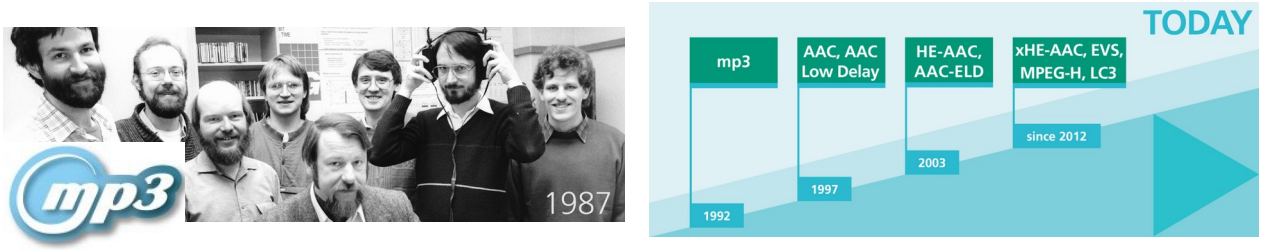
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Extendiendo OpenAL con ficheros MP3 y libmpg123

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

MP3 es [1], probablemente[1], el formato de audio más usado actualmente en los dispositivos electrónicos. Se desarrolló en el *Fraunhofer IIS*, entre el 1987 (véase Figura 1a) y el 1992. Actualmente hay muchos ficheros de audio que compiten por el "mercado" digital de este media, puede ver una pequeña introducción y caracterización de algunos de ellos en [8]. Por su diseño, MP3 se ha portado a todas las plataformas y, recientemente, las patentes han expirado [2], así que ya forma parte del dominio público. Lo cual es una ventaja respecto a otros como AAC y otras recientes innovaciones del propio *Fraunhofer IIS (*Figura 1b). Y con respecto a otros competidores que están obteniendo mejores resultados en las comparativas, como Vorbis y Opus, MP3 está disponible en la mayor parte de entornos de desarrollo actuales, aunque en el caso de Opus ya es habitual verlo también en todas partes.



a) *b)*

Figura 1: MP3 nace en el Fraunhofer IIS: (a) Equipo de desarrollo <https://www.mp3-history.com/> y logo <https://www.audioblog.iis.fraunhofer.com/mp3-software-patents-licenses/mp3-logo> y (b) línea de tiempo de los desarrollos del Fraunhofer IIS <https://www.iis.fraunhofer.de/en/ff/amm.html>.

OpenAL es [3] un motor de audio 3D capaz de *renderizar* el sonido que llega hasta el oyente, creando una escena sonora tridimensional que proporciona una inmersión mayor del usuario en el conjunto de estímulos sonoros que le llegan al oyente (usuario). Las operaciones básicas de OpenAL no se encargan de cargar audio desde disco, sino que operan cuando ya está cargado en memoria y sin compresión, asociándolo a una fuente de sonido, para crear el sonido espacial. Esta ha sido una decisión de diseño (véase [4] y [5]), que llevó a la creación de una componente, *Audio Library Utility Toolkit* o ALUT [6], para proporcionar una capa de nivel superior que se encargue de cargar ficheros de audio, en concreto, ofrece la posibilidad de cargar ficheros WAVE desde archivo.

En este artículo se verá el uso del formato de audio MP3 [7] y cómo utilizar mpg123 [10], para incorporar este formato a la dinámica de uso típica en OpenAL.

2 Objetivos

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

- Incorporar ficheros de formato MP3 a un desarrollo sobre OpenAL.

1 Véase el comentario "Top 10 Audio File Formats" en <https://www.sageaudio.com/blog/recording/top-10-audio-file-formats.php>.

- Identificar una posible estrategia para la carga completa en memoria de ficheros MP3.
- Instalar y compilar una aplicación que hace uso del formato MP3 sobre OpenAL, con las funciones de la biblioteca *libmpg123* en un sistema operativo GNU/Linux.

