Approches multi-résolutions : travaux pratiques

Objectif : comprendre et utiliser les représentations multi-résolutions (pyramides, ondelettes), utilisées pour l'analyse, le traitement et le codage de signaux et d'images.

Télécharger le code fourni à l'adresse : `https://uuu.enseirb.fr/~megret/Enseignement/Multicadences/`
Consignes générales : avant toute manipulation sous Matlab, bien suivre les étapes suivantes:
- dézipper l'ensemble des fichiers fournis en conservant l'arborescence des fichiers
*- créer un ou plusieurs script ou fonctions *.m afin d'y taper les commandes à exécuter*
(Rappel : exécution → F5, exécution partielle → F9).
- Les images à utiliser se trouvent dans le répertoire `images/`

Pyramides d'images

Les codes à utiliser pour les pyramides d'images se trouvent dans `matlab_pyramid/`.

Préliminaire : sur- et sous-échantillonnage des images

1. Tester le sous-échantillonnage et sur-échantillonnage brut 2D sur une image

```
I = double(imread('photographe256.png'));
Idown = downsample2(I);
Iup   = upsample2(I);

figure(1); imagesc(I,[0 255]);     colormap('gray');
figure(2); imagesc(Idown,[0 255]); colormap('gray');
figure(3); imagesc(Iup,[0 255]);   colormap('gray');
```

2. Programmer la réduction dyadique 2D (`reduce2.m`) avec le filtre anti-repliement 3x3 séparable associé au masque [1/4 1/2 1/4], appliqué séquentiellement en ligne et en colonne (`imfilter`).

```
function J = reduce2(I)
Ih  = imfilter(I,  [1 2 1]/4, 'symmetric');
Ihh = imfilter(Ih, [1 2 1]'/4, 'symmetric');
J   = downsample2(Ihh);
```

Comparer l'effet de `downsample2` et `reduce2` sur une image texturée:

```
I = double(imread('barbara512.png'));
Idown = downsample2(I);
Ireduce = reduce2(I);

% Affichage comme précédemment
```

3. Programmer l'expansion (`expand2.m`) avec un filtre d'interpolation identique au précédent.

```
function Jhh = expand2(I);
J   = upsample2(I);
Jh  = imfilter(J,  [1 2 1]/4, 'symmetric');
Jhh = imfilter(Jh, [1 2 1]'/4, 'symmetric');
Jhh = Jhh * ?; % Mettre ici le facteur correct
```

Quel facteur multiplicatif faut-il appliquer pour que la dynamique de l'image reste inchangée ? Pourquoi ?

Comparer l'effet de `upsample2` et `expand2` sur une image (par exemple *photographe256*).

Construction des pyramides

Les pyramides d'image sont une application directe du rééchantillonnage des images [1].

4. Programmer le calcul de la pyramide gaussienne `G` associée à une image d'entrée `I`, pour un nombre de niveaux *n* donné.

```
function G = pyramid(I,n)

G=cell(n,1);
G{1}=I;
for i=2:n
    G{i}=reduce2(G{i-1});
end
```

Chaque image est stockée dans une structure `G` de type "cell array", où l'on accède au niveau i à travers `G{i}`. Une telle structure permet de stocker des images de tailles différentes.

Pyramide gaussienne à 4 niveaux

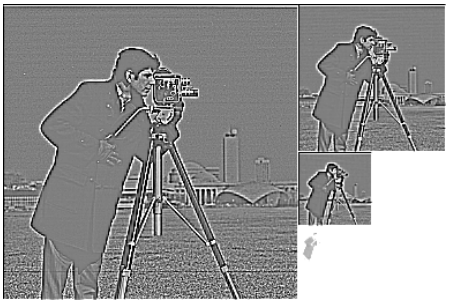
Pour l'affichage, on utilisera la fonction `pyrashow` fournie :

```
G = pyramid(I,4);
pyrashow(G);
```

5. Programmer le calcul de la pyramide laplacienne selon le prototype :

```
function [L,G,E]=laplacianpyramid(I,n)

G = pyramid(I,n)

E=cell(n-1,1);
L=cell(n,1);

for i=1:n-1
    E{i}=expand2(G{i+1});
    L{i}=G{i}-E{i};
end
L{n}=G{n};
```

Chaque niveau *i* de la pyramide laplacienne L peut être calculé à partir des niveaux *i* et *i+1* de la pyramide gaussienne (sauf le dernier niveau, qui est une simple copie). La pyramide `E` est la pyramide d'expansion.



