Flatten Anything: Unsupervised Neural Surface Parameterization

Qijian Zhang[1], **Junhui Hou**[1,*], **Wenping Wang**[2], **Ying He**[3]
[1]Department of Computer Science, City University of Hong Kong, Hong Kong SAR, China
[2]Department of Computer Science and Engineering, Texas A&M University, Texas, USA
[3]School of Computer Science and Engineering, Nanyang Technological University, Singapore
qijizhang3-c@my.cityu.edu.hk, jh.hou@cityu.edu.hk
wenping@tamu.edu, yhe@ntu.edu.sg

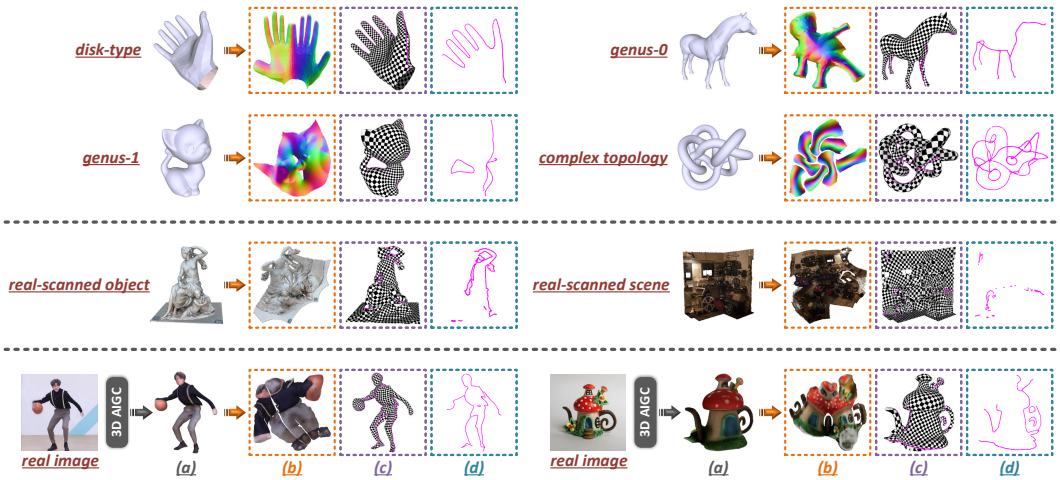


Figure 1: ***Flatten Anything Model (FAM)*** for neural surface parameterization: (a) input 3D models; (b) learned UV coordinates; (c) texture mappings; (d) learned cutting seams.

Abstract

Surface parameterization plays an essential role in numerous computer graphics and geometry processing applications. Traditional parameterization approaches are designed for high-quality meshes laboriously created by specialized 3D modelers, thus unable to meet the processing demand for the current explosion of ordinary 3D data. Moreover, their working mechanisms are typically restricted to certain simple topologies, thus relying on cumbersome manual efforts (e.g., surface cutting, part segmentation) for pre-processing. In this paper, we introduce the *Flatten Anything Model (FAM)*, an unsupervised neural architecture to achieve global free-boundary surface parameterization via learning point-wise mappings between 3D points on the target geometric surface and adaptively-deformed UV coordinates within the 2D parameter domain. To mimic the actual physical procedures, we ingeniously construct geometrically-interpretable sub-networks with specific functionalities of surface cutting, UV deforming, unwrapping, and wrapping, which are assembled into a bi-directional cycle mapping framework. Compared with previous methods, our FAM directly operates on discrete surface points without utilizing connectivity information, thus significantly reducing the strict requirements for mesh quality

[*]Corresponding author. This work was supported in part by the National Natural Science Foundation of China Excellent Young Scientists Fund 62422118, and in part by the Hong Kong Research Grants Council under Grants 11219324 and 11219422.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

and even applicable to unstructured point cloud data. More importantly, our FAM is fully-automated without the need for pre-cutting and can deal with highly-complex topologies, since its learning process adaptively finds reasonable cutting seams and UV boundaries. Extensive experiments demonstrate the universality, superiority, and inspiring potential of our proposed neural surface parameterization paradigm. Our code is available at https://github.com/keeganhk/FlattenAnything.

1 Introduction

Surface parameterization intuitively refers to the process of flattening a 3D geometric surface onto a 2D plane, which is typically called the parameter domain. For any 3D spatial point (x, y, z) lying on the underlying surface, we explicitly map it to a 2D planar coordinate (u, v) while satisfying certain continuity and distortion constraints. Building such point-to-point mappings is also known as UV unwrapping, which serves as an indispensable component in modern graphics rendering pipelines for texture mapping and is also widely used in many downstream geometry processing applications, such as remeshing, mesh completion/compression, detail transfer, surface fitting, and editing, etc.

Theoretically, for any two geometric surfaces with identical/similar topological structures, there exists a bijective mapping between them. Nevertheless, when the topology of the target 3D surface becomes complicated (e.g., with high genus), one must pre-open the original mesh to a sufficiently-developable disk along appropriate cutting seams. Consequently, the current industrial practice for implementing UV unwrapping is typically composed of two stages: (1) manually specifying some necessary cutting seams on the original mesh; (2) applying mature disk-topology-oriented parameterization algorithms (e.g., LSCM [15], ABF [31, 33]) which have been well integrated into popular 3D modeling software (e.g., Blender, Unity, 3Ds Max) to produce per-vertex 2D UV coordinates. In practice, such stage-wise UV unwrapping pipeline still shows the following limitations and inconveniences:

1) Commonly-used surface parameterization algorithms are designed for well-behaved meshes, which are typically produced by specialized 3D modelers and technical artists. However, with the advent of user-generated content (UGC) fueled by the rapidly growing 3D sensing, reconstruction [24, 14], and generation [26, 28, 18, 34] techniques, there emerges an urgent need for dealing with ordinary 3D data possibly with unruly anomalies, inferior triangulations, and non-ideal geometries.

