

Premiers pas avec l'API audio OpenAL



par **Laurent Gomila** ([Autres articles](#))

Date de publication : 30/04/2007

Dernière mise à jour : 30/04/2007

Ce tutoriel aborde la programmation audio avec l'API OpenAL. Après avoir vu comment l'installer, nous détaillerons le fonctionnement d'OpenAL et verrons comment jouer très simplement un son. Ce tutoriel est écrit pour la version 1.1 d'OpenAL.



- 1 - Introduction
- 2 - Installation et paramétrage
- 3 - Initialisation et libération
- 4 - Charger un son
- 5 - Jouer un son
- 6 - Les extensions
- 7 - Conclusion

1 - Introduction


OpenAL est une API libre (distribuée sous licence LGPL) et multiplateforme offrant une gestion bas niveau de l'audio : sons 3D, flux, capture, sons multi-canaux (jusqu'à 7.1), effets, etc. Ceux qui connaissent OpenGL ne seront pas dépayés : la philosophie et la syntaxe sont exactement les mêmes.

OpenAL est à la base développé par Loki Entertainment et Creative Labs, mais chaque constructeur de chipset audio peut y ajouter des fonctionnalités propres à son matériel via un système d'extensions.

Utilisée par les plus grands studios de jeux vidéo (Id Software, Epic, ...) OpenAL peut d'ores et déjà être considérée comme un standard sûr et performant.

Pour une présentation plus détaillée, je vous renvoie vers la page d' **OpenAL sur Wikipedia**, ainsi que sur le site officiel  www.openal.org.

2 - Installation et paramétrage

La première chose à faire est de se rendre sur le site officiel :  www.openal.org. Vous y trouverez à peu près tout ce qu'il vous faut : téléchargements, documentation, wiki, ...

Rendez-vous dans la section *downloads* : vous y trouverez le code source (on peut le laisser de côté pour l'instant), ainsi que les versions de développement d'OpenAL pour les différents systèmes d'exploitation (Windows, Linux, MacOS X). Téléchargez celle qui vous intéresse puis installez-la. Si vous êtes sous Windows prenez le SDK (13.2 Mo) ; l'installateur lui ne contient que les DLLs OpenAL (une bonne idée est de le fournir avec votre programme, pour les utilisateurs qui n'auraient pas déjà installé OpenAL sur leur machine).

Ne paniquez pas si vous tombez sur une page Creative Labs en cliquant sur un fichier : le fichier en question se trouve au bas de la page, et le téléchargement pourra s'effectuer une fois que vous aurez pris connaissance de la licence.

Le SDK contient tout ce qu'il faut : la documentation de référence, les spécifications de l'API, les fichiers en-tête et bibliothèques, l'installateur, et des exemples en C++ (avec les fichiers projet Visual C++ 7.1 et 8 fournis en prime).

Vous trouverez également sur la page *downloads* des téléchargements pour ALUT (*OpenAL Utility Toolkit*), l'équivalent de GLUT pour OpenGL -- pour ceux qui connaissent. Cette bibliothèque contient assez peu de fonctions, principalement pour créer et charger des sons au format WAV depuis un fichier ou depuis des données en mémoire. Ici nous ne l'utiliserons pas, vous êtes donc libre de la télécharger ou non.



Une fois le SDK installé, vérifiez que tout fonctionne correctement en exécutant les exemples fournis dans le répertoire /samples

Une fois les fichiers de développement installés, il faut paramétrer votre environnement de programmation pour qu'il trouve les fichiers d'OpenAL. Pour cela, ajoutez le sous-répertoire */include* du SDK aux répertoires de recherche du compilateur, et le sous-répertoire */libs* aux répertoires de recherche de l'éditeur de liens. Sous Visual Studio par exemple, cela se trouve dans le menu "Tools", "Options...", "Projects and solutions", "VC++ directories".

3 - Initialisation et libération

Nous voilà donc prêts à écrire un premier programme audio avec OpenAL. Première chose à faire : modifier les options de votre projet pour lier avec la bibliothèque OpenAL (OpenAL32.lib sous Windows). Puis il vous faudra inclure les deux en-têtes OpenAL :

```
#include <al.h>
#include <alc.h>
```

<al.h> est l'en-tête principal d'OpenAL, <alc.h> quant à lui définit les fonctions de manipulation du contexte audio. Le contexte est l'environnement qui permet d'exécuter les fonctions audio, et est typiquement très spécifique au système d'exploitation et au driver. Pour ceux qui connaissent OpenGL, c'est là une véritable révolution puisque ALC gère toutes ces spécificités, sans que vous ayez à écrire une tonne de code non portable comme c'est le cas pour OpenGL. Afin de bien différencier ces deux concepts, les fonctions d'ALC sont préfixées par "alc", et les fonctions d'OpenAL par "al".

