TP contours déformables : les snakes

4 mars 2015

1 Introduction

1.1 Objectif

Ce TP a pour objectif de vous faire appréhender une technique de détection par contours actifs : les SNAKES. Les composantes de ce TP sont les suivantes :

1 Implémentation C/C++ d'un algorithme de snakes
2 Caractérisation de l'algorithme en fonction des paramètres
3 Modification du modèle

1.2 À rendre

Consigne à observer Le TP se fait en C/C++ pour la partie algorithmique et sous *octave* pour la visualisation de l'évolution des snakes. Aucun IDE n'est imposé, cependant, si vous en utiliser un, nous conseillons fortement *Qtcreator* (le fichier cmake est fourni).

Consignes pour le rendu à remettre avant le 15 mars 23h55

À rendre sur le dépôt sur e-campus une archive nommée : TP_SNAKE_NOM1_NOM2.tar. Pour générer cette archive, utilisez la commande :

tar -cvf TP_SNAKE_NOM1_NOM2.tar fichier1 fichier 2... fichierN

qui génère l'archive TP_SNAKE_NOM1_NOM2.tar contenant les fichiers fichier1, fichier2 jusqu'à fichierN. Le contenu de l'archive est le suivant :

— un compte rendu qui explique ce que fait votre code en 5 ligne, et une page pour expliquer vos experiences
— votre code pour calculer l'evolution du snake (il faut que le code compile) qui comporte les fichier sources, un fichier pour compiler (makefile ou cmake) et les resources (images).
— un script pour afficher vos resultats.

Dans le cas ou votre script affiche des figures, n'oubliez pas de mettre des titres, des légendes, les labels, les conditions d'exécution, les paramètres de l'algorithme... tout ce qui permet de comprendre la figure car elle ne parle pas d'elle même (on voit clairement sur la figure... à bannir car le lecteur ne voit pas la même chose que vous).

Si votre script affiche des résultats dans la console, vous pouvez utiliser la commande :

printf('mon texte avec des variables %i %f', var1, var2)

qui vous permet d'afficher proprement du texte dans la console avec des valeurs (comme la fonction *printf* en C). Encore une fois, les valeurs ne parlent pas d'elle mêmes. Elles ont besoin d'avoir une interprétation. Est-ce qu'elles appartiennent à un domaine spécifique, est-ce des valeurs grandes/petites par rapport à quelque-chose, ont elles des unités, est-ce cohérent avec

ce à quoi vous vous attendiez et pourquoi, est-ce une valeur aléatoire ou déterministe, est-ce un paramètres, est-ce une valeur moyenne, y a-t-il besoin d'autant de chiffres après la virgule (significatif)... bref, expliciter les valeurs.

Remarque : touts les codes et scripts doivent être commentés.

2 Rappel de cours

2.1 Formulation du probleme

Cahier des charges On souhaite segmenter des objets dans une image en les entourant avec une courbe paramétrée fermée. On va donc utiliser les outils mathématiques liés aux courbes paramétriques. On rappelle qu'une courbe (Γ) peut se paramÃ©trer par l'application γ :

$$\begin{aligned} \gamma : [0,1] &\to \mathbb{R}^2 \\ s &\mapsto \begin{pmatrix} \gamma_x(s) \\ \gamma_y(s) \end{pmatrix} \end{aligned}$$

ou $\gamma_x(s)$ et $\gamma_y(s)$ sont les coordonnées de la courbe.

Une courbe avec de bonnes propriétés Deux propriétés sont dÃ©sirées pour une courbe : la continuité et la régularité (lisse). La continuité se traduit par une norme de la dÃ©rivée qui doit être fini et la régularité impose des derivees secondes de normes finies. Pour définir une énergie pénalisant les mauvaises configurations de Γ, on ajoute toutes les petites contributions de la norme de la dérivée et de la dérivée seconde sur le long de la courbe en pondérant les deux termes :

$$\mathcal{E}_{\text{int}}(\gamma) = \int_0^1 \frac{1}{2} \left(\lambda_1 \|\gamma'(s)\|_2^2 + \lambda_2 \|\gamma''(s)\|_2^2 \right) \mathrm{d}s$$

avec λ_1 et λ_2 des réels positifs modulant les effets de chaque terme.

