UNIVERSITAT POLITÈCNICA DE VALÈNCIA

OpenAL: usando *libsndfile para reproducción en* streaming *de ficheros de audio*

Apellidos, nombre	Agustí i Melchor, Manuel (magusti@disca.upv.es)
Departamento	Departamento de Informática de Sistemas y Computadores (DISCA)
Centro	Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València

1 Resumen de las ideas clave

OpenAL [1] es un motor de audio 3D, esto es, utiliza audio cargado en memoria para generar sonido espacial (también denominado *surround o audio envolvente)*. Esto permite *renderizar* el sonido que escucha un oyente inmerso en una escena sonora tridimensional, de modo que, él mismo, se "vea" envuelto entre las fuentes de sonido dispuestas a su alrededor.

Las opciones de importación de ficheros en OpenAL se limitan a la funcionalidad que ofrece la capa denominada "The OpenAL Utility Toolkit" [2] (abreviado como ALUT) en la organización interna de OpenAL. El cometido de ALUT[1] es implementar un mínimo interfaz de acceso al contenido de ficheros de audio, limitado a la lectura de ficheros WAVE monofónicos o estéreo. En ambos casos, los ficheros se leen y se cargan completos en la memoria principal antes de ser reproducidos. Cuando la cantidad de datos a cargar es importante, hay que valorar cuándo y cómo se puede hacer. Por ello caben básicamente dos estrategias de reproducción de audio: a partir de la precarga ("static playback" en la terminología de OpenAL) del audio o de la reproducción en continuo (o en streaming). En este trabajo nos vamos a centrar en la reproducción en *streaming* que, puesto que no la ofrece directamente ALUT, será necesario implementar; además del acceso al fichero de audio, lo cual llevaremos a cabo con *libsndfile*.

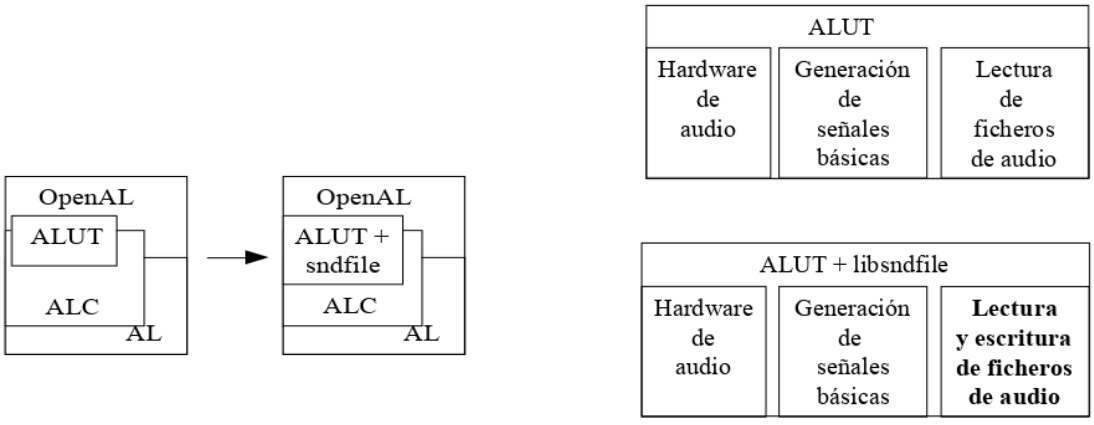

Figura 1: Esquema de bloques del estándar OpenAL y su extensión propuesta: (a) organización en niveles ALUT, ALC y AL y (b) modificación propuesta en el nivel ALUT al añadir libsndfile.

La Fig. 1 resume gráficamente el problema y la solución que vamos a abordar aquí. En la Fig. 1a podemos ver la organización en niveles original de OpenAL y como queremos modificar el nivel ALUT con la incorporación de *libsndfile*. Esta librería proporcionará a OpenAL nuevas operaciones tanto de lectura como de escritura de ficheros (Fig. 1b), de ellas nos vamos a centrar en las que permiten la lectura por "trozos" de los ficheros de audio.

