

# Utilisation des flux pour lire les fichiers OGG avec OpenAL



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

Date de publication : 07/05/2007

Dernière mise à jour : 07/05/2007

Dans ce tutoriel nous nous intéresserons à l'utilisation des flux audio avec OpenAL, et à leur application à la lecture de musiques au format OggVorbis.

- 1 - Introduction
- 2 - Ouverture et fermeture du fichier ogg
- 3 - Lecture des échantillons depuis le fichier OGG
- 4 - Alimenter un son pendant sa lecture
- 5 - Conclusion


## 1 - Introduction


Si vous débutez avec OpenAL, je vous invite à commencer par le **tutoriel d'introduction** avant d'aborder celui-ci.

Si vous avez bien suivi le tutoriel d'introduction, vous savez charger un son dans un tampon et le jouer à l'aide d'une source sonore. Cette technique sera très efficace pour des sons courts de type impacts, bruits de pas, etc. Mais comment faire pour lire des musiques de plusieurs minutes, occupant plusieurs dizaines de mégas en mémoire une fois décompressées ? Ou encore pour lire un son arrivant au fur et à mesure, par exemple d'internet ou du réseau local ? La solution tient dans ce que l'on appelle les flux (*stream* en anglais), et permettra d'alimenter une source sonore au fur et à mesure plutôt qu'avec un tampon contenant la totalité du son.

Nous allons ici illustrer ce procédé avec la lecture de musiques au format OGG (OggVorbis). Pour les rares qui ne le connaissent pas encore, le format OggVorbis est l'équivalent gratuit des MP3. Gratuit car il n'y a aucun brevet sur le format, et donc aucune histoire de droits pour le décoder. De plus, une bibliothèque de décodage open-source est fournie sur le site officiel.

## 2 - Ouverture et fermeture du fichier ogg

Tout d'abord rendez-vous sur le site officiel de OggVorbis :  [www.vorbis.com](http://www.vorbis.com) pour y télécharger la bibliothèque qui servira à décoder les fichiers OGG. Cliquez sur "développeurs", puis "downloads" dans le menu de gauche pour accéder à la page de téléchargements. Ici plusieurs bibliothèques et outils sont disponibles, mais tout ce qui nous intéresse sont les deux premières bibliothèques : libogg et libvorbis. Les fichiers compilés ne sont pas fournis, mais tout est fait pour que vous puissiez compiler ces deux bibliothèques en un clic. Sous Windows pas exemple, les fichiers Visual C++ 6 sont fournis, ainsi que des .bat pour compiler directement.

 Si vous cherchez des musiques gratuites au format OGG, vous pourrez en trouver quelques unes sur la page [music](#) du site officiel.

Une fois votre environnement de programmation correctement paramétré avec les chemins des en-têtes et bibliothèques de libogg et libvorbis, vous pourrez utiliser ces bibliothèques en incluant le fichier <vorbisfile.h> et en liant avec libogg, vorbis et vorbisfile. D'autres en-têtes et fichiers bibliothèques sont disponibles, mais ne nous seront pas utiles pour l'utilisation que nous allons en faire.

Vous êtes maintenant prêts à ouvrir un fichier OGG :

```
OggVorbis_File Stream;
FILE*      File      = NULL;
ALenum     Format     = 0;
ALsizei    SampleRate = 0;

bool OpenOgg(const std::string& Filename)
{
    // Ouverture du fichier
    File = fopen(Filename.c_str(), "rb");
    if (File)
        return false;

    // On ouvre un flux ogg-vorbis à partir du fichier que l'on vient d'ouvrir
    int Error = ov_open(File, &Stream, NULL, 0);
    if (Error < 0)
        return false;

    // Récupération des informations du son
    vorbis_info* Infos = ov_info(&Stream, -1);
    Format = Infos->channels == 1 ? AL_FORMAT_MONO16 : AL_FORMAT_STEREO16;
    SampleRate = Infos->rate;

    return true;
}
```

L'ouverture d'un fichier OGG se fait en deux temps : tout d'abord il faut ouvrir le fichier sous forme d'un FILE\* avec les fonctions de la bibliothèque standard C (fopen), puis ouvrir le flux OggVorbis sur ce fichier à l'aide de la fonction ov\_open. Notez que si vous voulez ouvrir votre flux OggVorbis d'une manière particulière, vous pouvez fournir des fonctions de rappel (*callback*) qui seront utilisées au lieu de celles par défaut :


```
size_t MyReadFunc(void *ptr, size_t size, size_t nmemb, void *datasource);
int MySeekFunc(void *datasource, ogg_int64_t offset, int whence);
int MyCloseFunc(void *datasource);
long MyTellFunc(void *datasource);

ov_callbacks Callbacks =
{
    &MyReadFunc,
    &MySeekFunc,
```

```
&MyCloseFunc,  
&MyTellFunc  
};  
  
ov_open_callbacks(Data, &Stream, NULL, 0, Callbacks);
```

Ici nous n'en auront pas besoin, les fonctions par défaut sachant très bien gérer un son OggVorbis stocké dans un fichier.

