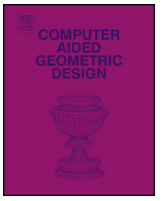
Comput. Aided Geom. Des. 111 (2024) 102313

Contents lists available at ScienceDirect

Computer Aided Geometric Design

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



Fast parameterization of planar domains for isogeometric analysis via generalization of deep neural network

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
Domain parameterization
Deep neural network

ABSTRACT

One prominent step in isogeometric analysis (IGA) is known as domain parameterization, that is, finding a parametric spline representation for a computational domain. Typically, domain parameterization is divided into two separate steps: identifying an appropriate boundary correspondence and then parameterizing the interior region. However, this separation significantly degrades the quality of the parameterization. To attain high-quality parameterization, it is necessary to optimize both the boundary correspondence and the interior mapping simultaneously, referred to as integral parameterization. In a prior research, an integral parameterization approach for planar domains based on neural networks was introduced. One limitation of this approach is that the neural network has no ability of generalization, that is, a network has to be trained to obtain a parameterization for each specific computational domain. In this article, we propose an efficient enhancement over this work, and we train a network which has the capacity of generalization—once the network is trained, a parameterization can be immediately obtained for each specific computational via evaluating the network. The new network greatly speeds up the parameterization process by two orders of magnitudes. We evaluate the performance of the new network on the MPEG data set and a self-design data set, and experimental results demonstrate the superiority of our algorithm compared to state-of-the-art parameterization methods.

1. Introduction

Isogeometric analysis (IGA) Hughes et al. (2005) is a promising approach for concurrent geometric design and physical simulations for geometric models. However, in the current CAD system, boundary representation is adopted and in order to simulate physical phenomena in a volume, one has to obtain the parametric representation of the volume. The process of computing the parametric representation of a domain from the boundary representation is known as domain parameterization. Domain parameterization involves establishing a mapping from a parametric domain to a specific computational domain, as depicted in Fig. 1. Traditional methods break down this task into two stages. First, an appropriate boundary correspondence is specified manually or through algorithms Zheng et al. (2019); Zheng and Chen (2021); Zhan et al. (2023). Then, interior parameterization is computed under the predefined boundary correspondence. Such methods are unable to simultaneously optimize both boundary correspondence and internal mapping, and they often fail to achieve globally optimal parameterization.

* 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.cagd.2024.102313

Available online 25 April 2024

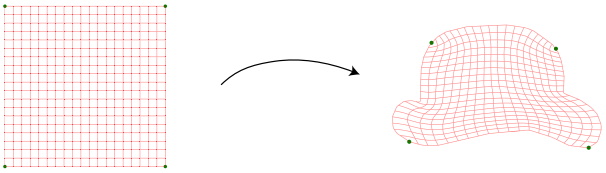


Fig. 1. A visualized example of planar domain parameterization.

To address this challenge, the concept of integral parameterization was introduced, aiming to optimize both boundary correspondences and internal mappings simultaneously Liu et al. (2020); Zhan et al. (2024). The approach presented by Liu et al. (2020) often converges to local optima, producing parameterizations with unsatisfactory boundary correspondences. In contrast, Zhan et al. (2024) utilizes neural networks to obtain integral parameterization with favorable boundary correspondences. Compared with previous parameterization methods Pan et al. (2018); Nian and Chen (2016); Ji et al. (2023); Falini et al. (2023); Liu et al. (2020), the neural network-based approach Zhan et al. (2024) produces superior parameterization results. However, a notable limitation of this approach is the necessity to retrain the neural network for each specific computational domain, that is, the network has no ability of generalization, which introduces significant computational overhead. In practice, fast parameterization is required in various scenarios, e.g., parameterization multiple patches of a computational domain at the same time, or parameterization of changing domains, etc.

In this paper, we propose an improved method to facilitate rapid and effective generation of high-quality parameterizations for planar domains. Similar to the prior research, we utilize a neural network to represent the inverse parameterization. The network consists of two parts. One part is used to extract the feature of the boundary of the computational domain, and the other part is a multi-layer perceptron (MLP) similar to the network in Zhan et al. (2024). The extracted feature is incorporated into the MLP network to include the information of each computational domain. As a result, the generalized network can be expressed as a mapping $G : \mathbb{R}^2 \times D \to \mathbb{R}^2, ((x, y), C) \mapsto (u, v)$, where D represents the set of computational domains and C is a computational domain in D. This mapping associates a point (x, y) in the computational domain $C \in D$, producing the corresponding parametric values (u, v) in the parametric domain. After the network is trained, it directly provides an inverse parameterization for any specified computational domain C. Subsequent post-processing and B-spline surface fitting lead to the forward parameterization. For each specific computational domain, the parameterization can be accomplished in approximately 0.1 seconds, which is one hundred times faster than the method in Zhan et al. (2024), and at the same time high quality parameterization is achieved.

The remainder of the paper is structured as follows. Section 2 presents a review of related works about domain parameterization. Section 3 details our algorithm, including network architecture design, defining the loss function, pre-processing, post-processing, etc. In Section 4, we conduct comparative analyses between our method and existing parameterization techniques, showcasing our experimental results. Finally, in Section 5 we conclude our work and outline potential directions for future research.

2. Related work

2.1. Internal parameterization methods

Internal parameterization methods are traditional and mainstream approaches, focusing on parameterizing the interior region based on given boundary mapping. Earlier methods in this category include techniques such as discrete Coons patches Farin and Hansford (1999) and spring models Gravesen et al. (2014). This type of method has minimal computational complexity, but often yields poorly behaved or even non-bijective parameterizations. Another class of linear methods relies on harmonic mappings, which reformulates the parameterization problem into solving partial differential equations (PDEs) with Dirichlet boundary conditions Ji et al. (2023); Nguyen and Jüttler (2012); Xu et al. (2013). While these linear methods are computationally efficient, they tend to produce suboptimal parameterizations. To achieve higher-quality parameterizations, various nonlinear optimization-based methods have been proposed Pan et al. (2018, 2020); Nian and Chen (2016); Wang and Qian (2014). These methods are computationally expensive, but often deliver high-quality bijective parameterizations. To further improve parameterization quality, locally refinable splines are used to represent the parametric equations on the computational domain Escobar et al. (2011); Zhang et al. (2012); Falini et al. (2015); Zheng and Chen (2022).

It's worth noting that all the aforementioned methods are primarily designed for simply connected domains. In cases of topologically complex domains, multi-patch parameterization becomes a preferred solution Xiao et al. (2018); Buchegger and Jüttler (2017); Xu et al. (2018); Pan et al. (2023). However, multi-patch parameterization introduces two new challenges: determining how to partition the domain effectively and ensuring global smoothness of parameterization.

2.2. Integral parameterization methods

The boundary correspondence significantly impacts the quality of parameterization Zheng et al. (2019); Zhan et al. (2023); Liu et al. (2020); Zhan et al. (2024), and thus there is a need for methods that handles both boundary correspondence and internal mapping simultaneously, referred to as integral parameterization methods. Currently, very few integral domain parameterization methods are devised (Liu et al. (2020) and Zhan et al. (2024)). Liu et al. (2020) iteratively optimizes the boundary correspondence

and the interior mapping in order to obtain an improved parameterization. This method enforces the boundary mapping error as a constraint during optimization. However, due to the strong boundary mapping constraint, the algorithm is often trapped in local minima, leading to poorly-behaved boundary correspondence. Zhan et al. (2024) utilizes a neural network to represent the inverse parameterization mapping, and a network is trained for each specific computational domain. In order to obtain a high quality parameterization, the loss function is designed to include three terms–angular and area distortion of the mapping, and the boundary fitting term. Compared to both internal parameterization methods and the method of Liu et al. (2020), the neural network approach achieves higher bijectivity ratio, lower distortion and more uniform parameterizations.

