UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Reproducción de ficheros MIDI con OpenAL

Apellidos, nombre	Agustí i Melchor, Manuel (magusti@disca.upv.es)
Departamento	Departamento de Informática de Sistemas y Computadores (DISCA)
Centro	Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València

1 Resumen de las ideas clave

OpenAL es [1] un motor de audio 3D capaz de renderizar el sonido que escucha un oyente al recrear una escena sonora tridimensional, de modo que el usuario se "vea" envuelto entre las fuentes de sonido dispuestas a su alrededor. OpenAL no se encargará de cargar audio de un fichero en disco, sino de enlazarlo a una fuente de sonido, cuando ya está cargado en memoria y sin compresión. Esta ha sido una decisión de diseño [2] y [3].

Si se quiere ampliar el conjunto de ficheros de los que OpenAL puede importar el audio se puede recurrir a librerías específicas. De manera que se puede ampliar la operativa, al tiempo que limitar el peso de este componente en una aplicación, a solo los formatos que se sepa que se van a utilizar. En este artículo se verá el uso del formato MIDI (*Musical Instrument Digital Interface)* de audio [4] y de las funciones de la librería *libwildmidi [5]* para determinar las propiedades del audio contenido en un fichero, extraer los datos de audio que este contiene y llevarlos a memoria en un formato que sea compatible con las estructuras de OpenAL para que este los reproduzca.

2 Objetivos

A partir del estudio del ejemplos que se aborda en este documento, el lector será capaz de:

- Exponer la estrategia para incorporar el uso del formato de audio MIDI en OpenAL.
- Instalar y compilar una aplicación que hace uso del formato MIDI sobre OpenAL, con las funciones de la biblioteca *libwildmidi*.
- Identificar una posible secuencia de instrucciones que operan sobre un fichero MIDI y recodificarlo como audio en forma de ondas, de forma que OpenAL pueda utilizarlo.

El presente documento está encaminado a ofrecer una perspectiva inicial de cómo ampliar el conjunto de formatos de ficheros que puede utilizar el motor de audio 3D OpenAL. No es el objetivo de este documento dar una solución global para todos los formatos, ni adentrarse en el detalle de cómo es MIDI por dentro, sino mostrar cómo incorporar este formato al repertorio de archivos que puede utilizar una aplicación basada en OpenAL.

Para plantear la solución, es necesario revisar la relación entre audio digital, su uso en un motor de audio como OpenAL, las peculiaridades de MIDI en cuanto a su forma de describir el sonido y cómo operar sobre el formato de fichero asociado a MIDI.

3 Introducción

MIDI es [4] un estándar de audio para el contexto musical. Esto es, describe un protocolo que sirve la conectar instrumentos musicales y también computadores, tabletas y teléfonos, de diferentes compañías. Todos ellos tienen en común ser digitales y tener una cierta capacidad de proceso de la información digital. Con MIDI se realizan [6] labores de creación, interpretación, aprendizaje y distribución de composiciones. La diferencia con los formatos de forma de onda típicos (p. ej. los archivos .WAVE o MP3) de los ficheros de sonido es que MIDI recoge la secuencia de eventos que representan la interpretación de la música. MIDI no transmite señales de audio. MIDI es únicamente datos: un

conjunto de instrucciones que las máquinas usan para “hablar” entre ellas, en el contexto de la interpretación musical.

Con la llegada del ordenador al mundo de la música [8] se populariza el uso de archivos MIDI”. Estos son, básicamente, como las partituras de los músicos. Los archivos MIDI, a pesar de toda la información numérica que contienen, por regla general, ocupan poco espacio con respecto a los archivos de audio en forma de onda (incluso aunque estos últimos suelen utilizar compresión). El motivo es que un archivo MIDI no guarda la información sonora, sino la información (los eventos) que permite recrearla. El sonido final será sintetizado (generado) a través del banco de sonidos de una tarjeta de audio o de un dispositivo MIDI externo conectado al ordenador. Así que la calidad de la interpretación de un archivo MIDI puede tener muy distintos resultados en función de la capacidad del dispositivo que "haga sonar" su contenido. La gran ventaja de los archivos MIDI con respecto a los ficheros en forma de onda es que pueden editarse hasta el más mínimo detalle. Es decir, un archivo MIDI nos permite cambiar los instrumentos, cambiar la velocidad de interpretación, el compás, el volumen, la altura, la duración, etc. Y todo esto con una precisión que llega hasta la especificación de una nota musical.