Voyons maintenant comment initialiser OpenAL. La première chose est d'ouvrir un device, puis de créer un contexte au sein de ce device. Si vous vous demandez ce que représente le device, on peut le voir comme le "périphérique" qui va être utilisé pour effectuer les sorties audio.

```
bool InitOpenAL()
{
    // Ouverture du device
    ALCdevice* Device = alcOpenDevice(NULL);
    if (!Device)
        return false;

    // Création du contexte
    ALCcontext* Context = alcCreateContext(Device, NULL);
    if (!Context)
        return false;

    // Activation du contexte
    if (!alcMakeContextCurrent(Context))
        return false;

    return true;
}
```

alcOpenDevice prend en paramètre le nom du device à ouvrir. Ici nous passons NULL pour choisir le device par défaut, mais il est possible de récupérer une liste des devices disponibles via la fonction alcGetString. En effet, si vous lui spécifiez l'option ALC_DEVICE_SPECIFIER, elle vous renverra une liste des devices disponibles sous forme d'une chaîne de caractères terminée par un double caractère nul, chaque device étant séparé par un simple caractère nul.

```
void GetDevices(std::vector<std::string>& Devices)
{
    // Vidage de la liste
    Devices.clear();

    // Récupération des devices disponibles
    const ALCchar* DeviceList = alcGetString(NULL, ALC_DEVICE_SPECIFIER);

    if (DeviceList)
    {
        // Extraction des devices contenus dans la chaîne renvoyée
        while (strlen(DeviceList) > 0)
```

```
    {  
        Devices.push_back(DeviceList);  
        DeviceList += strlen(DeviceList) + 1;  
    }  
}
```

Revenons à notre code d'initialisation, et plus particulièrement à la création du contexte. `alcCreateContext` prend en paramètre le device, puis un pointeur sur un tableau d'attributs. Ces derniers sont très peu utiles (en tout cas pour l'instant), nous pouvons donc passer NULL.


Il est également possible de créer plusieurs contextes, mais nous n'en aurons pas l'utilité ici, un seul étant largement suffisant.

Passons maintenant à la libération des ressources. Rien de magique : il s'agit de détruire le contexte, puis le device. Si vous ne les avez pas stockés, il est possible de les récupérer via les fonctions `alcGetCurrentContext` et `alcGetContextsDevice`.

```
void ShutdownOpenAL()  
{  
    // Récupération du contexte et du device  
    ALCcontext* Context = alcGetCurrentContext();  
    ALCdevice* Device = alcGetContextsDevice(Context);  
  
    // Désactivation du contexte  
    alcMakeContextCurrent(NULL);  
  
    // Destruction du contexte  
    alcDestroyContext(Context);  
  
    // Fermeture du device  
    alcCloseDevice(Device);  
}
```

4 - Charger un son

A présent que nous savons ouvrir et fermer correctement OpenAL, nous allons pouvoir faire des choses intéressantes, à savoir jouer des sons. Mais avant cela il faut les charger : en effet OpenAL ne fournit aucune fonctionnalité pour lire des fichiers audio. Si vous avez téléchargé ALUT vous pourrez grâce à lui charger des fichiers au format WAV, mais ici nous allons utiliser une autre bibliothèque : `libsndfile`. Un peu à l'image de `DevIL` pour les images, `libsndfile` permet de charger tout un tas de formats de fichiers audio (wav, raw, aiff, ...). Et pour couronner le tout, elle est gratuite, multiplateforme et open-source.

Rendez-vous sur le  [site officiel de libsndfile](#), et téléchargez les fichiers nécessaires au développement. Deux options vous sont proposées : télécharger les sources puis les recompiler (tout est fourni pour que ce soit le plus simple possible), ou alors si vous êtes sous Windows, téléchargez directement les fichiers précompilés. Si vous n'avez pas de fichier `.lib`, référez-vous au fichier `readme` et suivez les directives pour le générer.

Une fois muni des fichiers nécessaires à l'utilisation de `sndfile`, nous pouvons écrire une fonction pour charger n'importe quel fichier audio supporté par `libsndfile`.

