Published in Image Processing On Line on 2021–02–28.
Submitted on 2020–01–31, accepted on 2021–02–13.
ISSN 2105–1232 © 2021 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2021.292

An Algorithm for 3D Curve Smoothing

Daniel Santana-Cedrés[1], Nelson Monzón[1] and Luis Alvarez[1]

[1] CTIM (Centro de Tecnologías de la Imagen), University of Las Palmas de Gran Canaria, Spain
(daniel.santana104@alu.ulpgc.es, nelson.monzon@ulpgc.es, lalvarez@ulpgc.es)

Abstract

In this work we present an application of curve evolution techniques to the smoothing of 3D curves. We study two types of application scenarios: in the first one, the curve is just given by an ordered set of 3D points, and in the second one, the curve represents the medial axis of a 3D volume. In this last scenario, the input of the algorithm is the 3D volume and a 3D curve representing an approximation of the volume medial axis. We propose an algorithm for 3D curve smoothing, based on a curve evolution equation which includes both scenarios. We present a variety of experiments to show the performance of the proposed technique.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Compilation and usage instruction are included in the `README.txt` file of the archive. Besides, we include doxygen documentation (in `html`) regarding the main classes and methods.

Supplementary Material

Along with the code, we provide example files to test the method. An ASCII file containing the set of 3D points representing the original curve and a 3D volume (stored as unsigned char in Analyze format) are included. In this particular case, they represent an aorta centerline and its segmentation. Moreover, we also include the results in order to perform further comparisons (ASCII and .obj files with the 3D models). The 3D volume is optional and any curve given by an ordered set of 3D points can be used.

Keywords: 3D curve smoothing

[1] https://doi.org/10.5201/ipol.2021.292

1 Introduction

The regularization of 3D curves is an important issue from a theoretical and practical point of view. In particular, in 3D medical imaging (as for instance CT scans), 3D curves appear in a natural way as the centerlines of different organs. For instance, Alvarez et al., in [3], propose a variational model for the regularization of the centerline of the aortic vessel. In this paper, we present an algorithm to regularize 3D curves based on a curve evolution equation.

Let $\mathcal{C}(s)$ be a 3D curve, $\mathcal{C} : [0, |\mathcal{C}|] \to R^3$, where s represents the arc-length parameter and $|\mathcal{C}|$ the length of the curve. Let A be a 3D set. A can be a 3D volume and then $\mathcal{C}(s)$ is an approximation of the volume medial axis (skeleton), or A can be an initial curve, $\mathcal{C}^0$, that is $A = \{\mathcal{C}^0(s) : s \in [0, |\mathcal{C}^0|]\}$. We denote by $d_{\partial A}(x)$ the usual signed distance function to the set A given by

$$d_{\partial A}(x) = \begin{cases} d(x, \partial A) & if \quad x \notin A, \\ -d(x, \partial A) & if \quad x \in A. \end{cases} \tag{1}$$

where $d(x, \partial A)$ is the Euclidean distance between x and the set ∂A defined as $d(x, \partial A) = inf_{y \in \partial A} \|x - y\|$ and $\|.\|$ is the usual Euclidean norm. The curve smoothing algorithm we propose is based on the solution of the curve evolution equation

$$\begin{cases} \frac{d}{dt}\mathcal{C}^t(s) = -\nabla d_{\partial A}(\mathcal{C}^t(s)) + wk(t,s)\mathcal{N}(t,s) & \text{for} \quad t > 0 \text{ and } s \in [0, |C^t|] \\ \mathcal{C}^0(s) = \mathcal{C}(s) & \text{for} \quad s \in [0, |C|] \\ \mathcal{C}^t(0) = \mathcal{C}(0) & \text{for} \quad t \geq 0 \\ \mathcal{C}^t(|\mathcal{C}^t|) = \mathcal{C}(|\mathcal{C}|) & \text{for} \quad t \geq 0, \end{cases} \tag{2}$$

where s represents the arc-length of the curve $\mathcal{C}^t$. We denote by $k(t, s)$ the curvature of $\mathcal{C}^t(s)$, and by $\mathcal{N}(t, s)$ the corresponding unit normal direction. The curve evolution equation (2) is composed of two contributions:

- Advection by the negated distance gradient $\nabla d_{\partial A}(\mathcal{C}^t(s))$, which moves the curve towards the centerline of A.
- The curve shortening flow, which has a smoothing effect.

The parameter w represents a balance between both terms. As $t \to \infty$, the curve $\mathcal{C}^t$ converges, in our experiments, to a stationary curve which, as desired, is both smooth and close to the centerline of A. The curve evolution equation (2) is inspired by the following energy minimization problem proposed in [3]

$$\mathcal{C}_w = \underset{\mathcal{C}:\mathcal{C}(0)=p_0, \mathcal{C}(|\mathcal{C}|)=p_1}{\arg\min} E_S(\mathcal{C}) \equiv \int_0^{|\mathcal{C}|} d_{\partial A}(\mathcal{C}(s))ds + w|\mathcal{C}|. \tag{3}$$

However, (2) is not equivalent to the Euler-Lagrange equation for minimizing (3). Indeed, Equation (2) can be considered as a simplification of the Euler-Lagrange equation for minimizing (3). In addition, let us mention that, in the case that for some $p \in A$, $d_{\partial A}(p) + w < 0$, the energy (3) could be not bounded and the corresponding Euler-Lagrange equation could be unstable. Using the above energy model, we aim to maximize the distance between the curve and the boundary of set A, but keeping the curve smooth. The larger the value of w the stronger the regularization effect. In this paper we present, in detail, an algorithm to solve the curve evolution equation (2). The main issues we have to address to implement this algorithm are the following:

1. The computation of the signed distance function (1) for a given 3D set, A.

2. The implementation of an algorithm to solve the evolution equation (2).

3. An algorithm to reparameterize a 3D curve using a constant arc-length distance between the curve points. To avoid the curve degeneration, we use this algorithm in each iteration to solve the curve evolution equation (2). This step is important because otherwise, we have realized experimentally, that the curve can degenerate (points of the 3D curve collapsing, etc.) and the procedure to make evolve the curve can fail.

The main contribution of the paper is the design of an algorithm for 3D curve regularization using a curve evolution equation. The algorithm for calculating the 3D distance is not a relevant contribution of the work and in this sense other options are possible. The rest of the paper is organized as follows: in Section 2, we present some related works. In Section 3 we present the numerical scheme to solve the curve evolution equation (2). In Section 4, we study in detail the proposed algorithm for 3D curve smoothing. In Section 5, we show some experiments on curves with or without a 3D volume associated. Finally, in Section 6, we present some conclusions.

2 Related Work

Most of the work in the literature for curve regularization is devoted to the 2D case. In [5] and [10], some energies for curvature penalized minimal path are proposed to regularize 2D curves. In [6], the authors propose a minimal path approach for tubular structures segmentation in 2D images with applications to retinal vessel segmentation.

One important application framework of 3D curve regularization is medical imaging. Among the experiments presented in this paper an example of aorta centerline regularization is shown. Automatic aorta segmentation algorithms for CT scans have been previously developed (see [9] for a survey). For instance, in [13], the authors proposed an iterative method based on building a 2D region for segmenting the ascending aorta. In [4], a tracking procedure of the aorta centerline is presented. In [11], the authors introduce a method for the automatic segmentation of the aorta and the centerline estimation. In [1], an active contour method for the aorta segmentation is proposed. In [2], the aorta lumen contour is estimated using an *ellipse motion tracking.* In [12], authors use the aorta centerline and the aorta segmentation as reference structures to detect anatomical landmarks of the aorta in CTA images.

3 Numerical Scheme to Solve the Curve Evolution Equation

We present a basic numerical scheme to solve the curve evolution equation (2). In practice, a 3D curve $\mathcal{C}$ is given by a collection of 3D points $\{\mathcal{C}_i\}_{i=1,\dots,\chi(\mathcal{C})}$, where $\chi(\mathcal{C})$ represents the number of points of the curve, $\mathcal{C}_1 = p_0$ and $\mathcal{C}_{\chi(\mathcal{C})} = p_1$. We assume that

$$\begin{aligned} \|\mathcal{C}_i - \mathcal{C}_{i-1}\| &= h \quad \text{for all } i = 2, \dots, \chi(\mathcal{C}) - 1, \\ \left\|\mathcal{C}_{\chi(\mathcal{C})} - \mathcal{C}_{\chi(\mathcal{C})-1}\right\| &\leq h, \end{aligned} \tag{4}$$

that is, we use a curve parameterization with a constant arc-length $h > 0$.

