AUDIO ESPACIAL CON OPENAL: USO DE OPENAL-SOFT PARA LA REALIZACIÓN DE EFECTOS Y USO DE FICHEROS MULTICANAL.

Giovanny Tipantuña Topanta
giotitoa@fiv.upv.es

Resumen

Este documento es una memoria de trabajo donde se recogen los apartados más significativos surgidos durante la realización de la aplicación para el tratamiento de audio con biblioteca de funciones de OpenAL. Además sirve a modo de guía para su posterior entendimiento a la hora de realizar ampliaciones a las distintas partes que se han integrado en la aplicación final.

Palabras Clave: OpenAL, OpenGL, Echo, Reverb, doopler, audio, multicanal, 5.1, AL_EXT_MCFORMATS, alloopback, alAuxiliaryEffectSloti.

1 HERRAMIENTAS SOFTWARE Y HARDWARE

1.1 MÁQUINA VIRTUAL

El primer paso a realizar, es decidir la plataforma en la que desarrollaremos la aplicación. Actualmente, la aplicación fue construida en una máquina virtual que ejecuta **Kubutu12.4.5** como sistema operativo invitado, mientras que el sistema operativo anfitrión fue Windows XP 32 bits Service Pack 3. La elección de Kubutu 12.4.5 fue realizada para minimizar el espacio requerido en disco ya que investigando en la red, vimos que ocupaba 4.5 GB una vez instalado y se daba soporte hasta 2017, el cual nos asegura que teníamos acceso a los repositorios de canonical para descargar e instalar las bibliotecas con la típica orden "*sudo apt-get install openal-dev*".

Las características hardware del equipo en el que se ha probado la aplicación dispone de **2.5 GB de memoria *ram*** y un procesador de dos núcleos Genuine Intel(R) CPU 1.60 GHz.

En este punto cabe mencionar la importancia de decidir el entorno en el que se quiere desarrollar o del que se dispone para ello, tanto hardware como software. Ya que ejecutar la aplicación en una máquina virtual puede ofrecer menos calidad que si se ejecuta en una maquina normal, puesto que hay que dedicar memoria y tiempo de cpu a varias aplicaciones que están presentes a la vez. Como efecto positivo, se obliga al programador a optimizar los recursos a la hora de desarrollar, ya que si se consigue una ejecución buena en estas condiciones, en condiciones mejores en cuanto a hardware, la ejecución será muy buena.

En un primer momento, se optó por realizar la aplicación en entorno Linux bajo máquina virtual de vmware Workstation 6. Sin embargo, al abrir el sistema instalado Kubutu 12.4.5 se encontró que no había soporte para expandir la pantalla del sistema invitado, la cual inicialmente se creaba con una resolución de 800x600 píxeles, a la resolución total que el equipo podía mostrar 1200x800 píxeles.

Este problema dificultaba el desarrollo y como consecuencia se tuvo que cambiar de plataforma para la máquina virtual a VirtualBox, la cual actualmente se ofrece gratis. La versión utilizada de VirtualBox fue **VirtualBox 4.3.26 for Windows hosts** x86/amd64 que se puede conseguir en el **enlace [1].**

El sistema operativo Kubuntu 12.4.5 fue descargado en formato **.iso** del repositorio de versiones que se puede encontrar en el **enlace [2]**. Específicamente el descargado fue **kubuntu-12.04.5-desktop-i386** PC intel x86 desktop cd**.**

Para la instalación es necesario crear una máquina limpia, con características para alojar un sistema Linux. Los pasos se pueden encontrar en la red o simplemente siguiendo las instrucciones que presentan las ventanas al crear una maquina nueva. Lo importante es dedicarle cantidad de memoria ligeramente por encima de **512MB** para la instalación. Para instalar desde la iso es necesario acceder a la maquina creada en el apartado "almacenamiento" y colocar la iso como fichero a ser leído y luego arrancar la máquina.

Una vez instalado el sistema operativo, el siguiente paso es instalar las bibliotecas necesarias para el tratamiento del audio con OpenAL por un lado, y el tratamiento de gráficos con OpenGL por otro lado.

Llegados a este instante es necesario establecer la cantidad de memoria ram y **memoria gráfica** que se dedica a la máquina virtual. La aplicación fue comprobada con 128MB de memoria gráfica y una memoria base de 1900MB en la configuración de la máquina virtual.

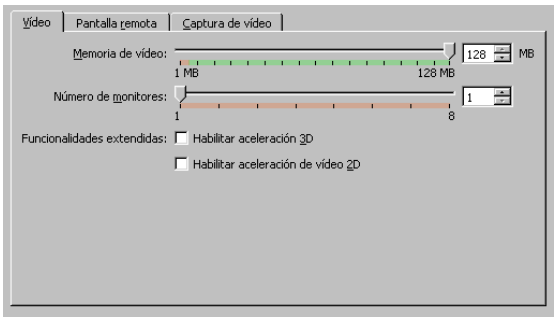

Figura 1. Configuración memoria gráfica.

Uno de los problemas encontrados en la configuración que debería ser tenido en cuenta es que establecer el parámetro "dispositivo apuntador" en "ratón/ps2" puede hacer que el sistema invitado se ejecute muy lentamente si no existe "integración directa". En nuestro caso, tuvimos que elegir, **"dispositivo apuntador -> tableta USB".**

Otro aspecto importante que puede agilizar la ejecución de la máquina virtual es desactivar todos aquellos aspectos de animaciones de escritorio y dejarlo todo en colores planos. Es decir, buscar en las opciones, y minimizar las opciones gráficas al mínimo. Reducir la resolución de pantalla del sistema invitado también mejora la rapidez.

A pesar de que esta opción deshabilita la rueda del ratón en el sistema invitado ha sido la más eficiente. Así mismo, es necesario mencionar que debido a la configuración de la arquitectura del procesador del equipo, VirtualBox no nos permitía emplear los 2 núcleos, sin embargo ha sido suficiente con uno para la realización del proyecto. Además investigando en la red se dedujo que para equipos lanzados con posterioridad al 2004 está disponible la opción "**virtualización**", lo cual hará que se empleen varios núcleos en la máquina virtual.

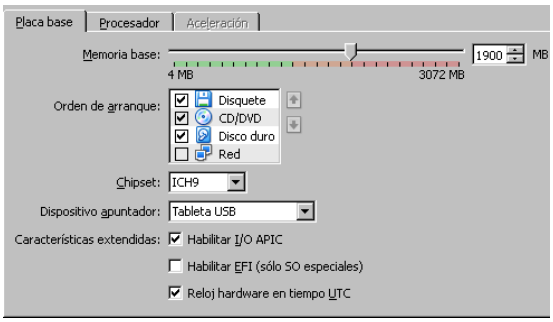

Figura 2. Configuración de memoria base y dispositivo apuntador de la máquina virtual.

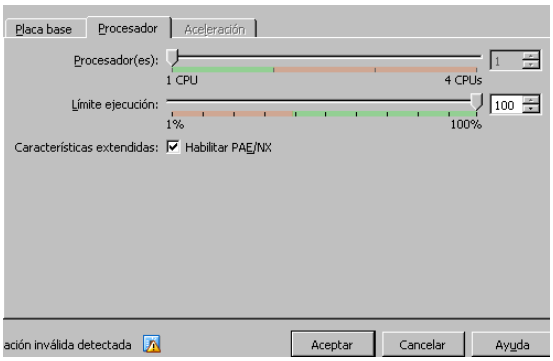

Figura 3. Configuración de *cpu* de la máquina virtual.

Otro aspecto a mencionar en este punto, es el de la configuración de audio de la máquina virtual, la cual se puede ver en la figura 4.

