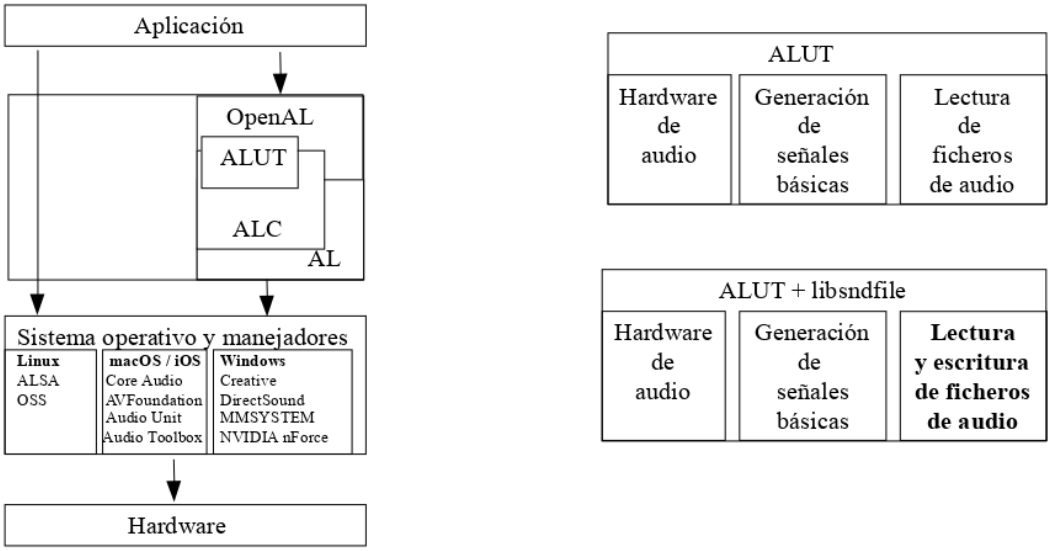
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

OpenAL: comparativa de ALUT y *libsndfile* *para reproducción en* precarga *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], un motor de audio 3D, 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 encuentran en un pequeño componente denominado "The OpenAL Utility Toolkit" [2] (ALUT, véase Fig. 1a) que junto a ALC y AL [3] conforman la jerarquía interna de OpenAL.



a) *b)*

Figura 1: Esquema de bloques del contexto de OpenAL y su extensión propuesta: (a) niveles originales en el estándar OpenAL y (b) bloques de funciones de ALUT y ALUT + libsndfile.

Actualmente, podemos encontrar un número muy alto de formatos de audio, por lo que en los desarrollos que utilizan OpenAL se opta por diferentes estrategias para ampliar esta interfaz. En un buen número de casos, se ha optado por seleccionar el uso de una biblioteca especializada solo en el formato que se vaya a utilizar para poder reducir la sobrecarga de código que puede no llegar a utilizarse en una aplicación. En el otro extremo, por las características de la aplicación desarrollada, tenemos a los desarrolladores que optan por una librería más genérica, en el caso que necesite emplear varios formatos.

ALUT [2] ha ido siendo intencionadamente restringido en su planteamiento inicial, limitándose a un pequeño número de operaciones relativas a la gestión de elementos complementarios a la funcionalidad del motor de audio espacial. Así vemos los tres grupos de operaciones que podemos encontrarnos en la versión original de ALUT (Fig. 1b) y que vamos a plantear aquí una opción para ampliar ese bloque de "Lectura de ficheros de audio" y extender las operaciones disponibles con una librería externa. Hemos escogido *libsndfile* que es [8] una biblioteca de funciones, realizada en lenguaje C, para el acceso (tanto para lectura, como para escritura) de un buen número de formatos de ficheros de audio (como WAVE, AIFF o FLAC, entre otros) y que ha sido publicada en código fuente y bajo una licencia GPL (*Gnu Lesser General Public License*).

2 Objetivos

Este artículo está enfocado a explorar una propuesta de cómo ampliar el conjunto de formatos de ficheros que puede utilizar OpenAL. Buscamos una opción sencilla, en cuanto a número de líneas de código, que nos permita precargar ("static playback" en la terminología de OpenAL) los ficheros de audio; para usarlos en nuestras aplicaciones con OpenAL.

