UNIVERSITAT POLITÈCNICA DE VALÈNCIA

OpenAL: efecto *Doppler*. Posicionamiento y velocidad del 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

Cuando escuchamos el sonido de una sirena de ambulancia, policía o bomberos nos hemos acostumbrado a que suena diferente en función de lo lejos o cerca que esté de nosotros; es el efecto *Doppler*. En este artículo se expondrá cómo reconstruir este efecto en una escena sonora, mediante el motor de audio 3D OpenAL. Para ilustrar el efecto se construirá, con OpenGL, una visualización de la escena con objetos tridimensionales sobre la que el usuario puede interactuar.

Con estas técnicas el ejemplo mostrado podrá ser llevado a cualquier sistema operativo y constituye una visualización de audio e imagen 3D. Se detallarán los aspectos referidos al código que tienen especial relevancia en este caso para entender cómo funciona. La intención es experimentar y aprender con el ejemplo, no desarrollarlo desde cero. El proyecto completo se puede obtener del autor de este documento a través del correo electrónico del mismo.

2 Objetivos

En el presente documento vamos a presentar qué hay que hacer para simular en nuestro computador el efecto *Doppler*. Para ello centraremos nuestra atención en:

- Cuáles son las propiedades de este efecto y cómo se implementa en OpenAL.
- Cómo se relacionan las propiedades que afectan al cálculo del sonido (OpenAL) con las que nos permiten visualizar la escena (OpenGL).

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[1] para este menester. También se utilizará la abreviatura “ec.” cuando se referencie a una ecuación o fórmula que se enuncia en el texto.

No es objetivo de este documento la introducción al desarrollo de aplicaciones con el motor OpenAL o con OpenGL, ni la generación de una escena gráfica tridimensional realista. La consulta a los enlaces sugeridos te ayudará si necesitas referencias en este sentido.

3 Introducción

Vamos a hablar de qué es, desde el punto de vista físico el efecto *Doppler*, para pasar a ver en qué propiedades y funciones se especifica en OpenAL.

3.1 Efecto *Doppler*

Es el cambio de frecuencia que percibimos, en el sonido, cuando se mueven los objetos que lo emiten [1]. Depende de la velocidad relativa entre la fuente de sonido y el oyente. Al acercarse el sonido parece más agudo, mientras que al alejarse da la impresión de ser más grave. Se hace notar a partir de que la velocidad de los objetos (v_F) es apreciable frente a la de transmisión del sonido en el medio (v) siguiendo la expresión que muestra la ec. 1. El frente de ondas

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

(f) se ve “empujado” por el emisor del mismo en el caso de un coche en movimiento (véase fig. 1a). Esta “compresión” de las ondas se traduce en que el sonido cambia hacia valores de frecuencia más alta (f’), por lo que sonará más agudo al oyente que esté en esa zona próxima al emisor.

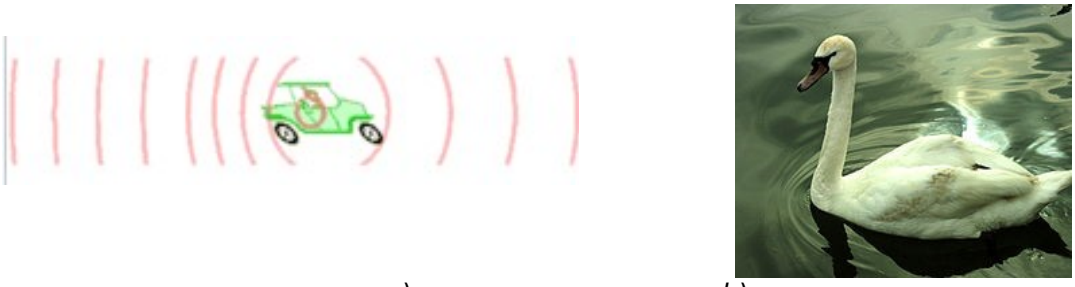
$$f' = f \cdot \frac{v}{v \mp v_F}$$

Ecuación 1: Efecto Doppler: relación entre frecuencia emitida y recibida.

