TME espaces d'échelles

NI600-AMO/IMA/Master Informatique/UPMC

Jeudi 16 janvier

Le matériel utile pour ce TME est disponible à cet endroit :

`/Infos/lmd/2013/master/ue/amo-2013oct`

recopiez le contenu de ce répertoire dans votre compte. Le répertoire contient quelques images. Vous en trouverez bien davantage sur l'Internet, en particulier ici : `http://www.hlevkin.com/default.html#testimages`.

1 Notations

Cas continu

Le domaine spatial Ω représente un rectangle (l'image). On le suppose de taille $[1, N_x] \times [1, N_y]$.
Soit la fonction $I : ... \rightarrow ...$ tel que $(x, y, t) \mapsto I(x, y, t)$ qui représente une famille d'images filtrées.
On note $I_t = \frac{\partial I}{\partial t}$ et de même pour I_x et I_y. On note $I_{xx} = \frac{\partial^2 I}{\partial x^2}$.

Cas discret

Le domaine Ω est discrétisé selon une grille régulière de pixels :

$$\begin{aligned} x_i &= i \qquad & i &= 1 \cdots N_x \\ y_j &= j \qquad & j &= 1 \cdots N_y \end{aligned}$$

et le temps par : $t_k = k \triangle t, \quad k = 0 \cdots k_{max}$. La fonction continue I est alors approchée par l'ensemble de valeurs :

$$I(x_i, y_j, t_k) \quad i = 1 \cdots N_x, \quad j = 1 \cdots N_y, \quad k = 0 \cdots k_{max}$$

Par la suite on note la fonction discrétisée $I_{i,j,k}$, et de fait on a $I_{i,j,k} = I(x_i, y_j, t_k), \forall i, j, k$.

2 Rappels de programmation

2.1 Lecture/écriture d'images

2.1.1 En C/C++

En C, on utilisera la bibliothèque `Imlib2`. Dans le répertoire C, vous trouverez le module `imlib-io.c` qui propose une routine de lecture et une d'écriture d'image. La routine de lecture lit une image, la transfère dans un tableau de flottants et retourne l'adresse de ce tableau. Les images couleurs sont transformées en niveaux de gris.

```
int Nx, Ny;

double *MonImage = read_grayscale( "monimage.jpg", &Nx, &Ny);
```

L'image contient Ny ligne de Nx pixels. Les pixels sont rangés ligne par ligne dans le tampon, ce qui fait que pour accéder au pixel de la ligne i et de la colonne j on écrira :

```
int i, j;

i=10; j=15;
printf("I(%d,%d)=%f\n", i, j, MonImage[j+i*Ny]);
```

L'écriture est simple :

```
write_grayscale("monimage.jpg", Nx, Ny, MonImage);
```

Le fichier `imlib-demo.c` est un exemple complet : il charge une image, applique un seuil et sauvegarde le résultat. Le nom des images en entrée et en sortie, ainsi que la valeur de seuil sont passés en paramètre de la commande.

Voici un exemple d'utilisation dans le terminal texte. Pour compiler le programme :

```
% gcc imlib-exemple.c -o imlib-test `imlib2-config --cflags --libs`
```

et pour le tester :

```
% imlib-exemple lena.png lenaSeuil.png 128
```

Et enfin pour voir le résultat :

```
% eog lenaSeuil.png
```

Dans la suite du TME, recopiez le fichier `imlib-demo.c` en lui donnant le nom de l'exercice, puis modifiez la fonction de traitement pour l'adapter à l'exercice.

En Octave/Matlab

On utilisera les fonctions `imread()` et `imsave()` pour lire/écrire des images.

```
# Lecture
Img = imread('monimage.jpg');
# Si image couleur, convertir en niveau de gris
Ndg = (Img(:,:,1)+Img(:,:,2)+Img(:,:,3))/3
# Ecriture
imsave('monimage2.png',Img);
# Visualisation
imshow(Img);
```

2.2 Calcul de dérivés d'images

Les opérateurs différentiels, tels que la dérivée partielle, sont approchés par des différences finies. Considérons la dérivée partielle par rapport à x. On écrira par exemple :

$$I_x(x_i, y_j, t_k) \simeq I_{i+1,j,k} - I_{i,j,k}$$

2.2.1 En C/C++

On doit calculer, pour tous les pixels de l'image la différence entre le pixel i, j et son voisin au Sud. D'où l'algorithme, en supposant qu'il n'y a pas d'indice k :

```
void diff_est( double *Iout, *I, int Nx, int Ny) {
  int i,j;
  for( i=0; i<Nx-1; i++)
```

```
    for( j=0; j<Ny; j++)
      Iout[j+i*Ny] = I[j+(i+1)*Ny] - I[j+i*Ny];
  /* cas de la derniere ligne non calculable: on la met a zero */
  i = Nx-1;
  for( j=0; j<Ny; j++)
    Iout[j+i*Ny] = 0;
}
```

2.3 En Octave/Matlab

En Matlab, faire des boucles sur les indices de tableau est inefficace. Pour calculer $I_{i+1,j,k} - I_{i,j,k}$, on remarque simplement qu'il faut prendre la matrice I, **la décaler d'un indice vers le haut** et la soustraire à I. Soit en Matlab :

```
Iout = translateImage(I,0,-1) - I;
```

La routine `translateImage` est fourni dans `translateImage.m`. Lisez le code et remarquez qu'elle se contente de translater l'image selon le vecteur entier que vous donnez en second et troisième paramètre.

