UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Arquitectura de niveles en OpenAL: AL, ALC y ¿ALUT?

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

Open Audio Library (OpenAL) [1], es un estándar que describe un conjunto de operaciones para la creación de escenas sonoras: esto es, calcula el audio que escucharía el oyente de la escena situado en ciertas coordenadas espaciales, al que le llegan una serie de señales sonoras en función de la localización de estas en el espacio y junto a otros posibles efectos y variaciones de los parámetros que definen el entorno, las características del oyente y las de cada señal de audio que es posible renderizar.

Este motor de audio 3D, cuyo logotipo podemos ver en la Fig. 1a, originalmente desarrollado por *Loki Software*[1] hacia el año 2000 está formado por un conjunto de funciones (en forma de un *Application Programming Interface* o API) multiplataforma que permite portar las aplicaciones, con él desarrolladas, a los diferentes sistemas operativos existentes. Sus capacidades de importación de ficheros (entre otras operaciones) se encuentran en un pequeño componente denominado "The OpenAL Utility Toolkit" [2] (ALUT, véase Fig. 1b) que junto a ALC y AL [3] conforman la arquitectura (o jerarquía) interna de OpenAL.

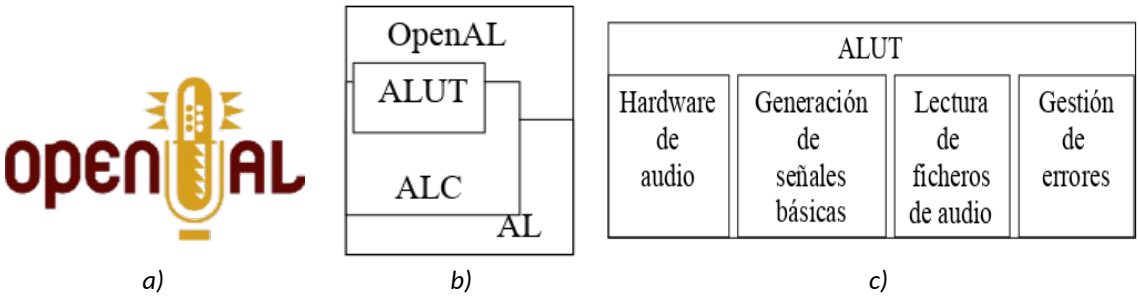


Figura 1: OpenAL: (a) logotipo, (b) niveles de la especificación del estándar (AL, ALC y ALUT) y (c) grupos de operaciones en ALUT .

ALUT [2] ha sido 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, con "la esperanza de que sirva para producir programas de demostración rápidamente y ayude a los nuevos desarrolladores a empezar con OpenAL. Para ello simplifica los tediosos detalles de inicio de una aplicación de audio y permite centrarse, sin distracciones (como cargar el contenido de audio de ficheros de disco), en el uso de OpenAL." [2].

En este artículo veremos los cuatro grupos principales de operaciones que podemos encontrarnos en ALUT (Fig. 1c). Nos vamos a detener a examinar qué hace ALUT y si se puede trabajar con OpenAL sin ALUT. Así que examinaremos ejemplos de código para que el lector pueda experimentar con esta situación y decidir, o reescribir con estas ideas, un ejemplo que pueda tener que desarrollar.

[1] Fue una empresa de desarrollo de videojuegos que liberó este estándar antes de su cierre <https://en.wikipedia.org/wiki/OpenAL>.

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.

