OpenAL on Android

By **martin < https://pielot.org/author/mpielot/>**

December 14, 2010 < https://pielot.org/2010/12/openal-on-android/>

Although being slightly late with 3D Audio Support for Android 2.3 announced – this tutorial shows how to compile OpenAL for Android, so you can provide 3D Sound in your apps with 2.2 and below. The code has successfully been tested with the Nexus One (2.2) and the G1 (1.6).

Update: **the resulting project for download as a single .zip file < http://pielot.org/wp-content/uploads/2011/01/HelloOpenAL.zip>** . To run the example create a directory called wav on your device's SD card and put a sound file called lake.wav into it.

Update: some people reported **latency issues**. It can possibly be fixed in the OpenAL source. See last paragraph for a possible solution.

Update: if you are using a **NativeActivity**, and the app crashes on device = alcOpenDevice(NULL); please take a look at Garen's fix to the getEnv() method: **http://pielot.org/2010/12/14/openal-on-android/#comment-1160 < http://pielot.org/2010/12/14/openal-on-android/#comment-1160>**

Preparation

Understand how to compile NDK resources

This tutorial requires working with the Android NDK. We will have to compile OpenAL from source into a native Shared Object and then build a Java Native Interface to work with it. The techniques I use to work with the NDK (on Windows) have been described in a **previous tutorial < http://pielot.org/?p=312>** . You might want to take a look to understand what exactly I am doing.

Remember to PRESS F5 after you COMPILED the SHARED OBJECT. Otherwise, Eclipse will not use the new .so.

Create HelloOpenAL Project

Create a normal Android SDK Project.

```
package org.pielot.helloopenal;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class HelloOpenAL extends Activity {
 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 }

 private native int play(String filename);
}
```

Compile OpenAL for Android

The first step will be to compile OpenAL for Android. The goal will be to produce a Shared Object library libopenal.so that can be loaded into Android apps.

Download patched source of OpenAL

Thanks to Martins Mozeiko and Chris Robinson a version of OpenAL exists that has been adapted to the Android platform. Go to **http://repo.or.cz/w/openal-soft/android.git < http://repo.or.cz/w/openal-soft/android.git>** and download latest version of the patched OpenAL sourcecode. For this tutorial I used version that can be downloaded **here < http://repo.or.cz/w/openal-soft/android.git/snapshot/ea44b95252ce15dd38fb7563477e9e35b1c147dc.zip>** .

Extract into project folder. Rename top folder of downloaded source from 'android' to 'openal'.

Create config.h

To compile OpenAL a file called config.h is needed.Copy it from <PROJECT_HOME>/openal/android/jni to <PROJECT_HOME>/openal/include.

Create Android.mk

To tell the NDK compiler what files to compile, we now need to create Android.mk in <PROJECT_HOME>/jni . The file should contain:

```
TARGET_PLATFORM := android-3
ROOT_PATH := $(call my-dir)

############################################################
include $(CLEAR_VARS)

LOCAL_MODULE     := openal
LOCAL_ARM_MODE   := arm
LOCAL_PATH       := $(ROOT_PATH)
LOCAL_C_INCLUDES := $(LOCAL_PATH) $(LOCAL_PATH)/../openal/i
LOCAL_SRC_FILES  := ../openal/OpenAL32/alAuxEffectSlot.c \
 ../openal/OpenAL32/alBuffer.c        \
 ../openal/OpenAL32/alDatabuffer.c    \
 ../openal/OpenAL32/alEffect.c        \
 ../openal/OpenAL32/alError.c         \
 ../openal/OpenAL32/alExtension.c     \
 ../openal/OpenAL32/alFilter.c        \
 ../openal/OpenAL32/alListener.c      \
 ../openal/OpenAL32/alSource.c        \
 ../openal/OpenAL32/alState.c         \
 ../openal/OpenAL32/alThunk.c         \
 ../openal/Alc/ALc.c                  \
 ../openal/Alc/alcConfig.c            \
 ../openal/Alc/alcEcho.c              \
 ../openal/Alc/alcModulator.c         \
 ../openal/Alc/alcReverb.c            \
 ../openal/Alc/alcRing.c              \
 ../openal/Alc/alcThread.c            \
 ../openal/Alc/ALu.c                  \
 ../openal/Alc/android.c              \
 ../openal/Alc/bs2b.c                 \
 ../openal/Alc/null.c                 \

LOCAL_CFLAGS     := -DAL_BUILD_LIBRARY -DAL_ALEXT_PROTOTYPES
LOCAL_LDLIBS     := -llog -Wl,-s

include $(BUILD_SHARED_LIBRARY)

############################################################
```

Compile OpenAL

Now compile the source code using the NDK. I used a technique described in another tutorial on **Using cygwin with the Android NDK on Windows < http://pielot.org/?p=312>** . I am creating a batch file make.bat in the projects directory containing:

```
@echo on

@set BASHPATH="C:\cygwin\bin\bash"
@set PROJECTDIR="/cygdrive/d/dev/workspace-android/hellooper
@set NDKDIR="/cygdrive/d/dev/SDKs/android-ndk-r4b/ndk-build'

%BASHPATH% --login -c "cd %PROJECTDIR% && %NDKDIR%

@pause:
```

Save the file and execute it. If there is no error you have just compiled the OpenAL library into a Shared Object! You can find it in <PROJECT_HOME>/libs/armeabi. Now let’s see how we can make use of it.

