Compression par transformée en cosinus discrète d'images en niveaux de gris

Devoir 1 OIM
Compression d'images

2014

NOTES IMPORTANTES :

Modalités de remise du projet : Vous devez déposer sous moodle une archive au format tar.gz (et uniquement à ce format là) nommée : prénom.nom-OIM1.tar.gz (remplacez prénom et nom par VOTRE prénom et VOTRE nom). Merci de ne pas mettre d'espaces ou d'accents dans le nom de votre archive.
Cette archive devra contenir UNIQUEMENT :
— les fichiers sources, respectant la norme C99, de votre projet,
— un fichier *Makefile* permettant de compiler les différents programmes réalisés,
— éventuellement un document au format pdf ou texte contenant les informations que vous souhaitez communiquer sur votre travail (Spécifications, descriptions des algorithmes, références bibliographiques, ...)

Modalités de correction : **Attention**, un outil de détection de la copie sera utilisé sur les archives déposées. Toute copie avérée entrainera la note de 0 au devoir et une convocation par le responsable de l'unité d'enseignement. Si un projet remis ne peut être compilé sur le système d'exploitation utilisé en TP (Linux), la note de 0 sera attribuée au projet. La qualité du code (lisibilité, réutilisabilité, modularité, ...) ainsi que ses performances (complexité en espace et en temps) constitueront une part importante de la note finale.

L'objectif de ce devoir est de programmer une méthode de compression d'images basée sur la transformée en cosinus discrète (DCT). Comme vu en cours, la norme JPEG est construite sur ce principe. Il n'est pas demandé d'implanter la norme JPEG mais une simplification de celle-ci. Plusieurs fonctions utilitaires vous sont fournies dans l'archive à télécharger sous Moodle afin de réaliser ce devoir (fonction de calcul de la DCT et de son inverse, ...). Dans le code que vous remettrez, ces fonctions doivent pouvoir être remplacées par toute autre implantation respectant les interfaces données.

1 Description de l'archive logicielle fournie

L'archive qui vous est fournie pour servir de base à votre travail est constituée des fichiers suivants :

1. Fichers sources
   — image.h et image.c : Definition du type *image* et des fonctions de chargement et de sauvegarde des différents formats d'images manipulés dans le devoir.
   — dct-idct.h et dct-idct.c : Fonctions de calcul de la transformée en cosinus discrète et de son inverse.
   — main.c : Programme principal, se limitant initialement à l'analyse de la ligne de commande.
   — Makefile : fichier de compilation et de déclenchement des tests.
2. Fichiers de test
   — 1234.pgm : simple image de test.
   — input1.pgm : image de test contenant des valeurs particulières de pixel.
   — lena.pgm : image de test standard en traitement d'images
   — mystery.xxx : une image compressée, à décompresser et insérer dans votre rapport.

3. Script de test
   — test_script.sh : script shell de test de votre programme.

Pour chaque étape du sujet, vous devrez gérer dans votre programme principal les arguments de la ligne de commande. Ceci est nécessaire pour que les tests automatiques fonctionnent et permettent de valider votre programme. Regardez dans le fichier main.c pour comprendre le détail du fonctionnement qui vous est proposé. En particulier les arguments de la ligne de commande doivent respecter la syntaxe suivante :

usage : ./compressor *mode in out*

mode : 0 : decompression, 1 : compression, 2 : save dct (pgm format),
3 : save quantize (pgm format), 4 : save vectorize (xxx format), 5 output compression loss

in : input filename, extension pgm if compression, save dct or save quantize, extension .xxx if decompression

out : output filename, extension xxx if compression, pgm if decompression, save dct or save quantize

2 Description de la méthode de compression et du travail à fournir

2.1 Compression

Soit une image en niveau de gris, codée sur 8 bits par pixels, de taille $8M \times 8N$. L'image est parcourue par blocs de pixels de taille 8×8, sans recouvrement.

Chacun de ces blocs de pixels sera traité de manière identique et indépendante des autres blocs, selon les étapes suivantes, que vous devrez programmer dans le fichier main.c :

1. calcul de la DCT (3 pts)
   En utilisant la fonction DCT fournie, calculer la DCT de chaque bloc. Nous notons par $B(f, g)$ la transformée en cosinus discrète du bloc $b(x, y)$, avec f, g, x et y allant de 0 à 7. Cette transformée est rangée dans le champs *image->data* d'une image transformée pour laquelle vous aurez alloué la mémoire suffisante. Stocker sur le disque (en utilisant la fonction C fournie
   void writePgm(**const char** *filename, **const** image *img);) l'image obtenue en juxtaposant les DCT de tous les blocs en respectant l'ordre de parcours des blocs. Pour permettre une bonne visualisation de la DCT, dans cette étape les valeurs stockées sont normalisées (divisée par 8.f) Afin de vérifier votre code, effectuer le test de votre logiciel par la commande *make tests*. Après cette étape, votre programme doit passer le test 1 :
   ```
   $ make tests
     dct [OK]
   ```
2. quantification (3 pts)
   Chaque éléments (pixel) de $B(f, g)$ est quantifié en utilisant un quantum différent, donné par la matrice de quantifiction Q suivante pour produire le bloc $B_q(x, y)$:

$$Q = \begin{Bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{Bmatrix}$$

   L'opération de quantification s'écrit sous la forme :

$$B_q(f, g) = \left[\frac{B(f, g)}{Q(f, g)} \right]$$

   pour f et g allant de 0 à 7. Nous indiquons par $[x]$ l'entier le plus proche de x.