Pyramide laplacienne à 4 niveaux

Visualiser en prêtant une attention particulière à la dynamique pour l'affichage : **les valeurs dans la pyramide laplacienne sont signées**, généralement centrées autour de 0 :

```
[L,G,E] = laplacianpyramid(I,4);
figure; pyrashow(L,[-20 20]);
figure; pyrashow(E);
```

6. Programmer la fonction `reconstruct.m`, qui calcule la reconstruction de l'image initiale (et éventuellement les pyramides G et E associées) à partir de la pyramide laplacienne L :

```
function [I,G,E]=reconstruct(L)

n=length(L);

G=cell(n,1);
E=cell(n-1,1);

G{n}=L{n};
for i=(n-1):-1:1
    E{i}=expand2(G{i+1});
    G{i}=E{i}+L{i};
end

I=G{1};
```

Vérifier que l'image reconstruite est bien identique à l'image d'origine. Justifier à l'aide de mesures numériques.

```
J = reconstruct(L);
```

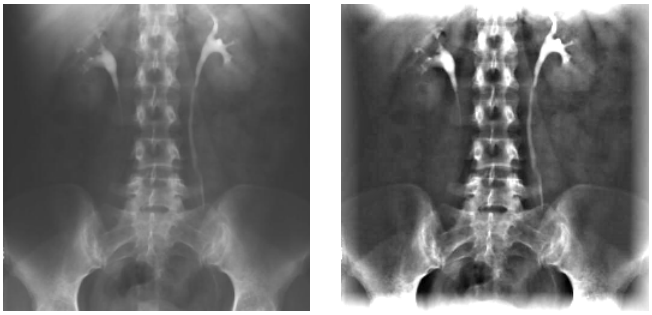
Application au réhaussement d'images

On désire rehausser les contours d'une image selon la technique [2].

7. Appliquer un facteur multiplicatif α à tous les étages de la pyramide laplacienne, sauf le dernier. Ceci a pour effet de rehausser les contenus associés aux fréquences moyennes et hautes, par rapport aux basses fréquences, associées au dernier niveau de la pyramide.

```
function [J L2 L]=enhance(I,n,alpha)

L = laplacianpyra(I,n);

L2{n}=L{n};
for i=1:(n-1)
    L2{i}=alpha*L{i};
end

J = reconstruct(L2);
J = min(255,max(0,J)); % Limite à 0..255
```

Appliquer ce réhaussement à une image médicale telle que `colonne512.png`, avec des valeurs de α dans l'intervalle 2..10. Comparer le résultat avec un réhaussement par multiplication directe de l'image par un coefficient.



Réhaussement d'images multi-résolutions [2]