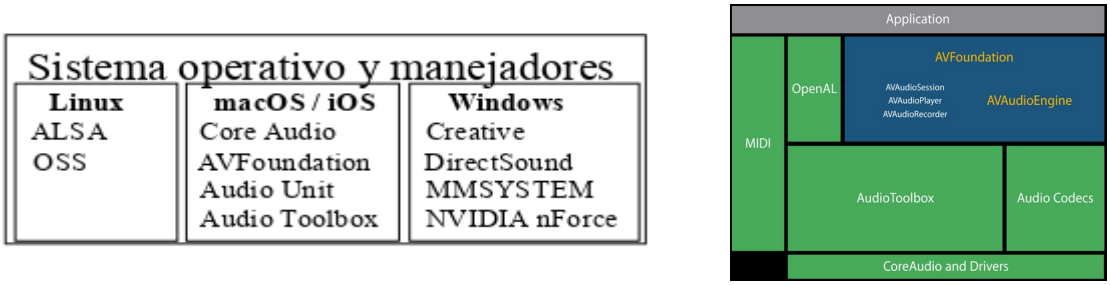
- Identificar en el código las funciones que pertenecen al nivel de ALUT.
- Identificar en el código las funciones que pertenecen a los niveles ALC o AL.
- Instalar y compilar una aplicación que hace uso de la librería OpenAL.

No es objetivo de ese artículo explicar el funcionamiento de audio 3D de OpenAL, queremos explorar en el papel de ALUT, de alto nivel de abstracción, frente a los otros dos componentes (ALC y AL) de más bajo nivel.

3 Introducción

Los desarrolladores de OpenAL [2] decidieron, en un momento dado, que era interesante ese nivel abstracción de ALUT y que debería tener ciertos contenidos, lo que daría lugar a la aparición de la versión 1.0.0 del mismo. Hay que hacer notar que podemos encontrar artículos y ejemplos que fueron desarrollados sobre las versiones anteriores, que llevan como números de versión 0.x.x por lo que podemos encontrar en ellos que usan algunas funciones que están declaradas como obsoletas en la versión más actual: la 1.1.

Para mantener la implementación de OpenAL de modo multiplataforma, se reconoce la plataforma sobre la que se ejecuta para apoyarse en los servicios que ofrece el sistema operativo (SO). Esto implica que ALC y AL implementan el acceso al *hardware* de audio a través de los manejadores o *drivers* soportados por el SO.



a) *b)*

Figura 2: Conexión con el Sistema Operativo o los manejadores existentes.

La Fig. 2a muestra como, en cada plataforma, hay un número de opciones (propias del SO o de los manejadores) entre los que OpenAL puede "escoger", seguramente [5]: OSS en Linux, *DirectSound* en Windows y *Core Audio* en mac OS (y iOS). De hecho, en este último[2], véase la Fig. 2b podemos encontrar repartidos los servicios de audio entre diferentes niveles:

[2] Información extraída del manual de referencia del desarrollador en <<https://developer.apple.com/library/archive/documentation/MusicAudio/Conceptual/CoreAudioOverview/WhatsinCoreAudio/WhatsinCoreAudio.html>.

- *Audio Toolbox* para los servicios a aplicaciones como *streaming*, sonidos de alertas del sistema, reproducción y grabación. Y los servicios de sesión, en el caso de iOS.
- *Audio Unit*, que agrupa a los códecs disponibles.
- *AVFoundation* que es la interfaz para *Objective-C*.
- *Core Audio* como nivel más básico definiendo los tipos de datos y los manejadores (servicios *hardware*).
- Y, para terminar, OpenAL encargado exclusivamente de los servicios de audio posicional y de baja latencia para tiempo real.

En el caso de la componente ALUT, no solo está el acceso al *hardware* de audio, sino que también ha de facilitar el acceso al sistema de archivos, así que todavía tiene más variaciones, por eso se le llama *wrapper* en algunos sitios. Esto ha propiciado que también esté disponible en plataformas móviles como Android, iOS y que lo estuviera para Blackberry y su *Tablet OS*[3]. Con el tiempo y, propiciado por la decisiones de *Creative Lab*, no se han producido cambios en ALUT, pero si que disponemos de una implementación con licencia *GNU Library Public License* (LGPL), denominada *Freealut*[4].

