Histogrammes et Applications

Définitions

L'histogramme est une mesure de la répartition d'une grandeur numérique (niveaux de gris, gradient, etc...) dans l'image. Il est analogue (à un coefficient 1/Surface près) à une densité de probabilité. Il est défini par :

h(i) = nombre de pixels de l'image dont le niveau de gris est égal à i.

L'histogramme cumulé représente lui la fonction de répartition statistique au coefficient 1/Surface près: c'est la probabilité pour qu'un point ait un niveau de gris inférieur ou égal à k. Il est défini à partir de l'histogramme simple par :

$$H(k) = \sum_{i=0}^{k} h(i)$$

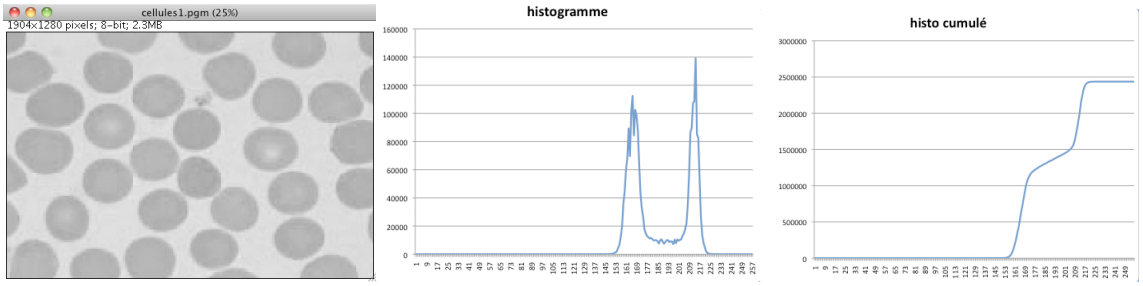

Figure 1 : Une image, son Histogramme et son histogramme cumulé

Amélioration de la perception par Modification d'histogrammes

L'égalisation d'histogramme est une opération ponctuelle qui transforme une image I(x,y) en une image R(x,y) par une fonction f valable pour tous les points de l'image. Cette fonction f est généralement monotone, afin de ne pas modifier le contraste auquel l'opérateur humain est habitué (certaines applications ont cependant utilisé des inversions locales de contraste). Parmi les applications pratiques de cette technique, citons :

- la modification du contraste et de la dynamique d'une image pour en faire ressortir certains détails.
- la correction de la réponse non linéaire d'un capteur quand cette non linéarité est identique pour tous les points de l'image (calibration).
- le passage d'une réponse en transmitance à une réponse en densité (fonction logarithme).
- la comparaison des documents entre eux. Pour comparer des images, les conditions moyennes doivent être identiques, ce qui est rarement le cas. Un histogramme plat permet de ramener toutes les images à une moyenne identique et d'avoir des documents comparables.

Etirement: Transformation linéaire d'un histogramme.

Il faut simplement ici étirer l'histogramme de manière à remplir toute la gamme utile des niveaux de gris par une transformation linéaire. Le niveau k de l'image de départ est donc transformé par :

$$S(k) = Nbgris * \frac{k - \min}{\max - \min}$$ où min (resp. max) est le minimum (resp maximum) de l'image.

Cette transformation doit être appliquée à tous les pixels de l'image en prenant garde au arrondi de calcul et à la division entière (au moins pour les langages dérivant du C).

Egalisation : Transformation d'un histogramme en histogramme plat.

Cette technique donne à la fois la meilleure dynamique possible et un fort contraste, ce qui améliore la visibilité des détails, mais aussi celle du bruit. Elle est souvent utilisée quand on doit comparer des images entre elles ou faire des opérations sur celles-ci. Leur histogramme plat est en effet identique, car il répartit uniformément toutes les valeurs de niveaux de gris sur l'ensemble des pixels de l'image. Chaque niveau de gris apparaît alors 256 fois dans notre cas.

Transformation par histogramme cumulé.

Soit une image I(x,y) avec 0 <= I(x,y) < Nbg, comportant Nbp pixels, et R l'image transformée par la loi S.
Dans le cas continue, on considère les pixels d'une image sont des variables aléatoires, à valeurs dans [0,Nbg − 1]. La loi de transformation S transforme un pixel de niveau k en un pixel de niveau l=S(k).
En choisissant pour S la fonction de répartition de la variable I, on montre qu'on obtient pour R une densité uniforme, ie S est la transformation suivante :

$$S(k) = \frac{Nbg-1}{Nl*Nc}\int_0^k h(i) \quad = \quad \frac{Nbg-1}{Nl*Nc}\ H(k)$$

Cette transformation doit être appliquée à tous les pixels de l'image.

