PROGRAMACIÓN DE AUDIO

Programa Avanzado en programación de videojuegos

TEMA 1: Fundamentos de la programación de audio y efectos de sonido.

Juan Mira Núñez

U-tad
CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL

Índice

- Introducción
- OpenAL
  - Buffer, source, listener
- Efectos de sonido
  - Efecto Doppler
  - EFX
    - Efecto
    - Slot
    - Filtro

Introducción

- La música y sonido de un videojuego permite dar una carga dramática al resto de elementos del juego: historia, interactividad, etc.
- Hay que dar importancia tanto a la música de nuestro juego, como al sonido ambiente, efectos...

Introducción

- Existen múltiples librerías de audio para C++: FMOD (de pago), IrrKlang (de pago), SDL_mixer (solo sonido 2D), Xaudio (Sólo Microsoft), OpenAL…
- En esta asignatura , trabajaremos con OpenAL.
- Ha sido utilizada en juegos como Bioshock, Quake 4, ColinMcRae , Prey, JediKnight…

OpenAL

- OpenAL es una librería de audio multiplataforma creada por Loki Software para portar juegos de Windows a Linux.
- Posteriormente , Creative Labs se hizo cargo de su desarrollo.
- Su diseño (convenio de nomenclatura de funciones , etc) está basado en OpenGL.

OpenAL

- Viene preinstalado o está disponible en los repositorios oficiales de las distribuciones Linux más utilizadas.
- En plataformas Apple (Mac, iOS) viene preinstalado y es la librería por defecto para el manejo de audio a bajo nivel.
- En Windows, debemos instalarnos los binarios y librerías de desarrollo

OpenAL

- Inicialmente era un proyecto de código libre, pero desde la versión 1.1, Creative cerró el código.
- No parece haber un gran esfuerzo recientemente por continuar el desarrollo de OpenAL.

OpenAL

- Ha surgido el proyecto OpenSL ES, liderado por el consorcio Khronos Group, responsables de la estandarización de OpenGL.
- Además, tras el cierre del código de OpenAL, han surgido alternativas como OpenAL Soft, que continúan siendo código libre.

  https://openal-soft.org/

OpenAL

- OpenAL permite la reproducción de sonidos en entornos tanto 2D como 3D, además de efectos de sonido como atenuación, efecto doppler, reverberación, etc.
- A pesar del estado de mantenimiento actual, sigue siendo una buena opción como librería multiplataforma.

