Piecewise 3D Euler spirals

David Ben-Haim* Technion

Gur Harary† Technion

Ayellet Tal‡ Technion

ABSTRACT

3D Euler spirals are visually pleasing, due to their property of having their curvature and their torsion change linearly with arc-length. This paper presents a novel algorithm for fitting piecewise 3D Euler spirals to 3D curves with G^2 continuity and torsion continuity. The algorithm can also handle sharp corners. Our piecewise representation is invariant to similarity transformations and it is close to the input curves up to an error tolerance.

1. INTRODUCTION

3D curves convey important information about the shape of objects. They are used as shape features, as strokes for sketch-based interaction, in non-photo realistic rendering, and in a variety of mesh analysis algorithms [2, 4, 10].

Traditionally, complex curves are represented in a piecewise manner. Usually, the primitives utilized are parametric polynomials defined using a set of geometric constraints, such as Bezier curves or NURBS [6]. These curves exhibit many attractive properties for curve design. However, unless imposed during the curve creation, fairness, which is often identified with how smoothly the curve bends [9], might not be maintained.

A planar Euler spiral is characterized by a linearly-evolving curvature along the curve [8, 15]. Several papers have addressed piecewise 2D Euler spirals. 2D Euler-spiral segments are fitted to geometric points and curvature constraints in [17, 18, 19]. They are also used for constructing splines [15, 20]. McCrae and Singh [14, 13] present methods for fitting a piecewise Euler spiral to a sketch. Their method is suitable for conceptual design applications, where aesthetic fairness is important. Baran et al. [1] present a different method, which better fits the input curve.

3D Euler spirals received less attention. They were recently introduced in [7] as the curves that have both a linearly-varying curvature and a linearly-varying torsion w.r.t the arc-length. Similarly to their 2D counterparts, these curves satisfy some desirable properties, including invariance to similarity transformations, symmetry, extensibility (i.e., the curve is refinable), smoothness, and roundness (i.e., if the boundary conditions lie on a circle, then the curve is a circle).

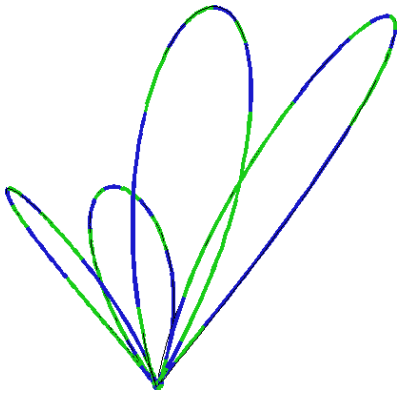


Figure 1: A piecewise 3D Euler spiral representation (40 segments). Each petal leaf is represented similarly, since Euler spirals are invariant to similarity transformations.

This paper addresses the problem of fitting piecewise 3D Euler spirals to general 3D curves, as illustrated in Figure 1. This representation is G^2-continuous, torsion-continuous, and manages to smooth noisy curves. It can thus be an appealing alternative to splines for certain applications.

Given a 3D curve, our algorithm approximates it by a small number of curves, each having a linearly-varying curvature and a linearly-varying torsion. This is performed by utilizing a dynamic programming approach that fits a connected set of line segments to the curvature values and to the torsion values of the input curve.

The rest of the paper is structured as follows. Section 2 describes our algorithm for constructing a piecewise 3D Euler spiral for representing a given 3D curve. Section 3 shows some results. We conclude in Section 4.

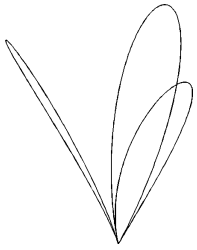
2. THE CONSTRUCTION ALGORITHM

The 3D Euler spiral is defined as the curve having both its curvature κ and its torsion τ evolve linearly along the curve [7]. That is to say, there exist constants $\kappa_0, \tau_0, \gamma, \delta \in \mathbb{R}$