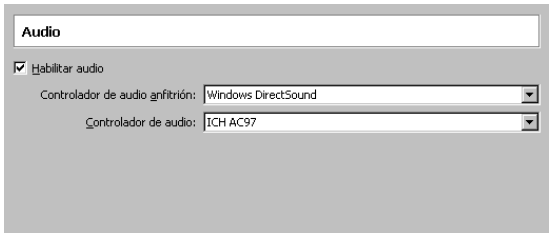

Figura 4. Configuración de audio de la máquina virtual.

Debemos mencionar que la opción ICH AC97 es la que se empleado ya que el equipo no dispone de tarjeta de sonido 5.1, sin embargo VirtualBox ofrece la posibilidad de hacer que el sistema invitado "vea" una tarjeta 5.1.

Finalmente en este aparado, mencionamos que se halló una solución encontrada para el *cliping* del sonido cuando se mueve la fuente mientras se está reproduciendo, la cual ha sido reducir el número de eventos que el teclado envía cuando se mantiene pulsada la tecla. Es decir, dentro de la *maquina virtual invitada*, accedemos al teclado y reducimos las pulsaciones por segundo a aproximadamente 15.

1.2 BIBLIOTECAS NECESARIAS: OpenAL, alut, OpenGL. freeglut, GLUI, SLD, OpenAL-soft-1.6

Esta sección la debemos comenzar mencionando que una vez hayamos abierto la maquina virtual y estemos dentro, es necesario disponer de las utilidades de GNU C, ya que muchas de las bibliotecas para programación C/C++ utilizan como base ésta y nuestra aplicación se encuentra actualmente escrita en C. Para ello podemos lanzar desde consola las siguientes instrucciones:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install build-essential
$ gcc -v
$ make -v
```

En cuanto a la biblioteca para **OpenGL** y **freeglut** (la versión de código abierto de glut en linux), se puede encontrar una guía extensa de instalación en el **enlace [3]**. Pasamos directamente a resumir a continuación las órdenes que se deben lanzar en la consola de Linux para conseguir descargar e instalar las dos bibliotecas.

Para OpenGL:

```
$ sudo apt-get install mesa-common-dev
```

Para freeglut:

```
$ sudo apt-get install freeglut3-dev
```

El correcto funcionamiento se puede comprobar compilando el ejemplo sencillo del **enlace [3].**

En cuanto a la biblioteca de audio OpenAL, la instalación fue realizada mediante el "*gestor de paquetes de muon*" que se dispone Kubuntu12.4.5. En él hemos escrito "*openal*" y hemos marcado aquellas referentes a desarrollo con sufijo "*dev*".

También se pueden lanzar las siguientes órdenes en la consola:

```
$ sudo apt-get install libopenal0a
libopenal-dev

$ sudo apt-get install libalut0 libalut-dev
```

Puede ser interesante el empleo de la biblioteca de funciones **GLUI** para crear una interfaz gráfica con la que proporcionar al usuario elementos como sliders verticales u horizontales para modificar los parámetros, por ejemplo, de los efectos. Esta opción se intentó introducir a la aplicación sin embargo habría que adaptar el código a C++ ya que GLUI está escrita en este lenguaje y utiliza como base glut o freeglut. Esta biblioteca facilita la creación de botones, sliders, etc. Actualmente, el menú desplegable que está anclado al botón derecho del ratón se realiza con funciones de la biblioteca glut.

La instalación de GLUI, puede realizarse lanzando las instrucciones desde la consola:

```
$ sudo apt-get install libglui2c2
$ sudo apt-get install libglui-dev
```

Para ver la versión de GLUI:

```
$ pkg-config glui --modversion
```

Hay que tener en cuenta la inicialización de *freeglut*, necesita que se le pase parámetros para que no de fallo en tiempo de ejecución:

```
glutInit(&argc, argv);
```

Para comprobar el funcionamiento de esta biblioteca, se puede encontrar un ejemplo en el directorio "*\codigo\partes\interfazGrafica*".

Por otro lado, la biblioteca **SDL** *(simple direct media layer)*, puede ser significativa desde el punto de vista que permite realizar manipulación sobre audio y video mediante funciones disponibles, como por ejemplo para extraer información referente al formato, número de bytes, muestras, etc. Esta biblioteca ha ido necesaria para compilar el ejemplo que realiza el tratamiento básico de la función "*alloopback()*" la cual se intentó emplear para abordar el apartado de la "*escritura de un fichero 5.1*" mediante un "reenvío" de la salida del audio y que actualmente no está disponible en la aplicación.

La copia local del ejemplo que emplea la función "*alloopback()"*, la cual pertenece a la biblioteca OpenAL-soft-1.6, se puede localizar en el directorio "*codigo\partes\multicanal\escritura\alloopbackINFO* " y el fuente original en el **enlace [4].**

Como vemos este ejemplo emplea OpenAL-soft-1.6 y SDL, por tal motivo pasamos a mencionar las órdenes para instalar estas bibliotecas:

Para SDL:

```
$ sudo apt-get install libsdl1.2-dev

$ pkg-config sdl --modversion
--cflags -libs

$ sudo apt-get install libsdl-
image1.2-dev
```

Para la biblioteca **OpenAL-soft-1.6** en la cual disponemos de los binarios de la función "*alloopback()*" necesaria para hacer un "*reenvío*" directo del audio renderizado de salida de OpenAL hacia un buffer para "almacenar" lo que se está oyendo en el escenario, el primer paso es descargar los ficheros que nos permitirán construir la biblioteca dinámica. Para ello, acudimos al **enlace [5]** y

descargamos "*Download -> openal-soft-1.16.0.tar.bz2*".

Como se puede observar en las instrucciones del **enlace [5]**, para crear los binarios de esta nueva biblioteca es necesario disponer de la herramienta **CMake**. Para abreviar la instalación pasaremos directamente han mostrar las órdenes, el orden en el que se han lanzado tras descargar y descomprimir lo mencionado anteriormente, y parte de la salida por consola para la correcta instalación de CMake y OpenAL-soft-1.6 en un directorio sin sobrescribir la versión de OpenAL ya instalada.

Pasos a seguir:

```
gio@gio-VirtualBox:~/Escritorio/imd_proyecto/openal-soft-1.16.0$ cd build/

gio@gio-VirtualBox:~/Escritorio/imd_proyecto/openal-soft-1.16.0/build$ cmake ..

El programa «cmake» no está instalado.
Puede instalarlo escribiendo:
sudo apt-get install cmake

gio@gio-VirtualBox:~/Escritorio/imd_proyecto/openal-soft-1.16.0/build$ sudo apt-get install cmake

[sudo] password for gio:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
...
ldconfig deferred processing now taking place

gio@gio-VirtualBox:~/Escritorio/imd_proyecto/openal-soft-1.16.0/build$ cmake ..