Une fois le fichier ouvert, nous pouvons récupérer des informations sur le son qui nous serviront à remplir nos tampons OpenAL, telles que le taux d'échantillonnage et le format (mono ou stéréo, déduit du nombre de canaux). Pour se faire il suffit d'appeler la fonction `ov_info`, qui renverra un pointeur vers une structure `vorbis_info` contenant (entre autres) les informations désirées.

 **Attention, les fichiers OggVorbis peuvent être encodés avec un taux d'échantillonnage variable (ce que l'on appelle VBR -- Variable Bit Rate), le code que nous allons écrire ne supporte pas de tels fichiers.**

La fermeture d'un fichier OGG est très simple : on clôture le flux OggVorbis, puis on ferme le fichier.

```
void CloseOgg()  
{  
    // Fermeture du flux ogg-vorbis  
    ov_clear(&Stream);  
  
    // Fermeture du fichier  
    fclose(File);  
}
```

### 3 - Lecture des échantillons depuis le fichier OGG

Une fois le fichier ouvert, il faudra pouvoir aller y lire nos échantillons sonores à plusieurs reprises afin d'alimenter des tampons OpenAL. Voici la fonction qui va réaliser la lecture :

```
void ReadOgg(ALuint Buffer, ALsizei NbSamples)
{
    // Tableau qui va recevoir les échantillons lus
    std::vector<ALshort> Samples(NbSamples);

    // Définition de variables utiles à la lecture
    ALsizei TotalRead = 0;
    ALsizei TotalSize = NbSamples * sizeof(ALshort);
    char* SamplesPtr = reinterpret_cast<char*>(&Samples[0]);

    // Tant qu'on n'a pas atteint la taille voulue, on lit
    while (TotalRead < TotalSize)
    {
        // Lecture des échantillons à partir du flux ogg-vorbis
        ALsizei Read = ov_read(&Stream, SamplesPtr + TotalRead, TotalSize - TotalRead, 0, 2, 1,
        NULL);

        if (Read > 0)
        {
            // La lecture a réussi, on avance du nombre d'octets lus
            TotalRead += Read;
        }
        else
        {
            // La lecture a échoué, on arrête de lire
            break;
        }
    }


    // Remplissage du tampon avec les données lues
    if (TotalRead > 0)
        alBufferData(Buffer, Format, &Samples[0], TotalRead, SampleRate);
}
```

La fonction importante ici est `ov_read`, c'est elle qui va aller récupérer les échantillons dans le fichier OGG ouvert. Ses paramètres sont les suivants :

- Un pointeur vers la structure associée au fichier ouvert
- L'adresse du tableau à remplir avec les échantillons lus
- La taille à lire en octets
- Le boutisme (*endianess*) : 0 pour little endian, 1 pour big endian
- La taille des échantillons en octets : ici nous manipulons des échantillons 16 bits, donc 2 octets
- Le signe des échantillons : 0 pour non signé, 1 pour signé
- Un pointeur vers un entier à remplir avec le numéro du bloc lu (cela peut être utile pour les fichiers à taux d'échantillonnage variable)

En cas de succès, `ov_read` renvoie le nombre d'octets lus.



Toutes ces informations peuvent être retrouvées dans la  **documentation de référence**.

Ici vous remarquez que nous effectuons une boucle pour lire plusieurs fois. Ceci est nécessaire, car `ov_read` ne lira au maximum qu'un bloc, ce qui peut représenter beaucoup moins que la taille que vous souhaitez réellement lire. Habituellement un bloc fait 4096 octets, ce qui serait très peu pour une lecture en flux.

Nous bouclons donc jusqu'à avoir lu la taille souhaitée, ou jusqu'à ce que `ov_read` renvoie 0 (fin de fichier atteinte) ou un nombre négatif (erreur), ce qui signifie dans les deux cas que nous devons arrêter la lecture.

Une fois le nombre d'échantillons voulu récupéré, nous pouvons alimenter un tampon OpenAL avec ceux-ci. Souvenez-vous que le format et le taux d'échantillonnage ont été récupérés lors de l'ouverture du fichier.

