Accélération de surfaces actives

Julien OLIVIER Julien MILLE Romuald BONÉ Jean-Jacques ROUSSELLE

Laboratoire Informatique

Université François-Rabelais de Tours
64, avenue Jean Portalis, 37200 TOURS

{julien.olivier, julien.mille, romuald.bone, rousselle}@univ-tours.fr

Concours Jeune Chercheur : Oui

Résumé

Les contours actifs et surface actives sont des modèles déformables utilisés respectivement pour la segmentation d'images 2D et 3D. Dans cet article, nous présentons deux méthodes développées afin d'améliorer la vitesse de ces processus de segmentation d'image. Elles reposent sur des adaptations à la 3D de méthodes développées en 2D pour les contours actifs. Nous les appliquons ici sur un modèle de surface 3D discrète (maillage) dont l'évolution est guidée par l'algorithme greedy, bien qu'elles puissent s'utiliser avec d'autres types d'implémentation des contours actifs, tels les courbes de niveaux.

Mots clefs

Segmentation d'images, contours actifs, surfaces actives, accélération.

1 Introduction

Les contours actifs ou *snakes* on été à l'origine développés par Kass *et al* dans [1]. Ce sont des outils de segmentation puissants, notamment grâce à leur robustesse au bruit.
De nombreux algorithmes ont été développés pour les contours actifs. L'algorithme *greedy* introduit dans [2] demeure l'un des plus populaire de par son efficacité et sa facilité d'implémentation. Bulpitt et Efford propose dans [3] une adaptation de l'algorithme *greedy* aux surfaces 3D. De nombreuses méthodes d'accélération des contours actifs 2D ont été développées dans [4]. Nous présentons dans cette article une adaptation des ces méthodes pour accélérer les surfaces actives 3D.
Dans la section 2 nous décrirons tout d'abord le modèle des surfaces active et ses énergies. Nous verrons aussi l'algorithme d'évolution pour les surfaces actives et le principe du remaillage. Les sections 3 et 4 décrivent les deux méthodes d'accélération que nous avons adapté aux surfaces actives : la méthode du voisinage décalé et la méthode du *line search*. La section 5 décrit nos résultats expérimentaux sur des modèles 3D et la section 6 conclut sur notre travail en envisageant les développements futurs.

2 Le modèle de surface active

Dans un domaine continu, un modèle déformable 3D est représenté par une surface paramétrée S qui, à un couple de paramètres (u, v), associe un point $(x, y, z)^T$:

$$\begin{aligned} S \quad : \quad & \Omega^2 \rightarrow \mathbb{R}^3 \\ & (u, v) \longmapsto (x(u, v), y(u, v), z(u, v))^T \end{aligned} \qquad (1)$$

Afin d'implémenter la surface active, nous utilisons la représentation explicite discrète décrite dans [5]. Cette dernière est un maillage triangulaire composé de n sommets connectés, dont les arêtes forment un ensemble de triangles adjacents. Afin de représenter la notion de connectivité (la topologie locale), nous considérons que chaque sommet p_i possède un ensemble de sommets adjacents notés A_i. Le maillage est construit à partir de plusieurs divisions successives d'un icosaèdre [6, 7], menant ainsi à une surface de géométrie sphérique avec une distribution homogène des sommets. La surface possède une fonctionnelle d'énergie discrète obtenue en sommant les énergies de chaque sommet (nous décrirons les différentes énergies des sommets plus loin). La surface va évoluer de manière à minimiser cette fonctionnelle, attirant les sommets vers les bordures de l'objet à segmenter tout en conservant une stabilité géométrique. Initialement développé pour les contours 2D par Williams et Shah [2], l'algorihme *greedy* est une méthode de minimisation d'énergie proposée à l'origine comme une alternative rapide à la méthode variationnelle [1] et à la programmation dynamique [8]. Il a récemment été utilisé pour la segmentation et le suivi 2D dans [9, 10]. Dans [3], cet algorithme est étendu à la 3D pour l'évolution de maillages dans des images volumétriques.