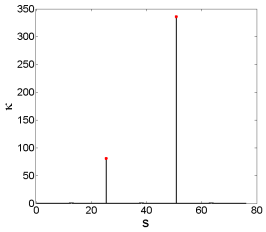
*davidbh@tx.technion.ac.il

†gur@tx.technion.ac.il

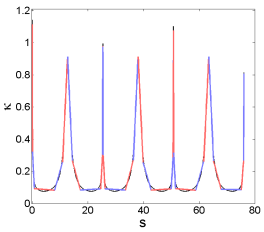
‡ayellet@ee.technion.ac.il



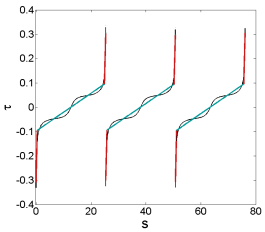
(a) Input



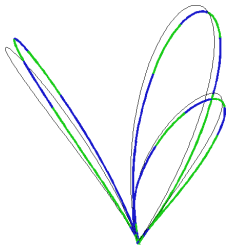
(b) Step 1: Corner detection (red)



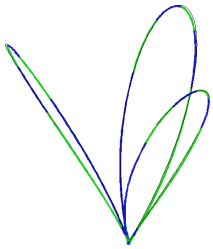
(c) Step 3(1): Curvature linear fitting



(d) Step 3(2): Torsion linear fitting



(e) Step 4: Euler-spiral construction



(f) Step 5: Applying rigid transformation

Figure 2: Algorithm outline

for which $\kappa(s) = \kappa_0 + \gamma s$ and $\tau(s) = \tau_0 + \delta s$, for $0 \leq s \leq L$ (L is the curve length). The definition is in the form of a set of differential equations, which requires the following initial conditions: a point $\mathbf{x}_0$ on the curve, a tangent $\vec{T}_0$ at $\mathbf{x}_0$, and a normal $\vec{N}_0$. This section describes the construction algorithm of piecewise 3D Euler spirals.

Given a 3D curve, our goal is to find a sequence of 3D Euler spirals that best fit the input curve. We require that

1. The number of spirals will be small.
2. The deviation of the resulting piecewise curve from the input curve will be small.
3. The representation will be robust to noise in the input curve, which is typical of curves detected by algorithms for edge detection on surfaces.
4. The piecewise curve will be G^2-continuous and torsion-continuous.

The key idea of the algorithm is to utilize the characterizing property of 3D Euler spirals – having a linear curvature and a linear torsion. Thus, the sequence of spirals is determined by fitting linear segments to the curvature and to the torsion values along the curve. Our algorithm is inspired by the work in 2D of McCrae et al. [14] and extends it to 3D, improving the robustness to noisy data. In particular, in the algorithm described below Steps 1,2,4 and the optimization in Step 3, are inherently different.

Our algorithm is outlined in Algorithm 1, illustrated in Figure 2, and explained thereafter. We assume that the curve is given as an ordered set of points, which is typical of most applications. If it is given analytically, we sample it. The algorithm consists of five steps. First, the curve is decomposed into sub-curves at the sharp corners. This is done in order to prevent biases at these points in subsequent steps. Next, the curvature and the torsion of each sub-curve are estimated. Having computed these values, we fit them to linear segments, such that the number of segments is minimized, while keeping the deviation small. These linear segments are constrained to connect at their end-points in order to yield G^2-continuity and torsion-continuity of the final piecewise curve. The fourth step utilizes the segments found previously to construct a piecewise 3D Euler spiral curve. Finally, a rigid transformation is applied to the resulting piecewise curve to improve the fitting, if needed.

Algorithm 1 Curve fitting algorithm

1: Corner detection
2: Curvature & torsion estimation
3: Line-segment fitting to the curvature & torsion using a dynamic programming algorithm
4: Piecewise 3D Euler spiral construction
5: Improvement by applying a rigid transformation

1. Corner detection: Sharp corners are points of discontinuity of the tangents, thus these are the points for which the curvature and the torsion are undefined. In practice, given a set of points, the curvature and the torsion can always be estimated, however, at corners their absolute values are very high relatively to the values of their neighborhoods. They appear as large spikes in curvature or torsion space. For instance, in Figure 2(b) the values of the curvature at the corners are as high as 340, while the values of the curvature at the other points range between 0 to 1.1 (Figure 2(c)). If the corners are not detected, our curvature-fitting algorithm will fit two linear segments around the pick, instead of a single point of discontinuity. This might result in erroneous spiral fitting, since most of the error would be contributed to representing the picks, while the rest of the curve would be represented by only a few spirals. Corner detection allows us to process each sub-curves adjacent to a corner separately.

