Utilisation de la bibliothèque FmodEx

par khayyam90

Date de publication : 16/05/2006

Dernière mise à jour : 16/05/2006

Cet article montre l'utilisation de la bibliothèque FmodEx, permettant de manipuler des fichiers son en C et en C++. Je vais y détailler le développement d'une classe de gestion de son.

I - Introduction

La bibliothèque FmodEx permet de manipuler simplement des fichiers sons (wav, mp3, ogg, wma ...). FmodEx peut être vue comme la Fmod 4, mais le nombre important de fonctions ajoutées ainsi que le design général qui a été revu lui ont donné un nouveau nom. C'est la raison pour laquelle vous ne trouverez pas de version 1, 2 ni même 3 de FmodEx. Les premières versions de FmodEx sont les versions Fmod.

FmodEx a l'énorme avantage d'être multi-plateforme. Elle a été implémentée pour fonctionner sur les systèmes Windows 32 et 64 bits, Linux 32 et 64 bits, Mac, Playstation 2 et 3, Xbox, GameCube ...

FmodEx est gratuite pour un usage non commercial. Dans le cas contraire, elle vous coûtera $100 pour un shareware amateur ou de $1500 à $6000 selon le type de diffusion commerciale que vous prévoyez d'en faire.

FmodEx possède une interface C++, que ne possédait pas Fmod. Ainsi, il est possible d'utiliser FmodEx en C et en C++. Toutes les fonctions ont été écrites pour ces 2 interfaces. Mais à l'heure actuelle, l'interface C++ ne fonctionne que sur les compilateurs MS Visual C++ / MS Visual Studio ; les autres compilateurs rencontrent des erreurs lors de l'édition de liens. Ceci sera peut-être corrigé dans les prochaines versions de FmodEx.

Donc, dans un souci de généricité, la classe que je vais développer va utiliser les fonctions de l'interface C.

II - Installation

Pour obtenir la FmodEx, rendez-vous sur le site officiel www.fmod.org, section download. L'installateur de la bibliothèque contient entre autres la documentation officielle de la FmodEx, tous les headers nécessaires à son utilisation, la bibiothèque compilée et des exemples des codes sources.

Copiez tous les fichiers d'entête dans un répertoire que votre compilateur saura trouver. /include/fmodex est un bon choix. Copiez aussi les fichiers de la bibliothèque compilée correspondant à votre compilateur. L'installateur de la bibliothèque comprend la bibliothèque déjà compilée pour les compilateurs MinGW, Visual C++, LCC et Borland C++ Builder. Copiez ceux qui vous conviennent dans un répertoire que votre compilateur saura trouver. /lib est un bon choix.

Pour utiliser l'interface C++ de FmodEx, il vous faudra utiliser les fichiers d'entête .hpp, et pour utiliser l'interface C, utilisez les fichiers .h

Une fois tous les fichiers copiés dans votre compilateur, il vous suffit de créer un projet pour votre compilateur préféré et de définir l'option d'édition de liens relative à FmodEx: -lfmodex.

Note : il est très facile de passer de l'interface C à l'interface C++ et vis et versa. Toutes les fonctions de l'interface C prennent en premier argument un pointeur vers la variable qui aurait appelé le traitement dans l'interface C++ ou bien directement cette variable. De plus, les noms sont très explicites quant aux classes C++ auxquelles une fonction de l'interface C fait référence. Exemple : FMOD_Sound_GetLength(FMOD_SOUND*, arguments) fait référence à FMOD::sound::getLength(arguments).

III - Ecriture de la classe de gestionnaire de sons

III-1 - Constructeur et destructeur

Notre classe permettra de créer un gestionnaire de sons, il nous faut donc tout d'abord les méthodes de création et de destruction.

La bibliothèque FmodEx comprend plusieurs classes de base, parmi lesquelles le système de sons, les sons, les canaux sonores... Et pour construire notre gestionnaire de sons, il va nous falloir créer un système de sons FmodEx et l'initialiser correctement. De même, pour détruire notre gestionnaire de sons, il va nous falloir détruire les objets que nous avons créés et appeler les méthodes propres à FmodEx pour fermer proprement tous les objets utilisés.