[1] *The OpenAL Utility Toolkit* (ALUT) <http://distro.ibiblio.org/rootlinux/rootlinux-ports/more/freealut/freealut-1.1.0/doc/alut.html>.

2 Objetivos

El presente documento está encaminado a ofrecer una perspectiva inicial de cómo ampliar la funcionalidad original de ALUT, para que sea posible implementar la estrategia de reproducción en *streaming* y tener una referencia práctica de cómo llevarla a cabo. Como que hay factores que dependen de la aplicación final a desarrollar, vamos a poner el enfoque en mostrar la funcionalidad, más que en conseguir el código más óptimo para usarlo en nuestras aplicaciones con OpenAL.

A partir del estudio del ejemplo que se aborda en este documento, el lector será capaz de:

- Incorporar ficheros de varios formatos a un desarrollo sobre OpenAL.
- Identificar una posible estrategia para el uso de ficheros en modo *streaming* o de reproducción en modo continuo, esto es, la carga progresiva en memoria de ficheros de audio.
- Instalar y compilar una aplicación que hace uso de la librería *libsndfile* sobre OpenAL.

3 Introducción

Para entender el contenido del ejemplo que desarrollaremos como solución, vamos a describir el contexto en que nos movemos: el funcionamiento interno de OpenAL, el formato del fichero WAV y las operaciones que proporciona *libsndfile* para leer estos ficheros.

3.1 Arquitectura de OpenAL

El esquema básico de trabajo que implementa OpenAL [1] se basa en el uso de tres objetos ([3], [4] y [10]): *buffers*, fuentes (*sources*) y el mezclador. La Fig. 2 resume los caminos de la precarga y el *streaming*, de manera gráfica, estableciendo la similitud y las diferencias entres estas dos estrategias:

- La precarga se realiza (paso 1 de la Fig. 2) rellenado un *buffer* y asignándolo de manera estática (fija) a una fuente. Los *buffers* son áreas de memoria que contienen sonidos sin compresión (obtenidos desde ficheros o generados de manera sintética). Las fuentes caracterizan los parámetros espaciales que se asocian a cada sonido y los modifican (paso 2 de la Fig. 2) para que reflejen los cambios que definen su situación, orientación y velocidad, etc. Finalmente, todos los sonidos son *renderizados* en la situación en la que define la posición del oyente a través del mezclador (paso 3 de la Fig. 2), para proporcionarle al oyente todos los sonidos adaptados con los valores globales de la escena como la velocidad de transmisión del sonido, el efecto *Doppler*, posición y velocidad del oyente, etc.

- OpenAL no proporciona explícitamente la funcionalidad de reproducción en continuo o *streaming* sobre un objeto [3], pero sí un mecanismo de asociación de un conjunto de *buffers* a una fuente. Esto permite que la aplicación pueda especificar cuántos, de qué tamaño y si se reutilizan o se descartan tras su uso. Así

se consigue maximizar la eficiencia de uso del recurso. Básicamente, consiste en cargar dinámicamente el audio: esto es, tener asociados varios trozos de sonido a una fuente, de manera que la reproducción del sonido dependa de que exista una pila de *buffers* (paso A de la fig. 2), que son desapilados (B y E) conforme se utilizan (C) y que van siendo rellenados (D), al tiempo que terminan de ser utilizados en la reproducción del audio.

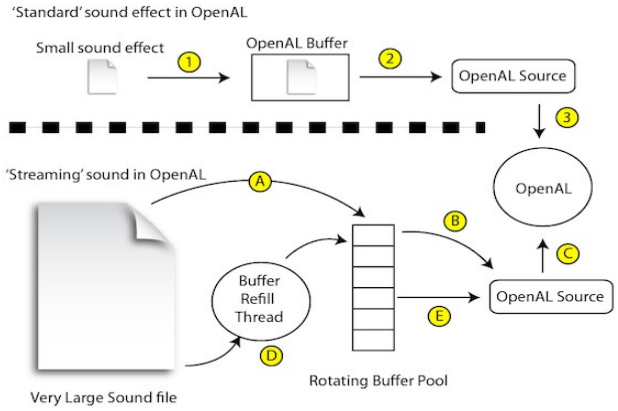

Figura 2: Esquema gráfico de las estrategias de precarga vs streaming en OpenAL. Imagen de [9].