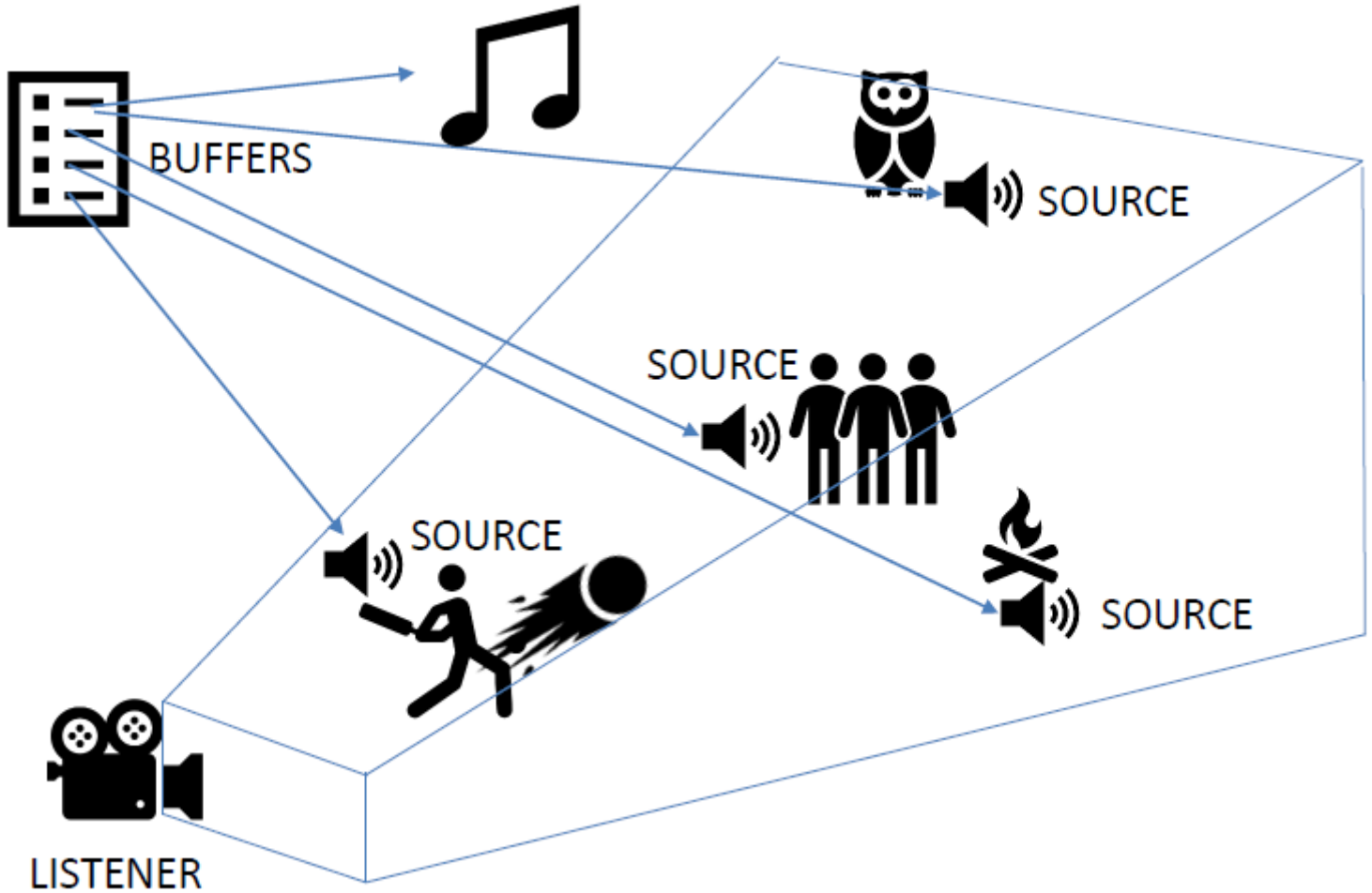
Buffer, Source y Listener

OpenAL funciona en base a 3 conceptos.

- **Buffers** donde se cargan los audios y que contienen la información de estos.
- **Sources** desde donde suenan los audios cargados en los buffers.
- **Listeners** que “escuchan” esos audios y generan el sonido que escuchamos.

Buffer, Source y Listener

OpenAL funciona en base a 3 conceptos.



Iniciando la librería

Lo primero es seleccionar el dispositivo que va a sonar. Esto lo hacemos con la función:

ALCdevice* alcOpenDevice (const ALCchar* devicename);

Ejemplo :

ALCdevice* Device = alcOpenDevice (NULL);

Con el parámetro a NULL seleccionamos el dispositivo por defecto.

Iniciando la librería

- Después hay que crear el context con ese dispositivo y hacer que sea el context actual.

ALCcontext * alcCreateContext (ALCdevice *device , ALCint * attrlist);

ALCboolean alcMakeContextCurrent (ALCcontext *context);

Ejemplo :

```
if (Device) {
ALCcontext * Context= alcCreateContext(Device,NULL);
alcMakeContextCurrent (Context);
}
```

Buffer

- Para poder reproducir sonidos, éstos deben de ser cargados desde un fichero de audio (WAV, OGG, MP3…) en un buffer de OpenAL.
- Los buffers contienen información de la onda sonora a reproducir.
- Es una estructura de datos que tiene un identificador, una etiqueta con el formato, el propio sonido, su tamaño y su frecuencia.

Buffer

Su constructor:

```
void alBufferData(
    Aluint buffer,
    Alenum format,
    const ALvoid* data,
    Alsizei size,
    Alsizei freq)
```

Buffer

void alBufferData(Aluint buffer, Alenum format, const Alvoid* data, Alsizei size, Alsizei freq)

- **buffer**: Identificador del buffer a rellenar.

