Travaux pratiques de traitement d'images numériques
Première séance
Institut Galilée
2010-2011

G. Dauphin et A. Beghdadi

Les travaux pratiques de traitement d'image sont répartis en trois séances de 8 heures chacune.

Quelques indications sur MatLab

Les images en niveaux de gris sont mémorisées sous la forme d'une matrice formée chacune d'entiers de 0 à 255 correspondant à un octet (format MatLab : `uint8`). Les images colorées sont parfois mémorisées sous la forme d'un tableau de chiffres chaque chiffre correspondant à un index dans une table de couleur, c'est souvent le format choisi pour les images en couleurs. Elles peuvent aussi être mémorisées sous la forme d'une matrice-3D formées chacune d'entiers de 0 à 255 (par exemple `image(1,1,1)` désigne l'intensité du rouge [3ème 1] présent dans le pixel qui est sur la colonne 1 [1er 1] et sur la ligne 1 [2ème 1]).

Le traitement d'image amène à faire des opérations sur les valeurs de chaque pixel. Il faut donc que ces valeurs soient représentées avec un type adapté appelé `double` en MatLab. Il est alors nécessaire que les valeurs doivent être comprises entre 0 et 1 ; cela afin de pouvoir utliser les commandes MatLab de visualisation des images.

MatLab dispose d'une aide en ligne sur la fenêtre de commandes : `help` permet d'afficher les différentes sections, `help` *nom d'une section* permet d'avoir la liste des commandes dans cette section, en particulier `help images` donne la liste des commandes relatives à l'image. `help` *nom d'une commande* donne une explication sur la commande et souvent un exemple qui peut vraiment être essayé. Les images peuvent être obtenues soit à partir du répertoire de MatLab `toolbox\images\imdemos`.

Les fonctions MatLab pour lire et enregistrer les images sont `imread` et `imwrite`. Les fonctions disponibles pour afficher les images sont `image`, `imagesc` et `imshow`. Attention à `imshow` qui considère que si l'image considérée est composée d'entiers alors l'image est supposée être en 0 et 256, tandis que si l'image est formée de `double`, alors l'image est supposée être entre 0 et 1. Dans le cadre de ces TP, on se placera dans ce deuxième cas. Plusieurs images peuvent être affichées sur une seule figure grâce aux fonctions `figure` et `subplot`. Une image en niveau de gris et plus généralement n'importe quelle matrice peuvent être considérées comme une surface : à chaque coefficient de la matrice, on associe un point dont l'abscisse et l'ordonnée sont déterminés par la position de ce coefficient et dont la cote est déterminée par la valeur du coefficient. Les fonctions MatLab disponibles sont `surf` et `mesh`, la première fonction colore les éléments de surface tandis que la deuxième colore seulement les bords des éléments de surface.

MatLab permet d'utiliser les commandes internes et les noms de programmes écrits comme des variables. Cependant il faut absolument éviter de faire cela.

Préambule

On appelle image naturelle, une image obtenue à partir du monde réel, par exemple au moyen d'un appareil photographique. Ici l'image naturelle est notée `image_n`. On appelle image synthétique, une image construite à partir d'un ordinateur, ici l'image synthétique est notée `image_s`. Dans le cadre de ces travaux pratiques, il est conseillé d'utiliser à la fois une image naturelle et une image synthétique. Sauf indication contraire, les commandes proposées sont valables pour des images en niveaux de gris (avec 256 niveaux)et de taille 256x256. Cependant on peut faire les traitements sur des images couleurs en appliquant ces traitements séparément sur les composantes rouges, vertes et bleues.

On peut obtenir une image en niveau de gris à partir d'une image en couleur en utilisant la commande `rgb2gray`. On peut obtenir une image de taille 256×256 en ne considérant qu'une partie de l'image en utilisant `imcrop` ou en réduisant la taille de l'image avec `imresize`.

