Computational Visual Media
https://doi.org/10.1007/s41095-021-0257-1 Vol. 8, No. 4, December 2022, 571–583

Research Article

Constructing self-supporting surfaces with planar quadrilateral elements

Long Ma[1,2,*], Sidan Yao[2,*], Jianmin Zheng[2], Yang Liu[3], Yuanfeng Zhou[1], Shi-Qing Xin[4], and Ying He[2] (✉)

Abstract We present a simple yet effective method for constructing 3D self-supporting surfaces with planar quadrilateral (PQ) elements. Starting with a triangular discretization of a self-supporting surface, we first compute the principal curvatures and directions of each triangular face using a new discrete differential geometry approach, yielding more accurate results than existing methods. Then, we smooth the principal direction field to reduce the number of singularities. Next, we partition all faces into two groups in terms of principal curvature difference. For each face with small curvature difference, we compute a stretch matrix that turns the principal directions into a pair of conjugate directions. For the remaining triangular faces, we simply keep their smoothed principal directions. Finally, applying a mixed-integer programming solver to the mixed principal and conjugate direction field, we obtain a planar quadrilateral mesh. Experimental results show that our method is computationally efficient and can yield high-quality PQ meshes that well approximate the geometry of the input surfaces and maintain their self-supporting properties.

Keywords self-supporting surfaces; planar quadrilateral (PQ) meshes; conjugate direction fields (CDFs)

* Long Ma and Sidan Yao contributed equally to this work.
1 School of Software, Shandong University, Jinan 250101, China. E-mail: L. Ma, malong@sdu.edu.cn; Y. Zhou, yfzhou@sdu.edu.cn.
2 School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798, Singapore. E-mail: S. Yao, syao003@e.ntu.edu.sg; J. Zheng, ASJMZheng@ntu.edu.sg; Y. He, yhe@ntu.edu.sg (✉).
3 Internet Graphics Group, Microsoft Research Asia, Beijing 100080, China. E-mail: yangliu@microsoft.com.
4 School of Computer Science and Technology, Shandong University, Qingdao 266237, China. E-mail: xinshiqing@sdu.edu.cn.
Manuscript received: 2021-07-23; accepted: 2021-10-01

1 Introduction

Self-supporting surfaces, as one of the most ancient and elegant techniques for building curved shapes, can be found in many architectural heritages, because of their advantageous structural properties and efficient use of material. A 3D surface is self-supporting if the internal stresses are along the tangent directions so that they balance the surface's weight, and there is no shear stress or bending moment within the surface [1–3]. Designing self-supporting surfaces is an over-constrained problem since there are 3 constraints in the equilibrium equation whereas the surface has only 2 degrees of freedom, which are the isotropic stress function and the z-coordinate.

Constructing self-supporting surfaces has received considerable attention in architectural geometry [4] this century, leading to several computational tools [5–9]. Most existing self-supporting surface generation methods target triangle meshes, the dominant representation in digital geometry processing thanks to its flexibility and ease of construction. We note that quadrilateral meshes are highly desired in architectural geometry since most smooth surfaces have two dominant local directions, associated with principal curvature directions, and aligning quad elements with given directions is a natural way to capture shape features. Using triangle meshes, however, necessitates an arbitrary choice of a third edge direction. Moreover, triangle meshes require more edges than quad meshes, so are heavier in architectural design [4].

In this paper, we are interested in modeling self-supporting surfaces with *planar quadrilateral* (PQ) meshes which are highly desired for glass structures [10–12]. Since PQ meshes are the discrete

counterpart of conjugate curve networks [13, 14], computing conjugate directions plays a critical role in PQ mesh construction [4, 11]. Existing methods [11, 12, 15, 16] apply global optimization to compute conjugate directions on polygonal meshes. Though they can yield smooth PQ meshes, these methods are computationally expensive, and it is not easy to find physically-valid solutions due to high non-linearity in the optimization.

To overcome the aforementioned challenges in existing methods, we propose a simple yet effective method for constructing self-supporting surfaces with planar quadrilateral elements. Our idea is to parameterize an existing triangle-mesh-based self-supporting surface and then extract a PQ mesh from the parameterization. Our method takes a triangular discretization of a self-supporting surface as input, and the principal curvatures and directions for each triangular face are calculated using a new discrete differential geometry approach, yielding more accurate results than existing methods. Then, the principal direction field is smoothed to reduce the number of singularities. Next, all faces are partitioned into two groups in terms of principal curvature differences. We compute a stretch matrix for faces with similar principal curvatures; it turns the principal directions into a pair of conjugate directions. For faces with large curvature difference, we keep the smoothed principal directions. Finally, a mixed-integer programming solver is applied to the mixed principal directions and conjugate directions to compute a planar quadrilateral mesh.

Since our method does not require any non-linear optimization for computing conjugate direction fields (CDFs), it is easy to implement and computationally efficient. Experimental results on various self-supporting surfaces (including both height and non-height fields) demonstrate the effectiveness and computational efficiency of our method when generating high-quality self-supporting surface with planar quadrilateral elements. Figure 1 illustrates an example generated by our method.

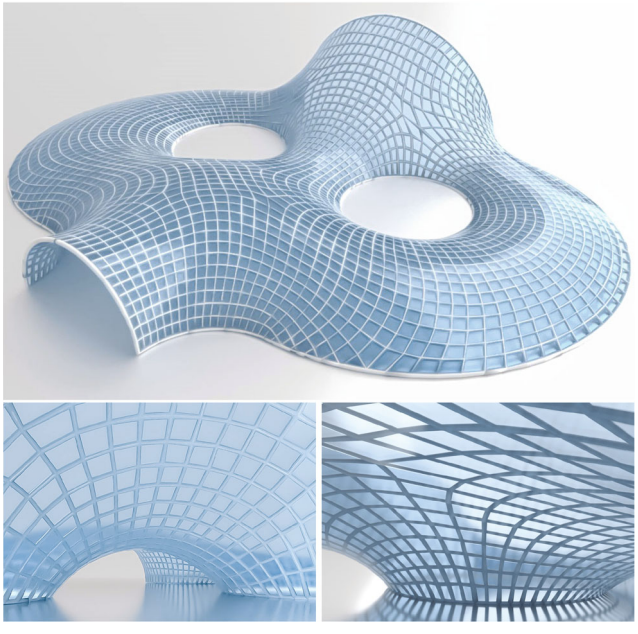


Fig. 1 Given a self-supporting surface as a triangular mesh, our method converts it into a planar quadrilateral mesh that approximates the input surface faithfully both geometrically and in physical performance.

2 Related work

2.1 Self-supporting surfaces

Self-supporting surfaces are often designed by TNA, a popular computational tool developed by Block and his colleagues [8, 17]. TNA approximates the stress field through uniaxial singular stresses and factors the over-constrained 3D equilibrium problem into horizontal and vertical sub-problems, making it easier to solve. Vouga et al. [18] applied TNA to approximate freeform shapes by self-supporting ones. Since their method solves an over-constrained linear system, it cannot guarantee convergence when vertices have high valence and/or large deformation is needed to make the reference mesh self-supporting.

Miki et al. [19] generalized the Airy stress function in the planar elasticity problem to the 3D thin shell problem and developed a method for computing self-supporting spline surfaces with high-order smoothness. However, the mean curvature of the thin shell causes the Airy stress function to be inconsistent with the true mechanical equilibrium. Therefore, this method can only compute approximate solutions.

Both Liu et al. [6] and de Goes et al. [5] projected 3D surfaces onto the 2D domain to simplify the problem. De Goes et al.'s method allows the user to specify the stress distribution, giving greater flexibility. Liu et al.'s method uses regular triangulation to ensure equilibrium in the x- and y-directions.

Ma et al. [9] proved that the hyper-generatrix of a 4D minimal hyper-surface of revolution is a 3D

self-supporting surface. As a result, constructing a 3D self-supporting surface is equivalent to 4D volume minimization, which can be further converted into a surface reconstruction problem with mean curvature constraint [20, 21]. Ma et al. showed that their method computes solutions of the over-constrained equilibrium problem more accurately than the TNA-based approaches [5, 6, 18]. The input of our method is a 3D self-supporting surface represented by a triangle mesh, as generated by Ma et al.'s method [9].

