FMOD

FMOD: APIs + Entorno de diseño

`https://www.fmod.com/` Conjunto completo de herramientas para

- El diseño sonoro de un videojuego 3D:
  - FMOD Studio (antiguamente: FMOD Designer Tool): entorno Middleware para el diseño de sonido interactivo
    - editor/herramienta de diseño de sonido no lineal
    - formato propio de bancos de sonido: colección estructurada de sonidos + parámetros de reproducción (eventos).

    Integración con Unity, Unreal, etc.
- Integración del sonido en el videojuego. Dos APIs en C/C++ (ahora también javascript y C# en Unity):
  - Low Level Programmer's API $\rightsquigarrow$ posicionamiento 3D, mezcla, etc (alternativa a OpenAL, mucho más completa).
  - FMOD Studio Programmer's API: carga y reproducción de bancos de sonido generados con FMOD Studio.

$\rightsquigarrow$ diferentes niveles de abstracción

✓ En constante actualización, muy eficiente.

✓ Buena documentación (navegable, intuitiva). Muchos vídeos

✓ Uso libre en proyectos no comerciales (no es código abierto)

FMOD: Low Level API

Low Level API

- Independiente de FMOD Studio.
- Librería de renderizado de audio 3D: posicionamiento 3D, mezcla en tiempo real, etc
  ... como OpenAL, pero mucho más completa
- Multiplataforma: Windows, Linux, Mac... Android, iOS, Game Cube, PS2, XBox, Nintendo...
- Soporta multitud de formatos de sonido (wav, ogg, mp3, midi, módulostracker,...)
- Efectos en tiempo real, extesionalidad (plugins),...
- ...funciones de análisis de espectro, drivers ASIO (Steinberg, baja latencia), ...

✓ Utilizado en más de 2000 juegos!!
(`https://www.fmod.com/games`).

- BioShock, Crysis, Diablo 3, Guitar Hero, Start Craft II, World of Warcraft.
- Integrada en motores de videojuegos: Unity3D, Unreal, CryEngine

Filosofía de FMOD

- FMOD puede asimilarse a una mesa de mezclas con distintos canales.

  channel en FMOD ≈ source OpenAL

- En cada canal se puede cargar un sonido que se puede gestionar independientemente: reproducir, parar, pausar...
- Cada canal puede situarse en una posición en el espacio y se le pueden asignar efectos.
- Varios canales pueden agruparse en otro canal virtual para manipularlos conjuntamente.
- Repertorio de funciones para controlar los canales.
- Permite asignar prioridades a los canales.

Los conceptos básicos de posicionamiento 3D en FMOD son los mismos que en OpenAL, pero el API es completamente distinto (mucho más directo en algunos aspectos).

Preparando el proyecto (Windows)

Librería estática de FMOD Low level:

- Enlazar la librería `/api/lowlevel/lib/fmod[L][64]_vc.lib`
  Opciones:
  - `L` es para habilitar los logs para depuración
  - `64` indica arquitectura de 64 bits (si no se pone nada es 32).

Se necesita también accesible en tiempo de ejecución la librería dinámica (dll) asociada:

- `fmod[L][64].dll`

Cabeceras

Accesibles en los directorios:

- `/api/lowlevel/inc` para el API Low Level
- `/api/studio/inc` para el API de FMOD Studio

```
#include "fmod.hpp"       // para utilizar el wrapper C++
#include "fmod_errors.h"  // para manejo de errores
...
```


Estructura del API Low Level

Clases básicas:

- System:
  - inicialización de la librería
  - selección de hardware y drivers
  - carga y reproducción de sonidos
  - control del LISTENER
  - consumo de CPU, ...

- Sound (unifica el tratamiento los distintos formatos de sonido):
  - control de buffers de sonido (se utiliza explícitamente para tareas específicas, menos frecuentes).
  - Puntos de sincronización de pistas (manejo de loops en tiempo real)

- Channel (hereda de ChannelControl). Control de parámetros de los canales:
  - volumen, panorama, etc
  - atributos para sonido 2D-3D, doppler, pitch
  - control de reproducción (play, pause, stop)

Más clases:

- ChannelGroup (hererda de ChannelControl): permite agrupar canales y obtener/modificar parámetros para todos ellos simultáneamente
- DSP: diseño y control de filtros, mezclas, reverberaciones... Posibilidades de expansión para el usuario.
- Geometry: simulación de recintos acústicos con obstrucción, oclusión, ...
- Reverb3D: manejo de reverbs.

