UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Extendiendo OpenAL con ficheros MP3 y libMAD

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

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Resumen de las ideas clave

MP3 es [1] probablemente el formato de audio más usado en Internet, desde que en la década de los 90 apareció este formato y porque se adapta muy bien para la transmisión en vivo (*streaming*). Por su diseño, MP3 se ha adaptado a todas las plataformas y las patentes han expirado [2], así que ya forma parte del dominio público; lo cual es una ventaja respecto a otros como AAC. Y con respecto a Vorbis y Opus, MP3 está disponible en todos los SDK de desarrollo de las plataformas actuales.

OpenAL es [3] un motor de audio 3D capaz de renderizar el sonido que llega hasta el oyente, creando una escena sonora envolvente que proporciona una inmersión mayor del usuario en el conjunto de estímulos sonoros que se le proporcionand. OpenAL no se encargará de cargar audio desde disco, sino que cuando ya está cargado en memoria y sin compresión, lo asigna 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 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.

Actualmente hay muchos ficheros de audio que compiten por el "mercado" digital de este formato, puede ver una pequeña introducción y caracterización de algunos de ellos en [8]. Con la aparición de nuevos formatos de audio, se puede recurrir a las librerías propias de cada uno de ellos para ampliar el conjunto de ficheros de los que OpenAL puede importar audio. De esta manera, se mantiene bajo el consumo de recursos que necesita una aplicación: ajustándose solo los formatos que se sepa que se van a utilizar. En este artículo se verá el uso del formato de audio MP3) [7] y cómo utilizar *MPEG Audio Decoder (MAD [10])*, para incorporar este formato a la dinámica de uso típica en OpenAL.

2 Objetivos

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

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

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.

Así que buscamos una opción sencilla, en cuanto a número de líneas de código, que nos permita precargar los ficheros MP3 para usarlos en nuestras aplicaciones que, con OpenAL, construyan una escena sonora tridimensional.

3 Introducción

El sonido en digital está [9] en todas partes: música, voz, sonido, comunicaciones, ….). 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 SDL, MPG123 o MAD.

MAD, implementada en la librería *libmad, ofrece [10]:*

1. *Sonido PCM con muestras de 16 o 24 bits.*
2. *Implementación basada en coma fija (enteros), que la hace interesante para dispositivos, como móviles y sistemas empotrados, con menor capacidad computacional o sin unidad de coma flotante.*
3. *Basado en los estándares ISO/IEC 11172-3 e ISO/IEC 11172-4 que definen MP3.*
4. *Distribuido bajo licencia GNU General Public License (GPL).*

Utilizaremos MAD/libmad 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.

3.1 Dentro de un fichero MP3

Un archivo MPEG de audio [11] no tiene una cabecera global, sino que está compuesto de una sucesión de trozos (*frames),* Figura 1a). Cada *frame* tiene su propia cabecera, dato de audio. Al inicio del archivo se pueden encontrar metadatos (autor, fecha, obra con la que está vinculada la pieza sonora e, incluso, una imagen), en formato ID3Tag o Xing[1].