Así que, en lo que respecta a la conexión con el *hardware* y dependiendo de qué sistema operativo se esté utilizando, la implementación inicial de OpenAL (conocida como OpenAL SI[5]) escogía a qué componente del SO puede encomendarle ciertas acciones. Ahora, con el avance de potencia de las CPU, tenemos una implementación totalmente software: OpenAL Soft[6] con licencia GPL.. Además, tiene un complemento, que es *Alure*[7], que forma parte del proyecto *OpenAL Soft*, con licencia *zlib* (que es compatible con GPL, pero que requiere que se documente el código fuente que se modifique) que suele acompañar a la instalación de *OpenAL Soft* y que cubre la funcionalidad del bloque ALUT.

4 Desarrollo

Para profundizar en el qué hace ALUT [2] por nosotros, que es nuestro objetivo principal, vamos a explorar un par de aplicaciones que nos permitan ver cómo se hacen con ALUT y sin él, esto es con la combinación de ALC y AL, algunas cuestiones de cada apartado como:

- Inicializar el *hardware*.
- Síntesis de audio, generando señales de audio básicas.
- Acceder a ficheros.
- Y en lo relativo a la gestión de errores.

[3] Todavía se puede acceder a la implementación para esta plataforma en <https://github.com/blackberry/OpenAL>.

[4] Encontrará más información sobre *Freealut* en <https://github.com/vancegroup/freealut/>.

[5] La *OpenAL Sample Implementation* es la original de Loki. <https://en.wikipedia.org/wiki/OpenAL >.

[6] Puede ampliar esta opción en su página web <https://openal-soft.org/>.

[7] La puede encontrar en <https://github.com/kcat/alure>.

Primero, para poder experimentar con el código que vamos a ver, deberíamos tener algunas dependencias instaladas. Para ello, en Linux (en concreto hemos utilizado Ubuntu 20.04 LTS), podemos utilizar la orden:

```
$ sudo apt-get install openal-dev alut-dev
```

Estoy subiendo los ejemplos que se muestran en este artículo a un repositorio de GitHub[8]. Puede descargarlos de allí y probarlos rápidamente.

4.1 Ejemplo de código con ALUT

El ejemplo de código que se muestra en el Listado 1 es un exponente de las funciones del API de alto nivel de OpenAL/ALUT. Se han remarcado algunas instrucciones para fijar la atención del lector en este ejemplo de código y facilitar la comparación con en el que veremos en el apartado 4.2. Para compilar este ejemplo habrá de utilizar una orden como:

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

En el Listado 1 podemos ver funciones de los cuatro grupos indicados en la Figura 1c:

- De gestión de errores (*alutGetErrorString* en la línea12).
- De gestión del *hardware*, tanto para inicialización (*alutInit* en línea 10) como liberación de recursos (*alutExit* en línea 47).
- Generación de señales básicas, como el *alutCreateBufferHelloWorld* (línea 21) que carga un vector predefinido con esa locución[9] o la generación de una señal senoidal con *alutCreateBufferWaveform* (en la línea 27).
- Lectura de ficheros WAVE con *alutCreateBufferFromFile* (línea 34). En implementaciones como la de *macOS*, se ha de utilizar la función *alutLoadWAVFile* (declarada obsoleta en ALUT 1.1).

Otras funciones como *alutSleep* (línea 32 y 41) permiten esa abstracción, que es la idea original de ALUT de subsumir los detalles diferentes de las plataformas soportadas para facilitar el desarrollo rápido de una prueba de concepto.

Y, lógicamente, hay funciones con el prefijo “al” que corresponde al nivel básico de OpenAL, que cualquier aplicación implementada sobre este estándar suele utilizar. Sólo destacar dos cosas:

- No aparecen funciones con el prefijo “alc”.
- Son instrucciones de una línea de código.

Lo que muestra que ALUT proporciona unas operaciones compuestas por una o más instrucciones, de más bajo nivel, facilitando su codificación, que suelen aparecer en secuencia y de forma habitual en este tipo de aplicaciones,

[8] Puede encontrarlo en la URL <https://github.com/magusti/OpenAL_examples/OpenAL_ALUT_vs_ALC_AL>.

[9] El fichero WAVE original está en <https://github.com/vancegroup/freealut/tree/master/src> y su versión hexadecimal en <https://github.com/vancegroup/freealut/blob/master/src/alutWaveform.c>.

