrar el oído izquierdo. Por la simetría de la configuración, **b** es igual a la distancia desde el centro de la cabeza hasta el frete de onda **W** a **W'** y por lo tanto:

$$b = r * \text{sen}(\theta) \qquad (1)$$

También del diagrama se desprende que **a** representa el arco de la circunferencia subtendido por el ángulo θ, por lo tanto la distancia **a + b** es igual a:

$$a + b = \left(\frac{\theta}{360}\right) 2\pi r + r\,sen(\theta) \qquad (2)$$

Puede verse que en el límite cuando θ tiende a cero, la longitud del camino es igual a 0. También, cuando θ tiende a 90º la longitud de **a+b** es en promedio 19.3 cm. y el retardo asociado es aproximadamente 560 μs. En la práctica, el ITD ha sido medido y ha resultado un poco más grande, posiblemente debido a la naturaleza no esférica de la cabeza, a complejas situaciones de difracción y a efectos producidos por las superficies [23].

Cabe mencionar igualmente que la respuesta espectral del oído externo modifica el sonido que llega al oído interno. Las complejas formas que pueden verse en la oreja resuenan a diferentes frecuencias dependiendo de la dirección de donde viene el sonido, modificando de esta forma el espectro de las señales de sonido antes de que alcancen la membrana del tímpano. El cerebro analiza esta información de ambos oídos como parte del proceso de localización, en conjunto con el IAD y el ITD.

1.3. HRTF

Es posible incorporar factores de sonido 3D dentro de una grabación. La forma más simple es crear una cabeza artificial e introducir en ella dos micrófonos. La cabeza artificial es una estructura que tiene las mismas dimensiones que una

cabeza promedio y cuenta con dos orejas de goma, cada una de las cuales contiene un micrófono en la posición donde se encontraría el canal auditivo. Cuando se hacen grabaciones con tales micrófonos, las ondas de sonido atraviesan la mayoría de modificaciones acústicas que sufrirían en un ambiente real. Por lo tanto, la mayoría de los factores de sonido 3D se incorporan en la grabación. En la figura 4 se muestra la respuesta en frecuencia para dos ubicaciones diferentes de la fuente de sonido, son estos efectos los que influyen cuando se quiere trabajar con la respuesta impulso del sistema cabeza – oreja – oido.

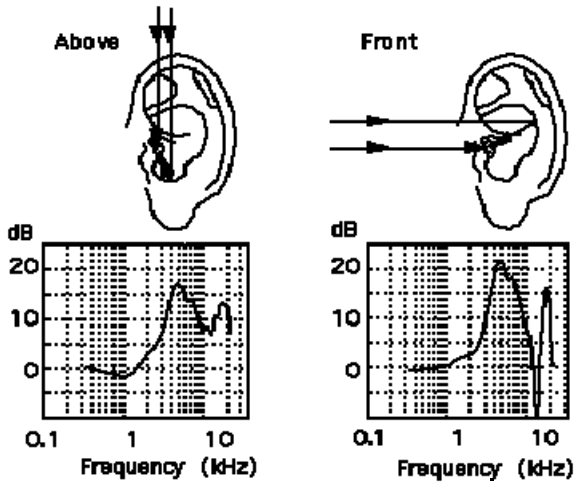


Figura 4. Respuesta en frecuencia para diferente ubicación de la fuente de sonido[4]

Esta tecnología puede ser llevada más allá, midiendo las características de una cabeza artifical (La Head-Related Transfer Function - HRTF), así, es posible sintetizar electrónicamente lo que la cabeza y oídos hacen acústicamente. Esto quiere decir que una grabación de sonido monofónica puede ser procesada para que parezca estar en el lugar del espacio tridimensional deseado.

[4] Disponible desde Internet <URL:http://interface.cipic.ucdavis.edu/CIL_tutorial/3D_psych/elev.htm>

Cabe señalar que la HRTF es función de la posición en donde se encuentra el sonido (relativa al usuario en un marco de referencia tridimensional), además es diferente para cada uno de los oídos ya que las características de recepción y la diferencia espacial existente entre cada uno de ellos así lo exigen.

Para el manejo de este modelo, se utilizaron las respuestas impulso contenidas en la base de datos de la HRTF del MIT[15] que se encuentra en la Internet y aparece como de libre acceso. Las mediciones en esta base de datos fueron realizadas en una cámara anecoica[5], utilizando un muñeco KEMAR (Knowles Electronics Mannequin for Acoustical Research- Maniquí de Knowles Electronics para Investigación Acústica)[13], localizado a 1,4 metros de la fuente sonora, obteniendo 512 muestras de 16 bits de la respuesta impulso, a una frecuencia de muestreo de 44,1 kHz. El maniquí tenía dos formas de orejas diferentes para obtener dos bases de datos diferentes. Un ejemplo de estas muestras puede verse en la Figura 5.

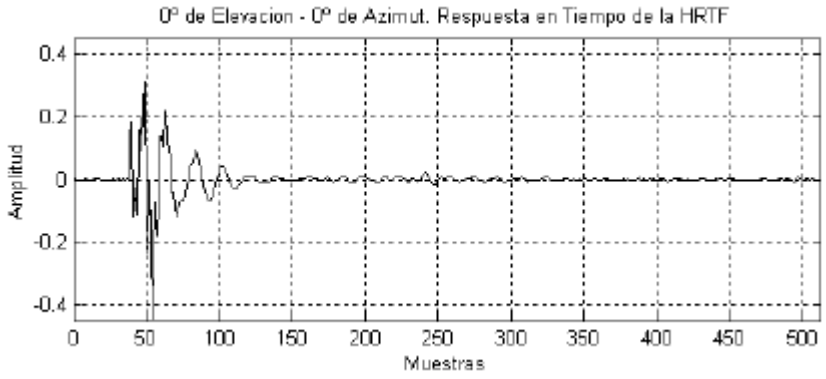


Figura 5. Ejemplo de Respuesta impulso para el oído del muñeco KEMAR en la posición 0º de elevación y 0º de Azimut. [6]

[5] Cámara al interior de la cual las ondas de sonido no tienen reflexiones apreciables y por lo tanto el efecto de eco es reducido al mínimo.

[6] Tomado de: HURTADO L., Jairo A. Generación de sonido binaural a partir de una entrada monofónica. Investigación de Maestría. Pontificia Universidad Javeriana. Bogotá D.C. 2002

1.4. INTERFAZ GRÁFICA

Dado que uno de los factores secundarios más importantes en el proceso de localización de sonidos es la interacción con las imágenes suministradas por el sentido de la vista, se decidió integrar un sistema de imagen y sonido tridimensional.

Para el desarrollo de la interfaz gráfica se utilizó C++ en conjunto con las librerías suministradas por el API (Application Programming Interface). El API es una herramienta creada para facilitar el trabajo de los programadores al interactuar con los dispositivos periféricos por ejemplo: la tarjeta de video y el monitor. Al utilizar este tipo de herramientas, el programador se libra del trabajo que resulta programar dispositivos a bajo nivel. Para este trabajo de grado se seleccionó el API OpenGL que es un estándar de industria y por lo tanto asegura portabilidad y simplicidad en el desarrollo.

Dada la importancia de la interacción entre el usuario y el ambiente gráfico, es importante trabajar no sólo en su concepción sino en la forma de mostrarlo al usuario. La API seleccionada ofrece herramientas para el manejo de cámaras y luces así como para el manejo de eventos provenientes de periféricos de entrada. Cabe mencionar que se guarda un vector con la posición actual de la cámara, la cual resulta útil para implementar los modelos de sonido descritos anteriormente.

1.5. PROGRAMACIÓN ORIENTADA POR OBJETOS

Después de la programación funcional, la programación orientada por objetos ha surgido como una opción para los programadores que aumenta el nivel al que se realiza la programación. Visual C++ 6.0 es un programa especialmente diseñado para facilitar este tipo de programación.

La programación orientada por objetos propone el encapsulamiento de datos y la creación de métodos para su consulta y modificación. Al encapsular los datos al interior de este tipo de estructuras se restringe el acceso y la modificación de los datos, esto permite que los datos sean modificados únicamente cuando el usuario quiere modificarlos y que no sean variables de fácil cambio en situaciones no previstas por el programador.

Así es como el programador puede construir una estructura modular compuesta por clases, cada una con características y responsabilidades muy específicas para luego integrar su funcionamiento en un programa principal.

Las clases, como se dijo anteriormente están compuestas por atributos y métodos. Los atributos son variables de cualquier tipo que son accesibles por las funciones (métodos) de la clase mientras que permanecen inaccesibles para funciones externas. Los métodos son funciones que implementan algoritmos para el cumplir con los objetivos de la clase, estas funciones pueden definirse como públicas o privadas, esto es, que solo puedan ser llamadas por otros métodos de la clase o que las demás clases también tengan acceso a ellas. Entre los métodos más comunes implementados dentro de las clases están los consultores y los modificadores, los cuales se encargan de permitir al programador tener un acceso indirecto a los atributos de la clase.

Además de las ventajas anteriormente citadas, la programación tiene algunas otras ventajas como la facilidad para la reutilización de código y la posibilidad de implementar herencia, esto es, la creación de clases (subclases) a partir de clases ya existentes (superclases) conservando los atributos y métodos de las mismas y agregando nueva funcionalidad.

Se considera que la programación orientada por objetos es apropiada para la aplicación que se busca realizar debido a la fácilidad para llevar a cabo la

estructura de tipo modular que se pretende alcanzar. Por otra parte, la API que se utilizó para la implementación del sonido se ubica también dentro del paradigma de la programación orientada por objetos y por lo tanto se enmarca de forma precisa en el esquema que se busca seguir. Cabe agregar que en DirectSound se encuentran diversas clases que pueden ser reutilizadas y adaptadas para conseguir el comportamiento deseado.

2. ESPECIFICACIONES

Para lograr los objetivos propuestos se desarrolló el sistema que se ilustra en la Figura 6. El sistema desarrollado está compuesto por un programa principal (Ver Anexo 1: Diagrama De Flujo Del Programa) que integra y permite la interacción entre dos módulos, uno encargado de la generación del audio y otro encargado de la generación del ambiente gráfico. Cabe anotar que estos módulos fueron implementados en forma de clases, con responsabilidades bien definidas en cuanto a la presentación gráfica o a la reproducción sonora. Las órdenes del usuario son recibidas con la ayuda de un dispositivo periférico e introducidas al programa para generar el conjunto sincronizado de variables de salida (sonido e imagen), las cuales son presentadas al usuario a través de dos dispositivos periféricos de salida.

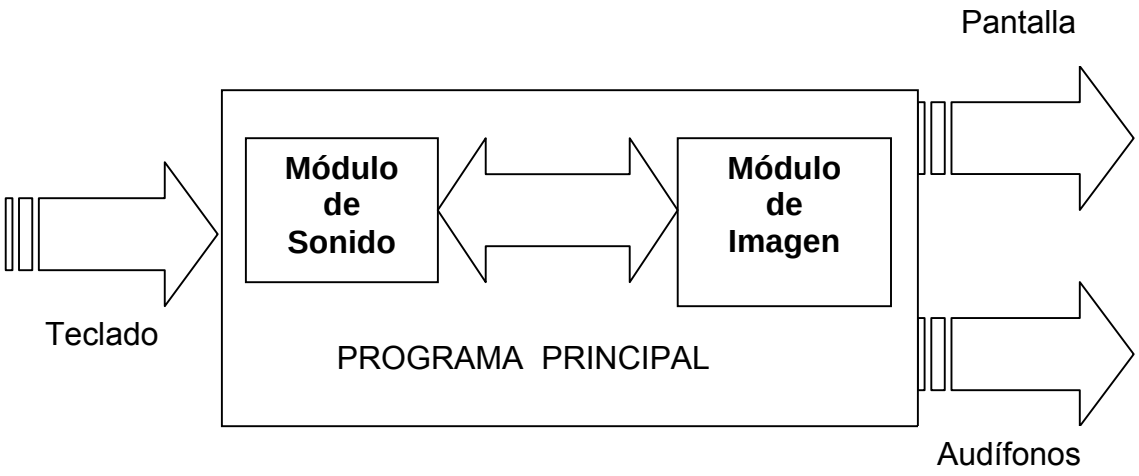


Figura 6. Diagrama en bloques

2.1. MÓDULO DE IMAGEN

Este módulo permite la visualización de un ambiente tridimensional, en donde el usuario tiene la percepción de estar inmerso y dentro del cual puede desplazarse libremente y a voluntad en dos dimensiones (plano azimutal).

El ambiente gráfico se modeló en 3D Max Studio, el cual brindó un gran número de herramientas y menús útiles para este fin. Luego de modelar en 3D Max Studio, se utilizó el programa 3D Exploration, a través del cual se exportó el ambiente desarrollado a un archivo plano (.obj), y por último se implementó en C++ un algoritmo que realiza la tarea de reconocer y volver a dibujar los diferentes vértices y texturas contenidos en el archivo plano teniendo como base para su diseño las librerías de la API OpenGL.

Los criterios para generar tal esquema fueron orientados principalmente hacia la objetividad a la hora de evaluar los resultados en cuanto a percepción de los diferentes usuarios que probaron el sistema. En la Figura 7, se muestra la vista superior del ambiente gráfico, en la cual se puede apreciar la distribución del mismo. La fuente de sonido podrá ser cualquiera de los patos que se muestran en la figura y corresponderá al usuario llegar hasta ella. Cabe anotar que los caminos no están separados por paredes sino por espacios vacíos para evitar el alto número de reflexiones que se podrían ocasionar si existieran tales paredes.

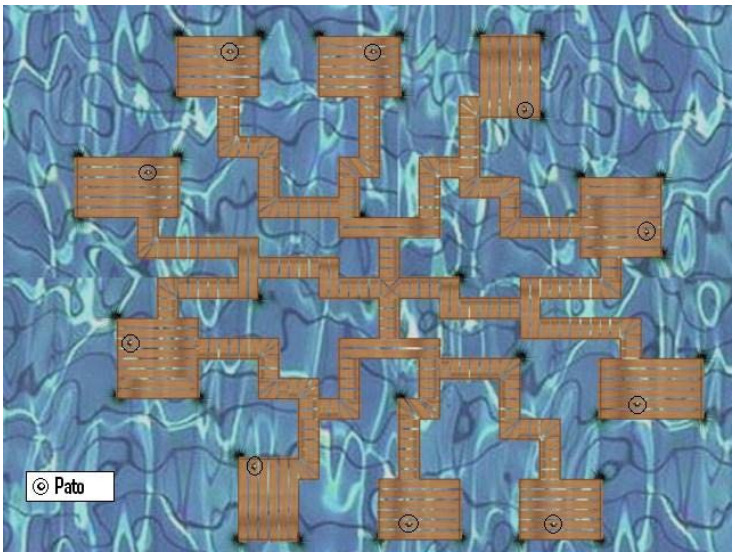


Figura 7: Ambiente Gráfico (Vista Superior)

El usuario está representado por la posición de la cámara, la cual presenta una vista perspectiva[7] de lo que ve un sujeto ubicado en el mismo lugar.

2.2. MÓDULO DE SONIDO

Este módulo es el encargado de generar el sonido para los canales derecho e izquierdo con base en la posición actual del usuario dentro del ambiente gráfico, en la posición de la fuente de sonido y en el modelo de sonido binaural en uso. Se puede seleccionar entre dos de los modelos que fueron desarrollados: el modelo IAD e ITD y el modelo HRTF (Head Related transfer function). Cada uno de los modelos es implementado independientemente sobre la misma interfaz gráfica con el fin de evaluar su desempeño de una forma más objetiva.

En el caso del modelo IAD e ITD el algoritmo hace uso de las ecuaciones para el cálculo de la diferencia en tiempo de llegada que se muestran en el Capítulo 1.

[7] Especificación de los objetos combinada con las especificaciones del observador mediante la cual una API presenta en el monitor una imagen que representa una vista tridimensional con un campo visual similar al de un ser humano.

Marco Teórico. Para el caso del modelo HRTF, el algoritmo hace uso asimismo de la base de datos que contiene las respuestas impulso medidas por el MIT sobre el muñeco KEMAR junto con las ecuaciones del Capitulo 1: Marco Teórico.

2.3. PROGRAMA PRINCIPAL

Los módulos se encuentran inmersos dentro del programa principal, este programa es el encargado de recibir las entradas por parte del usuario y generar las salidas. Para la generación de las salidas de imagen y audio se utilizan los respectivos módulos anteriormente mencionados. La plataforma de desarrollo fue una aplicación de consola WIN32 trabajada en Visual C++ 6.0.

Se manejaron al interior del programa las siguientes entradas y salidas:

2.3.1. Entradas

Para la interacción con el usuario, se utiliza como periférico de entrada el teclado del computador en donde se corre la aplicación. El usuario puede moverse con la ayuda de las cuatro teclas de flecha que se encuentran en un teclado estándar. En caso de que el usuario desee reproducir el sonido, conservando su posición y orientación actual puede hacer uso de la tecla “TAB”

2.3.2. Salidas

Una vez las entradas del usuario son procesadas, el programa genera dos salidas, una imagen y un sonido, los cuales son presentados al usuario a través de dos dispositivos de salida diferentes. Para presentar la salida de imagen se utiliza la pantalla del computador en el cual se esté corriendo el programa y para presentar

la salida de sonido se utilizan audífonos estándar de diadema rígidos para cancelación de ruido.

3. DESARROLLO

A continuación se presentan los pasos que se siguieron para obtener el sistema descrito anteriormente

3.1. MÓDULO DE IMAGEN

El módulo de imagen es una parte muy importante del trabajo desarrollado debido a que gracias a él los usuarios pueden interactuar con el sistema y ubicarse espacialmente. Para recrear un ambiente gráfico se procedió en primera instancia a utilizar las herramientas integradas en la librería GLUT (OpenGL Utility Toolkit) para crear formas básicas como cubos, esferas, prismas, etc. Sin embargo, después de cierto tiempo se encontró que este método de trabajo no estaba dando buenos resultados debido a la dificultad que implica crear ambientes complejos y creíbles a partir de formas básicas. Es por ello que se decidió optar por un modelado con ayuda de herramientas existentes que facilitan tal fin y posteriormente importar estos modelos desde visual C++ 6.0.

3.1.1. Modelado del ambiente gráfico.

El modelado fue realizado en el programa 3D Max Studio 6.0, tratando al máximo de usar figuras y sólidos de geometría primitiva, pocos materiales que utilicen texturas y en los casos que fueron necesarias, se trabajaron texturas de baja resolución. Todo esto se hizo buscando obtener un ambiente con buena apariencia y de tamaño moderado que facilite una implementación eficiente en C++. En las figuras 8, 9 y 10 se encuentran unas imágenes del laberinto modelado exportadas desde 3D Max Studio.

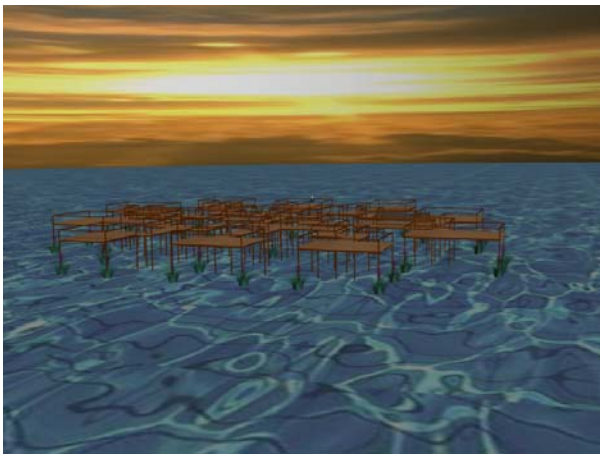


Figura 8: Perspectiva Total del Laberinto Modelado

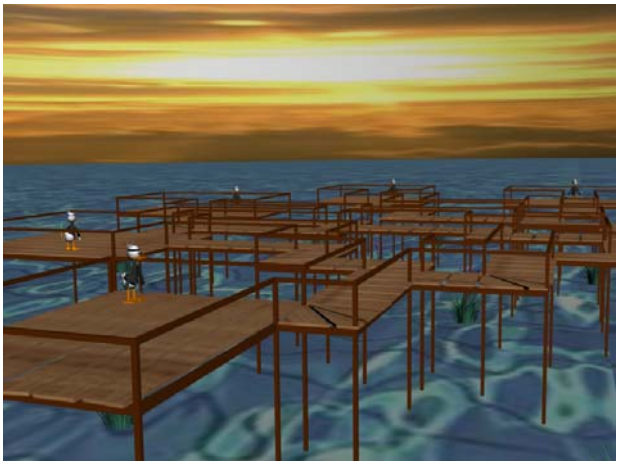


Figura 9: Perspectiva del Laberinto Modelado Implementado con Varios Patos

Figura 10: Acercamiento al Pato Fuente de Sonido

3.1.2. Carga del ambiente gráfico desde Visual C++ 6.0.

Después de terminar con la etapa de modelado se utilizó un programa llamado 3D Exploration, cuya función fue tomar el archivo generado en 3D Studio y exportarlo a dos archivos de texto plano (archivo .obj y archivo .mtl).

El trabajo que realiza el programa 3D Exploration consiste en tomar el ambiente gráfico que se encuentra en el archivo 3DS y expresarlo como una lista de triángulos, cada uno con una ubicación espacial y con unas propiedades espécificas que describen su comportamiento con la luz (color, coeficiente de reflexión ambiental, coeficiente de reflexión difusa, coeficiente de reflexión especular, coeficiente de brillo). Para poder utilizar el ambiente al interior de la plataforma de programación utilizada fue necesario crear una clase que se encargara de realizar la carga y la clasificacion de la información contenida en los archivos. La clase creada para tal fin se denominó Cargar01, en la Tabla 1 se presenta una descripción donde se incluyen los beneficios que se esperaban obtener con su implementación.

Clase	Cargar01
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Leer el archivo de vértices y el de materiales (.obj y .mtl)
2	Interpretar los archivos y almacenar arreglos con la información.
3	Permitir la consulta de la información por otras clases a través de consultores.

Tabla 1. Descripción de la Clase Cargar01

Para cumplir con estas responsabilidades se crearon al interior de la clase una serie de atributos y métodos encaminados a la adecuada carga y organización de la información.

En primer lugar se implementó el método leer, el cual tenía la función de copiar cada uno de los archivos en un arreglo de caracteres, estos arreglos son atributos de la clase y pueden encontrarse en el código con los nombres “buffer” y “buffer2”. Seguidamente se creó el método “contar” cuya función consistió en contar el número de triángulos, vértices, normales y coordenadas de textura que se encuentran en el archivo OBJ, así como los tipos de materiales descritos en el archivo MTL. De igual forma, era responsabilidad de este método crear arreglos de estructuras del tamaño respectivo.

Una vez creados los arreglos con los diversos tipos de estructuras necesarios, se procedió a rellenar tales estructuras con la información contenida en los respectivos archivos, para tal fin se crearon múltiples algoritmos auxiliares que también figuran como métodos de la clase y que permiten recorrer y localizar la información disponible en los arreglos ya mencionados. De esta forma se cumplieron a cabalidad las responsabilidades que se esperaban de la clase Cargar01.

3.1.3. Presentación del ambiente gráfico dentro de la aplicación.

Una vez cargado el mapa de vértices, normales y materiales y debidamente almacenado en la memoria, se procede a realizar el render utilizando las utilidades suministradas en las librerías de OpenGL. Todas las funciones encaminadas a tal fin están incluidas en la clase Graficar01 que puede ser consultada en el Anexo 2: “Código Fuente”. Asimismo se presenta en la Tabla 2 un resumen con las responsabilidades y las colaboraciones de tal clase dentro del programa.