2.2 Conjugate direction fields

Conjugate direction fields are at the core of constructing PQ meshes, and in discrete geometry as counterparts of conjugate curve networks. The first algorithm for designing PQ meshes, proposed by Liu et al. [11], optimized the fairness of mesh edges, with face planarity constraints. Zadravec et al. [12] computed smooth CDFs by optimizing the smoothness of the representation vector field. Their method can produce high-quality quad-dominant meshes with planar faces, but it is unable to handle $\pm k/4$ ($k \in \mathbb{Z}^2$) singularities, which are common for surfaces with convex corners such as a rounded cube. Liu et al.'s method [15] allows the presence of such singularities, but at the cost of solving a non-linear optimization problem.

3 Method

3.1 Overview

Our method takes a self-supporting surface represented by triangle mesh as input. We first compute the principal curvatures and directions for each triangular face using a new method for computing the curvature tensor on triangle meshes. Then we group the triangular faces whose principal curvature difference is small. We call these faces *free faces* and the remaining faces *fixed faces*. Since the principal directions on fixed charts are already conjugate, our goal is to construct a pair of conjugate directions for each free chart. Towards this goal, we compute a stretching matrix that turns the principal directions of free charts into conjugate directions. The principal directions of the fixed charts and the conjugate directions of the free charts induce a conjugate direction field on the input mesh. Applying the mixed integer quadrangulation method [22] to the CDF, we parameterize the triangle mesh and extract the planar quadrilateral elements. We present the algorithmic details in Sections 3.2–3.6. We show the algorithmic pipeline of our method in Fig. 2 and pseudocode in Algorithm 1. To ease reading, we list major notation in Table 1.

Algorithm 1 Constructing 3D self-supporting surfaces with planar quadrilateral elements

Require: A self-supporting surface as a triangular mesh $M = (V, E, F)$ and the threshold ε of principal curvature difference
Result: A planar quad mesh representing the surface
Compute the principal curvatures and directions using Algorithm 2
Smooth the principal directions to reduce singularities
for each face $f \in F$ **do**
  if $\kappa_1\kappa_2 < 0$ **then**
    Mark f *fixed*
  else
    Compute $\lambda = \min\{|\kappa_1|, |\kappa_2|\}/\max\{|\kappa_1|, |\kappa_2|\}$
    if $\lambda < \varepsilon$ **then**
      Mark f *free*
      Compute the stretch matrix $\boldsymbol{S}$ using Eq. (4)
      Stretch the principal directions using $\boldsymbol{S}$
    else
      Mark f *fixed*
    end if
  end if
end for
Apply the MIQ solver to the mixed cross and conjugate direction fields
Extract quad mesh from the parameterization

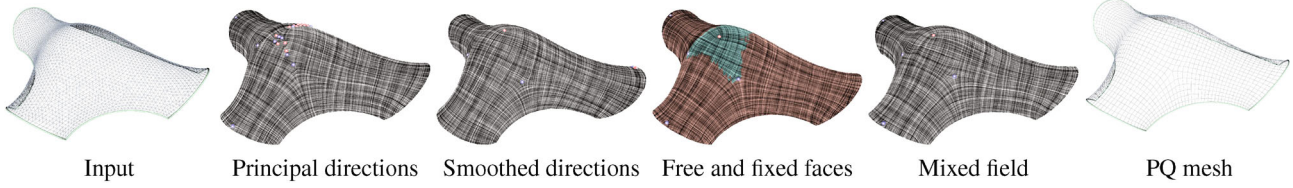


Fig. 2 Algorithmic pipeline. The input is a 3D self-supporting surface represented by a triangle mesh. First, we compute principal curvatures and principal directions for each triangular face. Then, we group the faces with similar principal curvatures, which are called free faces (green). We compute a smooth cross field on the free faces and generate a pair of conjugate directions by stretching the principal directions. Next, we parameterize the triangle mesh using the mixed principal direction field and conjugate direction field. Finally, we extract the planar quadrilateral mesh from the parameterization.

Table 1 Notations

Item	Description
$\Omega \subset \mathbb{R}^2$	2D domain
$u, v \in \Omega$	Parameters
$\boldsymbol{r}(u,v) \subset \mathbb{R}^3$	3D surface
$M = (V, E, F)$	Input 3D triangular mesh
$\boldsymbol{e}_1$, $\boldsymbol{e}_2$	Local coordinate axes for each face
$\boldsymbol{n}$	Unit normal
$\boldsymbol{C}$	Curvature tensor
κ_1, κ_2	Principal curvatures
$\boldsymbol{d}_1$, $\boldsymbol{d}_2$	Principal directions
$\boldsymbol{S}$	Stretch matrix
$\boldsymbol{u}$, $\boldsymbol{v}$	Conjugate tangent directions

3.2 Computing principal curvatures and directions

Curvature is the amount by which a curve deviates from a straight line, or a surface deviates from planar. For a smooth 3D surface $\boldsymbol{r}(u,v)$, the curvature tensor $\boldsymbol{C}$ satisfies $\boldsymbol{C}\boldsymbol{r}_u = -\boldsymbol{n}_u$ and $\boldsymbol{C}\boldsymbol{r}_v = -\boldsymbol{n}_v$, where $\boldsymbol{r}_u$, $\boldsymbol{r}_v$ are the tangent vectors of $\boldsymbol{S}$ and $\boldsymbol{n}_u$, $\boldsymbol{n}_v$ the partial derivatives of unit normal $\boldsymbol{n}$.

Since we deal with 3D self-supporting surfaces represented by triangle meshes, we need to discretize $\boldsymbol{C}$. The key idea is to consider $\boldsymbol{r}_u$, $\boldsymbol{r}_v$ as the movement of observation points on the surface. Consider a triangular face $\triangle ABC$ and its neighbors $\triangle CBD$, $\triangle ACF$, and $\triangle BAE$. Let $\boldsymbol{n}$, $\boldsymbol{n}_i$ $(i = 1, 2, 3)$ be the normals of $\triangle ABC$, $\triangle CBD$, $\triangle ACF$, and $\triangle AEB$, and $\delta\boldsymbol{n}_i$ be the derivatives of these normals.

Observe that each face normal is the cross product of two edge vectors, and each vertex normal is computed as the weighted average of face normals. Therefore, face normals are usually more accurate than vertex normals on triangle meshes. Our movement strategy is to use the centers O and O_i $(i = 1, 2, 3)$ of the circumscribed circles of $\triangle ABC$ and its neighbors $\triangle BCD$, $\triangle ACF$, and $\triangle BAE$. We compute the derivatives as

$$\delta\boldsymbol{r}_i = \overrightarrow{OO}_i \tag{1}$$

and

$$\delta\boldsymbol{n}_i = \boldsymbol{n} - \boldsymbol{n}_i \tag{2}$$

for $i = 1, 2, 3$. Since the curvature tensor $\boldsymbol{C}$ satisfies $\boldsymbol{C}\delta\boldsymbol{r}_i = -\delta\boldsymbol{n}_i$ $(i = 1, 2, 3)$, we form an over-constrained linear system:

$$\begin{bmatrix} \delta\boldsymbol{r}_1\cdot\boldsymbol{e}_1 & \delta\boldsymbol{r}_1\cdot\boldsymbol{e}_2 & 0 \\ 0 & \delta\boldsymbol{r}_1\cdot\boldsymbol{e}_1 & \delta\boldsymbol{r}_1\cdot\boldsymbol{e}_2 \\ \delta\boldsymbol{r}_2\cdot\boldsymbol{e}_1 & \delta\boldsymbol{r}_2\cdot\boldsymbol{e}_2 & 0 \\ 0 & \delta\boldsymbol{r}_2\cdot\boldsymbol{e}_1 & \delta\boldsymbol{r}_2\cdot\boldsymbol{e}_2 \\ \delta\boldsymbol{r}_3\cdot\boldsymbol{e}_1 & \delta\boldsymbol{r}_3\cdot\boldsymbol{e}_2 & 0 \\ 0 & \delta\boldsymbol{r}_3\cdot\boldsymbol{e}_1 & \delta\boldsymbol{r}_3\cdot\boldsymbol{e}_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \delta\boldsymbol{n}_1\cdot\boldsymbol{e}_1 \\ \delta\boldsymbol{n}_1\cdot\boldsymbol{e}_2 \\ \delta\boldsymbol{n}_2\cdot\boldsymbol{e}_1 \\ \delta\boldsymbol{n}_2\cdot\boldsymbol{e}_2 \\ \delta\boldsymbol{n}_3\cdot\boldsymbol{e}_1 \\ \delta\boldsymbol{n}_3\cdot\boldsymbol{e}_2 \end{bmatrix} \tag{3}$$

