Sistemas Informáticos

Manipulación y Edición de Sonido Digital

Universidad Complutense de Madrid
Ingeniería en Informática

Profesor
Jaime Sánchez Hernández

Alumnos
Miguel Gómez-Zamalloa Gil
Diego Vico Gómez
Luis María Alonso Martín

Índice

Tabla de contenido

Lista de tablas

Lista de ilustraciones

Introducción

Objetivos

El objetivo de este proyecto ha sido el desarrollo de un entorno gráfico de manipulación y edición de archivos de audio digital sin compresión en formato WAV.

En cuanto a la manipulación, la aplicación permite realizar todas las operaciones básicas que se pueden realizar sobre archivos: abrir, guardar como... etc. El desarrollo de la aplicación como un entorno MDI (*Multiple Document Interface*), capaz de mostrar múltiples ventanas de archivos dentro de un mismo entorno o ventana principal, ha permitido que ésta pueda trabajar hasta con ocho archivos simultáneamente. El entorno de trabajo de cada uno de estos archivos está dentro de su propia ventana, sin embargo es posible realizar operaciones entre archivos como intercambiar o copiar secciones de datos de audio entre los distintos archivos abiertos o incluso mezclar el contenido de los mismos en un nuevo archivo.

La aplicación muestra la representación gráfica del muestreo del sonido de cada archivo. Tiene una función de zoom en la representación que permite visualizar archivos de cualquier duración superior a 15 milisegundos.

Es también capaz de reproducir los archivos al completo o sólo las partes que sean seleccionadas y realizar todas las operaciones propias de un reproductor. Puede incluso reproducir varios archivos a la vez sin problema.

A nivel de edición, se han desarrollado toda una serie de efectos de edición de sonido que se pueden aplicar sobre las secuencias de muestras:

- Amplificación: Se puede amplificar o reducir el volumen de las muestras del archivo desde 0% al 300% y en el caso de sonido estéreo se puede realizar de manera distinta para cada una de los canales.

- Fade: Se pueden introducir efectos de fade-in y fade-out sobre cualquier parte del la secuencia utilizando distintas para ello: lineal, logarítmica, etc.

- Eco: Se puede añadir un efecto de eco al sonido. La intensidad del eco y el retardo del mismo son configurables. También tiene la opción de permitir una retroalimentación del efecto.

- Reverberación: El más importante de todos. Se puede añadir reverberación de una sala a un archivo de audio con toda una serie de parámetros configurables como los coeficientes de los materiales y la atmósfera de la habitación. Una de las principales características de este efecto es que además de ofrecer al usuario una serie de formas de habitación predefinidas también tiene la opción de diseñar sus propias formas de para la sala y situar el emisor y el receptor del sonido dentro de ésta.

Tecnologías de desarrollo

Para el desarrollo del proyecto se ha utilizado el lenguaje C++, más concretamente el entorno de programación Borland C++ Builder 5.

La utilización de este lenguaje se debe a que la aplicación desarrolla grandes cantidades de operaciones e intercambio de datos con archivos en disco durante la ejecución de las operaciones. Debido a estos requisitos, es más conveniente la utilización de lenguajes compilados como C o C++ frente a otros interpretados como Java. Siendo C++ y Java los lenguajes sobre los que el grupo tenía más experiencia y por las razones anteriores se decidió la utilización del lenguaje C++, más en concreto la utilización del entorno de Borland sobre el que ya se había trabajado anteriormente.

Gracias a la programación orientada a objetos de C++ se ha creado un conjunto de clases de la aplicación que permiten trabajar sobre los archivos de forma organizada y permitiendo realizar una escalación en niveles, separando las operaciones a bajo nivel como lectura de muestras del archivo de las operaciones de alto nivel como la aplicación de efectos.

La utilización del lenguaje C++ también permitió la utilización de las librerías gráficas de OpenGL para el trabajo de representación de las muestras de los archivo de audio. Estas librerías están diseñadas en principio para ser compatibles con muchos lenguajes de programación, sin embargo su utilización en el entorno de C++ es rápida y sencilla.

A través de OpenGL, la definición gráfica de la secuencia de muestras es reducida a la especificación de unos vértices sobre un plano 2D.

El funcionamiento de OpenGL está explicado adecuadamente en el Apéndice A.

Capítulo
1

Capítulo 1 : Manipulación de archivos

Introducción

En este capítulo se explicará detalladamente el funcionamiento interno de todas las operaciones básicas de manipulación de archivos. Esta manipulación está dividida en dos categorías claras:

- Manipulación de archivos aislados: Se refiere a las operaciones de apertura y cierre de archivos junto con otras básicas de edición como la eliminación o el silenciado de partes del archivo.

- Manipulación de múltiples archivos: Se refiere a las operaciones que afectan a varios archivos abiertos en el entorno. Estas pueden incluir desde la copia o traslado de partes de un archivo a otro u otros hasta la creación de un mezcla del contenido de varios de ellos.

Todo el manejo de archivos a bajo nivel está implementado sobre las clase *WaveFile* y sus subclases, de modo que la aplicación sólo realiza llamadas del tipo leer_muestras() o escribir_muestras(). Las muestras que se manejan son de tipo `unsigned char` para las de 8 bits y de tipo `short` para las de 16 bits.

El manejo de estos archivos se realiza directamente sobre ellos, ya que cargar en la memoria principal todas las muestras de varios archivos puede saturar la memoria incluso en equipos potentes. Por esta razón casi todas las funciones trabajan sobre un fichero destino distinto del original para mantener este intacto.

Sobre ésta, la clase *WaveFileFunc* contiene toda una serie de funciones estáticas que permiten realizar todas las operaciones anteriores y que trabajan sobre instancias de la clase *WaveFile*.

Manipulación de archivos aislados

Dentro de la manipulación de archivos aislados se encuentran dos tipos de operaciones que aunque son diferentes en su significado no presentan gran diferencia a nivel de operaciones: operaciones sobre archivos y operaciones sobre el contenido de los archivos.

Operaciones sobre archivos

La clase *WaveFileFunc* ofrece funciones para crear archivos, eliminarlos, renombrarlos o copiarlos. La eliminación y el renombrado se basan en rutinas propias de C++ de manejo de archivos. La ejecución de estas operaciones hace desaparecer el archivo original por eliminación o renombramiento.

La función de creación de archivos (*WaveFileFunc.Crear*) tiene los siguientes parámetros:

Parámetro	Tipo	Funcionalidad
`destino`	`AnsiString`	Nombre completo del nuevo archivo a crear
`duracion`	`float`	Duración del nuevo archivo en segundos
`sampleRate`	`int`	Frecuencia de muestreo del archivo
`bitsPerChannel`	`short int`	Número de bits por canal (8 ó 16) del archivo
`channels`	`short int`	Número de canales del archivo

• Tabla 1: Parámetros de la función WaveFileFunc.Crear

Su ejecución se limita a crear una cabecera de un archivo WAV con el formato indicado por los tres parámetros `sampleRate`, `bitsPerChannel` y `channels`. Tras esto crea un array de muestras a cero del tamaño indicado por la duración (`duracion * sampleRate`). En función de si el archivo se desea con 8 o con 16 bits por canal este array de muestras se creará de tipo `unsigned char` (nivel cero de muestras es el valor 128) o de tipo `short` (nivel cero de muestras es el valor 0). Finalmente escribe las muestras en el archivo y lo cierra.

La función de copia de archivos (*WaveFileFunc.Copiar*) tiene los siguientes parámetros:

Parámetro	Tipo	Funcionalidad
`origen`	`AnsiString`	Nombre completo del archivo a copiar
`destino`	`AnsiString`	Nombre completo del nuevo archivo a crear

• Tabla 2: Parámetros de la función WaveFileFunc.Copiar

Esta función es más simple que la anterior, realiza pasos similares de creación de cabecera del fichero, pero la copia de las muestras se realiza a través de funciones proporcionadas por la clase *WaveFile*.

Operaciones sobre el contenido de los archivos

En esta categoría se engloban las operaciones con silencios (inserción o sobrescritura), la eliminación de muestras y la función de remuestreo. Todas funciones trabajan creando un archivo nuevo, por lo que el archivo original nunca se perderá.

La operación de insertar un silencio (*WaveFileFunc.InsertarSilencio*) de una determinada duración tiene los siguientes parámetros:

Parámetro	Tipo	Funcionalidad
`origen`	`AnsiString`	Nombre completo del archivo original
`destino`	`AnsiString`	Nombre completo del nuevo archivo a crear
`ini`	`unsigned long int`	Muestra a partir del cual se va a insertar el silencio
`count`	`unsigned long int`	Número de muestras silencio que se van a insertar en el archivo

• Tabla 3: Parámetros de la función WaveFileFunc.InsertarSilencio

Esta función crea un archivo en blanco con el mismo formato que el original. Utilizando un buffer, del tipo adecuado (`unsigned char` - 8 bits ó `short` - 16 bits) y de tamaño suficiente, lee las `ini` primeras muestras del original y las escribe en el destino, luego escribe en el destino `count` muestras silencio y finalmente escribe el resto del archivo original en el destino y lo cierra.

La operación de silenciar (*WaveFileFunc.SilenciarSamples*) un determinado fragmento de un archivo tiene los siguientes parámetros:

Parámetro	Tipo	Funcionalidad
`origen`	`AnsiString`	Nombre completo del archivo original
`destino`	`AnsiString`	Nombre completo del nuevo archivo a crear
`count`	`unsigned long int`	Número de muestras silencio que se van a insertar sobrescribiendo el archivo
`ini`	`unsigned long int`	Muestra a partir del cual se va a insertar el silencio

• Tabla 4: Parámetros de la función WaveFileFunc.SilenciarSamples

Esta función opera de manera diferente a la anterior, primero crea una copia con todo el contenido del original y luego con el buffer sobrescribe las muestras originales con silencios. Para ello sólo hace falta desplazar el puntero de escritura en el archivo destino al punto indicado (`ini`) y volver a llama a la función de escritura. Esta manera de hacerlo requiere al final una corrección en el tamaño almacenado en la cabecera del archivo reduciéndolo ya que la función de escritura de *WaveFile* siempre asume inserción y no sobrescritura por lo que incrementa el tamaño de la sección de datos en la cabecera aunque físicamente no este ocurriendo así.

La función de eliminación de muestras (*WaveFileFunc.EliminarSamples*) tiene los siguientes parámetros:

Parámetro	Tipo	Funcionalidad
`origen`	`AnsiString`	Nombre completo del archivo original
`destino`	`AnsiString`	Nombre completo del nuevo archivo a crear
`ini`	`unsigned long int`	Muestra a partir del cual se va a realizar la eliminación
`count`	`unsigned long int`	Número de muestras que se van a eliminar del archivo

• Tabla 5: Parámetros de la función WaveFileFunc.EliminarSamples

El trabajo realizado en este caso es muy similar al de la función de insertar silencios. Realiza los mismos pasos excepto la inserción del silencio leyendo del original sólo las partes que interesa conservar en el archivo de destino.

La función más compleja de esta categoría es la función de remuestreo. Esta operación calcula a partir de la secuencia de muestras del archivo original otra secuencia de muestras con valores distintos de frecuencia de muestreo, bits por canal y número de canales y que sea lo más parecida posible a la original.

La función de remuestreo (*WaveFileFunc.CambiarFormato*) presenta los siguientes parámetros:

Parámetro	Tipo	Funcionalidad
`origen`	`AnsiString`	Nombre completo del archivo original
`destino`	`AnsiString`	Nombre completo del nuevo archivo a crear
`sampleRate`	`int`	Frecuencia de muestreo del archivo
`bitsPerChannel`	`short int`	Número de bits por canal (8 ó 16) del archivo
`channels`	`short int`	Número de canales del archivo

• Tabla 6: Parámetros de la función WaveFileFunc.CambiarFormato

El trabajo de esta operación se divide en secciones: Primero se convierte la secuencia de muestras que se ha leído del original transformando muestras de 8 bits (`unsigned char`) a 16 bits (`short`) mediante la fórmula X = (X-128)*256 o de 16 bits a 8 bits con la fórmula X = (X/256)+128. Simultáneamente se convierten de

estéreo a mono calculando la media de las muestras de cada canal o a la inversa copiando el mismo valor de muestra para los dos canales.

Tras estas operaciones se realiza el cambio de frecuencia de muestreo. Si la frecuencia final (f1) es mayor que la original (f2) el valor de cada muestra original (tras las transformaciones anteriores) se repite f1/f2 veces en la secuencia destino. Si ocurre al contrario, y es la frecuencia original la mayor, cada muestra de la secuencia final se obtiene calculando la media de f2/f1 muestras originales.

Finalmente se realiza la escritura de la secuencia de muestras obtenida en el archivo de destino y se cierra.

Dado que f1/f2 o f2/f1 son divisiones enteras, el único caso en el que no se va a producir una degradación del sonido es el caso de que f1 sea múltiplo de f2 o viceversa como ocurre en el caso de: 11025Hz, 22050Hz, 32075Hz, 44100Hz. Si tenemos en cuenta los tres parámetros que se modifican, es difícil conseguir que el cambio no se aprecie en el resultado, algo que sólo ocurrirá cuando se amplíen los valores de los parámetros y por tanto el tamaño del archivo.

Manipulación de múltiples archivos

En este campo se incluyen todas las operaciones que implican la participación de varios archivos de audio. Estas incluyen las operaciones básicas de copiar, cortar y pegar y la función de mezcla de archivos con todas sus opciones.

Para las funciones de copiar, pegar y cortar se ha tenido que utilizar sobrecarga de funciones ya que transmiten las muestras entre el archivo y el entorno a través de arrays de muestras que tienen tipo diferente según si son de 8 ó 16 bits. Por esta razón hay dos declaraciones de cada una.

Las funciones de copia (*WaveFileFunc.CopiarSamples*) tienen como objetivo leer del archivo original un conjunto de muestras que transmitirá al entorno para que se pueda introducir en otros archivos. Esta función tiene los siguientes parámetros:

Parámetro	Tipo	Funcionalidad
`origen`	`AnsiString`	Nombre completo del archivo original
`samples`	`unsigned char*` `short*`	Array de muestras donde se van a almacenar los datos leídos del archivo
`count`	`unsigned long int`	Número de muestras que se van a copiar del archivo
`inicio`	`unsigned long int`	Muestra a partir del cual se va a realizar la copia
`sampleRate`	`int&`	Devuelve la frecuencia de muestreo de la secuencia de muestras copiada
`bitsPerChannel`	`short int&`	Devuelve el número de bits por canal (8 ó 16) de la secuencia de muestras copiada
`channels`	`short int&`	Devuelve el número de canales de la secuencia de muestras copiada

• Tabla 7: Parámetros de las funciones WaveFileFunc.CopiarSamples

La operación se realiza sencillamente, basta con situar el puntero de lectura del archivo original en el comienzo de la sección de interés y copiar las muestras en el array pasado por parámetro. También devuelve los datos del formato del fragmento copiado para que el entorno pueda hacer un remuestreo de los datos si se necesita copiar en otro archivo con distinto formato.