Les fonctions FmodEx à appeler lors de la construction de notre gestionnaire de sons sont FMOD_System_Create et FMOD_System_Init. La première construit un système de sons FmodEx. Elle prend un argument : un pointeur vers un système de son non alloué. La seconde fonction va initialiser FmodEx pour un système de sons donné. Elle a besoin de connaître le nombre de canaux à utiliser, le type d'initialisation, et des informations supplémentaires facultatives à transmettre à la carte son.

Autant de canaux créés signifie autant de morceaux que vous pourrez jouer simultanément (chacun sur un canal différent).

```
FMOD_SYSTEM  *sys;
FMOD_System_Create(&sys);
FMOD_System_Init(sys, 4, FMOD_INIT_NORMAL, NULL);
```

La libération de mémoire se fait en appelant la fonction FMOD_System_Release.

```
FMOD_System_Release(sys);
```

La variable sys a besoin d'être connue partout dans notre gestionnaire de sons, il faut donc le définir en attribut de classe.

Pour l'instant, notre classe ressemble à

```
class sound_mgr{
    public:
        sound_mgr();
        ~sound_mgr();
    private:
            FMOD_SYSTEM  *sys;
};
```

avec

```
sound_mgr::sound_mgr(){
    FMOD_System_Create(&sys);
    FMOD_System_Init(sys, 4, FMOD_INIT_NORMAL, NULL);
}

sound_mgr::~sound_mgr(){
    FMOD_System_Release(sys);
}
```

III-2 - Lecture de fichiers musicaux

FmodEx permet de lire un grand nombre de formats musicaux. Les commandes sont les mêmes quel que soit le type de fichier que vous manipulez.

FmodEx (et Fmod aussi) différencie un flux d'un son. Un son sera entièrement chargé en mémoire avant d'être lu alors que pour un flux, FmodEx va acquérir les données au fur et à mesure qu'il joue le flux. Un flux peut être un simple fichier sur votre disque dur ou bien un flux disponible sur internet. En effet, FmodEx permet de jouer des flux réseau. Toutefois, à vos risques et périls, si le réseau est lent ou surchargé, le lecture sera exécrable.

Un flux a donc cet avantage qu'il commence à être joué aussitôt après l'appel de la fonction apropriée. Un flux ou un son est toujours manipulé au moyen d'un handle de type FMOD_SOUND*. Ce pointeur sera alloué par la fonction FMOD_System_CreateStream ou FMOD_System_CreateSound selon que vous souhaitez manipuler un flux ou un son.

La lecture d'un flux demande donc d'abord la création d'un flux relatif à un fichier ou à une adresse réseau puis la lecture elle-même. Cette dernière opération a besoin d'un pointeur vers un canal audio (FMOD_CHANNEL*).

```
FMOD_SOUND   *sound;
FMOD_CHANNEL *channel;
FMOD_System_CreateStream(sys, "machin.mp3", FMOD_HARDWARE | FMOD_LOOP_OFF | FMOD_2D, 0, &sound);
FMOD_System_PlaySound(sys, FMOD_CHANNEL_FREE, sound, 0, &channel);
```

On remarque que le troisième argument de FMOD_System_CreateStream est une liste de drapeaux. FMOD_HARDWARE spécifie l'utilisation de l'accélération matérielle. FMOD_LOOP_OFF spécifie que le morceau ne sera joué qu'une fois et FMOD_2D spécifie qu'aucun traitement 3D ne sera pris en compte pour le rendu du son.

Le handle sur le canal audio permet de paramétrer et manipuler les options relatives à ce canal. Une idée intéressante est de créer autant de handles de canaux audios que vous avez créé de canaux lors de l'initialisation de FmodEx (à l'aide d'un std::vector<FMOD_CHANNEL*> par exemple).

Et une fois qu'un son ou un flux est terminé ou bien ne nous est plus nécessaire, il faut libérer sa mémoire allouée avec FMOD_Sound_Release(FMOD_SOUND*);

Toute lecture audio se fait dans un thread, ainsi la lecture n'est pas une opération bloquante.

III-3 - Navigation dans un fichier musical

Certaines fonctions peuvent être utiles : mettre en pause, reprendre, avancer, reculer ...

Une fois qu'un son ou un flux est en cours de lecture, il devient intimement lié au canal sur lequel il est joué. Les handles relatifs au son et au canal utilisé doivent donc être connus dans toute la classe.

