Sonido en videojuegos

Jaime Sánchez Hernández

Departamento de Sistemas Informáticos y Computación, UCM

Curso 18/19

4. Librerías de renderizado de audio 3D

Distintas posibilidades

Libres:

- OpenAL (de código abierto): la mantiene Creative Labs.
  Implementada en C, pero se han creado algunos wrappers sobre C++ (OpenAL++), Python, Haskell.
  Muy bien documentada.
- FMOD (libre para proyectos no comerciales).
  Fácil de usar, muy completa, buen rendimiento, buena documentación, muy actualizada.
- Otras: Wwise (AudioKinetic), (Audiere, DirectX)

Comerciales: BASS, X-Audio, Miles...

OpenaAL (Open Audio Library)

http://www.openal.org/ (librería, ejemplos, documentación, links...)
Otra distro: OpenAl Soft https://github.com/kcat/openal-soft

Pequeña intro: https://es.wikipedia.org/wiki/OpenAL

- Desarrollada y mantenida por Loki Games (desaparecida en 2002), mantenida actualmente por Creative Labs.
  Su objetivo fue establecerse como un estandard multiplataforma (Microsoft DirectX(3D) y otros, como EAX estaban diseñados solo para windows).
- Licencia GPL.
- Implementada en C (hay algunos wrappers para C++, Python, Haskell, etc)
- Soporta DirectX y EAX.
- Utilizada en Doom3, Quake4, Battlefield 2, Race Driver GRID, Metal Gear 2.
  Motor de Unreal, programa de modelado Blender3D

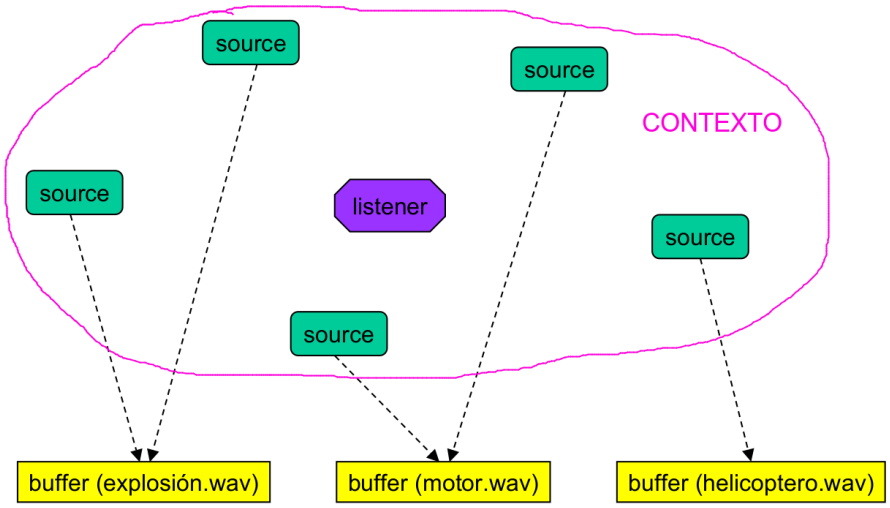
OpenAl. Descripción

- El API de OpenAL sigue la misma filosofía que OpenGL. . . dicen que “uno de los mejores hecho nunca”.
  Es una librería que sigue la filosofía de orientación a objetos, aunque no implementa clases
- Documentación (accesible desde su web):
  - OpenAL Programmer’s Guide: es la referencia básica para consultar la especificación de las funciones disponibles.
  - OpenAL 1.1 Specification and Reference: detalles de implementación de la librería.
    Documentación del alut.h (actualización distribuida aparte).

Los “objetos” de OpenAL

- OpenAL maneja 3 tipos de objetos básicos:
  - Buffers: almacenan los sonidos. Las muestras deben estar en formato PCM (pulse code modulation)
  - Sources: puntos en el espacio que emiten los sonidos. Las propiedades pueden modificarse dinámicamente (en tiempo de ejecución), en particular la posición y velocidad.
  - Listener: posición, orientación y velocidad del oyente. También se pueden modificar su propiedades dinámicamente
- Varios sources pueden utilizar los mismos buffers, pero no es posible asignar varios buffers al mismo source.
  - La lógica: un source representa un sonido que proviene de una posición en el espacio. Si se quieren varios sonidos procedentes de la misma posición habrá que poner varias fuentes en esa posición.
  - Puede ser útil asociar el mismo sonido varias sources, por ejemplo cuando tenemos varias entidades de la misma naturaleza, pero en distintas posiciones en la escena.

El mundo OpenAL



Los .wav en formato mono!