- **format**: Indica el número de bits y si el buffer es mono o estéreo. Puede tomar los valores:
  - AL_FORMAT_MONO8
  - AL_FORMAT_MONO16
  - AL_FORMAT_STEREO8
  - AL_FORMAT_STEREO16

Buffer

void alBufferData(Aluint buffer, Alenum format, const Alvoid* data, Alsizei size, Alsizei freq)

- **data**: Es un puntero a los datos de audio en formato PCM. Éste es el formato utilizado en los CDs, que se utiliza por ejemplo en el formato de fichero WAV (otros formatos, como OGG o MP3, necesitan ser decodificados a PCM).
- **size**: Tamaño del buffer pasado en el parámetro anterior.
- **freq**: Frecuencia del audio.

Buffer

- Vamos a ver las funciones más importantes de OpenAL para el manejo de buffers de sonido:

Buffer

- **void alGenBuffers(ALsizei n, ALuint* buffers)**

Genera el número de buffers indicado por el parámetro n , y guarda el dentificador de cada uno en el array apuntado por el parámetro buffers

Buffer

- **void alDeleteBuffers(ALsizei n, ALuint* buffers)**

Elimina de memoria *n* buffers del array apuntado por el parámetro buffers.

Buffer

Ejemplo:

```
alGetError();
alGenBuffers(NUM_BUFFERS, g_Buffers);
if ((error = alGetError()) != AL_NO_ERROR)
{
DisplayALError("alGenBuffers:", error);
return;
}
// Cargamos el archivo de audio en data…
alBufferData(g_Buffers[0],format,data,size,freq);
```

Cargando el wav

- Hay muchas maneras de cargar el wav.
- Lo importante es conocer el tipo de cabecera del archivo wav (está su cabecera en la práctica).
- Luego es leer el fichero cómo cualquier otro fichero.

Por ejemplo podemos hacer:

```
charbuffer[4];
std::ifstream in(fn, std::ios::binary);
in.read(buffer, 4);
if (strncmp(buffer, "RIFF", 4) != 0) {
std::cout<< "this is not a valid WAVE file" << std::endl;
returnNULL; }
in.read(buffer, 4);
….
```

Source

- Una fuente (source) es un lugar de la escena desde el que se emite un buffer de sonido.
- Tiene propiedades como pitch, ganancia, bucle, posición, orientación, velocidad…
- Sus funciones más importantes son:

Source

- **void alGenSources(ALsizei n, ALuint * sources)**

Genera n fuentes de sonido , y coloca sus identificadores en el array apuntado por el parámetro sources

Source

- **void alDeleteSources(ALsizei n, ALuint * sources)**

Elimina de memoria *n* fuentes, cuyos identificadores de encuentran en el array apuntado por sources.

Creando Sources

Ejemplo

```
alGenSources(1,source);
if ((error = alGetError ()) != AL_NO_ERROR)
{
DisplayALError(“alGenSources 1 : ", error);
return;
}
```

Opciones de una source (float)

- **void alSourcef(Aluint source, Alenum param, ALfloat value)**

Establece el valor de un determinado parámetro de tipo float de la fuente indicada por *source*.

Los valores de *param* pueden ser:

- **AL_PITCH**: Velocidad de reproducción.
- **AL_GAIN**: Volumen.
- **AL_MIN_GAIN / AL_MAX_GAIN**: Volúmenes mínimo y máximo.

Opciones de una source (float)

- **void alSourcef(Aluint source, Alenum param, ALfloat value)**
  - **AL_MAX_DISTANCE**: Distancia máxima a la que es audible el sonido
  - **AL_REFERENCE_DISTANCE**: Distancia a partir de la cual el volumen disminuye de forma constante.
  - **AL_ROLLOFF_FACTOR**: Cuando la distancia es mayor que AL_REFERENCE_DISTANCE, indica la velocidad de atenuación

Opciones de una source (float)

- **void alSourcef(Aluint source, Alenum param, Alfloat value)**
  - **AL_CONE_OUTER_GAIN**: Volumen del sonido cuando el listener está fuera del cono de la fuente.
  - **AL_CONE_INNER_ANGLE / AL_CONE_OUTER_ANGLE**: Ángulos que dan forma al cono de la fuente

Opciones de una source (Vector3)

- **void alSource3f(ALuint source, ALenum param, ALfloat x, Alfloat y, ALfloat z)**

Asigna valor a parámetros de la fuente que requieran un vector de tres coordenadas. Los valores para param pueden ser:

  - **AL_POSITION**: Establece la posición de la fuente
  - **AL_DIRECTION**: Establece la dirección de la fuente
  - **AL_VELOCITY**: Establece la velocidad de movimiento de la fuente (para el efecto doppler).