Por lo que a partir del estudio del ejemplo de código que se aborda en este documento, el lector será capaz de:

- Incorporar ficheros de varios formatos a un desarrollo sobre OpenAL, más allá, de los que ofrece el interfaz original de ALUT.
- Identificar una posible estrategia para la reproducción de audio en modo precarga. Lo que significa que se realizará la carga completa (en memoria) del contenido de los ficheros de audio. Así que se describirá una posible secuencia de pasos para, utilizando *libsndfile*, mimetizar el interfaz de ALUT en la carga de audio desde fichero.
- 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 desarrollo de aplicaciones de sonido que utilizan OpenAL como elemento básico. Además, expondremos un poco de la forma en que se estructura el formato de audio WAVE, para poder entender cómo se adapta la solución propuesta.

3.1 Arquitectura de OpenAL

El esquema básico de trabajo que ofrece OpenAL para "renderizar" una escena sonora tridimensional se basa en la idea [5] que muestra la Fig. 2. En ella se observa la funcionalidad de OpenAL que se sustenta en tres objetos:

- Los "buffer", que contienen los diferentes sonidos sin compresión y obtenidos desde ficheros o generados de manera sintética.
- Estos se asignan a objetos de tipo "source", que reciben como entrada el sonido y los parámetros que definen su situación, orientación y velocidad, así como también la atenuación, la direccionalidad de la fuente, etc. De esta forma se puede calcular cómo se van a comportar estas fuentes de sonido.
- Finalmente, todos los sonidos con *renderizados* en la situación que define la posición del oyente a través del mezclador ("output mix"), para proporcionarle al oyente todos los sonidos parametrizados con los valores globales de la escena como la velocidad de transmisión del sonido, si se tiene en cuenta el efecto *Doppler*, posición y velocidad del oyente, etc.

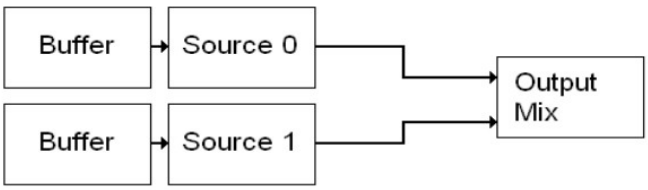


Figura 2: Arquitectura básica de OpenAL. Imagen extraída de [5].

3.2 Dentro de un fichero WAVE

Waveform Audio File Format (WAVE)[1] es [6] una especificación de fichero de audio en forma de ondas. Un fichero en este formato (generalmente con extensión ".wav"), 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 publicación inicial del formato, junto con su sencillez han hecho de él un formato de audio que ha sido portado a, posiblemente, todas las plataformas que hacen uso de audio digital.

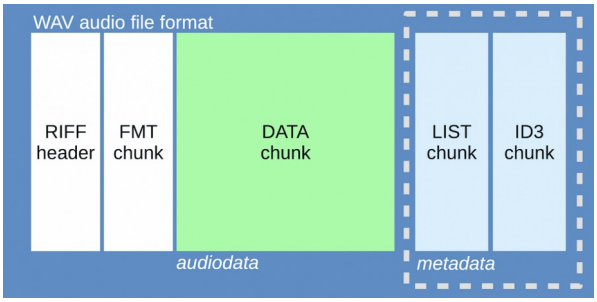


Figura 3: Bloques de un fichero de audio WAVE. Imagen extraída de [7].

El contenido de un fichero WAVE (como muestra gráficamente la Fig. 3) está organizado en una cabecera que contiene los parámetros que lo definen y los datos, que van agrupados en "chunks". Cada *chunk* tiene una cabecera que identifica su tipo y tamaño, más los datos que contiene. En WAVE cabe esperar de tipo: FMT (que describe el formato), de metadatos (LIST y/o ID3) y DATA (que es el audio en formato digital).

La cabecera del fichero WAVE especifica cuestiones como, por citar las más relevantes en nuestro caso, la frecuencia de muestreo (en un valor de 32bits), el tamaño de cada muestra de sonido digitalizado (entre 8 y 64 bits para valores enteros y entre 32 y 64 bits para valores reales), el número de canales (monofónico, estéreo y, a partir de una revisión del formato original, también multicanal o cuadrafónico), así como el tamaño máximo (actualmente en 4GBytes).

