Dynamic neighborhoods in active surfaces for 3D segmentation

Julien Olivier, Julien Mille, Romuald Boné, Jean-Jacques Rousselle

Université François Rabelais de Tours
Laboratoire Informatique
64 Avenue Jean Portalis, 37200 Tours, France
{julien.olivier, julien.mille, romuald.bone, rousselle}@univ-tours.fr

Abstract. Active contours and surfaces are deformable models used for 2D and 3D image segmentation. In this paper, we propose two methods developed in order to accelerate 3D image segmentation process. They are adaptations on active surfaces of two methods developed for 2D active contour. We use them on a discrete 3D surface model (mesh) evolving with the greedy algorithm. Those methods will be compared to the classical greedy algorithm and to a recent fast adaptation of the level set method.

Keywords: Active surface, mesh, greedy algorithm, fast level set

1 Introduction

Active contours or snakes, initially developed by Kass *et al* in [1], are powerful segmentation tools thanks to their noise robustness and ability to generate linked closed boundaries. Their 3D extensions, active surfaces, were developed according to several implementations (see [2] [3] for surveys on 3D deformable models). Among these implementations, meshes are explicit discrete representations [4], which represents the surface as a set of interconnected vertices. The model is deformed by direct modifications of vertices coordinates. Several evolution algorithms have been developed to deform them. One of the most popular is the greedy algorithm [5] because of its efficiency. An adaptation of the greedy algorithm on 3D surface was proposed by Bulpitt and Efford in [6].

Conversely, implicit implementations, based on the level set framework [7], handle the surface as the zero level of a hypersurface, defined on the same domain as the image (for 3D images, the hypersurface is a $\mathbb{R}^3 \longrightarrow \mathbb{R}$ application). Level sets are often chosen for their natural handling of topological changes and adaptiveness to any dimension. Their algorithmic complexity is proportional to the image resolution, making them time-consuming. Despite the development of accelerating methods (like the narrow band technic [7], the fast marching method [8] and the recent fast level set [9]), their computational cost remains high, preventing their use in time-critical applications. Moreover, mesh surfaces have several advantages over their implicit counterparts. Their representation is more intuitive, and thus allow easier modeling of *a priori* knowledge and user interaction. The main drawback is that meshes do not modify their topology naturally (technics for detection of topological changes must be implemented beside the evolution algorithm).

In many applications, the topology of the area of interest is known in advance. When segmenting images in which prior knowledge about the object topology is available, we believe that mesh-based approaches should be privileged over implicit surfaces. In this paper, we deal with a 3D triangular mesh driven by greedy algorithm. The model is able to perform remeshing, thus providing geometrical versatility (in the same manner that 2D reparameterization technics overcome the lack of geometrical flexibility of traditional snakes). Several methods have been developed in [10] in order to accelerate 2D active contours. In this paper, we study 3D adaptations of these methods in order to accelerate discrete active surfaces, as an extension of the work in [11].

The outline of this paper is as follows: section 2 presents the 3D model and its energies, the greedy algorithm for active surfaces and the remeshing principle. Section 3 and 4 describe the shifted neighborhood

method and the line search method. Section 5 describes the Fast Level Set implementation for active contours. Section 6 shows our experimental results on 3D models, comparing the performances of our acceleration methods with the basic greedy algorithm and the recent Fast Level Set implementation method. Section 7 concludes with our work expecting the future developments.

2 The 3D active surface model

In a continuous domain, a 3D deformable model is represented by a parameterized surface $\mathbf{S}$ mapping a couple of parameters (u, v) to a space point $(x, y, z)^T$:

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

The parameter domain is normalized : $\Omega = [0, 1]$. The surface is attached to the image I, which is a $\mathbb{R}^3 \rightarrow \mathbb{R}$ function, mapping each voxel (x, y, z) to a gray level $I(x, y, z)$. Basing ourselves on the work by [12], the surface is endowed with the energy functional $E(\mathbf{S})$.

$$E(\mathbf{S}) = \int_\Omega \int_\Omega \omega_{10} \left\| \frac{\partial \mathbf{S}}{\partial u} \right\|^2 + \omega_{01} \left\| \frac{\partial \mathbf{S}}{\partial v} \right\|^2 + 2\omega_{11} \left\| \frac{\partial^2 \mathbf{S}}{\partial u \partial v} \right\|^2 + \omega_{20} \left\| \frac{\partial^2 \mathbf{S}}{\partial u^2} \right\|^2 + \omega_{02} \left\| \frac{\partial^2 \mathbf{S}}{\partial v^2} \right\|^2 + P(\mathbf{S}) du dv \tag{2}$$

Segmentation of a region of interest in image I is performed by determining the optimal surface $\mathbf{S}^*$ minimizing E. The energy functional depends on two kinds of energies. The first one quantifies the geometrical regularity of the surface, whereas the second one is the external term, depending on the distance between the surface and the salient edges of the image. First and second order partial derivatives of $\mathbf{S}$ with respect to u and v are smoothing terms. Coefficients ω_{ij} weight the significance of regularizing components with respect to the external term. ω_{10} and ω_{01} are the elasticity coefficients, ω_{20} and ω_{02} are the rigidity coefficients and ω_{11} is the resistance to twist. Since our segmentation is boundary-based, P is the external term attracting surface points towards salient boundaries on the image. It must decrease as the edge magnitude increases, hence we choose $P = -\|\nabla I\|$. It should be noted that other attracting functions could be chosen, e.g. a distance map with respect to the thresholded edge image.

To implement the active surface, we use the discrete representation described in [13], which is a triangular mesh made up of n vertices, denoted $\mathbf{p}_i = (x_i, y_i, z_i)^T \in \mathbb{R}^3$, and edges connecting the vertices (making a set of adjacent triangles). In order to represent the connectivity notion, each vertex $\mathbf{p}_i$ has a set of adjacent vertices, denoted A_i. The mesh is built from successive subdivisions of an icosahedron [14][15], thus leading to a sphere-like surface with a homogeneous vertex distribution (see figure 1).

