UNIVERSIDAD POLITECNICA DE VALENCIA

Introducción al procesado de audio mediante OpenAL

Apellidos, nombre	Agustí Melchor, Manuel[1] (magusti@disca.upv.es)
Departamento	[1]Dpto. De Ing. De Sistemas y Computadores
Centro	Universidad Politécnica de Valencia

UNIVERSIDAD POLITECNICA DE VALENCIA

1 Resumen

En este artículo vamos a presentar una introducción al tema de procesado de audio. Para ello se va a utilizar el API de OpenAL [1],[2] y [3] como vehículo para realizar ejemplos. Aunque es un material de apoyo, está pensando de forma que la realización práctica de lo presentado en este documento conduzca a la obtención de una serie de resultados experimentales: código que permita generar “resultados audibles” a modo de demostración de lo expuesto.

Se va a presentar un enfoque que permita trabajar en cualquier sistema operativo desarrollando sistemas de características multimedia que hagan uso de operaciones sobre audio, en este caso, como un medio característico del que extraer y sobre el que mostrar información.

2 Introducción

El presente desarrollo está encaminado a ofrecer una perspectiva de presentación de ejemplos de uso de audio bajo la interfaz de programación (API) de OpenAL, pero no esta misma de forma exhaustiva. Para ello, se va a participar en la implementación de pequeños ejemplos independientes que utilizan el audio como medio principal de salida de información.

Figura 1: Logotipo de OpenAL tomados del sitio web.

OpenAL es una interfaz de programación multiplataforma[1] para audio multicanal 3D apropiada para el uso en aplicaciones de videojuegos y otras relativas al tratamiento de audio. Se puede ver el logotipo habitual en la fig. 1

En la mayoría de distribuciones de su sistema operativo de elección, se puede encontrar disponible (por ejemplo, en GNU/Linux busca los paquetes que empiecen con 'libalut' y con 'libopenal'), en su última versión está disponible también en la web de *Creative* [1].

3 Objetivos

Una vez que el alumno lea con detenimiento este documento y experimente con los ejemplos propuestos, será capaz de:

[1]Actualmente en Mac OS X, iPhone, GNU/Linux (tanto para OSS como para ALSA), BSD, Solaris, IRIX, Microsoft Windows, Xbox, Xbox 360 y MorphOS.

- Determinar qué pasos son necesarios para escribir una aplicación que utilice OpenAL. Dando así a conocer al alumno, de un modo participativo, el uso de OpenAL para llevar a cabo operaciones básicas de procesado de audio digital.
- Manipular sonido en un computador, en términos de OpenAL.
- Generar un ejecutable a partir del código creado y las bibliotecas de programación necesarias, sin instalar ningún entorno pesado, desde la línea de órdenes.
- Utilizar una plataforma de experimentación abierta y multiplataforma que permita al lector proseguir su autoaprendizaje de forma independiente.

4 Desarrollo

La librería permite modelizar una colección de fuentes de audio moviéndose en un espacio tridimensional que son oídos por un único oyente en algún lugar de ese espacio. Los objetos básicos en OpenAL son un oyente (*Listener*), una fuente de audio (*Source*) y una zona de memoria (*Buffer*) que contiene la información de audio. Cada *buffer* puede ser asignado a una o más fuentes que representan posiciones (definidas por coordenadas en un espacio tridimensional) de donde "brota" el audio. Siempre hay un oyente, que representa el punto donde se "escuchan" los sonidos que generan las fuentes (*rendering*).

La funcionalidad de OpenAL se estructura [2] en base a estos objetos:

- Una fuente (*source*) contiene un puntero a una zona de memoria (*buffer*), la velocidad, posición y dirección e intensidad del sonido.
- El oyente (*listener*) representa la velocidad, posición y dirección del mismo, así como la ganancia asociada a todos los sonidos. Aunque se pueden definir varios oyentes sólo uno puede estar activo.
- Los buffers contienen audio en formato PCM, en muestras de ocho o dieciséis bits, tanto en monofónico como en formato estéreo.

El motor de OpenAL realiza todos los cálculos necesarios como la atenuación debida a la distancia, el efecto Doppler, etc. El resultado final de todo esto para el usuario final es que las aplicaciones realizadas con OpenAL recrean un escenario aural cercano mientras el usuario se mueve en un espacio tridimensional.

