An Algorithm for 3D Curve Smoothing

Daniel Santana-Cedrés\textsuperscript{1}, Nelson Monzón\textsuperscript{2} and Luis Alvarez\textsuperscript{1}

\textsuperscript{1} CTIM (Centro de Tecnologías de la Imagen), University of Las Palmas de Gran Canaria, Spain
\textsuperscript{2} CMLA, École Normale Supérieure, Université Paris-Saclay, France,
(dsantana@ctim.es, monzon@cmla.ens-cachan.fr, lalvarez@ulpgc.es)

Abstract

In this article we present an application of variational techniques to the smoothing of 3D curves. We study 2 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 the minimization of a general variational model, 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\textsuperscript{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

\textsuperscript{1}https://ipolcore.ipol.im/demo/clientApp/demo.html?id=55555000008
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 discretize the variational model introduced in [3] and we generalize the method to the case of 3D curves which do not correspond to the medial axis of a 3D volume.

Let $C(s)$ be a 3D curve, $C : [0, |C|] \to R^3$, where $s$ represents the arc-length parameter and $|C|$ the length of the curve. Let $A$ be a 3D set. $A$ can be a 3D volume and then $C(s)$ is an approximation of the volumen medial axis (skeleton), or $A$ can be an initial curve, $C^0$, that is $A = \{C^0(s) : s \in [0, |C^0|]\}$. As in [3], the smoothing procedure is based in the minimization of the following variational model

$$C_w = \arg \min_{C : C(0) = p_0, C(|C|) = p_1} ES(C) \equiv \int_0^{|C|} d_{\partial A}(C(s))ds + w|C|, \quad (1)$$

where the parameter $w$ balances both energy terms and $d_{\partial A}(x)$ is the usual signed distance function to the set $A$ given by

$$d_{\partial A}(x) = \begin{cases} 
  d(x, \partial A) & \text{if } x \notin A, \\
  -d(x, \partial A) & \text{if } x \in A.
\end{cases} \quad (2)$$

Using the above energy model we aim to maximize the distance between the curve and the boundary of the set $A$, but keeping the curve smooth. The larger the value of $w$ the stronger the regularization effect.

In this paper we present, in details, an algorithm for the minimization of energy (1). The main issues we have to address to implement this algorithm are the following ones:

1. The computation of the signed distance function (2) for a given set 3D set $A$.
2. The implementation of a gradient descent type method to compute, by iterations, local minima of the energy (1).
3. An algorithm to reparametrize a 3D curve using a constant arc-length distance between the curve points. To avoid the curve degeneration, we use this algorithm in each gradient descent iteration to minimize energy (1). This step is important because otherwise, we have realized experimentally, that the curve can degenerate (points of the 3D curve collapsing, etc..) and the minimization procedure can fail.

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, introduced in [3], to find local minima of energy (1). In section 4, we study in details 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 [9], 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 showed. Automatic aorta segmentation algorithms for CT scans have been previously developed (see [8] for a survey). For instance, in [12], the authors proposed an iterative method based on building a 2-D region for segmenting the ascending aorta. In [4], a tracking procedure of the aorta centerline is presented. In [10], 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 and ellipse motion tracking. In [11], 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 minimize energy (1)

We present a basic numerical scheme introduced in [3] to minimize energy (1). In practice, a 3D curve $\mathcal{C}$ is given by a collection of 3D points $\{C^i\}_{i=1,..,\chi(\mathcal{C})}$, where $\chi(\mathcal{C})$ represents the number of points of the curve, $C^1 = p_0$ and $C^{\chi(\mathcal{C})} = p_1$. We assume that

$$\|C^i - C^{i-1}\| = h \quad \text{for all} \ i = 2, .., \chi(\mathcal{C}) - 1, \quad (3)$$