La minimisation globale de la fonctionnelle d'énergie est réalisée par plusieurs minimisations locales successives. En effet, à chaque itération de l'algorithme un voisinage cubique de taile w autour de chaque sommet est étudié (voir fig. 1). Pour chaque voxel du voisinage, l'algorithme calcule la fonctionnelle d'énergie correspondante et affecte alors le sommet de la surface active au voxel ayant

l'énergie minimale. Dans l'algorithme *greedy* classique le fenêtre de voisinage est centrée autour de chaque sommet. Dans les méthodes que nous présentons plus loin nous autorisons cette fenêtre à ne plus être centrée.

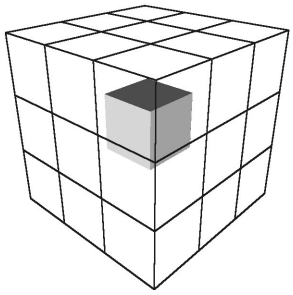

Figure 1 – *Sommet centré dans sa fenêtre de voisinage cubique* $3 \times 3 \times 3$

Afin de définir la position de chaque sommet dans son voisinage nous définissons un vecteur de décalage $\vec{\mathbf{s}}_i^{(t)} = (s_x, s_y, s_z)^T$, représentant les coordonnées du i^{ieme} sommet de la surface active à l'itération t par rapport à un voxel origine choisi dans la fenêtre de voisinage.
Au lancement de l'algorithme toutes les fenêtres de voisinage sont centrées autour de leurs sommets respectifs. Ainsi nous avons $\vec{\mathbf{s}}_i = (w/2, w/2, w/2)^T$.
Nous pouvons désormais définir le voisinage du i^{eme} sommet de la surface active à l'itération t :

$$\mathcal{N}_i^{(t)} = \left\{ \mathbf{p}_i^{(t)} + \mathbf{r} - \vec{\mathbf{s}}_i^{(t)} | \mathbf{r} \in [0, w-1]^3 \right\} \qquad (2)$$

Soit p_i la position initiale dans la fenêtre de voisinage $\mathcal{N}_i^{(t)}$, nous définissons p'_i comme la position du voxel testé par l'algorithme. Une fois que tous les voxels de $\mathcal{N}_i^{(t)}$ ont été explorés l'algorithme choisit la nouvelle position du sommet p_i :

$$\mathbf{p}_i^{(t+1)} = \arg \min_{\mathbf{p}'_k \in \mathcal{N}_i} E(\mathbf{p}'_k) \qquad (3)$$

L'énergie d'un sommet au voxel p'_i est la somme pondérée de différentes énergies internes et externes. Ces dernières sont normalisées sur le voisinage entier.

$$\begin{aligned} E(\mathbf{p}'_i) &= \alpha E_{cont}(\mathbf{p}'_i) + \beta E_{curv}(\mathbf{p}'_i) \\ &+ \gamma E_{grad}(\mathbf{p}'_i) + \delta E_{bal}(\mathbf{p}'_i) \end{aligned} \qquad (4)$$

Les paramètres (α, ..., δ) représentent les différents poids permettant de contrôler l'influence de chaque énergie. La continuité E_{cont} et la courbure E_{courb} sont les énergies internes qui permettent de garantir la stabilité géométrique de la surface. Ces deux énergies impliquent une prise en compte de la distance euclidienne et de l'ensemble de sommets adjacents. Dans un soucis de rapidité lors de l'execution, nous choisirons dans les équations suivantes de calculer la distance au carré plutôt que la distance elle même. Nous distinguons également $\|.\|$ la norme d'un vecteur de $|.|$ la valeur absolue d'un scalaire. Ainsi $\|\mathbf{p}_i - \mathbf{p}_j\|$ représente la distance euclidienne entre les sommets $\mathbf{p}_i$ et $\mathbf{p}_j$.

Nous allons maintenant décrire l'adaptation des différentes énergies à notre modèle 3D. Nos énergies sont des extensions intuitives de celles du modèle 2D, adaptées à la gestion d'un maillage.
L'énergie de continuité E_{cont} est une énergie interne permettant aux différents sommets d'être espacés de façon régulière sur la surface active. En minimisant cette énergie, l'écart entre la distance moyenne au carré $\overline{d^2}$ et la distance entre le sommet considéré p'_i et ses sommets adjacents est réduite.

$$\begin{aligned} E_{cont}(\mathbf{p}'_i) &= \sum_{j \in A_i} \left| \overline{d^2} - \|\mathbf{p}'_i - \mathbf{p}_j\|^2 \right| \\ \overline{d^2} &= \frac{1}{n} \sum_{i=1}^{n} \frac{1}{card(A_i)} \sum_{j \in A_i} \|\mathbf{p}_i - \mathbf{p}_j\|^2 \end{aligned} \qquad (5)$$