L'histogramme de l'image obtenue alors sera théoriquement plat. En réalité, du fait de la quantification des niveaux de gris (les valeurs de S(k) sont des réels et nous ne prenons que les valeurs entières les plus proches), les valeurs obtenues ne sont que des approximations et l'histogramme n'est pas complètement plat.

Remarque : Voici une façon simple d'obtenir un histogramme réellement plat à partir d'une image déjà égalisée :

- sélectionner itérativement un pixel aléatoirement sur l'image
- Si l'histogramme est plus grand que Nbc*Nbl/Nbg alors on ajoute aléatoirement 1 ou -1 à l'intensité du pixel

Cette procédure est appliquée itérativement.

Filtrage rétinien

Les opérations de type égalisation d'histogramme sont efficaces mais traite les pixels indépendamment de leur contexte. Les études des rétines réelles montrent qu'au contraire, les photo-recepteurs de l'œil sont en général capables de s'adapter à une large gamme de luminance. Ils sont donc plus sensibles dans les valeurs faibles (noir) tout en évitant la saturation dans les zones blanches. Le principe est donc de rendre la loi dépendante non seulement du point concerné, mais aussi des valeurs voisines immédiates et de la valeur moyenne globale. Leur fonction de réponse est du type :

$$R(i,j) = I(i,j) * \frac{255 + M(i,j) + \alpha\bar{I}}{I(i,j) + M(i,j) + \alpha\bar{I}}$$

$$où \quad M(i,j) = \frac{1}{(2*n+1)^2}\sum_{k=-n}^{n}\sum_{l=-n}^{n} I(i+k,j+l) \quad et \quad \bar{I} = \frac{1}{nl*nc}\sum_{k=0}^{nl-1}\sum_{l=0}^{nc-1} I(k,l)$$

Le paramètre α est typiquement de l'ordre de 0,5 et la taille n du voisinage utilisé de l'ordre de 3 ou 5.
C'est une version simplifié du filtre IPL de L. Meylan, D. Alleysson, and S. Süsstrunk, « Model of retinal local adaptation for the tone mapping of color filter array images », Vol. 24, No. 9/September 2007/ J. Opt. Soc. Am.

Contraste local

Une solution pour mesurer la qualité des transformations précédentes est d'utiliser une mesure de contraste local. Cette mesure s'applique sur une fenêtre locale de taille -N,N:

$$C(i,j) = \frac{\max_{(k,l)\in[-N,N]x[-N,N]}(I(i+k,j+l)) - \min_{(k,l)\in[-N,N]x[-N,N]}(I(i+k,j+l))}{\max_{(k,l)\in[-N,N]x[-N,N]}(I(i+k,j+l)) + \min_{(k,l)\in[-N,N]x[-N,N]}(I(i+k,j+l))}$$

La mesure globale sur l'image est simplement la moyenne de cette mesure de contraste sur l'ensemble de l'image

Travail à réaliser

Séances 1 et 2

L'objectif est de comparer l'efficacité des différentes méthodes de rehaussement de contraste présenté ci dessus. Les fonctions d'entrées sorties sur les images au format pgm sont données. On ne travaillera que sur des images en niveaux de gris sur 8 bits. Les fonctions et un exemple de code sont donnés sur le kiosque. Télécharger ces fichiers, compiler avec le makefile fourni. Ce programme lit une image au format pgm, construit une image en

vidéo inverse et stocke cette nouvelle image. Vérifier le bon fonctionnement en visualisant les images à l'aide de la commande display ou du logiciel imagej.

Exercice 1 :

Lire une image, construire son histogramme, trouver le min et le max puis effectuer un étirement d'histogramme et sauver la nouvelle image. Visualiser l'histogramme à l'aide de display ou imagej.
Evaluer qualitativement le résultat, puis quantitativement en réalisant la mesure du contraste. Que pouvez vous dire du résultat sur les images proposées (aquitain.pgm, muscle.pgm, pont.pgm, coucherdesoleil.pgm) ?

Exercice 2 :

Même travail mais en réalisant une égalisation d'histogramme. Visualiser l'histogramme à l'aide de display ou imagej. Que pouvez vous dire du résultat sur les images proposées ?
Améliorer le résultat avec la méthode proposée en remarque. Visualiser l'histogramme. Conclusion ?

Exercice 3 :

Même travail mais en utilisant le filtre rétine. Conclusion ?