También se da si el emisor está quieto y se mueve el oyente hacia el emisor, por lo que podemos ponerlo como una función de ellos:

$$Doppler_{OpenAL} = f(V_{Emisor}, V_{Oyente}, V_{Sonido}).$$

Y no es un efecto exclusivo del sonido, también se observa este fenómeno en otros frentes de ondas, como el de un cisne mientras nada en el agua (fig. 1b)



a) *b)*

Figura 1: Ejemplos gráficos del efecto Doppler (a) en el sonido y (b) en el agua. Imágenes extraídas de la wikipedia <https://es.wikipedia.org/wiki/Efecto_Doppler>.

¿Cómo se puede introducir este efecto en una aplicación propia? Vamos a dejar que lo haga un experto: OpenAL.

3.2 OpenAL

Este motor de audio 3D nos permitirá recrear una escena de audio teniendo en cuenta el posicionamiento tridimensional de las fuentes de sonido respecto al oyente, que es la posición del observador. También puede aplicar efectos sobre los sonidos, grabar desde micrófono, etc. El sitio web de OpenAL proporciona los recursos necesarios para su instalación y documenta el API de OpenAL con: las especificaciones [2], la guía del programador [3] y las extensiones [4]. La falta de ejemplos de uso en los que se elabora su uso se compensa con la cantidad de estos y de proyectos que hay disponibles en Internet[2].

OpenAL permite aplicar el efecto *Doppler* a partir de la especificación [3] de:

- La velocidad de las fuentes de sonido y del oyente, asi como de sus y su posiciones.
- La velocidad de transmisión del sonido en el medio. Hay que tener en cuenta que las unidades deben ser dadas en metros/segundo (m/s) y la velocidad del sonido está establecida, por defecto, en 343,3 m/s, que es

2 Puedes ver unos ejemplos rápidos en: “OpenAL 3D Audio Tutorial1: Introduction” <https://www.youtube.com/watch?v=BR8KjNkYURk >, “OpenAL short example” <https://ffainelli.github.io/openal-example/> y en “OpenAL” <https://www.ecured.cu/OpenAL>.

la de propagación en el aire. Estas unidades no están relacionadas con las de posición.

- El factor de ganancia. Por si quieres acentuar o relajar el efecto, se dispone de un factor adicional que se aplica al cálculo de la magnitud física.

Con todos estos elementos, se puede enunciar el efecto el desplazamiento en frecuencia calculado es proporcional al la velocidad del oyente y la fuente, a lo largo de la línea que los une. La fig. 2 Lo muestra gráficamente: con los valores vectoriales de posición y velocidad de la fuente (*S* y SV), posición y velocidad del oyente (*L* y *SL*), así como con las proyecciones (estos son valores escalare) de la velocidad de cada uno en la línea que une ambas posiciones (*vss* y *vls*).

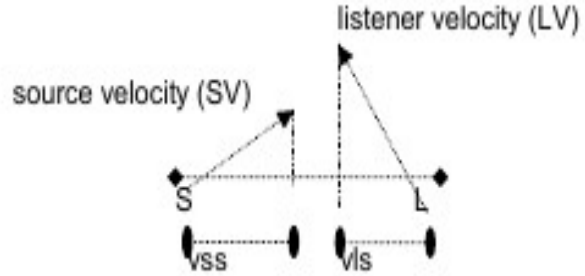


Figura 2: Relaciones geométricas entre emisor y receptor. Imagen obtenida de [3]

$$vls = \frac{SL \cdot LV}{|SL|}, vss = \frac{SL \cdot SV}{|SL|}$$
$$vss = min(vss, \frac{SS}{DF}), vls = min(vls, \frac{SS}{DF})$$

$$\Rightarrow f' = f * \frac{SS - DF * vls}{SS - DF * vss}$$

Ecuación 2: Cálculos para determinar el efecto Doppler en OpenAL.

A partir de este modelo y utilizando las operaciones matemáticas de norma o magnitud de un vector $|v|$ y producto escalar de dos vectores $v1 \cdot v2$, la implementación del efecto *Doppler* en OpenAL se describe con la ec. 2. En la que se representa con:

- *SS*, es la velocidad de transmisión del sonido en el medio que se quiera simular. En el API de OpenAL se llama *AL_SPEED_OF_SOUND.*
- *DF* (*DOPPLER FACTOR*), es el factor adicional, que vale 1.0, por defecto. En el API de OpenAL se llama *AL_DOPPLER_FACTOR.*
- *vls* y *vss,* son las proyecciones de SL y SV, respectivamente, en el eje que une a la fuente y el oyente.
- *f*,la frecuencia del sonido emitido.
- *f'*, la frecuencia del sonido percibida.