Entre estos objetos existen unas relaciones o asociaciones, por lo que se habla de otros dos objetos más: un contexto (*context*) y un dispositivo (*device*). Estos elementos están fuera de esta exposición, así que no los detallaremos más. La figura 2 muestra los objetos básicos de OpenAL y sus relaciones con los objetos de contexto y dispositivo.

Desde el punto de vista de la utilización de OpenAL en una aplicación, puede abstraerse como [3] un conjunto de instrucciones que permite especificar las fuentes de sonido y el oyente bajo un sistema de coordenadas espaciales en tres dimensiones, junto a órdenes que permiten controlar cómo serán *renderizados* esos sonidos. La inmediatez de estas instrucciones no está garantizada, puesto que existen latencias dependiendo de la implementación. Idealmente, estas no deberían ser apreciadas por el usuario.

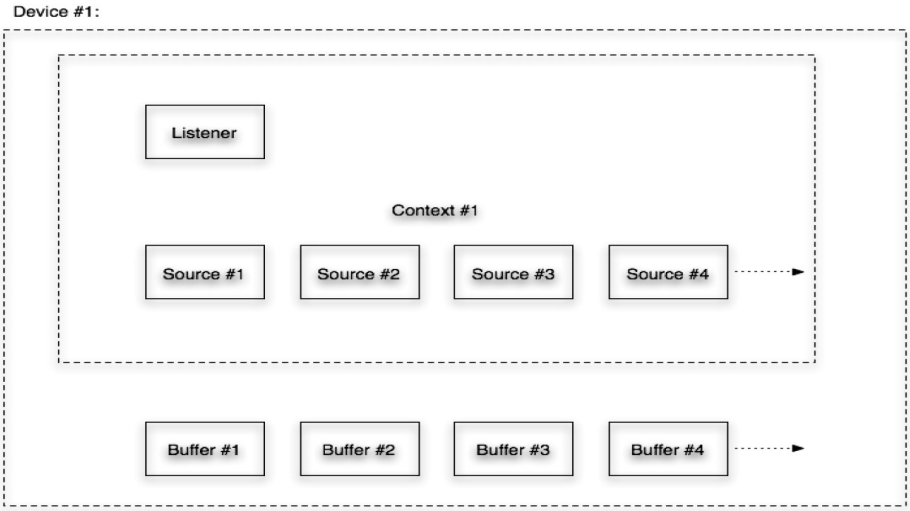

Figura 2: Un ejemplo de jerarquía entre objetos de OpenAL [2]

La especificación de OpenAL incluye dos subsecciones en la interfaz del programador: las de núcleo (AL) y las de contexto (ALC, *Audio Library Context*) que se utiliza para gestionar un contexto y el uso de recursos en las diferentes plataformas en que se puede utilizar. Sobre estas, existe un nivel superior de agrupación de funciones básicas y de otras funciones para facilitar el uso de formatos de ficheros de audio: ALUT (*The OpenAL Utility Toolkit, [4]*). Utilizaremos este mayormente para evitar tener que implementar la lectura de los diferentes formatos de ficheros de audio.

En este texto vamos presentar un esquema de trabajo: cómo se estructura y compila un programa que utiliza el API de OpenAL, qué parámetros pueden recibir y qué resultados devuelven. Es aconsejable tener la ayuda siempre a mano, puedes comprobar que está junto a la instalación realizada[2], otras referencias útiles están disponibles en [2], [3] y [4]. Los siguiente ejemplos tienen un interfaz en modo texto para centrarnos en la temática del audio. Son ejemplos de código para ver las cuestiones básicas.

4.1 'Hello, world' utilizando ALUT

Este, véase Listado 1, es el tradicional primer programa en cualquier lenguaje de programación o librería, pero este, en lugar de escribirlo en pantalla, dice “Hello, world!”. Se ha obviado la gestión de errores por simplicidad. ¡Qué más se le puede pedir al primer ejemplo!.