Las operaciones de corte (*WaveFileFunc.CortarSamples*) tienen los siguientes parámetros:

Parámetro	Tipo	Funcionalidad
origen	AnsiString	Nombre completo del archivo original
destino	AnsiString	Nombre completo del archivo resultante
samples	unsigned char* short*	Array de muestras donde se van a almacenar los datos leídos del archivo
count	unsigned long int	Número de muestras que se van a copiar del archivo
inicio	unsigned long int	Muestra a partir del cual se va a realizar la copia
sampleRate	int&	Devuelve la frecuencia de muestreo de la secuencia de muestras copiada
bitsPerChannel	short int&	Devuelve el número de bits por canal (8 ó 16) de la secuencia de muestras copiada
channels	short int&	Devuelve el número de canales de la secuencia de muestras copiada

• Tabla 8: Parámetros de las funciones WaveFileFunc.CortarSamples

La operación es similar a la anterior, pero en esta se modifica el archivo creando uno nuevo sin las muestras copiadas. Para ello se leen primero las muestras a copiar en el array del entorno y se le da valor a los parámetros de salida. Tras esto, se resetea el archivo de origen y, a través de buffer, se escriben sólo las secciones restantes en el archivo de destino de manera similar a la utilizada en la función de eliminación de muestras.

Las funciones para pegar segmentos (*WaveFileFunc.PegarSamples*) realizan la operación opuesta a las anteriores, ya que reciben un array de muestras del entorno que tienen que insertar sobrescribiendo una parte del archivo original. Los parámetros de estas funciones son:

Parámetro	Tipo	Funcionalidad
origen	AnsiString	Nombre completo del archivo original
destino	AnsiString	Nombre completo del archivo resultante
samples	unsigned char* short*	Array de muestras donde se almacenan los datos a escribir sobre el archivo
count	unsigned long int	Número de muestras que se van a copiar sobre el archivo
sel	unsigned long int	Número de muestras que se van a sobrescirbir del archivo original
inicio	unsigned long int	Muestra a partir del cual se va a realizar la copia
sampleRate	int&	Frecuencia de muestreo de la secuencia de muestras copiada
bitsPerChannel	short int&	Número de bits por canal (8 ó 16) de la secuencia de muestras copiada
channels	short int&	Número de canales de la secuencia de muestras copiada

• Tabla 9: Parámetros de las funciones WaveFileFunc.PegarSamples

Esta función realiza primero un remuestreo de la secuencia de muestras que recibe como parámetro, para ello crea una archivo auxiliar donde escribe las muestras recibidas con el formato de las mismas. Sobre este archivo auxiliar aplica la función *WaveFileFunc.CambiarFormato* explicada anteriormente creando otro archivo auxiliar con el contenido remuestreado para que se corresponda con el formato del archivo de destino. Después de esto el resto es sencillo, se lee del original las muestras anteriores a la parte a escribir y se escriben sobre el archivo destino, luego se escribe el contenido del archivo auxiliar y finalmente las muestras que faltaban del original.

La función de mezcla de archivos (*WaveFileFunc.Mezclar*) realiza la combinación de un conjunto de archivos originales en un archivo destino. El contenido de los originales no va a tener el mismo peso en el resultado para todos ellos, para cada uno se recibe un volumen relativo (entre 0 y 100) de modo que la suma de todos ellos sea 100. Esto permite introducir sonidos de fondo de menor intensidad que el principal. Los parámetros de esta función son:

Parámetro	Tipo	Funcionalidad
`origen`	`AnsiString*`	Array de nombres completos de archivos fuente
`destino`	`AnsiString`	Nombre completo del archivo resultante
`vol`	`int*`	Array de volumenes relativos de los archivos fuente volumen de archivo[i] = vol[i]
`nfuentes`	`int`	Número de archivos fuente
`sampleRate`	`int&`	Frecuencia de muestreo del archivo resultante
`bitsPerChannel`	`short int&`	Número de bits por canal (8 ó 16) del archivo resultante
`channels`	`short int&`	Número de canales del archivo resultante

• Tabla 10: Parámetros de la función WaveFileFunc.Mezclar

Esta función realiza primero un cambio de formato de los archivos originales al del archivo de destino creando un array de archivos auxiliares remuestreados. Luego utiliza un array de buffers para leer las muestras de cada archivo auxiliar calculando simultáneamente el número máximo de muestras de los archivos en `maxsamples`. Después de esto escribe una por una en un buffer de salida el resultado de la siguiente operación:

$$\sum_{j=0}^{\max samples-1} fuentes_j \cdot \frac{vol_j}{100}$$

Cuando j ha superado el número de muestras del archivo fuente el valor de $fuente_j$ es el nivel cero (128 para 8 bits o 0 para 16 bits). Si todo se ha realizado correctamente se realiza la escritura de las muestras en el archivo de destino y se eliminan todos los auxiliares.

La función de mezcla se suele utilizar con una función auxiliar (*WaveFileFunc.Bucle*) que crea un nuevo archivo a partir del archivo original con un silencio inicial especificado y en el que el contenido del original se repite n veces con silencios intermedios. Esto permite crear sonidos de base repetitivos que se pueden mezclar con otros archivos. Esta función tiene los siguientes parámetros:

Parámetro	Tipo	Funcionalidad
`origen`	`AnsiString`	Nombre completo del archivo original
`destino`	`AnsiString`	Nombre completo del archivo resultante
`duracion`	`float`	Duración en segundos del bucle de repetición
`intervalo`	`float`	Duración en segundos entre repeticiones
`inicial`	`float`	Duración del silencio inicial

• Tabla 11: Parámetros de la función WaveFileFunc.Bucle

La duración total del archivo resultante es `duracion` + `inicial`. El formato es el del archivo original. Primero, sobre el archivo destino, se escriben n muestras silencio (n = segundos de silencio inicial * frecuencia) y después se utiliza un bucle que escribe repetidamente el contenido del archivo original y m muestras silencio (m = segundos de silencio inicial * frecuencia) hasta alcanzar la duración indicada por `duracion`.

Capítulo
2

Capítulo 2 : Representación gráfica

Introducción

En este capítulo se explicará el modo en que se realiza la representación gráfica de la secuencia de muestras de un archivo en un formulario. Para este fin se utiliza la librería gráfica OpenGL que permite reducir esta representación al cálculo de vértices en un plano. El grueso de la comunicación con el hardware gráfico a bajo nivel se realiza por el motor de OpenGL y no es visible en código.

Estos cálculos se pueden realizar de manera sencilla simplemente estableciendo una relación entre cada muestra y un punto del plano, sin embargo, dado el gran número de muestras que contiene un archivo relativamente corto (un archivo mono de 8 bits y frecuencia 11025Hz de 1,5 segundos tiene unas 16500 muestras) el hacerlo de esta manera consumiría mucho tiempo y recursos para representarlas con una precisión que no se podría llegar a apreciar. Por esta razón se necesitan toda una serie de cálculos previos que permitan aproximar la representación a la realidad pero utilizando un número muy inferior de puntos del plano.

Todo el trabajo de cálculo de vértices y representación de los mismos se ha cargado en la clase *Visualizador* que trabaja con formularios *TWAVfileForm* que son los que incluyen el contexto de representación de OpenGL. Esta clase no sólo calcula los vértices sino que los formatea visualmente añadiendo escalas de valores y permitiendo operaciones de zoom, etc.

A la hora de manejar la visualización aparecen también otras clases relacionadas con OpenGL como *VControl* que contiene un control gráfico de la representación. Estas clases se explicarán en el capítulo explicativo del entorno de la aplicación al estar más relacionado con esto que con la propia representación.

Cálculo de vértices

En el momento en el que el entorno abre o crea un nuevo fichero se crea para él una nueva instancia de la clase *Visualizador* a la que se asocia, a través de la función *Visualizador.setWaveFile,* un puntero a un objeto *WaveFile* que es el que va a llevar a cabo la interacción con el archivo a bajo nivel.

Después de esto se abre un formulario de visualización que durante su creación (*TWAVfileForm.FormCreate*) realiza intercambia información con el visualizador sobre el tamaño de la pantalla y el tamaño del archivo y finalmente comienza el proceso de cálculo de vértices cuando se realiza la llamada a *TWAVfileForm.setCampoThread* que muestra una ventana de información de proceso mientras se crea otro hilo de ejecución que realiza los cálculos.

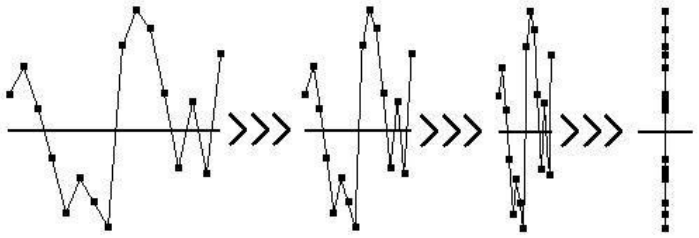
Campo de visualización

El campo de visualización es el plano 2D sobre el que se van a representar los vértices. Este campo se establece a través de la función *Visualizador.setCampo* que recibe como parámetros las dimensiones del mismo: el vértice inferior izquierdo (XL,YL) y el vértice superior derecho (XR,YR).

Estas dimensiones, aunque son proporcionales a las del formulario, son valores genéricos que sólo tienen utilidad para OpenGL. De este modo, aunque el ancho (XR – XL) se corresponde con el número de píxeles de anchura de la visualización el alto (YR – YL) no guarda ninguna relación con la ventana y se fija en un valor de 2*130*número de canales del archivo (los archivos estéreo muestran un canal sobre el otro). Se elige el valor 2*130 = 260 porque se van a distinguir 256 niveles distintos para las muestras, esto se corresponde con los valores de 8 bits mientras que los de 16 (65536 niveles) se redondean dividiendo su valor entre 256.

El ancho en píxeles de la representación marca el número de vértices que es necesario calcular para obtener una representación adecuada, ya que un número mayor provoca la superposición de los puntos al ser un píxel una unidad indivisible.

Se utilizan dos vértices por cada línea vertical de un píxel, uno superior y otro inferior formando una línea. Esto es así porque a cada línea se le hace corresponder un intervalo de la secuencia de muestras y la compresión de la línea que une los valores del intervalo de muestras a una anchura de un único píxel de lugar a una línea que une los valores máximo y mínimo de las muestras en el intervalo. Este fenómeno se aprecia en la siguiente figura:



• Ilustración 1: Compresión gráfica de un intervalo de muestras

Para realizar estas aproximaciones, la función *Visualizador.setCampo* realiza dos bucles anidados que recorren N intervalos (continua mientras haya muestras por leer en el archivo) para cada uno de los cuales calcula el máximo y el mínimo de sus M muestras (N = número de muestras / número de píxeles de anchura).

El principal problema de esta operación es la lectura de las muestras del disco que consume gran cantidad de tiempo. Para mejorar esto se utiliza para la lectura de las muestras la función *Visualizador.readSamples* que utiliza dos buffer de lectura (uno para cada canal si los tiene) para leer simultáneamente del archivo un número de muestras que se le pasa por parámetro. Esta operación mejora significativamente el rendimiento de la lectura con respecto a la utilizada en las versiones iniciales que leía las muestras una a una (*Visualizador.readSample*). La velocidad de ejecución varía según el tamaño de los buffers que se indique pero tras realizar varias pruebas se ha

llegado a la conclusión de que utilizando el tamaño número de muestras / número de píxeles de anchura se consigue un rendimiento adecuado para todo tipo de archivos, habiendo probado desde archivos de pocos KB a grandes archivos de 40 ó 50 MB.

Todas estas operaciones para la simplificación de la visualización no son necesarias para archivos extremadamente pequeños en los que el número de muestras no supera el ancho en píxeles del campo de visualización. En este caso no se realiza operación alguna y se obtienen vértices de cada una de las muestras.

Listas de vértices

La visualización requiere realizar el refresco de la pantalla continuamente y OpenGL necesita que se le pasen los valores de los vértices en cada refresco. Calcular los vértices cada vez que se refresca la pantalla requiere una velocidad de lectura del disco que no soportan los equipos normales. Por esta razón es necesario calcular los valores en momentos interesantes y conservarlos en memoria principal para poder realizar una lectura rápida en cada refresco.

Los vértices calculados con la función anterior sólo se calculan en momentos en los que se ha modificado el archivo y por tanto la visualización ya no se corresponde con la realidad del archivo, cuando el campo de visualización cambia de tamaño, por ejemplo al cambiar el tamaño de la ventana contenedora, o cuando cambia el rango de muestras a visualizar. Estas situaciones normalmente se dan con el suficiente intervalo de tiempo entre ellas para que no se vea afectado el trabajo.

Para almacenar los valores se utilizan una o dos (una para las muestras de cada canal del archivo) listas enlazadas de tipo *ListaPicos*. Estas listas utilizan tres punteros: uno al inicio de la lista, otro al final y uno desplazable que se utiliza para las lecturas secuenciales. Con estos tres punteros se han diseñado funciones de lectura capaces de recuperar muchas muestras en poco tiempo mejorando el rendimiento en el refresco de la pantalla. En cada nodo de la lista se almacena un valor, si el tamaño del archivo es grande se almacenan uno tras otro l valor máximo y mínimo de cada píxel horizontal junto con el número de secuencia horizontal de los valores y en el caso de archivos pequeños los valores de cada muestra junto con el número de secuencia horizontal de este valor.

Dado que sólo se almacenan, como máximo, cuatro valores por píxel y los monitores normales no suelen superar resoluciones con más de 2048 píxeles de anchura la carga en memoria principal por cada archivo abierto es pequeña, sin llegar siquiera al orden del MB. De este modo, aún cuando la aplicación este trabajando simultáneamente con ocho (número máximo de archivos que se pueden abrir en el entorno) archivos de gran tamaño, la ocupación de memoria principal dedicada a este tema será pequeña y no debería afectar al rendimiento del sistema.

Rango de muestras

La visualización también incluye la posibilidad de mostrar únicamente secciones interesantes del archivo, a esto se le denomina cambio del rango de muestras a visualizar. Esta operación es realizada por la función *Visualizador.setRango* que se limita a cambiar los valores de tres variables internas que indican al visualizador la muestra inicial y la final que se visualizan junto con el número máximo de muestras del archivo.

La función recibe como parámetro un valor entero definido en la propia clase que indica la opción a realizar con el rango de muestras: mostrar todo el archivo, ampliar,

reducir, ampliar la parte seleccionada, mover a derecha o izquierda y mover con el control gráfico. En cada caso cambia los valores de las variables de manera distinta cuidando de no salirse del rango de muestras del archivo y de cumplir las normas impuestas a la representación (no se representas secuencias inferiores a 15 milisegundos, las ampliaciones se realizan reduciendo 1/10 el rango original y las reducciones ampliando 1/9 el rango)

Esta función no realiza en ningún momento el recalculo de los vértices de representación, por lo que cualquier llamada a la función *Visualizador.setRango* ha de seguirse forzosamente por otra llamada a la función *Visualizador.setCampo*.

Representación

En este campo se explica los mecanismos que es necesario utilizar para a partir de las listas de vértices obtenidas llevar a cabo la representación gráfica utilizando OpenGL.