For the first term of the curve evolution equation (2), we consider in each point $\mathcal{C}_i$ a gradient descent type scheme of the form

$$\mathcal{C}_i^{n+1} = \mathcal{C}_i^n - \delta \nabla d_{\partial A}(\mathcal{C}_i^n), \tag{5}$$

where $\delta > 0$ represents the discretization step. To discretize the right part of the curve evolution equation (2), we consider the curvature shortening flow described, for instance, in [7]. This flow tends to reduce the length of the curve and can be formulated as follows

$$\mathcal{C}_i^{n+1} = \mathcal{C}_i^n + \delta k_i^n \mathcal{N}_i^n, \tag{6}$$

where k_i^n represents an approximation to the curvature and $\mathcal{N}_i^n$ the unit normal direction at the point in $\mathcal{C}_i^n$ using the Frenet–Serret frame. By combining both schemes and adding the weight w, we obtain the following discretization scheme for the curve evolution equation (2)

$$\mathcal{C}_i^{n+1} = \mathcal{C}_i^n - \delta \nabla d_{\partial A}(\mathcal{C}_i^n) + \delta w k_i^n \mathcal{N}_i^n \quad \text{for } i = 2, \ldots, \chi(\mathcal{C}) - 1, \text{ and } n > 0, \tag{7}$$

to simplify the notation we use δ as a generic time step that can be formally different in Equations (5), (6) and (7). We point out that, after each iteration, we need to reparameterize the curve $\mathcal{C}^{n+1}$ in order to preserve the constant arc-length condition (4). We compute the unit normal vector $\mathcal{N}_i^n$ as

$$\mathcal{N}_i^n = \begin{cases} \dfrac{\frac{\mathcal{C}_{i-1}^n + \mathcal{C}_{i+1}^n}{2} - \mathcal{C}_i^n}{\left\| \frac{\mathcal{C}_{i-1}^n + \mathcal{C}_{i+1}^n}{2} - \mathcal{C}_i^n \right\|} & \text{if} \quad \frac{\mathcal{C}_{i-1}^n + \mathcal{C}_{i+1}^n}{2} \neq \mathcal{C}_i^n, \\ \overrightarrow{0} & \text{otherwise,} \end{cases} \tag{8}$$

and the curvature k_i^n is approximated as the quotient between the angle, $\theta_i^n = \angle \mathcal{C}_{i-1}^n \mathcal{C}_i^n \mathcal{C}_{i+1}^n$, of the vectors $\overrightarrow{\mathcal{C}_{i-1}^n \mathcal{C}_i^n}$ and $\overrightarrow{\mathcal{C}_i^n \mathcal{C}_{i+1}^n}$ and the arc-length h, that is

$$k_i^n = \frac{\theta_i^n}{h}. \tag{9}$$

We use as stopping criterion of the iterative scheme (7) the condition

$$\frac{|E_S(\mathcal{C}^n) - E_S(\mathcal{C}^{n-1})|}{|E_S(\mathcal{C}^{n-1})|} < TOL, \tag{10}$$

where the computation of $E_S(\mathcal{C})$ is the energy defined in (3). $TOL > 0$ is a parameter to fix the stopping criterion of the scheme. In the experiments presented in this paper we use $TOL = 10^{-5}$.

Scheme (7) is a basic approximation of a minimizer of energy (3) but it is not derived from the Euler-Lagrange equation of (3) due to the choice of the curvature shortening flow introduced in (6). Nevertheless, despite this theoretical limitation, in the experiments we show that scheme (7) behaves as a good minimizer of (3) (see Figure 6).

Automatic estimation of the discretization step δ

We can estimate δ automatically using a two-step process: on the one hand, using as potential the normalized signed distance function, we have that $\|\nabla d_{\partial A}(\mathcal{C}_i^n)\| \leq 1$. Therefore, by imposing in the scheme (5) that

$$\delta \leq \frac{h}{2}, \tag{11}$$

we obtain that the point $\mathcal{C}_i^{n+1}$ is closer to $\mathcal{C}_i^n$ than to $\mathcal{C}_{i-1}^n$ and $\mathcal{C}_{i+1}^n$. On the other hand, with respect to the curvature part we impose that

$$\delta w k_i^n \leq \left\| \frac{\mathcal{C}_{i-1}^n + \mathcal{C}_{i+1}^n}{2} - \mathcal{C}_i^n \right\|, \tag{12}$$

which means that the curvature flow never makes the point $\mathcal{C}_i^n$ to move to the other side of the segment $\overline{\mathcal{C}_{i-1}^n \mathcal{C}_{i+1}^n}$. Using a straightforward computation, we obtain that the above condition is equivalent to

$$\delta w \frac{\theta_i^n}{h} \leq h \cos\left(\frac{\pi - \theta_i^n}{2}\right). \tag{13}$$

We observe that the function

$$f(\theta) = \delta w \frac{\theta}{h} - h \cos\left(\frac{\pi - \theta}{2}\right), \tag{14}$$

satisfies that

$$f(\pi) = \delta w \frac{\pi}{h} - h \leq 0 \Leftrightarrow \delta \leq \frac{h^2}{w\pi},$$

and we can easily check that if

$$\delta \leq \frac{h^2}{w\pi}, \tag{15}$$

then $f(\theta) \leq 0$ for any $\theta \in [0, \pi]$ and then (13) is satisfied. Therefore, joining both estimations of δ for each part of the numerical scheme, we can fix automatically δ as

$$\delta = \frac{\min\{\frac{h}{2}, \frac{h^2}{w\pi}\}}{2}. \tag{16}$$

We point out that with this choice of δ we have that

$$\left\|\mathcal{C}_i^{n+1} - \mathcal{C}_i^n\right\| \leq \delta + \delta w k_i^n \leq \frac{h}{4} + \frac{1}{2}\left\|\frac{\mathcal{C}_{i-1}^n - \mathcal{C}_i^n}{2} + \frac{\mathcal{C}_{i+1}^n - \mathcal{C}_i^n}{2}\right\| < \frac{3}{4}h, \tag{17}$$

which ensures that the velocity of point displacement is always smaller than the distance between points. Notice that this is not a theoretical proof of the stability of the scheme but, in practice, we experienced that using this choice for δ we obtain a stable numerical evolution of (7).

4 3D Curve Smoothing Algorithm

In this section, we describe the main steps of the 3D Curve Smoothing Algorithm we propose. We can identify three main steps which are described as algorithm pseudocodes in this section: first, we need a function to reparameterize a curve in such a way that its points are equally spaced (Algorithm 1); second, we implement a procedure to compute the distance in 3D inside and outside a 3D set (Algorithm 2); and finally we present the algorithm to smooth the curve using the numerical scheme explained in the previous section (Algorithm 3).

In Algorithm 1, we describe the function devoted to reparameterize a curve. This algorithm receives as input the initial curve, $\{\mathcal{C}_i^I\}_{i=1,\ldots,\chi(\mathcal{C}^I)}$, the constant distance, h, between consecutive points of the new curve, and the maximum length of the new curve. The objective is to return a new curve, $\{\mathcal{C}_m^O\}_{m=1,\ldots,\chi(\mathcal{C}^O)}$, in which the points are equally spaced by the indicated distance h, without overpass the maximum curve length allowed.

The algorithm begins assigning $\mathcal{C}_1^O = \mathcal{C}_1^I$. Then we iterate through the original curve. $\mathcal{C}_k^I$ represents the current position of the original curve and $\mathcal{C}_m^O$ the current position of the new curve. In the case $\|\mathcal{C}_k^I - \mathcal{C}_m^O\| \leq h$ we assign

$$\mathcal{C}_{m+1}^O = \mathcal{C}_m^O + h \frac{\mathcal{C}_k^I - \mathcal{C}_m^O}{\|\mathcal{C}_k^I - \mathcal{C}_m^O\|},$$

and we update m ($m = m + 1$). In the case $\|\mathcal{C}_k^I - \mathcal{C}_m^O\| > h$ we advance k ($k = k + 1$) until $\|\mathcal{C}_k^I - \mathcal{C}_m^O\| \leq h$. In this case, since $\left\|\mathcal{C}_{k-1}^I - \mathcal{C}_m^O\right\| < h$, there exists $\lambda \in (0, 1]$ such that

$$\left\|\mathcal{C}_{k-1}^I + \lambda\vec{v}_1 - \mathcal{C}_m^O\right\| = h,$$

where $\vec{v}_1 = \mathcal{C}_k^I - \mathcal{C}_{k-1}^I$. A straightforward calculation yields

$$\lambda = \frac{-(\vec{v}_2, \vec{v}_1) + \sqrt{(\vec{v}_2, \vec{v}_1)^2 - (\vec{v}_1, \vec{v}_1)\left((\vec{v}_2, \vec{v}_2) - h^2\right)}}{(\vec{v}_1, \vec{v}_1)},$$

where $\vec{v}_2 = \mathcal{C}_{k-1}^I - \mathcal{C}_m^O$. In this case, we assign $\mathcal{C}_{m+1}^O = \mathcal{C}_{k-1}^I + \lambda\vec{v}_1$ and we update m ($m = m + 1$). See the description of Algorithm 1 for more details.