Inicialización

Para facilitar la gestión de errores:

```
using namespace FMOD;

// para salidas de error
void ERRCHECK(FMOD_RESULT result){
  if (result != FMOD_OK){
    std::cout << FMOD_ErrorString(result) << std::endl;
    // printf("FMOD error %d - %s", result, FMOD_ErrorString(result));
    exit(-1); }
}
```

Inicialización básica:

```
int main() {
  System *system;
  FMOD_RESULT result;
  result = System_Create(&system);      // Creamos el objeto system
  ERRCHECK(result);

  // 128 canales (numero maximo que podremos utilizar simultaneamente)
  result = system->init(128, FMOD_INIT_NORMAL, 0);  // Inicializacion de FMOD
  ERRCHECK(result);
```


Cierre del sistema

- Se puede seleccionar el dispositivo/driver de salida (`System::getNumDrivers, System::getNumDrivers, System::setDriver`)
- puede además detectarlos automáticamente y cambiarlos dinámicamente (por ejemplo, si se conectan una tarjeta USB durante la ejecución).

Para cerrar el sistema:

```
result=system->release();
ERRCHECK(result);
```


Carga de un sonido

```
Sound *sound1;
result = system->createSound(
   "Battle.wav",     // path al archivo de sonido
   FMOD_DEFAULT,     // valores (por defecto en este caso: sin loop, 2D)
   0,                // informacion adicional (nada en este caso)
   &sound1);         // handle al buffer de sonido
```

Asignación a canal y reproducción del sonido:

```
Channel *channel;
result = system->playSound(
    sound,    // buffer que se "engancha" a ese canal
    0         // grupo de canales, 0 sin agrupar (agrupado en el master)
    false,    // arranca sin "pause" (se reproduce directamente)
    &channel); // devuelve el canal que asigna
// el sonido ya se esta reproduciendo
```

La variable `channel` permite modificar los parámetros de ese canal:

```
result = channel->setVolume(0.7f);
```

Para hacerlo efectivo,EN CADA VUELTA DEL BUCLE PPAL.:

```
result = system->update();
```

(simplePlayer,battle)

Carga de sonidos

Hemos cargado el sonido con:

```
system->createSound(..., FMOD_DEFAULT,...)
```

Esto significa que lo cargamos con la primera de las siguientes alternativas:

- `FMOD_LOOP_OFF`, `FMOD_LOOP_NORMAL`, `FMOD_LOOP_BIDI`: sin o con loop.
- `FMOD_2D` y `FMOD_3D`: posicionamiento 2D o 3D
- `FMOD_3D_INVERSEROLLOFF`, `FMOD_3D_LINEARROLLOFF`, `FMOD_3D_CUSTOMROLLOF`, etc
- etc

`FMOD_DEFAULT`: sonido 2D, sin loop
Pueden combinarse distintos modos (del tipo enumerado correspondiente) con el operador "`|`"

- FMOD soporta directamente 20 formatos de sonido (wav, aiff, flac, midi, mod, mpe, ogg, xm, it, etc). Y además, formato FSB, formato propio de FMOD.
- Con independencia del formato, los sonidos se cargan con `createSound` (FMOD se encarga de distinguirlos y cargarlos apropiadamente).

Reproducción de sonidos

Para reproducir hemos hecho:

```
Channel *channel;
result = system->playSound(
  sound,   // buffer que se "engancha" a ese canal
  0        // grupo de canales, 0 sin agrupar (agrupado en el master)
  false,   // arranca sin "pause" (se reproduce directamente)
  &channel); // devuelve el canal que asigna
```

- "0" indica el grupo de canales (en este caso sin agrupar) FMOD permite el agrupamiento de canales (`ChannelGroup`) en un canal virtual: permite modificar parámetros de todo el grupo de manera conjunta.
- Arrancar el sonido con *pause* lo deja disponible en memoria para:
  - modificar parámetros antes de la reproducción.
  - evitar latencias (por ejemplo, para sincronizar pistas).
- FMOD obtiene un canal libre (se podría indicar un canal concreto, pero no es habitual): devuelve un nuevo canal cada vez que se carga un sonido.