3 Diffusion linéaire

En continu, I vérifie l'équation 2D de la chaleur. Pour tout $(x, y) \in \Omega$ on a :

$$\begin{aligned} I_t(x,y,t) &= \frac{1}{2}(I_{xx} + I_{yy}) \quad t > 0 \\ I(x,y,0) &= I^0(x,y) \end{aligned}$$

où I^0 est l'image à filtrer.

En discret, l'équation de la chaleur est approchée en utilisant un schéma avant en temps et centré en espace :

$$I(i,j,k+1) = I(i,j,k) + \frac{1}{2}\triangle t[I(i-1,j,k) + I(i+1,j,k) + I(i,j-1,k) + I(i,j+1,k) - 4I(i,j,k)]$$

Exercices :

- Implémentez l'équation de la chaleur. Le programme admettra deux paramètres : le pas de temps $\triangle t$ et le nombre d'itérations k_{max}. On rappelle qu'à itération k, on obtient une représentation de I^0 à l'échelle $t = k\triangle t$. On remarquera qu'il est inutile de maintenir un indice k dans les tableaux puisque le calcul à l'étape k ne dépend que de l'image à l'étape $k-1$.
- Testez votre programme sur différentes images. On rappelle que le schéma utilisé ici en 1D est stable si $\triangle t < 1$ (voir cours 2 page 21). En 2D, la condition devient $\triangle t < \frac{1}{2}$. Vérifiez ce fait expérimentalement : choisissez une cinquantaine d'itérations et faîtes varier $\triangle t$. Choisissez une valeur en dessous et proche de 1 puis faîtes varier le nombre d'itérations.

4 Diffusion non linéaire isotropique

En continu, l'équation 2D de diffusion non linéaire s'écrit, pour $(x, y) \in \Omega$:

$$\begin{aligned} I_t(x,y,t) &= \nabla.(c(x,y,t)\nabla I) \quad t > 0 \\ I(x,y,0) &= I^0(x,y) \end{aligned}$$

où c est une fonction (elle représente la diffusivité). Perona et Malik proposent pour cette fonction :

$$c(x,y,t) = g\left(\|\nabla I(x,y,t)\|\right)$$

avec g une fonction réelle à décroissance rapide. Les deux choix possibles pour la fonction g sont :

— $g(x) = e^{-\left(\frac{x}{K}\right)^2}$, fonction de Tukey,

— $g(x) = \frac{1}{1+\left(\frac{x}{K}\right)^{1+\alpha}}, \quad \alpha > 0$, fonction de Lorentz.

Le paramètre K est un contraste sur la carte des contours (norme du gradient d'image). Sa plage de valeurs correspond à la dynamique de l'image à filtrer. On utilisera plutôt la fonction de Tukey qui contient moins de paramètres.

Perona et Malik ont proposé le schéma discret suivant (voir cours 3 page 14) :

$$I_{i,j,k+1} = I_{i,j,k} + \triangle t[C^N \nabla^N I + C^S \nabla^S I + C^W \nabla^N I + C^E \nabla^S I]_{i,j,k}$$

avec :

— $\nabla^N I_{i,j,k} = I_{i-1,j,k} - I_{i,j,k}$ et $C^N_{i,j,k} = g(\|\nabla^N I_{i,j,k}\|)$

— $\nabla^S I_{i,j,k} = I_{i+1,j,k} - I_{i,j,k}$ et $C^S_{i,j,k} = g(\|\nabla^S I_{i,j,k}\|)$

— $\nabla^W I_{i,j,k} = I_{i,j-1,k} - I_{i,j,k}$ et $C^W_{i,j,k} = g(\|\nabla^W I_{i,j,k}\|)$

— $\nabla^E I_{i,j,k} = I_{i,j+1,k} - I_{i,j,k}$ et $C^E_{i,j,k} = g(\|\nabla^E I_{i,j,k}\|)$

Exercices

— Comme pour l'exercice précédent, implémentez ce schéma numérique. Cette fois nous avons un paramètre supplémentaire K qui règle le type de diffusion : les contours dont le gradient sont plus petits en norme que K seront lissés, et les autres seront préservés. Une valeur élevée de K donne une diffusion linéaire alors qu'une valeur basse donne une diffusion non linéaire.

— Expérimentez les conditions d'instabilité du schéma en faisant varier $\triangle t$.