Por lo tanto, en una aplicación sobre OpenAL, para emplear e efecto *Doppler* es necesario que las fuentes y el oyente tengan asignados valores de posición.

También lo es que alguno de ellos tenga valores de velocidad distintos de cero (que se mueva, vamos). Además es posibles ajustar la velocidad de propagación

y se dispone de un control extra para enfatizar o rebajar la perturbación del sonido debida al efecto *Doppler*.

4 Desarrollo

Para construir este ejemplo de visualización de audio en 3D se han tomado elementos de diferentes fuentes, que se han ido combinando para obtener el resultado deseado: un ejemplo básico de ejemplo de OpenAL con varias fuentes estáticas [5] serviría de inicio (pero ya no está disponible), el proyecto PIGE [6]- lo he utilizado para montar una escena básica y asociar sonidos, la documentación de OpenGL [7] y ALUT [8] son necesarias para crear y juntar todas la piezas y, no menos importante, “Efecto Doppler” [9] era un pequeño documento que recordaba las opciones pero no las ponía en un ejemplo. Es por ello que se conservan los comentarios originales donde es posible. Como algunoa de estas referencias ya no está disponible he creído necesario poner aquí todo el código posible.

El desarrollo que abordamos es el de una aplicación que refuerce con la imagen lo que está sucediendo con el audio. Esto es, que permita observar como los elementos que aparecen en ella tienen un sonido asociado y que este refleja la posición y velocidad de los mismos respecto al oyente que es, al tiempo, el usuario que está viendo la escena. Así tenemos una idea como la que muestra la fig. 3, en la que se observa las tareas encargadas a OpenGL y a OpenAL. Por ello, las fuentes de sonido deberán tener una visualizacióon que permita diferenciarlas, así como también diferenciar su estado: en tanto si están reproduciendo el audio asociado o si están apagados.

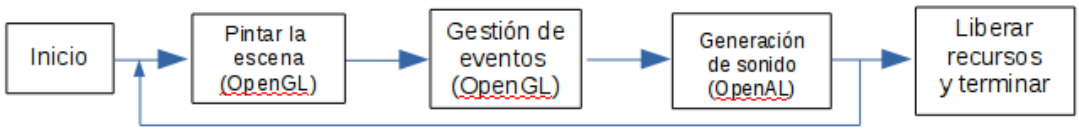


Figura 3: Idea inicial de la escena a construir.

Los controles (eventos que gestiona OpenGL) que se dejan en mano del usuario son encender/apagar las fuentes (con las teclas de '1', '2' y '3' las fijas y con '4' el objeto móbil), factor i, la velocitad del móbil ('v' la baja y 'V' la incrementa), la posición del oyente (con las teclas del cursor o con 'asqz' y el factor *Doppler* (con 'D' lo aumenta y 'd' lo hace decrecer).

OpenGL nos permitirá crear una escena gráfica en la que los objetos se representar en un espacio tridimensional. Existe una correlación entre los sistemas de coordenadas de ambos motores, por lo que es bastante directo reutilizar la posición y velocidad de los objetos a dibujar con OpenGL respecto a la cámara, en la recreación del sonido que los acompaña que genera OpenAL respecto al oyente. OpenGL y OpenAL son multiplataforma, por lo que los desarrollos realizados con ambas librerías son portables a otras plataformas.

4.1 Ejemplo desarrollado

Necesitarás cuatro ficheros de sonido *WAVE*, los puedes buscar en *Freesound*[3], en mi caso, estan dentro de un subdirectorio “wavdata” y son: "ambulancia.wav", “Battle.wav”, “Gun1.wav” y “Gun2.wav”. Si los cambias,

3 El sitio web esta en <*https://freesound.org/*>.

modifica las referencias a ellos. El código se puede compilar, desde la linea de ordenes, con:

```
$ gcc -o efectpDoppler efectpDoppler.c -lalut -lopenal -lglut -lGLU -lGL -lm
```

Este es el contenido completo del ejemplo:

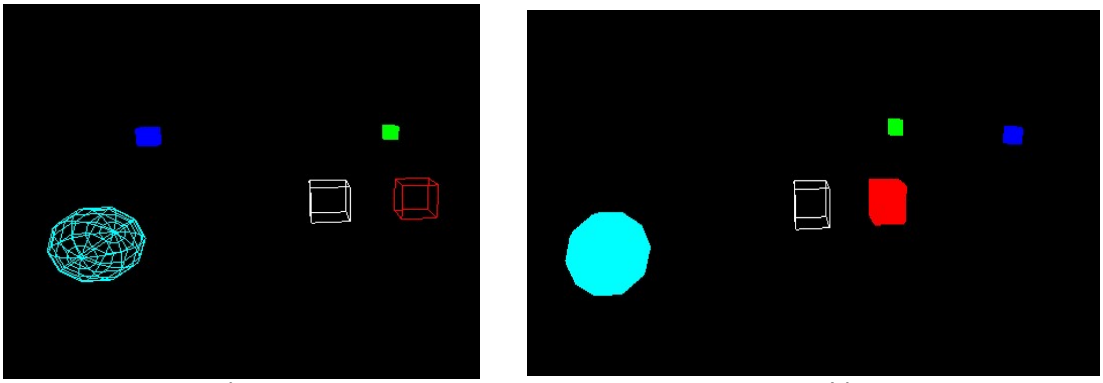
```
// efectpDoppler.c = [5] + [6] + [7] + [8] + [9]

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <AL/al.h>              // Soporte de audio
#include <AL/alc.h>
#include <AL/alut.h>
#include <GL/glut.h>            // Soporte de modo grafico
#include <GL/freeglut.h>        // FullScreen*

#define NUM_BUFFERS 4           // Maximum data buffers we will need.
#define NUM_SOURCES 4           // Maximum emissions we will need.
#define BATTLE 0                // These index the buffers and sources.
#define VEHICLE 1
#define GUN1   2
#define GUN2   3

ALuint Buffers[NUM_BUFFERS];                // Buffers hold sound data.
ALuint Sources[NUM_SOURCES];                // Sources are points of emitting sound.
ALfloat SourcesPos[NUM_SOURCES][3];         // Position of the source sounds.
ALfloat SourcesVel[NUM_SOURCES][3];         // Velocity of the source sounds.
ALfloat ListenerPos[] = { 0.0, 0.0, 0.0 };  // Position of the listener.
ALfloat ListenerVel[] = { 0.0, 0.0, 0.0 };  // Velocity of the listener.
// Orientation of the listener. (first 3 elements are "at", second 3 are "up")
ALfloat ListenerOri[] = { 0.0, 0.0, 1.0, 0.0, 1.0, 0.0 };
// Estaba 0,0, 0,0, -1.0, para alinearlo a la camara de OpenGL
ALboolean mute[NUM_SOURCES];
int GLwin;
#define DISTANCIA 10
ALfloat velocitatDelMovil = 80;             // En unidades de m/s
ALfloat factorPerAlDoppler = 0.0;
ALfloat velocitatPerAlDoppler = 0.0;

/*   This function will load our sample data from the disk using the alut
 *   utility and send the data into OpenAL as a buffer. A source is then
 *   also created to play that buffer.
 Simplificado con el uso de ALUT: alutCreateBufferFromFile            */
ALboolean LoadALData() {
  alGenBuffers(NUM_BUFFERS, Buffers);           // Load wav data into buffers.
  if(alGetError() != AL_NO_ERROR)      return AL_FALSE;

  Buffers[BATTLE]  = alutCreateBufferFromFile( "wavdata/Battle.wav" );
  Buffers[GUN1]    = alutCreateBufferFromFile( "wavdata/Gun1.wav" );
  Buffers[GUN2]    = alutCreateBufferFromFile( "wavdata/Gun2.wav" );
  Buffers[VEHICLE] = alutCreateBufferFromFile( "wavdata/ambulancia.wav" );
  mute[BATTLE] = 0;  mute[GUN1] = 0;  mute[GUN2] = 0;  mute[VEHICLE] = 0;

  alGenSources(NUM_SOURCES, Sources);  // Bind buffers into audio sources.
  if(alGetError() != AL_NO_ERROR)      return AL_FALSE;

  alSourcei (Sources[VEHICLE], AL_BUFFER,   Buffers[VEHICLE]   );
  alSourcef (Sources[VEHICLE], AL_PITCH,    1.0f              );
  alSourcef (Sources[VEHICLE], AL_GAIN,     1.0f              );
  alSourcefv(Sources[VEHICLE], AL_POSITION, SourcesPos[VEHICLE]);
  alSourcefv(Sources[VEHICLE], AL_VELOCITY, SourcesVel[VEHICLE]);
  alSourcei (Sources[VEHICLE], AL_LOOPING,  AL_TRUE           );

  alSourcei (Sources[BATTLE], AL_BUFFER,   Buffers[BATTLE]   );
  alSourcef (Sources[BATTLE], AL_PITCH,    1.0f              );
  alSourcef (Sources[BATTLE], AL_GAIN,     1.0f              );
  alSourcefv(Sources[BATTLE], AL_POSITION, SourcesPos[BATTLE]);
  alSourcefv(Sources[BATTLE], AL_VELOCITY, SourcesVel[BATTLE]);