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

int main (int argc, char **argv) {
  ALuint elBuffer[3], laFont;
  ALenum error;
  ALint sourceState;

  alutInit (&argc, argv);
  error = alGetError();
  if (error) printf("Error alutInit: %s\n", alutGetErrorString(error));

  printf ("La versio de ALUT instalada es %d.%d \n",
            alutGetMajorVersion(), alutGetMinorVersion());
  printf("OpenAL Renderer is '%s'\n", alGetString(AL_RENDERER) );
  printf("OpenAL Version is '%s'\n", alGetString(AL_VERSION) );
  printf("OpenAL Vendor is '%s'\n", alGetString(AL_VENDOR) );
  alGenSources (1, &laFont);

  elBuffer[0] = alutCreateBufferHelloWorld();
  error = alGetError();
  if (error) printf("Hello World: %s\n", alutGetErrorString(error));
  alSourcei( laFont, AL_BUFFER, elBuffer[0]);
  alSourcePlay( laFont );  alutSleep( 1 );  alSourceStop( laFont );

  elBuffer[1] = alutCreateBufferWaveform(ALUT_WAVEFORM_SINE,
                                         440.0, 0.0, 1.0);
  error = alGetError();
  if (error) printf("Waveform: %s\n", alutGetErrorString(error));
  alSourcei (laFont, AL_BUFFER, elBuffer[1]);
  alSourcePlay( laFont );  alutSleep( 1 );  alSourceStop( laFont );

  elBuffer[2] = alutCreateBufferFromFile( "footsteps-4.wav" );
  error = alGetError();
  if (error) printf("FromFile: %s\n", alutGetErrorString(error));
  alSourcei(laFont, AL_BUFFER, elBuffer[2]);
  alSourcePlay (laFont);
  alGetSourcei( laFont, AL_SOURCE_STATE, &sourceState);
  while (sourceState == AL_PLAYING){
    alutSleep( 1 );
    alGetSourcei( laFont, AL_SOURCE_STATE, &sourceState);
  }

  alDeleteBuffers(3, &elBuffer);
  alDeleteSources(1, &laFont);
  alutExit();
  return EXIT_SUCCESS;
} // fi de main
```

Listado 1: Ejemplo de aplicación de OpenAL que sí usa los servicios de ALUT.

4.2 Ejemplo de código sin ALUT

Este ejemplo surge de la experimentación propia sobre la documentación de OpenAL y de un buen número de ejemplos consultados, de entre los que podemos destacar [4] para la gestión del contexto, [6] para la generación de una señal senoidal, [7] para la función de carga de un fichero RAW o [8] para la reproducción de un fichero WAVE.

El código del ejemplo está repartido entre el Listado 2, el Listado 3 y el Listado 4. Como en el caso del apartado 4.1 se remarcan algunas instrucciones para facilitar la comparación entre ambas situaciones. Adelantamos ya que aquí aparecen muchas instrucciones con prefijo "alc". Veamos ahora, cómo cambia el código al utilizar directamente las operaciones de los niveles ALC y AL de OpenAL. Para compilar este ejemplo habrá de utilizar una orden como:

```
$ gcc openal__alc_al.c -o openal__alc_al  \
      $(pkg-config openal --cflags –libs) -lm
```

El Listado 2, aparte de las cabeceras iniciales propias del lenguaje C, contiene:

- Una implementación propia de la función *DisplayALError* (líneas 9 a la 11), por respetar el código original (visto en [4]) y la de la función *fill_buffer* (líneas 13 a la 23) que utilizaremos en el Listado 3.
- Y la inicialización del *hardware* entre las líneas 34 a la 54, con el uso de *alcCreateContext*, junto a *alcCreateContext* y *alcMakeContextCurrent*. ¿Recuerda la función *alutInit*? Pues el resultado es el mismo pero, ahora, con veinte líneas. Lo interesante es que podríamos haber permitido al usuario escoger si se dispone de varios dispositivos de sonido que funcionen bajo OpenAL, pero en nuestro caso solo hay uno: la versión software *de OpenAL Soft*.

Por su parte, el Listado 3, contiene dos bloques:

- En primer lugar, entre las líneas 55 y 70, continua con el código que permite averiguar las versiones y extensiones[10] de la versión instalada de OpenAL, el *hardware* de reproducción (líneas 71 a la 79) disponible y de grabación a través del micrófono (líneas 80 a la 92). Una curiosidad, entre cámaras web y panel delantero y trasero de mi equipo, ¡muestra hasta cuatro micrófonos disponibles!
- En segundo lugar, el uso de la función *fill_buffer* [6], líneas 98 a la 100, nos permite sintetizar una onda senoidal, al estilo de como lo hacía la función *alutCreateBufferWaveform* de ALUT.

Y, para finalizar, en el Listado 4, vemos:

- Cómo se reproduce el sonido de la señal senoidal generado anteriormente, líneas 101 a la 114, calculando el tiempo que durará esa señal creada matemáticamente; porque ahora no es un parámetro con el que se ha generado la secuencia, como lo era en ALUT.
- Cómo se puede leer un fichero RAW, a partir del código de Taverner [7], cuyos parámetros son conocidos de partida: un canal, muestras de 16 bits con signo y a razón de 44.100 muestras por segundo. Esto permite que el fichero no contenga esta información como cabecera y todo su contenido sea información de sonido.

[10] Cerca de treinta en la plataforma donde lo hemos probado (Ubuntu 20.04), pudiendo obtenerlas todas de golpe o preguntando por una en concreto.

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <math.h>
4.  #include <AL/alut.h>
5.  #include <string.h>
6.  #include <unistd.h>
7.  #include <limits.h>
8.
9.  void DisplayALError( char *titolMensatge, ALenum error) {
10.     printf("%s %s\n", titolMensatge, alGetString( error ) );
11. }// fi de DisplayALError
12.
13. #define TAMANY_BUFFER  100*1024
14. #define SAMPLERATE     22050
15. #define BYTESPERSAMPLE 1
16. #define NCANALS        1
17.
18. void fill_buffer(void *audioBuffer, size_t size, int frequency ) {
19.   char *dest = (char *)audioBuffer;
20.   for (int i=0; i<size; i++) {
21.         dest[i] = UCHAR_MAX * sin(frequency*(2*M_PI)*i/SAMPLERATE);
22.   }
23. }
24.
25. int main(int argc, char *argv[]) {
26.   ALCdevice *elDispositiu;
27.   ALCcontext *elContext;
28.   ALuint elBuffer, laFont;
29.   ALenum format, error, g_bEAX;
30.   const ALchar *pDeviceList, *llista;
31.   ALbyte data[ TAMANY_BUFFER ];
32.   int i;
33.
34.   elDispositiu = alcOpenDevice(NULL);
35.   if (elDispositiu) {
36.     elContext = alcCreateContext(elDispositiu,NULL);
37.     if (elContext != NULL) {
38.      alcMakeContextCurrent( elContext );
39.      error = alGetError();
40.      if (error) DisplayALError("Error  alcMakeContextCurrent:", error);
41.     } else {
42.      error = alGetError();
43.      if (error) DisplayALError("Error en alcCreateContext:", error);
44.     }
45.   }else {
46.     error = alGetError();
47.     if (error) DisplayALError("Error en alcOpenDevice:", error);
48.   }
49.   if (elDispositiu != NULL) {
50.     elContext = alcCreateContext(elDispositiu,NULL);
51.     if (elContext != NULL) {
52.         alcMakeContextCurrent( elContext );
53.     }
54.   }
...
```