2) Despite the existence of a few early attempts at heuristic seam generation [32, 8], the process of finding high-quality cutting seams in practical applications still relies on manual efforts and personal experience. Hence, the entire workflow remains subjective and semi-automated, leading to reduced reliability and efficiency, especially when dealing with complex 3D models that users are unfamiliar with. Besides, since cutting is actually achieved via edge selection, one may need to repeatedly adjust the distribution of mesh edges to allow the cutting seams to walk through.

3) The procedures of cutting the original mesh into a disk and then flattening the resulting disk onto the parameter domain should have been mutually influenced and jointly optimized; otherwise, the overall surface parameterization results could be sub-optimal.

In recent years, there has emerged a new family of neural parameterization approaches targeted at learning parameterized 3D geometric representations via neural network architectures. The pioneering works of FoldingNet [39] and AtlasNet [11] are among the best two representatives, which can build point-wise mappings via deforming a pre-defined 2D lattice grid to reconstruct the target 3D shape. RegGeoNet [41] and Flattening-Net [42] tend to achieve fixed-boundary rectangular structurization of irregular 3D point clouds through geometry image [12, 21] representations. However, these two approaches cannot be regarded as real-sense surface parameterization due to their lack of mapping constraints. DiffSR [2] and Nuvo [36] focus on the local parameterization of the original 3D surface but with explicit and stronger constraints on the learned neural mapping process. Their difference lies in that DiffSR aggregates multi-patch parameterizations to reconstruct the original geometric surface, while Nuvo adaptively assigns surface points to different charts in a probabilistic manner.

In this paper, we make the first attempt to investigate neural surface parameterization featured by both global mapping and free boundary. We introduce the Flatten Anything Model (FAM), a universal and fully-automated UV unwrapping approach to build point-to-point mappings between 3D points lying on the target geometric surface and 2D UV coordinates within the adaptively-deformed parameter domain. In general, FAM is designed as an unsupervised learning pipeline (i.e., no ground-truth UV

maps are collected), running in a per-model overfitting manner. To mimic the actual physical process, we ingeniously design a series of geometrically-meaningful sub-networks with specific functionalities of surface cutting, UV deforming, 3D-to-2D unwrapping, and 2D-to-3D wrapping, which are further assembled into a bi-directional cycle mapping framework. By optimizing a combination of different loss functions and auxiliary differential geometric constraints, FAM is able to find reasonable 3D cutting seams and 2D UV boundaries, leading to superior parameterization quality when dealing with different degrees of geometric and topological complexities. Comprehensive experiments demonstrate the advantages of our approach over traditional state-of-the-art approaches. Conclusively, our FAM shows the following several major aspects of characteristics and superiorities:

- Compared with traditional mesh parameterization approaches [15, 33, 29], FAM directly operates on discrete surface points and jointly learns surface cutting, which can be insensitive to mesh triangulations and is able to deal with non-ideal geometries and arbitrarily-complex topologies. Besides, such a pure neural learning architecture can naturally exploit the powerful parallelism of GPUs and is much easier for implementing, tuning, and further extension.
- Another distinct advantage lies in the smoothness of parameterization. Since mesh parameterization computes parameters solely for existing vertices, for any point on the mesh that is not a vertex, these approaches resort to interpolation within its encompassing triangle to determine the corresponding UV coordinate. Such practice fails to guarantee smoothness, particularly across edges, and this issue is further exacerbated in low-resolution meshes. By contrast, FAM exploits the inherent smooth property [30] of neural networks to learn arbitrary point-to-point mappings, thereby ensuring smoothness for all points on the target surface.
- Different from previous multi-patch local parameterization frameworks [11, 2, 36], FAM focuses on global parameterization, a more valuable yet much harder problem setting.
- Different from previous fixed-boundary structurization frameworks [41, 42], FAM deforms the 2D UV parameter domain adaptively and regularizes the learned neural mapping explicitly, thus significantly reducing discontinuities and distortions.

2 Related Works

2.1 Traditional Parameterization Approaches

Early approaches formulate mesh parameterization as a Laplacian problem, where boundary points are anchored to a pre-determined convex 2D curve [7, 9], tackled by sparse linear system solvers. Such a linear strategy is appreciated for its simplicity, efficiency, and the guarantee of bijectivity. However, the rigidity of fixing boundary points in the parameter domain usually leads to significant distortions. In response to these issues, free-boundary parameterization algorithms [33, 19] have been proposed, offering a more flexible setting by relaxing boundary constraints. Although these approaches bring certain improvements, they often struggle to maintain global bijectivity. More recent state-of-the-art approaches [29, 35] shift towards minimizing simpler proxy energies by alternating between local and global optimization phases, which not only accelerate convergence but also enhance the overall parameterization quality. OptCuts [16] explores joint optimization of surface cutting and mapping distortion. Despite the continuous advancements, the working mechanisms of all these approaches fundamentally rely on the connectivity information inherent in mesh structures, casting doubt on their applicability to unstructured point clouds. Moreover, mesh-oriented parameterization approaches typically assume that the inputs are well-behaved meshes possessing relatively regular triangulations. When faced with input meshes of inferior quality characterized by irregular triangulations and anomalies, remeshing becomes an indispensable procedure. Additionally, how to deal with non-ideal geometries and complex topologies (e.g., non-manifolds, multiply-connected components, thin structures) has always been highly challenging.