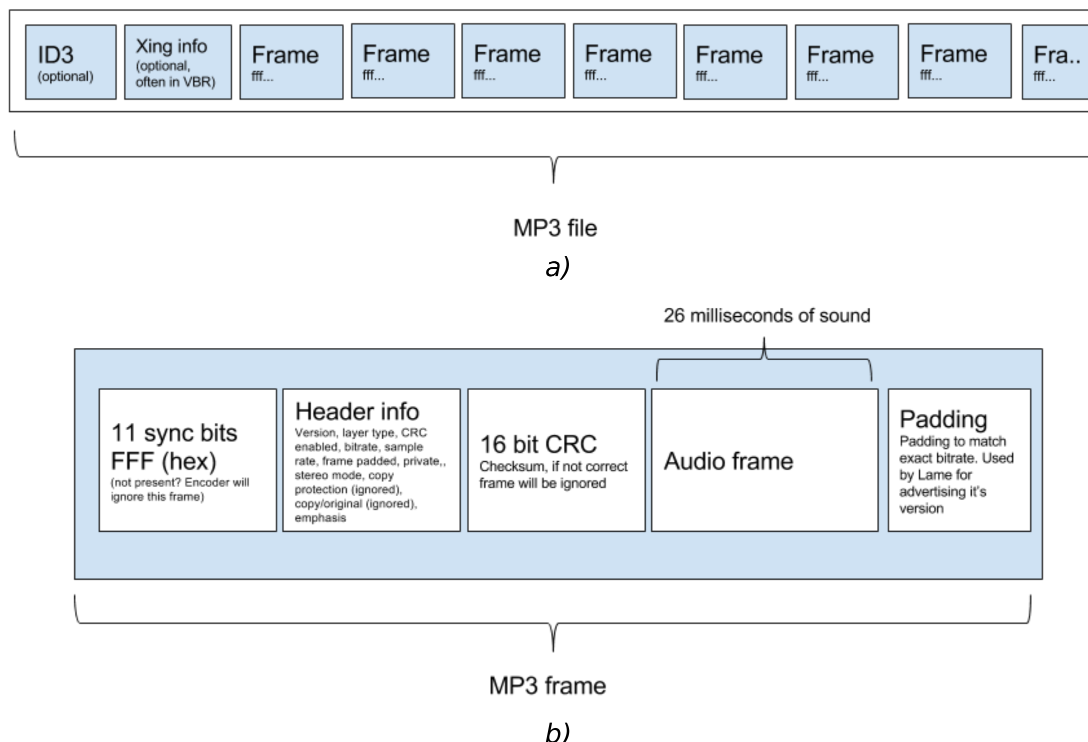

Figura 1: MP3: (a) contenido del fichero y (b) estructura de cada frame [1].

Cada *frame*. Figura 1b), empieza con los bits para sincronización y, después, la cabecera, que indica el *layer* de audio dentro del estándar MPEG, el tamaño de

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

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.

Dada la complejidad de procesar el contenido del fichero MP3, buscaremos apoyarnos en el trabajo de MAD [10] de Robert Leslie en *Underbit Technologies, Inc.*, disponible en los repositorios de Linux o en repositorios de *Github* como [12].

3.2 Propuesta de uso de MP3 con libmad

La librería *libmad* ofrece [13] dos perfiles o modalidades de acceso a los ficheros que se denominan *High-Level API* y *Low-Level API, que se encargan de:*

- *El de alto nivel,* permite asignar las funciones (*callbacks*) que serán notificadas a cada paso de la decodificación del fichero (*bitstream* o *stream*).
- El de bajo nivel, permite acceder a cada paso del proceso de decodificación explícitamente

La primera forma de uso está ejemplificada en un fichero (minimad.c) que mapea el fichero en memoria (con mmap) y encadena una secuencia de funciones que abrirán el stream (fichero), procesarán los frames y obtendrán de estos el audio digital. Al final de esta cadena, se llama a una función que será la encargada de convertir el audio digital en el formato que se necesite según la aplicación a desarrollar.

OpenAL es capaz de reproducir audio, en PCM, desde memoria, pudiendo trabajar en modo de precarga o de reproducción en continuo (*streaming*). La precarga supone llevar a memoria, por completo, el contenido del fichero de audio, para después reproducirlo. La fig. 2 muestra de forma gráfica como la librería *Audio Library Utility Toolkit* (ALUT) [6] complementa a OpenAL, en tanto 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.