Pour avoir la liste des images déjà disponibles sous MatLab

```
help imdemos,
```

Cas d'une image avec des vraies couleurs :

```
image_n=imread('autumn.tif');
```

On peut voir que cette `image_n` n'est pas une matrice mais un ensemble de trois matrices en faisant :

```
size(image_n),
```

On peut visualiser en niveaux de gris la composante rouge de l'image couleur

```
figure(1); imshow(image_n(:,:,1))
```

On peut visualiser en niveaux de gris la composante verte de l'image couleur

```
figure(1); imshow(image_n(:,:,2))
```

On peut visualiser l'ensemble des trois matrices sous la forme d'une image couleur

```
figure(1); imshow(image_n);
```

Cas d'une image avec un petit nombre de couleurs :

```
[image_ind,map]=imread('forest.tif');
```

Cette image couleur est manifestement stockée sous la forme d'une table d'indice qui renvoie à une table de couleurs parce que `image_ind` est une matrice.

```
size(image_ind),
```

Pour représenter cette image couleur sous la forme d'un triplet de matrice correspondant au rouge, vert et bleu.

```
image_n=ind2rgb(image_ind,map);
```

On peut vérifier qu'on s'est ramené au cas précédent avec

```
size(image_n)
```

On peut alors afficher l'image couleur ainsi obtenue.

```
figure(1); imshow(image_n);
```

Pour faire des calculs il est nécessaire de passer en format double puis aussi de convertir l'ensemble des valeurs en des valeurs contenus dans l'intervalle $[0, 1]$.

En effet on pourrait afficher une image couleur avec seulement ces instructions

```
im1=imread('autumn.tif');
figure(1); imshow(im1);
```

Cependant pour faire des calculs on est amené à convertir au format double et dans ce cas on ne voit plus rien.

```
figure(1); imshow(double(im1);
```

Il suffit alors de diviser par 256 pour obtenir le résultat souhaité.

```
figure(1); imshow(double(im1)/256);
```

Pour savoir si une image provient d'une palette finie de couleur, il existe une façon de faire avec la fonction MatLab `unique` en comptabilisant toutes les couleurs différentes au sein de l'image couleur (ce nombre de couleurs différentes est nécessairement inférieur ou égal au nombre de pixels de l'image et a priori différent de la somme du nombre de rouges différents du nombre de verts différents et du nombre de bleus différents). Comme les couleurs sont représentés par des triplets, une astuce pour les représenter sur un seul nombre est de les laisser sous la forme d'un triplet d'entier entre 0 et 255 et de convertir ce triplet en un entier entre 0 et 255*255*255 par le biais de ces commandes.

```
im1=imread('autumn.tif');
image_c=im1(:,:,1)+256*im1(:,:,2)+256*256*im1(:,:,3);
```

Le nombre de couleurs différentes est alors donné par

```
length(unique(image_c(:))),
```

Ce nombre est bien sûr différent de

```
length(unique(im1(:,:,1)))+length(unique(im1(:,:,2)))+length(unique(im1(:,:,3))),
```

Construction d'une image synthétique :

Soit un carré blanc noyé dans un fond uniforme gris, soit un disque noyé dans un fond uniforme. La première image peut se faire avec `ones(256,256)*0.5` pour le fond gris et avec `ones(15,15)` pour le carré. La deuxième image peut se faire avec `meshgrid` et `find` en utilisant le fait qu'un disque a pour équation : $(x-100)^2+(y-100)^2 \leq 400$. L'image synthétique choisie est notée `image_s`.

Pour le carré :

```
im1=ones(256)*0.5;
im1(128-7:128+7,128-7:128+7)=1;
figure(1); imshow(im1);
```

On peut vérifier que le carré est de bonne taille avec

```
sqrt(sum(sum(im1>=0.8))),
```

Pour le disque :

```
[x,y]=meshgrid(0:255,255:-1:0);
im1=((x-100).^2+(y-100).^2<=400);
figure(1); imshow(im1);
```

Quantification linéaire :

La fonction $N\_$ détermine pour chaque valeur de t, le nombre d'éléments de A inférieur à cette valeur de t.

