UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cálculo del espectro de frecuencias en FMOD

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

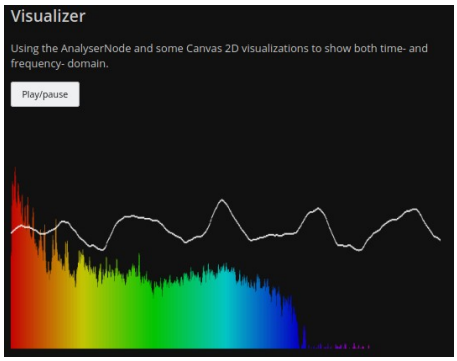
1 Resumen de las ideas clave

El sonido puede ser definido como una onda de presión que provoca la vibración de las moléculas en un medio, como puede ser el aire, y que cambia con el tiempo, por lo que se habla del sonido como un "frente de ondas" (véase Figura 1a). En el computador, a veces, solo se está interesado en su capacidad para reproducir esas sensaciones auditivas para el usuario, a partir de lo que se ha guardado en un fichero o de lo que puede sintetizar un dispositivo.

En otros casos, como sucede en aplicaciones para videoconsolas y para sistemas multimedia interactivos, el audio se quiere que acompañe y se ajuste a las acciones del usuario. Para conseguir esas acciones se recurre a las propiedades de la onda sonora como el volumen o la reverberación, que caracterizan el sonido en el dominio del tiempo. En cambio, otras acciones, se describen en el dominio de la frecuencia con más claridad, esto es, considerando el sonido como el resultado de una composición de señales más simples (como p. ej. ondas sinusoidales), caracterizadas por su frecuencia. Así, la Figura 1b) nos muestra como, en un instante de tiempo, el sonido que se escucha puede ser representado de manera gráfica por una curva que explica su variación de volumen en el tiempo (la línea blanca) y también por el conjunto de frecuencias que se escuchan en ese instante (las líneas verticales de colores). Esta respuesta, en el tiempo y en la frecuencia, que el sistema auditivo humano está acostumbrado a obtener de las señales sonoras, ha llevado a buscar modelos y métodos algorítmicos que permitan obtener la descripción de un sonido de forma numérica en ambos dominios.

ESTADO	MEDIO	VELOCIDAD DEL SONIDO (m/s)
Gaseoso	Aire (20ºC)	340
	Hidrógeno (0ºC)	1.286
	Oxígeno (0ºC)	317
	Helio (0ºC)	972
Líquido	Agua (25ºC)	1.493
	Agua de mar (25ºC)	1.533
Sólido	Aluminio	5.100
	Cobre	3.560
	Hierro	5.130
	Plomo	1.322
	Caucho	54
Vacío	Vacío	0

a)



b)

Figura 1: El sonido (a) como frente de ondas (imagen de ¿En qué medio se propaga mejor el sonido? <http://sonenacustica.blogspot.com.es/2013/11/en-que-medio-se-propaga-mejor-el-sonido.html >) y (b) Visualizer <https://webaudioapi.com/samples/visualizer/ >.

En este trabajo se introduce cómo se puede utilizar FMOD [1] para obtener la descomposición en frecuencias de una señal sonora. Para ello se revisará el proceso de instalación de FMOD y se explorará el código fuente de un ejemplo que permita obtener, en tiempo real, esta descomposición. Como requisito previo, se asume que se conoce el uso básico de FMOD; por lo que si el lector desconoce esta librería debería consultar [2] y

[5]. Este trabajo se ha realizado sobre la versión[1] 2.02.11 de FMOD y en plataforma Linux Ubuntu 20.04, pero es totalmente transportable a otros sistemas operativos.

2 Objetivos

La documentación de FMOD [4] está redactada en base al API[2] que proporciona: la explicación de los objetos y servicios que la compone. Pero hay que profundizar en los contenidos de la sección "White Papers | Getting Started", del "FMOD API User Manual 2.02"[3] para encontrar retazos de código que ayuden a pasar de la documentación a un ejemplo de código. Afortunadamente, la existencia de ejemplos en la distribución de esta librería ayuda a entender la documentación y a explorar aplicaciones pequeñas y completas. Pero no se dispone de ejemplos o una guía para el desarrollador que muestre ejemplos de uso de cada objeto o tipo de servicio que proporciona y, además, al publicarse nuevas versiones siempre aparecen cambios respecto a la forma de proceder de la versión anterior.