One major limitation of the neural network approach Zhan et al. (2024) is that, the neural network has to be retrained for each specific computational domain, resulting in substantial computational overhead. In this paper, we enhance the approach in Zhan et al. (2024) by training a network with the ability of generalization, that is, once the neural network is trained, a high-quality parameterization for any specific computational domain can be obtained by evaluating the network. Thus computational time can be greatly decreased.

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 parameterization problem Zhan et al. (2023, 2024); Giannelli et al. (2023); Falini et al. (2023). Below we briefly review these methods.

- Zhan et al. (2023) presented an approach that utilizes neural networks to rapidly generate boundary correspondence. The central component of the neural network is a U-Net, which takes discrete boundary information as its input and outputs the probability of each boundary point being a corner point.
- Giannelli et al. (2023) employed neural networks for surface parameterization of point clouds. The algorithm initially generates a graph structure on the point cloud and computes the corresponding line graph representation. Then, a graph convolutional network is utilized to obtain the weight of each edge in the parametric space, resulting in a meshless parameterization.
- Falini et al. (2023) transformed the planar domain parameterization problem into a set of partial differential equations and solved them using PINN neural networks. Unfortunately, this method is not suitable for complex computational domains.
- In contrast to the internal parameterization methods developed in Giannelli et al. (2023) and Falini et al. (2023), Zhan et al. (2024) introduced an integral parameterization approach through an MLP network. This method simultaneously optimizes the interior mapping and the boundary correspondence via training a network, and it can achieve high-quality parameterization for most computational domains.

3. Method

In this section, we present our method to enhance the neural network in Zhan et al. (2024) with generalization capacity. Our method can be broadly divided into two major steps: the first step involves training a neural network with generalization capacity to represent the inverse parameterization mapping, and the second step evaluates the trained neural network to obtain a parameterization for a specific computational domain. In the following, we describe the details of the two steps.

3.1. Problem formulation

As described before, our goal in this paper is to train a neural network such that, for any input of the boundary of a computational domain C, the network outputs a mapping from C to the standard parametric domain to represent the inverse mapping of the parameterization. The network should have the generalization capacity, that is, it only needs to be trained once for the parameterizations of all the computational domains. The problem is thus formulated as follows:

Formulation 1. *Given the parametric domain $P = [0,1]^2$, the set $\mathcal{D}$ of simply connected planar computational domains, find a mapping $G : \mathbf{R}^2 \times \mathcal{D} \mapsto \mathbf{R}^2$ which satisfies the following requirements for each specific computational domain $C \in \mathcal{D}$:*

1. *The mapping $G_C(\cdot) := G(\cdot, C)$ is bijective.*
2. *The mapping $G_C(\cdot) := G(\cdot, C)$ has low area and angular distortion.*
3. *The one-sided Hausdorff distance $h(G_C(\partial C), \partial P)$ is smaller than a threshold.*

In this work, we train a neural network in an unsupervised manner to represent the inverse mapping G. Thus we reformulate the problem as:

Formulation 2. *Train a neural network G, with the inputs ∂C and a point $(x, y) \in C$, and the output (u, v) which are the parametric values corresponding to (x, y) with respect to C, such that the following loss function is minimized:*

$$L_S = \frac{1}{\#S} \sum_{C \in S} L_C \tag{1}$$

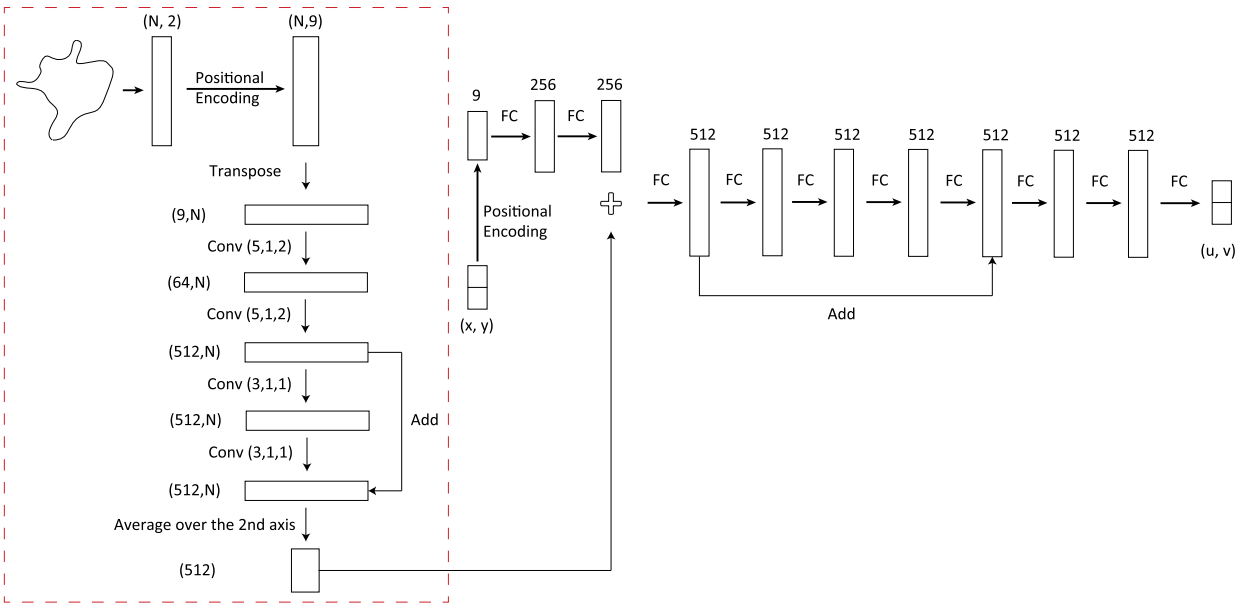


Fig. 2. Structure of our neural network. The network consists of two components: the component surrounded by the red dashed wireframe is for feature extraction of the computational domain, and the other component is an MLP for computing the parameterization mapping. In the mapping component, the feature vector is attached to the third layer of the MLP. $Conv(a,b,c)$ denotes a 1-dimensional convolutional layer whose $kernal\_size = a, stride = b, padding = c$ with a batch normalization layer, FC refers to a fully connected layer, the signature + stands for concatenation of two vectors, the numbers around the tensors denote their dimensions, the arrows marked with "Add" are skip connections.

where

$$\begin{aligned} L_C &= L_C^{inner} + L_C^{bound} \\ L_C^{inner} &= w_{area} L_C^{area} + w_{angle} L_C^{angle} + w_{smooth} L_C^{smooth} \\ L_C^{bound} &= w_{dist} L_C^{dist} + w_{inter} L_C^{inter} \end{aligned} \tag{2}$$

S is the training set, which is composed of a set of simply connected planar domains. L_C^{inner} represents the interior loss of mapping $G_C(\cdot) = G(\cdot, C)$ over the computational domain C, including area distortion, angular distortion, and a smooth term. L_C^{bound} represents the boundary loss of mapping G_C, which includes the distance from $G_C(\partial C)$ to ∂P and the constraints for non self-intersection of $G_C(\partial C)$. w_{area}, w_{angle}, w_{smooth}, w_{dist}, and w_{inter} are the weights. Further details will be provided in Section 3.3.

The loss function of our neural network is similar to that proposed in Zhan et al. (2024) except that a smooth term is added and the calculation of the distortion is modified in L_C^{inner}. The training process of the neural network is illustrated in Section 3.4. After the network is trained, to parameterize a specific simply connected planar domain C, we just follow the procedure in Section 3.6.