Dentro del *chunk* DATA, podemos encontrar las muestras de sonido (*samples*) en digital. Cada muestra 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 *frames*. Un *frame* es un conjunto de muestras para todos los canales utilizados, que han de sonar en un mismo instante de tiempo.

[1] 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>.

4 Desarrollo

Para llevar a cabo la extensión de ALUT, vamos primero a ver un ejemplo con esta capa de OpenAL. La Fig. 4a muestra la cadena de operaciones que llevan la información contenida en el fichero WAVE a un buffer de memoria que puede gestionar OpenAL. Desde aquí ya podremos exponer las modificaciones del mismo, con el uso de *libsndfile*, para la carga de audio desde fichero. Sustituyendo a ALUT por libsndfile, Fig. 4b, en la cadena de operaciones a emplear.

La Fig[2]. 4a resalta los nombres principales de las funciones sobre las que reside el peso de la operación y con las que veremos cómo implementar una funcionalidad equivalente a la función de ALUT (*alutCreateBufferFromFile*) con la de libsndfile (*sf_readf_short)* y *alBufferData* de la capa de bajo nivel (AL) de OpenAL.

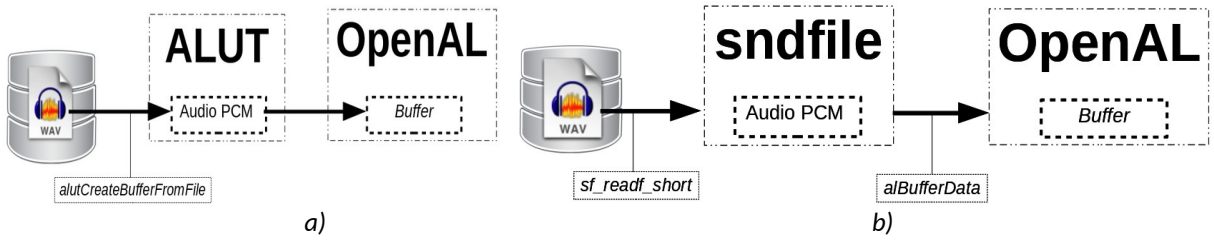


Figura 4: Esquema básico de gestión de formatos de ficheros en OpenAL: (a) con ALUT y (b) la propuesta de este trabajo utilizando libsndfile.

Como presentación de *libsndfile*[3] y para que pueda identificar su papel en esta propuesta, destacaremos que, de entre los más de veinticinco formatos soportados [9] 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* 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 que describen el audio contenido en el fichero.
- Las funciones de lectura (con prefijo sf_readf) para acceder a los frames de audio.

Con esas opciones, sustituiremos las operaciones de ALUT por las de *libsndfile* para leer el contenido de los ficheros de audio, como muestra la Fig. 4b.

[2] 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.

[3] El sitio web de libsndfile se puede encontrar en la URL <http://www.mega-nerd.com/libsndfile/>.

4.1 Ejemplo de código con ALUT

Para llevar a cabo el ejemplo de uso de *libsndfile*, primero veamos cómo se realiza la tarea en OpenAL con ALUT. En este contexto, la reproducción de un fichero WAVE se puede hacer como muestra el Listado 1 [4] y que pasamos a comentar brevemente.

Podemos ver cómo se inicializa la aplicación y OpenAL (líneas 13 a la 19), así como también la impresión de valores de las versiones de OpenAL y ALUT (líneas 21 y 24). Vamos comprobando la existencia de errores en cada paso, aunque no lo recalcaré para mantener la atención en la estructura del ejemplo. Si todo ha ido bien, podemos crear un *buffer* y cargarlo con ALUT, a partir de la ruta del archivo (línea 27).

Con el audio cargado en memoria, solo queda declarar una fuente (línea 31) y asociarle el buffer (línea 36) para poder hacerlo sonar (línea 37). Como no sabemos lo que va a durar, esperaremos a que termine de reproducirse (líneas 39 a la 44) para liberar los recursos solicitados (líneas 46 a la 47) y terminar la aplicación.

