



Introducción al uso del API de bajo nivel de FMOD

Apellidos, nombre	Agustí Melchor, Manuel (magusti@disca.upv.es)
Departamento	Dpto. de Ing. De Sistemas y Computadores
Centro	Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València

1 Resumen de las ideas clave

En el contexto de la realización de aplicaciones o sistemas que emplean audio espacial (también denominado audio 3D, tridimensional o envolvente) hay dos grandes librerías para realizar nuestras aplicaciones: OpenAL y FMOD. En este trabajo veremos una introducción a cómo se puede incorporar FMOD a una de nuestras aplicaciones, desde su instalación hasta la revisión de algunos ejemplos básicos y cómo construir el ejecutable.

Aunque la librería que es motivo de este trabajo es muy popular, por ser utilizada en plataformas de desarrollo de videojuegos, hay poca documentación sobre cómo instalarla y cómo utilizarla en una aplicación propia. Así que quiero ofrecer una serie de punteros a dónde acudir para obtener la librería, compilar los ejemplos y ver algunas secuencias de instrucciones típicas para la reproducción de archivos y para la manipulación de audio 3D. Este no es un trabajo de detalle de todas las posibilidades de FMOD, pero sí lo suficiente como para entender los ejemplos básicos.

Este trabajo se ha realizado sobre la versión de GNU/Linux Ubuntu 18.04, pero es totalmente transportable a otros sistemas operativos.

2 Objetivos

La falta de documentación de esta librería puede que sea debida a que no es un estándar abierto, también porque su uso está relacionado con las operaciones de gestión de audio a bajo nivel (que son siempre muy complejas) y, quizá también, porque no es habitual encontrar referencias a su carácter multiplataforma. **Este trabajo presenta** al lector la secuencia de pasos para tener instalados los componentes que permitan el desarrollo sobre esta librería y también dónde encontrará ejemplos para empezar a desarrollar con ella. Una vez que el lector haya leído este documento, será capaz de:

- Encontrar la versión de instalación más actual que se haya publicado de la librería FMOD e instalarla.
- Identificar en el código fuente los pasos necesarios para construir una aplicación que haga uso de ficheros y audio 3D.

3 Introducción

Para el desarrollo de aplicaciones de naturaleza multimedia que hacen uso del sonido y con características multiplataforma nos encontramos con librerías¹ como SDL (para operaciones con ficheros de diferentes formatos, conversiones, efectos y audio 2D), OpenAL (para audio 3D, que es un motor de audio 3D de pequeño consumo de recursos) y el que se puede considerar el estándar *de facto* que es FMOD, en tanto en cuanto que motores de desarrollo como *Unreal Engine* o *Unity3D* han optado por FMOD, véase la Figura 1, en sus últimas versiones. Lo cual le ha dado mucha popularidad en este campo y por el gran número de plataformas para las que ha sido portado.

FMOD ([1], [2]) es un motor de efectos de sonido propietario de *Firelight Technologies*. Está escrito en C++ y ha sido portado para su uso en las plataformas de escritorio (*Windows*, *macOS* y *Linux*), móviles (*iOS*, *BlackBerry* y *Android*) y videoconsolas (*Wii*, *Wii U*, *3DS*, *Switch*, *Xbox*, *Xbox 360*, *Xbox One*, *PlayStation 2*, *PlayStation 3*, *PlayStation 4*, *PlayStation Portable*, *PlayStation Vita*

¹ Véase <<https://www.libsdl.org/>>, <<https://www.openal.org/>> y <<http://www.fmod.com/>>.

y *Google Native Client*), con soporte para aceleración en plataformas basadas en *AMD TrueAudio* o *Sound Blaster*. Da soporte a un buen número de formatos de ficheros, incluyendo los nativos de las plataformas señaladas, formatos abiertos, MIDI y audio en bruto. Existen licencias de uso de FMOD para desarrollo, que son necesarias para redistribuir FMOD como parte de una aplicación comercial. No es necesaria una licencia para el desarrollo, además, explícitamente el EULA² que acompaña la descarga permite su uso con fines educativos y no comerciales³.

