Computing IGA-suitable planar parameterizations by PolySquare-enhanced domain partition

Shiwei Xiao\textsuperscript{a}, Hongmei Kang\textsuperscript{b,\*}, Xiao-Ming Fu\textsuperscript{a,\*}, Falai Chen\textsuperscript{a}

\textsuperscript{a} School of Mathematical Sciences, University of Science and Technology of China, China
\textsuperscript{b} School of Mathematical Sciences, Soochow University, China

\textbf{A R T I C L E  I N F O}

\textbf{Article history:}
Available online 27 March 2018

\textbf{Keywords:}
Isogeometric analysis
Analysis-suitable planar parameterization
Injectivity
Domain partition
PolySquare
Preconditioned Accelerated Proximal Gradient

\textbf{A B S T R A C T}

We propose an algorithm for computing injective IGA-suitable planar parameterizations with as uniform and orthogonal as possible iso-parametric structure. Central to this approach is a PolySquare-enhanced domain partition procedure that decomposes the input complex planar domain into coarse and square-like quad patches. First, we triangulate the input planar domain that may be multiply connected, and deform it to be a PolySquare-like structure by optimizing a boundary alignment energy. Then, the PolySquare-like structure is pixilated to make the input domain be a quadrilateral mesh that is decimated to generate a coarse patch layout, where each patch is an approximately squared quadrilateral. Finally, the sparse patches are subdivided to represent the input domain and produce the resulting partition. We parameterize each patch with continuous constraints to find the parameterization of the input domain. Compared with existing IGA-suitable planar parameterization methods, our method produces better parameterizations and fewer patches than the ones that also use domain partition strategy. We demonstrate the superiority of our method over various complex domains, including an example containing 30 holes.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

Isogeometric analysis (IGA) was originally introduced by Hughes et al. (2005), Cottrell et al. (2009) as a new approach for solving Partial Differential Equations (PDEs) by using the smooth spline basis that defines the geometry as the basis for analysis. IGA has stimulated the development of many fields such as locally refinable splines (Li et al., 2016) and the application in a variety of PDEs of engineering interest, such as linear elasticity (Auricchio et al., 2007), structural vibrations (Cottrell et al., 2006), phase transition phenomena (Gómez et al., 2008), shape optimization (Wall et al., 2008; Qian, 2010; Nguyen et al., 2012) and the references in the survey paper (Nguyen et al., 2015; da Veiga et al., 2014). The aim of IGA is to improve the interoperability between Computer Aided Design (CAD) and Computer Aided Engineering (CAE). However, CAD models are often given only by the boundaries. A fundamental and crucial problem in IGA is to compute a valid parametric spline representation of the computational domain from the given CAD model. This problem is called parameterization. In this paper, we consider the planar parameterizations of 2D computational domains.

A good planar parameterization can generate high accuracy and efficiency in the subsequent steps of IGA (Cohen et al., 2010; Xu et al., 2013a; Pilgerstorfer and Jüttler, 2014). Here the term good means that the parameterization should satisfy

\* Corresponding authors.
E-mail addresses: khm@suda.edu.cn (H. Kang), fuxm@ustc.edu.cn (X.-M. Fu).
the following properties: (1) it works for the domains with high genus and complex boundary; (2) it is injective (that is also called foldover-free, inversion-free), i.e., the determinant of the Jacobian of the parametrization is always positive; (3) its iso-parametric structure is nearly orthogonal and has as uniform as possible elements; (4) it has few control points, i.e., the matrix assembly is efficient.

Given the 2D boundary curve information with spline representations, many approaches have been proposed to solve the planar parameterization problem in the past decade. Some methods compute the high-quality parameterizations for computational domain bounded by four B-spline curves, such as the variational harmonic method (Xu et al., 2013c) and the Teichmüller mapping method (Nian and Chen, 2016). The computational domains with simple boundaries are considered in these methods, otherwise their nonlinear and non-convex formulations for computing parameterizations cause high distortions, even foldovers. Moreover, they can not work for the multiply connected domains that have holes. To handle these high genus domains, the partition-driven techniques were presented to decompose the input domains into coarse quad patches, like the skeleton-based decomposition method (Xu et al., 2015), multi-patch parametrization method (Buchegger and Jüttler, 2017), or approximate convex decomposition method (Xu et al., 2018). This category computes the parameterizations via the multi-patch representation, which glues patches together to define the results. However, these methods usually generate too many patches for the complex domains to be efficient for the subsequent matrix assembly. To support complex and high-genus domains, we also focus on the partition-based strategy that generates high-quality quad layouts for complex domains.

We present a new domain partition method, called PolySquare-enhanced Domain Partition (PeDP), to compute high-quality IGA-suitable planar parameterizations for the planar domains with holes and complex boundary curves. Our method generating a coarse quad layout and computing parameterizations consists of three steps (that are listed as follows): (1) PolySquare construction, (2) domain partition, and (3) parameterization computation.

1. Given the boundary curves of an input domain, we first triangulate it and employ a foldover-free and boundary-aligned deformation to align the normals of boundary edges to axis directions, generating a PolySquare-like structure that is pixelated to generate a PolySquare. The use of the PolySquare structure can help handle the complex and high genus domains well.

2. Then, we project the pixelated PolySquare back to the input domain to generate a quad mesh of the input domain. Because of no interior singular vertices, we utilize the poly-chord collapse operation (Daniels et al., 2008) based on a boundary-preserving metric to simplify the quad mesh to a coarse quad patch layout. This boundary-preserving simplification generates a small number of patches, which are square-like. We subdivide and optimize this layout to generate the final quad partition which is represented as a dense quad mesh.

3. Finally, we compute the tensor-product B-splines parameterization for each patch. Two adjacent patches meet with $C^0$ continuity. A tensor-product B-spline surface is first fitted to follow the dense quad mesh edge by the least square method, and then improved by optimizing a foldover-penalized energy. Because the final partition has almost uniform and orthogonal quadrilaterals, our parameterization computation is very easy to be injective and with high quality.

The key technique to make our method very effective is a use of the preconditioned accelerated proximal gradient method that can optimize the involved nonlinear and non-convex energies effectively and efficiently. Our method is able to produce the high-quality IGA-suitable planar parameterization for the complex domain with high genus. Compared with existing IGA-suitable planar parameterization methods, our method possesses higher parameterization quality and generates fewer patches. We demonstrate the practical robustness of our method on various complex domains and show the superiority to existing methods in terms of both parameterization quality and patch number.

The contribution of our method is summarized as:

- We present an efficient and practically robust method to compute high-quality IGA-suitable planar parameterizations via the PolySquare-enhanced domain partition strategy.
- Our method could process the various domains with high genus and complex boundaries without any extra inputs, e.g., cross fields. The generated patches are all square-like so that the subsequent parameterization computation process is very easy to satisfy the requirements of good parameterizations.