¡Ya lo tenemos! O casi: nos falta instalar las dependencias. 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 libsndfile1-dev
```

Ahora ya podemos compilar y ejecutar (asumiendo que lo ha guardado, como yo, en un fichero de nombre *openal_alut__precarga.c*) con la orden:

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

Y lo puede probar con un fichero WAVE (u otros formatos, FLAC, etc. soportados por *libsndfile*) que tenga a mano…, el nuestro se llama “escala.wav” y lo ejecutamos con:

```
$  openal_alut__precarga 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
```

Observará que el ejecutable escribe unos pocos mensajes en pantalla y empieza a reproducir el archivo ¿Lo está probando ya? ¡No espere, quiero que se convenza y no puedo adjuntar el audio en este documento!

Estoy subiendo los ejemplos que se muestran en este artículo a un repositorio de GitHub en la URL <https://github.com/magusti/OpenAL_examples/openal_libsndfile_precarga>. Puede descargarlos de allí y probarlos rápidamente.

```c
#include <stdio.h>
#include <stdlib.h>           // EXIT_SUCCESS
#include <AL/alut.h>

int main(int argc, char *argv[]) {
  ALuint bufferID;            // The OpenAL sound buffer ID
  ALuint sourceID;            // The OpenAL sound source
  ALint estado;
  ALenum error;
  char barraActivitat[4] = {'|', '/', '-', '\\'};
  int i;

  if( argc > 1) {
        printf("argv[1] = %s\n", argv[1]);
  }
  if (!alutInit(&argc, argv)) {
      printf("Fallo al iniciar openAL.\n");
      return( -1 );
  }

  printf ("La versio de OpenAL instalada es %s \n",
             alGetString(AL_VERSION) );
  printf ("La versio de ALUT instalada es %d.%d \n",
             alutGetMajorVersion(), alutGetMinorVersion());

  //Generar buffer y fuente
  bufferID = alutCreateBufferFromFile( argv[1] );
   if((error=alGetError()) != AL_NO_ERROR)
      printf("Error generando buffer desde %s\n", argv[1] );

  alGenSources (1, &sourceID);
  if((error=alGetError()) != AL_NO_ERROR)
     printf("Error generando la fuente: %s\n",
             alutGetErrorString(error));

  alSourcei (sourceID, AL_BUFFER, bufferID);
  alSourcePlay( sourceID );

  i=0;
  do {
     alutSleep( 0.1f );
     printf("%c\r", barraActivitat[i++%4]); fflush( stdout );
     alGetSourcei(sourceID, AL_SOURCE_STATE, &estado);
  } while(estado == AL_PLAYING);

  alDeleteSources( 1, &sourceID );
  alDeleteBuffers( 1, &bufferID );

  alutExit();
  return EXIT_SUCCESS;
}
```

Listado 1: Ejemplo de lectura de fichero WAVE con ALUT en OpenAL.

4.2 Variaciones en el código para el uso de *libsndfile*

Respecto al código mostrado en este apartado, se podría haber agrupado en una función de nombre, p. ej., *sndfileCreateBufferFromFile* o similar; que recoja todas las instrucciones de *libsndfile*. 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? Las líneas destacadas en negrita son en las que sugerimos al lector que centre su atención, puesto que son las que implementan la reproducción, a partir de la lectura del fichero con *libsndfile*. Este código es el resultado de pruebas propias, así como de la consulta de ejemplos, como en [8].

Para ver cómo queda con el uso de *libsndfile*, el Listado 2 muestra las modificaciones para sustituir las instrucciones de ALUT por las de *libsndfile*. Para ello aparecen las declaraciones de variables de nuevos tipos (líneas 14 y 15). Seguimos aprovechando las facilidades de ALUT para acceso al hardware (línea 23), tras lo que vemos mensajes sobre las versiones que estamos ejecutando de OpenAL, ALUT y *libsndfile* (entre las líneas 26 y 29). Las líneas 30 a la 36 sirven para inicializar la estructura de datos y abrir el fichero. Obtendremos los parámetros del audio y los mostraremos por pantalla (líneas 38 a la 56).

Siguiendo con el código del Listado 3 vemos cómo se obtienen los datos para configurar OpenAL (líneas 57 a la 69), lo que nos permite calcular la cantidad de espacio que hemos de reservar para guardar el audio (líneas 77 a 79) y ya podemos leer el contenido del fichero y asignárselo a un *buffer* (líneas 84 y 87). El resto de código es el mismo que en el Listado 1, excepto por la línea 109 que cierra el uso del fichero de partida.

¿Lo compilamos y ejecutamos? Venga, sí; vamos allá. Lo podemos hacer, asumiendo que lo hemos guardado en un fichero de nombre *openal_libsndfile__precarga.c*, con la orden:

```
$  gcc openal_libsndfile__precarga .c -o openal_ libsndfile__precarga $(pkg-config openal --cflags --libs)
```

Y lo puede probar con un fichero WAVE (o también, recuerde, FLAC y el resto de los que soporta *libsndfile*) que tenga a mano …, en nuestro caso se llama “escala.wav”:

```
$ openal_libsdnfile__precarga escala.wav
argv[1] = escala.wav
infilename = 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
escala.wav: canals 1, freqMostreig 44100, format 10002 [WAVE, 16bits], frames 4672512, frames Totals 4672512, 9345024 bytes i temps (105.952652 segons o 01:45 minuts)
Formats de 16 bits i 01 canal/s
Reservant lloc per a 9345024 bytes
Hem llegit 4672512 de 4672512 frames esperats.
Que sone la m�sica!
$
```

Pero también, recuerde, con FLAC y el resto de formatos soportados por *libsndfile*.

```
1.  #include <stdio.h>
2.  #include <stdlib.h>          // EXIT_SUCCESS
3.  #include <math.h>
4.  #include <AL/alut.h>
5.  #include <string.h> //memset
6.  #include <sndfile.h>
7.
8.  int main(int argc, char *argv[]) {
9.    ALuint bufferID, sourceID; // The OpenAL sound buffer & source
10.   char *datosAudio;          // The sound buffer data from file
11.   long tamanyBuffer;
12.   ALenum error, formatoAudio; //mono o stereo (1 o 2 canales)
13.   ALint nFrames,i,numCanales, frecuenciaMuestreo, bitsMuestra, estado;
14.   SNDFILE    *infile;
15.   SF_INFO    sfinfo ;
16.   char barraActivitat[4] = {'|', '/', '-', '\\'},
17.       infilename[128] = "escala.wav" ;
18.
19.   if( argc > 1) {
20.       strcpy(infilename, argv[1]);
21.       printf("infilename = %s\n", infilename);
22.   }
23.   if (!alutInit(&argc, argv))   {
24.       printf("Fallo al iniciar openAL.\n");     return 0;
25.   }
26.    printf ("OpenAL: %s \n", alGetString(AL_VERSION) );
27.    printf ("ALUT: %d.%d \n", alutGetMajorVersion(),
28.              alutGetMinorVersion());
29.    printf ("libsndfile: %s \n", sf_version_string());
30.
31.    memset (&sfinfo, 0, sizeof (sfinfo));
32.
33.   if (! (infile = sf_open (infilename, SFM_READ, &sfinfo))) {
34.       printf ("Not able to open input file %s.\n", infilename);
35.       puts (sf_strerror (NULL));
36.       return 1;
37.   } ;
38.   numCanales = sfinfo.channels;
39.   bitsMuestra = ((sfinfo.format & SF_FORMAT_PCM_16) > 0? 16 :
40.                  ((sfinfo.format & SF_FORMAT_PCM_S8) > 0? 8 : 0));
41.   frecuenciaMuestreo = sfinfo.samplerate;
42.   printf("%s: canals %d, freqMostreig %d, format %0x [%s, %s], frames
43.           %ld, frames Totals %ld, %ld bytes i temps (%04f segons o
44.           %02ld:%02ld minuts)\n",
45.           argv[1],
46.           numCanales,  frecuenciaMuestreo, sfinfo.format,
47.           ((sfinfo.format & SF_FORMAT_WAV) > 0? "WAVE": "no WAVE"),
48.           ((sfinfo.format & SF_FORMAT_PCM_16) > 0? "16 bits" :
49.               ((sfinfo.format & SF_FORMAT_PCM_S8) > 0? "8 bits" : "")),
50.           sfinfo.frames,
51.           (sfinfo.frames * sfinfo.channels), // Muestras en total?
52.              (sfinfo.frames * sfinfo.channels *
53.               ((sfinfo.format & SF_FORMAT_PCM_16) > 0? 2 : 1)),
54.           (float)sfinfo.frames / (float)sfinfo.samplerate,
55.           (sfinfo.frames  / sfinfo.samplerate) / 60,
56.           (sfinfo.frames  / sfinfo.samplerate) % 60   );
...
```

Listado 2: Ejemplo de lectura de fichero WAVE con libsndfile en OpenAL (parte 1).