Algorithm 1: Function `curve3D_reparameterization`($\mathcal{C}^I$,h,LC_{max})

input : $\mathcal{C}^I$ input curve (the number of points of $\mathcal{C}^I$ is $\chi(\mathcal{C}^I)$)
h: constant length of the segments of the new curve
LC_{max}: maximum length of the new curve
output : $\mathcal{C}^O$: reparameterized output curve (initially empty)

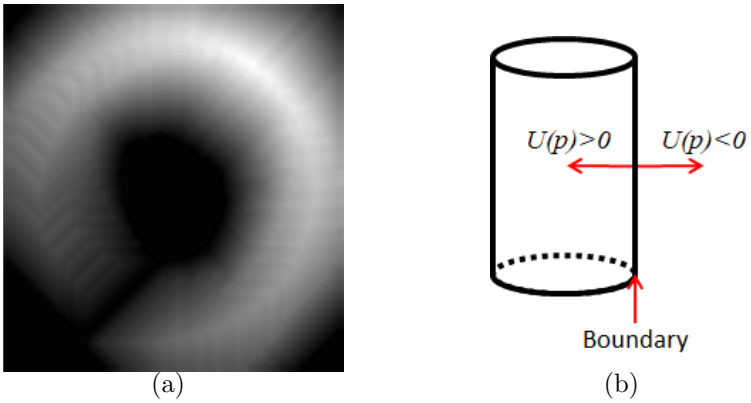
$\mathcal{C}_1^O \leftarrow \mathcal{C}_1^I$; *// Add the first point to the output curve*
$k \leftarrow 2$;
$m \leftarrow 1$; *// m represents in each iteration, $\chi(\mathcal{C}^O)$, that is, the number of points of $\mathcal{C}^O$*
while (*true*) **do**
  if ($|\mathcal{C}^O| \geq LC_{max}$) **then** *// Check the size of the new curve*
    break;
  $\vec{v} \leftarrow \mathcal{C}_k^I - \mathcal{C}_m^O$; *// Vector between the current point of $\mathcal{C}^I$ and the last one added to $\mathcal{C}^O$*
  if ($\|\vec{v}\| \geq h$) **then**
    $\mathcal{C}_{m+1}^O \leftarrow \mathcal{C}_m^O + h \cdot \frac{\vec{v}}{\|\vec{v}\|}$;
    $m \leftarrow m + 1$;
    continue;
  $k \leftarrow k + 1$;
  // Advance k until $\|\vec{v}\| \geq h$
  while ($k < \chi(\mathcal{C}^I)$ *and* $\|\vec{v}\| < h$) **do**
    $\vec{v} \leftarrow \mathcal{C}_k^I - \mathcal{C}_m^O$;
    if ($\|\vec{v}\| < h$) **then**
      $k \leftarrow k + 1$;
  if ($k == \chi(\mathcal{C}^I)$) **then**
    break;
  / In this case, $\|\mathcal{C}_m^O - \mathcal{C}_{k-1}^I\| < h$ and $\|\mathcal{C}_m^O - \mathcal{C}_k^I\| \geq h$. We compute a point in the segment $\overline{\mathcal{C}_{k-1}^I\mathcal{C}_k^I}$ with the prefixed distance h from the last point $\mathcal{C}_m^O$. See the text for details. */*
  $\vec{v}_1 \leftarrow \mathcal{C}_k^I - \mathcal{C}_{k-1}^I$;
  $\vec{v}_2 \leftarrow \mathcal{C}_{k-1}^I - \mathcal{C}_m^O$;
  $det \leftarrow \sqrt{(\vec{v}_1 \cdot \vec{v}_2)^2 - (\vec{v}_1 \cdot \vec{v}_1)((\vec{v}_2 \cdot \vec{v}_2) - (h)^2)}$;
  $\mathcal{C}_{m+1}^O \leftarrow \mathcal{C}_{k-1}^I + \frac{-(\vec{v}_1 \cdot \vec{v}_2) + det}{(\vec{v}_1 \cdot \vec{v}_1)}\ \vec{v}_1$;
  $m \leftarrow m + 1$;
// Check the difference between the last points of both curves and add the last point of $\mathcal{C}^I$ if necessary
if ($\|\mathcal{C}_{last}^I - \mathcal{C}_m^O\| > 0$) **then**
  $\mathcal{C}_{m+1}^O \leftarrow \mathcal{C}_{\chi(C_I)}^I$;
return $\mathcal{C}^O$;

Algorithm 2 describes the method to compute the 3D distance inside and outside a set A. In the case where A is a 3D curve with no interior points, the computation is done only for points outside

the curve. This function receives as parameter a 3D image, I, where the set A is included as a level set. We assume that $I(p) > 0$ if $p \in A$ and $I(p) \leq 0$ if $p \notin A$. The voxel size of the 3D image is included in the information associated to the 3D image. We also indicate the maximum values for the distance inside (Max_d^{in}) and outside (Max_d^{out}) of the provided level set, as well as the neighborhood type (6, 18, or 26 neighbors). As an outcome, the algorithm provides a 3D image with the signed distances to the boundary of A.

First, we store in 2 arrays (N and N_d respectively) the relative indexes in the 3D image and distances of one voxel to its neighbors. Such arrays depend on the type of neighborhood selected and the voxel size. These arrays are ordered in an increasing way accordingly to the distance of a point to its neighbors. We also use an auxiliary 3D image, V, to control the voxels that we have already visited and assigned a distance to the boundary of set A during the algorithm execution. We initialize the value of V as $V(p) = 10^8$ for all voxels p in order to identify the voxels that we have not visited yet.

Once the boundary has been initialized, the distance of the inner and outer regions are processed independently but using the same approach. Considering the limit of the maximum value indicated by the parameter, the idea is to propagate the distance with growing values, from the boundary toward inner or outer regions (Figure 1(b)). The only difference is the use of positive (inside) or negative (outside) values. In order to control how the region grows, we use the image with the visited voxels, V. In this way, the value of this mark increases or decreases depending on whether we are processing the region toward the inside or outside the set.



(a) (b)

Figure 1: Computation of the distance function: (a) example of one of the slices of the distance image (obtained by applying algorithm 2) to the synthetic curve described in Section 5.1, and (b) diagram of the 3D distance with reference to the 3D volume boundary.

Taking into account the considerations explained above, to simplify the presentation, in Algorithm 2, we present the computation of the distance inside the set A. The computation outside A is very similar. The algorithm starts by using the visited voxels nearest to the set boundary. In case we are in a visited voxel p, we explore its neighborhood. If a neighbor is an unvisited voxel, we update its distance value by adding the voxel distance from p to such a neighbor and we mark this neighbor as visited in the corresponding iteration. The global value for the increasing distance of the region ($MaxDis$) is updated according to the evolution of the distances in the neighborhood. This process is iteratively repeated until we reach the limit for the distance value Max_d^{in}, or the global region distance does not grow between iterations (because we have visited all the voxels). In case there are remaining unvisited voxels, they already have the appropriate values for the maximum distances, due to the fact that we have initialized the regions in the distance image to such values. Figure 1(a) depicts an example of one of the slices belonging to the result of Algorithm 2. The brighter the voxel, the higher the distance value. The neighborhood size is a parameter of the distance function. In the

experiments presented in this paper, we use a neighborhood of size 18. In fact, the neighborhood size has a strong influence on the quality of the approximation of the actual distance function with the computed one. We point out that the strategy of this iterative procedure to compute the 3D distance is similar to the well-known Dijkstra's algorithm [8] for finding the shortest paths between nodes in a graph.