Collage aux contours... Une autre bonne propriété de la courbe est quelle soit collée aux contours des objets que l'on souhaite segmenter. Pour cela, on cherche un critère qui se minimise lorsque la courbe vient se positionnée sur le contour de l'objet à segmenté. Plusieurs solutions sont possibles, parmi lesquelles on trouve :

1. la norme du gradient de l'image (son opposé) qui passe par un extremum sur les changements brusque d'intensités qui ont tendances à représenter des contours,
2. l'image elle même dans le cas de dessins : contours linéaires fins foncés sur fond clair,
3. la transformée de distance dans le cas des images binaires : le gradient étant nul loin des contours (2 pixels) la courbe ne sera pas attirée par les zones de transitions, cependant, la transformée de distance est élevée lorsqu'on est loin des contours d'un objet et nulle dans l'objet (point ã prendre en compte pour ne pas laisser le snake se contracter en une singularité).

Pour le premier cas, on définit l'énergie externe comme la contribution de touts les lieux de la courbe pondérés par la valeur de la norme du gradient au carrée (circulation le long de la courbe), et on l'écrit :

$$\mathcal{E}_{\text{ext}}(\gamma) = \int_0^1 -\lambda_3 \left(\frac{\partial \mathrm{I}}{\partial x}^2 + \frac{\partial \mathrm{I}}{\partial y}^2 \right) (\gamma) \mathrm{d}s$$

avec λ_3 un réel positif pondérant la contribution de l'énergie liée à l'image.

Au total, nous avons la formulation suivante de l'énergie :

$$\mathcal{E}_{\text{tot}}(\gamma) = \int_0^1 \tfrac{1}{2}\left(\lambda_1\|\gamma'(s)\|_2^2 + \lambda_2\|\gamma''(s)\|_2^2\right) - \lambda_3\left(\tfrac{\partial \mathrm{I}}{\partial x}^2 + \tfrac{\partial \mathrm{I}}{\partial y}^2\right)(\gamma)\mathrm{d}s \quad (1)$$

$$= \int_0^1 \mathrm{E}_{\text{int}}(\gamma', \gamma'') + \mathrm{E}_{\text{ext}}(\gamma)\mathrm{d}s \quad (2)$$

Pour minimiser $\mathcal{E}_{\text{tot}}$, nous allons avoir recours aux équations d'Euler-Lagrange.

2.2 Minimisation de fonctionnelle par equation d'Euler-Lagrange

On entend par fonctionnelle la generalisation des fonctions aux domaine des fonction. Par exemple, si f est une fonction de $\mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R}^m)$ l'espace des fonction continue de $\mathbb{R}^n$ dans $\mathbb{R}^m$, alors $G(f, f', f'') = \|f\| + \|f'\| + \|f''\|$ est une fonctionnelle. Dans le cas d'une formulation intégrale (1), comme la fonctionnelle $\mathcal{E}_{\text{tot}}$, les équations d'Euler-Lagrange donnent un critère pour minimiser en fonction de la courbe Γ qui est :

$$\gamma \in \underset{\gamma\in\mathcal{C}^\infty([0,1],\mathbb{R}^2)}{\arg\min}\ \mathcal{E}_{\text{tot}}(\gamma) \quad \text{tel que} \quad \frac{\partial E_{\text{tot}}}{\partial \gamma} - \frac{\mathrm{d}}{\mathrm{d}s}\left(\frac{\partial E_{\text{tot}}}{\partial \gamma'}\right) + \frac{\mathrm{d}^2}{\mathrm{d}s^2}\left(\frac{\partial E_{\text{tot}}}{\partial \gamma''}\right) = 0$$

avec $E_{\text{tot}} = E_{\text{int}} + E_{\text{ext}}$. En traduisant directement l'équation précédente, on trouve l'équation d'équilibre suivante :

$$\forall s \in [0,1], \quad 0 = \lambda_1\gamma''(s) - \lambda_2\gamma^{(4)}(s) + \lambda_3\nabla(\|\nabla I\|^2)$$