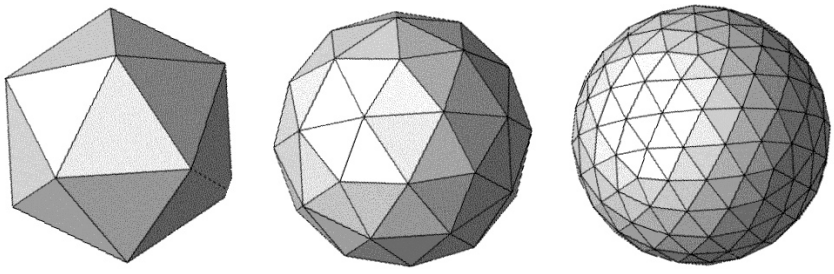

Fig. 1. The basic icosahedron and its first two tesselations

Initially developped for 2D active contours by Williams and Shah in [5], the greedy algorithm is an energy minimizing method first proposed as an alternative to the variationnal method [1] and the dynamic programming [16]. It has been recently used for 2D segmentation in [17] and [18]. In [6] one may find an extension of this algorithm for 3D meshes. Global energy minimization is performed via successive local optimizations. Each vertex is endowed with its own energy. Hence, unlike in eq. 2, the energy functional is the sum of vertex energies.

$$E(\mathbf{S}) = \sum_{i=1}^{n} E(\mathbf{p}_i) \tag{3}$$

At each iteration, a cubic neighborhood of side length w around each vertex is considered (see fig. 2). The energy is computed at each voxel belonging to the neighborhood and the vertex is moved to the location leading to the lowest energy. This approach avoids to perform gradient descent of a partial derivatives equation, derived from the energy functional using classical Euler-Lagrange scheme [14]. Hence, it is not necessary to derive analytically the energies, with respect to vertex coordinates.

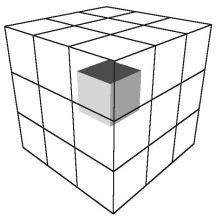

Fig. 2. Cubic centered neighborhood

Unlike in the classical greedy algorithm, our methods will deal with neighborhoods which are not necessarily centered around the vertices. We define $\mathcal{N}_i^{(t)}$, the neighborhood of the i^{th} vertex at iteration t:

$$\mathcal{N}_i^{(t)} = \left\{ \mathbf{p}_i^{(t)} + \mathbf{r} - \vec{\mathbf{s}}_i^{(t)} \mid \mathbf{r} \in [0, w-1]^3 \right\} \tag{4}$$

$\vec{\mathbf{s}}_i^{(t)} = (s_x, s_y, s_z)^T$ is the shift vector, representing the coordinates of the i^{th} vertex at iteration t relatively to an original voxel chosen on the corner of the neighborhood. At the beginning, all vertices are centered in their neighborhood, hence we have $\vec{\mathbf{s}}_i^{(0)} = (w/2, w/2, w/2)^T$.
The initial position being $\mathbf{p}_i$, we denote $\mathbf{p}'_i$ a tested location in the neighborhood. Once all energies have been computed the new location of vertex $\mathbf{p}_i$ is chosen:

$$\mathbf{p}_i^{(t+1)} = \arg \min_{\mathbf{p}'_k \in \mathcal{N}_i^{(t)}} E(\mathbf{p}'_k) \tag{5}$$

The energy of a vertex at location $\mathbf{p}'_i$ is a weighted sum of discrete internal and external energies, normalized on the whole neighborhood.

$$E(\mathbf{p}'_i) = \alpha E_{cont}(\mathbf{p}'_i) + \beta E_{curv}(\mathbf{p}'_i) + \gamma E_{grad}(\mathbf{p}'_i) + \delta E_{bal}(\mathbf{p}'_i) \tag{6}$$

The coefficients α, β, γ and δ are the weights defining the relative influence of the energies. The continuity E_{cont} and the curvature E_{curv} are discrete implementations of first and second order surface derivatives of eq. 2, respectively. Parameters α controls the surface elasticity whereas β is the rigidity. They have a similar role than coefficients ω_{ij} in the continuous model.

Let us describe the adaptation of the different energies to our 3D model. The energies are intuitive extensions of the 2D active contour ones, suitable to our mesh representation. While the discretization of internal energies is simple for a parametric snake, it is not obvious how to implement the surface derivatives of eq. 2 in a triangular mesh. For a discrete planar contour, the continuity is the distance between successive neighbors. However, like in [5], it is preferable to use the absolute difference between this distance and a default length $\overline{d}$, which may be the mean distance computed over all vertices. Extending this principle to the mesh, E_{cont} maintains the vertices evenly spaced along the surface. Minimizing it reduces the gap between the mean squared distance $\overline{d^2}$ and the distance between the considered vertex and its adjacent vertices.

$$\begin{aligned} E_{cont}(\mathbf{p}'_i) &= \sum_{j \in A_i} \left| \overline{d^2} - \left\| \mathbf{p}'_i - \mathbf{p}_j \right\|^2 \right| \\ \overline{d^2} &= \frac{1}{n} \sum_{i=1}^{n} \frac{1}{|A_i|} \sum_{j \in A_i} \left\| \mathbf{p}_i - \mathbf{p}_j \right\|^2 \end{aligned} \tag{7}$$

The second internal energy is the curvature E_{curv}, which minimization results in a local smoothing effect, by making the vertex get closer to the centroid of its adjacent vertices. In a 2D snake, curvature is equivalent to the squared distance between the vertex and the middle of its two neighbors. By extension to 3D, the curvature of the tested point $\mathbf{p}'_i$ is the squared distance between $\mathbf{p}'_i$ and the centroid of the neighbors of vertex $\mathbf{p}_i$.