No es el objetivo de este documento dar una solución global para todos los formatos, sino mostrar cómo incorporar este formato al repertorio de archivos que puede utilizar una aplicación basada en OpenAL. El presente documento está encaminado a ofrecer una perspectiva inicial de cómo ampliar el conjunto de formatos de ficheros que puede utilizar el motor de audio 3D OpenAL.

3 Introducción

El sonido en digital está en todas partes: música, voz, comunicaciones, ….) [9]. La complejidad de los formatos actuales que, como MP3, incorporan compresión con pérdidas, psicoacústica, *streaming* o sonido multicanal, propicia que se desarrollen librerías específicas como *mpg123*, SDL o MAD.

mpg123 es un proyecto que culmina en un reproductor de audio, basado en el decodificador implementado en la librería *libmpg123* [10]. En particular, *libmpg123* ofrece:

1 Capacidades de decodificación para los diferentes niveles de audio del estándar MPEG en tiempo real.

2 Está basado en los estándares[2] ISO/IEC 11172-3 e ISO/IEC 11172-4 que definen MP3.

3 Disponible para diferentes plataformas de computadores de escritorio, móviles, videoconsolas (tanto fijas como portables) y sistemas empotrados, con menor capacidad computacional o sin unidad de coma flotante.

4 Distribuido bajo licencia GNU Lesser General Public License (LGPL).

Utilizaremos *libmpg123* para extender el conjunto de formatos que se pueden importar en una aplicación sobre el motor de audio 3D de OpenAL y bajo lenguaje C. Y lo haremos de forma práctica, viendo qué hay que instalar y cómo compilar y ejecutar un ejemplo de código.

3.1 Estructura de un fichero MP3

Un archivo MPEG de audio [11] está compuesto de una sucesión de trozos o *frames*, como muestra la Figura 2a, no tiene un único bloque de cabecera y uno de datos. En su lugar, cada *frame* tiene su propia cabecera y datos (de audio). Al inicio del fichero se pueden encontrar los metadatos (autor, fecha, obra con la que está vinculada la pieza sonora e, incluso, una imagen), en formato ID3Tag o Xing[3].

Cada *frame* (Figura 2b), empieza con los bits para sincronización y, después, la las propiedades como el *layer* de audio dentro del estándar MPEG, el tamaño de muestra y la frecuencia de muestreo. Además, está el CRC (que permite comprobar la validez del *frame*) y el relleno (*padding*, que se desechará) para que el tamaño del *frame* sea fijo. Para obtener la información del audio digital es necesario recuperar el primer *frame* y leer su cabecera, siempre y cuando haya sido generado con un *bitrate* constante. Por limitaciones de la extensión

2 Puede ver los análisis de compatibilidad de decodificación de audio en MPEG en <https://www.underbit.com/resources/mpeg/audio/compliance/>.

3 Puede ver más sobre este tema, en <https://handwiki.org/wiki/MP3 >.

del documento, vamos a dejar fuera de nuestra solución a los ficheros de *bitrate* variable (vbr o *Variable Bit Rate*).