Este trabajo presenta al lector un caso de uso: el cálculo de frecuencias asociadas a una señal de sonido. Y lo acompaña de la secuencia de pasos para tener instalados los componentes que permitan el desarrollo sobre la librería FMOD; con lo que podrá experimentar de forma práctica mientra lee el documento. 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, descargarla y ejecutar los ejemplos que la acompañan.
- Identificar en el código fuente del ejemplo los pasos necesarios para construir una aplicación que calcule el espectro de frecuencias de un sonido, al tiempo que lo reproduce.
- Compilar y ejecutar el código del ejemplo.

3 Introducción

Para el desarrollo de aplicaciones de naturaleza multimedia que hacen uso del sonido y con características multiplataforma se dispone de librerías[4] como SDL (para operaciones con ficheros de diferentes formatos, conversiones, efectos y audio 2D), OpenAL (un motor de audio 3D de pequeño consumo de recursos) y FMOD que ha ganado mucha popularidad al ser utilizado en motores de desarrollo como *Unreal Engine* o Unity3D.

Veamos ahora en qué consiste el desarrollo propuesto y cómo se caracteriza la herramienta que se ha escogido para implementarlo.

---

1 Véase <https://www.fmod.com/docs/2.02/api/welcome-revision-history.html>.

2 Véase <https://en.wikipedia.org/wiki/API> para describir este término.

3 Disponible en <https://www.fmod.com/docs/2.02/api/white-papers.html>.

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

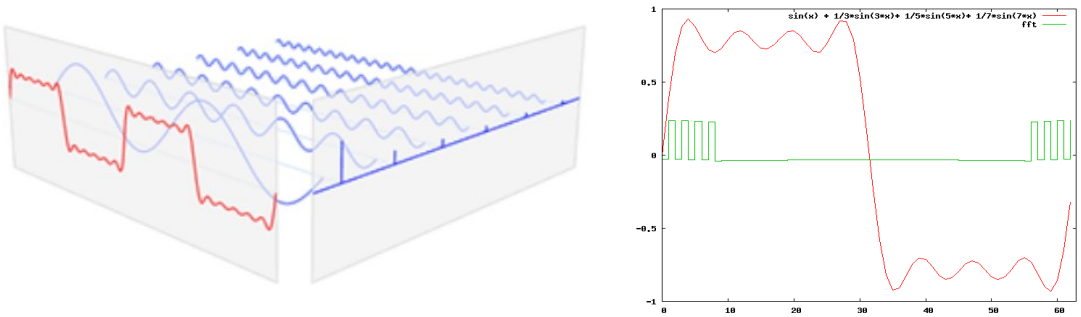
3.1 El sonido en el dominio de las frecuencias

De manera intuitiva se habla en el día a día de sonidos graves o agudos y se utilizan representaciones gráficas como las de la Figura 2; en ella, aparece el espectro de frecuencias, que es una técnica que permite expresar el rango de frecuencias audible por el ser humano (entre los 20 y los 20000Hz), o de manera más simplificada, agrupando por bandas de frecuencia. Y también admite otras interpretaciones (y no solo sería posible dibujarla, también cabe animar personajes…) con más interés, quizás, desde el punto de vista creativo.

a) *b)*

Figura 2: Representación gráfica del sonido en el dominio de las frecuencias: (a) espectro de frecuencias y (b) interpretación artística del espectro.

