Projet informatique : décomposition en ondelettes

IN 101 – ENSTA

`http://www.di.ens.fr/~pointche/enseignement/ensta/projet/`

Résumé

Le but de ce projet est de manipuler, au travers des images et donc de données bidimensionnelles, la décomposition d'un signal sur une base d'ondelettes. Cette décomposition nous servira à représenter l'image à l'aide d'un faible nombre de coefficients (moins de 1% de l'image d'origine) et à nous apercevoir que malgré cette compression, le contenu de l'image reste perceptible. Deux bases d'ondelettes seront envisagées, l'*ondelette de Haar* qui nous servira d'exemple, puis l'*ondelette de Daubechie*.

L'image de gauche représente l'image originale. À droite, on a représenté l'image obtenue par reconstruction du signal d'ondelettes de Haar après avoir gardé 0.677% *des points.*

1 Modalités de ce projet

Ce projet peut être effectué par binôme (2 personnes). Son évaluation sera effectuée sur la base

- d'une disquette ou d'un e-mail avec une archive compressée (`.tar.gz` ou `.tgz`) regroupant
  - les sources (veiller à la clarté des sources, et ne pas hésiter à commenter chaque étape) ;
  - un `Makefile` qui effectue la compilation de l'executable `projet`.

  Ce dernier sera testé sur l'image `lena.gif` (voir ci-dessus) de taille 512 sur 512, ainsi que sur une image quelconque (sauf avertissement contraire de votre part) ;
- d'un rapport écrit, d'au plus 4 pages, détaillant vos réflexions et les problèmes rencontrés ;
- d'un listing du code source, ou des modules essentiels (au plus 400 lignes ou 4 pages).

Le tout doit être remis le **21 décembre**, avec le contrôle de connaissances, pour les documents physiques, par e-mail pour le reste (également le 21 décembre, 12h00, au plus tard).

Consulter la page `http://www.di.ens.fr/~pointche/enseignement/ensta/projet` pour les fichiers à charger, ainsi que toute information supplémentaire. Une liste de questions/réponses y sera proposée.

2 Présentation du projet

2.1 La décomposition en ondelettes

De manière intuitive, la décomposition en ondelettes permet de séparer un signal en bandes de fréquences. À cette fin, plutôt que d'utiliser différents filtres passe-bande, on utilise un filtrage passe-haut et un filtrage passe-bas que l'on applique de manière itérative. La décomposition revient à représenter le signal d'une manière différente à chaque étape.

Nous allons désigner un signal numérique par le vecteur $\vec{s} = \{s_0, s_1, \ldots, s_k, \ldots, s_{2n+1}\}$. Lors de la première étape de la décomposition, le signal $\vec{s}$ est transformé comme suit :

$$\vec{s} = \{s_0, s_1, s_2, s_3, \ldots, s_{2p}, s_{2p+1}\} \longmapsto (\{m_0, \ldots, m_p\}, \{d_0, \ldots, d_p\}) = (\vec{m}, \vec{d}), \tag{1}$$

où $\vec{m}$ est le vecteur des valeurs ¡¡ moyennes ¿¿ et $\vec{d}$ est le vecteur des valeurs ¡¡ détails ¿¿. Dans la décomposition sur la base d'ondelettes de Haar les ¡¡ détails ¿¿ et les ¡¡ moyennes ¿¿ sont reliés au signal par les formules suivantes :

$$\forall\ 0 \leq k \leq p, \qquad m_k = \frac{s_{2k} + s_{2k+1}}{\sqrt{2}} \text{ et } d_k = \frac{s_{2k} - s_{2k+1}}{\sqrt{2}}. \tag{2}$$

Dans le cas général, on aura une combinaison linéaire d'ordre n du signal :

$$\forall\ 0 \leq k \leq p, \qquad m_k = \sum_{i=0}^{n-1} P_i s_{2k+i} \text{ et } d_k = \sum_{i=0}^{n-1} Q_i s_{2k+i}. \tag{3}$$

Remarque : Lorsque les indices sortent des bornes, on les fait revenir à zéro. En d'autres termes, $2k + i$ est pris modulo $2p + 2$ (dont le résultat est compris entre 0 et $2p + 1$).