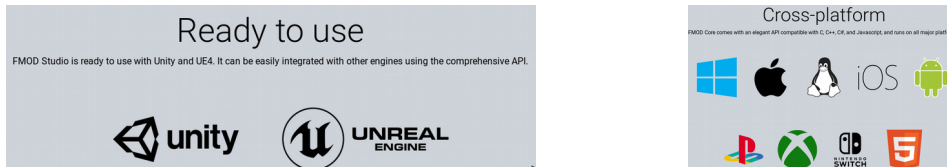


Figura 1: Motores de videojuegos y plataformas para los que existen versiones de FMOD. Imágenes de <<https://fmod.com/studio>>.

FMOD está formado por dos partes:

- *FMOD Studio*. Una herramienta visual con un funcionamiento similar a un sistema de edición de audio digital (*Digital Audio Workstation* o *DAW*) como *Ableton Live*, *Cubase*, *Pro Tools*, *Ardour* o *Rosegarden*. Se usan para digitalizar muestras de audio, grabar, conectar con dispositivos MIDI, convertir entre formatos y añadir efectos como operaciones habituales.
- *FMOD Core API*. Un API⁴ de desarrollo de aplicaciones en *C/C++* (y con enlaces para *C#* y *Javascript*) con funciones de bajo nivel como cargar y reproducir audio, aplicar efectos especiales, agrupar pistas en canales y mezcladores, así como generar audio tridimensional.

Así pues, el API de FMOD Core API ofrece [3] las primitivas de más bajo nivel de FMOD, las abstracciones básicas:

- Acceso al hardware, a ficheros y algoritmos de procesado de señal, como p. ej. *Effect Parameters* (los que configuran la acción del *Digital Signal Processor* o *DSP*), *System* (acceso a la tarjeta de sonido), grabación, sonido 3D y oclusiones.
En la ambientación espacial de audio, FMOD ofrece una solución para trabajar con múltiples oyentes⁵, aunque para ello impone ciertas restricciones.
- Objetos básicos, como *Sound* (que es donde se guarda el audio descomprimido), *Channel* (el objeto que simula la situación desde la que se origina el audio), *ChannelGroup* (un conjunto de canales a los que se aplica una misma operación y que pueden anidarse, también se los llama *bus* o *mixers*).

2 Son las siglas de *End-User License Agreement* o “Acuerdo de Licencia con el Usuario Final”, que son las condiciones que imponen los propietarios de un programa, aplicación o producto al usuario.

3 Véase la URL: <<https://www.fmod.com/resources/eula/>>.

4 Siglas de *Application Programming Interfaz*. Véase al respecto la URL <https://en.wikipedia.org/wiki/Application_programming_interface>.

5 Véase “White Papers | 3D Sounds. Split screen / multiple listeners”. Disponible en <<https://www.fmod.com/resources/documentation-api?version=2.0&page=white-papers-3d-sounds.html>>.

4 Desarrollo

Vamos a hablar aquí de cómo es el proceso de Instalación del SDK⁶, la generación de los ejemplos que lo acompañan y la secuencia básica de uso.

4.1 Instalación

La Figura 2 muestra el sitio web de FMOD. En la barra de menú de la parte superior hay un *Download* que lleva al mismo sitio que el botón que puedes ver en azul con la leyenda “Get FMOD”. Al llegar a esa URL te encontrarás con la necesidad de registrarte, Figura 3.