2. Related work

**IGA-suitable parameterizations** The parameterization methods for IGA can be grouped into four types (Xu et al., 2018): (1) IGA-aware optimal parameterizations (Cohen et al., 2010; Xu et al., 2013a; Pilgerstorfer and Jüttler, 2014); (2) volumetric spline parameterizations (Martin et al., 2009; Escobar et al., 2011; Zhang et al., 2012; Chen et al., 2017); (3) IGA-suitable planar parameterizations (Xu et al., 2015; Nian and Chen, 2016; Xu et al., 2018; Buchegger and Jüttler, 2017); (4) IGA-suitable volumetric parameterizations (Xu et al., 2013b, 2013c; Wang and Qian, 2014; Zhang et al., 2013). Here we review the latter two categories that are mostly related to ours.

Xu et al. (2011) constructed the injective planar parameterizations by a constrained optimization method. This nonlinear formulation is so hard to solve that the results may contain foldovers. The Teichmüller mapping is utilized in Nian and Chen (2016) to generate a bijective high-quality planar parameterization from four specified boundary curves. However this
method is not suitable for multiply connected input domains. To handle the complex domains with high genus, the domain partition method is proposed. Xu et al. (2015) used the skeleton of the input domain to guide the domain decomposition and compute the planar parameterization with $C^0$-continuity between patches. A systematic method is proposed (Buchegger and Jüttler, 2017) to explore the different possible parameterizations of a planar domain by collections of quadrilateral patches. An approximate convex decomposition and pattern based quad mesh generation is used to partition the input domain (Xu et al., 2018). The latter two methods usually produce much more patches, causing high cost of subsequent matrix assembly. However, our method uses the quad mesh simplification to generate the decomposition by a boundary-preserving metric that results in square-like patches and small number of patches.

Aigner et al. (2009) proposed a method for swept volumes which cover many free-form shapes in CAD system like blades or propellers. The nonlinear optimization framework (Xu et al., 2011) is extended for volume parameterizations (Xu et al., 2013b, 2014). A conformal solid T-spline from the boundary T-spline representation for the volume parameterization is constructed (Zhang et al., 2013). Given six boundary B-spline surfaces, the valid trivariate B-spline solids are computed by an efficient method that combines divide-and-conquer, constraint aggregation and the hierarchical optimization techniques (Wang and Qian, 2014). A reuse method is proposed by Xu et al. (2017) for a framework of computation reuse in IGA on a set of 3D models with similar semantic features. In this paper, we focus on the planar case.

Some methods focus on the parameterizations with non-standard B-splines, such as T-splines (Zhang et al., 2013), THB-splines (Falini et al., 2015), and Powell–Sabin splines (Speleers and Manni, 2015). In contrast, we focus on the standard B-spline that is used to define the parameterization of each patch with continuous constraints on the neighbor patches.

Automatic generation of coarse quad patches We decompose the input domain into multiple quad patches that could be easily parameterized by B-Splines. There are many methods focusing on generating quad meshes (Bommes et al., 2013b) or quad patches (Campen, 2017). The QuadCover (Kälberer et al., 2007) and MIQ (Bommes et al., 2009) methods are the pioneers to generate high quality quad meshes. But their results often do not possess coarse layout. To resolve this problem, some post-processing techniques are proposed, such as the helix removal (Bommes et al., 2011), domain simplicity (Tarini et al., 2011), aligned global parameterizations (Campen and Kobelt, 2014), perfect matching (Razafindrazaka et al., 2015; Razafindrazaka and Polthier, 2017). Another way is to increase the target quad edge lengths by reliable quad meshing algorithms (Bommes et al., 2013a; Campen et al., 2015). Some other methods generate the coarse quad layout by tracing the fields (Campen et al., 2012; Pietroni et al., 2016). On the one hand, these methods usually generate narrow rectangles (see the Fig. 11 in Bommes et al. (2011), Fig. 8 in Bommes et al. (2013a), and Fig. 11 in Razafindrazaka et al. (2015)). They are not suitable for IGA, because narrow rectangles result in large condition number of the Jacobian matrix of the parameterizations. On the other hand, the high quality of the quad patch layout is highly dependent on the situations (number, positions, and indices) of the singularities of the input cross field (see the Fig. 13 in Pietroni et al. (2016) and Fig. 8 in Bommes et al. (2013a)). Thus, the input cross field should be carefully tuned; otherwise, the generated patch layout may be not usable. However, our method is based on the simplification of a special type of quad meshes that are generated by PolySquare. They do not contain any interior singularities, so we only use poly-chord collapse operation (Daniels et al., 2008) to generate the coarse quad patches. Given a quad mesh, some methods also adopt the decimation technique to generate coarse quad layout by adjusting its geometry and connectivity using some well-defined local operations (Daniels et al., 2008; Tarini et al., 2010; Daniels et al., 2009a, 2009b). However, many interior singularities would still exist and the quad patches are often not uniform enough. Our employed poly-chord collapse operation does not introduce any other singularities, and our effective quad optimization results in as uniform as possible patches.

3. PolySquare-enhanced domain partition

3.1. PolySquare construction

Generally, CAD models are only given by boundaries. The planar parameterization problem in IGA is to find a global mapping $B(u, v) : \Omega_0 = [a, b] \times [c, d] \mapsto \Omega$ with the given boundaries, as illustrated in the right inset. However, for domains with complex boundaries or high genus, a global mapping is difficult to obtain. Domain partition is an advisable choice. The original computational domain is partitioned into several patches by some methods such that each patch is easier to be parameterized and the patch number is small. Adjacent patches are stitched with $C^0$ continuity. Our method is also based on the domain partition and consists of three steps: (i) PolySquare construction, (ii) domain partition, and (iii) parameterization computation. Its workflow is shown in Fig. 1.

We combine the fast normal-driven deformation (Fu et al., 2016) and the pixelation (i.e., 2D version of voxelization) (Yu et al., 2014) to generate a PolySquare of the input domain efficiently and effectively. This treatment makes our method suitable for the domains with complex boundary curves and high genus.
Triangulation of the input domain Our construction of PolySquare is based on triangular meshes that are an approximation of the input domains. The triangulation of the input domain consists of three steps: (1) uniformly sample the given boundary curves to generate the polyline; (2) compute the constrained Delaunay triangulation (CDT) of these polylines; (3) isoparametrically remesh the CDT by Fu et al. (2014) that uses the identity matrix as the Riemann metric. The Fig. 1(a) shows an example. The obtained triangular mesh is denoted as $\mathcal{M}$ containing $N_f$ facets $\{f_1, \ldots, f_{N_f}\}$ and $N_v$ vertices $\{v_1, \ldots, v_{N_v}\}$. Let $n$ boundary edges of $\mathcal{M}$ be $\{b_1, \ldots, b_n\}$, their corresponding normals be $\{n_1, \ldots, n_n\}$ and their midpoints be $\{c_1, \ldots, c_n\}$. In 2D, the normal $n_k$ of an edge $e_k := \overrightarrow{v_i v_j}$ is defined as $n_k = b_k^\perp/\|b_k\|$ where $b_k = v_j - v_i$ and $\perp$ denotes a counterclockwise rotation by $90^\circ$.