La deuxième énergie interne utilisée est l'énergie de courbure E_{courb}. Minimiser cette dernière revient à appliquer un effet lissant sur la surface active en permettant au sommet considéré de se rapprocher du centre de gravité des sommets adjacents.

$$E_{curv}(\mathbf{p}'_i) = \left\| \mathbf{p}'_i - \frac{1}{card(A_i)} \sum_{j \in A_i} \mathbf{p}_j \right\|^2 \qquad (6)$$

Afin d'attirer la surface active vers les bords de l'objet à segmenter, nous utilisant l'énergie de gradient E_{grad}. Cette énergie externe est basée sur l'amplitude de la norme du gradient g de l'image I en chaque voxel. En cas d'image bruitée, un filtre gaussien est appliqué en amont sur l'image. Dans les équations suivantes, G_σ est un noyau gaussien d'écart type σ et $*$ est l'opérateur de convolution.

$$\begin{aligned} g(\mathbf{p}) &= \|\nabla I(\mathbf{p}) * G_\sigma\| / g_{max} \\ E_{grad}(\mathbf{p}'_i) &= -g(\mathbf{p}'_i) \end{aligned} \qquad (7)$$

Pour le calcul de la norme du gradient, nous effectuons une détection de contours 3D, en convoluant l'image avec l'opérateur de Zucker-Hummel [11], composé de trois masques de taille $3 \times 3 \times 3$. Par exemple, le masque suivant filtre l'image selon l'axe x.

$$Z_x = \begin{bmatrix} -k_1 & 0 & k_1 \\ -k_2 & 0 & k_2 \\ -k_1 & 0 & k_1 \end{bmatrix} \begin{bmatrix} -k_2 & 0 & k_2 \\ -k_3 & 0 & k_3 \\ -k_2 & 0 & k_2 \end{bmatrix} \begin{bmatrix} -k_1 & 0 & k_1 \\ -k_2 & 0 & k_2 \\ -k_1 & 0 & k_1 \end{bmatrix} \qquad (8)$$

$$k_1 = \frac{\sqrt{3}}{3} \; ; \; k_2 = \frac{\sqrt{2}}{2} \; ; \; k_3 = 1 \qquad (9)$$

Afin d'augmenter le rayon d'action de la surface active, nous utilisons une énergie ballon E_{grad} basée sur la force d'inflation présentée par Cohen dans [12].

$$E_{bal}(\mathbf{p}'_i) = \|\mathbf{p}'_i - (\mathbf{p}_i + k\vec{\mathbf{n}}_i)\|^2 \qquad (10)$$

où $\vec{\mathbf{n}}_i$ est le vecteur normal à la surface, défini au sommet p_i. Nous calculons la normale du sommet $\mathbf{p}_i$ comme la moyenne des normales des triangles voisins de $\mathbf{p}_i$. Par abus de langage, la normale d'un triangle désigne le vecteur unitaire orthogonal au plan auquel appartient le triangle. Dans les expressions suivantes, T_i désigne l'ensemble des triangles voisins de $\mathbf{p}_i$.

$$\vec{\mathbf{n}}_i = \frac{\sum_{t \in T_i} \vec{\mathbf{n}}_t}{\left\| \sum_{t \in T_i} \vec{\mathbf{n}}_t \right\|} \tag{11}$$

La normale d'un plan est déterminée par le produit vectoriel normalisé de deux vecteurs de ce plan. Ainsi, $\vec{\mathbf{n}}_t$ est calculé comme suit :

$$\vec{\mathbf{n}}_t = s_t \frac{(\mathbf{p}_{t_2} - \mathbf{p}_{t_1}) \wedge (\mathbf{p}_{t_3} - \mathbf{p}_{t_1})}{\|(\mathbf{p}_{t_2} - \mathbf{p}_{t_1}) \wedge (\mathbf{p}_{t_3} - \mathbf{p}_{t_1})\|} \tag{12}$$

où $\mathbf{p}_{t_j}, j = 1...3$ sont les sommets du triangle t ($\mathbf{p}_i$ est obligatoirement l'un d'eux). $s_t = \pm 1$ est le signe qui change l'orientation de $\vec{\mathbf{n}}_t$, pour assurer que le vecteur pointe vers l'intérieur de la surface. Orienter les normales de cette façon est nécessaire pour une implémentation correcte du ballon.