3.2 Contenido de un fichero con formato WAVE

Waveform Audio File Format (WAVE o también abreviado como WAV)[2] es [5] una especificación de fichero de audio en forma de ondas, que nace de una iniciativa de IBM y Microsoft para especificar los archivos de naturaleza multimedia: los *Resource Interchange File Format* o RIFF[3]. Este formato describe cómo almacenar audio digital, sin compresión, permitiendo especificar el número de canales, la frecuencia de muestreo y el tamaño de muestra entre otros valores. La difusión de la publicación del formato, junto con su sencillez han hecho de él un formato de audio que ha sido utilizando en, posiblemente, todas las plataformas de computadores con más o menos precisión.

La Fig. 3a muestra los bloques que se pueden observar en la especificación de este formato. En ellos podemos encontrar una cabecera, seguida de una secuencia de *chunks* (trozos) de datos. De manera que, tras la cabecera, hay dos *chunks*: "fmt" que describe el formato de los datos y "data" que contiene los datos en sí. En la sección de datos, cada muestra (*sample*) de audio se ha codificado de forma independiente por canales, a partir de una determinada frecuencia de muestreo y con un tamaño de bits. Estos datos de audio se agrupan en uno o más *frames* de audio. Un *frame* es un conjunto de muestras para todos los canales utilizados; de manera que, todas las muestras de un *frame*, han de

[2] Encontrada en *Wave File Specfications*, en las páginas del Prof. Peter Kabal, MMSP Lab, ECE, McGill University. <http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html>.

[3] Microsoft sigue basando en esta especificación sus nuevos formatos, como puede verse en la documentación del API de *XAudio2* <https://docs.microsoft.com/es-es/windows/win32/xaudio2/resource-interchange-file-format—riff>.

sonar simultáneamente en los diferentes canales, así que un *frame* constituye un bloque de audio dentro del contenido del fichero.

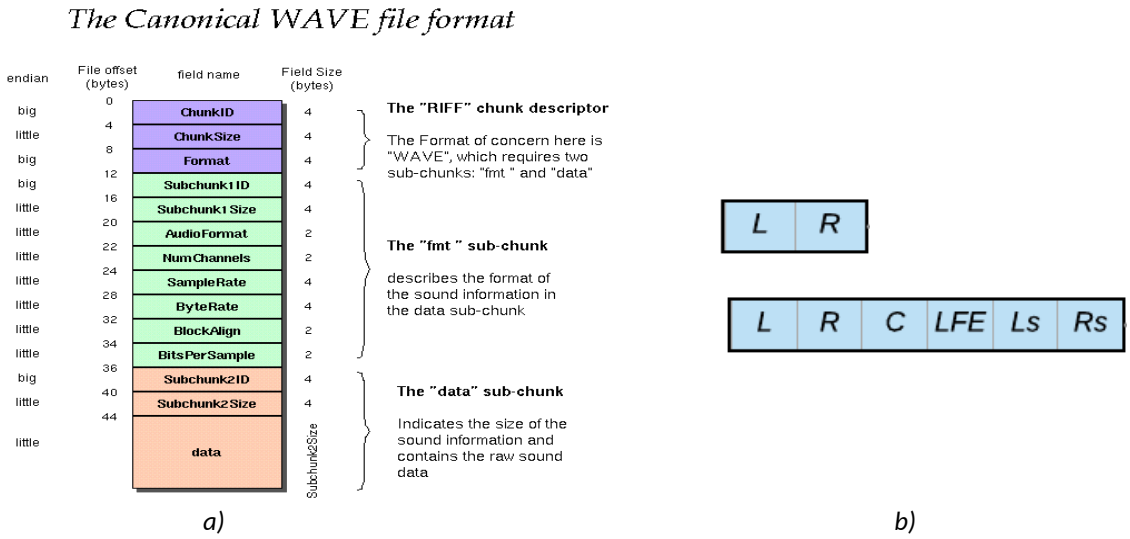

Figura 3: Contenido del fichero WAVE: (a) Estructura de WAVE [6] y (b) concepto de frame.