Clase	Graficar01
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Iniciar el formato de pixeles para el renderizado.
2	Pintar (Realizar renderizado)
3	Mover la cámara según eventos del usuario.
4	Detectar y evitar colisiones del usuario con objetos circundantes dentro del ambiente.
5	Desactivar OpenGL (Liberar recursos).
6	Entregar posición actual de cámara y fuente de sonido.

	Colaboraciones
1	Hallar ángulo entre la cámara y la fuente de sonido para la reproducción del sonido.
2	Hallar la distancia entre la cámara y la fuente de sonido para la reproducción del sonido (atenuación).

Tabla 2. Descripción de la Clase Graficar01

Son múltiples los métodos y atributos implementados en esta clase, vale la pena resaltar entre los atributos las tres estructuras tipo “punto” que encargadas de almacenar la posición y orientación del usuario así como la posición de la fuente (pueden encontrarse en el código con los nombres eye, at y pato). A continuación se presentan los métodos implementados para cumplir con cada una de las responsabilidades y colaboraciones necesarias descritas como objetivos de la clase.

3.1.3.1. Responsabilidades

En primera instancia, para iniciar las propiedades necesarias con el fin de hacer posible la renderización dentro el área de pintado designada, se creó el método “iniciarOpengl”, allí se obtiene el contexto de dispositivo y se crea y pone activo el contexto de renderizado, utilizando el formato de píxeles descrito en el método “iniciarPixelFormat”.

Una vez que se ha inicializado OpenGL puede llevarse a cabo la tarea más importante de esta clase que consiste en pintar una escena determinada. Debe quedar claro que el ambiente cargado describe hasta el último detalle del ambiente pero es OpenGL quien decide qué parte de este ambiente debe presentarse en la pantalla, basado en la posición y orientación de la cámara. Para pintar la escena, se implementó el método “display”, el cual se encarga de tomar los datos cargados con anterioridad y utilizar las funciones de OpenGL para convertir tales datos en imágenes. Como resultado de este proceso de renderización, surge una imagen como la que se muestra en la figura 11, para más información sobre el proceso de renderización desde el código hasta la presentación en pantalla, véase Angel, Interactive Computer Graphics [1].

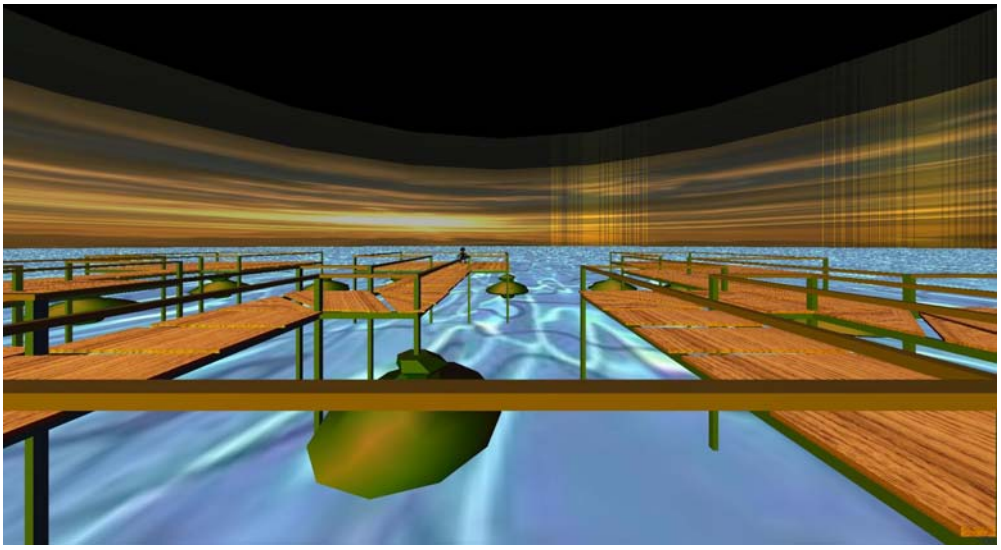


Figura 11. Vista del usuario dentro del ambiente gráfico

Después de presentar y ubicar al usuario en una escena específica, es necesario permitirle desplazarse dentro del ambiente y presentarle imágenes acordes con tal interacción, es por ello que se crearon cuatro métodos que permitieran mover la cámara hacia delantey hacia atrás así como rotarla para de esta forma presentar al usuario imágenes diferentes que coincidieran con sus movimientos. Estos métodos pueden encontrarse con los nombres “movercamaraadelante”, “movercamaraatras”, “movercamaraderecha” y “movercamaraizquierda”.

Sin embargo, al hacer realidad los métodos anteriormente descritos surge el problema de que el usuario no debería tener acceso a ciertos lugares del ambiente, que debería chocar con los objetos del ambiente y que éstos deberían restringir su paso. Para atacar este problema se creó el método “colisiones”, el cual se encarga de comprobar si es posible llevar a cabo el movimiento solicitado por el usuario, en caso de que el movimiento no sea válido no se permite al usuario llevarlo a cabo y se mantiene su posición actual, de lo contrario el movimiento de la cámara es ejecutado.

Finalmente, una vez terminada la aplicación deben eliminarse los objetos utilizados para el renderizado con el fin de devolver al sistema operativo el control de la memoria y los recursos, para ello existe un método denominado “disableOpenGL” que se encarga de la liberación de los recursos utilizados para el renderizado.

3.1.3.2. Colaboraciones

Una vez descrita la forma en que se implementaron las responsabilidades de la clase, se presentan los aspectos más relevantes en cuanto a la forma de poner en práctica sus colaboraciones.

La primera colaboración exigida a la clase es obtener el ángulo entre la cámara y la fuente de sonido. Esta función se facilitó debido a que constantemente se guarda la posición y orientación de la cámara así como la posición de la fuente de sonido. Utilizando estas ventajas se generó el método "anguloconpato" el cual, utilizando relaciones trigonométricas, devuelve un valor de ángulo entre $-\pi$ y π radianes.

Para atacar la segunda colaboración exigida a la clase, consistente en hallar la distancia entre la cámara y la fuente, se implementó el método "distanciaconpato", el cual halla la norma del vector resta entre el vector posición de la fuente de sonido y el vector posición del usuario.

Además de los métodos anteriormente descritos se implementaron algunos otros algoritmos que apoyaban la labor de los arriba citados, estos métodos auxiliares pueden también ser encontrados en el código fuente.

3.2. MÓDULO DE SONIDO

Tal como se mencionó en el marco teórico, existe una diferencia en el tiempo de llegada del sonido a los oídos izquierdo y derecho a menos que la fuente se encuentre en el plano que comprende posiciones que están a igual distancia de ambas orejas.

3.2.1. Implementación del Modelo IAD – ITD.

Gracias a la simetría del modelo de cabeza esférica, el tiempo retardo existente en llegar la señal al oído izquierdo cuando un frente de onda incide sobre el oído derecho con un ángulo determinado, es igual al tiempo de retardo que se experimenta en el oído derecho cuando el frente de onda llega al oído izquierdo

con un ángulo igual medido en sentido contrario, como se puede ver en la Figura 12.

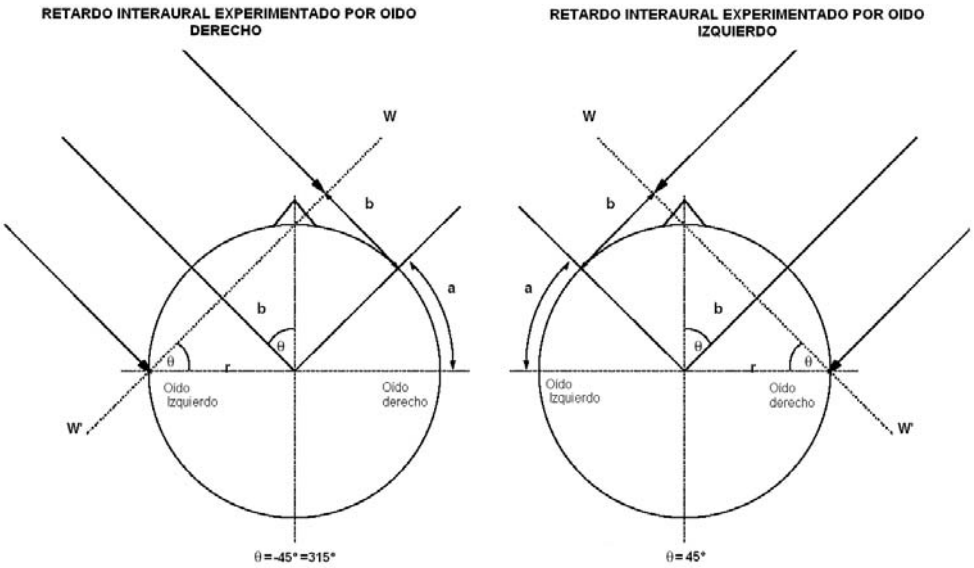


Figura 12: Diagrama de cabezas esféricas con desfase interaural igual experimentado por cada oído afectado.

De lo anterior se dedujo que los retardos aplicados al oído izquierdo cuando el frente de onda se encuentra en la Región D del esquema de la cabeza que se muestra en la Figura 13 son simétricamente iguales a los aplicados al oído derecho cuando el frente de onda se encuentra en la Región C.

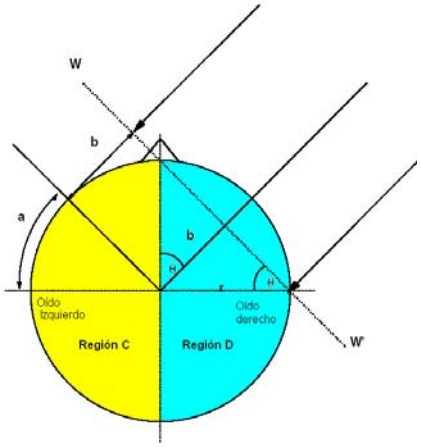


Figura 13: Diagrama de cabeza esférica para simetría entre los lados izquierdo y derecho

También, gracias a este modelo de cabeza esférica, se presenta simetría con los tiempos de retardo experimentados en la Región Frontal y la Región Posterior para ángulos equidistantes tal como se muestra en la Figura 14.

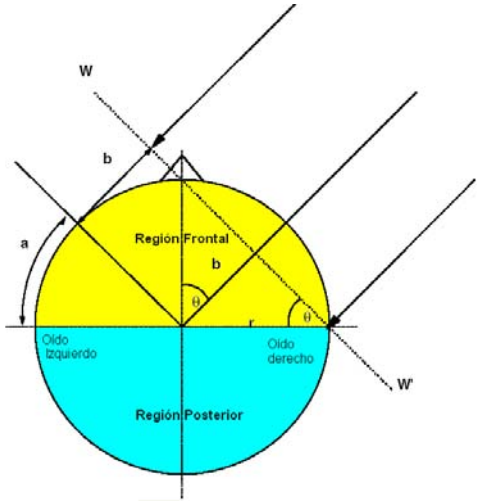


Figura 14: Diagrama de cabeza esférica para simetría entre región frontal y región posterior

Debido a la simetría descrita, para implementar este modelo se simplifican los cálculos y solo es necesario hallar 6 valores de retardos que son aplicados al inicio o al final de dos archivos monofónicos iguales, al inicio son aplicados al archivo correspondiente al oído que experimenta el retardo y al final son aplicados al archivo correspondiente al oído contrario, para luego juntar estos dos archivos en un nuevo archivo estéreo que permite tener la sensación de oír el sonido reproducido en un lugar determinado del espacio.

Los tiempos de retardo fueron calculados para los ángulos θ=15º,30º,45º,60º,75º y 90º y luego estos datos fueron utilizados para los demás ángulos.

Primero se calculó la distancia a+b para cada uno de los ángulos anteriormente mencionados:

$$a+b \quad = \quad \left(\frac{\theta}{360}\right)2\pi r \quad + \quad r\,sen(\theta) \qquad (3)$$

$$Donde: r \approx 8.125 cms$$

Aplicando la Ecuación 3 se calcularon los valores que se registran en la Tabla 3.

Ángulo	a+b[cms]
15º	4.23
30º	8.32
45º	12.13
60º	15.55
75º	18.48
90º	20.88

Tabla 3. Valores de distancia adicional recorrido por el frente de onda

Luego se procedió a hallar los tiempos de retardo entre los dos oídos.

$$t_d = \frac{(a+b)}{V_{SONIDO}} \qquad (4)$$

Al aplicar la fórmula que se muestra en la Ecuación 4 se obtuvieron los retardos (en microsegundos) que se muestran en la Tabla 4.

Ángulo	t_d[µseg]
15º	123.33
30º	239.94
45º	353.64
60º	453.35
75º	538.77
90º	608.74

Tabla 4. Valores de tiempo de retardo experimentados por el frente de onda entre los dos oídos

Después hallar los tiempos de retardo entre ambos oídos, se procedió a encontrar el número de muestras correspondientes a los retardos de cada ángulo. El archivo monofónico estaba grabado a una frecuencia de 22 kHz entonces:

$$\#\,Muestras = \frac{22000 Muestras * t_d}{1seg} \qquad (5)$$

Aplicando esta fórmula, se encuentra la cantidad de muestras de retardo correspondientes a cada ángulo que se presentan en la Tabla 5

Ángulo	# Muestras de Retardo	# Muestras Trabajadas
15º	2.71	3
30º	5.27	6
45º	7.78	8
60º	9.97	10
75º	11.85	12
90º	13.39	13

Tabla 5. Número de muestras de retardo para los respectivos valores de ángulo

En la tabla número 6 se encuentran las muestras de retardo correspondientes a cada oído, para la totalidad de los ángulos trabajados por el sistema.

# Muestras de Retardo	Retardos Sobre Oído Izquierdo	Retardos Sobre Oído Derecho
3	15º y 165º	195º y 345º
6	30º y 150º	210º y 330º
8	45º y 135º	225º y 315º
10	60º y 120º	240º y 300º
12	75º y 105º	255º y 285º
13	90º	270º

Tabla 6. Número de muestras de retardo para la totalidad de los ángulos

Para manejar la atenuación con la distancia, se hizo uso de la función SetVolume de DirectSound que modifica el nivel de atenuación del volumen de reproducción

del buffer, en centésimas de decibel. La atenuación se implementó como una función que aumenta con el cuadrado de la distancia, tal como se muestra en la ecuación número.

$$Atenuación\left[dB\right] = 20 * \log(dis\tan cia) \qquad (6)$$

3.2.2. Implementación del Modelo HRTF

Este modelo fue desarrollado usando el mismo archivo monofónico de sonido utilizado en el modelo anterior. Se trabajó con la base de datos que contiene las respuestas impulso medidas por el MIT para la oreja derecha del muñeco KEMAR. Estas respuestas impulso fueron convolucionadas con el archivo de sonido monofónico directamente para los distintos ángulos correspondientes al oído derecho, sin embargo para el oído izquierdo hubo la necesidad de encontrar un ángulo equivalente basados en las propiedades esféricas de la cabeza, debido a que no se contaba con una base de datos de respuestas impulso para oído izquierdo con un tipo de oreja igual a la derecha. Esto es, se contaba tan solo con la base de datos para el oído derecho y a partir de ellas se extrapolaron las del oído izquierdo.

En la Tabla 7 se presenta el ángulo equivalente en cuanto a respuesta impulso del oído izquierdo con respecto al oído derecho.

Oido Derecho	0º	15º	30º	45º	60º	75º
Oido Izquierdo	0º	345º	330º	315º	300º	285º

Oido Derecho	90º	105º	120º	135º	150º	165º
Oido Izquierdo	270º	255º	240º	225º	210º	195º

Oido Derecho	180º	195º	210º	225º	240º	255º
Oido Izquierdo	180º	165º	150º	135º	120º	105º

Oido Derecho	270º	285º	300º	315º	330º	345º
Oido Izquierdo	90º	75º	60º	45º	30º	15º

Tabla 7. Ángulo con respuesta impulso equivalente del oído izquierdo respecto al derecho

Luego de realizar las convoluciones para ambos oídos, se unieron los dos archivos monofónicos en un nuevo archivo estéreo que permite tener la sensación de oír el sonido reproducido en un lugar determinado del espacio.

En este modelo también se trabajó el retardo interaural entre los oídos y la atenuación con la distancia, descritos en el modelo anterior.

3.2.3. Reproducción del sonido dentro de la aplicación

Una vez trabajada la fuente monofónica y obtenida la base de datos del sonido con cada modelo para los diferentes ángulos, se procedió a implementar una serie de funciones agrupadas en una clase dedicadas a la escogencia del sonido.

En la Tabla 8 se presenta la descripción de la clase Sonar02, encargada de la escogencia del archivo y de la reproducción del mismo. De igual forma, en el Anexo 2: "Código Fuente" puede encontrarse la implementación de la clase.

Para cumplir con las responsabilidades de la clase es necesario, en primer lugar, inicializar los parámetros necesarios para asegurar el adecuado funcionamiento de DirectSound. Esta inicialización fue implementada con el método OnInit, el cual se utiliza al comienzo del programa principal.

Clase	Sonar02
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Encargarse del manejo de DirectSound (objeto y buffer)
2	Cargar el archivo de sonido
3	Reproducir el sonido con base en posición de la cámara y de la fuente.
4	Calcular y efectuar atenuación del sonido (volumen) de acuerdo a la distancia entre fuente y cámara.

Tabla 8. Descripción de la Clase Sonar02

El algoritmo desarrollado para la escogencia y reproducción del sonido toma el ángulo y la distancia entre el usuario y la fuente de sonido obtenidos en la clase Graficar01 para producir un sonido particular. Con base en el parámetro ángulo se escoge un archivo de sonido a reproducir, por ejemplo, si se encuentra que el ángulo entre el usuario y la fuente es igual a 28 grados, se toma el archivo obtenido después de tomar la fuente monofónica y pasarlo a través del modelo en uso para un ángulo igual a 30 grados, esta función es incorporada en la clase con el método CargarArchivo. Este método es utilizado cada vez que el usuario cambia su posición dentro del ambiente y es acompañado por el método CargarDirectSound el cual deja el archivo seleccionado listo para su reproducción ajustando los parámetros de Directsound. Finalmente, utilizando el parámetro distancia se aplica la atenuación correspondiente utilizando el método OnSound, este método también se ocupa de la reproducción del sonido hacia los audífonos.

3.3. PROGRAMA PRINCIPAL

El programa principal, que tiene a su cargo el manejo de la aplicación se encarga de iniciar un objeto de tipo Cargar01 con el fin de leer el archivo, posteriormente instancia un objeto de tipo Graficar01 para iniciar el ciclo de dibujado y finalmente se queda esperando un evento de teclado que lo obligue a volver a dibujar el ambiente en la pantalla, y a reproducir un sonido en concordancia con una nueva posición del usuario. Asimismo, es ella quien se encarga de dar inicio y fin a la ejecución de la aplicación, caso en el cual se encarga de eliminar los objetos y los buffers creados para de esta forma liberar la memoria utilizada.

3.4. OTRAS UTILIDADES

Además de las clases utilizadas para cargar los archivos, graficar el ambiente y reproducir el sonido, se implementaron algunas clases cuyo objetivo consistió en almacenar información de una forma estructurada para permitir a las otras clases un manejo más robusto y organizado.

La primera de estas clases es “Punto” cuyo principal objetivo fue permitir la creación de objetos con los tres vértices de un triángulo. Cabe recordar que el archivo suministrado por el 3D Exploration consiste en un archivo plano de vértices, normales, materiales y texturas correspondientes a triángulos que posteriormente deben ser dibujados. En la tabla 9 se presenta una descripción de esta clase.

Clase	Punto
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Guardar las tres coordenadas de un punto en el espacio 3D.
2	Permitir a usuarios externos (otras clases) su creación, destrucción, consulta y modificación

Tabla 9. Descripción de la Clase Punto

Otra clase implementada es “Normal”, estructura de datos que, al igual que “Punto” almacena tres valores, pero en este caso los valores corresponden a la normal de un triángulo en un vértice. Esta información es importante para el manejo de materiales y texturas ya que determina en buena medida la interacción del triángulo con los rayos de luz incidentes sobre el mismo (la forma en que son reflejados y refractados). En la Tabla 10 se presenta una descripción de la clase.

Clase	Normal
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Guardar las tres coordenadas de un vector normal a una superficie e un punto.
2	Permitir a usuarios externos (otras clases) su creación, destrucción, consulta y modificación

Tabla 10. Descripción de la Clase Normal

De igual forma, se implementó una clase denominada material. Esta clase contiene información sobre las características del material con que debe ser

dibujado un triángulo. Si es necesario aplicar una textura determinada, la clase contendrá el nombre del archivo gráfico que debe aplicarse, de lo contrario tendrá información sobre el color y las características del material con que debe dibujarse. En la Tabla 11 se describe la clase en cuestión.

Clase	Material
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Guardar los parámetros ka, kd, ks, ns e illum que caracterizan un material dentro de OpenGL[8]
2	Permitir a usuarios externos (otras clases) su creación, destrucción, consulta y modificación
3	Devolver el nombre del archivo fuente de una textura asociada a un material.

Tabla 11. Descripción de la Clase Material

Como se dijo anteriormente, existen ciertos triángulos que requieren el uso de texturas para su dibujado; para tales triángulos el 3D exploration adjunta una serie de coordenadas para cada vértice. Estas coordenadas corresponden a un mapa de bits donde se tiene la información de la textura. Para almacenar esta información se diseñó e implementó la clase "Textura", que se describe a continuación en la Tabla 12.

[8] Parámetros que describen la interacción entre la luz y un material, ka: coeficiente de reflexión ambiental, kd: Coeficiente de reflexión difusa, ks: Coeficiente de reflexión especular, ns: Coeficiente de brillo.

Clase	Textura
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Guardar las coordenadas (en 2D) de un archivo BMP al que corresponde un vértice determinado que se encuentra vinculado a una textura.
2	Permitir a usuarios externos (otras clases) su creación, destrucción, consulta y modificación

Tabla 12. Descripción de la Clase Textura

No solo es importante tener almacenada la información de vértices, normales y materiales, lo es también conocer cuál normal y qué material corresponden a un determinado vértice de un triángulo. Por ello se hizo necesario implementar una clase que almacenara información de correspondencia entre vértices, normales, materiales y coordenadas de textura (si aplica) para un determinado vértice. En la Tabla 13 se presenta la descripción de la clase Orden, encargada de estas funciones.

Clase	Orden
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Guardar la correspondencia entre vértices, normales, materiales y coordenadas de textura para cada uno de los triángulos exportados desde 3D Exploration.
2	Permitir a usuarios externos (otras clases) su creación, destrucción y consulta.
3	Devolver el nombre del archivo fuente de una textura asociada a un material.

Tabla 13. Descripción de la Clase Orden

Otra clase que se decidió crear es la clase Objeto, esta clase simplemente almacena la información global del archivo cargado, esto es, cuántos triángulos se cargaron, cuántos puntos, cuántos materiales y cuantas coordenadas de texturas fueron leídos. Cabe aclarar que debido a que existen triángulos que comparten puntos, el número de puntos, de normales y de coordenadas pueden diferir y es por ello que la clase Objeto entra a jugar un papel relevante. En la Tabla 14 se presenta una descripción de esta clase

Clase	Objeto
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Guardar la cantidad de triángulos, vértices, texturas, materiales y normales leídos desde el archivo.
2	Permitir a usuarios externos (otras clases) su creación, destrucción y consulta.