OpenAL trabaja [9] sobre el audio digitalizado en forma de ondas en *buffers* de memoria y los asocia a las fuentes (*sources*), que son las encargadas de posicionar el sonido en el espacio y aplicarle efectos. Es labor del desarrollador de una aplicación cargar el sonido a partir de, p. ej., el acceso a ficheros de audio, el uso del micrófono, la síntesis de voz o el uso de un dispositivo MIDI.

Para poder hacer sonar un fichero MIDI se necesita conocer el estándar [4] en profundidad. Así que no solo es cuestión de leer el contenido del fichero, sino que, además, hay que generar la información de audio descrita por los eventos que están contenido en el archivo MIDI. Por ello se va a revisar la operativa de OpenAL y cómo unirla con la extracción del sonido descrito por MIDI. Como desarrolladores debemos empezar por valorar las soluciones existentes para estos menesteres y, puesto que existen librerías que lo hacen con calidad, vamos a utilizar aquí libwildmidi como una posible opción para ello.

3.1 Reproducción y formatos de fichero de audio en OpenAL

OpenAL es capaz de reproducir audio, en PCM, desde memoria, pudiendo trabajar en modo de precarga o de reproducción en continuo (*streaming*). La precarga supone llevar a memoria, por completo, el contenido del fichero de audio, para después reproducirlo. La fig. 1 muestra de forma gráfica como la librería *Audio Library Utility Toolkit* (ALUT) [7] complementa a OpenAL, en tanto que es capaz de leer archivos WAVE de disco (con la llamada al sistema *read*) y extraer de ese contenido el audio en PCM para asociarlo (con *alutCreateBufferFromFile*) a un *buffer* de OpenAL.

Cuando el formato del fichero no es reconocido por ALUT, como p. ej. el caso de MIDI, es necesario entender el formato del fichero para ser capaz de extraer el audio. Para ello es necesario conocer la especificación a fondo de MIDI. La Figura 2 resume, de forma gráfica, los cambios respecto a la fig. Figura 1. En este caso, el uso de la librería *libwildmidi* permite generar el audio (asociado a la descripción de la interpretación codificada en MIDI) en PCM para asignárselo a un *buffer* de OpenAL a través de la instrucción *alBufferData.* En cualquier caso la asociación es estática (precarga), esto es, se puede cargar totalmente el audio en memoria para asignarlo un *buffer*. Cuando el fichero es grande, esta solución es muy costosa en términos de ocupación de recursos.

La segunda aproximación es cargar dinámicamente el audio: esto es, tener en memoria los datos necesarios para tener sonido durante un tiempo, mientras se

carga otro “trozo” de audio. Es lo que se conoce como reproducción en continuo o *streaming* [1]. OpenAL no proporciona la funcionalidad de *streaming* sobre un objeto [3], sino que proporciona un mecanismo más general de asociación de una pila de *buffers* a una fuente. Esto permite que la aplicación pueda especificar cuántos, de qué tamaño y si se reutilizan o se descartan tras su uso. Así se consigue maximizar la eficiencia de uso del recurso.

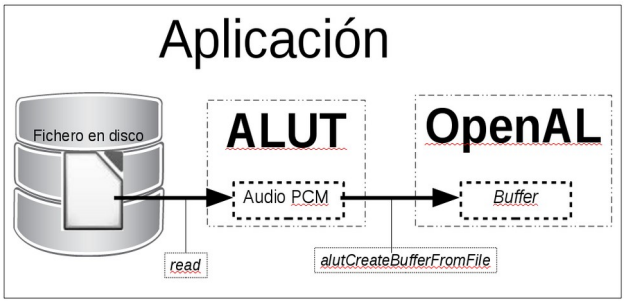

Figura 1: Esquema básico de gestión de formatos de ficheros en OpenAL con ALUT.

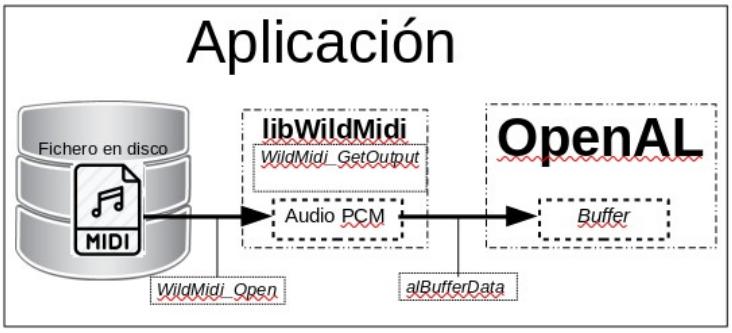

Figura 2: Esquema ampliado de la gestión de formatos de ficheros para OpenAL con librerías externas: en este caso libWildMIDI.