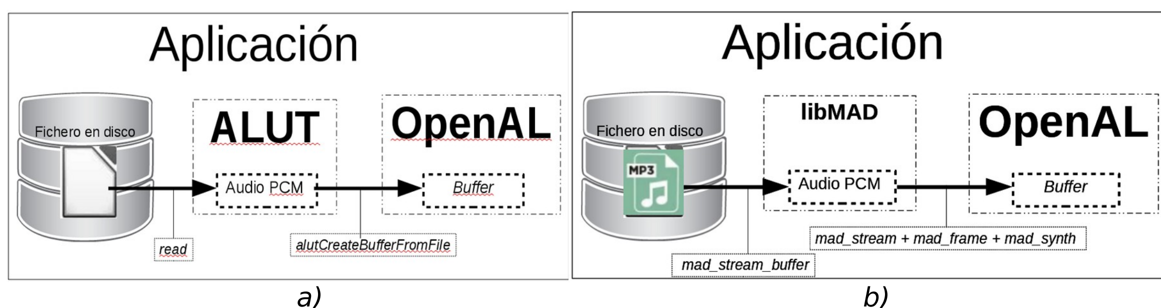

Figura 2: Esquema básico de gestión de formatos de ficheros en OpenAL: (a) con ALUT y (b) la variación propuesta en este trabajo con MAD/libmad.

Para poder ofrecer una visión alternativa, en este trabajo desarrollaremos la vertiente de "Low Level API" para cargar completamente el fichero en memoria y extraer el sonido. De este modo ofreceremos un interfaz en la mima línea que se dispone ALUT para archivos WAVE. La Figura 2 resume, de forma gráfica, los cambios que proponemos: mantener el flujo de trabajo que proporciona ALUT, creando una nueva función que haga transparente el uso del formato MP3. En este caso, el uso de la librería *libMAD* permite obtener el audio en PCM para asignárselo a un *buffer* de OpenAL.

4 Desarrollo

Para llevar a cabo el ejemplo que se muestra en este artículo, se ha partido del ejemplo minimad.c [13] de la distribución de la libmad y de los ejemplos de uso de OpenAL ([3] y [6]). Mantendremos los comentarios de los diferentes ficheros de código en que nos hemos inspirado por deferencia con sus autores y por la calidad de los mismos.

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

```
$ sudo apt-get install libmad-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? seguimos adelante.

De esta forma podremos compilar y ejecutar el ejemplo que vamos a mostrar con las órdenes:

```
$ gcc playerMAD_OpenAL.c -o playerMAD_OpenAL  -lalut $(pkg-config mad openal --cflags –libs)

