High-quality quad-mesh generation for self-intersecting parametric surfaces

Haolei Nan
Hangzhou Dianzi University
Gang Xu
gxu@hdu.edu.cn
Hangzhou Dianzi University
Haiyan Wu
Hangzhou Dianzi University
Renshu Gu
Hangzhou Dianzi University
Jinlan Xu
Hangzhou Dianzi University
Yang Liu
China Aerodynamics Research and Development Center

Research Article

Keywords: Self-intersecting surfaces, Quad partitioning, Quad meshing, Feature preservation

Posted Date: July 4th, 2024

DOI: https://doi.org/10.21203/rs.3.rs-4599197/v1

Additional Declarations: No competing interests reported.

High-quality quad-mesh generation for self-intersecting parametric surfaces

Haolei Nan[1], Gang Xu[1*], Haiyan Wu[1], Renshu Gu[1], Jinlan Xu[1], Yang Liu[2]

[1*]College of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, 310018, Zhejiang, China.
[2]China Aerodynamics Research and Development Center, Mianyang, 621000, Sichuan, China.

*Corresponding author(s). E-mail(s): gxu@hdu.edu.cn;
Contributing authors: nanhaolei@hdu.edu.cn;

Abstract

We propose a method for the high-quality quadrangulation of an input self-intersecting parametric surface. We first compute the algebraic representation of self-intersecting curves and cusp curves of the surface by the moving plane method. Sampling points are then generated on the singular curves and the boundary over the parameter domain. In order to ensure the correctness of mesh intersection topology, we compute feature points and the pair-points of the curve sampling points, which are fixed strictly throughout our method. A triangle mesh is generated over the domain with the singular curves as constraint lines, which is the input of quadrilateral mesh generation. We compute a cross-field aligned with the feature lines and construct a coarse layout of quad patches relying on the cross-field. As T-junctions are allowed in the layout, we solve an Integer Linear Program (ILP) problem to determine the number of edges, thereby ensuring that the generated mesh is conforming. Feature lines are preserved perfectly throughout the pipeline by imposing constraints. In conclusion, our method guarantees the neighborhood coordination of mesh intersections, while still producing high-quality quadrilateral meshes.

Keywords: Self-intersecting surfaces, Quad partitioning, Quad meshing, Feature preservation

1 Introduction

In Computer Aided Design (CAD) and Computer Aided Engineering (CAE), it is often necessary to simulate various industrial products during their design process. The method of describing 3D models is crucial for accurate simulation and emulation. Parametric representation is a common way to represent surfaces, which is defined by a vector-valued function that maps a 2D parametric domain onto a 3D surface [1]. Mesh generation for surfaces is widely used in numerical analysis and serves as computational cells in the Finite Element Method (FEM) [2].

In practical applications of industrial product design, there may be a large number of complex surfaces, such as surfaces with self-intersecting curves. Although software such as MATLAB [3] can easily generate meshes for parametric surfaces given a parametric representation, they fail to guarantee the correct mesh topology in the vicinity of self-intersecting curves. Mesh generation for self-intersecting surfaces requires ensuring the consistency of the mesh along the self-intersecting curves of the surface. The parametric surface mesh generation methods that currently exist, such as advancing front method [4], bubble packing method [5] and particle-based method [6], are unable to correctly handle self-intersecting surfaces. A recent technique proposed by Zeng et al. [7] improves the particle-based mesh generation method and preserves self-intersecting curves correctly. However, the above methods can only generate triangle meshes. In the field of finite element analysis, quadrilateral meshes have greater advantages over triangle meshes: better alignment with model geometry, improved computational accuracy, improved computational efficiency and more [8, 9].

In addition, preserving sharp features is a challenging task in the field of quadrilateral mesh generation. It is often difficult to balance the goals of minimizing singularities, ensuring quadrilateral mesh regularity and preserving features. Existing methodologies in quadrilateral mesh generation typically prioritize enhancing mesh quality, often at the expense of accurately preserving model features [10]. Tarini et al. [11] preserve feature lines in the final stages of the process by vertex snapping techniques. Some methods introduce soft constraints to maintain vertices along feature lines [12, 13]. Jakob et al. [14] take a different approach by employing an extrinsic parametrization method. This approach naturally aligns sharp features in the resulting mesh, providing a form of sharp-feature alignment without the need for explicit snapping or soft force techniques. However, these methods may not consistently preserve all targeted features, and ensuring that they don't compromise other mesh requirements, such as fold-over prevention, can be challenging.

We propose a method for generating high-quality quadrilateral meshes for self-intersecting parametric surfaces that ensures the consistency of the mesh along the singular curves of the surface, including self-intersecting curves and cusp curves. We extract the singular curves of the self-intersecting parametric surface represented in algebraic form. The singular curves will be reproduced in the final surface quad mesh. To realize this, we identify the singular curves as feature lines and preserve feature lines throughout the mesh generation process. A triangle mesh is generated first over the parametric domain based on sampling points along the singular curves and boundaries in preparation for quadrilateral mesh generation. To derive the surface quadrilateral

mesh, we generate the quadrilateral mesh over the parametric domain from the triangle mesh and map the mesh onto the surface based on the parametric equations.

In order to ensure the correctness of mesh intersection topology, a key step involves generating sampling points on the pre-images of the singular curves and computing the pair-points for each curve sampling point. The pair-point is defined as the point that maps to the same surface point as the given curve sampling point. These points remain fixed throughout the mesh generation process. Additionally, we connect the sampling points as feature-edges and strictly preserve these feature-edges. This ensures that the intersection lines between the faces of the final generated surface mesh are mesh edges.

Our quadrilateral mesh generation method belongs to the cross-field based coarse layout method. We partition the model into quad patches and quadrangulate each patch individually in a globally consistent manner. The layout is constructed by tracing paths aligned with the computed cross-field along the mesh edges. Since T-junctions are allowed in the layout, we solve an Integer Linear Program problem to determine the number of edges required for quadrangulation, ensuring global consistency. To preserve the feature lines, we make all feature-lines paths and introduce constraints during the ILP and quadrangulation.

2 Related Work

2.1 Surface self-intersection computation

In order to generate a conforming quadrilateral mesh for a self-intersecting parametric surface, it is usually necessary to extract the singular curves, which will be kept as feature lines in the mesh generation process.

Volino et al. and Lin et al. [15–17] find self-intersecting curves by generating a triangle mesh and computing the self-intersection of the faces in the triangle mesh. Krishnan et al., Galligo et al. and Park et al. [18–20] partition the parameter domain into a large number of small regions and derive the self-intersecting curves of the surfaces by computing the intersections between the surface faces corresponding to these small regions. The errors of these methods are usually large, and the lack of accuracy can lead to incorrect self-intersecting curves. Achieving high accuracy requires the partitioning of sufficiently small faces, which can lead to a significant increase in computational efficiency, although Toth et al. [21] accelerate the computational efficiency by using Bounding Volume Hierarchy (BVH) trees. Jia et al. [22] propose a moving plane method that can directly compute the algebraic representations of self-intersecting curves and cusp curves through algebraic operations. This method derives curves with high accuracy and reduces the time complexity.

2.2 Quadrilateral Mesh Generation

Generating high-quality quadrilateral meshes has been the focus of academic research for decades. Catmull et al. [23] subdivide the triangles to obtain quadrilaterals. Owen et al. [24] combine multiple triangles into a quadrilateral by the advancing front method. Remacle et al. [25] use the perfect matching algorithm to find the globally

optimal triangle pairing scheme. Bommes et al. [26] derive the seamless global parameterization by solving a mixed integer problem. Then Bommes et al. [27] improve the method based on global parameterization by solving a convex Mixed-Integer Quadratic Programming (MIQP) to ensure bijection. Fang et al. [13] propose a local method based on global mapping and Morse theory to handle complex feature constraints. We refer the reader to [28] for more details on classical methods of quadrilateral mesh generation.

Our method is based on the coarse layout construction [29]. These approaches typically involve constructing a coarse layout of patches. There are a variety of approaches to constructing layouts. Dong et al. and Ling et al. [30, 31] generate a quad layout based on Morse–Smale theory. Gould et al. [32] are based on the medial axis. Usai et al. [33] follow the curve skeleton of a triangle mesh. Tarini et al. and Bommes et al. [12, 34] extract a quad layout based on the original quadrilateral mesh. Campen et al. [35] build a dual layout from curvature-guided crossing loops on the surface. Campen et al. and Razafindrazaka et al. [36, 37] trace streamlines based on cross-field. The layout can be conforming or non-conforming. It is easy to generate a quadrilateral mesh over a conforming layout, but it is difficult to construct the conforming layout [38–40]. So many methods construct a non-conforming layout, i.e. they allow T-junctions in the layout, which simplifies the layout construction [41–43].

Our method is most closely related to cross-field based streamline tracing methods. This class of methods first computes the smooth cross-field over the model and determines the location of the singular points, then traces the field-aligned streamlines and constructs a layout over the model. In these methods, the cross-field has a large influence on the construction of the layout, because the streamlines are aligned with the field. Typically we align the cross-field to the curvature or boundary to make the streamlines more consistent with the geometry of the model [44]. Ideally, streamlines can only be orthogonal to each other [45], and tangential crossings of streamlines can lead to limit cycles or extremely thin regions in the quad layout [12].

Some methods use singular points as start and end points and trace out field-aligned separatrices connecting singular points to construct a quad layout. Kowalski et al. [46] derive the cross-field by solving partial differential equations using Lagrange multipliers and applying the Runge-Kutta numerical scheme to extend streamlines starting from singular points. However, the method is not robust, which can lead to tangential crossings of streamlines. Fogg et al. [47] improve this method and proposes a smooth cross-field generation method for an arbitrary number of constraints, which uses a fast marching method to propagate the cross-field. Razafindrazaka et al. [48] analyze the singular structure of the cross-field, constructs a graph of possible matches based on the distance between singular points, and then solves the best matching problem to derive the final layout result. Pietroni et al. [49] propose field-coherent streamlines, constructs candidate streamlines connecting two singular points or connecting a singular point and a boundary point, and then solves the optimization problem to select the final streamline. Myles et al. and Lyon et al. [50, 51] propose a modification of the layout construction based on the motorcycle graph [52, 53]. Schertler et al. [54] improve the motorcycle graph and proposes a generalized motorcycle graph that is applicable in the presence of imperfect or non-pure quad meshes

and can construct a quad layout more robustly. The above methods all generate a quad layout, but Pietroni et al. [55] propose a method to generate a non-pure quad layout with T-junctions and quadrangulate each patch with the method proposed by Takayama et al. [56], which can quadrangulate a patch with 2-6 sides by inputting the number of side subdivisions at its boundary.

