The Visual Computer (2023) 39:3767–3781
https://doi.org/10.1007/s00371-023-03001-x

ORIGINAL ARTICLE

A variational approach for feature-aware B-spline curve design on surface meshes

Rongyan Xu[1] · Yao Jin[1,2] · Huaxiong Zhang[1,2] · Yun Zhang[3] · Yu-kun Lai[4] · Zhe Zhu[5] · Fang-Lue Zhang[6]

Accepted: 9 June 2023 / Published online: 25 July 2023

Abstract
Robust curve design on surface meshes with flexible controls is useful in a wide range of applications but remains challenging. Most existing methods fall into one of the two strategies: one is to discretize a curve into a polyline, which is then optimized, and the other is to directly design smooth splines on meshes. While the former approach usually needs a sufficiently dense sampling of curve points, which is computational costly, the latter approach relaxes the sampling requirement but suffers from the lack of user control. To tackle these problems, we proposed a variational method for designing feature-aware B-spline curves on surface meshes. Given the recent advances in shell space construction methods, we could relax the B-spline curve inside a simplified shell mesh and evaluate its distance to the surface using equipped bijective mapping. To effectively minimize the distance between the curve and the surface, with additional controls in the form of both internal and external constraints, we applied the interior point method and adaptively inserted knots of the spline to increase its freedom and adjust the weighting during the iterations. When the curve is close enough to the surface, it can be efficiently sampled at any resolution and robustly projected to the surface. Experiments show that our method is more robust, has higher flexibility, and generates smoother results than existing methods.

Keywords B-spline curve · Surface meshes · Feature aware · Knot insertion · Variational optimization

1 Introduction

Curve design is an important problem in computer graphics and computer-aided geometric design. Although with a solid theoretical foundation in Euclidean space, this problem has different formulations in other spaces such as curved spaces (e.g., a triangle mesh). Such curves usually have a wide range of applications such as mesh cutting, feature specification, artistic design, manufacturing, virtual reality, and computational geometry on surfaces. Besides some common properties shared with the Euclidean space such as smoothness, interpolation, and possibly feature-awareness, in a curved space it requires the curve to be confined on the mesh surface (manifold constraint), which makes the problem more challenging.

To design such curves, most of the existing methods address this problem by discretizing the curve into a polyline and then optimizing it to fulfill all the requirements, e.g., interpolation, smoothness, and manifold constraints. Based on how to handle the manifold constraint, those methods can be classified into different categories such as projection-based [6, 9], smoothing-based [12] and parameterization-based [13]. The computational cost of these methods largely depends on the sampling density of the curve, so it could be costly if the sampling is dense. However, if the sampling is not sufficiently dense, it could lead to visually rough results.

Some other methods [18, 21] propose to design a smooth spline curve to approach the mesh and discretize the curve

✉ Huaxiong Zhang
zhxhz@zstu.edu.cn

Rongyan Xu
1536980672@qq.com

1 School of Computer Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China

2 Zhejiang Provincial Innovation Center of Advanced Textile Technology, Shaoxing 312000, China

3 College of Media Engineering, Communication University of Zhejiang, Hangzhou 310018, China

4 School of Computer Science and Informatics, Cardiff University, Cardiff, Wales CF24 4AG, UK

5 Mathworks, Natick, MA 01760, USA

6 School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6012, New Zealand

in the last step. Spline curves can be considered as an extension of polylines with high-order smooth bases but require fewer variables. Since spline curves are already smooth, the key lies in how to constrain them on the surface. Most methods design such curves by extending the De Casteljau's algorithm from Euclidean space to a manifold. While these methods can ensure the smoothness of the curves, they are hard to control, e.g., in the situation of interpolating given points [18, 20]. Panozzo et al. [21] embed the manifold to a high-dimensional Euclidean space, where B-splines can be flexibly designed and smoothly projected to the surface. However, the projection scheme is not robust and may fail in some cases. Furthermore, none of these methods can design feature-aware curves, e.g., geometric snakes [13].

In this paper, we proposed a novel approach for feature-aware B-spline curve design on surface meshes. While our approach is an extension of the shell-space-constrained approach [9], for robustness and efficiency we parameterized the ambient space of the surface into a simplified volumetric shell mesh equipped with a bijective mapping [8] and perform an alternating geometry optimization and adaptive knot insertion in the shell space. In summary, our contributions are threefold:

1. We presented a robust and customized shell-space constrained approach for B-spline curve design on surface meshes with flexible control.
2. We proposed a general feature-aware term which favors a variety of applications and can be seamlessly incorporated into the energy function for optimization.
3. We introduced a new knot insertion scheme, which is guided by a feature function and based on the difference between the curve and its projection.

2 Related work

The curve design problem on continuous and discrete manifolds has been well studied in the last decades, and there is a lot of literature related to this topic. Here our focus is on curve design on discrete surface meshes. According to the curve representations, existing methods can be classified as either implicit or explicit. We mainly introduced explicit methods, which directly design an explicit curve on the mesh to meet several key constraints, e.g., smoothness, interpolation, and manifoldness. Implicit methods are also known as level set approaches, which need to solve a PDE (partial differential equation) for a smooth scalar field on the mesh with specific boundary conditions, and the curve is extracted from its level set [1, 16, 17]. Such methods are robust, and the designed curve is guaranteed to satisfy the manifold constraint, but at the expense of high computational cost. So such methods are beyond the scope of this paper.

Unlike the Euclidean space situation, the key to designing curves on surface meshes is to confine the curve to lie on the surface. Most existing methods discretize the curve into a polyline and iteratively optimize it to reach an ideal state. Some of them eliminate the complicated manifold constraint by using local single-patch parameterizations [13, 14] so that they can flexibly design curves in the Euclidean plane and map the results to the surface. However, it is usually not easy to determine an appropriate part region for parameterization, and the mapping distortion may also affect the results. Some other methods relax the smoothness constraint while keeping the manifold constraint and gradually smooth an initial curve on the surface, driven by specific energy functions. Jung et al. [10] extended the well-known active contour model from the image space to surface meshes and diffused a closed polyline lying on the mesh edges by the snake energy. Ji et al. [7] improved the results by allowing the curve to pass across mesh faces, i.e., adaptively merging or splitting the vertices on the polyline during the iteration. Lawonn et al. [12] further smoothed the polyline by solving a weighted Laplacian energy, which can reduce the geodesic curvature. Due to the strict manifold constraint, these methods are robust but not efficient and tend to plunge into local minimum. Some other methods relax the manifold constraint, smooth the curve in the ambient space and project the result to the surface. Hofer et al. [6] defined a spline energy for the polyline and performed energy minimization in the vicinity of the surface by using the projection gradient method. Later, high-order versions of spline energy are proposed and used for data fitting on surface meshes with the same strategy [23]. Hofer et al. [5] summarized the methods and showed their variety of applications. However, the projection step relies on the local fitting of moving least squares (MLS), which is usually time-consuming and may sacrifice robustness. Jin et al. [9] recently applied shell space for robust distance evaluation and projection, which shares both advantages of smoothing-based and projection-based methods. Their bottleneck lies in constructing a shell mesh with proper thickness. Optimizing a polyline on the mesh is straightforward and popular, but on the other hand it has to predetermine the sampling density. Moreover, the improper estimation of density may primarily affect the smoothness of the curve or computational efficiency.

Instead of designing a polyline directly, some other methods generate a smooth spline approaching the surface meshes and discretize it in the final step. Among them, most try to extend the weighted averaging from Euclidean space to manifolds. Park and Ravani [22] firstly extended De Casteljau's algorithm to manifolds, where the control polygon is represented with geodesic paths. Wallner and Pottmann [25] generalized the Euclidean linear subdivision scheme by using the "geodesic averaging" method. Morera et al. [20] extended the recursive De Casteljau bisection and proposed the geometric Bézier algorithm. Sarlabous et al. [3] further proposed

geodesic conic Bézier curves, which are suitable for complicated geometries. Sharp et al. [24] recently proposed an approach that achieves better computational performance on the same algorithm [20] by using a fast geodesic computing method. Furthermore, Mancinelli et al. [18] extended both the recursive De Casteljau bisection and an open-uniform Lane–Riesenfeld subdivision scheme, which achieves higher performance. Such subdivision-based methods are robust but lack flexible control over the curve. And none of them can adapt to the feature of the mesh due to their forward design manner, which limits their applications. Panozzo et al. [21] embedded the mesh in a high-dimensional Euclidean space to compute weighted averages, generated a B-spline there, and mapped sampled points of the curve onto the surface by Phong projection. The method can be formulated as a variational problem, and is flexible in terms of user control, but the projection operator is not robust and it is hard to decide how to project the whole curve on the surface.

To robustly control the curve on surface meshes, we represented the curve as a B-spline and formulate the curve design problem in a variational way with the assistance of shell space [9]. One of the key elements of our method is the knot selection (number and positions) for the B-spline, which is vital in data fitting. Common methods usually predetermine initial knots and adaptively adjust them during iterations, e.g., changing knot positions [27], decreasing [11] or increasing [15] number of knots. Kang et al. [11] selected a set of active knots from a dense knot vector via a sparse optimization. Liang et al. [15] selected knots according to the constant increment of the feature integral, followed by an iterative knot insertion scheme to improve the fitting precision. Dung et al. [2] proposed a two-step method for fast knot optimization. Yeh et al. [26] proposed a curvature-based feature integral to guide the knot insertion. Mohanty and Fahnestock [19] applied particle swarm optimization combined with model selection to tackle the involved high-dimensional and non-convex optimization. We incorporated a new feature integral-based knot insertion scheme into the optimization to drive the B-spline effectively to approach the surface mesh. Overall, compared to the polyline-based methods, our B-spline representation avoids the need to determine the sampling density but retains the smoothness of the curve; compared to the existing spline-based methods, ours are robust, controllable, and feature-sensitive.

