

TP Traitement d'images 3D – recalage déformable 3D

(3 heures)
2018-2019

Ce TP a pour objectif de vous faire programmer une version de base de l'algorithme de recalage déformable par « démons » vu en cours.

- **Le TP est noté : le compte-rendu doit être envoyé sous forme électronique à : gerard.subsol@lirmm.fr avant le mercredi 12 décembre 2018 (minuit)**
- *Tout compte-rendu envoyé sous une mauvaise forme ou hors-délai sera sanctionné par un 0.*
- *Le compte-rendu doit inclure **la date, vos noms** et être composé d'au moins 1 à 2 pages de texte décrivant la méthode utilisée pour répondre aux questions ainsi que les **commandes** lancées **AVEC quelques captures d'écran** pour évaluer le résultat.*
- *Le tout doit être sous la forme d'un **unique fichier pdf**.*
- *Le TP peut se faire seul ou en binôme*
- *La participation active pendant le TP pourra aussi être prise en compte.*
- **Tout plagiat sera lourdement sanctionné.**

Soient deux images 3D F et G, de même taille, avec une dimension isotrope du voxel identique.

1. A partir du cours (voir ci-après), **écrire l'algorithme en pseudo-code** pour obtenir un champ de déformations 3D à appliquer sur l'image G pour qu'elle ressemble à l'image F. Programmer la fonction.
2. Il faut maintenant appliquer le champ de déformations 3D ? L'astuce consiste à utiliser le champ de déformation de G vers F pour déformer F qui sera en fait l'image flottante.
Expliquer l'idée sur un schéma. Programmer la fonction.
3. Programmer une **version la plus simple possible** de l'algorithme des « démons ». Le programme devra prendre en entrée 2 images, F et G, ainsi qu'un nombre d'itérations. Il déformera F vers G et devra donner en sortie une image déformée F' qui soit ressemblante à G.
4. Tester votre algorithme sur les images fournies (BRAIN.SMALL et BRAIN). On pourra aussi déformer l'atlas (qui se superpose sur l'image 1) pour avoir un atlas superposable sur l'image 2.

Annexe 1 : quelques fonctions Cimg qui peuvent être utiles

```
CimgList<Tfloat> get_gradient ( const char *const axes = 0,
                                const int      scheme = 3
                                ) const
```

Return image gradient.

Parameters

axes Axes considered for the gradient computation, as a C-string (e.g "xy").

scheme = Numerical scheme used for the gradient computation:

- -1 = Backward finite differences
- 0 = Centered finite differences
- 1 = Forward finite differences
- 2 = Using Sobel kernels
- 3 = Using rotation invariant kernels
- 4 = Using Deriche recursive filter.
- 5 = Using Van Vliet recursive filter.

```
Cimg<T>& blur ( const float  sigma_x,
                  const float  sigma_y,
                  const float  sigma_z,
                  const bool    boundary_conditions = true,
                  const bool    is_gaussian = false
                  )
```

Blur image.

Parameters

sigma_x Standard deviation of the blur, along the X-axis.

sigma_y Standard deviation of the blur, along the Y-axis.

sigma_z Standard deviation of the blur, along the Z-axis.

boundary_conditions Boundary conditions. Can be { false=dirichlet | true=neumann }.

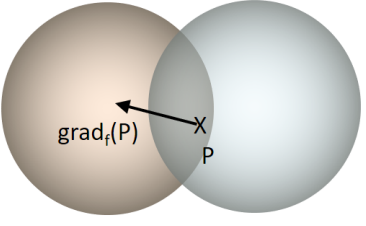
is_gaussian Tells if the blur uses a gaussian (true) or quasi-gaussian (false) kernel.

```
Tfloat linear_atXYZ ( const float fx,
                      const float fy = 0,
                      const float fz = 0,
                      const int   c = 0
                      ) const
```

Return pixel value, using linear interpolation and Neumann boundary conditions for the X,Y and Z-coordinates.

Annexe 2 : rappels de cours

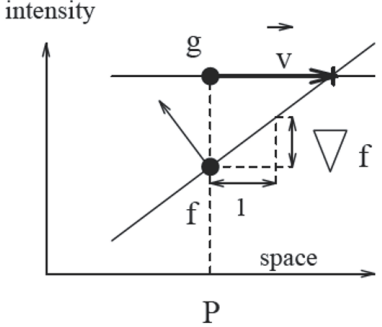
f : image fixe g : image flottante



- Au voxel P : $f = I_f(P)$ et $g = I_g(P)$
- On a $g > f$ (g plus clair que f)
- Il faudrait donc déplacer P vers un voxel où f est plus grand.
- Pour cela, on calcule $\text{grad}_f(P)$ qui donne la direction dans laquelle f augmente le plus autour de P.
- On « déplace » P dans la direction de $\text{grad}_f(P)$.

- Approximation linéaire de la variation de f le long du gradient
- Pour que $g = f$, il faut le déplacer suivant le vecteur :

$$\vec{v} = \frac{(g - f) \vec{\nabla} f}{\vec{\nabla} f^2}$$



P

- En tout voxel, on a un vecteur de déplacement.
- On régularise ce champ de vecteurs (ex. lissage gaussien des coordonnées)
- On applique la déformation ainsi définie à l'image flottante g
- On itère... jusqu'à ce que le champ de vecteurs de déformation soit "petit".