3 Overview

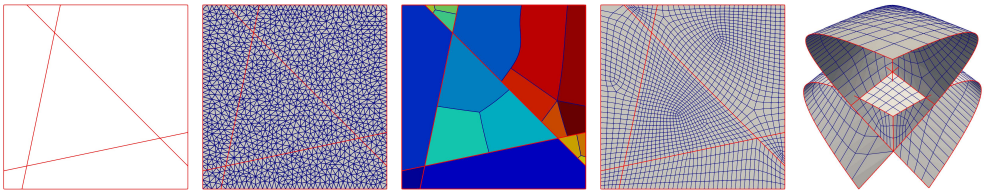

Fig. 1 The pipeline of our method. We first compute the singular curves of the surface based on the parametric equation (a). A triangle mesh is generated over the domain constrained by the singular curves (b). Then we construct a quad layout over the triangle mesh (c) and quadrangulate each patch to derive the quadrilateral mesh (d). Finally, the output quadrilateral surface mesh is obtained by mapping the mesh over the domain to the surface (e).

Singularity computation for surfaces. First, we compute a series of linearly independent moving planes that follow the given surface parametrization. This set of planes serves as a representation for rational parametric surfaces, from which we derive a matrix representation of the surface. Subsequently, we utilize this matrix representation to compute the singular factors on the given surface. Each singular factor corresponds to either a singular curve or an isolated singular point.

Triangle mesh generation over the parameter domain. Next, we project the singular curves into the parameter domain to obtain their pre-images. We decompose the singular factor into several functions, corresponding to different curve branches. We sample points on these curve branches respectively. Moreover, the pair-points, feature points and boundary points are computed. These sampling points are then interconnected to form edges, serving as constraint edges for the triangle mesh generation. Finally, we conduct constrained Delaunay triangulation utilizing these constraint edges to ensure the preservation of singular curves throughout the mesh generation.

Quadrilateral mesh generation with constraints. We then generate a quadrilateral mesh from the triangle mesh and map the quadrilateral mesh on the parameter domain to the parametric surface. This approach yields both a quadrilateral mesh on the parameter domain and another on the parametric surface. We first compute the cross-field on the triangle mesh and utilize path-tracing based on this cross-field to partition the mesh into patches. Notably, this partitioning process permits the existence of T-junctions within the layout. To ensure the generation of a conforming mesh in the presence of these T-junctions, we solve an Integer Linear Program to determine the

number of subdivisions for each patch side. Then we quadrangulate each patch individually and merge them to obtain the final quadrilateral mesh model. The constraint edges obtained previously as well as points on the constraint edges are still preserved throughout this process.

4 SINGULARITY COMPUTATION FOR SURFACES

We briefly describe how to determine singular curves and isolated singularities on parametric surfaces by the moving plane method [22].

4.1 Moving planes

A rational parameter surface in $\mathbb{R}^3$ can be represented by a map

$$\begin{aligned} &\varphi : \mathbb{R}^2 \to \mathbb{R}^3 \\ &(s,t) \mapsto \left(\frac{f_0(s,t)}{f_3(s,t)}, \frac{f_1(s,t)}{f_3(s,t)}, \frac{f_2(s,t)}{f_3(s,t)} \right) \end{aligned}$$

where $f_i(s,t), i = 0,1,2,3$ are polynomials with respect to s and t which are linearly independent of each other. To represent points at infinity, we adopt the homogeneous parametrization and this map becomes:

$$\begin{aligned} &\phi : \mathbb{R}^2 \to \mathbb{PR}^3 \\ &(s,t) \mapsto (f_0(s,t), f_1(s,t), f_2(s,t), f_3(s,t)) \end{aligned}$$

A *moving plane* is defined as a family of planes determined by the parameters s, t:

$$L(s,t;X) := l_0(s,t)\,x + l_1(s,t)\,y + l_2(s,t)\,z + l_3(s,t)\,w = 0$$

where $l_i(s,t), i = 0,1,2,3$ are polynomials with respect to s, t.

We can also represent the moving plane in vector form

$$\mathbf{L}(s,t) = (l_0(s,t), l_1(s,t), l_2(s,t), l_3(s,t))$$

We call the moving plane $\mathbf{L}(s,t)$ following the rational parametrization of the surface $\phi(s,t)$ when it satisfies the following equations, i.e., the moving plane $\mathbf{L}(s,t)$ always passes through the point $\phi(s,t)$ on the surface for each parameter pair (s,t).

$$\mathbf{L}(s,t) \cdot \phi(s,t) = \sum_{i=0}^{3} l_i(s,t)\, f_i(s,t) \equiv 0 \tag{1}$$

For a general rational surface $\phi(s,t)$, it is always possible to find a set of moving planes $[L_1, L_2, \ldots, L_k]$ which are linearly independent of each other such that each moving plane L_i satisfy Equation (1). For any point $P = \phi(s_0, t_0)$ on the surface, the point must be located on a moving plane $L_i(s_0, t_0)$.

4.2 Matrix representation of surfaces

Let d_1 be the order of the highest terms with respect to s in the rational parameter surface $\phi(s,t)$, and d_2 be the order of the highest terms with respect to t in the rational parameter surface $\phi(s,t)$.

We define a monomial basis vector $\mathbf{b} = [1, s, \ldots, s^{v_1}, t, st, \ldots, s^{v_1}t^{v_2}]$ with respect to s,t. The moving plane $[L_1, L_2, \ldots, L_k]$ can be written as the product of a monomial basis vector $\mathbf{b}$ and a matrix $M(X)$ of size $q \times k$:

$$[L_1, L_2, \ldots, L_k] = \mathbf{b} \cdot M(X) \tag{2}$$

The matrix $M(X)$ is called the *matrix representation* of the rational parameter surface $\phi(s,t)$, where $v_1 = 2d_1 - 1, v_2 = d_2 - 1, q = (v_1+1)(v_2+1)$.

Given a parametric surface $\phi(s,t)$, we construct the moving plane equations for the monomial basis vectors and the moving planes containing the unknown coefficients. The moving planes following the rational parametrization of the surface can be derived by solving Equation (1). Then we can derive the matrix representation of the surface $M(X)$ from Equation (2).

4.3 Computing singular factors

A *singular point* on a rational parameter surface is a point with more than one corresponding parameter pair. For a point $P = (x_0, y_0, z_0, w_0)$ on a parameter surface, the point P is said to be a *simple point* if there is only one parameter pair (s,t) such that $\phi(s,t) = P$. A point P is said to be a singular point if there are $N\,(N > 1)$ parameter pairs $(s_i, t_i), i = 1, 2, \ldots, N$ such that $\phi(s_i, t_i) = P$. If N is finite, the order of the point P is defined as N. Otherwise, there are infinitely many parameter pairs such that $\phi(s_i, t_i) = P$, in which case the point P is also called an isolated singular point. Each point on a singular curve of a surface is a singular point with finite corresponding parameter pairs.

We substitute the parametric surface $\phi(s,t)$ into the matrix representation of the surface $M(X)$ to derive the matrix with respect to s,t

$$N(s,t) = M(f_0(s,t), f_1(s,t), f_2(s,t), f_3(s,t))$$

We compute the minors $D_i(s,t)$ of order $q-1$ for the matrix $N(s,t)$ and compute the greatest common divisor of $D_i(s,t)$, denoted $H(s,t)$. We then factorize $H(s,t)$ to derive a number of factors $h_i(s,t)$, which we call *singular factors*.

The order of a singularity factor $h(s,t)$ is defined as the order of the corresponding singular curve. $h(s,t) = 0$ corresponds to a singular curve on the parameter surface if the order is finite. $h(s,t) = 0$ corresponds to an isolated singular point if the order is infinite.

5 TRIANGLE MESH GENERATION

After the computation of singular factors, we sample points along pre-images of the singular curves corresponding to these singular factors and generate a triangle mesh

from the sampling points for the subsequent quadrilateral mesh generation. Except for curve sampling points, we compute pair-points, feature points and boundary sampling points.

5.1 Sampling points generation

5.1.1 Computing curve sampling points

There may be several branches in the parameter domain after projecting the singular factor $h(s,t)$ into the parameter domain. In order to correctly connect the sampling points to get the constraint edges in the subsequent process, we need to know which branch each sampling point is located on and connect the sampling points located on each branch respectively. Therefore, we compute the function with its domain corresponding to each branch curve.

We define $D = I \times J \subset \mathbb{R}^2$ as the parameter domain of the parametric surface $\phi(s,t)$, where $s \in I \subset \mathbb{R}$, $t \in J \subset \mathbb{R}$. Given a singular factor $h(s,t)$, we can determine the function $t = g_i(s)$ with its domain such that $h(s, g_i(s)) \equiv 0$, where $s \in I_i \subseteq I$, $t \in J_i \subseteq J$.

The function $t = g_i(s)$ determined in this way may have multiple intervals within its domain. We decompose the functions so that each decomposed function works within a single interval. For a function $t = g_i(s)$ with domain I_i, we decompose it into $t = g_{ij}(s)$, where $s \in I_{ij}$, $I_i = \bigcup_{j=1}^{m} I_{ij}$ and $I_{ij-1} \cap I_{ij} = \emptyset$. In this case, each function $t = g_{ij}(s)$ corresponds to a distinct branch curve within the defined interval I_{ij}.

For example, for the equation $h(s,t) = s^2 - 3t^2 - 3 = 0, s \in [-2,2], t \in [-2,2]$, two functions can be determined (see Fig. 2):

$$t = g_1(s) = -\frac{\sqrt{3s^2-9}}{3}, s \in \left[-2,-\sqrt{3}\right] \cup \left[\sqrt{3},2\right]$$
$$t = g_2(s) = \frac{\sqrt{3s^2-9}}{3}, s \in \left[-2,-\sqrt{3}\right] \cup \left[\sqrt{3},2\right]$$

The domain of these two functions can be composed of two intervals respectively:

$$t = g_{11}(s) = -\frac{\sqrt{3s^2-9}}{3}, s \in \left[-2,-\sqrt{3}\right]$$
$$t = g_{12}(s) = -\frac{\sqrt{3s^2-9}}{3}, s \in \left[\sqrt{3},2\right]$$
$$t = g_{21}(s) = \frac{\sqrt{3s^2-9}}{3}, s \in \left[-2,-\sqrt{3}\right]$$
$$t = g_{22}(s) = \frac{\sqrt{3s^2-9}}{3}, s \in \left[\sqrt{3},2\right]$$

Next, we uniformly generate curve sampling points on the branch curve S_{ij} corresponding to $t - g_{ij}(s) = 0$ and add them to the sampling point set V_{ij}, where $V_{ij} = \left\{(s,t) \middle| t - g_{ij}(s) = 0\right\}$. Let $V = \bigcup_{i=0}^{n} \bigcup_{j=0}^{m} V_{ij}$ be the total sampling point set.