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

For the left part of energy (1), we consider in each point $C^i$ a gradient descent type scheme of the form

$$C^i_{n+1} = C^i_n - \delta \nabla d_{\partial A}(C^i_n). \quad (4)$$

To simplify the numerical scheme of the right part of energy (1) 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:

$$C^i_{n+1} = C^i_n + \delta k^i_n \mathcal{N}_n^i, \quad (5)$$

where $k^i_n$ represents an approximation to the 2D curvature and $\mathcal{N}_n^i$ the unit normal direction in $C^i_n$ of the curve restricted to the plane given by the points $C^{i-1}_n, C^i_n, C^{i+1}_n$. By combining both schemes and adding the weight $w$, we obtain the following minimization scheme for (1)

$$C^i_{n+1} = C^i_n - \delta \nabla d_{\partial A}(C^i_n) + \delta w k^i_n \mathcal{N}_n^i, \quad (6)$$

where $\delta > 0$ represents the discretization step. We point out that, after each iteration, we need to reparameterize the curve $C^i_{n+1}$ in order to preserve the constant arc-length condition (3). We compute the unit normal vector $\mathcal{N}_n^i$ as

$$\mathcal{N}_n^i = \begin{cases} \frac{C^{i-1}_n + C^{i+1}_n - C^i_n}{\|C^{i-1}_n + C^{i+1}_n - C^i_n\|} & \text{if } \frac{C^{i-1}_n + C^{i+1}_n}{2} \neq C^i_n, \\ \theta^i_n & \text{otherwise} \end{cases}, \quad (7)$$

and the curvature $k^i_n$ is approximated as the quotient between the angle, $\frac{\theta^i_n}{\theta^i_n} = \angle C^{i-1}_n C^i_n C^{i+1}_n$, of the vectors $C^{i-1}_n C^i_n$ and $C^i_n C^{i+1}_n$ and the arc-length $h$, that is

$$k^i_n = \frac{\theta^i_n}{h}. \quad (8)$$
We use as stopping criterion of the iterative scheme (6) the condition
\[
\frac{|E_S(C_n) - E_S(C_{n-1})|}{|E_S(C_{n-1})|} < TOL,
\]
(9)
where \(E_S(C)\) is the energy defined in (1). \(TOL > 0\) is a parameter to fix the stopping criterion of the scheme. In the experiments presented in this paper we use \(TOL = 10^{-5}\).

Scheme (6) is a basic approximation of a minimizer of energy (1) but it is not derived from the Euler-Lagrange equation of (1) due to the choice of the curvature shortening flow introduced in (5). Nevertheless, despite this theoretical limitation, in the experiments we show that scheme (6) behaves as a good minimizer of (1) (see Fig. 6).

**Automatic estimation of the discretization step \(\delta\)**

We can estimate \(\delta\) automatically using a two-step process: on the one hand, using as potential the normalized signed distance function, we have that \(\|\nabla d_{\partial A}(C_n^i)\| \leq 1\). Therefore, by imposing in the scheme (4) that
\[
\delta \leq \frac{h}{2},
\]
(10)
we obtain that the point \(C_{n+1}^i\) is closer to \(C_n^i\) than to \(C_{n-1}^i\) and \(C_{n+1}^i\). On the other hand, with respect to the curvature part we impose that
\[
\delta \omega_i^k \leq \left\| \frac{C_{n-1}^i + C_{n+1}^i}{2} - C_n^i \right\|,
\]
(11)
which means that the curvature flow never makes the point \(C_n^i\) to move to the other side of the segment \(C_{n-1}^iC_{n+1}^i\). Using a straightforward computation, we obtain that the above condition is equivalent to
\[
\delta \omega_i^\theta \leq h \cos \left( \frac{\pi - \theta_n}{2} \right).
\]
(12)
We observe that the function
\[
f(\theta) = \delta \omega_i^\theta - h \cos \left( \frac{\pi - \theta}{2} \right),
\]
(13)
satisfies that
\[
f(\pi) = \delta \omega_i^\pi - h \leq 0 \Leftrightarrow \delta \leq \frac{h^2}{w\pi},
\]
and we can easily check that if
\[
\delta \leq \frac{h^2}{w\pi},
\]
(14)
then \(f(\theta) \leq 0\) for any \(\theta \in [0, \pi]\) and then (12) is satisfied. Therefore, joining both estimations of \(\delta\) for each part of the numerical scheme, we can fix automatically \(\delta\) as
\[
\delta = \min \left\{ \frac{h}{2}, \frac{h^2}{w\pi} \right\}.
\]
(15)
In practice, we experienced that using this choice for \(\delta\) we obtain a stable numerical evolution of (6).
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, \(\{C_i\}_{i=1,...,\chi(C)}\), the constant distance, \(LS\), between consecutive points of the new curve, and the maximum length of the new curve. The objective is to return a new curve, \(\{C^m_i\}_{m=1,...,\chi(C)}\), in which the points are equally spaced by the indicated distance \(LS\), without overpass the maximum curve length allowed.