A la hora de representar gráficamente las muestras también se utiliza OpenGL para la representar en pantalla otros elementos necesarios como una escala numérica tanto para los valores de las muestras como para los valores de tiempo.

También se explican los mecanismos que se han utilizado para intentar hacer la aplicación lo más portable posible intentando corregir los errores más frecuentes que aparecen por la utilización de ciertas tarjetas gráficas.

Toda la labor de representación, incluyendo todo lo mencionado anteriormente, se realiza en la función *Visualizador.draw* y todas las funciones auxiliares que utiliza (*Visualizador.draw*XX*)*. Esta función es invocada por el formulario de representación cada vez que en este se realiza una llamada a *TWAVfileForm.GLScene*, por ejemplo en cada refresco de ventana. Esta última es la función principal en esta utilización de OpenGL ya que contiene todas las llamadas a funciones con código de representación de OpenGL.

Representación de las muestras

La función *Visualizador.draw* lee de cada lista existente (una para archivos mono o dos para archivos estéreo) el valor máximo y mínimo para cada píxel horizontal y el número de secuencia horizontal y realiza la conversión de este último a una coordenada horizontal real del campo de visualización a través de la función *Visualizador.convertX*. Este último paso no sería necesario para archivos grandes ya que el número de secuencia horizontal almacenado en la lista se correspondería con la coordenada horizontal del campo en ambos valores, pero para archivos pequeños no se almacenan máximos y mínimos sino valores de muestras, por lo que los dos valores leídos tendrán dos números de secuencia distintos y por tanto dos coordenadas distintas. Esto se hace de esta modo para ahorrar código innecesario y en el caso de archivos pequeños se debe entender como la lectura de dos valores consecutivos, no como la lectura del máximo y del mínimo.

Tras realizar todos estos pasos se han obtenido las coordenadas de dos vértices para cada canal del campo de visualización. Estos vértices se pasan al sistema de OpenGL a través de la instrucción `glVertex2f()`. OpenGL unirá todos estos vértices con una línea al estar todas las instrucciones `glVertex2f()` dentro de un bloque `glBegin(GL_LINE_STRIP)..glEnd()`.

Representación de elementos adicionales

Como elementos adicionales de la visualización se entienden las escalas de valores, la vertical para los niveles de las muestras y la horizontal con valores temporales adecuados para el rango de representación mostrado. También se incluyen en este campo a la representación de secciones seleccionadas mediante el resalte con otro color de la misma y la representación de la marca de reproducción que indica que punto del archivo que se está reproduciendo en ese instante.

La escalas utilizan campos de visualización distintos a las representaciones de las muestras y por tanto sus coordenadas no estarán relacionadas. No se debe confundir el campo de visualización que se puede utilizar en cada caso concreto y que se configura con la instrucción `gluOrtho2D()` con la pantalla de visualización o puerto de vista, que sí esta relacionada en coordenadas con el formulario contenedor y en donde se colocan los distintos campos que pueda haber con la instrucción `glViewport()`.

Las dos escalas introducen marcas equidistantes inamovibles durante toda la representación, lo único que cambia son los valores asociados a estas marcas. En el caso de la escala vertical existen dos posibilidades, indicar el volumen relativo en porcentaje ([-100%..100%] siendo 0% el silencio) o indicar el nivel real de las muestras ([0..255] para 8 bits y [0..65535] para 16 bits). En ambos casos se utiliza la función *Visualizador.drawN* que dibuja valores entre 0 y 99999 en OpenGL (internamente utiliza la función *Visualizador.drawD* que dibuja dígitos o algunos caracteres predefinidos). Para la escala vertical utiliza una única llamada a la función *Visualizador.drawET* que dibuja los tiempos de toda la escala, internamente utiliza la función *Visualizador.drawT* que recibe como parámetro un número de muestra y calcula el tiempo correspondiente a ese número en el archivo y lo dibuja adecuadamente formateado (si son minutos no llega a la precisión de milisegundos y si son milisegundos no dibuja los minutos)).

El resalte de las partes seleccionadas se basa en dos coordenadas horizontales que recibe como parámetro y que utiliza para cambiar el color del fondo entre ellas dibujando un rectángulo de otro color tras el resto de dibujos. De manera similar se introduce la marca de tiempo de reproducción que recibe una coordenada horizontal sobre la que dibuja una línea.

Portabilidad

Ante los problemas aparecidos con algunas tarjetas gráficas se han configurado dos opciones de video opcionales: el blending y el suavizado de polígonos. Ambas son utilizadas para ofrecer un aspecto mas conseguido a la representación gráfica como se explicará más adelante. A nivel de código, el mayor problema lo representa el permitir como opcional el blending ya que para que la representación sea correcta y no haya objetos de fondo superpuestos a objetos de primer plano es necesario que si el blending está desactivado definir todos los objetos en orden desde el primer plano hasta el fondo mientras que si está desactivado los objetos se definen en el orden contrario por lo que el código se duplica para realizar la misma operación.

Problemas de hardware

La biblioteca OpenGL trabaja a muy bajo nivel con el hardware gráfico para obtener mejor rendimiento y calidad de imagen, sin embargo si la tarjeta no es 100% compatible con OpenGL, es anticuada o tiene una arquitectura especial es necesario

configurar parámetros complejos e individuales para cada hardware para que todo funcione correctamente.

Debido a la relativa sencillez del resto de operaciones gráficas, los mayores problemas han aparecido con las opciones del blending y el suavizado de polígonos. Ambas se utilizan para reducir el efecto “cuadrícula” de los polígonos y para suavizar y mejorar visualmente la superposición de polígonos. Finalmente se decidió hacer opcionales estas mejoras para que los usuarios que tengan este tipo de problemas puedan utilizar la aplicación.

El grueso de las pruebas gráficas se han realizado sobre equipos con hardware gráfico de la compañía ATI en los que no ha aparecido ningún tipo de problema. Aunque sin estas opciones la aplicación debería funcionar correctamente sobre cualquier equipo compatible con OpenGL, no se puede asegurar al 100% un correcto funcionamiento en cualquier hardware dada la gran variedad de arquitecturas actuales y de hecho se ha comprobado que existen incompatibilidades con alguna de las tarjetas de video actuales o sus controladotes de OpenGL.

• Ilustración 2: Suavizado de polígonos con blending

Capítulo
3

Capítulo 3 : Reproducción de archivos

Introducción

En esta sección se va a describir todo lo relativo a la reproducción del proyecto a través del sistema multimedia de la computadora.

La funcionalidad implementada es la que contiene cualquier reproductor multimedia normal además de poder reproducir fragmentos de archivo, utilidad importante en el modo de edición de las canciones. Es decir:

- Reproducción de archivos, parcial o total.
- Avance
- Retroceso
- Avance hasta el final para añadir
- Retroceso hasta el principio
- Stop
- Pausa

Para llevarla a cabo había dos opciones, hacerla a bajo nivel, para lo cual había que hacer una investigación bastante intensiva de las posibilidades de cada una de las gamas de tarjetas de audio o bien basarse en la interfaz del sistema operativo para la reproducción de sonido, tras sopesar las posibilidades decidimos que usaríamos la interfaz del sistema operativo mientras que nos diera las posibilidades que nosotros buscamos, y si en algún caso necesitábamos algo extra pues tendríamos que recurrir a la implementación a bajo nivel del reproductor del sonido.

La interfaz del sistema operativo Windows para audio se denomina comandos MCI (Interfaz de control multimedia), el cual contiene un conjunto de comandos para la reproducción desde dispositivos y grabación de ficheros multimedia. Estos comandos son una interfaz genérica para cada tipo de dispositivo multimedia.

Comandos MCI de Windows

El comando principal MCI se denomina `mciSendCommand(..)` en el que sus parámetros deciden el tipo de acción a realizar ya sea apertura en modo lectura o grabación de un fichero, como la toma de muestras de la fuente de sonido para grabarlo posteriormente en un fichero

A parte de la función *mciSendCommand* existen unas cuantas constantes para definir el tipo de mensaje y el dispositivo destinatario del mismo, que se irán especificando a la vez que se explique la funcionalidad implementada:

Esta función devuelve un código de error o de éxito en función del resultado del mensaje, para obtener la causa del error está la otra función mci que usamos y que se encuentra bajo el nombre `mciGetErrorString(..)` la cual nos devuelve una cadena de texto con la descripción del error que se ha encontrado para la ejecución del mensaje.

Formato de los comandos MCI

Los mensajes MCI constan de los siguientes tres elementos:

- Una constante que indica la acción a realizar
- Una estructura que contiene los parámetros relativos al mensaje mandado al dispositivo, por ejemplo el nombre del fichero a abrir.
- Un conjunto de bits o banderolas que especifican la opciones para el comando y los campos de validación en el bloque de los parámetros.

El siguiente comando de ejemplo envía el mensaje de reproducción al dispositivo identificado por el numero de dispositivo "dispositivoID".

```
mciSendCommand(dispositivoID,           // id del dispositivo
    MCI_PLAY,                           // mensaje
    0,                                  // banderolas
    (DWORD)(LPVOID) &mciPlayParms);     // parm de reproducción
```

Para el primer parámetro existe la opción de especificar un "para todo" (`MCI_ALL_DEVICE_ID`) como identificador de dispositivo para cualquier comando que no devuelva información. Cuando se especifica "para todo", el interfaz MCI envía el comando secuencialmente a todos los dispositivos abiertos por la aplicación que envía el comando. La utilización de esta posibilidad es una manera apropiada para controlar todos los dispositivos pero no se debe usar para sincronizar todos los dispositivos, ya que el tiempo que pasa desde que se manda el comando al primer dispositivo hasta que se manda al último puede variar.

Un ejemplo de utilización de esta posibilidad "para todo" es cerrar todos los ficheros abiertos para la reproducción.

Dispositivos soportados

MCI reconoce un conjunto básico de tipos de dispositivos. Un tipo de dispositivo es un conjunto de drivers que comparten un conjunto de instrucciones y que son usadas para controlar dispositivos multimedia o ficheros de datos similares. Algunos de los comandos MCI, como puede ser `MCI_OPEN`, requiere que se especifique un tipo de dispositivo. La siguiente tabla muestra una lista de tipos de dispositivos que soporta la implementación de los comandos MCI usada para este proyecto. Dicha implementación incluye conjuntos de instrucciones diferentes para esos dispositivos.

Tipo de dispositivo	Constante	Descripción
cdaudio	MCI_DEVTYPE_CD_AUDIO	Reproductor de CD's de audio
dat	MCI_DEVTYPE_DAT	Reproductor de cintas de audio digital
digitalvideo	MCI_DEVTYPE_DIGITAL_VIDEO	Vídeo digital en una ventana
other	MCI_DEVTYPE_OTHER	Dispositivo MCI indefinido
overlay	LMCI_DEVTYPE_OVERLAY	Vídeo analógico en ventana
scanner	MCI_DEVTYPE_SCANNER	Scanner de imágenes
sequencer	MCI_DEVTYPE_SEQUENCER	Secuenciador MIDI
vcr	MCI_DEVTYPE_VCR	Reproductor de cintas de vídeo
videodisc	MCI_DEVTYPE_VIDEODISC	Reproductor de discos de vídeo
waveaudio	MCI_DEVTYPE_WAVEFORM_AUDIO	Dispositivo de audio que reproduce ficheros de ondas digitalizadas

• Tabla 12: Dispositivos soportados por los comandos MCI

Funcionalidad implementada

Abrir

Funcionamiento

Cada vez que se abre o se crea un archivo se crea una instancia de la clase reproductor que contiene todos los métodos necesarios para la reproducción.

Detalles de la implementación

Se invocará al constructor de la clase *Reproductor* que tiene el siguiente formato: "Reproductor(AnsiString nombreFichero, HWND hwnd)"

Parámetro	Tipo	Funcionalidad
`nombreFichero`	`AnsiString`	Contiene el nombre físico del archivo a reproducir
`hwnd`	`HWND`	Es el manejador de una ventana a la que se le notificarán los resultados de la reproducción.

• Tabla 13: Parámetros del constructor Reproductor.Reproductor

La función realiza las siguientes acciones:

Manda un comando del tipo `MCI_OPEN` al interfaz MCI con el parámetro del nombre del fichero, y devuelve éxito o fracaso y el numero de identificador de dispositivo asignado en caso de éxito que guardamos para realizar el resto de las acciones a las que da opciones esta clase.

Reproducir

Funcionamiento

Cada vez que se pulsa el botón "play" en el panel de botones de reproducción se llama a este método.

Si en el archivo a reproducir está seleccionada alguna parte del fichero se reproducirá la parte seleccionada, si no se procederá a la reproducción completa del fichero.

Detalles de la implementación

Se invocará a la función "play(long sampleStart, long sampleEnd)"

Parámetro	Tipo	Funcionalidad
`sampleStart`	`long`	Muestra de comienzo
`sampleEnd`	`long`	Muestra de final

• Tabla 14: Parámetros de la función Reproductor.Play

Ambos parámetros son opcionales, en caso de ser omitidos se toma sampleStart con valor 0 y sampleEnd lo asignamos a la última muestra del archivo.

La función realiza las siguientes acciones:

- Lo primero que hace es cambiar el formato en el que nos comunicamos con el sistema de comandos MCI para hablar en número de muestras
- Lo siguiente es mandar la orden de reproducción mediante la orden `mciSendCommand(wDispositivoID, MCI_PLAY, MCI_FROM | MCI_TO, (DWORD) (LPVOID) &mciPlayParams).`
- El último paso es volver a poner el formato de comunicación con el sistema MCI en milisegundos. Para que cuando se quiera obtener el tiempo actual de reproducción la respuesta sea en milisegundos.

Pausar

Funcionamiento

Cada vez que se pulsa el botón "play" en el panel de botones de reproducción se convierte en un botón "pausar" que cuando se pulsa se encarga de llamar a esta función.

Detalles de la implementación

Se invocará a la función "pause()"

Esta función no requiere ningún parámetro.

La función realiza las siguientes acciones:

- Manda la orden de pausa mediante la orden `mciSendCommand( wDispositivoID, MCI_PAUSE, 0, null).`

Avanzar

Funcionamiento

Cada vez que se pulsa el botón ">>" en el panel de botones de reproducción se llama a este método.

Lo que hace esta función es adelantar unos cuantos samples la reproducción en marcha del fichero wav.

Detalles de la implementación

Se invocará a la función "step()"

Esta función no requiere ningún parámetro.

La función realiza las siguientes acciones:

- Lo primero que hace es cambiar el formato en el que nos comunicamos con el sistema de comandos MCI para hablar en número de muestras
- Lo siguiente es mandar la orden de leer el sample actual mediante la instrucción:

```
mciStatusParams.dwItem = MCI_STATUS_POSITION;
mciSendCommand(wIDDispositivo, MCI_STATUS, MCI_STATUS_ITEM,
(DWORD) (LPMCI_STATUS_PARMS) &mciStatusParams);
```

- Ahora que ya tenemos la muestra que se está reproduciendo lo que hago es mover el puntero del fichero a unas cuantas muestras más adelante. Lo cual lo hago con las 2 siguientes instrucciones:

```
mciSeekParams.dwTo =  mciStatusParams.dwReturn + 10;
mciSendCommand(wIDDispositivo, MCI_SEEK, MCI_TO, (DWORD)
(LPMCI_SEEK_PARMS)  &mciSeekParams);
```

- El último paso es volver a poner el formato de comunicación con el sistema MCI en milisegundos. Para que cuando se quiera obtener el tiempo actual de reproducción la respuesta sea en milisegundos.