3.2. Neural network architecture

To represent the mapping G in Formulation 1 using a neural network, we design the network architecture as in Fig. 2. The network G consists of two parts–feature extraction and parameterization mapping. The feature extraction component encodes each computational domain into a vector, while the parameterization mapping component computes a mapping from a specific computational domain C to the parametric domain P. In the following, we explain the two components in details.

The feature extraction component is to encode the boundary of a computational domain into a vector. For a specific computational domain C, we first discretize its boundary into an ordered point sequence $\mathbf{b} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_N)$ with $\mathbf{b}_i = (x_i, y_i)$ uniformly (in our implementation, we set $N = 256$). In order to enhance the fitting capacity of the network, we expand each pair of coordinates (x, y) into a 9-dimensional vector $(x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3)$ through position encoding. Thus we obtain a $N \times 9$ tensor as the input of the network. The network is then followed by a series of convolutional layers to obtain a $512 \times N$ tensor whose i-th column represents the local feature extracted from the points nearby $\mathbf{b}_i$. Finally, we calculate the average vector of the columns of the tensor to obtain a 512-dimensional feature vector $\mathbf{v}$. This process is to ensure that the extracted feature is insensitive to the starting position of the sample points.

The parameterization mapping component is mainly an MLP, similar to that in Zhan et al. (2024). This part includes a series of fully connected layers and a skip connection. For each point $\mathbf{p} = (x, y)$ in the computational domain C, after positional encoding, it is fed into the network which also receives the feature vector $\mathbf{v}$ extracted by the feature extraction component as its input. The output of the network is a pair of parametric values (u, v) corresponding to (x, y) with respect to C.

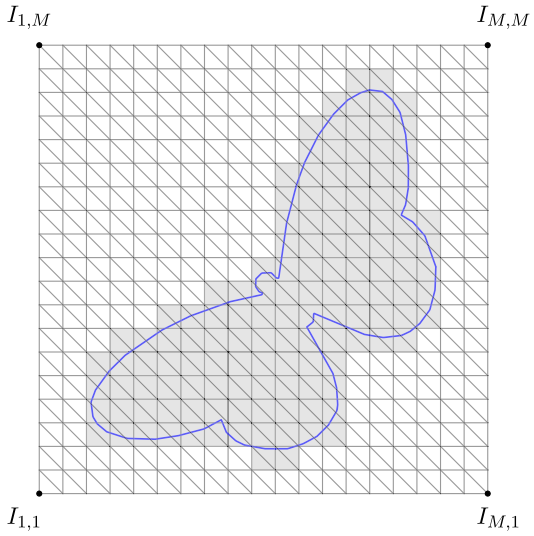


Fig. 3. This figure illustrates how to sample points to calculate the interior loss. The blue curve is the boundary ∂C of the computational domain C, and the cells that cover C are marked in gray.

In the network, we employ the rectified linear unit (ReLU) as the activation function. In the feature extraction component, we add a batch normalization layer following each convolutional layer. This enables the network to converge faster. Nevertheless, it is important to note that batch normalization is not applied in the parameterization mapping component. This is because batch normalization does not perform well in fitting tasks and can introduce numerical instability in the output.

In addition, we conduct an ablation study in Section 4 to validate the effectiveness of positional encoding. The experimental results demonstrate that positional encoding significantly enhances the performance of the network. This improvement may be due to the enlargement of the parametrization space represented by the neural network.

3.3. The loss function

We train our network in an unsupervised manner, thus a loss function that measures the parameterization quality is needed. The loss function we employ is similar to that in Zhan et al. (2024). As described in (1), the loss function L_C is composed of two parts: L_C^{bound} and L_C^{inner}. L_C^{bound} measures the distance between the image of the computational domain C and the parametric domain P, and it comprises two terms L_C^{dist} and L_C^{inter}. L_C^{inner} measures the parameterization quality which consists of three terms L_C^{area}, L_C^{angle} and L_C^{smooth}, where L_C^{area} and L_C^{angle} are area distortion and angular distortion respectively, and L_C^{smooth} is a smooth term which doesn't appear in Zhan et al. (2024). Below we explain the details about how these terms are calculated.

3.3.1. Boundary loss

The boundary loss $L_C^{bound} = w_{dist} L_C^{dist} + w_{inter} L_C^{inter}$ measures the boundary difference defined by the inverse parameterization G_C, where L_C^{dist} is the one-sided Hausdorff distance between $G_C(\partial C)$ and ∂P, and L_C^{inter} is a constraint that ensures $G_C(\partial C)$ forms a star-shaped region with respect to the point $A = (1/2, 1/2)$, thereby guaranteeing that $G_C(\partial C)$ has no self-intersection. The calculation of these two terms is described below.

First, let $B = \{\mathbf{b}_i\}_{i=1}^N$ be a list of points sampled on the boundary ∂C. Then

$$\begin{aligned} L_C^{dist} &= \frac{1}{N}\sum_{i=1}^{N}\left(\operatorname{dist}(G_C(\mathbf{b}_i), \partial P)\right)^2, \\ L_C^{inter} &= \frac{1}{N}\sum_{i=1}^{N}(-s_i)_+, \\ \text{with } s_i &= \frac{G_C(\mathbf{b}_i) - A) \times (G_C(\mathbf{b}_{i+1}) - A)}{\|G_C(\mathbf{b}_i) - A\|_2 \|G_C(\mathbf{b}_{i+1}) - A\|_2}. \end{aligned}$$

3.3.2. Smooth term

The smooth term serves two purposes. First, it makes the mapping G_C globally smoother, and second, it can accelerate the training process of the network.

As shown in Fig. 3, we uniformly sample an $M \times M$ mesh with grid points $I_C = \{I_{i,j}\}_{M\times M}$ which covers the computational domain C. The cells that cover C are marked in gray in the figure. We denote the width and height of the cells as h, denote the

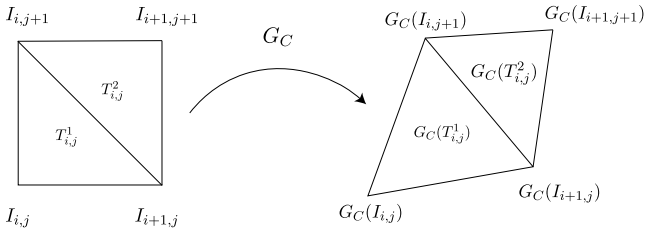


Fig. 4. The mapping G_C over a cell.

marked region which covers C as Ω and denote the indices (i,j) of the grid points that lies in Ω as $K = \{(i,j)|I_{i,j} \in \Omega\}$. Then the smooth term $L_C^{smooth}(i,j)$ at each grid point $I_{i,j}$ is defined as

$$L_C^{smooth}(i,j) = \left\| G_C(I_{i,j}) - \frac{1}{8} \sum_{\substack{(\hat{i},\hat{j})\neq(i,j) \\ \hat{i}=i-1,i,i+1 \\ \hat{j}=j-1,j,j+1}} G_C(I_{\hat{i},\hat{j}}) \right\|_2^2 .$$

