Computer Methods in Applied Mechanics and Engineering 441 (2025) 117988

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Integral parameterization of volumetric domains via deep neural networks

Zheng Zhan [a], Wenping Wang [b], Falai Chen [a],*

[a] School of Mathematical Sciences, University of Science and Technology of China, Hefei, Anhui 230026, PR China
[b] Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843-3112, USA

ARTICLE INFO

Keywords:
Isogeometric analysis
Volumetric domain parameterization
Deep neural network

ABSTRACT

Isogeometric Analysis (IGA) is a promising technique that integrates geometric modeling with numerical analysis. An essential step in IGA is domain parameterization, which aims to establish a parametric representation for a given computational domain. Specifically, it involves defining a spline-based mapping from the standard parametric domain to the computational domain. Typically, domain parameterization is performed in two stages: identifying an appropriate boundary correspondence and then parameterizing the interior region. However, this separation of the parameterization process often leads to a degradation in the quality of the parameterization. To attain high-quality parameterization, it is essential to optimize both the boundary correspondence and the interior mapping simultaneously. This approach is referred to as integral parameterization. Previous research has introduced integral parameterization methods for planar domains using neural networks. The goal of the current paper is to extend the method to handle integral parameterization of volumetric domains. We utilize Multi-Layer Perceptrons (MLPs) to represent the inverse parameterization mappings, incorporating efficient distortion measures into the loss function. To ensure stable training and achieve accurate results, we employ several techniques, including a four-stage training procedure and the smooth cuboid approach. The performance of our method is evaluated on multiple volumetric domains, and experimental results demonstrate its superiority over existing state-of-the-art techniques.

1. Introduction

Isogeometric analysis (IGA for short) [1] provides a promising tool that integrates geometric design and physical simulations for geometric models. This approach utilizes the same spline-based parameterization for both geometry and numerical analysis, eliminating the need for time-consuming operations such as mesh generation in traditional finite element analysis (FEA). In IGA, the domain parameterization step replaces the mesh generation process in FEA, which accounts for approximately 80% of the computation time in FEA [2]. For a given computational domain, domain parameterization constructs a parametric representation for it, i.e. a spline-based mapping from the standard parametric domain to the computational domain. An example for volumetric domain parameterization is illustrated in Fig. 1.

Traditional methods typically decompose domain parameterization into two stages. First, an appropriate boundary correspondence is specified manually or automatically through some algorithms [3–5]. Next, the interior parameterization is computed based on the predefined boundary correspondence. However, the boundary correspondences selected based on experience are often suboptimal. To overcome this limitation, the concept of integral parameterization was introduced, which aims to optimize both

* Corresponding author.
E-mail addresses: zz00822@mail.ustc.edu.cn (Z. Zhan), wenping@cs.hku.hk (W. Wang), chenfl@ustc.edu.cn (F. Chen).

https://doi.org/10.1016/j.cma.2025.117988
Received 9 January 2025; Received in revised form 5 March 2025; Accepted 31 March 2025
Available online 10 April 2025

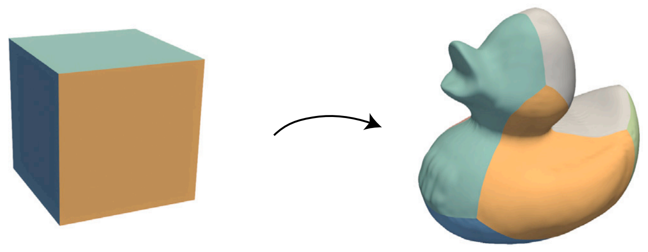


Fig. 1. Visualized example of volumetric domain parameterization.

boundary correspondences and interior mappings simultaneously [6–8]. The method proposed by Liu et al. [6] tends to converge to local optima, producing parameterizations with suboptimal boundary correspondences. Zhan et al. [7] and Zhan et al. [8] leverage neural networks to achieve integral parameterizations with more favorable boundary correspondences. Compared with traditional parameterization methods [6,9–12], the neural network-based integral parameterization approaches deliver superior parameterization results in less computational time. However, currently only planar domains are considered by this method. In this paper, we will extend the method to cope with volumetric domains.

The remainder of the paper is structured as follows. Section 2 provides a review of related work, while Section 3 introduces the necessary preliminaries. In Section 4, we present the details of our algorithm, including the network architecture design, loss function definition, etc. Section 5 includes a comparative analysis between our method and existing parameterization techniques, along with the presentation of our experimental results. Finally, Section 6 concludes the paper and discusses potential directions for future research.

2. Related works

2.1. Interior parameterization methods

For planar domain parameterization, discrete Coons patches [13] represent one of the earliest approaches. Subsequently, harmonic mapping-based approaches [11,14] are developed, offering some improvement in parameterization quality. However, these linear methods often produce unsatisfactory or even non-bijective mappings. To address this, nonlinear optimization algorithms have been employed to further enhance parameterization quality [9,10]. Additionally, replacing B-spline representation with local refinable splines has been shown to yield better results [15]. For volumetric domains, some of these methods can be directly extended to three dimensions, including harmonic mapping-based algorithms [16,17], optimization-based methods [14,18–21], and approaches utilizing locally refinable splines [22–25].

For topologically complex domains, multi-patch parameterization is often the preferred solution [14,26,27]. In these methods, the parametric domain is composed of multiple cuboid (or rectangular in planar case) regions, with each region corresponding to a part of the computational domain. Ensuring the continuity between these different patches is crucial and challenging.

2.2. Integral parameterization methods

Since the quality of parameterization is greatly influenced by the boundary correspondence [3,5–7], the separation of boundary correspondence and interior mapping in traditional parameterization methods often results in low-quality parameterizations. This shortcoming highlights the need for methods that can simultaneously optimize both boundary correspondence and interior mapping, known as integral parameterization methods. Currently, very few integral domain parameterization methods are devised [6–8]. Liu et al. [6] iteratively optimizes both the boundary correspondence and the interior mapping to improve parameterization quality. This method incorporates boundary mapping error as a constraint during the optimization process. However, due to the strong boundary mapping constraint, the algorithm is often trapped in local minima, resulting in poorly-behaved boundary correspondence. Zhan et al. [7] and Zhan et al. [8] utilizes neural networks to parameterize planar domains. To ensure high-quality parameterization, the loss function includes three major terms: angular and area distortion of the mapping, and boundary fitting error. Compared to traditional interior parameterization methods and the approach by [6], these neural network-based methods achieve higher bijectivity ratio, lower distortion and more uniform parameterizations.

2.3. Neural networks in parameterization

In recent years, deep learning methods have been applied to tackle various problems in geometric modeling and processing, including the parameterization problem [5,7,8,12,28–32]. Among these methods, [28–31] focus on surface parameterizations, which involves finding an appropriate mapping from a plane to a surface ($\mathbb{R}^2 \mapsto \mathbb{R}^3$). [32] use network-based method to generate high-quality hexahedral mesh, and thus construct volumetric spline. [5,7,8,12] address planar domain parameterization, mapping from planar parametric domains to planar computational domains. However, there is currently a lack of neural network based methods for volumetric domain parameterization that operate in $\mathbb{R}^3$.

3. Preliminary

In this section, we provide some preliminary concepts about B-spline representations and distortions in volumetric mapping.

3.1. B-spline representation

After we adopt the inverse parameterization represented by a neural network, we have to translate it into a B-spline represented forward parameterization as the final result. A trivariate tensor product B-spline is defined as

$$f(u,v,w)=\sum_{i=0}^{L}\sum_{j=0}^{M}\sum_{k=0}^{N}\mathbf{P}_{ijk}N_i^p(u)N_j^q(v)N_k^r(w), \tag{1}$$

where $\mathbf{P}_{ijk}\in\mathbb{R}^3$ are control points, N_i^p, N_j^q, N_k^r are the B-spline basis functions of degree p,q,r with respect to the knot sequences U,V,W, respectively. In this paper, we set degree $p=q=r=3$, and adopt quasi-uniform knot sequences, i.e.

$$\begin{aligned} U &= (0,0,0,0,1/L,2/L,\dots,1,1,1,1),\\ V &= (0,0,0,0,1/M,2/M,\dots,1,1,1,1),\\ W &= (0,0,0,0,1/N,2/N,\dots,1,1,1,1). \end{aligned}$$

3.2. Distortion measures of volumetric mapping

The quality of a volumetric mapping can be measured at a point $a=(u,v,w)$ by how it changes nearby a. In this paper, we measure the distortion by two major terms: the as-rigid-as-possible (ARAP) distortion [33] and the fairness of the mapping, which can be calculated pointwise by the Jacobian matrix and Hessian matrix respectively. Here we briefly introduce these two terms:

ARAP distortion. For a mapping $f=(f_1,f_2,f_3)$ that maps from the parametric domain $P\in\mathbb{R}^3$ to a computational domain $C\in\mathbb{R}^3$ and a point $a=(a_1,a_2,a_3)\in\mathbb{R}^3$, the Jacobian matrix $J(a)$ at point a can be represented by

$$J(a)=\left(\frac{\partial f_i(a)}{\partial a_j}\right)_{3\times 3}.$$

$J(a)$ can be decomposed by singular value decomposition as $J(a)=U^T\Sigma V$, in which $\Sigma=(\sigma_1^a,\sigma_2^a,\sigma_3^a)$ are the singular values, and U,V are orthogonal matrices. Then the ARAP distortion at point a is defined by $ARAP(f,a)=\sum_{i=1}^{3}(\sigma_i^a-1)^2$. Since we use the inverse mapping $G=f^{-1}$ of the parameterization in optimization, we calculate the ARAP distortion of the forward parameterization over the parametric domain P by

$$\begin{aligned}\int_P ARAP(f,a)da &= \int_P\sum_{i=1}^{3}(\sigma_i^a-1)^2da\\ &= \int_C \tilde{\sigma}_1^x\tilde{\sigma}_2^x\tilde{\sigma}_3^x\sum_{i=1}^{3}(\frac{1}{\tilde{\sigma}_i^x}-1)^2dx,\end{aligned}$$

where σ_i^a is the ith singular value of f at point $a\in P$, and $\tilde{\sigma}_i^x$ is the ith singular value of the inverse mapping G at point $x=f(a)$ which lies inside the computational domain C.

The ARAP distortion has a corresponding geometric explanation. The Jacobian matrix J represents a local linear transformation $\mathcal{J}$, and the singular value decomposition $J=U^T\Sigma V$ decomposes this transformation into three operations: rotation $\mathcal{V}$, scaling $\mathcal{E}$, and rotation $\mathcal{U}^*$. Then the singular values represent the scaling ratios in three directions.

Fairness. ARAP distortion measures the pointwise distortion by the Jacobian matrix, while fairness term measures how large the Jacobian matrix varies, i.e. how smooth the mapping is. We define the fairness of the mapping $f=(f_1,f_2,f_3)$ by

$$Fairness(f,x)=\|Hf_1(x)\|_F^2+\|Hf_2(x)\|_F^2+\|Hf_3(x)\|_F^2,$$

where $\{Hf_i\}_{i=1}^3$ are the Hessians of $\{f_i\}_{i=1}^3$, $\|\cdot\|_F$ is the Frobenius norm.