Our corner detection algorithm aims at finding large deviations between the tangents of subsequent points. The input is the tangents at all the sample points on the curve and the output is a list of corner points. A simple thresholding on the angle between the tangents will yield insufficient results, since noise looks like corners. Instead, we find the angles between adjacent tangents, after smoothing the tangents in the spirit of bilateral filtering [21]. Recall that bilateral filtering aims at edge-preserving noise smoothing. Similarly,

we aim at corner-preserving smoothing.

Our algorithm consists of three steps: First, to assure robustness to noise we smooth the tangents extracted from the curve in a corner-preserving fashion. Then, we calculate the angle between every pair of subsequent tangents. Finally, we threshold the angles to determine which points are corners. We elaborate below.

In the first step each tangent is replaced by a weighted average of its neighbors. Formally, the smoothed tangent $\vec{\hat{T}}_j$ at the j^{th} point is a weighted average of the tangents of its neighbors (in practice, we consider $n = 15$ neighbors):

$$\vec{\hat{T}}_j = \frac{\sum_{i=j-n}^{i=j+n} \omega_{ij} \cdot \vec{T}_{ij}}{\sum_{i=j-n}^{i=j+n} \omega_{ij}}.$$

The tangents are then normalized.

The question is how to set the weights, such that they would reflect two aspects: The first is the proximity of the neighbor to the given point – closer neighbors should get larger weights. The second is the similarity of the neighbor's tangent to the given point's tangent – neighbors with similar tangents should get larger weights.

We consider ω_{ij} as the product of the two weights described below:

$$\omega_{ij} = \alpha_{ij} \cdot \beta_{ij}.$$

Let s_j and $\vec{T}_j$ be the arc-length and the tangent of the j^{th} point, and s_{ij} and $\vec{T}_{ij}$ be the arc-length and the tangent of its i^{th} neighbor. Then,

$$\alpha_{ij} = exp\left(-\frac{(s_{ij} - s_j)^2}{2\sigma_d}\right), \quad \beta_{ij} = \max(\vec{T}_{ij} \cdot \vec{T}_j, 0).$$

α_{ij} considers spatial proximity, whereas β_{ij} considers proximity of the values. Hence, the more different the directions and the farther the points, the lower the weight.

In the second step of our algorithm the angle between every pair of subsequent smoothed tangents is calculated as: $\theta = \arccos\left(\vec{\hat{T}}_{j-1} \cdot \vec{\hat{T}}_j\right)$.

The final step of the algorithm is thresholding θ. A point j having a large θ is marked as a corner (in practice $\theta > 45^\circ$).

2. Curvature & torsion estimation: There exist various methods that estimate both the curvature and the torsion of spatial curves [3, 11, 12]. In this work we follow the *independent coordinates* method proposed by Lewiner et al. [12], which is shown to be robust to noise. Briefly, a cubic parametric curve is fitted to the coordinates of each sample point. For example, for the x coordinate we get:

$$\hat{x}(s) = x_0 + x_0' \cdot s + \frac{1}{2}x_0'' \cdot s^2 + \frac{1}{6}x_0''' \cdot s^3,$$

where x_0 is the position of the sample point, x_0', x_0'', x_0''' are the 1^{st}, 2^{nd}, and 3^{rd} derivatives respectively, and s is the arc-length. A similar procedure is performed for the y and z coordinates.

The parameters x_0', x_0'', x_0''' are found by minimizing a weighted square error for each sample point:

$$E_x(x_0', x_0'', x_0''') = \sum_{i=-n}^{i=n} w_i(x_i - (x_0' \cdot s + \frac{1}{2}x_0'' \cdot s^2 + \frac{1}{6}x_0''' \cdot s^3))^2,$$

where x_i is the position of a neighboring sample point w.r.t x_0 and w_i is its weight. In our implementation $w_i = 1, \forall i$.

Given these derivatives, the curvature and the torsion are computed according to their definition [5] (where $r = (x, y, z)$):

$$\kappa(s) = \frac{|r' \times r''|}{\|r'\|^3}, \quad \tau(s) = -\frac{(r' \times r'') \cdot r'''}{\|r' \times r''\|^2}.$$

3. Line-segment fitting to the curvature & torsion: The goal of this step is to fit a small number of line segments to the curve of the curvature (torsion). We will first discuss the basic algorithm and then show how to enhance it so as to guarantee G^2-continuity and torsion-continuity. Since our algorithm is performed similarly for the curvature and for the torsion, we will present it only for the curvature.