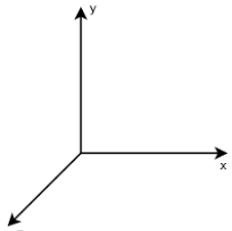
Convenciones en OpenAL

Unidades:

- Tiempo: segundos
- Frecuencia: hertzios
- Distancia: no hay unidades!!

  El programador puede trabajar libremente en metros, cm, o lo que quiera (si introduce/modifica algún parámetro que involucre unidades, como la velocidad de propagación del sonido en el aire, entonces todas las distancias en la misma unidad)

Sistema de referencia (coordenadas cartesianas):



Filosofía del API

El API permite modificar las propiedades (posición, velocidad,...) de los "objetos" listener, source, etc.

En vez de utilizar múltiples funciones de acceso del estilo

```
source.setSourcePosition(...)   source.setSourceOrientation(...)
```

proporciona una única función

```
alSourcefv(...)
```

que modifica las propiedades del "objeto" `source`. Por ejemplo:

```
alSourcefv(src, AL_POSITION, pos)
```

- `src`: identificador (handle) del source
- `AL_POSITION`: propiedad a cambiar
- `pos`: valor que se asigna a esa propiedad (vector con tres ALfloat)
- `fv`: indica que la función recibe un vector de float

Todas las funciones y los tipos empiezan por al, AL (o alut)

Capas del API

OpenAL también sigue la misma filosofía que OpenGL en las capas del API, teniendo la posibilidad de trabajar a más alto o más bajo nivel:

- gl.h → al.h (audio library)
- glu.h → ~~alu.h (audio library utility)~~ deprecated
- glut.h → alut.h (audio library utility toolkit)
  (tiene una versión actualizada que se distribuye por separado, junto con su documentación)

Incorpora además: alc.h (audio library utility context)

Preparando el proyecto OpenAL

Includes de OpenAL:

```
#include <path>\al.h
//#include <path>\alu.h   deprecated
#include <path>\alut.h
#include <path>\alc.h
```

Enlazar las librerías (en MSVC → opciones de proyecto → vinculador → entrada):

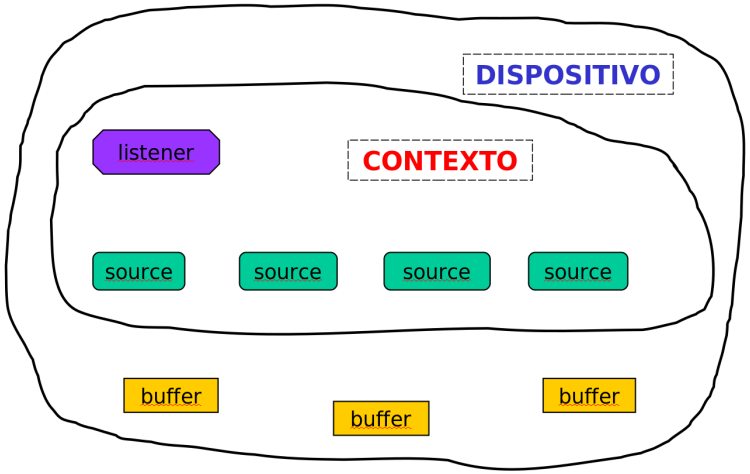
```
<path>\OpenAL32.lib
<path>\ALut.lib
```

Inicializando OpenAL

Necesitamos:

- Un dispositivo (device): tarjeta de sonido (puede haber varias)
- Un contexto (context): conjunto de datos que representan el estado de OpenAL. Cada contexto representa:
  - un listener y
  - múltiples sources (los buffers no forman parte del contexto).

  Es posible mantener varios contextos e intercambiarlos entre sí, pero solo puede haber un contexto activo en cada momento. En la mayoría de los casos esto no se utilizará, pero puede tener aplicación: por ejemplo, en un juego donde la acción puede trasladarse instantáneamente a dos (o más) escenas distintas (para cada una de ellas puede haber un contexto sonoro diferente). Se pueden suspender contextos (eso puede tener otras utilidades).