La décomposition de Haar est donc celle réalisée en prenant :

$$\begin{cases} \texttt{P[0]} = 1/\sqrt{2} = \texttt{0.70710678118654752440} & \texttt{Q[0] = P[0]} \\ \texttt{P[1]} = 1/\sqrt{2} = \texttt{0.70710678118654752440} & \texttt{Q[1] = -P[1]} \end{cases}$$

Une fois la première transformation réalisée, le processus de réarrangement est réitéré sur les vecteurs des ¡¡ moyennes ¿¿ jusqu'à épuisement des termes : en considérant le vecteur initial des ¡¡ moyennes ¿¿ $\vec{m}^0 = \{m_0^0, \ldots, m_{2^n-1}^0\} = \vec{s}$:

$$\left|\begin{array}{rcll} \vec{m}^0 & \longmapsto & (\{m_0^1, m_1^1, \ldots, m_{2^{n-1}-1}^1\}, \{d_0^1, d_1^1, \ldots, d_{2^{n-1}-1}^1\}) & = (\vec{m}^1, \vec{d}^1) \\ \vec{m}^1 & \longmapsto & (\{m_0^2, m_1^2, \ldots, m_{2^{n-2}-1}^2\}, \{d_0^2, d_1^2, \ldots, d_{2^{n-2}-1}^2\}) & = (\vec{m}^2, \vec{d}^2) \\ \ldots & & \ldots & \\ \vec{m}^{n-2} & \longmapsto & (\{m_0^{n-1}, m_1^{n-1}\}, \{d_0^{n-1}, d_1^{n-1}\}) & = (\vec{m}^{n-1}, \vec{d}^{n-1}) \\ \vec{m}^{n-1} & \longmapsto & (\{m_0^n\}, \{d_0^n\}) & = (\vec{m}^n, \vec{d}^n) \end{array}\right. \tag{4}$$

Si lors des transformations successives, seul le vecteur des moyennes est remplacé par sa transformation,

$$\begin{aligned} \vec{s} = \vec{m}^0 \mapsto (\vec{m}^1, \vec{d}^1) \mapsto ((\vec{m}^2, \vec{d}^2), \vec{d}^1) \mapsto \ldots &\ \longmapsto\ ((\ldots((\vec{m}^k, \vec{d}^k), \vec{d}^{k-1}), \ldots), \vec{d}^1) \\ \ldots &\ \longmapsto\ \ldots \mapsto ((\ldots((\vec{m}^n, \vec{d}^n), \vec{d}^{n-1}), \ldots), \vec{d}^1), \end{aligned}$$

à l'issue de la dernière transformation, le signal d'origine $\vec{s}$ est donc représenté par le terme final de ¡¡ moyenne ¿¿ m_0^n et tous les termes de ¡¡ détails ¿¿,

$$\{d_0^n\}, \{d_0^{n-1}, d_1^{n-1}\}, \ldots, \{d_0^k, d_1^k, \ldots, d_{2^{n-k}-1}^k\}, \ldots, \{d_0^2, d_1^2, \ldots, d_{2^{n-2}-1}^2\}, \{d_0^1, d_1^1, \ldots, d_{2^{n-1}-1}^1\}.$$

La reconstruction du signal se fera elle aussi de manière itérative en recomposant à partir de la dernière transformation, les moyennes du rang $p - 1$ étant calculées à partir des moyennes et des détails du rang p. Dans le cas de l'ondelette de Haar, on aura $m_k^{p-1} = \left(m_k^p + d_k^p\right)/\sqrt{2}$.