Opciones de una source (int)

- **Void alSourcei(ALuint source, ALenum param, ALint value)**

Establece un parámetro de la fuente de tipo entero .

Los posibles valores de param son:

  - **AL_SOURCE_RELATIVE**: Determina si las coordenadas de la fuente son relativas al oyente o a la escena
  - **AL_LOOPING**: Indica si el sonido se debe reproducir en bucle
  - **AL_BUFFER**: Asigna un buffer a la fuente

Relacionando buffer con source

Ejemplo:

```
alSourcei (source[0], AL_BUFFER, g_Buffers [0]);
if ((error = alGetError ()) != AL_NO_ERROR)
{
DisplayALError(“alSourcei AL_BUFFER 0 : ", error);
}
```

Source

- **void alSourcePlay(ALuint source)**
- **void alSourceStop(ALuint source)**
- **void alSourcePause(ALuint source)**

Reproduce, detiene la reproducción, o pone en pausa una fuente.

Source

- **void alGetSourcei(ALuint source, ALenum param, ALint* value)**

Obtiene un parámetro de tipo entero de la fuente especificada.

Coloca el resultado en la variable apuntada por value .

Los valores de param son:

- **AL_SOURCE_RELATIVE**: Indica si la posición de la fuente es relativa al oyente o a la escena.
- **AL_BUFFER**: Devuelve el identificador del buffer asociado a la fuente.
- **AL_SOURCE_STATE**: Devuelve AL_PLAYING, AL_STOPPED, o AL_PAUSED para indicar el estado de reproducción.

Listener

- El oyente o listener representa la transformación del **oyente** dentro de la escena.
- Lo habitual es que el oyente se sitúe en las coordenadas de la cámara.
- La posición relativa de cada fuente respecto del oyente indicará el posicionamiento del sonido para su salida por los altavoces.

Sus funciones más importantes son:

Listener

- **void alListener3f(ALenum param, ALfloat x, ALfloat y, ALfloat z)**

Establece un parámetro del oyente que toma un vector de tres coordenadas como valor.

Los valores de param pueden ser:

- **AL_POSITION**: Establece la posición del oyente
- **AL_ORIENTATION**: Establece la orientación del oyente
- **AL_VELOCITY**: Establece la velocidad por segundo del oyente (para el efecto doppler).

Eliminando recursos al final

Ejemplo:

```
Context= alcGetCurrentContext();
Device= alcGetContextsDevice(Context);
alcMakeContextCurrent(NULL);
alcDestroyContext(Context);
alcCloseDevice(Device);
```

EFECTOS DE SONIDO

Efecto doppler

- El efecto doppler es un fenómeno físico producido por la distorsión de las ondas como consecuencia del movimiento de cuerpos.

- Este efecto se puede visualizar al lanzar un objeto al agua. Se crean ondas que se alejan de forma radial del origen del impacto.

Efecto doppler

Efecto doppler

- Las ondas sonoras producen el mismo efecto. Cuando un objeto emite sonido, las ondas se dispersan desde la fuente de sonido.

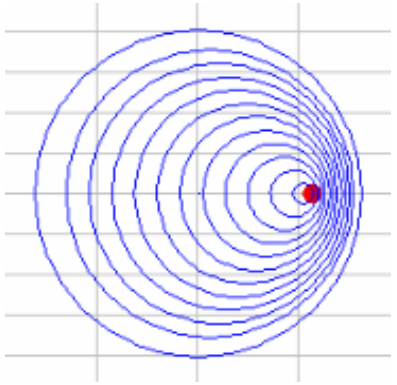
- Si el objeto que emite sonido, o el objeto que lo percibe (el oyente) están en movimiento, se produce una distorsión en la frecuencia de la onda (shifting).

Efecto doppler

- La frecuencia de la onda aumenta en la dirección en la que el emisor de sonido se desplaza.

