UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Extendiendo OpenAL con OGG Vorbis

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 es [1] un motor de audio 3D cuyo principal objetivo es situar fuentes de sonido en una escena sonora tridimensional, de modo que el oyente (el usuario) se vea envuelto por esa espacializacion del sonido a su alrededor. OpenAL no se encargará de cargar audio de un fichero en disco, sino de enlazarlo a una fuente de sonido. cuando ya está cargado en memoria y sin compresión. Esta ha sido una decisión de diseño ya presente en la primera versión debida a *Loki* [2] y que se mantiene en la actualidad [3].

Para poder ampliar el conjunto de ficheros de los que OpenAL puede **importar el audio** se puede recurrir a librerías específicas. De manera que se puede ampliar la operativa de OpenAL ajustándola a las necesidades y limitando el peso, de este componente en una aplicación, a solo los formatos que se sepa se van a utilizar.

Se verá en este artículo el uso del formato de audio OGG Vorbis [4] y de las funciones de la librería *libvorbis* para determinar las propiedades del audio contenido en un fichero, descomprimir los datos que este contiene y llevar a memoria el audio en un formato que sea compatible con las estructuras de OpenAL.

2 Objetivos

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. No es el objetivo de este documento dar una solución global para todos, sino mostrar cómo incorporar uno y que sirva esta discusión para otros casos similares.

A partir del estudio de los ejemplos que se abordan, el lector será capaz de:

- Explorar los **formatos de audio soportados** por la especificación del motor de audio 3D **OpenAL**.
- Explorar el conjunto de funciones de la biblioteca ***libvorbis***.
- Explorar **un ejemplo de código** para experimentar con la solución propuesta.
- Proponer otras posibles extensiones de OpenAL con futuros formatos de audio.

Para empezar, se introducirá qué es el audio digital, cómo lo gestiona un motor como OpenAL y cómo es el interfaz usado de *libvorbis*. Después se pasará a hablar de cómo juntarlo todo en la solución desarrollada.

3 Introducción

A lo largo del tiempo, el sonido se ha almacenado en una serie de soportes analógicos como el disco de vinilo o la casete, como muestra la fig. 1. Con la llegada del computador, el sonido se ha de codificar en **"formato digital"** para ser procesado. A partir de esta codificación aparecerán soportes como los discos ópticos y formatos como el CD-Audio y el MP3.

Esta digitalización del audio supone la conversión de la señal analógica en digital, lista así para usarse y almacenarse en un computador. El esquema de codificación binario que se utiliza se denomina PCM (de *Pulse Code Modulation*), en el cual el audio es muestreado en el tiempo a intervalos regulares y la amplitud del mismo codificada respecto a un número fijo de estados.

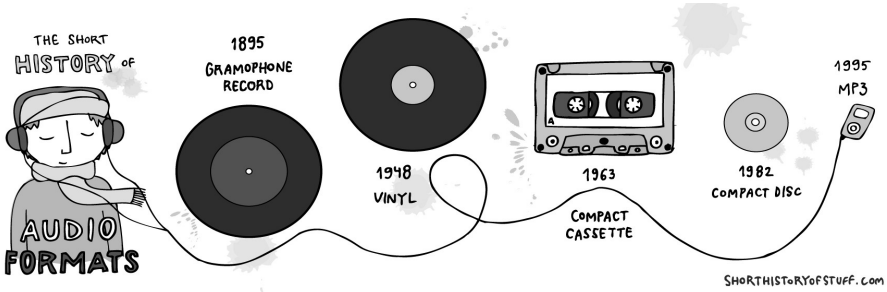

Figura 1: Ejemplos de soportes y formatos de audio. Imagen de [5].

Así, habitualmente, se habla de PCM para referirse a la serie de instantáneas del valor de la señal de sonido que se han adquirido con una **frecuencia de muestreo** dada y han sido convertidas en valores numéricos a partir de la elección de un **tamaño de muestra** (o número de bits que se emplean para indicar su valor actual dentro del rango dinámico de la señal). La fig. 2 muestra la señal original analógica (en rojo) y, sobre ella, sobrepuesta (en azul) los valores tomados en cada instante de muestreo utilizando PCM con 4 bits. Estos dos parámetros determinan la calidad de la digitalización, tanto más fiel a la señal original cuanto mayor son. Aunque también necesitaremos más espacio de memoria para representarla. Es nuestra decisión la elección de los valores más adecuados para cada caso.