La mise en pause et la reprise d'une lecture dépendent du canal audio qui joue le morceau et se font avec la fonction FMOD_Channel_SetPaused(channel, true/false);.

La longueur d'un morceau dépend du morceau lui-même tandis que la position courante dans un morceau dépend du canal audio qui le joue. L'acquisition de ces deux informations est très similaire :

```
unsigned int sound_mgr::getLength(){
    unsigned int i;
    FMOD_Sound_GetLength(sound, &i, FMOD_TIMEUNIT_MS);
    return i;
}
```

```
unsigned int sound_mgr::getCurrentPos(){
    unsigned int i;
    FMOD_Channel_GetPosition(channel, &i, FMOD_TIMEUNIT_MS);
    return i;
}
```

Le dernier paramètre des fonctions FMOD_Sound_GetLength et FMOD_Channel_GetPosition mérite qu'on s'y attarde. En effet, FmodEx nous permet d'obtenir ces durées de plusieurs manières : en millisecondes, en partitions PCM, en octets... FMOD_TIMEUNIT_MS correspond au type millisecondes.

Les fonctions avancer et reculer utilisent la fonction FMOD_Channel_SetPosition. Pour cela, il faut récupérer la position courante et regarder si l'emplacement à atteindre existe. Si oui, alors on effectue le dépacement. De même que pour FMOD_Sound_GetLength, FMOD_Channel_SetPosition permet d'utiliser plusieurs unités de temps.

```
void sound_mgr::avancer(unsigned int i){
    unsigned int c=getCurrentPos();
    unsigned int l=getLength();
    if(c+i<l)
        FMOD_Channel_SetPosition(channel, c+i,FMOD_TIMEUNIT_MS );
}

void sound_mgr::reculer(unsigned int i){
    unsigned int c=getCurrentPos();
    if(c-i>0)
        FMOD_Channel_SetPosition(channel, c-i,FMOD_TIMEUNIT_MS );
}
```

III-4 - Acquisition des TAGS MP3

Les tags MP3 permettent de stocker dans le fichier même des informations telles que le nom de l'artiste, le titre du morceau, le nom de l'album ...

FmodEx nous a fait des fonctions très pratiques pour extraire ces informations, en particulier FMOD_Sound_GetTag. Son utilisation est très simple, cette fonction prend en arguments un pointeur vers un canal audio, le nom d'un tag ou le numéro d'un tag ainsi qu'un pointeur sur une structure pour recevoir la valeur du tag.

```
string sound_mgr::getArtist(){
    FMOD_TAG tag;
    FMOD_Sound_GetTag(sound, "ARTIST", 0, &tag);
    string s((char*)tag.data);

    return s;
}
```

De même pour obtenir le titre d'un morceau :

```
string sound_mgr::getTitle(){
    FMOD_TAG tag;
    FMOD_Sound_GetTag(sound, "TITLE", 0, &tag);
    string s((char*)tag.data);

    return s;
}
```

III-5 - Les callbacks

Voilà un sujet très intéressant. Un callback est une fonction qui sera appelée lors d'un évènement particulier, par

exemple si un morceau est terminé, ou bien s'il débute...

Les fonction à appeler ont des profils différents selon l'évènement auquel elles se rapportent. Un callback peut dépendre d'un canal ou bien du système tout entier. Voici un exemple de gestion de callback relatif à un canal donné.

```
FMOD_RESULT F_CALLBACK onEnd(FMOD_CHANNEL *channel,
                             FMOD_CHANNEL_CALLBACKTYPE type,
                             int c,
                             unsigned int c1,
                             unsigned int c2){
        // traitement
    return FMOD_OK;
}
```

Cette fonction callback sera branchée sur l'évènement "fin d'un son" par

```
FMOD_Channel_SetCallback(channel, FMOD_CHANNEL_CALLBACKTYPE_END,  onEnd, 0);
```

Les différents paramètres du callback sont :

- un pointeur sur le canal qui a généré l'évènement
- le type d'évènement qui a appelé le callback
- la valeur passée en dernier argument de FMOD_Channel_SetCallback
- 2 entiers dont les valeurs sont assignées par le callback lui-même. Ces valeurs dépendent de l'évènement et ne sont utiles que dans de très rares cas (en gros, elles ne servent jamais)

