Programación de Audio

Práctica 1: Reproducción de buffers

En esta práctica, vamos a aprender a cargar buffers de audio desde un fichero WAV, a crear fuentes de sonido desde las que reproducir el buffer, y a configurar el oyente. Trabajaremos sobre el motor que se desarrolló en la asignatura Programación 2D.

Inicialización

Antes de utilizar audio en nuestro juego, debemos inicializar el sistema de audio. Se debe crear un dispositivo y un contexto de OpenAL, utilizando las funciones alcOpenDevice (con parámetro nullptr) y alcCreateContext (con el dispositivo creado en la llamada anterior y nullptr como parámetros, respectivamente). A continuación, estableceremos el contexto creado como activo con alcMakeContextCurrent.
Al salir de la aplicación, se deben eliminar el contexto y el dispositivo (en ese orden) con las funciones alcDestroyContext y alcCloseDevice.

Clase AudioBuffer

Implementaremos una clase para el manejo de buffers de sonido, con los siguientes métodos (además de las variables miembro, constructores y destructor que estimemos necesarios):

- `static AudioBuffer* load(const char* filename);`
- `uint32_t getAlBuffer() const;`

En el método load, obtendremos del fichero WAV los datos necesarios para cargar la fuente de sonido. Si el fichero de audio no se puede abrir o está mal formado, devolveremos nullptr. En caso contrario, devolveremos un puntero a un nuevo objeto AudioBuffer.

Para poder hacer esto, es necesario conocer la estructura de un fichero WAV. En primer lugar, encontramos una cabecera con los siguientes datos:

Dato	Bytes
ChunkID (“RIFF”)	4
RiffChunkSize	4
Format(“wave”)	4
SubChunkID (“fmt”)	4
FmtChunkSize	4
AudioFormat	2
Channels	2
SampleRate	4
ByteRate	4
BlockAlign	2
BitsPerSample	2
ExtraParamsSize	2
ExtraParams	X

Los dos últimos elementos sólo aparecen dependiendo del valor de AudioFormat. Si es un fichero PCM no comprimido (AudioFormat = 1), estos campos no están presentes. Con otro valor, puede estar presente o puede no estarlo. Podemos saber si debemos leer estos valores o no de la siguiente forma: Si el valor que hemos leído en FmtChunkSize es 16, entonces estos campos no están presentes. Si es mayor de 16, entonces debemos leer el valor de ExtraParamsSize y saltarnos ese número de bytes.

A continuación, debemos leer la cabecera de los siguientes bloques de datos (4 bytes), buscando uno que coincida con el string “**data**”. Mientras el bloque no tenga ese identificador, nos lo saltaremos leyendo su tamaño (4 bytes) y saltándonos el número indicado de bytes.

Una vez encontrado el bloque “**data**”, leeremos su tamaño (4 bytes), reservaremos memoria para ese tamaño, y leeremos ese número de bytes dentro de la memoria reservada.

Podemos encontrar más información sobre el formato WAV en el siguiente enlace: http://www.sonicspot.com/guide/wavefiles.html.

Una vez hecho esto, generaremos el buffer de OpenAL (que almacenaremos en una variable miembro y podremos devolver con la buffer getAlBuffer) y, con la función alBufferData, rellenaremos la información de dicho buffer, con los siguientes parámetros:

- **buffer**: El identificador del buffer generado en el paso anterior.
- **format**: Dependiendo del valor leído en BitsPerSample:
    - **BitsPerSample=8**: Si el valor de Channels leído del fichero es 1, el formato será AL_FORMAT_MONO8. En caso contrario, será AL_FORMAT_STEREO8.
    - **BitsPerSample=16**: Si el valor de Channels leído del fichero es 1, el formato será AL_FORMAT_MONO16. En caso contrario, será AL_FORMAT_STEREO16.

- **data**: El puntero a la memoria reservada donde hemos cargado el bloque "**data**".
- **size**: El tamaño del bloque "data".
- **freq**: El valor SampleRate leído del fichero.

Clase AudioSource

También necesitaremos una clase AudioSource que represente una fuente de sonido. Tendrá los siguientes métodos (además de las variables miembro necesarias y destructor si lo estimamos oportuno):

- `AudioSource(AudioBuffer* buffer);`
- `void setPitch(float pitch);`
- `void setGain(float gain);`
- `void setLooping(bool loop);`
- `void setPosition(float x, float y, float z);`
- `void setVelocity(float x, float y, float z);`
- `void play();`
- `void stop();`
- `void pause();`
- `bool isPlaying() const;`

En el constructor, crearemos la fuente de OpenAL, y llamaremos a los métodos que establecen los valores de la fuente pasándole valores por defecto (sin reproducción en bucle, pitch y gain valdrán 1, y la posición y velocidad serán 0 en todas sus coordenadas). Por último, indicaremos el buffer a utilizar con alSourcei.
El resto de métodos llamarán a las funciones apropiadas de OpenAL según aparece en las diapositivas del tema 1.

Listener

Tenemos que crear un Listener que represente a nuestro oyente. Como sólo hay un oyente presente en el mundo, lo podemos representar mediante una clase singleton o, en nuestro caso, utilizaremos simplemente funciones globales. Su interfaz es la siguiente:

- `void setListenerPosition(float x, float y, float z);`
- `void setListenerVelocity(float x, float y, float z);`
- `void setListenerOrientation(float x, float y, float z);`

Esta clase sigue el patrón singleton. En el constructor, llamaremos a los métodos que establecen las propiedades del oyente, pasándole 0 en todas ellas. Los otros métodos se implementarán llamando a las funciones de OpenAL de acuerdo con la información de las diapositivas del tema 1.

Ejercicio

Una vez hecho todo esto, probaremos que la reproducción de audio funciona cargando un buffer con el fichero **"data/music.wav"**. Generaremos una fuente de sonido, y reproduciremos el buffer generado.

Debemos abrir una ventana con GLFW, para poder recibir entrada del teclado. Cuando se pulsen las teclas de cursor, se debe hacer lo siguiente:

- Arriba: Aumentamos el pitch.
- Abajo: Disminuimos el pitch.
- Izquierda: La fuente se desplaza a la izquierda.
- Derecha: La fuente se desplaza a la derecha.