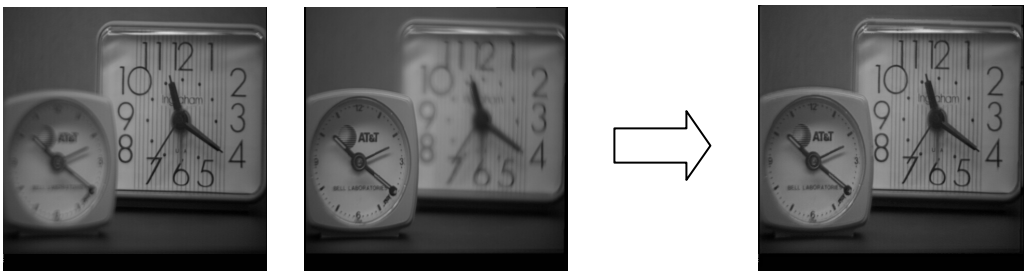
Application à la fusion d'images

8. Programmer la fonction qui produit une image fusionnée selon la technique « Max laplacian » [4]. Cette technique décompose les deux images en pyramides laplaciennes `L1` et `L2` et applique la fusion entre images à chaque niveau des pyramides laplaciennes, en sélectionnant pour chaque pixel le coefficient le plus large en valeur absolue (correspondant au plus grand contraste). L'image fusionnée correspond à la reconstruction à partir de la pyramide laplacienne fusionnée.

```
function [J,L] = laplacianfusion(I1, I2, n)

L1 = laplacianpyramid(I1,n);
L2 = laplacianpyramid(I2,n);

L = cell(1,n);

L{n} = (L1{n}+L2{n})/2;
for i=1:(n-1)
    M = (abs(L1{i}) > abs(L2{i}));
    L{i}=M.*L1{i}+(1-M).*L2{i};
end

J = reconstruct(L);
J = min(255,max(0,J));
```

Appliquer cette technique à plusieurs paires d'images
- `clock1.gif` et `clock2.gif` pour le défloutage par fusion d'images à mises au point multiples
- `s7708_01.jpg` et `s7708_05.jpg` pour la création d'image HDR à partir d'images à expositions multiples

Dans chaque cas, afficher la paire d'images originales, l'image fusionnée par max-laplacian, et une image fusionnée par simple moyenne des valeurs des pixels.
On pourra également afficher les pyramides laplaciennes des images `I1`, `I2` et `J` pour observer l'effet de la sélection des coefficients.



Défloutage par fusion d'images à focus multiples

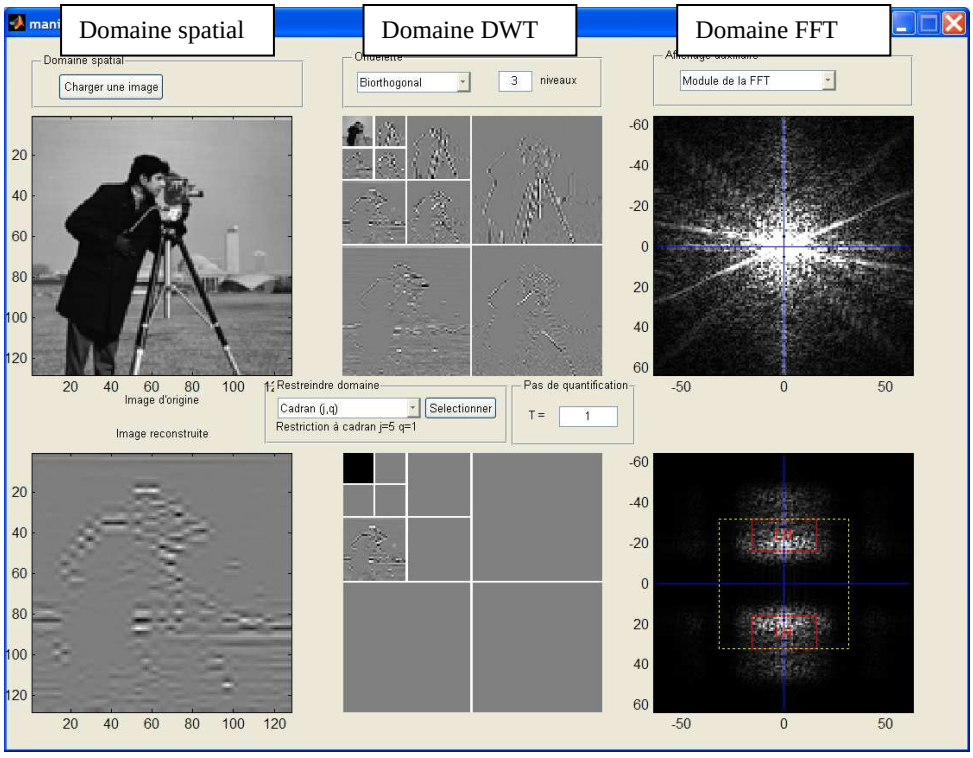
Références

[1] P.J. Burt, E.H. Adelson, "The Laplacian Pyramid as a Compact Image Code," IEEE Trans. on Communications, pp. 532--540, April 1983.
[2] page 344 de l'article de Dippel et al. : « Multiscale contrast enhancement for radiographies : laplacian pyramids vs. fast wavelet transforms », IEEE Trans. Medical Imaging, vol 21, n°4, 2002.
[3] J. Ogden, E. Adelson, J. Bergen, and P. Burt, "Pyramid Based Computer Graphics". RCA Engineer, volume 30, pages 4-15, 1985.
[4] Fusion max-laplacian d'images : http://www.cs.technion.ac.il/~ronrubin/Projects/fusion/