Matemáticamente, el análisis espectral se realiza con la transformada de Fourier (o **FT**, de *Fourier Transform*) que descompone una señal, como una composición de señales más simples (por ejemplo, señales senoidales, véase la Figura 3a) y permite así identificar las frecuencias. Este análisis se realiza para un intervalo de tiempo; por lo que, un sonido de cierta duración debe ser "troceado" en base al intervalo escogido y analizado de manera independiente de los demás. Si se aplica a todo el contenido de un fichero de sonido, seguramente, se verá que aparecen de casi todas las frecuencias y no resultará muy informativo. Desde el punto de vista computacional (digital) esta operación se implementa de manera eficiente con la transformada rápida de Fourier (**FFT**, de *Fast Fourier Transform*). Esta descomposición es reversible, así que se puede volver a regenerar (reconstruir o sintetizar) la señal en el domino del tiempo a partir del espectro de frecuencias, a este proceso se le denomina transformada inversa (IFT).



a) *b)*

Figura 3: Representación gráfica de la transformada de Fourier. (a) Descomposición en señales simples, imagen de <http://commons.wikimedia.org/wiki/Fourier_transform_time_and_frequency_domains_(small).gif> y (b) Comparación de espectros.

Para una señal como la que se observa en la Figura 3b que ha sido generada como la suma de 4 señales senoidales (representada con la línea en color rojo) se obtiene un espectro (representado en verde) con cuatro componentes distintas de cero. Se puede observar que el contenido del espectro es simétrico respecto a la mitad de la longitud de la ventana de la señal considerada: podemos omitir esa mitad al realizar el análisis de frecuencias por lo que veremos que el tamaño del espectro viene definido como la mitad del de la ventana de observación de la señal.

3.2 FMOD

FMOD ([1], [3]) 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, móbiles y videoconsolas. Da soporte a un buen número de formatos de ficheros, incluyendo los nativos de las plataformas señaladas, MIDI y audio en bruto. FMOD está formado por dos componentes. *FMOD Studio*, *que* es una herramienta visual con un funcionamiento similar a un sistema de edición de audio digital. Y *FMOD Core API*, un API[5] 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. Este API ofrece [4] 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.

Existen licencias de uso de FMOD para redistribuir FMOD como parte de una aplicación comercial. No es necesaria una licencia para este desarrollo, además, explícitamente el EULA[6] que acompaña la descarga permite su uso con fines educacionales y no comerciales.

4 Desarrollo

Vamos a hablar aquí de cómo es el proceso de Instalación del SDK[7], la generación de los ejemplos que lo acompañan y exploraremos un ejemplo de código para el calculo de la transformada en frecuencias. Para ello se habrá de incorporar la funcionalidad del cálculo de la FFT y ver cómo se puede parametrizar y recoger el resultado de esta función.

Puede encontrar el ejemplo que se va a explorar en el repositorio creado a tal efecto en *GitHub*[8], descárguelo y así podrá seguir la exploración del mismo que propone el resto de este apartado.

---

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

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

7 *Software Development Kit*, véase al respecto en la URL <https://en.wikipedia.org/wiki/Software_development_kit>.

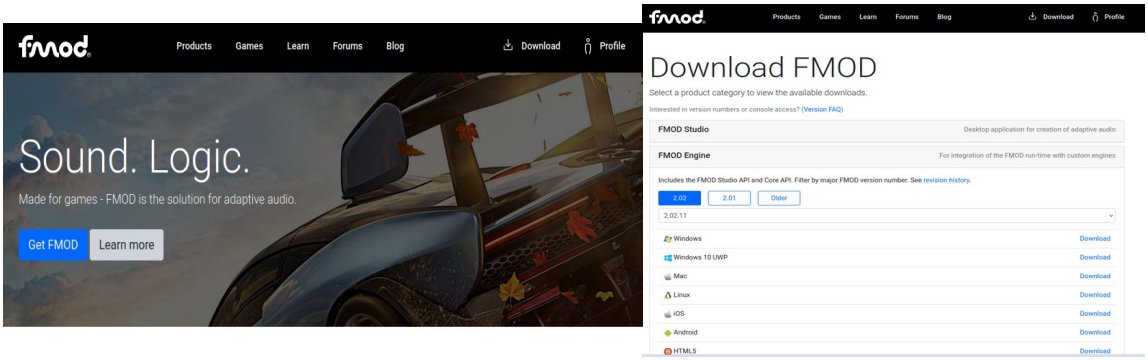
8 Encontrará el código que se expone en este artículo en el repositorio de Github <https://github.com/magusti/FMOD_examples/FMOD_FFT>.