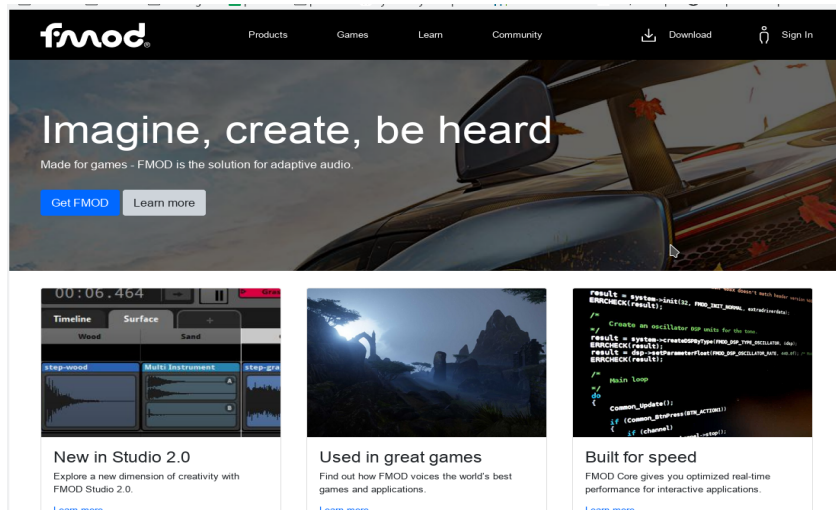


Figura 2: Sitio web de FMOD.

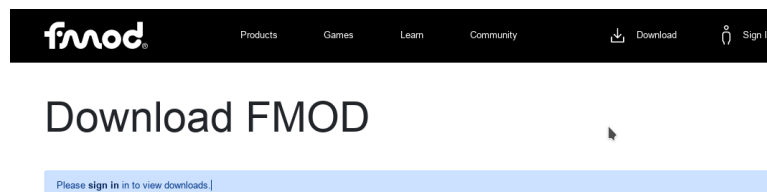


Figura 3: Apartado de descarga del sitio web de FMOD al acceder sin haberse registrado.

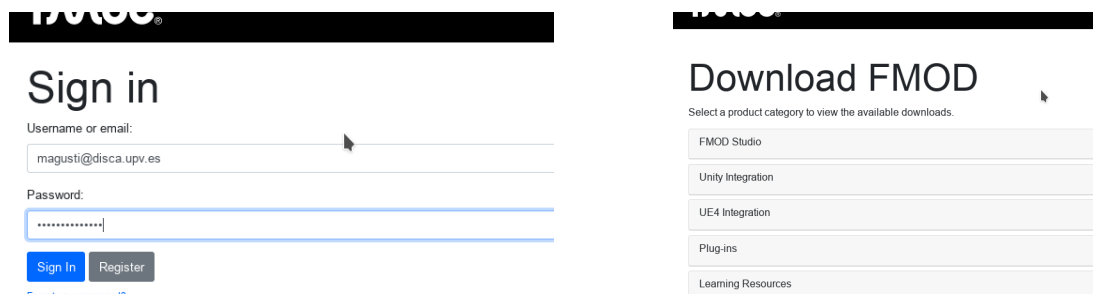


Figura 4: Registrándose en el sitio web de FMOD (a) y el nuevo contenido de la página de descargas tras el registro (b).

Ese proceso es relativamente rápido: tras la selección de un nombre de usuario y contraseña, como muestra la Figura 4a, se completa con unos pocos datos

6 Software Development Kit, véase al respecto en la URL https://en.wikipedia.org/wiki/Software_development_kit.

personales (que he obviado) y nos lleva otra vez a la sección de descarga, que ahora tiene nuevo contenido (como se ve en la Figura 4b). Escogeremos la opción “FMOD Studio”.

```
$ mv fmodstudioapi20000linux.tar.gz .
$ tar xvf fmodstudioapi20000linux.tar.gz
$ cd fmodstudioapi20000linux/
```

Listado 1: Órdenes ejecutadas para la instalación

Descargado el fichero, lo llevaremos a un directorio de nuestra elección y lo descomprimos como indica el Listado 1. Dentro del directorio está todo el contenido, como muestra la Figura 5: En particular veremos ejemplos del Core en *api/examples* y el manual en *doc/FMOD API User Manual*⁷.