La Fig. 3b[4] muestra cómo el tamaño del *frame* está en función del número de canales. En el caso de un *frame* para un sonido estéreo (es decir con dos canales) contiene la muestra para el canal izquierdo (L), seguida de la muestra para el canal derecho (R). Y en el caso de un *frame* para un sonido 5.1 (con seis canales) siempre codifica las muestras en la secuencia: canal izquierdo (L), canal derecho (R), canal central (C), subwofer (LFE), canal izquierdo trasero (Ls) y derecho trasero (Rs).

3.3 Introducción a libsndfile

La biblioteca de funciones que vamos a utilizar es l*ibsndfile* [7] y está implementada en lenguaje C, diseñada para lectura y escritura de formatos de ficheros de audio y publicada en código fuente, bajo una licencia GPL (*GNU Lesser General Public License*). De entre los más de veinticinco formatos soportados [8] por esta biblioteca, hay funciones para la lectura y escritura de formatos de audio estructurado, ficheros sin formato (RAW PCM) o formatos de audio en forma de ondas, entre los que destacaremos: *WAV*, *AIFF* y *FLAC*.

A nivel de desarrollo, el trabajo con *libsndfile* para la lectura de ficheros de audio, se basa en el uso de:

- El tipo SNDFILE que sirve de referencia al fichero en disco.
- El tipo SF_INFO, que es la estructura en la que se recogen los valores del chunk “fmt” (Fig. 3a) del fichero.
- Las funciones de lectura (con prefijo *sf_readf*) para acceder a los *frames* de audio.

[4] Extraída de Bitmovin (2021). “Separating and Combining Audio Streams”. Disponible en <https://bitmovin.com/docs/encoding/tutorials/separating-and-combining-audio-streams>.

4 Desarrollo

La Fig[5]. 4a muestra de forma gráfica cómo la librería *Audio Library Utility Toolkit* (ALUT) [2] proporciona a OpenAL un mecanismo básico que es capaz de leer archivos WAVE de disco (con la llamada al sistema *read*) y extraer de ese contenido el audio en PCM para asociarlo (con *alutCreateBufferFromFile*) a un *buffer* de OpenAL. Ahora buscamos poder leer ese fichero poco a poco, para lo que hemos de reescribir una secuencia de operaciones de libsndfile compatible con el funcionamiento de OpenAL.

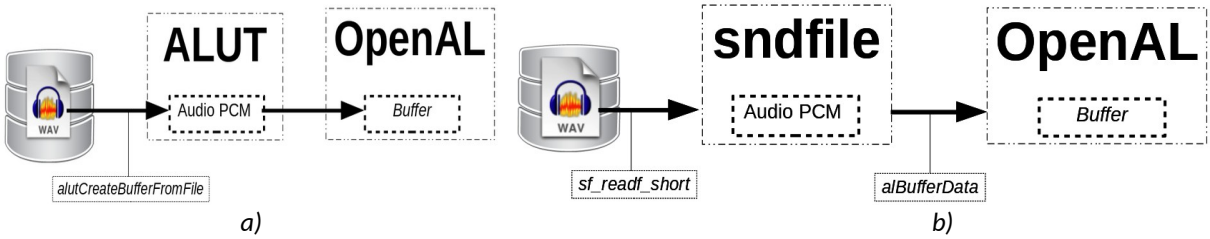

Figura 4: Esquema básico de gestión de formatos de ficheros en OpenAL: (a) con ALUT y (b) la propuesta de este trabajo utilizando libsndfile.

La Fig. 4b resume, de forma gráfica, los cambios que proponemos: mantener el flujo de trabajo que proporciona ALUT, buscando obtener con *libsndfile* la secuencia de operaciones que haga transparente el uso de ficheros en formato WAVE, obteniendo el audio, a trozos, en PCM para asignárselo a un *buffer* de OpenAL.