Le mouvement induit par la minimisation de l'énergie ballon est soit une dilatation soit une rétractation de la surface, selon le signe du coefficient δ. Celui-ci doit être choisi en fonction de la position initiale de la surface par rapport à l'objet.

Afin de permettre à la surface de s'adapter localement à la géométrie de l'objet à segmenter, un remaillage est effectué après chaque itération de l'algorithme *greedy*. Pour conserver la distance entre les sommets adjacents homogène, le maillage peut ajouter ou enlever des sommets, garantissant une distribution stable des sommets [7, 13, 14]. Ainsi chaque couple de sommets adjacents vérifie la contrainte :

$$d_{min} \leq \|\mathbf{p}_i - \mathbf{p}_j\| \leq d_{max} \tag{13}$$

où d_{min} et d_{max} sont deux seuils définis par l'utilisateur tels que $d_{max} \geq 2d_{min}$. Ajouter ou supprimer des sommets modifie localement la topologie de la surface, des contraintes topologiques doivent donc être vérifiées. Pour pouvoir ajouter ou supprimer des sommets, $\mathbf{p_i}$ et $\mathbf{p_j}$ doivent posséder deux sommets adjacents en commun : $|A_i \cap A_j| = 2$.
Si $\|\mathbf{p}_i - \mathbf{p}_j\| > d_{max}$ alors l'algorithme crée un nouveau sommet au milieu du segment $\mathbf{p}_i\mathbf{p}_j$ et le connecte à $\mathbf{p}_a$ et $\mathbf{p}_b$ (voir figure 2.b).
Si $\|\mathbf{p}_i - \mathbf{p}_j\| < d_{min}$, le sommet $\mathbf{p}_j$ est supprimé et $\mathbf{p}_i$ est alors déplacé au milieu du segment que formaient $\mathbf{p}_i$ et $\mathbf{p}_j$ (voir figure 2.c).

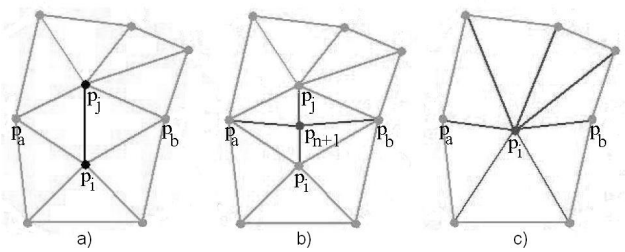

Figure 2 – *Remaillage sur la surface active : a) Maillage initial b) Ajout d'un sommet c) Suppression d'un sommet*

Maintenant que nous avons étudié l'algorithme *greedy* pour les surfaces actives, nous allons présenter les deux méthodes initialement développées en deux dimensions que nous avons adaptées aux surfaces actives 3D.

3 La méthode du voisinage décalé

La méthode du voisinage décalé a été initialement développée pour les contours actifs 2D dans [4]. Dans cette section nous allons décrire comment nous avons adapté cette méthode aux surfaces actives 3D.
A chaque itération nous allons agir sur le voisinage de chaque sommet de manière à orienter son espace de recherche dans la direction paraissant la plus intéressante. Afin de définir quelles sont ces directions, nous allons utiliser l'information déduite de l'itération précédente de l'algorithme *greedy*. En effet, grâce aux itérations précedentes nous sommes en mesure de connaître pour chaque sommet la direction qui a été suivie. Ainsi, nous allons décaler le voisinage de chaque sommet de la surface active d'un voxel dans la direction suivie précédemment. A chaque itération et pour tous les sommets, le prochain décalage à réaliser sera obtenu par :

$$\vec{\mathbf{d}}_i^{\,(t+1)} = \mathcal{B}(-1, 1, \mathbf{p}_i^{(t+1)} - \mathbf{p}_i^{(t)}) \tag{14}$$