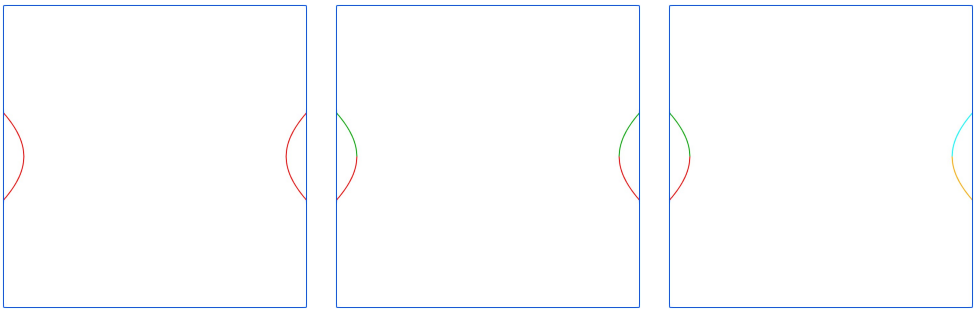

Fig. 2 An example of decomposing the singular curve $h(s,t) = 0$ (a). $h(s,t) = 0$ is decomposed into two functions $t = g_1(s)$ (red curves) and $t = g_2(s)$ (green curves) (b). Then each function with multiple intervals is decomposed into different branches (c).

5.1.2 Computing pair-points

From Section 4.3, it is evident that each point along the singular curve corresponds to multiple pre-image points. If there is a point $Q_1 = (s_1, t_1)$ and a point $Q_2 = (s_2, t_2)$ in parameter domain D, such that $\phi(s_1, t_2) = \phi(s_2, t_2)$, we call the point Q_1 is the pair-point of point Q_2 , or the point Q_2 is the pair-point of point Q_1.

The above sampling process may cause the loss of pair-points of the curve sampling points, that is, for a sampling point $Q_1 = (s_1, t_1) \in V$, there is a point $Q_2 = (s_2, t_2) \notin V$, such that $\phi(s_1, t_2) = \phi(s_2, t_2)$. This will lead to the mesh generated later being non-conforming (see Fig. 3). Consequently, for each sampling point Q in the sampling point set V, we compute its pair-points. Subsequently, we identify the branch S_{ij} where the pair-point is located and add pair-points to the corresponding sampling point set V_{ij}.

Given a point $Q_1 = (s_1, t_1)$ in the parameter domain D, we compute its corresponding surface point $P = \phi(s_1, t_1) = (x_1, y_1, z_1, w_1)$. Solving the system of Equations (3) results in the solution $Q_i = (s_i, t_i)$, which is the pair-point of point Q_1, where $Q_i \neq Q_1$.

$$\begin{cases} x_1 &= f_0(s,t) \\ y_1 &= f_1(s,t) \\ z_1 &= f_2(s,t) \\ w_1 &= f_3(s,t) \end{cases} \tag{3}$$

5.1.3 Computing feature points

Next, we locate feature points on the parameter domain, which include the intersection points of singular curves and the boundary of the parameter domain, the self-intersecting points of singular curves, the intersection points of different singular curves, and miter points [20]. The loss of feature points could also lead to a non-conforming mesh.

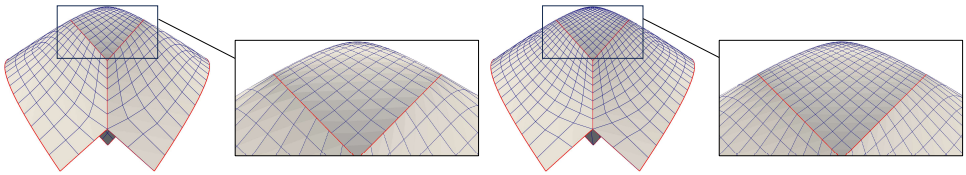

Fig. 3 Left is a non-conforming mesh due to the loss of the pair-points, and right is a conforming mesh.

The intersection points of the singular curve and the parameter domain are derived by solving equations $h(a,t)=0, h(b,t)=0, h(s,c)=0$ and $h(s,d)=0$ respectively.

The self-intersecting points of the singular curve are derived by solving Equation (4).

$$\begin{cases} h(s,t)=0 \\ \frac{\partial h(s,t)}{\partial s}=0 \\ \frac{\partial h(s,t)}{\partial t}=0 \end{cases} \tag{4}$$

The intersection points of different singular curves $h_1(s,t)$ and $h_2(s,t)$ are derived by solving Equation (5).

$$\begin{cases} h_1(s,t)=0 \\ h_2(s,t)=0 \end{cases} \tag{5}$$

The miter points are derived by solving Equation (6).

$$\begin{cases} h(s,t)=0 \\ \det\left(J^T J\right)=0 \end{cases} \tag{6}$$

where

$$J=\begin{pmatrix} \frac{\partial \frac{f_0(s,t)}{f_3(s,t)}}{\partial s} & \frac{\partial \frac{f_0(s,t)}{f_3(s,t)}}{\partial t} \\ \frac{\partial \frac{f_1(s,t)}{f_3(s,t)}}{\partial s} & \frac{\partial \frac{f_1(s,t)}{f_3(s,t)}}{\partial t} \\ \frac{\partial \frac{f_2(s,t)}{f_3(s,t)}}{\partial s} & \frac{\partial \frac{f_2(s,t)}{f_3(s,t)}}{\partial t} \end{pmatrix}$$

5.2 Triangulation with constraints

For each sampling point set V_{ij}, we execute the minimum spanning tree algorithm, which connects the points in the point set and produces the corresponding edge set E_{ij}. Let $E=\bigcup_{i=0}^{n}\bigcup_{j=0}^{m}E_{ij}$ be the set of constrained edges. We execute constrained Delaunay triangulation [57] to generate a triangle mesh with V as the input set of points and E as the constrained edges. During the triangle mesh generation process, it is forbidden to insert new vertices on constrained edges, but it is allowed to insert new vertices elsewhere to optimize mesh quality.

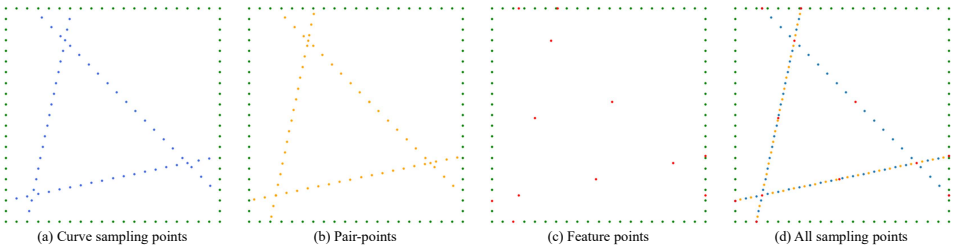

Fig. 4 An example of generating sampling points, including boundary sampling points (green points). We remove duplicate points and keep only one.

6 QUADRILATERAL MESH GENERATION

In this step of the pipeline, we generate the quadrilateral mesh based on the triangle mesh. The constraint-edges input during the generation of the triangle mesh also serves as the feature-edges in this step. These feature edges will be preserved throughout the process of quadrilateral mesh generation. We classify feature-edges into two types: *singular-edges* that correspond to the pre-images of singular curves, and boundary-edges that belong to the parameter domain boundary. Different maintenance strategies are applied to these two types of feature-edges. For the singular-edges, we need to ensure that the vertices of the feature-edges are not offset in any way and the insertion of new vertices into these feature-edges is prohibited. Conversely, for the boundary-edges, we can relax the restriction by allowing the boundary vertices to slide on the boundary and allowing the insertion or deletion of vertices on the boundary.

6.1 Computing cross-field

To compute a cross-field aligned with the feature-edges over the input triangle mesh, we initiate the process by establishing initial constraints for the cross-field. For each face adjacent to a feature-edge, we assign a tangent direction on the face aligned with the corresponding mesh edge. We then adopted the cross-field propagation method described in [40] to diffuse the cross-field inward, resulting in the generation of a smooth transition cross-field within the model. This diffusion process ensures that each face is assigned a cross-field value.

Finally, based on the cross-field value of each face, we determine the singular points defined on the mesh vertices. For non-singular vertices, a cross-field value is assigned, computed as the average of the cross-field values of the adjacent faces of the vertex. This comprehensive approach ensures the alignment of the cross-field with the feature-edges and facilitates the determination of singular points and assignment of cross-field values to the mesh vertices.

6.2 Layout Construction

We trace paths along the edges of the mesh to get a set of patch sides aligned with the cross-field. These patch sides divide the mesh into patches, resulting in a layout of quadrilateral patches. There are two processes for path tracing. The first process,

in which paths will avoid passing through singularities, divides the model into non-quadrilateral patches and makes it possible to have at most one singularity in each patch. The second process targets non-quadrilateral patches for further partitioning. Paths are traced from singularities inside the patch, and these paths divide the non-quadrilateral patch into a number of quadrilateral sub-patches.

6.2.1 Preprocessing

We first preprocess the mesh by cutting all internal feature-edges into two directed boundary-edges. The vertices on the feature-edges are then duplicated into two copies, corresponding to the new edges on each side of the original feature-edges.

We classify vertices on feature-edges into narrow, concave, flat and convex vertices based on the angle θ of the sector formed by the triangle faces surrounding each vertex. It's important to note that since the vertices on the original feature-edges are duplicated into two copies, the new vertices on both sides of the feature-edges are classified separately. When $315^\circ \leq \theta < 360^\circ$, the vertices are classified as narrow vertices, when $225^\circ \leq \theta < 315^\circ$, the vertices are classified as concave vertices, when $135^\circ \leq \theta < 225^\circ$, the vertices are classified as flat vertices, and when $\theta < 135^\circ$, the vertices are classified as convex vertices (see Fig. 5).