3.1.1. Boundary-aligned deformation We deform $\mathcal{M}$ to be a PolySquare-like structure whose boundary normals are almost axis-aligned similar to Fu et al. (2016). The deformed mesh is denoted as $\mathcal{M}_d$ and its vertices are $\{\hat{v}_1, \ldots, \hat{v}_{N_v}\}$. The deformation is expressed with a piecewise linear transformation $\Phi$. On any triangle $f_k := \Delta v_p v_q v_r$ of the mesh $\mathcal{M}$, the affine transformation $\Phi$ has a linear form $\Phi(x) = J_k x + t_k$, where $J_k$ is a $2 \times 2$ matrix and also the Jacobian of $\Phi$ for $\forall x \in f_k$. $J_k$ is represented by the deformed vertices linearly:

$$J_k = [\hat{v}_q - \hat{v}_p, \hat{v}_r - \hat{v}_p] \cdot [v_q - v_p, v_r - v_p]^{-1}. \tag{1}$$

Deformation energy Our mesh deformation is driven by a combined energy:

$$E_{\text{deform}} := E_{\text{iso}} + \lambda E_{\text{align}}, \tag{2}$$

where $\lambda$ is the positive factor balancing the isometric distortion energy $E_{\text{iso}}$ and the normal alignment energy $E_{\text{align}}$. Here we use the isometric AMIPS energy (Fu et al., 2015) to define $E_{\text{iso}}$:

$$\begin{align*}
\delta_k^{\text{conf}} &= \frac{\text{trace} J_k^T J_k}{2 \det J_k}, \\
\delta_k^{\text{area}} &= \frac{1}{2} \left( \det J_k + (\det J_k)^{-1} \right), \\
E_{\text{iso}} &= \begin{cases} 
+\infty, & \exists f_k, \det J_k \leq 0; \\
\frac{1}{N_f} \sum_{k=1}^{N_f} \exp \left( \frac{1}{2} \left( \delta_k^{\text{conf}} + \delta_k^{\text{area}} \right) \right), & \text{otherwise}.
\end{cases}
\end{align*} \tag{3}$$

Before deformation, i.e., $\mathcal{M}_d = \mathcal{M}$, $E_{\text{iso}}$ is minimum. This energy could prevent any degenerate triangle and penalize the maximal distortion. Minimizing this energy results in angle-preserving and length-preserving deformation.

The alignment energy $E_{\text{align}}$ is defined as follows.

$$E_{\text{align}} = \frac{1}{n} \sum_{i=1}^{n} (b_i \cdot \mathbb{P}(G_\sigma(n_i)))^2 \tag{6}$$

where $G_\sigma(\cdot)$ is the Gaussian function:

$$G_\sigma(n_i) = \sum_{b_j \in \mathcal{C}_i} ||b_j|| \exp \left( \frac{-\text{dist}(c_i, c_j)^2}{2\sigma^2} \right) \cdot n_j, \tag{7}$$
and the operator \( P(\cdot) \) maps a two-dimensional vector to its closest axis direction. The \( \text{dist}(c_i, c_j) \) is the geodesic distance between \( c_i \) and \( c_j \) on the boundary of \( \mathcal{M} \). Instead of directly using \( P(\cdot) \), we follow (Fu et al., 2016) to combine neighbor normals to deform the mesh to avoid the zigzags. Thus, different Gaussian kernel sizes \( \sigma \) result in different coarseness. Larger \( \sigma \) indicates more coarseness, i.e., fewer poly segments.

**Pre-rotation** If the boundary normals deviate from the targeted normals far away, the normal-aligned energy may be very large. To reduce the normal-aligned energy, we perform a global pre-rotation so that the boundary normals become as much closer to the targeted normals as possible. The rotating angle is obtained by optimizing the following problem:

\[
E_\theta = \sum_{i=1}^{n} (\mathbf{R}(\theta) \cdot \mathbf{b}_i \cdot P(\mathbf{n}_i))^2,
\]

where \( \theta \) is the rotation angle, and \( \mathbf{R}(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \) is the rotation matrix. This pre-rotation has no effect on \( E_{\text{iso}} \).

**Deformation procedure** Our method for generating PolySquare-like structure is as follows.

1. Pre-rotate the mesh by a \( \theta \) that is computed via optimizing \( E_\theta \).
2. Optimize \( E_{\text{deform}} \) to deform the mesh.
3. If the maximum change of vertex positions in Step 2 is less than \( \varepsilon_d \) or the maximum iteration is reached, stop the algorithm. Otherwise, increase \( \lambda \) by a constant multiply (e.g., 10) and go to Step 1.

The large \( \varepsilon_d \) indicates the mesh still exhibits large deformation and does not converge. And the very small one requires the deformation to take much more time to converge. During our experiments, we take a tradeoff and set \( \varepsilon_d = 10^{-4} \). In our experiments, we set the initial \( \lambda \) to be 1, and the maximum iteration number to be 6. We show a PolySquare-like structure in Fig. 1 (b).

### 3.1.2. PolySquare generation

**Pixelation** We extend the method (Yu et al., 2014) to generate the PolySquare. By embedding the normal-aligned mesh \( \mathcal{M} \) into a planar grid, we obtain an initial polysquare \( \mathcal{S} \) from the pixels which are inside or partially inside \( \mathcal{M} \). The size \( \eta \) of the pixel controls the coarseness of \( \mathcal{S} \). Note that an inappropriate \( \eta \) leads to loss of topology. But, it can be theoretically estimated through a Morse analysis as mentioned in Yu et al. (2014). Usually, we set \( \eta \) to be half of the average length of mesh edges if this value keeps the topology of the input. Otherwise, we use the estimated value via the Morse analysis.

**Morphological optimization** The initial \( \mathcal{S} \) may have many corners (i.e., \( \mathcal{S} \) is very complex) or large deviation from \( \mathcal{M} \). Fewer corners cause larger deviation. Thus, to achieve a tradeoff between simplicity and small deviation, we further optimize \( \mathcal{S} \) using the homotopic morphological operations (Yu et al., 2014). The optimization is driven by a combined metric:

\[
E_{\text{mor}} := E_{\text{simp}} + \alpha E_{\text{dev}},
\]

where \( E_{\text{simp}} \) is the simplicity metric defined as the number of corners, \( E_{\text{dev}} \) is the deviation metric, and \( \alpha \) is the weighing factor and we set \( \alpha = 20 \) in our experiments. We define \( E_{\text{dev}} \) as:

\[
E_{\text{dev}} = \sum_{\mathbf{v} \in P} |D(\mathbf{v})|,
\]

where \( P \) is the boundary vertex set of \( \mathcal{S} \), and \( D(\mathbf{v}) \) is the minimum distance from each boundary vertex \( \mathbf{v} \) of \( \mathcal{S} \) to \( \mathcal{M} \).