2.2 Première étape

- Écrire une fonction permettant de réaliser la transformation décrite dans l'équation (1). On supposera que la longueur du signal est une puissance de 2, soit $\vec{s} = \{s_0, ...s_{2^n-1}\}$. Cette fonction utilisera la décomposition de Haar, soit l'équation (2), pour calculer les moyennes et les détails. Cette fonction devrait suivre le prototype
  ```
  void wave_haar_step(double *input, int size)
  ```
  et modifiera le vecteur (un tableau de `double`) passé en argument `input`.
- Écrire une fonction `void wave_haar(double *input, int size)` appelant de manière itérative la fonction précédente pour réaliser la transformation complète décrite dans la formule (4). On remarquera que cela revient à appeler la fonction `wave_haar_step` en divisant `size` par 2 entre chaque appel. En effet, à chaque étape, on ne touche pas aux valeurs comprises au-delà de `size`. On ne considère que les valeurs entre `0` et `size-1`, sur lesquelles on effectue la transformation. Les nouvelles moyennes sont stockées dans le tableau initial aux indices entre `0` et `size/2-1`, et les nouveaux détails entre `size/2` et `size-1`.
- Écrire une fonction réalisant la transformation inverse, en utilisant plusieurs appels à la fonction réciproque de `wave_haar_step`, qu'il faudra tout d'abord définir :
  ```
  void iwave_haar_step(double *input, int size)
  ```

Vérifier le bon fonctionnement des fonctions en constatant que l'on retrouve bien le signal d'origine. Comment peut-on généraliser les fonctions précédentes à des signaux de longueur quelconque ?

3 Gestion des images

3.1 Représentation des images

Nous admettrons que la décomposition en ondelettes d'une image revient à réaliser itérativement une étape de la décomposition selon les lignes puis une étape de la décomposition selon les colonnes. Nous conviendrons de représenter les images par un tableau unidimensionnel. Une image de largeur **L** et de hauteur **H** sera donc représentée par un tableau de **L**×**H** pixels. Nous conviendrons de plus de respecter les conventions du langage `C` en faisant démarrer les indices de lignes ou de colonnes à zéro. Ainsi le pixel situé en haut à gauche de l'image possédera les coordonnées `(0,0)` et celui situé en bas à droite de l'image les coordonnées `(L-1,H-1)`.

3.2 Utilitaires à votre disposition

Les fichiers mis à votre disposition sur la page web du projet permettent de simplifier la manipulation des images. Il existe ainsi des fonctions de lecture/écriture d'images au format GIF[1] et des fonctions d'allocation/libération d'images. En plus de ces fonctions, sont définies plusieurs macros permettant d'accéder aux champs des structures. Toutes ces macros manipulent des pointeurs sur les structures images.

- Dans `image_defs.h` sont définies trois structures d'images :
  - type `IMAGE`, possédant des niveaux de gris codés sur un octet ;
  - type `DIMAGE`, dont le niveau de gris est codé sur un `double` ;
  - type `CIMAGE`, dont la couleur de chaque pixel est codée sur trois octets, pour le rouge, le vert et le bleu.

  C'est le type `DIMAGE` que nous utiliserons. Il est fourni avec les macros suivantes :
  - `LARG(im)` : la largeur de l'image, soit le nombre de colonnes qui la composent,

[1] Graphic Interchange Format est une marque déposée de Compuserve

  - `HAUT(im)` : la hauteur de l'image, soit le nombre de lignes qui la composent,
  - `Pix(im,i,j)` : le niveau de gris du pixel de la ligne `j` et de la colonne `i`, sachant que les lignes et les colonnes commencent à zéro,
  - `MINI(im)` : le plus petit niveau de gris ;
  - `MAXI(im)` : le plus grand niveau de gris.