where $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are the axes of the local coordinate system on $\triangle ABC$. Then we compute x_i $(i = 1, 2, 3)$ by solving a linear least squares problem. After that, we can form the curvature tensor $\boldsymbol{C}$:

$$\boldsymbol{C} = \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \end{bmatrix}$$

since $\boldsymbol{C}$ is symmetric. The eigenvalues κ_i and eigenvectors $\boldsymbol{d}_i$ $(i = 1, 2)$ of $\boldsymbol{C}$ are the principal curvatures and directions of face $\triangle ABC$. Algorithm 2 gives pseudocode for computing principal curvatures and directions on triangle meshes.

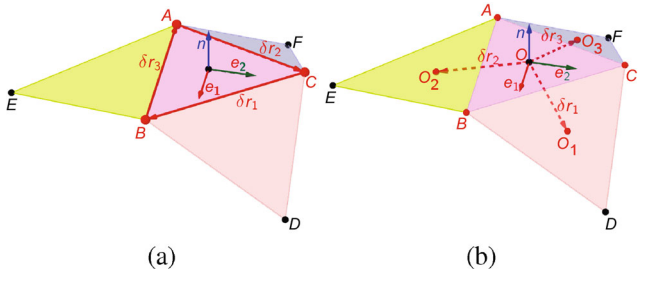


Fig. 3 Difference between the observation point movement strategy used in Rusinkiewicz's method [23] and our method. (a) In Ref. [23], the derivatives are $\delta\boldsymbol{r}_1 = \overrightarrow{CB}$, $\delta\boldsymbol{r}_2 = \overrightarrow{AC}$, and $\delta\boldsymbol{r}_3 = \overrightarrow{BA}$, and are based on mesh vertices. (b) In our method, we define $\delta\boldsymbol{r}_i = \overrightarrow{OO}_i$, where O and O_i are the centers of the circumscribed circles of $\triangle ABC$ and its neighbors.

Algorithm 2 Computing principal curvatures and directions on triangle meshes

Data: A triangular mesh $M = (V, E, F)$
Result: Principal curvatures and directions for each triangular face $f \in F$
for each $f \in F$ **do**
  Compute the circumscribed circle center of f
  Compute the per-face normal using cross product
  Construct a local coordinate system for f
end for
for each face $f \in F$ **do**
  for $i = 1$ to 3 **do**
    Compute $\delta\boldsymbol{r}_i$ and $\delta\boldsymbol{n}_i$ using Eqs. (1) and (2)
  end for
  Solve Eq. (3) using linear least squares
  Form matrix $\boldsymbol{C}$
  Compute the eigenvalues and eigenvectors of $\boldsymbol{C}$
end for

3.3 Smoothing principal directions

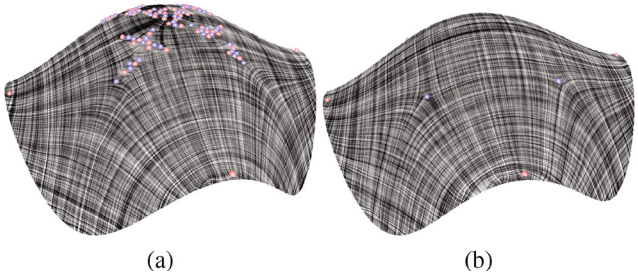
The computed principal directions are usually not smooth due to discretization. As a result, the

principal direction fields contain many undesired singularities, which compromises the parameterization quality. To reduce singularities, we apply an iterative method to locally adjust singularities. Specifically, the method moves singularities based on their indices: singularities of the same index repel, and of opposite index attract. When a pair of opposite-index singularities meet, they cancel each other out, reducing the number of singularities by two. After all singularities stop moving, we obtain a smoothed principal direction field with fewer singularities than the original. Figure 4 shows such a principal curvature direction field before and after optimization. We observe that the number of singularities is significantly reduced using this local smoothing method.

3.4 Distinguishing fixed and free faces

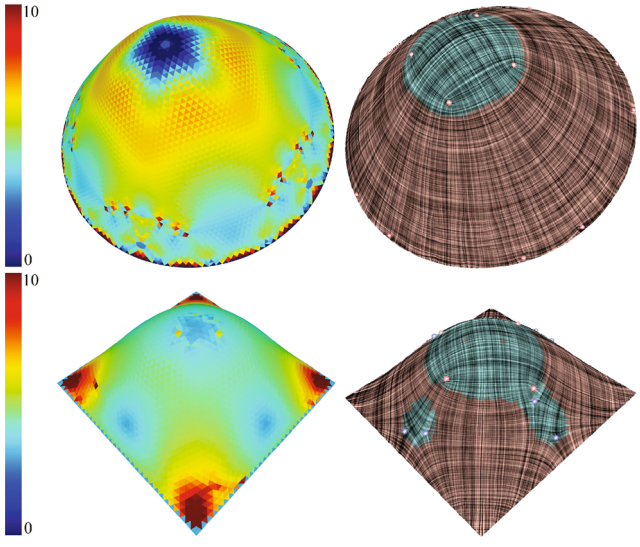
On a self-supporting surface, some parts typically exist which are spherical or nearly so (see Fig. 5(a)). It is difficult to calculate principal directions on these parts, and the direction field produced in Section 3.2 is inaccurate there. However, using the ideas of Liu et al. [11, 15], principal direction fields are not the only way to make a PQ mesh: conjugate direction fields can also be used for this purpose. Thus we need to manipulate these cross vectors into conjugate directions.

Regions with large principal curvature differences can be ignored, since we can directly use the computed principal directions, as they are already conjugate. We need to pay attention to regions with small principal curvature differences (i.e., almost spherical regions), on which principal directions are ill-defined. A face is classified as a *free face* when its principal curvatures satisfy



(a) (b)

Fig. 4 Smoothing principal directions. (a) There are many singularities in the principal direction field. (b) The singularities are reduced significantly after smoothing. The red and blue spheres are singularities with index 1/4 and −1/4, respectively.



(a) (b)

Fig. 5 Distinguishing fixed and free faces. (a) We compute the quantity λ from principal curvatures κ_1 and κ_2 on each triangular face, $\lambda = \min\{|\kappa_1|, |\kappa_2|\}/\max\{|\kappa_1|, |\kappa_2|\}$. (b) Faces with $\lambda \geqslant \varepsilon$ are marked as *fixed* (brown) and the remainder are *free* faces (green).

$\min\{|\kappa_1|, |\kappa_2|\}/\max\{|\kappa_1|, |\kappa_2|\} < \varepsilon$, where ε is a user-specified threshold. In our implementation, we set $\varepsilon = \sqrt{3}$. See Fig. 5 for two examples showing classification of fixed and free faces.

3.5 Computing the stretch matrix $\boldsymbol{S}$

To generate conjugate directions for free faces, we compute a matrix $\boldsymbol{S}$, which stretches the principal directions.

Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be the tangent vectors at point $\boldsymbol{p}$ on the surface. If $\boldsymbol{p}$, $\boldsymbol{p}+s\boldsymbol{u}$, $\boldsymbol{p}+t\boldsymbol{v}$, and $\boldsymbol{p}+s\boldsymbol{u}+t\boldsymbol{v}$ form a planar quadrilateral for some scalars s and t, they satisfy the following constraint:

$$(s\boldsymbol{u}+t\boldsymbol{v})^{\mathrm{T}}\boldsymbol{C}(s\boldsymbol{u}+t\boldsymbol{v}) = s^2\boldsymbol{u}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{u} + t^2\boldsymbol{v}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{v}$$

where $\boldsymbol{C}$ is the curvature tensor at $\boldsymbol{p}$. This implies the tangent directions $\boldsymbol{u}$ and $\boldsymbol{v}$ satisfy:

$$\boldsymbol{u}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{v} = 0$$

so are conjugate directions. If the edges of each quadrilateral face follow conjugate directions, its vertices are thus co-planar. Therefore, conjugate directions are often used in constructing planar quadrilateral meshes [11, 12, 15].

The surface curvature tensor $\boldsymbol{C}$ is

$$\boldsymbol{C} = L\boldsymbol{r}_u \otimes \boldsymbol{r}_u + M(\boldsymbol{r}_u \otimes \boldsymbol{r}_v + \boldsymbol{r}_v \otimes \boldsymbol{r}_u) + N\boldsymbol{r}_v \otimes \boldsymbol{r}_v$$

where L, M, and N are the coefficients of the second fundamental form, and $\otimes$ is tensor product. It is well known that the principal directions $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ are

perpendicular and satisfy $\boldsymbol{d}_1^{\mathrm{T}}\boldsymbol{C}\boldsymbol{d}_2 = 0$. Furthermore, the principal curvatures κ_1 and κ_2 are the eigenvalues of $\boldsymbol{C}$: $\boldsymbol{C}\boldsymbol{d}_i = \kappa_i \boldsymbol{d}_i$ $(i = 1, 2)$.

Consider a unit vector $\boldsymbol{v}$. $\boldsymbol{v}\otimes\boldsymbol{v}$ is a second order tensor. For an arbitrary vector $\boldsymbol{r}$, we have $(\boldsymbol{v}\otimes\boldsymbol{v})\boldsymbol{r} = (\boldsymbol{v}\cdot\boldsymbol{r})\boldsymbol{v}$. If $\boldsymbol{r}$ and $\boldsymbol{v}$ have the same or opposite direction, the result is equal to $\boldsymbol{r}$. If $\boldsymbol{r}$ and $\boldsymbol{v}$ are perpendicular, the result is the zero vector $\boldsymbol{0}$. Intuitively speaking, the tensor $\boldsymbol{v}\otimes\boldsymbol{v}$, when applied to $\boldsymbol{r}$, computes the projection of $\boldsymbol{r}$ onto $\boldsymbol{v}$.

Now consider the matrix $\boldsymbol{I} - \boldsymbol{v}\otimes\boldsymbol{v}$, where $\boldsymbol{I} \in \mathbb{R}^{2\times 2}$ is the identity matrix. Applying this matrix to vector $\boldsymbol{r}$ yields

$$(\boldsymbol{I} - \boldsymbol{v}\otimes\boldsymbol{v})\boldsymbol{r} = \boldsymbol{r} - (\boldsymbol{v}\cdot\boldsymbol{r})\boldsymbol{v}$$

Since vector $\boldsymbol{r} - (\boldsymbol{v}\cdot\boldsymbol{r})\boldsymbol{v}$ is perpendicular to $\boldsymbol{v}$, the tensor $\boldsymbol{I} - \boldsymbol{v}\otimes\boldsymbol{v}$, when applied to $\boldsymbol{r}$, extracts its component perpendicular to $\boldsymbol{v}$.

Therefore, the combination

$$(\boldsymbol{I} + (s-1)\boldsymbol{v}\otimes\boldsymbol{v})\boldsymbol{r} = (\boldsymbol{I} - \boldsymbol{v}\otimes\boldsymbol{v})\boldsymbol{r} + s(\boldsymbol{v}\otimes\boldsymbol{v})\boldsymbol{r}$$

works to stretch vector $\boldsymbol{r}$ by a factor $s \in \mathbb{R}_+$ along the direction of $\boldsymbol{v}$ while keeping fixed $\boldsymbol{r}$'s component perpendicular to $\boldsymbol{v}$. From the above analysis, the matrix $\boldsymbol{I} + (s-1)\boldsymbol{v}\otimes\boldsymbol{v}$ stretches a vector $\boldsymbol{r}$ by a factor s along v's direction when acting on $\boldsymbol{r}$.

Now we stretch the cross vectors on free faces along the principal directions $\boldsymbol{d}_i$ by a factor of $1/\sqrt{\kappa_i}$. The stretch matrix $\boldsymbol{S}_i$ for each direction $\boldsymbol{d}_i$ is defined as

$$\boldsymbol{S}_i = \boldsymbol{I} + \left(\frac{1}{\sqrt{\kappa_i}} - 1\right)\frac{1}{\boldsymbol{d}_i\cdot\boldsymbol{d}_i}\boldsymbol{d}_i\otimes\boldsymbol{d}_i, \quad i = 1, 2 \tag{4}$$

Since $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ are perpendicular, multiplying by $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ does not influence each direction.

Multiplying $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ yields the stretch matrix $\boldsymbol{S}$:

$$\boldsymbol{S} = \boldsymbol{S}_1\boldsymbol{S}_2 = \boldsymbol{I} + \sum_{i=1}^{2}\left(\frac{1}{\sqrt{\kappa_i}} - 1\right)\frac{1}{\boldsymbol{d}_i\cdot\boldsymbol{d}_i}\boldsymbol{d}_i\otimes\boldsymbol{d}_i \tag{5}$$

since $(\boldsymbol{d}_1\otimes\boldsymbol{d}_1)\boldsymbol{C}(\boldsymbol{d}_2\otimes\boldsymbol{d}_2) = 0$. Also note that

$$\frac{1}{\boldsymbol{d}_1\cdot\boldsymbol{d}_1}\boldsymbol{d}_1\otimes\boldsymbol{d}_1 + \frac{1}{\boldsymbol{d}_2\cdot\boldsymbol{d}_2}\boldsymbol{d}_2\otimes\boldsymbol{d}_2 + \boldsymbol{n}\otimes\boldsymbol{n} = \boldsymbol{I}$$

where $\boldsymbol{n}$ is the unit normal. We can thus re-write $\boldsymbol{S}$ as

$$\boldsymbol{S} = \sum_{i=1}^{2}\frac{1}{\sqrt{\kappa_i}}\frac{1}{\boldsymbol{d}_i\cdot\boldsymbol{d}_i}\boldsymbol{d}_i\otimes\boldsymbol{d}_i + \boldsymbol{n}\otimes\boldsymbol{n} \tag{6}$$

Matrix $\boldsymbol{S}$ is symmetric. Now we show that the matrix $\boldsymbol{S}$ turns two perpendicular tangent directions into a pair of conjugate directions. Since the principal curvatures κ_i and principal directions $\boldsymbol{d}_i$ satisfy the following equations:

$$\begin{cases}(\boldsymbol{d}_1\otimes\boldsymbol{d}_1)\boldsymbol{C}(\boldsymbol{d}_1\otimes\boldsymbol{d}_1) = \kappa_1(\boldsymbol{d}_1\cdot\boldsymbol{d}_1)\boldsymbol{d}_1\otimes\boldsymbol{d}_1\\ (\boldsymbol{d}_2\otimes\boldsymbol{d}_2)\boldsymbol{C}(\boldsymbol{d}_2\otimes\boldsymbol{d}_2) = \kappa_2(\boldsymbol{d}_2\cdot\boldsymbol{d}_2)\boldsymbol{d}_2\otimes\boldsymbol{d}_2\\ (\boldsymbol{d}_1\otimes\boldsymbol{d}_1)\boldsymbol{C}(\boldsymbol{d}_2\otimes\boldsymbol{d}_2) = 0\\ (\boldsymbol{d}_2\otimes\boldsymbol{d}_2)\boldsymbol{C}(\boldsymbol{d}_1\otimes\boldsymbol{d}_1) = 0\end{cases}$$

we obtain

$$\boldsymbol{S}\boldsymbol{C}\boldsymbol{S} = \sum_{i=1}^{2}\frac{1}{\boldsymbol{d}_i\cdot\boldsymbol{d}_i}\boldsymbol{d}_i\otimes\boldsymbol{d}_i = \boldsymbol{I} - \boldsymbol{n}\otimes\boldsymbol{n}$$

Given two arbitrary perpendicular tangent vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, we can easily verify that the stretched vectors $\boldsymbol{S}\boldsymbol{u}$ and $\boldsymbol{S}\boldsymbol{v}$ are conjugate because

$$(\boldsymbol{S}\boldsymbol{u})^{\mathrm{T}}\boldsymbol{C}(\boldsymbol{S}\boldsymbol{v}) = \boldsymbol{u}^{\mathrm{T}}(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{S})\boldsymbol{v} = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{u}^{\mathrm{T}}(\boldsymbol{n}\otimes\boldsymbol{n})\boldsymbol{v} = 0$$

See Fig. 6 for an illustration. Therefore, the stretch matrix $\boldsymbol{S}$ turns perpendicular tangent directions into conjugate directions. In our implementation, we stretch the cross vectors on free faces to generate conjugate directions.