Unlike surface meshes, unstructured point clouds lack information regarding the connectivity between points, and thus are much more difficult to parameterize. Hence, only a relatively limited amount of prior works [43] have focused on the parameterization of surface point clouds. Early studies investigate various specialized parameterization strategies for disk-type [10, 1, 40], genus-0 [44, 17], genus-1 [37], well-sampled [13] or low-quality [22] point clouds. Later approaches achieve fixed-boundary spherical/rectangular parameterization through approximating the Laplace-Beltrami operators [5] or Teichmüller extremal mappings [23]. More recently, FBCP-PC [4] further pursues free-boundary conformal parameterization, in which a new point cloud Laplacian approximation scheme is proposed for handling boundary non-convexity.

2.2 Neural Parameterization Approaches

Driven by the remarkable success of deep learning, there is a family of recent works applying neural networks to learn parameterized 3D geometric representations. FoldingNet [39] proposes to deform a uniform 2D grid to reconstruct the target 3D point cloud for unsupervised geometric feature learning. AtlasNet [11] applies multi-grid deformation to learn locally parameterized representations. Subsequently, a series of follow-up researches inherit such a "folding-style" parameterization paradigm as pioneered by [39, 11] and investigate different aspects of modifications. GTIF [3] introduces graph topology inference and filtering mechanisms, empowering the decoder to preserve more representative geometric features in the latent space. EleStruc [6] proposes to perform shape reconstruction from learnable 3D elementary structures, rather than a pre-defined 2D lattice. Similarly, TearingNet [27] adaptively breaks the edges of an initial primitive graph for emulating the topology of the target 3D point cloud, which can effectively deal with higher-genus or multi-object inputs. In fact, the ultimate goal of the above line of approaches is to learn expressive shape codewords by means of deformation-driven 3D surface reconstruction. The characteristics of surface parameterization, i.e., the mapping process between 3D surfaces and 2D parameter domains, are barely considered.

In contrast to the extensive research in the fields of deep learning-based 3D geometric reconstruction and feature learning, there only exist a few studies that particularly focus on neural surface parameterization. DiffSR [2] adopts the basic multi-patch reconstruction framework [11] and explicitly regularizes multiple differential surface properties. NSM [25] explores neural encoding of surface maps by overfitting a neural network to an existing UV parameterization pre-computed via standard mesh parameterization algorithms [38, 29]. DA-Wand [20] constructs a parameterization-oriented mesh segmentation framework. Around a specified triangle, it learns to select a local sub-region, which is supposed to be sufficiently developable to produce low-distortion parameterization. Inheriting the geometry image (GI) [12] representation paradigm, RegGeoNet [41] and Flattening-Net [42] propose to learn deep regular representations of unstructured 3D point clouds. However, these two approaches lack explicit constraints on the parameterization distortions, and their local-to-global assembling procedures are hard to control. More recently, Nuvo [36] proposes a neural UV mapping framework that operates on oriented 3D points sampled from arbitrary 3D representations, liberating from the stringent quality demands of mesh triangulation. This approach assigns the original surface to multiple charts and ignores the packing procedure, thus essentially differing from our targeted global parameterization setting.

3 Proposed Method

Our FAM operates on discrete spatial points lying on the target 3D geometric surface for achieving global free-boundary parameterization in an unsupervised learning manner. Denote by $\mathbf{P} \in \mathbb{R}^{N\times 3}$ a set of unstructured points without any connectivity information, we aim at point-wisely parameterizing (i.e., row-wisely mapping) $\mathbf{P}$ onto the planar domain to produce UV coordinates $\mathbf{Q} \in \mathbb{R}^{N\times 2}$.

Technical Motivation. The most straightforward way of learning such a surface flattening process is to directly impose certain appropriate regularizers over the resulting 2D UV coordinates $\mathbf{Q}$. *However, due to the missing of ground-truths, it is impossible to explicitly supervise the generation of* $\mathbf{Q}$*, and it empirically turns out that designing and minimizing such regularizers cannot produce satisfactory results.* Alternatively, another feasible scheme is to choose an opposite mapping direction to wrap the adaptively-deformed and potentially-optimal 2D UV coordinates $\hat{\mathbf{Q}} \in \mathbb{R}^{N\times 2}$ onto the target surface by generating a new set of unstructured points $\hat{\mathbf{P}} \in \mathbb{R}^{N\times 3}$, which should approximate the input $\mathbf{P}$ by minimizing certain point set similarity metrics. Under ideal situations where $\hat{\mathbf{P}}$ and $\mathbf{P}$ are losslessly matched, we can equivalently deduce the desired UV coordinate of each point in $\mathbf{P}$ (i.e., for a certain 3D point $\mathbf{p}_i \in \mathbf{P}$ perfectly matched with a generated 3D point $\hat{\mathbf{p}}_j \in \hat{\mathbf{P}}$, which is row-wisely mapped from $\hat{\mathbf{q}}_j \in \hat{\mathbf{Q}}$, then we know the UV coordinate of $\mathbf{p}_i$ should be $\hat{\mathbf{q}}_j$). *However, due to the practical difficulty of reconstructing point sets with high-precision point-wise matching, the generated* $\hat{\mathbf{P}}$ *is only able to coarsely recover the target surface, thereby* $\hat{\mathbf{Q}}$ *cannot be treated as the resulting 2D UV coordinates of the input* $\mathbf{P}$. Still, despite the inaccuracy of such inverse 2D-to-3D wrapping process, it provides valuable cues indicating how the target 3D surface is roughly transformed from the 2D parameter domain, which naturally motivates us to combine the two opposite mapping processes into a bi-directional joint learning architecture. *To build associations between the unwrapping and wrapping processes and promote mapping bijectivity, we further introduce cycle mapping mechanisms*

