

Effectuer des captures audio avec OpenAL



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

Date de publication : 14/05/2007

Dernière mise à jour : 14/05/2007

Dans ce tutoriel nous détaillerons le moyen de réaliser des captures audio à l'aide de l'API OpenAL, et de les sauvegarder dans des fichiers WAV.

- 1 - Introduction
- 2 - Ouverture et fermeture du device de capture
- 3 - Capturer le son
- 4 - Sauvegarde dans un fichier WAV
- 5 - Conclusion

1 - Introduction

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

Nous allons aborder ici la capture audio, c'est-à-dire le moyen d'enregistrer un son provenant de l'entrée de votre carte audio. OpenAL fournit pour cela un petit ensemble de fonctions bien utiles, que nous allons détailler sans plus attendre.

2 - Ouverture et fermeture du device de capture

Jusqu'à présent, vous deviez ouvrir un device ("périphérique") avant de pouvoir jouer des sons. La capture utilisant un périphérique différent, à savoir l'entrée de votre carte son (et non la sortie), il faudra ouvrir un nouveau device.

```
ALCdevice* Device;
ALCdevice* CaptureDevice;

// Device a été créé par l'appel à alcOpenDevice...

bool InitCapture()
{
    // On commence par vérifier que la capture audio est supportée
    if (alcIsExtensionPresent(Device, "ALC_EXT_CAPTURE") == AL_FALSE)
        return false;

    // Ouverture du device
    CaptureDevice = alcCaptureOpenDevice(NULL, 44100, AL_FORMAT_MONO16, 44100);
    if (!CaptureDevice)
        return false;

    return true;
}
```

Avant de tenter une quelconque opération de capture, il est possible de vérifier si le système supporte la capture audio via l'extension ALC_EXT_CAPTURE. Si elle n'est pas présente, vous pouvez abandonner tout de suite. Notez bien qu'il s'agit d'une extension de contexte (ALC), il faut donc la vérifier avec la fonction `alcIsExtensionPresent`. Le device qu'elle prend en paramètre est bien entendu l'autre, celui que vous avez créé pour initialiser OpenAL.



Toutes les fonctions de capture sont préfixées par `alcCapture`. Vous en aurez d'ailleurs vite fait le tour, puisqu'il n'en existe que 5.

L'ouverture du device de capture demande plusieurs paramètres. Le premier, à l'instar de `alcOpenDevice`, est le nom du périphérique de capture à utiliser. Ici nous pouvons passer `NULL` pour ouvrir le device par défaut. Notez qu'il est possible de récupérer la liste des devices de capture disponibles, avec la fonction `alcGetString` et l'option `ALC_CAPTURE_DEVICE_SPECIFIER`.

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

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

    if (DeviceList)
    {
        // Extraction des devices contenus dans la chaîne renvoyée
        while (strlen(DeviceList) > 0)
        {
            Devices.push_back(DeviceList);
            DeviceList += strlen(DeviceList) + 1;
        }
    }
}
```

Mais revenons à notre fonction `alcOpenDevice` et à ses paramètres.

Le deuxième paramètre est le taux d'échantillonnage (nombre d'échantillons à produire par seconde). Ici nous utilisons 44100, qui équivaut à un son de qualité CD.

Le troisième paramètre est le format des échantillons, il s'agit du même format demandé par `alBufferData`. Ici le son sera vraisemblablement mono, et nous voulons récupérer des échantillons codés sur 16 bits.

Enfin, le quatrième et dernier paramètre est la taille du tampon interne de capture. Trouver la bonne taille dépend beaucoup de votre application, cependant de manière générale évitez les tampons trop petits, cela pourrait causer des saccades. Ici nous choisissons un tampon de 44100 échantillons, capable donc de stocker une seconde d'enregistrement.

Avant de libérer OpenAL, il ne faudra pas oublier de fermer le device de capture.

```
void ShutdownCapture()  
{  
    // Fermeture du device de capture  
    alcCaptureCloseDevice(CaptureDevice);  
}
```

3 - Capturer le son

Une fois le device de capture correctement ouvert, nous pouvons démarrer la capture. Ceci se fait tout simplement avec la fonction `alcCaptureStart` :

```
// Lancement de la capture
alcCaptureStart(CaptureDevice);
```

Une fois la capture démarrée, il faudra récupérer les échantillons produits au fur et à mesure qu'ils seront disponibles. Ici nous allons les stocker dans un tableau d'échantillons (entiers 16 bits signés), afin de pouvoir les enregistrer par la suite dans un fichier. Notez que nous pourrions tout aussi bien écrire directement les échantillons dans le fichier au fur et à mesure que nous les récupérons, mais le passage par un tableau est ici plus commode. On pourrait également envoyer les échantillons sur le réseau pour les lire sur un autre ordinateur sous forme de flux, par exemple si nous voulions faire de la transmission de voix en temps réel par le réseau.

Enfin bref, ici nous n'utiliserons qu'un simple tableau.

