UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ALURE: interfaz de alto nivel para OpenAL. Servicios relacionados con la reproducción 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 es un estándar para el desarrollo [4] de escenas de audio 3D, formado [1] por un conjunto de abstracciones (dispositivo, contexto, fuentes, *buffers* y oyente). OpenAL proporciona propiedades de posicionamiento espacial para las fuentes de sonido, funciones para reproducción de audio, atenuación con la distancia, efecto Doppler, etc. Tiene un mecanismo interno para facilitar la tarea de reproducción en continuo (*streaming*), uso de audio multicanal, así como la grabación de audio con micrófono. También incluye un mecanismo para incorporar extensiones y permitir a las aplicaciones que pregunten por si están o no instaladas en tiempo de ejecución, así la extensión *Effects Extension* (EFX) [5] proporciona parámetros ambientales (características del medio, generalmente el aire), así como las que introducen los objetos y sus materiales (oclusiones y reverberaciones).

La Figura 1 muestra el bloque de OpenAL 1.1 (la última versión publicada de este estándar) con las capas que lo forman, esto es la arquitectura interna de OpenAL. El nivel más bajo es el que forma el núcleo de operaciones básicas que se conoce como *Audio Layer* (AL) y que se encarga de gestionar los *buffers* (que contienen el sonido en memoria sin modificar), las fuentes (que son los objetos que asociados a un *buffer* le confieren propiedades de posición y velocidad en la escena) y el oyente (que es el objeto que representa la posición y velocidad del usuario). Sobre este, el *Audio Library Context* (ALC) que implementa las abstracciones de dispositivo (acceso al hardware de audio) y contexto (la definición de los límites y características de la escena sonora). Por encima de estos dos niveles, *The OpenAL Utility Toolkit* [2] (ALUT) proporciona capacidades de importación de ficheros, generación de señales básicas y alguna otra función auxiliar.

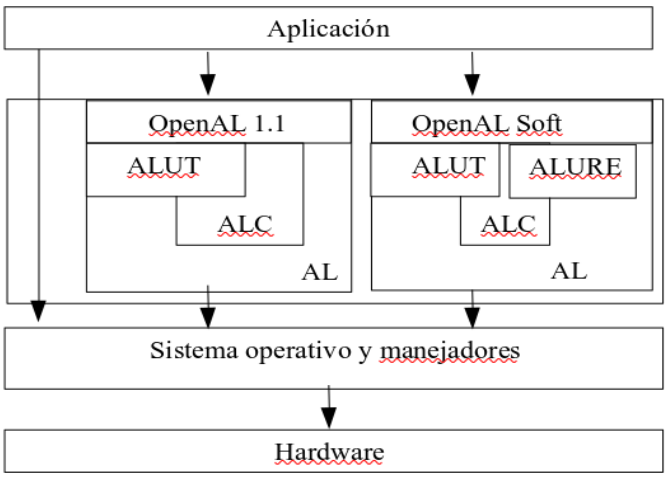

Figura 1: Correspondencia de los niveles del estándar OpenAL con la implementación de OpenAL Soft.

El otro bloque de la Figura 1, OpenAL Soft, es [6] la implementación (totalmente software) del API del estándar de OpenAL para audio 3D, realizado por Chris Robinson. Esta implementación da soporte a toda la jerarquía de niveles de OpenAL y ha ido actualizando sus servicios. Además, añade un bloque, al mismo nivel que ALUT, que se denomina *AL Utilities REtooled* (ALURE) y con el que el autor de *OpenAL Soft* puede tomar decisiones

respecto a su evolución. *ALURE* [7] expande las funcionalidades de OpenAL en temas relacionados con la carga de ficheros, la decodificación de sus contenido o la gestión de *streaming*.

La Figura 1 muestra el bloque de *OpenAL Soft* al lado del de OpenAL para que podamos hacernos la composición de lugar: *OpenAL Soft* es, seguramente, la versión que tendremos instalada, ¿por qué no utilizamos las aportaciones de esta implementación que si ha tenido un progreso, unas actualizaciones y unas aportaciones, que podemos utilizar en nuestros desarrollos?

En este artículo, vamos a presentar las opciones que tenemos para la reproducción de formatos de ficheros de audio; especialmente, si queremos aprovechar que en la instalación que podemos tener actualmente de OpenAL (en nuestros equipos) viene ya, seguramente, preinstalado ALURE, que es una mejora sustancial de ALUT.