We use the standard morphological operations (opening and closing) in the image processing to minimize the energy. The openings remove thin glitches and the closings fill small holes. We perform the homotopic opening or homotopic closing operations in a moving window to reduce the energy. For the homotopic operation each time, we have three choices: keeping, opening, and closing, where keeping means we do neither homotopic opening nor closing. If a homotopic opening (or closing) operation reduces \( E_{\text{mor}} \) most, then we choose to apply this operation. If neither reduces the energy \( E_{\text{mor}} \) more than a threshold value, even increases the energy, or changes the topology of \( \mathcal{M} \), we do keeping. Similar to Yu et al. (2014), we start from a big window size and gradually reduce it. For each window size, we keep moving the window along all boundary pixels on \( \mathcal{S} \) until the energy \( E_{\text{mor}} \) does not decrease. The final optimized PolySquare is denoted as \( \mathcal{S}^* \) (Fig. 1 (c)).

### 3.2. Domain partition

#### 3.2.1. Quad meshing

**Initial back projection** To generate the quad mesh of the input domain, we first project \( \mathcal{S}^* \) back to the normal-aligned mesh \( \mathcal{M} \). To this end, we first perform the projection on the boundary points that are mapped onto the closest points of \( \mathcal{M} \). In other words, for any boundary vertex of \( \mathcal{S}^* \), we move it to the position of its closest point on the boundary of \( \mathcal{M} \).
Foldover elimination. After this initial boundary projection, the mesh $S^*$ usually contains many flipped elements. By splitting each quad into four triangles virtually (see the right inset) to compose a triangular mesh $T$ that has $N_f^j$ triangles $\{f_1, ..., f_{N_f^j}\}$, we optimize the following foldover-penalized energy (Escobar et al., 2003) to eliminate the foldovers.

$$E_{\text{fold}} = \sum_{k=1}^{N_f^j} \left( \frac{||J_k||_F^2}{\det J_k + \sqrt{\det J_k^2 + \epsilon}} \right)$$

where $|| \cdot ||_F$ denotes the Frobenius norm and $\epsilon$ is a small positive number that is determined using the method of (Escobar et al., 2003). When some $\det J_k < 0$, $E_{\text{fold}}$ is very large so as to suppress the foldovers. The positions of the boundary and inner vertices of $S^*$ are optimized at the same time. And the boundary vertices are projected on the boundary of $\hat{\mathcal{M}}$ in each iteration.

Quad optimization. Once there are no flipped quads, we go to optimize the quad mesh to make each quad square-like. Similar to the foldover elimination, we virtually split each quad into four triangles and treat an isosceles right triangle as the source triangle. To distribute the vertices uniformly, thereby making the quads look like squares, we optimize the conformal AMIPS energy (Fu et al., 2015):

$$E_{\text{conf}} = \begin{cases} +\infty, & \exists f_k, \det J_k \leq 0; \\ 1 \sum_{k=1}^{N_f^j} \exp(\delta_k^\text{conf}), & \text{otherwise.} \end{cases}$$

Here the boundary and interior vertices are optimized simultaneously.

Re-projection and optimization. After the above steps, $S^*$ is projected onto $\hat{\mathcal{M}}$ well. To seek for the projection onto $\mathcal{M}$, we note that $\hat{\mathcal{M}}$ has the one-to-one correspondence with $\mathcal{M}$. For this reason, we can implement the barycentric projection of $S^*$ onto $\mathcal{M}$ as follows. We first calculate the barycentric coordinate of each vertex $v$ of $S^*$ w.r.t. the triangle $t$ of $\hat{\mathcal{M}}$ which contains $v$. By using the same barycentric coordinate on the corresponding triangle of $t$ in $\mathcal{M}$, we can interpolate a position of $v$ on $\mathcal{M}$. We treat this position as the initial projection of $v$ on $\mathcal{M}$. After finishing the barycentric projection of all vertices, we further repeat the foldover elimination and quad optimization to improve the quality of the projection. The difference during optimization is that the boundary vertices are projected on the boundary of $\mathcal{M}$, rather than $\hat{\mathcal{M}}$ in each iteration. This optimization finally generates a quality quad mesh (denoted as $Q$) whose boundary approximates the input domain boundary. Fig. 1 (d) shows a generated quad mesh. We show the back projection and quad optimization process in Fig. 2.

3.2.2. Boundary-preserving decimation

To obtain the sparse patch expression of the input domain, we use a boundary-preserving simplification of the quad mesh $Q$. Because of no singular vertices, we only use the poly-chord (see the chord formed by the red quads in the right inset) collapse operation (Daniels et al., 2008). Collapse of a poly-chord from the quad mesh means deleting all quadrilaterals through which it passes by merging the vertices of each edge contained in the selected poly-chord.

Boundary-preserving metric. On a quad mesh, there are usually more than one poly-chord sheet. One of our task is to decide which poly-chord to be collapsed each time. The prioritization of the operations is achieved by following the metric defined on a poly-chord $C$.
\[
\rho(C) = \beta \left(1 - e^{-\rho_q(C)}\right) + (1 - \beta) \left(1 - e^{-\rho_d(C)}\right),
\]
where \(\rho_q(C)\) is the geometric loss and \(\rho_d\) evaluates the area distortion (Daniels et al., 2008). \(\rho_q(C)\) is the larger 2D QEM metric of merging vertices between the first and the last boundary edges contained in the poly-chord \(C\). Similar to Garland and Heckbert (1997), we could easily define the 2D QEM matrices that are used to compute the 2D QEM metric. \(\rho_d(C)\) is defined as the length of the longest edge collapse due to the collapse of the poly-chord \(C\), similar to Daniels et al. (2008). 

**Termination** Each time we choose the poly-chord with least error metric to be collapsed. If the Hausdorff distance (scaled by the diagonal length of the bounding box of the input meshes) between the simplified mesh \(\tilde{Q}\) and \(Q\) is greater than a threshold \(D_H\), we stop the collapse. Larger \(D_H\) causes fewer patches, but larger geometric deviation and low-quality quad meshes, thereby low-quality parameterizations. Smaller \(D_H\) causes more patches, thereby high-quality parameterizations. We choose a tradeoff and set \(D_H\) as 0.1 w.r.t the bounding box diagonal for all examples. We show a simplified quad mesh in Fig. 1 (e). We demonstrate the simplification process in Fig. 3.

### 3.2.3. Layout subdivision and optimization

After simplification, we obtain the simplified quad mesh \(\tilde{Q}\) and the sparse quad patch layout \(\mathcal{P}\) of the original domain is defined as \(\tilde{Q}\). To generate a better domain partition with as uniform and orthogonal as possible patches, we apply subdivision (see the right inset) and optimization using the following method.

1. Set \(l = 1\) and \(\bar{Q}_1 = \tilde{Q}\).
2. Subdivide \(\tilde{Q}\), to generate a new quad mesh, and we project it onto \(\mathcal{M}\), then perform foldover elimination and further optimize it by minimizing \(E_{\text{cont}}\) to generate \(\bar{Q}_{l+1}\).
3. \(l := l + 1\). If the Hausdorff distance between \(\bar{Q}_l\) and \(Q\) is less than \(\varepsilon_{Q}\), stop the algorithm. Otherwise go to Step 2.