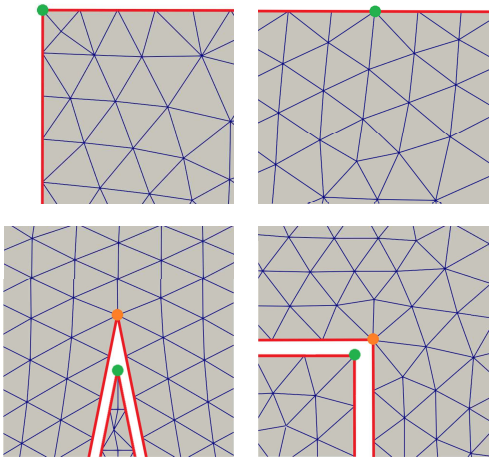

Fig. 5 Examples of different classes of boundary-vertices. The mesh is cut along the feature-edges, resulting in the vertices being copied in two copies of different classes (bottom).

We construct a directed weighted graph G on the mesh, where each vertex in the mesh will provide four nodes in the graph G. Each node represents a mesh vertex and a tangent direction of the field at that vertex. Since there is no corresponding cross-field value at the singularities, the singularities will be ignored in the construction of the graph G.

We define the weight between two nodes as

$$w_{ij} = (v_j - v_i) \cdot f^{\perp} \tag{7}$$

where f represents the average field direction between nodes v_i and v_j, and $f^{\perp}$ is the direction orthogonal to f.

6.2.2 Tracing paths for a non-quad layout

To ensure the quality of the constructed patches, the patches constructed in this step must adhere to the following three conditions, known as patch validity conditions:

1. Topological constraint: the genus of the patch is zero.
2. Convex constraint: only flat and convex vertices are allowed on the boundary of the patch.
3. Side constraint: the number of sides of the patch is between 3 and 6.

We first designate the feature-edges as patch boundaries and then proceed to trace paths to generate additional patch sides, facilitating the region decomposition process. The approach involves selecting nodes within the graph G as both starting nodes S and ending nodes E of the patch sides. Then we execute the Dijkstra algorithm to search for the shortest path between the starting node in S and the ending node in E, effectively determining the patch side.

The starting nodes S and the ending nodes E are chosen in two passes. During the first pass, nodes corresponding to two directions outward from each concave vertex and nodes corresponding to three directions outward from each narrow vertex are added to the starting nodes S. All remaining nodes corresponding to concave and narrow vertices are then added to the ending nodes E. This initial path-tracing process effectively eliminates the majority of concave and narrow vertices, thereby generating patches that conform to the convex constraints. However, there may still be patches that fail to meet the convex constraints, implying that the starting nodes S are non-empty. To address this, we include the nodes corresponding to the flat vertices in the ending nodes E and repeat the path-tracing process. This additional step ensures that all patches ultimately adhere to the convex constraints. During the second pass, nodes corresponding to flat vertices are randomly sampled. A portion of these sampled nodes is designated as the starting nodes S, while the remaining portion is assigned as the ending nodes E. The patch sides generated in this step assist in dealing with the patches that do not satisfy the topological constraints or side constraints.

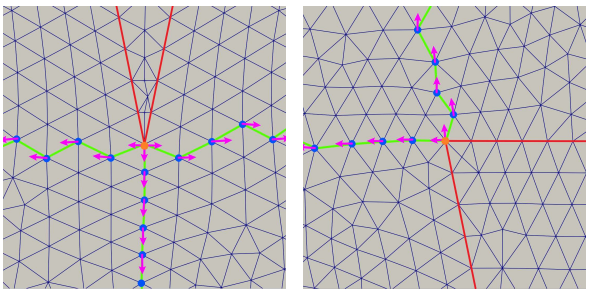

Fig. 6 Three paths are traced from a narrow vertex (left), and two paths are traced from a concave vertex (right).

After tracing paths in these two passes, initial patches are constructed. We have to check whether these patches satisfy the validity condition. For the patches that do not satisfy the validity conditions, we recursively repeat the above steps, partitioning the patch into smaller patches until all patches satisfy the validity condition. A difference from the above steps is that when tracing paths inside a patch, the new path terminates when it reaches to the boundary of the patch.

Following the construction process, there may be redundant patches. To optimize the final quadrangulation while ensuring validity conditions are met, we aim to merge certain patches with others, thereby reducing the overall number of patches. Fewer patches will help to improve the quality and efficiency of the final quadrangulation.

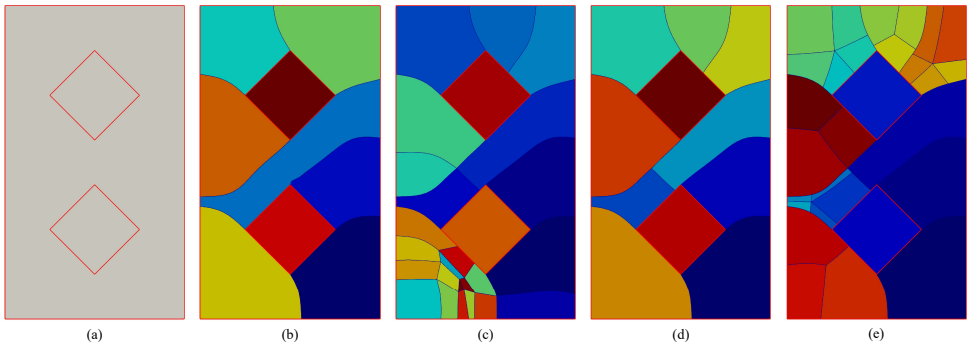

Fig. 7 Different processes of the construction of layout. Given an input triangle mesh with feature lines (a), an initial layout of patches that may not satisfy validity conditions is constructed (b). We partition the unsolved patches recursively (c), and merge the patches without violating validity conditions (d). Finally, a quad layout is constructed by tracing paths from the singularity inside the patch (e).

6.2.3 Partitioning for non-quad patches

Following the construction process described above, we proceed to independently partition each non-quadrilateral patch to construct a pure-quad layout. Due to the validity conditions, polygonal patches with relatively regular geometric boundaries are derived. These patches typically exhibit clear polygon-like boundaries with a small number of sides, typically not exceeding 6. Starting from a point inside the polygon and drawing rays to each boundary respectively, the polygon can be divided into several rectangles, with the number of rectangles equal to the number of patch sides. We will partition the non-quad patches with a similar spirit.

To address potential path-tracing failures due to insufficient mesh edges within the patch, we employ a re-meshing technique [58]. This involves implementing local operations such as edge-flip, edge-fold, and edge-split to refine the mesh and optimize its quality. It's crucial to ensure that feature-edges remain fixed throughout this process, and operations that may impact feature-edges are prohibited. Re-meshing the patch may introduce T-junctions along the patch sides. To ensure that the mesh is

conforming, we repair the mesh near these T-junctions. Subsequently, we compute the cross-field within the patch and identify field-singularities. Path tracing is initiated from each singularity, extending to each original patch side and thereby partitioning the patch into several quadrilateral sub-patches.

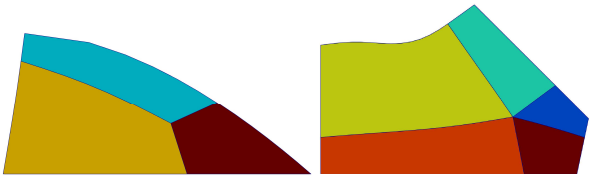

Fig. 8 Examples of partitioning for the triangle patch and the pentagonal patch.

This path-tracing procedure also relies on a directed weighted graph G. To accommodate path-tracing from the singularity, we construct a new graph on the patch. In this graph, we include the singularity as a node in G with a corresponding field value of zero. Additionally, we ensure that the weights between the singularity and all neighboring nodes are set to zero.

For each original patch side, we designate the node corresponding to the singularity as the starting node S, and we include the nodes corresponding to the vertices along the side (excluding the corners) as the ending nodes E. This configuration results in multiple paths originating from the singularity and terminating at the original patch side. From these paths, we select one as the final new patch side. For an N-sided patch, we repeat this process to determine N final new patch sides. These new patch sides effectively partition the patch into N quadrilateral sub-patches.

We first execute the Dijkstra algorithm to search for the shortest path from the singularity to the original patch side as candidate paths for the final new patch sides. The weight between two nodes in the path is computed by Equation (7), with the exception that the weight between the singularity and its neighbouring nodes is zero. In selecting the final patch sides, it's crucial that the chosen paths do not intersect or tangent any of the identified paths. Moreover, we aim to satisfy two quality conditions: (1) the angle with the selected path is close to 90° and (2) the total weight of the path is minimized. Meeting the first condition ensures that the constructed patch closely resembles a rectangle, while satisfying the second condition ensures alignment with the cross-field, thus enhancing the quality of the final generated quadrilateral mesh.

In cases where the patch is a triangle and contains a corner with an angle less than 45°, the singularity in the vicinity of the corner will degenerate to that corner. This can lead to an inability to accurately compute the singularity within the patch [39]. Therefore, when computing the cross-field for regions containing small corners, it's essential to optimize the boundary constraints of the cross-field. This optimization ensures that a three-valence singularity is introduced inside the patch.

For a given triangle ABC, where A is a small corner, we define the representation vector at A as the opposite vector of the average of the representation vectors of the two geometrical edges connected to A. We then linearly interpolate the representation

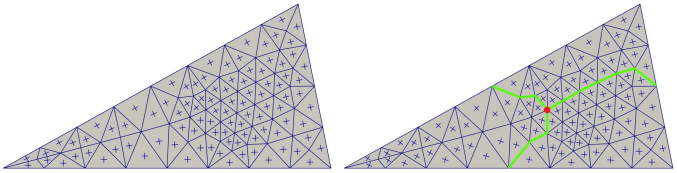

Fig. 9 The original cross-field is aligned with the boundary (left). We optimize the cross-field to introduce a new singular point (red point) inside (right). Paths (green lines) are traced from the singular point.

vectors along AB and AC to update the cross-field values along AB and AC. These updated values serve as boundary constraints for the new cross-field computation.

In order to eliminate ambiguity when interpolating the vectors, the representation vectors on AB are interpolated in the counterclockwise direction and the representation vectors on AC are interpolated in the clockwise direction. This ensures that the cross-field on the boundary rotates along the counterclockwise direction, introducing a three-valence singularity in the interior [59].