In Algorithm 3 we present the procedure to smooth a 3D curve accordingly to the numerical scheme presented in the previous section. The flowchart of this algorithm is presented in Figure 3. The procedure has as input the original curve $\mathcal{C}$, which is modified during the smoothing process. Moreover, we include other parameters, such as the distance image (U), a weight to balance the smoothing and data terms (w), a tolerance to stop iterations based on criterion (10) (TOL), and the maximum number of iterations allowed ($MaxIter$). In the experiments presented in this paper we use 1000 iterations as such maximum. For curve reparameterization we fix to 1 the constant arc-length. We notice that actual coordinates of a point in the distance function image (U) depend on the voxel size ($v = (v_x, v_y, v_z)$). Therefore, to evaluate the distance function at a curve point $\mathcal{C}_i = (x_i, y_i, z_i)$ using the image U, we have to compute $U((x_i/v_x, y_i/v_y, z_i/v_z))$. To simplify the notation in the algorithm, we express this computation as $U(\mathcal{C}_i/v)$. Initially, we compute the minimum voxel size, which can be known from the distance image. In order to ensure the points of the input curve are equally spaced, a reparameterization is applied (as described in Algorithm 1). Then, the length of the curve is obtained, by computing the sum of the norm of the distance between each pair of points. Using this length, the input weight, and the values in the distance image along the curve, we compute an initial energy. The idea is to minimize this energy during the process of smoothing the curve. Besides, we also compute the maximum step size, based on the description given in Section 3, concerning the computation of the discretization step δ.

The smoothing procedure stops if we reach the maximum number of iterations, or the energy is not improved enough (i.e., the relative difference between the energies of two consecutive iterations is lower than the tolerance TOL - see Equation (10) -). For the new version of the curve, we use the same initial and final points. For the rest of the points of the curve, the iterative optimization procedure uses the normal (N), the curvature ($Curv$), and the gradient of the distance function ($\nabla U(\mathcal{C}_i/v)$) computed at each point. In this way, the new curve is obtained by following a gradient descending type method to compute a local minima of the energy, described in Equation (3). Moreover, in order to preserve the constant arch-length condition, this new curve is reparameterized by applying Algorithm 1.

In Figure 2, we include a flowchart of the main algorithm. After reading the input curve, we process differently the cases in which the algorithm is provided with an input 3D volume or not. In case no 3D volume is provided, we compute the maximum and minimum 3D coordinates of the curve, and normalize its points. Moreover, we also compute a 3D volume by generating an image in which the points along the curve are marked as inner to the 3D volume. In any case, the curve is reparameterized before the smoothing procedure. This procedure is also fed with a distance image (described in Algorithm 2). Finally, we generate an .obj file, which contains a 3D representation of the result. In all the cases this model includes the input and output curves. If a 3D volume is provided, it is also added to the model. The result of the smoothed curve is also written in the disk as an ASCII file.

In Table 1, we include a summary of the main parameters of the algorithm, with the name, type (input, output, and if it is optional [ ]), their default value, and a short description. We provide to the algorithm the input curve, which is stored in an ASCII file. This file contains a header with the number of points and their 3D coordinates. Moreover, we indicate a weight parameter to balance both energy terms (see Equation (3)), a tolerance to stop iterations, as well as the maximum number of iterations. Optionally, we can also provide an input 3D image (of type unsigned char in Analyze format) which contains as level set the 3D volume for which the curve is the centerline.

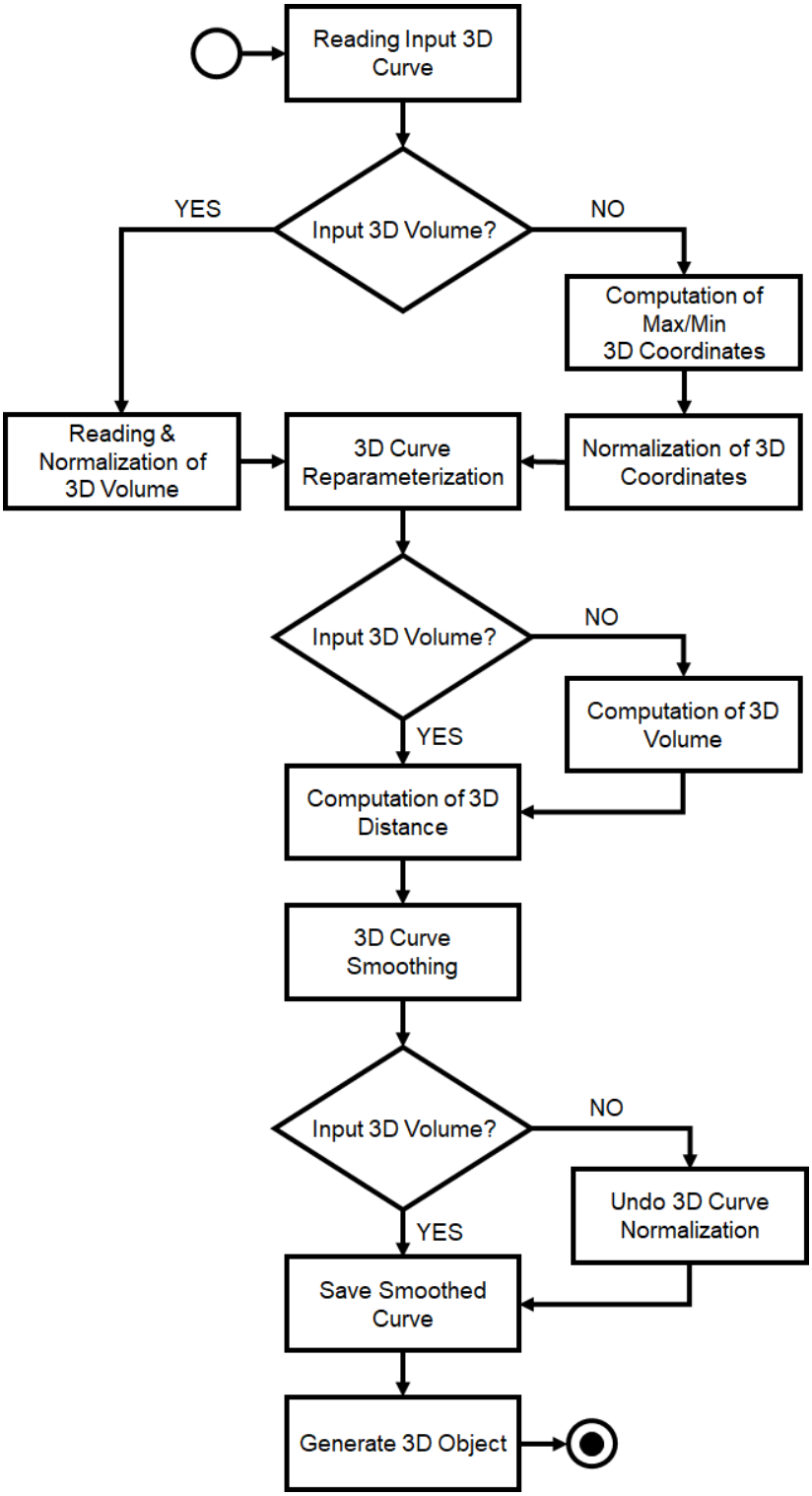


Figure 2: Flowchart of the main algorithm.