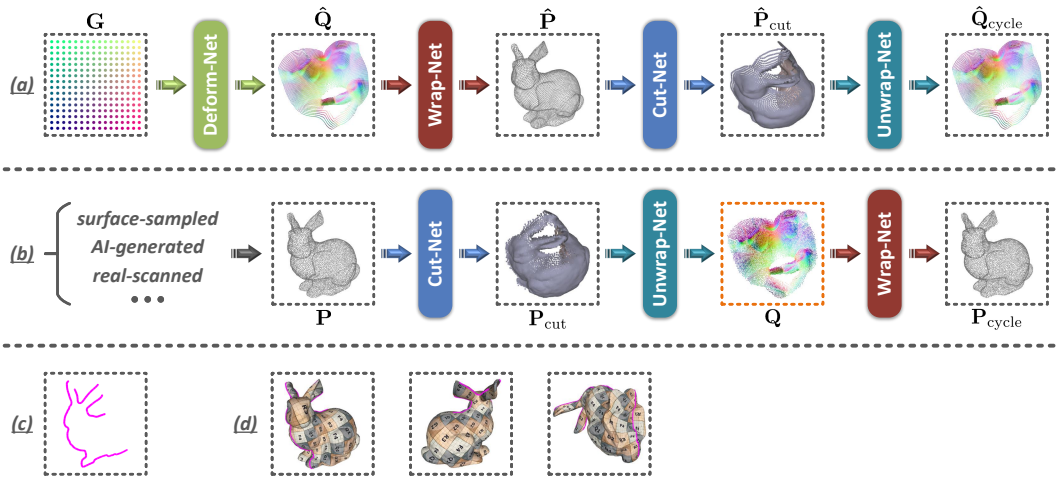


Figure 2: Illustration of bi-directional cycle mapping, composed of (a) 2D→3D→2D cycle mapping branch, and (b) 3D→2D→3D cycle mapping branch. Modules with the same color share network parameters. (c) shows the learned cutting seams. (d) shows the checker-image texture mapping.

and impose consistency constraints. Thus, our FAM is designed as a bi-directional cycle mapping workflow, in which all sub-networks are *parameter-sharing* and *jointly-optimized*.

3.1 Bi-directional Cycle Mapping

As illustrated in Figure 2, there are two parallel learning branches comprising a series of geometrically-meaningful sub-networks built upon point-wisely shared multi-layer perceptrons (MLPs):

- The deforming network $\mathcal{M}_d$ (**Deform-Net**) takes a uniform 2D lattice as input, and deforms the initial grid points to potentially-optimal UV coordinates.
- The wrapping network $\mathcal{M}_w$ (**Wrap-Net**) wraps the potentially-optimal 2D UV coordinates onto the target 3D geometric surface.
- The surface cutting network $\mathcal{M}_c$ (**Cut-Net**) finds appropriate cutting seams for transforming the original 3D geometric structure to an open and highly-developable surface manifold.
- The unwrapping network $\mathcal{M}_u$ (**Unwrap-Net**) assumes that its input surface has already been pre-opened, and flattens the 3D points onto the 2D parameter domain.

3.1.1 2D→3D→2D Cycle Mapping

The upper learning branch begins with a pre-defined 2D lattice with N grid points uniformly sampled within $[-1, 1]^2$, denoted as $\mathbf{G} \in \mathbb{R}^{N\times 2}$. To perform adaptive UV space deformation, we employ an offset-based coordinate updating strategy by feeding $\mathbf{G}$ into the Deform-Net to produce another set of 2D UV coordinates $\hat{\mathbf{Q}} \in \mathbb{R}^{N\times 2}$, which can be formulated as:

$$\hat{\mathbf{Q}} = \mathcal{M}_d(\mathbf{G}) = \xi''_d([\xi'_d(\mathbf{G}); \mathbf{G}]) + \mathbf{G}, \tag{1}$$

where $[*;*]$ denotes channel concatenation, $\xi'_d : \mathbb{R}^2 \to \mathbb{R}^h$ and $\xi''_d : \mathbb{R}^{(h+2)} \to \mathbb{R}^2$ are stacked MLPs. Intuitively, the initial 2D grid points are first embedded through ξ'_d into the h-dimensional latent space, concatenated with itself, and then mapped onto the 2D planar domain through ξ''_d. The learned offsets are point-wisely added to the initial grid points to produce the resulting $\hat{\mathbf{Q}}$.

After that, we perform 2D-to-3D wrapping by feeding the generated $\hat{\mathbf{Q}}$ into the Wrap-Net to produce $\hat{\mathbf{P}} \in \mathbb{R}^{N\times 3}$, which is expected to roughly approximate the target 3D geometric structure represented by the input 3D point set $\mathbf{P}$. In the meantime, we use three more output channels to produce normals $\hat{\mathbf{P}}^{\mathbf{n}} \in \mathbb{R}^{N\times 3}$. The whole network behavior can be described as:

$$[\hat{\mathbf{P}}; \hat{\mathbf{P}}^{\mathbf{n}}] = \mathcal{M}_w(\hat{\mathbf{Q}}) = \xi''_w([\xi'_w(\hat{\mathbf{Q}}); \hat{\mathbf{Q}}]), \tag{2}$$

where $\xi'_w : \mathbb{R}^2 \to \mathbb{R}^h$ and $\xi''_w : \mathbb{R}^{(h+2)} \to \mathbb{R}^6$ are stacked MLPs, and the resulting 6 output channels are respectively split into $\hat{\mathbf{P}}$ and $\hat{\mathbf{P}}^{\mathbf{n}}$.