Inicializando OpenAL

Podemos inicializar OpenAL:

- de forma "fácil" con ALUT, dejando que seleccione el dispositivo y la configuración, y creando el contexto actual por nosotros. Selecciona el "mejor" dispositivo disponible (preferred).

```
alutInint(0,NULL);
alGetError();  //limpiamos el bit de error
```

- a más bajo nivel con ALC, especificando el dispositivo y creando el contexto.

```
ALCdevice *Device = alcOpenDevice((Alubyte*) ``DirectSound3D'');
if (Device==NULL) exit(-1);

// creamos el contexto manualmente con los flags de configuracion
// (NULL suele ser suficiente)
ALCcontext *Context = alcCreateContext(Device,NULL);
alcMakeContextCurrent(Context);
alGetError();  //limpiamos el bit de error
```


Con la inicialización anterior hemos especificado "DirectSound3D" como dispositivo pero con ALC también es posible dejar que OpenAL seleccione el mejor (preferred) dispositivo por nosotros, pasándole el parámetro NULL:

```
ALCdevice *Device = alcOpenDevice(NULL);
//... el resto igual
```

Creación y carga de un buffer

Con `al`:

```
// creamos buffer
ALuint Buffer;
alGenBuffers(1, &Buffer);  //num de buffers y puntero a ellos
if(alGetError() != AL_NO_ERROR) exit(-1);

// variables de retorno de la apertura de un wav
ALenum format; ALsizei size; ALvoid* data; ALsizei freq; ALboolean loop;

// cargamos datos (en un espacio intermedio, una especie de clipboard)
alutLoadWAVFile("wavdata/holaMundo.wav",
                &format, &data, &size, &freq, &loop);

// los pasamos al buffer
alBufferData(Buffer, format, data, size, freq);

// descargamos datos del almacenamiento intermedio
alutUnloadWAV(format, data, size, freq);
```

Con `alut`:

```
ALuint Buffer = alutCreateBufferFromFile("wavdata/holaMundo.wav");
```


Hola mundo

```
#include <stdlib.h>
#include <AL/al.h>
#include <AL/alut.h>
#include <AL/alc.h>

int main () {
  // inicializamos
  alutInit(0,NULL);
  alGetError(); //limpiamos el bit de error

  // creamos buffer
  ALuint Buffer = alutCreateBufferFromFile("../wavdata/holaMundo.wav");
  if(alGetError() != AL_NO_ERROR) exit(-1);

  // creamos source
  ALuint Source;
  alGenSources (1, &Source);

  // enganchamos source al buffer
  alSourcei (Source, AL_BUFFER, Buffer);

  // reproducimos
  alSourcePlay (Source);

  // pausa para escuchar, liberacion de recursos y salida
  alutSleep (3);
  alutExit ();
  return EXIT_SUCCESS;
}
```


Posicionando el *listener*

```
// vector de posicion del listener
ALfloat ListenerPos[] = { 0.0, 0.0, 0.0 };

// vector de velocidad del listener
ALfloat ListenerVel[] = { 0.0, 0.0, 0.0 };

// Orientacion, mirando a la pantalla (los primeros 3 elementos
// son "at", los otros "up"). Todos los valores deben ser 1 o -1
ALfloat ListenerOri[] = { 0.0, 0.0, -1.0,  0.0, 1.0, 0.0 };
//                        ----- at -----   ----- up ----

alListenerfv(AL_POSITION,    ListenerPos);
alListenerfv(AL_VELOCITY,    ListenerVel);
alListenerfv(AL_ORIENTATION, ListenerOri);
```


Posicionando el *source*

```
// creamos un source
alGenSources(1, &Source); //num de sources, puntero a ellas

// posicion y velociddad: vectores en coordenadas cartesianas
ALfloat SourcePos[] = { 0.0, 0.0, 0.0 };
ALfloat SourceVel[] = { 0.0, 0.0, 0.0 };

// enganchamos el buffer
...

// propiedades del source
alSourcefv(Source, AL_POSITION, SourcePos);
alSourcefv(Source, AL_VELOCITY, SourceVel);
alSourcef (Source, AL_PITCH,    1.0 ); // frec de reproduccion
alSourcef (Source, AL_GAIN,     1.0 ); // ganancia de la fuente
alSourcei (Source, AL_LOOPING,  true); // para repr. ciclica
```

Un reproductor simple

```
ALubyte c = ' ';
while(c != 'q') {
  c = getche();
  switch(c) {
    // Pulsando 'p' empiza la reproduccion del sample
    case 'p': alSourcePlay(Source); break;
    // Con 's' se para la reproduccion
    case 's': alSourceStop(Source); break;
    // Con 'h' se pausa. Con 'p' se reinicia
    case 'h': alSourcePause(Source); break;
  };

  // eliminacion de buffers, sources y exit
  alDeleteBuffers(1, &Buffer);
  alDeleteSources(1, &Source);
  alutExit();
}
```

(reproductor)

Información sobre el wav

```
ALint frecuencia, bitsRes, canales, tamanio;

alGetBufferi(Buffer,AL_FREQUENCY, &frecuencia);
alGetBufferi(Buffer,AL_BITS, &bitsRes);
alGetBufferi(Buffer,AL_CHANNELS, &canales);
alGetBufferi(Buffer,AL_SIZE, &tamanio);

printf("\n\nFormato del wav:\n");
printf("Frec:     %i\n",frecuencia);
printf("BitsRes:  %i\n",bitsRes);
printf("Canales:  %i\n",canales);
printf("Tamanio:   %i\n",tamanio);
```