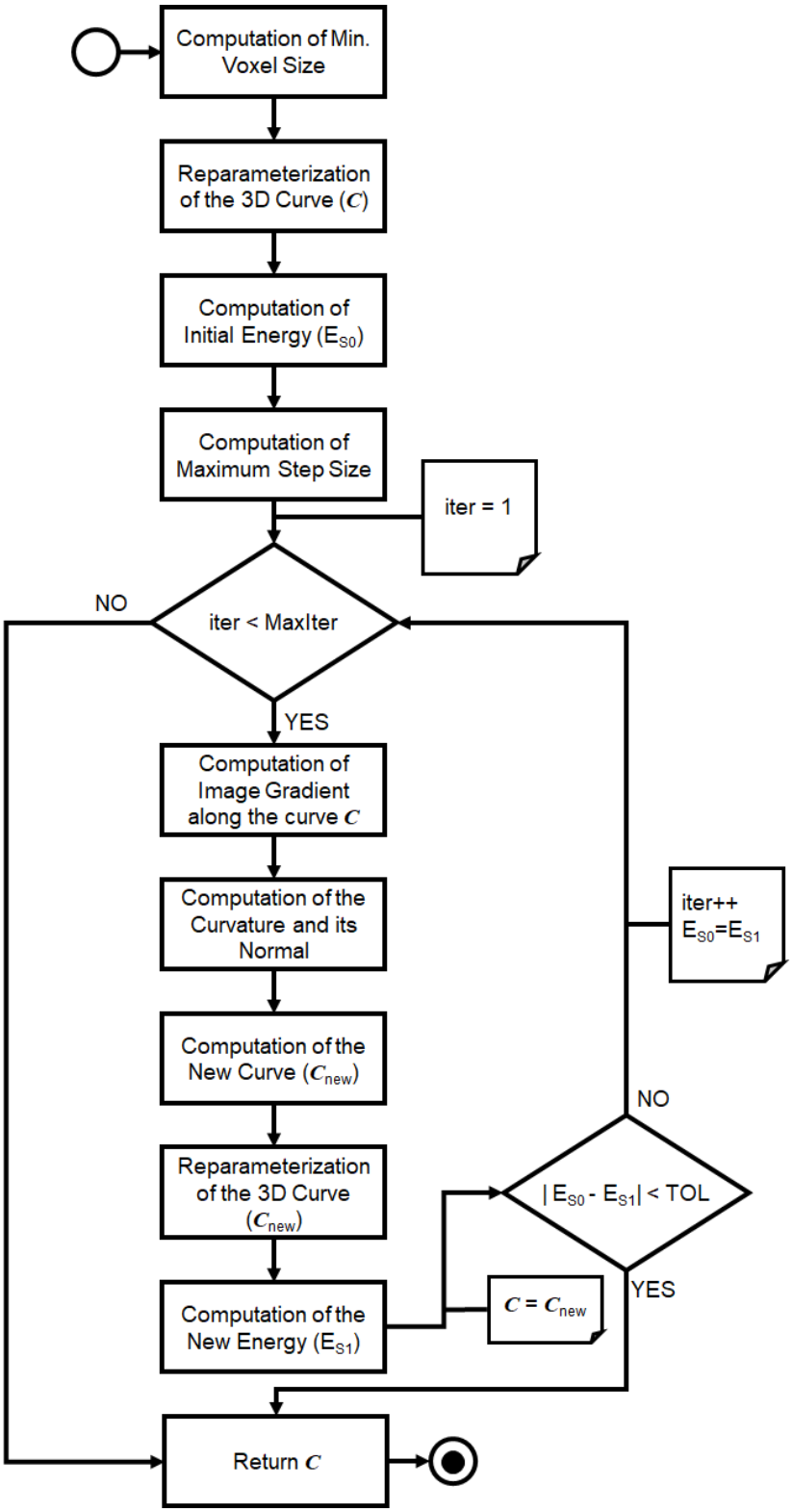


Figure 3: Flowchart of the algorithm to smooth 3D curves.

Algorithm 2: Function `distance_function3D`(I,Max_d^{in},Max_d^{out},NT)

input : I: input image where the set $A = \{p : I(p) > 0\}$
Max_d^{in}, Max_d^{out}: maximum distances to compute inside/outside the volume
NT: neighborhood type (6, 18, or 26 neighbors)
output : U: output image with the computed distance

/ In N and N_d we store the indexes and distances to the voxels in the neighborhood given by NT */*

$\forall p \in I : \begin{cases} U(p) \leftarrow Max_d^{in} & if \quad I(p) > 0 \\ U(p) \leftarrow -Max_d^{out} & otherwise \end{cases}$; *// Output initialization*

// Initial contour distance. V is a 3D image used to control visited voxels
$\forall p \in I : \ V(p) \leftarrow 10^8$;
for $(\forall p \in I)$ **do**
  if $(I(p) > 0)$ **then**
    for $(\forall k \in N)$ **do**
      if $(I(p + N(k)) \leq 0)$ **then**
        $U(p) \leftarrow 0.5 \cdot N_d(k)$;
        $V(p) \leftarrow 1$;
        break;
  else
    for $(\forall k \in N)$ **do**
      if $(I(p + N(k)) > 0)$ **then**
        $U(p) \leftarrow -0.5 \cdot N_d(k)$;
        $V(p) \leftarrow -1$;
        break;
$MaxDis \leftarrow 0$; $iter \leftarrow 0$;
while $(MaxDis < Max_d^{in})$ **do** *// region inside the set $I(p) > 0$*
  $iter \leftarrow iter + 1$; $TmpD \leftarrow MaxDis$;
  for $(\forall p \in I)$ **do**
    if $(I(p) \leq 0 \ OR \ V(p) \neq iter)$ **then**
      continue;
    for $(\forall k \in N)$ **do**
      if $(I(p + N(k)) > 0)$ **then**
        if $(V(p + N(k)) == 10^8)$ **then**
          $V(p + N(k)) \leftarrow iter + 1$;
          $U(p + N(k)) \leftarrow U(p) + N_d(k)$;
        else
          $temp = U(p) + N_d(k)$;
          if $(U(p + N(k)) > temp)$ **then**
            $U(p + N(k)) \leftarrow temp$;
        if $(U(p + N(k)) > MaxDis)$ **then**
          $MaxDis \leftarrow U(p + N(k))$;
  if $((TmpD == MaxDis))$ **then**
    break;
// The computation of the distance outside the set is very similar. See the code for more details.
return U;

As outcomes, the algorithm provides a new file with the curve smoothed, following the same format that the input one. Besides, an .obj file is generated, containing a comparison of the input and output curves. If a 3D volume is given as input, it is also included in the output .obj file.

Algorithm 3: Procedure `curve3D_smoothing`($\mathcal{C}$,U,w,$MaxIter$,TOL)

input : $\mathcal{C}$: input/output 3D curve
U: image with the signed distance function.
w: weight to balance the smoothing and the data terms
$MaxIter$: maximum number of iterations
TOL: tolerance to stop iterations

output : updated $\mathcal{C}$ with the smoothed version of the curve

// v is a 3D point with the voxel size (v_x, v_y, v_z) of the image given by U

$v_{min} \leftarrow \min\{v_x, v_y, v_z\}$; *// Minimum voxel size*

$\mathcal{C} \leftarrow$ `curve3D_reparameterization`($\mathcal{C}$,1.,10^{20}); *// Reparameterization of the input curve*

$l_0 \leftarrow \sum_{i=2}^{\chi(\mathcal{C})} \|\mathcal{C}_i - \mathcal{C}_{i-1}\|$; *// Initial length of the curve*

$E_{S0} \leftarrow w \cdot l_0 - \sum_{i=1}^{\chi(\mathcal{C})} U\left(\frac{\mathcal{C}_i}{v}\right)$; *// Initial energy*

$step \leftarrow 0.5 \cdot \min\{\frac{1}{\pi \cdot w}, \frac{v_{min}}{2}\}$; *// Maximum step size*

$iter \leftarrow 1$;

while $(iter < MaxIter)$ **do**

  $N(i) \leftarrow \frac{(\mathcal{C}^{i-1}+\mathcal{C}^{i+1})\cdot 0.5 - \mathcal{C}_i}{\|(\mathcal{C}_{i-1}+\mathcal{C}_{i+1})\cdot 0.5 - \mathcal{C}_i\|} : \forall i \in \{2, \ldots, \chi(\mathcal{C}) - 1\}$

; *// Computation of the normal, $N(i)$, to the curve.*

// Computation of the curvature, $Curv(i)$

$$Curv(i) \leftarrow \begin{cases} \frac{\pi - 2\arccos\overbrace{\left(\frac{\|(\mathcal{C}_{i-1}+\mathcal{C}_{i+1})\cdot 0.5 - \mathcal{C}_i\|}{v_{min}}\right)}^{t}}{v_{min}} & if \quad t \leq 1 \quad : \forall i \in \{2, \ldots, \chi(\mathcal{C}) - 1\}; \\ 0. & t > 1 \end{cases}$$

// Computation of the new curve

$\mathcal{C}_1^{new} \leftarrow \mathcal{C}_1$; $\mathcal{C}_{\chi(\mathcal{C})}^{new} \leftarrow \mathcal{C}_{\chi(\mathcal{C})}$; *// First and last points initialization*

/ To compute the new curve we use the current values, the image gradient along the current curve, as well as its normal and curvature */*

$\mathcal{C}_i^{new} \leftarrow \mathcal{C}_i + \nabla U(\mathcal{C}_i/v) \cdot step + N(i) \cdot Curv(i) \cdot w \cdot step : \forall i \in \{2, \ldots, \chi(\mathcal{C}) - 1\}$;

$\mathcal{C}^{new} \leftarrow$ `curve3D_reparameterization`($\mathcal{C}^{new}$,1.,10^{20}); *// Algorithm 1*

$l_1 \leftarrow \sum_{i=2}^{\chi(\mathcal{C}_{new})} \|\mathcal{C}_i^{new} - \mathcal{C}_{i-1}^{new}\|$; *// Length of the new curve*

$E_{S1} \leftarrow w \cdot l_1 - \sum_{i=1}^{\chi(\mathcal{C}^{new})} U\left(\frac{\mathcal{C}_i^{new}}{v}\right)$; *// New energy*

$\mathcal{C} \leftarrow \mathcal{C}^{new}$;

if $\left(\frac{|E_{S0} - E_{S1}|}{|E_{S0} + 10^{-20}|}\right) < TOL$ **then** *// Check energy evolution*

  break;

$iter \leftarrow iter + 1$;

$E_{S0} \leftarrow E_{S1}$;

5 Experimental Results

In order to assess the performance of the proposed algorithm, we have carried out experiments with synthetic and real data. In this way, we can evaluate quantitatively and qualitatively the ability of the proposal to smooth a 3D curve. In the following subsections we describe the results for synthetic and real data.