4.1 Instalación

La Figura 4a 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 puede ver en azul con la leyenda "Get FMOD". En el apartado de descargas, Figura 4b, se encuentran los diferentes productos de esta empresa. En este caso, estamos interesados en el ***FMOD Engine***, del que se ofrecen varias versiones. Proceda a descargar la correspondiente a nuestra plataforma: en el momento de la redacción de este artículo es la versión 2.02 y para Linux, aunque por supuesto, puede escoger la plataforma de su elección.

La descarga genera un fichero `fmodstudioapi20211linux.tar.gz` que se descomprime con la orden `tar`:

```
$ tar xvf fmodstudioapi20211linux.tar.gz
```



a) *b)*

Figura 4: Sitio web de FMOD: (a) Get FMOD y (b) FMOD Engine.

Dentro del directorio está todo el contenido: como la documentación en *doc/FMOD API User Manual*, ejemplos del *Core* en *api/examples*, etc. 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, véase la secuencia de órdenes utilizada en el Listado 1 para compilarlos. Tras lo cual, ya se puede ejecutar cualquiera de los ejemplos disponibles. Los que necesiten ficheros de audio para ejecutarse ya los encontrará el código, tranquilo. Diviértase y pruebe alguno, por ejemplo: *play_sound*, *play_stream*, *3d* y *multiple_speaker*.

```
$ cd fmodstudioapi20211linux/api/core/examples/make/
$ for i in *.makefile; do make -f $i CPU=x86_64 CONFIG=Release; done
$ play_sound
```

Listado 1: Órdenes ejecutadas para construir los ejemplos.

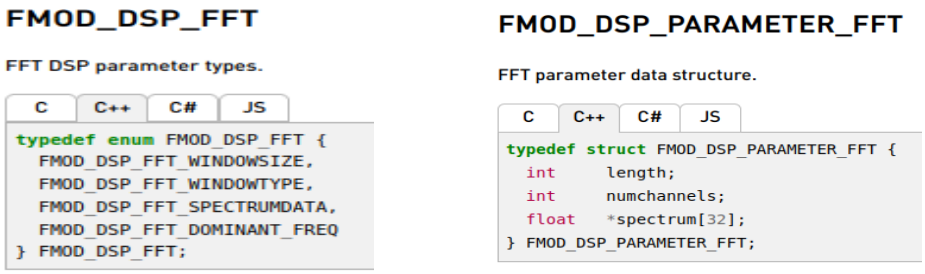
4.2 El ejemplo de código comentado

Sobre la base del ejemplo *play_stream*, se van a plantear las modificaciones para la obtención del espectro de frecuencias, por eso se llamará *play_stream_fft*. En esta versión 2.02 de FMOD esta funcionalidad ha cambiado[9]. Como sugiere la documentación, si tiene la

9 Véase el apartado "System::getSpectrum and System::getWaveData removed" de "White Papers | Transitioning from FMOD Ex", disponible en <https://fmod.com/docs/2.01/api/white-

paciencia de buscar, se basa ahora en dos grupos de acciones. Primero es necesario **añadir un módulo DSP** configurado para recoger a su entrada la señal de audio de la salida del sistema de FMOD; para ello se conectará, utilizando las instrucciones *System::getMasterChannelGroup* y *ChannelControl::getDSP*. Y, por otro, **configurar el módulo DSP para realizar la FFT** (véase Figura 5a); para lo que hay que crearlo con *System::createDSPByType* y tipo *FMOD_DSP_TYPE_FFT* y conectarlo con *ChannelControl::addDSP*. En tiempo de ejecución se podrá obtener el espectro (con *DSP::getParameterData*) o la frecuencia fundamental (con *DSP::getParameterFloat*).

El ejemplo *play_stream* que se acaba de nombrar, reproduce el contenido de un fichero en *streaming*, esto es, sin cargar totalmente el contenido del fichero en memoria [6], con lo que su consumo de recursos es menor que el ejemplo *play_sound* que carga completamente el fichero antes de empezar a reproducirlo. A cambio, la complejidad del código es mayor puesto que tiene que repetir la secuencia de operaciones de carga y reproducción hasta llegar al final del archivo. Nos interesa esta aproximación puesto que nos proporciona un enfoque que permite realizar las acciones en sincronía con el sonido que se está escuchando en cada instante, al proporcionar un "recorte" para poder así tener aislado el audio que suena.



a) *b)*

Figura 5: Estructuras de datos para el módulo DSP y el cálculo de la frecuencia. Fuente: documentación FMOD <https://fmod.com/docs/2.02/api/core-api-common-dsp-effects.html>.