2 Objetivos

Este artículo está enfocado a explorar los servicios que proporciona ALUT como API de alto nivel, por lo que a partir del estudio del ejemplo de código que se aborda en este documento, el lector será capaz de:

- Identificar en el código las funciones que pertenecen al nivel de ALUT.
- Identificar en el código las funciones que pertenecen a los niveles ALURE.
- Instalar y compilar una aplicación que hace uso de la librería OpenAL, con el API DE ALUT o de ALURE.
- Identificar las funciones que ofrecen el API DE ALUT o de ALURE para cargar y reproducir audio desde un fichero.

No es objetivo de ese artículo explicar el funcionamiento de audio 3D de OpenAL, Queremos explorar el papel de ALURE como posible (y recomendable) sustituto de ALUT. Tampoco lo es ser un mero listado exhaustivo de operaciones, sino mostrar las más habituales y proporcionar referencias para entender las operaciones y descubrir otras similares.

3 Introducción

OpenAL Soft proporciona [8] una implementación software de OpenAL, multiplataforma y publicada bajo licencia LGPL. Nace como un *fork* de la versión disponible del repositorio de SVN de openal.org que era de código abierto y estaba disponible solo para entorno Windows.

Sobre OpenAL Soft, ALURE [7] ofrece un interfaz en lenguaje C que se basa en OpenAL y lo expande en temas relacionados con la carga de ficheros, la decodificación de sus contenidos y la gestión de *streaming*. Desde la versión 1.1 de ALURE, está publicado bajo licencia X11/MIT[1], lo que permite su uso en aplicaciones de código abierto o cerrado, *freeware* o comercial. El propósito de esta librería (como lo es el de ALUT al que equivale) es proporcionar funcionalidades que se usen de forma habitual y que varían en función de

[1] Puede consultarse en <https://en.wikipedia.org/wiki/MIT_License> al respecto.

la plataforma para la que se realiza el desarrollo, como por ejemplo, la carga de ficheros de sonido en los *buffers* de OpenAL o la gestión de la cola de *buffers* para ofrecer una reproducción en continuo (*streaming*).

De entre todas las opciones que ofrece *ALURE*, en este artículo, nos centramos en cómo proporciona funciones para el acceso a los ficheros de audio en disco a una aplicación. Con esta idea, ALURE incluye [9] un lector y decodificador nativo básico de WAVE y AIFF, así como también se apoya en librerías externas para la lectura y decodificación de otros formatos populares como: extensiones de WAVE con *libSndFile*, Ogg Vorbis con *VorbisFile* y *OpusFile*, Ogg FLAC con *dr_flac*, MP3 con *minimp3. Además, incluye la opción de enlazar con otras librerías externas en tiempo de ejecución lo que permite seguir extendiendo sus capacidades sin tener que modificar el código de ALURE y la opción de deshabilitar estas librerías en tiempo de compilación*.

Acompañaremos esta presentación de sendos ejemplos de código que nos permitirán identificar las operaciones del API de ALUT y el de ALURE en el código y lo complementaremos con ejemplos de órdenes para la instalación y compilación en plataforma Linux, que es fácilmente portable (también en línea de órdenes) en otros sistemas operativos.

Si está preparado, estimado lector, empecemos con la instalación que es breve, con las órdenes:

```
$ sudo apt install libalure-dev
$ sudo apt install libopenal-dev libvorbis-dev libopusfile-dev libsndfile1-dev
```

y comprobaremos su instalación, preguntado por la versión instalada o las dependencias para compilar un programa con esta librería con las siguientes órdenes y el resultado que pueden/deben mostrar:

```
$ pkg-config alure --modversion
1.2
$ pkg-config alure --cflags
-I/usr/include/AL
$ pkg-config alure --libs
-lalure -lopenal
```

4 Desarrollo

Desde la especificación del estándar OpenAL [3] la secuencia de inicialización de OpenAL consiste en:

- Abrir un dispositivo, esto es acceder al hardware disponible e inicializarlo.
- Inicializar un contexto, esto es asociar una escena sonora con un oyente a ese dispositivo.
- Crear los *buffers* que contendrán el sonido y rellenarlos con muestras de sonido.

- Crear la/s fuente/s para posicionar el sonido y asociar el/los *buffer*/s que correspondan a cada una.
- Reproducir el audio .