$$E_{curv}(\mathbf{p}'_i) = \left\| \mathbf{p}'_i - \frac{1}{|A_i|} \sum_{j \in A_i} \mathbf{p}_j \right\|^2 \tag{8}$$

Note that for a given mesh vertex $\mathbf{p}_i$, $E_{curv}(\mathbf{p}_i) = 0$ if $\mathbf{p}_i$ and all its adjacent vertices lie on the same plane. To attract vertices towards salient edges, the external energy E_{grad} is a function of normalized gradient magnitude g of image I. In presence of noisy data, the image is smoothed with a gaussian filter prior to gradient operation. In the following equations, G_σ is a gaussian kernel with standard deviation σ, $*$ is the convolution operator and g_{max} is the maximal edge intensity in the image.

$$\begin{aligned} g(\mathbf{p}) &= \left\| \nabla I(\mathbf{p}) * G_\sigma \right\| / g_{max} \\ E_{grad}(\mathbf{p}'_i) &= -g(\mathbf{p}'_i) \end{aligned} \tag{9}$$

As regards gradient magnitude, real 3D edge detection is obtained by convolving the image with the Zucker-Hummel operator [19], which usually yields better edge localization than using direct finite differences. It is made up of three $3 \times 3 \times 3$ masks ZH_x, ZH_y and ZH_z, each mask filtering the image in one dimension.

$$ZH_x = \begin{bmatrix} -k_1 & 0 & k_1 \\ -k_2 & 0 & k_2 \\ -k_1 & 0 & k_1 \end{bmatrix} \begin{bmatrix} -k_2 & 0 & k_2 \\ -k_3 & 0 & k_3 \\ -k_2 & 0 & k_2 \end{bmatrix} \begin{bmatrix} -k_1 & 0 & k_1 \\ -k_2 & 0 & k_2 \\ -k_1 & 0 & k_1 \end{bmatrix} \tag{10}$$

$$ZH_y = \begin{bmatrix} -k_1 & -k_2 & -k_1 \\ 0 & 0 & 0 \\ k_1 & k_2 & k_1 \end{bmatrix} \begin{bmatrix} -k_2 & -k_3 & -k_2 \\ 0 & 0 & 0 \\ k_2 & k_3 & k_2 \end{bmatrix} \begin{bmatrix} -k_1 & -k_2 & -k_1 \\ 0 & 0 & 0 \\ k_1 & k_2 & k_1 \end{bmatrix} \tag{11}$$

$$ZH_z = \begin{bmatrix} -k_1 & -k_2 & -k_1 \\ -k_2 & -k_3 & -k_2 \\ -k_1 & -k_2 & -k_1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} k_1 & k_2 & k_1 \\ k_2 & k_3 & k_2 \\ k_1 & k_2 & k_1 \end{bmatrix} \tag{12}$$

$$k_1 = \frac{\sqrt{3}}{3} \ ; \ k_2 = \frac{\sqrt{2}}{2} \ ; \ k_3 = 1 \tag{13}$$

To increase the capture range, we introduce a balloon energy E_{bal} derived from the inflation force proposed in [20]. It allows the mesh to be initialized far from the object boundaries.

$$E_{bal}(\mathbf{p}'_i) = \|\mathbf{p}'_i - (\mathbf{p}_i + k\vec{\mathbf{n}}_i)\|^2 \tag{14}$$

where $\vec{\mathbf{n}}_i$ is the unit inward normal, defined at vertex $\mathbf{p}_i$. The normal of vertex $\mathbf{p}_i$ is the normalized sum of the normals of the neighboring triangles [15]. Rigorously, the normal of a triangle t is the unit vector orthogonal to the plane defined by t. In the following expressions, T_i is the set of neighboring triangles of $\mathbf{p}_i$. The normal $\vec{\mathbf{n}}_t$ of a given triangle is the normalized cross product between two vectors belonging to the corresponding plane.

$$\vec{\mathbf{n}}_i = \frac{\sum_{t\in T_i} \vec{\mathbf{n}}_t}{\left\| \sum_{t\in T_i} \vec{\mathbf{n}}_t \right\|} \quad ; \quad \vec{\mathbf{n}}_t = s_t \frac{(\mathbf{p}_{t_2} - \mathbf{p}_{t_1}) \times (\mathbf{p}_{t_3} - \mathbf{p}_{t_1})}{\|(\mathbf{p}_{t_2} - \mathbf{p}_{t_1}) \times (\mathbf{p}_{t_3} - \mathbf{p}_{t_1})\|} \tag{15}$$

where $\mathbf{p}_{t_j}, j = 1...3$ are vertices of triangle t ($\mathbf{p}_i$ must be one of them). $s_t = \pm 1$ is the sign changing the orientation of $\vec{\mathbf{n}}_t$ insuring that it points towards the interior of the surface. Such a calculation of the normal vector is necessary to a correct balloon implementation. The motion resulting from the balloon energy minimization is either an expansion or a retraction of the surface, depending on the sign of coefficient δ. This one must be chosen regarding the initial position of the surface with respect to the target object.

In order to adapt local topology, remeshing is performed after each iteration of the greedy algorithm. The mesh is allowed to add or delete vertices to keep the distance between adjacent vertices homogeneous, resulting in a stable vertex distribution [21][15][22]. It insures that every couple of adjacent vertices $(\mathbf{p}_i, \mathbf{p}_j)$ satisfies the constraint:

$$d_{min} \leq \|\mathbf{p}_i - \mathbf{p}_j\| \leq d_{max} \tag{16}$$

where d_{min} and d_{max} are two user-defined thresholds, such that $d_{max} \leq 2d_{min}$. We choose their values near the neighborhood width w, so that the surface sampling is consistent with the motion range of the vertices. Adding or deleting vertices modifies local topology, thus topological constraints should be verified. To perform vertex adding or deleting, $\mathbf{p}_i$ and $\mathbf{p}_j$ should share exactly two common adjacent vertices: $|A_i \cap A_j| = 2$. When $\|\mathbf{p}_i - \mathbf{p}_j\| > d_{max}$, a new vertex is created at the middle of line segment $\mathbf{p}_i\mathbf{p}_j$ and connected to $\mathbf{p}_a$ and $\mathbf{p}_b$ (see middle part of figure 3). When $\|\mathbf{p}_i - \mathbf{p}_j\| < d_{min}$, $\mathbf{p}_j$ is deleted and $\mathbf{p}_i$ is translated to the middle location (see right part of figure 3).