Avanzar todo

Funcionamiento

Cada vez que se pulsa el botón ">>|" en el panel de botones de reproducción se llama a este método.

Lo que hace esta función es detener la reproducción y mandar el puntero de la reproducción del fichero al final del mismo.

Detalles de la implementación

Se invocará a la función "next()"

Esta función no requiere ningún parámetro.

La función realiza las siguientes acciones:

- Lo único que realizar esta acción es mandar un comando MCI de mover el puntero al final de la pista lo cual lo hago con la instrucción:

```
mciSendCommand(wIDDispositivo,MCI_SEEK,MCI_SEEK_TO_END,hand)
```

Retroceder

Funcionamiento

Cada vez que se pulsa el botón "<<" en el panel de botones de reproducción se llama a este método.

Lo que hace esta función es retrasar unos cuantos samples la reproducción en marcha del fichero wav.

Detalles de la implementación

Se invocará a la función "back()"

Esta función no requiere ningún parámetro.

La función realiza las siguientes acciones:

- Lo primero que hace es cambiar el formato en el que nos comunicamos con el sistema de comandos MCI para hablar en número de muestras

- Lo siguiente es mandar la orden de leer el sample actual mediante la instrucción:

```
mciStatusParams.dwItem = MCI_STATUS_POSITION;
mciSendCommand(wIDDispositivo, MCI_STATUS, MCI_STATUS_ITEM,
(DWORD) (LPMCI_STATUS_PARMS) &mciStatusParams);
```

- Ahora que ya tenemos la muestra que se está reproduciendo lo que hago es mover el puntero del fichero a unas cuantas muestras más adelante. Lo cual lo hago con las 2 siguientes instrucciones:

```
mciSeekParams.dwTo =  mciStatusParams.dwReturn - 10;
mciSendCommand(wIDDispositivo, MCI_SEEK, MCI_TO, (DWORD)
(LPMCI_SEEK_PARMS)  &mciSeekParams);
```

- El último paso es volver a poner el formato de comunicación con el sistema MCI en milisegundos. Para que cuando se quiera obtener el tiempo actual de reproducción la respuesta sea en milisegundos.

Realmente la única diferencia de esta función con la de avanzar es que en lugar de sumar un número determinado de muestras a la actual lo que hacemos es restarlas para retroceder un poco en la reproducción.

Retroceder todo

Funcionamiento

Cada vez que se pulsa el botón "|<<" en el panel de botones de reproducción se llama a este método.

Lo que hace esta función es detener la reproducción y mandar el puntero de la reproducción del fichero al comienzo del mismo.

Detalles de la implementación

Se invocará a la función "prev()"

Esta función no requiere ningún parámetro.

La función realiza las siguientes acciones:

- Lo único que realizar esta acción es mandar un comando MCI de mover el puntero al comienzo de la pista lo cual lo hago con la instrucción:

```
mciSendCommand(wIDDispositivo,MCI_SEEK,MCI_SEEK_TO_START,hand)
```

Grabar

Funcionamiento

Cada vez que se pulsa el botón de grabar en el panel de botones de reproducción se llama a este método.

Si en el archivo a reproducir está seleccionada alguna parte del fichero se grabará encima la parte seleccionada desde la fuente de sonido, si no se procederá a la escritura del fichero desde el principio.

Detalles de la implementación

Se invocará a la función "recprd(long sampleStart, long sampleEnd)"

Parámetro	Tipo	Funcionalidad
sampleStart	long	Muestra de comienzo
sampleEnd	long	Muestra de final

• Tabla 15: Parámetros de la función Reproductor.recprd

La función realiza las siguientes acciones:

- Lo primero que hace es cambiar el formato en el que nos comunicamos con el sistema de comandos MCI para hablar en número de muestras

- Lo siguiente es mandar la orden de grabación mediante la orden

```
mciSendCommand(wIDDispositivo, MCI_RECORD, MCI_FROM |
MCI_RECORD_OVERWRITE | MCI_TO , (DWORD) (LPMCI_RECORD_PARMS)
&lpRecord);
```

- Luego lo que hago es grabarlo de memoria al fichero temporal habitual para guardar los cambios. Y por último volvemos a poner el formato de comunicación con el sistema MCI en milisegundos. Para que cuando se quiera obtener el tiempo actual de reproducción la respuesta sea en milisegundos.

Capítulo
4

Capítulo 4 : Edición de archivos

Introducción

En esta sección quedarán englobadas todas aquellas funcionalidades destinadas a la edición de ficheros wav desde un punto de vista semántico, es decir, la transformación del sonido que contienen. SoundEx permite aplicar sobre un fichero wav los siguientes efectos o transformaciones (ver Menú->Efectos):

- **Amplificar**: La señal de audio contenida en el fichero abierto queda amplificada positiva o negativamente según se especifique.
- **Fade**: Es un amplificación de la señal de audio pero no constante. La variabilidad en el coeficiente de amplificación depende de la función seleccionada que podrá ser logarítmica, exponencial, etc.
- **Ecos**: Típico efecto de eco, es decir, se produce la repetición de la señal original tras un cierto tiempo.
- **Reverberación**: Se simula el efecto que tendría el escuchar la onda original en un recinto acústico determinado.

Como ya se ha explicado en otras secciones, un fichero wav esta compuesto por una cabecera, y una colección de muestras (8 o 16 bits) que representan los valores obtenidos tras la discretización de la onda de audio analógica original. El número de muestras por segundo que se toman de la onda original es lo que se llama frecuencia de muestreo; un valor típico de este parámetro podría ser 44100Hz, lo que nos indica que por cada segundo de audio el fichero presentará 44100 muestras dispuestas de forma secuencial.

En general, las transformaciones enumeradas en esta sección seguirán el siguiente procedimiento:

A) Se tiene un fichero wav origen, a partir del cual deseamos aplicar el efecto seleccionado.

B) Se recorre secuencialmente muestra por muestra.

C) Para cada muestra del fichero origen, se realizan los cálculos necesarios de tal forma que escribimos en el fichero destino una muestra resultado de la modificación de la original.

D) En algunos casos, una vez recorrido por completo el fichero origen, será necesario seguir produciendo muestras que se escribirán en el fichero destino, produciéndose así un alargamiento de éste con respecto al fichero original.

Detalles generales de implementación

En el momento que se seleccione uno de los efectos (Menu->Efectos->xxxx), se invocará a la función correspondiente en la clase *Efectos*. Se tiene un método por cada uno de los efectos aplicables, que será responsable de:

- Comprobar que existe parte seleccionada en el fichero. Si no es así se devuelve el control al programa principal sin efectuarse efecto alguno.
- Mostrar el formulario correspondiente.
- Hacer un tratamiento de los datos recogidos como parámetros configurables en el formulario, comprobando en su caso posibles inconsistencias.
- Invocar a la función correspondiente del módulo *UFunciones* con los parámetros necesarios. En ocasiones puede ser necesaria una distinción de casos para determinar que función será invocada.
- Borrar el formulario.

A continuación se explicará de forma más detallada el funcionamiento y los detalles de implementación de cada uno de los efectos enumerados arriba.

Amplificación

Funcionamiento

Una vez seleccionemos el efecto "Amplificar" (Menu->Efectos->Amplificar), se presentará un formulario en el que podremos especificar determinados parámetros, que determinarán el resultado de la amplificación.

La siguiente tabla ilustra y explica el significado de los distintos parámetros configurables en el formulario correspondiente a la amplificación:

Parámetro	Rango	Funcionalidad
Canal izquierdo/Mono	0 - 300	Indica el porcentaje de amplificación que queremos aplicar a la señal original en su canal izquierdo si es estéreo, o en la señal completa si es mono.
Canal derecho	0 - 300	Porcentaje de amplificación que queremos aplicar a la señal original en su canal derecho.
Amplificación independiente por canal	True / False	Si esta desactivado entonces se obliga a que sean iguales los parámetros de porcentaje de amplificación de ambos canales. Si se quieren especificar valores diferentes de amplificación para cada canal entonces deberá estar activado.

• Tabla 16: Parámetros configurables de la amplificación

Al pulsar el botón Aplicar, se ejecutará la amplificación, con los parámetros especificados, sobre la sección seleccionada del fichero abierto; y quedará en pantalla el nuevo fichero ya modificado.

Detalles de la implementación

Se invocará a la función "Amplificar_Region(WAVfile, out, ampi, ampd, formprogreso->Pb1, ini, fin);"

Parámetro	Tipo	Funcionalidad
WAVfile	WaveFile*	Puntero al fichero origen
out	WaveFile*	Puntero al fichero destino
ampi	float	Factor de amplificación para canal izquierdo
ampd	float	Factor de amplificación para canal derecho
formprogreso->Pb1	TProgressBar*	Puntero a la barra de progreso para ir modificándola según se recorre el fichero
ini	unsigned long int	Índice del primer sample seleccionado
fin	unsigned long int	Índice del último sample seleccionado

• Tabla 17: Parámetros de la función UFunciones.Amplificar_Region

La función realiza las siguientes acciones:

- Se copia del fichero origen al destino la parte previa a la seleccionada si existe.
- En la parte seleccionada, se recorre muestra a muestra el fichero origen, y para cada muestra se escribe en el fichero destino la muestra amplificada. En pseudocódigo sería:

```
para cada muestra
  origen->leerMuestra(muestra);
  destino->escribirMuestra(muestra*ampi);
  si (estereo)
    origen->leerMuestra(muestra);
    destino->escribirMuestra(muestra*ampd);
  fsi
fpara
```

- Se copia del fichero origen al destino la parte posterior a la seleccionada si existe.

Fade

Funcionamiento

Una vez seleccionemos el efecto "Fade" (Menu->Efectos->Fade), se presentará un formulario en el que podremos especificar determinados parámetros, que determinarán el resultado del Fade aplicado. En el formulario se mostrará un dibujo, en forma de gráfica, de la envolvente que se aplicará al fichero wav original. La gráfica representa en su eje 'x' el tiempo, correspondiente a la parte seleccionada del fichero wav abierto, en la cual se aplicará el efecto; y en su eje 'y' el factor de amplificación que se aplicará a cada muestra. Se entiende que el rango de valores de factores de amplificación entre los que oscilará la curva, está acotado por dos parámetros de entrada configurables en el propio formulario.

La siguiente tabla ilustra y explica el significado de los distintos parámetros configurables en el formulario correspondiente al Fade:

Parámetro	Rango	Funcionalidad
Amplificación inicial (Ampi)	0 – 200	Es el valor inicial del factor de amplificación. Implica que la envolvente aplicada multiplicará a la primera muestra seleccionada por Ampi/100.
Amplificación final (Ampf)	0 - 200	Es el valor final del factor de amplificación. Implica que la envolvente aplicada multiplicará a la última muestra seleccionada por Ampf/100.
Tipo de función	{Lineal,Logarítmica, Exponencial,e^{-x}}	Especifica el tipo de función usada, que determinará el valor del factor de amplificación, interpolando entre Ampi/100 y Ampf/100 para cada muestra intermedia.
Parámetros de ajuste de la función	Diferente para cada función	Determinan lo pronunciada que será la función aplicada

• Tabla 18: Parámetros configurables del Fade

Así, la onda original quedará multiplicada por la función envolvente, que en general proporcionará un valor de factor de amplificación diferente para cada una de las muestras, obtenido interpolando con la función seleccionada entre los valores Ampi/100 y Ampf/100.

Al pulsar el botón Aplicar, se ejecutará el efecto de Fade, con los parámetros especificados, sobre la sección seleccionada del fichero abierto; y quedará en pantalla el nuevo fichero ya modificado.

Ejemplos de aplicación

- "Fade Out" con Ampi=100, Ampf=0 y Función=Log: Se produce una bajada progresiva logarítmica en el volumen de la señal de audio, desde su valor original hasta el silencio total. El efecto imitaría una situación en la cual el oyente se aleja progresivamente de forma lineal (en el espacio) con respecto a la fuente de sonido.

- "Fade In" con Ampi=0, Ampf=200 y Función=Exp: Se produce una subida progresiva logarítmica en el volumen de la señal de audio, desde el silencio hasta el doble del volumen original.

Detalles de la implementación

Visualización de la envolvente

Utilizaremos la librería OpenGL como soporte para el dibujo de gráficas. En cierto modo nosotros nos limitaremos a pintar una serie de puntos dados por sus coordenadas x e y, respecto al sistema de coordenadas que especifiquemos, y que se corresponderá con un área determinada de la pantalla dentro del formulario correspondiente. Se tienen dos objetos (cuadrícula y gráfica) cada uno de los cuales será responsable de dibujar en pantalla la parte que le corresponde:

- Objeto cuadricula: Responsable de dibujar una cuadrícula, consistente en un número (configurable) determinado de líneas horizontales y verticales.

- Objeto grafico: Responsable de pintar la gráfica de la función elegida, que será ajustada respecto a los parámetros de ajuste de la función. El proceso de pintado de esta gráfica consistirá en un recorrido discreto por el dominio de la función a aplicar, calculando en cada punto x, su valor f(x) asociado; y realizándose

posteriormente una transformación en el sistema de coordenadas para ajustar los puntos obtenidos al sistema especificado.

Aplicación del efecto sobre el archivo

Primeramente se hace un análisis de los parámetros de entrada del formulario. Se comprueba si ampi <= ampd (o lo contrario). De esta forma se determina si el Fade a aplicar es de tipo 'In' o 'Out', es decir, si la envolvente a aplicar será creciente o decreciente. De esta forma elegiremos la función apropiada (creciente o decreciente) cambiando de signo a la función base si fuese necesario. Así por ejemplo, si se desea aplicar un "Fade Out" usando una función logarítmica, deberemos pasar como parámetro final a la función –Logx, pues la función Logx es creciente en todo su dominio. El parámetro correspondiente al tipo de función aplicada tomará un valor de entre los siguientes: f(x) **?** {x,-x,Lnx,-Lnx,e^x,-e^x,e^{-x},-e^{-x}}

Posteriormente se invocará a la función "Fade_Region(WAVfile, out, funcion, x_min, x_max, amp_min, amp_max, formprogreso->Pb1, factor, ini, fin);"

Parámetro	Tipo	Funcionalidad
WAVfile	WaveFile*	Puntero al fichero origen
out	WaveFile*	Puntero al fichero destino
funcion	int	Determina el tipo de función
X_min	float	Parámetro de ajuste de la función
X_max	float	Parámetro de ajuste de la función
Amp_min	float	Factor de amplificación mínimo
Amp_max	float	Factor de amplificación máximo
formprogreso->Pb1	TProgressBar*	Puntero a la barra de progreso para ir modificándola según se recorre el fichero
factor	float	Parámetro de ajuste de la función
ini	unsigned long int	Índice del primer sample seleccionado
fin	unsigned long int	Índice del último sample seleccionado

• Tabla 19: Parámetros de la función UFunciones.Fade_Region

La función realiza las siguientes acciones:

- Se copia del fichero origen al destino la parte previa a la seleccionada si existe.
- En la parte seleccionada, se recorre muestra a muestra el fichero origen, y para cada muestra se escribe en el fichero destino la muestra amplificada según el valor correspondiente obtenido por interpolación en la función elegida. En pseudocódigo sería:

```
int nSamples = fin – ini + 1 ;
float rangox = x_max – x_min;
float y_min = min(f(x_min),f(x_max));
float y_max = max(f(x_min),f(x_max));
float rangoy = fabs(y_max – y_min);
float rangoamp = fabs(amp_max – amp_min);

float step = rangox/nSamples;
para cada muestra
  origen->leerMuestra(muestra);
  f = f(x);
  f_ajustado = amp_min + (rangoamp*(f – y_min))/rangoy;
  x = x + step;
  destino->escribirMuestra(muestra*f_ajustado);
```

```
  si (estereo)
    origen->leerMuestra(muestra);
    destino->escribirMuestra(muestra*f_ajustado);
  fsi
fpara
```

- Se copia del fichero origen al destino la parte posterior a la seleccionada si existe.