In order to construct cycle mapping, we further apply the Cut-Net on the reconstructed $\hat{\mathbf{P}}$, which is transformed into an open 3D surface manifold $\hat{\mathbf{P}}_{\text{cut}} \in \mathbb{R}^{N\times 3}$, as given by:

$$\hat{\mathbf{P}}_{\text{cut}} = \mathcal{M}_c(\hat{\mathbf{P}}) = \xi_c''([\xi_c'(\hat{\mathbf{P}}); \hat{\mathbf{P}}]) + \hat{\mathbf{P}}, \tag{3}$$

where $\xi_c' : \mathbb{R}^3 \rightarrow \mathbb{R}^h$ and $\xi_c'' : \mathbb{R}^{(h+3)} \rightarrow \mathbb{R}^3$ are stacked MLPs.

Once we obtain the highly-developable 3D surface manifold $\hat{\mathbf{P}}_{\text{cut}}$, it is straightforward to perform 3D-to-2D flattening through the Unwrap-Net, which can be formulated as:

$$\hat{\mathbf{Q}}_{\text{cycle}} = \mathcal{M}_u(\hat{\mathbf{P}}_{\text{cut}}) = \xi_u(\hat{\mathbf{P}}_{\text{cut}}), \tag{4}$$

where $\xi_u : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ is simply implemented as stacked MLPs, and the resulting $\hat{\mathbf{Q}}_{\text{cycle}} \in \mathbb{R}^{N\times 2}$ is expected to be row-wisely equal to $\hat{\mathbf{Q}}$.

3.1.2 3D→2D→3D Cycle Mapping

The bottom learning branch directly starts by feeding the input 3D point set $\mathbf{P}$ into the Cut-Net to produce an open 3D surface manifold $\mathbf{P}_{\text{cut}}$, which can be formulated as:

$$\mathbf{P}_{\text{cut}} = \mathcal{M}_c(\mathbf{P}) = \xi_c''([\xi_c'(\mathbf{P}); \mathbf{P}]) + \mathbf{P}. \tag{5}$$

After that, we apply the Unwrap-Net on the generated $\mathbf{P}_{\text{cut}}$ to produce the desired 2D UV coordinates $\mathbf{Q}$, as given by:

$$\mathbf{Q} = \mathcal{M}_u(\mathbf{P}_{\text{cut}}) = \xi_u(\mathbf{P}_{\text{cut}}). \tag{6}$$

In order to construct cycle mapping, we further apply the Wrap-Net on the generated $\mathbf{Q}$ for recovering the target 3D geometric structure, which can be formulated as:

$$[\mathbf{P}_{\text{cycle}}; \mathbf{P}^{\mathbf{n}}_{\text{cycle}}] = \mathcal{M}_w(\mathbf{Q}) = \xi_w''([\xi_w'(\mathbf{Q}); \mathbf{Q}]), \tag{7}$$

where $\mathbf{P}_{\text{cycle}}$ is expected to be row-wisely equal to $\mathbf{P}$. When the input surface points are accompanied by normals $\mathbf{P}^{\mathbf{n}} \in \mathbb{R}^{N\times 3}$, we also expect that the normal directions of $\mathbf{P}^{\mathbf{n}}$ and the generated $\mathbf{P}^{\mathbf{n}}_{\text{cycle}}$ should be row-wisely consistent.

Remark. Throughout the overall bi-directional cycle mapping framework, *only* $\mathbf{Q}$ *is regarded as the desired 2D UV coordinates that are row-wisely mapped from the input 3D surface points* $\mathbf{P}$, all the others (e.g., $\hat{\mathbf{Q}}$ and $\hat{\mathbf{Q}}_{\text{cycle}}$) just serve as intermediate results. Morevoer, it is also worth emphasizing that, *when the training is finished, any arbitrary surface-sampled 3D points can be fed into our FAM to obtain point-wise UV coordinates*.

Extraction of Cutting Seams. The extraction of points located on the learned cutting seams, which we denote as $\mathbf{E} = \{\mathbf{e}_i \in \mathbf{P}\}$, can be conveniently achieved by comparing the mapping relationships between the input $\mathbf{P}$ and the parameterized $\mathbf{Q}$.

Specifically, since $\mathbf{P}$ and $\mathbf{Q}$ are mapped in a row-wise manner, for each input 3D point $\mathbf{p}_i \in \mathbf{P}$ and its K-nearest neighbors $\{\mathbf{p}_i^{(k)}\}_{k=1}^{K_{\text{cut}}}$ within $\mathbf{P}$, we can directly know their 2D UV coordinates, denoted as $\mathbf{q}_i \in \mathbf{Q}$ and $\{\mathbf{q}_i^{(k)}\}_{k=1}^{K_{\text{cut}}}$. The maximum distance value η_i between $\mathbf{q}_i$ and its neighboring points can be computed as:

$$\eta_i = \max(\{\|\mathbf{q}_i - \mathbf{q}_i^{(k)}\|_2\}_{k=1}^{K_{\text{cut}}}). \tag{8}$$

Then, $\mathbf{p}_i$ will be determined to locate on the cutting seam if η_i is larger than a certain threshold T_{cut}. Collecting all such points within $\mathbf{P}$ can deduce a set of seam points $\mathbf{E}$, and it is observed that in this way no seam points would be identified if cutting the original 3D surface is actually unnecessary.