Parameter	I/O	Default Value	Description
$\mathcal{C}$	I	–	Name of the file with the input curve
w	I	1.	Weight parameter to balance the energy
TOL	I	0.00001	Tolerance to stop iterations
max_iter	I	1000	Maximum number of iterations
$\mathcal{C}_O$	O	–	Name of the file for the output curve
$.obj_{out}$	O	–	Output name for the .obj file with the 3D model comparing the input and output curves
$Seg.hdr$	[I]	–	3D image (unsigned char in Analyze format) including a 3D volume as level set for which the given 3D curve is the centerline

Table 1: Description of the main parameters of the 3D curve smoothing algorithm: parameter name, type (input, output, and if it is optional [ ]), default value, and a short description

5.1 Synthetic Data

For the synthetic case, we use the curve $\mathcal{C}^s(t) = ((6\pi - t)\cos(t), (6\pi - t)\sin(t), 3t)$ for $t \in [0, 6\pi]$ that we discretize using a constant arc-length parameterization with $h = 1$. To evaluate the performance of Algorithm 3, we use as input data the distance function to this curve, U, computed in a discrete way using the algorithm proposed in this paper. Moreover, as initial curve we provide $\mathcal{C}^i$, a noisy version of the curve $\mathcal{C}^s$, given by

$$\mathcal{C}^i = \mathcal{C}^s + (\mathcal{U}(-1,1), \mathcal{U}(-1,1), \mathcal{U}(-1,1))^T, \tag{18}$$

where $\mathcal{U}(-1,1)$ follows the uniform probability distribution in the interval $[-1,1]$. We use $\mathcal{C}$ as the initial guess for the scheme (7). Once the noise has been added, the curve is again reparameterized to preserve the constant arch-length condition. Figure 4(a) shows the original generated curve in black and the noisy one in red. In Figure 4(b) we include the result of the 3D curve smoothing for $w = 1$, with zooms in several regions, to illustrate the ability of the proposed solution to perform the smoothing. As observed, even in locations where the noisy curve is clearly separated from the ground truth, the method is able to smooth it.

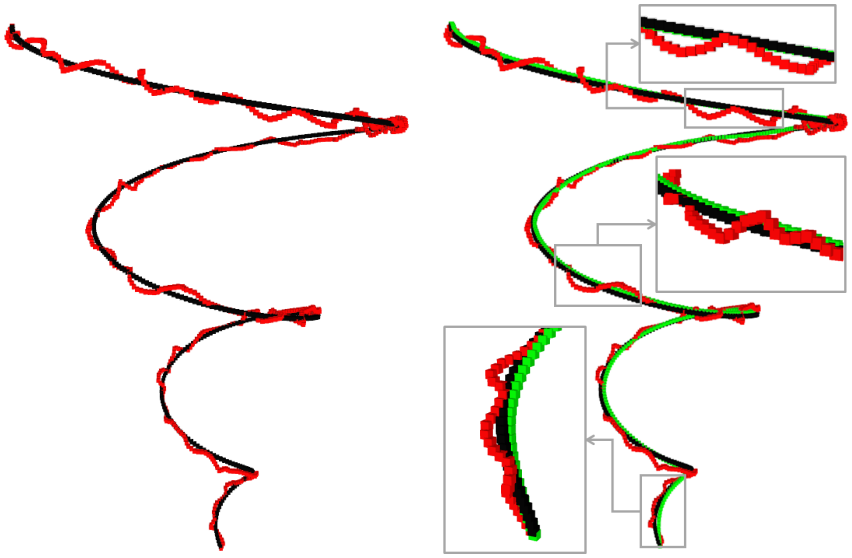
To measure the performance of the smoothing algorithm, we compute an approximation error of the distance between the ground-truth curve, $\mathcal{C}^s$, and the smoothed one in the iteration n, $\mathcal{C}^n$, by means of the expression

$$D(\mathcal{C}^s, \mathcal{C}^n) = \sqrt{\frac{1}{2\chi(\mathcal{C}^s)} \sum_{i=1}^{\chi(\mathcal{C}^s)} d^2(\mathcal{C}^s_i, \mathcal{C}^n) + \frac{1}{2\chi(\mathcal{C}^n)} \sum_{i=1}^{\chi(\mathcal{C}^n)} d^2(\mathcal{C}^n_i, \mathcal{C}^s)}, \tag{19}$$

where, for a given 3D point p and a curve $\mathcal{C}$, $d(p, \mathcal{C})$ is the Euclidean distance of p to the curve $\mathcal{C}$.

In Table 2 we show some results obtained by Algorithm 3 using different values of the smoothing parameter w. For each value of w, we include the number of iterations until the algorithm stopped, the final length, the energy, and the final approximation error given by (19) at the last iteration of the algorithm. Moreover, in Figure 5 we show the profile of the smoothed curves for the different values of w, and in Figures 6, 7, and 8 we present the evolution of the energy, length, and approximation error (Equation (19)) across the iterations of the algorithm.

As observed, the lowest values for the energy and error are obtained with small values of w. In fact, $w = 0$ provides the lowest value for both features. The reason is that in this experiment, we are using a smooth synthetic curve as ground truth and the 3D distance to this smooth curve in



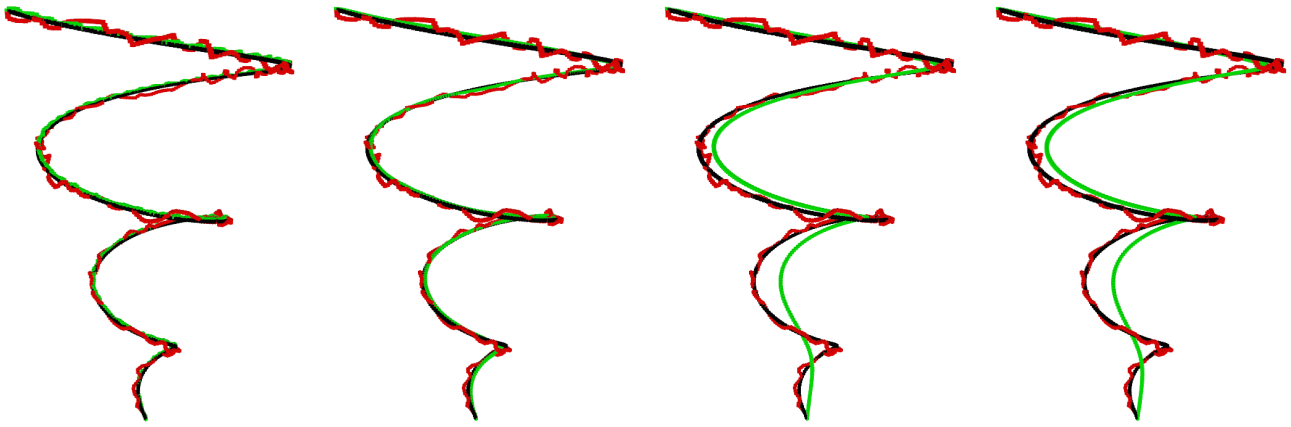
(a) (b)

Figure 4: Synthetic data used for the experiments: (a) the original generated curve (black) and curve with noise added (red), and (b) the result after applying the smoothing algorithm (green) with $w = 1$.

w	**N. Iterations**(10)	**Final Length**	**Final Energy** $E_S(\mathcal{C}_w^\infty)$ (3)	**Final Error** (19)
0	189	509.088	117.455	0.616 51
1	153	475.996	9682.823	0.787 999
10	1000	396.417	81 087.501	6.255 379
50	1000	388.503	390 499.807	6.897 268

Table 2: Quantitative results obtained for the synthetic curve with $w = 0, 1, 10, 50$, where $\mathcal{C}_w^\infty$ represents the final curve obtained after iterations stop.

the energy, so $w = 0$ provides a good result. In real applications, where we deal with noisy curves and the ground truth is not known, we need to use $w > 0$ to regularize the curves. An important issue in medical imaging is to compute the length of the curve (for instance, for stent implantation). If we study the evolution of the length for this synthetic curve, in the case of $w = 0$ we obtain a final length of 509.088 , whereas with $w = 1$ the curve length is 475.996 Considering that the length of the ground-truth is 487.608 , the result provided by $w = 1$ is closer than the one obtained with $w = 0$ (the differences between the ground-truth and both curves are 21.48 and 11.61 for $w = 0$ and $w = 1$ respectively). In other words, the regularization helps to compute more accurately the curve length and to avoid the curve zigzagging. Regarding higher values of w, the length of the curve is strongly reduced due to the fact that smooth curves tend to move away from the ground truth curve. This can be observed in Figures 5(c) and 5(d). As a consequence, as can be seen in Figure 8, this fact produces in both cases ($w = 10$ and $w = 50$) a higher error between the ground-truth and the smoothed curves. In particular we can observe that the values $w = 10$ and $w = 50$ are too high and the algorithm does not converge towards a curve close to the ground truth. So, in real applications we will never use such values for w. We have checked that the convergence rate of the algorithm depends on the parameter w. As observed in Figure 6, the value of the energy has a nice decreasing behavior across iterations. For small values of w (which are the ones of interest in practice), the convergence rate is quite good. For large values of w, we can observe in Figure 6 that the energy decay rate varies more smoothly and the algorithm stops after 1000 iterations because it considers that the curve is moving too far from the initial curve.