In some cases, there may still be non-quadrilateral sub-patches after the initial partitioning of the patches. The described operation is recursively performed on the non-quadrilateral patches until there are no longer any non-quadrilateral patches remaining.

6.3 Quadrangulation

After the construction of the layout, we generate quadrilateral mesh for each patch independently and then merge all sub-meshes into a complete mesh. In order to ensure that the merged mesh is conforming, we need to quantize each patch side first, that is, to determine the number of quantized edges along each patch side.

6.3.1 Quantization for patch sides

We split the patch sides of the layout at every T-junction, deriving a set of *arches* [55]. To determine the number of quantized edges for each arch, we solve an Integer Linear Program. We represent the number of quantized edges for each arch with an integer variable $x_i \geq 1$ and minimize the objective function over x_i. The objective function consists of the following constraints and objective terms.

1. Parity constraints

   A patch can be quadrangulated only if the sum of all quantized edges on the sides of the patch is even [56]. Thus for each patch, we impose the constraint that the sum of the number of quantized edges is even, by adding an auxiliary integer variable m:

   $$\sum x_i = 2m, m \in \mathbb{N}, \forall x \in K$$

   where K is the set of arches surrounding a patch.
2. Feature constraints

   We enforce a constraint that prohibits the insertion of new vertices on singular-edges. As detailed in 5.1.2, it is essential to compute and include pair-points for

each point on a singular curve to maintain mesh conformity. Therefore, after we locate all the curve sampling points, including pair-points and feature points, we must ensure their positions remain fixed, prohibiting any positional offsets. Additionally, it is strictly forbidden to insert new vertices on the singular-edges during quadrangulation. This prohibition simplifies the process by eliminating the need to find pair-points for any newly inserted vertices, which would otherwise complicate the process. However, splitting or merging feature-edges that are not singular-edges is permitted.

F is defined as the set of arches belonging to singular-edges. For each arch of F, we add a constraint so that the number of quantized edges along the arch is equal to the number of mesh edges e_i of the arch.

$$x_i = e_i, x_i \in F$$

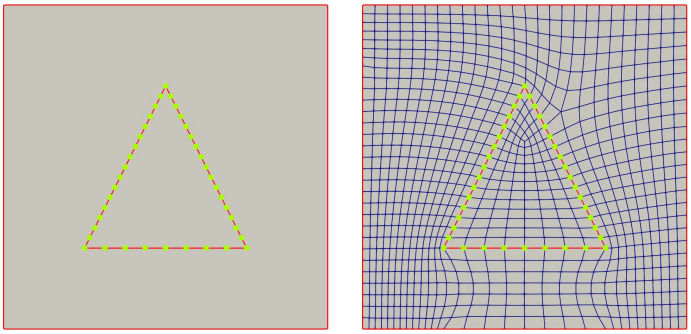

Fig. 10 The points of the singular-edges are preserved strictly during the mesh generation.

3. Regularity objective

It is optimal for a mesh to contain as few singularities as possible. A quadrilateral patch can be quadrangulated without inserting singularities when the number of quantized edges of two opposite sides of a patch is equal [60], i.e., the following conditions are satisfied:

$$\begin{cases} s_0 = s_1 \\ s_2 = s_3 \end{cases} \tag{8}$$

where s_j $(0 \leq j \leq 3)$ is the sum of a set of x_i along the side j of a patch.

Setting these conditions as hard constraints may lead to an unfeasible linear integer programming model. Therefore we add an auxiliary variable $w \geq 0$ for each patch and add w to the objective function. Then we impose the hard linear constraints:

$$|s_j - s_{j+2}| \leq w$$

Conditions (8) are satisfied as much as possible by minimizing w.

4. Isometry objective

   Given that we prohibit the insertion of new vertices on singular-edges and maintain fixed positions of mesh vertices along these edges, the lengths of the singular-edges are predetermined. In order to make the size of the mesh uniform, we make the length of the edges of the final mesh as close as possible to the average of the lengths of the singular edges l_{avg}. We compute the length L_i of each arch i and include an auxiliary variable $c \geq 0$ in the objective function for each arch. We impose constraints:

$$|\frac{L_i}{l_{avg}} - x_i| \leq c_i$$

Finally, we weight and normalize the regularity and isometry objectives separately to formulate the final objective function.

6.3.2 Quadrangulation for patches

After determining the number of quantized edges, we quadrangulate each patch by the mesh patterns provided by [56]. In this process, we constrain the vertices on the singular edges to be fixed and the vertices on the boundary-edges to be allowed to slide on the boundary.

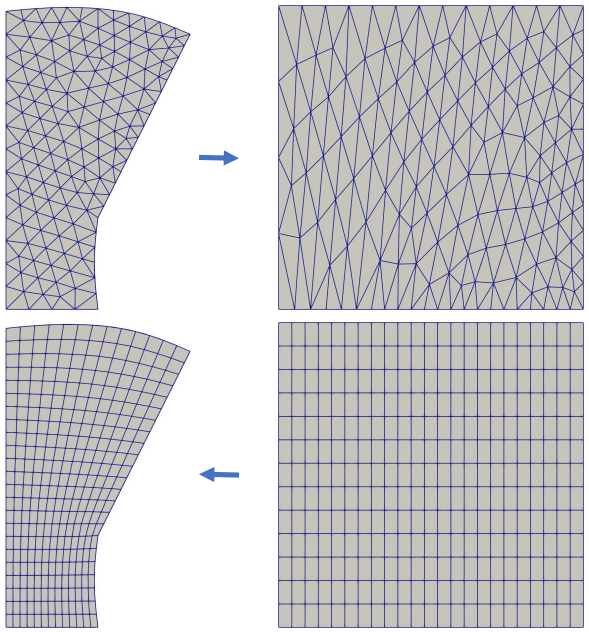

Fig. 11 The triangle mesh is parameterized into the parameter domain. Then the mesh pattern is mapped to the coordinate system where the triangle mesh is located through barycentric interpolation.

For the patch sides associated with singular-edges and those associated with model boundary-edges, we compute in advance the positions of the new vertices that will

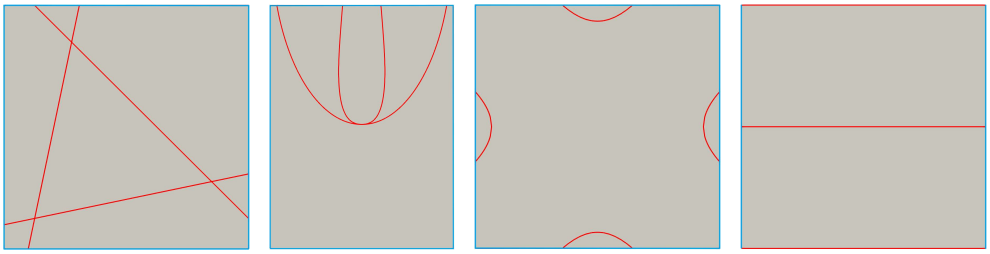

Fig. 12 Examples of pre-images of singular curves (red curves) for self-intersecting parametric surfaces.

be generated in the quadrilateral mesh based on the computed numbers of quantized edges. For the former, where the number of quantized edges is constrained to match the number of mesh edges, we establish a one-to-one correspondence between the resulting new vertices and the original vertices. Consequently, the position of the new vertex directly corresponds to the position of the corresponding old vertex. For the latter, where the patch sides are associated with model boundary-edges, we linearly interpolate to compute the positions of the new vertices.

Then we map the vertices on the patch sides to the parameter domain boundary of the mesh pattern. For the patch sides associated with feature-edges, we map them based on the positions of the pre-computed new vertices. Conversely, for the remaining patch sides, we uniformly map them based on the number of quantized edges. Subsequently, the mesh corresponding to the patch is parameterized into the parameter domain of the mesh pattern. We interpolate the positions of the vertices in the mesh pattern within the coordinate system of the original mesh. Then, we project the mesh pattern back onto the original mesh to determine the vertices in the quadrilateral mesh corresponding to the patch, excluding the fixed vertices.

Finally, we smooth the quadrilateral mesh to improve the quality, while also ensuring that vertices on the feature-edges remain fixed during the smoothing process.

7 RESULTS

We conduct our experiments on a Windows-based personal laptop equipped with an Intel Core i5-12500H processor, 16 GB of RAM, and operating at 2.8 GHz. We use SymPy [61] to compute singularity for surfaces, VcgLib [62] for mesh processing, and Gurobi [63] to solve the ILP problem. The generated quadrilateral meshes are evaluated in terms of mesh quality and feature preservation.

The parametric equations and their corresponding singular factors corresponding to the self-intersecting curves and/or cusp curves shown in Fig. 12 from left to right are as follows:

- Bi-quadric2. The parametric equation is:

$$\phi(s,t) = \left(s^2 + \frac{1}{2}t, t^2 + \frac{1}{2}s, st + s + t, 1\right)$$

The singular factors are: $h_1 = 2s+2t-1$ and $h_2 = 4t^2-20st-18t+4s^2-18s-27$.

- Rational. The parametric equation is:

$$\phi(s,t) = \left(st^2+t^2-12st+4s^3+4s^2+s+1,\right.$$
$$\left. s^2t^2+6t+4s^4+s^2, 6t^2, t^2+4s^2+1\right)$$

The singular factors are: $h_1 = 4s^2t^2+t^2-12t+16s^4+8s^2+1$ and $h_2 = t^2-12t+4s^2+1$.

- Enneper. The parametric equation is:

$$\phi(s,t) = \left(s-\frac{s^3}{3}+st^2, t-\frac{t^3}{3}+s^2t, s^2-t^2, 1\right)$$

The singular factors are: $h_1 = 3s^2-t^2+3$ and $h_2 = -3t^2+s^2+3$.

- Cusp. The parametric equation is:

$$\phi(s,t) = \left(\left(2\left(1-s^2\right)\right)\left(1-t^2\right)s, 2\left(1-s^2\right)^2 t, 8s^2t, 1\right)$$

The singular factors are: $h_1 = t$, $h_2 = t+1$ and $h_3 = t-1$.

