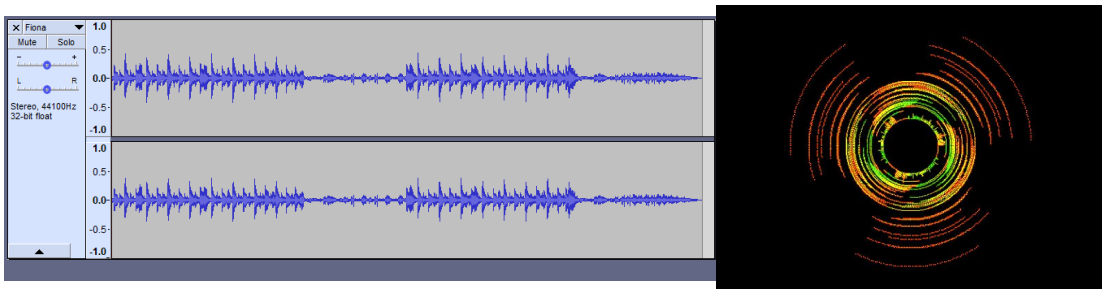
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

OpenAL y OpenGL: escuchar y ver el sonido

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

Posiblemente has utilizado algún editor de audio como *Audacity* (fig. 1a) que te permite grabar, reproducir, recodificar y aplicar operaciones y efectos sobre una sonido de forma gráfica, seleccionando la parte donde quieres aplicar el sonido o el punto donde quieres insertar otro. O quizá utilizas un complemento (o *plugin*) de visualización[1] (fig. 1b) en una aplicación de reproducción de listas de música.



a) *b)*

Figura 1: Representación del sonido en forma de ondas en (a) Audacity y (b) en el reproductor VLC.

Ambos casos son ejemplos de aplicaciones que visualizan el sonido, para operar sobre él o con fines mas artísticos. Tienen en común que son capaces de leer ficheros de audio (quizá comprimidos), extraer la información de audio y ofrecerla al computador para que este pueda reproducirla, esto es: transformarla de nuevo en ondas sonoras, para que lo podamos escuchar como estamos acostumbrados por nuestro oído.

El sonido se puede definir como una onda de presión. Es una señal que varia con el tiempo y cuyos valores nos llegan con más o menos intensidad o volumen. En este trabajo abordaré cómo se puede dibujar esa onda. Aunque se puede ser muy creativo en la forma de hacerlo, me voy a concentrar (por brevedad en la exposición), en el "pintado" de la variación del volumen del sonido, que es lo que se conoce como "forma de onda".

No esperes un representación tan exquisita como la de *Audacity*, sino solo los valores de audio representados en forma gráfica para "ver" el sonido. ¡Aunque todo es ponerse! El ejemplo completo se puede conseguir contactando con el autor a través del correo electrónico que figura en la portada de este documento.

Nota: En el texto, cuando me refiera a una de las figuras que se incluyen en él, utilizaré la abreviatura "fig." listada en la RAE[2] para este menester.

2 Objetivos

El objetivo del presente trabajo es obtener una representación gráfica del audio al tiempo que se reproduce, mediante las herramientas OpenGL y OpenAL, Los ejemplos que se muestras se han realizado en plataforma GNU/Linux y están escritos en lenguaje C.

A partir del estudio de los ejemplos que se abordan, el lector será capaz de:

---

1 Imagen extraída de "Audio Visualizations in VLC Media Player. VLC Help". Disponible en <https://www.vlchelp.com/audio-visualizations-vlc-media-player/>.

2 Lista de las abreviaturas convencionales más usuales en español. Disponible en <http://www.rae.es/diccionario-panhispanico-de-dudas/apendices/abreviaturas>.

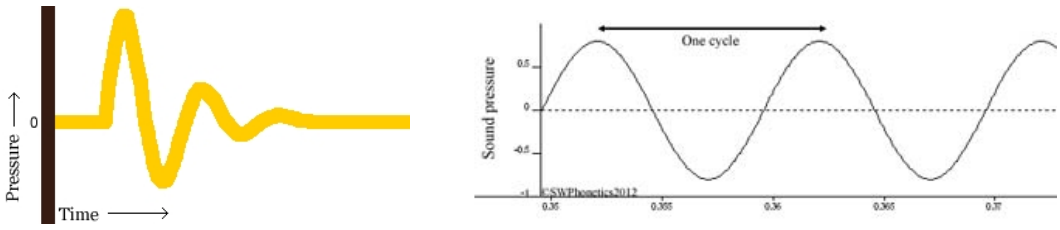
- Examinar las estrategias de reproducción de sonido digital y observar las diferencias entre precarga y *streaming*.
- Exponer la estrategia para incorporar el uso de del esquema de compresión de audio Opus en OpenAL.
- Instalar y compilar una aplicación que pueda hacer uso del formato Opus sobre OpenAL, con las funciones de la biblioteca *libopus*.