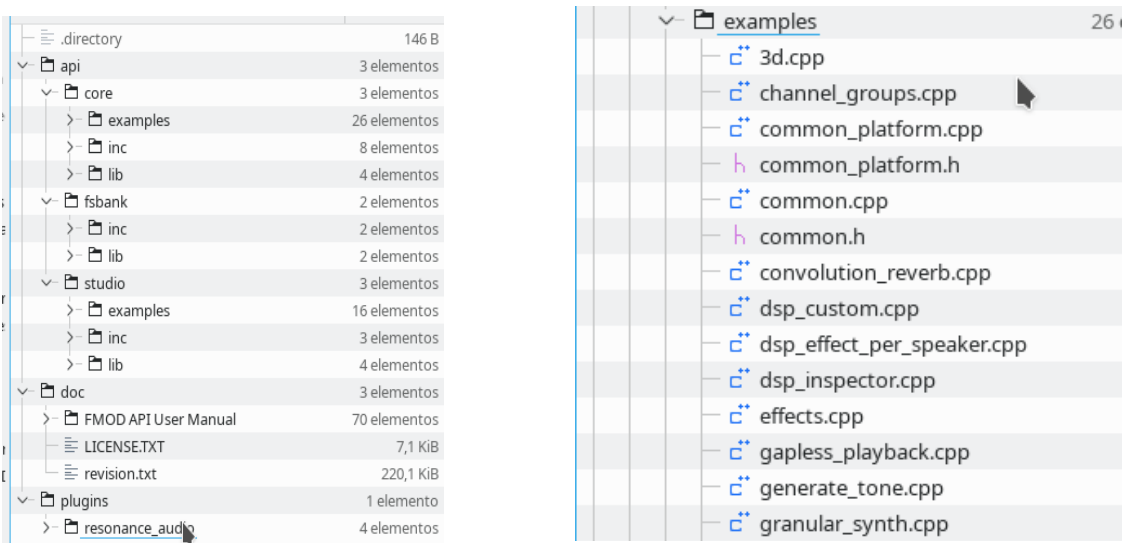


Figura 5: Contenido del SDK descargado (izquierda) y un detalle de los ejemplos (derecha) donde se ve el de *3d.cpp* que veremos .

Para compilarlos, véase la secuencia de órdenes utilizada en el Listado 2, entraremos en el directorio *api/core/examples* y utilizaremos los ficheros *Makefile* que hay dentro de *api/core/examples/make* para generar las versiones de los ejecutables cuyos código fuente están en el directorio superior.

```
$ cd api/core/examples/make/
$ for i in *.makefile; do echo $i;
make -f $i CPU=x86_64 CONFIG=Release; done
$ play_sound
```

Listado 2: Órdenes ejecutadas para construir los ejemplos.

He utilizado un bucle for para hacerlo y no tener que ir uno a uno. Tras lo cual, ya puedes ejecutar cualquiera de los ejemplos disponibles. Los que necesiten ficheros de audio para ejecutarse ya los encontrará el código, tranquilo. Diviértete y prueba alguno, te dejo tres sugerencias: *play_sound*, *3d* y *multiple_speaker*.

⁷ En este directorio está lo mismo que en <https://www.fmod.com/resources/documentation-api?version=2.0&page=core-guide.html>.

4.2 Reproducción de ficheros: ejemplos *play_sound.cpp* y *play_stream.cpp*

Estos dos ejemplos muestran cómo cargar sonidos desde fichero y reproducirlos. En pantalla veremos una salida como la que muestra la Figura 6 izquierda. Si se pulsa una de las teclas de los sonidos disponibles, se va incrementando el número de sonidos en ejecución, lo que se muestra en la línea “Channels Playing”, véase la Figura 6 derecha

```

=====
Play Sound Example.
Copyright (c) Firelight Technologies 2004-2019.
=====
Press 1 to play a mono sound (drumloop)
Press 2 to play a mono sound (jaguar)
Press 3 to play a stereo sound (swish)
Press Q to quit
Time 00:00:00/00:00:00 : Stopped
Channels Playing 0

=====
Play Sound Example.
Copyright (c) Firelight Technologies 2004-2019.
=====
Press 1 to play a mono sound (drumloop)
Press 2 to play a mono sound (jaguar)
Press 3 to play a stereo sound (swish)
Press Q to quit
Time 00:00:36/00:01:47 : Playing
Channels Playing 8

```

Figura 6: Resultado de la ejecución del ejemplo *play_sound*.