-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
...
-- Configuring done
-- Generating done
-- Build files have been written to: /home/gio/Escritorio/imd_proyecto/openal-soft-1.16.0/build

gio@gio-VirtualBox:~/Escritorio/imd_proyecto/openal-soft-1.16.0/build$ ll

total 112
drwxrwxr-x  4 gio gio  4096 abr 17 16:13 ./
drwxrwxr-x 11 gio gio  4096 ago 15  2014 ../
-rw-rw-r--  1 gio gio 20218 abr 17 16:13 CMakeCache.txt
drwxrwxr-x 10 gio gio  4096 abr 17 16:13 CMakeFiles/
-rw-rw-r--  1 gio gio  7098 abr 17 16:13 cmake_install.cmake
-rw-rw-r--  1 gio gio  4835 abr 17 16:13 config.h
-rw-rw-r--  1 gio gio     0 ago 15  2014 .empty
-rw-rw-r--  1 gio gio 54019 abr 17 16:13 Makefile
-rw-rw-r--  1 gio gio   261 abr 17 16:13 openal.pc
drwxrwxr-x  3 gio gio  4096 abr 17 16:13 utils/

gio@gio-VirtualBox:~/Escritorio/imd_proyecto/openal-soft-1.16.0/build$ make

Scanning dependencies of target common
[  1%] Building C object CMakeFiles/common.dir/common/atomic.c.o
[  3%] Building C object CMakeFiles/common.dir/common/rwlock.c.o
[  5%] Building C object CMakeFiles/common.dir/common/threads.c.o
[  7%] Building C object CMakeFiles/common.dir/common/uintmap.c.o
Linking C static library libcommon.a
...
[100%] Built target openal-info
```

Finalmente en el directorio *build* se habrán generado los binarios y deberíamos encontrar: `libopenal.so libopenal.so.1 libopenal.so.1.16.0`

Actualmente, se dispone de una versión construida para Linux32 bits, en el directorio "*openal-soft-1.16.0*", y a su vez se encuentra el fichero .zip que contiene los ficheros descargados para la construcción de las bibliotecas.

A partir de este momento, ya podemos decirle al compilador que en lugar de tomar los binarios de la versión instalada de OpenAL, tome los binarios del directorio *build* de openal-soft-1.6 y a su vez que los ficheros de cabecera *.h* los debe tomar del directorio *include* también del directorio openal-soft-1.6. El direccionamiento para esto depende de donde se encuentren los ficheros a ser compilados con referencia al directorio openal-soft-1.6, por ello hemos dejado la parte de inicial de cada fichero *.c* as ser compilado un comentario con la orden con la que fue comprobada la compilación.

En resumen, para compilar el ejemplo *loopbackMOD.c* del directorio "*codigo\partes\multicanal\escritura\alloopbackINFO*" empleando openal-soft-1.6 es necesario incluir el directorio *build* al path de bibliotecas dinámicas con la orden:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:../../../../../openal-soft-1.16.0/build/
```

Lo siguiente es compilar el fichero estableciendo los *flags* de inclusión de ficheros de cabecera con "**-I**" delante de la dirección donde queremos leerlos, así mismo "**-L**" delante de la dirección donde se

encuentran los binarios a ser empleados. Y por último le decimos al compilador que de esos "*includes*" y esos "*binarios*", queremos lo referente a *alut, openal, y SDL*, colocando un –l delante para indicar "*libalut, libopenal, libSDL*". La orden de compilación del ejemplo mencionado queda:

```
$ gcc loopbackMOD.c -I/usr/include/SDL/ -
I../../../../../openal-soft-1.16.0/include/
-L../../../../../openal-soft-1.16.0/build/ -
lalut -lopenal -lSDL -o loopbackMOD
```

Para lanzar la ejecución es necesario comprobar que el ejecutable va a emplear los binarios del directorio *build* con la orden:

```
$  ldd loopbackMOD
```

Y deberíamos encontrar una línea que indica "libpenal => ../../../../../openal-soft-1.16.0/build/"

También ha hecho falta añadir el *include* de *helpers.c* en *alhelpers.h*:

```
#include "alhelpers.c"
```

A partir de este momento no debería haber inconveniente para realizar modificaciones sobre este ejemplo.

2 DESARROLLO DE LAS PARTES

2.1 MÁQUINA DE ESTADOS

Para evitar que la aplicación caiga en un estado no contralado, se ha definido una máquina de estados en el fichero "*estados.h"* donde se define de forma eficiente mediante un dato de tipo de enumerado, 3 estados principales: *DOPPLER, EFECTOS, MULTICANAL.*

Dependiendo del estado, desde el fichero "*main.c"* se atienden los eventos de teclado, y ratón, y en función de la tecla o de la opción de menú pinchada con el ratón se invoca a las funciones del modo respectivo para realizar modificaciones en las fuentes, efectos, etc.

El funcionamiento de esta pequeña máquina de estados se basa en el empleo del parámetro *estadoActual* para minimizar las instrucciones a ejecutar en cada modo. Por ejemplo, para el modo *DOPPLER* es necesario realizar las transformaciones de las fuentes (cubos de colores), pero en modo *EFECTOS,* no es necesario. El mismo método se ha empleado para el empleo de algunas teclas en modo *EFECTOS,* pues se han dejado implementadas las funciones correspondientes a modificaciones de parámetros de efectos como, y por ello hay que distinguir si estamos en modo *DOPPLER* para lanzar las funciones de este modo, o en modo *EFECTOS* para lanzar las funciones de este otro modo.

2.2 INTERACCIÓN MEDIANTE TECLADO

En primer lugar, el tratamiento de la interactividad del usuario mediante el teclado se ha optado por realizarlo en dos partes. La primera es capturar los eventos de teclado en la función "*keyboard()*" del fichero "*main.c*", y la segunda consiste en atender dichos eventos según la tecla pulsada y lanzar la ejecución de las funciones que modifican los valores de algunas de las propiedades de las fuentes dependiendo del modo en el que se encuentre la aplicación.

Las funciones que atienden los eventos de teclado para realizar modificaciones en las fuentes cuando estamos en modo *DOPPLER* se encuentran definidas en el fichero denominado "*funciones.c*" el cual es incluido en la cabecera de "*main.c*".

Esto nos ha permitido observar la manera en la que OpenAL establece los valores de los identificadores de tipo "*ALuint*" cuando se crea un buffer o una fuente.

Por ejemplo, cuando se ejecuta la instrucción:

```
bufferNames[0] =
alutCreateBufferWaveform(ALUT_WAVEFOR
M_SINE, 262, 0.0, 1.0);
```

El valor que retorna la función "*alutCreatBufferWaveForm()*" es un identificador de buffer de OpenAL que contiene una forma de onda sintetizada. Este valor retornado no es fijo y depende de la ejecución, por ese motivo guardamos su valor en una variable del mismo tipo "*ALuint*", en este caso en la *posición 0* de "*bufferNames*".

Lo mismo ocurre con la creación de las fuentes cuando se lanza la instrucción:

```
alGenSources(NUM_SOURCES,
sourceNames);
```

Ahí, se crean un numero "*NUM_SOURCES*" de fuentes y se colocan los nombres de las fuentes (Identificadores de tipo *ALuint*) en el *Array* de nombre "*sourceNames*".

Es por este motivo, por el que al llamar a nuestras funciones que atienden eventos del teclado para modificar las fuentes, le tenemos que pasar el valor correcto del nombre de la fuente. Un ejemplo de esto

es la instrucción de parar la reproducción de la fuente:

```
stop(sourceNames[fuenteObjetivo]);
```

Donde *fuenteObjetivo* es un valor de posición de 0 a 4 dentro del *Array* porque tenemos 5 fuentes.

Puntos relevantes en la realización de las funciones:

- -En la función "*imprimirValoresFuente()*" se emplea el retorno de carro para machacar la última línea impresa y sobrescribirla, así no se producen saltos de línea excesivos.
- -En las funciones *mas__*y *menos__*se ha optado por extraer el valor actual de una determinada propiedad, almacenarlo en la variable correspondiente que luego será usada a la hora de imprimir, modificar el valor en la variable e insertar este nuevo valor modificado de la variable empleando *alSourcei(…,…,nuevoValor)*. Cada una de estas funciones, llama a "*imprimirValoresFuente()*"
- La función "*imprimirValoresFuente()*" ha sido modificada para que en lugar de consultar todos los valores de todas las fuentes, se imprima los últimos valores de la última *fuenteObjetivo* recogidos en las variables apropiadas para ello, lo cual alivia de carga al motor de OpenAL. Es decir, si se modifica el pitch de la *fuenteObjetivo* 1, esto se almacena en la variable "*p*" y se imprimen los valores anteriores *gain, vel, pos, dir, etc*, junto con el nuevo valor de *p*.
- -La forma de identificar la fuente a mover es mediante la variable "*fuenteObjetivo*" cuyos valores van de 0 a 4. Esto nos permite por un lado realizar comprobaciones para dibujar el área de influencia del la onda sonora en OpenGL:

```
...
if(fuenteObjetivo == 2){
/*pintaCircunferencia...*/
}
...
```

- Por otro lado nos permite consultar el identificador que el motor OpenAL ha asignado a la fuente, para realizar modificaciones a alguna propiedad de la fuente empleando ese identificador:

```
...
case 'V':
menosVel
(sourceNames[fuenteObjetivo]);
break;
…
```

2.3 INTERACCIÓN MEDIANTE RATÓN

Esta opción de la que se dispone en la aplicación ha sido generada con el propósito de mejorar la interacción, puesto que se dedican la mayoría de teclas clave para el modo *DOPPLER*, empleamos un menú desplegable para acceder y manipular elementos del modo *EFECTOS*.