```
// On va stocker les échantillons capturés dans un tableau d'entiers signés 16 bits
std::vector<ALshort> Samples;
```

Nous pouvons maintenant démarrer une boucle pour récupérer les échantillons capturés jusqu'à ce que l'enregistrement soit stoppé :

```
// ...Et c'est parti pour 5 secondes de capture
time_t Start = time(NULL);
while (time(NULL) - Start < 5)
{
    // On récupère le nombre d'échantillons disponibles
    ALuint SamplesAvailable;
    alcGetIntegerv(CaptureDevice, ALC_CAPTURE_SAMPLES, 1, &SamplesAvailable);

    // On lit les échantillons et on les ajoute au tableau
    if (SamplesAvailable > 0)
    {
        std::size_t Start = Samples.size();
        Samples.resize(Start + SamplesAvailable);
        alcCaptureSamples(CaptureDevice, &Samples[Start], SamplesAvailable);
    }
}
```

Pour faire simple, nous terminons ici l'enregistrement automatiquement au bout de 5 secondes.

La première chose à faire est de consulter le nombre d'échantillons disponibles, avec la fonction `alcGetIntegerv` et l'option `ALC_CAPTURE_SAMPLES`. S'il est positif (ie. s'il y a des échantillons en attente), nous les récupérons via la fonction `alcCaptureSamples`, qui prend en paramètre un pointeur sur le tableau à remplir ainsi que le nombre d'échantillons à récupérer.

Une fois les 5 secondes écoulées, nous pouvons stopper la capture. Cela se fait à l'aide de la fonction `alcCaptureStop` :

```
// On stoppe la capture
alcCaptureStop(CaptureDevice);
```

Puisqu'il s'est écoulé un petit moment, même infime, entre la dernière récupération et l'arrêt de la capture, nous vidons une dernière fois le tampon interne au cas où de nouveaux échantillons seraient prêts. Rien de nouveau ici, c'est un copier-coller de la boucle ci-dessus.

```
// On n'oublie pas les éventuels échantillons qu'il reste à récupérer
ALCint SamplesAvailable;
alcGetIntegerv(CaptureDevice, ALC_CAPTURE_SAMPLES, 1, &SamplesAvailable);
if (SamplesAvailable > 0)
{
    std::size_t Start = Samples.size();
    Samples.resize(Start + SamplesAvailable);
    alcCaptureSamples(CaptureDevice, &Samples[Start], SamplesAvailable);
}
```

4 - Sauvegarde dans un fichier WAV

Ici nous pourrions directement jouer le son capturé, en remplissant un tampon et en l'affectant à une source, mais voyons plutôt comme le sauvegarder dans un fichier audio.

La bibliothèque que nous allons utiliser ici est la même qui a servi à charger les fichiers audio dans le [tutoriel précédent](#) : libsndfile. C'est pourquoi je vous invite à relire le [chapitre correspondant](#) si vous n'avez pas installé correctement cette bibliothèque.

Voici la fonction de sauvegarde :

```
void SaveSound(const std::string& Filename, const std::vector<ALshort>& Samples)
{
    // On renseigne les paramètres du fichier à créer
    SF_INFO FileInfos;
    FileInfos.channels    = 1;
    FileInfos.samplerate  = 44100;
    FileInfos.format      = SF_FORMAT_PCM_16 | SF_FORMAT_WAV;

    // On ouvre le fichier en écriture
    SNDFILE* File = sf_open(Filename.c_str(), SFM_WRITE, &FileInfos);
    if (!File)
        return;

    // Ecriture des échantillons audio
    sf_write_short(File, &Samples[0], Samples.size());

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

Ici nous n'avons pas à faire appel à OpenAL, il s'agit uniquement de l'utilisation de libsndfile.

La première chose à faire est de remplir une instance de SF_INFO avec les paramètres du son : le nombre de canaux (1 puisque nous avons un son mono), le taux d'échantillonnage (rappelez-vous, nous avons choisi 44100) et le format. Ce dernier est une combinaison de deux champs : d'une part le format des échantillons (PCM 16 bits), d'autre part le format du fichier à créer (ici nous créons un fichier WAV).

Une fois ces informations remplies, nous pouvons ouvrir le fichier en écriture (il sera créé s'il n'existe pas).


Puis, on écrit les échantillons, qui sont pour l'instant dans notre tableau en mémoire, dans le fichier avec la fonction sf_write_short. Rien de compliqué ici. Si le format de fichier utilisé n'aime pas les échantillons sous forme d'entiers signés 16 bits aucun souci : libsndfile s'occupe de la conversion si besoin.

Enfin, n'oubliez pas de fermer le fichier. Et voilà, vous y êtes, vous pouvez enfin écouter la douce mélodie de votre voix !

5 - Conclusion

La capture audio n'est pas une tâche qui nécessite beaucoup de code, OpenAL nous mûche déjà assez remarquablement le travail. Ici nous pouvons même dire que nous avons fait un tour exhaustif de la capture : il n'existe en effet que les 5 fonctions que nous avons vues (pour l'instant tout du moins).

Le code source complet de ce tutoriel est disponible, avec les fichiers projets pour Visual Studio 2005 : [openal-capture-src.zip \(147 Ko\)](#)

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 !