Tabla 14. Descripción de la Clase Objeto

Finalmente, vale la pena mencionar una última clase cuya función consistió en cargar los archivos de mapa de bits utilizados por los la clase Graficar01 cuando era necesario dibujar un triángulo que requería el uso de una textura específica. En la tabla 15 se presenta una breve descripción de esta clase.

Clase	CTextura
Superclases	Ninguna
Subclases	Ninguna

	Responsabilidades
1	Leer, almacenar y habilitar un mapa de bits al cual se hace referencia como una textura

	Colaboraciones
1	Habilitar en OpenGL el mapeo de texturas

Tabla 15. Descripción de la Clase CTextura

4. ANÁLISIS DE RESULTADOS

Dado que dentro del objetivo general del proyecto se planteó la necesidad no solo de implementar sino también de evaluar el comportamiento de los modelos de especialización de sonido, y que uno de los objetivos específicos consistía en comparar los resultados de la percepción por parte de los diferentes usuarios en cada uno de los modelos de sonido implementados, se resolvió plantear un esquema de encuestas enfocado a dilucidar la eficiencia de cada uno de los modelos, probados de forma separada en condiciones ambientales y técnicas similares (ambiente gráfico, mismo computador, mismos audífonos, etc.)

4.1. PROTOCOLO DE PRUEBAS

Para realizar las pruebas se llevó a cabo el siguiente protocolo:

1. Ubicar al usuario, sentado frente a la pantalla del computador y colocarle los audífonos.

2. Explicar que se presentarían dos modelos y que con cada uno de ellos se realizarían dos pruebas (sin mencionar detalle alguno sobre tales modelos).
3. Mostrar una vista superior de la totalidad del ambiente gráfico con el fin de contextualizar al usuario dentro del mismo.

4. Indicar la forma de manejo de la aplicación (Interacción del usuario con el sistema).
5. Dar recomendaciones en cuanto a las condiciones de la prueba (e.g. dejar la cabeza quieta mirando hacia la pantalla)
6. Ejecutar la aplicación[9]
7. Tomar el tiempo que tarda el usuario en encontrar la fuente del sonido reproducido para cada una de las cuatro pruebas.
8. Solicitar al usuario que responda una pequeña encuesta (Ver Anexo 3: "Formato de Encuesta").

Con cada uno de los modelos se realizaron dos pruebas, que consistieron en:

1. Ubicar al usuario en un ambiente dentro del cual debía encontrar la fuente de un sonido reproducido a través de los audífonos. Esta fuente está representada por un pato; cabe aclarar que para esta primera prueba el pato es único y se encuentra modelado en algún punto del laberinto.

2. En la segunda prueba se ubicó nuevamente al usuario en un ambiente similar al de la primera prueba. En ésta el cambio radica en que no existe uno sino varios patos modelados dentro del ambiente pero sólo uno de ellos es el que emite el sonido reproducido en los audífonos. Esto obliga a que el usuario intente ubicarse auditivamente y no sólo se guíe por lo que ve al navegar dentro del ambiente.

Las pruebas fueron diseñadas de esta forma buscando que en la primera de ellas el usuario se familiarice con el ambiente y con la forma de interactuar con el sistema. La segunda prueba, según opinión del grupo, es la que aporta mayor información en cuanto al comportamiento del modelo debido a que existen varios

[9] Para la ejecución de la aplicación se alternaba el uso del modelo, esto es, para un usuario se presentaba primero el modelo IAD-ITD, para el siguiente se comenzaba con el HRTF.

modelos que visualmente pueden distraer la atención del usuario y por lo tanto el sonido entra a jugar un papel determinante en la identificación y escogencia de la fuente verdadera.

Al terminar la prueba se ha recolectado la siguiente información:

1. Edad y sexo del encuestado.
2. Tiempos en encontrar la fuente de sonido para las pruebas anteriormente mencionadas.
3. Encuesta con respuestas del usuario (4 preguntas)

4.2. CONSIDERACIONES SOBRE LA MUESTRA

Para la muestra se escogieron 52 usuarios, 26 hombres y 26 mujeres entre los 19 y 25 años de edad. Tanto para el grupo masculino como para el grupo femenino se encuestó a la mitad de los usuarios utilizando en primer lugar el modelo IAD-ITD y a la otra mitad comenzando con el modelo HRTF.

4.3. RESULTADOS DE LAS ENCUESTAS

4.3.1. Resultados Globales

A continuación se presentan los resultados tabulados que arrojaron las encuestas.

A la pregunta: ¿El sonido le ayudó a ubicarse y desplazarse dentro del ambiente gráfico? 51 de los 52 encuestados respondió afirmativamente.

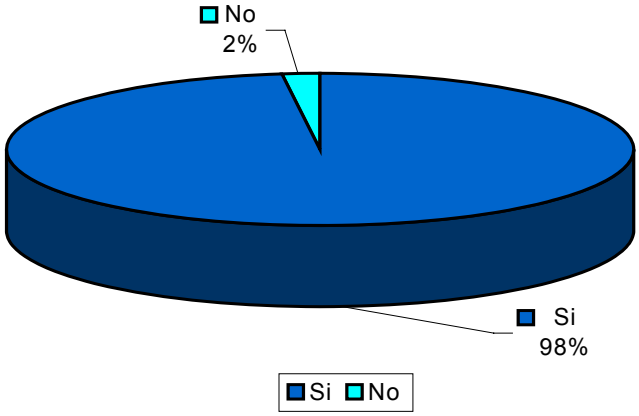


Figura 15. Ayuda del sonido en la ubicación espacial

En la siguiente pregunta se pidió a los encuestados marcar con una X el modelo de espacialización de sonido que les permitió una mejor ubicación. En este caso, el 65% de los encuestados se inclinó por el modelo HRTF, el 12% por el modelo IAD – ITD mientras que el 23% no encontró diferencia significativa entre ambos modelos.

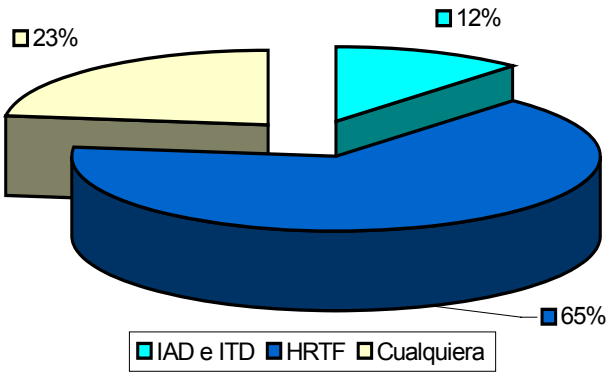
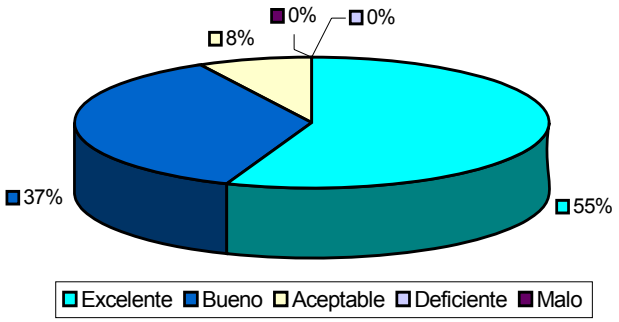


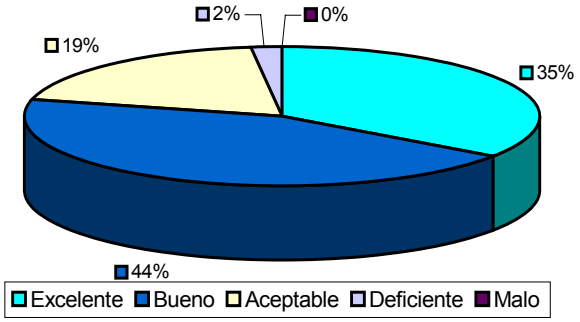
Figura 16. Modelo que permitió mejor ubicación

Finalmente se solicitó a los usuarios calificar la concordancia del movimiento de la cámara con el sonido reproducido para cada uno de los modelos. En el caso del Modelo HRTF, tal como se muestra en la Figura 17 (a), el 55% de los encuestados opinó que la concordancia era excelente, 37% que era buena y 8% que era

aceptable mientras que en el caso del modelo IAD – ITD, como se muestra en la Figura 17 (b), el 35% de los encuestados respondió que la concordancia era excelente, 44% que era buena, 19% que era aceptable y 2% que era deficiente.



(a)



(b)

Figura 17. Calificación de la concordancia entre imagen y sonido para (a) Modelo HRTF; (b) Modelo IAD – ITD.

4.3.2. Resultados Clasificados por Sexos.

A continuación se presentan los resultados a las mismas preguntas clasificados por sexos

4.3.2.1. Resultados para Hombres.

En el caso de los hombres, el 65% de los encuestados prefirieron el Modelo HRTF, el 12% prefirió el modelo IAD – ITD y el 23% no encontró diferencia entre los modelos.

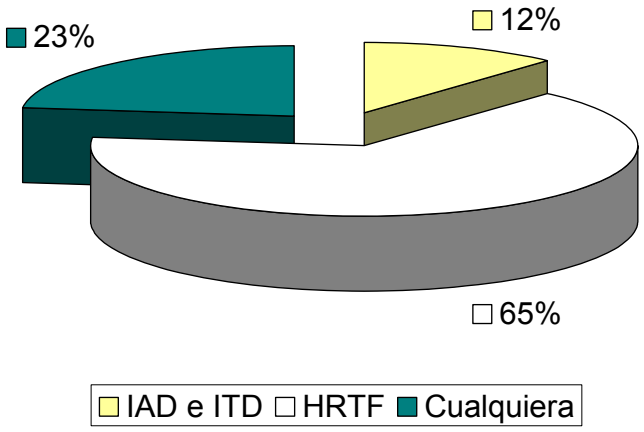


Figura 18. Modelo que permitió mejor ubicación (Hombres)

En cuanto a la concordancia entre la cámara y el sonido reproducido para el modelo IAD – ITD el 35% lo consideró excelente, el 42% bueno y el 23 lo encontró aceptable. Respecto a la concordancia entre movimiento y sonido en el modelo HRTF los hombres lo calificaron excelente en el 42% de los casos, bueno en el 46% y aceptable en el 12%. Ningún modelo fue considerado deficiente ni malo.

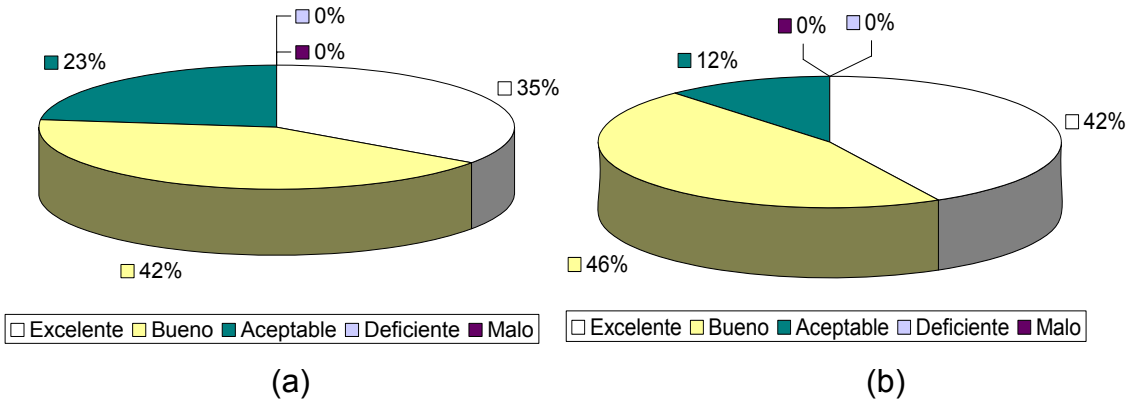


Figura 19: (a) Naturalidad del Sonido Reproducido Modelo IAD e ITD (Hombres)
(b) Naturalidad del Sonido Reproducido Modelo HRTF (Hombres)

4.3.2.2. Resultados para Mujeres.

En el caso de las mujeres, las encuestadas respondieron en un 65% que el modelo que mejor ubicación les brindó fue el HRTF, 12% se inclinó por el IAD ITD mientras que el 23% no encontró ninguna diferencia entre los dos modelos.

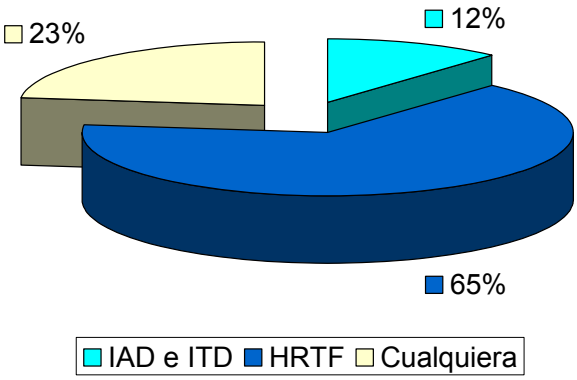
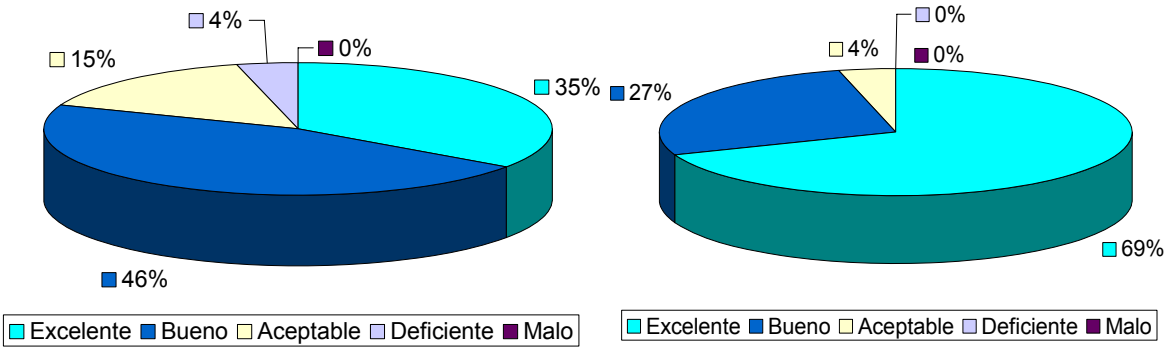


Figura 20. Modelo que permitió mejor ubicación (Mujeres)

Concerniente a la relación entre la cámara y el sonido, las encuestadas opinaron que el modelo IAD – ITD era excelente en un 35%, bueno en un 46%, aceptable en 15% y deficiente en un 4%. Para el modelo HRTF opinaron en 69% que era excelente, 27% afirmaron que era bueno y 4% que era aceptable



(a) (b)

Figura 21: (a) Naturalidad del Sonido Reproducido Modelo IAD e ITD (Mujeres)
(b) Naturalidad del Sonido Reproducido Modelo HRTF (Mujeres)

4.4. ANÁLISIS ESTADÍSTICO

4.4.1. Tiempos de Identificación de la Fuente.

A continuación se estiman las medias poblacionales μ del tiempo (en segundos) que tarda una persona en identificar la fuente de sonido para cada una de las pruebas realizadas con los modelos de espacialización de sonido.

Para tal fin se utiliza la media muestral ($\bar{x}$) dado que es un estimador no sesgado y en la mayoría de tiene una varianza más pequeña que la de cualquier otro estimador de μ.

4.4.1.1. Modelo IAD – ITD un Solo Pato

En la Figura 22 se muestra en histograma la distribución de los tiempos que los usuarios tardaron en encontrar la fuente de sonido utilizando el modelo IAD - ITD en un ambiente con un sólo modelo (donde el modelo es la fuente del sonido). A partir de tales datos se halló la media muestral y la desviación estándar:

$$\bar{x} = 47.83 \qquad \sigma = 34.94 \qquad (7)$$

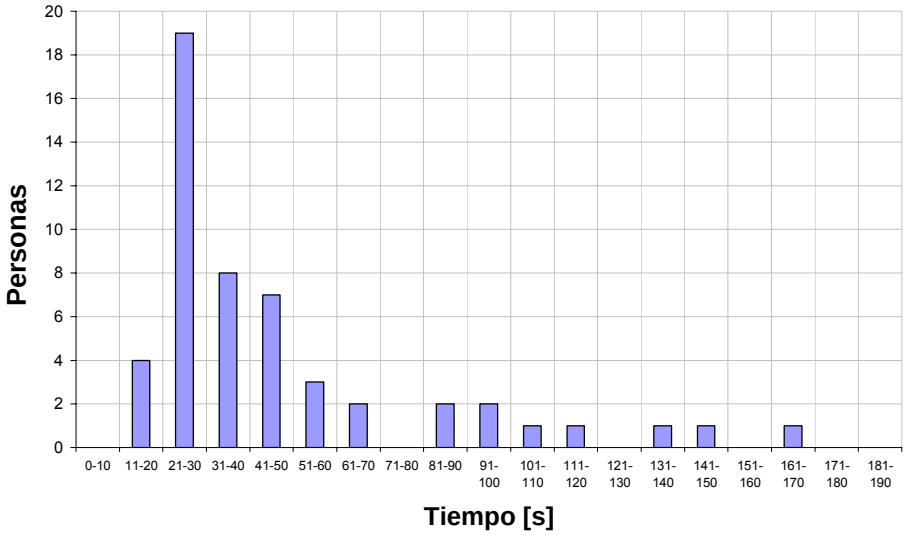


Figura 22: Histograma para la distribución de tiempos en prueba con un solo pato utilizando modelo IAD - ITD

Con base en estos parámetros, se encontró el intervalo de confianza de 95% para el tiempo medio de identificación de la fuente de sonido:

$$38.33\ s < \mu < 57.32\ s \qquad (8)$$

4.4.1.2. Modelo IAD – ITD Varios Patos

En la Figura 23 se muestra en histograma la distribución de los tiempos que los usuarios tardaron en encontrar la fuente de sonido utilizando el modelo IAD – ITD en un ambiente con varios modelos (patos que no emiten sonido alguno). A partir de tales datos se halló la media muestral y la desviación estándar:

$$\bar{x} = 67.88 \qquad \sigma = 35.22 \qquad (9)$$

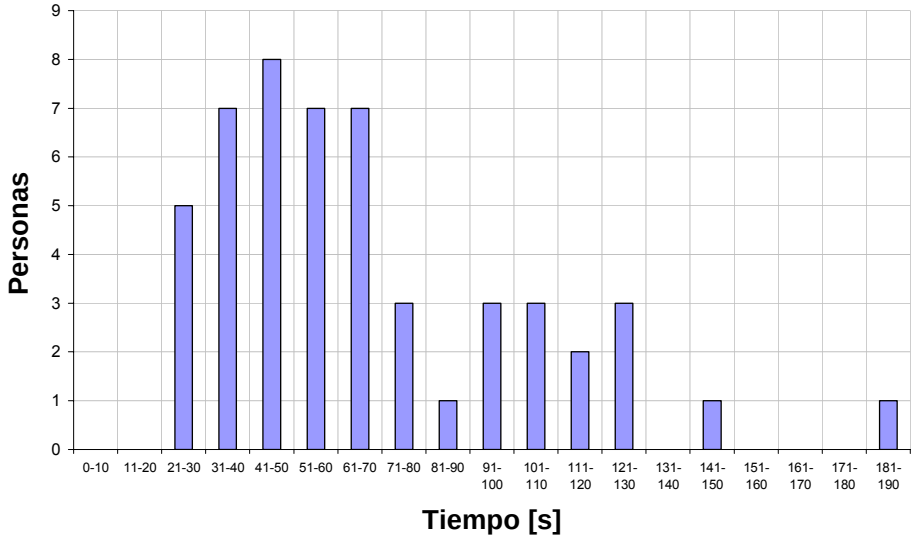


Figura 23: Histograma para la distribución de tiempos en prueba con un varios patos utilizando modelo IAD - ITD

Con base en estos parámetros, se encontró el intervalo de confianza de 95% para el tiempo medio de identificación de la fuente de sonido:

$$58.21\,s < \mu < 77.54\,s \qquad (10)$$

4.4.1.3. Modelo HRTF un Solo Pato

En la figura 24 se muestra en histograma la distribución de los tiempos que los usuarios tardaron en encontrar la fuente de sonido utilizando el modelo HRTF en un ambiente con un sólo modelo (donde el modelo es la fuente del sonido). A partir de tales datos se halló la media muestral y la desviación estándar:

$$\overline{x} = 46.29 \qquad \sigma = 32.04 \qquad (11)$$

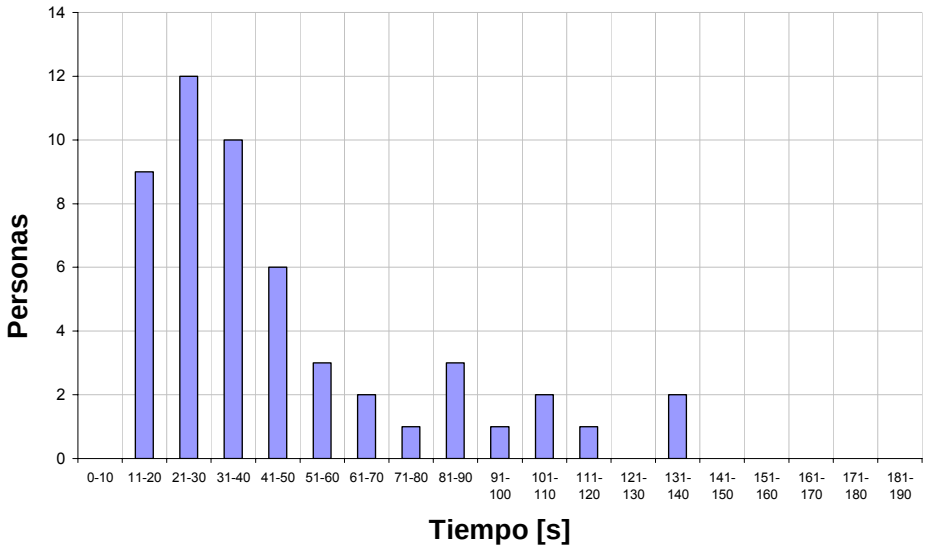


Figura 24: Histograma para la distribución de tiempos en prueba con un solo pato utilizando modelo HRTF

Con base en estos parámetros, se encontró el intervalo de confianza de 95% para el tiempo medio de identificación de la fuente de sonido:

$$37.58\ s < \mu < 55.00\ s \qquad (12)$$

4.4.1.4. Modelo HRTF Varios Patos

En la Figura 25 se muestra en histograma la distribución de los tiempos que los usuarios tardaron en encontrar la fuente de sonido utilizando el modelo HRTF en un ambiente con varios modelos (patos que no emiten sonido alguno). A partir de tales datos se halló la media muestral y la desviación estándar:

$$\bar{x} = 61.29 \qquad \sigma = 26.36 \qquad (13)$$

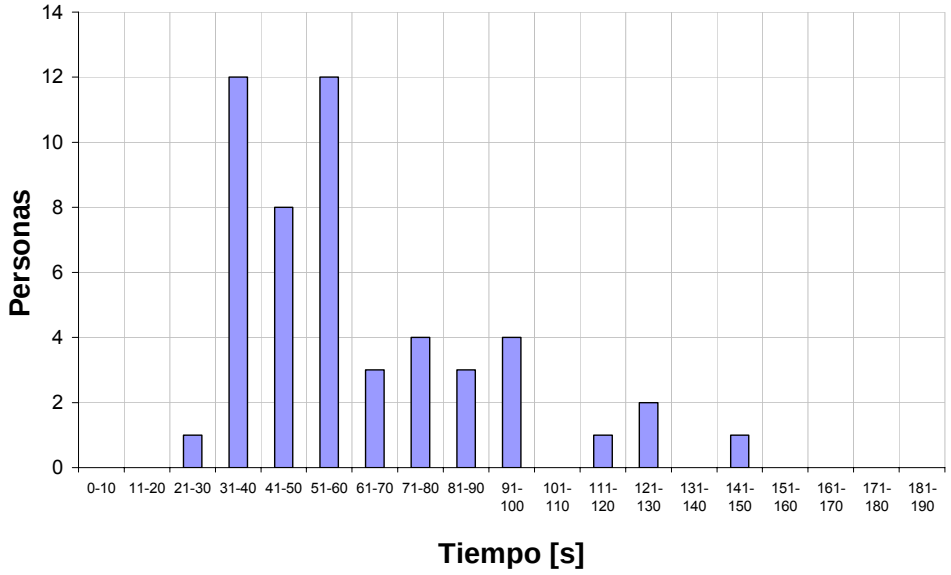


Figura 25: Histograma para la distribución de tiempos en prueba con un varios patos utilizando modelo HRTF

Con base en estos parámetros, se encontró el intervalo de confianza de 95% para el tiempo medio de identificación de la fuente de sonido:

$$54.06\ s < \mu < 68.53\ s \qquad (14)$$

4.4.1.5. Resumen

Como resumen, se muestra en la Figura 26 los intervalos de confianza (del 95%) de la media poblacional para el tiempo que un usuario tarda en encontrar la fuente de sonido en cada una de las pruebas.