When \(\bar{Q}_l\) is close to \(Q\), we stop the algorithm. Large \(\varepsilon_Q\) leads to great deviation between \(\bar{Q}_l\) and \(Q\), indicating that the resulting \(\bar{Q}_l\) cannot represent the input domain. Small \(\varepsilon_Q\) requires more iterations to terminate the algorithm. To achieve a good balance between efficiency and quality, we set \(\varepsilon_Q = 10^{-3}\) for all examples.

Finally, the original domain is partitioned using the topology of \(\mathcal{P}\) (see the red lines in Fig. 1 (f) and 7), but with the corresponding vertices of \(\bar{Q}_l\) to define the positions of the patch edges. Each quad element of this dense quad mesh is square-like. Clearly, all the edges of \(\bar{Q}_l\) correspond to the layout \(\mathcal{P}\) before subdivision. At a time, \(\bar{Q}_l\) is subdivided, the edges of \(\bar{Q}_l\) that correspond to the layout \(\mathcal{P}\) are partitioned into two edges of \(\bar{Q}_{l+1}\) that also correspond to \(\mathcal{P}\). Except these edges, other newly produced edges do not correspond to \(\mathcal{P}\). Thus, the edges that correspond to the layout \(\mathcal{P}\) can be determined and recorded during every subdivision. When the layout subdivision finishes, all the edges that correspond to the layout \(\mathcal{P}\) are already recorded. And the red contours in all figures are these edges corresponding to the layout \(\mathcal{P}\). Fig. 4 demonstrates the subdivision process.

### 3.3. IGA-suitable planar parameterization computation

From the above domain partition step, not only patches layout are produced but also discrete points in the tensor-product structure on each patch are supplied, this greatly simplifies the parametrization process for each patch. Since the output discrete points are optimized with low distortion, injectivity and orthogonality, each patch is parameterized by a least square approximation with splines to follow the given discrete points as possible. Adjacent patches are stitched with \(C^0\) continuity.

In order to guarantee \(C^0\) continuity, we firstly fit the segmented curves and boundaries of the computational domain with the B-spline curves by the least square method. Then each patch is parametrized into a B-spline surface with fixed boundaries by the least square method. In this paper, bicubic B-spline surfaces are used for the parametrization. For the complicated input boundary curves with degree larger than three, B-spline surfaces with higher degree will be used.
Fig. 4. Subdivision process on the model Woody. (a) the sparse quad patch layout $\mathcal{P}$; (b–f) the subdivision mesh using $\mathcal{P}$ after 1, 2, 3, 4, and 5 times subdivision. The subdivision process starts from (a) with all the edges colored red. In each subdivision, only the new edges produced by the partition of a red edge are colored red, which means these newly produced edges correspond to the layout $\mathcal{P}$. And the black edges do not correspond to $\mathcal{P}$. Thus, with these red edges, the final large patches of the subdivision mesh can be determined.

**B-splines** A B-spline curve $c(t)$ of degree $p$ defined over a knot vector $U = \{t_0, t_1, \ldots, t_{m+p+1}\}$ with $t_i < t_{i+p+1}$, $i = 0, 1, \ldots, m$, is defined as

$$c(t) = \sum_{i=0}^{m} C_i N_i^p(t), \ t \in [t_p, t_{m+1}],$$

where $C_i$ are control points and $N_i^p(t)$ is a B-spline basis function defined recursively as follows

$$N_0^p(t) = \begin{cases} \prod_{\ell=0}^{p} \frac{t - t_{\ell}}{t_{\ell+1} - t_{\ell}}, & t \in [t_{\ell}, t_{\ell+1}) \\ 0, & \text{otherwise} \end{cases},$$

$$N_i^p(t) = \frac{t - t_i}{t_{i+p} - t_i} N_i^{p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1}^{p-1}(t), \quad p \geq 1.$$

Let $\{C_{ij}\}_{i=0}^m, j=0 \in \mathbb{R}^d$ be $(m + 1)(n + 1)$ control points, a B-spline surface of degree $(p, q)$ is defined by

$$B(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} C_{ij} N_i^p(u) N_j^q(v) = \sum_{i=0}^{m} \sum_{j=0}^{n} C_{ij} B_{ij}(u, v).$$

where $B_{ij}(u, v) = N_i^p(u) N_j^q(v)$, $N_i^p(t)$ and $N_j^q(t)$ are the B-spline basis functions of degree $p$ and $q$ defined on knot vectors $U = \{u_0, u_1, \ldots, u_{m+p+1}\}$ and $V = \{v_0, v_1, \ldots, v_{n+q+1}\}$, respectively.

**Least square approximation with splines** Given a set of data $\{P_i\}_{i=1}^N, P_i \in \mathbb{R}^2$ equipped with parameter values $\{(s_i, t_i)\}_{i=1}^N$, then the least square approximation with splines defined by Eq. (14) is defined as

$$\min_{B(u, v)} \sum_{i=1}^{N} (B(s_i, t_i) - P_i)^2.$$  

(16)

When the degree and knot vectors of splines are fixed, Problem (16) is simplified as

$$\min_{X} \|WX - P\|^2,$$

(17)

where $P = (P_1, P_2, \ldots, P_N)^T$, and $X = (X_1, X_2, \ldots, X_K)^T$, $X_i \in \mathbb{R}^2$, $K = (m + 1)(n + 1)$ is the control points vector, and $W = (w_{ij})_{N \times K}$ with $w_{ij} = N_i^p(s_j) N_0^q(t_i), l_0 + j_0(m + 1) = j$.

If $W$ has rank $K$, then the solution of Problem (17) is $X = (W^TW)^{-1}W^TP$. The sufficient and necessary condition for $W$ having full column rank is stated in Schoenberg and Whitney (1953). If $W$ doesn’t have full column rank, $X$ is defined as the solution with minimal $\|X\|_2$ among all the solutions.

**Post-optimization** After the least square approximation, we further optimize the $E_{fold}$ to eliminate the foldovers (if they exist) by replacing the discrete Jacobian with the Jacobian of $B(u, v)$. Finally, the $E_{conf}$ is optimized with respect to the control points to make the parameterizations have better quality. The boundary control points are fixed during these two optimization processes. One example is shown in Fig. 1 (g).
Algorithm 1 Preconditioned Accelerated Proximal Gradient method (PAPG).