Para empezar voy a revisar la relación entre audio digital, su uso en un motor de audio como OpenAL [1] y las posibilidades de uso de un formato de fichero como OGG y la codificación Opus.

3 Introducción

Voy a hablar de qué es el audio digital para entender qué información se guarda en un fichero de audio. El sonido es [2] un frente de ondas que se propaga en un medio, como el aire. El sonido llega a nosotros a través de nuestros oídos y también, un poco, a través de la piel. Lo cual es posible porque ambos sentidos convierten el sonido en impulsos eléctricos que el cerebro ha aprendido a reconocer.

Hay sonidos, como el de un aplauso que son cortos y decaen con rapidez, fig. 2a. Mientras que hay otros, como el caso ideal de un tono puro, fig. 2b, que se prolonga sostenido en el tiempo y cuya representación es [3] más constante, en tanto que la onda es periódica en el tiempo. En ambos casos, se ha representado el sonido de forma gráfica como la envolvente del sonido: esto es, la curva formada por la unión de los valores de volumen que representan la evolución de la señal conforme avanza el tiempo.



a) *b)*

Figura 2: Un sonido es una señal de presión que varía con el tiempo y su representación como la envolvente de esa onda.

Es posible registrar el sonido mediante el uso de micrófonos que, a partir de las vibraciones de la membrada que hay en su interior, convierten la señal de sonido en una señal eléctrica. Esto implica un cambio de representación en tanto que, ahora, se tiene un valor de corriente eléctrica que varía a lo largo del tiempo. Esta señal se puede registrar y volver a convertir en una señal audible con un altavoz, que hace vibrar su membrana a partir de la señal eléctrica que le llega.

El audio digital es la codificación de esa señal eléctrica en valores que un computador puede procesar. Para ello se utilizan los conversores analógico-digital (*Analog-to-Digital Converter* o ADC), que muestrean el voltaje de una señal eléctrica. Esto es, se discretiza: la señal es medida muchas veces por segundo y, a cada medida realizada, se le asigna un valor en una escala. La fig. 3 muestra la señal analógica original (en amarillo) y, sobre ella, los puntos representan las muestras tomadas por el ADC.

Esta representación discreta se puede volver a convertir en una señal continua con un conversor digital-analógico (*Digital-to-Analog Converter* o DAC), que hace la función contraria al ADC. Esta señal es enviada a los auriculares o altavoces para volver a tomar forma de una señal audible. Ambos conversores se

encuentran en el *hardware* del subsistema de sonido de un computador, bien como una tarjeta de sonido, o bien integrado en la placa base.

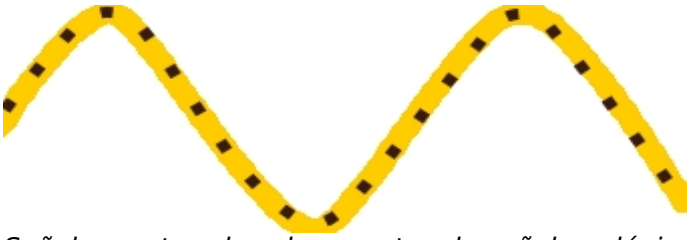


Figura 3: Señal muestreada sobrepuesta a la señal analógica original.

El número de muestras por segundo se denomina la velocidad de muestreo y se mide en Hercios (Hz). El número de valores que puede tomar cada medida se denomina tamaño de la muestra y es el número de bits (*N*) que se utilizan para determinar la escala de la conversión que estará en el rango de 0 a 2^N-1. En un CD-Audio se utilizan valores de 44.100 Hz o "44 Khz" de velocidad de muestreo, estéreo y 16 bits de tamaño de muestra. Cuantas más muestras se tomen y más valores tiene el convertidor en su escala, mayor es la precisión de la representación. Y también lo que ocupa al ser almacenada o transmitida.