5 Diffusion non linéaire anisotropique

L'équation généralisée de la diffusion s'écrit :

$$\begin{aligned} I_t(x,y,t) &= \nabla.(D(x,y,t)\nabla I) \quad t > 0 \\ I(x,y,0) &= I^0(x,y) \end{aligned}$$

où D est un tenseur 2D, c'est-à-dire une matrice symétrique définie positive. Le tenseur permet de définir un processus qui dépend cette fois de la direction de diffusion. En pratique, cette direction de diffusion sera calculée à partir des configurations locales de l'image. D étant symétrique, on note a, b et c ses éléments, de sorte que $D = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$.

Voici une approximation possible de l'équation précédente (voir cours 3, page 48) :

$$\begin{aligned} I_{i,j,k+1} = I_{i,j,k} + \triangle t \Bigg[& -\frac{b_{i-1,j} + b_{i,j+1}}{4} I_{i-1,j+1,k} + \frac{c_{i,j+1} + c_{i,j}}{2} I_{i,j+1,k} \\ & + \frac{b_{i+1,j} + b_{i,j+1}}{4} I_{i+1,j+1,k} + \frac{a_{i-1,j} + a_{i,j}}{2} I_{i-1,j,k} \\ & - \frac{a_{i-1,j} + 2a_{i,j} + a_{i+1,j} + c_{i,j-1} + 2c_{i,j} + c_{i,j+1}}{2} I_{i,j,k} \\ & + \frac{a_{i+1,j} + a_{i,j}}{2} I_{i+1,j,k} + \frac{b_{i-1,j} + b_{i,j-1}}{4} I_{i-1,j,k} \\ & + \frac{c_{i,j-1} + c_{i,j}}{2} I_{i,j-1,k} - \frac{b_{i+1,j} + b_{i,j-1}}{4} I_{i+1,j-1,k} \Bigg] \end{aligned} \tag{1}$$

Comme on le voit, ce schéma est un peu long à écrire. Il vous est fourni avec dans le matériel du TME (fichiers `M/tnld.m` et `C/tnld.c`). Pour utiliser ce schéma, il suffit alors de "plugger" les bonnes valeurs dans a, b et c.

Diffusion "Edges Enhancing"

On choisit $D = R_\sigma^T L R_\sigma$ avec $R_\sigma = \begin{pmatrix} I_x^\sigma & I_y^\sigma \\ -I_y^\sigma & I_x^\sigma \end{pmatrix}$ $L = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$ où I^σ est la représentation à l'échelle σ^2 de I. Autrement dit à l'instant t, on prend I et on la lisse avec un noyau gaussien d'écart-type σ. R_σ exprime un changement de repère, on se place dans le repère orienté selon ∇I^σ. Les paramètres λ_1 et λ_2 règlent la quantité de diffusion selon respectivement les directions ∇I^σ et la direction orthogonale à ∇I^σ. En calculant explicitement D, on trouve :

$$a = \frac{\lambda_1 \left(I_x^\sigma\right)^2 + \lambda_2 \left(I_y^\sigma\right)^2}{\|\nabla I^\sigma\|^2} \tag{2}$$

$$b = \frac{(\lambda_1 - \lambda_2) I_x^\sigma I_y^\sigma}{\|\nabla I^\sigma\|^2} \tag{3}$$

$$c = \frac{\lambda_2 \left(I_x^\sigma\right)^2 + \lambda_1 \left(I_y^\sigma\right)^2}{\|\nabla I^\sigma\|^2} \tag{4}$$

Le paramètre λ_1 contrôle la diffusion dans la direction $\nabla^\sigma I$ et λ_2 contrôle la diffusion dans la direction orthogonale. Pour lisser le long des contours, il faut donc que $\lambda_2 >> \lambda_1$. Weickert propose :

$$\lambda_1 = \frac{1}{5}\lambda_2 \tag{5}$$

$$\lambda_2 = \exp\left(-\frac{\|\nabla^\sigma I\|^2}{K^2}\right) \tag{6}$$

Exercices :

- Implanter la diffusion "Edges Enhancing" en utilisant le schéma (**??**) fourni et en définissant les bonnes valeurs pour a, b et c. Pour cela nous aurons besoin d'appliquer une convolution à l'image I_k avec un noyau gaussien. Les sources de la convolution gaussienne vous sont également fournies :
  - pour Matlab, le fichier `M/gD.m` (et le fichier auxiliaire `M/convSepBrd.m`). Pour l'utiliser, se reporter au cours 1, page 59.
  - en C, le fichier `C/bconvol.c` contient la routine gblur() qui s'utilise ainsi :

```
double sigma = 3;
/* in et out sont deux tampons double a Nx colonnes et Ny lignes */
gblur( out, in, Nx, Ny, sigma);
```

- Le programme possède les paramètres suivants : $\triangle t$, k_{max}, σ, et K. σ contrôle la taille du voisinage dans lequel on calcule la direction locale du gradient de I^σ. K contrôle la diffusion non linéaire. Expérimentez le code avec différentes valeurs pour ces paramètres.