Initialize \( x_1 = x_1 = x_0 \) which satisfies \( Ax_0 = b, t_1 = 1, t_0 = 0, \eta \in (0, 1), \delta > 0, c_1 = f(x_1), q_1 = 1; \)
while not converged do
    \( y_k = x_k + \frac{1}{t_k}(x_k - x_k) + \frac{q_{k-1}}{t_k}(x_k - x_{k-1}); \)
    \( p^*_k = \mathbb{D}(y_k); \)
    \( x_{k+1} = \text{linesearch}_{0, \alpha_k} E(y_k + \alpha_k \cdot p^*_k); \)
    if \( f(x_{k+1}) \leq c_k - \delta \|x_{k+1} - y_k\|^2 \) then
        \( x_{k+1} = z_{k+1}; \)
    else
        \( p^*_k = \mathbb{D}(x_k); \)
        \( v_{k+1} = \text{linesearch}_{0, \alpha_k} E(x_k + \alpha_k \cdot p^*_k); \)
        if \( f(x_{k+1}) \leq f(v_{k+1}) \) then
            \( x_{k+1} = z_{k+1}; \)
        else
            \( x_{k+1} = v_{k+1}; \)
        end if
    end if
end while

3.4. Preconditioned accelerated proximal gradient method

To minimize the involved energies \( (E_{\text{deform}}, E_{\text{fold}}, E_{\text{cont}}) \) efficiently and effectively, we use the preconditioned accelerated proximal gradient method (PAPG), which is based on the accelerated proximal gradient method (APG) \( (\text{Li and Lin, 2015}) \). The optimization problem is:

\[
\min_{x} E(x) \\
\text{s.t. } Ax = b,
\]

where \( x \) is the column stack of the vertex positions. The linear constraints are common, such as the positional constraints of some vertices or control points. The detailed procedure is illustrated in Algorithm 1. The step size \( \alpha_k \) is determined by a backtracking line search to decrease the objective function. The operator \( \mathbb{D}(\cdot) \) computes the descent direction \( p_k \) with respect to \( x_k \). In this paper, we minimize the following problem to compute \( p_k \):

\[
\min_{p_k} \|Hp_k + \nabla E(x_k)\|_2^2 \\
\text{s.t. } Ap_k = 0,
\]

where \( H \) is the selected preconditioner, as which is chosen the Laplacian matrix of the mesh by default. We find \( p_k \) by solving the linear KKT system:

\[
\begin{bmatrix}
    H & A^T \\
    A & 0
\end{bmatrix}
\begin{bmatrix}
    p_k \\
    \lambda
\end{bmatrix} = 
\begin{bmatrix}
    -\nabla E(x_k) \\
    0
\end{bmatrix}.
\]

Because the preconditioner \( H \) is constant, we pre-factorize it once during the preprocessing. It turns out that this algorithm performs efficiently and effectively when minimizing these nonlinear energy functions. Fig. 5 shows the comparison with two other optimization methods (L-BFGS and APG) on optimizing \( E_{\text{fold}} \) to eliminate the foldovers. Both L-BFGS and APG failed to converge in 3000 iterations, containing some flipped triangles (see the yellow triangles in the zoom-in views in Fig. 5). However, our PAPG converges in 344 iterations (1.36 seconds) and uses 177 iterations to eliminate all foldovers.

4. Experiments and applications

We applied our PolySquare-enhanced domain partition method to various input domains. Our method was implemented in C++ and all experiments were run on a desktop computer with an Intel Core i7-6560U processor and 8GB memory.

Quality metric for IGA-suitable parameterizations For a given parametrization \( B(u, v) \), let \( J \) be the Jacobian matrix of the mapping \( B \). Suppose \( B(u, v) = (x(u, v), y(u, v))^T, x(u, v), y(u, v) \in \mathbb{R} \), then

\[
J = (B_u \ B_v) = \begin{bmatrix}
    \frac{\partial x(u, v)}{\partial u} & \frac{\partial x(u, v)}{\partial v} \\
    \frac{\partial y(u, v)}{\partial u} & \frac{\partial y(u, v)}{\partial v}
\end{bmatrix}.
\]

In order to evaluate the quality of the parameterization, we employ two indicators, one is the scaled Jacobians of the mapping \( B(u, v) \) defined as the determinant of the matrix \( J_B \).

\[
J_B(u, v) = \frac{J(u, v)}{\|B_u\|\|B_v\|},
\]
Fig. 6. (a) L-BFGS, (b) APG, (c) PAPG. Comparison with L-BFGS and APG (Li and Lin, 2015) on a 2D fixed boundary mapping problem. The input mesh is a square and mapped to the domain with prescribed boundary. The initial mapping is the discrete harmonic mapping, which contains flipped triangles. Three methods try to eliminate them by optimizing $E_{\text{fold}}$ while fixing the boundary vertices. The color on triangles encodes the conformal distortion, with white being optimal. We use the metric of Fu et al. (2015) to measure the conformal distortion $\varphi_k^\text{inv} := \sigma_k^{\text{max}}/\sigma_k^{\text{min}}$, where $\sigma_k^{\text{max}}$ and $\sigma_k^{\text{min}}$ are the maximum and minimum singular values of $J_k$. The regions colored with yellow indicate the flipped triangles.

the other one is the condition number of the Jacobian matrix $J$ defined as

$$\kappa(J) = \|J\| \|J^{-1}\| = (\text{tr}(J^T J))^{1/2}. \quad (23)$$

It can be proved that the closer to 1.0 the scaled Jacobians of a parametrization is, the better the parametrization is. Similarly, the closer to 2.0 the condition number of Jacobian of a parametrization is, the better the parametrization is.

In the following examples, we use the colormap in Fig. 1 to demonstrate the scaled Jacobians of the iso-parametric curves of a parametrization which is a multiple B-spline surface. The statistic data, meaning the minimum value, average value and maximum value, of the scaled Jacobians and the condition number of all examples presented in this paper is shown in Table 1.

4.1. IGA-suitable planar parameterizations

Different $\sigma$ The Gaussian kernel size $\sigma$ affects the range of normal smoothing. We show one comparison using different $\sigma$ in Fig. 6. Although the PolySquares are different, the final partition is similar because the simplification is stable to reduce most poly-chords, thereby resulting in similar quad layout.

Various domains We test our algorithm on 34 various input domains, including high-genus ones, and show 18 ones in Fig. 7. Other results are provided in the supplemental material. The patch numbers for all examples are small, even for the domain with 30 holes, and the number of control points are less than 3100. Thus, the computational cost of the matrix assembly for IGA is acceptable. The average scaled Jacobian metrics of all examples are very high, indicating our parameterizations have high quality. The minimum scaled Jacobian metrics are greater than zero, but approach zero. All the same, our parameterizations are reasonable and good, because the minimum scaled Jacobian occurs on the edge of iso-parametric elements of B-spline surfaces. In IGA application, the Jacobians are evaluated at the Gauss integral points which are in the interior of the iso-parametric elements. Thus it has less effect for the IGA.

Timings We collected the running time for the triangular mesh Rabbit (Fig. 8 (d)) with 4290 vertices. The alignment process took 1.40 seconds. The pixelation took 0.47 seconds. The back projection process took about 2.36 seconds. The simplification took about 0.30 seconds. And the subdivision and optimization took 1.31 seconds. The parametrization phase took 2.87 seconds. For another triangular mesh Woody with one hole (the leftmost one in the third row of Fig. 7) and 617 vertices, our method ran about 0.14 seconds in the alignment process, 0.11 seconds in the pixelation, 0.38 seconds in the back projection and 0.28 seconds in the simplification, 0.85 seconds in the subdivision and optimization, and 3.41 seconds in the parametrization phase.
### Table 1

Statistic data of planar parametrization by our method for the models shown in this paper, here \( N_p \) indicates the patch number and \( DOF \) means the total number of control points of all patches.