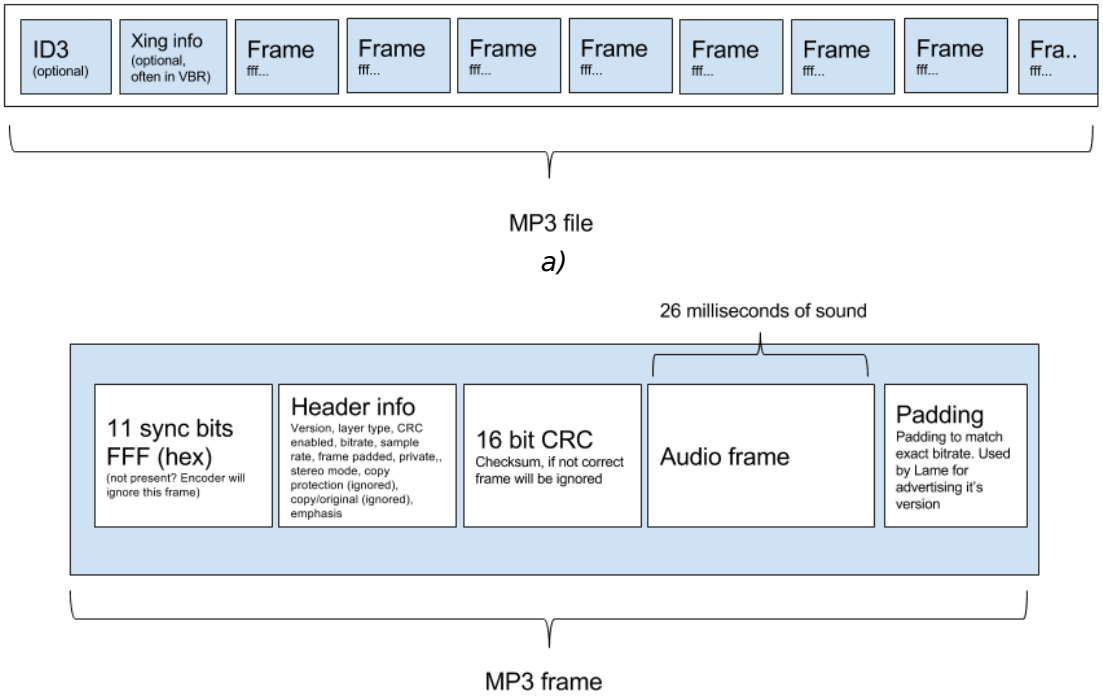


Figura 2: MP3: (a) contenido del fichero y (b) estructura de cada frame [1].

3.2 Propuesta de uso de MP3 con mpg123

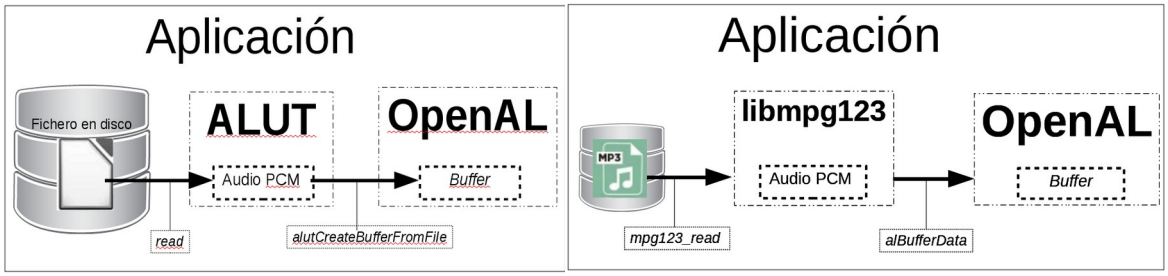
Dada la complejidad de procesar el contenido de un fichero MP3, buscaremos apoyarnos en los servicios que ofrece el API de *mpg123* a través de la biblioteca de funciones *libmpg123: esta* ofrece [10] una serie de operaciones a llevar a cabo sobre un fichero MP3 de forma bastante transparente a todos los detalles de la estructura interna del fichero. Todas ellas giran en torno a la estructura *mpg123_handle* que encapsula los datos que el decodificador necesita para, no solo leer el fichero, sino también extraer de la secuencia de *frames* leídos, la información de audio descomprimiéndola sobre la marcha.

Para poder leer los *frames*, libmpg123 proporciona la función *mpg123_decode* con lo que podremos identificar si es el *frame* de formato o de metadatos y obtener con *mpg123_getformat*, las propiedades del audio y los *frames* que componen el fichero. Tras lo cual, las sucesivas invocaciones a *mpg123_decode*, serán capaces de extraer (decodificar) el audio.

OpenAL es capaz de reproducir audio, en PCM, cargándolo totalmente en memoria o en modo de reproducción en continuo (*streaming*), esta segunda técnica se adapta perfectamente a esta capacidad de leer a trozos el fichero MP3 que nos ofrece *libmpg123* e ir reproduciéndolo sin esperar a tener todo el fichero cargado en memoria.