El contenido de cada fichero se carga y descomprime completamente en memoria, a diferencia del ejemplo *play_stream*, cuya salida se puede ver en la Figura 7. Este ejemplo utiliza un fichero especial: un banco de sonidos o FSB (FMOD Studio Bank⁸) que optimiza la carga y acceso frente a varios ficheros sueltos.

```

=====
Play Stream Example.
Copyright (c) Firelight Technologies 2004-2019.
=====
Press 1 to toggle pause
Press Q to quit
Time 00:11:34/00:11:46 : Playing
Channels Playing 0

```

Figura 7: *play_stream*

Entre *play_sound* y *play_stream* no hay mucha diferencia en la demanda de recursos al sistema, Figura 8a. Pero si se tiene en cuenta que *play_stream* accede a un fichero de sonidos, *wave_vorbis.fsb*, cuya capacidad, en disco, es casi tres veces mayor que la de los tres que utiliza *play_sound* (*drumloop.wav*, *jaguar.wav* y *swish.wav*), Figura 8b, entonces si que se ve una diferencia, ¿verdad?.

Nombre	Usuario	CPU %	Memoria	Memoria compartida
play_stream	magusti		1.412 K	6.204 K
play_sound	magusti		1.404 K	6.312 K

a)

```

du -sh drumloop.wav jaguar.wav swish.wav wave_vorbis.fsb
84K drumloop.wav
80K jaguar.wav
36K swish.wav
316K wave_vorbis.fsb

```

b)

Figura 8: Ocupación de memoria y disco de los ejemplos (a) y de los ficheros de audio asociados (b).

Estos ejemplos son extensos, así que he eliminado la parte de la presentación en pantalla que no son instrucciones relativas al audio. Además, sugiero al lector profundizar en los contenidos de la sección “White Papers | Getting Started”, del

⁸ Véase en el apartado “White Papers | Studio API Banks” del “FMOD API User Manual 2.00”.

“FMOD API User Manual 2.00” para revisar las instrucciones que he remarcado y que constituyen la secuencia básica de operaciones a realizar en el código que he transcrito aquí (Listado 3 y Listado 4) del primer ejemplo citado.

```
//Play Sound Example Copyright (c), Firelight Technologies Pty, Ltd 2004-2019.
#include "fmod.hpp"
#include "common.h"

int FMOD_Main() {
    FMOD::System *system; FMOD::Sound *sound1, *sound2, *sound3;
    FMOD::Channel *channel = 0; FMOD_RESULT result;
    unsigned int version; void *extradriverdata = 0;

    Common_Init(&extradriverdata);
    // Create a System object and initialize
    result = FMOD::System_Create(&system);    ERRCHECK(result);
    result = system->getVersion(&version);    ERRCHECK(result);
    if (version < FMOD_VERSION) {
        Common_Fatal("FMOD lib version %08x doesn't match header version %08x",
                    version, FMOD_VERSION); }
    result = system->init(32, FMOD_INIT_NORMAL, extradriverdata); ERRCHECK(result);
    result = system->createSound(Common_MediaPath("drumloop.wav"),
                                FMOD_DEFAULT, 0, &sound1); ERRCHECK(result);
    result = sound1->setMode(FMOD_LOOP_OFF); /* drumloop.wav has embedded
    loop points which automatically makes looping turn on, */
    ERRCHECK(result); /* so turn it off here. We could have also just put
    FMOD_LOOP_OFF in the above CreateSound call. */
    result = system->createSound(Common_MediaPath("jaguar.wav"), FMOD_DEFAULT,
                                0, &sound2); ERRCHECK(result);
    result = system->createSound(Common_MediaPath("swish.wav"), FMOD_DEFAULT,
                                0, &sound3); ERRCHECK(result);

    // Main loop
    do {
        Common_Update();
        if (Common_BtnPress(BTN_ACTION1)) {
            result = system->playSound(sound1, 0, false, &channel); ERRCHECK(result); }
        if (Common_BtnPress(BTN_ACTION2)){
            result = system->playSound(sound2, 0, false, &channel); ERRCHECK(result); }
        if (Common_BtnPress(BTN_ACTION3)) {
            result = system->playSound(sound3, 0, false, &channel); ERRCHECK(result);}
        ...
    }
```