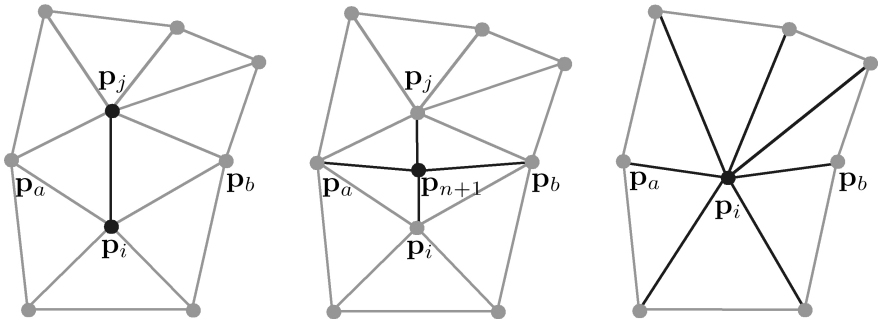

Fig. 3. Remeshing operations: vertex adding and deleting

Algorithm 1 Shifted Neighborhood Method: 3D Model

for $t \leftarrow 1$ to T **do**
  for $i \leftarrow 1$ to n **do**
    $\mathbf{p}_i^{(t+1)} = \arg\min_{\mathbf{p}'_k \in \mathcal{N}_i^{(t)}} E(\mathbf{p}'_k)$
    $\vec{\mathbf{d}}_i^{(t+1)} = \mathcal{B}(-1, 1, \mathbf{p}_i^{(t+1)} - \mathbf{p}_i^{(t)})$
    $\vec{\mathbf{s}}_i^{(t+1)} = \mathcal{B}(1, w-2, \vec{\mathbf{s}}_i^{(t)} - \vec{\mathbf{d}}_i^{(t+1)})$
    $\mathcal{N}_i^{(t+1)} = \left\{ \mathbf{p}_i^{(t+1)} + \mathbf{r} - \vec{\mathbf{s}}_i^{(t)} \mid \mathbf{r} \in [0, w-1]^3 \right\}$
  end for
end for

3 The Shifted Neighborhood Method

In order to improve active surfaces completion time, we used the shifted neighborhood method developed for 2D snakes in [10]. We adapted this greedy-based method on active surfaces. For each vertex and at each iteration, we modify the neighborhood in order to direct the searching space of each vertex to the directions that seems the most interesting. To define where these directions are, we use the information of the direction followed by each vertex during the last iteration. So each neighborhood will be shifted from one voxel in the direction followed previously. At each iteration, we compute the next shift applied to the vertex with:

$$\vec{\mathbf{d}}_i^{\,(t+1)} = \mathcal{B}(-1, 1, \mathbf{p}_i^{(t+1)} - \mathbf{p}_i^{(t)}) \tag{17}$$

The vector quantity $\vec{\mathbf{d}}_i^{(t)}$ represents the displacement applied on the neighborhood of the i^{th} vertex at iteration t. $\mathcal{B}$ is a shift limiting function, bounding the vector coordinates between two scalars:

$$\mathcal{B}(b_1, b_2, \vec{\mathbf{u}}) = \begin{pmatrix} \max(b_1, \min(b_2, u_x)) \\ \max(b_1, \min(b_2, u_y)) \\ \max(b_1, \min(b_2, u_z)) \end{pmatrix} \tag{18}$$

The displacement $\vec{\mathbf{d}}_i^{(t+1)}$ given by equation (17) allows us to define the new shift vector $\vec{\mathbf{s}}_i^{(t+1)}$ for each vertex of the active surface. Thus we have:

$$\vec{\mathbf{s}}_i^{(t+1)} = \mathcal{B}(1, w-2, \vec{\mathbf{s}}_i^{(t)} - \vec{\mathbf{d}}_i^{(t+1)}) \tag{19}$$

The next iteration of the greedy algorithm will be helded with these new neighborhoods. At this stage, we can define the algorithm for the shifted neighborhood method. This last consists in computing the new neighborhood $\mathcal{N}_i^{(t+1)}$ with equations (4), (17) and (19) at the end of each iteration, once all the vertices of the active surface have been modified. When included in the greedy algorithm for an active surface of n vertices and T iterations, the shifted neighborhood method is described in algorithm 1.

4 The Line Search Method

The line search method [10] originally applied for two-dimensional active contours allows to reduce completion time efficiently. We adapted this method on 3D active surfaces. The principle of this approach is to anticipate on the next iteration of the greedy algorithm using the information taken from the previous one. This method is launched at the end of each iteration of the greedy algorithm, once all the vertices have been translated.