<table>
<thead>
<tr>
<th>Model</th>
<th>( N_p )</th>
<th>DOF</th>
<th>Scaled Jacobians minimum</th>
<th>Scaled Jacobians average</th>
<th>Scaled Jacobians maximum</th>
<th>Condition number minimum</th>
<th>Condition number average</th>
<th>Condition number maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaf_1hole</td>
<td>10</td>
<td>490</td>
<td>0.0020</td>
<td>0.8803</td>
<td>1.0000</td>
<td>2.0002</td>
<td>3.7157</td>
<td>1043.2000</td>
</tr>
<tr>
<td>Leaf_2hole</td>
<td>15</td>
<td>375</td>
<td>0.0052</td>
<td>0.8544</td>
<td>1.0000</td>
<td>2.0003</td>
<td>4.0293</td>
<td>652.7040</td>
</tr>
<tr>
<td>Leaf_3hole</td>
<td>19</td>
<td>931</td>
<td>0.0061</td>
<td>0.8693</td>
<td>1.0000</td>
<td>2.0001</td>
<td>2.9260</td>
<td>344.0880</td>
</tr>
<tr>
<td>Map_1hole</td>
<td>9</td>
<td>441</td>
<td>0.0049</td>
<td>0.8318</td>
<td>1.0000</td>
<td>2.0001</td>
<td>3.1872</td>
<td>414.6180</td>
</tr>
<tr>
<td>Map_2hole</td>
<td>11</td>
<td>539</td>
<td>0.0366</td>
<td>0.8779</td>
<td>1.0000</td>
<td>2.0001</td>
<td>2.6218</td>
<td>96.1585</td>
</tr>
<tr>
<td>Map_3hole</td>
<td>19</td>
<td>931</td>
<td>0.0181</td>
<td>0.9254</td>
<td>1.0000</td>
<td>2.0000</td>
<td>2.4573</td>
<td>144.3380</td>
</tr>
<tr>
<td>Rabbit_1hole</td>
<td>12</td>
<td>300</td>
<td>0.0080</td>
<td>0.9115</td>
<td>1.0000</td>
<td>2.0011</td>
<td>2.8999</td>
<td>249.2080</td>
</tr>
<tr>
<td>Rabbit_2hole</td>
<td>19</td>
<td>931</td>
<td>0.0198</td>
<td>0.9199</td>
<td>1.0000</td>
<td>2.0000</td>
<td>2.5564</td>
<td>170.8120</td>
</tr>
<tr>
<td>Rabbit_3hole</td>
<td>19</td>
<td>475</td>
<td>0.0020</td>
<td>0.9408</td>
<td>1.0000</td>
<td>2.0000</td>
<td>2.7313</td>
<td>1026.1000</td>
</tr>
<tr>
<td>Woody_1hole</td>
<td>15</td>
<td>735</td>
<td>0.0055</td>
<td>0.8924</td>
<td>1.0000</td>
<td>2.0004</td>
<td>3.2819</td>
<td>364.6330</td>
</tr>
<tr>
<td>Woody_2hole</td>
<td>15</td>
<td>735</td>
<td>0.0100</td>
<td>0.8892</td>
<td>1.0000</td>
<td>2.0001</td>
<td>2.7213</td>
<td>200.8860</td>
</tr>
<tr>
<td>Woody_3hole</td>
<td>25</td>
<td>625</td>
<td>0.0155</td>
<td>0.9136</td>
<td>1.0000</td>
<td>2.0001</td>
<td>2.6697</td>
<td>129.8090</td>
</tr>
<tr>
<td>Catus</td>
<td>16</td>
<td>400</td>
<td>0.0148</td>
<td>0.9334</td>
<td>1.0000</td>
<td>2.0005</td>
<td>2.5148</td>
<td>138.9220</td>
</tr>
<tr>
<td>LetterS</td>
<td>5</td>
<td>80</td>
<td>0.8289</td>
<td>0.9778</td>
<td>1.0000</td>
<td>2.0021</td>
<td>2.2869</td>
<td>3.8062</td>
</tr>
<tr>
<td>Sharp_3hole</td>
<td>22</td>
<td>352</td>
<td>0.1366</td>
<td>0.9531</td>
<td>1.0000</td>
<td>2.0000</td>
<td>2.4105</td>
<td>15.3130</td>
</tr>
<tr>
<td>Woody (Fig. 1 (g))</td>
<td>10</td>
<td>250</td>
<td>0.0100</td>
<td>0.9478</td>
<td>1.0000</td>
<td>2.0002</td>
<td>2.8470</td>
<td>200.1140</td>
</tr>
<tr>
<td>Jigsaw (Fig. 8 (a))</td>
<td>1</td>
<td>625</td>
<td>0.2635</td>
<td>0.7654</td>
<td>1.0000</td>
<td>2.0102</td>
<td>5.1701</td>
<td>88.389</td>
</tr>
<tr>
<td>Jigsaw (Fig. 8 (b))</td>
<td>21</td>
<td>1029</td>
<td>0.0051</td>
<td>0.9662</td>
<td>1.0000</td>
<td>2.0004</td>
<td>2.3631</td>
<td>395.1100</td>
</tr>
<tr>
<td>Rabbit (Fig. 8 (c))</td>
<td>1</td>
<td>176</td>
<td>0.0019</td>
<td>0.8003</td>
<td>1.0000</td>
<td>2.0005</td>
<td>3.6177</td>
<td>1906.4000</td>
</tr>
<tr>
<td>Rabbit (Fig. 8 (d))</td>
<td>10</td>
<td>250</td>
<td>0.0162</td>
<td>0.9256</td>
<td>1.0000</td>
<td>2.0004</td>
<td>2.7053</td>
<td>137.3060</td>
</tr>
<tr>
<td>Zhong (Fig. 9 (b2))</td>
<td>206</td>
<td>5150</td>
<td>0.0166</td>
<td>0.9581</td>
<td>1.0000</td>
<td>2.0000</td>
<td>2.2869</td>
<td>122.9100</td>
</tr>
<tr>
<td>Zhong (Fig. 9 (d2))</td>
<td>15</td>
<td>735</td>
<td>0.0333</td>
<td>0.9765</td>
<td>1.0000</td>
<td>2.0004</td>
<td>2.3418</td>
<td>60.9190</td>
</tr>
<tr>
<td>Shape (Fig. 9 (a2))</td>
<td>48</td>
<td>1200</td>
<td>0.2320</td>
<td>0.9011</td>
<td>1.0000</td>
<td>2.0004</td>
<td>2.5464</td>
<td>8.6836</td>
</tr>
<tr>
<td>Shape (Fig. 9 (c2))</td>
<td>20</td>
<td>980</td>
<td>0.0074</td>
<td>0.9689</td>
<td>1.0000</td>
<td>2.0001</td>
<td>2.6523</td>
<td>276.0260</td>
</tr>
<tr>
<td>Exam_10hole</td>
<td>83</td>
<td>2075</td>
<td>0.0144</td>
<td>0.9281</td>
<td>1.0000</td>
<td>2.0001</td>
<td>2.5415</td>
<td>138.4400</td>
</tr>
<tr>
<td>Exam_20hole</td>
<td>119</td>
<td>2975</td>
<td>0.0061</td>
<td>0.9047</td>
<td>1.0000</td>
<td>2.0000</td>
<td>2.5908</td>
<td>398.3100</td>
</tr>
<tr>
<td>Exam_30hole</td>
<td>192</td>
<td>3072</td>
<td>0.0068</td>
<td>0.8577</td>
<td>1.0000</td>
<td>2.0000</td>
<td>3.0655</td>
<td>292.3000</td>
</tr>
</tbody>
</table>