Listado 3: play_sound.cpp (1ª parte).

```

...
    result = system->update();ERRCHECK(result);
{ unsigned int ms = 0, lenms = 0; bool playing = 0, paused = 0;
  int channelsplaying = 0;
  if (channel) { FMOD::Sound *currentsound = 0;
    result = channel->isPlaying(&playing);
    if ((result != FMOD_OK) && (result != FMOD_ERR_INVALID_HANDLE) && (result !=
FMOD_ERR_CHANNEL_STOLEN)) { ERRCHECK(result); }
    result = channel->getPaused(&paused);
    if ((result != FMOD_OK) && (result != FMOD_ERR_INVALID_HANDLE) && (result !=
FMOD_ERR_CHANNEL_STOLEN)){ ERRCHECK(result); }
    result = channel->getPosition(&ms, FMOD_TIMEUNIT_MS);
    if ((result != FMOD_OK) && (result != FMOD_ERR_INVALID_HANDLE) && (result !=
FMOD_ERR_CHANNEL_STOLEN)) { ERRCHECK(result); }
    channel->getCurrentSound(&currentsound);
    if (currentsound) { result = currentsound->getLength(&lenms, FMOD_TIMEUNIT_MS);
      if ((result != FMOD_OK) && (result != FMOD_ERR_INVALID_HANDLE) && (result !=
FMOD_ERR_CHANNEL_STOLEN)) { ERRCHECK(result); }
    }
  }
  system->getChannelsPlaying(&channelsplaying, NULL);
  [...]
}
Common_Sleep(50);
} while (!Common_BtnPress(BTN_QUIT));
// Shut down
result = sound1->release(); ERRCHECK(result);
result = sound2->release(); ERRCHECK(result);
result = sound3->release(); ERRCHECK(result);
result = system->close(); ERRCHECK(result);
result = system->release(); ERRCHECK(result);
Common_Close();
return 0;
}

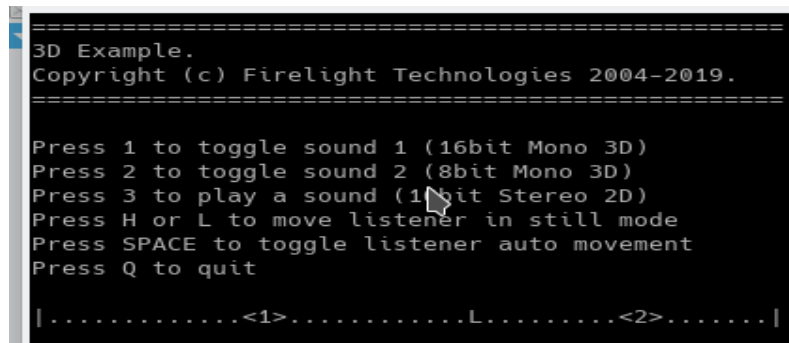
```

Listado 4: play_sound.cpp (2º parte).

4.3 Audio tridimensional: ejemplo 3d.cpp

Dentro de los ejemplos del SDK este que nos ocupa aborda la especialización del sonido. En pantalla veremos un,a salida como la que muestra la Figura 9 en la que se indican las posibles acciones a realizar. Además, en la línea inferior una letra 'L' (que se refiere a la posición del *Listener*, el oyente de la escena sonora) se va a mover en el espacio (representado por una línea de puntos), pasando cerca de la posición de dos sonidos (observe que "sound 1" y "sound 2" tienen diferentes tamaños de muestras, 16 y 8 bits, pero ambos son "3D") de forma que se oirá cómo influyen esas posiciones. Con las teclas 'h' y 'l' se puede mover a voluntad la posición del oyente para explorar esta escena. Un tercer

sonido, "sound 3" se reproduce como "2D", así que no se verá afectado por la posición del oyente.



```
=====
3D Example.
Copyright (c) Firelight Technologies 2004-2019.
=====

Press 1 to toggle sound 1 (16bit Mono 3D)
Press 2 to toggle sound 2 (8bit Mono 3D)
Press 3 to play a sound (16bit Stereo 2D)
Press H or L to move listener in still mode
Press SPACE to toggle listener auto movement
Press Q to quit

|.....<1>.....L.....<2>.....|
```

Figura 9: Figura 6: Resultado de la ejecución del ejemplo 3d.cpp.