3 Problem and algorithm overview

Let $\mathcal{S} =< V_{\mathcal{S}}, F_{\mathcal{S}} >$ denote a self-intersection free and orientable manifold triangle mesh embedded in Euclidean space $\mathbb{R}^3$, where $V_{\mathcal{S}}$ and $F_{\mathcal{S}}$ stand for the vertex and face sets of the mesh. And the mesh possibly associates a scalar field representing some user-defined features (depending on specific applications), $\mathcal{I} : \mathcal{S} \to \mathbb{R}$. Our aim is to design a "visually smooth" curve on the surface $\mathcal{S}$ interpolating a sequence of points $\mathbf{X} = \{\mathbf{x}_i \in \mathcal{S}\}_{i=1}^{N}$ while adapting to the scalar field at the same time. The curve is already discrete because it is embedded in the mesh surface. We thus defined the "smoothness" of such curves discretely, analogous to the case of continuous curves.

To solve the problem, instead of directly designing a discrete curve like [6, 9], we used a B-spline to model the curve here to improve the quality of results. Let the B-spline curve be represented as, $\mathbf{P}(t) = \sum_{i=0}^{n} \mathbf{C}_i B_{i,k}(t)$ with a knot vector $\mathcal{T} = \{t_0, t_1, \ldots, t_{n+k+1}\}$, where $\mathcal{P} = \{\mathbf{C}_i\}_{i=0}^{n} \in \mathbb{R}^3$ are $n+1$ control points and $\mathcal{B} = \{B_{i,k}(t)\}$ are basis functions which can be defined recursively as follows,

$$B_{i,0}(t) = \begin{cases} 1 \; t \in [t_i, t_{i+1}) \\ 0 \quad \text{otherwise} \end{cases} \tag{1}$$

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} B_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(t). \tag{2}$$

Similar to the geometric snake model [13], we could thus find such a curve by solving the following variational problem,

$$\begin{aligned} &\min_{\mathcal{P},\mathcal{T},\mathcal{I}} E_{\text{int}}(\mathcal{P}, \mathcal{T}) + \mu E_{\text{ext}}(\mathcal{P}, \mathcal{T}, \mathcal{I}) \\ &\quad \text{s.t.} \max_{t \in [0,1]} D(\mathbf{P}(t), S) < \varepsilon \\ &\quad \mathbf{P}(t_{k_i}) = \mathbf{x}_i, i = 1, 2, \ldots, N, \end{aligned} \tag{3}$$

where $(t_{k_i}, \mathbf{x}_i)$ is the i^{th} pair of parameter and interpolatory point and ε (e.g., ten percent of average length of the mesh edges) is a small constant to ensure that the B-spline is sufficiently close to the surface mesh.

One part of the objective $E_{\text{int}}(\mathcal{P}, \mathcal{T})$ is the internal energy. It considers the fairness of the curve and can be commonly defined as the combination of ℓ_2 norms of the first and second derivatives, which is also known as splines in tensions [6]:

$$E_{\text{int}}(\mathcal{P}, \mathcal{T}) = \frac{1}{L^2} \int_0^1 \left(\lambda \|\mathbf{P}'(t)\|^2 + (1 - \lambda) \|\mathbf{P}''(t)\|^2 \right) dt, \tag{4}$$

where $L = \sum_{i=1}^{N-1} \|\mathbf{x}_i - \mathbf{x}_{i+1}\|$ is an approximation for the length of the curve, and λ is the weight used to balance the two terms. The energy for open curves is slightly different from closed curves, and for the later case it should prescribe continuous condition for the end points.

The other part $E_{\text{ext}}(\mathcal{P}, \mathcal{T}, \mathcal{I})$ is the external energy. It is determined by the features defined on the mesh, which could guide the curve to snap to the local extremum of the scalar field. And it can be defined as:

$$E_{\text{ext}}(\mathcal{P},\mathcal{T},\mathcal{I}) = \int_0^1 c\left(\mathcal{I}\left(\mathcal{M}_{\mathcal{S}}\left(\mathbf{P}(t)\right)\right)\right) dt, \tag{5}$$

where $\mathcal{M}_{\mathcal{S}}$ is the projection operator which maps a curve point to the mesh surface $\mathcal{S}$, and $c : \mathbb{R} \to \mathbb{R}^+ \cup \{0\}$ is a composite function for the scalar field, which will all be elaborated in the following sections.

The constraint in Eq. 3 is a relaxation of the manifold constraint, where $D(\mathbf{P}(t), \mathcal{S})$ is the distance function measuring the distance from a curve point to the mesh surface and is defined as:

$$D(\mathbf{P}(t), \mathcal{S}) = \inf_{\mathbf{p}\in\mathcal{S}} \|\mathbf{P}(t) - \mathbf{p}\|. \tag{6}$$

By now, we could convert the constrained optimization (3) to an unconstrained one by taking manifold and interpolation constraints into penalties,

$$\begin{aligned}\min_{\mathcal{P},\mathcal{T},\mathcal{I}} E_{\text{int}}(\mathcal{P},\mathcal{T}) + \mu E_{\text{ext}}(\mathcal{P},\mathcal{T},\mathcal{I}) \\ + \alpha E_{\text{dist}}(\mathcal{P},\mathcal{T}) + \beta E_{\text{fit}}(\mathcal{P},\mathcal{T})\end{aligned} \tag{7}$$

where $E_{\text{dist}}(\mathcal{P},\mathcal{T})$ is the energy term compositing the distance function (6) with a barrier function g. It maps the distance value to a nonnegative value:$\mathbb{R}^+ \cup \{0\} \to \mathbb{R}^+ \cup \{0\}$,

$$E_{\text{dist}}(\mathcal{P},\mathcal{T}) = \int_0^1 g(D(\mathbf{P}(t), \mathcal{S}))dt. \tag{8}$$

And $E_{\text{fit}}(\mathcal{P},\mathcal{T})$ is the soft positional constraint with a big weight β (e.g., $\beta = 10^6$ by default) to meet the interpolation constraints,

$$E_{\text{fit}}(\mathcal{P},\mathcal{T}) = \frac{1}{NL^2}\sum_{i=1}^{N} ||\mathbf{P}(t_{k_i}) - \mathbf{x}_i||^2, \tag{9}$$

where $\frac{1}{NL^2}$ is used to make the energy term dimensionless.

To solve the optimization (7), we first gave the definition of the distance function (6). Like [9], we built a discrete shell space and defined an appropriate in-between scalar field to approximately evaluate the point-to-surface distance. However, it is non-trivial to extend the algorithm [9] for B-spline curves directly. Firstly, it is hard to obtain an initial B-spline curve inside the shell, which is essential for the adopted interior point solver. Secondly, a valid shell mesh with a proper thickness, i.e., without inverted elements, is difficult to build. In certain circumstances, such a shell does not even exist, which makes it impossible to design the curves. Thirdly, the problem of selecting appropriate knots of the B-spline that can reasonably fit the surface mesh should also be solved.

To this end, we built our approach upon the recently published robust shell construction algorithm [8] and proposed a new algorithm for the B-spline curve in such shells.

4 Our algorithm

Our algorithm consists of the following steps: scalar field construction (Sect. 4.1), initial curve generation (Sect. 4.2), followed by the alternating iterations between curve optimization (Sect. 4.3) and knot insertion (Sect. 4.4), and finally the obtained curve is robustly projected onto the mesh surface with defined mapping $\mathcal{M}$ when the distance error is below the predetermined threshold to ensure that the curve is on the surface. Algorithm 1 illustrates our whole algorithm.

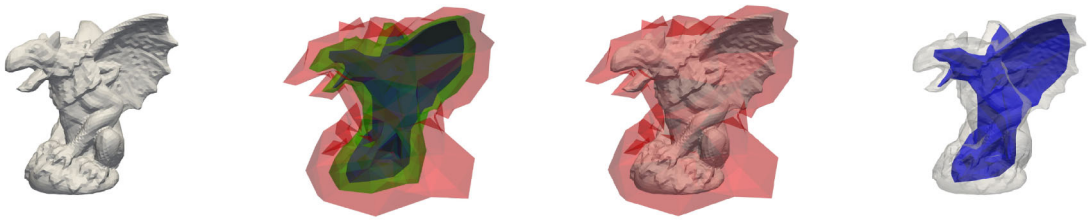
Algorithm 1 B-spline curve design in the shell space

Require: a mesh $\mathcal{S}$ with a scalar field $\mathcal{I}$, the shell space for $\mathcal{S}$: $\Omega_{\mathcal{S}}$, and a sequence of interpolatory points: $\mathbf{X} = \{\mathbf{x}_{i=1}^{N} \in \mathcal{S}\}$.
Ensure: a feature-aware curve $\mathbf{Q}$ on the surface interpolating $\mathbf{X}$.
1: Generate an initial B-spline curve with $< \mathcal{P}, \mathcal{T} >$(Sec. 4.2)
2: **repeat**
3: Update weights with Eq. (18)
4: Curve optimization: update $\mathcal{P}$ (Sec. 4.3)
5: Knot insertion: update $\mathcal{T}$ (Sec. 4.4)
6: **until** the stop conditions are satisfied
7: Project the B-spline curve onto the mesh
8: **return** the projected curve $\mathbf{Q}$

4.1 Scalar field construction

For fast and robust evaluation of the distance value for a given point near the surface mesh, it is helpful to construct a foldover-free shell space tessellated with tetrahedrons surrounding the surface mesh. Here, directly extruding two layers of prism mesh from the surface by mesh deformation might be prone to foldover tetrahedron elements. Instead, we applied the shell meshes equipped with a bijective map $\mathcal{M}$ on the shelf [8], which is proved to be robust experimentally. Here, we briefly reviewed the method to make our description self-contained. It starts from a triangle mesh, then finds an ideal direction of extrusion for each vertex by a local optimization, builds the prism shell with the calculated directions, and performs optimization to simplify it in topological and geometric aspects. The final obtained shell meshes are highly simplified, which hold the advantages of robustness and efficiency for further computation (see Fig. 1).