  alSourcei (Sources[BATTLE], AL_LOOPING,  AL_TRUE           );

  alSourcei (Sources[GUN1], AL_BUFFER,   Buffers[GUN1]   );
  alSourcef (Sources[GUN1], AL_PITCH,    1.0f            );
```

```
  alSourcef (Sources[GUN1], AL_GAIN,     1.0f           );
  alSourcefv(Sources[GUN1], AL_POSITION, SourcesPos[GUN1]);
  alSourcefv(Sources[GUN1], AL_VELOCITY, SourcesVel[GUN1]);
  alSourcei (Sources[GUN1], AL_LOOPING,  AL_FALSE       );

  alSourcei (Sources[GUN2], AL_BUFFER,   Buffers[GUN2]   );
  alSourcef (Sources[GUN2], AL_PITCH,    1.0f           );
  alSourcef (Sources[GUN2], AL_GAIN,     1.0f           );
  alSourcefv(Sources[GUN2], AL_POSITION, SourcesPos[GUN2]);
  alSourcefv(Sources[GUN2], AL_VELOCITY, SourcesVel[GUN2]);
  alSourcei (Sources[GUN2], AL_LOOPING,  AL_FALSE       );

  // Do another error check and return.
  if(alGetError() != AL_NO_ERROR)  return AL_FALSE;
  return AL_TRUE;
}

/*   We already defined certain values for the listener, but we need
 *   to tell OpenAL to use that data. This function does just that.        */
void SetListenerValues() {
  alListenerfv(AL_POSITION,    ListenerPos);
  alListenerfv(AL_VELOCITY,    ListenerVel);
  alListenerfv(AL_ORIENTATION, ListenerOri);
}

/*   We have allocated memory for our buffers and sources which needs
 *   to be returned to the system. This function frees that memory.    */
void KillALData() {
  alDeleteBuffers(NUM_BUFFERS, Buffers);
  alDeleteSources(NUM_SOURCES, Sources);
  alutExit();
}

void ajuda() { // Por si no recuerdas los controles (las teclas) que puedes pulsar
  printf(" '1', '2' i '3'  on/off les fonts 1, 2 i 3. El '4' on/off el vehicle.\n\
La posició del oient es canvia en les tecles del cursor o en 'asqz'\n\
Doppler: 'D'/'d' per al factor i 'v'/'V' per a la velocitat\n\n");
}

void display(void) {                                  // Actualiza la escena visual
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) ;
  glPushMatrix() ;
  glRotatef(20.0,1.0,1.0,0.0) ;

  glPushMatrix() ;  glColor3f(1.0,0.0,0.0) ;
  glTranslatef(SourcesPos[BATTLE][0], SourcesPos[BATTLE][1], SourcesPos[BATTLE][2]);
  if (mute[BATTLE] == 1)       glutWireCube(0.5);
  else                         glutSolidCube(0.5);  glPopMatrix() ;

  glPushMatrix() ;  glColor3f(0.0,1.0,0.0) ;
  glTranslatef(SourcesPos[GUN1][0], SourcesPos[GUN1][1], SourcesPos[GUN1][2]) ;
  if (mute[GUN1] == 1)     glutWireCube(0.5);
  else                     glutSolidCube(0.5);   glPopMatrix() ;

  glPushMatrix() ;
  glTranslatef(SourcesPos[GUN2][0], SourcesPos[GUN2][1], SourcesPos[GUN2][2])
  glColor3f(0.0,0.0,1.0) ;
  if (mute[GUN2] == 1)     glutWireCube(0.5);
  else                     glutSolidCube(0.5);   glPopMatrix() ;

  glPushMatrix() ;  glColor3f(0.0,1.0,1.0) ;
  glTranslatef(SourcesPos[VEHICLE][0], SourcesPos[VEHICLE][1], SourcesPos[VEHICLE]
[2]);
  if (mute[VEHICLE] == 1)      glutWireSphere(0.5, 10, 10);
  else                         glutSolidSphere(0.5, 10, 10);  glPopMatrix() ;

  glPushMatrix() ;  glColor3f(1.0,1.0,1.0) ;  //the listener
  glTranslatef(ListenerPos[0], ListenerPos[1], ListenerPos[2]) ;