Listado 2: Ejemplo de aplicación de OpenAL con ALC/AL, que no usa los servicios de ALUT (parte 2).

```
...
55.   printf("OpenAL Renderer is '%s'\n", alGetString(AL_RENDERER) );
56.   printf("OpenAL Version is '%s'\n", alGetString(AL_VERSION) );
57.   printf("OpenAL Vendor is '%s'\n", alGetString(AL_VENDOR) );
58.   g_bEAX = alIsExtensionPresent("EAX2.0");
59.   printf ("alIsExtensionPresent(\"EAX2.0\") retorna %d.\n", g_bEAX );
60.   llista = alGetString(AL_EXTENSIONS); i=1;
61.   if (llista) {
62.       printf("\nExtensions de OpenAL disponibles:");
63.       printf("\n%2d - ", i);
64.       while (*llista)         {
65.          if (*llista != ' ' ) printf("%c", *llista);
66.          else
67.           printf("\n%2d - ", ++i);   llista++;
68.          }
69.       }
70.   printf("\n");
71.   pDeviceList = alcGetString(NULL, ALC_DEVICE_SPECIFIER);
72.   i=1;
73.   if (pDeviceList)      {
74.       printf("Drivers/Manejadores disponibles de audio:\n");
75.       while (*pDeviceList)    {
76.           printf("%2d - %s\n", i, pDeviceList);
77.           pDeviceList += strlen(pDeviceList) + 1;   i++;
78.       }
79.   }
80.   printf("Manejador/Driver por defectofecte:\n%s\n",
81.            alcGetString(NULL, ALC_DEFAULT_DEVICE_SPECIFIER) );
82.   //Dispositius de captura (entrada de audio, micròfons)
83.   pDeviceList = alcGetString(NULL, ALC_CAPTURE_DEVICE_SPECIFIER);
84.   i=1;
85.   if (pDeviceList)    {
86.       printf("\nAvailable Capture Devices are:\n");
87.       while (*pDeviceList)        {
88.           printf("%2d - %s\n", i, pDeviceList);
89.           pDeviceList += strlen(pDeviceList) + 1;
90.              i++;
91.       }
92.   }
93.   alGetError(); // clear error code
94.   alGenBuffers(1, &elBuffer);
95.   if ((error = alGetError()) != AL_NO_ERROR)     {
96.         DisplayALError("alGenBuffers :", error);
97.   }
98.   fill_buffer(data, TAMANY_BUFFER, 440 ); // Genera una señal básica
99.   alBufferData( elBuffer, AL_FORMAT_MONO8, (const Alvoid*)data,
100.                TAMANY_BUFFER, SAMPLERATE );
...
```

Listado 3: Ejemplo de aplicación de OpenAL con ALC/AL, que no usa los servicios de ALUT (parte 2).

La función *load*, la usamos entre las líneas 115 y la 122, *no la hemos incluido*[11] *por brevedad en la exposición*, se encarga de leer el fichero de disco y nos devuelve el número de bytes leídos (*dataSize*) y el contenido (*data2*). La línea 124 lo hace

[11] Pero sí que estará en el repositorio mencionado de *GitHub*.

sonar y, en la línea 126, hacemos un cálculo de lo que tardará, en tiempo, su reproducción.

```
...
101.  alGenSources(1,&laFont);
102.  if ((error = alGetError()) != AL_NO_ERROR)  {
103.        DisplayALError("alGenSources 1 : ", error);
104.    }
105.  alSourcei(laFont, AL_BUFFER, elBuffer);   // Attach buffer to source
106.  if ((error = alGetError()) != AL_NO_ERROR)    {
107.        DisplayALError("alSourcei AL_BUFFER : ", error);
108.  }
109.  alSourcePlay( laFont );
110.  printf("Espera %f segons\n", (float)TAMANY_BUFFER /
111.                      (float) (SAMPLERATE* BYTESPERSAMPLE * NCANALS) );
112.  sleep( round( (double)TAMANY_BUFFER /
113.                (double)(SAMPLERATE* BYTESPERSAMPLE * NCANALS) ) );
114.
115.  printf("Cargar un fitxer RAW: footsteps.raw\n");
116. {
117.   long dataSize;
118.   const ALvoid* data2;
119.   data2 = load( "footsteps.raw", &dataSize );
120.   printf("He llegit de %s %ld bytes\n", "footsteps.raw", dataSize );
121.   alBufferData( elBuffer, AL_FORMAT_MONO16, data2, dataSize, 44100 );
122.   free( (void*)data2 );
123.   alSourcei( laFont, AL_BUFFER, elBuffer);
124.   alSourcePlay( laFont );
125.   printf("Espera %f segons\n", (float)dataSize/(float)(44100* 2 * 1));
126.   sleep( round( (double)dataSize/(double)(44100 * 2 * 1) ) );
127. }
128.
129.   //...
130.
131.  elContext = alcGetCurrentContext();
132.  elDispositiu = alcGetContextsDevice( elContext);
133.  alcMakeContextCurrent(NULL);
134.  alcDestroyContext( elContext );
135.  alcCloseDevice( elDispositiu );
136.  return EXIT_SUCCESS;
137.}
```

