Modèles de transformation pour le recalage d'images

D. Legland

24 août 2010

Résumé

Notes sur l'implémentation des classes de transformation géométrique utilisées pour recaler les images.

Table des matières

1 Architecture globale

L'idée est de pouvoir manipuler simplement des familles de transformations paramétrées, de manière à identifier les paramètres optimaux pour un modèle de transformation donné. L'identification des paramètres optimaux se fait via des algorithmes d'optimisation non détaillés ici.

Pour la conception, je suis parti d'une architecture de classes, de la plus générale à la plus particulière. Les classes de transformations peuvent être génériques pour les dimensions (exemple : les transfos affines). Quand ce n'est pas possible, j'ajoute un suffixe « 2D » ou « 3D » selon les cas.

La classe de base est la classe Transform. La classe ParametricTransform ajoute la manipulation des paramètres à optimiser. On trouve aussi des classes permettant de limiter la ré-écriture des transformations spécialisées.

1.1 Transform

Il s'agit de la classe la plus générique. Elle définit principalement la méthode transformPoint, ainsi que quelques méthodes utilitaires.

transformPoint transforme un point en un autre

transformVector transforme un vecteur en un autre

getJacobian renvoie la matrice jacobienne (des dérivées premières en fonction des coordonnées) pour une position donnée

compose renvoie un nouvelle transformation, résultat de l'application de la transformation passée en paramètre suivie de cette transformation

Toutes ces méthodes sont abstraites, et nécessitent d'être implémentées dans les classes dérivées.

1.2 ParametricTransform

Cette classe (abstraite) ajoute la gestion des paramètres. Elle définit un champs « params », sous la forme d'un vecteur ligne, qui peut être modifié via les méthodes appropriées.

Pour distinguer les transformations paramétrées des transformations non paramétrées, j'ajoute parfois le suffixe « Model » au nom de la classe. Exemple : « TranslationModel ».

Méthodes implémentées

On a tout d'abord quelques méhodes pour manipuler les paramètres :

getParameters renvoie le vecteur de paramètres

setParameters modifie le vecteur de paramètres

getParameterLength renvoie la taille du vecteur de paramètres

On a aussi quelques méthodes pour faciliter l'interprétation

getParameterNames renvoie un tableau de chaînes contenant le nom de chaque paramètre

getParameterName renvoie le nom du i-ème paramètre

Enfin, on a aussi une méthode pour calculer les dérivées mais en fonction des paramètres :

getParametricJacobian calcule la matrice jacobienne, qui a autant de lignes que le nombre de dimensions, et autant de colonnes que le nombre de paramètres

1.3 AffineTransform

Cette classe abstraite sert de base aux autres transformations affines. Elle implémente plusieurs méthodes qui se basent sur la matrice de transformation associée. À noter que la matrice de transformation n'est pas définie dans cette classe, mais dans les classes dérivées.

Méthodes abstraites

getAffineMatrix renvoie la matrice affine associées à cette transformation

Il s'agit de la seule méthode abstraite de cette classe, et donc la seule qui nécessite une implémentation par les classes dérivées.

Méthodes implémentées

Les méthodes suivantes définies par les interfaces sont implémentées par la classe AffineTransform :

transformPoint transforme un point en un autre

transformVector transforme un vecteur en un autre

getJacobian renvoie la matrice jacobienne (des dérivées premières en fonction des coordonnées) pour une position donnée

Note : pour une transformation affine, la matrice jacobienne est obtenue en isolant la partie linéaire de la matrice de transformation.

compose renvoie un nouvelle transformation, résultat de l'application de la transformation passée en paramètre suivie de cette transformation

On a aussi une méthode spécifique aux transformations affines

getInverse renvoie la transformation affine inverse

2 Implémentations utilitaires

2.1 MatrixAffineTransform

Implémentation classique d'une transformation affine, sous la forme d'une matrice. Utilisées pour des tests, ou pour implémenter des transformations classiques.

2.2 ComposedTransform

Cette classe stocke un tableau de transformations. Lors de l'appel à une des méthodes de calcul (par exemple « transformPoint »), elle appelle la méthode correspondante pour chaque transformation stockée, et combine les résultats.