Efecto doppler

- De igual forma, la frecuencia de la onda disminuye en la dirección opuesta al movimiento del emisor
- Este fenómeno afecta a la forma en la que el oyente percibe el sonido



Efecto doppler

- OpenAL simula el efecto doppler, utilizando el algoritmo siguiente

```
shift = DOPPLER_FACTOR * freq * (DOPPLER_VELOCITY - listener.velocity) /
(DOPPLER_VELOCITY - source.velocity)
```

- Estos parámetros se establecen con las siguientes funciones:

Efecto doppler

- **void alDopplerFactor(ALfloat factor)**

Establece el valor de DOPPLER_FACTOR. El parámetro debe ser positivo. Modificamos la magnitud de la ecuación de la siguiente forma:

- **factor = 0**: Desactiva el efecto doppler.
- **0 < factor < 1**: Disminuye el efecto doppler.
- **factor = 1**: Valor del efecto doppler por defecto
- **factor > 1**: Intensifica el efecto doppler.

Efecto doppler

- **void alDopplerVelocity(ALfloat velocity)**

Establece el valor de DOPPLER_VELOCITY , que indica la velocidad a la que se desplaza el sonido (cambia el medio en el que el sonido se desplaza -aire , agua…-).

El valor por defecto es 343.3 m/s, que corresponde a la velocidad del sonido en el aire (20º C, al 50% de humedad y a nivel del mar).

EFX

- Las tarjetas de audio soportaban la simulación de ciertos efectos sonoros mediante hardware. Las extensiones para soportar estos efectos se llaman Environmental Audio Extensions (EAX)

- Desde la llegada de Windows Vista, se desactivó el soporte para EAX por hardware, y ha caído en desuso

EFX

- No obstante, OpenAL Soft soporta un conjunto de extensiones para implementar estos efectos mediante software. A estas extensiones se les llama Effects Extension (EFX).

- ¿Qué tipo de efectos se pueden conseguir con EFX?

EFX

- Imaginemos un juego donde el personaje se mueve de espacios abiertos a espacios cerrados, se sumerge en el agua, etc.

- Si un jugador lanza una granada dentro de una casa, el sonido debe oírse de diferente manera para el que está en el interior que para el que lo oye desde fuera.

EFX

- De igual forma, el sonido no se percibe de la misma manera estando bajo el agua que en la superficie.
- Con OpenAL, podremos definir efectos de reverberación, distorsión, etc.

Veamos los elementos de OpenAL necesarios para producir estos efectos:

Efecto

- Un efecto (effect) modifica el sonido con un tipo de efecto y unos parámetros determinados. Por ejemplo, podríamos crear un efecto de reverberación, y según sus parámetros, simular los entornos de un baño, una cueva, agua…

Efecto

Un efecto se crea con la siguiente función

- **void alGenEffects(ALsizei n, ALuint * effects)**

Y se destruye con:

- **void alDeleteEffects ALsizei n, ALuint * effects)**

Efecto

El tipo de efecto a simular se establece con la función

- **void alEffecti (ALuint effect, ALenum param, ALint value)**

El valor de param debe ser **AL_EFFECT_TYPE**

Se pueden simular muchos tipos de efectos :

AL_EFFECT_REVERB, AL_EFFECT_CHORUS,

AL_EFFECT_DISTORTION, AL_EFFECT_ECHO,

AL_EFFECT_FLANGER…

Slot

- Un slot es un contenedor para un efecto. Se puede ligar una fuente de sonido a un slot determinado, y el sonido emitido por esa fuente será afectado por el efecto del slot.

Slot

Crearemos slots con la función

- **void alGenAuxiliaryEffectSlots(ALsizei n, ALuint* slots)**

Y los destruiremos con:

- **void alDeleteAuxiliaryEffectSlots(ALsizei n, ALuint * slots)**

Slot

Para fijar un efecto en el slot, haremos:

- **alAuxiliaryEffectSloti(ALuint slot, ALenum param, ALint value)**

Pasaremos **AL_EFFECTSLOT_EFFECT** como valor de param , y el efecto correspondiente como value

Slot

Es necesario ligar una fuente al slot cuyo efecto queremos aplicar . Esto se hace con la función

- **void alSource3i(ALuint source, ALenum param, ALint v1, ALint v2, ALint v3)**

El valor de param debe ser **AL_AUXILIARY_SEND_FILTER** , v1 será el slot a ligar, y v2 y v3 serán 0.