En utilisant la matrice de quantification ci-dessus, calculer, pour chaque bloc $B(f,g)$, le résultat $B_q(f,g)$ de la quantification par Q.

Stocker sur le disque, en utilisant writePgm, l'image obtenue en juxtaposant les DCT **quantifiées** de tous les blocs. Afin de vérifier votre code, effectuer le test de votre logiciel par la commande *make tests*. Après cette étape, votre programme doit passer le test 2 :

```
$ make tests
  dct [OK]
  quantify [OK]
```

3. parcours en Z de la DCT quantifiée de chaque bloc (3 pts)

   Pour chaque DCT quantifiée $B_q(f,g)$, programmer la fonction qui la transforme en un vecteur de taille 64 en utilisant le parcours en Z vu en cours et en TD (voir illustration ci-dessous). Nous notons par $B_q^v(x,y)$ la version vectorisée de $B_q(f,g)$.

   Stocker sur le disque l'image obtenue en juxtaposant les versions vectorisées $B_q^v(x,y)$ de tous les blocs, en utilisant la fonction writePgm.

   Cette fonction écrit dans le fichier la taille de l'image, puis un bloc de données de img−>size valeurs lue à partir de img−>data. Afin de vérifier votre code, effectuer le test de votre logiciel par la commande *make tests*. Après cette étape, votre programme doit passer le test 3 :

```
$ make tests
  dct [OK]
  quantify [OK]
  vectorize [OK]
```

4. détection des valeurs utiles pour chaque bloc (2 pts)

   Pour chaque vecteur $B_q^v(x,y)$, détecter la position p à partir de laquelle toutes les valeurs suivantes sont égales à 0. Stocker tout ou partie du vecteur $B_q^v(x,y)$ courant dans le champs *image->data* en écrivant dans un premier temps le nombre de valeur significatives du vecteur, suivi par ces valeurs.

   Lorsque toute l'image aura été transformée, modifier le champs *image->size* qui devra contenir le nombre de valeurs écrites dans *image->data*. Sauvegardez l'image résultante en utilisant cette fois ci la fonction

```
void writeCompressed(const char *filename, const image *img)
```

   Vérifier votre programme en effectuant le test de votre logiciel par la commande *make test*. Après cette étape, votre programme doit passer le test 4 :

```
$ make tests
  dct [OK]
  quantify [OK]
  vectorize [OK]
  compression [OK]
```

2.2 Décompression (2 pts)

Reconstruire l'image, à partir des données stockées en étape 6, en parcourant les étapes de 1 à 6 dans l'ordre inverse. Vérifier votre programme en effectuant le test de votre logiciel par la commande *make test*. Après cette étape, votre programme doit passer le test 5 :

```
$ make tests
  dct [OK]
  quantify [OK]
  vectorize [OK]
  compression [OK]
  decompression [OK]
```

2.3 Erreur de compression (1 pt)

Calculer l'erreur de compression résultante de la quantification. Pour cela vous devez comparer l'image originale avec l'image compressée puis décompressée en faisant la somme des différences au carré des

niveaux de gris obtenus, pixel à pixel, divisée par le nombre de pixel. Votre programme doit afficher l'erreur de compression pour l'image input.pgm en utilisant la commande

```
./compressor 5 input.pgm dummy.xxx
```

dummy.xxx ne sera pas pris en compte dans votre programme.

2.4 Optimisation (3 pts)

La fonction fournie idct est sous-optimale. Optimisez la pour éviter les calculs redondants. Pour cette étape, vous détaillerez vos modifications dans votre rapport.

2.5 Parallélisation (3pts)

La plupart des boucle de votre programme peuvent être parallélisées. En utilisant une bibliothèque comme openMP (http://openmp.org/), ou une autre de votre choix, paralléliser votre programme pour obtenir des meilleures performance. Mesurer le gain de performances avec la commande time.

Pour optimiser au mieux votre programme, vous pouvez aussi utiliser la programmation *sse* ou *avx* des processeurs modernes.

Pour cette étape, vous détaillerez vos optimisation dans votre rapport et indiquerez les gains de performances obtenus sur des images de taille différentes dans votre rapport.