The basic algorithm follows the approach proposed by [14], which utilizes dynamic programming in order to minimize both the number of line segments and the fitting error.

Given two points on the curve p_0 and p_N, we denote by $M(p_0, p_N)$ the cost of the configuration of the set of connected line segments from p_0 to p_N. We aim at minimizing the cost of the line-segment configuration:

$$M(p_0, p_N) = \min_{p_i}(M(p_0, p_i) + M(p_i, p_N), E(p_0, p_N) + E_{cost}).$$

In this equation, $M(p_0, p_i)$ (similarly, $M(p_i, p_N)$) is the cost of the line-segment configuration from p_0 to some point p_i residing between p_0 to p_N on the input curve. E_{cost} is a penalty term for adding a new line segment, hence restricting the number of segments used in the approximation. $E(p_0, p_N)$ denotes the error of fitting the line segments to the curvature values of the original curve. Recalling the sought-after linear curvature, we define this error as:

$$E(p_0, p_N) = \min_{\kappa_0, \gamma}\left(\sum_{i=0}^{N}(\kappa_0 + \gamma * s(p_i) - \kappa(p_i))^2\right).$$

Here, $s(p_i)$ and $\kappa(p_i)$ are the arc-length and the curvature at point p_i on the input curve. κ_0 and γ are found by comparing the deviation of $E(p_0, p_N)$ w.r.t. κ_0 and γ to zero:

$$\frac{\partial E(p_0, p_N)}{\partial \kappa_0} = 0\ , \ \frac{\partial E(p_0, p_N)}{\partial \gamma} = 0.$$

The algorithm proceeds in a bottom up fashion, where the solutions to large subproblems build on the solutions to smaller subproblems. It starts by calculating M for each pair of subsequent points. Then, for larger segments p_i is chosen so as to minimize $M(p_0, p_N)$.

Note that the term E_{cost} is of substantial importance, since it has a strong influence on the number of segments. Its value is adjusted to the input curve, as follows. First, the curve's curvature is segmented at the curve's local minima and maxima. Second, the value of the error $E(p_0, p_N)$ that suits this segmentation is calculated. Finally, E_{cost} is set to $\epsilon \cdot E(p_0, p_N)$. ϵ is a user-defined parameter that lets the user control the error tolerance. In all the examples we present in this paper (except for Figure 7) $\epsilon = 0.1$.

Up to this point we found the best linear fit for each segment separately. This might result in G^2-discontinuity at the contact points between adjacent spirals. This is due to curvature segments that do not intersect properly in curvature space, as illustrated by the red segments in Figure 3.

The straightforward solution to this problem is to modify the spirals, so that they meet at the curvature-line intersections (the red intersections in Figure 3). This solution is

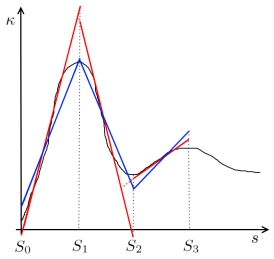


Figure 3: Obtaining G^2-continuity. The best linear fitting (red) to the curvature (black) results in undesirable intersections of the red line segments. To obtain G^2-continuity, we aim at the blue fitting.

problematic because the resulting curvature will differ considerably from that of the input curve.

Instead, our solution slightly modifies the parameters of the curvature segments (the blue segments in Figure 3). The idea is to find all the linear parameters (κ_{0i}, γ_i) simultaneously, imposing constraints on the continuity.

This is performed by solving the following optimization problem. Suppose that we have K curvature segments, where segment i starts at arc-length (of the input curve) S_{i-1} and ends at arc-length S_i and contains M_i sample points, $1 \leq i \leq K$. Let κ_{0i} be the initial curvature of the i^{th} spiral and γ_i be its slope. We minimize the error E_{tot}:

$$E_{tot} = \sum_{i=1}^{K} \sum_{j=0}^{M_i} \left(\kappa_{0i} + \gamma_i \cdot ds(p_j) - \kappa(S_{(i-1)} + ds(p_j))\right)^2,$$

where $ds(p_j)$ is the arc-length relative to the segment's initial point. In other words, we minimize the distance between a point on the blue segment to its corresponding point (having the same arc-length) on the black curve in Figure 3.