$\vec{\mathbf{d}}_i^{(t)}$ représente le déplacement appliqué au i^{eme} sommet à l'itération t. $\mathcal{B}$ est une fonction de limitation de décalage, permettant de borner le décalage entre deux entiers :

$$\mathcal{B}(b_1, b_2, \vec{\mathbf{u}}) = \begin{pmatrix} \max(b_1, \min(b_2, u_x)) \\ \max(b_1, \min(b_2, u_y)) \\ \max(b_1, \min(b_2, u_z)) \end{pmatrix} \tag{15}$$

Connaissant le déplacement $\vec{\mathbf{d}}_i^{(t+1)}$ grâce à l'équation (14), nous pouvons définir le prochain décalage $\vec{\mathbf{s}}_i^{(t+1)}$ à appliquer à chaque sommet de la surface active :

$$\vec{\mathbf{s}}_i^{(t+1)} = \mathcal{B}(1, w-2, \vec{\mathbf{s}}_i^{(t)} - \vec{\mathbf{d}}_i^{(t+1)}) \tag{16}$$

Nous pouvons maintenant définir l'algorithme pour la méthode du voisinage décalé, qui consiste à calculer à la fin de chaque itération et pour chaque sommet la nouvelle fenêtre de voisinage avec les équations (14) (16) et (2), une fois que tous les sommets de la surface active ont été déplacés. En incluant la méthode du voisinage décalé dans l'algorithme *greedy* nous obtenons :

Algorithme 1 Méthode du voisinage décalé : modèle 3D

Pour $t \leftarrow 0$ à $T-1$ **faire**
  Pour $i \leftarrow 0$ à $n-1$ **faire**
    $\mathbf{p}_i^{(t+1)} = \underset{\mathbf{p'}_k \in N_i}{\arg\min}\, E(\mathbf{p'}_k)$
    $\vec{\mathbf{d}}_i^{(t+1)} = \mathcal{B}(-1, 1, \mathbf{p}_i^{(t+1)} - \mathbf{p}_i^{(t)})$
    $\vec{\mathbf{s}}_i^{(t+1)} = \mathcal{B}(1, w-2, \vec{\mathbf{s}}_i^{(t)} - \vec{\mathbf{d}}_i^{(t+1)})$
    $\mathcal{N}_i^{(t+1)} = \left\{ \mathbf{p}_i^{(t+1)} + \mathbf{r} - \vec{\mathbf{s}}_i^{(t)} | \mathbf{r} \in [0, w-1]^3 \right\}$
  Fin Pour
Fin Pour

4 La méthode du *line search*

La méthode du *line search* a elle aussi été développée dans [4] pour les contours actifs 2D. Nous proposons ici son adaptation aux surfaces actives. Son principe est similaire à la méthode du voisinage décalé : il s'agit d'anticiper la prochaine itération de l'algorithme *greedy* en utilisant les informations obtenues lors des précédentes itérations. La différence majeure est que nous n'allons plus modifier le voisinage de chaque sommet, mais explorer une ligne de voxels dans la direction suivie précédemment.

Cette méthode est lancée à la fin de chaque itération de l'algorithme *greedy*, une fois que tous les sommets ont été déplacés. La direction suivie par chaque sommet $\mathbf{p}_i$ est conservée et un nombre fixé de voxels est alors observée vers celle ci. L'énergie globale de chaque voxel de la ligne d'exploration est alors calculée avec l'équation (4). Le sommet courant $\mathbf{p}_i$ est alors affecté au voxel d'énergie minimale, si elle est inférieure à la sienne.

L'algorithme 2 représente l'intégration de la méthode du *line search* dans l'algorithme *greedy*. Définissons $\mathcal{T}$ comme le nombre d'itérations de l'algorithme *greedy* et l comme le nombre de voxels de la ligne d'exploration du *line search*.

Algorithme 2 Méthode du *line search* : modèle 3D

Pour $t \leftarrow 0$ à $T-1$ **faire**
  Pour $i \leftarrow 0$ à $n-1$ **faire**
    $\mathbf{p}_i^{(t+1)} = \underset{\mathbf{p'}_k \in N_i}{\arg\min}\, E(\mathbf{p'}_k)$
    Déterminer la direction $\vec{\mathbf{v}} = \frac{\mathbf{p}_i^{(t+1)} - \mathbf{p}_i^{(t)}}{\left\| \mathbf{p}_i^{(t+1)} - \mathbf{p}_i^{(t)} \right\|}$
    Line Search $d = \underset{s \in [0,l]}{\operatorname{argmin}} \left\{ E(\mathbf{p}_i^{(t)} + s\vec{\mathbf{v}}) \right\}$
    Mise à jour : $\mathbf{p}_i^{(t)} \leftarrow \mathbf{p}_i^{(t)} + dv$
  Fin Pour
Fin Pour