A los valores discretos de audio digital que se observan en la fig. 3 se les conoce como sonido en PCM (del inglés *Pulse Code Modulation*) y representa el sonido en digital y sin compresión. En los ficheros en disco, este audio puede aparecer así directamente (como en un CD-Audio o un fichero WAVE) o comprimido para reducir sus exigencias de espacio (como en MP3, OGG/Vorbis, Opus, ...).

4 Desarrollo

Para que una aplicación sobre OpenAL pueda utilizar los sonidos desde ficheros Opus [5], puede cargarlos de forma estática o dinámica. Ambas opciones están disponibles para experimentar con ellas en los ejemplos de [6]. Uno de ellos es el que utilizaré de base en este trabajo. Veamos cuál es el escogido.

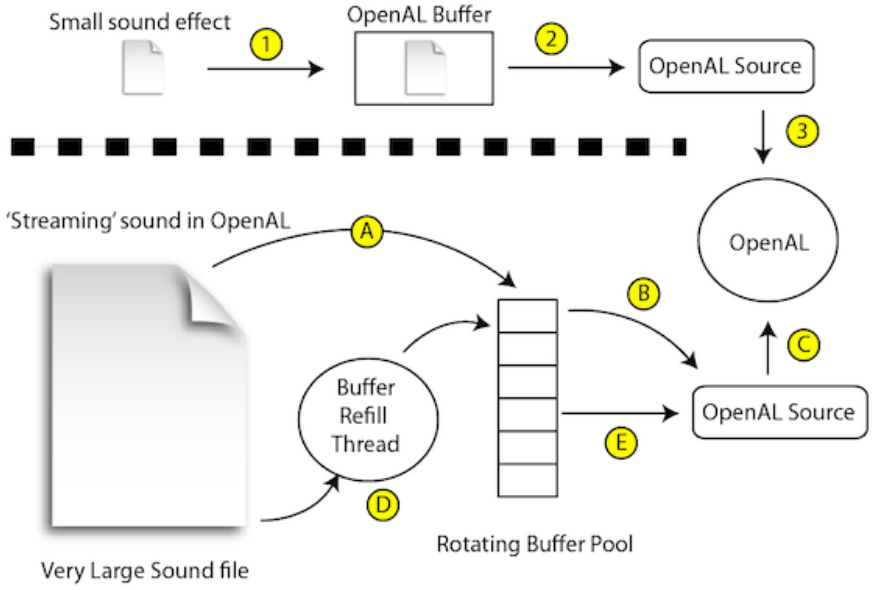


Figura 4: Esquema gráfico de las estrategias de precarga vs streaming en OpenAL y enumeración de pasos involucrados. Imagen extraída de [6.

En la parte superior de la fig. 4 se muestra la carga estática. Se utiliza *libopusfile* para leer (y descomprimir) el contenido del fichero en el paso 1. Lo carga todo en PCM en un buffer en memoria, lo asigna (paso 2) a una fuente de OpenAL (con *alBufferData*) y ya lo puede reproducir (paso 3). Esta forma de realizar la carga es razonable cuando el fichero es de pequeño tamaño. En cambio, cuando el tamaño del fichero se hace grande, el tiempo de espera y el consumo de recursos son importantes en esta solución.

En ese caso, la carga dinámica es la alternativa a valorar. Se ilustra en la parte inferior de la fig. 4. Se repite la secuencia para acceder al fichero, dejar parte de su contenido en PCM en un *buffer* y reproducirlo (en los pasos A B y C). Pero se debe seguir leyendo el resto de trozos del fichero y mantener una lista de ellos para que el motor de OpenAL pueda ir sacando de esa lista un nuevo trozo cuando ya haya hecho sonar el anterior (pasos D y E).

Visto que esta segunda opción es la que nos ofrece la posibilidad de ir recorriendo el fichero, podemos saber qué trozo está sonando y utilizar esa misma información para pintar el sonido. Para dibujar la onda, habrá que acceder a los valores en PCM que hay en memoria y que se están reproduciendo ahora y dibujar esos puntos respecto a un sistema de coordenadas, uniéndolos con segmentos. Los ejes a utilizar son el tiempo (en horizontal) y el volumen (en el vertical) como rango de valores en los que toman valores las muestras de sonido.

Ya tenemos la base para realizar el “render” del sonido: un ejemplo que combine la realización de la parte sonora, con una representación gráfica sincronizada. Voy a ocuparme ahora de cómo implementar las dos partes del ejemplo.