4. Method

Given a volumetric computational domain represented by its boundary mesh, we develop an integral parameterization method based on neural network to achieve high quality volumetric parameterization. In this section, we introduce this approach in detail.

Instead of computing a forward mapping from the parametric domain to a computational domain directly, our approach first compute an inverse mapping from a computational domain to the parametric domain and leverages a neural network to represent the inverse parameterization with carefully designed loss functions. The reason is that calculating the boundary error of an inverse parameterization is much easier than that of a forward parameterization [7,8]. Specifically, the network is modeled as a mapping $G:\mathbb{R}^3\to\mathbb{R}^3,(x,y,z)\mapsto(u,v,w)$, where a point (x,y,z) in the computational domain is mapped to the corresponding parametric coordinates (u,v,w) in the parametric domain. Once trained, the network provides the inverse parameterization for a specific computational domain C. Subsequently, B-spline fitting is applied to obtain the forward parameterization.

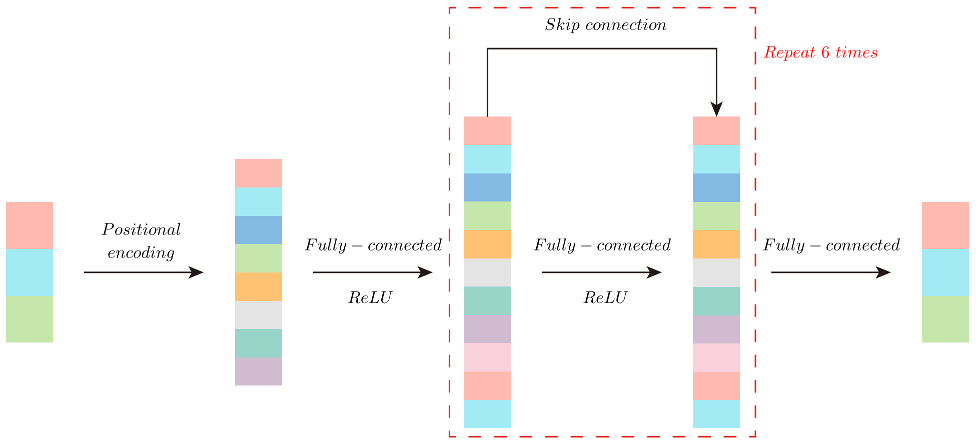


Fig. 2. Structure of our neural network. The part in the red wireframe represents residual modules.

4.1. *Problem formulation*

As described before, our goal in this paper is to train a neural network to represent a mapping that maps from a given volumetric domain C to the standard parametric domain $P = [0,1]^3$, which represent the inverse mapping of the parameterization. The problem is thus formulated as follows:

Formulation 1. *Given the parametric domain $P = [0,1]^3$ and a simply connected computational domain $C \subset \mathbb{R}^3$, find a mapping $G : C \mapsto P$ which satisfies the following requirements:*

1. *The mapping G is bijective.*
2. *The mapping G is smooth and exhibits low distortion.*
3. *The mapped boundary surface $G(\partial C)$ fits well with ∂P.*

The bijectivity can be ensured pointwise by the positivity of the Jacobian determinant. As discussed in Section 3.2, smoothness and distortion can be controlled through a fairness term and the ARAP energy. The boundary fitting error can be assessed by the distance between surfaces $G(\partial C)$ and ∂P and their normals. In this work, we use a neural network to represent the inverse mapping G and design a suitable loss function to achieve these requirements. The problem can be reformulated as:

Formulation 2. *Train a neural network G over the computational domain C in an unsupervised manner, such that the following loss function is minimized:*

$$L = L_{inner} + L_{bound} \tag{2}$$

where

$$\begin{aligned} L_{inner} &= w_{bij} L_{bij} + w_{arap} L_{arap} + w_{fair} L_{fair}, \\ L_{bound} &= w_{dist} L_{dist} + w_{norm} L_{norm} + w_{star} L_{star} \\ &\quad + w_{cor} L_{cor} + w_{edge} L_{edge}. \end{aligned} \tag{3}$$

L_{inner} represents the interior loss of mapping G over the computational domain C, including bijectivity term L_{bij}, ARAP distortion L_{arap} and fairness distortion L_{fair}. L_{bound} represents the boundary fitting error, which includes the distance from $G(\partial C)$ to ∂P, the difference between the normals of $G(\partial C)$ and ∂P, and the constraint L_{star} for non self-intersection of $G(\partial C)$. It is worth noting that, L_{dist} only limits the one-sided distance from $G(\partial C)$ to ∂P, as the two-sided Hausdorff distance $H(G(\partial C), \partial P)$ can be controlled by $h(G(\partial C), \partial P)$ [7,8]. L_{cor}, L_{edge} are only used in the last stages of training, representing the corner points mapping error and edge mapping error, respectively. Constants w_{bij}, w_{arap}, w_{fair}, w_{dist}, w_{norm}, w_{star}, w_{cor} and w_{edge} are nonnegative weights. Further details will be provided in Sections 4.3 and 4.4.

The training process of the neural network is illustrated in Section 4.3. After the network is trained, we use a tensor product trivariate B-spline to approximate the forward parameterization.

4.2. *Neural network architecture*

To represent the mapping G in Formulation 1 using a neural network, we design the network architecture as in Fig. 2. For each input point $v = (x, y, z) \in \mathbb{R}^3$, positional encoding is first utilized to map it to a higher dimensional vector. Then the vector is fed into an MLP which contains multiple residual module (in the red wireframe), and finally obtain the corresponding point (u, v, w) in the parametric space. In the following, we explain the positional encoding and the MLP in detail.

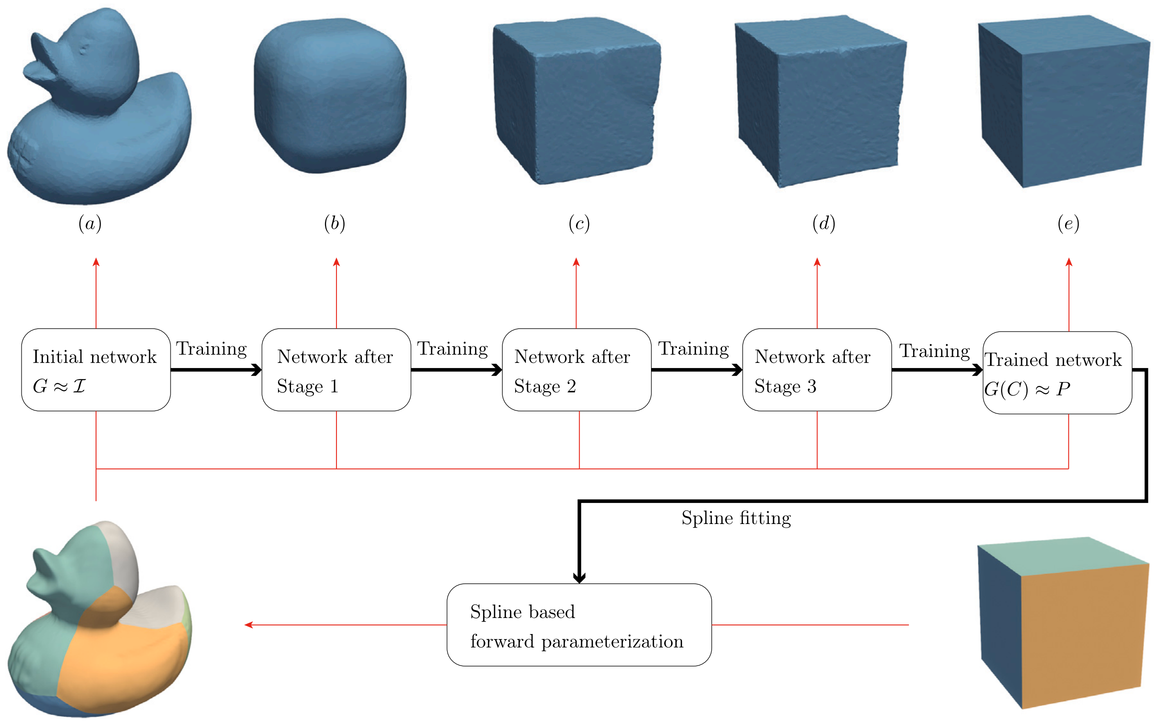


Fig. 3. Visualization of network training and spline fitting. Figures (a)–(e) show the mapped computational domain $G(C)$ in different training stages.

Positional encoding. Positional encoding is utilized to provide some positional information of the input, and it is widely used in Neural Radiance Fields (NeRF) [34] and Transformer [35]. Generally, positional encoding maps the inputs into a higher dimensional space using high-frequency functions, enhancing the model's ability to fit data with high-frequency variations. In this study, we employ positional encoding to map 3-D coordinates (x_1, x_2, x_3) into a $(6N + 3)$-dimensional vector

$$\left(x_1, x_2, x_3, \left\{ \frac{sin(nkx_i)}{nk}, \frac{cos(nkx_i)}{nk} \right\}_{n=1,i=1}^{N,3} \right).$$

In this expression, k is a trainable parameter initialized by 1 which represent the scaling ratio of the input. In practice, we set $N = 10$ and obtain a 63 dimensional encoding.

MLP. As shown in Fig. 2, our network is an MLP consisting of multiple residual modules. The architecture includes only fully connected layers, activation functions, and skip connections. We utilize the rectified linear unit (ReLU) as the activation function. The network accepts a $6N+3$ dimensional encoding as input and maps it to a $W = 512$ dimensional vector through a fully connected layer with ReLU activation. Subsequently, multiple residual modules process this vector, and finally, a fully connected layer produces the final output.

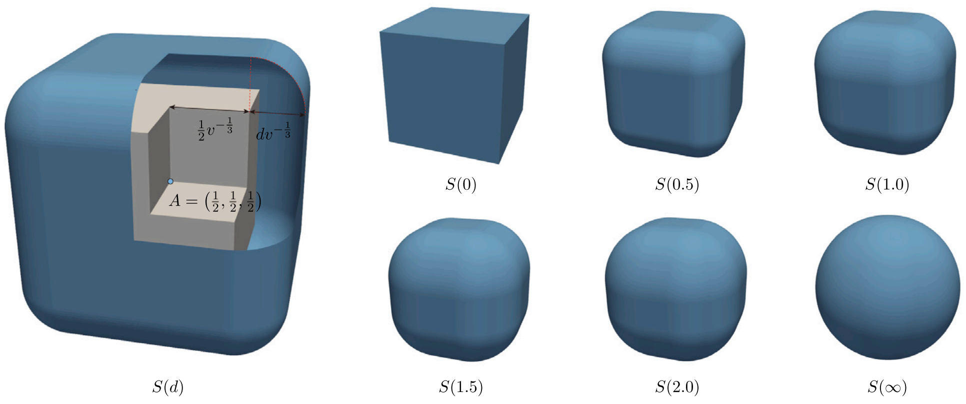
4.3. Algorithm pipeline

Given a volumetric domain C, we first scale the domain to ensure the volume of C is 1, here we still denote the domain by C. Then, we parameterize the domain via two main steps: training a neural network to represent the inverse mapping, and use a trivariate tensor product B-spline function to approximate the forward mapping. In the following, we discuss the two major steps in detail.