La de *streaming* está dirigida a grandes cantidades de sonido y permite asociar varios trozos de sonido a una fuente, de manera que la reproducción del sonido dependa que exista una serie de *buffers*, sobre los que se extrae el audio l aparte de la aplicación que entiende el formato del fichero (en nuestro caso *libWildmidi* con operaciones como *WildMidi_Open* y *WildMidi_GetOutput*) y de los que va a ser obtenido el audio conforme OpenAL vaya necesitándolo para la reproducción del audio.

4 Desarrollo

Vamos a revisar un ejemplo completo de uso de libWildMidi y OpenAL para ilustrar de forma práctica el uso de MIDI como extensión a OpenAL. Está basado en WildMIDI [11], este es un reproductor (con interfaz en modo texto), creado para ilustrar el uso de la librería libWilMidi.

De este ejemplo hemos extraído la parte que trabaja con OpenAL (como manejador, esto es, como interfaz con el hardware de audio). Por ese motivo se incluye todo el código en este artículo y se ha optado por mantener los comentarios originales como parte del reconocimiento a la referencia de donde se ha extraído este código. Veamos con este ejemplo parte del API de *libWildMidi* para hacernos una idea de cómo se puede utilizar MIDI en nuestra aplicación .

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #ifndef __APPLE__
4. #include <al.h>
5. #include <alc.h>
6. #else
7. #include <OpenAL/al.h>
8. #include <OpenAL/alc.h>
9. #endif
10.#include "wildmidi_lib.h"
11.
12.#define NUM_BUFFERS 4
13.static ALCdevice *device;
14.static ALCcontext *context;
15.static ALuint sourceId = 0, buffers[NUM_BUFFERS], frames = 0;
16.static char config_file[1024];
17.
18.static int write_openal_output(int8_t *output_data, int output_size);
19.static void close_openal_output(void);
20.static int open_openal_output(void);
21.
22.int main(int argc, char **argv) {
23.     static unsigned int rate = 32072;
24.     struct _WM_Info *wm_info;
25.     int i, res = 0;
26.     uint16_t mixer_options = 0;
27.     void *midi_ptr = 0;
28.     uint8_t master_volume = 100, *output_buffer;
29.     uint32_t perc_play, pro_mins, pro_secs, apr_mins, apr_secs, samples;
30.     static char spinner[] = "|/-\\";
31.     static int spinpoint = 0;
32.     char * ret_err = NULL;
33.     long libraryver;
34.
35.     if (argc == 0) {
36.         fprintf(stderr, "ERROR: No midi file given\r\nUtilitzar: %s
nomFitxer.mid", argv[0]); return( 1 );
37.     }
...
```

Listado 1: Listado de wildmidi_openal (parte 1)

```
...
38.     printf("Initializing Sound System\n");
39.     if (open_openal_output() == -1) {
40.         printf("No es pot obrir OpenAL\n"); return (1);
41.     }
42.     printf("libWildMidi %ld.%ld.%ld\n", LIBWILDMIDI_VER_MAJOR,
LIBWILDMIDI_VER_MINOR, LIBWILDMIDI_VER_MICRO);
43.     if (WildMidi_Init("/etc/wildmidi/wildmidi.cfg",
44.                       rate, mixer_options) == -1) {
45.         fprintf(stderr, "%s\r\n", WildMidi_GetError());
46.         WildMidi_ClearError(); return (1);
47.     }
48.     samples = 16384; // == 16K == 16*1024
49.     output_buffer = (int8_t *) malloc(16384);
50.     if (output_buffer == NULL) {
51.         fprintf(stderr, "Not enough memory, exiting\n");
52.         WildMidi_Shutdown(); return (1);
53.     }
54.     WildMidi_MasterVolume(master_volume);
55.     WildMidi_ClearError();
56.     midi_ptr = WildMidi_Open(argv[1]);
57.     if (midi_ptr == NULL) {
58.        ret_err = WildMidi_GetError(); printf(" Error:  %s\r\n", ret_err);
59.        close_openal_output(); free(output_buffer); return 3;
60.     }
61.     wm_info = WildMidi_GetInfo(midi_ptr);
62.     apr_mins = wm_info->approx_total_samples / (rate * 60);
63.     apr_secs = (wm_info->approx_total_samples % (rate * 60)) / rate;
64.
65.     mixer_options = wm_info->mixer_options;
66.     printf("\r\n[Approx %2um %2us Total]\r\n", apr_mins, apr_secs);
67.     fprintf(stderr, "\r");
68.     printf("Samples = %d\n", samples);
...
```

Listado 2: Listado de wildmidi_openal (parte 2).

Se necesita instalar los paquetes de desarrollo de libWildMido, lo que puede hacer con la orden:

```
$ sudo apt-get install libwildmidi-dev
```

Esperamos que el lector lo esté instalando para comprobar lo que decimos. ¿Lo tiene? Ha sido sencillo, ¿verdad? Seguimos que viene lo más interesante.

Puede comprobar que la instalación está operativa compilando el ejemplo de test[1] o directamente el que vemos en este apartado. Para compilar lo haremos con la ayuda de *pkg-config*, con la orden

```
$ gcc wildmidi_openal.c -o wildmidi_openal -lWildMidi `pkg-config openal --cflags --libs` && wildmidi_openal midis/bluesIntro.mid
```

o directamente especificando las librerías, con la orden

```
$ gcc wildmidi_openal.c -o wildmidi_openal -lWildMidi -I/usr/include/AL -lopenal && wildmidi_openal midis/bluesIntro.mid
```

El Listado 1 muestra la parte de declaraciones del código: dependencias, funciones auxiliares y estructuras de datos del código principal. En él hemos destacado las estructuras donde se guardará la información MIDI.

En los Listado 2 y Listado 3 se agrupa la secuencia de operaciones sobre el fichero MIDI, Es la parte encargada de traer la información MIDI desde fichero y convertirla a audio digital en una zona de la memoria central del computador. En ellos se ha resaltado la cadena de operaciones que son la base del acceso al contenido del fichero y la conversión de los eventos en sonido digital:

- *WildMidi_Init*[2] *(línea 43) que inicializa elementos internos de la reproducción como el fichero de configuración de los instrumentos de que dispone libWildMidi o la frecuencia de muestreo (debe ser un valor entre 11025 y 65000) para la generación de audio digital.*
- *WildMidi_Open* (línea 56) que abre el fichero y comprueba si su formato se corresponde con el estándar MIDI.
- *WildMidi_GetOutput* (línea 70) lee el fichero MIDI hasta obtener un número de bytes de audio digital. Este número debe ser múltiplo de cuatro (en el código toma el valor es de 16384 o 16KB) ya que los datos devueltos están formados por muestras (*samples*) de 16 bits estéreo (2 bytes para el canal izquierdo y otros dos para el canal derecho).

También se ha destacado el uso de *WildMidi_GetInfo* (líneas 61 y 72), que nos permite acceder a las propiedades del fichero, de las que destacamos:

- *current_sample*. Es el número de *samples* de audio digital ya obtenidos (sintetizados) a partir de la lectura del MIDI. Sirve para determinar el punto actual de la reproducción a partir del instante actual (líneas 75 y 76) o como porcentaje (línea 73).
- *approx_total_samples*. Es el número total de muestras estéreofónicas que se espera obtener del fichero MIDI y que se utiliza para calcular el tiempo total de la reproducción del MIDI (línea 62 y 63).
- *total_midi_time*. Es el tiempo total, en milésimas de segundo, de la ejecución del MIDI.

Con estos datos se podría reescribir este ejemplo en el modo de precarga ya comentado, ya que nos dice el total de *samples* contenidos en el fichero. En el presente código se han utilizado para dar una muestra del avance de la reproducción del archivo, tanto en minutos y segundos, como en porcentaje.

Y aunque menos destacado, las funciones *WildMidi_GetError* y *WildMidi_ClearError*, ofrecen acceso al control de errores interno de la librería.

---

1 Lo encontrará en el Github de libWildMidi [11] <https://github.com/Mindwerks/wildmidi/blob/master/test/test.c>.

2 Como hemos instalado la librería podemos acceder a la documentación con las páginas del manual en línea de órdenes de cualquiera de las funciones, p. ej.
$ man WildMidi_Init.

