
TD 1 : représentation temporelle du son

Les fichiers de ce TD se trouvent à l'adresse suivante :

<http://dept-info.labri.fr/~sm/Licence/INF106/TD1>

1 Outils pour le son

1.1 Mixeur

Plusieurs “mixeurs” sont disponibles et permettent d’effectuer plusieurs réglages concernant votre carte son (le volume notamment). Nous pouvons citer comme exemples de mixeurs : `alsamixer` (projet ALSA), `aumix` (mode texte), `kmix` (projet KDE), `gmix` (projet GNOME), `xmix`, *etc.*

Exercice 1 : Utiliser un (ou plusieurs) mixeur(s) de façon à régler correctement le volume général. Certains mixeurs gèrent séparément le volume de la sortie PCM. Faut-il aussi tenir compte de ce volume (en plus du volume principal), et pourquoi ?

1.2 Jouer des fichiers sonores

Exercice 2 : Tester les commandes `play`, `esdplay`, `wavplay`, `soxplay` à partir de fichiers sons (et vérifier en même temps si ces commandes sont disponibles...). Ces commandes permettent de jouer le fichier son passé en paramètre.

1.3 Conversion de formats

La commande `sox` permet de convertir de nombreux formats sonores.
Tester `sox -help` pour voir les options possibles.

Exercice 3 : Convertir un fichier `aiff` au format `wav`. Écouter les deux fichiers. Existe-t-il une différence ?

Exercice 4 : Convertir un fichier `wav` en un fichier `raw`. Regarder la différence de taille entre les deux fichiers. Expliquer.

Exercice 5 : Convertir un fichier `raw` en un fichier `wav`. Quelles informations est-il nécessaire de préciser ? Pourquoi ? Trouver le format du fichier `son.raw`.

1.4 Compression sans perte...

Compresser un fichier au format `wav` ou `aiff` en utilisant `gzip` par exemple. Essayer sur `harmonica.wav` et `toms.aiff`. Conclusions ?

1.5 Taux d'échantillonnage optimal

Faire :

```
sox toms.aiff -r 22050 tmp.aiff
play toms.aiff
play tmp.aiff
```

Comparer la qualité (en écoutant) et la taille des fichiers `toms.aiff` et `tmp.aiff`. Conclusions?
Même chose mais avec 11025 au lieu de 22050. La qualité est-elle toujours aussi bonne ? Expliquer.

2 Développement en langage C

Nous allons à présent développer un module en C d'ouverture et d'écriture de fichiers au format wav.

2.1 Type `trame`

Nous définissons comme une `trame` une portion de N échantillons de la représentation temporelle d'un son.

Exercice 6 : Comment cela s'exprime-t-il en langage C, si chaque échantillon est un réel compris entre -1.0 et 1.0 ? Dans la suite, on fixera $N = 1024$.

2.2 Conversion au format brut

Nous souhaitons à présent lire les échantillons d'un fichier son au format wav (échantillonné à 44100 Hz, 16 bits et monophonique). Pour cela, nous allons dans un premier temps supprimer l'en-tête à l'aide de `sox`, de façon à créer un fichier *temporaire* au format brut.

Exercice 7 : Trouver la commande complète utilisant `sox` et qui permet d'effectuer cette conversion.

2.3 Ouverture de fichier

Nous allons écrire une fonction `sound_file_open_read` qui effectue l'ouverture d'un fichier son dont le nom est passé en paramètre.

Dans un premier temps cette fonction doit convertir le fichier wav en fichier au format brut `raw`. La fonction `system` permet d'exécuter une commande système dans un programme C. Par exemple, le programme suivant effectue un listing des fichiers du répertoire courant :

```
#include <stdlib.h>

int main ()
{
    system ("ls -l");
    return EXIT_SUCCESS;
}
```

Exercice 8 : Écrire la partie de la fonction `sound_file_open_read` qui permet la conversion d'un fichier wav en un fichier temporaire `raw` que nous appellerons `tmp-in.raw`.