The constructed shell mesh Ω is associated with a scalar field $f : \Omega \to \mathbb{R}$. In the setup, we specified the scalar values for the vertices of the three layers, i.e., top $\mathcal{S}_T$, middle $\mathcal{S}_M$ and bottom $\mathcal{S}_B$ of the shell as $1, 0, -1$, respectively. Moreover, those points inside the shell can be linearly interpolated based on the tetrahedron elements where they locate. Here, we needed to calculate the distance from any point inside the shell space to the surface mesh (8). Thus, we built a new scalar field $d(\mathbf{p})$ such that the scalar value of any point inside as its distance to the mesh surface. In such a scalar



(a) The input mesh (b) The simplified shell (c) Input mesh & Top (d) Input mesh & Bottom

Fig. 1 The simplified shell generated by Jiang et al. [8]

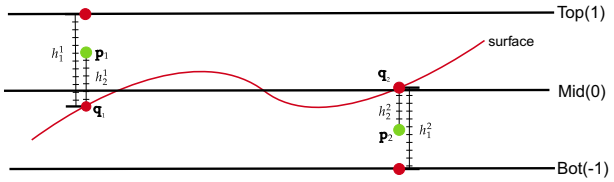


Fig. 2 Illustration of symbols h_1, h_2 defined in the shell, where the calculation h_1 depends on the relative positions between $\mathbf{p}$ and its projection $\mathcal{M}_{\mathcal{S}}(\mathbf{p})$ onto the surface mesh

field, the value for the boundary of the shell is set to 1, i.e., $d(\mathbf{p})|_{\mathbf{p}\in\partial\Omega} = 1$, and the original mesh surface $\mathcal{S}$ is set as zero level set, i.e., $d(\mathbf{p})|_{\mathbf{p}\in\mathcal{S}} = 0$. After that, the value d of any point inside the shell can be obtained by linearly scaling the original scalar field f. We would later composite the scalar field with a smooth barrier function g with $g(1) = \infty$ to confine the curve inside the shell space. With the help of the bijective map $\mathcal{M}$ between any pair of sections provided by the shell, we could efficiently find the projection point on the mesh surface of any sampled point on the curve $\mathbf{p}$: $\mathcal{M}_{\mathcal{S}}(\mathbf{p})$. Here, the subscript of $\mathcal{M}$ denotes the section of the projection, e.g., $\mathcal{M}_{\mathbf{T}}$ and $\mathcal{M}_{\mathbf{B}}$ are maps when the projection sections correspond to the top and bottom layers.

In detail, we calculated the new scalar value for the point $\mathbf{p}$ as the relative distance between $\mathbf{p}$ and its projection $\mathcal{M}(\mathbf{p})$, $\mathcal{D}(\mathbf{p}, \mathcal{S}) = d(\mathbf{p}) = h_2/h_1$, where $h_2 = |f(\mathbf{p}) - f(\mathcal{M}_{\mathcal{S}}(\mathbf{p}))|$ denotes the absolute difference of the scalar values between $\mathbf{p}$ and $\mathcal{M}_{\mathcal{S}}(\mathbf{p})$, and h_1 indicates the thickness of the shell at the projection point $\mathcal{M}_{\mathcal{S}}(\mathbf{p})$. An illustration can be seen in Fig. 2. In all, the new scalar field can be calculated as follows,

$$d(\mathbf{p}) = \begin{cases} \frac{f(\mathbf{p}) - f(\mathcal{M}_{\mathcal{S}}(\mathbf{p}))}{1 - f(\mathcal{M}_{\mathcal{S}}(\mathbf{p}))} & \text{if } f(\mathbf{p}) \geq f(\mathcal{M}_{\mathcal{S}}(\mathbf{p})) \\ \frac{f(\mathcal{M}_{\mathcal{S}}(\mathbf{p})) - f(\mathbf{p})}{1 + f(\mathcal{M}_{\mathcal{S}}(\mathbf{p}))} & \text{otherwise} \end{cases}. \tag{10}$$

As illustrated, the two situations are based on the relative positions of $\mathbf{p}$ and $\mathcal{M}_{\mathcal{S}}(\mathbf{p})$. The scaling ensures that points on the mesh surface are of 0 distance, and points on the top and bottom sections have distance of 1, and other points are assigned linearly in between.

To compute h_1 and h_2 for the point $\mathbf{p}_i$ sampled from the curve, we first needed to find the tetrahedron where it locates. Instead of directly searching for the element by an AABB tree [8], we adopt a more robust and efficient scheme that uses the information of the previous sampling point $\mathbf{p}_{i-1}$ on the curve. We traced the ray $\overline{\mathbf{p}_{i-1}\mathbf{p}_i}$ from the element where $\mathbf{p}_{i-1}$ locates and find all the intersection points of tetrahedrons until it reaches the endpoint $\mathbf{p}_i$.

During the subsequent optimization, we had to further composite the new scalar field with a barrier function g as introduced in (8), so as to constrain the curve inside the shell space. We thus selected the function which enjoys nice properties as introduced in [9]:

$$g_h(x) = \frac{\pi}{2h} x \tan\left(\frac{\pi}{2h} x\right), x \in [0, h). \tag{11}$$

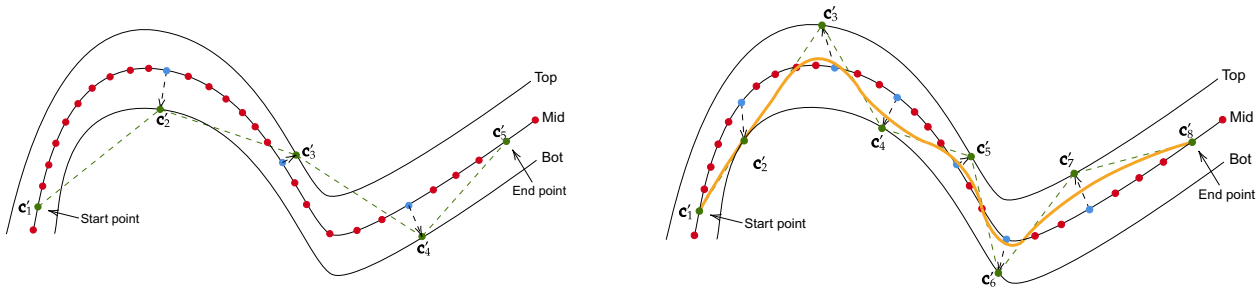
In the equation, the threshold h denotes the upper bound of the scalar value for the curve $\mathbf{P}(t)$, which can be calculated as follows:

$$h = \min\{\max_{t_i} d(P(t_i)) + \varepsilon_b, 1\} \tag{12}$$

where ε_b is a user prescribed small positive value used for better numerical stability and is set to 0.02 by default.

4.2 Initial curve generation

An initial B-spline curve is required to lie inside the shell space to bootstrap the iterative optimization procedure. However, it is challenging to generate a B-spline strictly inside the given shell with a theoretical guarantee. Hence we present a practical scheme by using the convex hull property of the B-spline, i.e., a B-spline curve is contained in the convex hull of its control polygon, which helps to alleviate the problem. We adaptively built a control polygon inside the shell space for the initial B-spline curve. In practice, we kept the first and last interpolatory points, i.e., $\mathbf{x}_1, \mathbf{x}_N$ fixed. Then, we sequentially and alternately projected the remaining points to the top and bottom layers of the shell as its control points with the bijective maps $\{\mathcal{M}_{\mathbf{T}}, \mathcal{M}_{\mathbf{B}}\}$. If the segment between any two adjacent control points intersects the boundary of the shell, calculate the Dijkstra path between the corresponding interpolatory points on the mesh. And the middle point $\mathbf{x}_m$



(a) The control polygon intersects with the shell boundary (b) Final control polygon and the initial curve

Fig. 3 Illustration of generating an initial curve. The Dijkstra path, interpolatory points, control points and the initial curve are marked in red, blue, green and yellow, respectively

on the path is then projected to the boundary. Finally, we inserted a new point into the sequence of the control points. This procedure repeats until no segment intersects the shell boundary. The pseudo-code of the algorithm can be seen in Algorithm 2, and an illustration is also shown in Fig. 3).

We have found rare failure cases in a large number of experiments. For robustness consideration, we assumed that parts of the curve have the chance of being out of the shell. In that case, we thus redefined the scalar field d by scaling it with a ratio η to avoid infinity value when it is composited with the barrier function, i.e., $d(\mathbf{p}) \Leftarrow \eta d(\mathbf{p})$ ($\eta = 0.99$ in default). We set the scalar value of each point outside the shell the same as that on the boundary. In such a setting, the value of the barrier function on or out of the boundary becomes a finite value, i.e., $g_h(\mathcal{S}_b) = g_h(\mathcal{S}_t) = g_h(\eta)$, which is beneficial for further optimization. The projection of the point $\mathbf{p}$ outside can be defined in the same way as that of the projection of its nearest point $\mathcal{NP}(\mathbf{p})$ on the shell boundary, i.e. $\mathcal{M}_{\mathcal{S}}(\mathbf{p}) = \mathcal{M}_{\mathcal{S}}(\mathcal{NP}(\mathbf{p}))$.

Note that the shell may contain singularity vertices whose 1-ring neighboring triangles do not associate prism elements, and thus no scalar field is defined there. If a curve crosses over such regions with a segment $\langle \mathbf{P}(t_{k_1}), \mathbf{P}(t_{k_2}) \rangle$, we set the scalar value for the point $\mathbf{P}(t)$ $\left(t \in (t_{k_1}, t_{k_2})\right)$ by linear interpolation:

$$d\left(\mathbf{P}(t)\right) = \frac{t_{k_2} - t}{t_{k_2} - t_{k_1}} d\left(\mathbf{P}(t_{k_1})\right) + \frac{t - t_{k_1}}{t_{k_2} - t_{k_1}} d\left(\mathbf{P}(t_{k_2})\right). \quad (13)$$

4.3 Curve optimization

With a fixed knot vector $\mathcal{T}$, we could numerically solve the optimization problem (7). Though we could calculate the energy term E_{int} (4) by using an analytical method since the B-spline curve has a closed-form expression, we instead discretized both differentials with the finite difference method to reduce computation time, i.e. $\mathbf{P}'(t_i) = (\mathbf{q}_i - \mathbf{q}_{i+1})/(t_i - t_{i+1})$.