3.2 Training Objectives

As an unsupervised learning framework, FAM is trained by minimizing a series of carefully-designed objective functions and constraints, whose formulations and functionalities are introduced below.

Unwrapping Loss. The most fundamental requirement for $\mathbf{Q}$ is that any two parameterized coordinates cannot overlap. Hence, for each 2D UV coordinate $\mathbf{q}_i$, we search its K-nearest-neighbors $\{\mathbf{q}_i^{(k)}\}_{k=1}^{K_u}$, and penalize neighboring points that are too close, as given by:

$$\ell_{\text{unwrap}} = \sum_{i=1}^{N}\sum_{k=1}^{K_{\text{u}}} \max(0, \epsilon - \|\mathbf{q}_i - \mathbf{q}_i^{(k)}\|_2), \tag{9}$$

where the minimum distance value allowed between neighboring points is controlled by a threshold ϵ.

Wrapping Loss. The major supervision of the upper learning branch is that the generated point set $\hat{\mathbf{P}}$ should roughly approximate the original 3D geometric structure represented by $\mathbf{P}$. Since $\hat{\mathbf{P}}$ and $\mathbf{P}$ are not row-wisely corresponded, we use the commonly-used Chamfer distance $\texttt{CD}(*;*)$ to measure point set similarity, as given by:

$$\ell_{\text{wrap}} = \texttt{CD}(\hat{\mathbf{P}}; \mathbf{P}). \tag{10}$$

Here, there is no need to impose the unwrapping loss over $\hat{\mathbf{P}}$, since we empirically find that optimizing the CD loss naturally suppresses clustered point distribution. Besides, we do not optimize $\texttt{CD}(\hat{\mathbf{Q}}; \mathbf{Q})$, since we observe degraded performances, possibly because in earlier training iterations $\hat{\mathbf{Q}}$ and $\mathbf{Q}$ are far from optimal and thus can interfere with each other.

Cycle Consistency Loss. Our bi-directional cycle mapping framework aims to promote consistencies within both 3D and 2D domains, which are natural to learn thanks to our symmetric designs of the functionalities of sub-networks. Thus, the cycle consistency loss is formulated as:

$$\ell_{\text{cycle}} = \|\mathbf{P} - \mathbf{P}_{\text{cycle}}\|_1 + \|\hat{\mathbf{Q}} - \hat{\mathbf{Q}}_{\text{cycle}}\|_1 + \texttt{CS}(\mathbf{P^n}; \mathbf{P^n_{\text{cycle}}}), \tag{11}$$

where $\texttt{CS}(*;*)$ computes point-wise cosine similarity between $\mathbf{P^n_{\text{cycle}}}$ and ground-truth normals $\mathbf{P^n}$, which can be removed when ground-truth normals are not provided or hard to compute.

Mapping Distortion Constraint. We regularize the distortion of the learned neural mapping function by exploiting differential surface properties, which can be conveniently deduced via the automatic differentiation mechanisms in common deep learning programming frameworks.

Throughout the bi-directional cycle mapping framework, 2D UV coordinates $\mathbf{Q}$ and $\hat{\mathbf{Q}}$ are respectively mapped to 3D surface points $\mathbf{P}_{\text{cycle}}$ and $\hat{\mathbf{P}}$. For convenience, here we denote the two neural mapping functions as $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ and $g : \mathbb{R}^2 \rightarrow \mathbb{R}^3$. We compute the derivatives of f and g with respect to $\mathbf{Q}$ and $\hat{\mathbf{Q}}$ at each 2D UV point (u, v) to obtain the Jacobian matrices $\mathbf{J}_f \in \mathbb{R}^{3\times2}$ and $\mathbf{J}_g \in \mathbb{R}^{3\times2}$:

$$\mathbf{J}_f = (f_u \;\; f_v), \; \mathbf{J}_g = (g_u \;\; g_v) \tag{12}$$

where f_u, f_v, g_u, g_v are three-dimensional vectors of partial derivatives. Then we further deduce the eigenvalues of $\mathbf{J}_f^T\mathbf{J}_f$ and $\mathbf{J}_g^T\mathbf{J}_g$, which are denoted as $(\lambda_f^1, \lambda_f^2)$ and $(\lambda_g^1, \lambda_g^2)$. Thus, we can promote conformal (i.e., angle-preserving) parameterizations by constraining the following regularizer:

$$\ell_{\text{conf}} = \sum_{\mathbf{q}\in\mathbf{Q}} \|\lambda_f^1 - \lambda_f^2\|_1 + \sum_{\hat{\mathbf{q}}\in\hat{\mathbf{Q}}} \|\lambda_g^1 - \lambda_g^2\|_1. \tag{13}$$

4 Experiments

We collected a series of 3D surface models with different types and complexities of geometric and/or topological structures for experimental evaluation. We made qualitative and quantitative comparisons with SLIM [29], a state-of-the-art and widely-used mesh parameterization algorithm. Additionally, we also compared our FAM with FBCP-PC [4] for unstructured and unoriented (i.e., without normals) 3D point cloud parameterization. Finally, we conducted ablation studies to demonstrate the effectiveness of the proposed bi-directional cycle mapping framework and the necessity of Cut-Net, together with different aspects of verifications for comprehensively understanding the characteristics of our approach. Due to page limitations, detailed technical implementations and additional experimental results are presented in our Appendix

Comparison with SLIM [29]. Considering that SLIM is only able to be applied on disk-topologies or surfaces with open boundaries, we collected 8 such testing models for parameterization, using its officially-released code. For the training of our FAM, we uniformly sample $10,000$ points from the original mesh vertices at each optimization iteration. During testing, the whole vertex set is fed into our FAM to produce per-vertex UV coordinates. Then, we can perform checker-map texture mappings for qualitative evaluation. For quantitative evaluation, we computed conformality metrics (*the lower, the better*) by measuring the average of the absolute angle differences between each corresponding angle of the 3D triangles and the 2D parameterized triangles. As shown in Figure 3 and Table 1, our FAM outperforms SLIM both qualitatively and quantitatively on all testing models.