Ensuite, la fonction doit ouvrir en lecture le fichier `raw` créé. Pour cela, utiliser la fonction `fopen` qui prend en paramètres un nom de fichier et un mode d'ouverture (ici `"rb"` pour la lecture (*read*, *r*) de données binaires (*b*)).

```
#include <stdio.h>

int main ()
{
    FILE *fp;
    fp = fopen ("test.raw", "rb");
    ...
}
```

Exercice 9 : Écrire la partie de la fonction `sound_file_open_read` qui permet l’ouverture en lecture du fichier temporaire `tmp-in.raw`.

2.4 Lecture du fichier

Nous allons à présent écrire une fonction `sound_file_read` qui recopie dans une trame les N échantillons du fichier. Cette fonction prend donc en paramètres un pointeur de fichier et une trame.

La fonction `fread` permet de remplir un buffer `ptr` de `nmemb` éléments de taille `size` octets dans le fichier dont le pointeur est `fp`. Elle retourne le nombre d’éléments lus (s’il n’y a pas d’erreur, cela doit être `nmemb`) :

```
#include <stdio.h>

int main ()
{
    FILE *fp;
    short tmp[100];
    int n;
    ...
    n = fread (tmp, sizeof(short), 100, fp);
    if (n != 100)
        { /* erreur ou fin de fichier atteinte */ }
    ...
}
```

Exercice 10 : Écrire la fonction `sound_file_read` qui récupère les échantillons du fichier temporaire `raw` et les copie dans un tableau de réels. Attention : les échantillons lus (de type `short`, entiers signés sur 16 bits) doivent ensuite être convertis en “réels” (type `double`) compris entre -1.0 et 1.0 .

2.5 Fermeture du fichier

A la fin de la lecture de tous les échantillons, le fichier temporaire `raw` peut être fermé. La fonction `fclose` doit être utilisée : elle prend en paramètre le pointeur du fichier ouvert précédemment.

```
#include <stdio.h>

int main()
{
    FILE *fp;
    ...
    fclose (fp);
}
```

Exercice 11 : Écrire la fonction `sound_file_close_read` qui ferme le fichier temporaire.

2.6 Affichage des échantillons

L’utilitaire `gnuplot` peut être utilisé pour afficher facilement des courbes (ici la représentation temporelle d’une trame). Il suffit pour cela de créer un fichier de coordonnées, constitué d’une suite de lignes au format : abscisse ordonnée `<RET>`, comme dans l’exemple suivant (fichier `fonction.txt`) :

```
0 0
1 1
2 4
3 9
```

Puis, depuis l'interprète gnuplot :

```
plot 'fonction.txt' with lines
pause 2 "Attendre 2 secondes..."
```

Ces commandes peuvent être regroupées dans un script `script.gp`, puis l'appel depuis le *shell* se fait ainsi :

```
gnuplot < script.gp
```

et l'appel depuis le langage C :

```
system ("gnuplot < script.gp");
```

Il existe aussi un module d'interface C très pratique, appelé `gnuplot_i.h`.

Dans un premier temps, il faut créer une fonction `plot_init` qui va initialiser l'affichage. Cette fonction définit un contrôleur de type `gnuplot_ctrl *` (dans l'exemple ci-dessous la variable `h`). Ensuite, le style de la courbe peut être défini à l'aide de la fonction `gnuplot_setstyle`.

```
#include "gnuplot_i.h"

static gnuplot_ctrl *h = NULL;

...
h = gnuplot_init ();
gnuplot_setstyle (h, "lines");
```

Exercice 12 : Écrire la fonction `plot_init` d'initialisation de l'affichage.

Ensuite, il faut préciser l'axe des abscisses en définissant un tableau de réels (par exemple `x_axis`). La fonction `gnuplot_plot_xy` prend en paramètres le contrôleur, l'axe des abscisses, l'axe des ordonnées (valeurs à afficher, ici les échantillons), le nombre de valeurs, et enfin le nom du graphe (chaîne de caractères).

```
gnuplot_plot_xy (h, x_axis, data, N, "son");
```

Exercice 13 : Écrire la fonction `plot_temporal` d'affichage des échantillons. Attention : il est nécessaire une fois la courbe affichée à l'écran de suspendre l'exécution du programme (pour avoir le temps de voir le résultat). Vous pourrez faire appel à la fonction `sleep` qui prend en paramètre un entier indiquant un nombre de secondes.