Therefore, we evenly sampled points on the parametric line, $\mathbf{T}' = \{t'_i\}$, as well as the corresponding points on the curve $\mathbf{Q} = \{\mathbf{q}_i = \mathbf{P}(t'_i)\}$. We then discretized the integrals of all energy terms by Gaussian quadrature. Suppose M points are sampled on the curve, and each point is associated with a segment. The integral on each segment can be calculated by sampling new points and accumulating their quantities.

Thus, we could discretize the internal energy E_{int} (4) as follows:

$$E_{\text{int}}(\mathcal{P}) = \frac{1}{ML^2} \sum_{i=1}^{M} \sum_{j=1}^{M'} \omega_j (\lambda \|\mathbf{P}'(t'_{i,j})\|^2 + (1-\lambda)\|\mathbf{P}''(t'_{i,j})\|^2), \quad (14)$$

Algorithm 2 Control polygon generation for the initial B-spline

Require: a mesh $\mathcal{S}$ with a shell space $\Omega_{\mathcal{S}}$, and a sequence of interpolatory points $\mathbf{X} = \{\mathbf{x}_{i=1}^{N} \in \mathcal{S}\}$.
Ensure: a control polygon $\mathcal{C}$ inside the shell space $\Omega_{\mathcal{S}}$.

$\mathcal{C} := \{\mathbf{c}_1 = \mathbf{x}_1\}$
$flag = f(\mathbf{x}_2) > 0\ ?\ 1 : 0$
for $j := 2, \cdots, N-1$ **do**
  $\mathbf{c}_j := flag\ ?\ \mathcal{M}_{\mathbf{T}}(\mathbf{x}_j) : \mathcal{M}_{\mathbf{B}}(\mathbf{x}_j)$
  $flag := !flag$
  if $\overline{\mathbf{c}_{j-1}\mathbf{c}_j} \cap \partial\Omega_{\mathcal{S}} \neq \emptyset$ **then**
    $R := \text{Dijsktra}(\mathbf{x}_{j-1}, \mathbf{x}_j)$
    $\mathbf{x}_m := \text{Mid}\left(R(\mathbf{x}_{j-1}, \mathbf{x}_j)\right)$
    $\mathbf{c}_m := flag\ ?\ \mathcal{M}_{\mathbf{T}}(\mathbf{x}_m) : \mathcal{M}_{\mathbf{B}}(\mathbf{x}_m)$
    $\mathcal{C} := \mathcal{C} \cup \{\mathbf{c}_m\}$
    $flag := !flag$
    $\mathbf{c}_j := flag\ ?\ \mathcal{M}_{\mathbf{T}}(\mathbf{x}_j) : \mathcal{M}_{\mathbf{B}}(\mathbf{x}_j)$
    $flag := !flag$
    while $\overline{\mathbf{c}_m\mathbf{c}_j} \cap \partial\Omega_{\mathcal{S}} \neq \emptyset$ **do**
      $\mathbf{x}_m := \text{Mid}\left(R(\mathbf{x}_m, \mathbf{x}_j)\right)$
      $\mathbf{c}_m := flag\ ?\ \mathcal{M}_{\mathbf{T}}(\mathbf{x}_m) : \mathcal{M}_{\mathbf{B}}(\mathbf{x}_m)$
      $\mathcal{C} := \mathcal{C} \cup \{\mathbf{c}_m\}$
      $flag := !flag$
      $\mathbf{c}_j := flag\ ?\ \mathcal{M}_{\mathbf{T}}(\mathbf{x}_j) : \mathcal{M}_{\mathbf{B}}(\mathbf{x}i.e.,j)$
    end while
  end if
  $\mathcal{C} := \mathcal{C} \cup \{\mathbf{c}_j\}$
end for
$\mathcal{C} := \mathcal{C} \cup \{\mathbf{x}_N\}$
return $\mathcal{C}$

where ω_j is the weight of Gaussian quadrature. With the same quadrature scheme, the external energy E_{ext} (5) can also be rewritten as:

$$E_{\text{ext}}(\mathcal{P}, \mathcal{I}) = \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{M'} c\left(\mathcal{I}\left(\mathcal{M}\left(\mathbf{P}(t'_{i,j})\right)\right)\right). \tag{15}$$

And the distance energy is discretized as well:

$$E_{\text{dist}}(\mathcal{P}) = \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{M'} \omega_j g_h(d(\mathbf{q}_{i,j})). \tag{16}$$

To minimize (7), we adopted the interior-point method. Starting from an initial B-spline curve (Sect. 4.2), we iteratively updated the positions of control points by using the Newton's solver with the following step:

$$\mathcal{P}^{m+1} \longleftarrow \mathcal{P}^m + \alpha_m H_m^{-1} \nabla E(\mathcal{P}, \mathcal{T}), \tag{17}$$

where H_m is the $3(n+1) \times 3(n+1)$ Hessian matrix of $E(\mathcal{P})$ evaluated at $\mathcal{P}^m$, m is the index of the iteration, and $\alpha_m \in (0, 1]$ is the step size calculated with the backtracking line search strategy. In each iteration, we needed to frequently find the tetrahedron in which each updated sampled point is located. Based on the observation that the next sampled point is likely to be located to the same or neighboring tetrahedron, we thus adopted the ray-tracing method instead of spatial searching (e.g., an AABB tree), e.g., tracing the tetrahedron from the ray's starting point to its ending point by computing all the intersections between the ray and shell cells. This method is faster than spatial searching [8] since we used a simplified shell, and reduces the number of intersections while brings considerable efficiency improvements.

It is easy to verify that the energy terms $E_{\text{int}}(\mathcal{P})$, $E_{\text{ext}}(\mathcal{P}, \mathcal{I})$ and $E_{\text{fit}}(\mathcal{P})$ are all quadratic w.r.t. $\mathcal{P}$, and the term $E_{\text{dist}}(\mathcal{P})$ is also convex. Thus, the Hessian of the whole energy H is positive-definite and its inverse H^{-1} in (17) always exists during the iterations. This nice property ensures the convergence of the optimization.

Equation (7) has several weighting parameters. For different cases, users may need to adjust these parameters accordingly to achieve convergence. To make this process more robust, we applied an adaptive weighting scheme before the new optimization. By fixing the weight of the energy terms E_{int} and E_{ext}, we adaptively updated the weight of the term E_{dist} as follows:

$$\alpha^{\text{new}} = \min\left\{\alpha_{\min} \max\left\{\frac{E_{\text{int}} + \mu E_{\text{ext}} + \beta E_{\text{fit}}}{E_{\text{dist}}}, 1\right\}, \alpha_{\max}\right\}, \tag{18}$$

where we heuristically set $\alpha_{\min} = 1$, $\alpha_{\max} = 10^8$. To satisfy the interpolation constraint, we could also adaptively adjust the weight of E_{fit} by using a similar scheme as above, where we set $\beta_{\min} = 10$, $\beta_{\max} = 10^{10}$ heuristically.

4.4 Knot insertion

It is known that a fixed number of knot points determined at the beginning may not be sufficient to optimize the curve to be close enough to the mesh surface. We thus present an adaptive scheme to insert new knot points to increase the degree of freedom and optimized its distribution while fixing the B-spline curve $\mathcal{P}$. Inspired by the recent knot-insertion work [15, 26], where they use a feature function to guide the insertion operation, we defined a new feature function specifically for our problem.

We denoted the projection of the B-spline curve on the surface as "shadow curve", which is considered the optimal approximation on the mesh or the ground-truth curve. We thus measured the difference between the B-spline curve and its shadow curve, with which a feature function, known as cumulative distribution function (CDF) can be defined:

$$\begin{aligned} F(t) = &\int_0^t \lambda \|\mathbf{P}'(t) - \mathbf{Q}'(t)\|^2 dt + \\ &\int_0^t (1-\lambda) \|\mathbf{P}''(t) - \mathbf{Q}''(t)\|^2 dt + \\ &\int_0^t \alpha g(d(\mathbf{P}(t))) dt, \end{aligned} \tag{19}$$

where $\mathbf{Q}(t)$ is the shadow of $\mathbf{P}(t)$ on the mesh surface.

The above-defined CDF tells us the information about the amount of difference, which can be used to guide the insertion operation. Intuitively, more knots should be inserted for intervals with a more significant difference and vice versa.