The smooth term L_C^{smooth} over the domain C is defined as

$$L_C^{smooth} = \frac{1}{\#K} \sum_{(i,j)\in K} L_C^{smooth}(i,j).$$

3.3.3. Area and angular distortion terms

The area distortion L_C^{area} and the angular distortion L_C^{angle} are calculated based on the Jacobian matrix of the mapping G_C. Zhan et al. (2024) proposed calculating the Jacobian matrix using a finite difference method at each sample point. On contrast, we compute the Jacobian matrix over a triangular mesh of the computational domain in this paper. The advantage of this approach is a reduction in computational cost: for the same number of sampled points, fewer function (G_C) values need to be computed. As shown in Fig. 3, the $M \times M$ mesh is triangulated to get a triangular mesh $T = \{T_i\}_i$. The Jacobian matrix $J_C(T_i)$ can be calculated as follows for the two types of triangles in Fig. 4:

$$J_C(T^1_{i,j}) = \begin{pmatrix} \frac{G_C(I_{i+1,j}) - G_C(I_{i,j})}{h} & \frac{G_C(I_{i,j+1}) - G_C(I_{i,j})}{h} \end{pmatrix}$$
$$J_C(T^2_{i,j}) = \begin{pmatrix} \frac{G_C(I_{i+1,j+1}) - G_C(I_{i,j+1})}{h} & \frac{G_C(I_{i+1,j+1}) - G_C(I_{i+1,j})}{h} \end{pmatrix}$$

The prior work Zhan et al. (2024) measures the area distortion by $|J_p| + 1/|J_p| - 2$ and the angular distortion using the Beltrami coefficient μ with $|\mu|^2 = \frac{||J_p||_F^2 - 2|J_p|}{||J_p||_F^2 + 2|J_p|}$, where $|\cdot|$ refers to determinant, and $||\cdot||_F$ stands for the Frobenius norm. In this paper, we define the area distortion $L_C^{area}(T_i)$ and the angular distortion $L_C^{angle}(T_i)$ over each triangle T_i by

$$\begin{aligned} L_C^{area}(T_i) &= P_t(|J_C(T_i)|), \\ L_C^{angle}(T_i) &= \frac{||J_C(T_i)||_F^2 - 2|J_C(T_i)|}{||J_C(T_i)||_F^2 + 2|J_C(T_i)|}, \end{aligned}$$

where

$$P_t(x) = \begin{cases} (1 - \frac{1}{t^2})x + \frac{2}{t} - 2, & x < t \\ x + \frac{1}{x} - 2, & x \geq t \end{cases}$$

is an improvement over the function $x + 1/x - 2$, plotted in Fig. 5. $P_t(x)$ produces a large positive value when $|J_C(T_i)| < 0$, i.e., G_C is not bijective. In our implementation, t is set to be 0.01, this allows the neural network to effectively avoid non-bijective results.

Subsequently, the area distortion L_C^{area} and the angular distortion L_C^{angle} over the computational domain C are calculated as follows:

$$\begin{aligned} L_C^{area} &= \frac{1}{\#T} \sum_{T_i \in T} L_C^{area}(T_i), \\ L_C^{angle} &= \frac{1}{\#T} \sum_{T_i \in T} L_C^{angle}(T_i). \end{aligned}$$

It is worth noting that, during the training process, the mesh I_C changes in each iteration to avoid network overfitting. Before training, we sample an initial mesh I_C for each domain C in the dataset, and then calculate the corresponding region Ω, index set K and triangle set T. In each iteration, we randomly choose a vector $\vec{v} \in [-h/2, h/2]^2$, and then the new mesh becomes $\hat{I}_C = I_C + \vec{v}$.

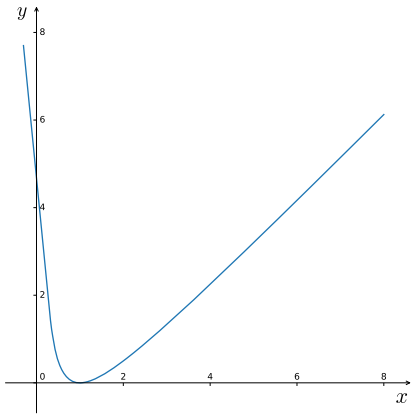


Fig. 5. The function $y = P_t(x)$ is shown in figure with $t = 0.3$.

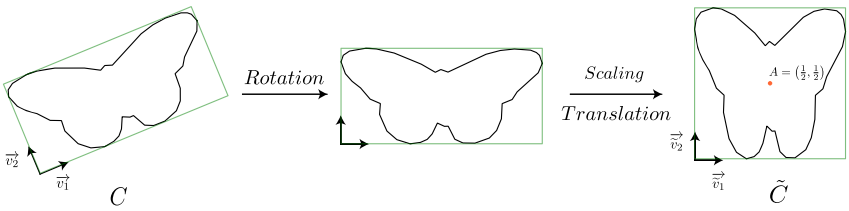


Fig. 6. The pre-processing of the computational domain C. The vectors $\overrightarrow{v_1}, \overrightarrow{v_2}$ and $\overrightarrow{\tilde{v}_1}, \overrightarrow{\tilde{v}_2}$ are principal directions.

To avoid redundant computation, we keep the index set K unchanged, take the cover region as $\hat{\Omega} = \Omega + \vec{v}$, and update the set of triangles as $\hat{T} = T + \vec{v}$. Additionally, to ensure that the region $\hat{\Omega}$ covers the domain C, the region Ω has to be slightly enlarged to include the computational domain C.

3.4. Training and testing

For a dataset $\mathcal{D}$ that consists of a set of simply connected planar domain $\{C_i\}_i$, we train and test a neural network in the following steps:

1. Neural network pretraining. We initialize the network to be an identity mapping, meaning that for any point (x, y) and any domain C, it holds that $G(x, y, C) \approx (x, y)$.
2. Apply some pre-processing for each domain in $\mathcal{D}$, whose details will be described in Section 3.5. Then we divide the dataset $\mathcal{D}$ into a training set $\mathcal{D}_{train}$, a validation set $\mathcal{D}_{val}$, and a test set $\mathcal{D}_{test}$. The details of the dataset and the division can be found in Section 4.2.
3. Train the neural network with the loss function (1) over the training set $\mathcal{D}_{train}$ for N_{epoch} epochs. Choose the epoch with minimal validation loss as the training result. The specific training parameters can be found in Section 4.2.
4. Use the trained neural network to parameterize the computational domains in $\mathcal{D}_{test}$, following the procedure described in Section 3.6.

3.5. Pre-processing

The pre-processing step has been employed in the previous work Zhan et al. (2024), which is modified slightly in this paper.

First, we employ Principal Component Analysis (PCA) to compute two orthogonal principal directions $\overrightarrow{v_1}, \overrightarrow{v_2}$ of the computational domain C, and compute an oriented bounding box (whose directions are $\overrightarrow{v_1}, \overrightarrow{v_2}$ respectively) to include C. Subsequently, rotation transformation is performed to make the principal directions axis-aligned. Following the rotation, we proceed to apply scaling and translation operations, ensuring that the bounding box forms a unit square centered at the coordinates $A = (0.5, 0.5)$. We denote the pre-processing as a mapping p and denote the resulting domain as $\tilde{C} = p(C)$. The entire pre-processing step is shown in Fig. 6.

3.6. Parameterize a specific computational domain

After the neural network is trained, we can parameterize each specific computational domain C by a similar approach to the post-processing step in Zhan et al. (2024). Specific procedure is as follows (as shown in Fig. 7):

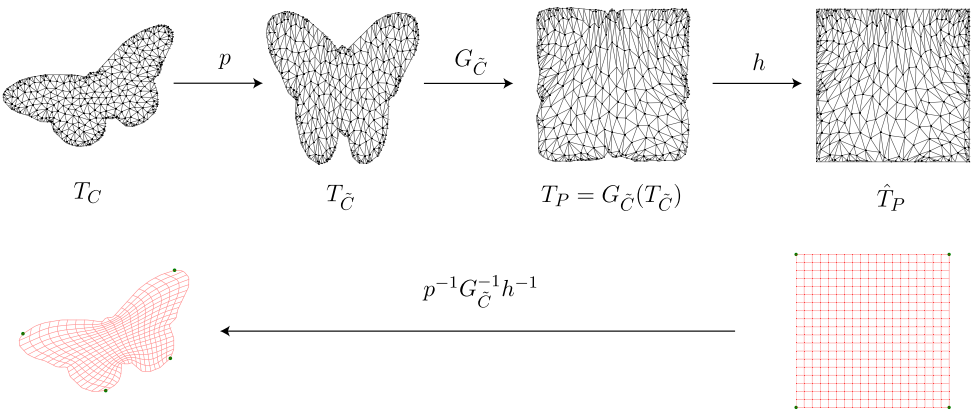


Fig. 7. The process of parameterizing a computational domain C using the mapping $G_{\tilde{C}}$.