2.7 Test d'affichage des échantillons

Exercice 14 : En utilisant les fonctions précédentes, écrire un programme principal (fonction `main`) qui affiche les N (par exemple $N = 1024$) premiers échantillons d'un son dont le nom est passé en paramètre. Par exemple, la commande `main son.wav` doit afficher les N premiers échantillons de `son.wav` (représentation temporelle du son).

2.8 Représentation temporelle du son complet

Nous souhaitons à présent afficher successivement la représentation temporelle d'un son au format `wav`, par segments de N échantillons. Ainsi, l'affichage d'un son composé de $10 \times N$ échantillons doit donner lieu à l'affichage successif de 10 graphes. Chaque graphe est alors affiché durant un temps limité (2 secondes par exemple). Ensuite, le graphe suivant est affiché durant le même temps, et ainsi de suite.

Exercice 15 : Modifier le programme précédent de façon à pouvoir afficher les représentations temporelles successives d'un son dont le nom est passé en paramètre.

2.9 Ouverture de fichier en écriture

Nous souhaitons proposer des fonctions qui permettent de sauvegarder un son au format wav. Pour cela, nous allons reprendre le principe de la lecture d'un fichier wav à l'envers.

Nous allons d'abord écrire une fonction `sound_file_open_write` qui effectue l'ouverture en écriture d'un fichier son dont le nom est passé en paramètre. Nous utiliserons cette fonction pour écrire les échantillons dans un fichier temporaire `tmp-out.raw`. Pour cela le principe est le même que dans la deuxième partie de la fonction `sound_file_open_read`. La seule différence ici provient du mode d'ouverture (paramètre de la fonction `fopen`) qui est ici "wb", écriture (*write*, w) de données binaires (b).

Exercice 16 : Écrire la fonction `sound_file_open_write`.

2.10 Écriture du fichier

Nous allons à présent écrire une fonction `sound_file_write` qui recopie les N échantillons d'une trame dans un fichier. Cette fonction prend donc en paramètre un descripteur de fichier et une trame.

La fonction `fwrite` permet de remplir le fichier dont le pointeur est `fp` avec `nmemb` éléments de taille `size` octets contenus dans le buffer `ptr`. Elle retourne le nombre d'éléments écrits (s'il n'y a pas d'erreur, cela doit donc être exactement `nmemb`) :

```
#include <stdio.h>

int main()
{
    FILE *fp;
    double tmp[100];
    int n;
    ...
    n = fwrite (tmp, sizeof(double), 100, fp);
    if (n != 100)
        { /* erreur */ }
    ...
}
```

Exercice 17 : Écrire la fonction `sound_file_write` qui récupère les échantillons du tableau de réels d'une trame et les copie dans le fichier temporaire. Attention : les échantillons (de type double, compris entre -1.0 et 1.0) doivent d'abord être convertis en entiers signés sur 16 bits (type short).

2.11 Fermeture du fichier

À la fin de l'écriture de tous les échantillons, le fichier temporaire peut être fermé. La fonction `fclose` doit être utilisée à nouveau.

Exercice 18 : Écrire la fonction `sound_file_close_write` qui ferme le fichier temporaire.

2.12 Conversion au format wav

Le fichier temporaire doit ensuite être converti au format wav. Pour cela, comme pour la conversion inverse, il est possible de faire appel à la commande `sox`.

Exercice 19 : Trouver la commande complète utilisant `sox` et qui permet d'effectuer cette conversion.

Exercice 20 : Écrire la partie de la fonction `sound_file_close_write` qui permet la conversion du fichier temporaire `tmp-out.raw` en un fichier au format wav.

Exercice 21 : Vérifier la validité du fichier obtenu avec la commande `file`, qui vérifie le format du fichier passé en paramètre. En profiter pour vérifier le format du fichier `son.java`.

Exercice 22 : Lire le fichier `wav` obtenu sous `emacs`, en passant en mode hexadécimal (`M-x hexl-mode`), et vérifier sa structure. . .

3 Sortie sur la carte son

Exercice 23 : Étudier le programme `silence.c` et le modifier pour générer, à la place de la minute de silence, une sinusoïde de fréquence 440 Hz et d'amplitude 0.5 pendant 1 seconde, dont l'équation est :

$$a(t) = 0.5 \sin(2\pi 440t) \quad (0 \leq t < 1)$$