  Attention à l'ordre des coordonnées dans l'utilisation de `Pix(im,i,j)`.
- dans `image_func.h` sont définis les prototypes des fonctions permettant de gérer ces formats d'images. Les fonctions étant définies dans le module `image_func.c`. Notamment, les fonctions de lecture/écriture au format GIF :
  - `DIMAGE *read_GIF_file_DIMAGE(char *name)` — lecture d'une image au format GIF. Elle retourne un pointeur sur une structure `DIMAGE` fraîchement allouée codant l'image au format GIF contenue dans le fichier `name`.
  - `int write_GIF_file_DIMAGE(char *name, DIMAGE *im)` — écriture d'une image au format GIF dans le fichier `name`. La fonction retourne 0 si tout se passe bien et une valeur non nulle en cas d'erreur. L'image n'est pas modifiée.
- La librairie GIF est fournie sous forme d'une archive `libgif.a` (compilée pour PC/Linux), avec son fichier d'en-têtes `gif_lib.h`. Cependant, elle est peut-être déjà installée sur votre système, dans quel cas vous n'avez rien à faire, sinon copiez ces fichiers dans votre répertoire courant.

Vous pourrez alors compiler chaque module séparément :

```
gcc -c -Wall -I. image_func.c
gcc -c -Wall projet.c
```

Votre projet sera dans le module `projet.c`, qui inclura les headers,

```
#include "image_defs.h"
#include "image_func.h"
```

mais vous pouvez également le séparer en plusieurs sous-modules.

Enfin, vous construisez l'exécutable en éditant les liens entre les différents modules et les librairies mathématique `-lm` et GIF `-lgif`. Pour cela, vous utilisez la commande :

```
gcc image_func.o projet.o -o projet -L. -lm -lgif
```

Les options de compilation `-I.` et `-L.` précisent au compilateur qu'il peut aussi chercher, respectivement, le header à inclure (par `#include <gif_lib.h>`, d'où le `-I.`), et la librairie GIF (à charger par `-lgif`, d'où le `-L.`), dans le répertoire courant (ce qui est le cas lorsque vous avez copié ces fichiers, pas nécessaire dans le cas contraire).

4 La décomposition des images

4.1 Décomposition dans la base d'ondelettes de Haar

Afin de décomposer une image dans la base d'ondelettes de Haar, on peut réutiliser la fonction `wave_haar()` en lui apportant quelques modifications afin d'obtenir deux fonctions :

- `void wave_haar_h(DIMAGE *im, int JJ, int size)` qui réalise une étape de la décomposition selon les lignes pour la ligne `JJ`,
- `void wave_haar_v(DIMAGE *im, int II, int size)` qui réalise une étape de la décomposition selon les colonnes pour la colonne `II`.

En supposant que l'image `im` soit carrée (*i.e.* `H = L`), La décomposition totale sera obtenue en suivant le synopsis présenté figure 1.

```
size = LARG(im);
tant que (size != 0) faire {
  pour toutes les lignes JJ
    wave_haar_h(im,JJ,size)
  pour toutes les colonnes II
    wave_haar_v(im,II,size)
  diviser size par 2
}
sauver l'image
```

FIG. 1 – Décomposition d'une image bidimensionnelle

Écrire la fonction permettant de réaliser la décomposition totale pour une image possédant une largeur et une hauteur qui soient l'une et l'autre des puissances de deux. Comment généraliser ce qui vient d'être fait pour des images de taille quelconque ?

4.2 Élimination des détails

Une fois l'image décomposée dans la base de Haar, il est possible d'éliminer les détails en donnant une valeur nulle aux pixels dont le niveau est inférieur à un certain seuil.

Écrire un programme

```
haar <input.gif> <output.gif> <seuil>
```

- qui charge une image dont le nom du fichier `input.gif` est passé en paramètre de la ligne de commande ;
- décompose cette image dans la base de Haar ;
- met à zéro les détails dont la valeur est inférieure à un seuil `seuil` passé sur la ligne de commande (gérer un compteur du nombre de pixels ainsi supprimés, pour avoir une idée du taux de compression obtenu) ;
- recompose l'image ;
- sauvegarde l'image recomposée au format GIF dans le fichier `output.gif` passé sur la ligne de commande.

Faites varier le seuil et analysez l'évolution de l'image obtenue. Quelles conclusions pouvez-vous en tirer ?