En el fichero "*funcionesEfectos.c*" se encuentran las funciones que atienden los eventos del menú desplegable con el botón derecho del ratón para realizar la creación del contexto para este modo, las modificaciones en los parámetros cuando nos encontramos en modo *EFECTOS*, y la salida y liberación de recursos reservados al entrar en este modo.

La manera de crear un menú desplegable, es realizada, en resumen, empleando las funciones *glutAddMenuEntry()* para añadir una nueva entrada al menú y *glutAddSubMenu()* para crear un submenú.

En el fichero *main.c* vemos que es necesario guardar el identificador del menú creado cuando se invoca a la función *glutCreateMenu();* para crear un menú principal, o un submenú. Además es necesario, pasarle el nombre de la función que realizará la atención de los eventos que lleguen de ese menú o submenú.

Para entrar más en detalle, puesto que puede ser útil ampliar las funcionalidades del menú desplegable, vamos a explicar el funcionamiento sencillo de creación de más entradas en el menú, partiendo del siguiente fragmento de código recogido de *main.c*:

```
  menu = glutCreateMenu(entradaPrincipal);
  glutAddMenuEntry("Terminar Programa", 3);
  glutAddMenuEntry("-", 4);
  glutAddSubMenu("Activar Efectos",
submenu1);
  glutAddSubMenu("Efecto", submenu2);
  glutAddMenuEntry("Reproducir Marcados",
1);
  glutAddMenuEntry("Detener Marcados", 2);
  glutAddMenuEntry("-", 5);
  glutAddMenuEntry("Leer 5.1", 6);

  glutAttachMenu(GLUT_RIGHT_BUTTON);
```

Con esto se crean entradas en el menú en el orden que están escritas las instrucciones. Para cada *MenuEntry* se debe indicar el valor que se "*lanzará*" hacia la función "*EntradaPrincipal()*"cuando se pulse sobre dicha entrada. Por ejemplo Terminar programa lanzará el valor 3, y dentro de la función *EntradaPrincipal()* hay un *switch* con un *case 3:* donde se invoca a la función de finalización del programa.

Para el caso de *SubMenu* lo que se hace es pasar un identificador del submenú. Por ejemplo, en:

```
  glutAddSubMenu("Activar Efectos",
submenu1);
```

Se establece que al pasar por encima de la entrada "*Activar Efectos*" se debe desplegar un nuevo submenú cuyo identificador es el valor "*submenu1*"

2.4 MOVER Y ROTAR

Para realizar la rotación de una fuente o del oyente se ha optado por una solución bastante acertada, en la que se emplea un *Array* donde almacenamos el valor del ángulo de giro de cada fuente:

```
ALfloat angulo[NUM_SOURCES] = { 0.0,
0.0, 0.0, 0.0, 0.0 };
```

Al atender el evento de teclado de la pulsación de la tecla d 'r' o la tecla 'R', se incrementa o decrementa respectivamente el valor del ángulo de rotación. Esto permite, pasarle a OpenGL el valor en grados, y a OpenAL el valor en radianes empleando la macro DEG2RADque convierte grados a radianes:

```
case 'r':
/*actualizamos */
angulo[fuenteObjetivo] +=
incrementoAngulo;
/*reseteamos si pasa de 360 */
angulo[fuenteObjetivo]
= (angulo[fuenteObjetivo] >= 360.0) ?
0.0 : angulo[fuenteObjetivo];
/*al modificar esta variable, la
funcion display() rotara la caja */
/*aplicamos a openal */
rotarFuenteEnOpenAL
(sourceNames[fuenteObjetivo],
angulo[fuenteObjetivo]);
break;
```

Esto nos permite mantener la consistencia en el ángulo de giro, y se puede probar en la salida por consola de la actualización del vector de dirección, ya que cuando rotamos la fuente de forma que se quede mirando hacia la *derecha* el valor del vector de dirección imprimido es *{1 0 0}*, lo cual nos asegura que la rotación es coherente en ambos entornos.

Una comprobación de este funcionamiento se puede realizar mediante la rotación para la fuente 1 (verde), la cual dispone de un ángulo externo = 30 grados, y se comprueba que al rotar y quedar el oyente fuera del área del cono, no se escucha el sonido.

Además cabe mencionar que la aplicación tiene implementada la actualización del *step de desplazamiento en OpenGL* para que cuando la velocidad de la fuente aumente debido a la pulsación de la tecla correspondiente '*v*', en *OpenAL* también se incremente la velocidad de desplazamiento. El porcentaje de incremento es de 1%, y cuando se mueva la fuente se deberá pasar este *step actualizado* a OpenAL a través de la función "*moverFuenteEnOpenAL()*". No actualizamos cuando NO se haya modificado la *vel* en OpenAL (limites [0,300]).

2.5 TIPOS DE FUENTE: DIRECCIONAL, OMNIDIRECCIONAL, CON ATENUACIÓN, SIN ATENUACIÓN.

Para conseguir un tipo de fuente direccional necesitamos que el ángulo interno sea menor que el ángulo externo, así por ejemplo podemos establecer:

```
alSourcef (sourceNames[1],
AL_CONE_INNER_ANGLE, 5.0f);
alSourcef (sourceNames[1],
AL_CONE_OUTER_ANGLE, 30.0f);
```

Esto genera una duda sobre la ganancia que queremos que exista cuando el oyente este fuera del área de influencia de estos ángulos, por ello, para este caso establecemos que cuando el oyente este fuera del área de influencia, no se escuche nada poniendo una ganancia 0:

```
alSourcef (sourceNames[1],
AL_CONE_OUTER_GAIN, 0.0f);
```

Con esto conseguimos un efecto de fuente "muy direccional" (fuente de color verde en el ejemplo).

También podemos crear una fuente direccional pero más parecida a la realidad, es decir que el ángulo externo abarque *360 grados*, lo cual nos permitirá comprobar que el volumen de la fuente es mas "*fuerte*" cuando está apuntando directamente de frente al oyente, que cuando está "*dándole la espalda*". Este comportamiento es de más aproximado a la realidad ya que se propaga en todas las direcciones.

Por otro lado, para conseguir un tipo de fuente omnidireccional necesitamos que el ángulo interno sea de 360 grados, y evidentemente el exterior también. Una vez conseguido esto, el volumen (intensidad sonora) de la fuente es igual si esta "de cara" o "dándole la espalda" al oyente. En este tipo de fuentes, poner ganancia 0 fuera del área de influencia *no tiene efecto*:

```
  alSourcef (sourceNames[0],
AL_CONE_OUTER_GAIN, 0.0f);

  /* NO tiene efecto cuando
outer_angle = 360*/
```

La mejor forma de comprobar esto, en nuestro programa "*main*" es colocar la fuente frente al oyente y rotarla. (Fuente roja en el ejemplo).

Una idea breve sobre el funcionamiento de los tipos de fuente puede apreciarse en el vídeo "*openal_efectoDoppler.avi*" del directorio *videos* el cual ha sido codificado con h.264 para que ocupe poco espacio en disco.