Listado 4: Ejemplo de aplicación de OpenAL con ALC/AL, que no usa los servicios de ALUT (parte 3).

- Hemos dejado, en la línea 129, un espacio abierto que el lector interesado podrá encontrar rellenado en el repositorio mencionado de *GitHub* (donde están los ejemplos de código completos que estamos examinando, por brevedad de la exposición). Ahí se podrá ver el uso del código de Vicent [8] para cargar un fichero WAVE sin ayuda de librerías externas (como SDL, sndfile, etc.). Como en el caso del fichero RAW, se puede hacer encargándose uno mismo de todo el trabajo, partiendo de la especificación del formato de fichero, pero estamos seguros que el lector entenderá (al ver el número de líneas y la complejidad de las operaciones), que no es una opción trivial y pueden aparecer errores por malinterpretaciones de la especificación del formato.
- Y, para terminar, entre las líneas 131 y 135, se procede a liberar los recursos concedidos al inicio., con *alcDestroyContext* y *alcCloseDevice*.

5 Conclusiones y cierre

OpenAL está dividido en tres componentes. ALUT, la de más alto nivel, es vista como una parte de la que se podría prescindir. En este artículo hemos explorado su planteamiento y las operaciones que provee. Nuestro objetivo era analizar (con un desarrollo práctico) si es posible desarrollar en OpenAL sin ALUT.

La respuesta rápida que es que sí se puede desarrollar sin ALUT. Pero hemos visto que la alternativa de trabajar sin una librería de alto nivel, para determinadas acciones, no es interesante en tanto que obliga a reinventar la rueda. ALUT nos permite trabajar en construir una demo de forma rápida y multiplataforma; de hecho, si sirve el acceso que proporciona a ficheros en formato WAVE y no necesitamos de una interfaz gráfica, tenemos lo suficiente incluso para desarrollar una aplicación final con poco consumo de recursos por parte de la librería de OpenAL. Si necesitamos un formato de fichero concreto, podríamos complementar ALUT con una librería específica (para ese formato) o una solución más integrada como las opciones que ofrece ALURE[12].

Espero que, a estas alturas, tenga ejecutándose el código propuesto y comprobando que puede oír el audio con OpenAL con y sin ALUT ¿No es así? Pues descárguelo del repositorio en *GitHub* indicado 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] Deckhead. (2020).The Complete Guide to OpenAL with C++ – Part 1: Playing a Sound. Disponible en <https://indiegamedev.net/2020/02/15/the-complete-guide-to-openal-with-c-part-1-playing-a-sound/>.

[6] openal-surround-test. Repositorio de Github disponible en <**https://github.com/aib/openal-surround-test**>.

[7] Tony Tavener (2021) Cranial Burnout: OpenAL Soft -- demonstration of binaural 3D audio. <http://cranialburnout.blogspot.com/2012/08/openal-soft-demonstration-of-binaural.html>.

[8] Vincent_M . (2010). Loading WAVs For OpenAL. Disponible en la URL <https://gamedev.net/forums/topic/573599-loading-wavs-for-openal/4660752/>.

[12] Podemos encontrar su página web en <https://kcat.tomasu.net/alure.html>.