1. Convert the domain C into the domain $\tilde{C}$ by the pre-processing step outlined in 3.5. At the same time, triangulate C into a triangular mesh T_C, and obtain the corresponding triangulation $T_{\tilde{C}} = p(T_C)$.
2. Employ the trained neural network to compute a corresponding triangulation in the parametric domain, denoted as $T_P = G_{\tilde{C}}(T_{\tilde{C}})$.
3. Modify the mesh T_P into the mesh $\hat{T}_P$ — a triangular mesh that aligns well with the boundary ∂P. This can be achieved through mean value coordinates (MVC) parameterization Floater (2003). We denote the mapping by h. Fig. 8 provides several further examples to illustrate the mapping.
4. Utilize a bi-quadratic B-spline function F_C to fit the mapping $p^{-1}G_{\tilde{C}}^{-1}h^{-1}$ that maps $\hat{T}_P$ to T_C, resulting in a forward parameterization. In this paper, the B-spline function F_C contains 53 knots in both directions with 50×50 control coefficients.
   Previous work (Zhan et al. (2024)) employs the least squares method to fit the mapping $p^{-1}G_{\tilde{C}}^{-1}h^{-1}$. However, the fitting process introduces some error which may result in a non-bijective parameterization (Zhan et al. (2024)).
   To address this issue, we conduct further optimization for the B-spline parameterization. We employ the Adam's optimization algorithm (Kingma and Ba (2017)) as the optimizer, and set the control coefficients of F_C to be the decision variables. The objective function contains two parts. The first part is defined in the same way as L_C^{inner} in (1), which represents the interior mapping distortion. The second part is the fitting part defined by $\frac{1}{n}\sum_{i=1}^{n} \|F_C(u_i) - x_i\|_2^2$, in which $\{u_i\}_{i=1}^n$ are $n = 400$ points uniformly sampled on $\partial([0,1]^2)$ and $\{x_i = p^{-1}G_{\tilde{C}}^{-1}h^{-1}(u_i)\}_{i=1}^n$ are the targets. This part serves as a soft constraint on boundary mapping. In the optimization process, the learning rate (step of the optimizer) is set to be 10^{-3}, and the maximum number of iterations is 100.

4. Experiments and evaluation

4.1. Dataset

There are two datasets used in our approach: MPEG-7 dataset and a self designed dataset (Hand-Draw), some examples of these datasets are shown in Fig. 9. The MPEG-7 dataset is an open source dataset that contains 1406 silhouettes. Zhan et al. (2024) constructed a set of simply connected planar domains from it (977 samples). For convenience, we still refer to the dataset as the MPEG-7 dataset. Additionally, we design some computational domains with hand-draw to create another dataset, denoted as Hand-Draw dataset. This dataset contains 799 computational domains. Finally, we scale all the computational domains such that their areas are all 1.

4.2. Training and testing

We divide the MPEG-7 dataset into three parts: training set $\mathcal{D}_{train}$, validation set $\mathcal{D}_{val}$ and test set $\mathcal{D}_{test}$, which contain 677, 150, 150 samples respectively. Then, we train our neural network on the training set $\mathcal{D}_{train}$ for N_{epoch} epochs, and choose the epoch with minimal validation loss as the training result (denoted by Net-M). After that, we test our method and conduct comparison experiments with other parameterization methods Ji et al. (2023); Pan et al. (2018); Liu et al. (2020); Zhan et al. (2024) on the test set $\mathcal{D}_{test}$. To further examine the network's generalization capability, we retrain the network from scratch on the dataset Hand-Draw and denote the training result as Net-H. In this process, the Hand-Draw dataset is divided into a training set and a validation set, consisting of 699 and 100 samples respectively. After training, we conduct tests for Net-M and Net-H on the set $\mathcal{D}_{test}$. Since the test set $\mathcal{D}_{test}$ is selected from the MPEG-7 dataset, the generalization capability of the network can be examined if Net-H perform similarly to Net-M on $\mathcal{D}_{test}$.

In the training process, we use Adam optimizer, and we set the training epoch $N_{epoch} = 100000$, the learning rate $lr = 10^{-4}$ and the batch size $bs = 64$. Additionally, the validation loss is calculated over the validation test $\mathcal{D}_{val}$ in each epoch, then the model with minimum validation loss will be saved as the training result.

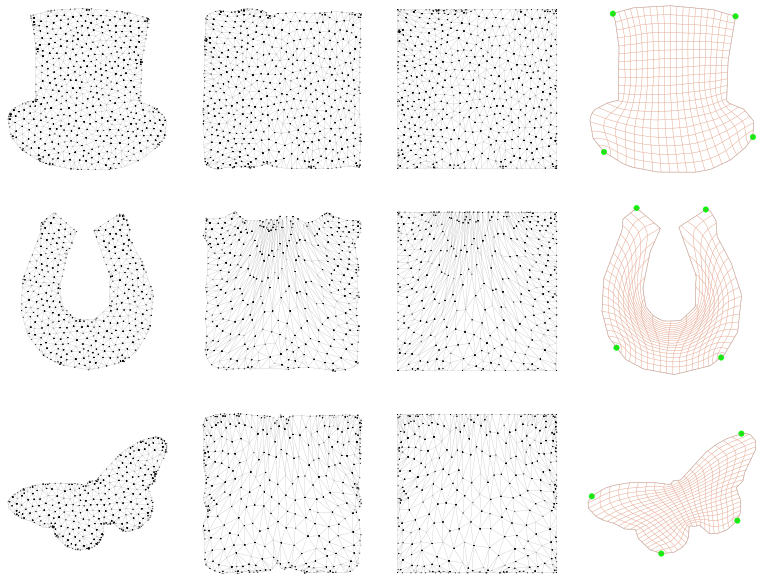


Fig. 8. Further examples are provided to illustrate the modification of the triangular mesh T_P by the mapping h.

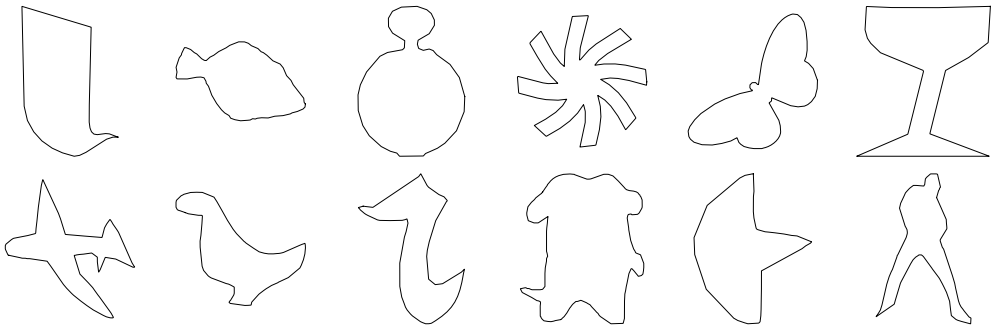


Fig. 9. Some examples in the dataset MPEG-7 (the top row) and dataset Hand-Draw (the second row).