(a) $w = 0$ (b) $w = 1$ (c) $w = 10$ (d) $w = 50$

Figure 5: Comparison of the results for the smoothing of the synthetic curve for different values of w included in Table 2.

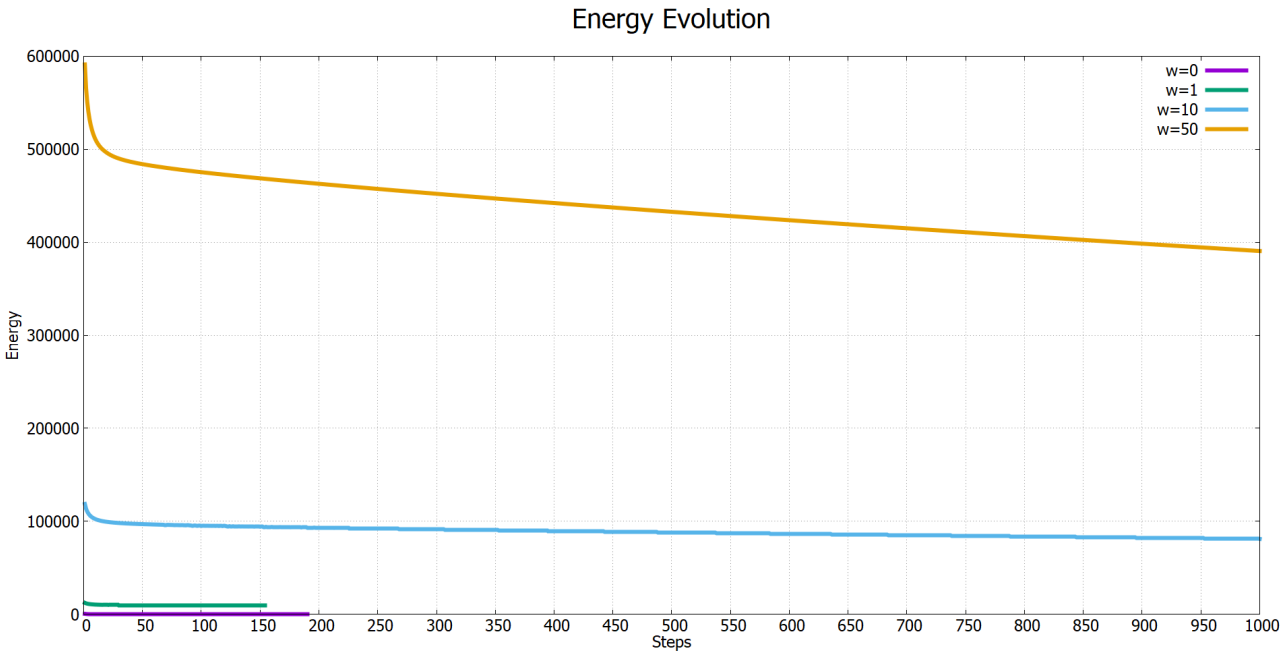


Figure 6: Evolution of $E_S(\mathcal{C}^n)$ for the synthetic curve with different values of w.

5.2 Real Data

For the experiments with real data, we use the result provided by [4]. In this article, the authors present a technique to automatically obtain the aortic lumen by tracking its cross-sections. As outcomes, this procedure provides a segmentation of the aortic lumen, that we use in our experiments as the 3D volume, and the aorta centerline, that we use as an initialization of the 3D volume medial axis. In Figure 9, we include the result for a real case, using $w = 1$, with zooms in a couple of regions. As observed, the proposed algorithm is able to smooth the curve, even in the cases in which the input is very irregular, as in the case depicted in the zoom at the bottom.

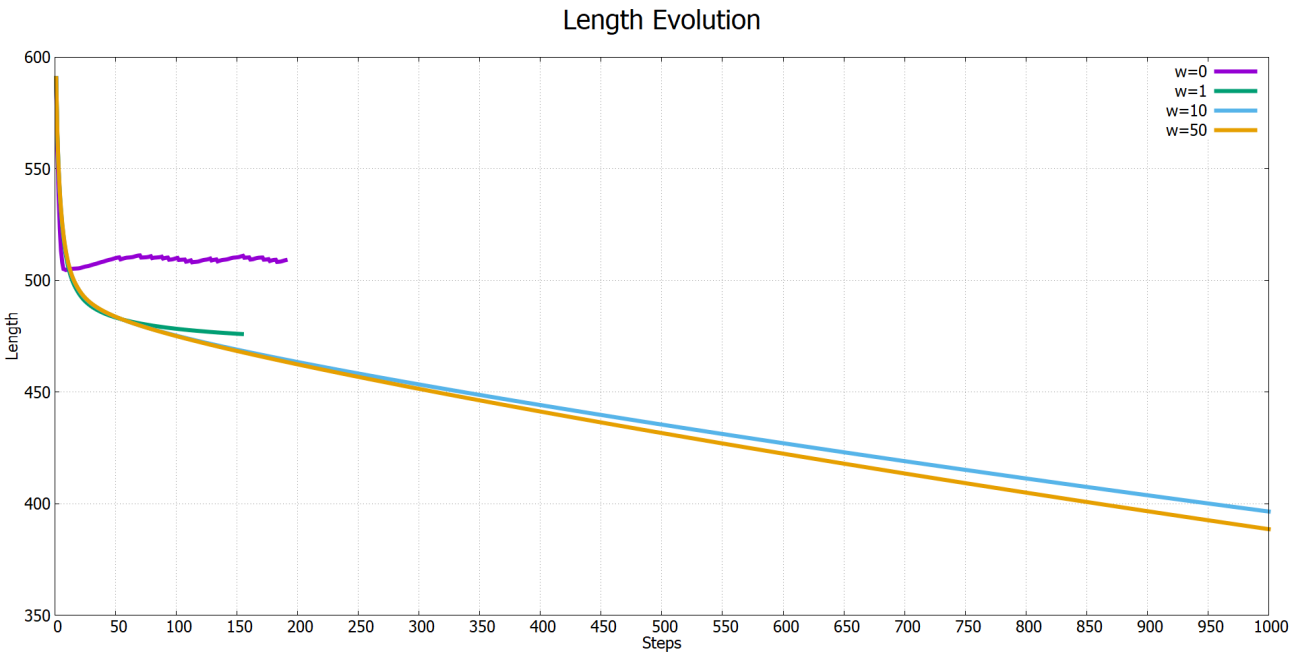


Figure 7: Evolution of curve length for the synthetic curve with different values of w.