The direction followed by each vertex $\mathbf{p}_i$ is memorized and we look toward it for a fixed number of voxels, which creates a linear neighborhood. These lasts are compared by computing their global energies in a similar way as it is done with the cubic neighborhood (see equation (6)). The voxel giving the lowest energy is then chosen for the new location of the current vertex. As a result, for each vertex, two neighborhoods are

Algorithm 2 Line Search Method: 3D Model

1: **for** $t \leftarrow 1$ to T **do**
2: **for** $i \leftarrow 1$ to n **do**
3: $\mathbf{p}_i^{(t+1)} = \arg\min_{\mathbf{p}'_k \in \mathcal{N}_i^{(t)}} E(\mathbf{p}'_k)$
4: Determine the direction $\vec{\mathbf{v}} = \dfrac{\mathbf{p}_i^{(t+1)} - \mathbf{p}_i^{(t)}}{\left\|\mathbf{p}_i^{(t+1)} - \mathbf{p}_i^{(t)}\right\|}$
5: Line Search: $m = \underset{k\in[0,l]}{\operatorname{argmin}} \left\{E(\mathbf{p}_i^{(t)} + k\vec{\mathbf{v}})\right\}$
6: Update: $\mathbf{p}_i^{(t+1)} \leftarrow \mathbf{p}_i^{(t+1)} + m\vec{\mathbf{v}}$
7: **end for**
8: **end for**

scanned consecutively: the cubic centered neighborhood and the linear neighborhood. The second algorithm describes the line search method integrated in the greedy algorithm for active surfaces. Let T be the number of iterations to be done by the greedy algorithm and l the number of voxels to be explored (length of the linear neighborhood).

5 Level Set and Fast Level Set implementation

In order to compare our methods with recent breaktroughs in the image segmentation domain we adapted the Fast Level Set implementation method to our 3D segmentation problem.
Basing ourselves on the work of Osher *et al* [23] and [7], we implemented our active surface model with level sets. We consider the parameterized surface $\mathbf{S}$ defined in section 2, on which we add a time dependency.

$$\begin{aligned} \mathbf{S} : \Omega^2 \times \mathbb{R}^+ &\longrightarrow \mathbb{R}^3 \\ (u,v,t) &\longmapsto (x(u,v,t), y(u,v,t), z(u,v,t))^T \end{aligned} \tag{20}$$

The surface evolves according to the following partial differential equation:

$$\frac{\partial \mathbf{S}(u,v,t)}{\partial t} = F\vec{\mathbf{n}} \tag{21}$$

where $\vec{\mathbf{n}}$ is the inward normal vector and F is the speed function applied on the surface points. The surface boundary is implicitly represented as the zero-level of a function $\psi : \mathbb{R}^3 \times \mathbb{R}^+ \to \mathbb{R}$. It comes:

$$\psi(\mathbf{S}(u,v,t),t) = 0 \qquad \forall (u,v) \in \Omega^2, \forall t \geq 0 \tag{22}$$

Function ψ evolves on its zero-level according to the equation:

$$\frac{\partial \psi(\mathbf{S}(u,v,t),t)}{\partial t} = F(u,v,t)\,\|\nabla\psi(\mathbf{S}(u,v,t),t)\| \tag{23}$$

If the speed function F is defined over the entire domain $\mathbb{R}^3 \times \mathbb{R}^+$ then eq. 23 can be extended such as:

$$\frac{\partial \psi(\mathbf{x},t)}{\partial t} = F(\mathbf{x},t)\,\|\nabla\psi(\mathbf{x},t)\| \qquad \forall \mathbf{x} \in \mathbb{R}^3,\ \forall t \geq 0 \tag{24}$$

The main advantage of the level-set implementation is its ability to automatically handle topological changes. Indeed, the curve can naturally split or merge with others without any additionnal implementation. Unfortunately the level-set method requires significant computationnal time. Several methods have been developed in order to accelerate the level-set implementation. The narrow band implementation [8] allows to

decrease the complexity of the algorithm from $O(n^2)$ to $O(n)$, n being the size of the image grid. The author considers a band around the zero-level of the level-set function. The partial differential equation (PDE) is solved only inside this region and not in the entire definition domain of ψ. Although this technique allows to significally reduce computational time of the level-set implementation, it still limits the use of the level-set in real time applications such as object tracking. Shi *et al* have recently developed in [9] an acceleration method based on the narrow band implementation called the Fast Level Set method. As it was originally designed for 2D segmentation, we extended it to 3D. The Fast Level Set method uses two lists to represent the surface: the list of outside boundary points L_{out} and the list of inside boundary points L_{in}.

$$\begin{aligned} L_{out} &= \{\mathbf{x} \mid \psi(\mathbf{x}) > 0, \exists\, \mathbf{y} \in N(\mathbf{x}), \psi(\mathbf{y}) < 0\} \\ L_{in} &= \{\mathbf{x} \mid \psi(\mathbf{x}) < 0, \exists\, \mathbf{y} \in N(\mathbf{x}), \psi(\mathbf{y}) > 0\} \end{aligned} \tag{25}$$

where $N(\mathbf{x})$ is the discrete neighborhood of $\mathbf{x}$.

The authors assume $\psi(\mathbf{x})$ to take only four integer values, according to the position of x:

$$\psi(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in L_{out} \\ -1 & \text{if } \mathbf{x} \in L_{in} \\ 3 & \text{if } \mathbf{x} \text{ is outside } \mathbf{S} \text{ and } \mathbf{x} \notin L_{out} \\ -3 & \text{if } \mathbf{x} \text{ is inside } \mathbf{S} \text{ and } \mathbf{x} \notin L_{in} \end{cases} \tag{26}$$

The authors define the dicrete optimality condition for the surface as:

The surface $\mathbf{S}$ ***with boundary points*** L_{in} ***and*** L_{out} ***is optimal if the speed function*** F ***satisfies:***

$$F(\mathbf{x}) < 0 \ \forall \mathbf{x} \in L_{out} \text{ and } F(\mathbf{x}) > 0 \ \forall \mathbf{x} \in L_{in} \tag{27}$$

While this optimality condition is not reached, the speed function F is calculated for every point in L_{in} and L_{out}. If $F(\mathbf{x}) > 0$ at a point in L_{out} the surface is moved outward. If $F(\mathbf{x}) < 0$ at a point in L_{in} the surface is moved inward. Once the surface has moved, the two lists are updated.