4.3.1. Inverse parameterization

In this step, our goal is to map the domain C to the parametric domain $P = [0, 1]^3$. First, the network mapping G is initialized to be an identity map, i.e. $G(x) = x, \ \forall x \in C$. As shown in Fig. 3(a), the mapped computational domain $G(C)$ is identical to the original domain C in the initial network. Then a network is trained which maps C to a sequential shapes as shown in Fig. 3(b)–(e).

1. Map domain C to a smooth cuboid $S(d)$.
   Since the discontinuity of the face normals of the cube boundary ∂P likely leads to unstable network training and local optima, we first map the domain to a smooth cuboid, which is shown in Fig. 4 and defined as follows:



$\frac{1}{2}v^{-\frac{1}{3}}$
$dv^{-\frac{1}{3}}$
$A = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$
$S(d)$
$S(0)$
$S(0.5)$
$S(1.0)$
$S(1.5)$
$S(2.0)$
$S(\infty)$

Fig. 4. Visualization of the smooth cuboid $S(d)$ with different parameter d.

Table 1
The weights corresponding to the terms in the loss function. "0" refers to the loss function term not being used.

Training stage	w_{bij}	w_{arap}	w_{fair}	w_{dist}	w_{norm}	w_{star}	w_{cor}	w_{edge}
1	10^6	1	2×10^{-3}	10^3	10	10^3	0	0
2	10^6	1	2×10^{-3}	10^3	10	10^3	0	0
3	10^6	1	2×10^{-3}	10^4	10	10^3	10^3	0
4	10^6	1	2×10^{-3}	10^4	10	10^3	10^3	10^4

Definition 1. Let K be a cube, and let the region $R(K,d) = \left\{\mathbf{x} \in \mathbb{R}^3 \mid dist(\mathbf{x}, K) \le d\right\}, \forall d > 0$. The boundary $\partial R(K,d)$ forms an isometric surface of K, and we call $R(K,d)$ a smooth cuboid.
It can be calculated that the volume of the region $R(P,d)$ is $v = \frac{4}{3}\pi d^3 + 3\pi d^2 + 6d + 1$. Since P and C both have volume 1, we scale $R(P,d)$ to obtain a smooth cuboid $S(d)$ which centered at $A = (1/2, 1/2, 1/2)$ with volume 1:

$$S(d) = \left\{ v^{-\frac{1}{3}}\mathbf{x} + (1 - v^{-\frac{1}{3}})A \mid \mathbf{x} \in R(P,d) \right\}.$$

In this step of network training, the parametric domain is the cuboid $S(d)$. The boundary loss term L_{dist} measures the distance from $G(\partial C)$ to $\partial S(d)$, and L_{norm} measures the difference of the normal vectors between $G(\partial C)$ and $\partial S(d)$. In practice, the loss terms $L_{bij}, L_{arap}, L_{fair}, L_{dist}, L_{norm}, L_{star}$ are used in this step and the training takes 1000 iterations. We use $S(2)$ as the parametric domain for the first 600 iterations and use $S(1)$ for the next 400 iterations. After this stage, $G(C)$ is close to $S(1)$ as shown in Fig. 3(b).

2. Map domain C into a unit cube.
   We set the parametric domain as P and continue the training. Now the boundary loss term L_{dist} measures the distance between $G(\partial C)$ to ∂P, and L_{norm} measures the difference of the normal vectors of $G(\partial C)$ and ∂P. This step still uses the loss function terms $L_{bij}, L_{arap}, L_{fair}, L_{dist}, L_{norm}, L_{star}$ and requires 1000 iterations. Fig. 3(c) shows the resulting $G(C)$.
3. Training with fixed corner mapping.
   Now the mapped computational domain $G(C)$ is nearly a cube, but there is still error between $G(\partial C)$ and ∂P. To further reduce the boundary error, in the current step, the parametric domain is still P and 8 corner points are selected based on the current mapping. In the training process, we add a loss term L_{cor} with fixed corner mapping, and enlarge the weight w_{dist}. This step takes about 500 iterations. Section 4.4 explains how to choose the corner mapping.
4. Training with fixed edge mapping.
   Since the corner points are accurately mapped, i.e. $G(v_{c_i}) = P_i$, we deal with the edge mapping in this step. The edge mapping will be controlled by the loss function term L_{edge} defined in Section 4.4. This step utilizes all the loss function terms $L_{bij}, L_{arap}, L_{fair}, L_{dist}, L_{norm}, L_{star}, L_{cor}, L_{edge}$ and takes 1000 iterations. Fig. 3(e) shows the resulting $G(C)$. Section 4.4 discusses how to obtain the edge mapping.

Among these steps, we use the loss function in Eq. (2) to ensure accurate boundary mapping and low-distortion interior mapping. Loss term L_{cor} is only applied in step 3,4 and L_{edge} is only adopted in step 4. Each term is described in detail in Section 4.4, and the weights are listed in Table 1. The network training time for each computational domain is about 200 s.

4.3.2. Forward parameterization

After the network training, we use a trivariate tensor product B-spline function F to approximate the forward mapping s.t. $G(F(x)) \approx x, \forall x \in [0,1]^3$. The spline function F has $30 \times 30 \times 30$ control points with tri-degree $(3,3,3)$. The procedure consists of three steps:

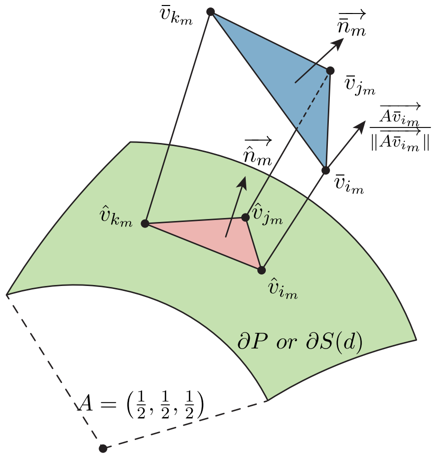

$\bar{v}_{k_m}$
$\overrightarrow{\bar{n}_m}$
$\bar{v}_{j_m}$
$\frac{\overrightarrow{A\bar{v}_{i_m}}}{\|\overrightarrow{A\bar{v}_{i_m}}\|}$
$\overrightarrow{\hat{n}_m}$
$\bar{v}_{i_m}$
$\hat{v}_{k_m}$
$\hat{v}_{j_m}$
$\hat{v}_{i_m}$
∂P or $\partial S(d)$
$A = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$

Fig. 5. Calculate the boundary loss of a facet.

1. Determine boundary mapping.
   ∂C is represented by a triangle mesh (V, T), and after network training, the mapped mesh $(G(V), G(T))$ approximates ∂P, although a small error remains. In this step, we find the nearest point v_P on ∂P for each vertex $\bar{v}$ of $(G(V), G(T))$, and then obtain a target mesh (V_P, T_P) on ∂P. Here each vertex $v \in V$ has a corresponding vertex $v_P \in V_P$, and each facet $t \in T$ has a corresponding facet $t_P \in T_P$.
2. Least squares approximation.
   In this step, we compute the control points of F by minimizing $\int_P \|G(F(x)) - x\|^2 dx$ via least squares method. The integration is computed by Gaussian quadrature.
3. Optimization.
   In this step, the spline parameterization F will be further optimized using optimization algorithms. The control points of F serve as the design variables, and the objective function consists of two components: an interior term and a boundary term. The interior term is defined similarly to the interior loss of the neural network, which will not be repeated here. The boundary term quantifies the boundary mapping errors, including vertex position errors and normal vector errors:
   $$\frac{1}{\#V}\sum_{v\in V}\|F(v_p) - v\|^2, \qquad \frac{1}{\#T}\sum_{t\in T}\|n(F(t_p)) - n(t)\|^2,$$
   with $n(\cdot)$ representing the normal vector. The optimization step takes 100 iterations and costs about 40 s.

4.4. Calculation of the loss function

We train our network in an unsupervised manner, thus a loss function that measures the parameterization quality is required. As described in (2), the loss function L is composed of two parts: L_{bound} and L_{inner}. L_{bound} measures the error between the mapped boundary $G(\partial C)$ and the target boundary ∂P (or $\partial S(d)$), and it is comprised of five terms L_{dist}, L_{norm}, L_{star}, L_{cor} and L_{edge}. L_{inner} measures the interior mapping distortion which consists of three terms L_{bij}, L_{arap} and L_{fair}. Below we explain the details about how these terms are calculated.

4.4.1. Boundary loss

Assume that the boundary ∂C of the volumetric domain C is represented by a triangular mesh (V, T), in which $V = (v_1, \ldots, v_N), T = (t_1, \ldots, t_M)$ represent the vertices and faces. The neural network maps the boundary mesh to $(G(V), G(T))$, where $G(V) = (\bar{v}_1, \ldots \bar{v}_N)$. We denote their nearest points on the target boundary ∂P or $\partial S(d)$ by $(\hat{v}_1, \ldots \hat{v}_N)$. It is worth noting that $\hat{v}_i$ will be updated in each iteration in corresponding to $\bar{v}_i$. Fig. 5 shows a facet $G(t_m) = (\bar{v}_{i_m}, \bar{v}_{j_m}, \bar{v}_{k_m})$ and its target $(\hat{v}_{i_m}, \hat{v}_{j_m}, \hat{v}_{k_m})$. The norm of the two facets can be calculated by:

$$\overrightarrow{\bar{n}_m} = \frac{\overrightarrow{\bar{v}_{i_m}\bar{v}_{j_m}} \times \overrightarrow{\bar{v}_{i_m}\bar{v}_{k_m}}}{\|\overrightarrow{\bar{v}_{i_m}\bar{v}_{j_m}}\|\|\overrightarrow{\bar{v}_{i_m}\bar{v}_{k_m}}\|}, \quad \overrightarrow{\hat{n}_m} = \frac{\overrightarrow{\hat{v}_{i_m}\hat{v}_{j_m}} \times \overrightarrow{\hat{v}_{i_m}\hat{v}_{k_m}}}{\|\overrightarrow{\hat{v}_{i_m}\hat{v}_{j_m}}\|\|\overrightarrow{\hat{v}_{i_m}\hat{v}_{k_m}}\|}.$$

Then we define L_{dist} and L_{norm} by:

$$\begin{aligned} L_{dist} &= \frac{1}{N}\sum_{i=1}^{N}\|\bar{v}_i - \hat{v}_i\|^2, \\ L_{norm} &= \frac{1}{M}\sum_{m=1}^{M}\|\overrightarrow{\bar{n}_m} - \overrightarrow{\hat{n}_m}\|^2. \end{aligned}$$

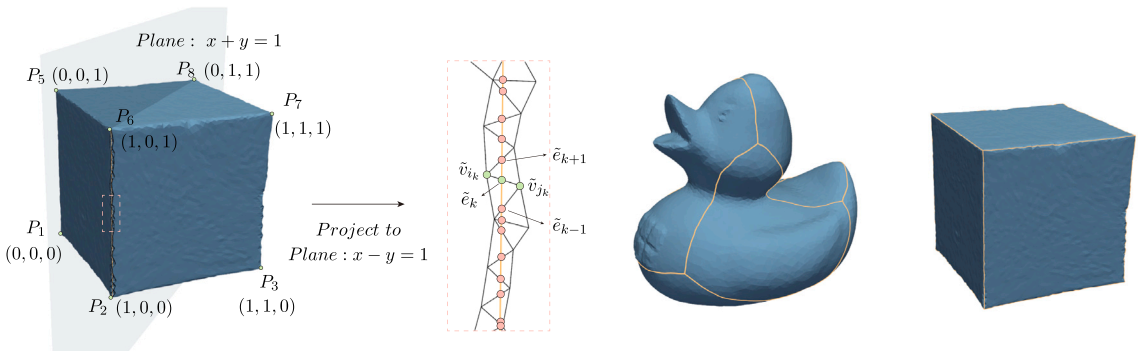


Fig. 6. The edges on (V,T) and $(G(V),G(T))$.