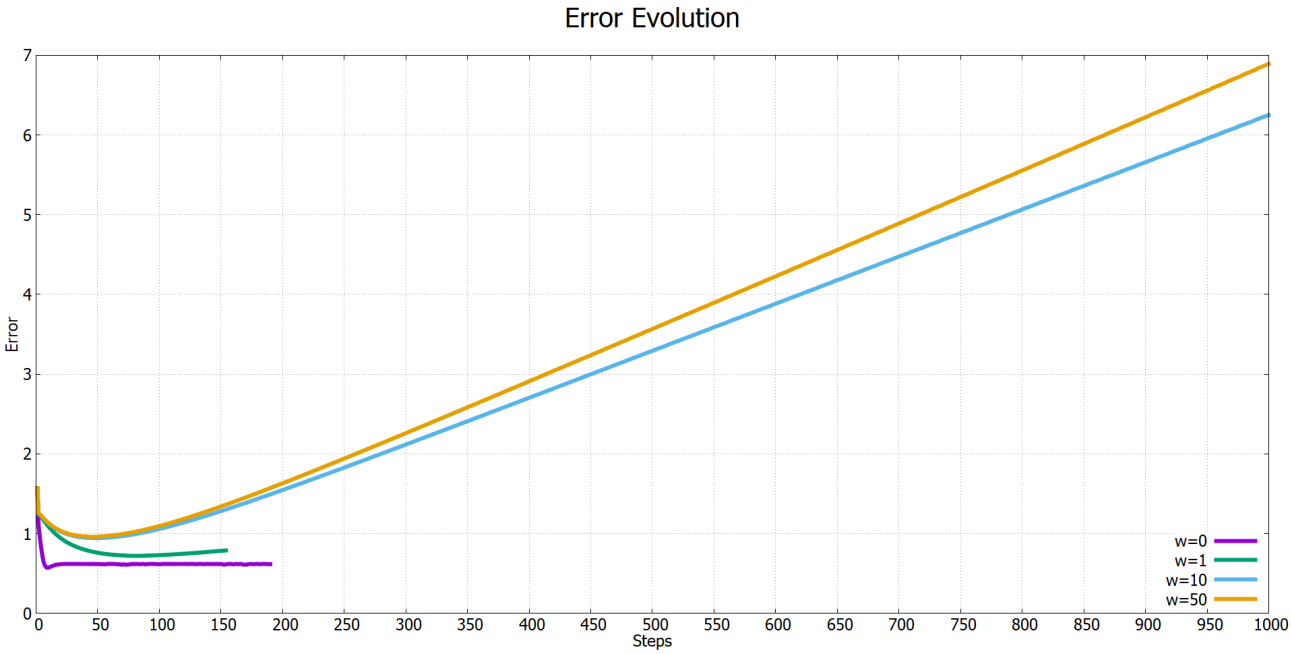


Figure 8: Evolution of error between the curve ground-truth and the smoothed one with different values of w.

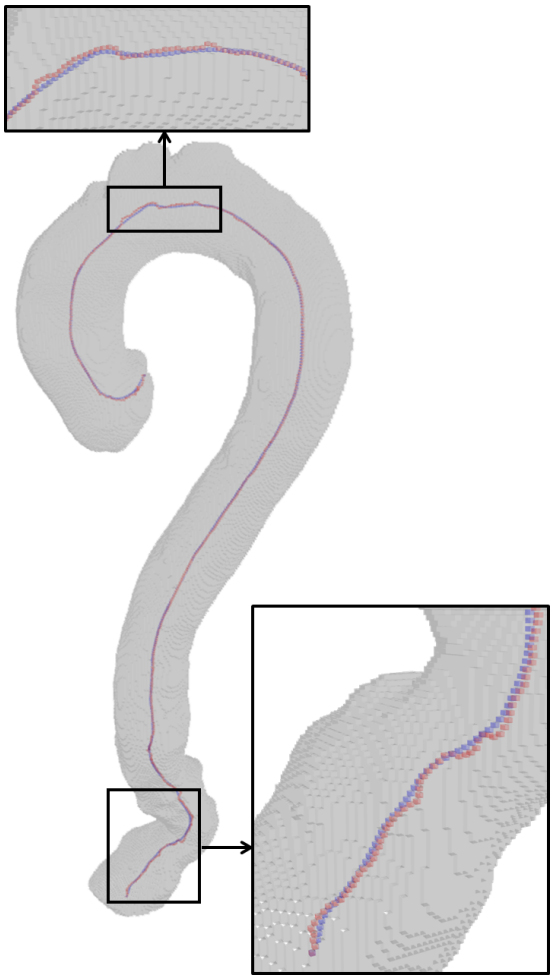


Figure 9: Result of applying the smoothing algorithm with $w = 1$ to a real case: original centerline (red), smoothed one (blue), and the segmentation (3D volume) in gray.

Computational Complexity We have measured the execution computational time required by the different parts of the algorithm. We used an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz processor. We check the computational time for the experiment on real data presented above. The average execution time of the curve reparameterization is $2.3 \cdot 10^{-5}$ seconds. The average execution time of the computation of the distance function is 7.64 seconds and the average time of each iteration of the curve smoothing algorithm is $1.76 \cdot 10^{-4}$. For the experiment with the real data and $w = 1$, the curve smoothing algorithm takes 76 iterations to converge. So globally, the algorithm is reasonably fast and most of the time is devoted to the distance to surface computation.

6 Conclusions

In this paper, we presented an algorithm for 3D curve smoothing based on the curve evolution equation (2). The algorithm can be used to smooth a 3D curve and optionally it can also address the smoothing of the medial axis of a 3D volume. We described the main steps of the algorithm using flowcharts and pseudocodes. We have also presented a variety of experiments to show the performance of the algorithm. These experiments show that the algorithm works well. The regularization effect depends strongly on the weight parameter w. The larger the value of w the stronger the regularization obtained.

References

[1] M. Alemán-Flores, D. Santana-Cedrés, L. Alvarez, A. Trujillo, L. Gómez, P.G. Tahoces, and J.M. Carreira, *Segmentation of the aorta using active contours with histogram-based descriptors*, in MICCAI Workshop: Intravascular Imaging and Computer Assisted Stenting and Large-Scale Annotation of Biomedical Data and Expert Label Synthesis, Springer International Publishing, 2018, pp. 28–35. https://doi.org/10.1007/978-3-030-01364-6_4.

[2] L. Alvarez, E. González, C. Cuenca, A. Trujillo, P.G. Tahoces, and J.M. Carreira, *Ellipse motion estimation using parametric snakes*, Journal of Mathematical Imaging and Vision, 60 (2018), pp. 1095–1110. https://doi.org/10.1007/s10851-018-0798-9.

[3] L. Alvarez, D. Santana-Cedrés, P.G. Tahoces, and J.M. Carreira, *Aorta centerline smoothing and registration using variational models*, in International Conference on Scale Space and Variational Methods in Computer Vision, Springer, July 2019, pp. 447–458. https://doi.org/10.1007/978-3-030-22368-7_35.

[4] L. Alvarez, A. Trujillo, C. Cuenca, E. González, J. Esclarín, L. Gomez, L. Mazorra, M. Alemán-Flores, P.G. Tahoces, and J.M. Carreira, *Tracking the Aortic Lumen Geometry by Optimizing the 3D Orientation of Its Cross-sections*, in Medical Image Computing and Computer-Assisted Intervention - MICCAI 2017, Cham, 2017, Springer International Publishing, pp. 174–181. https://doi.org/10.1007/978-3-319-66185-8_20.

[5] D. Chen, J-M. Mirebeau, and L.D. Cohen, *Global Minimum for Curvature Penalized Minimal Path Method*, in British Machine Vision Conference (BMVC), BMVA Press, September 2015, pp. 86.1–86.12. https://dx.doi.org/10.5244/C.29.86.

[6] D. Chen, J. Zhang, and L. D. Cohen, *Minimal Paths for Tubular Structure Segmentation With Coherence Penalty and Adaptive Anisotropy*, IEEE Transactions on Image Processing, 28 (2019), pp. 1271–1284. https://doi.org/10.1109/tip.2018.2874282.

[7] K-S. Chou and X-P. Zhu, *The Curve Shortening Problem*, Chapman and Hall/CRC, 2001. ISBN 9781584882138.

[8] E.W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[9] D. Lesage, E.D. Angelini, I. Bloch, and G. Funka-Lea, *A review of 3D vessel lumen segmentation techniques: Models, features and extraction schemes*, Medical Image Analysis, 13 (2009), pp. 819 – 845. https://doi.org/10.1016/j.media.2009.07.011.

[10] J-M. Mirebeau, *Fast-Marching Methods for Curvature Penalized Shortest Paths*, Journal of Mathematical Imaging and Vision, 60 (2018), pp. 784–815. https://doi.org/10.1007/s10851-017-0778-5.

[11] P.G. Tahoces, L. Alvarez, E. González, C. Cuenca, A. Trujillo, D. Santana-Cedrés, J. Esclarín, L. Gomez, L. Mazorra, M. Alemán-Flores, and J.M. Carreira, *Automatic estimation of the aortic lumen geometry by ellipse tracking*, International Journal of Computer Assisted Radiology and Surgery, 14 (2019), pp. 345–355. https://doi.org/10.1007/s11548-018-1861-0.

[12] P.G. Tahoces, D. Santana-Cedrés, L. Alvarez, M. Alemán-Flores, A. Trujillo, C. Cuenca, and J.M. Carreira, *Automatic detection of anatomical landmarks of the aorta in* CTA *images*, Medical & Biological Engineering & Computing, (2020). https://doi.org/10.1007/s11517-019-02110-x.

[13] S. Wang, L. Fu, Y. Yue, Y. Kang, and J. Liu, *Fast and Automatic Segmentation of Ascending Aorta in MSCT Volume Data*, in International Congress on Image and Signal Processing, Oct 2009, pp. 1–5. https://doi.org/10.1109/CISP.2009.5305569.