La Figura 3a muestra, de forma gráfica, como la componente ALUT [6] de OpenAL, es capaz de leer archivos 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, para los archivos MP3 que esta componente no soporta, la solución propuesta sigue la idea de leer el contenido de disco y llevarlo a la zona de memoria (*buffer*) desde donde puede

acceder OpenAL, véase Figura 3b. El uso de la librería *libmpg123* permite ir leyendo los trozos (*frames*) en que está dividido interiormente el archivo MP3, así que hay que averiguar cuál es el tamaño de estos para dimensionar las estructuras de datos en memoria que gestiona la aplicación.



a) *b)*

Figura 3: Esquema básico de gestión de formatos de ficheros en OpenAL: (a) con ALUT y (b) la variación propuesta en este trabajo con libmpg123.

4 Desarrollo sobre mpg123

Integrado en el reproductor de audio de línea de órdenes que implementa el proyecto *mpg123* tenemos un posible ejemplo de código que podría ser la solución a este problema[4], en el fichero *openal.c*: que ofrece una salida de audio hacia un servidor sobre OpenAL. Pero no es un ejemplo independiente con el que podamos experimentar. Así que nos basaremos en el ejemplo[5] de Han [6], que ofrece una solución (solo para el sistema Windows) y que extenderemos para un uso multiplataforma (la probaremos sobre Linux) y lo completaremos con las indicaciones de instalación y generación del ejecutable que no incorpora este ejemplo.

Se necesita instalar los paquetes de desarrollo de *libmpg123*, lo que puede hacer en Linux (en concreto hemos utilizado Ubuntu 20.04 LTS) con la orden:

```
$ sudo apt-get install libmpg123-dev
```

¿Lo está instalando? Bien hecho. Compruebe que lo que se dice aquí es cierto. Entonces, aproveche también para localizar algún fichero MP3 en su máquina. ¿Lo tiene? ¿Todavía no? Bueno, todavía no lo necesitamos, pero no tardaremos. Seguimos adelante.

Si ha descargado la versión que hemos realizado a partir del ejemplo original de Han [6], ahora podremos compilar y ejecutar el ejemplo (asumiendo que el fichero se llama *openal_mpg123.c*) que vamos a mostrar y comentar, en los próximos párrafos, con la orden:

```
$ gcc openal_mpg123.c -o openal_mpg123 $(pkg-config  --cflags --libs libmpg123) $(pkg-config  --cflags --libs  openal)
```

El código completo, con las modificaciones que ahora se comentarán, está alojado en un repositorio de *GitHub*[6] para facilitar su acceso. El código de este ejemplo está repartido entre los Listado 1 al Listado 3. ¿Lo tenemos ya?

4 Véase el repositorio <https://github.com/gypified/libmpg123/blob/master/src/output/openal.c>.

5 Este ejemplo está referenciado en otros como “Use openal and mpg123 to play MP3, (transfer) “ <https://blog.actorsfit.com/a?ID=00500-d87f3bc7-b1d2-430e-af81-dd92dab07c56 >.

Entonces seguimos con la explicación del código de ejemplo de uso de *libmpg123* con OpenAL.

Empieza en el Listado 1, donde podemos encontrar:

- Las definiciones de variables relativas a OpenAL (empiezan con el prefijo AL) y, en especial, en la línea 21 está la referencia al fichero (observe que *mpg123* es del tipo *mpg123_handle* que hemos mencionado anteriormente) que nos permitirá acceder a la información del MP3.
- Y las inicializaciones y comprobaciones propias de libmpg123:
  - Primero corresponde establecer el funcionamiento de *libmpg123* con *mpg123_init* (línea 23).
  - Después, se ha de inicializar la estructura que nos sirve de referencia al fichero MP3, con *mpg123_new* (línea 26) y donde se escoge el decodificador de MPEG a utilizar, en nuestro caso *mpg123_decoders()[0]*, corresponde a "AVX", que es el conjunto de instrucciones optimizadas de los procesadores de la familia x86 que se diseñaron para el tratamiento de datos multimedia.
  - Ahora ya se puede obtener la referencia del fichero MP3 con *mpg123_open* (línea 28), que asociará la misma al fichero cuya ruta en disco recibe. Lo que permite ya acceder a los contenidos del fichero de audio, observe que lo primero será acceder a las propiedades del *frame* de formato con *mpg123_getformat* (línea 33), de donde se obtienen el número de canales y de donde se deriva el tamaño del *buffer* donde se transferirán los *frames* de datos conforme se vaya leyendo.