The main idea of this method is thus to make the surface evolve without solving the PDE of 24 which requires significant computationnal time but only by computing the speed function F. Again, segmentation is boundary-based, hence the surface should stop on strong edges. As a result, F is a function of curvature κ and gradient magnitude.

$$F(\mathbf{x}) = -\left\|\nabla I(\mathbf{x})\right\| - \lambda_\kappa \kappa(\mathbf{x}) \tag{28}$$

The curvature regularizes ψ and is weighted by coefficient λ_κ. It is expressed as follows:

$$\begin{aligned} \kappa &= \operatorname{div}\left(\frac{\nabla\psi}{\|\nabla\psi\|}\right) \\ &= \frac{\psi_{xx}(\psi_y^2+\psi_z^2)+\psi_{yy}(\psi_x^2+\psi_z^2)+\psi_{zz}(\psi_x^2+\psi_y^2)-2\psi_{xy}\psi_x\psi_y-2\psi_{xz}\psi_x\psi_z-2\psi_{yz}\psi_y\psi_z}{(\psi_x^2+\psi_y^2+\psi_z^2)^{3/2}} \end{aligned} \tag{29}$$

with notations $\psi_x = \dfrac{\partial\psi}{\partial x}, \psi_{xx} = \dfrac{\partial^2\psi}{\partial x^2}$, ...

For more details and testings on the method one can refer to [9]. The next section describes and compares the results obtained on tested images with our greedy acceleration techniques.

6 Experimental results

In this section, we present our experiments on 3D images. We compare the shifted neighborhood method and the line search method to the classical greedy algorithm. This comparative study also includes the results obtained with implicit surface modeling implemented with the fast level set method. Each tested image is made of several slices of gray objects embedded in white backgrounds, highly corrupted with gaussian noise. Different values of neighborhood width w are tested (obviously, the shifted neighborhood is not experimented with $w = 3$). For each image, the surface is initialized as a sphere with identical center and radius for all evolution methods, independently from its implementation (a mesh or a level set).

In order to evaluate segmentation, we use a function taking into account the overall distance between the estimated boundary and ground truth. Let $\mathcal{R}$ be the set of voxels belonging to the real boundary, and $\mathcal{E}$ the set of voxels belonging to the estimated boundary. For each voxel on the estimated boundary, we consider the distance to the nearest voxel on the real boundary, and conversely. The modified Hausdorff distance $\mathcal{H}_{mean}$ introduced in [24] measures the average fitting of the surface to the real boundary.

$$\begin{aligned} \mathcal{H}_{mean}(\mathcal{E},\mathcal{R}) &= \max(h_{mean}(\mathcal{E},\mathcal{R}), h_{mean}(\mathcal{R},\mathcal{E})) \\ h_{mean}(\mathcal{E},\mathcal{R}) &= \frac{1}{|\mathcal{E}|}\sum_{\mathbf{p}\in\mathcal{E}} \min_{\mathbf{q}\in\mathcal{R}} \|\mathbf{p}-\mathbf{q}\| \end{aligned} \tag{30}$$

In what follows, we compare computational times obtained on final surfaces having equivalent qualities, with respect to the modifief Hausdoff distance. Typically, H_{mean} have values around 1, which corresponds to good fitting of the real boundaries.

The first image is a $400 \times 400 \times 400$ data set representing a spiral and was chosen in order to test the methods when vertex adding is enabled. The active surface is initialized inside the 3D model with only 12 vertices. The final meshes for the three methods contain about one thousand vertices. We choose $\alpha = 0$, $\beta = 0.5$, $\gamma = 2$ and $\delta \in [-0.6, -1.1]$.

The second image is a $200 \times 200 \times 200$ model of a vase and is interesting to test the infiltration of the model into the concavities. For this particular 3D dataset, remeshing of the model is disabled. We use $\alpha = 0$, $\beta = 0.5$ for the greedy algorithm, 0.4 for the shifted neighborhood method and 0.3 for the line search, $\gamma = 2$ and $\delta = 0.8$. We initialize the meshes with 2562 vertices.

The third image represents three ellipsoids and allows to have both salient and smooth angles on the same model. The image size is $200 \times 200 \times 200$. We prevent remeshing of the model and initialize it with 2562 vertices. The parameters are $\alpha = 0.5$, $\beta = 0.3$ for the greedy algorithm and 0.4 for the shifted neighborhood method and the line search, $\gamma = 2$ and $\delta \in [0.3, 0.7]$.

In order to compare boundary fitting ability of the methods on real data, we tested them on computed tomography (CT) data sets of the abdomen, in which the active surface was used to detect boundaries of the aorta. Such segmentation is done in the framework of abdominal aortic aneurysm diagnosis. The mesh was initialiazed as a small sphere inside the aorta and inflated thanks to the balloon energy. Figure 4 shows one slice of each image and the visual 3D results obtained with the line search method. Tables 1 and 2 list computational times obtained on synthetic and CT images, respectively. Each method and each window width is systematically tested on the images (tests were made on a Pentium IV 1.7 GHz with 512 Mb RAM).

Mesh-based methods (initial greedy algorithm, LS and SN) need fewer iterations than the level set surface to reach the boundaries, which is mainly due to the level set front. A voxel neighboring the front (the L_{out} list) needs more than one iteration to change its status, from outer to inner voxel. An iterations of the level set method is also more computationally intensive than one of the explicit methods. The discretization of the evolving front is the same as the image grid, so that each voxel located on the front needs to be considered. On the mesh, the motion of a vertex does not affect its coordinates but also all neighboring triangles, which cover

many voxels (the quantity of voxels depends on the sampling resolution w). As regards distance measures, it is interesting to note that explicit methods lead to more accurate results than the level set approach, except on the "Vase" data set. We may assume that level sets are globally more sensitive to noise and prone to boundary leakage issues, because of the implicit formulation of the regularizing curvature energy, which has a limited range. On the mesh, the internal energy of a vertex affects a larger portion of the surface. The better performance of the implicit surface on the "Vase" data set is explained by the presence of sharp angles at the shape borders. Curvature prevents the mesh from fitting angular parts accurately, whereas the level set surface is not limited thanks to its discretization. Indeed, it can easily grow voxels into small concave parts of the boundary.