2.6 EFECTO DOPPLER Y VELOCIDAD DE LA FUENTE.

El factor *doppler* lo podemos obtener mediante la función:

```
d = alGetFloat(AL_DOPPLER_FACTOR);
```

El valor de este factor tiene un rango *de 0.0 a 10.0* y un valor por defecto de *1.0.* Esta es una propiedad de una fuente y está definida de la misma manera que la propiedad de *Factor Doppler global* proporcionado en *OpenAL* y *DirectSound*. Es importante tener en mente que el *factor doppler de una fuente* es un ***multiplicador*** de la propiedad de *Factor Doppler global.*

Un valor *0.0* deshabilita el efecto del *desplazamiento Doppler* para la fuente correspondiente. Un valor de *1.0* proporciona un efecto *Doppler natural* de acuerdo al movimiento de la fuente relativo al oyente. Un valor por encima de *1.0* exagera este efecto.

La forma más coherente de probar el efecto *doppler* es variar la propiedad de velocidad de la fuente. Por ejemplo, si tenemos una velocidad de *33.3 m/s (120 km/h)* y desplazamos la fuente de tal forma que se acerque, luego pase cerca del oyente, finalmente y se aleje, podremos observar cómo el motor de OpenAL aplica el efecto *doppler*, dando como resultado que al pasar cerca del oyente la *frecuencia* de la onda sonora es más rápida que cuando está lejos del oyente.

Para ello se ha realizado una función que permite al usuario variar la velocidad de la fuente, la cual inicialmente se ha establecido que tenga *33.3 m/s (120 km/h)*. Los valores negativos no tienen sentido en esta propiedad sin embargo el motor de OpenAL no tiene esto en cuenta, permitiendo colocar valores negativos y teniendo que controlar nosotros los valores negativos. Por defecto, OpenAL establece la velocidad del sonido como *speedOfSound = 343.3m/s,* por tanto, ese es nuestro limite. Al acercarnos a ese valor, el motor de OpenAL empieza a exagerar el efecto doppler en gran medida. Sin embargo, se puede modificar este valor para "simular" otro entorno que no sea el aire, para ello se dispone de la función:

```
void alSpeedOfSound(  ALfloat value
);
```

En este punto cabe mencionar un inconveniente surgido durante la modificación del parámetro *d* correspondiente al *factor de multiplicación doppler* en ejecución, y es, que debido a condiciones de la máquina virtual, el motor de OpenAL puede dar errores, por lo que se ha optado por dejar desactivada la función que realiza esta modificación, ya que lo más lógico es variar la velocidad de la fuente y probar el efecto en lugar de exagerar el *factor de multiplicación doppler*.

2.7 ATENUACIÓN BASADA EN LA DISTANCIA Y RECORTADO POR DISTANCIA.

En cuanto a la atenuación de la ganancia de la fuente basada en la distancia, podemos decir que es una de las tareas de las que se encarga OpenAL. Para ello, existen tres parámetros clave:

```
AL_REFERENCE_DISTANCE
AL_MAX_DISTANCE
AL_ROLLOFF_FACTOR
```

Manipulando estos parámetros, hemos creado fuentes que difieren de una fuente normal. Por ejemplo, la fuente *azul* es de tipo *omnidireccional* con "*recortado por distancia*". Para esta fuente, dado que la versión 1.1 de OpenAL no ofrece esta característica ("*At this time OpenAL does not support culling at all*"), ha sido necesario crear una función que basada en la distancia entre fuente y oyente "fuerce" la ganancia 0.0 cuando sea mayor que una determinada distancia.

Esto debe hacerse así por ahora porque, cuando establecemos para una fuente la propiedad

```
AL_MAX_DISTANCE = radioCircunferencia
```

No se produce un "*recorte*" (apagado) del sonido en esa distancia si no que se mantiene el último nivel de ganancia independientemente de la distancia. Es decir, *AL_MAX_DISTANCE* es el valor por encima del cual se "*desatiende*" la atenuación de la ganancia de la fuente, dejando de realizar los cálculos necesarios y "*aliviando*" el motor de OpenAL.

Para esta fuente azul, se llama a la función propia "*recortarFuentePorDistancia()*" para "*recortar*" según la distancia actual.

En otro sentido, la fuente *amarilla* es un ejemplo de "*atenuación desatendida*", ya que es de tipo *direccional* y además tiene un valor máximo de distancia por encima de la que ya no se calcula la atenuación.

Finalmente, la fuente *magenta* es un tipo de fuente en la que se pretende comprobar que si establecemos

```
AL_REFERENCE_DISTANCE =
AL_MAX_DISTANCE
```

Igualamos la mínima con la máxima, la fuente se escucha con el mismo nivel de ganancia y sin ninguna atenuación.

Para un mejor entendimiento del modelo matemático que se emplea en OpenAL para realizar el cálculo de la atenuación basada en la distancia, podemos observar el diagrama de la figura 5, en la que se presenta uno de los modelos disponibles, el modelo lineal.

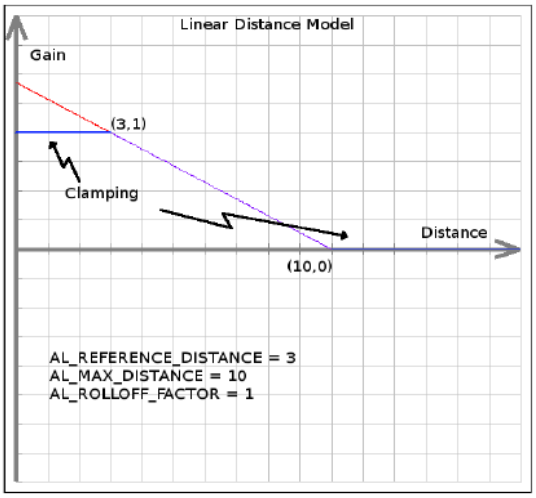

Figura 5. Modelo de atenuación por distancia lineal.

Este diagrama es el más sencillo de explicar y muestra el modelo lineal extendido para garantizar que para distancias entre oyente y fuente que se encuentran por debajo de *AL_REFERENCE_DISTANCE = 3*, la ganancia de la fuente correspondiente se queda "*fijada*". En resumen lo que se observa en el figura 5 es que mientras la distancia entre oyente y fuente se encuentre en el rango [0, 3] el nivel de ganancia percibido por el oyente se mantiene fijo en valor 1 (línea horizontal azul) dando la sensación de intensidad sonora máxima emitida desde la fuente. A partir de esa distancia de 3, la ganancia empieza a reducirse de forma lineal conforme aumenta la distancia entre fuente y oyente, hasta que finalmente al sobrepasar el valor *AL_MAX_DISTANCE = 10*, la ganancia se mantiene en nivel 0.

El último detalle de este modelo, es el factor de atenuación *AL_ROLLOFF_FACTOR = 1*, el cual establece la rapidez con la que se llega desde el nivel de ganancia 1 con distancia 3, al nivel de ganancia 0 con distancia 10. Para este ejemplo en el diagrama se establece que el factor de atenuación es de 1.

De manera análoga, el *modelo inverso*, que actualmente es el utilizado en la aplicación al ser el modelo que por defecto establece OpenAL, se puede encontrar en la guía del programador de OpenAL, donde encontramos que el funcionamiento es similar al lineal con la salvedad de que no existe distancia máxima en el modelo, por lo que actuar sobre este parámetro no tendría efecto y tan solo se "desatiende" la atenuación.

Para una explicación más extensa se puede consultar la guía cuya copia local "*OpenAL_Programmers_Guide.pdf*" se encuentra en el directorio "*docs\openal_v1_1*", y también podemos observar la figura 6.