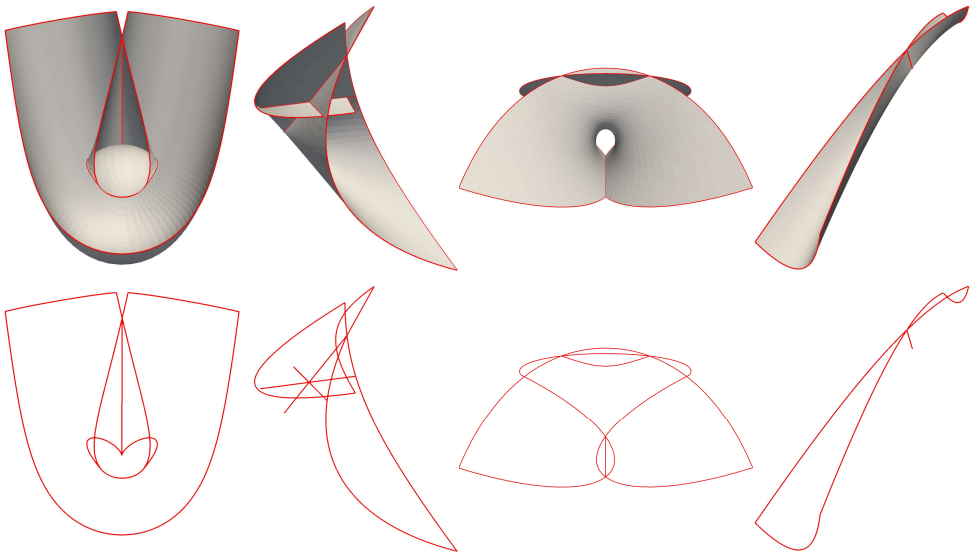

Fig. 13 Examples of boundaries and self-intersecting lines extracted from our generated mesh.

The quality of the mesh is evaluated by the number of singularities (#S), the average angle deviation from 90° (AD), the average Scaled Jacobian (ASJ) and the minimum Scaled Jacobian (MSJ) [65]. The precision of the generated mesh on feature-edges is evaluated by *one-sided Hausdorff distance* d_H and the *average distance* d_A.

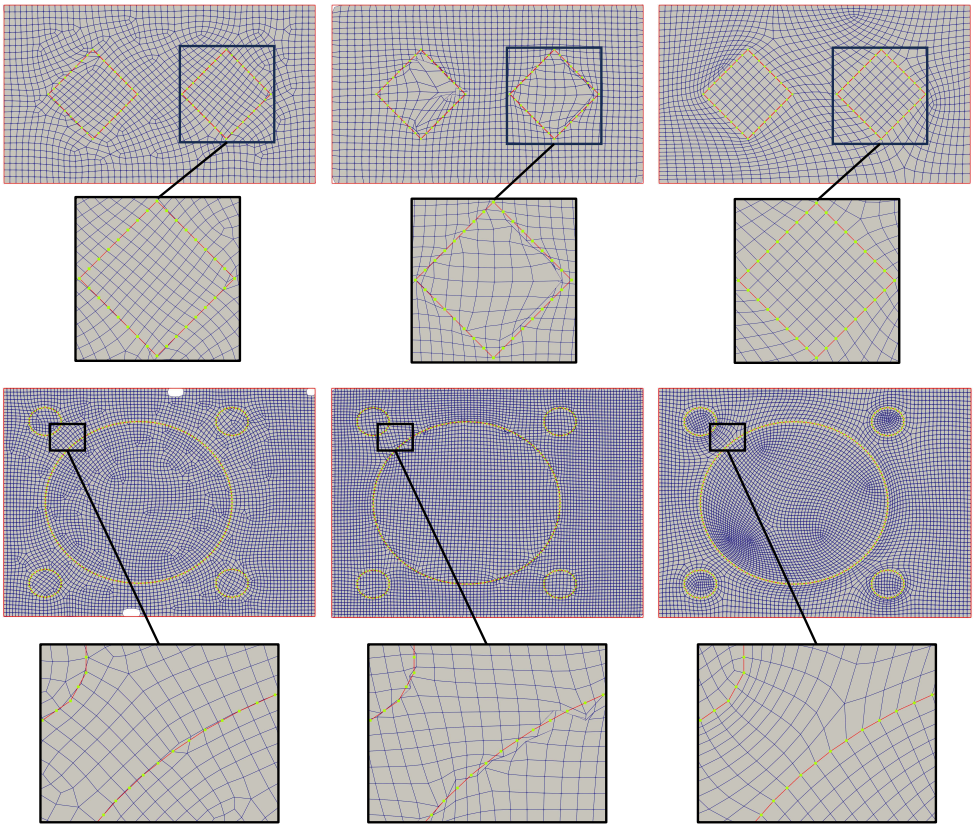

Fig. 14 A comparison of meshes over domain between Instant-Mesh [14] (left), Quadriflow [64] (middle) and the proposed method (right). Our method correctly preserves the feature lines (red lines) and fixes the feature vertices (green vertices).

The set of sampling points P is generated uniformly with a distance of 10^{-2} on the singular curves and boundary of the parametric surface. We compute its Hausdorff distance d_H from the set of feature points C as well as the average distance d_A.

$$d_H(P, C) = max_{\mathbf{p}\in P} min_{\mathbf{c}\in C} \|\mathbf{p} - \mathbf{c}\|_2$$
$$d_A(P, C) = \frac{1}{|P|} \sum_{\mathbf{p}\in P} min_{\mathbf{c}\in C} \|\mathbf{p} - \mathbf{c}\|_2$$

Table 1 reports the results and metrics of our generated meshes compared to those generated by Instant-Mesh [14] and Quadriflow [64]. As shown in the table, the minimum Scaled Jacobian of our result mesh is much better than that of other methods. Specifically, the minimum Scaled Jacobian of the Enneper mesh generated by our method is 0.43, while that of the mesh generated by Quadriflow is -0.99. Our method is able to preserve the feature line and still guarantee a better quality of mesh near

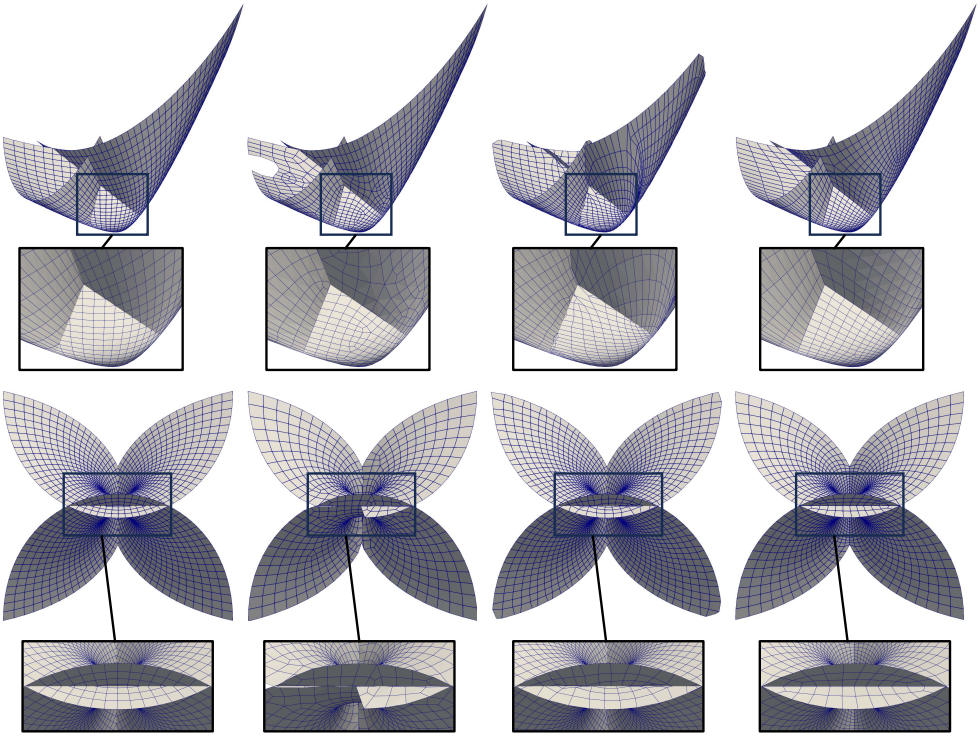

Fig. 15 A comparison of generated surface meshes between MATLAB [3] (left), Quadriflow [64] (middle) and the proposed method (right). The proposed method generates conforming meshes and preserves the boundaries better.

the feature line, whereas Quadriflow generates more flipped faces near the feature line (see Fig. 16). Additionally, the number of singularities in our method is smaller. While Instant-Mesh introduces 136 singularities to improve the mesh quality, our method introduces only 20 singularities in the Rational mesh. Our method balances the number of singularities and the quality of the mesh by building and solving the ILP problem.

Our result meshes have the smallest Hausdorff distance and average distance, better preserving the feature lines. Fig. 13 shows the self-intersecting curves and boundaries extracted from the quadrilateral mesh generated by our method. The self-intersecting curves could not be recognized and extracted from the generated quadrilateral mesh of Quadriflow and Instant-Mesh because they were unable to correctly preserve the self-intersecting curves.

Fig. 14 provides a comparison of the quadrilateral mesh generated over the parameter domain. Our method effectively preserves feature-edges while maintaining high quality near these features, ensuring no offset in the location of feature-vertices. In contrast, the quality of Quadriflow near the feature lines is extremely poor, characterized by numerous flipped faces. Moreover, Quadriflow exhibits significant errors in preserving feature lines and boundaries of the original mesh. Although Instant-Mesh

Table 1 Measurements on the result meshes compared with Instant-mesh [14] and Quadriflow [64].