```
...
69.      while ( res > -1) { // (1)
70.        res = WildMidi_GetOutput(midi_ptr, output_buffer, samples);
71.        if (res <= 0) break;
72.        wm_info = WildMidi_GetInfo(midi_ptr);
73.        perc_play = (wm_info->current_sample * 100) /
74.                      wm_info->approx_total_samples;
75.        pro_mins = wm_info->current_sample / (rate * 60);
76.        pro_secs = (wm_info->current_sample % (rate * 60)) / rate;
77.        fprintf(stderr, "[%2um %2us Processed] [%2u%%] %c  \r",
78.                pro_mins, pro_secs, perc_play, spinner[spinpoint++ % 4]);
79.        if (write_openal_output(output_buffer, res) < 0) {
80.                 printf("\r");
81.        }
82.      } // while (res > -1)
83.      if (WildMidi_Close(midi_ptr) == -1) {
84.      ret_err = WildMidi_GetError();
85.      fprintf(stderr, "OOPS: failed closing midi handle!\r\n%s\r\
n",ret_err);
86.      }
87.     close_openal_output();
88.     free(output_buffer);
89.     if (WildMidi_Shutdown() == -1) {
90.         ret_err = WildMidi_GetError();
91.         fprintf(stderr,
92.              "OOPS: failure shutting down libWildMidi\r\n%s\r\n",
ret_err);
93.         WildMidi_ClearError();
94.      }
95.     printf("\r\n");
96.     return (0);
97.} // main
...
```

Listado 3: Listado de wildmidi_openal (parte 3).

El Listado 4 recoge las operaciones ahora desde el punto de OpenAL, esto es, cómo recoger la información de audio digital en memoria central y llevarla al motor de audio de OpenAL quien la enviará al dispositivo físico de audio del computador. En ese caso se han destacado las operaciones de trasvase y adecuación de la información en memoria central a los “buffers” de OpenAL como *alBufferData*. a*lSourceQueueBuffers* o *alSourcePlay*.

```
98. static int write_openal_output(int8_t *output_data, int output_size) {
99.     ALint processed = 0, state, bufid;
100.       if (frames < NUM_BUFFERS) { // initial state: fill the buffers
101.           alBufferData(buffers[frames], AL_FORMAT_STEREO16, output_data,
102.                        output_size, rate);
103.           if (++frames == NUM_BUFFERS){ // Now queue and start playback!
104.               alSourceQueueBuffers(sourceId, frames, buffers);
105.               alSourcePlay(sourceId);
106.           }
107.           return 0;     }
108.       /* Get relevant source info */
109.       alGetSourcei(sourceId, AL_SOURCE_STATE, &state);
110.       if (state == AL_PAUSED) { // resume it, then..
111.        alSourcePlay(sourceId);
112.        if (alGetError() != AL_NO_ERROR) {
113.        fprintf(stderr, "\nError restarting playback\r\n"); return (-1);}
114.       }
115.       while (processed == 0) { // Wait until we have a processed buffer
116.          alGetSourcei(sourceId, AL_BUFFERS_PROCESSED, &processed);     }
117.       /* Unqueue and handle each processed buffer */
118.       alSourceUnqueueBuffers(sourceId, 1, &bufid);
119.       /* Read the next chunk of data, refill the buffer, and queue it
120.       * back on the source */
121.       alBufferData(bufid, AL_FORMAT_STEREO16, output_data, output_size,
                  rate);
122.       alSourceQueueBuffers(sourceId, 1, &bufid);
123.       if (alGetError() != AL_NO_ERROR) {
124.           fprintf(stderr, "\nError buffering data\r\n"); return (-1);
   }
125.       /* Make sure the source hasn't underrun */
126.       alGetSourcei(sourceId, AL_SOURCE_STATE, &state);
127.       if (state != AL_PLAYING) {
128.           ALint queued;
129.           /* If no buffers are queued, playback is finished */
130.           alGetSourcei(sourceId, AL_BUFFERS_QUEUED, &queued);
131.           if (queued == 0) {
132.               fprintf(stderr, "\nNo buffers queued for playback\r\n");
133.               return (-1);         }
134.           alSourcePlay(sourceId);     }
135.       return (0);}
```

Listado 4: Listado de wildmidi_openal (parte 4).