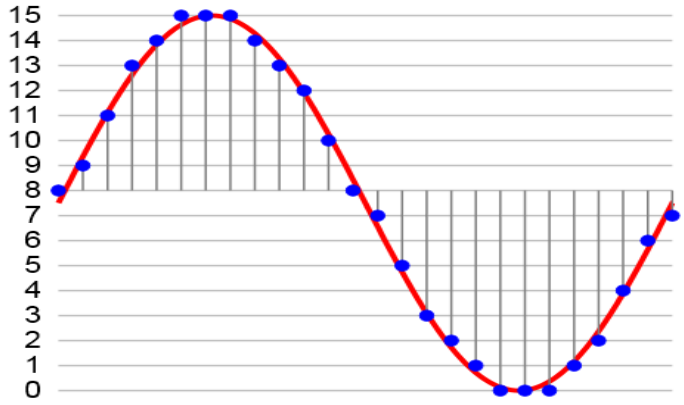

Figura 2: Señal analógica (en rojo) y su correspondiente versión digitalizada (azul). Imagen extraída de <https://en.wikipedia.org/wiki/Pulse-code_modulation>.

Para reproducir el audio, hay que saber exactamente qué valores se han utilizado en la codificación digital para poder interpretar los datos que contiene el fichero. Originalmente los formatos para ficheros de audio asumieron que los parámetros eran fijos; pero con el tiempo y las mejoras en el *hardware*, estos se guardan en la cabecera del propio fichero.

Además, los formatos de audio han ido incorporando mecanismos de compresión [6] para reducir la ocupación de los ficheros. Así, se habla de dos tipos de compresión. La primera se denomina “sin pérdidas” (*lossless*) en tanto que aprovecha la redundancia de información para reescribirla con una menor cantidad de símbolos; lo que hace más complejas las operaciones de escribirla y leerla de fichero, en tanto que es necesaria esta recodificación de audio “plano”

en una codificación que depende del contenido del fichero. La segunda variante es la que se denomina “con pérdidas” (*lossy*) y que elimina cierta información de audio basándose en características del sistema auditivo humano; además de recodificarla, para obtener mayores tasas de compresión. Estas técnicas son todavía más complejas que las anteriores, en tanto en cuanto necesitan tomar más aspectos en consideración para codificar el audio y, por tanto, también para su lectura y decodificación.

En este trabajo se quieren **utilizar ficheros *OGG Vorbis***, cuyo contenido es audio comprimido. Así que no solo es cuestión de leer el contenido del fichero, sino que además hay que descomprimir la información de audio contenido en él. Por ello, **se revisará la operativa de OpenAL y la especificación de *OGG Vorbis*** para encontrar cómo unir ambos.

3.1 OpenAL

La información de audio, en un contexto típico de uso de un motor de audio como es OpenAL (como se muestra en la fig. 3), recorre un camino que va desde la información guardada en un fichero en disco (p. ej.) hasta llegar al sistema de audio. Por lo que va pasando por:

- Los formatos de fichero. Han evolucionado hasta incluir esquemas de codificación que, además de la información (en este caso, de audio), la acompañan de los metadatos necesarios (entre otros y si se ha utilizado) para su decodificación y descompresión.
- Las bibliotecas de formatos. Son el nivel encargado de la interfaz entre los formatos de ficheros y el formato PCM que utiliza internamente OpenAL. De esta manera, el núcleo de OpenAL es pequeño y está totalmente enfocado a las operaciones de procesado de audio.
  Las funciones para ofrecer un mínimo soporte en las tareas de acceso al contenido de ficheros y mantener la portabilidad entre plataformas se agruparon con este propósito en la *OpenAL Utility Toolkit* (ALUT) [7], Este complemento de Openal proporciona la capacidad básica para generar señales básicas e importar sonido desde archivos WAVE[1]. No está pensado hacer crecer estas operaciones para dar cabida a más.
- EL motor. Se encarga primero de traer (a la memoria del sistema o de la tarjeta de sonido) el audio en “bruto” (*raw*), sin formato, esto es con unos valores ya establecidos de frecuencia de muestreo y cuantización. Después construye la *renderización* del sonido, teniendo en cuenta la distribución espacial de las fuentes de audio alrededor del oyente de la escena. Por último, aplica otras características ambientales del sonido como la atenuación por la distancia o el efecto *Doppler*. OpenAL puede funcionar enteramente por *software* o aprovechar la aceleración que le ofrezca el sistema de audio disponible.
- El sistema de audio es el *hardware* encargado de la captura y reproducción del sonido. Puede estar en una tarjeta de sonido o integrado en la placa base del computador.