Un comentario, en la versión original del código, se utiliza un fichero FSB que es un formato[10] definido por FMOD que "empaqueta" varios sonidos en un mismo fichero. Por ello se verá que utiliza la función *getSubSound*. En los ejemplos de uso que se dan en este artículo se proponen archivos OGG, Wave o MP3, para no crear confusión.

En el código podemos ver dos grandes partes:

- La primera es la que corresponde a la **creación de los elementos que van a componer el sistema**, corresponde con el Listado 1 y hasta la línea 64 del Listado 2. Cabe destacar que toda la aplicación se asienta sobre la variable *system* (línea 17) y el playSound (de la línea 66, Listado 2), que empieza a cargar el fichero en memoria. Dentro de este bloque, hay tres secciones. La primea sección, entre las líneas 1 a la 26 del Listado 1, es la parte común a todas las aplicaciones sobre esta librería y que realiza la inicialización de la librería FMOD. Ya se ha comentado que no se entrará en detalle sobre estos conceptos generales de FMOD en este artículo,

papers-transitioning-from-fmodex.html#system_getspectrum-and-system_getwavedata-removed">.

10 Más sobre formatos soportados por FMOD en <https://www.fmod.com/docs/2.02/api/core-guide.html#file-formats-support-for-the-most-optimal-format-for-games-fsb >.

pero no se pueden dejar pasar algunas cosas como los “números mágicos” para que no se confundan después. La línea 23 inicializa el sistema con 32 canales como valor máximo. Este número corresponde con el número de voces o notas que va a poder generar, en un mismo instante, la aplicación (y que se podría modificar, claro). Por eso la variable *freq* de la línea 32 es de 32 elementos.

- La segunda sección corresponde a la inicialización del módulo que realizará la FFT. Este es un módulo genérico de procesado de la señal (por eso se llama DSP, de *Digital Signal Processor*). De las líneas 26 a la 49 del Listado 1 se encuentra la primera parte de lo que se está añadiendo para calcular el espectro y que ahora corresponde a la instanciación del módulo que realiza la FFT. De esta parte cabe destacar la creación del módulo (línea 36, *createDSPByType*) y sus parámetros; estos son el tipo de ventana (línea 38) que define cómo se trocea el sonido y el tamaño (línea 41) del intervalo del sonido. En FMOD, este número debe ser una potencia de dos con un valor entre 128 y 16384 (esto es 128, 256, 512, 1024, 2048, 4096, 8192 ó 16384) y que por defecto se establece en 2048. Una vez creado hay que "conectarlo" (línea 47) con *addDSP*. Y la tercera sección de este primer bloque, véase el Listado 2, corresponde al origen del sonido a reproducir en *streaming* (línea 50), esto es el contenido del fichero que se recibe como argumento por la aplicación y que se inicia con *playSound* (línea 60). Observe que se guarda la frecuencia de muestreo del sonido cargado (línea 63), se necesitará un poco más tarde,

Y la segunda parte del código de esta aplicación, que va desde la línea 64 a la 94 del Listado 2, es **la que obtiene el espectro**. En ella se pueden distinguir dos secciones:

- La primera corresponde al bucle que comprueba la interacción del usuario. La entrada del usuario es actualizada con la instrucción *Common_Update* (línea 66), el avance del audio reproducido se realiza con system->update (línea 74) y se ha plegado (en la línea 76), todas las demás instrucciones que generan el interfaz por brevedad de la exposición.