4.1 “Render” sonoro del audio

El ejemplo de reproducción de un fichero Opus en streaming [6] me sirve de apoyo para partir de una base en la que está construida la parte necesaria para hacer sonar un fichero de audio. En esta referencia hay un ejemplo completo que reproduce un fichero Opus en modo continuo (o *streaming*).

Esto es, reproduce el trozo que tiene cargado en memoria y va aprovechando, mientras esto se produce, para ir cargando el siguiente trozo. Cuando ha terminado de reproducir un trozo, tomará el siguiente y, mientras, se reaprovecha el espacio del primer trozo para cargar sobre él el siguiente a reproducir. Esta secuencia se repetirá mientas queden trozos del fichero por traer de disco.

Así, se minimiza el peso en memoria de ese recurso y tenemos un punto donde sabemos exactamente qué sonido (en PCM) se está escuchando. Así se facilitará el sincronismo entre la etapa de sonorización y la de representación gráfica.

Voy a comentar cómo está implementado el ejemplo de *streaming* de audio de [6]. **No esperes ver aquí todo ese código.** Voy a exponer el algoritmo que utiliza y, sobre él, abordaré con más detalle los puntos donde se introducen las modificaciones.

El algoritmo del listado 1 implementa el esquema de funcionamiento visto en la parte inferior de la fig. 4. Observa las correspondencias:

- Las etapas 2, 3.1 y 3.2 se corresponde con el paso A, es la inicialización de un primer *buffer* y de la fuente.
- La etapa 3.3 corresponde a los pasos B, E y C, donde se escoge el primer *buffer* disponible de la lista y se hace sonar.
- La etapa 3 se corresponde al paso D, donde se realiza el rellenado de la lista de *buffers*.

```
1       Initializar OpenAL
2       Obtener información del fichero OPUS → stream opus
3       Mientras haya datos que leer → update_stream
   3.1  Rellenar los buffers → fill_buffer
          Descomprimir y decodificar Opus → op_read
   3.2  Añadir el buffer a la lista → alSourceQueueBuffers
   3.3  Asignar a la fuente un primer buffer relleno de la lista
        y reproducir → alSourcePlay
   3.4  Sacar buffer leído  de la lista → alSourceUnqueueBuffers
4       Liberar recursos utilizados
```

Listado 1: Algoritmo de las etapas del reproductor de audio [6] en streaming.

4.2 “Render” gráfico del audio

Voy a indicar dónde, en el código del ejemplo tomado de partida, se puede introducir la funcionalidad de acceder al contenido del audio y plantear, a partir de ahí, cómo proceder a su dibujado.

No esperes ver aquí todo ese código (insisto, está completo en [6] y por eso no aparece aquí). Las modificaciones a realizar en el código original se centran en dos partes: la inicialización y el acceso a los datos que suenan, para dibujarlos.

4.2.1 Inicialización y gestión de eventos

El punto 1 del listado 1 ha de ser ampliado para incluir la inicialización y configuración de la parte gráfica. Esto se traduce en las líneas de:

- Listado 2, que contiene las cabeceras que deberán ir al principio del código, junto a las ya existentes.

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/freeglut.h>
```

Listado 2: Cabeceras de OpenGL a añadir.

- Listado 3, contiene las operaciones de gestión de eventos gráficos. La aplicación responderá a la pulsación de la tecla ‘z’ haciendo que el tamaño de la representación gráfica se haga más pequeño o ampliándolo con la tecla ‘Z’. También responderá al uso de la tecla *Esc*, cerrando la aplicación de forma ordenada.
  La ventana también tiene registradas dos acciones más que tienen que ver con la parte de refresco del gráfico (*idle*) y con el redimensionado de la ventana (para permitir ponerlo a pantalla completa o no) con la función *reshape*. Debe incluirse antes del *main* del ejemplo de partida.
- Listado 4 reúne las instrucciones para la creación y configuración de la ventana y el modo gráfico de esta. Estableciendo, entro otras cosas, un tamaño inicial, y las funciones indicadas de gestión de eventos de

teclado y del sistema gráfico. Debe incluirse después del código del listado 3 y antes del *main* del ejemplo de partida.