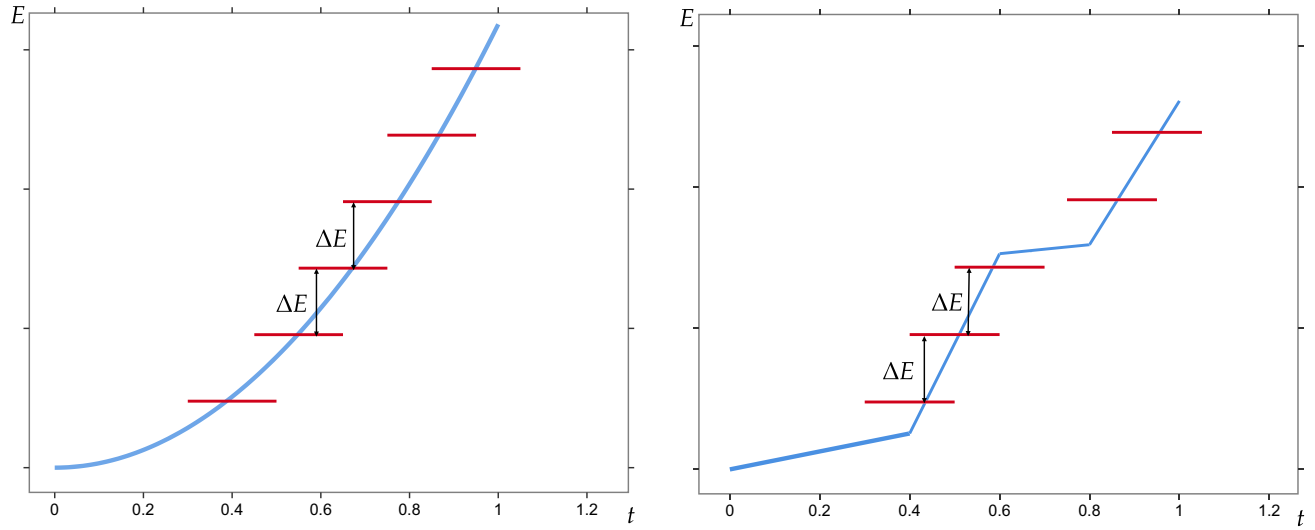
In practice, we carried out Gaussian quadrature for the integral (19) and calculate the feature value for each knot interval:

$$\Delta F_i = F(t_{i+1}) - F(t_i). \tag{20}$$

We then divided ΔF_i equally into several sub-intervals and inserted knots at the position of the corresponding points, as illustrated in Fig. 4a.

We computed the integration for each knot interval to simplify the computation and determine the number of new knots in the interval (t_i, t_{i+1}). The number of new knots that need to be inserted to the interval is calculated as $\Delta F_i / \Delta_{\text{avg}} F$. Then, the new knots are placed uniformly in each interval (see Fig. 4b).

$$\Delta_{\text{avg}} F = \int_0^1 F(t) dt / l, \tag{21}$$



(a) Gaussian integral of continuous function (b) Gaussian integral in discrete case

Fig. 4 The CDF is divided by several constant functions (Red) of the interval ΔF, so that knots can be placed at the breakpoints of their intersections (**a**). The number of new knots in an interval is the number of intersections between CDF (blue and constant functions (Red)) (**b**)

$$l = rN_{\text{int}}, \tag{22}$$

where N_{int} is the number of intervals and r is a user prescribed weight ($r = 1$ by default).

To avoid inserting too many knots, we rejected a knot into the interval with a small span:

$$t_{i+1} - t_i < \varepsilon_{\text{int}}, \tag{23}$$

where ε_{int} is the minimal span of an interval. And we applied the following three stopping conditions for the adaptive knot insertion:

1. The integral value of all intervals is less than the threshold ε_d;
2. $\left|\Delta F^{n+1} - \Delta F^n\right| < \varepsilon_d$ (n is the iteration index);
3. $t_{i+1} - t_i < \varepsilon_{\text{int}}$ for all intervals (no knot can be inserted any more).

Iterations stop when any of conditions are met.

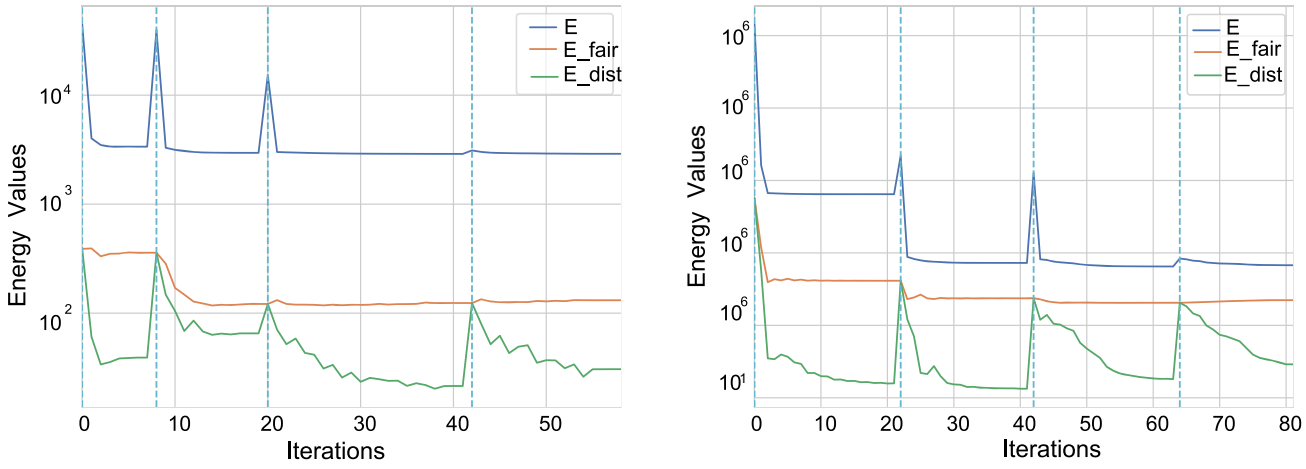
When the B-spline curve is close enough to the mesh surface, we needed to project it to the surface to obtain a polyline with the operator provided by Jiang et al. [8]. To ensure the curve being strictly on the surface, we had to sequentially project each sampled segment $\overline{\mathbf{p}_i\mathbf{p}_{i+1}}$ to the mesh $\mathcal{S}$. If the projected segment $\overline{\mathcal{M}(\mathbf{p}_i)\mathcal{M}(\mathbf{p}_{i+1})}$ is not on the same mesh face, we took $\mathcal{M}(\mathbf{p}_i)$ as the starting point and projected $\mathbf{p}_{i+1}$ to that face so as to find the the intersection point $\mathbf{p}'_i$ between the new segment and the face boundary. Then we inserted $\mathbf{p}'_i$ to the polyline and took it as the new starting point, and repeated the procedure until the intersection point $\mathbf{p}'_i$ and $\mathcal{M}(\mathbf{p}_{i+1})$ share the same face.

5 Experiments

We implemented the above algorithm as a single-threaded C++ application on a PC equipped with a 4.1 GHz Intel Core i5 CPU and 32GB of RAM. Linear algebra-related computation is carried out by Eigen lib [4]. We validated our algorithm on many meshes and compared it with state-of-the-art works. In the experiment, we set default parameters as follows, $\mu = 1, r = 1, k = 3$, which works well in most cases.

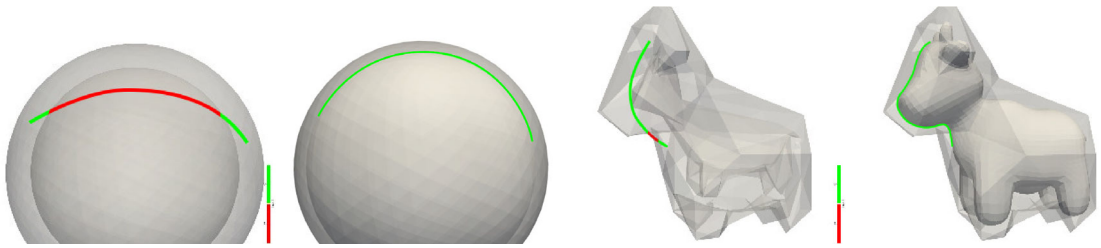
To validate the convergence behavior of our algorithm, we tested several examples (some results can be seen in Figs. 14c and 15d) and drew their energy-value graphs during the iterations (see Fig. 5). In the figure, the dotted lines (peaks of curves) indicate the new starting points of the optimization, where the energy values change sharply. This is because new knots are inserted, the weights of energy terms are updated. In each round of optimization, we solved a convex problem that guarantees convergence.

In practice, the initial curves generated by our scheme rarely appear to be outside the shell space (4.2). But we provided the scheme to handle the case when parts of the initial curve are outside. And it can still succeed to optimize practically. Figure 6 shows two of the examples which are constructed by omitting the adaptive knot insertion scheme (Algorithm. 2), and our algorithm is capable of optimizing both curves to the state that lies on the mesh surface.



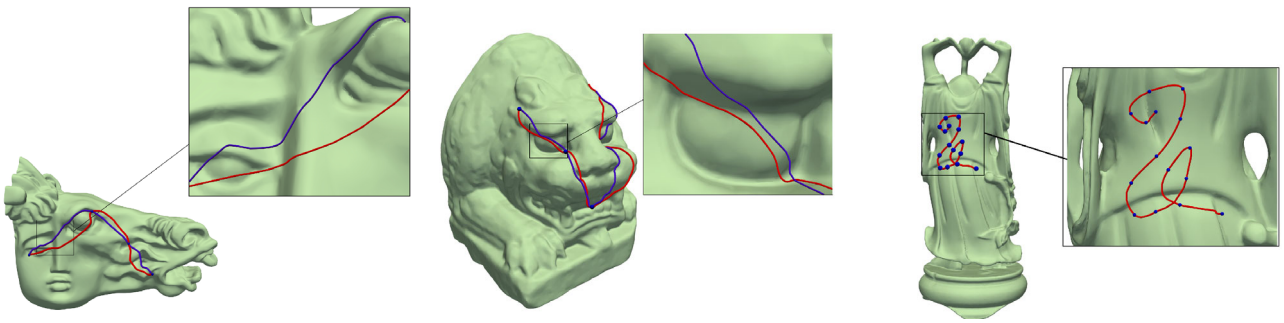
(a) Energy graph for the armadillo model (b) Energy graph for the cup model

Fig. 5 The convergence graphs of two curves. The peaks denote the starting points of the new optimization



(a) Initial curve I (b) Final curve I (c) Initial curve II (d) Final curve II

Fig. 6 Two examples whose initial curves are partially outside the shell space



(a) Curves on the horse head model (b) Curves on the lion model (c) The curve on the Maitreya model

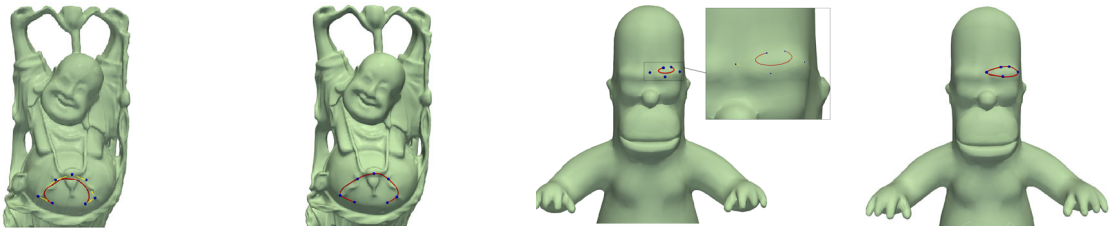
Fig. 7 Comparison result with the method [21]. The red and purple curves are results of ours and the method [21], respectively

5.1 Curve design without scalar fields

When the parameter μ in (7) is set to zero, i.e. the scalar field is not considered, our algorithm degenerates to solving the traditional curve interpolation problem on a surface. We showed several examples that benefits from our algorithms in the following sections.