Ecos

Funcionamiento

Una vez seleccionemos el efecto "Delay" (Menu->Efectos->Delay), se presentará un formulario en el que podremos especificar determinados parámetros, que determinarán el resultado del Eco aplicado. A continuación explicaremos el significado de cada uno de los parámetros configurables en el formulario:

- Volumen de mezcla: Es el porcentaje de volumen que tendrá la señal repetida con respecto a la señal original.

- Tiempo de retardo: Es el tiempo en milisegundos que tardará en repetirse la señal original. Esa repetición vendrá multiplicada por el factor de amplificación determinado por el volumen de mezcla.

- Feedback (Realimentación): Es el porcentaje de volumen que tendrá la señal de eco para las sucesivas repeticiones. Así por ejemplo si feedback es igual cero, se escuchará un eco con volumen "Volumen de mezcla" con un retardo de "Tiempo de retardo", y no se volverán a escuchar más repeticiones en tiempos 2*"Tiempo de retardo", etc.

Si el archivo sobre el que queremos aplicar el efecto de eco fuese estéreo, el formulario presentará una doble pestaña, ofreciéndonos la opción de especificar distintos parámetros para cada uno de los canales. El resultado es el mismo que se obtendría si se tratase al archivo estéreo como si fuesen dos archivos mono distintos, aplicáramos el efecto de eco con distintos parámetros para cada uno, y posteriormente los juntáramos en un archivo único estéreo.

Supóngase la aplicación de un efecto de eco sobre un archivo wav, compuesto por una colección de 8 muestras, donde el tiempo de retardo abarca dos muestras. La siguiente figura ilustra la forma en la operará nuestro algoritmo, y el resultado que se obtendrá como archivo de salida (colección de muestras de salida):

Muestras originales	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
1ª Repetición			M1	M2	M3	M4	M5	M6	M7	M8
2ª Repetición					M1	M2	M3	M4	M5	M6
Muestras de salida	M1	M2	*1	*2	*3	*4	*5	*6	*7	*8

Donde:
*1 = M3 + volumen*M1.

*2 = M4 + volumen*M2.
3 = M5 + volumen(M3 + feedback*M1).
4 = M6 + volumen(M4 + feedback*M2).
5 = M7 + volumen(M5 + feedback*M3 + feedback2*M1).
6 = M8 + volumen(M6 + feedback*M4 + feedback2*M2).
7 = M9 + volumen(M7 + feedback*M5 + feedback2*M3 + feedback3*M1).
8 = M10 + volumen(M8 + feedback*M6 + feedback2*M4 + feedback3*M2).

La siguiente tabla ilustra y explica el significado de los distintos parámetros configurables en el formulario correspondiente:

Parámetro	Rango	Funcionalidad
Volumen de mezcla	0 – 200	Porcentaje de volumen que tendrá la señal repetida con respecto a la señal original.
Tiempo de retardo	1 - 1000	Es el tiempo en milisegundos que tardará en repetirse la señal original.
Feedback	0 - 100	Es el porcentaje de volumen que tendrá la señal de eco para las sucesivas repeticiones.
Expandir el fichero original	True | False	Si esta activado, el fichero original podría aumentar su duración mientras los ecos acumulados posean una señal de pico por encima de un cierto umbral.

• Tabla 20: Parámetros configurables del Eco

Así, la onda original quedará modificada de forma que cada muestra vendrá como resultado de sumar la muestra original, y los ecos que se van acumulando de muestras anteriores. Además, de estar activa la opción “Expandir el fichero original”, se añadirán muestras adicionales, calculadas a partir de ecos acumulados de muestras anteriores, al final del fichero de salida produciéndose así un alargamiento de éste.

Al pulsar el botón Aplicar, se ejecutará el efecto de eco, con los parámetros especificados, sobre la sección seleccionada del fichero abierto; y quedará en pantalla el nuevo fichero ya modificado.

Detalles de la implementación

Primeramente se hace un tratamiento de los parámetros de entrada, y en función de si se desea hacer un eco estéreo (si el archivo lo es) o un eco simple se procede a la llamada de la función correspondiente.

Eco Simple

Se invocará a la función Delay_Region(WAVfile, out, tpo, vol, feedb, exp, suma, formprogreso->Pb1, ini, fin)

Parámetro	Tipo	Funcionalidad
`WAVfile`	`WaveFile*`	Puntero al fichero origen
`out`	`WaveFile*`	Puntero al fichero destino
`tpo`	`unsigned long int`	Tiempo de retardo
`vol`	`float`	Volumen de mezcla/100
`feedb`	`float`	Feedback/100
`exp`	`bool`	Activo para expandir el fichero
`suma`	`bool`	No se usa en esta versión(siempre True)
`formprogreso->Pb1`	`TProgressBar*`	Puntero a la barra de progreso para ir modificándola según se recorre el fichero

ini	unsigned long int	Índice del primer sample seleccionado
fin	unsigned long int	Índice del último sample seleccionado

• Tabla 21: Parámetros de la función UFunciones.Delay_Region

El algoritmo procederá a realizar un recorrido secuencial de las muestras del archivo origen, aunque en realidad el recorrido se hace de 'step' en 'step' muestras, siendo 'step' el número de muestras que equivalgan al tiempo de retardo pasado como entrada.

Se usarán dos buffers de muestras:

- Buf[0..step-1]: Almacena muestras como resultado de sumar las muestras del 'step' anterior y las muestras correspondientes a steps más antiguos que habrán ido multiplicando(disminuyendo) su valor por el coeficiente del feedback.
- WavBuffer[]: Usado por razones de eficiencia para ir haciendo lecturas en grupos de muestras y tenerlas así disponibles en memoria.

La función realiza las siguientes acciones:

- Se copia del fichero origen al destino la parte previa a la seleccionada si existe.
- Se hace la inicialización de las variables necesarias:

```
int nSamples = fin - ini + 1;//Numero de muestras
seleccionados
expandir = ((expandir) && ((fin + 1) >= fileIn-
>GetNumSamples()));

LlenarBuffer(fileIn,wavBuffer,0,long_wavBuffer,nSamples);
  //long_wavBuffer será en general potencia de 2, por ej.
16384

step=(int)(tpo*fileIn->GetSampleRate());
float* buf = new float[step];
```

- En la parte seleccionada, se recorre muestra a muestra el fichero origen, y para cada muestra se escribe en el fichero destino la muestra resultante de sumar la muestra original y los ecos correspondientes acumulados de muestras anteriores. En pseudocódigo sería:

```
para cada muestra
  muestra = wavBuffer[j];
    //equivalente a origen->leerMuestra(muestra)
  wavBuffer[j-1] = (muestra + buf[k]*volumen);
  buf[k] = muestra + buf[k]*feedback;
    //k es el índice correspondiente en el buffer buf y nos
indicará cuando se van cumpliendo los steps.
  k++;
  si (estereo)
    muestra = wavBuffer[j];
    wavBuffer[j-1] = (muestra + buf[k]*volumen);
    buf[k] = muestra + buf[k]*feedback;
    k++;
  fsi
```

```
  si (k == step)
    k = 0;
  fsi

  si (j >= long_wavBuffer)
    j = 0;
    VaciarBuffer(fileOut,wavBuffer,long_wavBuffer);

LlenarBuffer(fileIn,wavBuffer,i+1,long_wavBuffer,nSamples);
  Fsi
fpara
```

- Si esta activa la opción "Expandir fichero original" se procederá a añadir al final del archivo de salida calculado hasta el momento, las muestras resultantes de los ecos acumulados de muestras anteriores mientras el valor de pico en cada paso este por encima de un cierto umbral, en este caso será 0.05.

```
si (expandir)
  bool silencio = false;
  float discrim = 0.05;
  mientras (!silencio)
    para cada i entre k y step
       destino->escribeMuestra(volumen*buf[i]);
       buf[i] = buf[i]*feedback;
       si (fabs(buf[i]) > pico)
         pico = buf[i]; fsi
    fpara
    k = 0;
    si (pico < valor umbral)
      silencio = true;
      pico = 0.0;
    fsi
  fmientras
fsi
```

Eco Estéreo

Se invocará a la función: Delay_Estereo_Region(WAVfile, out, tpo1, vol1, feedb1, tpo2, vol2, feedb2, exp, suma, formprogreso->Pb1, ini, fin)

Parámetro	Tipo	Funcionalidad
WAVfile	WaveFile*	Puntero al fichero origen
out	WaveFile*	Puntero al fichero destino
tpo1	unsigned long int	Tiempo de retardo para canal izquierdo
vol1	float	Volumen de mezcla/100 para canal izquierdo
feedb1	float	Feedback/100 para canal izquierdo
tpo2	unsigned long int	Tiempo de retardo para canal derecho
vol1	float	Volumen de mezcla/100 para canal derecho
feedb1	float	Feedback/100 para canal derecho
exp	bool	Activo para expandir el fichero
suma	bool	No se usa en esta versión(siempre True)
formprogreso->Pb1	TProgressBar*	Puntero a la barra de progreso para ir modificándola según se recorre el fichero

ini	unsigned long int	Índice del primer sample seleccionado
fin	unsigned long int	Índice del último sample seleccionado

• Tabla 22: Parámetros de la función UF.Delay_Estereo_Region

El algoritmo procederá a realizar un recorrido secuencial de las muestras del archivo origen, teniendo en cuenta que el archivo es estéreo y que las muestras de cada canal van intercaladas.

Se usarán tres buffers de muestras:

- Buf1[0..step1-1]: Almacena muestras como resultado de sumar las muestras del 'step' anterior y las muestras correspondientes a steps más antiguos que habrán ido multiplicando(disminuyendo) su valor por el coeficiente del feedback. Todo ello aplicado al canal izquierdo.

- Buf2[0..step2-1]: Almacena muestras como resultado de sumar las muestras del 'step' anterior y las muestras correspondientes a steps más antiguos que habrán ido multiplicando(disminuyendo) su valor por el coeficiente del feedback. Todo ello aplicado al canal derecho.

- WavBuffer[]: Usado por razones de eficiencia para ir haciendo lecturas en grupos de muestras y tenerlas así disponibles en memoria.

La función realiza las siguientes acciones:

- Se copia del fichero origen al destino la parte previa a la seleccionada si existe.

- Se hace la inicialización de las variables necesarias:

```
int nSamples = fin - ini + 1;//Numero de muestras
seleccionados
expandir = ((expandir) && ((fin + 1) >= fileIn-
>GetNumSamples()));

LlenarBuffer(fileIn,wavBuffer,0,long_wavBuffer,nSamples);
  //long_wavBuffer será en general potencia de 2, por ej.
16384

step1=(int)(tpo1*fileIn->GetSampleRate());
step2=(int)(tpo2*fileIn->GetSampleRate());
float* buf1 = new float[step1];
float* buf2 = new float[step2];
```

- En la parte seleccionada, se recorre muestra a muestra el fichero origen, y para cada muestra se escribe en el fichero destino la muestra resultante de sumar la muestra original y los ecos correspondientes acumulados de muestras anteriores; teniendo en cuenta que encontraremos alternadas las muestras de cada canal. En pseudocódigo sería:

```
para cada muestra
  muestra = wavBuffer[j];
    //equivalente a origen->leerMuestra(muestra)
  wavBuffer[j-1] = (muestra + buf1[k1]*volumen1);
  buf1[k1] = muestra + buf1[k1]*feedback1;
```

```
    //k1 es el índice correspondiente en el buffer buf1 y nos
indicará cuando se van cumpliendo los steps del canal
izquierdo.
  k1++;
  muestra = wavBuffer[j];
  wavBuffer[j-1] = (muestra + buf2[k2]*volumen2);
  buf2[k2] = muestra + buf2[k2]*feedback2;
  k2++;
  fsi
  si (k1 == step1)
    k1 = 0;
  fsi
  si (k2 == step2)
    k2 = 0;
  fsi

  si (j >= long_wavBuffer)
    j = 0;
    VaciarBuffer(fileOut,wavBuffer,long_wavBuffer);

LlenarBuffer(fileIn,wavBuffer,i+1,long_wavBuffer,nSamples);
  Fsi
fpara
```

- Si esta activa la opción “Expandir fichero original” se procederá a añadir al final del archivo de salida calculado hasta el momento, las muestras resultantes de los ecos acumulados de muestras anteriores mientras el valor de pico en cada paso este por encima de un cierto umbral, en este caso será 0.05. Análogo al eco simple.

Reverberación

Simulando recintos acústicos

Una vez seleccionemos el efecto “Reverb” (Menú->Efectos->Reverb), aparecerá una ventana nueva en la que vamos a pintar el recinto el cual va a ser objeto de estudio para la generación de este efecto.

Dicho recinto a parte de por su forma se caracteriza por otros parámetros de configuración como son el coeficiente de elasticidad de las paredes, el coeficiente de perdida de amplitud por unidad de longitud del medio de transmisión etc.

La siguiente tabla ilustra y explica el significado de los distintos parámetros configurables para que el efecto quede como el usuario desee:

Parámetro	Rango	Funcionalidad
Coeficiente de elasticidad de las paredes	0 – 1	Indica el coeficiente máximo de amplitud de una onda que ha chocado con una pared, por ejemplo, si una onda viene con amplitud máxima y choca con la pared, la amplitud de salida será este coeficiente
Coeficiente de rozamiento del medio de transmisión	0 – 1	Pérdida de amplitud por metro del medio por el que se ha de transmitir el sonido.

Número de rebotes	0 – 1000*	Es el número de rebotes que se estudiarán para obtener el eco en caso de que la onda todavía tenga una amplitud mayor que 0
Precisión	0.1 – 180*	Es la diferencia en grados entre dos exploraciones del recinto consecutivas. Si el primer rayo de exploración sale con ángulo 0 grados el siguiente saldrá con un ángulo equivalente a la precisión establecida.