Attention ! Un callback ne peut être invoqué que si le système de sons réévalue les évènements. Cette réévaluation se fait avec la fonction FMOD_System_Update(sys). Donc, si vous avez une boucle qui tourne tout le long de votre programme, il faudra y ajouter un FMOD_System_Update(sys) (ou une fonction propre à la classe sound_mgr qui l'appelera). Si vous n'avez pas de telle boucle, il va falloir vous arranger pour invoquer FMOD_System_Update(sys) d'une manière ou d'une autre. Quitte à créer un thread qui va appeler FMOD_System_Update(sys) par exemple tous les dixièmes de seconde. Cette dernière solution n'est pas propre, mais je n'ai pas trouvé d'autre moyen de réévaluer les évènements.

Les callbacks ne sont donc pas très pratiques à utiliser dans FmodEx. Si vous avez besoin de tester des évènements tels que la fin d'un morceau, il devient préférable de tester par vous-même si un canal est toujours en train de jouer ou non. Pour cela, utilisez la fonction FMOD_CHANNEL_isPlaying(FMOD_CHANNEL, bool*) qui met à jour le booléen passé en paramètre selon que le canal est en train de jouer ou non.

IV - Effets sonores

Voici des exemples d'effets sonores proposés par FmodEx

IV-1 - Echo

L'écho correspond à une répétition d'un son peu après son émission. Le paramètre principal de cet effet est la durée séparant le son et son écho. Pour créer un effet sonore, il faut enregistrer la variable relative à cet effet sonore auprès du système de sons.

```
FMOD_DSP *echo;
FMOD_System_CreateDSPByType(sys, FMOD_DSP_TYPE_ECHO, &echo);

FMOD_DSP_SetParameter(echo, FMOD_DSP_ECHO_DELAY, 15); // écho de 15 ms
FMOD_System_AddDSP(sys, echo);
```

Ceci aura pour effet de rajouter de l'écho sur tous les sons joués par le système de sons. Si vous souhaitez ne mettre de l'écho que sur certains canaux, il faut utiliser de la même manière FMOD_Channel_AddDSP(channel, echo)

Et pour retirer l'écho, il faudra faire FMOD_DSP_Remove(echo)

IV-2 - Filtrage de fréquences

Le filtrage de fréquences permet de ne jouer que les fréquences comprises entre 2 valeurs. En réalité, il s'agit de ne jouer que les fréquences qui sont supérieures à une valeur d'une part et que celles qui inférieures à une autre valeur d'autre part.

Le filtrage s'effectue donc avec 2 effets sonores. Le filtrage par limite haute est un effet de type FMOD_DSP_TYPE_LOWPASS. Cet effet ne jouera que les fréquences inférieures à une limite donnée.

```
FMOD_DSP *lim_haute;
FMOD_System_CreateDSPByType(sys, FMOD_DSP_TYPE_LOWPASS, &lim_haute);
FMOD_DSP_SetParameter(lim_haute, FMOD_DSP_LOWPASS_CUTOFF, 1000);
FMOD_System_AddDSP(sys, lim_haute);
```

Ainsi, seules les fréquences inférieures à 1000 Hz seront jouées. De même, pour fixer une limite basse :

```
FMOD_DSP *lim_basse;
FMOD_System_CreateDSPByType(sys, FMOD_DSP_TYPE_HIGHPASS, &lim_basse);
FMOD_DSP_SetParameter(lim_basse, FMOD_DSP_HIGHPASS_CUTOFF, 5000);
FMOD_System_AddDSP(sys, lim_basse);
```

Où seules les fréquences supérieures à 5000 Hz seront jouées.

V - Téléchargement

- sound_mgr.h : le fichier d'entête de notre classe
- sound_mgr.cpp : le corps de notre classe

VI - Liens et remerciements

VI-1 - Liens

- www.fmod.org le lien magique. Attention, la doc officielle de FmodEx n'est pas en ligne. Pour l'instant le site ne met en ligne que la doc de Fmod. La doc officielle de FmodEx se situe dans l'installateur FmodEx, en section download.

VI-1 - Remerciements

Je tiens à remercier toute l'équipe des sections C/C++ et Jeux, et en particulier fearyourself, pour la relecture de cet article.