Ondelettes : manipulation de la DWT

Les codes à utiliser pour la DWT se trouvent dans `matlab_wavelet/` et `demo_wavelet/`.

La fenêtre interactive **`manipulate_dwt.m`**, permet de manipuler la décomposition en ondelettes discretes (DWT), ainsi que visualiser ses caractéristiques au niveau fréquentiel, entropique, qualitatif...
Attention :
- Pour que l'interface fonctionne, bien définir le répertoire courant comme le répertoire `demo_wavelet`
- Afin de limiter les temps de calcul, l'interface redimensionne automatiquement les images à 128×128.



Sur la ligne du haut sont affichées les informations relatives à l'image d'entrée I et sa DWT. Il est possible de choisir le type d'ondelette à utiliser (biorthogonale CDF9/7, Haar, ou sinus cardinal), ainsi que le nombre de niveaux de décomposition.
La ligne du bas correspond à l'image reconstruite R par transformée inverse de Wr, obtenue après deux traitements : application d'un pas de quantification T à la transformée W, mise à zéro des coefficients d'ondelette de certains cadrans.
La colonne de gauche correspond à l'image I ou à sa reconstruction R, la colonne du milieu à sa transformée W ou Wr. La colonne de droite contient un affichage modifiable : transformée de Fourier de l'image, histogramme des valeurs de l'image I (avec l'entropie associée), histogramme des valeurs de la transformée W (avec entropie) ou bien image d'erreur entre I et R.

Questions d'autotest :
1. Pour une décomposition à 1 niveau, à quoi correspond chaque cadran ?
2. Pour une décomposition à 2 niveaux (ou plus), à quoi correspond chaque cadran ?
3. En observant les histogrammes des valeurs et des coefficients d'ondelette, mettre en évidence l'utilité de la DWT pour le codage d'images.
4. En observant la DWT et l'image d'erreur, mettre en évidence l'effet de la quantification.
5. En mettant à zéro les cadrans en dessous d'une certaine échelle, mettre en évidence le type d'erreur que lorsqu'on commet en fonction du cadran dans lequel cette erreur intervient.
6. Quelles sont les différences entre l'ondelette biorthogonale et l'ondelette de Haar ?
7. Pourquoi l'ondelette sinus cardinal (utilisant des filtres parfaits dans le domaine fréquentiel) n'est-elle pas utilisée en pratique ?

Compression d'images par ondelettes

L'objectif est de mettre en œuvre un codeur par ondelette et d'étudier ses caractéristiques et performances. Ce codeur illustre une version simplifiée du codage JPEG2000 [1, 2, 3].

Décomposition, visualisation et reconstruction

1. Réaliser la décomposition `W` de l'image `I` issue du fichier `photographe256.png` sur 3 niveaux et calculer sa reconstruction `Ir` à partir de cette décomposition :

```
I=double(imread('images/photographe256.png'));
n=log2(size(I,1));
Jmin=n-3;  Jmax=n-1;
options=struct('wavelet_type','biorthogonal'); % ondelette CDF 9/7
W  = perform_wavelet_transform(I,Jmin,+1,options);
```

   Le résultat est représenté sous la forme d'une image `W` de la même taille que l'image d'origine `I`.

```
imagesc(W);
```

   Les différents niveaux de la décomposition ayant des dynamiques différentes, la fonction suivante permet d'afficher le résultat avec adaptation automatique de la dynamique pour chaque niveau :

```
show_wavelet(W,Jmin,'real');
```

   Il est possible d'afficher le cadran de son choix (échelle `j` et cadran n°`q`) à l'aide de

```
show_wavelet_cadran(W,Jmin,'real',j,q);
```