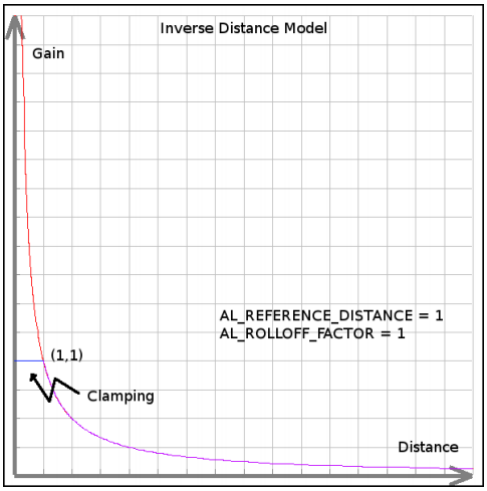

Figura 6. Modelo de atenuación por distancia inverso.

Los datos relevantes de este modelo encontrados en la depuración de la aplicación han sido que la *distancia máxima AL_MAX_DISTANCE* entre fuente y oyente para simular ese modelo que tiende a infinito, por defecto, cuando se crea la fuente se coloca un valor enorme:

```
340282346638528859811704183484516925440
0.000000
```

Eso significa que se mantiene el último valor de ganancia independientemente de lo lejos que se encuentre la fuente a partir de cierta distancia.

Para encontrar este tipo de valores "*por defecto*" se puede emplear rápidamente la función:

```
void alGetSourcef(
ALuint source,
ALenum param,
ALfloat *value
);
```

Así mismo, se ha comprobado que la propiedad de distancia de referencia

AL_REFERENCE_DISTANCE establece una distancia mínima a considerar entre fuente y oyente. Es decir, de 0 a *AL_REFERENCE_DISTANCE* se aplica el "*clamping*" o fijación de la ganancia como se observa en la línea azul de la figura 6. El valor por defecto que establece OpenAL cuando se crea es *AL_REFERENCE_DISTANCE = 1.0*

El otro parámetro participante en este modelo, es el factor de atenuación el cual tiene un valor por defecto en la creación de la fuente de *AL_ROLLOFF_FACTOR = 1.0*

2.8 EFECTOS CON OPENAL SOFT.

El primer paso realizado en este apartado, ha sido la comprobación de la presencia de los efectos en la versión instalada tanto en laboratorio de prácticas de *IMD* como la presente en la máquina virtual.

Para ello nos hemos servido de un fichero existente en la red y que se puede descargar del **enlace [6].** Para compilar ese ejemplo lanzamos la orden:

```
$ gcc info-openal.c -lalut -lopenal -
o info-openal
```

Con la ejecución hemos comprobado que los efectos presentes son:

```
Supported filters: Low-pass
Supported effects: EAX Reverb,
Reverb, Echo, Ring Modulator
```

A pesar de lo indicado, hemos comprobado manualmente con instrucciones condicionales que sólo existe soporte para *Reverb* y para *Echo*, y hemos procedido a realizar un ejemplo de partida en el que se ponen de manifiesto los pasos necesarios para emplear los efectos de *OpenAL soft*, y los cuales se encuentran recogidos en el fichero "*openalEfectos.c*" dentro del directorio "*codigo\partes\efectos*".

Este ejemplo, ha sido elaborado siguiendo los tutoriales presentes en el documento "*EfectosExtensionsGuide.pdf*" cuya copia local se encuentra en el directorio "*docs\openal_v1_1*".

En resumen, tenemos los siguientes pasos:

- Crear un "Device"
- Crear un "Context" con Efectos "auxiliares"
- Crear fuentes
- Crear Slots para los efectos
- Adjuntar efectos a los slots
- Configurar envíos. (AL_DIRECT_FILTER / AL_AUXILIARY_SEND_FILTER = dry / wet)

Un ejemplo de ejecución se puede ver en el video "*openalEfectos.avi*", del directorio "*codigo\partes\efectos*".

Para profundizar más en el entendimiento del uso de efectos en las fuentes, nos serviremos del diagrama de la figura 7, donde podemos apreciar que cuando creamos un contexto en OpenAL sin efectos podemos interactuar básicamente sobre un número reducido de parámetros como volumen y pitch, sin embargo cuando creamos un contexto con efectos existen otros elementos presentes en el contexto, es decir, se añaden los elementos de tipo "*slots*", "*sends*", y los efectos adjuntos a dichos slots.

La figura 7 nos da una muestra de ello, dejando ver, que los reguladores gráficos típicos de los programas de mezclas de audio por debajo emplean elementos funciones similares a las que OpenAL nos ofrece para el "*envío directo*" y el "*envió con efecto*".

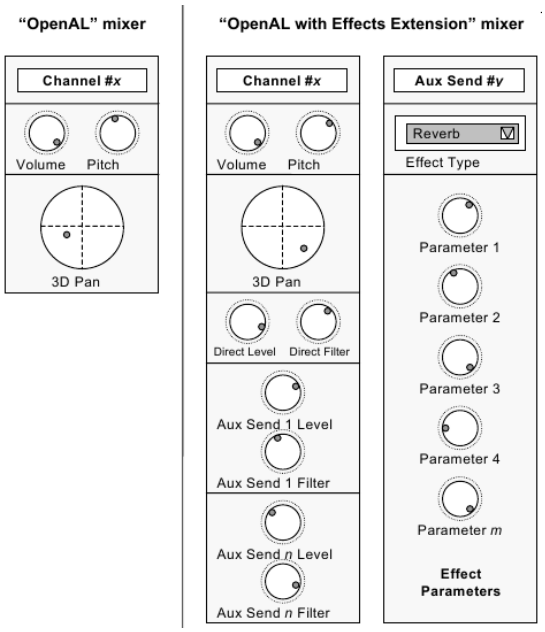

Figura 7. Contexto de OpenAL con efectos.

A continuación mostraremos ejemplos de instrucciones para generar efectos:

Crear Slots:

```
alGenAuxiliaryEffectSlots(1,
&uiEffectSlot[uiLoop]);
```

Establecer el tipo de efecto:

```
alEffecti(uiEffect[0], AL_EFFECT_TYPE,
AL_EFFECT_REVERB);
```

Establecer un parámetro del efecto:

```
alEffectf(uiEffect[0],
AL_REVERB_DECAY_TIME,
reverbDecayTime);
```

Adjuntar el efecto a un slot:

```
alAuxiliaryEffectSloti(uiEffectSlot[0]
,AL_EFFECTSLOT_EFFECT, uiEffect[0]);
```

Configurar envíos auxiliares:

```
alSource3i(sourceNameReverb,
AL_AUXILIARY_SEND_FILTER,
      uiEffectSlot[0],      /*id del
slot de efecto auxiliar Reverb*/
      0,                    /*numero
de envio auxiliar*/
AL_FILTER_NULL);            /* id de
filtro opcional (NULL puede dar
problemas, mejor AL_FILTER_NULL)*/
```

Adjuntar filtros (efectos) a la fuente:

```
  alSourcei(sourceNameEcho,
AL_DIRECT_FILTER, AL_FILTER_NULL);
*DRY*/

alSource3i(sourceNameEcho,AL_AUXILIARY
_SEND_FILTER, uiEffectSlot[1], 0,
AL_FILTER_NULL);
/*WET*/
```

Los pasos con más detalle se encuentran secuenciados en el fichero "*funcionesEfectos.c*", donde además se dispone de las funciones para modificar parámetros de los efectos creados "*reverb*" y "*echo*", de la que cabe mencionar que la función que crea los *strings* con los valores actuales de los parámetros de estos dos efectos, *cargarTextoValoresEfectos()*, ha tenido que sacarse fuera del *glutmainloop()* de OpenGL debido a que ralentizaba la aplicación.