La commande `inline` permet de définir cette commande. Le symbole `.''` signifie en fait transposée de la matrice auquel il s'applique. L'apostrophe est doublée parce qu'elle est incluse dans une chaîne de caractère qui va être interprêtée lors de l'exécution de la fonction.

```
N_=inline(...
  'reshape(sum(ones(length(t(:).''),1)*(A(:).'')<t(:)*ones(1,length(A)),2),size(t))'...
  ,'t','A');
```

Cette commande qui réalise une fonction en escalier permet d'illustrer le fonctionnement de cette commande

```
figure(1); plot(0:0.01:10,N_(0:0.01:10,0:10));
```

Grâce à cette commande la quantification linéaire devient

```
im1=double(imread('coins.png'))/256;
figure(1); imshow(im1);
Scale=0:1/8:1;
ImQuant=Scale(N_(im1,Scale));
```

Le nombre de niveaux de quantifications est déterminé par le vecteur `Scale`.
Quantification non-linéaire :
La quantification non-linéaire est réalisée de façon presque similaire.

```
im1=double(imread('coins.png'))/256;
figure(1); imshow(im1);
Scale=(0:1/8:1).^2;
ImQuant=Scale(N_(im1,Scale));
```

On peut vérifier que l'image a été quantifiée en utilisant

```
length(unique(ImQuant)),
```

1 Analyse du signal image

On considère une image synthétique de taille 256×256 notée `image_s` puis une image naturelle de même taille, notée `image_n`.

1. On souhaite réduire par un facteur de quatre la taille de l'image choisie.

   (a) Une première solution consiste à prendre un pixel sur quatre (i.e. un pixel par carré de quatre pixels). Utiliser ce procédé pour réduire la taille de l'image. Réitérer ce processus de réduction (3 fois) jusqu'à obtenir une image de taille 32×32 (on parle de décomposition multirésolution). Afficher avec des tailles différentes les images ainsi obtenues.

   ```
   deci=inline('im(1:2:end,1:2:end)','im');
   im2=deci(im1);
   ```

   (b) A chaque étape, discuter du critère de Nyquist et adapter le filtre de restitution vu en préparation pour réconstituer l'ensemble de l'image. Comparer les versions restituées avec la version originale de l'image.

   ```
   inte=inline('filter2(ones(2),upsample(upsample(im1.'',2).'',2))','im1');
   ```

   (c) Une deuxième solution consiste à remplacer un groupe de pixels (2×2) par un seul dont le niveau de gris est égal à la moyenne des quatre pixels du groupe, l'image est alors notée `image_rd2`. Comparer les deux approches.

   Une implémentation consiste à construire une fonction en ligne qui remplace un groupe de 2×2 pixels par un seul dont le niveau de gris est égal à la moyenne des quatre pixels. En fait c'est ici inutile car la fonction `mean2` le fait déjà.

   ```
   A=[1 0;0 1];
   mean2(A),
   ```

   La suite de l'implémentation consiste à utiliser `blkproc` pour appliquer cette fonction à tous les groupes de 2×2 pixels de l'image.

```
im2=blkproc(im1,[2 2],@mean2);
```

Une deuxième implémentation consiste à filtrer l'image par le moyenneur de masque `ones(2)/4` puis à décimer. Dans la deuxième implémentation un grand nombre de calculs en plus ont été effectués mais les pixels contenant les résultats de ces nouveaux calculs ne sont plus dans l'image résultante.

```
im2=deci(filter2(ones(2)/4,im1));
```

2. On souhaite maintenant retrouver une image de même taille que l'originale en partant de la dernière version réduite. Une première solution consiste à reproduire le même pixel plusieurs fois de façon à retrouver la taille initiale. Expérimenter cette solution, notée `image_mr` et discuter son effet. Proposer une solution plus efficace et expérimenter la. Afficher pour chacune des solutions le signal image erreur sur fond gris notée `image_err` et analyser le aux différents étages de la décomposition multirésolution.