- Y la segunda sección de esta segunda parte es la que actualiza el cálculo de la FFT para la ventana de audio que se está reproduciendo en ese momento. Para ello, la instrucción *getParameterData* (línea 77), recibe el parámetro *FMOD_DSP_FFT_SPECTRUMDATA* y devuelve sobre otro parámetro, *fftparameter (véase Figura 5b)*, los valores de frecuencia correspondientes a la transformada de Fourier. El doble bucle de las líneas 80 a la 94 muestra cómo se puede acceder a los valores de la FFT, para mostrarlos en pantalla en las líneas 85 y 86 (de cada componente de frecuencia y para cada canal) o para realizar un sencillo cálculo de determinación de la frecuencia media del sonido (líneas 87 a la 93), en las que se obvian los valores inferiores a 0,0001 que es un valor arbitrario escogido como filtro para lo que se considera ruido. Ese componente *spectrum*, que es accesible en esta versión 2.02 desde *fftparameter*, es el espectro de frecuencias que se andaba buscando. Son valores normalizados entre 0,0 y 1,0, organizados por canales de la señal original y el orden de rangos de frecuencia que se ha configurado en la línea 41 del Listado 1. El ancho de estos intervalos es el cálculo de la línea 83, para el que hemos guardado el valor de la frecuencia de muestreo en la línea 63.

```
1.  #include "fmod.hpp"
2.  #include "common.h"
3.
4.  // int FMOD_Main()
5.  int main(int argc, char *argv[]) {
6.     FMOD::System     *system;
7.     FMOD::Sound      *sound, *sound_to_play;
8.     FMOD::Channel    *channel = 0;
9.     FMOD_RESULT       result;
10.    unsigned int      version;
11.    void             *extradriverdata = 0;
12.    int               numsubsounds;
13.
14.    if (argc < 2) {  printf("Falta el fitxer.\n"); exit(1); }
15.    Common_Init(&extradriverdata);
16.    result = FMOD::System_Create(&system);
17.    ERRCHECK(result);
18.
19.    result = system->getVersion(&version);
20.    ERRCHECK(result);
21.    if (version < FMOD_VERSION)   { // Comprueba si es la 2.02.11
22.         Common_Fatal("FMOD lib version %08x doesn't match header
   version %08x", version, FMOD_VERSION);    }
23.    result = system->init(32, FMOD_INIT_NORMAL, extradriverdata);
24.    ERRCHECK(result);
25.
26. #define  TAMANY_CADENA 256
27.    int sampleSize = 64;
28.    FMOD::DSP *elDSP;
29.    FMOD::ChannelGroup *mastergroup;
30.    FMOD_DSP_PARAMETER_FFT *fftparameter;
31.    int rate, chan, nyquist;
32.    float val, freq[32];
33.    unsigned int len;
34.    char s[TAMANY_CADENA];
35.    //  Create the DSP effects.
36.    result = system->createDSPByType(FMOD_DSP_TYPE_FFT, &elDSP);
37.    if ((result != FMOD_OK) ) { ERRCHECK(result); }
38.    elDSP->setParameterInt(FMOD_DSP_FFT_WINDOWTYPE,
39.                           FMOD_DSP_FFT_WINDOW_RECT );
40.    ERRCHECK(result);
41.    elDSP->setParameterInt((int)FMOD_DSP_FFT_WINDOWSIZE,
                              sampleSize*2);
42.    ERRCHECK(result);
43.    //   Connect up the DSP network
44.    result = system->getMasterChannelGroup(&mastergroup);
45.    if ((result != FMOD_OK) ) { ERRCHECK(result); }
46.    result = mastergroup->addDSP(0, elDSP);
47.    ERRCHECK(result);
48.    result = elDSP->setActive(true);
49.    ERRCHECK(result);
   ...
```