$ playerMAD_OpenAL fichero.mp3
```

Todavía no hemos visto, pero si quiere probarlo primero el lector habrá de copiar en un fichero con el nombre indicado el código de los Listado 1 al Listado 4. ¿Lo tenemos ya? Entonces seguimos con la explicación del código.

Vamos ya a ver un ejemplo de uso de libmad con OpenAL (que denominamos *playerMAD_OpenAL.c*) que está repartido entre el varios listados.

Empezando con el Listado 1 podemos encontrar en él:

- Las definiciones de variables globales, en especial, en la línea 28 está el vector que contendrá las muestras de audio digital al final del proceso de decodificación (*stream*).
- El inicio de la función *madCreateBufferFromFile* que engloba las acciones de *libmad* para decodificar el fichero MP3. En este listado está la parte inicial que mapea el fichero (línea 37) en una zona de memoria principal para su acceso posterior.

El Listado 2 muestra la parte de decodificación del fichero MP3, esto es la secuencia de pasos que hemos indicado en la Figura 2b:

- En la línea 39, se obtiene el nivel superior (*stream*) con *mad_stream_buffer*.
- Y ya dentro del bucle, líneas 40 a la 51, a los *frames* (if (*mad_frame_decode*) y, de cada uno de ellos, se extraen las muestras de audio digital (*mad_synth_frame*) y de este se escribirán(con *output*) en la estructura de datos donde recopilaremos los valores de 16 bits que necesita OpenAL.
- Entre la línea 54 a la 60 podemos ver cómo se accede a las propiedades del audio contenido en el fichero. En realidad no es necesario esperar hasta este momento, pero como ya hemos comentado, no hay una cabecera global que los contenga, así que es necesario a ver procesado, al menos, un *frame* ya se podrían haber escrito en pantalla. Las divisiones por 1000 es para facilitar la lectura de los datos de *bitrate* (en kbps) y la frecuencia de muestreo (kHz).
- En la línea 63 se genera el objeto de OpenAL que contendrá el audio ya descomprimido del MP3 y, en la línea 66, se asigna con la función *alBufferData*. Tras lo cual es posible liberar ya los recursos relativos al uso del fichero MP3 con libMAD.

El Listado 3 muestra:

- el final de las comprobaciones del buffer creado para OpenAL y el resultado que devuelve la función *madCreateBufferFromFile*.
- En las línea 70 empieza el programa principal, empezando por la inicialización en la línea 83. Y la llama a la recién creada *madCreateBufferFromFile* que acabamos de resalta cómo gestiona el contenido del fichero MP3 con *libmad* y rellena una estructura de datos que puede ser usada desde OpenAL.
- En la línea 95 llega por fin el momento de poder escuchar la reproducción de audio, Y entre las líneas 97 a la 102 tendremos cuenta de que se así sucede y, mientras esperamos, actualizaremos un mensaje con el punto actual de la reproducción (el *offset*) del sonido.
- Al terminar el bucle, liberaremos recursos de OpenAL (líneas 104 a la 106) y terminaremos la aplicación.

Ya, en el Listado 4 se agrupan dos funciones para manipular el sonido en su última etapa de conexión entre libmad y OpenAL. Especialemnte cabe resaltar l función *output* que

- Se encarga de recoger el sonido en PCM del MP3 y rellenar con él una estructura de datos en que las muestras de canal izquierdo y derecho se entremezclan (líneas 130 a la 138) y se escalan a valores de 16 bits (recordemos que *libmad* utiliza 24 bits).
- Como la estructura del MP3 es una secuencia de frames para la que no se conoce su longitud, ha sido necesario guardan un valor de en qué posición (variable *base*, línea 118) de la variable *stream* vamos acumulando el audio digital extraído en cada frame.

5 Conclusiones y cierre

El motor de audio 3D, 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 ampliar solo a ficheros WAVE utilizando una librería genérica como ALUT.

La existencia de tantos formatos de audio que podemos encontrar actualmente hace interesante que el desarrollador pueda seleccionar el uso de bibliotecas especializadas en el conjunto de formatos que vaya a utilizar para poder reducir la sobrecarga de código que puede no utilizarse en una aplicación. En ese sentido, se ha revisado el camino que sigue la información de audio desde ficheros MAD para su inclusión en forma de trabajo habitual de OpenAL y, así, proponer un ejemplo de uso de audio basado en la librería *libMAD*.

Se puede optimizar el código mostrado, haciendo una primera lectura de los *frames*, sin decodificarlos, para acceder a los parámetros del audio y conocer exactamente su longitud. Eso permitiría hacer un consumo de recursos más ajustado y que hemos obviado por brevedad de al exposición.

Recuerde también que hemos concretado en el caso de precarga, no en el *streaming* (más interesante para ficheros de gran tamaño) y que está explicando en el mencionado ejemplo *minimad.c* que acompaña a *libmad*. El lector puede ahora 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 MP3. ¿No es así? Pues no cierre el documento sin haberlo hecho.

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <sys/stat.h>
5. #include <sys/mman.h>
6. #include <unistd.h> // usleep
7. #include <stdint.h> // uint32_t
8. #include <mad.h>
9. #include <assert.h>
10.#include <AL/alut.h>
11.
12.struct mad_stream mad_stream;
13.struct mad_frame mad_frame;
14.struct mad_synth mad_synth;
15.
16.#define LAYER_A_STR(v) (v==3? "Layer III" : (v==2? "Layer II" : "Layer
I"))
17.#define MODE_A_STR(v) (v==0? "SINGLE CHANNEL" : (v==2? "DUAL CHANNEL" :
(v==3? "JOINT STEREO" : "STEREO")))
18.
19.#define TAMANY_STREAM 1152*4*1024
20.static unsigned char stream[TAMANY_STREAM];
21.void output(struct mad_header const *header, struct mad_pcm *pcm);

22.static ALuint madCreateBufferFromFile( char *filename ) {
23.   int resultat, fd, nMostres = 0;
24.   ALenum err, format;   ALuint bufferOpenAL;
25.   uint32_t slen;    struct stat metadata;    FILE *fp;
26.   char *input_stream;
27.   fp = fopen(filename, "r");  if ( !fp ){ return( (ALuint)0 ); }
28.   fd = fileno(fp);
29.   if (fstat(fd, &metadata) >= 0) {
30.         printf("File size %d bytes\n", (int)metadata.st_size);}
31.   else {fclose(fp); return( (ALuint)0 ); }
32.     // Initialize MAD library
33.   mad_stream_init(&mad_stream);
34.   mad_frame_init(&mad_frame);
35.   mad_synth_init(&mad_synth);
36.
37.   input_stream = mmap(0, metadata.st_size, PROT_READ, MAP_SHARED, fd, 0);
...
```