4.1 API de ALUT para la carga y reproducción de ficheros de audio

El API o conjunto de operaciones que ofrece ALUT [3], en su apartado de *Sound Sample File Loading* nos habla de diez operaciones que podemos agrupar en

- De generación de buffers. Que permiten asignar, a un nuevo *buffer*, el contenido de un fichero con:
  - *alutCreateBufferFromFile. Que carga los contenidos de un fichero en un buffer nuevo de OpenAL.*
  - *alutCreateBufferFromFileImage. Hace lo propio, pero lo deja en un área de memoria central.*
  - *alutCreateBufferHelloWorld. Carga un buffer con la grabación de S. Baker diciendo "Hello, World!" y que está codificada en el propio código fuente de ALUT.*
  - *alutCreateBufferWaveform. Carga un buffer con una señal sintetizada en tiempo de ejecución. Puede generar ondas senoidales, cuadradas, triangular, diente de sierra y ruido blanco.*
- De carga en memoria principal. De manera análoga al grupo anterior, pero ahora teniendo como destino de la operación una posición de memoria, contamos con *alutLoadMemoryFromFile*, *alutLoadMemoryFromFileImage*, *alutLoadMemoryHelloWorld* y *alutLoadMemoryWaveform*.
- *alutGetMIMETypes*, nos indicará los tipos de ficheros que se soportan por las operaciones de lectura desde fichero.
- Y obsoletas (*deprecated WAV loaders*), que nos dice que se mantienen por compatibilidad hacia atrás con las versiones de ALUT 0.x.x. Y es que, en algunas plataformas como macOS, siguen siendo las únicas funciones que están disponibles, porque la implementación que se utiliza no se ha actualizado.

  En este caso nos encontraremos con alutLoadWAVFile, alutLoadWAVMemory y *alutUnloadWAV*.

El Listado 1 nos muestra un ejemplo de este grupo de operaciones que ofrece ALUT. Hemos recalcado cuándo aparecen para facilitar su identificación al lector. Le sugerimos que pruebe a ejecutar el código mostrado y, así, la experimentación le llevará a entender el desarrollo mostrado. Para compilarlo habrá de ejecutar la línea:

```
$ gcc openal_fitxer_alut.c -o openal_fitxer_alut  $(pkg-config freealut openal --cflags –libs)
```

Y para ejecutarlo necesitamos tener un fichero WAVE (supongamos que se llama "unFichero.wav"):

$ openal_fitxer_alut unFichero.wav

```
1.  #include <stdio.h> // printf
2.  #include <stdlib.h>
3.  #include <AL/alut.h>
4.
5.  int main (int argc, char **argv) {
6.    ALuint buffer, fuente;
7.    int error;
8.    ALint sourceState;
9.
10.   if (argc == 1)  {
11.    printf("Falta un parámetro\n: %s nombreFicheroWAVE\n", argv[0]);
12.    return( -1 );
13.   }
14.   else {
15.    alutInit( &argc, argv );
16.
17.    printf("Reproduint %s ...\n", argv[1]);
18.    buffer = alutCreateBufferFromFile( argv[1] );
19.    error = alGetError();
20.    if (error) printf("%s\n", alutGetErrorString(error));
21.
22.    alGenSources (1, &fuente);
23.    alSourcei(fuente, AL_BUFFER, buffer);
24.
25.    alSourcePlay (fuente);
26.    alGetSourcei( fuente, AL_SOURCE_STATE, &sourceState);
27.    while (sourceState == AL_PLAYING) {
28.     alutSleep( 0.25 );
29.     alGetSourcei( fuente, AL_SOURCE_STATE, &sourceState);
30.    }
31.    alDeleteSources(1, &fuente);
32.    alDeleteBuffers(1, &buffer);
33.    alutExit ();
34.    return EXIT_SUCCESS;
35.   } // if (argc == 1)
36. }// main
```

Listado 1: Ejemplo de reproducción de un fichero de audio WAVE con ALUT: openal_fitxer_alut.c.

El Listado 1 nos muestra el contenido de *openal_fitxer_alut.c*, es decir, la secuencia de acciones que pueden dar lugar a que se lea y reproduzca el fichero que se le pasa como parámetro al programa. Tras las inicializaciones de las líneas 10 a la 14, en la línea 15 encontramos la función *alutInit* que prepara el estado de la librería e inicializa el dispositivo de audio por defecto (el que le indica el sistema operativo).

Junto a un mensaje (línea 17), para que el usuario sepa lo que está haciendo el programa, la línea 18 contiene a la función protagonista: *alutCreateBufferFromFile*. La cual abrirá el fichero indicado, lo leerá y decodificará; y ya, con los datos de audio, creará un *buffer* y se los asignará. Todo el contenido del fichero está cargado en memoria, lo que puede ser un peso importante en algunos casos.