Avant d'écrire cette fonction, il faut bien comprendre comment fonctionne OpenAL, et ce que nous allons faire des données chargées. OpenAL repose sur 3 concepts de base : les tampons (*sound buffer*), les sources (*sound source*) et l'écouteur (*sound listener*). Un tampon contient des données audio, ce que l'on appelle habituellement des échantillons. Une source est un moyen de jouer un tampon, avec des propriétés spécifiques (position 3D, volume, amplitude, ...) ; ainsi typiquement on peut placer plusieurs sources dans une scène pour jouer un ou plusieurs tampons audio. Quant à l'écouteur, il est toujours unique et représente l'utilisateur. Vous pouvez lui affecter les mêmes propriétés qu'une source : position 3D, volume, etc.

Pour l'instant nous ne voulons que charger des données audio, ce sont donc les tampons qui vont nous intéresser. A l'instar d'OpenGL, les tampons OpenAL sont identifiés par des entiers. Toutes les fonctions relatives aux tampons sont préfixées par `alBuffer`. Les habitués d'OpenGL reconnaîtront immédiatement les fonctions utilisées ici.

Ici nous commençons par ouvrir le fichier audio avec les fonctions de la bibliothèque `libsndfile`. Les paramètres sont le nom du fichier, le mode d'ouverture (lecture, écrire, ou lecture / écriture) puis un pointeur vers une structure à remplir avec les informations sur le son.

```
#include <sndfile.h>

ALuint LoadSound(const std::string& Filename)
{
    // Ouverture du fichier audio avec libsndfile
    SF_INFO FileInfos;
    SNDFILE* File = sf_open(Filename.c_str(), SFM_READ, &FileInfos);
    if (!File)
        return 0;
```

De ces informations nous pouvons extraire le taux d'échantillonnage (nombre d'échantillons à lire par seconde) ainsi que le nombre d'échantillons contenus dans le fichier.

```
// Lecture du nombre d'échantillons et du taux d'échantillonnage (nombre d'échantillons à lire
// par seconde)
ALsizei NbSamples = static_cast<ALsizei>(FileInfos.channels * FileInfos.frames);
ALsizei SampleRate = static_cast<ALsizei>(FileInfos.samplerate);
```

Nous pouvons donc ensuite lire les échantillons, toujours avec les fonctions de libsndfile. Ici nous récupérerons les échantillons sous forme d'entiers 16 bits signés : il s'agit du format le plus courant -- un peu comme les entiers non signés 32 bits pour les formats de pixels. libsndfile s'occupe de la conversion automatiquement, ainsi même si les échantillons sont stockés sous un autre format dans le fichier audio, vous pourrez les récupérer dans le format voulu.

```
// Lecture des échantillons audio au format entier 16 bits signé (le plus commun)
std::vector<ALshort> Samples(NbSamples);
if (sf_read_short(File, &Samples[0], NbSamples) < NbSamples)
    return 0;
```

Une fois les échantillons récupérés dans un tableau, nous pouvons fermer le fichier, nous n'aurons plus besoin de libsndfile.

```
// Fermeture du fichier
sf_close(File);
```

Avant de remplir un tampon OpenAL avec nos échantillons, il faut déterminer le format OpenAL de ceux-ci. De base il n'en existe que 4 :

- AL_FORMAT_MONO8
- AL_FORMAT_STEREO8
- AL_FORMAT_MONO16
- AL_FORMAT_STEREO16

Plus de formats sont disponibles (4.0, 5.1, 6.1, 7.1, ...) mais via le système d'extensions, que nous aborderons plus tard.

Ici ce sera donc AL_FORMAT_MONO16 ou AL_FORMAT_STEREO16, puisque nous avons récupéré nos échantillons sous forme d'entiers 16 bits. Afin de déterminer s'il s'agit d'un son mono ou stéréo, il faut consulter le nombre de canaux, récupérés par libsndfile. 1 canal signifiant un son mono, 2 signifiant un son stéréo.

```
// Détermination du format en fonction du nombre de canaux
ALenum Format;
switch (FileInfos.channels)
{
    case 1 : Format = AL_FORMAT_MONO16; break;
    case 2 : Format = AL_FORMAT_STEREO16; break;
    default : return 0;
}
```

Puis vient enfin la création du tampon à proprement parler. Comme déjà précisé, le tampon sera manipulé via un identificateur de type entier non-signé (ALuint).

```
// Création du tampon OpenAL
ALuint Buffer;
alGenBuffers(1, &Buffer);
```



Prenez l'habitude de manipuler les types définis par OpenAL, c'est un gage de portabilité et d'évolutivité.