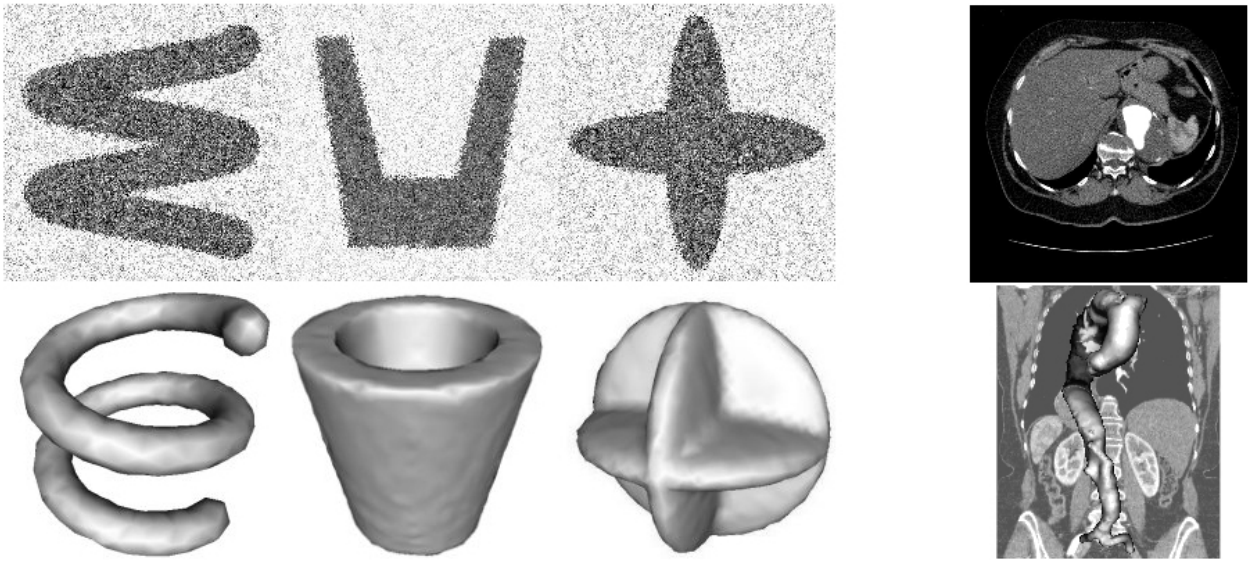

Fig. 4. 2D slices of 3D noisy images (top) and surface results obtained with the line search method (bottom)

7 Conclusion and future work

In this article we have described two acceleration methods for active surfaces evolving with the greedy algorithm. The first one is based on a smart orientation of the neighbourhood grid of each vertex regarding the directions followed in the preceding iterations. The second one uses the same direction information to make each vertex search for a better position along an exploration line. The main application domain of our methods is time-dependent segmentation with *a priori* knowledge about the topology of the object, such as 3D videos. We compared our two acceleration methods with a recent fast implicit implementation of active contour based on the level-set approach.

As shown in table 1 and 2, our methods allowed us to accelerate the greedy algorithm for active surfaces. Mesh-based methods (the greedy algorithm and our acceleration methods) leads to performances turning out to be far beyond the level set-based method ones. Accelerations methods tend to make completion time fall below 1s (for the best configuration) whereas the fast level set approach exceeds 30s. The best acceleration method for 2D active contours was the shifted neighborhood but we detected the line search as the best to be applied on 3D active surfaces. The explication is that in three dimensions an exploration line

stays a line whereas a square neighborhood becomes cubic, rising the completion times added by the shifted neighborhood method.

We also tested the Deformed Neighborhood method discribed in [10] but it was not efficient on active surfaces for the same reasons. We can also notice that the contribution of the shifted neighborhood method is better with a large neighborhood. Indeed, the shiftings are dependent of its size.

We are developing an hybrid model based on the shifted neighborhood method and the physic-based approach of parametric active contours [25]. The main idea is to use the information of the force vector to direct the shift of the neighborhood grid. We also plan to upgrade the performances of our methods by using a multi-resolution approach.

Image	**Neighborhood**	**Method**	**# iterations**	**Time (s)**	$\mathbf{H_{mean}}$
Spiral ($400 \times 400 \times 400$ voxels)	w=3	Greedy	400	0.30	0.497
		LS	65	0.16	0.499
	w=5	Greedy	195	0.52	0.723
		LS	63	0.30	0.734
		SN	138	0.31	0.728
	w=7	Greedy	145	0.63	0.985
		LS	60	0.57	0.997
		SN	97	1.11	0.991
	Fast level set		1060	44.17	0.995
3 ellipsoids ($200 \times 200 \times 200$ voxels)	w=3	Greedy	47	0.63	0.534
		LS	19	0.41	0.545
	w=5	Greedy	25	1.05	0.716
		LS	12	0.69	0.734
		SN	16	0.98	0.721
	w=7	Greedy	18	2.24	0.961
		LS	12	1.86	0.985
		SN	14	2.07	0.972
	Fast level set		120	30.82	1.030
Vase ($200 \times 200 \times 200$ voxels)	w=3	Greedy	155	0.98	0.665
		LS	68	0.72	0.687
	w=5	Greedy	120	1.27	0.875
		LS	50	0.85	0.896
		SN	92	1.08	0.892
	w=7	Greedy	109	1.98	1.337
		LS	43	1.13	1.554
		SN	60	2.20	1.423
	Fast level set		210	47.04	0.815

Table 1. Comparison of completion times between classical greedy algorithm, line search method (LS), shifted neighborhood method (SN)and fast level set on synthetic data sets