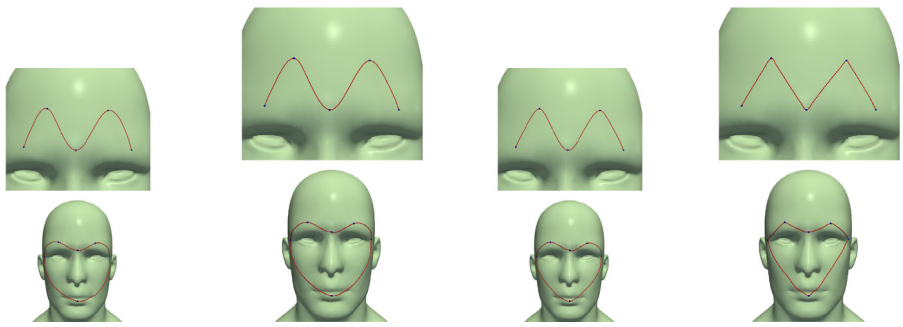
Our algorithm can control the curve shape flexibly by tuning the parameter λ. Figure 9 shows the results for both open and closed curves by adjusting λ from 1 to 0. It can be seen that the results present approximated piece-wise straightest paths when $\lambda = 1$ for both cases, which is in line with the explanation of [6, 9]. As λ decreases, the effect of the ℓ_1 norm derivatives gradually becomes dominant, and the resulted curves become smoother.

Our algorithm can generate B-spline curves with different orders on the mesh surface. Figure 10 displays the results of curves with order $k = 2, 3, 4$ for the same set of interpola-



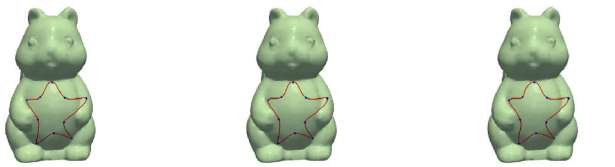
(a) The open curve on the Maitreya model (b) The open curve on the homer model

Fig. 8 Comparison with the method [24]. The left sub-figure is the result of method [24], where the control points, control polygon and the Bézier curve are colored in blue, yellow and red, respectively. The blue points are the interpolatory points of our method. The red curve is the resulting curve



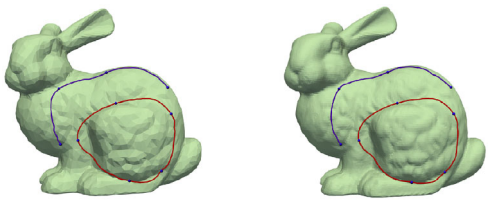
(a) $\lambda = 0$ (b) $\lambda = 0.4$ (c) $\lambda = 0.7$ (d) $\lambda = 1$

Fig. 9 Curves with different values of λ on the face model for both open and closed curves



(a) 2nd-order (b) 3rd-order (c) 4th-order

Fig. 10 Comparison of generated B-splines with different orders but the same set of interpolatory points



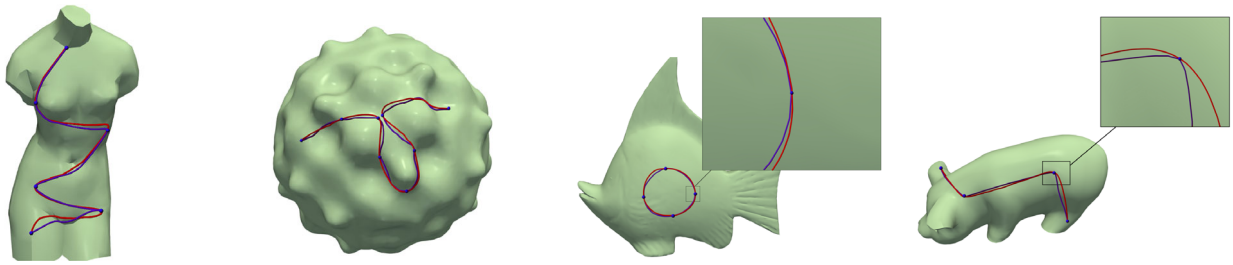
(a) Low-resolution result (b) High-resolution result

Fig. 11 Comparison of results with the same interpolatory points on meshes and different resolutions. The left bunny has 3485 vertices and 6966 faces, while the right bunny has 34834 vertices and 69451 faces

tory points ($\lambda = 0.5$). It can be seen that the shapes of the three curves are very similar in general. However, we found that fewer knots are sufficient for higher orders, in this case they need 113, 91, 59 knots, respectively. It is consistent with the fact that higher order curves are more flexible, and thus can interpolate the same set of points with fewer knots. Noting that B-splines with higher orders may produce unstable results for designing feature-aware curves.

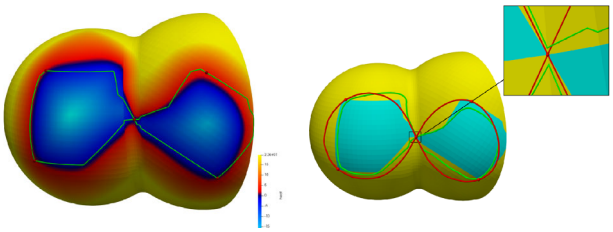
Our algorithm produces similar results on the same model with different resolutions. An example can be seen in Fig. 11, where the same set of interpolatory points are prescribed in two bunny models. The results show that the open and closed curves on the two meshes display visually similar shapes. This demonstrates that the resolution of the mesh has little impact on the designed curve by our method.

Our algorithm has several advantages over polyline-based algorithms. We compared our approach with the state-of-the-art algorithm [9]. Figure 12 shows the results, where the curves of both algorithms are densely sampled with a similar number of points. Both results are nice, but ours are slightly smoother near the interpolatory points. Besides, our algorithm can sample points on the calculated smooth spline curve in high resolutions at a meager cost. In contrast, it will take much time for the algorithm [9] since it needs to recompute the curve with more variables. In Fig. 12a, we set $\lambda = 0.5$ and $M = 200$. After the B-spline curve is generated, we resampled 5132 points on it. The whole process costs 320ms, but it takes 731ms to generate the curve with the same number of sampling points for [9]. A similar result can also be seen in Fig. 12b, where the two curves are sampled with a relatively low rate, e.g., 600([9]) and 584(ours) points in Fig. 12b, respectively.



(a) Curves on the Venus model. (b) Curves on the bumpy sphere model (c) Curves on the fish model (d) Curves on the pig model

Fig. 12 Comparison results with the method [9]. Curves of our method and Jin et al.'s method [9] are colored in red and yellow, respectively



(a) The initial curve ([17]) (b) Generated Curves

Fig. 13 Comparison results with the heat-flow method [17]

Next, we designed curves on meshes with a lower sampling rate. Examples can be seen in Fig. 12c, d, where 358 ([9]) and 288 (ours) points are sampled, respectively, for the former, and 165 ([9]) 119 (ours) for the later. It shows that our algorithm can still keep smoothness even though the sampling rate of the curve is remarkably reduced, while the algorithm [9] suffers from obvious aliasing problems.

Compared to the implicit algorithms, e.g., heat-flow based method [17], our algorithm can be used to design complicated curves with self-intersections (see the red curve in Fig. 13b), while the implicit algorithms cannot handle such cases (see the green curve in Fig. 13a, b) because the curve is extracted from the level set of a certain smooth scalar field. Besides, our algorithm can generate much smoother results than [17], since in their method smoothness of the result depends on the mesh resolutions.

As is known, splines have excellent properties and are frequently used to design curves on meshes. We thus compared ours with some representative works [21, 24] to show the advantages of our approach. The algorithm [21] can be used to design B-spline curves on the embedded high-dimensional Euclidean space, and it can generates similar results as our method (Fig. 7). However, their algorithm projects the sampled points onto the mesh but cannot guarantee that the whole curve is on the mesh. Moreover, the projection operator is not robust. As is shown in Fig. 7a, b, both curves generated by [21] break at local segments due to the projection failure while ours can always keep the continuity of the curve which is guaranteed by the robust projection $\mathcal{M}$. Another example is shown in Fig. 7c, where their algorithm fails to obtain the whole curve, whereas our approach successfully generates curves with good quality. We then compared our approach with the algorithm [24] which computes geodesic Bézier curves via the generalized De Casteljau method. However, this kind of approach lacks user controls, e.g., interpolating given points (Fig. 8a), adjusting curve shapes, and designing smooth closed curves (Fig. 8b). Moreover, none of them [21, 24] can extend to capture the feature of a mesh via a given scalar field.