El Listado 2 muestra la parte de inicialización de OpenAL:

- En la línea 42, se obtiene el subsistema de audio por defecto que ofrece el sistema a las aplicaciones *co*n alcOpenDevice. De él se genera una escena inicial con un oyente con *alcCreateContext* (línea 44), los *buffers* (línea 49 con *alGenBuffers*) y la única fuente sonora que se utilizará en este ejemplo (línea 53 con *alGenSources*). El código establece un número fijo de *buffers*, podría buscarse ese valor en tiempo de ejecución; pero, para simplificar el ejemplo, lo dejaremos con el valor original que suele funcionar bien.
- Y podemos empezar a ver la relación entre las acciones de *libmpg123* y OpenAL. Las instrucciones de las líneas 56 a la 60, crean la zona de memoria que se utilizará para llevar la información de audio del fichero al sistema de renderización de audio de OpenAL. En el bucle podemos ver la secuencia de las tres operaciones involucradas, que veremos repetidas más tarde:
  - Por la parte de *libmpg123*, la función *mpg123_read* lee un *frame* de disco, lo decodifica y deja el audio descomprimido en *pData*,
  - Por parte de OpenAL,
    - *alBufferData* copia este audio en uno de los cuatro *buffers* creados.

6 Puede encontrarlo en el directorio OpenAL_libmpg123 de *Github* de *magusti/OpenAL_examples* en <https://github.com/magusti/OpenAL_examples>.

```
1.  /* Modificaciones sobre la versión original
2.   https://www.cnblogs.com/fancycode/archive/2012/06/02/2531879.html
3.  *AUTHOR bowman han.  EMAIL fancycode+blogcn123@gmail.com.  DATA  6/2/2012 */
4.  #include <stdio.h>
5.  #include <unistd.h>
6.  #include <curses.h>
7.  #include <AL/al.h>
8.  #include <AL/alc.h>
9.  #include <mpg123.h>
10.
11. #define NUM_BUFFERS 4
12. int main( int argc, char **argv ) {
13.     ALuint g_Buffers[NUM_BUFFERS], uiSource, uiBuffer;
14.     ALCdevice * pDevice = NULL;             ALCcontext *pContext = NULL;
15.     ALboolean g_bEAX;                       ALenum error, eBufferFormat;
16.     ALint iDataSize, iFrequency;            ALuint ulFormat;
17.     int iLoop,iBuffersProcessed, iTotalBuffersProcessed, iEncoding, iChannels,
18.         encoding, iState, iQueuedBuffers, iMpg123_error;
19.     void *pData = NULL;                      long lRate;
20.     unsigned long ulDataSize = 0, ulFrequency = 0, ulBufferSize, ulBytesWritten;
21.     mpg123_handle *mpg123;
22.
23.     if(MPG123_OK != (iMpg123_error = mpg123_init()))  {//init mpg123 library
24.         printf("failed to init mpg123\n"); return -1;    }
25.     //open a default mpg123 decoder
26.     mpg123 = mpg123_new(mpg123_decoders()[0], &iMpg123_error);
27.     //try to open a mp3 file, modify to your own mp3 files
28.     if(MPG123_OK != (iMpg123_error = mpg123_open(mpg123, argv[1])))   {
29.         fprintf(stderr,"error in open mp3 file\n");
30.         return -1;
31.     }
32.     //get mp3 format infomation
33.     mpg123_getformat(mpg123, &lRate, &iChannels,  &iEncoding);
34.     if(iChannels ==2)    { //there only parse stereo mp3 file
35.         ulFormat = alGetEnumValue("AL_FORMAT_STEREO16");
36.         ulBufferSize = lRate;  //set buffer to 250ms
37.         ulBufferSize -= (ulBufferSize % 4);  //set pcm Block align
38.         ulFrequency = lRate;  //set pcm sample rate
39.     } else  { printf("channels info%i\n", iChannels);  return -3; }
40.
 ...
```