Image	Neighborhood	Method	# iterations	Time (s)	H_{mean}
CT 1 ($512 \times 512 \times 810$ voxels)	w=3	Greedy	580	25.30	1.324
		LS	98	17.7	1.458
	w=5	Greedy	330	9.65	1.694
		LS	117	7.23	1.734
		SN	256	8.03	1.710
	w=7	Greedy	240	8.69	1.936
		LS	102	6.68	2.034
		SN	168	9.73	1.967
	Fast level set		1227	302.41	3.345
CT 2 ($512 \times 512 \times 400$ voxels)	w=3	Greedy	425	18.57	1.125
		LS	96	13.99	1.241
	w=5	Greedy	242	7.08	1.439
		LS	85	5.30	1.476
		SN	187	5.89	1.454
	w=7	Greedy	176	6.37	1.646
		LS	86	4.95	1.756
		SN	136	7.16	1.672
	Fast level set		750	185.07	2.381
CT 3 ($512 \times 512 \times 400$ voxels)	w=3	Greedy	313	14.34	0.754
		LS	75	10.2	0.832
	w=5	Greedy	178	5.23	0.965
		LS	65	4.47	0.988
		SN	137	4.86	0.974
	w=7	Greedy	132	4.82	1.104
		LS	62	3.81	1.159
		SN	96	5.49	1.121
	Fast level set		735	182.446	2.016

Table 2. Comparison of completion times between classical greedy algorithm, line search method (LS), shifted neighborhood method (SN) and fast level set on CT (Computed Tomography) data sets

References

1. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: active contour models. International Journal of Computer Vision **1** (1988) 321–331
2. Montagnat, J., Delingette, H., Ayache, N.: A review of deformable surfaces: topology, geometry, and deformation. Image and Vision Computing **19** (2001) 1023–1040
3. Mille, J., Boné, R., Makris, P., Cardot, H.: 3D segmentation using active surface: a survey and a new model. In: 5^{th} Int. Conf. on Visualization, Imaging & Image Processing (VIIP), Benidorm, Spain (2005) 610–615
4. Zhang, Z., Braun, F.: Fully 3D active surface models with self-inflation and self-deflation forces. In: IEEE Computer Vision and Pattern Recognition (CVPR), San Juan, Puerto Rico (1997) 85–90
5. Williams, D., Shah, M.: A fast algorithm for active contours and curvature estimation. Computer Vision, Graphics, and Image Processing: Image Understanding **55** (1992) 14–26
6. Bulpitt, A., Efford, N.: An efficient 3D deformable model with a self-optimising mesh. Image and Vision Computing **14** (1996) 573–580
7. Malladi, R., Sethian, J., Vemuri, B.: Shape modeling with front propagation: a level set approach. IEEE Transactions on Pattern Analysis and Machine Intelligence **17** (1995) 158–175

8. Adalsteinsson, D., Sethian, J.: A fast level set method for propagating interfaces. Journal of Computational Physics **118** (1995) 269–277
9. Shi, Y., Karl, W.: A fast level set method without solving PDEs. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP). Volume 2., Philadelphia, USA (2005) 97–100
10. Olivier, J., Boné, R., Rousselle, J.J.: Comparison of active contour acceleration methods. In: 5^{th} Int. Conf. on Visualization, Imaging & Image Processing (VIIP), Benidorm, Spain (2005)
11. Olivier, J., Mille, J., Boné, R., Rousselle, J.J.: Active surfaces acceleration methods. In: Computational Modelling of Objects Represented in Images: fundamentals, methods and applications (CompIMAGE), Coimbra, Portugal (2006)
12. Cohen, L., Cohen, I.: Finite element methods for active contour models and balloons for 2D et 3D images. IEEE Transactions on Pattern Analysis and Machine Intelligence **15** (1993) 1131–1147
13. Mille, J., Boné, R., Makris, P., Cardot, H.: Greedy algorithm and physics-based method for active contours and surfaces: a comparative study. In: IEEE International Conference on Image Processing (ICIP), Atlanta, USA (2006) 1645–1648
14. McInerney, T., Terzopoulos, D.: 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** (1995) 69–83
15. Park, J.Y., McInerney, T., Terzopoulos, D.: A non-self-intersecting adaptive deformable surface for complex boundary extraction from volumetric images. Computer & Graphics **25** (2001) 421–440
16. Amini, A., Weymouth, T., Rain, R.: Using dynamic programming for solving variational problem in vision. IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1991) 855–867
17. Ji, L., Yan, H.: Attractable snakes based on the greedy algorithm for contour extraction. Pattern Recognition **35** (2002) 791–806
18. Lam, C., Yuen, S.: An unbiased active contour algorithm for object tracking. Pattern Recognition Letters **19** (1998) 491–498
19. Zucker, S., Hummel, R.: A three-dimensional edge operator. IEEE Transactions on Pattern Analysis and Machine Intelligence **3** (1981) 324–331
20. Cohen, L.: On active contour models and balloons. Computer Vision, Graphics, and Image Processing: Image Understanding **53** (1991) 211–218
21. Lachaud, J., Montanvert, A.: Deformable meshes with automated topology changes for coarse-to-fine three-dimensional surface extraction. Medical Image Analysis **3** (1999) 187–207
22. Slabaugh, G., Unal, G.: Active polyhedron: surface evolution theory applied to deformable meshes. In: IEEE Computer Vision and Pattern Recognition (CVPR). Volume 2., San Diego, USA (2005) 84–91
23. Osher, S., Sethian, J.: Fronts propagation with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. Journal of Computational Physics **79** (1988) 12–49
24. Dubuisson, M.P., Jain, A.: A modified Hausdorff distance for object matching. In: 12^{th} International Conference on Pattern Recognition (ICPR), Jerusalem, Israel (1994) 566–568
25. Lobregt, S., Viergever, M.: A discrete dynamic contour model. IEEE Transactions on Medical Imaging **14** (1995) 12–24