El planteamiento de la solución pasa por ver cómo ALUT permite ofrecer a OpenAL el sonido contenido en un fichero de audio en formato WAVE. Para ello se revisará el interfaz para el desarrollo de aplicaciones (*Application Programming Interface* o API) de una y otra, para recalcar cómo se conectan. Utilizaremos esa base para plantear la extensión a otro formato. Solo se destacarán las funciones para enunciar el papel de cada una en la solución. **Se**

1 “Audio File Format Specifications”
<http://www-msp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html>

verá el uso de las funciones del API más detallado en los ejemplos del apartado 4 Ejemplo de implementación de estudio.

La especificación de OpenAL [3] permite asociar audio a sus objetos a través de la función alBufferData y así describe la función:

```
void alBufferData (ALuint bufferName, ALenum format,
                   const ALvoid *data, ALsizei size, ALsizei frequency);
```

como la encargada de copiar en un *bufferName* los datos de audio en codificados en PCM que son apuntados por *data* y que ocupan un número *size* de bytes. Describiendo si se refieren a 1 o 2 canales y a 8 o 16 bits de tamaño de muestra en *format*; así como con un valor de frecuencia de muestreo como se se indica en *frequency*.

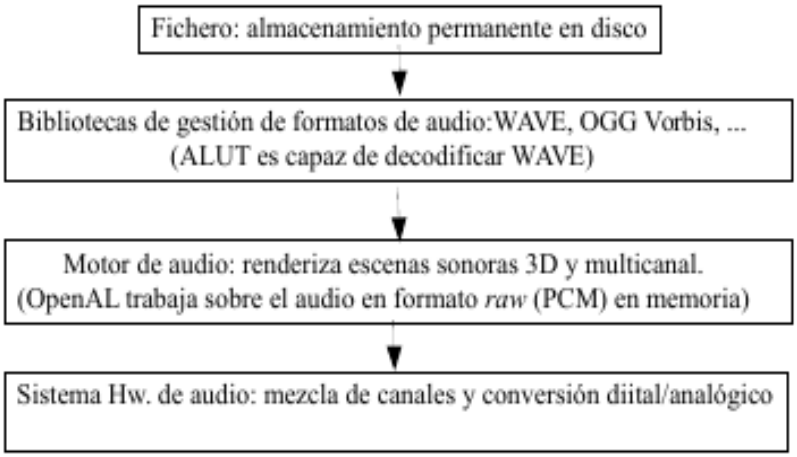

Figura 3: Esquema de la ruta que sigue la información de audio desde fichero hasta el subsistema de audio.

Por su parte, ALUT ofrece [7] funciones para crear e inicializar esos *buffers*, a partir de sonido pregrabado (alutCreateBufferHelloWorld), de la generación de sonidos básicos (alutCreateBufferWaveform) o de la lectura de ficheros en disco (alutCreateBufferFromFile). Esta última es la más similar a nuestro objetivo, permite leer ficheros de los tipos audio/basic (como el au o snd), audio/x-raw (ficheros raw) y audio/x-wav (ficheros WAVE) y tiene la forma:

```
ALuint alutCreateBufferFromFile (const char *filename);
```

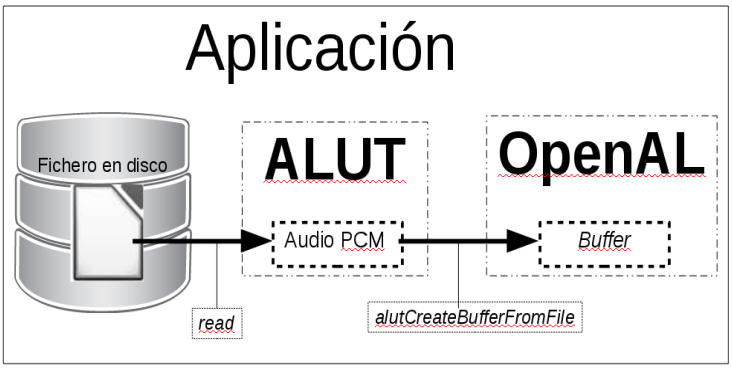

Figura 4: Diagrama de flujo de la carga de un buffer de OpenAL con ALUT.