From Lemma 1 in the Appendix, $(G(V),G(T))$ forms a star-shaped domain with the center $A = \left(\frac{1}{2},\frac{1}{2},\frac{1}{2}\right)$ if and only if $\langle \overrightarrow{A\bar{v}_{i_m}}, \overrightarrow{\bar{n}_m}\rangle \geq 0, m = 1,\ldots,M$. Thus we define L_{star} by:

$$L_{star} = \frac{1}{M}\sum_{i=m}^{M}\left(-\langle \frac{\overrightarrow{A\bar{v}_{i_m}}}{\|\overrightarrow{A\bar{v}_{i_m}}\|}, \overrightarrow{\bar{n}_m}\rangle\right)_+ .$$

As mentioned in Section 4.3, after the second stage of the training process, the mapped computational domain $G(C)$ is nearly a cube. We denote the eight corners $\{0,1\}^3$ of P by $\{P_i\}_{i=1}^8$, and locate the eight corner points $\{\bar{v}_{c_i}\}_{i=1}^8$ by

$$c_i = \underset{j}{\operatorname{argmax}}\langle \overrightarrow{A\bar{v}_j}, \overrightarrow{AP_i}\rangle.$$

c_i will not be updated in training, and we define

$$L_{cor} = \frac{1}{8}\sum_{i=1}^{8}\|\bar{v}_{c_i} - P_i\|^2.$$

After the third optimization stage, corners are accurately mapped, i.e. $\bar{v}_{c_i} = P_i, i = 1,\ldots,8$. As shown in Fig. 6, we use the plane $x+y=1$ to cut the edge between corner points $P_2 = (1,0,0)$ and $P_6 = (1,0,1)$. The cut facets are projected onto the plane $x-y=1$, and the plane $x+y=1$ is projected to be a line l. Here we mark the projected points by $\{\tilde{v}_i\}$, and obtain a list of points $\tilde{e}_k = \alpha_k\tilde{v}_{i_k} + (1-\alpha_k)\tilde{v}_{j_k}, k = 1,\ldots,K$ on the line l. Then we obtain the corresponding points $e_k = \alpha_k v_{i_k} + (1-\alpha_k)v_{j_k}, k = 1,\ldots,K$ on the mesh (V,T) of boundary ∂C. Here we define $\bar{e}_k = G(e_k)$, $l_k = \frac{\sum_{i=2}^{k}\|\bar{e}_i-\bar{e}_{i-1}\|}{\sum_{i=2}^{K}\|\bar{e}_i-\bar{e}_{i-1}\|}$, then define $\hat{e}_k = l_k P_6 + (1-l_k)P_2$ as the target point of $\bar{e}_k$ on $\overrightarrow{P_2P_6}$. It is worth noting that e_k and $\hat{e}_k$ will not be updated in training. The loss function term L^1_{edge} of edge $\overrightarrow{P_2P_6}$ can be defined by

$$L^1_{edge} = \frac{1}{K}\sum_{i=1}^{K}\|\bar{e}_i - \hat{e}_i\|^2 + \frac{\lambda}{K-1}\sum_{i=1}^{K-1}\left\|\frac{\overrightarrow{\bar{e}_i\bar{e}_{i+1}}}{\|\overrightarrow{\bar{e}_i\bar{e}_{i+1}}\|} - \frac{\overrightarrow{\hat{e}_i\hat{e}_{i+1}}}{\|\overrightarrow{\hat{e}_i\hat{e}_{i+1}}\|}\right\|^2,$$

where λ is a constant which we take as 10^{-3}.

The remaining 11 edges and corresponding loss function terms $L^i_{edge}, i = 2,\ldots,12$ are obtained similarly and shown in Fig. 6. The edge mapping loss L_{edge} is defined as

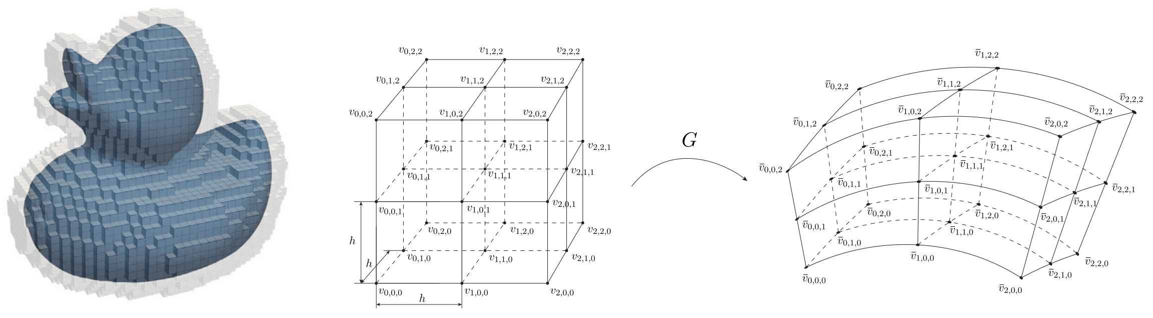
$$L_{edge} = \frac{1}{12}\sum_{i=1}^{12}L^i_{edge}.$$

4.4.2. Interior loss

To calculate the interior loss, which includes L_{arap} and L_{fair}, we first use a grid to discretize the domain C into small cubes as shown in Fig. 7(a). Each small cube has a size $h\times h\times h$ and the cubes covers the domain C. In practice, we set $h = 0.04$ such that there are about $\frac{1}{h^3} = 15625$ sub-cubes inside the domain C. Here the covered region is slightly larger than C, because we will move the sub-cubes randomly by a vector $\vec{m} \in [-h/2,h/2]^3$ in each training iteration to avoid overfitting. Also, we let the sub-cubes axis-aligned to simplify the calculation of Jacobian matrix and Hessian matrix. We denote the grid points–each of which are surrounded by eight sub-cubes by $Q = \{q_1,\ldots,q_R\}$.

Fig. 7(b) shows the local network mapping around a grid point $q_r \in Q$. We denote the 27 grid points in the $2\times2\times2$ cubes as $\{v_{i,j,k}\}_{i,j,k\in\{0,1,2\}}$, where $v_{i,j,k} = q_r + ((i-1)h,(j-1)h,(k-1)h)$, and the mapped vertices as $\{\bar{v}_{i,j,k}\}$. For a grid point q_r, we denote the discrete Jacobian matrix by J and the discrete Hessian matrices by H_1,H_2,H_3, where $H_k = (H^k_{ij})_{3\times3}, k = 1,2,3$. Let $H = (\overrightarrow{H_{ij}})_{3\times3}$ where $\overrightarrow{H_{ij}} = (H^1_{ij},H^2_{ij},H^3_{ij})^T$, then we have

$$J = \frac{1}{2h}\left(\bar{v}_{2,1,1} - \bar{v}_{0,1,1}, \bar{v}_{1,2,1} - \bar{v}_{1,0,1}, \bar{v}_{1,1,2} - \bar{v}_{1,1,0}\right)$$



(a) Sampling (b) The local network mapping G over 8 cubes.

Fig. 7. Sampling to calculate the interior loss.

$$
\begin{aligned}
\overrightarrow{H_{11}} &= \frac{1}{h^2}(\bar{v}_{2,1,1} - 2\bar{v}_{1,1,1} + \bar{v}_{0,1,1}) \\
\overrightarrow{H_{22}} &= \frac{1}{h^2}(\bar{v}_{1,2,1} - 2\bar{v}_{1,1,1} + \bar{v}_{1,0,1}) \\
\overrightarrow{H_{33}} &= \frac{1}{h^2}(\bar{v}_{1,1,2} - 2\bar{v}_{1,1,1} + \bar{v}_{1,1,0}) \\
\overrightarrow{H_{12}} &= \overrightarrow{H_{21}} = \frac{1}{4h^2}(\bar{v}_{2,2,1} - \bar{v}_{2,0,1} - \bar{v}_{0,2,1} + \bar{v}_{0,0,1}) \\
\overrightarrow{H_{13}} &= \overrightarrow{H_{31}} = \frac{1}{4h^2}(\bar{v}_{2,1,2} - \bar{v}_{2,1,0} - \bar{v}_{0,1,2} + \bar{v}_{0,1,0}) \\
\overrightarrow{H_{23}} &= \overrightarrow{H_{32}} = \frac{1}{4h^2}(\bar{v}_{1,2,2} - \bar{v}_{1,2,0} - \bar{v}_{1,0,2} + \bar{v}_{1,0,0})
\end{aligned}
$$

The singular values of J are $(\sigma_1, \sigma_2, \sigma_3)$, then we define the bijectivity loss and ARAP loss term by

$$L^r_{bij} = \left(\alpha - \sigma_1\sigma_2\sigma_3\right)_+$$

and

$$L^r_{arap} = \sigma_1\sigma_2\sigma_3 \sum_{d=1}^{3} \left(\frac{1}{\sigma_d} - 1\right)^2.$$

L^r_{bij} ensures that $\det(J) = \sigma_1\sigma_2\sigma_3 > \alpha$, and we set $\alpha = 0.1$ in our experiment. L^r_{arap} evaluate the ARAP energy of the forward parameterization G^{-1} at point $\bar{v}_{1,1,1}$.

The fairness loss term is defined by

$$L^r_{fair} = \|H_1\|_F^2 + \|H_2\|_F^2 + \|H_3\|_F^2,$$

where $\|\cdot\|_F$ refers to the Frobenius norm.