Con la misma idea del apartado anterior de no hacer larga la exposición con los detalles de salida en pantalla, he eliminado parte del código para resaltar lo relativo al audio y he remarcado también la secuencia de pasos propia de este ejemplo. Puede el lector profundizar en la operativa de esas operaciones y sus parámetros explorando los contenidos de la sección "White Papers | 3D Sounds"⁹, del "FMOD API User Manual 2.00".

```
// 3D Example. Copyright (c), Firelight Technologies Pty, Ltd 2004-2019.
#include "fmod.hpp"
#include "common.h"
[[ ... ]]
int FMOD_Main() {
[[ ... ]]
    Common_Init(&extradriverdata);
    // Create a System object and initialize.
    result = FMOD::System_Create(&system);    ERRCHECK(result);
[[ ... ]]
    // Set the distance units. (meters/feet etc).
    result = system->set3DSettings(1.0, DISTANCEFACTOR, 1.0f);    ERRCHECK(result);
    // Load some sounds
    result = system->createSound(Common_MediaPath("drumloop.wav"), FMOD_3D, 0, &sound1);
    result = sound1->set3DMinMaxDistance(0.5f * DISTANCEFACTOR, 5000.0f * DISTANCEFACTOR);
    result = sound1->setMode(FMOD_LOOP_NORMAL);    ERRCHECK(result);
[[ ... ]]
    // Play sounds at certain positions
    {
        FMOD_VECTOR pos = { -10.0f * DISTANCEFACTOR, 0.0f, 0.0f };
        FMOD_VECTOR vel = { 0.0f, 0.0f, 0.0f };
        result = system->playSound(sound1, 0, true, &channel1); ERRCHECK(result);
        result = channel1->set3DAttributes(&pos, &vel); ERRCHECK(result);
        result = channel1->setPaused(false); ERRCHECK(result);    }
    ...
}
```

Listado 5: Instrucciones a destacar del ejemplo 3d.cpp.

9 Disponible en los archivos instalados y en la URL <https://www.fmod.com/resources/documentation-api?version=2.0&page=white-papers-3d-sounds.html> >.

La diferencia más importante entre este ejemplo y los dos anteriores es cómo se ha establecido que el fichero a la hora de cargarse ha de ser gestionado como tridimensional (*createSound* y *sound1*) y que la posición del mismo en el espacio (*channel*) está en función de los valores de coordenadas en el espacio (*pos*) y de velocidad (*vel*).

El resto se ha omitido porque se parece mucho a los anteriores y si no se va a emplear esta interfaz en modo texto (que seguramente no lo hará en el 98% de los casos, estimado lector), puede omitir sus entresijos.

5 Conclusión

La falta de difusión de la documentación existente de FMOD puede que sea el óbice a su uso en aplicaciones realizadas en C/C++. Este trabajo ha presentado desde cómo instalar las librerías, hasta cómo compilar los ejemplos que se encuentran en el SDK para poder tener elementos de experimentación y validar su posible uso.

Ahora que se ha podido ver que la librería ofrece muchas operaciones al desarrollador es hora de ponerse a hacer un desarrollo propio. Para hacerlo más divertido yo propondría ponerle un interfaz gráfico tridimensional a estos ejemplos que hemos señalado. ¿Te animas a poner a OpenGL a pintar una sencilla escena con cubos que marquen la posición de las fuentes de audio y ver y oír cómo queda esa escena del ejemplo 3d.cpp? Voy a buscar algunos ejemplos de sonidos de motores, disparos y explosiones para darle emoción. ¿Te animas?

6 Bibliografía

[1] Sitio web de FMOD. Disponible en <<https://fmod.com/>>.

[2] *Wikipedia contributors. FMOD. Wikipedia, The Free Encyclopedia.* Disponible en <<https://en.wikipedia.org/w/index.php?title=FMOD&oldid=893068049>>.

[3] FMOD API User Manual 2.00. Disponible en <<https://www.fmod.com/resources/documentation-api?version=2.0&page=welcome.html>>.