* Un numero grande de rebotes o una precisión muy grande pueden afectar significativamente el tiempo de proceso de la escena.

Para dibujar el recinto bastará con ir haciendo clic en cada una de las esquinas que contenga y para ayudar al usuario a que el recinto sea perfectamente cerrado, cuando se haya cliqueado el último vértice con el botón izquierdo del ratón, se pulsará el botón derecho del ratón y el recinto se procederá a cerrarse. Posteriormente el usuario tiene que designar un punto de emisión del sonido y otro de recepción que deben estar en el interior del recinto.

Una vez hecho esto el usuario puede ver cómo va a ser la exploración haciendo clic con el botón izquierdo del ratón, cada vez que lo haga se irá aumentando el ángulo de salida del rayo de exploración.

Finalmente para generar el efecto bastará con ir al menú salir y aceptar el resultado o bien cancelar.

Introducción

Con este efecto lo que nosotros hemos pretendido es simular lo que una persona podría escuchar en recintos grandes, como por ejemplo, un auditorio o una catedral, cuando hay puntos de emisión de sonido dentro de él, por ejemplo, actores interpretando una obra o gente hablando mientras que visita la catedral, respectivamente siguiendo el ejemplo anteriormente expuesto.

Después de un gran esfuerzo en la exploración de un recinto a partir de un determinado punto hemos conseguido un efecto, que si bien no ha sido todo lo bueno que esperábamos tampoco nos ha decepcionado, ya que sí que hemos conseguido esa sensación de escuchar ecos como si estuviésemos en un recinto con una buena acústica.

Para lograr este efecto hemos dispuesto un editor de recintos 2D para que el usuario pueda simula la sala que desee y pruebe como sería lo que se escucha desde cada punto de escucha que desee, todo esto lo hemos hecho a través de una tubería gráfica muy popular hace algunos años y que todavía se utiliza en la actualidad, que es la que hemos usado para toda la parte gráfica de la aplicación en general: OpenGL.

Por otra parte, la representación de recinto, los ecos, y los puntos de escucha y de emisión, así como los resultados de la exploración del registro están estructurados en clases de tal manera que pueda sacarse el mayor partido a estos datos.

Utilidad gráfica para el dibujo de recintos acústicos

La utilidad gráfica para el dibujo de los recintos y su exploración es una ventana de OpenGL donde a través de la interacción con el usuario se va dibujando el recinto vértice por vértice y los puntos de emisión y recepción de sonido tal y como se ha explicado antes en el apartado inicial de esta sección.

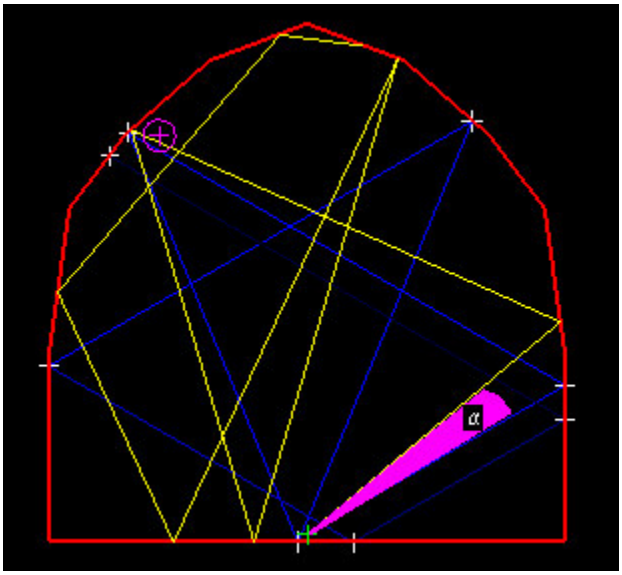
La representación del recinto se lleva a través de la clase escena que contiene una lista de segmentos que delimitan el recinto y la cual se va llenando cada vez que el usuario elige un vértice del recinto, además también consta de una escala para que el usuario tenga una sensación de las dimensiones del recinto que está dibujando, en todo caso la escala es meramente informativa ya que si es necesario se pueden elegir vértices y puntos que caigan a la izquierda o debajo de la escala.

A la escena también la hemos dotado de un zoom potencialmente infinito, es decir, que se puede acercar o alejar la cámara para dibujar recintos o muy grandes o muy pequeños con la única limitación de las limitaciones que posea el entorno gráfico que nos facilita el OpenGL.

Para facilitar el diseño de recintos hemos añadido unos menús con unos recintos preestablecidos para el usuario que quiera probar el sistema y quiera obtener un buen efecto, no se tenga que molestar en dibujar el recinto, sino que ya nosotros se lo facilitamos.

Estudio del comportamiento del sonido en el recinto

Como se puede ver en la siguiente figura este es el comportamiento del programa en dos pasos de exploración del recinto.



• Ilustración 3: Simulación de recintos acústicos

Funcionalmente, lo que se hace es lanzar rayos con ángulos cada vez α veces mayor, siendo α el ángulo mostrado en la figura y que se puede configurar en los parámetros de este módulo del programa, por cada rayo que se lanza se investiga si se cruza con el área de recepción del sonido, si es así se anota el tiempo que tarda en llegar el sonido y la amplitud con la que llega para ese rayo, así sucesivamente hasta llegar a recorrer la circunferencia completa. Con la lista de pares (tiempo, amplitud se procede a aplicar el efecto de reverberación que posteriormente en la siguiente sección se explica con detalle.

Decimos área de recepción del sonido porque consideramos que el punto de recepción no es un punto sin dimensiones sino que lo convertimos en una circunferencia de radio pequeño. Es decir el punto matemático se convierte en lo que

denominamos un punto "gordo". De ahí la representación gráfica del punto con una circunferencia rodeándolo.

Detalles de la implementación

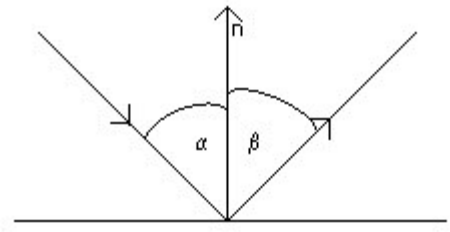
Con respecto a los detalles de implementación la cosa se vuelve mucho más compleja ya que para generar un rayo con todos los rebotes en las paredes hay varios detalles físicos que hay que explicar con detalle, entre ellos la reflexión del rayo.

El método que contiene la exploración del recinto se denomina *devVectorEcos(...)* y que está dentro de la clase en la que guardamos la Escena, al cual se le pasan como parámetros los 4 parámetros indicados anteriormente al inicio de esta sección que lo que hace es devolver la lista de pares (tiempo, amplitud) para que se le pueda pasar al método que genera el efecto en el fichero de muestras.

Dentro de este método lo que se hace básicamente es un bucle que recorre toda la circunferencia, para ello se va incrementando un ángulo que se usará para generar el rayo a partir del punto de emisión del sonido, y dentro de el se genera el rayo con ese ángulo de salida, posteriormente se explora ese rayo de sonido para ver o no si se cruza con el área de recepción del sonido, si es así se añade la el par (tiempo, amplitud) al resultado.

Veamos ahora con más detalle estos dos métodos que se encuentran dentro del bucle principal:

- ecos(número de rebotes, ángulo de salida): lo que hace es que mientras pueda rebotar explora si se corta la dirección actual de la recta con los lados del recinto, si es así me quedo con la que tenga menor distancia entre el punto de cruce y el punto de salida del rayo, y generamos un choque para el cual se usa la normal y la dirección actual del rayo y se averigua la nueva mediante la reflexión



• Ilustración 4: Reflexión del rayo contra la pared

  Donde el cos(α) tiene que ser igual al cos(β). n es la normal al punto de reflexión y α es el ángulo que forma el rayo de entrada con la normal y β es el ángulo que forman la normal y el rayo de salida.

- exploraSolucion(coeficiente del aire, coeficiente de las paredes): lo que hace es básicamente un doble bucle que mira si hay intersección entre las aristas de la línea de eco generada y las aristas de la zona de recepción del sonido, en cuanto se encuentra un cruce se averigua la distancia al origen de emisión y con ella el tiempo, mediante la fórmula e = v * t de la que despejo t = e / v donde t es el tiempo, e el espacio y v una velocidad constante, en este caso usaremos la

velocidad del sonido que es 340 m/s. Y también se obtiene la amplitud relativa al origen del sonido con la que llega al punto de recepción.

Merece la pena en este punto es discutir la fórmula que hemos usado para averiguar la amplitud que llega al punto de recepción, cualquiera que sea la amplitud del sonido que estemos tratando ahora la trataremos como si fuera amplitud 1 y a partir de esa amplitud original se van descontando atenuación del medio de transmisión y el coeficiente de las paredes. Dicha fórmula es:

coefActual = (coefActual - distancia * coefAire) * coefParedes;

En la cual coefActual es la amplitud que posee el rayo en último punto de rebote o en el punto de emisión en el caso inicial. Distancia es la distancia que hay entre el punto en el que nos encontramos y el punto origen de la recta actual y coefAire y coefParedes, son el coeficiente de atenuación por unidad de longitud del medio de transmisión y el coeficiente de rebote de las paredes.

Esta es una aproximación lineal a la atenuación del sonido, pero no es la única ya que existen otras aproximaciones no lineales. Hemos optado por esta ya que es la que en un principio más se acercaba a lo que estábamos buscando obtener.

Finalmente mencionar algunas dificultades que hemos encontrado a la hora de implementar esta parte, como por ejemplo como hallar las normales para que sea cual sea el recinto las normales siempre apunten hacia fuera, en un principio no conseguimos generarlas hacia fuera, por lo cual obligábamos al usuario a generar el recinto dibujándolo en sentido contrario a la agujas del reloj.

Posteriormente tras tratar problemas discretos en la asignatura de Geometría Computacional encontramos una fórmula que nos devolvía el sentido en el que esta dibujado un polígono. Esta fórmula la usamos para averiguar el sentido del dibujo y si esta dibujado en el sentido de las agujas del reloj cambiamos el sentido de la dirección de las normales de los lados y así solucionamos esta restricción.

Aplicación del efecto de reverb

El efecto de simulación de un recinto acústico, se reducirá finalmente a la formación de un archivo de salida constituido por una serie de muestras, que vendrán determinadas por la suma de la onda original y los distintos ecos que se producen por los rebotes de las ondas en las paredes. Por tanto, se trata en realidad de aplicar una serie de ecos simples en nuestro archivo origen, cada uno de los cuales viene determinado por el factor de atenuación, equivalente al volumen de mezcla del efecto de eco ya comentado; y el tiempo de retardo (ya comentado en el efecto de eco).

Tendremos así como salida del proceso de análisis del comportamiento del recinto acústico especificado, y como entrada de la función “reverb” que utilizaremos, una lista de parejas tiempo/volumen, y nos limitaremos a aplicar el algoritmo que a continuación se explica.

Detalles de la implementación

El comportamiento del algoritmo que se explica a continuación, sería equivalente a la aplicación sucesiva de efectos de eco (ver 6.3) sobre el archivo origen, cada uno de ellos con un elemento de la lista de parejas; pero por razones evidentes de eficiencia y debido a los enormes tamaños de los archivos wav, es necesario el diseño de un algoritmo más sofisticado.

- Entrada:
  - Factors[0..n]: Son los factores de atenuación de cada uno de los n ecos a aplicar.
  - Tiempos[0..n]: Son los tiempos de retardo de cada uno de los n ecos. Así cada <Factors[i],Tiempos[i]> formarían la lista de parejas explicada.
- Variables:
  - Steps[0..n]: Es el número de muestras que abarca cada uno de los tiempos de retardo.
  - Long_buf: Es el máximo de Steps[1..n].
  - Buf[0..Long_Buf]: Buffer circular donde se van almacenando los ecos que posteriormente se sumarán a la muestra original.
  - K: Entero usado como índice de la posición actual del buffer circular Buf.
- Pasos:
  - Se recorre muestra a muestra el archivo origen.
  - Para cada muestra que se lee:
    - Se calcula la muestra de salida como suma de la muestra original y Buf[k], que contendrá la suma de todos los ecos de muestras originales anteriores que se oirán en el instante correspondiente a la muestra actual.
    - Se calcula la repercusión como eco que tendrá esa muestra para muestras sucesivas. Para ello se recorre la lista de parejas y para cada una se calcula en que índice del buffer circular irá emplazado el eco correspondiente a la muestra actual. Se deberá sumar con los ecos ya acumulados en la posición correspondiente del buffer circular. Posteriormente esa posición se consumirá en el momento que k avance (modulo n) hasta la posición en cuestión.

Entrando más en detalle, se invocará a la función: “Reverb_v2(WAVfile, out, tpos, factors, nPares, formprogreso->Pb1, ini, fin)”

La siguiente tabla ilustra el significado de cada uno de los parámetros de entrada de la función *UFunciones.Reverb_v2*:

Parámetro	Tipo	Funcionalidad
`WAVfile`	`WaveFile*`	Puntero al fichero origen
`out`	`WaveFile*`	Puntero al fichero destino
`tpos`	`float*`	Tiempos de retardo de cada uno de los n ecos
`factors`	`float*`	Factores de atenuación o volumen de mezcla de cada uno de los n ecos
`nPares`	`int`	Número de parejas <Factor[i],Tiempo[i]>
`formprogreso->Pb1`	`TProgressBar*`	Puntero a la barra de progreso para ir modificándola según se recorre el fichero
`ini`	`unsigned long int`	Índice del primer sample seleccionado
`fin`	`unsigned long int`	Índice del último sample seleccionado

• Tabla 23: Parámetros de la función UFunciones.Reverb_v2

La función realiza las siguientes acciones:

- Se copia del fichero origen al destino la parte previa a la seleccionada si existe.
- Se hace la inicialización de las variables necesarias:

```
expandir=((expandir) && ((fin + 1) >= fileIn-
>GetNumSamples()));

LlenarBuffer(fileIn,wavBuffer,0,long_wavBuffer,nSamples);
   //wavBuffer es el buffer usado por razones de eficiencia
(ver   //apartado anterior, efectos de eco)

unsigned int* steps = new unsigned int[nPares];
unsigned int max_long_buf = 0;

for (int i = 0;i < nPares;i++){
  steps[i] =
((int)(tpos[i]*fileIn->GetSampleRate()))*fileIn-
>GetNumChannels();
  if (steps[i] > max_long_buf)
    max_long_buf = steps[i];
  }//for
unsigned int long_buf =
(max_long_buf + 1)*fileIn->GetNumChannels();

float* buf = new float[long_buf];
for (unsigned i = 0;i<long_buf;i++)
   buf[i] = 0.0;

unsigned int k = 0;
```

- En la parte seleccionada, se recorre muestra a muestra el fichero origen y se procede a realizar los cálculos explicados anteriormente en el buffer de muestras y la muestra correspondiente del archivo de salida. En pseudocódigo sería:

```
para cada muestra i:
  muestra = wavBuffer[j];//Lectura de muestra del archivo
origen
  j++;
```

```
  para cada l entre 0 y nPares
    punt = (k+steps[l]) mod long_buf;
    buf[punt] = buf[punt] + muestra*factors[l];
  Fpara

wavBuffer[j-1]=muestra+buf[k];//Escritura en fichero wav
destino

buf[k] = 0.0;  //El buffer se consume y se borra
k=(k+1)%long_buf;//Se incrementa el apuntador del buffer
circular

si (j >= long_wavBuffer)
  j = 0;
  VaciarBuffer(fileOut,wavBuffer,long_wavBuffer);
  LlenarBuffer(fileIn,wavBuffer,i+1,long_wavBuffer,nSamples);
Fsi

fpara
```

- Si el fin de la parte seleccionada coincide con el final del fichero wav se procederá a añadir al final del archivo de salida calculado hasta el momento, las muestras resultantes de los ecos acumulados de muestras anteriores.

```
si (expandir)
  para  cada muestra del buffer circular
    destino->EscribeMuestra(buf[k]);
    k = (k+1)%long_buf;
  fpara
fsi
```

Capítulo
5

Capítulo 5 : Entorno de aplicación

Introducción

En este capítulo se estudiará el desarrollo y la arquitectura del entorno de la aplicación haciendo referencia a la intercomunicación entre los distintos objetos de la aplicación para poder desarrollar conjuntamente las operaciones descritas anteriormente y la interacción del entorno con el usuario a través de los distintos formularios de la aplicación.