The algorithm starts assigning \(C^0 = C_i\). Then we go through the original curve by iterations. \(C^k_i\) represents the current position of the original curve and \(C^m_0\) the current position of the new curve. In the case \(\|C^k_i - C^m_0\| \leq LS\) we assign

\[
C^{m+1}_O = C^m_O + LS \frac{C^k_i - C^m_O}{\|C^k_i - C^m_O\|}
\]

and we update \(m\) \((m = m + 1)\). In the case \(\|C^k_i - C^m_0\| > LS\) we advance \(k\) \((k = k + 1)\) until \(\|C^k_i - C^m_0\| \leq LS\). In this case, since \(\|C^{k-1}_i - C^m_0\| < LS\), there exists \(\lambda \in (0, 1]\) such that

\[
\|C^{k-1}_i + \lambda \vec{v}_1 - C^m_0\| = LS
\]

where \(\vec{v}_1 = C^k_i - C^{k-1}_i\). An straightforward calculation yields to

\[
\lambda = -\frac{(\vec{v}_2, \vec{v}_1) + \sqrt{(\vec{v}_2, \vec{v}_1)^2 - (\vec{v}_1, \vec{v}_1)((\vec{v}_2, \vec{v}_2) - (LS)^2)}}{(\vec{v}_1, \vec{v}_1)}
\]

where \(\vec{v}_2 = C^{k-1}_i - C^m_0\). In this case we assign \(C^{m+1}_O = C^{k-1}_i + \lambda \vec{v}_1\) and update \(m\) \((m = m + 1)\). See the description of algorithm 1 for more details.

Algorithm 2 describes the method to compute the 3D distance inside and outside a set \(A\). In the case \(A\) is a 3D curve with no interior points, the computation is done only for points outside the curve. This function receives as parameters 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 distances inside and outside the set we want to compute, 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 \(A\) along 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 (fig. 1(b)). The only difference is the use of positive (inside) or negative
Algorithm 1: Function \texttt{curve3D_reparameterization}(C_I, LS, LC_{\text{max}})

\begin{verbatim}
input : \( C_I \) input curve (the number of points of \( C_I \) is \( \chi(C_I) \))
LS: constant length of the segments of the new curve
LC_{\text{max}}: maximum length of the new curve

output : \( C_O \): reparameterized output curve (initially empty)

\( C^0_O \leftarrow C^1_I; \) // Add the first point to the output curve
k \leftarrow 2;
\( m \leftarrow 1; \) // \( m \) represents in each iteration, \( \chi(C_O) \), that is the number of points of \( C_O \)

while (true) do
   if \( |C_O| \geq LC_{\text{max}} \) then // Check the size of the new curve
      break;
   \( \vec{v} \leftarrow C^k_I - C^m_O; \) // Vector between the current point of \( C_I \) and the last one added to \( C_O \)
   if \( \|\vec{v}\| \geq LS \) then
      \( C^{m+1}_O \leftarrow C^m_O + LS \cdot \vec{v} / \|\vec{v}\|; \)
      \( m \leftarrow m + 1; \)
      continue;
   end
   k \leftarrow k + 1;
   \end{verbatim}