Las funciones que modifican los parámetros de los efectos, se han implementado siguiendo el mismo criterio que las que modifican parámetros en el modo *DOPPLER*, como posición, pitch, etc. Es decir, disponemos de una función para establecer el "*parametroEfxObjetivo*" que identifica el parámetro a ser manipulado ya que los hemos numerado de 1 a 10. De esta forma, cuando se pulsa la tecla "*m*" (*más*) se lanza la función "*incrementarParametroEfxObjetivo()*" la cual en función de cuál sea el número del parámetro objetivo a manipular actualmente lanzara:

```
  case 1:
    masreverbDecayTime();
    break;
  case 2:
    masreverbRoomRolloffFactor();
    break;
  case 3:
    masreverbReflectionsDelay();
    break;
  case 4:
    masreverbReflectionsGain();
    break;
..
```

El mismo mecanismo se sigue para realizar decrementos mediante la tecla '*n*'.

Los efectos que se encuentran actualmente presentes en la aplicación son Reverb y Echo de los cuales pasamos a mencionar datos relevantes:

Reverb:

- El algoritmo tiene en cuenta la orientación del "listener" y trabaja con Vectores de dirección.
- Revotes de sonido en el ambiente
- El tiempo en el que llega el rebote es corto (simulación de pasillo, etc)
- Algunos Parámetros:
- RoomRolloffFactor – afecta solo al sonido de una fuente (atenuación del sonido reflejado)

Echo:

- Más sencillo de calcular (son instancias "retrasadas" de la señal de entrada)
- Algunos Parámetros:
- Delay: retraso en segundos,
- LRDelay: (tiempo entre rebote izq y der)
- Damping: controla la amortiguación (decaimiento) de las frecuencias altas aplicadas a cada echo.
- Feedback: retroalimentación de la señal de salida (efecto cascada)

2.8 GUARDAR Y RECUPERAR CONTEXTO AL CAMBIAR DE ESTADO

Para mantener un correcto funcionamiento de los contextos creados sobre el mismo dispositivo en OpenAL, ha sido necesario crear una función para guardar el contexto del escenario del modo *DOPPLER*, es decir, es necesario almacenar la posición, pitch, gain, y el resto de parámetros de cada una de las fuentes presentes, así como del oyente de este contexto. Esta funcionalidad se encuentra en la función "*void guardarEstadoDoppler(ALuint * sourceNames)*" del fichero "*funciones.c*".

El funcionamiento para salvar el contexto emplea estructuras de tipo array de floats para salvar de forma iterativa los valores de cada fuente, así como un puntero al contexto y puntero al dispositivo.

Como se observa la función *guardarEstadoDoppler(..)* recibe como parámetro, un puntero al vector de identificadores de fuentes o nombres de fuentes. Con ello se puede recorrer las posiciones e ir haciendo un "*alGetSourcef(…)*"; para almacenar el valor en el correspondiente float.

En el mismo sentido, la función "*void recuperarEstadoDoppler(ALuint * sourceNames)*" recorre los *arrays* y "*reinserta*" en las fuentes los valores guardados. Esto se realiza de esta forma, ya que al cambiar de contexto, OpenAL dispone de otro oyente y otras fuentes cuyos identificadores pueden dar conflicto con los que ya se encuentran actualmente al estar en el mismo *device*.

2.9 BÚSQUEDA DE FICHEROS DE AUDIO MULTICANAL 5.1

El primer paso para realizar la lectura ha sido conseguir ficheros multicanal. Para ello, nos hemos servido de los ficheros de audio 5.1 en formato de datos en crudo *.raw* disponibles en el **enlace [7]** junto con el ejemplo que realiza el tratamiento de dichos ficheros que se encuentra en el **enlace [8].**

Del mismo modo que en el resto de enlaces, se dispone de las copias locales, en este caso en el directorio
"*codigo\partes\multicanal\lectura\openal13*"

El segundo paso, ha sido comprobar con herramientas software, que los ficheros verdaderamente contiene 5.1 canales y el formato correcto en cuanto muestras por segundo (Hz), número de bits por muestra, y organización de los bytes. Este paso es importante, ya que una incorrecta interpretación del formato nos puede llevar a leer de forma incorrecta el audio obteniendo como resultado tan solo ruido, audio ralentizado, etc.

Nos hemos servido de "*Audacity*" el cual es un programa de código abierto para manipulación de audio mediante la visualización de la forma de onda. Se puede descargar del **enlace [9]** para Windows, o se puede lanzar la orden en la consola de Linux:

```
$ sudo apt-get install audacity
```

Entrando ya en detalle, lo que se ha realizado es la importación de datos en crudo una vez que hemos descargado los ficheros de audio 5.1. La manera de importar los ficheros *.raw* a *Audacity* y observar las formas de onda, precisa acceder al menú "*Archivo*" y saber el número de muestras por segundo en el que se han guardado los ficheros.

Paso 1: Conocer el formato, número de muestras por segundo, número de bits por muestra, organización de los bytes y número de canales. Esta información se puede encontrar en el fichero que realiza el tratamiento de estos ficheros *.raw*. En la figura 8, podemos observar una captura del código referente al número de muestras por segundo, canales y bits por muestra con la que han sido grabados estos ficheros *.raw* y con los que se realiza la carga a memoria.

```
checkForErrors();
alGenSources(5, &source[0]);
checkForErrors();

LoadBuffer("stereo_section_16_44.raw",buffer[0], AL_FORMAT_STEREO16, 22050);
LoadBuffer("quad_section_16_44.raw",buffer[1], AL_FORMAT_STEREO16, 22050);
LoadBuffer("quad_raw_16_44.raw",buffer[2], AL_FORMAT_QUAD16, 44100);
LoadBuffer("Sdotl_16_11.raw",buffer[3], AL_FORMAT_STEREO16, 11025);
LoadBuffer("51_raw_16_44.raw",buffer[4], AL_FORMAT_51CHN16, 44100);

alSourcei(source[0], AL_BUFFER, buffer[0]);
alSourcei(source[1], AL_BUFFER, buffer[1]);
alSourcei(source[2], AL_BUFFER, buffer[2]);
```

Figura 8. Formato de los ficheros de audio 5.1 en crudo .raw

Paso 2: Importar el fichero a *Audacity* para comprobar visualmente la aparición de los distintos canales. Accedemos a "*Archivo->Importar->Datos en bruto*". A continuación en la ventana que aparece debemos establecer el formato con el que se van "leer" los datos del fichero. Para nuestro caso, empezaremos con el fichero 5.1 denominado "*51_raw_16_44.raw*" ya que es el que más nos interesa, el resto de ficheros de 4 y dos canales se importan de manera similar.

La figura 9 nos indica los parámetros con los que se debe importar a Audacity este fichero, en los que podemos observar que se deben corresponder con lo establecido en la figuro 8, es decir, 16 bits por muestra, Little-endian, 6 canales (5 + 1) y 44100 muestras/s.

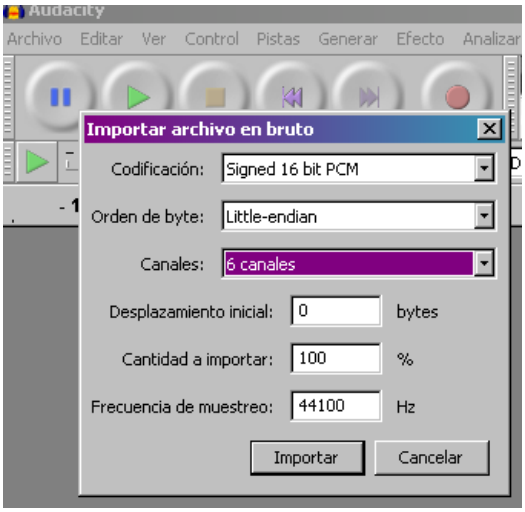

Figura 9. Importación de *51_raw_16_44.raw* a Audacity como datos en bruto.

El resultado de la importación se puede observar en la figura 10.