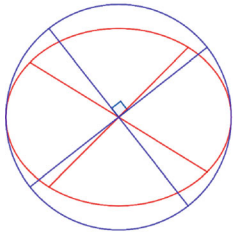


Fig. 6 Consider a circle (blue) with two perpendicular diameters. Stretching the circle with matrix $\boldsymbol{S}$ yields an ellipse and the two perpendicular diameters are conjugate.

As a local operation, computing stretch matrix $\boldsymbol{S}$ is computationally efficient. Figure 7 shows examples of conjugate directions generated by stretching principal directions.

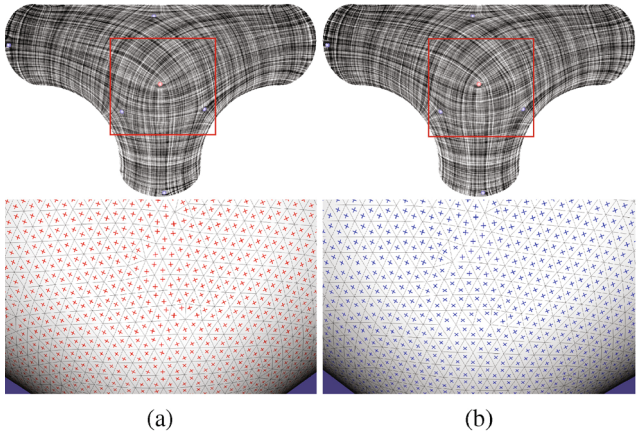


Fig. 7 Computing conjugate directions on free faces. (a) Smoothed principal directions. (b) Conjugate directions.

3.6 Parameterization & generating PQ meshes

Let $(\boldsymbol{u}, \boldsymbol{v}, -\boldsymbol{u}, -\boldsymbol{v})$ be the directional field, consisting of principal directions on fixed faces and conjugate directions on free faces. It is used as a guide to parameterize the triangle mesh and extract PQ elements.

We need to cut the input mesh into a topological disk so that the singularities of the directional field lie on the disk boundary. To do that, we take the singularities as seeds and apply Dijkstra's algorithm to compute a Voronoi diagram on the input mesh. The dual graph of the Voronoi diagram is a triangulation with singularities as vertices. We then segment the triangle mesh into patches along the edges of the Delaunay triangulation. Then, starting with an arbitrary patch, we iteratively glue the patches together by traversing them in breadth-first-search order. Since each patch is used only once, the resulting shape is a topological disk with singularities on its boundary.

To compute the parameterization, we adapt the global parametrization used in Liu et al.'s work [15]. Denote by (s, t) the parameters assigned to each vertex of the input triangle mesh. We minimize the following energy function:

$$E_p = \sum_{f_i \in F} A_i \left[\left(\frac{\nabla s_i}{\|\nabla s_i\|} \cdot \boldsymbol{u}_i^{\perp} \right)^2 + \left(\frac{\nabla t_i}{\|\nabla t_i\|} \cdot \boldsymbol{v}_i^{\perp} \right)^2 \right] \quad (7)$$

where A_i is the area of face f_i and $\perp$ denotes 90° rotation about the normal. To make the parameterization seamless, for each edge on the cut graph, we require the projections of parameter gradients on the two adjacent faces to match, leading to integer translations on the parameters.

Equation (7) is a nonlinear energy function with integer variables introduced at the topology cut and singularities [22]. Like in Liu et al.'s method [15], we solve the mixed integer programming using a greedy heuristic. We perform three operations in each iteration. First, we relax the integer constraints and minimize the function using the L-BFGS method [24]. Second, we pick an integer variable that is closest to integer, round it off, and fix it. We repeat the first two steps until all integer variables are fixed.

4 Experimental results

We implemented our algorithm in C++ and evaluated it on a laptop with a 2.90 GHz Intel Core i7 CPU and 8 GB memory.

4.1 Principal curvatures and directions

To calculate the curvature tensor $\boldsymbol{C}$, our method uses the observation point movement strategy, based on the centers of circumscribed circles. Here we compare our method to Rusinkiewicz's method [23] in terms of accuracy and robustness on a torus model:

$$((R + r\cos v)\cos u,\ (R + r\cos v)\sin u,\ r\sin v)^{\mathrm{T}}$$

setting $R = \sqrt{10}$ and $r = 1$. The principal curvatures are $\kappa_1 = \cos v/(R + r\cos v)$ and $\kappa_2 = 1/r$, while the principal directions are $\boldsymbol{d}_1 = (-\sin u, \cos u, 0)^{\mathrm{T}}$ and $\boldsymbol{d}_2 = (-\sin v\cos u, -\sin v\sin u, \cos v)^{\mathrm{T}}$. We generated two sets of triangle meshes with different degrees of anisotropy τ and resolutions. The anisotropy degree τ for a triangle f is defined as $\tau(f) = PH/(2\sqrt{3}A)$, where P, H, A are the half-perimeter, longest edge length, and area of f, respectively. $\tau \geqslant 1$ for all triangles and equality holds for equilateral triangles. The numerical results show that both methods converge with increasing number of resolutions. However, our method is more accurate and robust than theirs, especially for anisotropic meshes. See Fig. 8. Our method has better accuracy than Rusinkiewicz's method [23] since we adopt per-face normals in computing the curvature tensor $\boldsymbol{C}$, while Rusinkiewicz's method [23] uses per-vertex normals, which are more sensitive to triangulation quality than per-face normals and often result in large accumulation errors on anisotropic meshes.

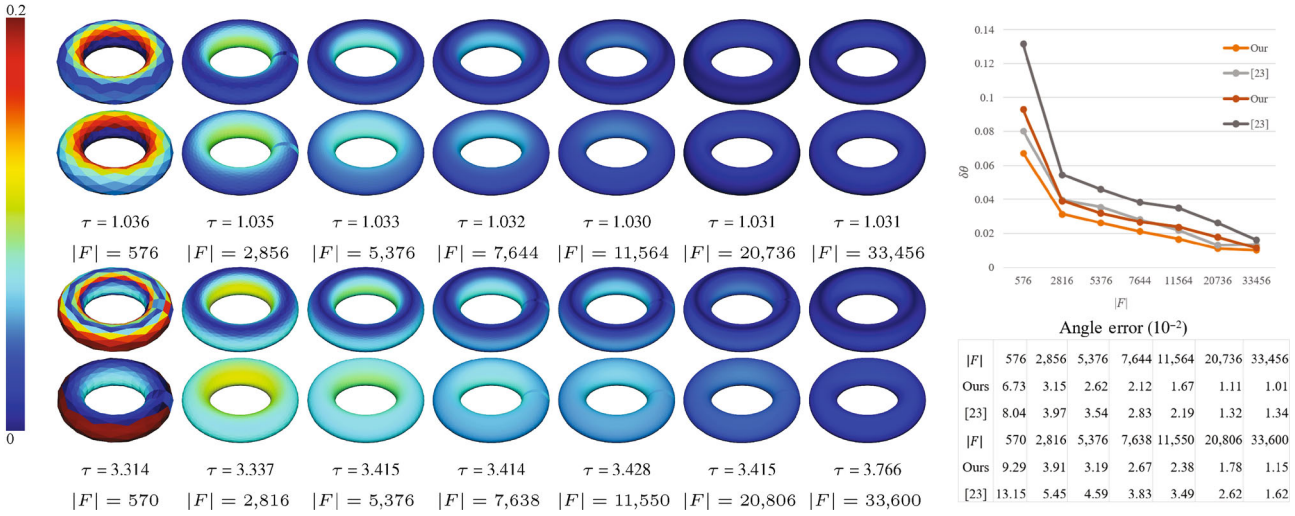
4.2 Evaluation

We evaluated our method and other approaches in terms of both physical and geometrical properties. Like Ref. [9], we adopt three physical measures: normal displacement θ, stress tensor error σ_{S}, and differential stress σ_{D}, to evaluate the stability of self-supporting surfaces. We also measure the smoothness of parameter lines and the planarity of the resulting PQ meshes.