5 Résultats expérimentaux

Nous avons réalisé une série de tests visant à comparer l'algorithme *greedy* aux méthodes du voisinage décalé et du *line search*. Chaque image de test est composée de plusieurs coupes d'un objet en niveaux de gris. Nous avons volontairement ajouté un bruit gaussien de moyenne nulle. Plusieurs largeurs de voisinage ont été testées sur chaque image. La méthode du voisinage décalé n'a pas été testée avec un largeur $w = 3$ afin de respecter les contraintes de la méthode (le voisinage n'aurait jamais été décalé). La première image représente une spirale. Elle a été choisie afin de tester la validité de nos deux méthodes lorsque la surface évolue en dilatation avec création de sommets (le remaillage est activé). La surface active a été initialisée à l'intérieur du modèle 3D avec seulement 12 sommets. Pour les 3 méthodes le maillage final contient environ un millier de sommets. Les paramètres choisis sont $\alpha = 0, \beta = 0.5, \gamma = 2$ et $\delta \in [-1.1, -0.6]$. La deuxième image est le modèle 3D d'un vase. Celle-ci présente l'intérêt de pouvoir tester l'infiltration de la surface active dans les concavités, ce qui a toujours été un des principaux inconvénients de l'algorithme *greedy*. Les paramètres sont $\alpha = 0$, $\beta = 0.5$ pour l'algorithme *greedy*, 0.4 pour la méthode du voisinage décalé et 0.3 pour le *line search*, $\gamma = 2$ et $\delta = 0.8$. Le remaillage n'a pas été autorisé et la surface active a été initialisée avec 2562 sommets. La troisième image représente trois ellipsoïdes imbriquées et permet de tester aussi bien la reconstruction d'angles saillants que d'angles faibles. Les paramètres étaient $\alpha = 0.5$, $\beta = 0.3$ pour l'algorithme *greedy* et 0.4 pour les méthodes du voisinage décalé et du *line search*, $\gamma = 2$ et $\delta \in [0.3...0.7]$.

Chaque exécution a été réalisée sur un Intel Pentium IV 2.8Ghz avec 512Mo RAM. La figure 4 donne les résultats en termes de temps d'exécution. La figure 3 représente dans sa partie supérieure une coupe en niveaux de gris de chaque image à segmenter et dans sa partie inférieure leurs reconstructions 3D respectives obtenues avec la méthode du *line search*.

Nos deux méthodes nous ont permis de réduire de manière significative le nombre d'itérations nécessaires à l'algorithme pour segmenter les frontières de l'objet. Pour les contours actifs 2D la meilleure méthode d'accélération est la méthode du voisinage décalé. Nos résultats nous permettent d'estimer qu'en trois dimensions la méthode du *line search* est la plus performante.

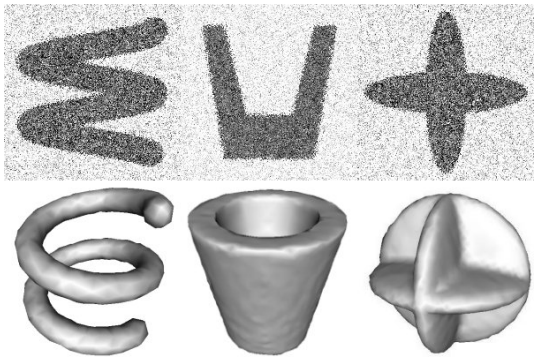

Figure 3 – *Résultats obtenus avec la méthode du line search.*

Image	Voisinage	Méthode	Itérations	Temps d'exécution (s)
Spirale	3	Greedy	400	0.3
		LS	65	0.16
	5	Greedy	195	0.52
		LS	63	0.3
		VD	138	0.31
	7	Greedy	145	0.63
		LS	60	0.57
		VD	97	1.11
3-Elipsoides	3	Greedy	47	0.63
		LS	19	0.41
	5	Greedy	25	1.05
		LS	12	0.69
		VD	16	0.98
	7	Greedy	18	2.24
		LS	12	1.86
		VD	14	2.07
Vase	3	Greedy	155	0.98
		LS	68	0.72
	5	Greedy	120	1.27
		LS	50	0.85
		VD	92	1.08
	7	Greedy	109	1.98
		LS	43	1.13
		VD	60	2.2