Voyons maintenant ce que nous allons faire de ces tampons, et comment utiliser la lecture en flux.

## 4 - Alimenter un son pendant sa lecture

La lecture en flux utilise les mêmes entités OpenAL qu'une lecture classique : une source sonore, et des tampons. Mais au lieu d'avoir cette fois un seul tampon contenant la totalité du son à jouer, nous aurons plusieurs tampons contenant une petite partie du son. Ces tampons sont organisés en pile : lorsqu'un tampon a été lu, il peut être dépilé, re-rempli avec les nouvelles données à lire, et réinjecté dans la pile.

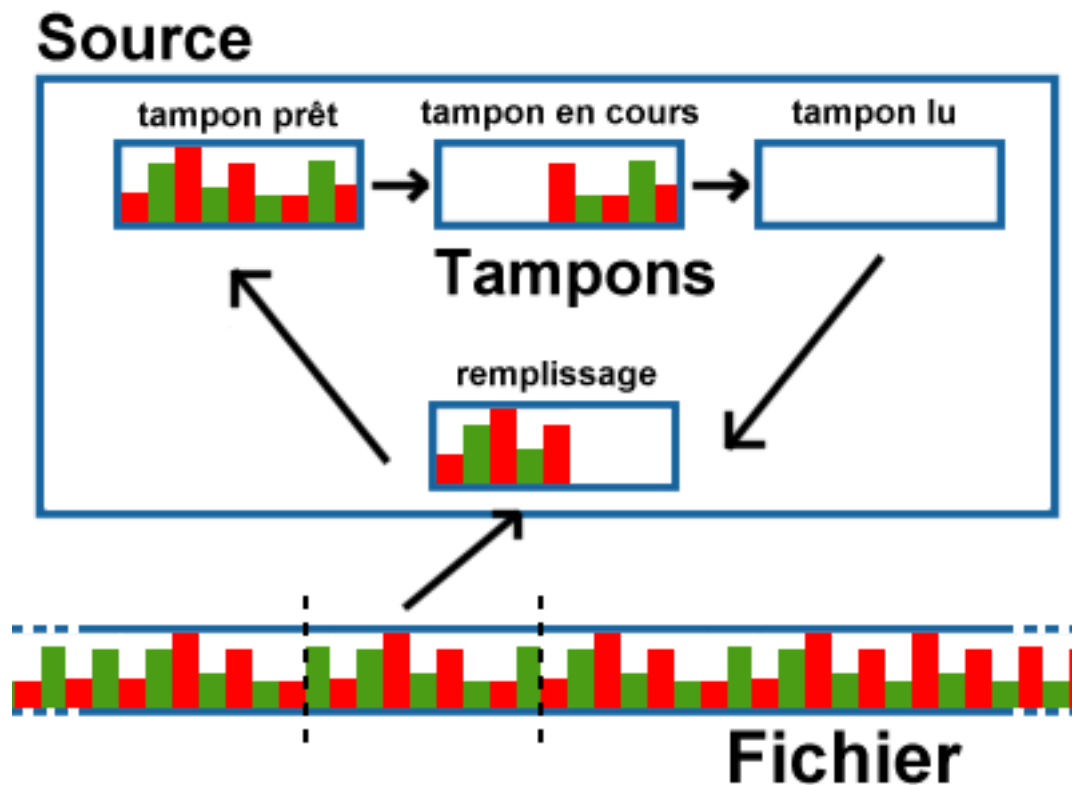


Schéma de fonctionnement de la lecture en flux

Il faudra donc veiller à ce que la lecture ne soit jamais interrompue, sous peine de causer des saccades. Le meilleur moyen pour cela est d'avoir des tampons de taille suffisante pour que les tampons vides aient largement le temps de se remplir. Nous pourrions également augmenter le nombre de tampons, mais cela surchargerait l'API audio pour le même résultat. En fait, seuls deux tampons sont nécessaires : l'un est lu pendant que l'autre se remplit. C'est exactement le même procédé que le *double-buffering* en programmation graphique : on passe à l'écran une surface à afficher pendant qu'on dessine sur l'autre.

Il n'y a aucune règle pour déterminer la taille optimale des tampons. L'important étant de ne pas choisir une taille trop faible, ni trop gigantesque. Un bon compromis est d'utiliser des tampons pouvant contenir 1 seconde d'échantillons, ce qui laisse largement le temps de lire les données et est loin de surcharger la mémoire. Par exemple pour un taux d'échantillonnage de 44100, nous aurions  $44100 \text{ (échantillons)} \times 2 \text{ (octets)} \times 2 \text{ (tampons)} = 172 \text{ Ko}$  de mémoire occupée, pour un fichier audio qui prendrait plusieurs dizaines de Mo s'il était chargé en totalité.