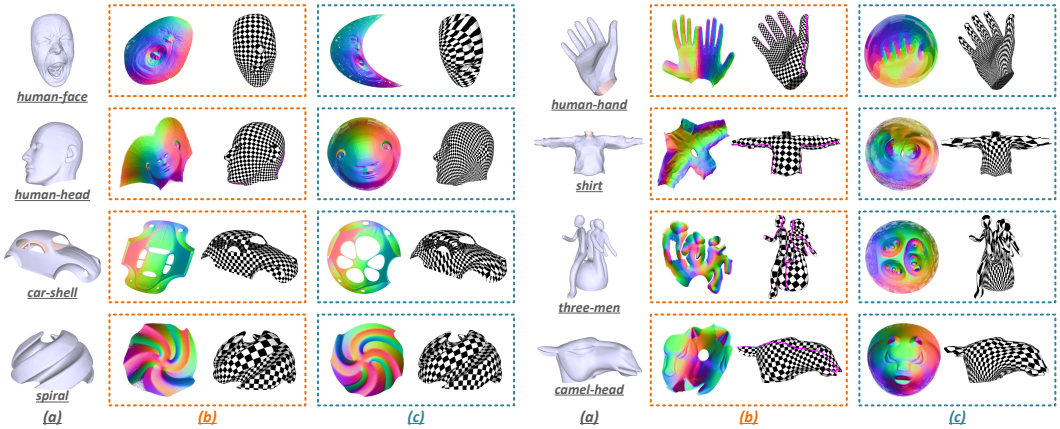


Figure 3: Comparison of UV unwrapping and texture mapping results on different (a) open surface models produced by (b) our FAM and (c) SLIM, where the 2D UV coordinates are color-coded by ground-truth point-wise normals to facilitate visualization.

Table 1: Quantitative comparisons of our FAM and SLIM in terms of parameterization conformality.

Model	*human-face*	*human-head*	*car-shell*	*spiral*	*human-hand*	*shirt*	*three-men*	*camel-head*
SLIM	0.635	0.254	0.411	0.114	0.609	0.443	0.645	0.349
FAM	**0.074**	**0.094**	**0.037**	**0.087**	**0.145**	**0.166**	**0.162**	**0.088**

Table 2: Quantitative conformality metrics of our parameterization results.

Model	*brain*	*cow*	*fandisk*	*human-body*	*lion*	*nail*	*nefertiti*
Conf. Metric	0.263	0.210	0.105	0.198	0.192	0.162	0.117
Model	*bucket*	*dragon*	*mobius-strip*	*rocker-arm*	*torus-double*	*three-holes*	*fertility*
Conf. Metric	0.179	0.311	0.062	0.143	0.126	0.142	0.166

Table 3: Conformality metrics of our FAM and FBCP-PC for point cloud parameterization.

Model	*cloth-pts (#Pts=7K)*	*julius-pts (#Pts=11K)*	*spiral-pts (#Pts=28K)*
FBCP-PC	**0.021**	**0.019**	**0.023**
FAM	0.037	0.058	0.117

Parameterization with Different Geometric and Topological Complexities. We conducted more comprehensive experimental evaluations, as displayed in Figure 4 and Table 2, showing the universality and robustness of our approach.

Comparison with FBCP-PC [4]. Since FAM operates on discrete surface-sampled points without any dependence on connectivity information, applying our approach to parameterize unstructured and unoriented 3D point clouds is straightforward and basically seamless. The only modification is to remove the supervision of point-wise normals, i.e., the last term $\texttt{CS}(\mathbf{P^n}; \mathbf{P^n_{\text{cycle}}})$ in Eqn. (11). As compared in Figure 5 and Table 3, our performances are not better than FBCP-PC. However, we must point out that FBCP-PC requires manually specifying indices of boundary points (arranged in order) as additional inputs. Hence, the comparisons are actually quite unfair to us.

Ablation Studies. We evaluated the necessity of our two-branch joint learning architecture. First, we removed the upper 2D→3D→2D branch to show the resulting $\mathbf{Q}$. Second, we removed the bottom 3D→2D→3D branch to show UV coordinates $\hat{\mathbf{Q}}$ by performing nearest-neighbor matching between $\mathbf{P}$ and $\hat{\mathbf{P}}$. As illustrated in Figure 6, removing any of the two branches will cause different degrees of performance degradation. Furthermore, we verified the necessity of Cut-Net in the whole learning pipeline. As shown in Figure. 7, removing Cut-Net does not lead to obvious performance degradation for models that are simpler to open (e.g., *human-face* and *mobius-strip*), yet for the other complex models the removal of Cut-Net causes highly-distorted surface flattening. Without learning offsets, the inherent smoothness of neural works can impede "tearing" the originally-continuous areas on the 3D surface, thus hindering the creation of cutting seams.