Para ejecutarlo se compila rápidamente desde la línea de órdenes con

```
$ gcc `pkg-config freealut --cflags --libs` helloWorld.c -o helloWorld
```

Atención a las comillas, que son las de ejecución: es como poner una tilde abierta a un espacio en blanco. Se ejecuta con la orden

```
$ ./helloWorld
```

[2]En GNU/Linux es habitual que se encuentre en /usr/share/doc/opencv-doc/ref/opencvref_cv.htm.

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

int main (int argc, char **argv)
{
 ALuint helloBuffer, helloSource;

 alutInit (&argc, argv);

 helloBuffer = alutCreateBufferHelloWorld ();
 alGenSources (1, &helloSource);
 alSourcei (helloSource, AL_BUFFER, helloBuffer);
 alSourcePlay (helloSource);
 alutSleep (1);

 alDeleteSources( 1, &helloSource );
 alDeleteBuffers( 1, &helloBuffer );
 alutExit ();
 return EXIT_SUCCESS;
}
```

Listado 1. Un primer programa en OpenAL,

Es breve: dice “Hello, world!” reproduciendo una secuencia de bytes tomada de la grabación de una locución de una persona real. ¿Simple, verdad? ¡Es un primer paso, no esta mal!

Como se habrá observado, todo gira en torno al uso de *buffers* y a las operaciones de proceso sobre los objetos de tipo fuente de audio *(source*) empiezan con el prefijo '*al*'. Las funciones de ALUT nos ofrecen agrupaciones de estas operaciones de OpenAL y tienen el prefijo '*alut*'.

La implementación utilizada en el ejemplo se basa en definir tipos simples, enteros que sirven como identificadores, puesto que no manipularemos las estructuras de audio directamente en estos ejemplos. Las funciones hacen todo el trabajo. ¡Deberías tomarte el tiempo necesario para localizarlas en la documentación antes de seguir!. El esquema de trabajo utilizado en el ejemplo se basa en:

- Inicializar el sistema y el audio con `alutInit` y el audio con `alutCreateBufferHelloWorld.`
- Crear la fuente de audio con `alGenSources`, `alSourcei` y `alSourcePlay.`
- Asignar las propiedades de la fuente, en este caso el sonido asociado que está en memoria en un *buffer*, con `alSourcei.`
- Reproducir el audio de la fuente con `alSourcePlay` y darle tiempo a que acabe con `alutSleep.`
- Liberar recursos y terminar, con `alutExit.`

Ahora debes reflexionar sobre el código propuesto, por que las funciones señalas serán habituales en la mayoría de casos. Revisa el código, asegúrate que sabes qué hace cada función. Por ejemplo:

- ¿Si destruyo el buffer justo antes de `alSourcePlay`, funcionará?

- ¿El tiempo en alutSleep, en qué unidades está? ¿Que pasa si lo quito? ¿Y si pongo otro valor?

4.2 Generando señales simples

Se pueden generar, aparte del la síntesis de “Hello world”, otros tipos de ondas básicas como: senoidales, cuadradas, triangulares, impulsioanales y ruido blanco.. Para ello recurriremos a la llamada *aAlutCreateBufferWaveform, l*os parámetros permiten especificar la forma de la onda, la frecuencia, ... Con lo que podemos hacer variaciones sobre el primer esquema, cambiando la función que genera la onda a reproducir. La podemos ver descrita en la documentación que mantiene *Creative Labs* y que tiene el siguiente interfaz:

```
ALuint alutCreateBufferWaveform (ALenum waveshape, ALfloat frequency,
                                 ALfloat phase, ALfloat duration);
```

El código siguiente, genera un tipo de onda diferente como respuesta a la tecla pulsada. No es tarea de OpenAL ocuparse del teclado, así que en anexo “” tienes una implementación del getche de MS/Windows para terminales de otros operativos.

Lo compilaremos con la línea de órdenes:

```
$ gcc `pkg-config freealut --cflags --libs` helloWorld2.c getch.c -o helloWorld2
```