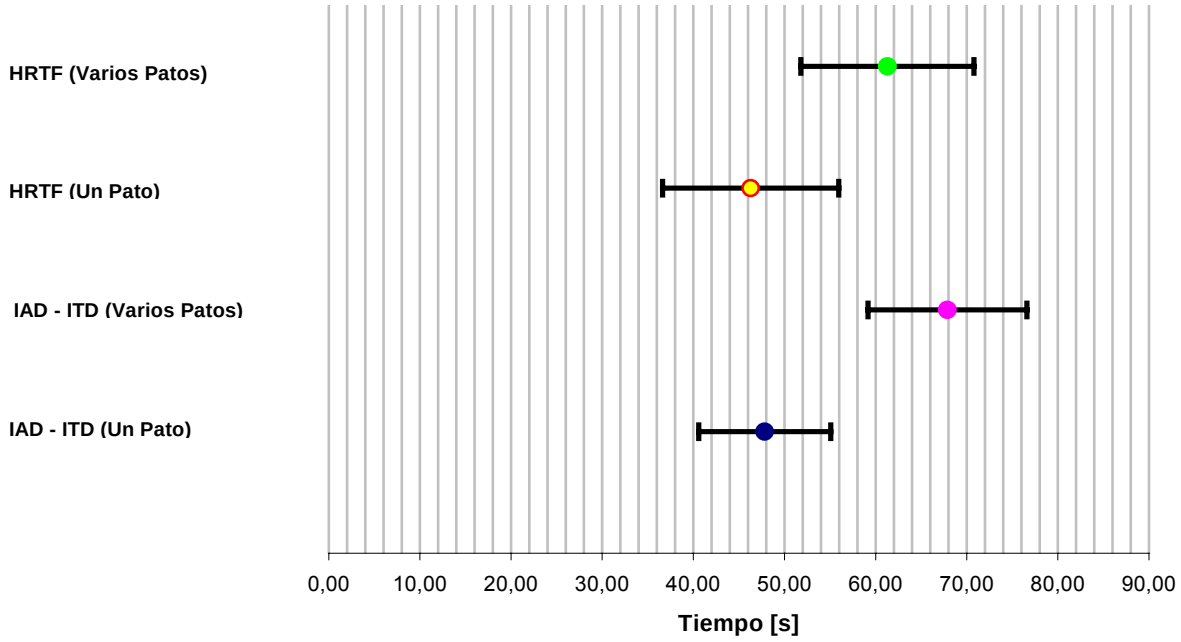


Figura 26: Comparación entre intervalos de confianza para tiempos de identificación de la fuente con cada modelo

En la figura 26 se aprecia que el modelo HRTF tiene medias muestrales ligeramente inferiores en ambas pruebas, sin embargo se observa un alto grado de sobrelapamiento en los intervalos de confianza de la primera prueba (con un pato) y por lo tanto no puede afirmarse categóricamente que el tiempo medio de identificación de la fuente es menor para el modelo HRTF que para el modelo IAD – ITD a pesar de encontrarse un leve indicio al respecto. En el caso de la segunda prueba (Varios patos) el grado de sobrelapamiento es menor y se estima que el tiempo medio en el que una persona encuentra la fuente de sonido es menor con el modelo HRTF que con el modelo IAD – ITD;

4.4.2. Gráficas de Caja y Extensión para los Resultados de Tiempo de Identificación de la Fuente

De igual forma se muestran a continuación las gráficas de caja y extensión en donde se comparan los resultados para los diferentes modelos en la prueba con un pato y en la prueba con varios patos. Debe recordarse que estas gráficas muestran el percentil 25, el percentil 75 y la mediana en un conjunto de datos.

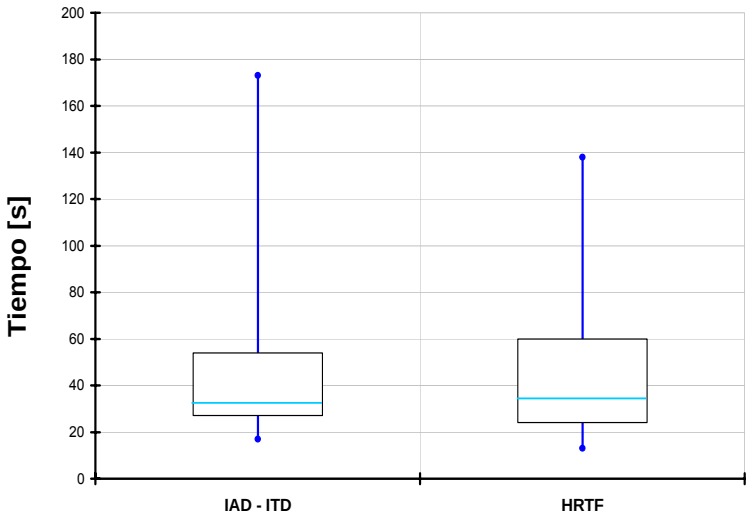


Figura 27: Gráfica de caja y extensión para la prueba con un pato

Como el lector puede apreciar, en la Figura 27 se observan medianas muy similares en la prueba con un pato y a pesar de la dispersión ligeramente mayor en los datos del modelo HRTF no puede establecerse una mejor calificación al modelo IAD ITD debido a la cercanía entre los resultados obtenidos con las dos pruebas.

En cuanto a los resultados para la prueba con varios patos puede inferirse a partir de la Figura 28, un mejor comportamiento del modelo HRTF debido a que tiene una mediana menor y la dispersión de los datos es significativamente menor.

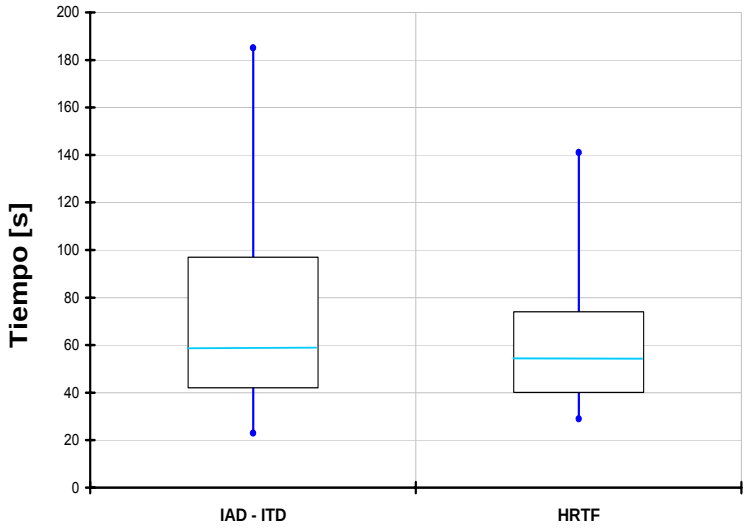
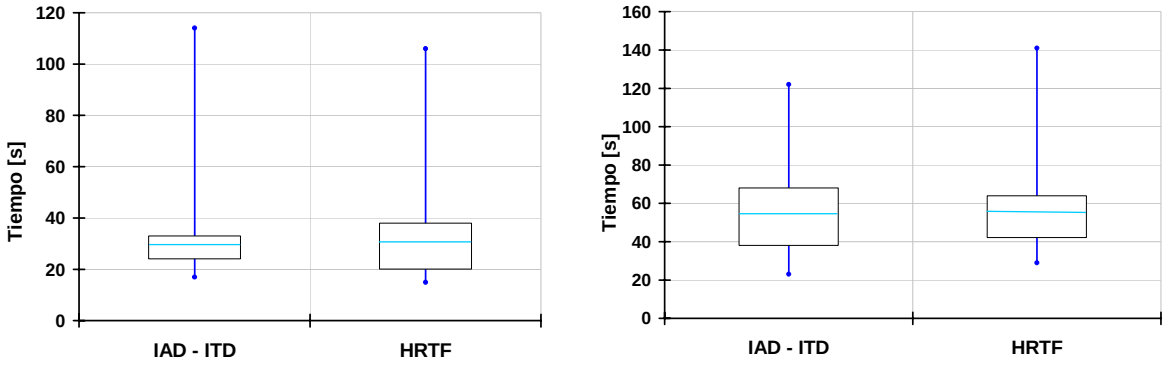


Figura 28: Gráfica de caja y extensión para la prueba con varios patos.

4.4.2.1. Gráficas de Caja y Extensión Clasificadas por Sexo.

Asimismo se presentan en las Figura 29 y 30 los diagramas de caja y extensión para los tiempos (en segundos) que tardaron los hombres y mujeres en encontrar la fuente de sonido para cada una de las dos pruebas.



a) b)

Figura 29: Gráfica de caja y extensión para la prueba en hombres:
a) Con un pato, b) Con varios patos.

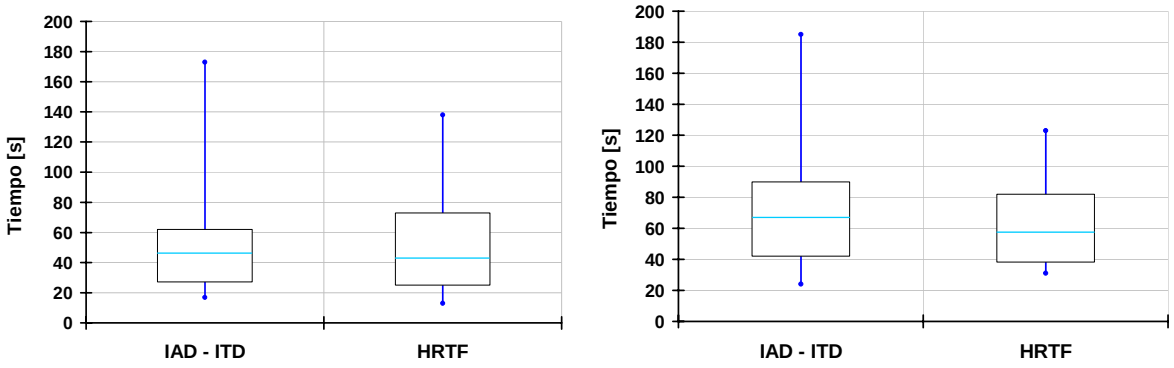


Figura 30: Gráfica de caja y extensión para la prueba en mujeres: a) Con un pato, b) Con varios patos.

Se observa al comparar las Figuras 29 y 30 una tendencia ligeramente mayor en las mujeres a desempeñarse mejor utilizando el modelo HRTF. En los hombres se observa un desempeño similar tanto con el modelo HRTF como con el modelo IAD – ITD.

4.5. COMPARACIÓN CON RESULTADOS ANTERIORES.

En el trabajo de maestría "Generación de sonido binaural a partir de una entrada monofónica"[10] se encontraron algunos resultados que coinciden con los encontrados en el presente proyecto, los cuales se enuncian a continuación:

En el género femenino se observó una tendencia a preferir el modelo de HRFT sobre los demás modelos. Este resultado se evidencia en las respuestas a las preguntas realizadas en cuanto a la concordancia entre sonido e imagen para cada uno de los modelos tal como aparece demostrado en el numeral 4.3.2.

[10] HURTADO L., Jairo A. Generación de sonido binaural a partir de una entrada monofónica. Investigación de Maestría. Pontificia Universidad Javeriana. Bogotá D.C. 2002.

En cuanto a los hombres, aunque en su mayoría prefirieron el modelo HRTF, no puede emitirse un juicio de forma tan concluyente como en el caso anterior ya que aunque se evidenció una tendencia por el modelo HRTF en lo que se refiere a concordancia entre sonido e imagen, la inclinación hacia el resultado “excelente” no fue tan contundente como en el caso femenino.

4.6. COMPARACIÓN DE LOS RESULTADOS CON RESPECTO A LOS OJETIVOS PLANTEADOS.

En el objetivo principal del proyecto se determinó implementar y evaluar el desempeño de modelos de espacialización de sonido al utilizarlos dentro de un programa que integrara en tiempo real el sonido generado por estos modelos junto con un ambiente gráfico dentro del cual el usuario pudiera cambiar de posición en dos dimensiones y a su vez, el sonido que se reprodujera fuera modificado con base en la nueva ubicación.

Al respecto, se considera que el objetivo fue alcanzado en su totalidad ya que, en primera instancia, fue posible crear un programa que integró un ambiente gráfico tridimensional totalmente interactivo con el usuario en cuanto al cambio de posición junto con la reproducción en tiempo real de los sonidos resultado de la implementación de los modelos de espacialización de sonido.

Por otra parte, la implementación se realizó de tal manera que fue posible evaluar adecuadamente tanto el modelo de retardo interaural como el modelo de la función de transferencia de la cabeza (HRTF).

En cuanto a los objetivos específicos, se considera de la misma forma que fueron exitosamente alcanzados ya que:

- Se desarrolló en software un sistema en donde el usuario puede cambiar sus coordenadas en el plano azimutal
- Se implementaron los modelos de sonido binaural enfocados al movimiento azimutal en tiempo real, totalmente ligados al movimiento del usuario (representado por la cámara) dentro del ambiente gráfico creado por los integrantes del trabajo.
- Se realizaron encuestas sobre el aplicativo que permitieron comparar los resultados de la percepción por parte de los 52 usuarios encuestados en cada uno de los modelos de sonido implementados y posteriormente se realizaron las valoraciones estadísticas tendientes a formular condiciones.

4.7. ESTUDIO DE COSTOS.

A pesar de que se presentaron algunas variaciones en los costos ocasionados en el presente trabajo, el costo total se mantuvo cercano al proyectado inicialmente. El costo final del trabajo estuvo en $ 109 780 por debajo del costo proyectado junto con el valor de los imprevistos.

En el campo de los recursos humanos se presentó un incremento de $ 3 240 000 las variaciones fueron ocasionadas por el incremento o decremento de las horas destinadas por las personas que intervinieron en el trabajo. En el campo de los recursos técnicos se presentó una disminución de $ 1 160 000, producto de la reducción de las horas de utilización de estos recursos. En el campo de otros, los gastos fueron realizados según las proyecciones. Para mas detalles favor remitirse a los Anexos 4 “Presupuesto Inicial” y 5 “Costo final del proyecto”

En general puede concluirse que los gastos se ajustaron al presupuesto presentado inicialmente.

5. CONCLUSIONES

Como fruto del proceso de implementación y pruebas del sistema se ha confirmado lo supuesto por Hurtado y Vizcaya [12] respecto a la importancia del sentido de la vista en el proceso de ubicación del ser humano. Se encontró que no se presentaron las dificultades allí documentadas en lo que concierne a la diferenciación de un sonido que viene de adelante o de atrás. Es importante anotar que en un sistema en el cual no se integren sonido y visión sino sólo sonido, el cerebro intenta crear una visión conjunta, sin embargo, dado que no existe concordancia entre ellas se producen fallas en la ubicación del usuario tales como, ubicar atrás un sonido que debería ser localizado adelante o viceversa. Es por esto que tan sólo con suministrar al cerebro informaciones consistentes entre imagen y sonido se puede lograr una mejora sustancial en el proceso de ubicación espacial del usuario. Por lo descrito anteriormente se evidencia que el módulo de imagen y la facilidad de interacción del usuario con el sistema fueron factores claves en la ubicación espacial de los usuarios.

Por otra parte, es importante mencionar que se obtuvo una buena calificación por parte de los usuarios en cuanto a la calidad de la implementación y a la concordancia entre imagen y sonido, sin importar el modelo utilizado. Esto muestra que el modelado del ambiente gráfico, el proceso de selección de las herramientas de desarrollo de software y la forma de implementarlas, brindaron frutos a la hora de presentar al usuario final un sistema rápido, coherente (entre imagen y sonido) y estéticamente agradable.

Es interesante asimismo confirmar las suposiciones que se tenían en torno a la posible mayor aceptación del modelo de función de transferencia de la cabeza (HRTF) frente al modelo de retardo interaural y de magnitud (IAD – ITD). Tal como lo muestran múltiples estadísticas el modelo HRTF evidenció fortalezas tanto en la evaluación directa de los usuarios, como en la medida indirecta que se hizo de los mismos al comparar los tiempos utilizados por los encuestados en las diversas pruebas realizadas.

Respecto a la hipótesis según la cual se sugería utilizar modelos diferentes al de HRTF debido a la posible dificultad en la implementación del mismo en cuanto a una mayor carga de procesamiento y a un mayor espacio en memoria para su adecuada presentación, no se encontraron diferencias significativas a la hora de implementar cada uno de los modelos mencionados anteriormente. Es por ello que se considera que el modelo HRTF presenta un mejor comportamiento y una mayor aceptación sin que su complejidad llegue a ser un obstáculo a la hora de realizar aplicaciones rápidas y consistentes.

Se considera que aunque se obtuvo una buena percepción de la espacialización del sonido por parte de los usuarios, para aplicaciones más exigentes ésta puede mejorarse aún más utilizando una mayor cantidad de ángulos con respecto a la fuente de sonido en el plano azimutal (en la aplicación desarrollada se utilizaron pasos de 15 grados). Esto permitirá al usuario sentir un poco mas suave el cambio a la hora de rotar dentro del ambiente, sin embargo debe tenerse cuidado ya que esto puede llegar a incrementar la carga de procesamiento tanto en la parte auditiva como en la parte de imagen (renderización del ambiente gráfico).

Finalmente, es claro el potencial que puede llegar a tener la utilización de los modelos de espacialización de sonido en diversas aplicaciones, especialmente en

aquellas que trabajan en conjunto con presentaciones visuales hacia el usuario tales como juegos de video (de hecho la aplicación desarrollada puede considerarse un pequeño juego de video) y otras aplicaciones comerciales tendientes a la comercialización de ambientes donde se presente al usuario remoto una aproximación del producto ofrecido.

BIBLIOGRAFÍA

1. ANGEL, Edward. Interactive Computer Graphics, A Top-Down Approach with OpenGL. Adisson Wesley Longman Inc.

2. AUREAL CORPORATION. 3D Audio Primer. Technical Papers Library.

3. BAMFORD, Jeffery S. Ambisonic Sound for the Masses. Audio Research Group. University of Waterloo. Notario.

4. BARGEN, Bradley. A Fondo DirectX. Serie de programación Microsoft. Madrid ; Buenos Aires : McGraw Hill, 1998.

5. BEGAULT, Durand R.3-D Sound For Virtual Reality And Multimedia . NASA 2000.

6. BROWN, Phllip & Duda, Richard O. A Structural Model for Binaural Sound Synthesis. IEEE Transaction on Speech And Audio Processing. Vol 6, No 5, September 1998.

7. BRUNGART, Douglas S. & RABIOWITZ, William R. Auditory Localization in the Near-Field. Research Laboratory of Electronics. Proceedings of ICAD (International Conference on Auditory Display).

8. CLEMOW, Richard & SIBBALD, Alastair. MultiDrive Technology. Sensaura. 3D Positional Audio.

9. DEITEL H. M. & DEITEL P.J., Cómo programar en C++. Prentice Hall Inc., 1998.

10. HeadWize Technical Series Paper. The Elements of Musical Perception.

11. HeadRoom Corporation. The Psychoacoustics of Headphone.

12. HURTADO L., Jairo A., VIZCAYA, Pedro Generación de sonido binaural a partir de una entrada monofónica. Investigación de Maestría. Pontificia Universidad Javeriana. Bogotá D.C. 2002.