Then the overall loss term L_{bij}, L_{arap} and L_{hes} are defined by

$$L_{bij} = \frac{1}{R}\sum_{i=1}^{R} L^i_{bij},$$

$$L_{arap} = \frac{1}{R}\sum_{i=1}^{R} L^i_{arap},$$

$$L_{fair} = \frac{1}{R}\sum_{i=1}^{R} L^i_{fair}.$$

4.4.3. Choice of weights

The choice of the weights in the loss function is summarized as follows. Firstly, without loss of generality, we set the weight $w_{arap} = 1$. The weight w_{fair} controls the smoothness of the mapping, and a too large value of w_{fair} may lead to large average distortion. Based on experimental experience, we choose the proper weight $w_{fair} = 2 \times 10^{-3}$. The bijectivity and star-shaped boundary are controlled by weights w_{bij} and w_{star}. These properties have to be ensured, so we use large weights $w_{bij} = 10^6$ and $w_{star} = 10^3$. The terms $L_{dist}, L_{norm}, L_{cor}, L_{edge}$ control the boundary mapping error. Based on our experiments, we choose $w_{norm} = 10, w_{cor} = 10^3, w_{edge} = 10^4$. In the first and second stages of training, we use a smaller weight $w_{dist} = 10^3$ to optimize an appropriate boundary correspondence. In the third and fourth stages, we increase the weight w_{dist} to 10^4 to reduce the boundary mapping error. Table 1 lists the weights corresponding to the terms in the loss function.

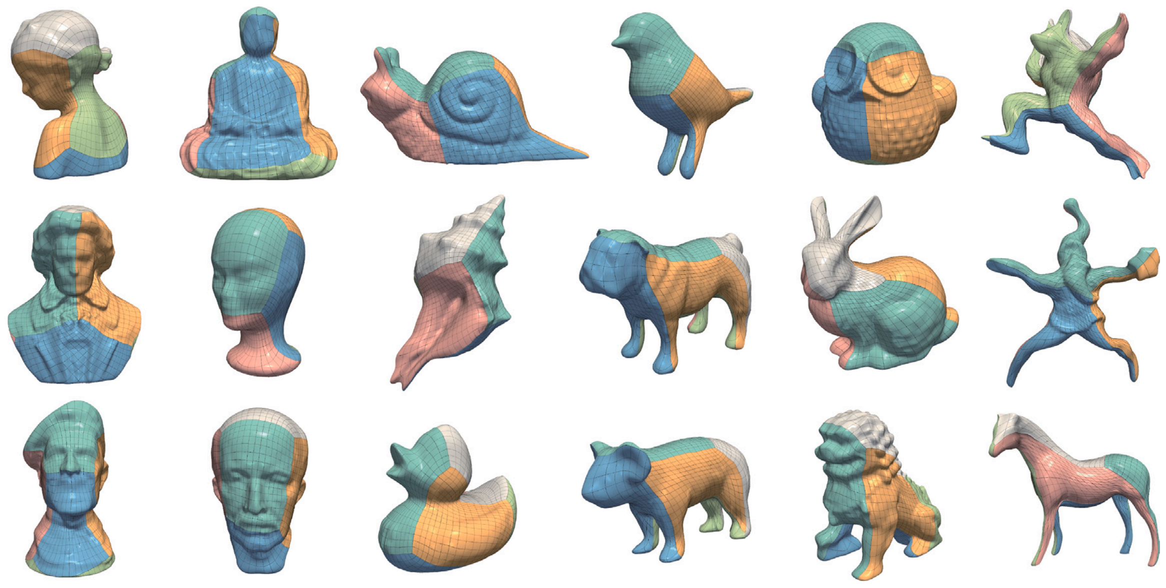


Fig. 8. Overview of our parameterization results. Different colors mark the different mapped faces of the parametric domain $[0,1]^3$.

5. Experiments and evaluation

In this section, we will apply our method on a set of computational domains and compare our results with some other parameterization methods, including internal parameterization methods [20,21,24] and integral parameterization method [6]. An example is also provided to illustrate the effectiveness of our method in IGA applications.

5.1. Implementation details

Our approach is implemented in python, using the PyTorch framework and Cuda acceleration. The experiments are conducted on a machine equipped with an Intel i7-12700F CPU and an Nvidia GTX 3080 Ti GPU. Adam optimizer is used in the training process, and we set the learning rate $lr = 5 \times 10^{-4}$. Also, to accelerate the 3×3 SVD, we use the "Fast CUDA 3×3 SVD" program in [36].

5.2. Quality metrics and experimental results

To evaluate the parameterization quality and compare with other methods, we use the bijectivity, ARAP distortion and the boundary error as the metrics. In the visualization of the ARAP distortion, we use $tanh(\frac{1}{2}\sqrt{arap})$ to ensure good display and to ensure the values lie in $(0,1)$. We apply our method on several computational domains and an overview is shown in Fig. 8. We also apply our method on the benchmark 3D dataset G1 provided by Cherchi and Livesu [37] and list the quantitive result in Appendix B. Our neural network generates bijective results for all the 25 models in the dataset. On the dataset provided by Liu et al. [6] which contains 215 models, our neural network achieves 209 bijective inverse mapping. In comparison, the methods in Su et al. [20],Zheng et al. [24] and Liu et al. [6] achieve $74, 116, 173$ bijective parameterizations, respectively.

5.3. Comparisons

Comparison with internal parameterization methods. First we compare our method with three traditional internal parameterization methods [20,21,24], over the models "duck", "koala" and "bust2". For the internal parameterization methods [20,21,24], boundary mapping are manually designed. In Figs. 9 to 11, we visualize the parameterization results from different perspectives and cut the parametric domain $[0,1]^3$ off to visualize the interior parameterization. The detailed metrics are listed in Table 2, and we observe that our method achieves both smaller boundary error and lower ARAP distortion. Fig. 9 shows the result of volumetric parameterization of the "duck" model by the three methods. Our method achieves smaller distortion and preserves more surface details. Fig. 10 shows the result of model "koala". In this model, our method selects similar corner correspondences and achieves lowest average ARAP distortion. For the model "bust2" which is shown in Fig. 11, our method chooses more reasonable corner correspondence and attains much better parameterization quality.

Table 2
Comparison with Su et al. [20],Ji et al. [21] and Zheng et al. [24], the best metrics are marked in bold.

Model	Method	Bijectivity	$ARAP_{mean}$	$ARAP_{max}$	Boundary
duck	Su et al. [20]	Y	2.842	54.795	0.00828
	Zheng et al. [24]	Y	2.879	80.979	0.00828
	Ji et al. [21]	Y	2.880	54.868	0.00824
	Ours	Y	**1.984**	**41.877**	**0.00184**
koala	Su et al. [20]	Y	4.114	59.303	0.00221
	Zheng et al. [24]	Y	4.131	76.295	0.00197
	Ji et al. [21]	Y	4.161	**31.136**	0.00237
	Ours	Y	**2.745**	65.879	**0.00193**
bust2	Su et al. [20]	Y	5.061	30.986	0.00390
	Zheng et al. [24]	Y	5.024	66.626	0.00304
	Ji et al. [21]	N	5.109	425.86	0.00364
	Ours	Y	**1.926**	**24.466**	**0.00234**

Table 3
Comparison with Liu et al. [6], the best metrics are marked in bold.

Model	Method	Bijectivity	$ARAP_{mean}$	$ARAP_{max}$	Boundary
bird	Liu et al. [6]	Y	2.549	91.321	0.00245
	Ours	Y	**1.730**	**42.346**	**0.00109**
cartoon3	Liu et al. [6]	Y	**1.136**	13.053	0.00124
	Ours	Y	1.399	**11.110**	**0.00120**
conch	Liu et al. [6]	Y	2.470	96.124	0.00738
	Ours	Y	**1.468**	**50.818**	**0.00169**
horse	Liu et al. [6]	N	7.052	199.418	0.01053
	Ours	Y	**4.340**	**128.453**	**0.00349**
santa4	Liu et al. [6]	Y	5.656	100.542	0.00636
	Ours	Y	**3.936**	**95.842**	**0.00527**

Comparison with integral parameterization methods. Now we compare our method with the integral parameterization method [6] over five models “bird”, “cartoon3”, “conch”, “horse” and “santa4”. The results are visualized in Fig. 12, and the metrics are shown in Table 3. For the model “bird”, Liu’s method gives a more symmetric boundary correspondence. However, from the picture we can find that our asymmetric parameterization leads to lower distortion. In the sectional views, we observe that the asymmetry makes the sectional surface is closer to a rectangle, and this may explain why we achieve lower distortion. For the model “cartoon3”, different corner correspondences are selected. Since the model is approximately spherical, both corner correspondences are reasonable. In this model, Liu’s method generates the parameterization with smaller average ARAP distortion, larger maximal ARAP distortion and bigger boundary error. For the model “conch”, our method chooses much more reasonable corner correspondence and thus achieve much lower distortion. For the model “horse”, Liu’s method fails to generate bijective parameterization, while our method achieve bijective parameterization with much lower distortion and boundary error. Finally, for the model “santas4”, both methods choose reasonable corner correspondence and generate bijective parameterizations while our method offers smaller distortion.

Comparison of computational time. Tables 4 and B.6 present the computational time of our algorithm. As demonstrated by the results, our method parameterizes each computational domain in about 4 min, regardless of model complexity. On the other hand, the traditional interior parameterization methods [20,21,24] require approximately 10 s for simple models (e.g., “duck”) but exceed 200 s for complex geometries (e.g., “koala”). For extremely complex models (e.g., “santa4”), these methods demand over 10 min. The integral parameterization method proposed by Liu et al. [6] requires substantially longer computational time, ranging from 10 to 40 min per model. Notice that all the timings are extracted from the original publications [6,20,21,24], with the caveat that they were obtained under varying hardware configurations.

5.4. Parameterization with specified corner points

Our method automatically selects eight corner points on the input model, and in most cases, it provides reasonable correspondence. However, in some instances, either due to practical needs or because the automatically selected corner points are not suitable, we need to specify the corner mapping. In this case, we only need to include the loss term L_{cor} in the first and second stages of network training, with a weight of $w_{cor} = 10^2$. The rest of the algorithm are kept unchanged. Fig. 13 shows examples over the “duck” and “bust2” models with and without specified corner points, the corresponding quantitive results are listed in Table 5. It can be observed that, our algorithm generally chooses corner points which lead to lower ARAP energy.