Avec cette formulation, on a une condition sur la courbe lorsqu'elle est déjà sur le contour de l'objet, lorsqu'elle est stationnaire. On ne peut pas faire bouger la courbe à partir de cette équation. On introduit alors un autre paramètre, le temps t, pour écrire une équation d'évolution :

$$\forall s \in [0,1], t \in \mathbb{R}^+, \quad \frac{\partial \gamma}{\partial t}(s,t) = \lambda_1\frac{\partial^2\gamma}{\partial s^2}(s,t) - \lambda_2\frac{\partial^4\gamma}{\partial s^4}(s,t) + \lambda_3\nabla(\|\nabla I\|^2)(\gamma(s,t)) \quad (3)$$

On dispose donc de deux équations scalaires pour décrire l'évolution de la courbe, une pour chacune de ses composantes.

2.3 Discrétisation et évolution temporelle

L'équation (3) étant sous sa forme continue, il faut la discrétiser pour la rendre utilisable. Il faut donc faire deux typse de discrétisation, une temporelle et une spatiale. On notera avec un exposant k les itérations temporelles et un indice n les repères spatiaux. On utilise des notations vectorielles pour les coordonnées de la courbe, γ_x^k et γ_y^k. Ces vecteurs regroupent toutes les coordonnées de la courbe discrète à l'itération k :

$$\gamma_x^k = (\gamma_{x,0}^k, \gamma_{x,1}^k \ldots, \gamma_{x,N-1}^k)^t, \quad \text{et} \quad \gamma_y^k = (\gamma_{y,0}^k, \gamma_{y,1}^k \ldots, \gamma_{y,N-1}^k)^t$$

avec N le nombre de point de discrétisation de la courbe. À partir de ces vecteurs, on peut écrire de façon matricielle les opérations de dérivations. On note la dérivée seconde et quatrième

respectivement D_2 et D_4.

$$D_2 = \begin{pmatrix} -2 & 1 & 0 & \dots & 0 & 1 \\ 1 & -2 & 1 & & 0 & 0 \\ 0 & 1 & -2 & \ddots & & \\ & & \ddots & \ddots & & \\ 0 & 0 & & & -2 & 1 \\ 1 & 0 & \dots & & 1 & -2 \end{pmatrix} \quad \text{et} \quad D_4 = \begin{pmatrix} 6 & -4 & 1 & 0 & \dots & 1 & -4 \\ -4 & 6 & -4 & \ddots & & 0 & 1 \\ 1 & -4 & 6 & \ddots & \ddots & & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & & 0 \\ 0 & & \ddots & \ddots & & & 1 \\ 1 & 0 & & & & -6 & -4 \\ -4 & 1 & 0 & \dots & 1 & -4 & 6 \end{pmatrix}$$

La discrétisation temporelle se fait simplement en prenant la différence entre deux instants :

$$\frac{\partial \gamma}{\partial t} \simeq \frac{\gamma^k - \gamma^{k-1}}{\Delta t}$$