5.2 Feature-aware curve design

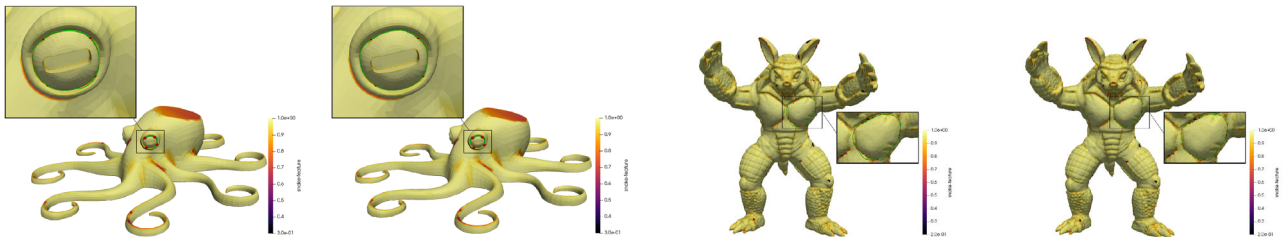
With the equivalent distance field defined in shell space, our algorithm can be extended and used to design feature-aware curves by simply augmenting solely mesh-dependent energies. Users can design specific scalar fields on the mesh in different applications. Our algorithm can drive the curve to regions with lower scalar values while keeping other constraints. Next we gave several examples.

In the first example, feature lines are needed to capture in concave regions on mesh. We thus applied normal variations of the neighboring faces to associate a scalar value at each vertex of the mesh [13] as follows,

$$\mathcal{I}(v) = \min_{f} \left\{ \mathbf{n}_v^T \mathbf{n}_f \right\}, \tag{24}$$

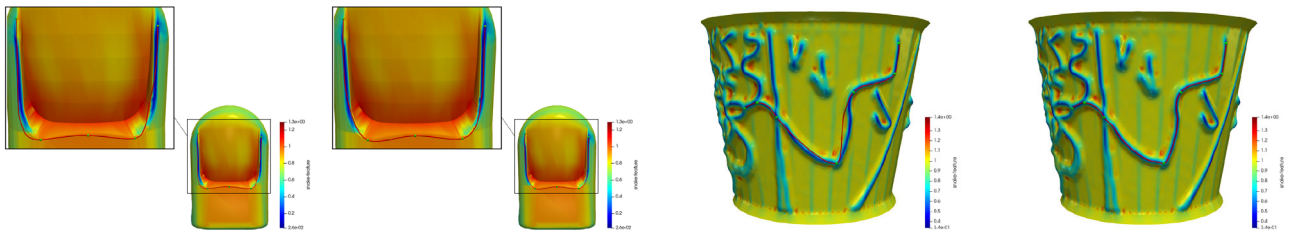
where $\mathbf{n}_v$ is the normal of vertex v and $\mathbf{n}_f$ is the normal of its neighboring face f. And the composite function $c(\cdot)$ is defined as a square function, e.g., $c(x) = x^2$. The result can be seen in Fig. 14, where after augmenting the feature-aware external energy, the algorithm can snap the curve to the feature region well for both the octopus (eyeballs in Fig. 14b) and armadillo (pectoral muscles in Fig. 14d) models.

In the second example, we had to extract sharp feature lines of meshes. We then set another feature field, i.e. maximal principal curvature field on the mesh, and the composition of the scalar value for each vertex v is defined as,



(a) Result with $\mu = 0$ (b) Result with $\mu = 1$ (c) Result with $\mu = 0$ (d) Result with $\mu = 1$

Fig. 14 Results of the designed curves on the octopus and armadillo models with and without the external energy. The color maps denote the defined scalar fields



(a) Result with $\mu = 0$ (b) Result with $\mu = 1$ (c) Result with $\mu = 0$ (d) Result with $\mu = 1$

Fig. 15 Results of the designed curves on the finger and cup models with and without the external energy. The color maps denote the defined scalar fields

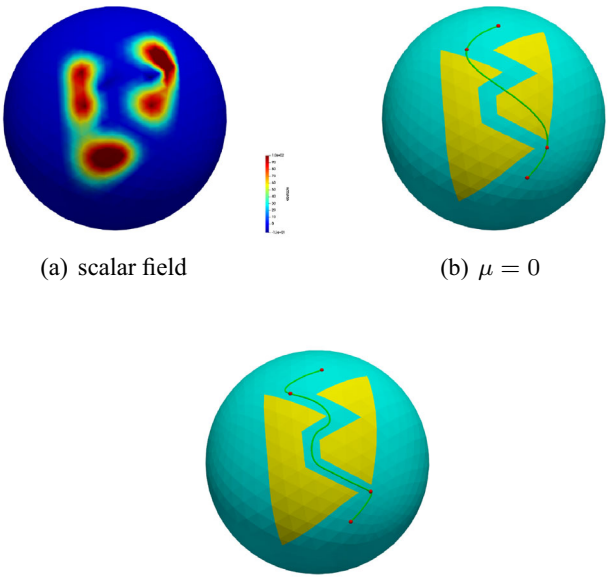
$$c(\mathcal{I}(v)) = e^{-\kappa_{\max}(v)}, \tag{25}$$

where $\kappa_{\max}$ is the maximal principal curvature. The curves on both the finger nail model (Fig. 15a) and vase model with geometric textures (Fig. 15c) are generated by ignoring the feature-aware energy, and they cannot capture the features. In contrast, after considering the feature energy, the curves on both models exactly correspond to the features, as shown in Fig. 15b and Fig. 15d).

In the third example, we designed a curve in the presence of obstacles on the mesh surface to avoid the obstacles. To achieve such a goal, we constructed a scalar field where large scalar values are set on the obstacle regions R and zero values for the remaining regions. To encourage stable convergence for our algorithm, we calculated a smooth scalar field in obstacle regions by solving Bi-Laplacian equations under boundary conditions to propagate the boundary values to the inner region:

$$\Delta^2 \mathcal{I} = 0, \quad \mathcal{I}|_{\partial R} = s, \quad \nabla \mathcal{I}|_{\partial R} = \mathbf{n}, \tag{26}$$

where $\mathbf{n}$ is the vector field defined on the boundary of the obstacle regions and is perpendicular to the boundary. Figure 16a shows the scalar field calculated by the above equation, where the red-colored region denotes the obstacle corresponding to the yellow region in Fig. 16b. We selected four interpolatory points between two obstacles. Moreover, the curve generated by our feature-aware algorithm avoids them successfully (Fig. 16c) while it crosses the obstacles by using the algorithm without the feature-aware energy (Fig. 16b).



(a) scalar field (b) $\mu = 0$ (c) $\mu = 1$

Fig. 16 An obstacle avoiding example

5.3 Performance

The main computational cost of our algorithm mainly consists of two parts. One is the step of curve updating. Our

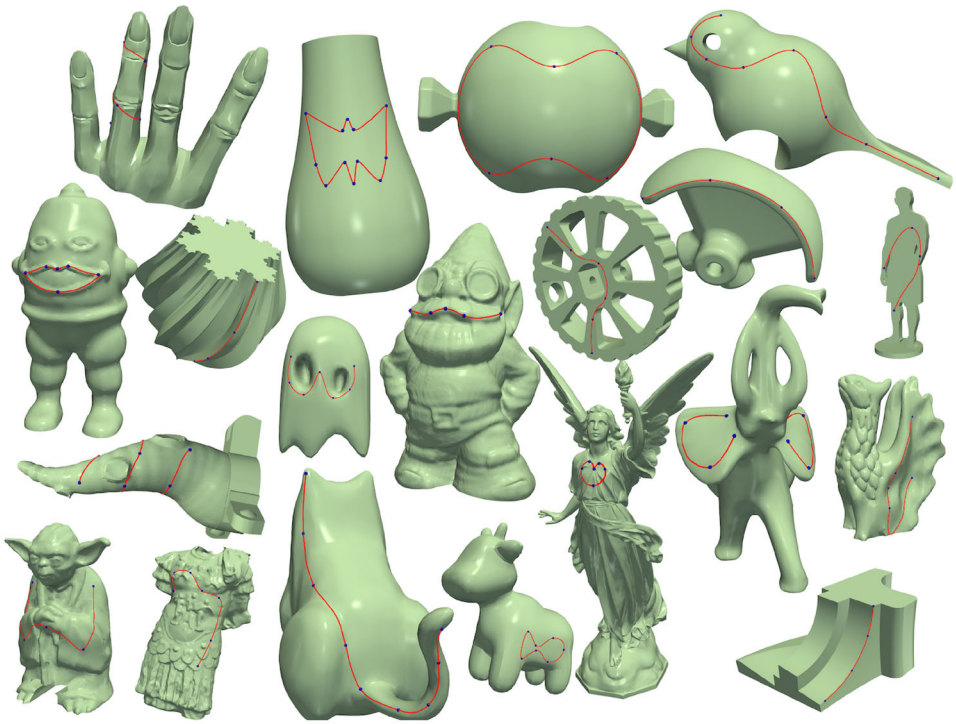


Fig. 17 A gallery of models and spline curves created with our method

Table 1 Statistics: The columns from left to right show the number of vertices of mesh models (interpolatory points), knots for the initial and the final curve, and the computational times

Model	#V/N	Knots (I)	Knots (F)	T (s)
Ghost (Fig. 17)	1K(7)	10	32	0.073
Bunny (Fig. 11a)	3K(4)	8	34	0.098
Squirrel (Fig. 10b)	8K(10)	23	91	0.863
Elephant (Fig. 17)	24K(4)	9	51	0.497
Armadillo (Fig. 14c)	50K(6)	11	23	1.023
Cup (Fig. 15c)	300K(7)	11	39	3.228
Cup (Fig. 15d)	300K(7)	11	71	50.981

algorithm has fewer variables than polyline-based methods in general. However, it requires additional efforts for knot insertion and re-optimization of the curve, which needs to change the matrix structure of Hessian and pay much cost in solving Newton's step (17). The other is the step of projecting any point inside the shell to the mesh surface, which is required in evaluating point-to-surface distance and the scalar value of the point. Though the AABB tree has accelerated the projection, frequent evaluations especially on high-resolution meshes still take much time. Table 1 shows the performance of our algorithm for some of the testing examples (most of other examples cost around 1s). As is seen from the table, mesh resolution and knot number of the curve primarily affect the efficiency. Moreover, the feature-aware algorithm usually takes more time than the version without it. It is because the former one needs to calculate the scalar values for the sampling points, which involves projection.

