PROGRAMACIÓN DE AUDIO

Máster en Programación de
Videojuegos

TEMA 1: Fundamentos de la programación de audio.

Javier Alegre Landáburu

Contenido

- Introducción
- OpenAL
  - Buffer, source, listener

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 (http://kcat.strangesoft.net/openal.html).

OpenAL

- OpenAL permite la reproducción de sonidos en entornos tanto 2D como 3D, además de efectos de sonido como atenuación, efecto doppler, reverberación, etc.

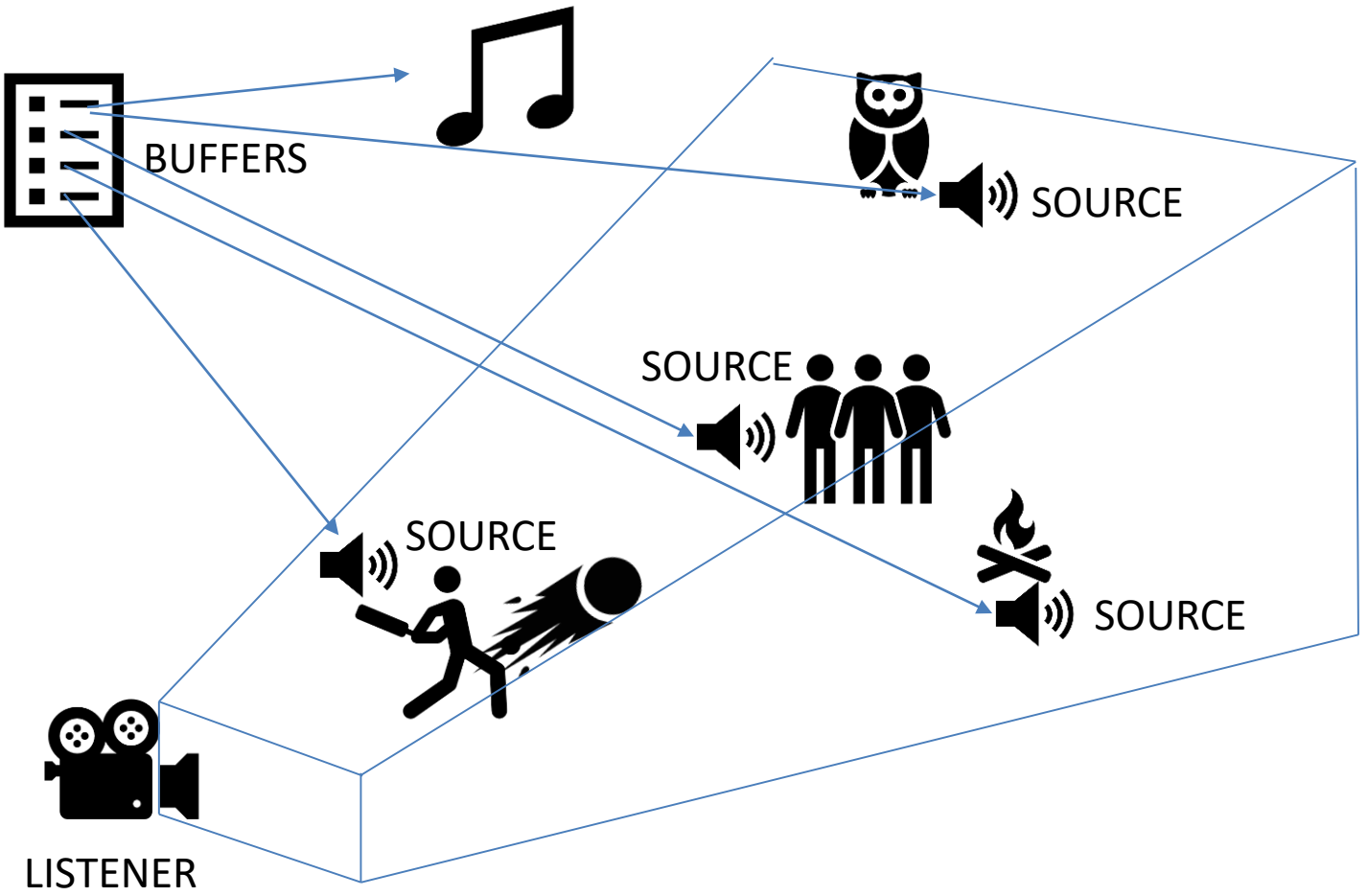
- A pesar del estado de mantenimiento actual, sigue siendo una buena opción como librería multiplataforma.

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

- **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

- **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 identificador 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:

```
char buffer[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;
return NULL; }
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 se 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)

- 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)

- 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:

Opciones de una Source (Vector3)

- 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:

Listener

- 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);
```