The code 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, the whole training process lasts about 50 hours.

The weights w_{area}, w_{angle}, w_{smooth}, w_{dis}, w_{fold} in the loss function are selected based on the network's performance on the training set. More precisely, we choose appropriate weights to ensure reasonable boundary correspondences and uniform interior parameterization. In our implementation, the weights are set to be $w_{area} = 1$, $w_{angle} = 4$, $w_{smooth} = 20$, $w_{dis} = 2000$, $w_{fold} = 50$.

4.3. Evaluation and comparisons

In the beginning, an ablation experiment is conducted to validate the effectiveness of positional encoding in Section 3.2, which encodes the coordinates (x, y) into a 9-dimensional vector $(x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3)$. We trained the neural network G on the MPEG-7 dataset under two situations: with and without positional encoding. The effectiveness of the encoding can be observed from the difference of the loss functions under both situations as illustrated in Fig. 10.

Next we perform comparisons of the current method with four state-of-the-art methods including Pan et al. (2018); Zheng et al. (2019); Ji et al. (2023) and Zhan et al. (2024), and evaluate the parameterization results over the test set $\mathcal{D}_{test}$, which contains 150 computational domains. When evaluate the internal parameterization methods (Pan et al. (2018); Ji et al. (2023)), suitable boundary correspondences are generated beforehand using the method based on optimal mass transport (OMT) in Zheng et al. (2019). We evaluate all the results over the test set using 7 metrics: $Area^{ave}$, $Area^{max}$, $Angle^{ave}$, $Angle^{max}$, $Bound_{dist}$, $Bijective$ and $Time$.

The metrics $Bound_{dist}$, $Bijective$ and $Time$ represent the Hausdorff distance between two boundaries $F_C(\partial P)$ and ∂C, the bijectivity of the parameterization mapping F_C and the computational time, respectively. The other metrics are defined as follows:

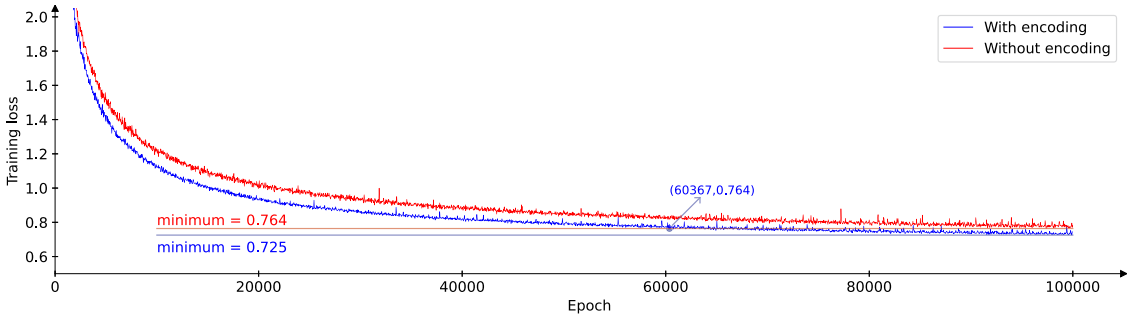


Fig. 10. Comparison of loss functions in the ablation experiments of positional encoding.

Table 1
Comparisons between our method and the four methods: Ji et al. (2023); Pan et al. (2018); Liu et al. (2020); Zhan et al. (2024). *Bijective* stands for the number of the bijective parameterizations, the other quantities are the average metrics over the test set. The metric Time-GPU shows the computational time when GPU acceleration is applied. The best metrics are marked in bold.

Method	$Area^{ave}$	$Area^{max}$	$Angle^{ave}$	$Angle^{max}$	Bijective	$Bound_{dist}$	Time(s)	Time-GPU(s)
Ji et al. (2023)+OMT	0.37447	0.83122	0.43620	0.78991	114/150	0.00770	0.524	-
Pan et al. (2018)+OMT	0.39458	0.91147	0.41724	0.78342	115/150	0.00846	24.1	-
Liu et al. (2020)	0.91468	0.99417	0.91703	0.97456	15/150	0.01851	27.6	-
Zhan et al. (2024)	0.13316	**0.60718**	0.24795	**0.65322**	146/150	**0.00454**	-	17.3
Net-M	**0.11447**	0.75576	**0.22948**	0.68650	**149**/150	0.00671	**0.161**	**0.093**
Net-H	0.13146	0.77108	0.24874	0.70719	147/150	0.00697	0.238	0.113

For a computational domain C and a corresponding bijective parameterization mapping F_C, we denote the Jacobian matrix at each point $p \in C$ as J_p, and define the area and angular distortion as

$$\begin{aligned} Area(p) &= 1 - \frac{1}{|J_p| + 1/|J_p| - 1}, \\ Angle(p) &= \frac{\|J_p\|_F^2 - 2|J_p|}{\|J_p\|_F^2 + 2|J_p|}. \end{aligned} \tag{3}$$

In this definition, both the area and angular distortions have a range of $[0, 1]$ for bijective mappings, and a smaller value indicates less distortion. The average and maximum values of $Area(p)$ and $Angle(p)$ over the domain C are denoted by $Area^{ave}$, $Area^{max}$, $Angle^{ave}$ and $Angle^{max}$, respectively. For a non-bijective parameterization, we set these metrics to be the maximum value 1.

To compare the parameterization quality of our method and the four state-of-the-art methods, we perform parameterization over the test dataset $\mathcal{D}_{test}$ and compute 7 evaluation metrics as shown in Table 1. Both the computational times with and without GPU acceleration are presented in the table. From the table, it can be observed that: (1) methods based on deep neural network produce better parameterization results than other state-of-the art methods; and (2) the current method achieves as good parameterization quality as the method Zhan et al. (2024) but with two orders of magnitude speedup, which demonstrates the excellent generalization capacity of our neural network for domain parameterization. Furthermore, the result Net-H performs similarly to Net-M. Fig. 11 presents parameterization results for several computational domains.

Based on the above parameterization results, the deep learning-based integral parameterization methods demonstrate several advantages compared to other methods:

1. Higher accuracy in boundary maps.
   As shown in Fig. 11, the boundaries of the parameterization mapping generated by Liu et al. (2020); Pan et al. (2018); Ji et al. (2023) often fail to align with the target boundaries, while the method Zhan et al. (2024) and the current method obtain better boundary mapping. This can also be observed by the metric $Bound_{dist}$ in Table 1.
2. Better correspondence of corner points.
   As shown in Fig. 11, the corner points produced by Liu et al. (2020) often lie on concave regions, resulting in non-bijective parameterizations. Furthermore, the corner points selected by Zheng et al. (2019) are occasionally unevenly distributed. The method Zhan et al. (2024) and the current method can both generate more reasonable corner points.
3. More uniform parameterization results.
   In the parameterizations generated by Liu et al. (2020); Pan et al. (2018); Ji et al. (2023), large distortions often occur in certain regions of the computational domains, while the learning based approaches produce more uniform distortions.
4. More likely to generate bijective results.
   It can be found in Table 1 that, neural network based methods generate more bijective results on the test set, though there is no theoretic guarantee that every parameterization is bijective. This is also attributed to better selection of corner points and more

Fig. 11. Comparisons between our method and the methods of Ji et al. (2023); Pan et al. (2018); Liu et al. (2020); Zhan et al. (2024) on several computational domains. The method of Ji et al. (2023) fails to produce a bijective map on the domain 6.

uniform internal mappings by the suitable design of the loss function term L^{area} and the post-processing step in B-spline fitting described in Section 3.6.