13. Knowles Electronics. KEMAR ((Knowles Electronics Mannequin for Acoustical Research) developers. http://www.knowles.com/html/comp_knowles.htm

14. KRAEMER, Alan. SRS Labs Inc.Two Speakers Are Better Than 5.1. IEEE Spectrum. May 2001.

15. MARTIN, Keith & GARDNER, Bill. HRTF Measurements of a KEMAR Dummy-Head Microphone. Media Lab Perceptual Computing. MIT. Mayo 1994

16. MARTENS, William L. Ph.D. Spatial Audio Terminology.. University of Aizu.

17. MICROSOFT CORP., DirectX 9.0 SDK Update, Microsoft Press, Summer 2003.

18. PHILP, Adam. Enviroment FX. Audio Enviroment Modelling. Sensaura. 3D Positional Audio.

19. STOICA, Petre & MOSES, Randolph. Introduction To Spectral Analysis. Prentice Hall. USA. 1997

20. SIBBALD, Alastair. Virtual Ear Technology. Sensaura. 3D Positional Audio.

21. SIBBALD, Alastair. Virtual Audio for Headphones. Sensaura. 3D Positional Audio.

22. SIBBALD, Alastair. Hearing in Three Dimensions. Sensaura. 3D Positional Audio.

23. SIBBALD, Alastair. An Introduction To Sound And Hearing. Sensaura. 3D Positional Audio.

24. SIBBALD, Alastair. Digital Ear Technology. Sensaura. 3D Positional Audio.

25. SPHAR, Chuck. Aprenda Microsoft Visual C++ 6.0 Ya. Microsoft Press. Madrid. 1999.

26. STARKEY TECHNICAL SERVICES DIVISION, Introduction To Audiometry And Amplification.

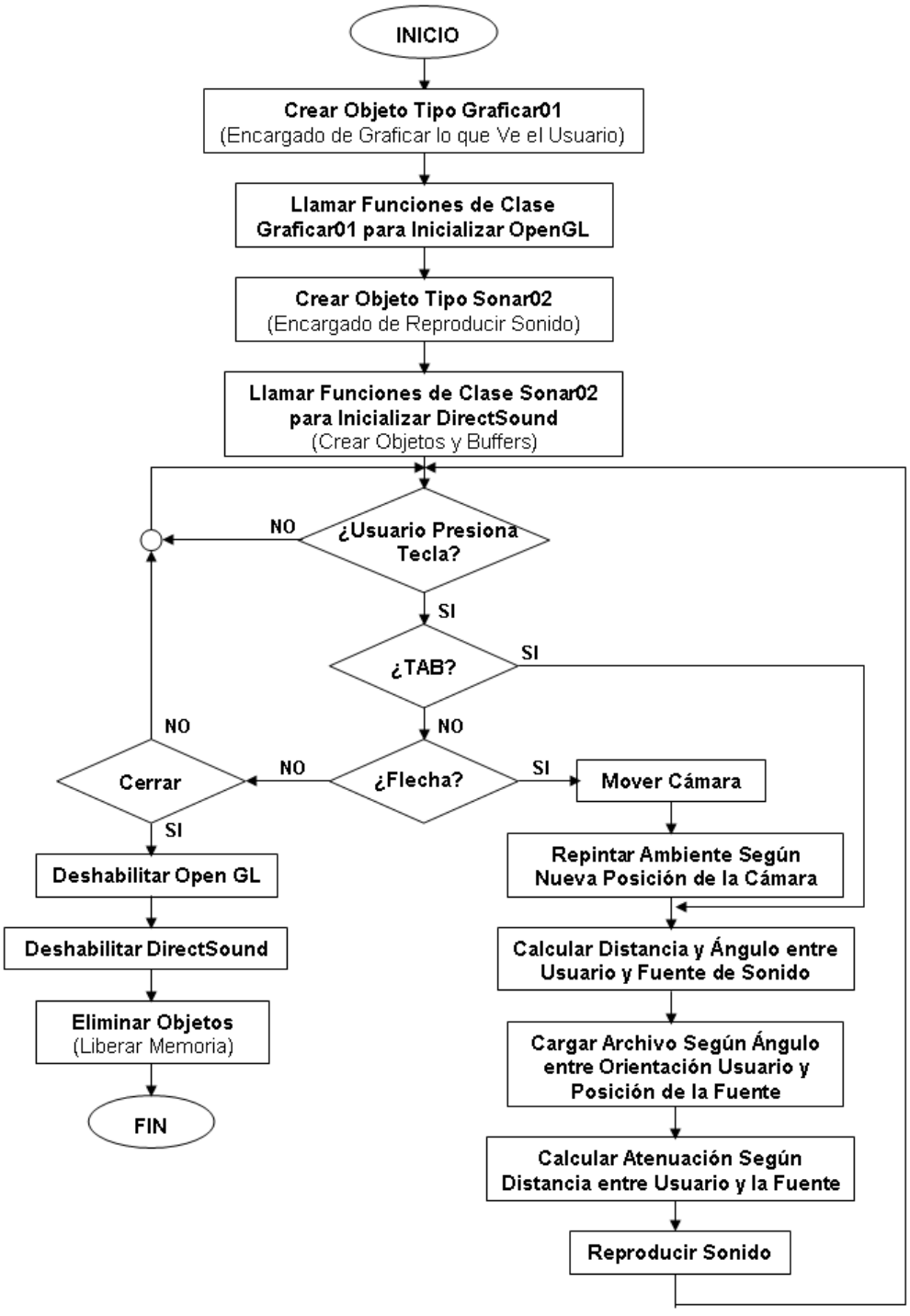
27. Audio Engineering Society. http://www.aes.org

28. Diccionario bilingüe de Audio Profesional. http://www.doctorproaudio.com/doctor/diccionario.htm

29. Readings in 3-D Sound and Spatial Audio
http://www.northwestern.edu/musicschool/classes/3D/pages/3DclassReadings.html

30. 3-D Sound Tutorial
http://interface.cipic.ucdavis.edu/CIL_tutorial/3D_sys2

ANEXO 1: DIAGRAMA DE FLUJO DE PROGRAMA



ANEXO 2: CÓDIGO FUENTE

A. Programa Principal

```
// Maneja de eventos y llama a las funciones encargadas de mostrar las imágenes y reproducir los sonidos
// prueba04.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"
#include "Graficar01.h"
#include "Sonar02.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // The title bar text



Graficar01 *graficador;
bool modelo=true;   //true: HRTF
                                                //false: IADITD
Sonar02 *sonidohrtf;


// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_PRUEBA04, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_PRUEBA04);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
```

```
                }
        }

        return msg.wParam;
}


ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc   = (WNDPROC)WndProc;
        wcex.cbClsExtra             = 0;
        wcex.cbWndExtra             = 0;
        wcex.hInstance              = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_PRUEBA04);
        wcex.hCursor                = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground          = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName           = (LPCSTR)IDC_PRUEBA04;
        wcex.lpszClassName          = szWindowClass;
        wcex.hIconSm                = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}


BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;

   hInst = hInstance; // Store instance handle in our global variable

   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      0, 0, 1890, 1168, NULL, NULL, hInstance, NULL);

   if (!hWnd)
   {
      return FALSE;
   }


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

        graficador = new Graficar01();
        graficador->iniciarOpengl(hWnd);
        graficador->cicloDibujado();


        sonidohrtf = new Sonar02();
        sonidohrtf->OnInit(hWnd);
        sonidohrtf->CargarDirectSound("Duck00.wav");
        sonidohrtf->OnSound(graficador->distanciaconpato());

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



   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}
```

```
//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND - process the application menu
//  WM_PAINT         - Paint the main window
//  WM_DESTROY   - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;float p;

        switch (message)
        {

                case WM_CREATE:

                        break;

                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
                                  break;
                                case IDM_EXIT:
                                  DestroyWindow(hWnd);
                                  break;
                                case IDM_HRTF:
                                        modelo=true;
                                        graficador->irainicio();
                                        graficador->setpato(219.0f,93.0f);
                                        break;
                                case IDM_IADITD:
                                        modelo=false;
                                        graficador->irainicio();
                                        graficador->setpato(-264.0f,-58.0f);
                                        break;
                                default:
                                  return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;

                case WM_PAINT:
                        graficador->cicloDibujado();
                        break;

                case WM_KEYDOWN:
                        switch ( wParam )
                        {

                        case VK_ESCAPE:
                                PostQuitMessage(0);
                                return 0;
                        case VK_UP:
                                if (!graficador->colisiones(0))
                                {
                                        graficador->movercamaraadelante();
                                        graficador->cicloDibujado();
                                }
```

```
                    sonidohrtf->Borrar();
                    p=graficador->anguloconpato();
                    if (modelo)
                            sonidohrtf->CargarDirectSound(sonidohrtf->CargarArchivo(p));
                    else
                                    sonidohrtf->CargarDirectSound(sonidohrtf-    …
                                                                    ...   >CargarArchivoIAD(p));
                    sonidohrtf->OnSound(graficador->distanciaconpato());
                    return 0;
case VK_RIGHT:
        graficador->movercamaraderecha();

                    sonidohrtf->Borrar();
                    p=graficador->anguloconpato();
                    if (modelo)
                            sonidohrtf->CargarDirectSound(sonidohrtf->CargarArchivo(p));
                    else
                            sonidohrtf->CargarDirectSound(sonidohrtf-      …
                                                                    ...   >CargarArchivoIAD(p));
                    sonidohrtf->OnSound(graficador->distanciaconpato());
        return 0;

case VK_LEFT:
        graficador->movercamaraizquierda();

                    sonidohrtf->Borrar();
                    p=graficador->anguloconpato();
                    if (modelo)
                            sonidohrtf->CargarDirectSound(sonidohrtf->CargarArchivo(p));
                    else
                            sonidohrtf->CargarDirectSound(sonidohrtf-      …
                                                                    …   >CargarArchivoIAD(p));
                    sonidohrtf->OnSound(graficador->distanciaconpato());
        return 0;

case VK_DOWN:
        if (!graficador->colisiones(1))
        {
                    graficador->movercamaraatras();
                    graficador->cicloDibujado();
        }

                    sonidohrtf->Borrar();
                    p=graficador->anguloconpato();
                    if (modelo)
                            sonidohrtf->CargarDirectSound(sonidohrtf->CargarArchivo(p));
                    else
                            sonidohrtf->CargarDirectSound(sonidohrtf-   …
                                                            …       >CargarArchivoIAD(p));
                    sonidohrtf->OnSound(graficador->distanciaconpato());
        return 0;

case VK_TAB:

                    sonidohrtf->Borrar();
                    p=graficador->anguloconpato();
                    if (modelo)
                            sonidohrtf->CargarDirectSound(sonidohrtf->CargarArchivo(p));
                    else
                            sonidohrtf->CargarDirectSound(sonidohrtf-    …
                                                            …           >CargarArchivoIAD(p));
                    sonidohrtf->OnSound(graficador->distanciaconpato());
        return 0;
default:
        return 0;
        break;
}
```

```
                        case WM_DESTROY:
                                graficador->DisableOpenGL(hWnd);
                                delete graficador;
                                sonidohrtf->Destroy();
                                delete sonidohrtf;

                                PostQuitMessage(0);
                                break;
                        default:
                                return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                                return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
   return FALSE;
}
```

B. Clase Cargar01

Declaración de la clase Cargar01

```
// argar01.h: interface for the Cargar01 class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_ARGAR01_H__EE612D5F_0FE3_4401_AB85_548D2B3E1593__INCLUDED_)
#define AFX_ARGAR01_H__EE612D5F_0FE3_4401_AB85_548D2B3E1593__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Objeto.h"
#include "punto.h"
#include "textura.h"
#include "normal.h"
#include "orden.h"
#include "material.h"
#include "Objeto.h"

class Cargar01
{

private:

        char buffer[5200000];           //Buffer donde se almacenará el archivo obj
        char buffer2[10000];            Buffer donde se almacenará el archivo mtl
        Objeto* atributos;
        textura** t;
        normal** n;
        orden** o;
        material** m;
        punto** p;
public:

        void contar(void);
        bool leer(void);
        Cargar01();
        virtual ~Cargar01();
        void almacenar(void);
        long saltarlinea(long i);
        int saltarlinea2(int j);
        float adquirirdatof(long i);
        float adquirirdato2f(int i);
        int adquirirdatoi(long i,int sig);
        void cargarmaterial(void);
        punto* consultarp(long indice);
        textura* consultart(long indice);
        normal* consultarn(long indice);
        orden* consultaro(long indice);
        material* consultarm(int indice);
        Objeto* consultaratrib();
};

#endif // !defined(AFX_ARGAR01_H__EE612D5F_0FE3_4401_AB85_548D2B3E1593__INCLUDED_)
```

Implementación de la clase Cargar01

```
//argar01.cpp: implementation of the Cargar01 class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "argar01.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <math.h>
#include <io.h>
#include <iostream.h>
#include <fcntl.h>
#include <GL/glu.h>
#include <GL/gl.h>


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Cargar01::Cargar01()
{
	atributos = new Objeto();

}

Cargar01::~Cargar01()
{

}

// Esta función se encarga de contar el número de vértices, normales, texturas, materiales y triángulos a ser dibujados
void Cargar01::contar(void)
{
	//------------------------------------------------------------------//
	// Cuenta objetos
				//
	//------------------------------------------------------------------//
	long i, j=0;
	for(i=0; i<5200000-1; i++)
	{

		if(buffer[i] == 103  && buffer[i+1] == 32)
			j++;
	}
	atributos->setnumobjects(j);



	//------------------------------------------------------------------//
	// Cuenta vertices y crea matriz de puntos con el número de vértices//
	//------------------------------------------------------------------//


	j=0;
	for(i=0; i<5200000-1; i++)
	{
		if(buffer[i] == 118  && buffer[i+1] == 32)
			j++;
	}
	atributos->setnumvertices(j);
	p=new punto*[j];
```

```
for (i=0;i<j;i++)
		p[i]= new punto();


//---------------------------------------------------------------------//
// Cuenta normales y crea matriz de normales con el número de normales//
//---------------------------------------------------------------------//

j=0;
for(i=0; i<5200000-1; i++)
{
		if(buffer[i] == 118  && buffer[i+1]== 110)
			j++;
}
atributos->setnumnormales(j);
n=new normal*[j];
for (i=0;i<j;i++)
		n[i]= new normal();


//---------------------------------------------------------------------//
// Cuenta texturas y crea matriz de texturas con el número de texturas//
//---------------------------------------------------------------------//

j=0;
for(i=0; i<5200000-1; i++)
{
		if(buffer[i] == 118  && buffer[i+1]== 116)
			j++;
}
atributos->setnumtexturas(55);

t=new textura*[j];
for (i=0;i<j;i++)
		t[i]= new textura();



//---------------------------------------------------------------------//
// Cuenta órdenes y crea matriz de órdenes con el número de órdenes//
//---------------------------------------------------------------------//

j=0;
for(i=0; i<5200000-1; i++)
{
		if(buffer[i] == 102 && buffer[i+1] == 32)
			j++;
}
atributos->setnumordenes(j);

o=new orden*[j];
for (i=0;i<j;i++)
		o[i]= new orden();


//---------------------------------------------------------------------//
// Cuenta materiales y crea matriz de materiales con el número de materiales//
//---------------------------------------------------------------------//

int l,ma;
ma=0;
for(l=0; l<10000-7; l++)
{
		if((buffer2[l] == 110 && buffer2[l+1] == 101) && (buffer2[l+2] == 119) )
			ma++;
```

```
	}
	atributos->setnummateriales(18);

	m=new material*[18];
	for (l=0;l<18;l++)
		m[l]= new material();



}

//Esta función se encarga de leer el archivo de texto plano .obj y dejarla en el buffer
bool Cargar01::leer(void)
{
	long nbytes = 5200000, bytesread, fh;
	int i=0,i1=0, nbytes2 = 10000, bytesread2,fh1;


	if( (fh = _open("Mod2Varios.obj", _O_RDONLY )) == -1 )
	{
		return false;
	}


    //Read in input:
	if( ( bytesread = _read( fh, buffer, nbytes ) ) <= 0 )
		return false;

	if( (fh1 = _open( "Mod2Varios.mtl", _O_RDONLY )) == -1 )
	{
		return false;
	}

	// Read in input:
	if( ( bytesread2 = _read( fh1, buffer2, nbytes2 ) ) <= 0 )
		return false;
	else
	{
		for(i=0;i<(5200000-bytesread-1);i++)
			buffer[bytesread+i]=32;
		return true;
	}
}
//Esta función tiene como objetivo almacenar de forma estructurada la información de vértices, normales, materiales y
texturas en //arreglos creados para tal fin
void Cargar01::almacenar(void)
{
	long  i=0, numvertact=0, numtexact=0, numnoract=0, numordact=0, numvert, numtex, numnor, bandera2;
	bool tienetextura=true;
	long j,  bandera, numord=0;
	float v1,v2,v3, n1,n2,n3, t1, t2;
	char* nombremat;


	for (long p=0; p<atributos->consulobjects();p++)
	{
		while (buffer[i] != 103)			//Se busca el caracter "g" para encontrar el primer objeto
			i=saltarlinea(i);
		i=saltarlinea(i);



	//Adquiriendo vértices


		bandera = i;			//Se guarda en bandera la dirección del primer vértice del objeto
		while (buffer[i] != 35)				//Se busca el lugar donde aparece el número total de vértices
```

```
		i=saltarlinea(i);
	i=i+2;

	numvert=adquirirdatoi(i,32);	//Se guarda en numvert el número de vértices del objeto
	i= bandera;					//Se retorna al primer vértice
	j=0;

	do
	{
		i=i+2;
		v1 = adquirirdatof(i);// Se convierte del array a punto flotante la primera componente
		while(buffer[i] != 32)
			i++;
		i++;
		v2 = adquirirdatof(i);// Se convierte del array a punto flotante la segunda componente
		while(buffer[i] != 32)
			i++;
		i++;
		v3 = adquirirdatof(i);// Se convierte del array a punto flotante la tercera componente
		i=saltarlinea(i);
		this->p[numvertact]->setpunto(v1,v2,v3);// Se guardan las componentes en el objeto
								//  punto: un vértice
		numvertact++;
		j++;
	}while(j<numvert);


//Adquiriendo texturas

	i=saltarlinea(i);

	if (buffer[i+1]==110)
	{
		i-=3;
		tienetextura = false;
	}
	else
	{
		bandera = i;		//Se guarda en bandera la dirección de la primera textura del objeto
		while (buffer[i] != 35)		//Se busca el lugar donde aparece el número total de texturas
			i=saltarlinea(i);
		i=i+2;
		numtex=adquirirdatoi(i,32);	//Se guarda en numtex el número de texturas del objeto
		i= bandera;				//Se retorna a la primera textura
		j=0;
		do
		{
			i=i+3;
			t1 = adquirirdatof(i); // Se convierte del array a punto flotante la primera componente
			while(buffer[i] != 32)
				i++;
			i++;
			t2 = adquirirdatof(i); |		// Se convierte del array a punto flotante la segunda
							//componente
			i=saltarlinea(i);
			this->t[numtexact]->settextura(t1,t2);	// Se guardan las componentes en el objeto
									//punto: un vértice
			numtexact++;
			j++;
		}while(j<numtex);

	}

//Adquiriendo normales

	i=saltarlinea(i);
	bandera = i;			//Se guarda en bandera la dirección de la primera normal del objeto
```

```
while (buffer[i] != 35)                    //Se busca el lugar donde aparece el número total de vértices
        i=saltarlinea(i);
i=i+2;
numnor=adquirirdatoi(i,32);     //Se guarda en numvert el número de normales del objeto
i= bandera;                                          //Se retorna a la primera normal
j=0;
do
{
        i=i+3;
        n1 = adquirirdatof(i);// Se convierte del array a punto flotante la primera componente
        while(buffer[i] != 32)
                i++;
        i++;
        n2 = adquirirdatof(i);// Se convierte del array a punto flotante la segunda componente
        while(buffer[i] != 32)
                i++;
        i++;
        n3 = adquirirdatof(i);// Se convierte del array a punto flotante la tercera componente
        i=saltarlinea(i);
        this->n[numnoract]->setnormal(n1,n2,n3); // Se guardan las componentes en el objeto punto
        numnoract++;
        j++;
}while(j<numnor);


//Guardando el orden

        i=saltarlinea(i);
        if (buffer[i] == 117)
        {
                while(buffer[i] != 32)
                        i++;
                i++;
                bandera2=1;
                while (buffer[i+bandera2] != 10)
                        bandera2++;
                nombremat = new char[bandera2+1];
                for (long k=0; k<bandera2;k++)
                        nombremat[k]=buffer[i+k];
                *(nombremat+bandera2)='\0';
                i=saltarlinea(i);
        }


        bandera=i;

        //Se cuentan ordenes
        while(buffer[i] != 35)
        {
                if(buffer[i] == 102)
                        numord++;
                i=saltarlinea(i);
        }
        i=bandera;

        for(j=0;j<numord;j++)
        {

                if(buffer[i] == 117)
                {
                        while(buffer[i] != 32)
                                i++;
                        i++;
                        bandera2=1;
                        while (buffer[i+bandera2] != 10)
                                bandera2++;
                        nombremat = new char[bandera2+1];
```

```
                                        for (long k=0; k<bandera2;k++)
                                                nombremat[k]=buffer[i+k];
                                        *(nombremat+bandera2)='\0';
                                        i=saltarlinea(i);
                                }
                                i+=2;
                                o[numordact+j]->setprimver(adquirirdatoi(i,47));
                                while(buffer[i] != 47)
                                        i++;
                                i++;
                                if (tienetextura)
                                        this->o[numordact+j]->setprimtex(adquirirdatoi(i,47));
                                while(buffer[i] != 47)
                                        i++;
                                i++;
                                this->o[numordact+j]->setprimnor(adquirirdatoi(i,32));
                                while(buffer[i] != 32)
                                        i++;
                                i++;
                                this->o[numordact+j]->setsegver(adquirirdatoi(i,47));
                                while(buffer[i] != 47)
                                        i++;
                                i++;
                                if (tienetextura)
                                        this->o[numordact+j]->setsegtex(adquirirdatoi(i,47));
                                while(buffer[i] != 47)
                                        i++;
                                i++;
                                this->o[numordact+j]->setsegnor(adquirirdatoi(i,32));
                                while(buffer[i] != 32)
                                        i++;
                                i++;
                                this->o[numordact+j]->settercver(adquirirdatoi(i,47));
                                while(buffer[i] != 47)
                                        i++;
                                i++;
                                if (tienetextura)
                                        this->o[numordact+j]->setterctex(adquirirdatoi(i,47));
                                while(buffer[i] != 47)
                                        i++;
                                i++;
                                this->o[numordact+j]->settercnor(adquirirdatoi(i,10));
                                this->o[numordact+j]->setnommat(nombremat);
                                i=saltarlinea(i);

                        }
                        numordact=numordact+numord;
                        numord=0;
                }
}
//  Funciones auxiliaries para desplazamiento dentro del buffer
long Cargar01::saltarlinea(long i)
{
        while (buffer[i]!=10)
                i++;
        i++;
        return(i);
}

int Cargar01::saltarlinea2(int j)
{
        while (buffer2[j]!=10)
                j++;
        j++;
        return(j);
}
```

```
float Cargar01::adquirirdatof(long i)
{
	float a;
	long j=0;
	while(buffer[i+j] != 32)
		j++;
	char strn1[20]= "                ";
		for (int k=0;k<20;k++)
			if (k<j)
				strn1[19+k-j]=buffer[i+k];
	a=(float) atof(strn1);
	return(a);
}

float Cargar01::adquirirdato2f(int i)
{
	float a;
	long j=0;
	while(buffer2[i+j] != 32)
		j++;
	char strn1[20]= "                ";
		for (int k=0;k<20;k++)
			if (k<j)
				strn1[19+k-j]=buffer2[i+k];
	a=(float) atof(strn1);
	return(a);
}

int Cargar01::adquirirdatoi(long i,int sig)
{
	int a;
	int j=0;
	while(buffer[i+j] != sig)
		j++;
	char strn1[20]= "                ";
		for (int k=0;k<20;k++)
			if (k<j)
				strn1[19+k-j]=buffer[i+k];
	a=atoi(strn1);
	return(a);
}

// Función para cargar la información del archivo mtl en arreglos de forma estructurada.
void Cargar01::cargarmaterial()
{

	GLfloat ambient[4]={0.2f,0.2f,0.2f,1.0f}, diffuse[4]={1.0f,0.8f,0.0f,1.0f}, specular[4]={1.0f,1.0f,1.0f,1.0f};
	char* nombre;
	char* nombrematerial;
	int i1=0, bandera;
	while (buffer2[i1] == 35)			//Se busca el caracter "tab" para encontrar el primer objeto
		i1=saltarlinea2(i1);

	for(int j=0;j<atributos->consulmateriales();j++)
	{
		while (buffer2[i1] != 32)
			i1++;
		i1++;

		bandera=1;
		while (buffer2[i1+bandera] != 10)
			bandera++;
		nombrematerial = new char[bandera+1];
		for (int k=0; k<bandera;k++)
			nombrematerial[k]=buffer2[i1+k];
		*(nombrematerial+bandera)='\0';
		m[j]->setnombre(nombrematerial);
```

```
                    i1=saltarlinea2(i1);

                    while (buffer2[i1] != 9)                    //Se busca el caracter "tab" para encontrar el primer objeto
                            i1++;
                    i1+=5;
                    m[j]->setprimka(adquirirdato2f(i1));
                    while(buffer2[i1] != 32)
                            i1++;
                    i1++;
                    m[j]->setsegka(adquirirdato2f(i1));
                    while(buffer2[i1] != 32)
                            i1++;
                    i1++;
                    m[j]->settercka(adquirirdato2f(i1));
                    i1=saltarlinea2(i1);
                    i1+=5;
                    m[j]->setprimkd(adquirirdato2f(i1));
                    while(buffer2[i1] != 32)
                            i1++;
                    i1++;
                    m[j]->setsegkd(adquirirdato2f(i1));
                    while(buffer2[i1] != 32)
                            i1++;
                    i1++;
                    m[j]->setterckd(adquirirdato2f(i1));
                    i1=saltarlinea2(i1);
                    i1+=5;
                    m[j]->setprimks(adquirirdato2f(i1));
                    while(buffer2[i1] != 32)
                            i1++;
                    i1++;
                    m[j]->setsegks(adquirirdato2f(i1));
                    while(buffer2[i1] != 32)
                            i1++;
                    i1++;
                    m[j]->settercks(adquirirdato2f(i1));
                    i1=saltarlinea2(i1);
                    i1+=8;
                    m[j]->setillum(adquirirdato2f(i1));
                    i1=saltarlinea2(i1);
                    i1+=5;
                    m[j]->setns(adquirirdato2f(i1));
                    i1=saltarlinea2(i1);

                    if (buffer2[i1] != 110)
                    {
                            i1+=3;
                            while (buffer2[i1] != 32)               //Se busca el caracter "tab" para encontrar el primer
                                                                    //objeto
                                    i1++;
                            i1++;
                            bandera=1;
                            while (buffer2[i1+bandera] != 10)
                                    bandera++;
                            nombre = new char[bandera+1];
                            for (int k=0; k<bandera;k++)
                                    nombre[k]=buffer2[i1+k];
                            *(nombre+bandera)='\0';
                            m[j]->setnombrearchivo(nombre);
                    }
            }
}
```