Todos los casos se basan en reservar memoria y rellenarla con datos de sonido a partir de una de esas tres formas de generarlo. Y, en particular, esta función, como muestra la fig. 4, lee de fichero con la habitual llamada al sistema *read* y asigna la información de audio leída (en PCM) a un *buffer* de OpenAL.

3.2 El interfaz del formato OGG Vorbis

Para obtener una solución como la que proporciona ALUT, es necesario leer un fichero Vorbis, llevarlo a memoria y asignar, a una fuente, el sonido leído con alBufferData. La especificación de Vorbis da una solución para este problema.

Ogg Vorbis es[2] un formato de audio **comprimido** que aparece en el año 2000 y que obtiene resultados en la línea de lo que MPEG-4 (AAC) ofrece. Tiene un especial hincapié en su naturaleza abierta y neutral (sin propietario y sin restricciones debidas a patentes). La biblioteca *libvorbis*[3] es la implementación de referencia de *Vorbis* y constituye un API que da soporte a este formato.

En nuestro caso se va a utilizar *libvorbisfile* [8], ya que no tiene especial interés el acceso a bajo nivel de los datos contenidos en el fichero. Por ello, esta es la mejor solución en tanto en cuanto que, de forma trasparente a la aplicación del usuario, tendrá cuenta de la demultiplexación, descompresión, etc. de este complejo formato. Para ello, *libvorbisfile* se estructura como una capa sobre los servicios que ofrecen *libogg* (que se ocupa de la gestión del formato de fichero) y *libvorbis* (que se encarga de las tarea de descompresión) y que se estructura en torno a dos funciones:

- La inicialización de estructuras de datos y comprobaciones de formato con ov_open. Si el fichero tiene el formato *OGG Vorbis*, esta función obtiene la descripción completa del archivo en una estructura OggVorbisFile.
  De esta nos interesan especialmente las propiedades del audio al descomprimirlo, contenidas en el elemento vorbis_info y que son el número de canales y la frecuencia de muestreo.
  No está guardado el tamaño de muestra utilizado, puesto que Vorbis utiliza valores reales para codificar las muestras de sonido. Se habrá de utilizar la función ov_pcm_total para averiguar el número de muestras que han sido recogidas y que son obtenidas al leer el fichero.
- La lectura y descompresión del flujo de audio con ov_read. Esta función nos devolverá hasta un cierto número de bytes de audio descomprimido en PCM que se especifique. Para ello, se leerá cuanto sea necesario del fichero, no tiene que ser necesariamente el fichero por completo. Además, se puede pedir que conviertan las muestras de audio de los valores reales con que son guardadas en el fichero a valores enteros de, p. ej. 2 bytes (16 bits), como valor más habitual por ofrecer un buen ratio entre tamaño de la muestra y precisión del valor que representa.

4 Ejemplo de implementación de estudio

Siguiendo el planteamiento propuesto en el apartado anterior podemos, véase fig. 5, concretar los pasos para incorporar a una aplicación, que utilice OpenAL, el uso de ficheros OGG Vorbis para cargar los sonidos a utilizar. Y que son: pedir al sistema operativo un descriptor de un fichero (con *fopen*), leer su contenido

2 La información completa sobre OGG Vorbis se puede encontrar en el sitio web de *Xiph Org Foundation* <https://xiph.org/vorbis/>.

3 EL API de *libvorbis* está disponible en <https://xiph.org/vorbis/doc/libvorbis/overview.html>.

(validándolo y descomprimiéndolo con *ov_open* y *ov_read*) y asignar el audio, en PCM, a un *buffer* de OpenAL (con *alBufferData*).