We define the smoothness measure by averaging the discrete geodesic curvatures of the parameter lines (see Fig. 9):

$$\phi = \frac{1}{2n} \sum_{i=1}^{n} (|\pi - \alpha_i| + |\pi - \beta_i|) \quad (8)$$

where n is the number of vertices of the quad mesh. Given a PQ mesh with m patches, we define the planarity measure γ by averaging the volume-to-base ratio:



Angle error (10^{-2})

$\|F\|$	576	2,856	5,376	7,644	11,564	20,736	33,456
Ours	6.73	3.15	2.62	2.12	1.67	1.11	1.01
[23]	8.04	3.97	3.54	2.83	2.19	1.32	1.34
$\|F\|$	570	2,816	5,376	7,638	11,550	20,806	33,600
Ours	9.29	3.91	3.19	2.67	2.38	1.78	1.15
[23]	13.15	5.45	4.59	3.83	3.49	2.62	1.62

Fig. 8 Comparison with Rusinkiewicz's method [23] in terms of accuracy and convergence plots. We generate sequences of isotropic (first two rows) and anisotropic meshes (last two rows) with increasing number of faces. The heat colour map shows the angle differences $\delta\theta$ between the computed principal directions and the ground-truth; warmer colors are larger errors. For each model, we show our results above and theirs below. τ is the anisotropy measure and $|F|$ is the face count.

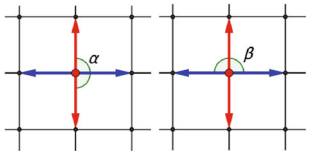


Fig. 9 Discrete geodesic curvatures of the iso-parameter lines.

$$\gamma = \frac{1}{m}\sum_{i=1}^{m}\frac{V_i}{\sqrt{A_i^3}} \tag{9}$$

where V_i is the volume of the tetrahedron formed by the four vertices of the i-th quad patch and A_i is the area of the patch. Obviously $\gamma = 0$ for a completely planar quad mesh. Therefore, the lower the value γ, the better the quality of a PQ mesh in terms of planarity.

4.3 Comparison

We compared our method to the mixed integer quadrangulation (MIQ) method [22] using the above physical and planarity measures. Table 2 shows that the MIQ method produces quad meshes with higher self-supporting performance than that of our method. However, our method outperforms MIQ in terms of planarity. This comparison indicates that a conjugate direction field is important for constructing self-supporting surfaces with planar quadrilateral elements. Without a conjugate direction field, the resulting meshes are far from PQ meshes, and so difficult to construct using glass structures.

We also compared our method with Ma et al.'s method [9] and Liu et al.'s method [15] using the aforementioned physical and geometrical measures. Table 3 shows that Ma et al.'s method produces the most accurate self-supporting surfaces in terms of the three physical measures, but their results are triangular meshes. Both Liu et al.'s method and ours produce planar quadrilateral meshes. To make a fair comparison with their method, we generated PQ meshes with a similar number of quads. We observe that Liu et al.'s results are smoother than ours, since their method minimizes a global smoothness energy. Also, the two methods yield results with comparable planarity. However, our method is more accurate in terms of physical measures. Furthermore, Liu et al.'s method solves a complex non-linear optimization problem, which typically takes 1–3 s, to compute CDF. In contrast, our method computes CDF using local computation, which takes less than 1 s for all test models. Our method is more computationally efficient than theirs.

Table 2 Evaluation of the physical measures θ, σ_D, and σ_S, and planarity measure γ. For each model, we show the MIQ result above and our result below. For each measure, the lower the value, the higher the quality

Model	θ (10^{-3})	σ_D	σ_S	γ (10^{-5})
Eich (Fig. 10, row 1, right)	1.51	0.79%	0.88%	9.64
	4.04	0.91%	1.27%	2.11
Tea (Fig. 10, row 3, left)	2.09	0.46%	0.73%	12.06
	3.38	0.81%	0.79%	2.56
Oct (Fig. 10, row 4, left)	2.41	0.33%	0.53%	2.92
	3.11	0.38%	1.08%	0.45

Table 3 Comparison of physical measures θ, σ_D, and σ_S, geometric measures ϕ and γ, and running time t_{CDF} and t_P. For each model, we show Ma et al.'s result (first row), our result (second row), and Liu et al.'s result (third row). For each quality measure, lower values indicate higher quality. #□ is the number of quadrilateral elements. t_{CDF} is the running time (in second) to calculate the CDF and t_P is the running time (in second) to extract the PQ mesh

Model	θ	σ_D	σ_S	#□	γ (10^{-5})	ϕ	t_{CDF}	t_P
Coin	2.36‰	2.59%	5.34%	—	—	—	—	—
Fig. 10	2.28‰	1.29%	3.54%	1,662	1.05	0.210	0.18	1.07
(row 1, L)	7.19‰	3.46%	3.20%	1,666	0.95	0.207	2.72	1.07
Eich	0.51‰	0.37%	0.41%	—	—	—	—	—
Fig. 10	4.04‰	0.91%	1.27%	2,410	2.88	0.217	0.115	0.43
(row 1, R)	13.3‰	3.42%	4.21%	2,378	2.11	0.228	2.910	0.43
Snow	0.36‰	0.26%	0.29%	—	—	—	—	—
Fig. 10	1.31‰	0.58%	0.70%	2,377	0.67	0.309	0.146	1.82
(row 2, L)	13.9‰	0.63%	1.09%	2,458	0.72	0.302	1.250	1.82
Star	1.96‰	1.28%	2.83%	—	—	—	—	—
Fig. 10	8.46‰	2.27%	2.16%	1,850	1.92	0.349	0.049	0.37
(row 2, R)	11.5‰	1.67%	4.60%	1,832	1.79	0.331	0.510	0.37
Tea	0.46‰	0.29%	0.39%	—	—	—	—	—
Fig. 10	3.38‰	0.81%	0.79%	1,830	2.56	0.199	0.101	0.72
(row 3, L)	11.3‰	1.46%	3.79%	1,865	2.59	0.194	1.180	0.72
Hall	1.48‰	1.68%	1.89%	—	—	—	—	—
Fig. 10	10.5‰	1.83%	2.47%	1,951	4.07	0.275	0.178	2.11
(row 3, R)	33.2‰	4.15%	5.16%	1,953	4.13	0.260	2.720	2.11
Oct	0.48‰	0.16%	0.22%	—	—	—	—	—
Fig. 10	3.11‰	0.38%	1.08%	2,532	0.45	0.190	0.134	0.74
(row 4, L)	27.4‰	2.18%	5.86%	2,516	0.60	0.172	1.520	0.74
Bottle	0.67‰	0.51%	0.62%	—	—	—	—	—
Fig. 10	2.02‰	0.61%	0.82%	3,109	1.34	0.188	0.187	0.68
(row 4, R)	9.89‰	5.11%	7.55%	3,130	0.99	0.178	2.230	0.68
Ring	3.96‰	2.33%	3.50%	—	—	—	—	—
Fig. 11	8.49‰	0.99%	1.61%	2,027	2.37	0.265	0.184	1.02
(row 4, R)	70.8‰	5.65%	2.13%	2,091	3.10	0.238	1.810	1.02