When comparing the two deep learning-based methods—the current method and Zhan et al. (2024), each has its own strengths and weaknesses:

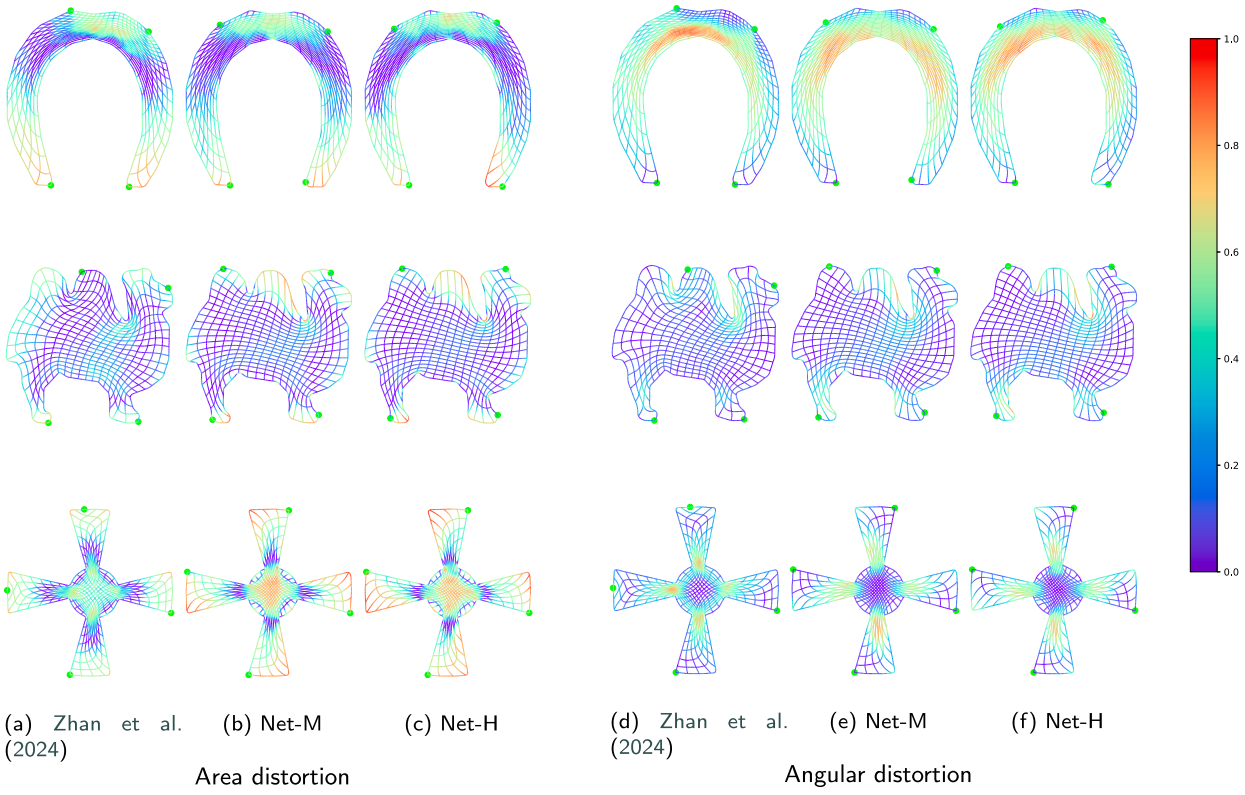


Fig. 12. The area and angular distortion of the two learning-based approaches for three examples.

1. In regular computational domains, such as 1, 2, and 3 in Fig. 11, both methods yield nearly identical results. However, the domain 4 demonstrates that the current approach achieves a more uniform parameterization compared to Zhan et al. (2024). This improvement is not solely due to the inclusion of a smoothing term in the loss function, but also a result of training on a large dataset. Generally, a uniform parameterization minimizes the loss function in most computational domains. However, for some complex domains, the loss function might not be a perfect indicator of parameterization quality, resulting in minimal loss which doesn't guarantee a uniform parameterization. In this work, we train the network to minimize the average loss function across the dataset, enabling uniform parameterization even on complex domains.
2. As seen in domains 5 and 6 in Fig. 11, for extremely complex computational domains, the current method does not achieve parameterization quality as good as Zhan et al. (2024). By observing the distribution of the grid lines in parameterization results by the current method, it is evident that grid lines are sparser around the boundaries and denser in the central areas, which indicates significant distortion near the boundaries. We also visualize the area and angular distortion of these examples in Fig. 12.
3. In example 4 and 6 in Fig. 11, the current method chooses better corner points.
4. In Zhan et al. (2024), retraining the neural network for each different computational domain results in heavy computational burden. However, there's no requirement for repetitive network training with the current method, enabling rapid parameterization in significantly much shorter time. It can be observed in Table 1 that, the computational time of the current method is around 0.2 seconds on CPU, and around 0.1 seconds with GPU acceleration, while the computational domain by the method Zhan et al. (2024) is usually around 20 seconds.

In summary, the current method can produce satisfactory parameterization results for general computational domains in a short period of time. However, for extremely complex computational domains, the method of Zhan et al. (2024) exhibits better performance. Thus in these extreme cases, one may improve the parameterization results of our method by continuing the training only for the current domain, with the parameters of the feature extraction component fixed. This procedure can be finished in less than 200 iterations within 3 seconds. It can be observed from Fig. 13 that this approach indeed improves the parameterization results to some extend.

5. Conclusion

In this paper, we enhance the neural work presented in Zhan et al. (2024) for domain parameterization with exceptional generalization capacity. A neural network is trained in an unsupervised manner to approximate the inverse parameterization, and once the

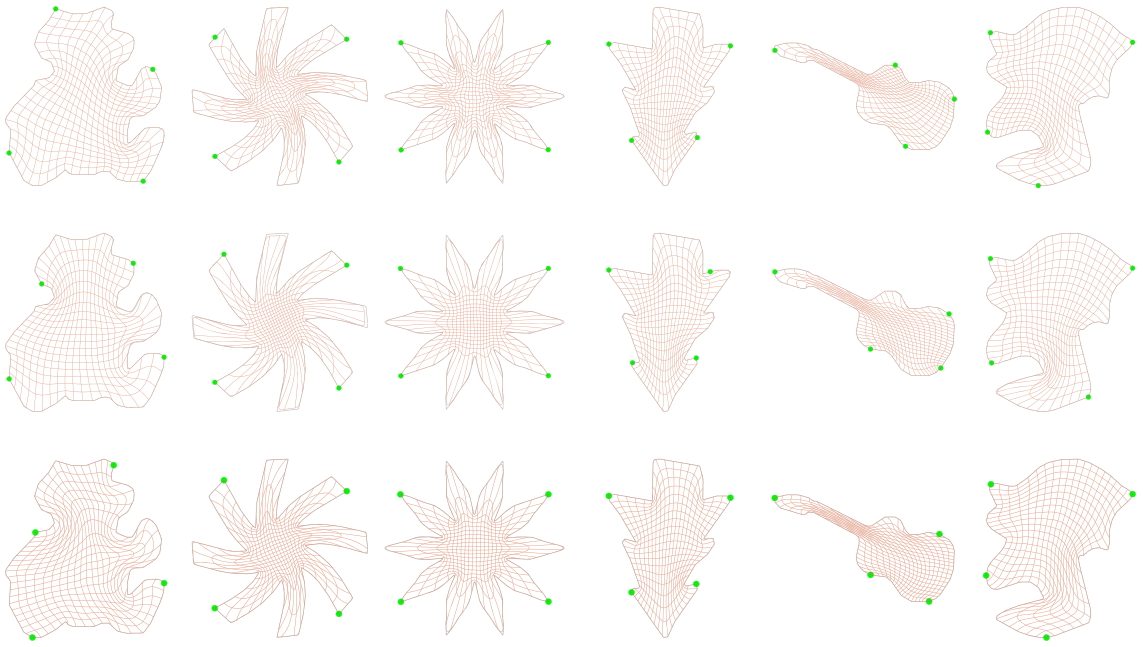


Fig. 13. Comparison over complex computational domains between the method of Zhan et al. (2024) (top), the current method Net-M (middle) and Net-M with further training (bottom).

network is trained, the parameterization is obtained by evaluating the network. The new approach achieves parameterization results as good as Zhan et al. (2024) with two orders of magnitude speedup. Compared to the traditional approaches Liu et al. (2020); Pan et al. (2018); Ji et al. (2023), the new method achieves higher bijectivity ratio, lower distortion and more uniform parameterization. Experimental examples are provided to illustrate the effectiveness of our approach.