Filtro

- Un filtro permite pasar un efecto establecido por una serie de modificadores, que permiten simular por ejemplo que el sonido lo estamos escuchando a través de una pared, o por medio de un teléfono.

Filtro

Creamos un filtro con la función

- **void alGenFilters(ALsizei n, ALuint * filters);**

Y lo destruiremos con:

- **void alDeleteFilters(ALsizei n, ALuint * filters);**

Filtro

Una vez creado el filtro , indicaremos el tipo de filtrado que debe de realizar , utilizando la función

- **void alFilteri(ALuint filter, ALenum param, ALint value)**

El valor de param debe ser **AL_FILTER_TYPE** , y value puede ser AL_FILTER_NULL , AL_FILTER_LOWPASS , AL_FILTER_HIGHPASS ,

AL_FILTER_BANDPASS.

Filtro

Un filtro puede modificar la onda de un emisor directamente, o la onda que es enviada a un slot.

Para hacer lo primero, utilizaremos **alSourcei** con el parámetro **AL_DIRECT_FILTER**

Para hacer lo segundo, cuando establecemos el slot con **alSource3i** , los dos últimos parámetros deberán ser 1 y el identificador del filtro, respectivamente.

Ejemplo de uso de Effects Extension

Hay que incluir “efx.h”.

Para poder usar las funciones tenemos que asignar punteros a función, por ejemplo para geneffects:

```
LPALGENEFFECTS alGenEffects = nullptr;
alGenEffects = (LPALGENEFFECTS)alGetProcAddress(“alGenEffects”);
```

Ejemplo de uso de Effects Extension

En efx.h se puede observar todo lo disponible:

```
/* Effect object function types. */
typedef void (AL_APIENTRY *LPALGENEFFECTS)(ALsizei, ALuint*);
typedef void (AL_APIENTRY *LPALDELETEEFFECTS)(ALsizei, const ALuint*);
typedef ALboolean (AL_APIENTRY *LPALISEFFECT)(ALuint);
typedef void (AL_APIENTRY *LPALEFFECTI)(ALuint, ALenum, ALint);
typedef void (AL_APIENTRY *LPALEFFECTIV)(ALuint, ALenum, const ALint*);
typedef void (AL_APIENTRY *LPALEFFECTF)(ALuint, ALenum, ALfloat);
typedef void (AL_APIENTRY *LPALEFFECTFV)(ALuint, ALenum, const ALfloat*);
typedef void (AL_APIENTRY *LPALGETEFFECTI)(ALuint, ALenum, ALint*);
typedef void (AL_APIENTRY *LPALGETEFFECTIV)(ALuint, ALenum, ALint*);
typedef void (AL_APIENTRY *LPALGETEFFECTF)(ALuint, ALenum, ALfloat*);
typedef void (AL_APIENTRY *LPALGETEFFECTFV)(ALuint, ALenum, ALfloat*);
```

```
#ifdef AL_ALEXT_PROTOTYPES
AL_API ALvoid AL_APIENTRY alGenEffects(ALsizei n, ALuint *effects);
AL_API ALvoid AL_APIENTRY alDeleteEffects(ALsizei n, const ALuint *effects);
AL_API ALboolean AL_APIENTRY alIsEffect(ALuint effect);
AL_API ALvoid AL_APIENTRY alEffecti(ALuint effect, ALenum param, ALint iValue);
AL_API ALvoid AL_APIENTRY alEffectiv(ALuint effect, ALenum param, const ALint *piValues);
AL_API ALvoid AL_APIENTRY alEffectf(ALuint effect, ALenum param, ALfloat flValue);
AL_API ALvoid AL_APIENTRY alEffectfv(ALuint effect, ALenum param, const ALfloat *pflValues);
AL_API ALvoid AL_APIENTRY alGetEffecti(ALuint effect, ALenum param, ALint *piValue);
AL_API ALvoid AL_APIENTRY alGetEffectiv(ALuint effect, ALenum param, ALint *piValues);
AL_API ALvoid AL_APIENTRY alGetEffectf(ALuint effect, ALenum param, ALfloat *pflValue);
AL_API ALvoid AL_APIENTRY alGetEffectfv(ALuint effect, ALenum param, ALfloat *pflValues);
```

¿Dudas?