// Advance \( k \) until \( \|\vec{v}\| \geq LS \)
while \((k < \chi(C_I) \text{ and } \|\vec{v}\| < LS)\) do
   \( \vec{v} \leftarrow C^k_I - C^m_O; \)
   if \( \|\vec{v}\| < LS \) then
      k \leftarrow k + 1;
   end
end

if \( (k = \chi(C_I)) \) then // Check the difference between the last points of both curves and add the last point of \( C_I \) if necessary
   break;
   */ In this case, \( \|C^0_O - C^{k-1}_I\| < LS \text{ and } \|C^0_O - C^k_I\| \geq LS \). We compute a point in the segment \( C^{k-1}_I \) with the prefixed distance \( LS \) from the last point \( C^m_O \). See the text for details. */
   \( \vec{v}_1 \leftarrow C^k_I - C^{k-1}_I; \)
   \( \vec{v}_2 \leftarrow C^k_I - C^m_O; \)
   \( \text{det} \leftarrow \sqrt{(\vec{v}_1 \cdot \vec{v}_2)^2 - (\vec{v}_1 \cdot \vec{v}_1)(\vec{v}_2 \cdot \vec{v}_2) - (LS)^2); \)
   \( C^{m+1}_O \leftarrow C^{k-1}_I + \frac{-(\vec{v}_1 \cdot \vec{v}_2) + \text{det}}{\|\vec{v}_1\|^2} \vec{v}_1; \)
   \( m \leftarrow m + 1; \)
end

// Check the difference between the last points of both curves and add the last point of \( C_I \) if necessary
if \( (\|C^O_{\text{last}} - C^m_O\| > 0) \) then
\( C^{m+1}_O \leftarrow C^\chi(C_I); \)
end

return \( C_O; \)
\end{verbatim}
remaining unvisited voxels, they already have the appropriate value for the maximum distances, due to we have initialized the regions in the distance image to such values. Fig. 1(a) depicts an example of one of the slices belonging to the result of the algorithm 2. The brighter the voxel, the higher the distance value.

![Image](image)

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.

In algorithm 3 we present the procedure to smooth a 3D curve accordingly with the numerical scheme presented in the previous section. The flow of this algorithm can be seen on the flowchart in figure 3. The procedure has as input the original curve \( C \), which is modified during the smoothing process. Moreover, we include other parameters, such as the distance image, a weight to balance the smoothing and data terms, a tolerance to stop iterations based on criterium (9) and the maximum number of iterations allowed. 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 in a curve points \( C^i = (x_i, y_i, z_i) \) 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 \( U(C^i/v) \). The flowchart of this algorithm is presented in fig. 3. 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 \( E_0 \). 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 \( \delta \).

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 - eq. 9). 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 to the curve \( N \), its curvature \( Curv \), and the gradient of the distance function along the curve \( \nabla U(C^i/v) \). 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 (1). Moreover, in order to preserve the constant arch-length condition, this new curve is reparameterized by applying algorithm 1.

In fig. 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 coordinates, and normalize the
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 $NT$ */

\[ \forall p \in I : \begin{cases} U(p) \leftarrow Max_{d}^{in} & \text{if } I(p) > 0 \\ U(p) \leftarrow Max_{d}^{out} & \text{otherwise} \end{cases} \] // 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 (\( \forall p \in I \))

if (\( I(p) > 0 \))

for (\( \forall k \in N \))

if (\( I(p + N(k)) \leq 0 \))

\[ U(p) \leftarrow 0.5 \cdot N_d(k); \]
\[ V(p) \leftarrow 1; \]
break;

else

for (\( \forall k \in N \))

if (\( I(p + N(k)) > 0 \))

\[ U(p) \leftarrow -0.5 \cdot N_d(k); \]
\[ V(p) \leftarrow -1; \]
break;

\[ MaxDis \leftarrow 0; \]
\[ iter \leftarrow 0; \]
while (\( MaxDis < Max_{d}^{in} \))

\[ \forall p \in I \]

if (\( I(p) \leq 0 \) OR \( V(p) \neq iter \))

continue;

for (\( \forall k \in N \))

if (\( I(p + N(k)) > 0 \))

if (\( V(p + N(k)) == 10^8 \))

\[ V(p + N(k)) \leftarrow iter + 1; \]
\[ U(p + N(k)) \leftarrow U(p) + N_d(k); \]

else

\[ temp = U(p) + N_d(k); \]
if (\( U(p + N(k)) > temp \))

\[ U(p + N(k)) \leftarrow temp; \]

if (\( U(p + N(k)) > MaxDis \))

\[ MaxDis \leftarrow U(p + N(k)); \]

if (\( (TmpD == MaxDis) \))

break;