```
#include <stdio.h> //getchar, printf
#include <stdlib.h>
#include <AL/alut.h>
#include "getch.h"

int main (int argc, char **argv)
{
 ALuint buffers[6], fuente;
 char c;

 alutInit (&argc, argv);
 c = 'h';

  buffers[0] = alutCreateBufferHelloWorld ();
  buffers[1] = alutCreateBufferWaveform(ALUT_WAVEFORM_SINE, 440.0, 0.0, 1.0);
  buffers[2] = alutCreateBufferWaveform(ALUT_WAVEFORM_SQUARE,  440.0, 0.0, 1.0);
  buffers[3] = alutCreateBufferWaveform(ALUT_WAVEFORM_SAWTOOTH,  440.0, 0.0, 1.0);
  buffers[4] = alutCreateBufferWaveform(ALUT_WAVEFORM_WHITENOISE,  440.0, 0.0, 1.0);
  buffers[5] = alutCreateBufferWaveform(ALUT_WAVEFORM_IMPULSE, 440.0, 0.0, 1.0);

  alGenSources (1, &fuente);
  printf("'h'ellow, 's'ine, sq'u'are, sa'w'tooh, white'n'oise, 'i'mpulse ('q' para salir): \n");

  do
  {
   switch ( c )
   {
  case 'h':  alSourcei (fuente, AL_BUFFER, buffers[0]);
          break;
  case 's': alSourcei (fuente, AL_BUFFER, buffers[1]);
          break;
  case 'u':  alSourcei (fuente, AL_BUFFER, buffers[2]);
```

```
        break;
  case 'w':  alSourcei (fuente, AL_BUFFER, buffers[3]);
        break;
  case 'n':  alSourcei (fuente, AL_BUFFER, buffers[4]);
        break;
  case 'i':  alSourcei (fuente, AL_BUFFER, buffers[5]);
        break;
 }// switch

 alSourcePlay (fuente);
 alutSleep (1);
 c = getch();
 printf("%c\r", c); fflush(stdout);
 } while ((c != 'q') && (c != 'Q'));

 printf("\nTerminando ... \n");

alDeleteSources(1, &fuente);
alDeleteBuffers(6, buffers);

alutExit ();
return EXIT_SUCCESS;
}
```

Listado 2. Generación de señales de audio básicas con OpenAL.

Esto ya es un poco más divertido, ¿eh? Pero también se ha hecho más complejo. Deberás comprobar que el esqueleto del programa principal no ha variado básicamente respecto del ejemplo inicial. El cambio está básicamente en la función *alutCreateBufferWaveform*, que es la genera una cantidad de bytes en memoria (en un buffer) que sigue una cierta función matemática y que representa a un sonido muy simple y, por cierto, muy limpio para lo que estamos acostumbrados en el mundo real.

¿Has analizado el código? Deberías, por que esta dinámica va a ser extensible a todas las situaciones, sólo que habrá que crear un número diferente de sonidos o de fuentes de audio y asignar las propiedades correspondientes. Venga, vamos a complicarlo un poco más.

4.3 Trabajando con ficheros de audio

Siguiendo con nuestra aproximación vamos a ver una variación del holaMundo que ahora tomará la señal de audio a hacer sonar desde un fichero con la información extraída a partir de la documentación sobre ALUT [4]. Observar como en este caso no se deja sonar un tiempo fijo, sino que se va consultando y, mientras queden datos, se mantiene el programa en espera.

```
… // Recuerda que trabajamos sobre el código del listado 2 como esqueleto básico
int main (int argc, char **argv)
{
 ALuint buffer, fuente;
 int error;
 ALint sourceState;

 if (argc == 1)
```

```
{
…
}
else
{
  alutInit (&argc, argv);
...
   buffer = alutCreateBufferFromFile( argv[1] );
…
    alGenSources (1, &fuente);
    alSourcei(fuente, AL_BUFFER, buffer);

    alSourcePlay (fuente);
    // Lo vamos a dejar sonar mientras hayan datos.
  alGetSourcei( fuente, AL_SOURCE_STATE, &sourceState);
  while (sourceState == AL_PLAYING)
   alGetSourcei( fuente, AL_SOURCE_STATE, &sourceState);