Compte rendu :

Comparer qualitativement et quantitativement les 3 solutions sur les images proposées (aquitain.pgm, muscle.pgm, pont.pgm, coucherdesoleil.pgm)

Annexe : fonctions fournies

Principales fonctions d'entrées sorties, de création et conversion

Lecture d'image 8bits dans un fichier pgm

```
unsigned char ** lectureimagepgm(char *name, int* prows, int* pcols)
```

Lecture dans le fichier "filename" de format pgm d'une image en niveau de gris de *prows lignes et *pcols colonnes. Le nombre de ligne et de colonnes est lu dans le fichier "filename". L'image est créée dynamiquement, les pixels remplis puis l'image est retournée. Retourne NULL en cas d'echec

Sauvegarde d'image 8 bits dans un fichier pgm

```
void ecritureimagepgm(char *name, unsigned char **im, int rows, int cols)
```

Sauvegarde dans le fichier "filename" de format pgm d'une image en niveau de gris de rows lignes et cols colonnes. Les nombres de lignes et colonnes doivent être connus. L'image est créée dynamiquement, les pixels remplis puis l'image retournée.

Lecture d'image 8bits dans un fichier pgm

```
double** lectureimagedoubleraw( char *filename, int rows, int cols)
```

Lecture dans le fichier "filename" de format raw d'une image de réels de rows lignes et cols colonnes. Le nombre de ligne et de colonnes est lu dans le fichier "filename". L'image est cree dynamiquement, les pixels remplis puis l'image est retournée. Retourne NULL en cas d'echec

Sauvegarde d'image de réels dans un fichier raw

```
void ecritureimagedoubleraw(char *filename, double **im, int rows, int cols)
```

Sauvegarde dans le fichier "filename" de format raw (ie seulement les données images, pas d'entete) d'une image de réels double de rows lignes et cols colonnes.

Creation dynamique d'image d'octets

```
unsigned char ** alloue_image(int nl, int nc)
```

Allocation dynamique d'un tableau 2D d'octets de nl lignes et nc colonnes. Retourne l'image ainsi créée. La zone de données image est allouée en une seule allocation ce qui permet d'utiliser indifféremment les notations im[i][j] avec 0<=i<n let 0<=j<nc ou (*im)[k] avec 0<=k<nl*nc pour parcourir l'image.

libération d'image

```
void libere_image(unsigned char** im)
```

libere la mémoire allouée pour l'image im.

Creation dynamique d'image de réels double précision

```
double ** alloue_image_double(int nl, int nc)
```

idem précédemment mais pour une image contenant des réels double précision

Conversion d'image d'octets en image de réels double précision

```
double** imuchar2double(unsigned char **im, int nl, int nc)
```

Crée une image de double et recopie l'image d'octets

Conversion d'image de réels double précision en image d'octets

```
unsigned char**imdouble2uchar(double** im,int nl, int nc)
```

Crée une image d'octets et recopie l'image de réels SANS mise à l'echelle. Attention aux dépassements de capacité des octets.

Recopie d'une partie d'image

```
unsigned char** crop(unsigned char **im, int oi, int oj, int fi, int fj)
```

Recopie la partie comprise entre les indices (oi,oj) et (fi,fj) dans une nouvelle image allouée dynamiquement. Retourne cette nouvelle image.

Principales fonctions liées à la FFT

Puissance de 2

```
int nextpow2( double num )
```

Détermine la puissance de 2 juste supérieure ou égale à num

```
int ispowerof2(double num)
```

vrai si num est une puissance de 2

Transformée de fourrier rapide et inverse

```
int fft( double** ims_reel, double** ims_imag, double** imd_reel, double** imd_imag , int dimx, int dimy)
```

Calcul la FFT de l'image complexe (partie réelle `ims_reel` et imaginaire `ims_imag`). Le résultat complexe est donné par les images partie réelle `imd_reel` et imaginaire `imd_imag`
Les dimensions `dimx` et `dimy` doivent être des puissances de 2.

```
int ifft( double** ims_reel, double** ims_imag, double** imd_reel, double** imd_imag , int dimx, int dimy)
```

idem, mais transformée inverse

```
void fftshift( double** imsr, double** imsi, double** imdr, double** imdi, int nl, int nc )
```

Centre la transformée de fourrier au milieu de l'image

```
double** padimdforfft(double** im, int* pnl, int* pnc)
```

Si les dimensions `*pnl, *pnc` ne sont pas des puissances de 2, crée une image dynamique de taille 2^k, copie l'image initiale de réels double précision et complète par des zéros. **Change** les valeurs `*pnl, *pnc`. Retourne la nouvelle image créée.

```
double** padimucforfft(unsigned char** im, int* pnl, int* pnc)
```

Idem, mais pour une image d'octets.