   Les conventions utilisées pour la numérotation de j et de q sont les suivantes :
   - q prend la valeur 0,1,2 ou 3. Le cadran q=0 correspond aux basses fréquences (LL), q=1 aux BF horizontales et HF verticales (LH), q=2 aux HF horizontales et BF verticales (HL), q=3 aux hautes-fréquences (HH).
   - j indique le niveau dans la décomposition hiérarchique. Il prend ses valeurs dans [| J_{min}…J_{max} |]. Le niveau j correspond à la décomposition pour laquelle chaque quadrant est de taille $2^j \times 2^j$. Ainsi le niveau j correspond à la décomposition en 4 sous-cadrans du cadran q=0 du niveau $j+1$.

2. Vérifier que la reconstruction est parfaite, aux approximations numériques près.

3. Calculer l'image `I` reconstruite à partir d'une décomposition `W` nulle partout sauf pour un seul coefficient non nul (On pourra utiliser l'interface `view_dwt` en sélectionnant « Coefficient Simple »).

```
W=zeros(128,128); Jmin=3;
options=struct('wavelet_type','biorthogonal');
W(15,45) = 255;
Ir = perform_wavelet_transform(W,Jmin,-1,options);
```

   Les images `Ir` ainsi obtenues correspondent aux formes d'onde associées à chaque coefficient de la DWT, et qui représentent une base orthogonale de l'espace des images. Afficher `Ir` et `W` côte à côte et tester pour plusieurs choix de la position du coefficient. Discuter l'apparence de la forme d'onde présente dans `Ir` en fonction du cadran de `W` contenant le coefficient non nul.

```
subplot 121; show_wavelet(W,Jmin,'real',1); colormap('gray');
subplot 122; imagesc(Ir);
```

   Comparer les résultats avec ceux obtenus en utilisant une ondelette de Haar

```
options=struct('wavelet_type','haar');
```

Quantification

Afin de pouvoir être représentée numériquement sous une forme compacte, les images de la transformée doivent être quantifiées. Le standard JPEG2000 prévoit l'utilisation potentielle d'un pas de quantification spécifique à chaque cadran. Dans notre cas, nous simplifierons en utilisant le même pas de quantification `T` pour tous les cadrans.

4. La quantification uniforme de pas `T` avec zone morte est définie par l'index entier *k* et la valeur quantifiée réelle $\hat{x}$ associées à la valeur *x*:

$$k = \begin{cases} 0 & \text{si } |x| < T \\ \left\lfloor \frac{|x|}{T} \right\rfloor \text{signe}(x) & \text{sinon} \end{cases} \qquad \hat{x} = \begin{cases} (k-0.5)\times T & \text{si } k < 0 \\ 0 & \text{si k} = 0 \\ (\text{k}+0.5)\times T & \text{si } k > 0 \end{cases}$$

   Cette quantification, et sa fonction inverse sont fournies par la fonction *perform_quantization* qui peut traiter aussi bien des scalaires que des matrices.

```
x = [-10:0.01:10];  T = 2.0;
[xhat, k] = perform_quantization(x, T, 1);
```

   Tracer le graphe exprimant la valeur quantifiée $\hat{x}$ en fonction de *x*. Tracer également le graphe de l'erreur $e(x) = \hat{x} - x$ en fonction de *x*. Illustrer la notion de zone morte.

5. Quantifier les coefficients d'ondelettes avec zone morte en utilisant le même pas `T` pour tous les cadrans :

```
[W_quant, W_int] = perform_quantization(W, T);
```

6. On définit l'erreur RMS comme la racine carrée de l'erreur quadratique moyenne entre l'image initiale I et l'image reconstruite après quantification $\hat{I}$

$$RMS = \sqrt{\frac{1}{N}\sum_{n=1..N}(\hat{I}(n) - I(n))^2}$$

   Il est également courant d'utiliser également le facteur de qualité PSNR (*Peak Signal to Noise Ratio* = Rapport Signal sur Bruit*)*, défini comme le ratio entre la puissance crête du signal (255^2 dans notre cas) et la puissance du bruit (erreur entre les images). Exprimé en dB :

$$PSNR_{dB} = 10\log_{10}\left(\frac{255^2}{P_e}\right) \quad \text{avec} \quad P_e = \frac{1}{N}\sum_{n=1..N}\left(\hat{I}(n) - I(n)\right)^2$$