Se centrará la explicación en el formulario principal (*TMainForm*) que es el que contiene el núcleo del manejo de los archivos de la aplicación y del formulario de visualización de archivos (*TWAVfileForm*) que concentra todas las operaciones sobre un archivo concreto. Para este último se han diseñado además una serie de clases para mostrar controles de reproducción y de visualización gráficos a través de OpenGL para hacer más cómodo el manejo de la aplicación para el usuario.

Otros formularios para la intercomunicación con el usuario para la realización de funciones específicas se explican en las capítulos relacionados con estas funciones concretas.

Formulario principal

El formulario principal (*TMainForm*) es el padre del entorno MDI (*Multiple Document Interface*) y por tanto el que contiene toda la información del estado de la aplicación en cada momento. Esta información incluye todos los datos relacionados con cada uno de los archivos abiertos en el entorno y número de ellos, los datos de configuración del entorno elegidos por el usuario y leídos del archivo de configuración "*CentroSonido.ini*", datos de ubicación de los archivos de la aplicación, etc.

Posee acceso a cada uno de los elementos clave del entorno como los visualizadores y formularios de visualización y permite realizar una configuración de los mismos a nivel global. También es el que mantiene las secuencias de muestras copiadas de archivos abiertos para permitir su inserción en otros archivos abiertos o que se abrirán durante la sesión.

El formulario principal es, en resumen, el encargado de realizar las operaciones de manipulación de archivos explicadas en su capítulo correspondiente así como de realizar y mantener la configuración global de la aplicación.

Manipulación de archivos

En el apartado de manipulación de archivos, el formulario principal es el encargado de abrir, cerrar, guardar o renombrar los archivos de audio que sean manejados por el entorno.

Para realizar la gestión de los archivos que estén siendo utilizados en el entorno el formulario almacena los siguientes arrays de datos:

Variable	Tipo	Funcionalidad
`WAVfileOpen[MAXFILES]`	`bool`	Marca de existencia para indicar si el archivo[i] está abierto
`WAVfile[MAXFILES]`	`WaveFile*`	Punteros a los manejadores de bajo nivel de los archivos
`visual[MAXFILES]`	`Visualizador*`	Punteros a los visualizadores de los archivos
`Vform[MAXFILES]`	`TWAVfileForm*`	Punteros a los fromularios de visualización de los archivos
`nombres[MAXFILES]`	`AnsiString`	Nombre y ruta completa de los archivos

• Tabla 24: Variables de manipulación de archivos

El valor `MAXFILES` es el número máximo de archivos que se pueden abrir simultáneamente en el entorno, este valor se ha fijado por razones de rendimiento en ocho aunque no sería difícil aumentar este número.

Al crear el formulario se inicializan todas las marcas de existencia de `WAVfileOpen` a falso, se crean los visualizadores sin asignarles ningún archivo y se declaran nulos los puntero a formularios de visualización.

Cuando se crea o se abre un nuevo archivo primero se comprueban las marcas de existencia para ver si hay espacio para abrir un nuevo archivo y en caso afirmativo comprueba que el archivo no esté ya abierto en el entorno. Si se pasan las dos pruebas anteriores se pasa a hacer el enlace de las estructuras de datos con el archivo: la primera marca i se pone a cierto, se abre el archivo con el manejador de bajo nivel `WAVfile[i]`, se le asigna al visualizador i el manejador anterior, se abre un nuevo formulario de visualización al que se le pasan todos los parámetros de visualización actuales y se almacena la ruta completa del archivo en `nombres[i]`. Si todo se ha realizado de forma correcta y no se ha producido ningún error tendremos un nuevo archivo manejado por el entorno.

Para cerrar, guardar o renombrar un archivo simplemente se le indica al formulario de visualización que esté activo en ese momento que lo haga ya que es éste el que realiza este trabajo. Si el archivo se renombra se recupera el nuevo nombre para almacenarlo en `nombres`. En el caso del cierre, para indicar al entorno que el archivo se ha cerrado (no se ha cancelado el cierre) el formulario de visualización tiene un puntero a la marca de existencia correspondiente que pondrá a falso antes de cerrarse.

Cuando se cierra la aplicación, el formulario solicita una confirmación de cierre. En el caso de que se confirme el cierre se recorren todos los formularios de visualización hijos indicándoles que deben cerrar el archivo que manejen. Todo este proceso se realiza después de realizar la confirmación del cierre pero antes de que éste se lleve a cabo ya que, al ser los formularios de visualización hijos del formulario principal, se cerrarían sin realizar las operaciones necesarias si se cerrara el formulario principal.

La mezcla de archivos se comporta como la apertura de un nuevo archivo que se crea en ese momento a partir de los datos que se solicitan. Una vez el nuevo archivo se ha creado se abre un formulario de visualización marcado como visualizador de mezcla. La única diferencia en esto es que el nuevo formulario de visualización aparecerá con un icono y una cabecera de título distintas para reconocer que el archivo procede de una mezcla de archivos. Esta marca sólo perdura mientras el archivo esté abierto, si se cierra y se vuelve a abrir aparecerá como un archivo normal.

En el formulario principal se almacenan también las muestras copiadas de un archivo para que puedan escribirse en otro archivos en el entorno. Las variables relacionadas con esta operación son:

Variable	Tipo	Funcionalidad
copiado	bool	Marca para indicar si hay muestras copiadas
samples8 samples16	unsigned char* short int*	Array de muestras copiadas de 8 bits Array de muestras copiadas de 16 bits
count	unsigned long int	Número de muestras copiadas
sampleRate	int	Frecuencia de muestreo de las muestras
bitsPerChannel	short int	Número de bits por canal (8 ó 16) de las muestras
channels	short int	Número de canales de las muestras

• Tabla 25: Variables de copia de muestras

A este nivel, las operaciones para leer o escribir muestras de un archivo a otro se limitan a realizar llamadas a funciones de los formularios de visualización que reciben o envían secuencias de muestras en cada caso junto con los datos de frecuencia de muestreo, número de canales y número de bits por canal.

Parámetros de configuración

Se han definido una serie de parámetros opcionales que son configurables por el usuario y que determinan el comportamiento de la aplicación en ciertas operaciones. Son en total cinco parámetros relacionados con cuatro tipos de operaciones:

E) Directorio (dir): Indica el directorio donde se buscarán los archivos por defecto cuando se abre o se guarda un archivo.

F) Tiempo (tmp): Duración en milisegundos que se le asigna por defecto a los archivos nuevos. Es necesario establecer un tiempo por defecto ya que la aplicación trabaja sobre archivos de disco que han de tener como mínimo 15 milisegundos.

G) Pantalla inicial (pnt): Indica si se desea que se muestre la pantalla de presentación inicial al comienzo de la ejecución del programa.

H) Suavizado de polígonos (pol): Indica si se desea utilizar el suavizado de polígonos en la representación gráfica.

I) Blending (bln): Indica si se desea utilizar el blending en la representación gráfica.

Estos parámetros se almacenan en el archivo *"CentroSonido.ini"* y se pueden modificar a través de un formulario de configuración. El contenido del archivo será similar al siguiente (en las tres últimas opciones 0 significa desactivado y 1 activado):

```
[DIRECTORIO]
dir=C:\Clase\Quinto\Sistemas&Informaticos
[NUEVO]
tmp=15000
[INICIO]
pnt=1
[OPENGL]
pol=1
bln=1
```

Estas opciones de configuración se leen al iniciarse la aplicación mediante llamadas a la función *TmainForm.leerOpciones*. En caso de no encontrar el archivo de configuración la función crea uno nuevo con unos valores por defecto e indica al usuario que debería reconfigurar los parámetros.

Formulario de visualización

El formulario de visualización (*TWAVfileForm*) recibe este nombre al ser el formulario contenedor del contexto de representación gráfica de OpenGL. Sin embargo, además de realizar la visualización de las muestras, en este formulario se realizan o controlan otras muchas operaciones sobre el archivo como la reproducción, la edición o el control de cambios sobre el archivo.

Cada formulario de visualización es un formulario hijo (*child*) del formulario principal y por tanto va a aparecer dentro del contexto de éste: cuando se maximice o minimice lo hará dentro del entorno y si el formulario principal se cierra la harán también todos sus hijos.

El formulario es el contenedor del contexto de representación gráfica de OpenGL. OpenGL sólo maneja un contexto de representación en cada momento por lo que, dado el hecho de que puede haber varios formularios de visualización abiertos (y por tanto varios contextos de representación), es necesario que cada vez que se active el formulario (el resto pasa a un segundo plano) se indique el contexto de representación actual al motor de OpenGL. Esta acción se realiza con la instrucción `wglMakeCurrent(HDC,HGLRC)` que se realiza cada vez que se hace un refresco de la pantalla. `HDC` (*Handle Device Context*) fija el "área cliente" de la ventana que queremos pintar y `HGLRC` (*Handle OpenGL Rendering Context*) fija el contexto en el que OpenGL dibujará sus gráficos.

La mayor parte del control a nivel gráfico se realiza con el ratón. Aunque se han diseñado controles gráficos estos realmente sólo son una visualización de las acciones que se realizan con el ratón en función de su posición y estado (a través de los eventos *TWAVfileForm.FormMouseDown*, *TWAVfileForm.FormMouseMove*, *TWAVfileForm.FormMouseUp*), es decir, aunque se pulse con el ratón sobre un control gráfico la acción a realizar está determinada por la posición del ratón sobre la pantalla. En el caso de que no se pinche sobre ningún control gráfico se realiza el proceso de selección de muestras utilizando las funciones de conversión de coordenadas de pantalla a coordenadas del campo de visualización (*TWAVfileForm.convCoord*XX)

Los controles gráficos se dibujan con superposición de polígonos convexos para que OpenGL realice correctamente el coloreado de los mismos. Esta característica es la que hace que estos controles sean los más afectados por la utilización o no de las opciones de suavizado de polígonos y blending.

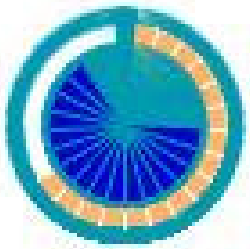
Visualización

Los trabajos de representación de las muestras son realizados en su mayor parte por la clase *Visualizador* como ya se ha explicado en el capítulo de representación gráfica. Para poder trabajar el formulario mantiene un puntero al visualizador correspondiente al archivo. Al nivel de este formulario no se realiza trabajo de representación sino de control de la misma, esta control incluye la selección de los valores de las escalas, la forma de la onda o el rango de muestras a visualizar.

En cuanto a las dos primeros, se limita a ofrecer dos posibilidades para cada uno: para el primero se puede elegir entre valores reales de muestras o valores relativos en porcentaje y para el segundo entre la forma real de la onda o la forma simulada. La forma simulada de la onda, una onda sinusoidal de la misma duración que el archivo, se introdujo en las primeras versiones para poder manejar archivos grandes sin verse afectado por los tiempos de espera debidos a los cálculos de la representación gráfica. Al mejorar el rendimiento de estos cálculos esta función ha perdido su utilidad aunque se mantiene.

El control del zoom de la visualización se realiza a través de menú permitiendo las opciones típicas de ampliación y reducción, para ello únicamente se realizan llamadas a la función *Visualizador.setRango*.

Para el control del rango de muestras se ha diseñado también un control gráfico (*VControl*) que permite al usuario un control rápido y sencillo. Este control se ubica siempre en la esquina superior derecha del formulario y muestra información no sólo de la representación sino también de la reproducción en curso. Aparece como un círculo en el que la banda externa de color anaranjado indica el porcentaje de archivo reproducido mientras que la parte interna de color azul indica la sección de archivo representado siendo el inicio la parte superior del círculo y siguiendo el sentido de las agujas del reloj hasta el final otra vez en la parte superior. Esta sección azul es móvil y se puede desplazar con el ratón pinchando sobre ella y moviéndole ratón alrededor del centro del círculo. Al desplazarla se cambia el rango de muestras visualizado. Al realizar el desplazamiento en función del ángulo y no con respecto a la posición del ratón se pueden obtener diferentes velocidades de desplazamiento para una mayor o menor precisión (con el ratón cerca del centro del círculo el desplazamiento es grande para un movimiento pequeño del ratón mientras que a gran distancia del centro el desplazamiento es mucho menor recorriendo la misma distancia con el ratón)



- Ilustración 5: Control gráfico de visualización

Reproducción

Para poder llevar a cabo el control de la reproducción el formulario mantiene un puntero a la clase *Reproductor* que es la que se va a encargar de manejar el archivo en cuanto a la reproducción. El control sobre la reproducción se basa en una serie de funciones (*TWAVfileForm.repPlayPause* <play / pausa>, *TWAVfileForm.repStop* <detener>, *TWAVfileForm.repPrev* <rebobinar>, *TWAVfileForm.repNext* <avanzar todo>, *TWAVfileForm.repBack* <paso atrás>, *TWAVfileForm.repStep* <paso adelante>, *TWAVfileForm.repRecord* <grabar>) que envían las órdenes necesarias al reproductor.

El control de reproducción se puede llevar a cabo a través de menú o a través del control gráfico diseñado para ello (*SControl*). Este control es similar al anterior y se ubica siempre en la esquina inferior izquierda de la ventana, con forma circular que

contiene siete botones con todas las operaciones posibles apareciendo resaltadas aquellas que se pueden realizar. El botón central de reproducción cambia en función del estado de la reproducción, así mientras se está reproduciendo el archivo el botón se muestra como el botón de pausa mientras que cuando la reproducción está detenida o pausada aparece como el botón de inicio de reproducción.