The Native Interface

The next steps will be to create a Java Native Interface that allows us to access to OpenAL Shared Object.

Define Native Interface in Activity

Extend the HelloOpenAL Activity, so it looks like

```
package org.pielot.helloopenal;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class HelloOpenAL extends Activity {
 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);

 System.loadLibrary("openal");
 System.loadLibrary("openaltest");
 int ret = play("/sdcard/wav/lake.wav");
 Log.i("HelloOpenAL", ""+ret);
 }

 private native int play(String filename);
}
```

Implement Native Interface

In <PROJECT_HOME> execute

```
javah.exe -classpath bin -d jni org.pielot.helloopenal.Hello
```

to create the c header for the native function. Now <PROJECT_HOME>/jni should contain the file org_pielot_helloopenal_HelloOpenAL.h. Create org_pielot_helloopenal_HelloOpenAL.c in <PROJECT_HOME>/jni and fill it with

```
#include "org_pielot_helloopenal_HelloOpenAL.h"

JNIEXPORT jint JNICALL Java_org_pielot_helloopenal_HelloOpen
 (JNIEnv * env, jobject obj, jstring filename) {
 return 0;
}
```

Compile Native Interface

Add new library to Android.mk

```
############################################################

include $(CLEAR_VARS)

LOCAL_MODULE     := openaltest
LOCAL_ARM_MODE   := arm
LOCAL_PATH       := $(ROOT_PATH)
LOCAL_C_INCLUDES := $(LOCAL_PATH)/../openal/include
LOCAL_SRC_FILES  := org_pielot_helloopenal_HelloOpenAL.c

LOCAL_LDLIBS     := -llog -Wl,-s

LOCAL_SHARED_LIBRARIES := libopenal

include $(BUILD_SHARED_LIBRARY)

############################################################
```

Now compile again. The above created make.bat will do. You now should have two libraries in <PROJECT_HOME>/libs/armeabi/, namely libopenal.so and libopenalwrapper.so.

I you want you can run the app now. It should not crash and print 'HelloOpenAL 0' into the log.

Testing OpenAL

Now we have two libraries, one containing OpenAL and the other a native interface. We will now fill the latter with life to demonstrate the use of OpenAL.

Initialize and Release Audio Components

Therefore open org_pielot_helloopenal_HelloOpenAL.c and extend the existing code by:

```
#include "org_pielot_helloopenal_HelloOpenAL.h"

#include <stdio.h>
#include <stddef.h>
#include <string.h>
#include <AL/al.h>
#include <AL/alc.h>

JNIEXPORT jint JNICALL Java_org_pielot_helloopenal_HelloOper
 (JNIEnv * env, jobject obj, jstring filename) {

 // Global Variables
 ALCdevice* device = 0;
 ALCcontext* context = 0;
 const ALint context_attribs[] = { ALC_FREQUENCY, 22050, 0 }

 // Initialization
 device = alcOpenDevice(0);
 context = alcCreateContext(device, context_attribs);
 alcMakeContextCurrent(context);

 // More code to come here ...

 // Cleaning up
 alcMakeContextCurrent(0);
 alcDestroyContext(context);
 alcCloseDevice(device);

 return 0;
}
```

This code will now acquire the audio resource and release them. You should be able to compile the code and execute the HelloOpenAL app. However, nothing will yet happen, as we still have to load and play sound.

Methods for Loading Audio Data

Now we need to load audio data. Unfortunately, OpenAL does not come with functions for loading audio data. There has been the very popular ALut toolkit, but this is not part of OpenAL anymore. We therefore need to provide our own methods to load .wav files.

The following code snippets have been posted by **Gorax < http://www.gamedev.net/community/forums/profile.asp?mode=display&id=73484>** at **www.gamedev.net < http://www.gamedev.net/community/forums/topic.asp?topic_id=505152&whichpage=1&#3296091>** . These are one struct and two methods methods to load .wav data and buffer it in the memory.