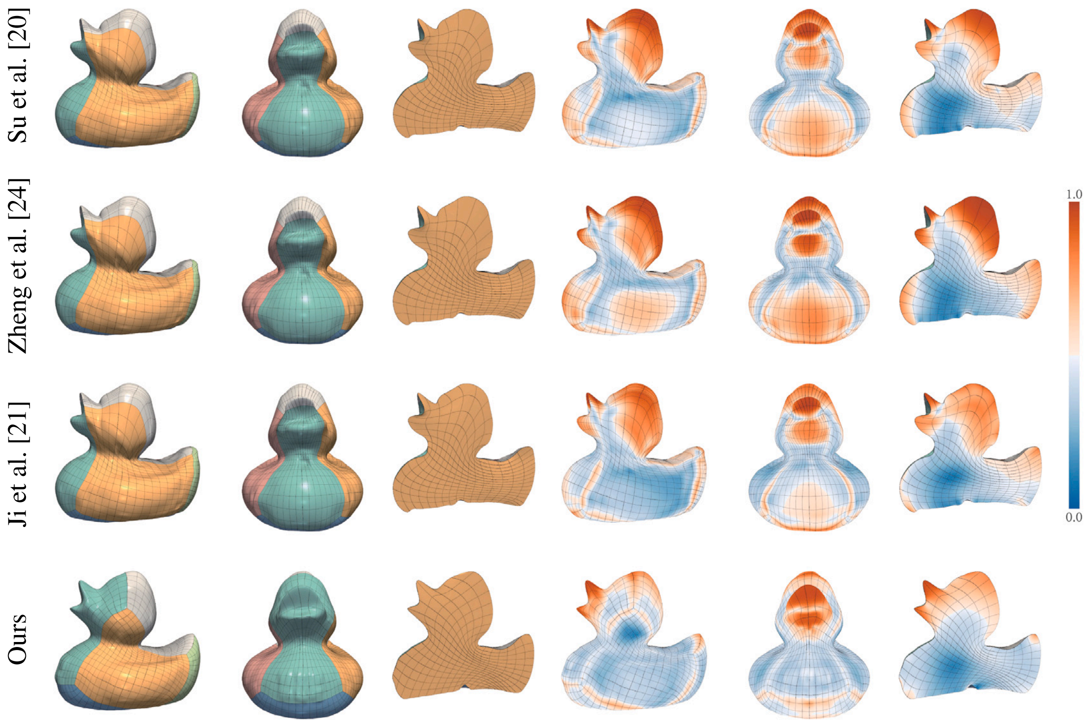


Fig. 9. Comparison with Su et al. [20],Zheng et al. [24] and Ji et al. [21] over the model "duck", the distortion is shown on the right.

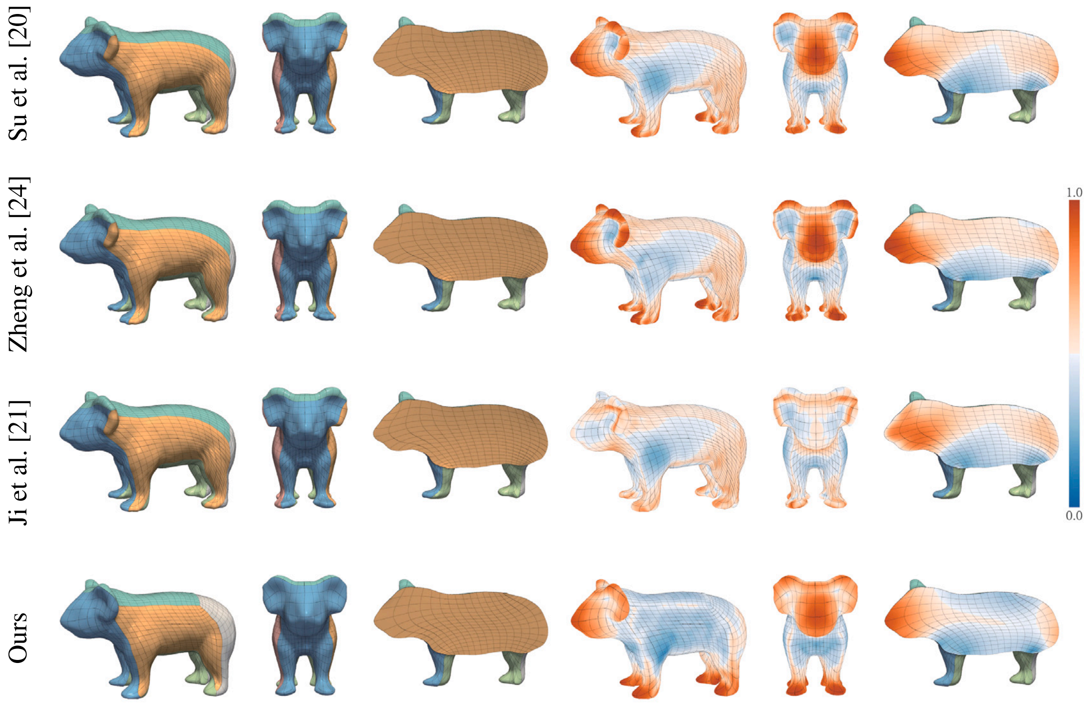


Fig. 10. Comparison with Su et al. [20],Zheng et al. [24] and Ji et al. [21] over the model "koala", the distortion is shown on the right.

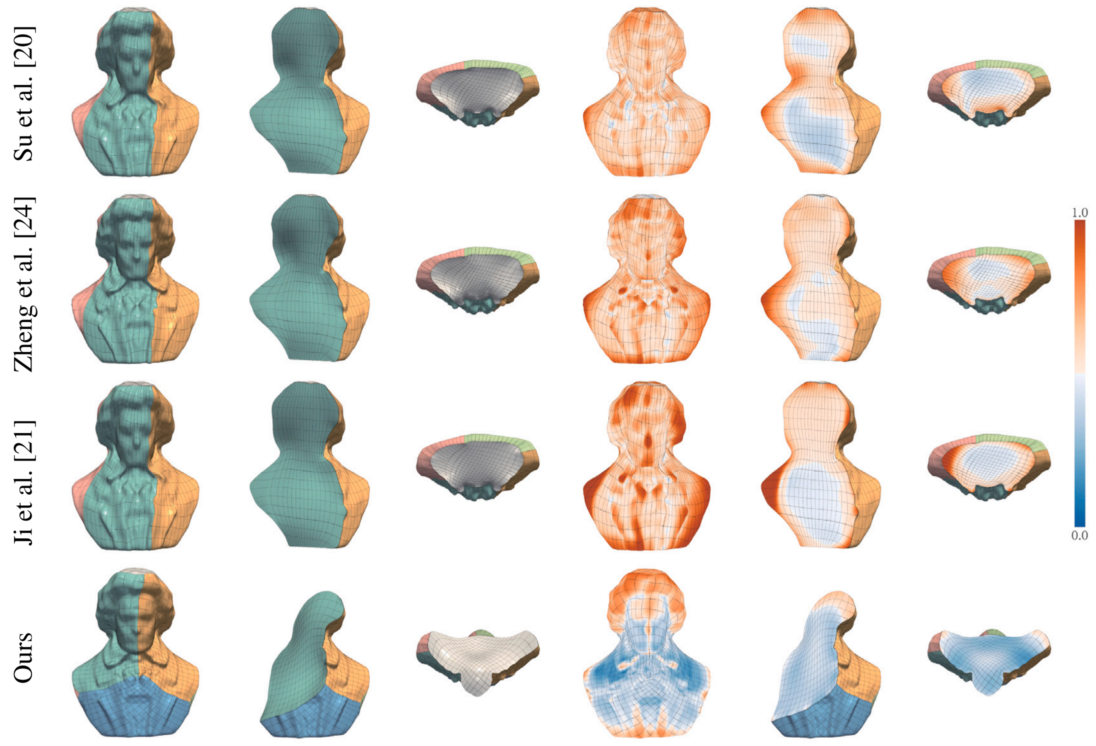


Fig. 11. Comparison with Su et al. [20],Zheng et al. [24] and Ji et al. [21] over the model "bust2", the distortion is shown on the right.

Table 4
Computational time of our method. t_{train} and t_{fit} represent the time for network training and spline fitting, respectively.

Model	duck	koala	bust2	bird	cartoon3	conch	horse	santa4
#Vertex	12 762	19 069	19 271	18 790	19 092	15 496	19 263	13 341
t_{train}(s)	186.98	186.08	186.30	177.91	184.68	182.38	190.73	194.23
t_{fit}(s)	39.68	38.51	38.69	38.99	38.53	39.02	39.71	38.71
t_{total}(s)	226.66	224.58	224.99	216.90	223.22	221.40	230.44	232.93

Table 5
Parameterizations of the "duck" and "bust2" models with and without specified points.

Model		Bijectivity	$ARAP_{mean}$	$ARAP_{max}$	Boundary
duck	Automatic	Y	**1.983914**	**41.87654**	0.001839
	Specified corners	Y	2.712664	79.03929	**0.001461**
bust2	Automatic	Y	**1.925911**	**24.46535**	0.002337
	Specified corners	Y	3.317379	67.37399	**0.002216**

5.5. Application in IGA simulation

In this subsection, we apply our domain parameterization results to solve three-dimensional PDEs, validating their effectiveness for IGA simulation. As an example, we solve the Poisson's equation with Dirichlet boundary condition over the "conch" and "santa4" models. The Poisson's equation is as follows:

$$\begin{cases} -\Delta u = f & in \;\; C \\ u = g & on \;\; \partial C, \end{cases} \tag{4}$$

where C is the computational domain. In our experiment, we set the exact solution u to be $u(x,y,z) = e^{x+z}\sin(y)$, and set f to be $f(x,y,z) = -\Delta u(x,y,z) = -e^{x+z}\sin(y)$. The error distribution is shown in Fig. 14. We also compare our method with the method of Liu et al. [6] for solving the above equation. It can be observed that, our parameterizations result in smaller analysis error, which demonstrates that our algorithm is more suitable for IGA simulation.