Model	Metric	Ours	Quadriflow	IM
Bi-quadric2	#V	1759	1668	1601
	#F	1680	1589	1521
	#S	9	18	91
	AD(°)	23.03	33.60	29.21
	ASJ	0.90	0.77	0.84
	MSJ	0.22	-0.99	-0.06
	d_H	0.06	0.44	0.33
	d_A	0.02	0.05	0.04
Enneper	#V	2402	2604	2243
	#F	2318	2503	2170
	#S	6	8	20
	AD(°)	9.94	9.35	10.10
	ASJ	0.97	0.97	0.97
	MSJ	0.43	-0.99	0.39
	d_H	0.07	0.25	0.10
	d_A	0.02	0.03	0.03
Rational	#V	6118	6336	6259
	#F	5962	6170	6103
	#S	20	18	136
	AD(°)	31.15	31.30	32.60
	ASJ	0.80	0.80	0.78
	MSJ	0.30	-0.98	-0.19
	d_H	0.05	0.18	0.08
	d_A	0.02	0.07	0.03
Cusp	#V	2867	2809	2809
	#F	2760	2704	2704
	#S	0	0	0
	AD(°)	38.18	38.60	38.20
	ASJ	0.70	0.70	0.70
	MSJ	0.01	0.00	0.01
	d_H	0.13	0.14	0.14
	d_A	0.03	0.04	0.04
Conoid	#V	1681	1681	1681
	#F	1600	1600	1600
	#S	0	0	0
	AD(°)	1.98	3.40	2.08
	ASJ	0.98	0.98	0.98
	MSJ	0.00	-0.98	-0.14
	d_H	0.02	0.02	0.02
	d_A	0.00	0.01	0.01
Riemann	#V	1681	1681	1681
	#F	1600	1600	1600
	#S	0	0	0
	AD(°)	2.18	5.70	2.30
	ASJ	0.99	0.97	0.99
	MSJ	0.89	0.49	0.88
	d_H	0.15	0.16	0.16
	d_A	0.05	0.05	0.05

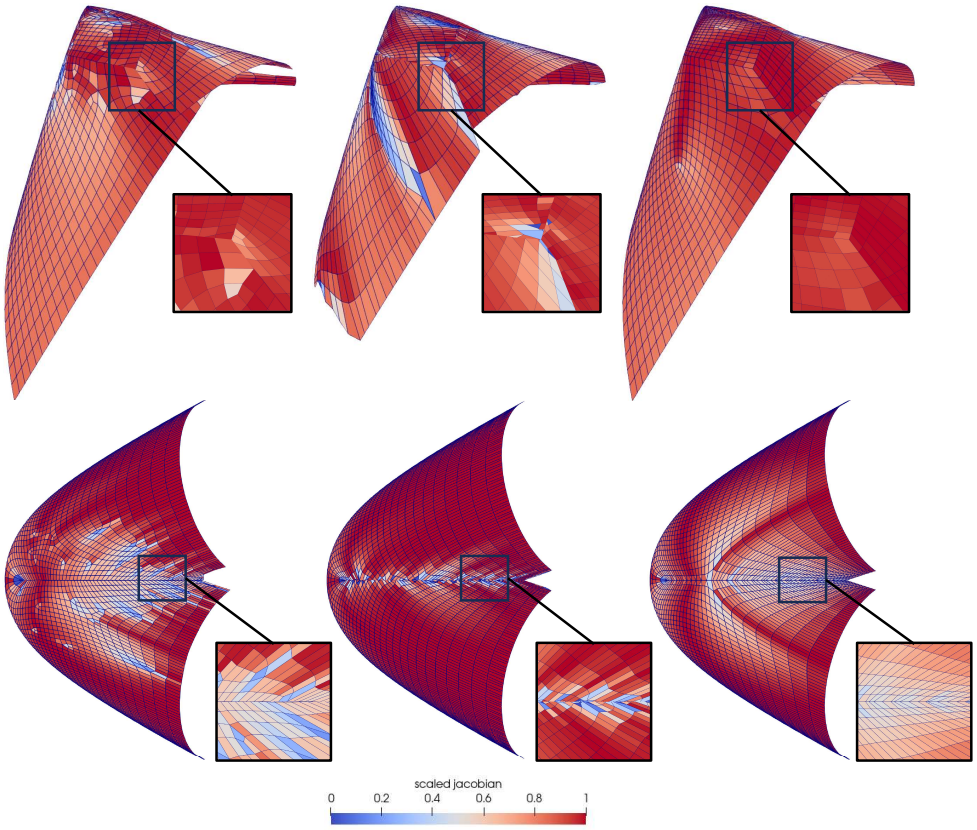

Fig. 16 A comparison of the quality visualization of generated meshes between Instant-Mesh [14] (left), Quadriflow [64] (middle) and the proposed method (right). Scaled Jacobian is color-coded from blue (SJ $\leq$ 0) to red (SJ = 1). The proposed method generates conforming meshes with fewer low-quality and flipped faces.

roughly preserves the shape of feature lines, it fails to maintain the shape of boundaries and does not strictly fix feature vertices.

Fig. 15 shows a comparison of the generated surface quadrilateral meshes. Our method can align the mesh correctly at the self-intersecting curves and preserve the boundaries. In contrast, the meshes generated by MATLAB [3] and Quadriflow fail to correctly align the mesh at the self-intersecting curves.

8 DISCUSSION

In this paper, we propose a fully automatic generation method of quadrilateral meshes for self-intersecting parametric surfaces, which can generate high-quality, conforming and pure-quadrilateral meshes with perfectly preserved features. Our method can correctly handle the self-intersecting lines of the surface, resulting in meshes devoid of flipped faces and erroneous intersections. This ensures that the topology of the surface

mesh is consistent with the parametric surface, attaining a high degree of accuracy. Compared with the state-of-the-art, our method can better preserve boundaries and features while maintaining mesh quality at a high standard. This is evident in various aspects such as quad shapes, the number of singular vertices, and other relevant metrics. Moreover, our method effectively preserves even the most sharp features present on the mesh (see Fig. 17).

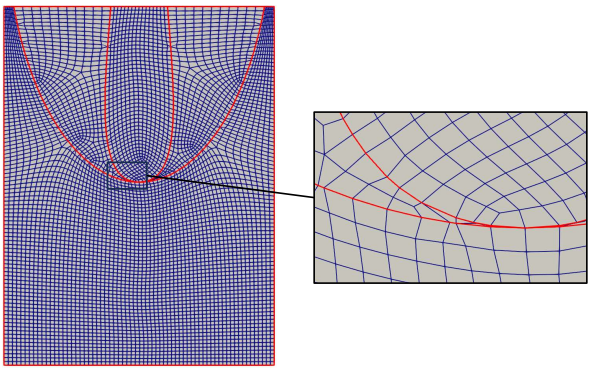

Fig. 17 Feature-edges are correctly preserved even in scenarios where the feature lines form extremely small angles.

Limitations and future work. It's acknowledged that while our method excels in generating high-quality quadrilateral meshes over the parametric domain, there are limitations when these meshes are mapped onto the parametric surface. This mapping process can sometimes cause distortion in the resulting surface mesh (see Fig. 18). Exploring the mapping between the parameter domain and the surface as a metric for mesh generation could indeed be a promising direction for future research. By incorporating considerations of the mapping process into the mesh generation pipeline, it may be possible to mitigate distortions and improve the quality of the resulting surface mesh. This approach could involve optimizing the parameter domain mesh such that it better conforms to the surface geometry during the mapping process, thereby leading to more accurate and faithful representations of the original surface.

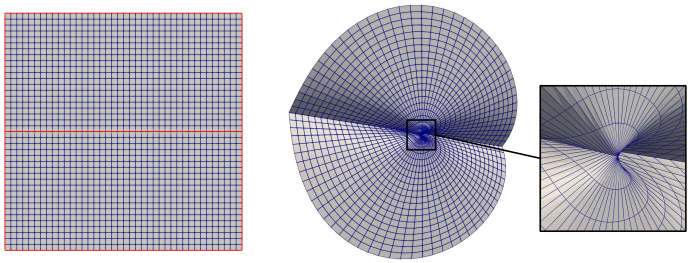

Fig. 18 The faces are distorted after mapping the domain mesh (left) to the surface mesh (right).

Additionally, the results of our method rely on the solution of the ILP problem, and the choice of solver can influence the results. [66] developed a rapid approximate solver for solving the Minimum-Deviation-Flow Problem in bi-directed networks (Bi-MDF) exactly. It can be employed to solve T-Mesh quantization problems more efficiently.

Declarations

Funding. This research was supported by the National Key R&D Program of China under Grant No.2023YFB3309100, the National Natural Science Foundation of China (No.U22A2033), the Zhejiang Provincial Science and Technology Program in China under Grant 2021C01108.

Conflict of interest. The authors have no competing interests to declare that are relevant to the content of this article.

Data availability. Data will be made available on request.

References

[1] Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., Lévy, B.: Polygon Mesh Processing. CRC press, United Kingdom (2010)

[2] Pepper, D.W., Heinrich, J.C.: The Finite Element Method: Basic Concepts and Applications. Taylor & Francis, United Kingdom (2005)

[3] Inc., T.M.: MATLAB Version: 9.13.0 (R2022b), The MathWorks Inc., Natick, Massachusetts, United States (2022). https://www.mathworks.com

[4] Cuillière, J.-C.: An adaptive method for the automatic triangulation of 3d parametric surfaces. Computer-aided design **30**(2), 139–149 (1998)

[5] Shimada, K., Gossard, D.C.: Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis. Computer Aided Geometric Design **15**(3), 199–222 (1998)

[6] Zhong, Z., Guo, X., Wang, W., Lévy, B., Sun, F., Liu, Y., Mao, W., *et al.*: Particle-based anisotropic surface meshing. ACM Trans. Graph. **32**(4), 99–1 (2013)

[7] Zheng, Z., Xiaohong, J., Mingyang, Z., Shiqing, X., Dongming, Y.: Mesh generation for self-intersecting rational parametric surfaces. Journal of Computer-Aided Design & Computer Graphics **35**(12), 1920–1934 (2023) https://doi.org/10.3724/SP.J.1089.2023.2023-00006

[8] D'azevedo, E.F.: Are bilinear quadrilaterals better than linear triangles? SIAM Journal on Scientific Computing **22**(1), 198–217 (2000)

[9] Shepherd, J.F., Johnson, C.R.: Hexahedral mesh generation constraints. Engineering with Computers **24**(3), 195–213 (2008)

[10] Kälberer, F., Nieser, M., Polthier, K.: Quadcover-surface parameterization using branched coverings. In: Computer Graphics Forum, vol. 26, pp. 375–384 (2007). Wiley Online Library

[11] Tarini, M., Pietroni, N., Cignoni, P., Panozzo, D., Puppo, E.: Practical quad mesh simplification. In: Computer Graphics Forum, vol. 29, pp. 407–418 (2010). Wiley Online Library

[12] Bommes, D., Lempfer, T., Kobbelt, L.: Global structure optimization of quadrilateral meshes. In: Computer Graphics Forum, vol. 30, pp. 375–384 (2011). Wiley Online Library

[13] Fang, X., Bao, H., Tong, Y., Desbrun, M., Huang, J.: Quadrangulation through morse-parameterization hybridization. ACM Transactions on Graphics (TOG) **37**(4), 1–15 (2018)

[14] Jakob, W., Tarini, M., Panozzo, D., Sorkine-Hornung, O., *et al.*: Instant field-aligned meshes. ACM Trans. Graph. **34**(6), 189–1 (2015)