4.1 Código de ejemplo propuesto

Para conseguir los objetivos propuestos y con la idea básica de la Fig. 4, la reproducción de un fichero WAVE se puede hacer como muestra el código que se encuentra repartido entre los tres listados que se muestran en este apartado. Las líneas destacadas en negrita son las que sugerimos que el lector centre su atención, puesto que son las que implementan la reproducción en *streaming* a partir de la lectura del fichero con *libsndfile*. Este código es el resultado de pruebas propias, así como de la consulta de ejemplos en [7] y [11].

El Listado 1 contiene las inicializaciones del hardware de audio (línea 30), los mensajes de las versiones de las diferentes librerías (y sus capas) entre las líneas 33 a la 35, la creación de la pila de *buffers* (línea 37) y la de la fuente de audio (línea 40), así como la apertura del fichero WAVE con *libsndfile* (línea 43) que obtiene los datos de la cabecera del archivo.

El Listado 2, deduce cuánto audio (en *frames*, en bytes y en tiempo de reproducción) se encuentra en el archivo (líneas 57 a la 78), los muestra y selecciona el tipo de audio para configurar los objetos de OpenAL. También calcula la cantidad de datos que componen cada *frame* (línea 79 a la 90) y ya podremos ir cargando el contenido del fichero en bloques (*frames*) de este valor (líneas 91 a la 102).

[5] Para la realización de la Fig. 4 se han utilizado iconos de https://es.wikipedia.org/wiki/Waveform_Audio_Format#/media/Archivo:AudacityWAV.png y de https://wiki.documentfoundation.org/Design/Whiteboards/LibreOffice_Initial_Icons.

```
1.  #include <stdio.h>
2.  #include <stdlib.h>     // EXIT_SUCCESS
3.  #include <math.h>
4.  #include <AL/alut.h>
5.  #include <sndfile.h>
6.  #include <string.h>     // memset
7.  #include <sndfile.h>    // libsndfile
8.
9.  #define   BUFFER_LEN      2*1024  // longitud del buffer de frames
10. #define   MAX_CHANNELS    2       // Mono (1) y estéreo (2)
11. short     *data;                  // buffer para las muestras de sonido
12. SNDFILE   *infile;                // descriptor de fichero
13. SF_INFO   sfinfo ;          // datos de la cabecera del fichero SNDFILE
14. char      infilename[128] = "escala.wav" ;
15. #define   NUM_BUFFERS 3
16. ALuint    buffers[NUM_BUFFERS];
17.
18. int main(int argc, char *argv[]){
19.     ALenum error, formatoAudio;           //1 o 2 canales
20.     ALint buffersLibres,queued, estado;
21.     ALuint alsource, bufferID;
22.     sf_count_t samples = 0, mostresLLegides = 0, numTotalMostres = 0;
23.     size_t frame_size;
24.     int bitsMuestra, frecuenciaMuestreo, i=0, val=0;
25.     ALsizei mostresEnBytes;
26.     char barraActivitat[4] = {'|', '/', '-', '\\'};
27.
28.     memset (&sfinfo, 0, sizeof (sfinfo));
29.     if( argc > 1) { strcpy(infilename, argv[1]); }
30.     if (!alutInit(&argc, argv)){
31.         printf("Fallo al iniciar OpenAL.\n"); return( -1 );
32.     }
33.     printf ("La versio de OpenAL instalada es %s \n",
34.              alGetString(AL_VERSION), alGetString(AL_EXTENSIONS));
35.     printf ("La versio de ALUT instalada es %d.%d \n",
36.              alutGetMajorVersion(), alutGetMinorVersion());
37.     printf ("La versio de libsndfile instalada es %s \n",
38.              sf_version_string());
39.
40.     alGenBuffers(NUM_BUFFERS, buffers);
41.     if((error = alGetError()) != AL_NO_ERROR)
42.         printf("Error generando buffers\n");
43.     alGenSources(1, &alsource);
44.     if((error = alGetError()) != AL_NO_ERROR)
45.         printf("Error generando la fuente: %s\n",
46.                  alutGetErrorString(error));
47.     if (!(infile = sf_open(infilename, SFM_READ, &sfinfo)) ) {
48.         printf("Not able to open input file %s.\n", infilename) ;
49.         puts(sf_strerror( NULL )); return( -2 );
50.     }
51.     if (sfinfo.channels > MAX_CHANNELS)    {
52.         printf ("%s té %d canals? No processem si més de %d canals\n",
53.                  infilename, sfinfo.channels, MAX_CHANNELS) ;
54.         sf_close (infile) ;
55.         return ( -1 );
56.     }
...
```