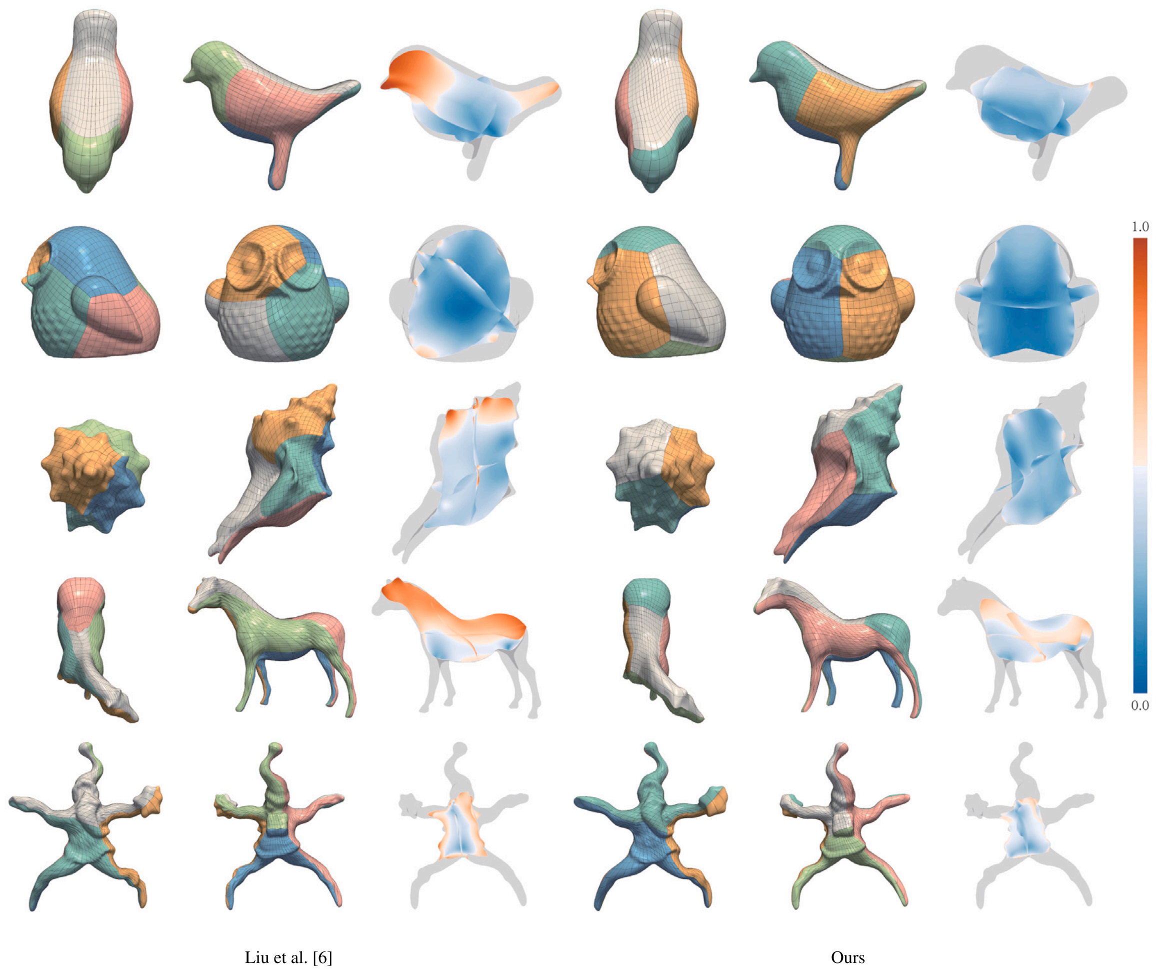


Fig. 12. Comparison with Liu et al. [6] over models “bird”, “cartoon3”, “conch”, “horse” and “santa4”. Let $P_x = \{(x, y, z)|x = \frac{1}{2}\} \cap P$, $P_y = \{(x, y, z)|y = \frac{1}{2}\} \cap P$, $P_z = \{(x, y, z)|z = \frac{1}{2}\} \cap P$, the distortion on the surfaces $G(P_x), G(P_y), G(P_z)$ is visualized in the figure.

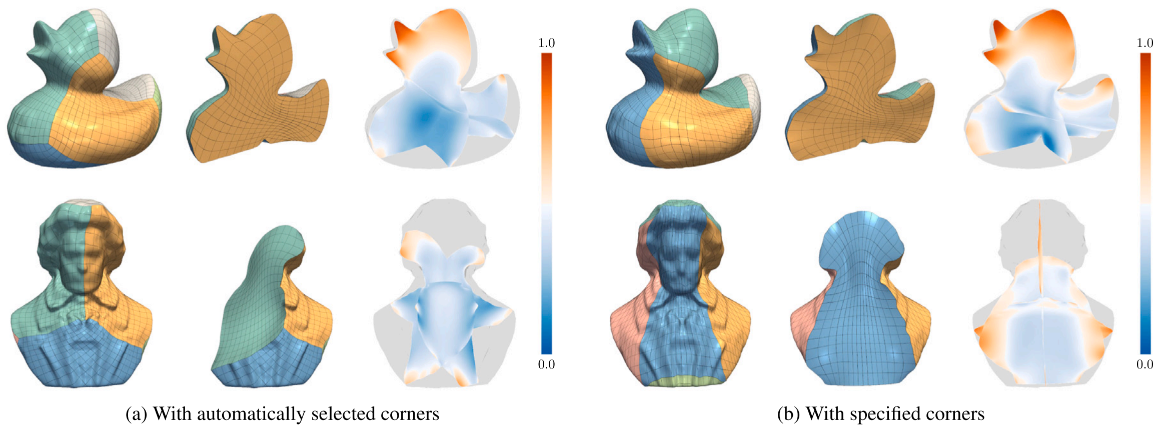

1.0
0.0
(a) With automatically selected corners
1.0
0.0
(b) With specified corners

Fig. 13. Parameterizations of the “duck” and “bust2” models with and without specified corner points. The ARAP distortion is shown in the figure.

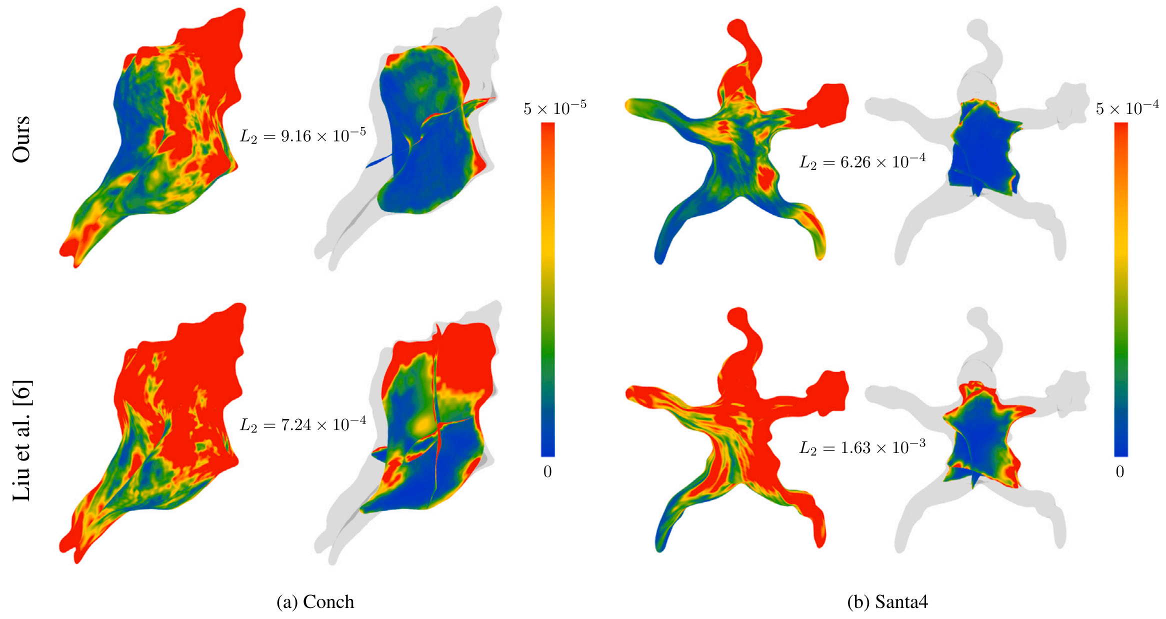

Ours
Liu et al. [6]
$L_2 = 9.16 \times 10^{-5}$
$L_2 = 7.24 \times 10^{-4}$
5×10^{-5}
0
$L_2 = 6.26 \times 10^{-4}$
$L_2 = 1.63 \times 10^{-3}$
5×10^{-4}
0
(a) Conch
(b) Santa4

Fig. 14. Error distribution in solving Poisson's equation on models "conch" and "santa4". The top row shows the result by our method and the below shows the result by Liu et al. [6]. The L_2 errors over the computational domains are marked in the figure.

Table B.6
Quantitive results on benchmark dataset G1 [37]. The models are scaled to have volume 1.

Model	Bijectivity	$ARAP_{mean}$	$ARAP_{max}$	Boundary	Time (s)
bimba	Y	0.928	15.921	0.000708	232.48
bird	Y	2.744	37.446	0.000961	232.34
blade	Y	1.569	28.229	0.001498	239.61
bone	Y	2.303	22.687	0.009318	227.24
bumpy_sphere	Y	0.322	6.283	0.000704	218.53
bunny	Y	0.835	47.179	0.000782	233.19
buste	Y	1.454	23.202	0.001353	228.05
camile_hand	Y	1.390	129.299	0.001271	238.24
chinese_dragon	Y	0.896	35.364	0.002506	243.35
david	Y	0.609	17.917	0.002063	234.73
devil	Y	0.600	62.906	0.000733	230.46
dilo	Y	2.660	184.823	0.002697	226.08
duck	Y	0.341	13.269	0.000846	226.50
foot	Y	0.799	18.873	0.000396	225.27
gargoyle	Y	1.274	36.548	0.001653	244.07
hand	Y	1.621	65.612	0.001303	231.38
homer	Y	1.799	69.896	0.001726	233.32
isidora_horse	Y	1.353	46.715	0.001409	235.39
lion	Y	2.839	546.323	0.048660	237.59
moai	Y	0.903	9.648	0.000746	231.29
mouse	Y	0.567	24.789	0.000716	233.80
octa_flower	Y	0.610	39.612	0.001494	224.43
pig	Y	0.776	42.099	0.000565	225.70
ramses	Y	1.741	23.986	0.001741	230.52
sphinx	Y	0.794	13.102	0.000902	230.25

6. Conclusion

In this paper, we propose a neural network based algorithm for volumetric integral parameterization. In the optimization, the boundary correspondence and the interior mapping are treated simultaneously. To obtain proper boundary correspondence, we optimize the inverse parameterization and the smooth cuboid is employed to avoid local optima. The loss function terms of bijectivity and star-shape eliminate flipping points, and the ARAP distortion and a fairness term are optimized to ensure high-quality interior mapping. The experiment shows that our method produces bijective and high quality parameterizations, and outperforms state-of-the-art methods in the comparisons.

There are some limitations in the current work. First, due to the complexity of three dimensional domain parameterization, our network lacks the generalization ability. As a result, we need to train the network separately for each computational domain. Fortunately, our algorithm can complete the parameterization for most models within four minutes regardless the complexity of the geometric models (Tables 4, B.6), which is acceptable for practical applications and faster than many optimization-based methods. A future research target is to generalize the neural network to avoid repeated network training. Second, the method only works for single block parameterization for computational domain with zero genus. A future research is to properly decompose the complex computational domains or use the poly-cube as the parametric domain.

CRediT authorship contribution statement

Zheng Zhan: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation. **Wenping Wang:** Writing – review & editing, Conceptualization. **Falai Chen:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the Key Project of the National Natural Science Foundation of China (No. 12494550 and 12494555), and the National Natural Science Foundation of China (No. 12371383).

Appendix A. Conditions for a star-shaped domain

In order to generate a bijective inverse map G, we want the triangle mesh $(G(V), G(T))$ to form a star-shaped domain with the center $A = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. The following lemma gives a condition for a star-shaped domain.