2 Quantification

Appliquer une quantification uniforme sur `image_s` puis sur une image naturelle `image_n` en utilisant 30,10 puis 5 niveaux de quantifications. Indiquer les différences perceptibles entre les images quantifiées et les images d'origine.

3 Couleur, luminance, espace de couleur

1. Calculer le signal luminance d'une image couleur, d'abord en extrayant les composantes rouge, puis verte puis bleue, et ensuite en utilisant seulement `rgb2gray`.
   Explication *on pourra utiliser* $L = 0.3 * R + 0.57 * G + 0.11 * B$.

2. Afficher successivement en niveau de gris les signaux de luminance et chrominance. Puis reformer l'image couleur et afficher l'image obtenue qui doit être identique à celle du départ.
   Explication *Il existe différents espaces de couleurs dans lesquels cette transformation peut se faire, mais dans tous les espaces, il y a forcément deux signaux de chrominance qui comportent une information sur la teinte et la saturation (proportion de blanc) et un signal de luminance (sombre ou claire). Les fonctions* `rgb2ycbcr` *et* `ycbcr2rgb` *peuvent être utilisées.*

4 Impact sur l'image d'une réduction de la résolution de la chrominance ou de la luminance

Sous-échantillonner le signal de chrominance et reformer l'image. Puis sous-échantillonner le signal de luminance et reformer cette autre image. Observer et commenter les images obtenues.

5 Impact sur l'image d'une sous-quantification de la chrominance ou de la luminance

On considère des images en niveaux de gris, ces valeurs de gris étant déjà quantifiées sur 256 niveaux. Simuler des quantifications sur un certain nombre de niveaux. Observer et commenter les images obtenues.

Explication *La quantification sur* N *niveaux se fait en appliquant la fonction* $g \mapsto \frac{E(Ng)}{N}$ *à la valeur de chaque pixel (celui-ci étant a priori entre* 0 *et* 1*).*

Une méthode pour implémenter cette quantification consiste à définir une fonction en ligne `inline` qui réalise cette opération et à l'appliquer à l'image.

On considère maintenant une image couleur. Séparer le signal chrominance du signal de luminance. Quantifier le signal de chrominance et reformer l'image. Puis quantifier le signal de luminance et reformer cette autre image. Observer et commenter les images obtenues.

Présentation de quelques mesures

On considère ici deux images : une image origine $[g^o_{m,n}]$ et une image bruitée $[g^b_{m,n}]$. L'appréciation visuelle de la ressemblance entre ces deux images est ce qu'on appelle un critère subjectif. Les mesures de qualité suivantes forment des critères dits objectifs. Elles sont plus rapides à utiliser, mais souvent moins fiables parce qu'elles ne mesurent que des distortions entre images sans pouvoir vraiment prédire dans quelle mesure cette distortion gène effectivement l'obervateur, ce qui pourtant est le plus important.

- L'erreur quadratique moyenne est aussi appelée MSE (mean square error) :

$$EQM = \frac{1}{M \times N} \sum_m \sum_n \left(g^o_{m,n} - g^b_{m,n} \right)^2$$

```
EQM=1/M/N*sum(sum((image_o-image_b).^2));
```

- Le PSNR, encore appelé `peak signal noise ratio`, signifie

$$dB = 10 \log_{10} \left(\frac{(\max(g) - \min(g))^2}{EQM} \right)$$

Ces mesures n'évoluent pas de façon proportionnelle, ainsi le calcul du quotient de l'une par rapport à l'autre peut mettre en évidence certains aspects.

6 Graphes

Construire deux graphiques représentant pour une image donnée en niveaux de gris, l'évolution du PSNR en fonction du niveau de quantification d'une part, et d'autre part l'évolution du PSNR en fonction de la résolution de l'image.