Une fois le tampon généré à l'aide de `alGenBuffers`, vous pouvez le remplir avec `alBufferData`, à qui il faudra donner à manger toutes les données du son que nous avons pris soin de récupérer auparavant : format, échantillons, taille du tableau d'échantillons, et taux d'échantillonnage.

```
// Remplissage avec les échantillons lus
alBufferData(Buffer, Format, &Samples[0], NbSamples * sizeof(ALushort), SampleRate);

// Vérification des erreurs
if (alGetError() != AL_NO_ERROR)
    return 0;

return Buffer;
}
```

N'oubliez pas de gérer correctement les erreurs, en appelant `alGetError()` après un appel de fonction OpenAL. Différents codes d'erreur peuvent être renvoyés, vous pouvez consulter la documentation de référence pour voir lesquels chaque fonction peut générer, ainsi que leur signification.

5 - Jouer un son

Afin de jouer un son, il faut créer une source sonore. Une source fera référence à un tampon contenant les échantillons audio à jouer, auquel elle ajoutera des propriétés particulières telles que la position 3D, le volume, l'amplitude, etc.

Les fonctions pour manipuler les sources sont prefixées par `alSource`. Pour créer une nouvelle source, de la même manière que pour un tampon, il faudra appeler `alGenSources`.

```
// Création d'une source
ALuint Source;
alGenSources(1, &Source);
```

Nous pouvons ensuite attacher notre tampon à la source fraîchement créée.

```
// On attache le tampon contenant les échantillons audio à la source
alSourcei(Source, AL_BUFFER, Buffer);
```



Notez bien le "i" qui suffixe la fonction `alSourcei` : il indique que le paramètre passé sera de type entier. Cette convention s'applique à toutes les fonctions OpenAL qui changent / récupèrent un paramètre. Les autres suffixes possibles sont "3i" (3 entiers), "iv" (tableau d'entiers), "f" (flottant), "3f" (3 flottants), et "fv" (tableau de flottants).

Notre source est maintenant prête à jouer le son que nous avons chargé précédemment. Pour se faire, il suffit d'appeler la fonction `alSourcePlay`.

```
// Lecture du son
alSourcePlay(Source);
```

Il existe d'autres fonctions pour contrôler la lecture : `alSourceStop`, `alSourcePause`, et `alSourceRewind`. Les versions suffixées par "v" de ces fonctions existent également, elles prennent cette fois un tableau de sources pour le cas où vous voudriez effectuer une action sur plusieurs sources simultanément.

Une fois le son lancé, tout ce que nous ferons ici est d'attendre qu'il se termine. Afin de ne pas attendre sans rien faire, nous allons également afficher la position de lecture en secondes.

```
ALint Status;
do
{
    // Récupération et affichage de la position courante de lecture en secondes
    ALfloat Seconds = 0.f;
    alGetSourcef(Source, AL_SEC_OFFSET, &Seconds);
    std::cout << "\rLecture en cours... " << std::fixed << std::setprecision(2) << Seconds << "
sec";

    // Récupération de l'état du son
    alGetSourcei(Source, AL_SOURCE_STATE, &Status);
}
while (Status == AL_PLAYING);
```

Comme vous le voyez, il est possible de récupérer toute sorte d'informations intéressantes avec `alSource` (n'oubliez pas le bon suffixe en fonction du type du paramètre !). Ici nous récupérons la position de lecture en secondes avec `AL_SEC_OFFSET`, mais il aurait également été possible de la récupérer en octets ou en nombre d'échantillons

L'état de lecture de la source est quant à lui récupéré avec `AL_SOURCE_STATE`. Il existe 4 états : `AL_INITIAL`, `AL_STOPPED`, `AL_PLAYING`, et `AL_PAUSED`.

Une fois terminé, n'oubliez pas de détruire le tampon et la source.

```
// Destruction du tampon
alDeleteBuffers(1, &Buffer);

// Destruction de la source
alSourcei(Source, AL_BUFFER, 0);
alDeleteSources(1, &Source);
```

Avant de détruire la source n'oubliez pas de détacher le tampon (en mettant sa propriété `AL_BUFFER` à 0), sans quoi vous pourriez obtenir une erreur.