3.3. Deformation Regularization

Chapter 3. Background

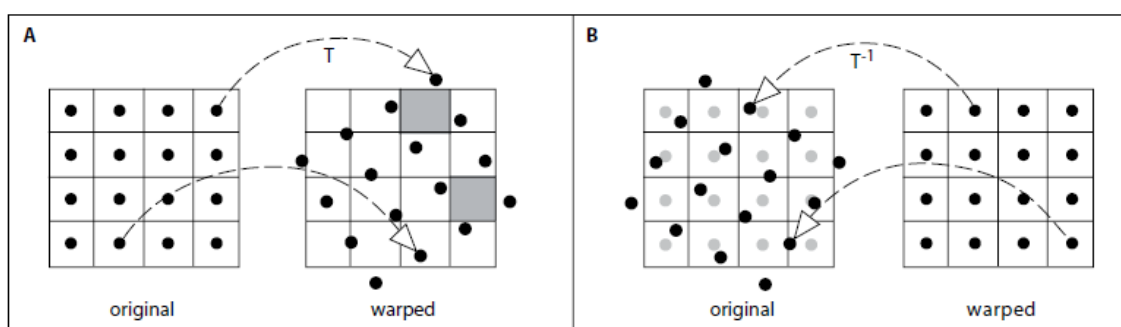
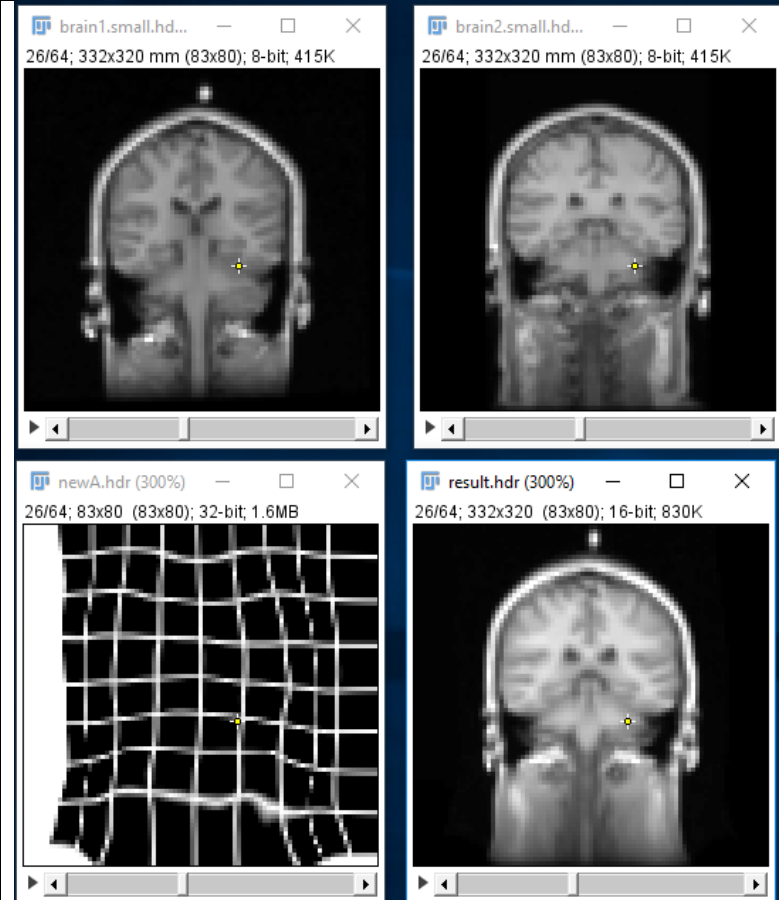
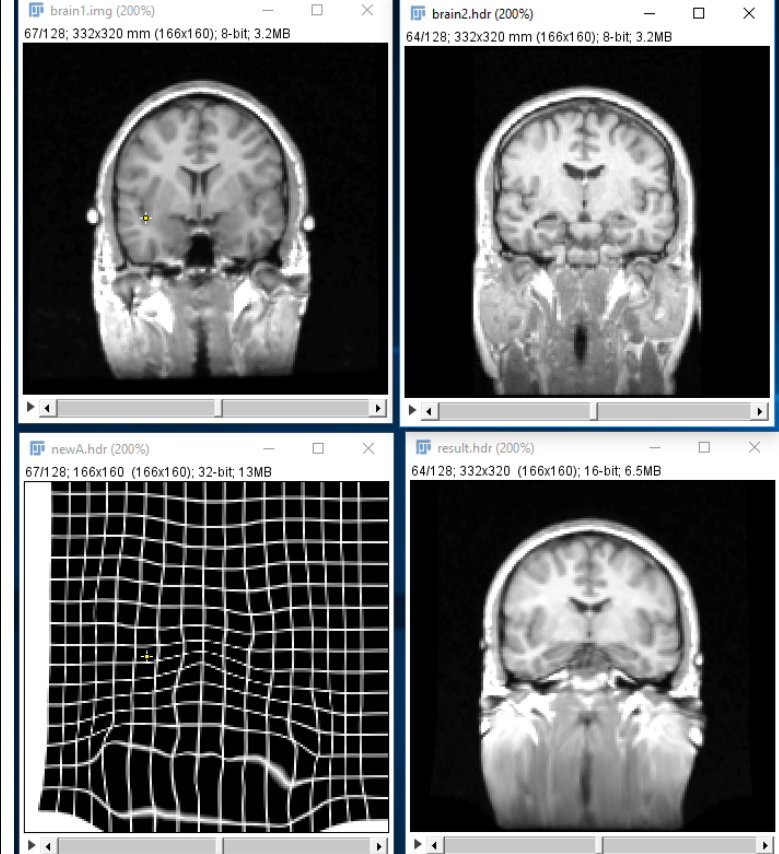


Figure 3.2.: Forward and backward image warping. In the case of forward warping (A), holes can occur in the warped image, marked in gray. Backward warping (B) eliminates this problem since intensities at locations that do not coincide with pixel coordinates can be obtained from the original image using an interpolation scheme.

				Brain_small Sigma=2 1000 itérations
				Brain Sigma=3 1000 itérations