Aucune hypothèse n'est faite sur les transformation stockées, elles doivent juste hériter de « Transform », et être définies pour les mêmes dimensions.

3 Modèles de transformation

Toutes les transformations décrites ici héritent à la fois de Transform (et même de AffineTransform pour le moment) ainsi que de ParametricTransform.

3.1 TranslationModel

Une translation en dimension arbitaire, définie par un vecteur de translation.
La matrice associée est donnée par :

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & \theta_1 \\ 0 & 1 & \theta_2 \\ 0 & 0 & 1 \end{bmatrix}$$

en deux dimensions, ou par

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & \theta_1 \\ 0 & 1 & 0 & \theta_2 \\ 0 & 0 & 1 & \theta_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

en trois dimensions.
La matrice jacobienne en fonction des paramètres est donnée par

$$\mathbf{J}_\theta(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.2 CenteredMotionTransform2D

Il s'agit d'une transformation définie par une rotation autour d'un centre donné (par défaut l'origine), suivie d'une translation. On a donc 3 paramètres. L'angle de rotation est stocké en degrés, et converti en radians pour les calculs. Le centre de la rotation est un paramètre non optimisable.

La matrice de transformation est donnée par :

$$\mathbf{M} = \begin{bmatrix} \cos\theta & -\sin\theta & c_x(1-\cos\theta) + c_y\sin\theta + u_x \\ \sin\theta & \cos\theta & c_y(1-\cos\theta) - c_x\sin\theta + u_y \\ 0 & 0 & 1 \end{bmatrix}$$

où c_x et c_y sont les coordonnées du centre de la transformation, et u_x et u_y sont les composantes du vecteur de translation.

La matrice jacobienne des dérivées par rapport aux paramètres est donnée par

$$\mathbf{J}_\theta(\mathbf{x}) = \begin{bmatrix} -(x-c_x)\sin\theta - (y-c_y)\cos\theta & 1 & 0 \\ (x-c_x)\cos\theta - (y-c_y)\sin\theta & 0 & 1 \end{bmatrix}$$

avec $\mathbf{x} = (x, y)$

3.3 CenteredEulerTransform3D

Cette classe représente une transformation rigide obtenue en combinant trois rotations successives suivies d'une translation. On a donc 6 paramètres.

Il y a de multiples manières de choisir les angles d'Euler. Pour cette classe, les rotations sont choisies ainsi, dans cet ordre :

1. une rotation autour de l'angle O_x selon un angle φ
2. une rotation autour de l'angle O_y selon un angle θ
3. une rotation autour de l'angle O_z selon un angle ψ

Pour une rotation autour de l'origine, on obtient la matrice de transformation suivante :

$$\mathbf{R}_{XYZ,\mathbf{o}} = \begin{bmatrix} c_y c_z & s_x s_y c_z - c_x s_z & c_x s_y c_z + s_x s_z & 0 \\ c_y s_z & s_x s_y s_z + c_x c_z & c_x s_y s_z - s_x c_z & 0 \\ -s_y & s_x c_z & c_x c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

avec $c_x = \cos\varphi$, $s_x = \sin\varphi$, $c_y = \cos\theta$...

La transformation autour d'un centre arbitraire s'obtient numériquement, en composant avec les matrices de transformation des translations directes et inverses.

$$\mathbf{R}_{XYZ,\mathbf{c}} = \mathbf{T_c}\mathbf{R}_{XYZ,o}\mathbf{T}_{-\mathbf{c}}$$

Pour la matrice Jacobienne, les calculs sont un peu laborieux, mais on retombe sur la version codée dans ITK.

4 Conclusions

On a une hiérarchie de classes de transformations, génériques selon la dimension.
Pour le moment, on a uniquement des transformations affines.
Parmi les points à travailler pour la suite :

- gestion des transfos élastiques
- gestion des fonctions de régularisation associées aux transformations
  - écart à l'origine pour les déplacements
  - intégrale de la norme du jacobien pour les transfos fluides ?