**Fig. 6.** Different \( \sigma \). \( \sigma = 2.0 \bar{L} \) for the first row and \( \sigma = 3.0 \bar{L} \) for the second row, where \( \bar{L} \) is the average edge length of the input triangular mesh. The simplified quad layouts (a4) and (b4) have the same topology and different positions. The final partitions (a5) and (b5) are similar because of the subdivision and optimization.

### 4.2. Comparisons

We compare our PeDP method with two state-of-the-art methods, i.e., Teichmüller mapping method (Nian and Chen, 2016) and approximate convex decomposition method (Xu et al., 2018).
Fig. 7. Domain partition and parameterization results on eighteen planar domains. The domains are partitioned by the red lines. The scaled Jacobian of the resulting parameterizations are color-coded using the colorbar in Fig. 1. From the first row to the fourth row, the models are named Leaf, Map, Rabbit, and Woody. From left to right in the fifth row, they are Cactus, LetterS, and Sharp. In the sixth row, the examples with 10, 20, and 30 holes are shown.

Fig. 8. Comparison with Nian and Chen (2016). The parameterizations of Nian and Chen (2016) are shown in (a, c), and ours are in (b, d). We use the colormap in Fig. 1 to encode the scaled Jacobian. The parameterization results of Nian and Chen (2016) are kindly provided by the authors.
Comparison with Nian and Chen (2016)  In Fig. 8, two simply connected domains are tested for comparing our method with Nian and Chen (2016). From the color-coded scaled Jacobian, our method achieves much higher quality. Although a Teichmüller map is injective, the computational result is just an approximation and their results leave room for reducing the distortion. To generate the results in Fig. 8(a) and (c), the Teichmüller method (Nian and Chen, 2016) spends about 18 minutes and 30 minutes, respectively. The corresponding run timings of our algorithm are 29.09 seconds and 8.71 seconds, respectively.

Comparison with Xu et al. (2018)  In Fig. 9, we compare with Xu et al. (2018) on two examples. We obtain the comparable average scaled Jacobian with Xu et al. (2018) and much fewer patches than theirs. Their key idea utilizes the pattern based quad mesh generation to partition the polygons that are generated by the approximate convex decomposition of the input domain. This method usually generates much more patches than ours, leading to higher computational cost for the subsequent matrix assembly. For Xu et al. (2018), they generate the parameterizations of these two models using 50.14 seconds and 251.08 seconds, while our algorithm only takes 18.01 seconds and 7.87 seconds, respectively.

5. Conclusion and discussion

We present an effective method to compute the IGA-suitable planar parameterizations for complex and high genus domains with low distortion and injectivity. Our method exhibits higher parameterization quality and fewer patches than previously existing methods. A few limitations do exist, and we would like to address them in future work.

Theoretical guarantee on foldover elimination  Given the target positions of the boundary vertices of an input domain, the problem of computing a foldover-free mapping while satisfying the boundary position constraints is discussed here. Note that the input domain is fixed in this discussion. When the target domain boundary is convex, the Tutte’s embedding methods (Tutte, 1963) could always guarantee on eliminating all foldovers. In addition, if the input and target domains are disk topology, the method (Weber and Zorin, 2014) has a theoretical guarantee. However, for any high genus domains and arbitrary boundary position constraints, no general algorithm have theoretical guarantees on always eliminating all foldovers in our view. Therefore, some optimization methods are proposed to try to eliminate the foldovers, for example the Simplex Assembly method (SA) (Fu and Liu, 2016). We give a comparison with the SA in the right inset. The input includes a triangle mesh and the target positions of its boundary vertices. Our method optimizes the $E_{fold}$ using our PAG solver. The color on the results indicates the conformal distortion (whose definition is in the caption of Fig. 5). Our method generates the comparable result to the SA. The run timings of the SA and ours are 3.47 seconds and 1.54 seconds, respectively. Our method is faster than the SA. The foldover elimination technique with theoretical guarantee for arbitrary boundaries is a promising research direction.

Besides, some methods subdivide the input domains to achieve the foldover-free mappings, such as Massarwi et al. (2017). This problem is different from our considered one where the input domain is fixed during the mapping computation.
Minimum scaled Jacobian The minimum scaled Jacobian is very small in our method because the boundary angle between two curves of one patch may be near 180°. Maybe some necessary patches are required, but they are difficult to determine. Especially, if we add more patches, it will lead to the inefficient matrix assembly. Thus, it is an interesting future work.

Integrate B-splines into domain partition Currently, the parametrization consists of two steps: domain partition and B-spline surface patches fitting. In the second step, the fitting is simply based on the discrete points output from the first step without optimization. A possible way which may be considered is to integrate the second step into the first step, which means we use the B-spline mapping instead of a linear mapping. However this leads to less freedom and complicated optimization problems. We will consider this in the future.

IGA-suitable volumetric parameterizations Intuitively, we could extend our method for decomposing the input volumetric domain into cube-like subdomains that can be parameterized easily. However, how to make this extension robust for general input domains is very challenging because the 3D versions of the components of our method are hard to handle and implement.

Limitation Our method cannot work well on the model with too many sharp features. And an example is shown in the right inset. Due to the PolySquare structure, our algorithm is not good at handling this case. Thus, we obtain a layout (b) and introduce some singularities on the boundary (see the regions bounded with the blue circles in (c)). However, a better partition in math is shown in (d) with an internal singular vertex. The partition in (d) results in a better IGA-suitable planar parameterization than the one in (c).

Acknowledgements

We thank Gang Xu for providing the results of Xu et al. (2018), Maodong Pan for generating the results of Nian and Chen (2016) and the anonymous reviewers for their constructive feedback. This work is supported by the Fundamental Research Funds for the Central Universities, the Scientific Research Foundation of Soochow University (No. Q410700117) and the National Natural Science Foundation of China (No. 11571338).

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cagd.2018.03.008.

References