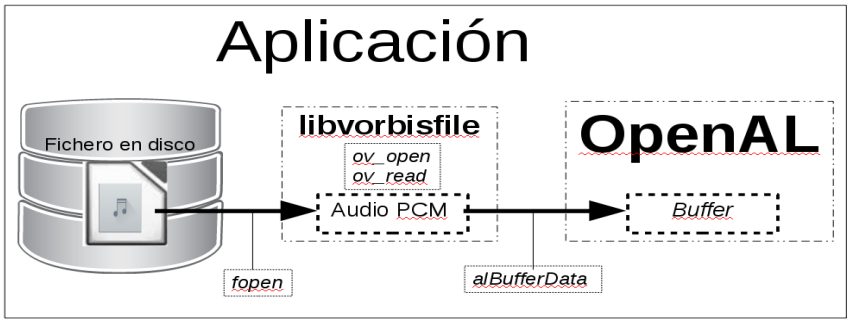

Figura 5: Propuesta de solución para cargar un fichero OGG Vorbis en OpenAL.

Para poder utilizar esas funciones es necesario instalar el soporte para el desarrollo en un determinado lenguaje. Así, p. ej. en la plataforma GNU/Linux para utilizar el API de OpenAL, el de ALUT y el de *vorbisfile* de *OGG Vorbis*, Se puede instalar el soporte de desarrollo de aplicaciones en C para ellos con la orden

```
$ sudo apt-get install freealut libvorbis-dev
```

Obteniendo, en el momento de escribir este documento, una versión 1.1.0 de ALUT y 1.3.2 de *vorbisfile*.

El ejemplo que se muestra aquí es una versión del de Yuen [9]: hay "..." en algunos puntos donde se ha omitido el código, se han quitado las comprobaciones de errores para reducir su extensión y se ha ampliado la información que proporciona con el cálculo del espacio requerido para leer en una sola operación el contenido del fichero y entregarlo como lo hace ALUT. Se han mantenido los comentarios originales donde no se ha variado el código. Se ha reescrito el código C++ en C, para tener "a vista" todos los pasos.

```
#include <stdio.h>            // Reescrito el código utilizando solo C y no C++
#include <stdlib.h>                  // malloc
...
char *errorString(int code) {
...
} // Fin de errorString

long LoadOGG(char *fileName, char **buffer, ALenum *format, ALsizei *freq) {
  int bitStream;   FILE *f;   char *p;
  //Next, we declare some variables that the Ogg Vorbis SDK uses.
  vorbis_info *pInfo;   OggVorbis_File oggFile;   int result;
  // Para contabilizar las operaciones de carga del fichero de audio
  long nMostres, nFramesLlegits; bytesLlegits, tamanyAudioEnBytes, tamanyAudioEnBytesLlegit;
   // Open for binary reading.   Up to this point, things should look very familiar.
  // The function simply uses the fopen() function to open the given file for binary reading.
  f = fopen(fileName, "rb");
  if (f == NULL)  { printf("Could not open Ogg file.\n");    exit( 1 );  }
```

```
  // Opening file for decoding. Then comes the act of passing control over to the SDK.
  // Note that there is no need to call fclose() anymore once this is done.
  if ((result = ov_open(f, &oggFile, NULL, 0)) < 0 )
  { fclose(f);  printf("Could not open Ogg stream: %s\n", errorString(result)); exit( 2 );   }
…
  pInfo = ov_info(&oggFile, -1);                    // Get some information about the OGG file
  ...
  tamanyAudioEnBytes = ((long)pInfo->channels * 2 * (long)ov_pcm_total(&oggFile,-1));
  printf("Reservando %ld bytes, %f timeTotal (en segons) \n",
          tamanyAudioEnBytes, ov_time_total(&oggFile,-1) ); fflush( stdout );
  *buffer = (char *)malloc( tamanyAudioEnBytes );
  ...
  // Decoding the data. Especificat en <https://xiph.org/vorbis/doc/vorbisfile/decoding.html>.
  // Now we are ready to decode the OGG file and put the raw audio data into the buffer.
  …
  // La versión original utilizaba instrucciones de C++ para la gestión de memoria. Ahora, se hace en C
  // y con una instrucción, a partir del cálculo previo de bytes que va ocupar la información descomprimida
     bytesLlegits = ov_read(          &oggFile, *buffer,  tamanyAudioEnBytes,
                                      0, 2, 1, &bitStream);          // 2 bytes por muestra, sin signo
       ...
   /* Clean up. Now all the audio data has been decoded and stuffed into the buffer.  We can release the
file resources (resource leaks are bad!). Note that there is no need to call fclose(). It is already done for us.
Neat.*/
   ov_clear(&oggFile);
   return( tamanyAudioEnBytesLlegit );
} // Fin de  LoadOGG


int main(int argc, char *argv[]) {
  ...
  /* Decoding OGG files. At this point, the system is all ready to go. The one thing that is missing is the
actual sound data! OK, let's write a function that can load OGG files into memory.*/
  tamanyBuffer = LoadOGG(argv[1], &bufferData, &format, &freq);
  ...
   /* Playing the sound. It is now time to get back to our main(). The next step is to upload the raw audio
data to the OpenAL sound buffer and attach the buffer to the source. */
  alBufferData(bufferID, (int)format, (char *)bufferData, tamanyBuffer, (int)freq);   error = alGetError();
   ...
  alSourcei(sourceID, AL_BUFFER, bufferID);              // Attach sound buffer to source
  alSource3f(sourceID, AL_POSITION, 0.0f, 0.0f, 0.0f);
  // Finally, play the sound!!!
  alSourcePlay(sourceID);   i = 0;
  ...
} // end of main
```