Selección del dispositivo de salida

`alut` es la opción más cómoda, pero `alc` ofrece más control. Detección y selección de dispositivo de salida (de audio). Con la llamada:

```
(char *) alcGetString(NULL, ALC_DEVICE_SPECIFIER);
```

Obtener la lista de los dispositivos disponibles y utilizar el que queramos.

```
if (alcIsExtensionPresent(NULL,(ALubyte*)"ALC_ENUMERATION_EXT")==AL_TRUE) {
  // preguntamos por los dispositivos de salida disponibles
  char *deviceList = (char *)alcGetString(NULL, ALC_DEVICE_SPECIFIER);

  // sacamos en pantalla las posibilidades
  int i=0;
  while ((deviceList[i]!=0) || (deviceList[i+1]!=0)) {
    printf("%c",deviceList[i]); i++;
  }

  // podemos utilizar cualquiera de estos dispositivos o el de por def.
  // ALCdevice *Device = (char *)alcGetString(NULL, ALC_DEFAULT_DEVICE_SPECIFIER);

  Device=...;
  alcOpenDevice(Device);
  // Creacion del contexto
  // ... etc
```

(dispositivos)

Convenciones en OpenAL

OpenAL proporciona los tipos habituales, pero con el prefijo “AL”:

`AL{ boolean | char | byte | ubyte | short | ushort | int | uint | enum | float | double | sizei }` (como uint de 32 bits)

Códigos de error (bastante explícitos):

`AL_NO_ERROR | AL_INVALID_NAME | AL_INVALID_ENUM | AL_INVALID_VALUE | AL_INVALID_OPERATION | AL_OUT_OF_MEMORY`

Importante: detectar si se produce error después de cada operación susceptible de producirlo (y limpiar el bit de error antes de hacer operaciones críticas, para chequear adecuadamente los posibles errores).

Creación de objetos:

```
void alGen{ Buffers | Sources }(ALsizei n, ALuint *nombreObjs);
```

Liberación de recursos:

```
void alDelete{ Buffers | Sources }(ALsizei n, ALuint *nombreObjs);
```

Modificación de atributos de objetos

Las funciones de OpenAL siguen la siguiente convención:

```
void al{Objeto}{n}{i|f}{v}({ALuint obj,} ALenum param, T valores);
```

donde:

- `Objeto` es el nombre del objeto (buffer, listener, source)
- `n` indica el número de valores que se pasan (se omite si es solo uno)
- `i|f` indica si el tipo de los valores es integer "i" o float "f"
- `v` indica que se pasa un vector del tipo anterior ("i" o "f")
- `{ALuint obj}` (si aparece) indica el objeto al que afecta esta llamada
- `ALenum param` indica el atributo del objeto a modificar
- `T valores` son los valores concretos que se pasan (de acuerdo con los tipos indicados por "i|f" y "v")

Ejemplos:

```
   alSourcef(Source, AL_PITCH,    1.0 );
alListenerfv(        AL_POSITION, ListenerPos);
```


Consulta de atributos

Para obtener atributos de los objetos:

```
void alGet{Objeto}{n}{i|f}{v}(ALuint obj, ALenum param, T * valor);
```

Para el estado global de openAL (no para objetos concretos): Para valores simples:

```
AL{Tipo} alGet{Tipo}(ALenum param);
```

Para vectores:

```
void alGet{Tipo}{v}(ALenum param, ALtipo * valor);
```

La librería `alc` también proporciona una serie de funciones `alc{...}` de consulta/manipulación del contexto, con la misma filosofía y sintaxis.

Echando a andar

```
// posicion y velocidad de la fuente
ALfloat SourcePos[] = { 0.0, 0.0, 0.0 };
ALfloat SourceVel[] = { 0.0f, 0.0f, 0.1f };
...

// una forma de inventarse un renderizado
ALint time = 0,  elapse = 0;
while(!kbhit()) {
  elapse += clock() - time;
  time += elapse;
  gotoxy(0,10);
  printf("Elapse: %i  time: %i",elapse,time); //salida
  if (elapse > 50){
    // reseteamos
    elapse = 0;
    // SourcePos[0] += SourceVel[0];
    // SourcePos[1] += SourceVel[1]
    SourcePos[2] += SourceVel[2];  // este es el que cambia, la Z

    // actualizamos la posicion del source
    alSourcefv(Source, AL_POSITION, SourcePos); }
}
```

(sirena)