   Observer la reconstruction `Ir_quant`, l'image d'erreur `Ir_quant-I`, la valeur de l'erreur L_2 et le PSNR pour plusieurs pas de quantification (par exemple 1, 16, 64, 256).

```
Ir_quant = perform_wavelet_transform(W_quant,Jmin,-1,options);
imagesc(Ir_quant); colorbar;
rms_val  = norm(I(:)-Ir_quant(:))
psnr_val = psnr(I,Ir_quant)
```

   Quels effets visuels sont introduits par cette quantification ? Où se trouvent les dégradations les plus visibles ?

   Tracer la courbe du PSNR en fonction du pas de quantification *T* pour `T` variant entre 1 et 256 (une dizaine de valeurs de T suffisent, justifiez votre façon de les choisir).

Codage

On étudie à présent la performance de compression obtenu après transformation par ondelettes. Pour simplifier, le codage sera effectué par un parcours simple de la transformée associé à une quantification globale (en pratique le codage est plutôt contextuel pour de meilleures performances : EBCOT, EZW…).

7. Calculer et afficher l'histogramme de l'image originale. Calculer également l'entropie totale et l'entropie par pixel de cette image.

```
[hI,  bins_I]=integerhist(I(:),[0:255]);
bar(bins_I, hI);

[HI, HI_pix] = integerentropy(I(:));         % Entropie de l'image I
```

8. Quantifier la transformée `W` pour un pas `T`=*5*. Calculer et afficher l'histogramme après quantification, ainsi que l'entropie totale et par pixel.

```
T=5;
[W_quant, W_int]=perform_quantization(W, T);

[hW, bins_W]=integerhist(W_int(:));
bar(bins_W, hW);

[HW, HW_pix] = integerentropy(W_int(:)); % Entropie de W_int
```

9. Comparer l'histogramme et l'entropie de l'image originale et du résultat de cette quantification. Rappeler le lien entre l'entropie et les performances de codage afin de conclure sur l'intérêt de la transformée en ondelette pour la compression d'images.

10. Coder `W_int` en utilisant le codeur arithmétique (la séquence binaire au format *uint8* issue du codage arithmétique se trouve dans le vecteur `y`, et le nombre de bits total dans `nbits`).

```
[y,nbits] = perform_arithmetic_coding(W_int(:),1); % Nb bits total
nbits_pix = nbits / length(W_int(:));              % Nb bits par pixel
```

11. Tracer la courbe du PSNR en fonction du nombre de bits par pixel pour `T` variant entre 1 et 256.

12. Comparer les résultats sur les images "lena256.png" et "rectangles.png" pour les ondelettes "biorthogonal" et "haar" (superposer les courbes pour pouvoir les comparer). Discuter ces résultats. Faire le lien avec la question 3.

13. Ecrire une fonction `dwtcompress` qui prend une image carrée de taille 2^n et renvoie un flux binaire sous la forme d'un vecteur d'entiers 0..255 (uint8) selon la convention suivante : [n (uint8), Jmin (uint8), T (uint8), y (séquence de uint8)]. La compression utilisera l'ondelette biorthogonale.
Vérifier que la fonction `dwtuncompress` fournie arrive à décoder votre flux binaire.

14. Question ouverte : proposer une amélioration à la technique présentée dans ce sujet, qui permettrait d'améliorer le taux de compression ou bien d'obtenir un codage *scalable*, et illustrer sa pertinence.

Références

[1] M.D. Adams, "The JPEG-2000 still image compression standard," ISO/IEC JTC 1/SC 29/WG 1 N 2412. http://www.ece.uvic.ca/~mdadams
[2] D. S. Taubman, M. W. Marcellin. Jpeg2000: Image Compression Fundamentals, Standards, and Practice. 2001.
[3] http://www.jpeg.org/jpeg2000/index.html?langsel=fr