  glutWireCube(0.5) ;   glPopMatrix() ;

  glPopMatrix() ;
  glutSwapBuffers() ;
```

```
}

void reshape(int w, int h) {
  glViewport(0,0,(GLsizei)w,(GLsizei)h) ;
  glMatrixMode(GL_PROJECTION) ;
  glLoadIdentity() ;
  gluPerspective(60.0,(GLfloat)w/(GLfloat)h,1.0,30.0) ;
  glMatrixMode(GL_MODELVIEW) ;
  glLoadIdentity() ;
  glTranslatef(0.0,0.0,-6.6) ;
}

void keyboard(unsigned char key, int x, int y)  {
  ALint state;
  switch(key)    {
    case 'h':    case 'H':   ajuda();         break;
    case '1':
      alGetSourcei(Sources[BATTLE], AL_SOURCE_STATE, &state);
      if(state != AL_PLAYING) { alSourcePlay(Sources[BATTLE]); mute[BATTLE] = 0; }
         else             { alSourceStop(Sources[BATTLE]); mute[BATTLE] = 1; } break;
    case '2':
      alGetSourcei(Sources[GUN1], AL_SOURCE_STATE, &state);
      if(state != AL_PLAYING) { alSourcePlay(Sources[GUN1]);  mute[GUN1] = 0;      }
      else           { alSourceStop(Sources[GUN1]);  mute[GUN1] = 1;      } break;
    case '3':
      alGetSourcei(Sources[GUN2], AL_SOURCE_STATE, &state);
      if(state != AL_PLAYING) { alSourcePlay(Sources[GUN2]);  mute[GUN2] = 0;      }
      else           { alSourceStop(Sources[GUN2]);  mute[GUN2] = 1;      } break;
    case '4':
      alGetSourcei(Sources[VEHICLE], AL_SOURCE_STATE, &state);
      if(state != AL_PLAYING){ alSourcePlay(Sources[VEHICLE]); mute[VEHICLE] = 0;  }
      else          { alSourceStop(Sources[VEHICLE]); mute[VEHICLE] = 1;  } break;
    case 'f': case 'F':    glutFullScreenToggle();        break;
    case 'a': case 'A':    ListenerPos[0]-=0.1;
                alListenerfv(AL_POSITION, ListenerPos); break ;
    case 's':  case 'S':
     ListenerPos[0] += 0.1; alListenerfv(AL_POSITION, ListenerPos); break ;
    case 'q':     case 'Q':
     ListenerPos[2]-=0.1; alListenerfv(AL_POSITION, ListenerPos); break ;
    case 'z':     case 'Z':
     ListenerPos[2] += 0.1; alListenerfv(AL_POSITION, ListenerPos); break ;
    case 'v':     case 'V':
      velocitatDelMovil = (key == 'v'? -10.0: 10.0);
      SourcesVel[VEHICLE][0] += velocitatDelMovil;
      SourcesVel[VEHICLE][1] += velocitatDelMovil;
      SourcesVel[VEHICLE][2] += velocitatDelMovil;
      alSourcefv(Sources[VEHICLE], AL_VELOCITY, SourcesVel[VEHICLE]);     break;
    case 'd':    case 'D':
      factorPerAlDoppler += (key == 'd'? -0.1: 0.1.);
      alDopplerFactor( factorPerAlDoppler); break;
    case 'e':    case 'E':
      velocitatPerAlDoppler += (key == 'd'? -1.0: 1.0.);
       alDopplerFactor( velocitatPerAlDoppler);      break;
    case 27: //Tecla ESC
      alSourceStop(Sources[GUN2]); alSourceStop(Sources[GUN1]);
      alSourceStop(Sources[BATTLE]);
      alutExit(); glutLeaveMainLoop(); glutDestroyWindow(GLwin) ; exit(0) ; break ;
    default: break;
    }
  glutPostRedisplay() ;
}

void specialKeys(int key, int x, int y){
  switch(key)    {
    case GLUT_KEY_RIGHT: ListenerPos[0] += 0.1;
               alListenerfv(AL_POSITION, ListenerPos); glutPostRedisplay() ; break;
    case GLUT_KEY_LEFT:  ListenerPos[0] -= 0.1;