Tras comprobar si ha habido algún error y mostrarlo (líneas 19 a la 20) con *alutGetErrorString*, si es necesario, se pasa a crear el objeto fuente (línea 22), al que asociamos el *buffer* recién creado (línea 23) y ya lo podemos hacer sonar (línea 24).

Las líneas 26 a la 30 se ocupan de mantener el programa esperando mientras suena la música, esto es, mientras no han terminado de reproducirse los datos cargados desde fichero.

Entre las líneas 31 a la 34 nos ocupamos de que la aplicación tenga una salida ordenada.

4.2 API de ALURE para la carga y reproducción de ficheros de audio

Ya hemos visto lo que hace y cómo ALUT, veamos ahora cómo lo llevaríamos a cabo con ALURE. De la documentación de ALURE[2] podemos ver que el módulo de *File Loading* ofrece funciones para:

- Cargar audio desde archivo
  - *Con alureCreateBufferFromFile, carga los contenidos del fichero en un nuevo buffer de OpenAL*
  - Mientras que con *alureBufferDataFromFile*, hace lo propio pero con un *buffer* ya existente. Esta es una buena idea cuando puedes reaprovechar los objetos creados, no lo vamos a ver por brevedad en la exposición.
- Cargar audio desde memoria
  - De manera similar al grupo anterior, ahora, es posible cargar una imagen de un fichero en memoria en un nuevo *buffer* de OpenAL con *alureCreateBufferFromMemory*.
- Y *alureBufferDataFromMemory* hace lo propio pero reescribiendo el contenido de un *buffer* ya existente.

Nos vamos a centrar en la primera del primer grupo, puesto que las otras recogen el resultado desde memoria, no desde un fichero.

La parte de la reproducción del fichero se basa en el ejemplo de ALURE alplay.c[3]. Así que verá los comentarios del fichero original también. El original utiliza todavía la librería *libsndfile* directamente para acceder al contenido de los ficheros, aquí hemos actualizado el código con las nuevas funciones de ALURE, con lo cual se hace más evidente lo transparente que es para el desarrollador reconocer y trabajar con diferentes formatos. Solo hay que esperar a que lean los datos y hacerlos sonar.

El Listado 2 muestra el contenido del fichero *openal_fitxer_alure.c*, que es un ejemplo de código que utiliza ALURE para la carga y reproducción de ficheros de sonido. Tras las inicializaciones de las línea 14 a la 18, encontramos la función *alureInitDevice* (línea 19) que prepara el estado de la librería e inicializa el dispositivo de audio por defecto (el que le indica el sistema operativo).

---

[2] La podemos encontrar en el sistema de archivos local si hemos instalado ALURE y también en <https://kcat.tomasu.net/alure-docs/files/alure-cpp.html>.

[3] Está disponible en el repositorio de Github de ALURE: <https://github.com/kcat/openal-soft/blob/master/examples/alplay.c>.

```c
#include <stdlib.h>
#include <stdio.h>
#include <AL/alure.h>
#include <string.h>
#include <AL/al.h>
#include <AL/alext.h>

int main(int argc, char **argv) {
  ALuint ALURE_major, ALURE_minor;
  ALuint source, buffer;
  ALenum state, error;
  ALfloat offset;

  if (argc == 1)  {
   printf("Falta un parámetro\n: %s nombreFichero\n", argv[0] );
   return( -1 );
  }

  if(!alureInitDevice(NULL, NULL))   {
     fprintf(stderr, "ALURE:: Failed to open OpenAL device: %s\n",
               alureGetErrorString());
     return( -1);
  }

  printf("Reprodueix %s\n", argv[1] );
  buffer = 0;
  alGenBuffers(1, &buffer);
  buffer = alureCreateBufferFromFile( argv[1] );
  alGenSources(1, &source);
  alSourcei(source, AL_BUFFER, (ALint)buffer);
  alSourcePlay(source);
  do {
        alureSleep( 0.1 );
        alGetSourcei(source, AL_SOURCE_STATE, &state);
        /* Get the source offset. */
        alGetSourcef(source, AL_SEC_OFFSET, &offset);
        printf("\rOffset: %f  ", offset);
        fflush(stdout);
  } while(alGetError() == AL_NO_ERROR && state == AL_PLAYING);
  printf("\n");

  /* All done. Delete resources, and close down OpenAL. */
  alDeleteSources(1, &source);
  alDeleteBuffers(1, &buffer);

  if( alureShutdownDevice() == AL_FALSE )  {
       printf(" ALURE:: error al lliberar el dispositiu.\n" );
       return( -5 );
  }

  return 0;
}
```