• Ilustración 6: Control gráfico de reproducción

Control de cambios

Es desde este formulario desde donde se lanzan todas las funciones que introducen los efectos de modificación de audio (ecos, reverberación, etc. en la clase *Efectos*) además de servir de puente entre el formulario principal y las funciones de copiado y pegado de muestras en los archivos transmitiendo los arrays de muestras en uno u otro sentido. Mantiene la integridad del archivo original frente a los cambios hasta que estos se guarden y ofrece también la posibilidad de deshacer el último cambio efectuado en el archivo.

Para poder realizar este control sobre los cambios sobre archivos se utilizan archivos auxiliares y marcas de modificación. Las variables implicadas en el control de cambios son:

Variable	Tipo	Funcionalidad
`modificado`	`bool`	Marca que indicar si el fichero original ha sido modificado
`nombreOrig`	`AnsiString`	Ruta completa del archivo original
`nombreFich`	`AnsiString`	Ruta completa del archivo en actualmente en uso
`nombreUndo`	`AnsiString`	Ruta completa del archivo anterior al último cambio

• Tabla 26: Variables de control de cambios

Partiendo de un archivo original, la realización de cualquier modificación en el archivo pasa por una llamada a la función *TWAVfileForm.modificarFichero* que toma el archivo destino de la modificación y lo renombra con el mismo nombre del archivo original pero con extensión ".tmp". También marca el archivo como modificado y crea una copia del archivo original con extensión ".und" que será el archivo para deshacer cambios. En sucesivos cambios en el archivo se renombrará el archivo .tmp a .und sobrescribiendo el anterior y se guardará el archivo resultante del cambio con el mismo nombre que el .tmp anterior.

Al guardar el archivo con la función *TWAVfileForm.guardarFichero* se sobrescribe el archivo original con el .tmp y se marca el archivo como no modificado. Si al cerrar el formulario no se ha guardado el archivo se eliminan todos los archivos temporales .tmp y .und.

La función *TWAVfileForm.deshacer* es la que se encarga de deshacer la última modificación realizada en el archivo. Para ello simplemente copia el contenido del archivo .und sobre el archivo actualmente en uso. Al contener los datos del archivo en uso anteriores a la realización del cambio el efecto es el de recuperar el estado del archivo en uso anterior al cambio.

La utilización de varios archivos temporales por cada archivo abierto en el entorno y el gran tamaño de los archivos WAV sin compresión hacen que los requisitos de disco de este entorno puedan llegar a ser altos si se trabaja con varios archivos de gran tamaño simultáneamente.

Temporización y formato

El formulario de visualización posee un panel de información de temporización y formato en la esquina superior izquierda en el que se muestra el tiempo de reproducción (desde el comienzo o hasta el final), el tiempo total del archivo y de la parte seleccionada y datos sobre la frecuencia de muestreo, el número de bits por canal y el número de canales del archivo. También ofrece la posibilidad de acceder a datos más detallados del archivo a través del botón “Propiedades” o de cambiar el formato del archivo a través del botón “Formato”.

Los datos de temporización se almacenan en las variables `tiempoI`, `tiempoF` y `tiempoT`, las dos primeras almacenan en cada momento el tiempo inicial y final en segundos de la parte seleccionada mientras que la última guarda la duración total del archivo en segundos.

Los datos de formato se leen directamente del archivo al abrir el formulario y no se modifican hasta que se pulse el botón “Formato” y se realice una llamada a la función *WaveFileFunc.CambiarFormato* a la que se le pasa el nuevo formato como parámetro por lo que nunca es necesario almacenar esos datos.

Los datos más específicos que se presentan al abrir el formulario de propiedades se leen directamente del archivo. Dado que estos datos se utilizarán en pocas ocasiones tampoco será necesario almacenarlos en variables del formulario.

Apéndice
A

Apéndice A : OpenGL

Descripción

OpenGL es una interfaz software de hardware gráfico. Es un motor 3D cuyas rutinas están integradas en tarjetas gráficas 3D. OpenGL posee todas las características necesarias para la representación mediante computadoras de escenas 3D modeladas con polígonos, desde el pintado más básico de triángulos, hasta el mapeado de texturas, iluminación o NURBS.

La compañía que desarrolla esta librería es Sillicon Graphics Inc (SGI), en pro de hacer un estándar en la representación 3D gratuito y con código abierto (open source). Esta basado en sus propios sistemas operativos y lenguajes IRIS, de forma que es perfectamente portable a otros lenguajes. Entre ellos C, C++, etc. y las librerías dinámicas permiten usarlo sin problema en Visual Basic, Visual Fortran, Java, etc.

OpenGL soporta hardware 3D, y es altamente recomendable poseer este tipo de hardware gráfico. Si no se tiene disposición de el, las rutinas de representación correrán por software, en vez de hardware, decrementando en gran medida su velocidad.

OpenGL como una máquina de estados

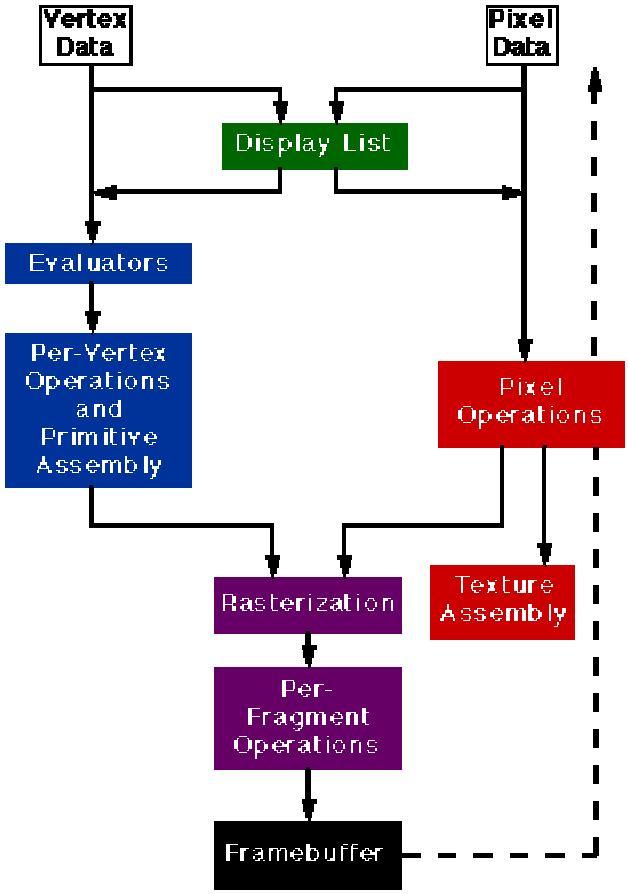
OpenGL es una máquina de estados. Cuando se activan o configuran varios estados de la máquina, sus efectos perdurarán hasta que sean desactivados. Por ejemplo, si el color para pintar polígonos se pone a blanco, todos los polígonos se pintarán de este color hasta cambiar el estado de esa variable. Existen otros estados que funcionan como booleanos (on ó off, 0 ó 1). Estos se activa mediante las funciones `glEnable` y `glDisable`.

Todos los estados tienen un valor por defecto, y también alguna función con la que conseguir su valor actual. Estas pueden ser mas generales, del tipo `glGetDoublev`() o `glIsEnabled`(), que devuelven un flotate y un valor booleano, respectivamente; o mas especificas, como `glGetLight`() o `glGetError()`, que devolverían una luz o un código de error..

El pipeline de renderizado de OpenGL

La mayor parte de las implementaciones de OpenGL siguen un mismo orden en sus operaciones, una serie de plataformas de proceso, que en su conjunto crean lo que se suele llamar el “OpenGL Rendering Pipeline”.

El siguiente diagrama describe el funcionamiento del pipeline:



• Ilustración 7: Funcionamiento del pipeline de OpenGL

En este diagrama se puede apreciar el orden de operaciones que sigue el pipeline para renderizar. Por un lado tenemos el "vertex data", que describe los objetos de nuestra escena, y por el otro, el "píxel data", que describe las propiedades de la escena que se aplican sobre la imagen tal y como se representa en el buffer. Ambas se pueden guardar en una "display list", que es un conjunto de operaciones que se guardan para ser ejecutadas en cualquier momento.

Sobre el "vertex data" se pueden aplicar "evaluators", para describir curvas o superficies parametrizadas mediante puntos de control. Luego se aplicaran las "per-vertex operations", que convierten los vértices en primitivas. Aquí es donde se aplican las transformaciones geométricas como rotaciones, translaciones, etc., por cada vértice. En la sección de "primittive assembly", se hace clipping de lo que queda fuera del plano de proyección, entre otros.

Por la parte de "píxel data", tenemos las "píxel operations". Aquí los píxeles son desempaquetados desde algún array del sistema (como el framebuffer) y tratados (escalados, etc.). Luego, si estamos tratando con texturas, se preparan en la sección "texture assembly".

Ambos caminos convergen en la "Rasterization", donde son convertidos en fragmentos. Cada fragmento será un píxel del framebuffer. Aquí es donde se tiene en cuenta el modelo de sombreado, la anchura de las líneas, o el antialiassing.

En la última etapa, las "per-fragmet operations", es donde se preparan los texels (elementos de texturas) para ser aplicados a cada píxel, la fog (niebla), el z-buffering, el blending, etc. Todas estas operaciones desembocan en el framebuffer, donde obtenemos el render final.

Sintaxis de OpenGL

Todas las funciones de OpenGL comienzan con el prefijo "gl" y las constantes con "GL_". Como ejemplos, la función `glClearColor()` y la constante `GL_COLOR_BUFFER_BIT`.

En muchas de las funciones, aparece un sufijo compuesto por dos caracteres, una cifra y una letra, como por ejemplo `glColor3f()` o `glVertex3i()`. La cifra simboliza el número de parámetros que se le deben pasar a la función, y la letra el tipo de estos parámetros.

En OpenGL existen 8 tipos distintos de datos, de una forma muy parecida a los tipos de datos de C o C++. Además, OpenGL viene con sus propias definiciones de estos datos (`typedef` en C). Los tipos de datos:

Sufijo	Tipo de dato	Correspondencia de tipo de C++	Definición en OpenGL
B	Entero 8-bits	signed char	GLbyte
S	Entero 16-bits	short	GLshort
I	Entero 32-bits	int o long	GLint, GLsizei
F	Punto flotante 32-bits	float	GLfloat, GLclampf
D	Punto flotante 64-bits	double	GLdouble, GLclampd
Ub	Entero sin signo 8-bits	unsigned char	GLubyte, GLboolean
Us	Entero sin signo 16-bits	unsigned short	GLushort
Ui	Entero sin signo 32-bits	unsigned int	GLuint,GLenum, GLbitfield

• Tabla 27: Tipos de datos de OpenGL

Librerías relacionadas con OpenGL

OpenGL contiene un conjunto de primitivos comandos, a muy bajo nivel. Además la apertura de una ventana en el sistema grafico que utilicemos (win32, X11, etc.) donde pintar no entra en el ámbito de OpenGL. Por eso las siguientes librerías son muy utilizadas en la programación de aplicaciones de OpenGL:

- OpenGL Utility Library (GLU): contiene bastantes rutinas que usan OpenGL a bajo nivel para realizar tareas como transformaciones de matrices para tener una orientación especifica, subdivisión de polígonos, etc.

- GLX y WGL: GLX da soporte para maquinas que utilicen X Windows System, para inicializar una ventana, etc. WGL seria el equivalente para sistemas Microsoft.

- OpenGL Utility Toolkit (GLUT): es un sistema de ventanas, escrito por Mark Kilgard, que seria independiente del sistema usado, dándonos funciones tipo abrir_ventana().

Apéndice
B

Apéndice B : Clase WaveFile

Descripción

Para la manipulación de archivos de audio a bajo nivel se han utilizado unas clases en lenguaje C++ que fueron especialmente diseñadas para ello, estas clases son la clase *WaveFile* y sus clases auxiliares *RiffFile* y *RiffChunk*.

La clase *WaveFile* es la clase encargada de realizar la manipulación a bajo nivel de archivos de audio digital sin compresión en formato WAV. Esta clase desarrolla una serie de métodos de lectura y escritura de los archivos de audio que permiten modificarlos y obtener toda la información útil de ellos de manera fácil y eficiente.

Operaciones

La clase *WaveFile* permite realizar un gran número de operaciones sobre los archivos de manera eficiente. Las más importantes se describen a continuación.

Operaciones de apertura y cierre

- *OpenRead*: Apertura del archivo en modo lectura.
- *OpenWrite*: Apertura del archivo en modo escritura.
- *Close*: Cierre del archivo. Si el archivo era de escritura y se ha modificado comprueba que la información de la cabecera sea correcta.

Operaciones de lectura y escritura

- *ResetToStart*: Desplaza el puntero de lectura-escritura al principio de la sección de datos.
- *MovePointer*: Desplaza el puntero de lectura-escritura a la muestra indicada por parámetro.
- *ReadSample*: Lee la siguiente muestra apuntada por el puntero de lectura-escritura. Es un método sobrecargado para cada uno de los tipos de muestra que se pueden leer (8bits mono, 8bits estéreo, etc.)
- *WriteSample*: Escribe una muestra en la posición apuntada por el puntero de lectura-escritura. Es un método sobrecargado para cada uno de los tipos de muestra que se pueden leer (8bits mono, 8bits estéreo, etc.)
- *ReadSamples*: Lee una serie de muestras a partir de la posición apuntada por el puntero de lectura-escritura. Es un método sobrecargado para cada uno de los

tipos de muestra que se pueden leer (8bits mono, 8bits estéreo, etc.). Es más eficiente usar este método para leer una serie de muestras que usar muchas veces el método *ReadSample* debido al uso de buffers.

- *WriteSamples*: Escribe una serie de muestras a partir de la posición apuntada por el puntero de lectura-escritura. Es un método sobrecargado para cada uno de los tipos de muestra que se pueden leer (8bits mono, 8bits estéreo, etc.). Es más eficiente usar este método para escribir una serie de muestras que usar muchas veces el método *WriteSample* debido al uso de buffers.

Operaciones sobre la cabecera

Hay toda una serie de métodos que permiten acceder o modificar la información almacenada en la cabecera del archivo o incluso generar otro tipo de información a partir de la de la cabecera.

La información a la que se puede acceder es la siguiente:

A) Formato (*FormatType*)

B) Compresión (*Compressed*)

C) Número de canales (*NumChannels*)

D) Frecuencia de muestreo (*SampleRate*)

E) Bytes por segundo (*BytesPerSecond*)

F) Bytes por muestra (*BytesPerSample*)

G) Bits por canal (*BitsPerChannel*)

H) Número de muestras (*NumSamples*)

I) Duración en segundos (*NumSeconds*)

J) Bytes de la sección de datos (*DataLength*)

Operaciones de formato

- *FormatMatches*: Comprueba si el formato de el archivo coincide con el del archivo que se le pasa por parámetro.

- *CopyFormatFrom*: Copia en la cabecera del archivo la información de formato de la cabecera del archivo que se le pasa por parámetro.

- *SetupFormat*: Modifica la información del formato estableciendo la que se le pasa por parámetro.

Operaciones sobre varios archivos

- *CopyFrom*: Copia el contenido de la sección de datos del archivo que se le pasa por parámetro a la sección de datos.