               alListenerfv(AL_POSITION, ListenerPos); glutPostRedisplay() ; break;
    case GLUT_KEY_UP: ListenerPos[2] -= -0.1;
               alListenerfv(AL_POSITION, ListenerPos); glutPostRedisplay() ; break;
    case GLUT_KEY_DOWN: ListenerPos[2] += 0.1;
                alListenerfv(AL_POSITION, ListenerPos); glutPostRedisplay() ; break;
```

```
  }
}

int main(int argc, char *argv[]){
  ALint state, nIteracio;
  double theta;

  printf("Visualizar fuentes de sonido en movimiento.\n\
Basado en:\n\
MindCode's OpenAL Lesson 3: Multiple Sources \n\
y \n\
PIGE-OpenAL. Chad Armstrong on Mon Jul 29 2002.\n\n");
  printf("Para ver la asignación de teclas: 'h') \n");
  // Initialize OpenAL and clear the error bit.
  alutInit(NULL, 0);
  alGetError();
  printf("Versió OpenAL: %s\n", alGetString( AL_VERSION ) );
  printf("Versió ALUT: %d.%d\n", alutGetMajorVersion(), alutGetMinorVersion() );
  if(LoadALData() == AL_FALSE)     return 0;             // Load the wav data.
  SetListenerValues();
  atexit(KillALData);                                    // Setup an exit procedure.
  //Inicializar OpenGL y lanzar interfaz gráfico: Initialise glut
  glutInit(&argc, argv) ;    printf( "Versió GLUT %d\n", GLUT_API_VERSION );
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH) ;
  glutInitWindowSize(1024,468) ;
  GLwin = glutCreateWindow("Fuentes en movimiento ejemplo sobre OpenAL + OpenGL");
  glutDisplayFunc(display) ;
  glutKeyboardFunc(keyboard) ;
  glutSpecialFunc(specialKeys);
  glutReshapeFunc(reshape) ;

  nIteracio = 0;
  alSourcePlay(Sources[BATTLE]);   // Begin the battle sample to play.
  // Go through all the sources and check that they are playing. Skip the first
  // source because it is looping anyway (will always be playing).
  // i tammé el segon, que serà el que es meneja en una velocitat variable
  // pegant voltes respecte a l'oritge)
  while( 1 )    {
     glutMainLoopEvent();
      // Menejar l'ambulància en cerles
      nIteracio = (nIteracio + (int)round(SourcesVel[VEHICLE][0])) % 360;

      for(int i = 1; i < NUM_SOURCES; i++)  {
       alGetSourcei(Sources[i], AL_SOURCE_STATE, &state);
       if ((i == VEHICLE) && (state == AL_PLAYING)) {
              theta = (double) (nIteracio) * 3.14 / 180.0;
              SourcesPos[i][0] = -float(cos(theta)) * 4.0;
              SourcesPos[i][1] = -0.1 * 5.0; // DISTANCIA;
              SourcesPos[i][2] = -float(sin(theta)) * 5.0;
          }
       if((i> VEHICLE) && (state != AL_PLAYING)){
         // Pick a random position around the listener to play the source.
              theta = (double) (rand() % 360) * 3.14 / 180.0;
           // L'aparte un poc: DISTANCIA
              SourcesPos[i][0] = -float(cos(theta)) * DISTANCIA;
              SourcesPos[i][1] = -float(rand()%2) * DISTANCIA;
              SourcesPos[i][2] = -float(sin(theta)) * DISTANCIA;
       }
       alSourcefv(Sources[i], AL_POSITION, SourcesPos[i]);
       if ((i> VEHICLE) && (mute[i] == 0))   alSourcePlay(Sources[i]);
       else        alutSleep( 0.1 );
     }
      display();
    }
  return 0;
}
```