```
Float zoom=1.0f;          // variables para la parte de gráficos
int ample, alt, idFinestra;
void keyboard(unsigned char key, int x, int y) {
  switch(key)
  {
    case 'z':  zoom-=1;  break;
    case 'Z':  zoom+=1;  break ;
    case 'f': case 'F':   glutFullScreenToggle(); break ;
    case 27: // ESC → cerrar OpenAL  OpenGL
      alSourceStop( testSource );
      alcMakeContextCurrent(0);
      alcDestroyContext(ctx);
      alcCloseDevice(dev);
      glutDestroyWindow( idFinestra ) ;
      exit(0);
      break ;
    default:break ;
    }
   glutPostRedisplay() ;
}
void reshape(int w, int h) {
  ample = w;   alt = h;
  glViewport(0,0,(GLsizei)w,(GLsizei)h) ;
  glMatrixMode(GL_PROJECTION) ;
  glLoadIdentity() ;
  gluPerspective(50.0,(GLfloat)h/(GLfloat)w,1.0,30.0) ;
  glMatrixMode(GL_MODELVIEW) ;
  glLoadIdentity() ;
  glTranslatef(0.0,0.0,-6.6) ;
}
static void idle(void) {
  glutPostRedisplay();
}
```

Listado 3: Gestión de los eventos que recibe el interfaz gráfico.

```
void initOpenGL( int argc, char **argv, int *GLwin ) {
  glutInit(&argc, argv);
  ample = 800;   alt = 400;
  glutInitWindowSize( ample, alt );
  glutInitWindowPosition(0,0);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  *GLwin = glutCreateWindow("Dibuja la onda - M. Agustí") ;
  glutReshapeFunc(reshape) ;
  glutKeyboardFunc(keyboard);
  glutIdleFunc(idle);
  glEnable(GL_DEPTH_TEST);
  glDepthFunc(GL_LESS);
  glClearColor(1.0,1.0,1.0,1.0) ;
}
```

Listado 4: Inicialización de los gráficos.

4.2.2 Dibujado de la onda

El punto 3 del listado 1 referido a la reproducción del audio es el punto clave donde se han de situar la mayor parte de las modificaciones. Puesto que al ritmo que suene la música, quiero que se vaya actualizando la representación gráfica de la misma: así se conseguirá que estén sincronizados el audio y los gráficos.

```
int fill_buffer(ALuint buffer, OggOpusFile *file)
{
… // El código de estas líneas no necesita modificaciones
  alBufferData(buffer, format, buf,
               samples_read*num_channels*2, 48000);
  dibujarBuffer( buf );
  return samples_read;
}
```

Listado 5: La función fill_buffer es el lugar donde se insertará la actualización del dibujo de la señal.

Para ello:

- El listados 5 muestra dónde se va a introducir el dibujado de la señal sonora.
- El listado 6, muestra qué instrucciones componen el dibujado. .Básicamente es un reescalado (*valors*) de los valores PCM entre su máximo valor posible (2^{16}-1 o 65536), multiplicado por una constante (en este caso 10), para permitir que se pueda variar el "zoom" con que se visualiza la señal.
  Estos valores son convertidos a coordenadas horizontales y verticales

(*fx* y *fy*) para lanzar el dibujo de una polilínea que une cada pareja de puntos vecinos. Para ello se utilizan los valores de ancho y alto de la ventana actual, buscando ocupar el máximo espacio disponible.

```
// El mismo numero de muestras que se reproducen se han de pintar
#define N_MUESTRAS  960*2*2
void dibujarBuffer( int16_t buf[N_MUESTRAS] ) {
  int i;
  float valors[N_MUESTRAS],
        fx=0.0, fy=0.0;
  float separacioEnX;
// Reescalar valores: por el 10*maximo_valor(int16) para Zoom
  for (i=0; i<N_MUESTRAS; i++)     {
      valors[i] = (ALint)(((ALint*)buf)[i]/ (10* 65535));
  }
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  glEnd();
  glColor3d(0.0f,1.0f,0.0f);  // En verd, que queda més bonic
  glBegin(GL_LINE_STRIP);
  for (i=0; i<N_MUESTRAS; i++)     {
      fx = ( i / (float)ample) - 1.0f;
      fy = ( (valors[i] + 1.0) /((float)alt ));
      glVertex3f(fx,fy,zoom); // La Z para zoom
 }
  glEnd();
  glutSwapBuffers();
  glutMainLoopEvent(); //
}
```

Listado 6: Función de dibujado del trozo de audio que se está reproduciendo.

Para poder utilizar esas funciones es necesario instalar el soporte para el desarrollo en un determinado lenguaje. Así, p. ej., en la plataforma GNU/Linux para utilizar las funciones del API de Opus, deberemos instalar el soporte de desarrollo de aplicaciones en C para ellos. Se puede haacer desde un terminal con la orden

