Correction des problèmes de son sous Xspect

Gauthier LEBRUN

19 juin 1997

1 Introduction

Xspect se doit de fonctionner sous SGI et ALPHA. Comme ce sont deux architectures différentes, la gestion du son n'est pas la même. Ainsi, sous ALPHA la gestion du son est laissée à la charge d'un serveur audio dont on peut fixer la fréquence de sortie (actuellement à 44.1 kHz).
Cette différence de gestion du son implique donc deux écritures de la fonction `playFromSelection`. Pour indiquer au compilateur quelle fonction il faut compiler, le fichier play.c comporte deux parties :

```
#if defined(__sgi)
        :
        :
        Code pour SGI
        :
        :
#elif defined(__alpha)
        :
        :
        Code pour ALPHA
        :
        :
#endif
```

2 Machines ALPHA : fusion des sources de playsf

De nombreux problèmes existaient lorsqu'on jouait des sons à partir de Xspect. Ainsi, sur les machines ALPHA, les sons étaient joués à la vitesse d'échantillonnage du serveur audio : un son échantillonné à 11kHz joué sur un serveur à 44kHz était entendu 4 fois trop vite...

De plus, on désire pouvoir régler le volume directement à partir de Xspect. Cette fonctionalité avait été implantée pour les machines SGI, mais elle ne l'était pas pour les machines ALPHA.

Rolf Woehrman a développé le programme playsf qui permet de jouer tous les sons quelque soient leur fréquence d'échantillonnage, leur type (float ou short), et indépendamment de l'architecture sur laquelle ils ont été conçus (en Big ou Little Endian).

Ce programme fonctionnant bien, notamment pour les machines ALPHA, je m'en suis donc largement inspiré pour réécrire la fonction `playFromSelection` de Xspect.

2.1 Fonction playFromSelection (play.c)

Cette fonction permet de jouer un segment de son. Elle est appelée par la fonction *callback* `XsCbPlayFromSelection`, en réponse à un appel au menu Marks/Play Segment.

Prototype : `void playFromSelection(View *view,int iselFrom)`

- **view :** spécifie la vue contenant le buffer et la sélection. Il nous permet de récupérer l'adresse du buffer et le numéro du canal à jouer. En effet, la fonction `playFromSelection` ne joue qu'un canal.
- **iselFrom :** spécifie le numéro de la sélection.

Une fois la sélection récupérée, j'initialise le serveur audio par la fonction `InitAudio`.

Je calcule ensuite le début et la fin du segment (en position en secondes), puis j'initialise une mémoire tampon de lecture (bufferToPlay) positionnée au début du segment. Enfin, j'effectue une boucle de lecture des échantillons. La sortie audio se fait dans cette boucle par la fonction `writeAudio`.

C'est dans cette boucle que ce font également différentes conversions :

- Conversion en Short par la fonction `STconvertToShortBuffer`.
- Si nécessaire, une inversion de l'ordre des octets (conversion Big Endian en Little Endian), par la fonction `STswap16`.

Lorsque le buffer a été bien converti, on peut modifier le volume de sortie en multipliant chaque échantillon par la variable level. Cette variable récupère la valeur output Level de la fenêtre Context/Play.

2.2 Fonctions InitAudio et writeAudio

Ces fonctions proviennent de playsf.

`void Initaudio (Buffer *buffer)` : Cette fonction crée une connection vers le serveur indiqué par la variable AUDIOFILE en fonction du taux d'échantillonnage du buffer. C'est dans cette fonction qu'on va initialiser (s'il y a lieu) la variable audioConversion qui va permettre de convertir le buffer à jouer à la bonne fréquence d'échantillonnage.

`int writeAudio (char *buf, int samples, int sampleWidth)` : Cette fonction envoie un nombre de *samples* échantillons (contenu dans la variable buf). Si la fréquence de l'échantillon est différente de celle du serveur, il faut convertir cet échantillon de manière à ce qu'il soit audible. C'est la fonction `AConvert`. Puis, la variable buf est envoyée au serveur audio : fonction `AFPlaySamples`.

2.3 Nouvelles bibliothèques

Pour pouvoir intégrer les sources de playsf (fonctions InitAudio, writeAudio), de nouvelles bibliothèques ont dûes être inclues au Makefile. Ainsi, ces fonctions utilisent la bibliothèque ST (pour *Sound Tools*), que Rolf Woehrman a développée. Cette bibliothèque se trouve actuellement dans `/u/sol/woehrman/src/STtools/libST`.

Pour utiliser les fonctions AConvertAllocInit et Aconvert, il faut inclure la bibliothèque aconvert. On peut la trouver dans `/u/formes/src/xspect/aconvert` avec ses sources.

Le fichier Makefile de Xspect a donc été modifié de manière à tenir compte de ces nouvelles bibliothèques :

```
LIB_STOOLS_DIR   =       /u/sol/woehrman/src/STtools/libST/\$(ARCH)
LIB_PRIV_DIR     =       /u/formes/src/xspect/aconvert

INCL_PRIV_DIR    =       /u/formes/src/xspect/aconvert
INCL_STOOLS_DIR  =       /u/sol/woehrman/src/STtools/include
```

> *ATTENTION : L'ordre des bibliothèques sur la ligne de compilation est important. Ainsi, il faut d'abord mettre les bibliothèques les plus dépendantes en premier lieu, suivies des plus indépendantes. Par exemple, la bibliothèques* aconvert *ne doit pas se trouver APRÈS les bibliothèques* AF *et* AFUtil *car elle les utilise. On indiquera donc sur la ligne de commande la liste des bibliothèques suivantes :* cc ... -laconvert -lAF -lAFUtil

3 Correction des bugs des machines SGI

La lecture des sons sur les machines SGI fonctionnait bien pour quasiment tous les sons sauf pour ceux échantillonné en flottants, et nécessitant une inversion des octets (Big $\rightleftarrows$ Little Endian).

Après étude du code existant, je me suis apperçu que la fonction `copyShortToPlay` effectue la conversion en short et l'inversion éventuelle de l'ordre des octets.
Il s'est avéré qu'elle était éronnée dans le cas des échantillons en float. En effet, 2 erreurs se trouvaient dans la partie DATA_TYPE_FLOAT :

1. La multiplication par 32768 était effectuée sur le pointeur et non sur la valeur pointée par celui-ci ! D'où (gros) problème...

2. La multiplication se faisait avant le swapFloat. Cette fonction est utilisée pour inverser l'ordre des octets, lorsque cela est nécessaire. Il faut donc effectuer la multiplication APRES l'inversion des octets.

J'ai donc aboutit au code suivant (où `pvalues` pointe sur le buffer à jouer) :

```
case DATA_TYPE_FLOAT:
   if (buffer->swap)
      for (i = 0; i < length && pvalues < pvaluesMax; i++)
      {
         float tmp = *((float *) pvalues);
         swapFloat(&tmp);
         tmp = tmp * 32768.0 * level;

         pvalues += step;
         *(ptr++) = (short) tmp;
      }
   else
      for (i = 0; i < length && pvalues < pvaluesMax; i++)
      {
         *(ptr++) = (short) (*((float *) pvalues) * 32768.0 * level);
         pvalues += step;
      }
break;
```