[15] Volino, P., Thalmann, N.M.: Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. In: Computer Graphics Forum, vol. 13, pp. 155–166 (1994). Wiley Online Library

[16] Volino, P., Magnenat Thalmann, N.: Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. In: Computer Animation and Simulation'95: Proceedings of the Eurographics Workshop in Maastricht, The Netherlands, September 2–3, 1995, pp. 55–65 (1995). Springer

[17] Lin, M., Gottschalk, S.: Collision detection between geometric models: A survey. In: Proc. of IMA Conference on Mathematics of Surfaces, vol. 1, pp. 602–608 (1998)

[18] Krishnan, S., Manocha, D.: An efficient surface intersection algorithm based on lower-dimensional formulation. ACM Transactions on Graphics (TOG) **16**(1), 74–106 (1997)

[19] Galligo, A., Pavone, J.P.: A sampling algorithm computing self-intersections of parametric surfaces. In: Algebraic Geometry and Geometric Modeling, pp. 185–204 (2006). Springer

[20] Park, Y., Hong, Q.Y., Kim, M.-S., Elber, G.: Self-intersection computation for freeform surfaces based on a regional representation scheme for miter points. Computer Aided Geometric Design **86**, 101979 (2021)

[21] Toth, C.D., O'Rourke, J., Goodman, J.E.: Handbook of Discrete and Computational Geometry. CRC press, United Kingdom (2017)

[22] Jia, X., Chen, F., Yao, S.: Singularity computation for rational parametric surfaces using moving planes. ACM Transactions on Graphics (TOG) **42**(1), 1–14 (2022)

[23] Catmull, E., Clark, J.: Recursively generated b-spline surfaces on arbitrary topological meshes. In: Seminal Graphics: Pioneering Efforts that Shaped the Field, pp. 183–188 (1998)

[24] Owen, S.J., Staten, M.L., Canann, S.A., Saigal, S.: Q-morph: an indirect approach to advancing front quad meshing. International journal for numerical methods in engineering **44**(9), 1317–1340 (1999)

[25] Remacle, J.-F., Lambrechts, J., Seny, B., Marchandise, E., Johnen, A., Geuzainet, C.: Blossom-quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. International journal for numerical methods in engineering **89**(9), 1102–1119 (2012)

[26] Bommes, D., Zimmer, H., Kobbelt, L.: Mixed-integer quadrangulation. ACM transactions on graphics (TOG) **28**(3), 1–10 (2009)

[27] Bommes, D., Campen, M., Ebke, H.-C., Alliez, P., Kobbelt, L.: Integer-grid maps for reliable quad meshing. ACM Transactions on Graphics (TOG) **32**(4), 1–12 (2013)

[28] Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., Zorin, D.: Quad-mesh generation and processing: A survey. In: Computer Graphics Forum, vol. 32, pp. 51–76 (2013). Wiley Online Library

[29] Campen, M.: Partitioning surfaces into quadrilateral patches: A survey. In: Computer Graphics Forum, vol. 36, pp. 567–588 (2017). Wiley Online Library

[30] Dong, S., Bremer, P.-T., Garland, M., Pascucci, V., Hart, J.C.: Spectral surface quadrangulation. In: Acm Siggraph 2006 Papers, pp. 1057–1066 (2006)

[31] Ling, R., Huang, J., Jüttler, B., Sun, F., Bao, H., Wang, W.: Spectral quadrangulation with feature curve alignment and element size control. ACM Transactions on Graphics (TOG) **34**(1), 1–11 (2014)

[32] Gould, J., Martineau, D., Fairey, R.: Automated two-dimensional multi-block meshing using the medial object. In: Proceedings of the 20th International Meshing Roundtable, pp. 437–452 (2012). Springer

[33] Usai, F., Livesu, M., Puppo, E., Tarini, M., Scateni, R.: Extraction of the quad layout of a triangle mesh guided by its curve skeleton. ACM Transactions on Graphics (TOG) **35**(1), 1–13 (2015)

[34] Tarini, M., Puppo, E., Panozzo, D., Pietroni, N., Cignoni, P.: Simple quad

domains for field aligned mesh parametrization. In: Proceedings of the 2011 SIGGRAPH Asia Conference, pp. 1–12 (2011)

[35] Campen, M., Bommes, D., Kobbelt, L.: Dual loops meshing: quality quad layouts on manifolds. ACM Transactions on Graphics (TOG) **31**(4), 1–11 (2012)

[36] Campen, M., Zorin, D.: Similarity maps and field-guided t-splines: a perfect couple. ACM Transactions on Graphics (TOG) **36**(4), 1–16 (2017)

[37] Razafindrazaka, F.H., Polthier, K.: Optimal base complexes for quadrilateral meshes. Computer Aided Geometric Design **52**, 63–74 (2017)

[38] Panozzo, D., Puppo, E., Tarini, M., Pietroni, N., Cignoni, P.: Automatic construction of quad-based subdivision surfaces using fitmaps. IEEE Transactions on Visualization and Computer Graphics **17**(10), 1510–1520 (2011)

[39] Kowalski, N., Ledoux, F., Frey, P.: Automatic domain partitioning for quadrilateral meshing with line constraints. Engineering with Computers **31**, 405–421 (2015)

[40] Viertel, R., Osting, B., Staten, M.: Coarse quad layouts through robust simplification of cross field separatrix partitions. arXiv preprint arXiv:1905.09097 (2019)

[41] Bommes, D., Vossemer, T., Kobbelt, L.: Quadrangular parameterization for reverse engineering. In: Mathematical Methods for Curves and Surfaces: 7th International Conference, MMCS 2008, Tønsberg, Norway, June 26-July 1, 2008, Revised Selected Papers 7, pp. 55–69 (2010). Springer

[42] Myles, A., Pietroni, N., Kovacs, D., Zorin, D.: Feature-aligned t-meshes. ACM Transactions on Graphics (TOG) **29**(4), 1–11 (2010)

[43] Campen, M., Bommes, D., Kobbelt, L.: Quantized global parametrization. Acm Transactions On Graphics (tog) **34**(6), 1–12 (2015)

[44] Vaxman, A., Campen, M., Diamanti, O., Panozzo, D., Bommes, D., Hildebrandt, K., Ben-Chen, M.: Directional field synthesis, design, and processing. In: Computer Graphics Forum, vol. 35, pp. 545–572 (2016). Wiley Online Library

[45] Viertel, R., Osting, B.: An approach to quad meshing based on harmonic cross-valued maps and the ginzburg–landau theory. SIAM Journal on Scientific Computing **41**(1), 452–479 (2019)

[46] Kowalski, N., Ledoux, F., Frey, P.: A pde based approach to multidomain partitioning and quadrilateral meshing. In: Proceedings of the 21st International Meshing Roundtable, pp. 137–154 (2013). Springer

[47] Fogg, H.J., Armstrong, C.G., Robinson, T.T.: Automatic generation of multi-block decompositions of surfaces. International Journal for Numerical Methods in Engineering **101**(13), 965–991 (2015)

[48] Razafindrazaka, F.H., Reitebuch, U., Polthier, K.: Perfect matching quad layouts for manifold meshes. In: Computer Graphics Forum, vol. 34, pp. 219–228 (2015). Wiley Online Library

[49] Pietroni, N., Puppo, E., Marcias, G., Scopigno, R., Cignoni, P.: Tracing field-coherent quad layouts. In: Computer Graphics Forum, vol. 35, pp. 485–496 (2016). Wiley Online Library

[50] Myles, A., Pietroni, N., Zorin, D.: Robust field-aligned global parametrization. ACM Trans. Graph. **33**(4), 135–1 (2014)

[51] Lyon, M., Campen, M., Bommes, D., Kobbelt, L.: Parametrization quantization with free boundaries for trimmed quad meshing. ACM Transactions on Graphics (TOG) **38**(4), 1–14 (2019)

[52] Eppstein, D., Erickson, J.: Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. In: Proceedings of the Fourteenth Annual Symposium on Computational Geometry, pp. 58–67 (1998)

[53] Eppstein, D., Goodrich, M.T., Kim, E., Tamstorf, R.: Motorcycle graphs: Canonical quad mesh partitioning. In: Computer Graphics Forum, vol. 27, pp. 1477–1486 (2008). Wiley Online Library

[54] Schertler, N., Panozzo, D., Gumhold, S., Tarini, M.: Generalized motorcycle graphs for imperfect quad-dominant meshes. ACM Transactions on Graphics **37**(4) (2018)

[55] Pietroni, N., Nuvoli, S., Alderighi, T., Cignoni, P., Tarini, M., *et al.*: Reliable feature-line driven quad-remeshing. ACM Transactions on Graphics **40**(4), 1–17 (2021)

[56] Takayama, K., Panozzo, D., Sorkine-Hornung, O.: Pattern-based quadrangulation for n-sided patches. In: Computer Graphics Forum, vol. 33, pp. 177–184 (2014). Wiley Online Library

[57] Chew, L.P.: Constrained delaunay triangulations. In: Proceedings of the Third Annual Symposium on Computational Geometry, pp. 215–222 (1987)

[58] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: Mesh optimization. In: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, pp. 19–26 (1993)

[59] Li, W.-C., Vallet, B., Ray, N., Lévy, B.: Representing higher-order singularities

in vector fields on piecewise linear surfaces. IEEE Transactions on Visualization and Computer Graphics **12**(5), 1315–1322 (2006)

[60] Tarini, M.: Closed-form quadrangulation of n-sided patches. Computers & Graphics **107**, 60–65 (2022)

[61] Team, S.D.: SymPy: a library for symbolic mathematics. https://docs.sympy.org/latest/index.html (2024)

[62] CNR: The Visualization and Computer Graphics Library. http://vcg.isti.cnr.it/vcglib/. (2013)

[63] Optimization, L.G.: Gurobi Optimizer Reference Manual. http://www.gurobi.com (2018)

[64] Huang, J., Zhou, Y., Niessner, M., Shewchuk, J.R., Guibas, L.J.: Quadriflow: A scalable and robust method for quadrangulation. In: Computer Graphics Forum, vol. 37, pp. 147–160 (2018). Wiley Online Library

[65] Stimpson, C., Ernst, C., Knupp, P., Pébay, P., Thompson, D.: The verdict library reference manual. Sandia National Laboratories Technical Report **9**(6), 8 (2007)

[66] Heistermann, M., Warnett, J., Bommes, D.: Min-deviation-flow in bi-directed graphs for t-mesh quantization. ACM Trans. Graph **42**(4) (2023)