// The computation of the distance outside the set is very similar. See the code for more details.

return $U$;

points of the curve. 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. Anyway, 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 wavefront model, which contains a 3D representation of
An Algorithm for 3D Curve Smoothing

**Algorithm 3**: Procedure `curve3D_smoothing(C, U, w, MaxIter, TOL)`

**Input**: `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 `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`

```plaintext
v_min ← min{v_x, v_y, v_z};  // Minimum voxel size
```

```plaintext
C ← curve3D_reparameterization(C, 1..10^20);  // Reparameterization of the input curve
```

```plaintext
l_0 ← ∑_{i=2}^{χ(C)} ||C_i - C_{i-1}||;  // Initial length of the curve
```

```plaintext
E_0 ← w · l_0 - ∑_{i=1}^{χ(C)} U(C_i / v);  // Initial energy
```

```plaintext
step ← 0.5 · min{1 / w, v_min / 2};  // Maximum step size
```

```plaintext
iter ← 1;
```

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

```plaintext
N ← ∀i ∈ [2, .., χ(C) - 1]: (C_i - C_{i-1}) · 0.5 · C_i
```

**// Curvature**

```plaintext
Curv ← ∀i ∈ [2, .., χ(C) - 1]: \[
\pi/2 \arccos \left( \frac{\left\langle (C_i - C_{i+1}) · 0.5 · C_i, \\cdot \right\rangle}{v_{\text{min}}} \right) \text{ if } t \leq 1; \\
0. \text{ otherwise}
\]
```

**// Computation of the new curve**

```plaintext
C_{new}^{1} ← C_1;  // First and last points initialization
```

```plaintext
C_{new}/C ← C_χ(C);
```

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

```plaintext
C_{new}^{2,χ(C)-1} ← ∀i ∈ [2, .., χ(C) - 1]: C_i + ∇U(C_i / v) · step + N_k · Curv_k · w · step;
```

```plaintext
C_{new} ← curve3D_reparameterization(C_{new}, 1..10^20);  // Algorithm 1
```

```plaintext
l_1 ← ∑_{i=2}^{χ(C)} ||C_{new} - C_{i-1}||;  // Length of the new curve
```

```plaintext
E_1 ← w · l_1 - ∑_{i=1}^{χ(C)} U(C_{new} / v);  // New energy
```

```plaintext
C ← C_{new};
```

**if** (|E_0 - E_1| / |E_0 + 10^{-20}|) < TOL **then**

```plaintext
break;
```

```plaintext
iter ← iter + 1;
```

```plaintext
E_0 ← E_1;
```

the result. In all the cases this model includes the input and output curves. Besides, 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 also indicate a weight parameter to balance the energy, a tolerance to stop iterations, as well as its maximum number. 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.

```plaintext
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>input</code></td>
<td><code>Curve3D</code></td>
<td><code>C</code></td>
<td>input curve</td>
</tr>
<tr>
<td><code>output</code></td>
<td><code>Curve3D</code></td>
<td><code>C</code></td>
<td>output curve</td>
</tr>
<tr>
<td><code>w</code></td>
<td>real</td>
<td><code>0.5</code></td>
<td>weight to balance smoothing and data terms</td>
</tr>
<tr>
<td><code>MaxIter</code></td>
<td>int</td>
<td><code>100</code></td>
<td>maximum number of iterations</td>
</tr>
<tr>
<td><code>TOL</code></td>
<td>real</td>
<td><code>10^{-3}</code></td>
<td>tolerance to stop iterations</td>
</tr>
</tbody>
</table>
```
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.
5 Experimental Results