//Consultores de los arreglos en donde reposa la información de normales, vértices, texturas y materiales.

punto* Cargar01::consultarp(long indice)
{

```
	return (p[indice]);
}

textura* Cargar01::consultart(long indice)
{
	return (t[indice]);
}

normal* Cargar01::consultarn(long indice)
{
	return (n[indice]);
}

orden* Cargar01::consultaro(long indice)
{
	return (o[indice]);
}

material* Cargar01::consultarm(int indice)
{
	return (m[indice]);
}

Objeto* Cargar01::consultaratrib()
{
	return (atributos);
}
```

C. Clase Graficar01

Declaración de la clase Graficar01

```
// Graficar01.h: interface for the Graficar01 class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_GRAFICAR01_H__D3307F36_18B9_4AEA_BA28_4E2D5ACC1B46__INCLUDED_)
#define AFX_GRAFICAR01_H__D3307F36_18B9_4AEA_BA28_4E2D5ACC1B46__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "punto.h"
#include "textura.h"
#include "normal.h"
#include "orden.h"
#include "material.h"
#include "Objeto.h"
#include "argar01.h"
#include "Textura1.h"
#include <GL/glu.h>
#include <GL/gl.h>

class Graficar01
{

private:
	PIXELFORMATDESCRIPTOR pfd;
	punto *eye, *at, *pato;
	Cargar01* archivo;
	HDC hDC;
	HGLRC hRC;
	float altura;
	CTextura **cargadortextura;

public:
	Graficar01();
	virtual ~Graficar01();

	void cicloDibujado();
	void iniciarPixelFormat(void);
	void iniciarOpengl(HWND hWnd);
	void init(void);
	void display();
	void movercamaraadelante(void);
	void movercamaraderecha(void);
	void movercamaraizquierda(void);
	void movercamaraatras(void);
	float normax(float* );
	double ABS(double );
	bool colisiones(int);
	void DisableOpenGL(HWND hWnd);
	void cargartextura(int indice);
	int esmaterial(long numorden);
	float anguloconpato(void);
	float distanciaconpato(void);
	punto* consuleye(void);
	punto* consulat(void);
	punto* consulpato(void);
	void irainicio(void);
	void setpato(float,float);
};

#endif // !defined(AFX_GRAFICAR01_H__D3307F36_18B9_4AEA_BA28_4E2D5ACC1B46__INCLUDED_)
```

Implementación de la clase Graficar01

```
// Graficar01.cpp: implementation of the Graficar01 class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Graficar01.h"

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <math.h>
#include <io.h>
#include <iostream.h>
#include <fcntl.h>
#include <gl/gl.h>
#include <gl/glu.h>

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Graficar01::Graficar01()
{
	altura=20.0f;
	eye = new punto();
	at  = new punto();
	pato = new punto();
	archivo = new Cargar01();

	if(archivo->leer())
	{
		archivo->contar();
		archivo->cargarmaterial();
		archivo->almacenar();
	}
}


Graficar01::~Graficar01()
{
	delete archivo;
	delete eye;
	delete at;
	delete pato;
}

//Iniciar el formato del pixel
void Graficar01::iniciarPixelFormat(void)
{
	int format;

	// set the pixel format for the DC
	ZeroMemory( &pfd, sizeof( pfd ) );
	pfd.nSize = sizeof( pfd );
	pfd.nVersion = 1;
	pfd.dwFlags = PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER;
	pfd.iPixelType = PFD_TYPE_RGBA;
	pfd.cColorBits = 24;
	pfd.cDepthBits = 16;
	pfd.iLayerType = PFD_MAIN_PLANE;
	format = ChoosePixelFormat( hDC, &pfd );
	SetPixelFormat( hDC, format, &pfd );
}
```

```
// Inicialización de los parámetros de OpenGL
void Graficar01::init(void)
{
	glColor3f(0.0,0.0,0.0);
	glClearColor (0.0, 0.0, 0.0, 0.0);
	glShadeModel (GL_SMOOTH);
	glLoadIdentity ();

	GLfloat light_ambient[]={0.5,0.5,0.5,1.0};
	GLfloat light_diffuse[]={1.0,1.0,1.0,1.0};
	GLfloat light_specular[]={1.0,1.0,1.0,1.0};

	GLfloat light0_pos[]={70.0,70.0,50.0,1.0};
	GLfloat light0_dir[]={-1.0,0.0,0.0,0.0};
	glEnable(GL_SMOOTH);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
	glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
	glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
	glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
	GLfloat global_ambient[]={0.7f,0.7f,0.7f,1.0f};
	glLightModelfv(GL_LIGHT_MODEL_AMBIENT,global_ambient);

	pato->setpunto(20.0f,-232.0f,altura);
	glViewport (0, 0, 1890, 1168);
		eye->setpunto(1.0, -5.0, altura);
		at->setpunto(5.0, -5.0, altura);
	glMatrixMode (GL_PROJECTION);
	glLoadIdentity ();
		glFrustum (-1.0, 1.0, -0.9, 0.9, 0.5, 7000.0);
		glRotatef(90,0.0,1.0,0.0);
		glRotatef(90,-1.0,0.0,0.0);
		glTranslatef(-1.0,5.0,-altura);
	glMatrixMode (GL_MODELVIEW);



	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glEnable(GL_DEPTH_TEST);
	glLoadIdentity (); //

	// Dejando listas las texturas
	bool cargobien=false;
	int j=this->archivo->consultaratrib()->consulmateriales();
	cargadortextura=new CTextura*[j];
	for (int i=0;i<j;i++)
	{
		cargadortextura[i]= new CTextura();
	}

	for ( i=0;i<this->archivo->consultaratrib()->consulmateriales();i++)
	{
		if(archivo->consultarm(i)->haytextura())
		{
			if(this->cargadortextura[i]->LeerTextura(archivo->consultarm(i)->consulnombrearchivo())==1)
			{
				cargobien=true;
			}
		}
	}
	CTextura :: HabilitarTexturas();
}

// Se pasan las manijas de la consola para que OpenGL pueda graficar
void Graficar01::iniciarOpengl(HWND hWnd)
```

```
{
	// get the device context (DC)
	hDC = GetDC( hWnd );

	// create and enable the render context (RC)
	hRC = wglCreateContext( hDC );

	//-------------------------------------------------//
	// Iniciar el format de pixeles para el renderizado //
	//-------------------------------------------------//
	this->iniciarPixelFormat();


	// create and enable the render context (RC)
	hRC = wglCreateContext( hDC );
	wglMakeCurrent( hDC, hRC );

	this->init();

}

//Realiza algunas inicializaciones justo antes de comenzar a dibujar
void Graficar01::cicloDibujado()
{

	wglMakeCurrent( hDC, hRC );
	//------------------//
	//  Volver a pintar //
	//------------------//
	glLoadIdentity ();
	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glEnable(GL_DEPTH_TEST);
	glLoadIdentity ();
	glClear( GL_COLOR_BUFFER_BIT );
		float theta = 0.0f;

	this->display();
	wglMakeCurrent( hDC, hRC );
	SwapBuffers( hDC );
}



//Se dibuja toda la información leída del archivo .obj
void Graficar01::display()
{
	int material;
	float  px[3][3];
	bool txton=false;
	char* otro="hola2";
	material=-1;
		for (long j=0;j<archivo->consultaratrib()->consulordenes();j++)
		{
			if (esmaterial(j) != material)
			{
				material=esmaterial(j);
				if (material==2)
					otro=archivo->consultarm(material)->consulnombrearchivo();

				this->cargartextura(material);
				if(archivo->consultarm(material)->haytextura())
				{
					this->cargadortextura[material]->PonerActiva();
					txton=true;
				}
				else
					txton=false;
```

```
                        }


                        px[0][0]=this->archivo->consultarp((archivo->consultaro(j))->consulprimver())->consulpuntox();
                        px[0][1]=this->archivo->consultarp((archivo->consultaro(j))->consulprimver())->consulpuntoy();
                        px[0][2]=this->archivo->consultarp((archivo->consultaro(j))->consulprimver())->consulpuntoz();
                        px[1][0]=this->archivo->consultarp((archivo->consultaro(j))->consulsegver())->consulpuntox();
                        px[1][1]=this->archivo->consultarp((archivo->consultaro(j))->consulsegver())->consulpuntoy();
                        px[1][2]=this->archivo->consultarp((archivo->consultaro(j))->consulsegver())->consulpuntoz();
                        px[2][0]=this->archivo->consultarp((archivo->consultaro(j))->consultercver())->consulpuntox();
                        px[2][1]=this->archivo->consultarp((archivo->consultaro(j))->consultercver())->consulpuntoy();
                        px[2][2]=this->archivo->consultarp((archivo->consultaro(j))->consultercver())->consulpuntoz();

                        glBegin(GL_TRIANGLES);
                                if (txton)
                                        glTexCoord2f(archivo->consultart(archivo->consultaro(j)-
>consulprimtex())->consultex1(),archivo->consultart(archivo->consultaro(j)->consulprimtex())->consultex2());
                                glNormal3f(archivo->consultarn((archivo->consultaro(j))->consulprimnor())-
>consulnormalx(),archivo->consultarn((archivo->consultaro(j))->consulprimnor())->consulnormaly(),archivo-
>consultarn((archivo->consultaro(j))->consulprimnor())->consulnormalz());
                                glVertex3f(px[0][0],px[0][1],px[0][2]);
                                if (txton)
                                        glTexCoord2f(archivo->consultart(archivo->consultaro(j)-
>consulsegtex())->consultex1(),archivo->consultart(archivo->consultaro(j)->consulsegtex())->consultex2());
                                glNormal3f(archivo->consultarn((archivo->consultaro(j))->consulsegnor())-
>consulnormalx(),archivo->consultarn((archivo->consultaro(j))->consulsegnor())->consulnormaly(),archivo-
>consultarn((archivo->consultaro(j))->consulsegnor())->consulnormalz());
                                glVertex3f(px[1][0],px[1][1],px[1][2]);
                                if (txton)
                                        glTexCoord2f(archivo->consultart(archivo->consultaro(j)-
>consulterctex())->consultex1(),archivo->consultart(archivo->consultaro(j)->consulterctex())->consultex2());
                                glNormal3f(archivo->consultarn((archivo->consultaro(j))->consultercnor())-
>consulnormalx(),archivo->consultarn((archivo->consultaro(j))->consultercnor())->consulnormaly(),archivo-
>consultarn((archivo->consultaro(j))->consultercnor())->consulnormalz());
                                glVertex3f(px[2][0],px[2][1],px[2][2]);
                        glEnd();
                }
                glFlush();
}

//Función para mover la cámara en el evento en que el usuario oprima la flecha hacia arriba (avanzar cámara hacia delante)
void Graficar01::movercamaraadelante()
{
        float p[3];
        p[0]=at->consulpuntox()-eye->consulpuntox();
        p[1]=at->consulpuntoy()-eye->consulpuntoy();
        p[2]=at->consulpuntoz()-eye->consulpuntoz();

        glMatrixMode (GL_PROJECTION);
                glTranslatef(eye->consulpuntox(),eye->consulpuntoy(),eye->consulpuntoz());
                glTranslatef(-at->consulpuntox(),-at->consulpuntoy(),-at->consulpuntoz());
        glMatrixMode (GL_MODELVIEW);

        float d=normax(p);

        eye->setpunto(at->consulpuntox(),at->consulpuntoy(),eye->consulpuntoz());
        at->setpunto(at->consulpuntox()+p[0],at->consulpuntoy()+p[1],eye->consulpuntoz());
}

//Función para mover la cámara en el evento en que el usuario oprima la flecha derecha (rotar la cámara hacia la derecha)
void Graficar01::movercamaraderecha()
{
        float p[3],teta;

        p[0]=at->consulpuntox()-eye->consulpuntox();
        p[1]=at->consulpuntoy()-eye->consulpuntoy();
```

```
	p[2]=at->consulpuntoz()-eye->consulpuntoz();
	glMatrixMode (GL_PROJECTION);
		glTranslatef(eye->consulpuntox(),eye->consulpuntoy(),eye->consulpuntoz());
		glRotatef(-15,0.0,0.0,-1.0);
		glTranslatef(-eye->consulpuntox(),-eye->consulpuntoy(),-eye->consulpuntoz());
	glMatrixMode (GL_MODELVIEW);
	float d=normax(p);


	teta=(float)atan2((double) (p[1]) ,(double) (p[0]));
	teta = teta - (3.14159f/12);

	if(d>0.0) for(int i=0;i<2;i++) p[i]/=(d);
	at->setpunto(eye->consulpuntox()+(float) (4*(cos(teta))),eye->consulpuntoy()+(float) (4*(sin(teta))),eye-
>consulpuntoz());
}

//Función para mover la cámara en el evento en que el usuario oprima la flecha izquierda (rotar la cámara hacia la izquierda)
void Graficar01::movercamaraizquierda()
{
	float p[3],teta;

	p[0]=at->consulpuntox()-eye->consulpuntox();
	p[1]=at->consulpuntoy()-eye->consulpuntoy();
	p[2]=at->consulpuntoz()-eye->consulpuntoz();
	glMatrixMode (GL_PROJECTION);
		glTranslatef(eye->consulpuntox(),eye->consulpuntoy(),eye->consulpuntoz());
		glRotatef(15,0.0,0.0,-1.0);
		glTranslatef(-eye->consulpuntox(),-eye->consulpuntoy(),-eye->consulpuntoz());
	glMatrixMode (GL_MODELVIEW);
	float d=normax(p);


	teta=(float)atan2((double) (p[1]) ,(double) (p[0]));
	teta = teta + (3.14159f/12);

	if(d>0.0) for(int i=0;i<2;i++) p[i]/=(d);
	at->setpunto(eye->consulpuntox()+(float) (4*(cos(teta))),eye->consulpuntoy()+(float) (4*(sin(teta))),at-
>consulpuntoz());
}

//Función para mover la cámara en el evento en que el usuario oprima la flecha hacia abajo (retroceder la cámara)
void Graficar01::movercamaraatras()
{
	float p[3];
	p[0]=at->consulpuntox()-eye->consulpuntox();
	p[1]=at->consulpuntoy()-eye->consulpuntoy();
	p[2]=at->consulpuntoz()-eye->consulpuntoz();
	glMatrixMode (GL_PROJECTION);
		glTranslatef(eye->consulpuntox(),eye->consulpuntoy(),eye->consulpuntoz());
		float d=normax(p);
		at->setpunto(eye->consulpuntox(),eye->consulpuntoy(),eye->consulpuntoz());
		eye->setpunto(eye->consulpuntox()-p[0],eye->consulpuntoy()-p[1],eye->consulpuntoz()-p[2]);
		glTranslatef(-eye->consulpuntox(),-eye->consulpuntoy(),-eye->consulpuntoz());
	glMatrixMode (GL_MODELVIEW);

}

//Funciones adicionales, utilerías usadas por las funciones relevantes
float Graficar01::normax(float* p)
{
	float d=0.0f;
	for(int i=0;i<3;i++)  d+=(*(p+i))*(*(p+i));
	d=(float)sqrt(d);
	return d;
}
```

```
//Funciones adicionales, utilerías usadas por las funciones relevantes
double Graficar01::ABS(double n)
{
	if (n<0)
		n=-n;
	if (n>3.14159)
		n-=3.14159;
	return n;
}

//Función para la detección de colisiones
bool Graficar01::colisiones(int modo)
{
	float px[3][3], eyeat[3], ab[3],ab1[3],ca[3],bc[3],ac[3], acxab[3], t,t2,t2a,ta,t2b,tb,t3,t4,t5,t6,intersec[3],
aintersec[3],bintersec[3],cintersec[3];
	float d, disaintersec,disbintersec,discintersec, altura1=10.6f;
	int k;
	eyeat[0]=at->consulpuntox()-eye->consulpuntox();
	eyeat[1]=at->consulpuntoy()-eye->consulpuntoy();
	eyeat[2]=at->consulpuntoz()-eye->consulpuntoz();
	d=normax(eyeat);
	for (k=0;k<3;k++) eyeat[k]/=d;

	for (long j=0;j<archivo->consultaratrib()->consulordenes();j++)
	{

		px[0][0]=archivo->consultarp((archivo->consultaro(j))->consulprimver())->consulpuntox();
		px[0][1]=archivo->consultarp((archivo->consultaro(j))->consulprimver())->consulpuntoy();
		px[0][2]=archivo->consultarp((archivo->consultaro(j))->consulprimver())->consulpuntoz();
		px[1][0]=archivo->consultarp((archivo->consultaro(j))->consulsegver())->consulpuntox();
		px[1][1]=archivo->consultarp((archivo->consultaro(j))->consulsegver())->consulpuntoy();
		px[1][2]=archivo->consultarp((archivo->consultaro(j))->consulsegver())->consulpuntoz();
		px[2][0]=archivo->consultarp((archivo->consultaro(j))->consultercver())->consulpuntox();
		px[2][1]=archivo->consultarp((archivo->consultaro(j))->consultercver())->consulpuntoy();
		px[2][2]=archivo->consultarp((archivo->consultaro(j))->consultercver())->consulpuntoz();

		for (k=0;k<3;k++) ab1[k]=px[1][k]-px[0][k];
		d=normax(ab1);
		for (k=0;k<3;k++) ab[k]=ab1[k];
		for (k=0;k<3;k++) ab[k]/=d;

		for (k=0;k<3;k++) ca[k]=px[0][k]-px[2][k];
		d=normax(ca);
		for (k=0;k<3;k++) ca[k]/=d;

		for (k=0;k<3;k++) ac[k]=px[2][k]-px[0][k];

		for (k=0;k<3;k++) bc[k]=px[2][k]-px[1][k];
		d=normax(bc);
		for (k=0;k<3;k++) bc[k]/=d;

		acxab[0]=ac[1]*ab1[2]-ac[2]*ab1[1];
		acxab[1]=ac[2]*ab1[0]-ac[0]*ab1[2];
		acxab[2]=ac[0]*ab1[1]-ac[1]*ab1[0];

		t=(-(acxab[0]*(eye->consulpuntox()-px[1][0])+acxab[1]*(eye->consulpuntoy()-px[1][1])+acxab[2]*…
		   …(altura1-px[1][2]))/(acxab[0]*eyeat[0]+acxab[1]*eyeat[1]+acxab[2]*eyeat[2]));

		intersec[0]=eye->consulpuntox()+t*eyeat[0];
		intersec[1]=eye->consulpuntoy()+t*eyeat[1];
		intersec[2]=altura1;

		if (modo==0)
		{
			if (t<5.0 && t>0.0)
			{
```

```
for (k=0;k<3;k++) aintersec[k]=intersec[k]-px[0][k];
disaintersec=normax(aintersec);
for (k=0;k<3;k++) aintersec[k]/=disaintersec;

for (k=0;k<3;k++) bintersec[k]=intersec[k]-px[1][k];
disbintersec=normax(bintersec);
for (k=0;k<3;k++) bintersec[k]/=disbintersec;

for (k=0;k<3;k++) cintersec[k]=intersec[k]-px[2][k];
discintersec=normax(cintersec);
for (k=0;k<3;k++) cintersec[k]/=discintersec;

t2=(px[1][0]-px[0][0]+(bc[0]*(px[0][1]-px[1][1]))/bc[1])/…
    …(aintersec[0]-(bc[0]*aintersec[1])/bc[1]);
t3=(px[1][1]-px[0][1]+(bc[1]*(px[0][0]-px[1][0]))/bc[0])/…
   …(aintersec[1]-(bc[1]*aintersec[0])/bc[0]);

t=(px[1][2]-px[0][2]+(bc[2]*(px[0][1]-px[1][1]))/bc[1])/…
  …(aintersec[2]-(bc[2]*aintersec[1])/bc[1]);
t4=(px[1][1]-px[0][1]+(bc[1]*(px[0][2]-px[1][2]))/bc[2])/
    …(aintersec[1]-(bc[1]*aintersec[2])/bc[2]);

t5=(px[1][0]-px[0][0]+(bc[0]*(px[0][2]-px[1][2]))/bc[2])/…
   …(aintersec[0]-(bc[0]*aintersec[2])/bc[2]);
t6=(px[1][2]-px[0][2]+(bc[2]*(px[0][0]-px[1][0]))/bc[0])/…
    …(aintersec[2]-(bc[2]*aintersec[0])/bc[0]);


if ((t>0.0f && disaintersec<=t) || (t2>0.0f && disaintersec<=t2 )|| (t3>0.0f …
  …&& disaintersec<=t3 ) || (t4>0.0f && disaintersec<=t4 ) || (t5>0.0f
  …&& disaintersec<=t5 ) || (t6>0.0f && disaintersec<=t6 ))

{
        t2a=(px[2][1]-px[1][1]+(ca[1]*(px[1][0]-px[2][0]))/ca[0])/…
            …(bintersec[1]-(ca[1]*bintersec[0])/ca[0]);
        ta=(px[2][1]-px[1][1]+(ca[1]*(px[1][2]-px[2][2]))/ca[2])/…
            …(bintersec[1]-(ca[1]*bintersec[2])/ca[2]);
        t3=(px[2][0]-px[1][0]+(ca[0]*(px[1][1]-px[2][1]))/ca[1])/…
            …(bintersec[0]-(ca[0]*bintersec[1])/ca[1]);
        t4=(px[2][2]-px[1][2]+(ca[2]*(px[1][1]-px[2][1]))/ca[1])/…
            …(bintersec[2]-(ca[2]*bintersec[1])/ca[1]);
        t5=(px[2][0]-px[1][0]+(ca[0]*(px[1][2]-px[2][2]))/ca[2])/…
            …(bintersec[0]-(ca[0]*bintersec[2])/ca[2]);
        t6=(px[2][2]-px[1][2]+(ca[2]*(px[1][0]-px[2][0]))/ca[0])/…
            …(bintersec[2]-(ca[2]*bintersec[0])/ca[0]);

        if ((ta>0.0f && disbintersec<=ta) || (t2a>0.0f && disbintersec<=t2a ) ||…
          …(t3>0.0f && disbintersec<=t3 ) || (t4>0.0f && disbintersec<=t4 ) ||…
          …(t5>0.0f && disbintersec<=t5 ) || (t6>0.0f && disbintersec<=t6 ))
        {
                t2b=(px[0][0]-px[2][0]+(ab[0]*(px[2][1]-px[0][1]))/ab[1])/…
                    …(cintersec[0]-(ab[0]*cintersec[1])/ab[1]);
                t3=(px[0][1]-px[2][1]+(ab[1]*(px[2][0]-px[0][0]))/ab[0])/…
                    …(cintersec[1]-(ab[1]*cintersec[0])/ab[0]);

                tb=(px[0][2]-px[2][2]+(ab[2]*(px[2][1]-px[0][1]))/ab[1])/…
                    …(cintersec[2]-(ab[2]*cintersec[1])/ab[1]);
                t4=(px[0][1]-px[2][1]+(ab[1]*(px[2][2]-px[0][2]))/ab[2])/…
                    …(cintersec[1]-(ab[1]*cintersec[2])/ab[2]);

                t5=(px[0][0]-px[2][0]+(ab[0]*(px[2][2]-px[0][2]))/ab[2])/…
                    …(cintersec[0]-(ab[0]*cintersec[2])/ab[2]);
                t6=(px[0][2]-px[2][2]+(ab[2]*(px[2][0]-px[0][0]))/ab[0])/…
                    …(cintersec[2]-(ab[2]*cintersec[0])/ab[0]);
                if ((tb>0.0f && discintersec<=tb) || (t2b>0.0f &&…
                  …discintersec<=t2b )|| (t3>0.0f && discintersec<=t3 ) ||…
                  …(t4>0.0f && discintersec<=t4 ) || (t5>0.0f &&…
```

```
                                                …discintersec<=t5 ) || (t6>0.0f && discintersec<=t6 ))
                                                        return true;
                                        }
                                }
                        }
}
else if (modo==1)
{
        if (t>-5.0 && t<0.0)
        {
                for (k=0;k<3;k++) aintersec[k]=intersec[k]-px[0][k];
                disaintersec=normax(aintersec);
                for (k=0;k<3;k++) aintersec[k]/=disaintersec;

                for (k=0;k<3;k++) bintersec[k]=intersec[k]-px[1][k];
                disbintersec=normax(bintersec);
                for (k=0;k<3;k++) bintersec[k]/=disbintersec;

                for (k=0;k<3;k++) cintersec[k]=intersec[k]-px[2][k];
                discintersec=normax(cintersec);
                for (k=0;k<3;k++) cintersec[k]/=discintersec;

                t2=(px[1][0]-px[0][0]+(bc[0]*(px[0][1]-px[1][1]))/bc[1])/…
                    …(aintersec[0]-(bc[0]*aintersec[1])/bc[1]);
                t3=(px[1][1]-px[0][1]+(bc[1]*(px[0][0]-px[1][0]))/bc[0])/…
                     (aintersec[1]-(bc[1]*aintersec[0])/bc[0]);

                t=(px[1][2]-px[0][2]+(bc[2]*(px[0][1]-px[1][1]))/bc[1])/…
                   …(aintersec[2]-(bc[2]*aintersec[1])/bc[1]);
                t4=(px[1][1]-px[0][1]+(bc[1]*(px[0][2]-px[1][2]))/bc[2])/…
                    …(aintersec[1]-(bc[1]*aintersec[2])/bc[2]);

                t5=(px[1][0]-px[0][0]+(bc[0]*(px[0][2]-px[1][2]))/bc[2])/…
                     …(aintersec[0]-(bc[0]*aintersec[2])/bc[2]);
                t6=(px[1][2]-px[0][2]+(bc[2]*(px[0][0]-px[1][0]))/bc[0])/…
                     …(aintersec[2]-(bc[2]*aintersec[0])/bc[0]);


                if ((t>0.0f && disaintersec<=t) || (t2>0.0f && disaintersec<=t2 )|| (t3>0.0f &&…
                   …disaintersec<=t3 ) || (t4>0.0f && disaintersec<=t4 ) || (t5>0.0f &&
                   …disaintersec<=t5 ) || (t6>0.0f && disaintersec<=t6 ))

                {
                        t2a=(px[2][1]-px[1][1]+(ca[1]*(px[1][0]-px[2][0]))/ca[0])/…
                              …(bintersec[1]-(ca[1]*bintersec[0])/ca[0]);
                        ta=(px[2][1]-px[1][1]+(ca[1]*(px[1][2]-px[2][2]))/ca[2])/…
                             …(bintersec[1]-(ca[1]*bintersec[2])/ca[2]);
                        t3=(px[2][0]-px[1][0]+(ca[0]*(px[1][1]-px[2][1]))/ca[1])/…
                             …(bintersec[0]-(ca[0]*bintersec[1])/ca[1]);
                        t4=(px[2][2]-px[1][2]+(ca[2]*(px[1][1]-px[2][1]))/ca[1])/…
                             …(bintersec[2]-(ca[2]*bintersec[1])/ca[1]);
                        t5=(px[2][0]-px[1][0]+(ca[0]*(px[1][2]-px[2][2]))/ca[2])/…
                             …(bintersec[0]-(ca[0]*bintersec[2])/ca[2]);
                        t6=(px[2][2]-px[1][2]+(ca[2]*(px[1][0]-px[2][0]))/ca[0])/…
                             …(bintersec[2]-(ca[2]*bintersec[0])/ca[0]);

                        if ((ta>0.0f && disbintersec<=ta) || (t2a>0.0f && disbintersec<=t2a ) ||…
                           …(t3>0.0f && disbintersec<=t3 ) || (t4>0.0f && disbintersec<=t4 )||…
                           …(t5>0.0f && disbintersec<=t5 ) || (t6>0.0f && disbintersec<=t6 ))
                        {
                                t2b=(px[0][0]-px[2][0]+(ab[0]*(px[2][1]-px[0][1]))/ab[1])/…
                                      …(cintersec[0]-(ab[0]*cintersec[1])/ab[1]);

                                t3=(px[0][1]-px[2][1]+(ab[1]*(px[2][0]-px[0][0]))/ab[0])/…
                                     …(cintersec[1]-(ab[1]*cintersec[0])/ab[0]);
                                tb=(px[0][2]-px[2][2]+(ab[2]*(px[2][1]-px[0][1]))/ab[1])/…
                                    …(cintersec[2]-(ab[2]*cintersec[1])/ab[1]);
```

```
                                                t4=(px[0][1]-px[2][1]+(ab[1]*(px[2][2]-px[0][2]))/ab[2])/…
                                                    …(cintersec[1]-(ab[1]*cintersec[2])/ab[2]);

                                                t5=(px[0][0]-px[2][0]+(ab[0]*(px[2][2]-px[0][2]))/ab[2])/…
                                                    …(cintersec[0]-(ab[0]*cintersec[2])/ab[2]);
                                                t6=(px[0][2]-px[2][2]+(ab[2]*(px[2][0]-px[0][0]))/ab[0])/…
                                                    …(cintersec[2]-(ab[2]*cintersec[0])/ab[0]);
                                                if ((tb>0.0f && discintersec<=tb) || (t2b>0.0f && discinterse…
                                                  …<=t2b )|| (t3>0.0f && discintersec<=t3 ) || (t4>0.0f &&…
                                                  …discintersec<=t4 ) || (t5>0.0f && discintersec<=t5 ) || …
                                                  …(t6>0.0f && discintersec<=t6 ))
                                                        return true;
                                        }
                                }
                        }
                }
                else
                        return false;
        }
        return false;
}

// Función para deshabilitar OpenGL una vez se cierre la aplicación
void Graficar01::DisableOpenGL(HWND hWnd)
{
        wglMakeCurrent( NULL, NULL );
        wglDeleteContext( hRC );
        ReleaseDC( hWnd, hDC );
}

//Función para colocar una textura determinada antes de graficar el triángulo correspondiente
void Graficar01::cargartextura(int indice)
{
        GLfloat ambient[4]={0.2f,0.2f,0.2f,1.0f}, diffuse[4]={1.0f,0.8f,0.0f,1.0f}, specular[4]={1.0f,1.0f,1.0f,1.0f};


        ambient[0]=archivo->consultarm(indice)->consulprimka();
        ambient[1]=archivo->consultarm(indice)->consulsegka();
        ambient[2]=archivo->consultarm(indice)->consultercka();
        ambient[3]=1;

        diffuse[0]=archivo->consultarm(indice)->consulprimkd();
        diffuse[1]=archivo->consultarm(indice)->consulsegkd();
        diffuse[2]=archivo->consultarm(indice)->consulterckd();
        diffuse[3]=1;

        specular[0]=archivo->consultarm(indice)->consulprimks();
        specular[1]=archivo->consultarm(indice)->consulsegks();
        specular[2]=archivo->consultarm(indice)->consultercks();
        specular[3]=1;



        glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,ambient);
        glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,diffuse);
        glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,specular);
        glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,archivo->consultarm(indice)->consulns());
}

//Función en donde se detecta si es necesario cargar una textura.  En caso de ser necesario se consulta el nombre del
//mapa de bits que es necesario cargar.
int Graficar01::esmaterial(long numorden)
{
        char* nombremat;
        char* nombreobj;
        nombreobj=this->archivo->consultaro(numorden)->consulnommat();
```

```
	for(int i=0;i<this->archivo->consultaratrib()->consulmateriales();i++)
	{

		nombremat=archivo->consultarm(i)->consulnombre();
		int j=0,k=0;
		while(nombremat[j]!='\0')
			j++;
		while(nombreobj[k]!='\0')
			k++;
		if(k<j)
			k=j;
		if(strncmp(nombreobj, nombremat, k)==0)
			return i;
	}
	return -1;
}

//Función que halla el ángulo entre el vector que indica la orientación del usuario y el vector que apunta (desde el usuario)
//hacia la fuente de sonido.
float Graficar01::anguloconpato(void)
{
	float teta,teta1,p[2],teta2;
	p[0]=at->consulpuntox()-eye->consulpuntox();
	p[1]=at->consulpuntoy()-eye->consulpuntoy();


	teta1=(float)atan2((double) (p[1]) ,(double) (p[0]));


	p[0]=pato->consulpuntox()-eye->consulpuntox();
	p[1]=pato->consulpuntoy()-eye->consulpuntoy();

	teta2=(float)atan2((double) (p[1]) ,(double) (p[0]));
	teta=teta1-teta2;
	if (teta<-3.14159)
		teta = (float) (teta + 2*3.14159);
	if (teta>3.14159)
		teta = (float) (teta - 2*3.14159);;
	return teta;
}

// Consultor de la posición de la cámara
punto* Graficar01::consuleye(void)
{
	punto* dos=eye;
	return dos;
}

//Consultor del vector con el cual se encuentra la orientación del usuario (hacia donde está mirando el usuario)
punto* Graficar01::consulat(void)
{
	punto* dos=at;
	return dos;
}

//Consultor de la posición de la fuente de sonido
punto* Graficar01::consulpato(void)
{
	punto* dos=at;
	return dos;
}

//Función que encuentra la distancia entre el usuario (la cámara) y la fuente de sonido
float Graficar01::distanciaconpato(void)
{
	float p[3];
	p[0]=pato->consulpuntox()-eye->consulpuntox();
```

```
	p[1]=pato->consulpuntoy()-eye->consulpuntoy();
	p[2]=pato->consulpuntoz()-eye->consulpuntoz();
	float d=normax(p);
	return d;

}

//Función que devuelve al usuario a la posición de partida (se mueven las coordenadas de la cámara)
void Graficar01::irainicio(void)
{
	eye->setpunto(1.0, -5.0, altura);
	at->setpunto(5.0, -5.0, altura);
	glMatrixMode (GL_PROJECTION);
	glLoadIdentity ();
		glFrustum (-1.0, 1.0, -0.7, 0.7, 0.5, 7000.0);
		glRotatef(90,0.0,1.0,0.0);
		glRotatef(90,-1.0,0.0,0.0);
		glTranslatef(-1.0,5.0,-altura);
	glMatrixMode (GL_MODELVIEW);

	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glEnable(GL_DEPTH_TEST);
	glLoadIdentity ();
	this->cicloDibujado();
}

//Función donde se establecen las coordenadas de la fuente de sonido
void Graficar01::setpato(float a,float b)
{
	pato->setpunto(a,b,altura);
}
```