Lemma 1. *Let C be a simply-connected volumetric domain bounded by a triangle mesh (V, T) and A be an inner point of C. For each facet $t \in T$, we denote its outer norm by $\overrightarrow{n_t}$. Then the following two conditions are equivalent:*

(1) C is a star-shaped domain with the center A.
(2) $\forall t \in T, \exists v \in l_t$ s.t. $\langle \overrightarrow{Av}, \overrightarrow{n_t} \rangle \geq 0$.

Proof. For each facet t, we denote the plane where t lies on by l_t. It divides $\mathbb{R}^3$ into two half-spaces: L_t^+–to which the normal vector $\overrightarrow{n_t}$ points from l_t, and L_t^-–the opposite half-space of L_t^+. Then, $A \in L_t^-$ if and only if $\exists v \in l_t$ such that $\langle \overrightarrow{Av}, \overrightarrow{n_t} \rangle \geq 0$. Define the kernel of the polyhedra C by $\ker(C) = \cap_{t \in T} L_t^-$, then $A \in \ker(C)$ if and only if the condition (2) holds. On the other hand, A is the center of the star-shaped domain C if and only if $A \in \ker(C)$ [38]. Therefore, $(1) \Leftrightarrow (2)$. □

Appendix B. Quantitive result on benchmark dataset G1

We tested our method on the benchmark dataset G1 [37] and present the results in Table B.6. Bijective inverse parameterization mappings, represented by the network, are successfully obtained for all 25 models.

Data availability

Data will be made available on request.

References

[1] T.J.R. Hughes, J.A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Comput. Methods Appl. Mech. Engrg. 194 (39) (2005) 4135–4195, http://dx.doi.org/10.1016/j.cma.2004.10.008.
[2] J.A. Cottrell, T.J.R. Hughes, Y. Bazilevs, Isogeometric Analysis: Toward Integration of CAD and FEA, first ed., Wiley Publishing, 2009.
[3] Y. Zheng, M. Pan, F. Chen, Boundary correspondence of planar domains for isogeometric analysis based on optimal mass transport, Comput.- Aided Des. 114 (2019) 28–36, http://dx.doi.org/10.1016/j.cad.2019.04.008.
[4] Y. Zheng, F. Chen, Volumetric boundary correspondence for isogeometric analysis based on unbalanced optimal transport, Comput.- Aided Des. 140 (2021) 103078, http://dx.doi.org/10.1016/j.cad.2021.103078.
[5] Z. Zhan, Y. Zheng, W. Wang, F. Chen, Boundary correspondence for iso-geometric analysis based on deep learning, Commun. Math. Stat. 11 (1) (2023) 131–150, http://dx.doi.org/10.1007/s40304-023-00337-7.
[6] H. Liu, Y. Yang, Y. Liu, X.-M. Fu, Simultaneous interior and boundary optimization of volumetric domain parameterizations for IGA, Comput. Aided Geom. Design 79 (2020) 101853, http://dx.doi.org/10.1016/j.cagd.2020.101853.

[7] Z. Zhan, W. Wang, F. Chen, Simultaneous boundary and interior parameterization of planar domains via deep learning, Comput.- Aided Des. 166 (2024) 103621, http://dx.doi.org/10.1016/j.cad.2023.103621.

[8] Z. Zhan, W. Wang, F. Chen, Fast parameterization of planar domains for isogeometric analysis via generalization of deep neural network, Comput. Aided Geom. Design 111 (2024) 102313, http://dx.doi.org/10.1016/j.cagd.2024.102313.

[9] M. Pan, F. Chen, W. Tong, Low-rank parameterization of planar domains for isogeometric analysis, Comput. Aided Geom. Design 63 (2018) 1–16, http://dx.doi.org/10.1016/j.cagd.2018.04.002.

[10] X. Nian, F. Chen, Planar domain parameterization for isogeometric analysis based on Teichmüller Mapping, Comput. Methods Appl. Mech. Engrg. 311 (2016) 41–55, http://dx.doi.org/10.1016/j.cma.2016.07.035.

[11] Y. Ji, K. Chen, M. Möller, C. Vuik, On an improved PDE-based elliptic parameterization method for isogeometric analysis using preconditioned Anderson acceleration, Comput. Aided Geom. Design 102 (2023) 102191, http://dx.doi.org/10.1016/j.cagd.2023.102191.

[12] A. Falini, G.A. D'Inverno, M.L. Sampoli, F. Mazzia, Splines parameterization of planar domains by physics-informed neural networks, Mathematics 11 (10) (2023) 2406, http://dx.doi.org/10.3390/math11102406.

[13] G. Farin, D. Hansford, Discrete coons patches, Comput. Aided Geom. Design 16 (7) (1999) 691–700, http://dx.doi.org/10.1016/S0167-8396(99)00031-X.

[14] G. Xu, B. Mourrain, R. Duvigneau, A. Galligo, Analysis-suitable volume parameterization of multi-block computational domain in isogeometric applications, Comput.- Aided Des. 45 (2) (2013) 395–404, http://dx.doi.org/10.1016/j.cad.2012.10.022.

[15] A. Falini, J. Špeh, B. Jüttler, Planar domain parameterization with THB-Splines, Comput. Aided Geom. Design 35–36 (2015) 95–108, http://dx.doi.org/10.1016/j.cagd.2015.03.014.

[16] X. Gu, Y. Wang, S.-T. Yau, Volumetric harmonic map, Commun. Inf. Syst. 3 (3) (2003) 191–202, http://dx.doi.org/10.4310/CIS.2003.v3.n3.a4.

[17] T. Nguyen, B. Jüttler, Parameterization of contractible domains using sequences of harmonic maps, in: Curves and Surfaces, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 501–514, http://dx.doi.org/10.1007/978-3-642-27413-8_32.

[18] M. Pan, F. Chen, W. Tong, Volumetric spline parameterization for isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 359 (2020) 112769, http://dx.doi.org/10.1016/j.cma.2019.112769.

[19] X. Wang, X. Qian, An optimization approach for constructing trivariate -spline solids, Comput.- Aided Des. 46 (2014) 179–191, http://dx.doi.org/10.1016/j.cad.2013.08.030.

[20] J.-P. Su, X.-M. Fu, L. Liu, Practical foldover-free volumetric mapping construction, Comput. Graph. Forum 38 (7) (2019) 287–297, http://dx.doi.org/10.1111/cgf.13837.

[21] Y. Ji, M.-Y. Wang, M.-D. Pan, Y. Zhang, C.-G. Zhu, Penalty function-based volumetric parameterization method for isogeometric analysis, Comput. Aided Geom. Design 94 (2022) 102081, http://dx.doi.org/10.1016/j.cagd.2022.102081.

[22] J.M. Escobar, J.M. Cascón, E. Rodríguez, R. Montenegro, A new approach to solid modeling with trivariate T-Splines based on mesh optimization, Comput. Methods Appl. Mech. Engrg. 200 (45) (2011) 3210–3222, http://dx.doi.org/10.1016/j.cma.2011.07.004.

[23] Y. Zhang, W. Wang, T.J.R. Hughes, Solid T-Spline construction from boundary representations for genus-zero geometry, Comput. Methods Appl. Mech. Engrg. 249–252 (2012) 185–197, http://dx.doi.org/10.1016/j.cma.2012.01.014.

[24] Y. Zheng, F. Chen, et al., Volumetric parameterization with truncated hierarchical B-splines for isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 401 (2022) 115662, http://dx.doi.org/10.1016/j.cma.2022.115662.

[25] L. Liu, Y. Zhang, Y. Liu, W. Wang, Feature-preserving T-mesh construction using skeleton-based polycubes, Comput.- Aided Des. 58 (2015) 162–172, http://dx.doi.org/10.1016/j.cad.2014.08.020.

[26] G. Xu, T.-H. Kwok, C.C.L. Wang, Isogeometric computation reuse method for complex objects with topology-consistent volumetric parameterization, Comput.- Aided Des. 91 (2017) 1–13, http://dx.doi.org/10.1016/j.cad.2017.04.002.

[27] L. Chen, G. Xu, S. Wang, Z. Shi, J. Huang, Constructing volumetric parameterization based on directed graph simplification of ℓ 1 Polycube Structure from complex shapes, Comput. Methods Appl. Mech. Engrg. 351 (2019) 422–440, http://dx.doi.org/10.1016/j.cma.2019.01.036.

[28] L. Morreale, N. Aigerman, V.G. Kim, N.J. Mitra, Neural surface maps, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 4639–4648.

[29] C. Giannelli, S. Imperatore, A. Mantzaflaris, F. Scholz, Learning meshless parameterization with graph convolutional neural networks, in: 2023 World Conference on Smart Trends in Systems, Security and Sustainability, London, United Kingdom, 2023.

[30] M. De Vita, C. Giannelli, S. Imperatore, A. Mantzaflaris, Parameterization learning with convolutional neural networks for gridded data fitting, in: Advances in Information and Communication, Springer Nature Switzerland, Cham, 2024, pp. 393–412, http://dx.doi.org/10.1007/978-3-031-53963-3_27.

[31] C. Giannelli, S. Imperatore, A. Mantzaflaris, F. Scholz, BIDGCN: Boundary-informed dynamic graph convolutional network for adaptive spline fitting of scattered data, Neural Comput. Appl. 36 (28) (2024) 17261–17284, http://dx.doi.org/10.1007/s00521-024-09997-0.

[32] Y. Yu, Y. Fang, H. Tong, Y.J. Zhang, DL-Polycube: Deep learning enhanced polycube method for high-quality hexahedral mesh generation and volumetric spline construction, 2024, http://dx.doi.org/10.48550/arXiv.2410.18852, arXiv:2410.18852.

[33] L. Liu, L. Zhang, Y. Xu, C. Gotsman, S.J. Gortler, A Local/Global approach to mesh parameterization, Comput. Graph. Forum 27 (5) (2008) 1495–1504, http://dx.doi.org/10.1111/j.1467-8659.2008.01290.x.

[34] B. Mildenhall, P.P. Srinivasan, M. Tancik, J.T. Barron, R. Ramamoorthi, R. Ng, NeRF: Representing scenes as neural radiance fields for view synthesis, in: Computer Vision – ECCV 2020, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2020, pp. 405–421, http://dx.doi.org/10.1007/978-3-030-58452-8_24.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, 2023, http://dx.doi.org/10.48550/arXiv.1706.03762, arXiv:1706.03762.

[36] M. Gao*, X. Wang*, K. Wu*, A. Pradhana, E. Sifakis, C. Yuksel, C. Jiang, GPU optimization of material point methods, ACM Trans. Graph. (Proc. SIGGRAPH ASIA 2018) 37 (6) (2018) (*Joint First Authors).

[37] G. Cherchi, M. Livesu, VOLMAP: A large scale benchmark for volume mappings to simple base domains, Comput. Graph. Forum 42 (5) (2023) e14915, http://dx.doi.org/10.1111/cgf.14915.

[38] F.P. Preparata, M.I. Shamos, Intersections, in: Computational Geometry: An Introduction, Springer, New York, NY, 1985, pp. 266–322, http://dx.doi.org/10.1007/978-1-4612-1098-6_7.