It is worth noting that our method and Liu et al.'s method [15] adopt different strategies to deal with faces with small principal curvature difference. Their method solves a global optimization problem to find a pair of conjugate directions $\boldsymbol{v}_p$ and $\boldsymbol{w}_p$ for each point p that satisfy

$$\kappa_{p,1}(\boldsymbol{v}_p \cdot \boldsymbol{e}_{p,1})(\boldsymbol{w}_p \cdot \boldsymbol{e}_{p,1}) + \kappa_{p,2}(\boldsymbol{v}_p \cdot \boldsymbol{e}_{p,2})(\boldsymbol{w}_p \cdot \boldsymbol{e}_{p,2}) = 0$$

where $\kappa_{p,1}$ and $\kappa_{p,2}$ are the principal curvatures at p, and $\boldsymbol{e}_{p,1}$, $\boldsymbol{e}_{p,2}$ are the corresponding principal directions. Since the principal directions are not accurate in sphere-like regions, the errors may accumulate as the iterative algorithm proceeds. To improve planarity, Liu et al. [15] introduced a planarity term in the objective function. However, the side effect of this term is larger shape approximation error, which in turn may compromise the equilibrium condition of the generated shape. In contrast, our method generates conjugate directions by manipulating the principal directions using a stretch matrix. We calculate smooth conjugate directions without planarity optimization. Therefore, our method is able to well approximate the input geometry and yields better results in terms of both geometric and physical properties.

Figures 10 and 11 show results for height and non-height fields, respectively. We visualize the physical measures using color maps; warmer colors indicate larger errors or distortion. We observe that our results are comparable to Ma et al.'s method [9] that produces triangle meshes and are better than Liu et al.'s method [15] that produces PQ meshes.

We may also qualitatively compare our method to Vouga et al.'s method [7]. Their method allows the user to design self-supporting surfaces using a reference surface. To generate PQ meshes, they solve a complicated non-linear optimization problem that takes equilibrium condition, shape smoothness, and planarity into consideration. Our method is based on local computation of a conjugate direction field and then constructs PQ meshes via mixed integer based parameterization. Our method aims to preserve the geometry of the input shape which is already self-supporting, so does not need a reference surface. Our method (including both CDF computation and global parameterization) is more efficient than theirs.

5 Conclusions and future work

We have developed a simple yet effective method for generating 3D self-supporting surfaces with planar quadrilateral elements. Our method takes a triangular discretization of a 3D self-supporting surface as input and generates a PQ mesh by constructing a conjugate direction field. In contrast to existing methods that compute conjugate direction fields using global optimization, our method is a light-weight approach since it computes a local stretch matrix for each triangular face with small principal curvature difference and turns principal directions into conjugate directions. Computational results

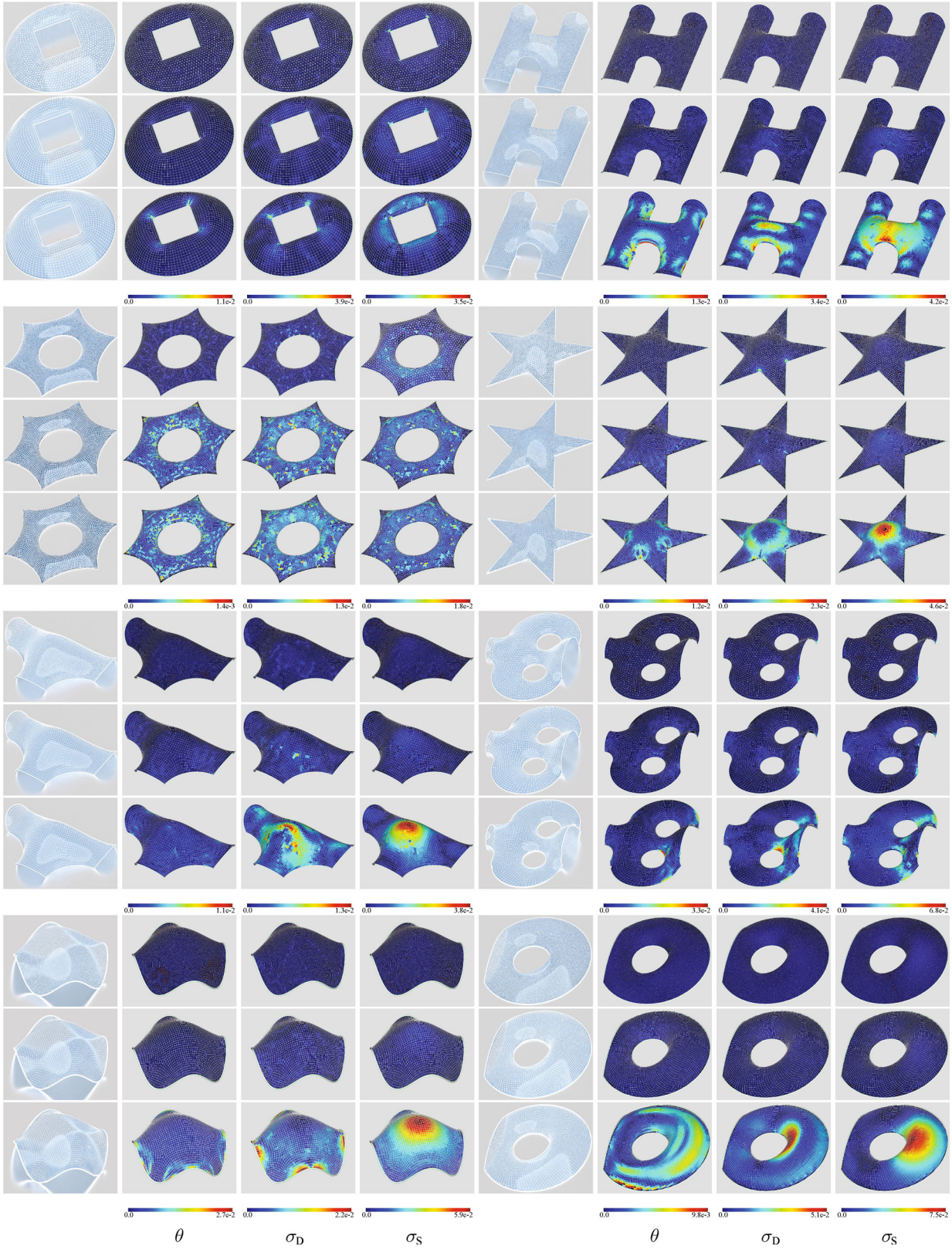


Fig. 10 Results and comparisons. For each model, the three rows show the triangle mesh produced by Ma et al.'s method and the PQ meshes produced by our method and Liu et al.'s method respectively. We visualize the quality of the results using 3 quantitative measures: normal displacement θ, stress tensor error σ_{S}, and differential stress σ_{D}. The lower the measures, the higher the quality of the resulting PQ meshes.

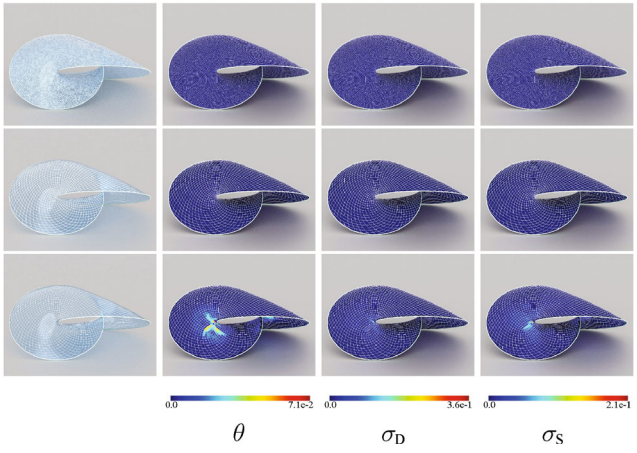


Fig. 11 Non-height fields. Row 1: the triangle meshes generated by Ma et al.'s method [9]; Rows 2 and 3: the PQ meshes produced by our method and Liu et al.'s method [15], respectively.

show that the generated PQ meshes approximate the input shape very well, thereby satisfying the balance constraints. Comparisons with existing methods demonstrate the effectiveness of our method.