Asumiendo que el código está en un fichero de nombre SimpleOGG.c, para compilar el ejemplo se puede utilizar la línea de órdenes:

```
$ gcc SimpleOGG.c -o SimpleOGG -lalut -lopenal -lvorbisfile
-lvorbis -logg -lm
```

y para ejecutarlo hay que darle una ruta hasta un fichero OGG Vorbis, p. ej.

```
$ SimpleOGG drHouse.ogg
```

5 Conclusión

El motor de audio 3D, OpenAL, parte de una especificación reducida para conseguir un impacto mínimo en el consumo de recursos de una aplicación. Se puede ampliar la gestión de ficheros utilizando ALUT. También se pueden utilizar otras bibliotecas genéricas como ALURE o SDL.

La aparición de nuevos formatos hace necesario el uso de bibliotecas especializadas en esos formatos. En ese sentido, se ha revisado el camino que sigue la información de audio desde un fichero hasta los componentes de OpenAL, para proponer un ejemplo de uso del formato de audio comprimido con OGG Vorbis y a través del API de *libvorbis*.

El resultado es bueno, solo que la ocupación de memoria es importante cuando la duración del archivo crece. Aproximadamente a razón de 10 MB por minuto de grabación, si se utilizan 44100 muestras por segundo, 16 bits de tamaño de muestra y dos canales. Por ello la solución propuesta utiliza un bucle de lectura en "trozos" que es fácilmente adaptable al uso de *streaming* de OpenAL. **¿Por qué no lo comprueba?**

6 Bibliografía

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

[2] Loki Software. (2000). oalspecs-annote.pdf (versió 1.0.1, 61 pàgs. Agost 2000).

[3] Creative. Tech. Ltd. (2005).OpenAL 1.1 Specification.pdf (62 pàgs. June, 2005). Disponible en <http://www.openal.org/documentation/openal-1.1-specification.pdf>.

[4] Xiph.org. Vorbis audio compresión. Disponible en <https://xiph.org/vorbis/>.

[5] Lang, Z. (2013). The short history of audio formats. Disponible en <https://shorthistoryofstuff.wordpress.com/2013/06/28/the-short-history-of-audio-formats/ >.

[6] RHA. (2014). *An Introduction To High Resolution Audio Formats - Part 1*. Disponible en <https://www.rha-audio.com/ca/blog/92028/an-introduction-to-high-resolution-audio-formats-part-1>.

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

[8] Xiph.Org. (2010). Vorbisfile Documentation. Disponible en <https://xiph.org/vorbis/doc/vorbisfile/index.html >.

[9] A. Yuen. (2003). Introduction to Ogg Vorbis. General and Gameplay Programming. Disponible en <https://www.gamedev.net/articles/programming/general-and-gameplay-programming/introduction-to-ogg-vorbis-r2031/>.