We add to this set of equations the following continuity constraints on the curvatures:

$$\kappa_{0i} + \gamma_i S_i = \kappa_{0(i+1)} + \gamma_{(i+1)} S_i, \quad 1 \leq i \leq K-1.$$

That is to say, we require that the curvatures of adjacent spirals at their intersection point are equal. Extracting κ_{0i} we get:

$$\kappa_{0i} = \kappa_{01} + \gamma_1 S_1 + \sum_{k=2}^{i-1} \gamma_k (S_k - S_{k-1}) - \gamma_i S_{i-1}.$$

Therefore, parameters κ_{0i} $\forall 2 \leq i \leq K$ are represented as a function of the parameters $\Gamma = (\kappa_{01}, \gamma_1, \gamma_2, ..., \gamma_K)^T$. The error E_{tot} can now be re-formulated in a matrix notation as:

$$E_{tot} = \|A\Gamma - B\|^2,$$

where vector B (of length equal to the number of sample points along the input curve) holds the curvature values κ of the input curve and the rows of matrix A are the coefficients of the parameters in Γ.

E_{tot} is minimized in a standard manner by the product of the pseudo-inverse of matrix A and vector B:

$$\Gamma = (A^T A)^{-1} A^T B.$$

4. Piecewise 3D Euler spiral construction: This step constructs the piecewise 3D Euler spiral. After the previous step, for each segment i we have the curvature parameters κ_{0i}, γ_i, the torsion parameters τ_{0i}, δ_i, and the length L_i. If we were also given the initial point $\mathbf{x}_{0i}$, the initial tangent $\vec{T}_{0i}$, and the initial normal $\vec{N}_{0i}$, these parameters would fully define the 3D Euler spiral, as described in [7].

The first segment is constructed using the initial conditions of the input curve $\mathbf{x}_0, \vec{T}_0, \vec{N}_0$. Jointly with the segment parameters $\kappa_{01}, \gamma_1, \tau_{01}, \delta_1, L_1$, the first segment is defined as:

$$\mathcal{C}_1(s) = \int_0^s \left[\int_0^t \frac{d\vec{T}_1(u)}{du} du + \vec{T}_0 \right] dt + \mathbf{x}_0,$$

where $0 \leq s \leq L_1$ and $\frac{d\vec{T}_1(u)}{du}$ is calculated according to the Frenet-Serret equations [5].

This equation also defines the first segment's endpoint, $\mathbf{x}_{f1} = \mathcal{C}_1(L_1)$. The other end conditions (tangent and normal) are found by the tangent's and normal's definitions:

$$\vec{T}_{f1} = \int_0^{L_1} \frac{d\vec{T}_1(u)}{du} du + \vec{T}_0 \ , \ \vec{N}_{f1} = \int_0^{L_1} \frac{d\vec{N}_1(u)}{du} du + \vec{N}_0.$$

The end conditions of the first segment are considered the initial conditions of the second segment: $\mathbf{x}_{02} = \mathbf{x}_{f1}$, $\vec{T}_{02} = \vec{T}_{f1}$, $\vec{N}_{02} = \vec{N}_{f1}$. We can therefore calculate the second spiral. Similarly, we proceed to calculate all the subsequent Euler spirals, where spiral $i+1$ is computed using the information gained after calculating spiral i. This results in the sought piecewise 3D Euler spiral representation of the input curve.

In case of corners (points of tangent discontinuity) using the end-tangent (/normal) of preceding segment as the initial tangent of the following is erroneous. At these points we set the angle between the tangents to be equal to the angle between the tangents at the corresponding points (having the same arc-length) on the input curve. This angle is calculated using the smoothed tangents $\vec{T}^s$, computed using the results obtained in Step 2:

$$\vec{T}^s(s) = \frac{r'}{\|r'\|} \ , \ \vec{N}^s(s) = \frac{r'' - (r'' \cdot \vec{T}^s) \cdot \vec{T}^s}{\|r'' - (r'' \cdot \vec{T}^s) \cdot \vec{T}^s\|}.$$

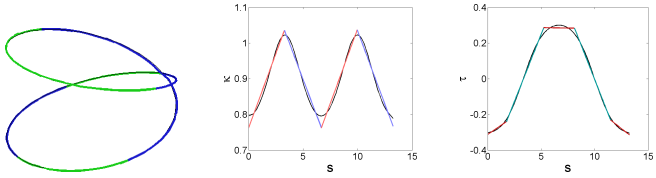
For closed curves, the resulting approximation is closed using the curve completion of [7].