Listado 1: Listado de playerMAD_OpenAL (parte 1).

```
...
38.     printf("Creant stream buffer\n");
39.     mad_stream_buffer(&mad_stream, input_stream, metadata.st_size);
40.     while (1) { // Decode frame from the stream
41.         if (mad_frame_decode(&mad_frame, &mad_stream)) {
42.             if (MAD_RECOVERABLE(mad_stream.error)) {
43.                 printf("-"); fflush(stdout); continue;
44.             } else if (mad_stream.error == MAD_ERROR_BUFLEN) {
45.                 printf("\\"); fflush(stdout); break; //continue;
46.             } else {printf("|"); fflush(stdout); break;             }
47.         }
48.         printf("."); fflush(stdout);
49.         // Synthesize PCM data of frame
50.         mad_synth_frame(&mad_synth, &mad_frame);
51.         output(&mad_frame.header, &mad_synth.pcm);
52.     }
53.      output(&mad_frame.header, &mad_synth.pcm);
54.     printf("\n%s: Layer %s (%d) Modo/Canals %s (%d)  bitrate %lu kbps
Canals %d, frecuencia %3.1f kHz duració %ld (segons)\n",
55.             filename, LAYER_A_STR( mad_frame.header.layer ),
mad_frame.header.layer,
56.             MODE_A_STR( mad_frame.header.mode ), mad_frame.header.mode ,
57.             mad_frame.header.bitrate/1000,  // kps
58.             MAD_NCHANNELS(&mad_frame.header),
59.             (float)(mad_frame.header.samplerate) / 1000.0,  // kHz
60.             mad_frame.header.duration.seconds);
61.
62.     nMostres = 0; bufferOpenAL = 0;
63.     alGenBuffers(1, &bufferOpenAL);
64.     if (mad_synth.pcm.channels == 1) format = AL_FORMAT_MONO16;
65.     else format = AL_FORMAT_STEREO16;
66.     alBufferData(bufferOpenAL, format, stream, sizeof(stream),
mad_synth.pcm.samplerate );
67.     fclose(fp);
68.     // Free MAD structs
69.     mad_synth_finish(&mad_synth);
70.     mad_frame_finish(&mad_frame);
71.     mad_stream_finish(&mad_stream);
...
```

Listado 2: Listado de playerMAD_OpenAL (parte 2)