Add the following code to org_pielot_helloopenal_HelloOpenAL.c. Add the code **above** JNIEXPORT jint JNICALL Java_org_pielot_helloopenal_HelloOpenAL_play

```
typedef struct {
 char  riff[4];//'RIFF'
 unsigned int riffSize;
 char  wave[4];//'WAVE'
 char  fmt[4];//'fmt '
 unsigned int fmtSize;
 unsigned short format;
 unsigned short channels;
 unsigned int samplesPerSec;
 unsigned int bytesPerSec;
 unsigned short blockAlign;
 unsigned short bitsPerSample;
 char  data[4];//'data'
 unsigned int dataSize;
}BasicWAVEHeader;

//WARNING: This Doesn't Check To See If These Pointers Are V
char* readWAV(char* filename,BasicWAVEHeader* header){
 char* buffer = 0;
 FILE* file = f open(filename,"rb");
 if (!file) {
 return 0;
 }

 if (f read(header,sizeof(BasicWAVEHeader),1,file)){
 if (!(//these things *must* be valid with this basic header
 memcmp("RIFF",header->riff,4) ||
 memcmp("WAVE",header->wave,4) ||
 memcmp("fmt ",header->fmt,4)  ||
 memcmp("data",header->data,4)
 )){

 buffer = (char*)malloc(header->dataSize);
 if (buffer){
 if (f read(buffer,header->dataSize,1,file)){
 f close(file);
 return buffer;
 }
 free(buffer);
 }
```

```
  }
  }
  f close(file);
  return 0;
 }

 ALuint createBufferFromWave(char* data,BasicWAVEHeader heade

  ALuint buffer = 0;
  ALuint format = 0;
  switch (header.bitsPerSample){
  case 8:
  format = (header.channels == 1) ? AL_FORMAT_MONO8 : AL_FORM
  break;
  case 16:
  format = (header.channels == 1) ? AL_FORMAT_MONO16 : AL_FOR
  break;
  default:
  return 0;
  }

  alGenBuffers(1,&buffer);
  alBufferData(buffer,format,data,header.dataSize,header.samp
  return buffer;
 }
```

WARNING, I had to put spaces into the word f open, f read, and f close. **Delete the spaces** when you copy that piece of code. For some reason, WordPress does not accept the words without space in a post.

Load Audio Data into Buffer

In method JNIEXPORT jint JNICALL Java_org_pielot_helloopenal_HelloOpenAL_play located the comment

```
// TODO More Code comes here
```

and replace it by

```
// Create audio buffer
 ALuint buffer;
 const char* fnameptr = (*env)->GetStringUTFChars(env, filen
 BasicWAVEHeader header;
 char* data = readWAV(fnameptr,&header);
 if (data){
 //Now We've Got A Wave In Memory, Time To Turn It Into A Us
 buffer = createBufferFromWave(data,header);
 if (!buffer){
 free(data);
 return -1;
 }

 } else {
 return -1;
 }

 // TODO turn buffer into playing source

 // Release audio buffer
 alDeleteBuffers(1, &buffer);
```

This piece of code tries to load PCM .wav audio data from the passed filename. The audio data is loaded into an OpenAL buffer. The buffer itself is merely the cached audio data but cannot be played. It therefore has to be attached to a sound source.

Create a playing source from the buffer

In method JNIEXPORT jint JNICALL Java_org_pielot_helloopenal_HelloOpenAL_play located the comment

```
// TODO turn buffer into playing source
```

and replace it by

```
 // Create source from buffer and play it
 ALuint source = 0;
 alGenSources(1, &source );
 alSourcei(source, AL_BUFFER, buffer);

 // Play source
 alSourcePlay(source);

 int        sourceState = AL_PLAYING;
 do {
 alGetSourcei(source, AL_SOURCE_STATE, &sourceState);
 } while(sourceState == AL_PLAYING);

 // Release source
 alDeleteSources(1, &source);
```

This piece of code creates a sound source from the buffer and plays it once.

Test on the Device

Compile the native code again by using make.bat. It should compile without errors. If you are using Eclipse, select the project HelloOpenAL in the Package Explorer and press F5. Otherwise, Eclipse might not be aware that the Shared Objects were updated.

Next, go to your devices SD Card and add a .wav file. I created a folder called “wav” and put a mono .wav file called “lake.wav” into this folder. Make sure it matches the filename you passed play(String filename) in the HelloOpenAL activity.

Now it time for the big test! Once you start the app, the .wav file should be played once.

This has been tested on the Nexus One and the G1/HTC Dream.

Solving Latency Issues

Some people seem to have experienced a 0.5 sec lag between triggering the sound and the actual playback. In the comments *aegisdino* suggested the following solution:

In alcOpenDevice() of ALc.c source,
"device->NumUpdates" seems to apply the lag issue.
In normal cases, device->NumUpdates will be 4, then I can feel about 0.5sec lag.
But when I fix it to 1, the lag disappeared.

I did not test the solution, but as NumUpdates was 1 in my version of ALc.c it could be the solution.

Share this:

< https://www.addtoany.com/share#url=https%3A%2F%2Fpielot.org%2F2010%2F12%2Fopenal-on-android%2F&title=OpenAL%20on%20Android>