5.4 Applications

Our curve design method is robust and flexible and can be used in many applications, e.g., drawing patterns, feature extraction, cutting holes, and segmentation on surface meshes. Here we provided more pattern drawing and feature extraction examples on different models, as shown in the gallery (Fig. 17).

6 Conclusion

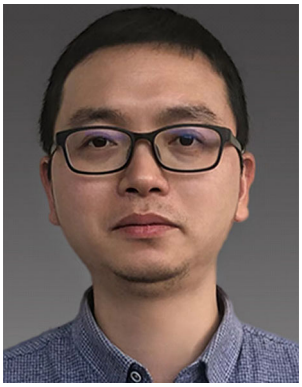
We proposed a variational B-spline curve design algorithm on surface meshes by using the scalar field defined in the shell space. The algorithm can robustly generate a smooth feature-aware curve at any resolution. It is also flexible for different controls, e.g., shape of the curve, interpolations, etc. Although our adaptive knot-insertion scheme is effective in practice, we had not dived into an optimal scheme for knot optimization. And how to insert as few knots as possible and optimize their positions is our future work.

Acknowledgements We would like to thank all the anonymous reviewers for their valuable comments. We would also like to thank Wanqiang Shen from Jiannan University for her helpful discussions. This work was supported by Key R&D Programs of Zhejiang Province (No. 2022C01220,2023C01224) and National Natural Science Foundation of China (No. 61702458) and the Fundamental Research Funds of Zhejiang Sci-Tech University (No. 23232106-Y). Yun Zhang was partially supported by the Zhejiang Province Public Welfare Technology Application Research (No. LGG22F020009), Key Lab of Film and TV Media Technology of Zhejiang Province (No. 2020E10015), and Teaching Reform Project of Communication University of Zhejiang (No. jgxm202131).

References

1. Cheng, L.T., Burchard, P., Merriman, B., Osher, S.: Motion of curves constrained on surfaces using a level-set approach. J. Comput. Phys. **175**(2), 604–644 (2002)
2. Dung, V.T., Tjahjowidodo, T.: A direct method to solve optimal knots of b-spline curves: An application for non-uniform b-spline curves fitting. PLoS ONE **12**(3), 1–24 (2017)
3. Estrada Sarlabous, J., Hernández Mederos, V., Martínez Morera, D., Velho, L., López Gil, N.: Conic-like subdivision curves on surfaces. Vis. Comput. **28**, 971–982 (2012)
4. Guennebaud, G., Jacob, B., et al.: Eigen. URl: http://eigen.tuxfamily.org**3** (2010)
5. Hofer, M.: Constrained optimization with energy-minimizing curves and curve networks: a survey. In: Proceedings of the 23rd Spring Conference on Computer Graphics, pp. 27–35 (2007)
6. Hofer, M., Pottmann, H.: Energy-minimizing splines in manifolds. ACM Trans. Graph. (TOG) pp. 284–293 (2004)
7. Ji, Z., Liu, L., Chen, Z., Wang, G.: Easy mesh cutting. Comput. Graph. Forum **25**(3), 283–291 (2006)
8. Jiang, Z., Schneider, T., Zorin, D., Panozzo, D.: Bijective projection in a shell. ACM Trans. Graph. **39**(6), 1–18 (2020)
9. Jin, Y., Song, D., Wang, T., Huang, J., Song, Y., He, L.: A shell space constrained approach for curve design on surface meshes. Comput. Aided Des. **113**, 24–34 (2019)
10. Jung, M., Kim, H.: Snaking across 3d meshes. In: 12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings., pp. 87–93. IEEE (2004)
11. Kang, H., Chen, F., Li, Y., Deng, J., Yang, Z.: Knot calculation for spline fitting via sparse optimization. Comput. Aided Des. **58**, 179–188 (2015)
12. Lawonn, K., Gasteiger, R., Rössl, C., Preim, B.: Adaptive and robust curve smoothing on surface meshes. Comput. Graph. **40**, 22–35 (2014)
13. Lee, Y., Lee, S.: Geometric snakes for triangular meshes. Comput. Graph. Forum **21**(3), 229–238 (2002)
14. Lee, Y., Lee, S., Shamir, A., Cohen-Or, D., Seidel, H.P.: Mesh scissoring with minima rule and part salience. Comput. Aided Geom. Des. **22**(5), 444–465 (2005)
15. Liang, F., Zhao, J., Ji, S., Fan, C., Zhang, B.: A novel knot selection method for the error-bounded b-spline curve fitting of sampling points in the measuring process. Meas. Sci. Technol. **28**(6), 065,015 (2017)
16. Liu, Z., Zhang, H., Wu, C.: On geodesic curvature flow with level set formulation over triangulated surfaces. J. Sci. Comput. **70**(2), 631–661 (2017)
17. Livesu, M.: A heat flow based relaxation scheme for n dimensional discrete hyper surfaces. Comput. Graph. **71**, 124–131 (2018)
18. Mancinelli, C., Nazzaro, G., Pellacini, F., Puppo, E.: b/surf: Interactive b\'ezier splines on surface meshes. IEEE Trans. Comput. Graph. Vis. (to be appear) (2023)
19. Mohanty, S.D., Fahnestock, E.: Adaptive spline fitting with particle swarm optimization. Comput. Stat. **36**(1), 155–191 (2021)
20. Morera, D.M., Carvalho, P.C., Velho, L.: Modeling on triangulations with geodesic curves. Vis. Comput. **24**(12), 1025–1037 (2008)
21. Panozzo, D., Baran, I., Diamanti, O., Sorkine-Hornung, O.: Weighted averages on surfaces. ACM Trans. Graph. **32**(4), 1–12 (2013)
22. Park, F.C., Ravani, B.: Bézier curves on Riemannian manifolds and lie groups with kinematics applications. In: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. 12846, pp. 15–21. American Society of Mechanical Engineers (1994)
23. Pottmann, H., Hofer, M.: A variational approach to spline curves on surfaces. Comput. Aided Geom. Design **22**(7), 693–709 (2005)
24. Sharp, N., Crane, K.: You can find geodesic paths in triangle meshes by just flipping edges. ACM Trans. Graph. **39**(6), 1–15 (2020)
25. Wallner, J., Pottmann, H.: Intrinsic subdivision with smooth limits for graphics and animation. ACM Trans. Graph. **25**(2), 356–74 (2006)
26. Yeh, R., Nashed, Y.S., Peterka, T., Tricoche, X.: Fast automatic knot placement method for accurate b-spline curve fitting. Comput.-Aided Design **128**, 102,905 (2020)
27. Zheng, W., Bo, P., Liu, Y., Wang, W.: Fast B-spline curve fitting by L-BFGS. Comput. Aided Geom. Design **29**(7), 448–462 (2012)

Rongyan Xu is currently a software engineer at Xingxin Technology Co., Ltd. He received M.S. degree at the School of Computer Science and Technology (School of Artificial Intelligence), Zhejiang Sci-Tech University, China, in 2022. His research interests include computer graphics and geometry processing.

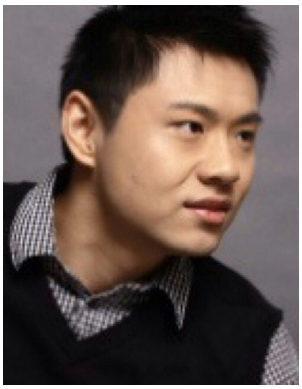


Yao Jin is currently an associate professor at the School of Computer Science and Technology (School of Artificial Intelligence), Zhejiang Sci-Tech University, China. He received his doctoral degree from Zhejiang University in 2015. His research interests include computer graphics and geometry processing. He is a member of CCF.

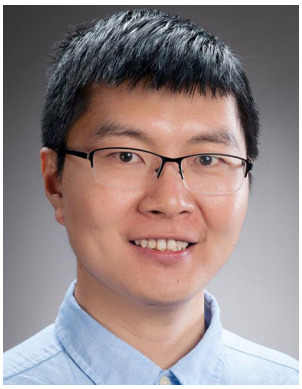
Huaxiong Zhang is currently a professor at the School of Computer Science and Technology (School of Artificial Intelligence), Zhejiang Sci-Tech University. His research interests include intelligent information processing and generation of digital art graphics.

Yun Zhang is currently a Professor at the Communication University of Zhejiang in China. He received his doctoral degree from Zhejiang University in 2013. Before that, he received Bachelor and Master degrees from Hangzhou Dianzi University in 2006 and 2009, respectively. He visited the Visual Computing Group of Cardiff University in 2018 and 2023, and the Computational Media Innovation Centre (CMIC) of Victoria University of Wellington in 2019. His research interests include Computer Graphics, Image and Video Editing, Computer Vision. He is a senior member of CCF.

Yu-Kun Lai received his bachelor's degree and PhD degree in computer science from Tsinghua University in 2003 and 2008, respectively. He is currently a Professor in the School of Computer Science & Informatics, Cardiff University. His research interests include computer graphics, geometry processing, image processing and computer vision. He is on the editorial boards of Computer Graphics Forum and The Visual Computer.



Zhe Zhu is currently a Senior Software Engineer at Mathworks. Before that he was a Senior Research Associate at Duke University. He got his Ph.D. in the Department of Computer Science and Technology, Tsinghua University in 2017. He received his bachelor's degree in Wuhan University in 2011. His research interests are in computer vision and computer graphics.



Fang-Lue Zhang is currently a senior lecturer with Victoria University of Wellington, Wellington. He received the Bachelor's degree from Zhejiang University, Hangzhou, in 2009, and the Doctoral degree from Tsinghua University, Beijing, in 2015. His research interests include image and video editing, mixed reality, and image-based graphics. He is a member of IEEE and ACM. He received Victoria Early Career Research Excellence Award in 2019. He is on the editorial board of Computer & Graphics. He is a committee member of IEEE Central New Zealand sector.