Listado 1: Ejemplo de lectura de fichero WAVE con libsndfile en OpenAL (parte 1).

```
...
57.     printf("%s: canals %d, freqMostreig %d, format %0xd [%s, %s], total
58.  (%ld mostres %ld bytes) i temps (%04f segons o %02ld : %02ld
59.  minuts:segons; frames %ld)\n",
60.             argv[1], sfinfo.channels, sfinfo.samplerate, sfinfo.format,
61.             ((sfinfo.format & SF_FORMAT_WAV) > 0? "WAVE": "no WAVE"),
62.             ((sfinfo.format & SF_FORMAT_PCM_16) > 0? "16bits" :
63.             ((sfinfo.format & SF_FORMAT_PCM_S8) > 0? "8bits" : "")),
64.             numTotalMostres, //sfinfo.frames,
65.             (sfinfo.frames * sfinfo.channels *
66.                 ((sfinfo.format & SF_FORMAT_PCM_16) > 0? 2 : 1)),
67.             (float)sfinfo.frames / (float)sfinfo.samplerate,
68.             (sfinfo.frames  / sfinfo.samplerate) / 60,
69.             (sfinfo.frames  / sfinfo.samplerate) % 60,
70.             (sfinfo.frames * sfinfo.channels) );
71.     numCanales = sfinfo.channels;
72.     bitsMuestra = ((sfinfo.format & SF_FORMAT_PCM_16) > 0? 16 :
73.     ((sfinfo.format & SF_FORMAT_PCM_S8) > 0? 8 : 0));
74.     if (bitsMuestra == 0)
75.         printf("Error en el fichero %s: bitsMuestra = %d\n",
76.                 infilename, bitsMuestra);
77.     frecuenciaMuestreo = sfinfo.samplerate;
78.
79.     formatoAudio = 0;
80.     if(sfinfo.channels == 1)
81.         formatoAudio = AL_FORMAT_MONO16;
82.         else if(sfinfo.channels == 2)
83.             formatoAudio = AL_FORMAT_STEREO16;
84.             else {
85.                 printf("Error: número de canales: %d\n",
86.                         sfinfo.channels);                return( -2 );
87.             }
88.             frame_size = (size_t)(BUFFER_LEN * sfinfo.channels) *
89.                                 sizeof(short);
90.             data = malloc(frame_size);
91.             for(i = 0; i < NUM_BUFFERS; i++) {
92.                 samples = sf_readf_short(infile, data, BUFFER_LEN);
93.                 if (samples > 0) {
94.                     mostresLLegides += samples;
95.                     printf("Ha llegit %ld mostres i en porta %ld\n",
96.                             samples, mostresLLegides);
97.                     mostresEnBytes = samples* sfinfo.channels *
98.                                     (sf_count_t)sizeof(short);
99.                     alBufferData(buffers[i], formatoAudio, data,
100.                                 mostresEnBytes, frecuenciaMuestreo);
101.                 }
102.             }
103.             alSourceQueueBuffers(alsource, NUM_BUFFERS, buffers);
104.             if((error=alGetError()) != AL_NO_ERROR)
105.                 printf("Error rellenando los buffers por 1a vez: %s\n",
106.                         alutGetErrorString(error));
107.             alSourcePlay(alsource);
108.             if((error=alGetError()) != AL_NO_ERROR)
109.                 printf("Que comence la música? Error: %s\n",
110.                         alutGetErrorString(error));
...
```