5. Improvement by rigid transformation: The previous step created a piecewise Euler spiral curve approximation, which often suffices. However, when the initial tangent & normal are inaccurate (due to noise) or in the existence of corners, it can be improved. In this case, to decrease the fitting error, we apply a rigid transformation for the entire piecewise curve, followed by rotations of the parts of the curve bounded by corners (Figure 2(f)).

The rigid transformation is found by solving the following weighted least-squares minimization problem, similarly to [16]. The input curve and the piecewise curve are sampled uniformly into corresponding sets of n points: $\{(x_0^I, y_0^I, z_0^I), \cdots, (x_{n-1}^I, y_{n-1}^I, z_{n-1}^I)\}$ and $\{(x_0^P, y_0^P, z_0^P), \cdots, (x_{n-1}^P, y_{n-1}^P, z_{n-1}^P)\}$, respectively. We seek the rotation matrix R and the translation vectors T and T_0 that minimize the following error:

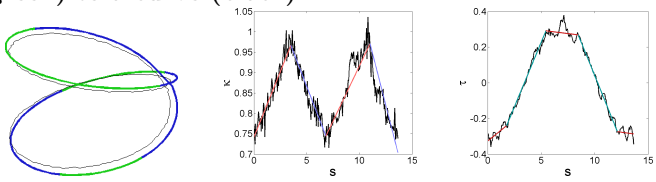
$$E_R = \sum_{i=0}^{n} \|R(r_i^I + T_0) + T - r_i^P\|^2,$$

where $r_i^I = (x_i^I, y_i^I, z_i^I)^T$ and $r_i^P = (x_i^P, y_i^P, z_i^P)^T$. The local rotation matrices are found similarly.



(a) Curve fitting (b) Curvature fitting (c) Torsion fitting

Figure 4: Fitting piecewise 3D Euler spirals (blue & green) to a curve (black).



(a) Curve fitting (b) Curvature fitting (c) Torsion fitting

Figure 5: Fitting piecewise 3D Euler spirals (blue & green) to a noisy curve (black).

3. RESULTS

Figure 4 demonstrates our fitting result, given synthetic example. In this case, our piecewise representation, which consists of seven segments, nicely approximate the input curve using only a handful of spirals – the input (black) curve is hardly noticeable.

Figure 5 shows an example of a noisy version of the curve in Figure 4. White noise was added to every coordinate of the input points. The variance of the noise is defined for the x-coordinate (and similarly for the y and z coordinates) as: $\sigma_x = \frac{1}{4n}\sum_{i=1}^{n}|x_i - x_{i-1}|$. It can be seen that though the curvature and the torsion are noisy, our piecewise representation is similar to that in Figure 4. The measured mean squared error (MSE) has increased from 7.78e-04 to 0.029.

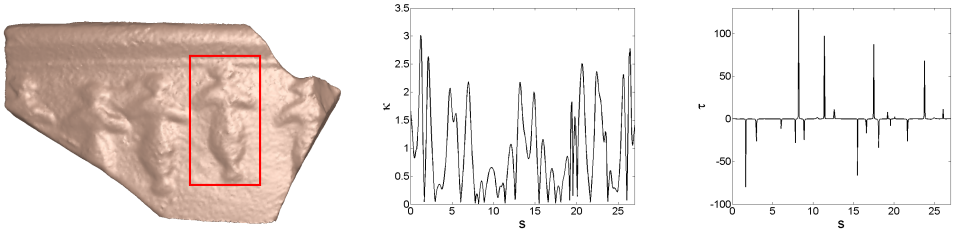
Figure 6 shows our approximation of a real example of a relief of a "dancer" extracted from a Hellenistic broken vase from the first century BCE. This curve is much noisier than the synthetic examples, yet our algorithm manages to represent it with only 70 3D Euler spirals.