La fig. 4a muestra un momento de la ejecución en la que la fuente de sonido móvil y las fuentes fijas están apagadas. Por ello se representan como objetos huecos: las fijas con cubos y el móvil por una esfera .



a) *b)*

Figura 4: El dibujo de los objetos informa de su posición, movimiento y de su estado: (a) fuentes apagadas y (b) fuentes emitiendo sonido .

En otro instante de la ejecución (fig. 4b), las fuentes fijas irán apareciendo en posiciones aleatorias y no se moverán hasta que no terminen de emitir su sonido. El móvil si que va cambiando de posición mientras emite su sonido de sirena para que se pueda apreciar el efecto *Doppler*.

5 Conclusión y trabajos futuros

A lo largo de este artículo hemos visto como el motor de audio 3D OpenAL es capaz de aplicar el efecto *Doppler* a los objetos en movimiento y de qué parámetros depende este efecto. Se ha construido una escena tridimensional, tanto en visualización como en sonido, para que el oyente pueda experimentar con su posición dentro de la escena, así como también las de las fuentes y los valores que se utilizan para calcular el efecto *Doppler*.

A partir del estudio del ejemplo, **el lector será capaz de**:

- Enunciar las funciones de OpenAL que permiten caracterizar los sonidos con su posición y velocidad.
- Experimentar con los rangos de valores aceptables para modelar este efecto.
- Llevar el ejemplo comentado a otros sistemas operativos, puesto que el uso de las librerías escogidas facilita la portabilidad.

Por supuesto que las cosas no se acaban aquí. Espero que el lector interesado tome la decisión de modificar el ejemplo, incluyendo otros sonidos, otros objetos o aumentando cantidad y calidad de la salida gráfica. ¿Y si cambiamos la velocidad de propagación[4] del sonido? Se pone una imagen de fondo para decorar cada “ambiente” y a divertirse. ¡¡ÁNIMO!!

6 Bibliografía

[1] Coronado, G. y Fernández, J. L. (2013). Efecto Doppler. FisicaLab. Disponible en <https://www.fisicalab.com/apartado/efecto-doppler>.

4 Tienes algunos ejemplos en la Wikipedia <https://es.wikipedia.org/wiki/Velocidad_del_sonido >.

[2] -. (2005). OpenAL 1.1 Specification and Reference. Disponible en <http://www.openal.org/documentation/openal-1.1-specification.pdf>

[3] -. (2007).OpenAL_Programmers_Guide. OpenaL versions 1.0 and 1.1. Disponible en <http://www.openal.org/documentation/OpenAL_Programmers_Guide.pdf>. Consultada el 27/03/018.

[4] Creative. Tech. Ltd. (2006). Effects Extension Guide. v. 1.1. Disponible en <http://kcat.strangesoft.net/misc-downloads/Effects%20Extension%20Guide.pdf>. Consultada el 27 de marzo de 2018.

[5] MindCode's OpenAL Lesson 3: Multiple Sources <http://forum.devmaster.net/t/openal-lesson-3-multiple-sources/2890>.

[6] Armstrong, C. (2002). PIGE-OpenAL <http://www.edenwaith.com/products/pige/tutorials/openal.php >.

[7] The freeglut Project: API Documentation <http://freeglut.sourceforge.net/docs/api.php#EventProcessing>.

[8] The OpenAL Utility Toolkit (ALUT) <http://distro.ibiblio.org/rootlinux/rootlinux-ports/more/freealut/freealut-1.1.0/doc/alut.html>.

[9] Maurais, J. (2003). OpenAL Lesson 7: The Doppler Effect. Disponible en <http://devmaster.net/p/2894/openal-lesson-7-the-doppler-effect>. Consultado en 05/10/2017.