Enfin, nous avons également parlé tout à l'heure d'un écouteur : ici nous n'avons pas besoin de le définir, les propriétés par défaut étant suffisantes. Il faudra jouer avec celui-ci lorsque vous voudrez gérer des sons 3D (pour représenter position, vitesse et orientation de la caméra), ou pour influencer sur le volume global par exemple.

Pour modifier les paramètres de l'écouteur rien de bien compliqué, il faut appeler les fonctions préfixées par `alListener` :


```
// Définition de la position de l'écouteur (ici l'origine)
alListener3f(AL_POSITION, 0.f, 0.f, 0.f);

// Définition de la vitesse de l'écouteur (ici nulle)
alListener3f(AL_VELOCITY, 0.f, 0.f, 0.f);

// Définition de l'orientation de l'écouteur (ici il regarde vers l'axe des Z)
ALfloat Orientation[] = {0.f, 0.f, 1.f, 0.f, 1.f, 0.f};
alListenerfv(AL_ORIENTATION, Orientation);
```

6 - Les extensions

Tout comme OpenGL, OpenAL s'est fendu d'un système d'extensions afin de pouvoir supporter les fonctionnalités particulières de chaque constructeur de chipset audio, et de pouvoir évoluer sans avoir à sortir une nouvelle version du SDK à chaque modification. En gros, plutôt que d'utiliser une fonction / constante définie dans `al.h`, vous irez la demander directement au driver de manière dynamique avec les fonctions qui vont bien. Attention, il faudra bien distinguer les extensions propres au contexte (préfixées par `ALC`) du reste (préfixées par `AL`).

La liste des extensions disponibles classées par systèmes peut être consultée sur le site officiel :  **OpenAL extensions**.

Afin de lister toutes les extensions supportées par votre système, vous pouvez utiliser `alGetString` avec l'option `AL_EXTENSIONS`.

```
const ALchar* Extensions = alGetString(AL_EXTENSIONS);
```

Pour vérifier si une extension est bien supportée pas la peine de parcourir cette chaîne : la fonction `alIsExtensionPresent` le fera pour vous.

```
bool IsMultiChannelSupported = (alIsExtensionPresent("AL_EXT_MCFORMATS") == AL_TRUE);
```

Lorsqu'une extension est supportée, elle est généralement accompagnée de fonctions et / ou constantes, qu'il faudra aller chercher dynamiquement via les fonctions `alGetProcAddress` et `alGetEnumValue`. `alGetProcAddress` permet d'obtenir un pointeur vers une fonction, et `alGetEnumValue` permet de récupérer la valeur d'une constante.

Par exemple, si vous voulez gérer les formats ayant plus de deux canaux, vous pouvez récupérer des formats supplémentaires (si votre environnement le supporte) :

```
ALenum Format = 0;
switch (FileInfos.channels)
{
    case 1 : Format = AL_FORMAT_MONO16;           break;
    case 2 : Format = AL_FORMAT_STEREO16;         break;
    case 4 : Format = alGetEnumValue("AL_FORMAT_QUAD16"); break;
    case 6 : Format = alGetEnumValue("AL_FORMAT_51CHN16"); break;
    case 7 : Format = alGetEnumValue("AL_FORMAT_61CHN16"); break;
    case 8 : Format = alGetEnumValue("AL_FORMAT_71CHN16"); break;
}
```


Les extensions `ALC` (relatives au contexte donc) sont manipulées exactement de la même manière, excepté qu'il faudra utiliser les fonctions préfixées par `"alc"` plutôt que `"al"` (`alcIsExtensionPresent`, `alcGetProcAddress`, `alcGetEnumValue`).

7 - Conclusion

OpenAL est une vraie révolution au niveau de la programmation audio : elle permet de développer de manière abordable des applications audio de qualité sans se soucier des soucis de portabilité, et en évitant d'avoir recours à des moteurs pas toujours gratuits comme FModEx ou Bass. De plus elle sera très facile d'accès pour les développeurs qui utilisent OpenGL, comme vous avez pu le constater.

Pour ceux qui voudraient aller plus loin, d'autres tutoriels suivront abordant la lecture de flux ou encore la capture audio.

Le code source complet de ce tutoriel est disponible, avec les fichiers projets pour Visual Studio 2005 : [openal-src.zip \(201 Ko\)](#)

Si vous recherchez des sons ou musiques gratuits pour vos développements, pensez à faire un tour par notre  [page de ressources gratuites](#) !

Si vous avez des suggestions, remarques, critiques, si vous avez remarqué une erreur, ou bien si vous souhaitez des informations complémentaires, n'hésitez pas à me contacter !