Finally, Figure 7 shows our 3D Euler approximation of an extremely noisy example. The given model is a seal from the early Iron Age, 11^{th} century BCE. As typical of such ancient artifacts, the seal was found noisy and eroded. The 3D curves on the surface were extracted using the algorithm of [22] (Figure 7(b)). Though from a certain viewpoint, the curve may look almost planar, it is inherently non-planar, as can be seen in Figure 7(e) where the curves are drawn from different viewpoints. This can also be noticed in the noisy graphs of the curvature and the torsion in Figures 7(c,d). Nevertheless, our piecewise representation manages to capture the shape of the curves (Figure 7(e)). As expected, in this case, many spirals are needed (about 200). Obviously, the larger the error allowed, the less spirals would be used. For instance, if we allow only 100 spirals, we would get the image in Figure 7(f), which illustrates that our approximation still maintains the general shape.

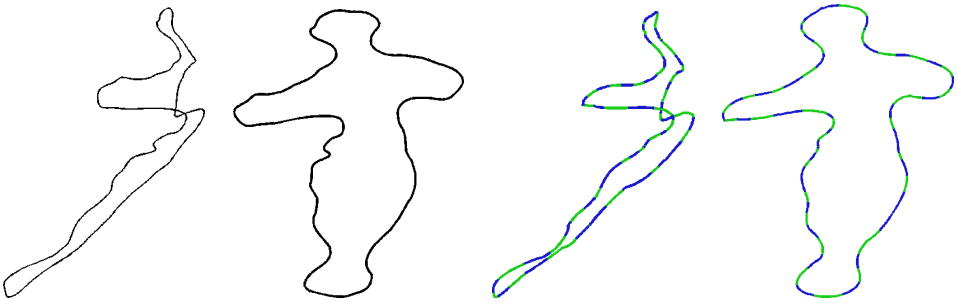
4. CONCLUSION

This paper presented an algorithm for fitting a piecewise 3D Euler spiral to a 3D curve with G^2-continuity. This representation is capable of handling noisy curves, as well as sharp corners. Our representation is constructed using a dynamic programming approach that fits a connected set of line segments to the curvature values and similarly, a connected set of lines to the torsion values of the input curve.

Since our piecewise 3D Euler spiral representation is similarity invariant, it may be used in the future for shape analysis applications, such as symmetry detection or retrieval.

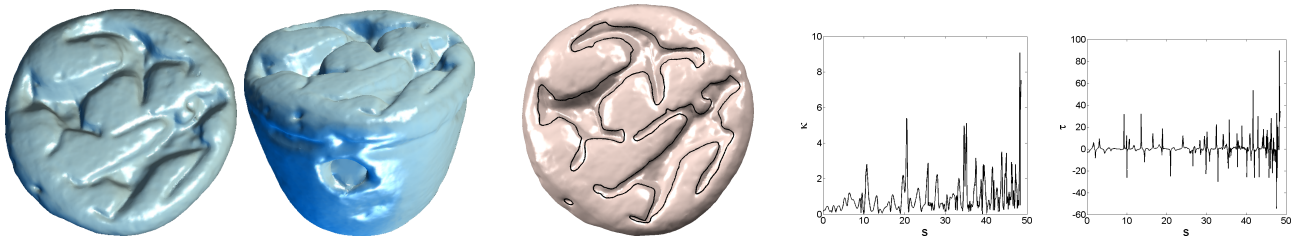


(a) Input model (b) Curvature (c) Torsion

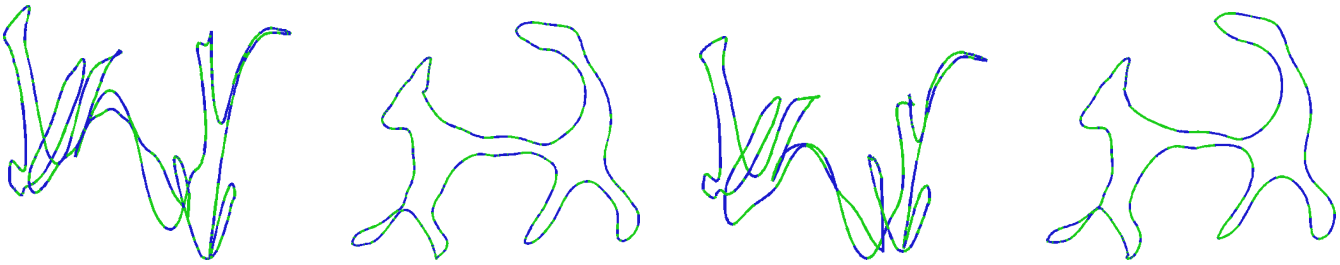


(d) Input curve : two views (e) Our fitting : two views

Figure 6: Fitting piecewise 3D Euler spirals (blue & green) to the curve of a dancer from a Hellenistic broken amphora from the 1^{st} century BCE.