Figure 4 – *Comparaison de trois méthodes : algorithme greedy (greedy), voisinage décalé (VD) et line search (LS)*

En effet, le passage en trois dimensions favorise la méthode du *line search* car au lieu d'explorer un nombre fixe de pixels, on explore le même nombre de voxels donc l'augmentation du temps de calcul apportée par la 3D n'est pas significative. En ce qui concerne la méthode du voisinage décalé, les calcul de décalage ne se font plus sur une fenêtre carrée mais cubique. L'augmentation du temps de calcul par rapport à la 2D est donc beaucoup plus importante.

6 Conclusion et perspectives

Dans cet article nous avons présenté deux méthodes d'accéleration des surfaces actives qui sont sont des adaptations à la 3D de méthodes 2D existantes. En termes de temps d'exécution la méthode du *line search* apparaît comme étant la plus performante. Nous avons également testé la troisième méthode décrite dans [4] (le voisinage déformé) mais ses performances en 3D sont réduites pour les mêmes raisons que la méthode du voisinage décalé, nous n'avons donc pas jugé utile de l'inclure à cette étude.

Nous étudions actuellement le possibilité de réaliser un modèle hybride de surface active s'appuyant à la fois sur une approche physique et sur l'algorithme *greedy*. Nous pourrons alors comparer ce modèle aux algorithmes d'évolution accélérés que nous venons de décrire.

Références

[1] M. Kass, A. Witkin, et D. Terzopoulos. Snakes : active contour models. *International Journal of Computer Vision*, 1(4) :321–331, 1987.

[2] D.J. Williams et M. Shah. A fast algorithm for active contours and curvature estimation. *Computer Vision and Image Processing : Image Understanding*, 55(1) :14–26, January 1992.

[3] A.J. Bulpitt et N.D. Efford. An efficient 3D deformable model with a self-optimising mesh. *Image and Vision Computing*, 14(8) :573–580, 1996.

[4] J. Olivier, R. Boné, et J.J. Rousselle. Comparison of active contour acceleration methods. Dans 5^{th} *Conf. on Visualization, Imaging & Image Processing (VIIP)*, pages 518–523, Benidorm, Spain, 2005.

[5] J. Mille, R. Boné, P. Makris, et H. Cardot. 3D segmentation using active surface : a survey and a new model. Dans 5^{th} *Conf. on Visualization, Imaging & Image Processing (VIIP)*, pages 610–615, Benidorm, Spain, 2005.

[6] T. McInerney et D. Terzopoulos. A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4D image analysis. *Computerized Medical Imaging and Graphics*, 19(1) :69–83, 1995.

[7] J-Y. Park, T. McInerney, et D. Terzopoulos. A non-self-intersecting adaptive deformable surface for complex boundary extraction from volumetric images. *Computer & Graphics*, 25(3) :421–440, June 2001.

[8] A.A. Amini, T.E. Weymouth, et R.J. Rain. Using dynamic programming for solving variational problem in vision. *IEEE Trans. PAMI*, 12(9) :855–867, September 1991.

[9] L. Ji et H. Yan. Attractable snakes based on the greedy algorithm for contour extraction. *Pattern Recognition*, 35(4) :791–806, April 2002.

[10] C.L. Lam et S.Y. Yuen. An unbiased active contour algorithm for object tracking. *Pattern Recognition Letters*, 19(5-6) :491–498, April 1998.

[11] S.W. Zucker et R.A. Hummel. A three-dimensional edge operator. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(3) :324–331, 1981.

[12] L.D. Cohen. On active contour models and balloons. *Computer Vision, Graphics, and Image Processing : Image Understanding*, 53(2) :211–218, 1991.

[13] J.O. Lachaud et A. Montanvert. Deformable meshes with automated topology changes for coarse-to-fine three-dimensional surface extraction. *Medical Image Analysis*, 3(2) :187–207, 1998.

[14] G. Slabaugh et G. Unal. Active polyhedron : surface evolution theroy applied to deformable meshes. Dans *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 84–91, San Diego, 2005.