Passons à présent à la pratique. Nous commençons par générer les 2 tampons et la source sonore :

```
// Création des deux tampons OpenAL que nous allons utiliser pour la lecture en flux
ALuint Buffers[2];
alGenBuffers(2, Buffers);
```

```
// Création de la source qui jouera le son
ALuint Source;
alGenSources(1, &Source);
```

Puis nous remplissons nos deux tampons avec les premiers échantillons du fichier OGG :

```
// Remplissage des tampons avec le début du son à jouer
ReadOgg(Buffers[0], 44100);
ReadOgg(Buffers[1], 44100);
```

Enfin, nous injectons les tampons dans la file de la source à l'aide de la fonction `alSourceQueueBuffers`.

```
// On ajoute les 2 tampons à la file de la source
alSourceQueueBuffers(Source, 2, Buffers);
```

A ce stade la source est prête, et la lecture peut démarrer.

```
// On peut commencer la lecture
alSourcePlay(Source);
```

Une fois la lecture entamée, il va falloir boucler jusqu'à ce que le son soit stoppé, en effectuant régulièrement le dépileage / remplissage / réinjection des tampons afin d'assurer que la source ne sera jamais à cours de données.

```
ALint Status;
do
{
    // On récupère le nombre de tampons qui ont été traités (ie. qui sont prêts à être réutilisés)
    ALint NbProcessed;
    alGetSourcei(Source, AL_BUFFERS_PROCESSED, &NbProcessed);


    // On les re-remplit et on les réinjecte dans la file
    for (ALint i = 0; i < NbProcessed; ++i)
    {
        // On sort le tampon de la file
        ALuint Buffer;
        alSourceUnqueueBuffers(Source, 1, &Buffer);

        // On le remplit avec les données du fichier
        ReadOgg(Buffer, 44100);

        // On le réinjecte dans la file
        alSourceQueueBuffers(Source, 1, &Buffer);
    }

    // On récupère l'état de la source
    alGetSourcei(Source, AL_SOURCE_STATE, &Status);
}
while (Status == AL_PLAYING);
```

Le nombre de tampons qui ont été traités est récupéré à l'aide de la fonction `alGetSourcei` et de l'option `AL_BUFFERS_PROCESSED`. S'il y en a qui sont prêts, nous pouvons les dépiler (`alSourceUnqueueBuffers`), les remplir et les réinjecter (`alSourceQueueBuffers`).

 **Attention** : tous les tampons présents dans la pile d'une même source doivent tous avoir le même format et le même taux d'échantillonnage, sans quoi la fonction

*alSourceQueueBuffers* générera une erreur. C'est entre autres pour cela que nous ne pouvons pas gérer directement les musiques à taux d'échantillonnage variable.

Une fois la lecture stoppée, il faut purger la file de tampons avant de pouvoir détruire la source :

```
// On purge la file de tampons de la source
ALint NbQueued;
ALuint Buffer;
alGetSourcei(Source, AL_BUFFERS_QUEUED, &NbQueued);
for (ALint i = 0; i < NbQueued; ++i)
    alSourceUnqueueBuffers(Source, 1, &Buffer);
alSourcei(Source, AL_BUFFER, 0);
```

Attention, ici nous récupérons la totalité des tampons (AL\_BUFFERS\_QUEUED), pas seulement ceux qui sont vides (AL\_BUFFERS\_PROCESSED) ; même si dans notre cas ils devraient tous être vides puisque le son est arrivé à son terme.

Enfin, nous pouvons tout détruire proprement.


```
// On détruit les deux tampons
alDeleteBuffers(2, Buffers);

// On détruit la source
alDeleteSources(1, &Source);
```

## 5 - Conclusion

Voilà qui clôt ce tutoriel sur la lecture en flux. C'est un procédé qui requiert plus d'efforts que la lecture classique, mais il vous sera utile bien des fois. Ainsi il est conseillé d'utiliser les flux pour jouer les musiques de plusieurs minutes, afin de ne pas surcharger la mémoire. La lecture en flux sera également utile pour recevoir et jouer des sons ou des voix à partir du réseau.

Le code source complet de ce tutoriel est disponible, avec les fichiers projets pour Visual Studio 2005 : **[openal-flux-ogg-src.zip \(4.31 Mo\)](#)**

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 !