Listado 1: Listado de mp3_openal.c (parte 1).

```
...
41.     //open a openal default device
42.     pDevice = alcOpenDevice(NULL); //select the perfered device
43.     if(pDevice){
44.         pContext = alcCreateContext(pDevice, NULL);
45.         alcMakeContextCurrent(pContext);
46.     }else {
47.         printf("failed to get a openal decice\n");  return -2;
48.     }
49.     alGenBuffers(NUM_BUFFERS, g_Buffers);   //Generate openal  Buffers
50.     if((error = alGetError()) != AL_NO_ERROR) {
51.         fprintf(stderr, "alGenBuffers : error %d\n", error);
52.     }
53.     alGenSources(1, &uiSource);
54.
55.     pData = malloc(ulBufferSize);
56.     for(iLoop= 0; iLoop < 4; iLoop++) {    //feed data to openal buffer
57.         mpg123_read(mpg123, (char *)pData, ulBufferSize,&ulBytesWritten);
58.         alBufferData(g_Buffers[iLoop], ulFormat, pData, ulBytesWritten,
59.                         ulFrequency);
60.         alSourceQueueBuffers(uiSource, 1, &g_Buffers[iLoop]);
61.     }
...
```

Listado 2: Listado de mp3_openal.cL (parte 2).

- Y *alSourceQueueBuffers* que pone en cola ese trozo de sonido para que se pueda reproducir en *streaming*, esto es cada vez que se haya reproducido el contenido de uno de los *buffers* ser marcará como libre y se continuará con el siguiente. Así hasta que lo estén todos.

Por último, en el Listado 3 empezamos a hacer sonar la música que acaba de cargar con la instrucción *alSourcePlay* de la línea 62. A continuación observamos la secuencia de instrucciones (línea 64 a la 79) encargadas de continuar la lectura de los *frames* de disco, rellenando los *buffers* conforme se van liberando tras ser reproducidos:

- Primero se comprueba si hay ya trozos utilizados en la reproducción del sonido, la instrucción *alGetSourcei* de la línea 66. Observe que el resultado ahí obtenido regula el bucle *while* más interno que repite la secuencia anterior de liberar esos *buffers* y rellenar con otros *frames* del fichero MP3 con *alSourceUnqueueBufffers*, *mpg123_read*, *alBufferDat*a y *alSourceQueueBuffers*. La diferencia con el rellenado inicial, que ha sido un poco confiado hay que admitirlo, es que ahora se comprueba si el tamaño del fichero da para rellenar todos los *buffers*, esto es, si la lectura del archivo de audio no ha llegado a su fin.

- El resto del código de este Listado 3 mantiene la reproducción mientras quedan datos de audio y, al terminar, se liberan recursos. Especialmente hemos de mencionar la línea 88 que es la instrucción *mpg123_close* que libera los recursos utilizados por ese fichero.

```
...
62.    alSourcePlay(uiSource);
63.    iTotalBuffersProcessed = 0;     printf("playing\n");
64.    while(1) {
65.        iBuffersProcessed = 0; usleep( 20000 ); // microsecond intervals
66.        alGetSourcei(uiSource, AL_BUFFERS_PROCESSED, &iBuffersProcessed);
67.        iTotalBuffersProcessed += iBuffersProcessed;
68.        printf("Buffers total Processed %d\r", iTotalBuffersProcessed);
69.      while(iBuffersProcessed)        {
70.       uiBuffer = 0;
71.       alSourceUnqueueBuffers(uiSource, 1, &uiBuffer);
72.       mpg123_read(mpg123, (char *)pData, ulBufferSize, &ulBytesWritten);
73.        if(ulBytesWritten)    {
74.          alBufferData(uiBuffer, ulFormat, pData, ulBytesWritten,
75.                       ulFrequency);
76.          alSourceQueueBuffers(uiSource, 1, &uiBuffer);
77.        }
78.       iBuffersProcessed--;
79.      }
80.      alGetSourcei(uiSource, AL_SOURCE_STATE, &iState);
81.      if(iState != AL_PLAYING)    {
82.        alGetSourcei(uiSource, AL_BUFFERS_QUEUED, &iQueuedBuffers);
83.        if(iQueuedBuffers)  {
84.            alSourcePlay(uiSource);//buffers have data, play it
85.        } else { break; }  //there is no data any more
86.      }
87.   }
88.   mpg123_close( mpg123 );  //close mpg123
89. //stop the source and clear the queue
90. alSourceStop(uiSource);
91. alSourcei(uiSource, AL_BUFFER, 0);
92. free(pData);
93. pData = NULL;
94. return 0;
95.}
```