(a) A seal from the Iron Age (b) Extracted curves [22] (c) Curvature (d) Torsion



(e) Approximation with 200 spirals: two views (MSE=0.0091) (f) Approximation with 100 spirals: two views (MSE=0.0293)

Figure 7: Approximating a real curve extracted from an archaeological artifact. As can be seen, the input curve, its curvature and torsion, are inherently noisy. Yet, our approximation manages to maintain the shape of the curve and smooth the noise.

5. REFERENCES

[1] I. Baran, J. Lehtinen, and J. Popovic. Sketching clothoid splines using shortest paths. In *Computer Graphics Forum*, 2010.

[2] M. Bokeloh, A. Berner, M. Wand, H. Seidel, and A. Schilling. Symmetry detection using feature lines. In *Computer Graphics Forum*, volume 28, pages 697–706, 2009.

[3] M. Boutin. Numerically invariant signature curves. *Int. J. of Computer Vision*, 40(3):235–248, 2000.

[4] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics*, 22(3):848–855, 2003.

[5] M. do Carmo. *Differential geometry of curves and surfaces*. Prentice Hall, 1976.

[6] G. Farin. *Curves and surfaces for computer aided geometric design*. Academic Press Prof. Inc., 1993.

[7] G. Harary and A. Tal. 3D Euler spirals for 3D curve completion. In *ACM Symposium on Computational Geometry*, 2010.

[8] B. Kimia, I. Frankel, and A. Popescu. Euler spiral for shape completion. *Int. J. Comp. Vision*, 54(1):159–182, 2003.

[9] D. Knuth. Mathematical typography. *American Mathematical Society*, 1(2):337–372, 1979.

[10] M. Kolomenkin, I. Shimshoni, and A. Tal. On edge detection on surfaces. pages 2767–2774. CVPR, 2009.

[11] T. Langer, A. Belyaev, and H. Seidel. Asymptotic analysis of discrete normals and curvatures of polylines. In *Spring conf. on Computer graphics*, pages 229–232, 2005.

[12] T. Lewiner, J. Gomes, H. Lopes, and M. Craizer. Curvature and torsion estimators based on parametric curve fitting. *Computers & Graphics*, 29(5):641–655, 2005.

[13] J. McCrae and K. Singh. Sketch-based path design. *Graphics Interface*, pages 95–102, 2009.

[14] J. McCrae and K. Singh. Sketching piecewise Clothoid curves. *Computers & Graphics*, 33(4):452–461, 2009.

[15] D. Meek and D. Walton. The use of Cornu spirals in drawing planar curves of controlled curvature. *J. of Comp. and Applied Mathematics*, 25(1):69–78, 1989.

[16] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *ACM Trans. on Graphics*, 24(3):471–478, 2005.

[17] A. Nutbourne, P. McLellan, and R. Kensit. Curvature profiles for plane curves. *Computer-Aided Design*, 4(4):176–184, 1972.

[18] T. Pal and A. Nutbourne. Two-dimensional curve synthesis using linear curvature elements. *Computer-Aided Design*, 9(2):121–134, 1977.

[19] A. Schechter. Synthesis of 2D curves by blending piecewise linear curvature profiles. *Computer-Aided Design*, 10(1):8–18, 1978.

[20] R. Schneider and L. Kobbelt. Discrete fairing of curves and surfaces based on linear curvature distribution. *Curve and Surface Design*, pages 371–380, 1999.

[21] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *International Conference on Computer Vision*, pages 839–846, 1998.

[22] R. Zatzarinni, A. Tal, and A. Shamir. Relief analysis and extraction. *ACM Transactions on Graphics*, 28(5):136:1–9, 2009.