D. Clase Sonar02

Declaración de la clase Sonar02

```
// Sonar02.h: interface for the Sonar02 class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_SONAR02_H__2DC87B12_627C_4935_8A98_E6D015666710__INCLUDED_)
#define AFX_SONAR02_H__2DC87B12_627C_4935_8A98_E6D015666710__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#include "DSUtil.h"


class Sonar02
{
private:
	LPDIRECTSOUND lpds;
	LPDIRECTSOUNDBUFFER lpdsbufferSonido;
public:
	Sonar02();
	virtual ~Sonar02();
	int CargarDirectSound(char* filename);
	int OnInit(HWND hWnd);
	char* CargarArchivo(float angulo);
	char* CargarArchivoIAD(float angulo);
	void Borrar();
	void OnSound(float distancia);
	void Destroy();
};

#endif // !defined(AFX_SONAR02_H__2DC87B12_627C_4935_8A98_E6D015666710__INCLUDED_)
```

Implementación la clase Sonar02

```
int Sonar02::CargarDirectSound(char* filename)
{
        LPDIRECTSOUNDBUFFER* buffer;
        buffer=&lpdsbufferSonido;

/*Se verifica que el archivo de sonido a cargar posea una estructura de tipo WAVEFORMATEX debido a que DirectSound
solo trabaja con archivo de modulación PCM de tipo wav */

//Se abre el archivo (Handle del archivo)
    HMMIO wavefile;
        wavefile = mmioOpen(filename, 0, MMIO_READ|  MMIO_ALLOCBUF);
        if (wavefile == NULL) return(0);

//Se buscan los datos del wav  (Informacion del bloque padre)
        MMCKINFO parent;
        memset(&parent, 0, sizeof(MMCKINFO));
        parent.fccType = mmioFOURCC('W', 'A', 'V', 'E');
        mmioDescend(wavefile, &parent, 0, MMIO_FINDRIFF);

// Se busca y verifica el formato de los datos (Informacion del bloque)
        MMCKINFO child;
        memset(&child, 0, sizeof(MMCKINFO));
        child.fccType = mmioFOURCC('f', 'm', 't', ' ');
        mmioDescend(wavefile, &child, &parent,0);

// Se lee el formato ( Informacion de formato de sonido, se verifica formato RIFF)
        WAVEFORMATEX wavefmt;
        mmioRead(wavefile, (char*)&wavefmt, sizeof(wavefmt));
        if(wavefmt.wFormatTag != WAVE_FORMAT_PCM) return(0);

// Se busca el chunk de datos
        mmioAscend(wavefile, &child, 0);
        child.ckid = mmioFOURCC('d', 'a', 't', 'a');
        mmioDescend(wavefile, &child, &parent, MMIO_FINDCHUNK);

// ***Se crea un DirectSoundBuffer para contener los datos***
//Se utiliza la estructura DSBUFFERDESC con los parámetros del buffer y apuntador a estructura WAVEFORMATEX.
Además se activa la bandera DSBCAPS_CTRLVOLUME necesaria para el manejo de la atenuación del volume.
Los parámetros del buffer primario se trabajan con los establecidos por defecto: 22kHz, 2 canales y 8 bits por muestra

        DSBUFFERDESC bufdesc;
        memset(&bufdesc, 0, sizeof(DSBUFFERDESC));
        bufdesc.dwSize = sizeof(DSBUFFERDESC);
        bufdesc.dwFlags = DSBCAPS_CTRLVOLUME ;
// Se trabaja el buffer con atributos del archivo wav
        bufdesc.dwBufferBytes = child.cksize;
        bufdesc.lpwfxFormat = &wavefmt;
//Se crea el Buffer
        if(DS_OK != (lpds->CreateSoundBuffer(&bufdesc, &(*buffer), NULL))) return(0);

//***Se escriben los datos del archivo en el buffer que se acaba de crear***
        void *write1 = 0, *write2 = 0;
        unsigned long length1, length2;
//Se bloquea el Buffer (Garantiza estabilidad en el buffer)
        (*buffer)->Lock(0, child.cksize, &write1, &length1, &write2, &length2, 0);
//Se copian los datos en la parte bloqueada del buffer
        if(write1 > 0)          mmioRead(wavefile, (char*)write1, length1);
        if(write2 > 0) mmioRead(wavefile, (char*)write2, length2);
//Se desbloquea el Buffer
        (*buffer)->Unlock(write1, length1, write2, length2);
//Se cierra el archivo de sonido
        mmioClose(wavefile, 0);
        return(1);
        }
```

```
//Se crea el objeto  (cuando el form se inicie)
int Sonar02::OnInit(HWND hWnd)
{
  if ((DirectSoundCreate(0, &lpds, NULL)) != DS_OK) return(0);
  if ((lpds->SetCooperativeLevel(hWnd, DSSCL_NORMAL)) != DS_OK) return(0);
  // Se carga el sonidos
 return(1);
}



//Se reproduce el sonido correspondiente al archivo de un ángulo específico
void Sonar02::OnSound(float distancia)
{
//Se atenua la intensidad del volumen dependiendo de la ubicación de la cámara
        lpdsbufferSonido->SetVolume((long) (-2000*(log10(distancia/10))));
//Se ubica el apuntador en la posición inicial del buffer
        lpdsbufferSonido->SetCurrentPosition(0);
//Se reproduce el sonido
        lpdsbufferSonido->Play(0, 0, 0);
}



// Se liberan los objetos y el buffer de sonido
void Sonar02::Destroy()
{
        lpdsbufferSonido->Release();
        lpds->Release();
}


// Se libera el buffer de sonido
void Sonar02::Borrar()
{
        lpdsbufferSonido->Release();
}



// Función que recibe el ángulo de ubicación de la cámara con respecto a la fuente sobre el plano azimutal y selecciona un
archivo de sonido correspondiente a este ángulo para ser reproducido por el modelo HRTF

char* Sonar02::CargarArchivo(float angulo)
{
        float ang=0;
        char* filename;
        if(angulo==0.0f)
                ang=0;

        if(angulo<0)
                ang=(angulo+2*pi)*180/pi;
        else
                ang=angulo*180/pi;

        if ((ang>=0 && ang<=7.5)||(ang>352.5))
        {
                filename="Duck00.wav";
                return filename;
        }

        if (ang>7.5 && ang<=22.5)
        {
```

```
		filename="Duck15.wav";
		return filename;
}

if (ang>22.5 && ang<=37.5)
{
		filename="Duck30.wav";
		return filename;
}

if (ang>37.5 && ang<=52.5)
{
		filename="Duck45.wav";
		return filename;
}

if (ang>52.5 && ang<=67.5)
{
		filename="Duck60.wav";
		return filename;
}

if (ang>67.5 && ang<=82.5)
{
		filename="Duck75.wav";
		return filename;
}

if (ang>82.5 && ang<=97.5)
{
		filename="Duck90.wav";
		return filename;
}

if (ang>97.5 && ang<=112.5)
{
		filename="Duck105.wav";
		return filename;
}

if (ang>112.5 && ang<=127.5)
{
		filename="Duck120.wav";
		return filename;
}

if (ang>127.5 && ang<=142.5)
{
		filename="Duck135.wav";
		return filename;
}

if (ang>142.5 && ang<=157.5)
{
		filename="Duck150.wav";
		return filename;
}

if (ang>157.5 && ang<=172.5)
{
		filename="Duck165.wav";
		return filename;
}

if (ang>172.5 && ang<=187.5)
{
		filename="Duck180.wav";
```

```
		return filename;
}

if (ang>187.5 && ang<=202.5)
{
		filename="Duck195.wav";
		return filename;
}

if (ang>202.5 && ang<=217.5)
{
		filename="Duck210.wav";
		return filename;
}

if (ang>217.5 && ang<=232.5)
{
		filename="Duck225.wav";
		return filename;
}

if (ang>232.5 && ang<=247.5)
{
		filename="Duck240.wav";
		return filename;
}

if (ang>247.5 && ang<=262.5)
{
		filename="Duck255.wav";
		return filename;
}

if (ang>262.5 && ang<=277.5)
{
		filename="Duck270.wav";
		return filename;
}

if (ang>277.5 && ang<=292.5)
{
		filename="Duck285.wav";
		return filename;
}

if (ang>292.5 && ang<=307.5)
{
		filename="Duck300.wav";
		return filename;
}

if (ang>307.5 && ang<=322.5)
{
		filename="Duck315.wav";
		return filename;
}

if (ang>322.5 && ang<=337.5)
{
		filename="Duck330.wav";
		return filename;
}

if (ang>337.5 && ang<=352.5)
{
		filename="Duck345.wav";
		return filename;
```

```
	}
	return "ninguno";
}
```
// Función que recibe el ángulo de ubicación de la cámara con respecto a la fuente sobre el plano azimutal y selecciona un archivo de sonido correspondiente a este ángulo para ser reproducido por el modelo IAD e ITD

```
char* Sonar02::CargarArchivoIAD(float angulo)
{
	float ang=0;
	char* filename;
	if(angulo==0.0f)
		ang=0;

	if(angulo<0)
		ang=(angulo+2*pi)*180/pi;
	else
		ang=angulo*180/pi;


	if ((ang>=0 && ang<=7.5)||(ang>352.5))
	{
		filename="IDuck00.wav";
		return filename;
	}

	if (ang>7.5 && ang<=22.5)
	{
		filename="IDuck15.wav";
		return filename;
	}

	if (ang>22.5 && ang<=37.5)
	{
		filename="IDuck30.wav";
		return filename;
	}

	if (ang>37.5 && ang<=52.5)
	{
		filename="IDuck45.wav";
		return filename;
	}

	if (ang>52.5 && ang<=67.5)
	{
		filename="IDuck60.wav";
		return filename;
	}

	if (ang>67.5 && ang<=82.5)
	{
		filename="IDuck75.wav";
		return filename;
	}

	if (ang>82.5 && ang<=97.5)
	{
		filename="IDuck90.wav";
		return filename;
	}

	if (ang>97.5 && ang<=112.5)
	{
		filename="IDuck105.wav";
		return filename;
	}
```

```
if (ang>112.5 && ang<=127.5)
{
	filename="IDuck120.wav";
	return filename;
}

if (ang>127.5 && ang<=142.5)
{
	filename="IDuck135.wav";
	return filename;
}

if (ang>142.5 && ang<=157.5)
{
	filename="IDuck150.wav";
	return filename;
}

if (ang>157.5 && ang<=172.5)
{
	filename="IDuck165.wav";
	return filename;
}

if (ang>172.5 && ang<=187.5)
{
	filename="IDuck180.wav";
	return filename;
}

if (ang>187.5 && ang<=202.5)
{
	filename="IDuck195.wav";
	return filename;
}

if (ang>202.5 && ang<=217.5)
{
	filename="IDuck210.wav";
	return filename;
}

if (ang>217.5 && ang<=232.5)
{
	filename="IDuck225.wav";
	return filename;
}

if (ang>232.5 && ang<=247.5)
{
	filename="IDuck240.wav";
	return filename;
}

if (ang>247.5 && ang<=262.5)
{
	filename="IDuck255.wav";
	return filename;
}

if (ang>262.5 && ang<=277.5)
{
	filename="IDuck270.wav";
	return filename;
}

if (ang>277.5 && ang<=292.5)
```

```
	{
		filename="IDuck285.wav";
		return filename;
	}

	if (ang>292.5 && ang<=307.5)
	{
		filename="IDuck300.wav";
		return filename;
	}

	if (ang>307.5 && ang<=322.5)
	{
		filename="IDuck315.wav";
		return filename;
	}

	if (ang>322.5 && ang<=337.5)
	{
		filename="IDuck330.wav";
		return filename;
	}

	if (ang>337.5 && ang<=352.5)
	{
		filename="IDuck345.wav";
		return filename;
	}

	return "ninguno";
}
```