5 Une base d'ondelettes de Daubechie

Le but est maintenant d'améliorer la qualité de la décomposition en utilisant une base d'ondelettes plus riche, celle de Daubechie d'ordre 3. Il suffit de remplacer dans la formule générale (3) les valeurs de P et Q :

$$\begin{cases} P[0] = 0.33267055295999998 & Q[0] = P[5] \\ P[1] = 0.80689150931099995 & Q[1] = -P[4] \\ P[2] = 0.45987750211799999 & Q[2] = P[3] \\ P[3] = -0.13501102001000001 & Q[3] = -P[2] \\ P[4] = -0.08544127388199999 & Q[4] = P[1] \\ P[5] = 0.03522629188200000 & Q[5] = -P[0] \end{cases}$$

On pourra utiliser le bout de code proposé figure 2 pour la fonction inverse. Programmer les

```
nh = size / 2; fsize = 6;
for (i=0,r=nh-(fsize/2-1); i<n; r=r+1,i=i+2) {
   b[i]=0.0;
   b[i+1]=0.0;
   for(k=fsize-2,j=0; k>=0; k=k-2,j=j+1) {
      b[i] += P[k]*Pix(a,(r+j)%nh,JJ) + Q[k]*Pix(a,nh+(r+j)%nh,JJ);
      b[i+1] += P[k+1]*Pix(a,(r+j)%nh,JJ) + Q[k+1]*Pix(a,nh+(r+j)%nh,JJ);
   }
}
```

FIG. 2 – Recomposition dans la base de Daubechie

fonctions de décomposition dans la base de Daubechie pour des images de taille quelconque. On rappelle que la décomposition s'arrête si le nombre de moyennes disponibles est inférieur à 6.

Réalisez le même programme de seuillage que précédemment.

```
daubechie <input.gif> <output.gif> <seuil>
```

Que pouvez-vous conclure ?

6 Objectifs et/ou Extensions Possibles

Le but de ce projet est de vous familiariser avec la programmation en C. Il vous faudra notamment comprendre les prototypes des fonctions mises à votre disposition, ainsi que certains nouveaux types. Puis, en raison de l'utilisation de fonctions externes, vous aurez à gérer et compiler un projet à plusieurs modules (voir le polycopié pages 58–62).

Il est également conseillé de faire un `Makefile` afin de faciliter les compilations répétées (voir le polycopié pages 74–77).

6.1 Objectif minimal

Vous avez finalement à produire un exécutable

```
projet -h/-d <input.gif> <output.gif> <seuil>
```

- qui charge l'image contenue dans le fichier `input.gif` ;
- la décompose dans la base de Haar (si `-h`) ou de Daubechie (si `-d`) ;
- met à zéro les détails dont la valeur est inférieure au seuil `seuil` ;
- recompose l'image et la sauvegarde au format GIF dans le fichier `output.gif`.

Lors de l'exécution, afficher le taux de compression effectué (pourcentage des pixels mis à zéro).

6.2 Extensions possibles

Pour les plus entreprenants, il existe une structure d'image couleur `CIMAGE` possédant trois champs de données. Il serait intéressant d'appliquer cette technique sur chaque canal (le rouge, le vert et le bleu).

Il est également possible d'utiliser la librairie de thread `libpthread.so` pour réaliser une version parallèle de la décomposition d'images en couleur selon le schéma :

```
pthread_create(appel sur les données rouges)
pthread_create(appel sur les données vertes)
pthread_create(appel sur les données bleues)
```

```
pthread_join(données rouges)
pthread_join(données vertes)
pthread_join(données bleues)
```

Pour avoir des informations sur les différents types et fonctions :

- voir `image_defs.h`, `image_func.h` et `image_func.c`, pour les images en couleur ;
- ainsi que `/usr/include/pthread.h` et le `man` des deux fonctions ci-dessus `pthread_create` et `pthread_join`, pour la gestion de processus parallèles.