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

```
...
72.    err = alGetError(); // Check if an error occured, and clean up if so.
73.     if(err != AL_NO_ERROR) {
74.         fprintf(stderr, "OpenAL Error: %s\n", alGetString(err));
75.         if(buffer && alIsBuffer(buffer)) alDeleteBuffers(1, &buffer);
76.         return 0; }
77.    return bufferOpenAL;
78.} // madCreateBufferFromFile
79.int main(int argc, char **argv) {
80.    ALuint source, buffer;   ALfloat offset;   ALenum state;
81.
82.    if (argc != 2) {return 255;    } // Parse command-line arguments
83.    alutInit (&argc, argv);      // Initialize OpenAL
84.    printf("madCreateBufferFromFile en libMAD versió %s\n", mad_version);
85.    buffer = madCreateBufferFromFile(argv[1]);// Load the sound
86.    if (alGetError() != AL_NO_ERROR )
87.        printf("Falla al crearel buffer <-- madCreateBufferFromFile \n");
88.    if( !buffer ) {  alutExit(); //CloseAL();  return( 1 );   }
89.    // Create the source to play the sound with
90.    source = 0;
91.    alGenSources(1, &source);
92.    alSourcei(source, AL_BUFFER, buffer);
93.    if (alGetError() != AL_NO_ERROR )
94.       printf("Failed to setup sound source\n");
95.    alSourcePlay(source);      // Play the sound until it finishes
96.    alSourcef(source, AL_GAIN, 10.0);
97.    do {
98.        usleep(1000);
99.        alGetSourcei(source, AL_SOURCE_STATE, &state);
100.           alGetSourcef(source, AL_SEC_OFFSET, &offset);
101.           printf("\rOffset: %f  ", offset); fflush(stdout);
102.      } while(alGetError() == AL_NO_ERROR && state == AL_PLAYING);
103.      // All done. Delete resources, and close down OpenAL
104.      alDeleteSources(1, &source);
105.      alDeleteBuffers(1, &buffer);
106.      alutExit ();
107.     return EXIT_SUCCESS;
108.   }
...
```

Listado 3: Listado de flac_Openal (parte 3)

```
...
109.  // Some helper functions, to be cleaned up in the future
110.  int scale(mad_fixed_t sample) {
111.       sample += (1L << (MAD_F_FRACBITS - 16));      // round
112.       // clip
113.       if (sample >= MAD_F_ONE)       sample = MAD_F_ONE - 1;
114.       else if (sample < -MAD_F_ONE) sample = -MAD_F_ONE;
115.       return sample >> (MAD_F_FRACBITS + 1 - 16); // quantize
116.  }
117.
118.  static long int base = 0;
119.  void output(struct mad_header const *header, struct mad_pcm *pcm) {
120.      register int nsamples = pcm->length;
121.      mad_fixed_t const *left_ch = pcm->samples[0],
122.                        *right_ch = pcm->samples[1];
123.      /* 1152 because that's what mad has as a max; *4 because
124.    there are 4 distinct bytes per sample (in 2 channel case) */
125.       static char spinner[] = "|/-\\";
126.      if (pcm->channels == 2) {
127.          while (nsamples--) {
128.              signed int sample;
129.              sample = scale(*left_ch++);
130.              stream[base + ((pcm->length-nsamples)*4)] =
131.                                                   ((sample >> 0) & 0xff);
132.              stream[base + ((pcm->length-nsamples)*4 +1)] =
133.                                                    ((sample >> 8) & 0xff);
134.              sample = scale(*right_ch++);
135.              stream[base + ((pcm->length-nsamples)*4+2)] =
136.                                                    ((sample >> 0) & 0xff);
137.              stream[base + ((pcm->length-nsamples)*4 +3)] =
138.                                                    ((sample >> 8) & 0xff);
139.          }
140.      } // Sería análogo para el caso de monofónicos: 1 canl).
141.      base += 1024*4;
142.  }
```

Listado 4: Listado de flac_Openal (parte 4)

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] MAD: MPEG Audio Decoder. Disponible en <https://www.underbit.com/products/mad/>.

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

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

[13] libmad - MPEG audio decoder library. Documentación. Disponible en <http://m.baert.free.fr/contrib/docs/libmad/doxy/html/index.html>.