E. Clase Punto

Declaración de la clase Punto

```
// punto.h: interface for the punto class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_PUNTO_H__7C88B5DB_4B7E_4E48_A31F_BB61174067AA__INCLUDED_)
#define AFX_PUNTO_H__7C88B5DB_4B7E_4E48_A31F_BB61174067AA__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class punto
{
	//atributos
	private:
		float x;
		float y;
		float z;
	public:
		//bool newpunto(char[]);
		punto();
		virtual ~punto();
		void setpunto(float v1, float v2, float v3);
		float* consulpunto() ;
		float consulpuntox();
		float consulpuntoy();
		float consulpuntoz();
};

#endif // !defined(AFX_PUNTO_H__7C88B5DB_4B7E_4E48_A31F_BB61174067AA__INCLUDED_)
```

Implementación de la clase Punto

```
// punto.cpp: implementation of the punto class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "punto.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

punto::punto()
{
	x=y=z=0;
}

punto::~punto()
{

}

//Modificador
void punto::setpunto(float v1, float v2, float v3)
{
	x=v1;
	y=v2;
	z=v3;
}

//Consultores
float* punto::consulpunto()
{
	float a[3];
	a[0]=x;
	a[1]=y;
	a[2]=z;
	return(a);
}

float punto::consulpuntox()
{
	float a;
	a=x;
	return(a);
}

float punto::consulpuntoy()
{
	float a;
	a=y;
	return(a);
}

float punto::consulpuntoz()
{
	float a;
	a=z;
	return(a);
}
```

F. Clase Normal

Declaración de la clase Normal

```
// normal.h: interface for the normal class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_NORMAL_H__63C37A44_BBF1_4171_9A48_6DCD94B2E93E__INCLUDED_)
#define AFX_NORMAL_H__63C37A44_BBF1_4171_9A48_6DCD94B2E93E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class normal
{
private:
                        float x;
                        float y;
                        float z;
public:
        normal();
        virtual ~normal();
        void setnormal(float v1, float v2, float v3);
        float* consulnormal() ;
        float consulnormalx();
        float consulnormaly();
        float consulnormalz();
};

#endif // !defined(AFX_NORMAL_H__63C37A44_BBF1_4171_9A48_6DCD94B2E93E__INCLUDED_)
```

Implementación de la clase Normal

```
// normal.cpp: implementation of the normal class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "normal.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

normal::normal()
{
	x=y=z=0;
}

normal::~normal()
{

}

//Modificador
void normal::setnormal(float v1, float v2, float v3)
{
	x=v1;
	y=v2;
	z=v3;
}

//Consultores
float* normal::consulnormal()
{
	float a[3];
	a[0]=x;
	a[1]=y;
	a[2]=z;
	return(a);
}

float normal::consulnormalx()
{
	float a;
	a=x;
	return(a);
}

float normal::consulnormaly()
{
	float a;
	a=y;
	return(a);
}

float normal::consulnormalz()
{
	float a;
	a=z;
	return(a);
}
```

G. Clase Material

Declaración de la clase Material

```
// material.h: interface for the material class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_MATERIAL_H__E5125A79_8630_4DA4_935D_0DAF46A1A239__INCLUDED_)
#define AFX_MATERIAL_H__E5125A79_8630_4DA4_935D_0DAF46A1A239__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class material
{
private:
	float ka1,ka2,ka3;
	float kd1,kd2,kd3;
	float ks1,ks2,ks3;
	float illum,ns;
	char* nombrearchivo;
	char* nombre;
	bool haytext;
public:
	material();
	virtual ~material();
	float consulprimka();
	float consulsegka();
	float consultercka();
	float consulprimkd();
	float consulsegkd();
	float consulterckd();
	float consulprimks();
	float consulsegks();
	float consultercks();
	float consulillum();
	float consulns();
	char* consulnombrearchivo();
	char* consulnombre();
	void setprimka(float v1);
	void setsegka(float v2);
	void settercka(float v3);
	void setprimkd(float n1);
	void setsegkd(float n2);
	void setterckd(float n3);
	void setprimks(float t1);
	void setsegks(float t2);
	void settercks(float t3);
	void setillum(float il);
	void setns(float n);
	void setnombrearchivo(char* nombrea);
	void setnombre(char* nombrea);
	bool haytextura();
};

#endif // !defined(AFX_MATERIAL_H__E5125A79_8630_4DA4_935D_0DAF46A1A239__INCLUDED_)
```

Implementación de la clase Material

```
// material.cpp: implementation of the material class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "material.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

material::material()
{
	ka1=ka2=ka3=kd1=kd2=kd3=ks1=ks2=ks3=0.0;
	nombrearchivo="inicializacion";
	nombre="initipmaterial";
	haytext=false;
}

material::~material()
{

}

//Cosultores
float material::consulprimka()
{
	float a;
	a=ka1;
	return a;
}
float material::consulsegka()
{
	float a;
	a=ka2;
	return a;
}
float material::consultercka()
{
	float a;
	a=ka3;
	return a;
}
float material::consulprimkd()
{
	float a;
	a=kd1;
	return a;
}
float material::consulsegkd()
{
	float a;
	a=kd2;
	return a;
}
float material::consulterckd()
{
	float a;
	a=kd3;
	return a;
}
float material::consulprimks()
{
	float a;
	a=ks1;
```

```
	return a;
}
float material::consulsegks()
{
	float a;
	a=ks2;
	return a;
}
float material::consultercks()
{
	float a;
	a=ks3;
	return a;
}
float material::consulillum()
{
	float a;
	a=illum;
	return a;
}
float material::consulns()
{
	float a;
	a=ns;
	return a;
}
char* material::consulnombrearchivo()
{
	return nombrearchivo;
}
char* material::consulnombre()
{
	return nombre;
}

//Modificadores
void material::setprimka(float v1)
{
	ka1=v1;
}
void material::setsegka(float v2)
{
	ka2=v2;
}
void material::settercka(float v3)
{
	ka3=v3;
}
void material::setprimkd(float n1)
{
	kd1=n1;
}
void material::setsegkd(float n2)
{
	kd2=n2;
}
void material::setterckd(float n3)
{
	kd3=n3-1;
}
void material::setprimks(float t1)
{
	ks1=t1;
}
void material::setsegks(float t2)
{
	ks2=t2;
```

```
}
void material::settercks(float t3)
{
	ks3=t3;
}
void material::setillum(float il)
{
	illum=il;
}
void material::setns(float n)
{
	ns=n;
}

void material::setnombrearchivo(char* nombrea)
{
	nombrearchivo=nombrea;
	haytext=true;
}
void material::setnombre(char* nombrea)
{
	nombre=nombrea;
}

//Devuelve true si es necesario mapear una textura para el triángulo en cuestión
bool material::haytextura()
{
	return	haytext;
}
```

H. Clase Textura

Declaración de la clase Textura

```
// textura.h: interface for the textura class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_TEXTURA_H__0179DCDF_A6CF_4390_84EB_2881873DECC8__INCLUDED_)
#define AFX_TEXTURA_H__0179DCDF_A6CF_4390_84EB_2881873DECC8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class textura
{
private:
        float tex1,tex2;
public:
        textura();
        virtual ~textura();
        float consultex1();
        float consultex2();
        float* consultex();
        void settextura(float t1, float t2);

};

#endif // !defined(AFX_TEXTURA_H__0179DCDF_A6CF_4390_84EB_2881873DECC8__INCLUDED_)
```

Implementación de la clase Textura

```
// textura.cpp: implementation of the textura class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "textura.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

textura::textura()
{
	tex1=tex2=0;
}

textura::~textura()
{

}

//Modificador
void textura::settextura(float t1, float t2)
{
	tex1=t1;
	tex2=t2;
}

//Consultores
float* textura::consultex()
{
	float a[2];
	a[0]=tex1;
	a[1]=tex2;
	return(a);
}

float textura::consultex1()
{
	float a;
	a=tex1;
	return(a);
}

float textura::consultex2()
{
	float a;
	a=tex2;
	return(a);
}
```

I. Clase Orden

Declaración de la clase Orden

```
// orden.h: interface for the orden class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_ORDEN_H__A3015EF7_E787_4E19_B7D3_37ABEBF23602__INCLUDED_)
#define AFX_ORDEN_H__A3015EF7_E787_4E19_B7D3_37ABEBF23602__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class orden
{
private:
	long primvertice, segvertice, tercvertice;
	long primnormal,segnormal,tercnormal;
	long primtext,segtext,terctext;
	char*	nommat;
public:
	orden();
	virtual ~orden();
	long consulprimver();
	long consulsegver();
	long consultercver();
	long consulprimnor();
	long consulsegnor();
	long consultercnor();
	long consulprimtex();
	long consulsegtex();
	long consulterctex();
	char* consulnommat();
	void setprimver(long v1);
	void setsegver(long v2);
	void settercver(long v3);
	void setprimnor(long n1);
	void setsegnor(long n2);
	void settercnor(long n3);
	void setprimtex(long t1);
	void setsegtex(long t2);
	void setterctex(long t3);
	void setnommat(char* nombrea);
};

#endif // !defined(AFX_ORDEN_H__A3015EF7_E787_4E19_B7D3_37ABEBF23602__INCLUDED_)
```

Implementación de la clase Orden

```
// orden.cpp: implementation of the orden class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "orden.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

orden::orden()
{
	primvertice = segvertice = tercvertice = 0;
	primnormal = segnormal = tercnormal = 0;
	primtext = segtext = terctext = 0;
	nommat="initipmaterial";
}

orden::~orden()
{

}

// Consultores
long orden::consulprimver()
{
	long a;
	a=primvertice;
	return a;
}
long orden::consulsegver()
{
	long a;
	a=segvertice;
	return a;
}
long orden::consultercver()
{
	long a;
	a=tercvertice;
	return a;
}
long orden::consulprimnor()
{
	long a;
	a=primnormal;
	return a;
}
long orden::consulsegnor()
{
	long a;
	a=segnormal;
	return a;
}
long orden::consultercnor()
{
	long a;
	a=tercnormal;
	return a;
}
long orden::consulprimtex()
{
	long a;
	a=primtext;
```

```
	return a;
}
long orden::consulsegtex()
{
	long a;
	a=segtext;
	return a;
}
long orden::consulterctex()
{
	long a;
	a=terctext;
	return a;
}
char* orden::consulnommat()
{
	return nommat;
}

// Modificadores
void orden::setprimver(long v1)
{
	primvertice=v1-1;
}
void orden::setsegver(long v2)
{
	segvertice=v2-1;
}
void orden::settercver(long v3)
{
	tercvertice=v3-1;
}
void orden::setprimnor(long n1)
{
	primnormal=n1-1;
}
void orden::setsegnor(long n2)
{
	segnormal=n2-1;
}
void orden::settercnor(long n3)
{
	tercnormal=n3-1;
}
void orden::setprimtex(long t1)
{
	primtext=t1-1;
}
void orden::setsegtex(long t2)
{
	segtext=t2-1;
}
void orden::setterctex(long t3)
{
	terctext=t3-1;
}
void orden::setnommat(char* nombrea)
{
	nommat=nombrea;
}
```

J. Clase Objeto

Declaración de la clase Objeto

```
// Objeto.h: interface for the Objeto class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_OBJETO_H__169CD9C3_84F3_4A99_8302_E394F27A3A35__INCLUDED_)
#define AFX_OBJETO_H__169CD9C3_84F3_4A99_8302_E394F27A3A35__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Objeto
{

private:
	long vertices;
	long normales;
	long texturas;
	long ordenes;
	long objetos;
	long objects;
	int materiales;

public:
	Objeto();
	void setnumvertices(long);
	void setnumobjects(long);
	void setnumnormales(long);
	void setnumtexturas(long);
	void setnumordenes(long);
	void setnummateriales(int);
	long consulvertices(void);
	long consulobjects(void);
	long consulnormales(void);
	long consultexturas(void);
	long consulordenes(void);
	int consulmateriales(void);
	virtual ~Objeto();

};

#endif // !defined(AFX_OBJETO_H__169CD9C3_84F3_4A99_8302_E394F27A3A35__INCLUDED_)
```

Implementación de la clase Objeto

```
// Objeto.cpp: implementation of the Objeto class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Objeto.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

// Constructor y destructor
Objeto::Objeto()
{
	vertices = normales = texturas = ordenes = objects = materiales = 0;
}

Objeto::~Objeto()
{

}
//Modificadores
void Objeto::setnumvertices(long numvert)
{
	vertices = numvert;
}
void Objeto::setnumobjects(long numvert)
{
	objects = numvert;
}
void Objeto::setnumnormales(long numnor)
{
	normales=numnor;
}

void Objeto::setnumordenes(long numord)
{
	ordenes = numord;
}

void Objeto::setnumtexturas(long numtext)
{
	texturas = numtext;
}

void Objeto::setnummateriales(int nummateriales)
{
	materiales = nummateriales;
}

//Consultores
long Objeto::consulvertices(void)
{
	long a = vertices;
	return (a);
}
long Objeto::consulnormales(void)
{
	long a = normales;
	return (a);
}
long Objeto::consultexturas(void)
{
	long a = texturas;
```

```
        return (a);
}
long Objeto::consulordenes(void)
{
        long a = ordenes;
        return (a);
}
long Objeto::consulobjects(void)
{
        long a = objects;
        return (a);
}
int Objeto::consulmateriales(void)
{
        int a = materiales;
        return (a);
}
```

K. Clase CTextura

Declaración de la clase CTextura

```
// Textura1.h: interface for the CTextura class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_TEXTURA1_H__DCF55198_CD77_4543_A66E_5D6517BDFCBA__INCLUDED_)
#define AFX_TEXTURA1_H__DCF55198_CD77_4543_A66E_5D6517BDFCBA__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CTextura
{

public :
        // Constructor
        CTextura(void);

        // Habilita las texturas
        static void HabilitarTexturas(void);

        // Leer la textura desde un archivo BMP de Windows de 24 bits
        // Devuelve 1 si se ha leido la textura
        // Devuelve 0 en otro caso
        int LeerTextura(const char *Archivo);

        // Pone la textura como la activa de OpenGL
        void PonerActiva(void);

        // Destructor
        virtual ~CTextura();

        // Puntero a la imagen de la textura
        unsigned char *Imagen;

        // Tamanyo de la textura
        int TamX,TamY;


};

#endif // !defined(AFX_TEXTURA1_H__DCF55198_CD77_4543_A66E_5D6517BDFCBA__INCLUDED_)
```

Implementación de la clase CTextura

```
// Textura1.cpp: implementation of the CTextura class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Textura1.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
////////////////////////////////////
// IMPLEMENTACION DE LA CLASE CTextura
////////////////////////////////////

#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
#include <stdio.h>

//Constructor de la clase
CTextura :: CTextura(void)
{
	Imagen = NULL;
}

//Función que lee la textura especificada en una cadena de caracteres
int CTextura :: LeerTextura(const char *Textura)
{
	BOOL b = FALSE;

	int namelen = strlen(Textura);
	if ( namelen > 4 )
	{
		char suffix[5];
		strncpy(suffix, Textura + namelen - 4, 4);
		suffix[4] = 0;
		_strupr(suffix);

		// is it a BMP?
		if ( !strcmp(suffix, ".BMP") )
		{
			FILE* file = fopen(Textura, "rb");
			if ( file != NULL )
			{
				BITMAPFILEHEADER fileheader;
				BITMAPINFOHEADER infoheader;

				if ( fread(&fileheader, sizeof(BITMAPFILEHEADER), 1, file) == 1
					&& (char)fileheader.bfType== 'B'&&*(((char*)&fileheader.bfType)+1)== 'M'
					&& fread(&infoheader, sizeof(BITMAPINFOHEADER), 1, file) == 1
					&& infoheader.biBitCount == 24&& infoheader.biCompression==BI_RGB)
				{

					rewind( file );
					long offset = fileheader.bfOffBits;
					fseek(file, offset, SEEK_SET);

					int bufsize = infoheader.biWidth * abs(( int ) infoheader.biHeight) * 3;
					unsigned char* imagebuf = new unsigned char[bufsize];

					if ( (int)fread(imagebuf, 1, bufsize, file) == bufsize )
					{
```

```
                                                // Flip the RGB components
                                                for ( int i = 0; i < bufsize / 3; ++i )
                                                {
                                                        unsigned char c = imagebuf[ i * 3 ];
                                                        imagebuf[ i * 3 ] = imagebuf[ ( i * 3 ) + 2 ];
                                                        imagebuf[ ( i * 3 ) + 2 ] = c;
                                                }

                                                TamX = infoheader.biWidth;
                                                TamY = infoheader.biHeight;
                                                b = TRUE;
                                        }


                                        Imagen = imagebuf;
                                }

                                fclose(file);
                        }
                }
        }

        return b;
}

//Función que pone activa la imagen recién cargada
void CTextura :: PonerActiva(void)
{
        if(Imagen)
                glTexImage2D(GL_TEXTURE_2D,0,3,TamX, TamY,0,GL_RGB,GL_UNSIGNED_BYTE, Imagen);
}

//Función que habilita el mapeo de texturas en OpenGL
void CTextura :: HabilitarTexturas(void)
{
        glPixelStorei(GL_UNPACK_ALIGNMENT,1);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glEnable(GL_TEXTURE_2D);
}

//Función destructura, libera la memoria
CTextura :: ~CTextura()
{
        if(Imagen)
                delete Imagen;
}
```

ANEXO 3: FORMATO DE ENCUESTA

Nombre: _____

Edad: _____ **Sexo:** (M) (F)

1. Se permite que el usuario pruebe el sistema con cada uno de los módulos de sonido con un solo pato.

¿Cuánto tiempo se demoró en encontrar al pato fuente de sonido?

Tiempo Modelo 1 : _____ **Tiempo Modelo 2 :_____**

2. Ahora el usuario prueba el sistema con cada uno de los módulos de sonido pero con múltiples patos.

¿Cuánto tiempo se demoró en encontrar al pato fuente de sonido?

Tiempo Modelo 1 : _____ **Tiempo Modelo 2 :_____**

- ¿El sonido le ayudó a ubicarse y desplazarse dentro del ambiente gráfico?

  Si___ **No___**

- Marque con una X el modelo de espacialización de sonido que le permitió una mejor ubicación espacial.

  ___ Modelo 1. **___ Modelo 2.** **___Cualquiera de los Dos.**

- Califique la concordancia del movimiento de la cámara con el sonido reproducido.

  Modelo 1:

  Excelente___ **Bueno___** **Aceptable___** **Deficiente___** **Malo___**

  Modelo 2:

  Excelente___ **Bueno___** **Aceptable___** **Deficiente___** **Malo___**

ANEXO 4: PRESUPUESTO INICIAL

Recursos Humanos

Ítem	Fuente	Costo Unitario Valor Hora	Cantidad Horas Total	Costo Total
Director de Trabajo de grado	PUJ[11]	$ 45 000	48 (2 horas semanales * 24 semanas)	$ 2 160 000
Asesores (1)	PUJ	$ 35 000	24 (1 hora semanal * 24 semanas)	$ 840 000
Investigadores (2)	Propia	$ 20 000	1200 (2 investigadores * 5 horas diarias * 5 días a la semana * 24 semanas)	$ 24 000 000
Encuestados	Propia	$ 5 000	60	$ 300 000
SUBTOTAL 1				**$ 27 300 000**

Recursos Técnicos

Ítem	Fuente	Costo Unitario	Cantidad	Costo Total
Alquiler de Computador con tarjeta de sonido y licencias de MS Windows 2000, MS Office MS Visual C++ con Open GL, Direct X, 3D Studio y Matlab	PUJ	$95 000 semanales	24	$ 2 280 000
Alquiler de cubículo para trabajo de investigadores.	PUJ	$ 50 000 semanales	24	$ 1 200 000
Audífonos	Propia	$ 80 000	1	$ 80 000
SUBTOTAL 2				**$ 3 560 000**

[11] Pontificia Universidad Javeriana

Otros Costos

Ítem	Fuente	Costo Unitario	Cantidad	Costo Total
Resma de papel tamaño Carta	Propia	$ 8 000	2	$ 16 000
Empaste	Propia	$ 4 000	12	$ 48 000
Cartuchos de impresora	Propia	$90 000	2	$ 180 000
CD's	Propia	$ 2 000	10	$20 000
Fotocopias	Propia	$ 50	400	$20 000
Dispositivo de almacenamiento USB (depreciación)	Propia	$ 25 000	1	$25 000
Energía	Propia	$ 227	500	$113 500
SUBTOTAL 3				**$422 500**

SUBTOTAL 1	$ 27 300 000
SUBTOTAL 2	$ 3 560 000
SUBTOTAL 3	$ 422 500
SUBTOTAL 4	$ 31 282 500
Imprevistos (7%)	$ 2 189 780

Costo TOTAL= $ 33 472 280

ANEXO 5: COSTO FINAL DEL PROYECTO

Recursos Humanos

Ítem	Fuente	Costo Unitario Valor Hora	Cantidad Horas Total	Costo Total
Director de Trabajo de grado	PUJ[12]	$ 45 000	24 (1 hora semanal * 24 semanas)	$ 1 080 000
Asesores (1)	PUJ	$ 35 000	16 (1 hora semanal * 16 semanas)	$ 560 000
Investigadores (2)	Propia	$ 20 000	1440 (2 investigadores * 6 horas diarias *5 días a la semana * 24 semanas)	$ 28 800 000
Encuestados	Propia	$ 5 000	20	$ 100 000
SUBTOTAL 1				**$ 30 540 000**

Recursos Técnicos

Ítem	Fuente	Costo Unitario	Cantidad	Costo Total
Alquiler de Computador con tarjeta de sonido y licencias de MS Windows 2000, MS Office, MS Visual C++ con Open GL, Direct X, 3D Studio y Matlab	PUJ	$95 000 semanales	16	$ 1 520 000
Alquiler de cubículo para trabajo de investigadores.	PUJ	$ 50 000 semanales	16	$ 800 000
Audífonos	Propia	$ 80 000	1	$ 80 000
SUBTOTAL 2				**$ 2 400 000**

[12] Pontificia Universidad Javeriana

Otros Costos

Ítem	Fuente	Costo Unitario	Cantidad	Costo Total
Resma de papel tamaño Carta	Propia	$ 8 000	2	$ 16 000
Empaste	Propia	$ 4 000	12	$ 48 000
Cartuchos de impresora	Propia	$90 000	2	$ 180 000
CD's	Propia	$ 2 000	10	$20 000
Fotocopias	Propia	$ 50	400	$20 000
Dispositivo de almacenamiento USB (depreciación)	Propia	$ 25 000	1	$25 000
Energía	Propia	$ 227	500	$113 500
SUBTOTAL 3				**$422 500**

SUBTOTAL 1	$ 30 540 000
SUBTOTAL 2	$ 2 400 000
SUBTOTAL 3	$ 422 500
TOTAL	**$ 33 362 500**