There are a few interesting directions for future work. First, our method requires the input shape to be self-supporting. From the application point of view, it is highly desirable to either construct PQ meshes directly from user-specified boundary curves or to take a reference surface as input and then optimize its geometry to fulfil the physical requirements of being self-supported. To make the method more flexible, we will also take the user-specified soft and/or hard directional constraints into consideration.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive comments. This work was partially supported by National Natural Science Foundation of China (62172257, 61802228), Singapore Ministry of Education (T2EP20220-0014), and the RIE2020 Industry Alignment Fund–Industry Collaboration Projects (IAF–ICP) Funding Initiative, as well as cash and in-kind contribution from the industrial partner, Rolls-Royce.

Declaration of competing interest

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

References

[1] Green, A. E.; Zerna, W. *Theoretical Elasticity*. Courier Corporation, 1992.

[2] Truesdell, C. Book review: *Finite Deformation of an Elastic Solid*. *Bulletin of the American Mathematical Society* Vol. 58, No. 5, 577–580, 1952.

[3] Van der Heijden, A. M. A. *W. T. Koiter's Elastic Stability of Solids and Structures*. Cambridge University Press, 2008.

[4] Pottmann, H.; Eigensatz, M.; Vaxman, A.; Wallner, J. Architectural geometry. *Computers & Graphics* Vol. 47, 145–164, 2015.

[5] De Goes, F.; Alliez, P.; Owhadi, H.; Desbrun, M. On the equilibrium of simplicial masonry structures. *ACM Transactions on Graphics* Vol. 32, No. 4, Article No. 93, 2013.

[6] Liu, Y.; Pan, H.; Snyder, J.; Wang, W. P.; Guo, B. N. Computing self-supporting surfaces by regular triangulation. *ACM Transactions on Graphics* Vol. 32, No. 4, Artide No. 92, 2013.

[7] Vouga, E.; Höbinger, M.; Wallner, J.; Pottmann, H. Design of self-supporting surfaces. *ACM Transactions on Graphics* Vol. 31, No. 4, Article No. 87, 2012.

[8] Block, P.; Ochsendorf, J. Thrust network analysis: A new methodology for three-dimensional equilibrium. *Journal of the International Association for Shell and Spatial Structures* Vol. 48, No. 3, 167–173, 2007.

[9] Ma, L.; He, Y.; Sun, Q.; Zhou, Y. F.; Zhang, C. M.; Wang, W. P. Constructing 3D self-supporting surfaces with isotropic stress using 4D minimal hypersurfaces of revolution. *ACM Transactions on Graphics* Vol. 38, No. 5, Article No. 144, 2019.

[10] Glymph, J.; Shelden, D.; Ceccato, C.; Mussel, J.; Schober, H. A parametric strategy for free-form glass structures using quadrilateral planar facets. *Automation in Construction* Vol. 13, No. 2, 187–202, 2004.

[11] Liu, Y.; Pottmann, H.; Wallner, J.; Yang, Y. L.; Wang, W. P. Geometric modeling with conical meshes and developable surfaces. *ACM Transactions on Graphics* Vol. 25, No. 3, 681–689, 2006.

[12] Zadravec, M.; Schiftner, A.; Wallner, J. Designing quad-dominant meshes with planar faces. *Computer Graphics Forum* Vol. 29, No. 5, 1671–1679, 2010.

[13] Bobenko, A. I.; Sullivan, J. M.; Schröder, P.; Ziegler, G. M. *Discrete Differential Geometry*. Basel: Birkhäuser Basel, 2008.

[14] Sauer, R. *Differenzengeometrie*. Berlin, Heidelberg: Springer, 1970.

[15] Liu, Y.; Xu, W. W.; Wang, J.; Zhu, L. F.; Guo, B. N.; Chen, F. L.; Wang, G. General planar quadrilateral mesh design using conjugate direction field. *ACM Transactions on Graphics* Vol. 30, No. 6, 1–10, 2011.

[16] Eigensatz, M.; Kilian, M.; Schiftner, A.; Mitra, N. J.; Pottmann, H.; Pauly, M. Paneling architectural freeform surfaces. In: Proceedings of the ACM SIGGRAPH 2010 Papers, Article No. 45, 2010.

[17] Block, P.; Lachauer, L. Closest-fit, compression-only solutions for freeform shells. In: Proceedings of the IABSE-IASS Symposium, 2011.

[18] Vouga, E.; Höbinger, M.; Wallner, J.; Pottmann, H. Design of self-supporting surfaces. *ACM Transactions on Graphics* Vol. 31, No. 4, Article No. 87, 2012.

[19] Miki, M.; Igarashi, T.; Block, P. Parametric self-supporting surfaces via direct computation of airy stress functions. *ACM Transactions on Graphics* Vol. 34, No. 4, Article No. 89, 2015.

[20] Meyer, M.; Desbrun, M.; Schröder, P.; Barr, A. H. Discrete differential-geometry operators for triangulated 2-manifolds. In: *Visualization and Mathematics III. Mathematics and Visualization*. Hege, H. C.; Polthier, K. Eds. Springer Berlin Heidelberg, 35–57, 2003.

[21] Lu, C. L.; Cao, Y.; Mumford, D. Surface evolution under curvature flows. *Journal of Visual Communication and Image Representation* Vol. 13, Nos. 1–2, 65–81, 2002.

[22] Bommes, D.; Zimmer, H.; Kobbelt, L. Mixed-integer quadrangulation. *ACM Transactions on Graphics* Vol. 28, No. 3, Article No. 77, 2009.

[23] Rusinkiewicz, S. Estimating curvatures and their derivatives on triangle meshes. In: Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 486–493, 2004.

[24] Liu, D. C.; Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming* Vol. 45, Nos. 1–3, 503–528, 1989.

Long Ma is an associate researcher at the School of Software, Shandong University. He holds his M.S. degree in mathematics and Ph.D. degree in computer science from Shandong University. His research focuses on computer graphics, geometry modeling, and mechanical simulation.

Sidan Yao is a Ph.D. student in the School of Computer Science and Engineering, Nanyang Technological University. She received her bachelor degree from the School of Computer Software, Tianjin University. Her research focuses on architectural geometry and differential geometry.

Jianmin Zheng is a full professor in the School of Computer Engineering at Nanyang Technological University. He received his bachelor and Ph.D. degrees from Zhejiang University. His research interests include computer aided geometric design, computer graphics, geometric modeling, CAD, visualization, and interactive digital media.

Yang Liu is a principal researcher in the Internet Graphics Group at Microsoft Research Asia which he joined in 2010. He received his Ph.D. degree in computer science from the University of Hong Kong in 2008, and his master and bachelor degrees in computational mathematics from the University of Science and Technology of China, in 2003 and 2000 respectively. He worked in the Alice group at INRIA/LORIA as a post-doctoral researcher from 2008. His research interests span geometric modeling and optimization, mesh generation, computer-aided geometric design, and architectural geometry.

Yuanfeng Zhou is a professor in the School of Software, Shandong University. He received his B.S., M.S., and Ph.D. degrees from Shandong University. His research focuses on intelligent graphics and image processing, geometric modeling and optimization, computational medicine, and virtual reality.

Shi-Qing Xin is an associate professor at the School of Computer Science and Technology, Shandong University. He received his Ph.D. degree in applied mathematics from Zhejiang University. His research focuses on geometric calculation, geometric modeling, and scene understanding.

Ying He is an associate professor in the School of Computer Engineering, Nanyang Technological University. He received his B.S. and M.S. degrees in electrical engineering from Tsinghua University, and his Ph.D. degree in computer science from Stony Brook University. His research interests fall into the general areas of visual computing and he is particularly interested in problems which require geometric analysis and computation.