Listado 1: Ejemplo play_streanm_fft.cpp.

```
...
50.     result = system->createStream(Common_MediaPath( argv[1] ),
51.                           FMOD_LOOP_NORMAL | FMOD_2D, 0, &sound);
52.     ERRCHECK(result);
53.     result = sound->getNumSubSounds(&numsubsounds);
54.     ERRCHECK(result);
55.     if (numsubsounds)  {
56.         sound->getSubSound(0, &sound_to_play);
57.         ERRCHECK(result);
58.     } else  {   sound_to_play = sound;    }
59.     // Play the sound.
60.     result = system->playSound(sound_to_play, 0, false,
61.                                &channel);
62.     ERRCHECK(result);
63.     result = system->getSoftwareFormat(&rate, 0, 0);
64.     // Main loop.
65.     do    {
66.      Common_Update();
67.      if (Common_BtnPress(BTN_ACTION1))     {
68.             bool paused;
69.             result = channel->getPaused(&paused);
70.             ERRCHECK(result);
71.             result = channel->setPaused(!paused);
72.             ERRCHECK(result);
73.      }
74.      result = system->update();
75.      ERRCHECK(result);
76.      { /^Mensajes de progreso de play_stream original*/ }
77.      result = elDSP->getParameterData(FMOD_DSP_FFT_SPECTRUMDATA,
78.                 (void **)&fftparameter, &len, s, TAMANY_CADENA);
79.      ERRCHECK(result)
80.      for (chan = 0; chan < fftparameter->numchannels; chan++) {
81.       float average = 0.0f, power = 0.0f;
82.       for (int i = 0; i < fftparameter->length; ++i) {
83.         float hz = i * (rate * 0.5f) / (nyquist - 1);
84.          int index = i; // + (16384* chan);
85.          Common_Draw( "fftparameter->spectrum[%d][%d] %f ",
86.                     chan, i, fftparameter->spectrum[chan][i]);
87.          if (fftparameter->spectrum[chan][i] > 0.0001f)    {
88.           average += fftparameter->spectrum[chan][i] * hz;
89.           power += fftparameter->spectrum[chan][i];
90.          }
91.         }
92.         if (power > 0.001f) { freq[chan] = average / power; }
93.         else    { freq[chan] = 0; }
94.       }
95.       Common_Sleep(50);
96.     } while (!Common_BtnPress(BTN_QUIT));
```

Listado 2: Ejemplo play_streanm_fft.cpp (2ª parte).

El resto de las líneas de código no se muestran por brevedad de la exposición, pero están en el repositorio de Github creado y se encargan de las tareas propias de limpieza y liberación de recursos.

5 Conclusión

Como el lector habrá podido observar al seguir el artículo, este trabajo ha presentado un caso de uso completo sobre FMOD, desde cómo instalar la librería, cómo compilar los ejemplos que se encuentran en el SDK. Así podrá experimentar y validar el cálculo del espectro de frecuencias de un sonido al tiempo que lo reproduce.

Ahora que se dispone de la librería instalada, un ejemplo, cómo compilar y ejecutarlo, es ocasión para planteamientos más "divertidos": yo propondría ponerle un interfaz gráfico tridimensional a estos ejemplos que hemos señalado. ¿Te animas a pintar con OpenGL o SDL los valores del espectro de frecuencias en forma de gráfica o de manera más artística? También podemos modificar el espectro y volver a convertirlo en la representación temporal del sonido y así presentar como funciona un ecualizador… Pero todo eso será en otra ocasión. Espero que, a estas alturas, tenga ejecutándose el código propuesto y comprobando que en pantalla aparecen los valores de frecuencia, a tiempo que suena la música o el contenido del fichero que le haya pasado como parámetro. Recuerde que puede descargar el código explicado del repositorio creado a tal efecto en *GitHub* y no cierre este el documento sin haberlo comprobado antes. ¡¡ÁNIMO!!

Un último comentario: "¿A qué suena un árbol que cae si no hay nadie para oírlo?". El sonido tiene mucho que decir todavía[11].

6 Bibliografía

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

[2] Agustí Melchor, M. (2019). Introducción al uso del API de bajo nivel de FMOD. Disponible en <http://hdl.handle.net/10251/123301 >.

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

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

[5] Kay. (2012)- Cutting Your Teeth on FMOD Part 1: Build environment, initialization and playing sounds. Disponible en <https://katyscode.wordpress.com/2012/10/05/cutting-your-teeth-on-fmod-part-1-build-environment-initialization-and-playing-sounds/>.

[6] Agustí Melchor, M. (2018). Reproducción de ficheros Opus con OpenAL: precarga vs "streaming". Disponible en <http://hdl.handle.net/10251/109211>.

11 Una sugerencia para pensar: "¿Hace ruido un árbol al caer si nadie está ahí para escucharlo? La respuesta de la ciencia y la filosofía" <https://www.bbc.com/mundo/noticias-56780694>.