Listado 3: Listado de mp3_openal.c (parte 3).

5 Conclusiones y cierre

OpenAL ofrece la capacidad de importar solo ficheros en formato WAVE utilizando un componente interno como es ALUT. La existencia de un alto número de formatos de audio que podemos encontrar actualmente, hace interesante que el desarrollador pueda seleccionar el uso de bibliotecas especializadas para incorporar un formato no soportado directamente por OpenAL. En ese sentido, se ha revisado la estructura de un fichero MP3 y cómo el proyecto *mpg123* proporciona la librería *libmpg123* que hemos mostrado cómo utilizar para proponer un ejemplo de reproducción sobre el formato MP3.

Dado que nuestro objetivo era un caso de uso concreto, dejamos al lector interesado en ampliar el conocimiento de esta librería, para lo que sugerimos empezar alguno de los varios pequeños ejemplos[7] de la documentación y que animamos a ponerlos en marcha para comprobar otras opciones.

Espero que, a estas alturas, tenga ejecutándose el código propuesto y comprobando que puede oír el contenido de sus ficheros en formato MP3[8]. ¿No es así? Pues no cierre el documento sin haberlo hecho.

6 Bibliografía

[1] MP3. Cast Protocols. Disponible en <https://cast.readme.io/docs/mp3 >.

[2] MP3. (2017). Fraunhofer Institute for Integrated Circuits IIS. Disponible en < https://www.iis.fraunhofer.de/en/ff/amm/consumer-electronics/mp3.html >.

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

[4] Loki Software. (2000). OpenAL 1.0.Specification. Disponible en <https://pdfs.semanticscholar.org/831a/72e74a6f63dafb1ff74dfa5e311f416bc238.pdf>.

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

[6] The OpenAL Utility Toolkit. Disponible en <http://distro.ibiblio.org/rootlinux/rootlinux-ports/more/freealut/freealut-1.1.0/doc/alut.html >.

[7] The mp3 History. Fraunhifer IIS. Disponible en <https://www.mp3-history.com/ >.

[7] Free Lossless Audio Codec. Sitio web. Disponible en <https://xiph.org/flac/>.

[8] -. (2021). Audio File Formats Explained. Mastering the Mix. <https://www.masteringthemix.com/blogs/learn/audio-file-formats-explained>.

[9] Fraunhofer IIS. (2020). Digital music is everywhere. Google Arts & Culture. Disponible en <https://artsandculture.google.com/asset/_/oAFSPmRyy0vP-g>.

[10] mpg123 - Fast console MPEG Audio Player and decoder library. Disponible en <http://www.mpg123.de/ >.

[11] G. Bouvigne. (2001). MPEG Audio Layer I/II/III frame header. Disponible en <http://www.mp3-tech.org/programmer/frame_header.html >.

7 Se pueden encontrar esos ejemplos en la documentación del API en la URL: <https://www.mpg123.de/api/group__mpg123__examples.shtml>.

8 Si no tiene ninguno a mano, sería extraño, puede buscar en <https://myfreemp3.to/>.

[12] M. J. Buenconsejo. libmad - MPEG audio decoder library. Github. Disponible en < https://github.com/markjeee/libmad >.

[13] API documentation for libmpg123, libout123, and libsyn123. Disponible en <http://www.mpg123.de/api/>.

[14] B. han. (2012). fancy han: openal, mpg123 y MP3. Disponible en <https://www.cnblogs.com/fancycode/archive/2012/06/02/2531879.html> .