```
...
57.   if (bitsMuestra == 0)
58.     printf("Error en el fichero, bitsMuestra = %d\n", bitsMuestra);
59.
60.   formatoAudio = 0;
61.   if(bitsMuestra == 8)  {
62.    if(numCanales == 1) formatoAudio = AL_FORMAT_MONO8;
63.    else if(numCanales == 2)  formatoAudio = AL_FORMAT_STEREO8;
64.   }
65.   else if(bitsMuestra == 16) {
66.    if(numCanales == 1) formatoAudio = AL_FORMAT_MONO16;
67.    else if(numCanales == 2) formatoAudio = AL_FORMAT_STEREO16;
68.   }
69.   if(!formatoAudio) {printf("Formato incompatible \n"); return( -2); ]
70.
71.   // Create sound buffer and source
72.   alGenBuffers(1, &bufferID);
73.   error = alGetError();
74.   if (error) printf("Error al crear el buffer:: alGetError: %s\n",
75.                     alutGetErrorString(error));
76.
77.   tamanyBuffer = (sfinfo.frames * sfinfo.channels *
78.                   ((sfinfo.format & SF_FORMAT_PCM_16) > 0? 2 : 1)) ;
79.   datosAudio = (char *)malloc( tamanyBuffer );
80.   if (datosAudio == NULL) {
81.         printf ("Error : no hay suficiente memoria para %ld bytes de
82.                  audio.\n", tamanyBuffer) ;          return( -1 );
83.     }
84.   nFrames = (int) sf_readf_short(infile, (short *) datosAudio,
85.                                  tamanyBuffer);
86.   alBufferData(bufferID, formatoAudio, datosAudio,
87.                tamanyBuffer, sfinfo.samplerate);
88.
89.   error = alGetError();
90.   if (error) printf("%s\n", alutGetErrorString(error));
91.
92.   alGenSources(1, &sourceID);
93.   error = alGetError();
94.   if (error) printf("Error al crear la font:: alGetError: %s\n",
95.                     alutGetErrorString(error));
96.
97.   alSourcei(sourceID, AL_BUFFER, bufferID);
98.   error = alGetError();
99.   if (error) printf("%s\n", alutGetErrorString(error));
100.
101.  alSourcePlay( sourceID );
102.  i=0;
103.  do {
104.     alutSleep( 0.1f );
105.     printf("%c\r", barraActivitat[i++%4]); fflush( stdout );
106.     alGetSourcei(sourceID, AL_SOURCE_STATE, &estado);
107.  } while(estado == AL_PLAYING);..
108.
109.  sf_close (infile) ;   /* Close input and output files. */
110. }
```

Listado 3: Ejemplo de lectura de fichero WAVE con libsndfile en OpenAL (parte 2).

El código ha sido comprobado con ficheros FLAC y 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

En el caso de OpenAL, se 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 a ficheros WAVE, utilizando el componente ALUT. En ese sentido, se ha revisado el camino que sigue la información de audio desde fichero en la forma de trabajo habitual de OpenAL y, así, proponer un ejemplo de uso de audio basado en la librería *libsndfile*.

Nos hemos centrado en el caso de precarga, no en el *streaming* (más interesante para ficheros de gran tamaño), pero ahora el lector ya puede experimentar con esta base para incorporarla a su aplicación. ¿Por qué no lo comprueba?

Espero que, a estas alturas, tenga ejecutándose el código propuesto y comprobando que puede oír el contenido de sus ficheros en formato WAVE, FLAC y el resto de los soportados por *libsndfile* ¿No es así? Pues busque alguno de estos ficheros en su equipo y no cierre el documento sin haberlo comprobado antes.

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] 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 >.

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

[7] "How to open WAV files [Waveform Audio File Format] - Guide " <https://samplerateconverter.com/wav>.

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

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