Au total, on trouve pour chaque composante les équations :

$$\frac{\gamma_x^k - \gamma_x^{k-1}}{\Delta t} = (\lambda_1 D_2 - \lambda_2 D_4)\gamma_x^k + \lambda_3 \nabla(\|\nabla I\|^2)_x \quad (4)$$

$$\frac{\gamma_y^k - \gamma_y^{k-1}}{\Delta t} = (\lambda_1 D_2 - \lambda_2 D_4)\gamma_y^k + \lambda_3 \nabla(\|\nabla I\|^2)_y \quad (5)$$

avec $\nabla(\|\nabla I\|^2)_x$ et $\nabla(\|\nabla I\|^2)_y$ les composantes selon x et y de $\nabla(\|\nabla I\|^2)$. Le système d'évolution liant les deux instants k et $k-1$ peut s'écrire :

$$(I + \Delta t(\lambda_2 D_4 - \lambda_1 D_2))\gamma_x^k = (\gamma_x^{k-1} + \Delta t \lambda_3 \nabla(\|\nabla I\|^2)_x) \quad (6)$$

$$(I + \Delta t(\lambda_2 D_4 - \lambda_1 D_2))\gamma_y^k = (\gamma_y^{k-1} + \Delta t \lambda_3 \nabla(\|\nabla I\|^2)_y) \quad (7)$$

avec I la matrice identité. On obtient donc un système linéaire pour chaque composante de la courbe.

3 Pratique

3.1 Les outils

Pour ce TP, on vous demande de réalise les calculs en C/C++ et la visualisation avec *octave*.

Nous mettons à votre disposition une classe *C_matrix.h* pour gérer les matrices et faire des opérations simples comme l'addition, la soustraction, la multiplication, ainsi que des opérations plus complexes comme l'inversion de matrice, la convolution, le calcul de gradient...

Exemple 1 Exemple de code pour la création d'une matrice :

```
#include <C_matrix.h>//gestion des operations matricielles
#include <iostream>

int main(int argc, char *argv[])
{
    //creation d'un objet
    C_matrix<double> my_matrix(3,5);
    //initialisation random de la matrice
    my_matrix.randomf();
   //affichage sur la console
    my_matrix.show();
    //sauvegarde de l'objet dans un fichier ``myMatrixFile''
    my_matrix.save("myMatrixFile");

    //affichage de la somme de tout les elements de my_matrix
    std::cout << my_matrix.sum() << std::endl;

    return 1;
}
```

Exemple 2 Pour le chargement et l'affichage d'image, la classe *C_imgMatrix* qui hérite de la classe *C_matrix* utilise la librairie *CImg* pour charger et afficher les images. La librairie *CImg* peut être installée sur votre ordinateur avec la commande

apt-get install cimg-dev

ou, si vous n'avez pas les droits nécessaires, vous pouvez cloner le repository git de *CImg* en tapant la commande suivante :

git clone http ://git.code.sf.net/p/cimg/source CImg

Remarque : *git* est un gestionnaire de version. Les gestionnaires de versions servent à gérer les versions d'un projet et de pouvoir revenir à touts moments à une version précédente pour pouvoir retrouver un code/version qui fonctionne. Je vous recommande fortement d'utiliser des gestionnaires de versions dans tout ce que vous faites. Par exemple, vous commencez un TP, vous créez un repository que vous actualiserez tout au long de la séance de sorte à ce que vous puissiez retrouver une version qui fonctionne ou qui fait autre chose (une autre partie d'un exercice). Pour plus d'information concernant *git*, je vous renvoie vers le lien ou un autre lien.

Exemple de code pour charger une image à l'aide de la classe *C_imgMatrix* :

```
#include <C_imgMatrix.h> //inclu la library de chargement d'image CImg
#include <iostream>

int main(int argc, char *argv[])
{
    if(argc!=2)
    {
        std::cout << "Commande : " << argv[0] << " fileName<string>" << std::endl;
        return -1;
    }
    //charge l'image argv[1] dans la matrice myImg
    C_imgMatrix<double> myImg(argv[1]);
    //montre l'image
    myImg.display(0);
    //calcul le gradient dans la direction X
    C_imgMatrix<double> myGradX = myImg.gradX();
    //affiche l'image du gradient
    myGradX.display(1);
    return 1;
}
```

3.2 Questions

3.2.1 Charger et afficher une image

Tester le chargement d'une image avec la classe *C_imgMatrix* et afficher votre image.