Listado 2: Ejemplo de lectura de fichero WAVE con libsndfile en OpenAL (parte 2).

```
...
111.            i=0;
112.            while (mostresLLegides < numTotalMostres) {
113.                alGetSourcei(alsource, AL_BUFFERS_PROCESSED,
114.                             &buffersLibres);
115.                if( buffersLibres > 0 ){
116.                  while(buffersLibres > 0) {
117.                   samples = sf_readf_short(infile, data, BUFFER_LEN);
118.                   if (samples > 0) {
119.                      mostresLLegides += samples;
120.                      alSourceUnqueueBuffers(alsource, 1, &bufferID);
121.                       mostresEnBytes = samples* sfinfo.channels *
122.                                        (sf_count_t)sizeof(short);
123.                       alBufferData(bufferID, formatoAudio,
124.                  (short*)data, mostresEnBytes, frecuenciaMuestreo);
125.                       alSourceQueueBuffers(alsource, 1, &bufferID);
126.                       buffersLibres--;
127.                       printf("%7ld / %7ld  mostres \r", sfinfo.frames,
128.                              mostresLLegides); fflush( stdout );
129.                       val=0;
130.                       alGetSourcei(alsource, AL_SOURCE_STATE, &val);
131.                       if(val != AL_PLAYING) alSourcePlay( alsource );
132.                    }//while( buffersLibres-- )
133.                }//if( buffersLibres > 0)
134.            } //while (mostresLLegides < numTotalMostres)
135.
136.           printf("\n");
137.
138.            i=0;
139.            do {
140.              alutSleep( 0.1f );
141.              printf("%c\r", barraActivitat[i++%4]); fflush( stdout );
142.              alGetSourcei(alsource, AL_SOURCE_STATE, &estado);
143.            } while(estado == AL_PLAYING);}
144.
145.            sf_close (infile); // Close input and output files
146.            free( data );
147.
148.            alDeleteSources(1, &alsource);
149.            alDeleteBuffers(NUM_BUFFERS, buffers);
150.            alutExit();
151.
152.            return EXIT_SUCCESS;
153.}
```

Listado 3: Ejemplo de lectura de fichero WAVE con libsndfile en OpenAL (parte 3).

Una vez sean añadidos a la pila de *buffers* (línea 103), se puede empezar a reproducir (línea 107) el audio en la fuente asociada .

El Listado 3 es el encargado de mantener la lista de *buffers* de la pila rellenados, conforme se va haciendo sonar su contenido (líneas 112 a la 134). Y, dependiendo, de la relación entre el tamaño del archivo y el de los *buffers*, puede que sea necesario esperar a que termine la reproducción del audio (líneas 138 a la 143). Tras lo cual, seguro que ha terminado y ya se pueden liberar los recursos asignados (líneas 145 a la 150).

Ya casi lo tenemos todo para probarlo, Nos falta indicar que es necesario instalar los paquetes de desarrollo de *libsndfile*, lo que puede hacer en Linux (en concreto hemos utilizado Ubuntu 20.04 LTS) con la orden:

```
$ sudo apt-get install libsnfdile1-dev
```

¿Lo está instalando? Bien hecho. Compruebe que lo que se dice aquí es cierto. Entonces aproveche también para localizar algún fichero WAV en su máquina, que lo vamos a necesitar enseguida. ¿Lo tiene? Yo tengo uno que se llama *escala.wav*, cuyas características son:

$ file escala.wav

escala.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 44100 Hz

Venga, seguimos adelante. Ahora ya podemos compilar y ejecutar el ejemplo (asumiendo que lo ha guardado, como yo, en un fichero de nombre *openal_libsdnfile__streaming.c*) con las órdenes:

```
$
 gcc openal_libsdnfile__streaming.c \
   -o openal_libsdnfile__streaming $(pkg-config openal --cflags --libs);
```