Listado 2: Ejemplo de reproducción de un fichero de audio WAVE con ALURE: openal_fitxer_alure.c. También se puede utilizar para OGG, FLAC o MP-)

Junto a un mensaje (línea 24), para que el usuario sepa lo que está haciendo el programa, la línea 27 contiene a la función protagonista: *alureCreateBufferFromFile*. La cual abrirá el fichero indicado, lo leerá y decodificará; y ya, con los datos de audio, creará un *buffer* y se los asignará. Recuerde que todo el contenido del fichero se está cargando completamente en memoria, como en el caso de ALUT.

Ahora, líneas 28 a la 30, ya se puede crear el objeto fuente, al que asociamos el *buffer* recién creado y ya lo podemos hacer sonar. Estos tres pasos se hacen con funciones de AL, como en el caso de ALUT. Las líneas 31 a 39 se ocupan de mantener el programa esperando mientras suena la música; además, este ejemplo nos va mostrando cuánto lleva sonando del fichero. Entre las líneas 41 a la 48 nos ocupamos de que la aplicación tenga una salida ordenada.

Como puede comprobar la equivalencia de los dos ejemplos es muy similar, así que el cambio a usar ALURE no es complejo. Lo mejor de este código, y la mayor diferencia con el ejemplo de ALUT, este que ALURE puede manejar otros formatos, lo podemos comprobar recopilando otros ficheros y cambiando la ruta, p. ej., he ido haciendo conversiones con *SoX*[4] y comprobando que ALURE los reproduce. No ha sido necesario cambiar ninguna línea del código … ¡Bien por ALURE!

¿Lo está comprobando? ¿No? Venga, se puede probar el código ejecutándolo con un nombre de fichero como parámetro:

```
$ openal_fitxer_alure fichero.wav
Reprodueix  fichero.wav
Offset: 20.433559
 …
$ openal_fitxer_alure fichero.ogg
Reprodueix  fichero.ogg
Offset: 20.433559
…
```

Y de igual manera con estos otros. De verdad, pruébelo:

```
 $ openal_fitxer_alure fichero.flac
 $ openal_fitxer_alure fichero.mp3
 $ openal_fitxer_alure fichero.aif
```

5 Conclusión y cierre

OpenAL está dividido en tres componentes. ALUT, la de más alto nivel, con el tiempo, es vista como una parte de la que se podría prescindir o que, al menos, precisa ser actualizada. En este artículo hemos explorado su planteamiento y las operaciones que provee. Nuestro objetivo era analizar (con un desarrollo práctico) si es posible utilizar ALURE que viene proporcionado por OpenAL Soft para tal fin, ya que esta es la implementación, de OpenAL, que más probablemente tendrá instalada en su equipo.

[4] La página web de “SoX - Sound eXchange ” está en <http://sox.sourceforge.net/>.

Podemos decir que sí se puede sustituir, en buena parte ,el papel de ALUT con el uso de ALURE y que, a nivel de uso de ficheros como entrada de datos, es mucho mayor el conjunto de formatos de ficheros de audio que podemos utilizar con ALURE. Aunque no hemos utilizado todas las funciones disponibles en ALUT o en ALURE, hemos remarcado las diferencias en los resultados obtenidos por la ejecución de los dos ejemplos de código propuestos.

Espero que no cierre el documento sin haber comprobado antes lo que pueden hacer estos dos ejemplos. Hemos dado las pautas para instalar, compilar y ejecutar el código propuesto. Solo queda que pueda oír sus propios ficheros a través de OpenAL con y sin ALUT ¿No es así? Bueno, el código podrá descargarlo del repositorio creado a tal efecto en *GitHub* en la URL <https://github.com/magusti/OpenAL_examples/OpenAL_ALURE>.

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] Peacock, D, Harrison, P., D'Orta, A., Carpentier, V., Cooper, E. (2006). Effects Extension Guide v 1.1. Disponible en <http://kcat.strangesoft.net/misc-downloads/Effects%20Extension%20Guide.pdf >.

[6] OpenAL Soft – Software 3D Audio. Disponible en <https://openal-soft.org/>.

[7] C. Robinson. ALURE Homepage. Disponible en < https://kcat.tomasu.net/alure.html>.

[8] OpenAL Soft – repositorio en Github. Disponible en <https://github.com/kcat/openal-soft>.

[9] ALURE – repositorio en Github. Disponible en <https://github.com/kcat/alure>.