3.2.2 Effectuer des opérations simple

Filtrez une image par un noyau gaussien (Attention ! Le noyau doit être de dimension impaire). Affichez le résultats. Affichez la valeur absolue de la différence entre l'image d'origine et l'image filtrée. Sur une image binarisée, utilisez la méthode *bwdistEuclidean* pour calculer la carte de distance, la visualiser.

3.3 Génération de deux courbes

Pour des soucis de simplicités conceptuelles, nous utiliserons des objets qui encoderont les points et les courbes, respectivement les classes *C_point* et *C_curve*. La classe *C_point* contient des attributs de position x et y, de vitesse vx et vy ainsi que d'énergie image Px et Py. La classe *C_curve* contient un vecteur de point ainsi que deux méthodes pour initialiser la positions des points de la courbe, une en forme de cercle et l'autre en forme de carré. Regardez rapidement de quoi sont faites ces classes avant de commencer. Dans la classe *C_curve*, créez deux méthodes pour extraire les coordonnées x et y de la courbe et les stocker dans deux matrices colonnes. Créez une matrice pour calculer la dérivée de la courbe, calculez la dérivée et sauvegardez les résultats dans des fichiers. Calculez de l'énergie intérieure de la courbe au sens des snakes. Sous octave, chargez les fichiers et les afficher à l'aide des fonctions *load* et *plot*.

3.4 Énergie intérieure du Snake

Dans la classe *C_snakes*, dont un squelette est propose, implémentez deux méthodes qui génèrent les matrices de dérivation $D1$ et dérivation seconde $D2$. Les deux matrices $D1$ et $D2$ sont des attributs de la classe. Créez ensuite une méthode qui réalise le calcul de l'énergie interne de la courbe.

3.5 Carte d'énergie extérieure

Dans la classe *C_snakes*, créez une méthode qui calcule l'énergie image en chaque point de l'image, ainsi que les composantes du gradient de E_{ext} (énergie image). On stockera l'énergie image dans *imgEnergy* et les composantes du gradient dans les attributs PX et PY.

3.6 Évolution temporelle du snake

Créez une méthode qui génère la matrice de dérivée $4^{\text{ème}}$ $D4$. Implémentez une méthode qui génère le système défini à l'équation (6) et (7) et qui stocke l'inversion du système dans la matrice Ainv. Créez une methode qui met à jour les attributs Px et Py de chaque points de la courbe. Implémentez deux méthodes qui retournent les seconds membres des équations (6) et (7). Pour faire évoluer la courbe depuis un point de départ fixe, définissez une méthode qui initialise tout les éléments nécessaire à l'évolution, puis créez une méthode qui encode un pas temporel de l'évolution du snake. Enfin, rassemblez tout les éléments codés auparavant de sorte à ce qu'une seule méthode soit appelée dans le *main*.

Testez votre code sur les images qui vous sont fournies et déterminez les paramètres λ_1, λ_2, λ_3 et dt pour ces images. Visualisez les effets de chaque paramètre pour les différentes images. Montrez l'évolution du snake et de l'énergie totale en fonction du temps.

sur les différentes images, faire évoluer le snake et trouver des bons paramètres pour trouver les contours désirés.

3.7 Pour aller plus loin

Si vous avez le temps pendant la séance de TP, vous pouvez explorer les pistes suivantes :

1. Changez la forme de l'énergie intérieure
2. Appliquez une force de pression sur le snake en ajoutant une force extérieure selon la normale à la courbe. Vous pouvez vous referez à l'article On active contour models and ballons
3. Pour arriver à detecter les contours dans une forme concave, vous pouvez utiliser une technique qui consiste à faire évoluer le snake dans un champ de vecteur. Vous pouvez vous référer à la publication Snakes, shapes, and gradient vector flow