```
$ sudo apt-get install libopusfile-dev libopus-dev
```

Este dibujado podrá ser visto en pantalla como se muestra en la fig. 5. Para ello será necesario, asumiendo que el fichero se llama “dibujaLaOnda.c”, compilarlo con la orden:

```
$ gcc -o dibujaLaOnda dibujaLaOnda.cpp `pkg-config opusfile
--cflags --libs` `pkg-config openal --cflags --libs` -lglut -lGLU
-lGL -lm
```

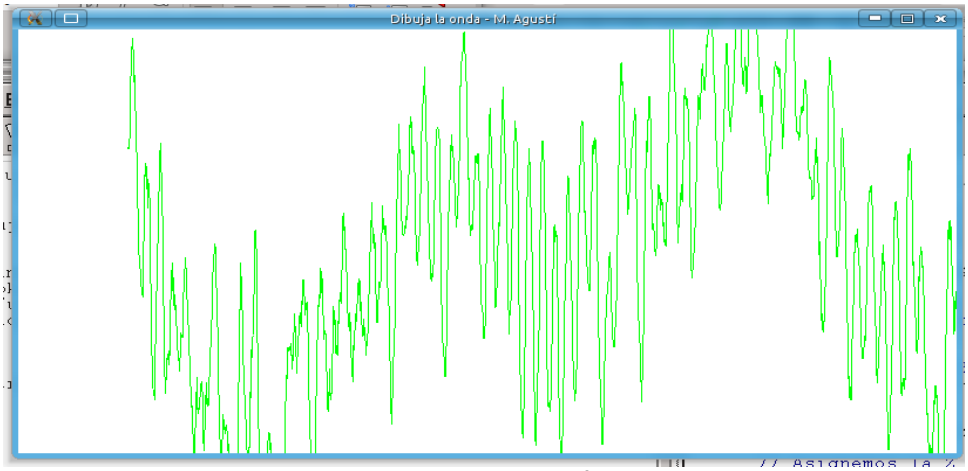


Figura 5: Un instante de la ejecución de dibujaLaOnda.

5 Cierre

En este trabajo se ha ilustrado cómo llegar a una representación gráfica de la señal de audio almacenada en un fichero en PCM o "forma de ondas". Para ello se han utilizado OpenGL para la salida gráfica y OpenAL para la sonora.

La exposición de la representación del audio en digital ha permitido valorar la complejidad de la codificación del audio en un fichero y la necesidad de acceder a los mismos valores que representan el audio pero trasladándolos a una representación gráfica en la que se utilizan otras escalas y otras unidades. El uso de la técnica de reproducción en continuo o *streaming* ha permitido sincronizar ambas representaciones del sonido.

Quiero agradecer a Aitor, Alexis, Clara, Rafa y Carles su participación en diferentes versiones de esta propuesta que aquí se ha expuesto.

Para comprobar que realmente has estado conmigo en este desarrollo, te animo a que pruebes a variar los colores o los parámetros de tamaño de *buffer* que se utilizan. Y, si te animas, probar otros tipos de representaciones gráficas y otros dibujos de motivos más artísticos te llevarán a un siguiente paso hacia la diversión. ¡¡ÁNIMO!!

6 Bibliografía

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

[2] "FLOSS Manuals: Audacity". Disponible en <http://write.flossmanuals.net/audacity/what-is-digital-audio/>.

[3] Wood, S. (2011), Understanding waveforms. Disponible en <https://swphonetics.com/praat/tutorials/understanding-waveforms/>

[4] Valdés, J-. Dibujando ondas de audio. Disponible en <http://hombrealto.com/web/?q=node/70>. Última consulta 27/01/2012.

[5] - . Opus Interactive Audio Codec. Disponible en <https://opus-codec.org/>.

[6] Valin. J.M., Vos, K y Terriberry, T. (2012). Definition of the Opus Audio Codec. Disponible en <https://tools.ietf.org/html/rfc6716>.

[6] Gow, D.. (2013). Using Opus with OpenAL. Disponible en <https://davidgow.net/hacks/opusal.html>.