However, there are a few limitations in the current approach. First, the neural network is designed for two-dimensional computational domains and can not be directly applied to three-dimensional domains. Thus, an important future research direction is to improve the current work and extend it to the parameterization of three-dimensional computational volumes. Second, our method can only parameterize simply connected domains by single-patch parameterization. Developing a network which divides multi-genus geometry into multiple genus-zero geometries for subsequent multi-patch parameterization is another important future research target.

CRediT authorship contribution statement

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

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.

Data availability

Data will be made available on request.

Acknowledgements

We would like to thank Dr. Ye Zheng, Maodong Pan and Ye Ji for providing partial experimental results and the source codes. This work is supported by NSF of China (no. 61972368, no. 12371383).

References

Buchegger, F., Jüttler, B., 2017. Planar multi-patch domain parameterization via patch adjacency graphs. Comput. Aided Des. 82, 2–12. https://doi.org/10.1016/j.cad.2016.05.019.

Escobar, J.M., Cascón, J.M., Rodríguez, E., Montenegro, R., 2011. A new approach to solid modeling with trivariate t-splines based on mesh optimization. Comput. Methods Appl. Mech. Eng. 200, 3210–3222. https://doi.org/10.1016/j.cma.2011.07.004.

Falini, A., D'Inverno, G.A., Sampoli, M.L., Mazzia, F., 2023. Splines parameterization of planar domains by physics-informed neural networks. Mathematics 11, 2406. https://doi.org/10.3390/math11102406.

Falini, A., Špeh, J., Jüttler, B., 2015. Planar domain parameterization with thb-splines. Comput. Aided Geom. Des. 35–36, 95–108. https://doi.org/10.1016/j.cagd.2015.03.014.

Farin, G., Hansford, D., 1999. Discrete coons patches. Comput. Aided Geom. Des. 16, 691–700. https://doi.org/10.1016/S0167-8396(99)00031-X.

Floater, M.S., 2003. Mean value coordinates. Comput. Aided Geom. Des. 20, 19–27. https://doi.org/10.1016/S0167-8396(03)00002-5.

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

Gravesen, J., Evgrafov, A., Nguyen, D.M., Nørtoft, P., 2014. Planar parametrization in isogeometric analysis. In: Mathematical Methods for Curves and Surfaces. Springer, Berlin, Heidelberg, pp. 189–212.

Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y., 2005. Isogeometric analysis: cad, finite elements, nurbs, exact geometry and mesh refinement. Comput. Methods Appl. Mech. Eng. 194, 4135–4195. https://doi.org/10.1016/j.cma.2004.10.008.

Ji, Y., Chen, K., Möller, M., Vuik, C., 2023. On an improved pde-based elliptic parameterization method for isogeometric analysis using preconditioned Anderson acceleration. Comput. Aided Geom. Des. 102, 102191. https://doi.org/10.1016/j.cagd.2023.102191.

Kingma, D.P., Ba, J., 2017. Adam: a method for stochastic optimization. https://doi.org/10.48550/arXiv.1412.6980. arXiv:1412.6980, 2017.

Liu, H., Yang, Y., Liu, Y., Fu, X.M., 2020. Simultaneous interior and boundary optimization of volumetric domain parameterizations for iga. Comput. Aided Geom. Des. 79, 101853. https://doi.org/10.1016/j.cagd.2020.101853.

Nguyen, T., Jüttler, B., 2012. Parameterization of contractible domains using sequences of harmonic maps. In: Curves and Surfaces. Springer, Berlin, Heidelberg, pp. 501–514.

Nian, X., Chen, F., 2016. Planar domain parameterization for isogeometric analysis based on Teichmüller mapping. Comput. Methods Appl. Mech. Eng. 311, 41–55. https://doi.org/10.1016/j.cma.2016.07.035.

Pan, M., Chen, F., Tong, W., 2018. Low-rank parameterization of planar domains for isogeometric analysis. Comput. Aided Geom. Des. 63, 1–16. https://doi.org/10.1016/j.cagd.2018.04.002.

Pan, M., Chen, F., Tong, W., 2020. Volumetric spline parameterization for isogeometric analysis. Comput. Methods Appl. Mech. Eng. 359, 112769. https://doi.org/10.1016/j.cma.2019.112769.

Pan, M., Zou, R., Tong, W., Guo, Y., Chen, F., 2023. G1-smooth planar parameterization of complex domains for isogeometric analysis. Comput. Methods Appl. Mech. Eng. 417, 116330. https://doi.org/10.1016/j.cma.2023.116330.

Wang, X., Qian, X., 2014. An optimization approach for constructing trivariate -spline solids. Comput. Aided Des. 46, 179–191. https://doi.org/10.1016/j.cad.2013.08.030.

Xiao, S., Kang, H., Fu, X.M., Chen, F., 2018. Computing iga-suitable planar parameterizations by polysquare-enhanced domain partition. Comput. Aided Geom. Des. 62, 29–43. https://doi.org/10.1016/j.cagd.2018.03.008.

Xu, G., Li, M., Mourrain, B., Rabczuk, T., Xu, J., Bordas, S.P., 2018. Constructing iga-suitable planar parameterization from complex cad boundary by domain partition and global/local optimization. Comput. Methods Appl. Mech. Eng. 328, 175–200. https://doi.org/10.1016/j.cma.2017.08.052.

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

Zhan, Z., Wang, W., Chen, F., 2024. Simultaneous boundary and interior parameterization of planar domains via deep learning. Comput. Aided Des. 166, 103621. https://doi.org/10.1016/j.cad.2023.103621.

Zhan, Z., Zheng, Y., Wang, W., Chen, F., 2023. Boundary correspondence for iso-geometric analysis based on deep learning. Commun. Math. Stat. 11, 131–150. https://doi.org/10.1007/s40304-023-00337-7.

Zhang, Y., Wang, W., Hughes, T.J.R., 2012. Solid t-spline construction from boundary representations for genus-zero geometry. Comput. Methods Appl. Mech. Eng. 249–252, 185–197. https://doi.org/10.1016/j.cma.2012.01.014.

Zheng, Y., Chen, F., 2021. Volumetric boundary correspondence for isogeometric analysis based on unbalanced optimal transport. Comput. Aided Des. 140, 103078. https://doi.org/10.1016/j.cad.2021.103078.

Zheng, Y., Chen, F., 2022. Volumetric parameterization with truncated hierarchical b-splines for isogeometric analysis. Comput. Methods Appl. Mech. Eng. 401, 115662. https://doi.org/10.1016/j.cma.2022.115662.

Zheng, Y., Pan, M., Chen, F., 2019. Boundary correspondence of planar domains for isogeometric analysis based on optimal mass transport. Comput. Aided Des. 114, 28–36. https://doi.org/10.1016/j.cad.2019.04.008.