```
...
136.  static void close_openal_output(void) {
137.      if (!context) return;
138.      printf("Shutting down sound output\r\n");
139.      alSourceStop(sourceId);          /* stop playing */
140.      alSourcei(sourceId, AL_BUFFER, 0);  /* unload buffer from source
      */
141.      alDeleteBuffers(NUM_BUFFERS, buffers);
142.      alDeleteSources(1, &sourceId);
143.      alcDestroyContext(context);
144.      alcCloseDevice(device);
145.      context = NULL;
146.      device = NULL;
147.      frames = 0;
148.  }
149.  //setup our audio devices and contexts
150.  static int open_openal_output(void) {
151.      device = alcOpenDevice(NULL);
152.      if (!device) {
153.          fprintf(stderr, "OpenAL: Unable to open default device.\r\n");
154.          return (-1);
155.      }
156.      context = alcCreateContext(device, NULL);
157.      if (context == NULL || alcMakeContextCurrent(context) ==
      ALC_FALSE) {
158.          if (context != NULL)
159.              alcDestroyContext(context);
160.          alcCloseDevice(device);
161.          context = NULL;
162.          device = NULL;
163.          fprintf(stderr,
164.                  "OpenAL: Failed to create the default context.\r\n");
165.          return (-1);
166.      }
167.      /* setup our sources and buffers */
168.      alGenSources(1, &sourceId);
169.      alGenBuffers(NUM_BUFFERS, buffers);
170.      return (0);
171.  }
```

Listado 5: Listado de wildmidi_openal (parte 5).

Por último, el Listado 5 recoge las instrucciones para inicializar y liberar los recursos del sistema relacionados con el hardware de sonido que gestiona OpenAL.

5 Conclusiones y cierre

El motor de audio 3D, OpenAL, parte de una especificación reducida para conseguir un impacto mínimo en el consumo de recursos de una aplicación. Sus capacidades de importación de ficheros se pueden ampliar utilizando una librería genérica como ALUT.

Pero la existencia de un número tan alto de formatos de audio hace interesante que el desarrollador pueda seleccionar el uso de bibliotecas especializadas en un formato para poder reducir la sobrecarga de código que puede no utilizarse en una aplicación. En ese sentido, se ha revisado el camino que sigue la información de audio desde fichero hasta los componentes de OpenAL para proponer un ejemplo de uso de audio en MIDI basado en el uso de la librería *libWildMidi*.

El lector puede ahora experimentar con esta base para incorporarla a su aplicación. ¿Por qué no lo comprueba? Ah, que no tiene un MIDI a mano, debería puede empezar por ahí[3].

6 Bibliografía

[1] OpenAL. Disponible en <http://www.openal.org>.

[2] Loki Software. (2000). OpenAL 1.0.Specification. Disponible en <https://pdfs.semanticscholar.org/831a/72e74a6f63dafb1ff74dfa5e311f416bc238.pdf>.

[3] OpenAL 1.1 Specification. (2005). Disponible en <http://www.openal.org/documentation/openal-1.1-specification.pdf>.

[4] MIDI Association. Disponible en <https://www.midi.org/>.

[5] WildMIDI: A Software Synthesizer. Mindwerks. Disponible en <https://www.mindwerks.net/projects/wildmidi/>.

[6] Joan. (2019). ¿Qué es el MIDI?:La guía del principiante para la herramienta musical más poderosa. Disponible en <https://blog.landr.com/es/que-es-el-midi-la-guia-del-principiante-para-la-herramienta-musical-mas-poderosa/>.

[7] The OpenAL Utility Toolkit. Disponible en <http://distro.ibiblio.org/rootlinux/rootlinux-ports/more/freealut/freealut-1.1.0/doc/alut.html >.

[8] -. Ventajas Generales. MIDI Herramienta Esencial. Disponible en <https://sites.google.com/site/midiherramientaesencial/home/ventajas-generales>.

[9] M. Agustí. (2011). Introducción al procesado de audio mediante OpenAL. Disponible en <http://hdl.handle.net/10251/12694>.

[10] M. Agustí. (2018). Reproducción de ficheros Opus con OpenAL: precarga vs "streaming". Disponible en <http://hdl.handle.net/10251/109211 >.

[11] Github de wildmidi. Disponible en <https://github.com/Mindwerks/wildmidi >.

3 Puede encontrar en muchos sitios en Internet como “Free MIDI Files – Where to Find Them” <https://beatlabacademy.com/free-midi/> o “Free Midi Files Download” <https://freemidi.org/>.