  alDeleteSources( 1, &fuente );
  alDeleteBuffers( 1, &buffer );
  alutExit ();
  return EXIT_SUCCESS;
 } // if (argc == 1)
}// main
```

Listado 3. Accediendo a ficheros de audio con ALUT en OpenAL.

El lector curioso habrá notado que se sigue utilizando la misma estructura básica, sólo que vamos introduciendo nuevos elementos:

- Para cargar el fichero de audio en un *buffer* en memoria hemos recurrido a *alutCreateBufferFromFile*.
- Existe una función para preguntar por una cierta propiedad de un objeto, en este caso de una fuente de sonido, *AlGetSourcei* con la que decidimos cuánto tiempo es necesario esperar hasta que se termina de reproducir un fichero.

Pero me gustaría que se preguntara otras cosas y que le encuentre la repuesta experimentando. Por ejemplo: compruebe que soporta la reproducción de ficheros WAVE mono y estereofónicos; busque al menos un par ficheros que contengan valores diferentes de número de canales y tamaño de las muestras.

5 Concluyendo

A lo largo de este objeto de aprendizaje hemos visto qué estructura tienen las aplicaciones que utilizan el API de OpenAL. La experimentación sobre el código propuesto le permitirá al lector:

- Determinar qué pasos son necesarios para escribir una aplicación que utilice OpenAL.
- Manipular sonido en un computador, en términos de OpenAL.
- Generar un ejecutable a partir del código creado y las bibliotecas de programación necesarias, sin instalar ningún entorno pesado, desde la línea de órdenes.

Ya se que los conceptos son muy básicos, pero te recuerdo que nuestros objetivos en este caso han sido cumplidos. Las nociones más básicas han sido presentadas, ahora te toca a ti atreverte a pensar en el audio como en un tipo de datos más que puedes procesar. Quien sabe, entre todos conseguiremos que los computadores puedan generar sonido como respuesta a sus acciones o a la interacción con el usuario con medios mucho más naturales para nosotros.

6 Bibliografía

[1] Creative Labs: Connect:: OpennAL. http://connect.creativelabs.com/openal/default.aspx

[2] OpenAL 1.1 Specification and Reference. 2005. Version 1.1, <http://connect.creativelabs.com/openal/Documentation/OpenAL 1.1 Specification.pdf>

[3] Garin Hiebert et al. OpenAL Programmer's Guide, OpenAL Versions 1.0 and 1.1. Creative Technology Limited, 2006

[4] The OpenAL Utility Toolkit (ALUT). http://connect.creativelabs.com/openal/Documentation/The OpenAL Utility Toolkit.htm

7 Anexo I

El siguiente código está sacado de <http://wesley.vidiqatch.org/code-snippets/alternative-for-getch-and-getche-on-linux/> y cito textualmente:

> getch() and getche() functionality for UNIX,
> based on termios (terminal handling functions)
>
> This code snippet was written by Wesley Stessens (wesley@ubuntu.com)
> It is released in the Public Domain.

Yo lo he separado en estos dos ficheros para su inclusión en los ejemplos tratados en este artículo: véase listados 4 y 5.

```
#ifndef GETCH_H
#define GETCH_H

char getch(void);
char getche(void);

#endif
```

Listado 4. Contenido del fichero getch.h

```
#include <termios.h>
#include <stdio.h>

static struct termios old, new;

/* Initialize new terminal i/o settings */
void initTermios(int echo) {
   tcgetattr(0, &old); /* grab old terminal i/o settings */
   new = old; /* make new settings same as old settings */
   new.c_lflag &= ~ICANON; /* disable buffered i/o */
   new.c_lflag &= echo ? ECHO : ~ECHO; /* set echo mode */
   tcsetattr(0, TCSANOW, &new); /* use these new terminal i/o settings now */
}

/* Restore old terminal i/o settings */
void resetTermios(void) {
   tcsetattr(0, TCSANOW, &old);
}

/* Read 1 character - echo defines echo mode */
char getch_(int echo) {
   char ch;
   initTermios(echo);
   ch = getchar();
   resetTermios();
   return ch;
}

/* Read 1 character without echo */
char getch(void) {
   return getch_(0);
}

/* Read 1 character with echo */
char getche(void) {
   return getch_(1);
}

/* Let's test it out
int main(void) {
   char c;
   printf("(getche example) please type a letter: ");
   c = getche();
   printf("\nYou typed: %c\n", c);
   printf("(getch example) please type a letter...");
   c = getch();
   printf("\nYou typed: %c\n", c);
   return 0;
}
*/
```

Listado 5. Contenido del fichero getch.c.