In order to assess the performance of the proposed algorithm, we have carried out experiments in 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
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>I/O</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>I</td>
<td>–</td>
<td>Input curve</td>
</tr>
<tr>
<td>$w$</td>
<td>I</td>
<td>1.0</td>
<td>Weight parameter to balance the energy</td>
</tr>
<tr>
<td>$tol$</td>
<td>I</td>
<td>0.00001</td>
<td>Tolerance to stop iterations</td>
</tr>
<tr>
<td>$max_iter$</td>
<td>I</td>
<td>1000</td>
<td>Maximum number of iterations</td>
</tr>
<tr>
<td>$C_{out}$</td>
<td>O</td>
<td>–</td>
<td>Output curve</td>
</tr>
<tr>
<td>$obj_{out}$</td>
<td>O</td>
<td>–</td>
<td>Output name for the obj file with the 3D model</td>
</tr>
<tr>
<td>$Seg.hdr$</td>
<td>I</td>
<td>–</td>
<td>3D image (unsigned char in Analyze format) including a 3D volume as level set for which the given 3D curve is the centerline</td>
</tr>
</tbody>
</table>

and real data.

5.1 Synthetic Data

For the synthetic case, we use the curve $C_{s}(t) = ((6\pi - t) \cos (t), (6\pi - t) \sin (t), 3t)$ for $t \in [0, 6\pi]$. To evaluate the performance of algorithm 3 we use as input data the distance function, $U$, to this curve and as initial curve, $C$, a noisy version of the curve, $C_{s}$, given by

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

where $\mathcal{U}(-1, 1)$ follows the uniform probability distribution in the interval $[-1, 1]$. We use $C$ as the initial guess for the scheme (6). Once the noise has been added, the curve is again reparameterized to preserve the constant arch-length condition. Fig. 4(a) shows the original generated curve in black and the noisy one in red. In fig. 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, $C_{s}$, and the smoothed one in the iteration $n$, $C_{n}$, by means of the expression:

$$d(C_{s}, C_{n}) = \sqrt{\frac{1}{2\chi(C_{s})} \sum_{i=1}^{\chi(C_{s})} d^2(C_{s}, C_{n}) + \frac{1}{2\chi(C_{n})} \sum_{i=1}^{\chi(C_{n})} d^2(C_{n}, C_{s})},$$

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

In table 2 we show some results obtained by the 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 (17) at the last iteration of the algorithm. Moreover, in Fig. 5 we show the profile of the smoothed curves for the different values of $w$, and in Figs. 6, 7, and 8 we present the evolution of the energy, length, and approximation error (eq. (17)) 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. However, if we study the evolution of the
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 \( w = 1 \).

Table 2: Quantitative results obtained for the synthetic curve with \( w = 0, 1, 10, 50 \).

<table>
<thead>
<tr>
<th>( w )</th>
<th>N. Iterations(9)</th>
<th>Final Length</th>
<th>Final Energy ( E_S(C_w) ) (1)</th>
<th>Final Error (17)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>189</td>
<td>509.088</td>
<td>117.455</td>
<td>0.61651</td>
</tr>
<tr>
<td>1</td>
<td>153</td>
<td>475.996</td>
<td>9682.823</td>
<td>0.787999</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>396.417</td>
<td>81087.501</td>
<td>6.255379</td>
</tr>
<tr>
<td>50</td>
<td>1000</td>
<td>388.503</td>
<td>390499.807</td>
<td>6.897268</td>
</tr>
</tbody>
</table>

length, with \( w = 0 \) 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). Regarding higher values of \( w \), the length of the curve is strongly reduced because of the smooth curves tends to move away from the curve ground truth. This can be observed in figs. 5(c) and 5(d). As a consequence, as can be seen in fig. 8, this fact produces in both cases \( (w = 10 \) and \( w = 50 \)) a higher error between the ground-truth and the smoothed curves.

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 fig. 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 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 \( E_S(C^n) \) for the phantom curve with different values of \( w \).

6 Conclusions

In this paper we presented an algorithm for 3D curve smoothing based on the variational model introduced in [3]. 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 and the results are consistent with the underlying variational model. In particular the regularization effect depends strongly on the weight parameter \( w \). The larger the value of \( w \) the stronger the regularization obtained.
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.
References