- La reproducción termina:
  - con `channel->Stop()`, o
  - cuando eln sonido termina de reproducirse (y no está en loop)

  En ambos casos: se libera el canal asociado.
- Puede pausarse la reproducción (sin liberar el canal) con `channel->setPaused(true)` y reiniciarse con `channel->setPaused(false)`
- También hay un método `channel->getPaused(&paused)`
  - Podemos hacer un método

```
void TogglePaused(FMOD::Channel* channel) {
  bool paused;
  channel->getPaused(&paused);
  channel->setPaused(!paused);
}
```

- Hay que liberar el buffer de sonido después de utilizarlo:

```
result = sound->release(); ERRCHECK(result);
```


Afinando la reproducción

Es posible comenzar la reproducción de un canal en un instante de tiempo (o sample) dado:

```
channel->setPosition(500, FMOD_TIMEUNIT_MS);
```

Comienza medio segundo después del instante inicial (también pueden utilzarse samples PCM, bytes, etc).

También podemos definir de antemano el número de repeticiones de un sonido en loop:

```
// se repite indefinidamente
channel->setLoop(-1)

// Se repite una sola vez
channel->setLoopCount(0);

// Tres veces
channel->setLoopCount(2);
```

Formatos en los buffers

Los sonidos pueden cargarse de tres modos en los buffers:

- Samples: se cargan las muestras (PCM) en memoria tal cual. Útil para los efectos de sonido que utilizan poca memoria.
- Streams: muestras grandes que ocuparían mucha memoria. Se cargan un buffer circular (como hacíamos en OpenAL). Los streams se reproducen de manera secuencial: no se puede hacer reproducción múltiple (simultáneo) del sonido de un stream. Útiles para música, pistas de voz, sonidos de ambiente, etc.
- Samples comprimidos (mp3, vorbis, etc): se dejan comprimidos en memoria. No tienen las limitaciones de los streams, ocupan menos en memoria, pero hay que descomprimir en tiempo real!

Por defecto, `System::createSound` los carga como samples (sin compresión).

- Pueden cargarse como streams con `System::createStream` (o con `System::createSound` y el valor `FMOD_CREATESTREAM`). Se puee hacer streaming incluso desde sonidos de internet.
- Pueden cargarse como samples comprimidos con `System::createSound` y el valor `FMOD_CREATECOMPRESSEDSAMPLE`

Modificando parámetros en tiempo real

- Volumen: `channel->setVolume(val);`
  `val` (float) en el intervalo [0,1] (silencio, volumen normal).
- Silencio: `channel->setMute(true);` y `channel->setMute(false);`
  Silencia el canal. Cuando se reactiva conserva el volumen que tuviese previamente.
- Modificación del volumen en sonidos multicanal. Por ejemplo, en para un sonido estéreo:

```
channel->setMixLevelsOutput(frontleft,frontright,center,...,backleft,backright);
```

- Modificación del pitch: `channel->setPicth(2.0f);`
  También existe `channel->setFrequency(rateHz);` (frecuencia de reproducción en Hz.)

Un pequeño reproductor

```
printf("[P] Pausar/Despausar\n[V/v] Subir/bajar volumen\n[Q] Salir\n");

bool paused;   float volume=1.0;
while (true) {
  if (kbhit()) {
    int key = getch();
    if ((key == 'P') || (key=='p')) {
      result=channel->getPaused(&paused);  ERRCHECK(result);
      result=channel->setPaused(!paused);  ERRCHECK(result);}
    else if (key=='V') {
      if (volume<1.0) {
        volume=volume+0.1;
        result = channel->setVolume(volume);  ERRCHECK(result);
        printf("Volume: %f\n",volume); }
    else if (key=='v'){
      if (volume>0) {
        volume=volume-0.1;
        result = channel->setVolume(volume);  ERRCHECK(result);
        printf("Volume: %f\n",volume); } }
    else if ((key=='Q') || (key=='q')) break;
  }
  result = system->update();
}
```

(simplePlayer2)