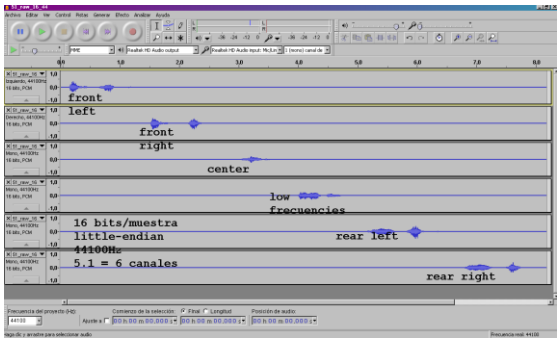

Figura 10. Formas de onda de los 6 canales del fichero *51_raw_16_44.raw*

Para facilitar la comprobación de primera mano de la importación del resto de ficheros *.raw* se pueden encontrar imágenes de la importación con comentarios de los respectivos formatos para su importación en el directorio "*codigo\partes\multicanal\ficheros\link1*". A modo de refuerzo para la posterior comprobación con diferentes ficheros, se pueden encontrar ficheros de audio 5.1 en formato *.wav* en el directorio "*codigo\partes\multicanal\ficheros\link2*".

2.10 LECTURA DE UN FICHERO 5.1 Y SEPARACIÓN DE CADA CANAL.

En el fichero "*funcionesLectura51.c*" se pueden encontrar las distintas partes en las que se ha adaptado el fichero del **enlace [10]**, del que podemos destacar que lo primero a realizar es averiguar si existe soporte para formatos multicanal:

```
if(alIsExtensionPresent("AL_EXT_MCFORM
ATS") == AL_TRUE)
```

El siguiente paso ha sido generar un buffer de tipo 5.1 donde alojar los datos, y 6 buffers de tipo mono para alojar el audio de cada canal separado.

```
alGenBuffers(1, &buffer51);
checkForErrors51();
alGenSources(1, &source51);
checkForErrors51();
```

Una vez tenemos generados los buffers, lo siguiente es *adjuntar los datos* al buffer. Esto se realiza en la función *LoadBuffer()*, donde podemos encontrar que para realizar la reserva de memoria en primer lugar se calcula el número total de bytes a reservar, luego se reserva esa cantidad de memoria con la instrucción:

```
memBuffer = malloc(readMem);
```

Despues de la reserva, ya solo queda leer desde disco:

```
fread(memBuffer, readMem, 1, file);
```

En la parte de la separación de los canales, nos apoyamos en el empleo de 6 buffers y 6 fuentes.

```
alGenBuffers(6, bufferCanal);
/*6 buffers separados*/
    checkForErrors51();
    alGenSources(6, sourceCanal);
/*6 fuentes separadas*/
    checkForErrors51();
```

Disponiendo de los buffers generados, se procede a reservar la memoria para alojar los datos de audio de un solo canal. Para ello dividimos entre 6 el total de bytes del fichero.

```
    numTotalBytesDeUnCanal =
(ALuint)(readMem/6);
    muestrasDeUnCanal =
malloc(numTotalBytesDeUnCanal * (2 *
(sizeof(char)) )  );
```

Finalmente, el bucle que realiza la separación de cada canal emplea un puntero para desplazarse por los bytes, y la función *memcpy()* junto con la función:

```
alBufferData(bufferCanal[c],
AL_FORMAT_MONO16, muestrasDeUnCanal,
numBytesLeidos, sampleRate*2);
```

Una vez separados los canales, ya podemos realizar el cambio en las 5 fuentes que tenemos en el modo *DOPPLER*. Esto se realiza con la función "*void intercambiarBuffers(ALuint *sourceNames)*" o con la función "*void intercambiarFuentes(ALuint *sourceNames)*".

Para lanzar esta "copia" se dispone de la opción "*Leer 5.1*" en el menú desplegable que se definió en apartados anteriores en este documento. No existe problema de contexto ya que lo que se hace es llevar los buffers al contexto que se encuentra ya creado en el modo *DOPPLER*.

En el diagrama de la figura 11 podemos observar la forma que se empleado para extraer e ir agrupando los bytes de los distintos canales teniendo en cuenta que el puntero debe saltar 6 canales para recolocarse en los siguientes 8 bits de la misma muestra.

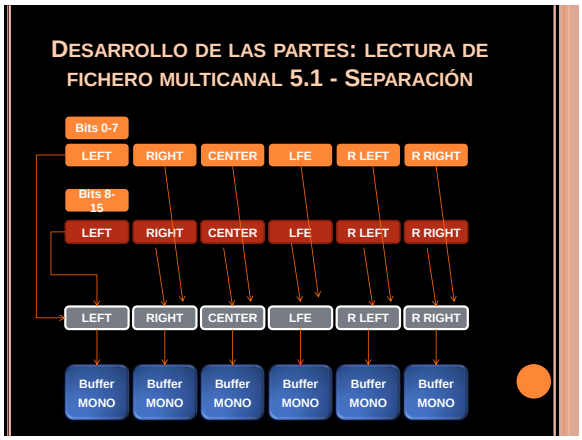

Figura 11. Diagrama de separación de canales.

3 CONCLUSIONES

A continuación pasaremos a mencionar algunos de los inconvenientes y soluciones encontradas para mejorar el rendimiento de la aplicación:

Las instrucciones que se lanzan no siempre pueden ser realizadas, por ello hay que comprobar constantemente si OpenAL ha podido realizar la acción.

Las modificaciones de parámetros de efectos pueden generar errores (ejemplo Reverb acepta 2 y rechaza el resto).

La integración de los motores de OpenAL y OpenGL en ocasiones se sobrecarga. (ejemplo dibujar el texto de los parámetros de los efectos)

Limitaciones debidas a la máquina virtual. Realizar modificaciones en las propiedades de las fuentes mediante pocas veces.

En lugar de realizar un *alGetSourcef(...)* cada vez que se quiere consultar el último valor, guardar el último valor en un float.

Limitar la orden *glutSwapBuffers() de* OpenGL

Emplear glPushAtrib*()* en lugar de *glPushMatrix()*

Depurar errores de OpenAL con *alutGetErrorString()* y otras funciones.

Finalmente, sería interesante que en futuras ampliaciones se empleara la biblioteca de GLUI para construir sliders con los que permitir al usuario enviar nuevos valores a los parámetros de los efectos, así como para añadir o mejor el movimiento de las fuentes mediante *trackball*. Tambíen sería interesante incorporar un apartado que permita capturar el audio del micrófono para insertarlos en una fuente con efectos y comprobar de primera mano cómo el *reverb* o el *echo* afecta a una voz humana.

Realizar la integración de los distintos apartados ha sido la parte que más nos ha obligado a volver hacia atrás y readaptar versiones de código para que la aplicación no se quede en un estado no controlado.

Al principio de este documento se hable de "*alloopback*" para motivar a los futuros lectores con intención de que se retomen las ampliaciones para permitir escritura de un fichero 5.1

Referencias

[1] https://www.virtualbox.org/wiki/Downloads

[2] http://cdimage.ubuntu.com/kubuntu/releases/12.04/release/

[3] http://www.codeproject.com/Articles/182109/Settingup-an-OpenGL-development-environment-in-Ub

[4] https://github.com/irungentoo/openal-soft-tox/tree/master/examples

[5] http://kcat.strangesoft.net/openal.html

[6] https://github.com/garinh/openal-soft/blob/master/examples/openal-info.c

[7] https://github.com/garinh/openal-soft/tree/899eb0747644923239d48dd706fe2d3cde02734a/build

[8] https://github.com/garinh/openal-soft/tree/master/examples

[9] http://sourceforge.net/projects/audacity/?lang=es

[10] https://github.com/garinh/openal-soft/blob/master/examples/openal-mctest.c