$ openal_libsndfile__streaming escala.wav

```
argv[1] = escala.wav
La versio de OpenAL instalada es 1.1 ALSOFT 1.19.1
La versio de ALUT instalada es 1.1
La versio de libsndfile instalada es libsndfile-1.0.28
Format 10002d: PCM_S8 (1d)?: 0d vs PCM_S16 (2d)?: 2d
escala.wav: canals 1, freqMostreig 44100, format 10002d [WAVE, 16bits],
total (4672512 mostres 9345024 bytes) i temps (105.952652 segons o 01 : 45
minuts:segons; frames 4672512)
```

El código ha sido comprobado con ficheros WAV de 1 y 2 canales, con muestras de 16 bits y con frecuencias de muestreo de hasta 48000 Hz. ¿No me diga que se va a quedar sin probar si funciona?

5 Conclusiones y cierre

OpenAL parte de una especificación reducida para conseguir un impacto mínimo en el consumo de recursos de una aplicación. Sus capacidades de importación de ficheros se pueden aplicar solo en modo precarga a ficheros WAVE utilizando el componente ALUT.

La existencia de tantos formatos de audio que podemos encontrar actualmente hace interesante que el desarrollador pueda seleccionar el uso de otras bibliotecas para gestionar los formatos de audio con mayor especificidad. En este caso, hemos utilizado *libsndfile* para leer e interpretar correctamente el contenido del fichero WAVE e ir

cargándolo en los bloques que su especificación define. De esta forma se reducen los requisitos de memoria, lo cual es interesante cuanto mayor es la duración del archivo.

Respecto al código mostrado, se puede agrupar en dos grandes funciones: una que inicialice la pila de *buffers* de la fuente y otra que vaya actualizando los contenidos de esta en cada llamada. Lo hemos obviado por brevedad de la exposición. Dejamos al lector que experimente con esta base para incorporarla a su aplicación. ¿Por qué no lo comprueba?

Por si le da pereza teclear el código, estoy subiendo el ejemplo a un repositorio de GitHub en la URL <https://github.com/magusti/OpenAL_examples/openal_libsndfile_streaming>. Espero que, a estas alturas, tenga ejecutándose el código propuesto y comprobando que puede oír el contenido de sus ficheros en formato WAVE en modo continuo (*streaming*). ¿No es así? Pues no tarde.

6 Bibliografía

[1] OpenAL. Disponible en <http://www.openal.org>.

[2] Panne, S. (2006). "The OpenAL Utility Toolkit (ALUT)". Disponible en <http://distro.ibiblio.org/rootlinux/rootlinux-ports/more/freealut/freealut-1.1.0/doc/alut.html>.

[3] - . (2005).*OpenAL 1.1 Specification*. Disponible en <http://www.openal.org/documentation/openal-1.1-specification.pdf>.

[4] Peacock, D, Harrison, P., Hiebert. G . (2007). OpenAL Programmer's Guide. OpenAL Versions 1.0 and 1.1 Disponible en <https://www.openal.org/documentation/OpenAL_Programmers_Guide.pdf>.

[5] IBM Corp. And Microsoft Corp. (1991).Multimedia Programming Interface and Data Specifications 1.0. <https://www.loc.gov/preservation/digital/formats/fdd/fdd000001.shtml>.

[6] "WAVE PCM soundfile format " <http://soundfile.sapp.org/doc/WaveFormat/ >.

[7] *The libsndfile Home Page*. Disponible en <https://libsndfile.github.io/libsndfile/ >.

[8] *The libsndfile API*. Disponible en <https://libsndfile.github.io/libsndfile/api.html>.

[9] Britten, B. (2010). Streaming in OpenAL. Disponible en <http://benbritten.com/2010/05/04/streaming-in-openal/ >.

[10] Daniel Peacock. D., Harrison, P. D'Orta, A., Carpentier, V. y Cooper, E . (2006).Effects Extension Guide. Disponible en <https://usermanual.wiki/Pdf/Effects20Extension20Guide.90272296/view >.

[11] C. Robinson. GitHub de OpenAL Soft. Disponible en <https://github.com/kcat/openal-soft/>.