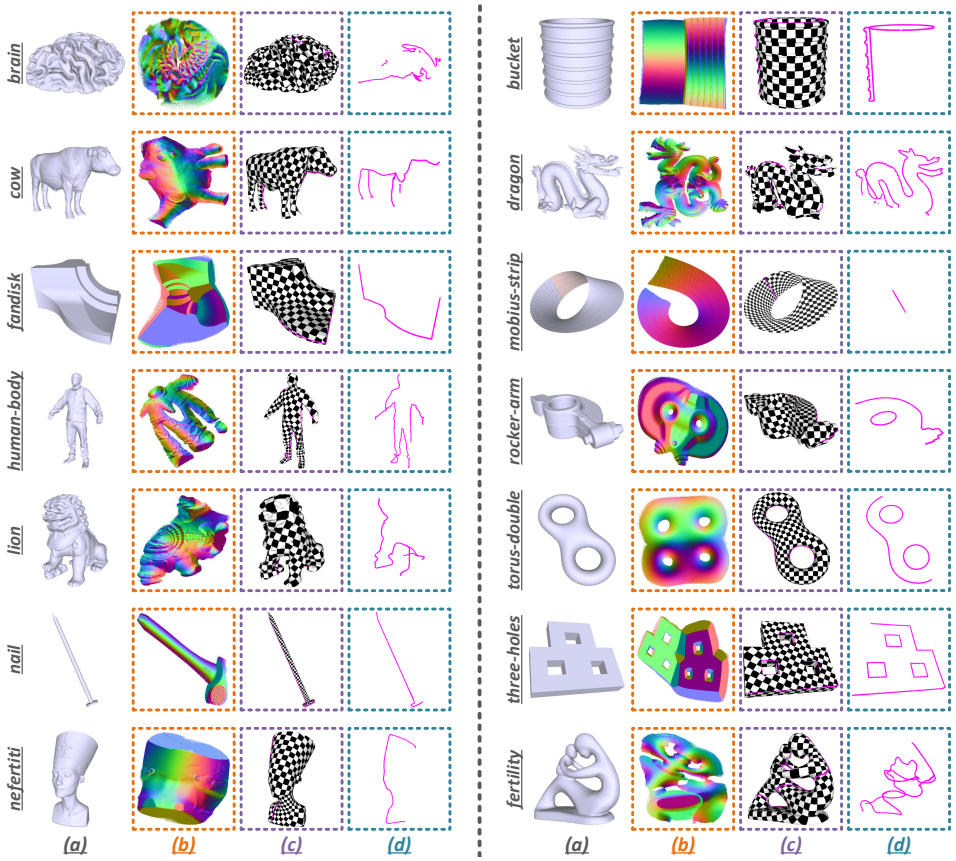


Figure 4: Display of surface parameterization results produced by our FAM. (a) input 3D models; (b) learned UV coordinates; (c) texture mappings; (d) learned cutting seams.

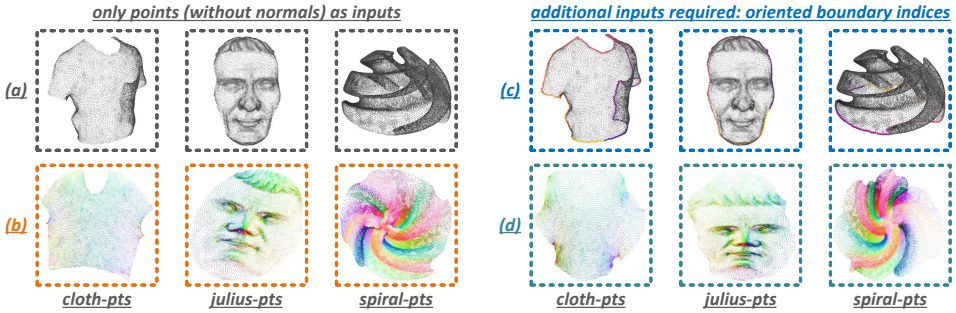


Figure 5: Point cloud parameterization achieved by our FAM (left) and FBCP-PC (right).

In addition, we conducted quantitative evaluations of self-intersection. Given a triangular mesh, we check if any pair of triangles overlaps in the UV space, and then measure the proportion of overlapped pairs to the total amount of triangle pairs. As reported in Table 4, self-intersection are inevitable both in FAM and SLIM, but the proportions of self-intersected triangles are very small. In practice, it is not hard to apply post-processing refinement to slightly adjust the distribution of UV coordinates to further relieve or even eliminate self-intersection issues. For point cloud parameterization, we evaluated our robustness to noises in Figure 8, where we can observe that FAM shows stable performances to noisy input conditions (with 1%, 2%, and 4% Gaussian noises for point position perturbation).

Finally, we performed stress tests on a Hilbert-space-filling-shaped cylinder model in Figure 9a. It is observed that our FAM obtains the basically optimal solution. Still, as shown in Figure 9b, processing the highly-complicated ShapeNet-style CAD model with rich interior structures and many multi-layer issues shows inferior UV unwrapping quality. Although our learned cutting seams are generally reasonable and texture mapping looks relatively regular from outside, it is hard to deal with the complicated interior structures.

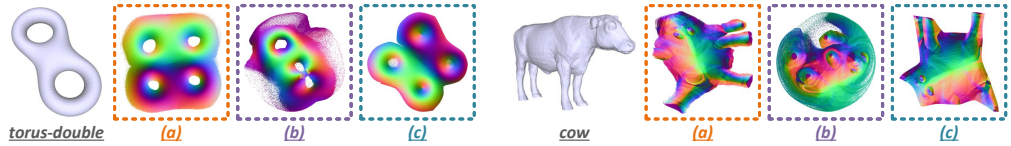


Figure 6: Ablation results produced by (a) our complete FAM framework, (b) FAM without the upper 2D→3D→2D learning branch, (c) FAM without the bottom 3D→2D→3D learning branch.

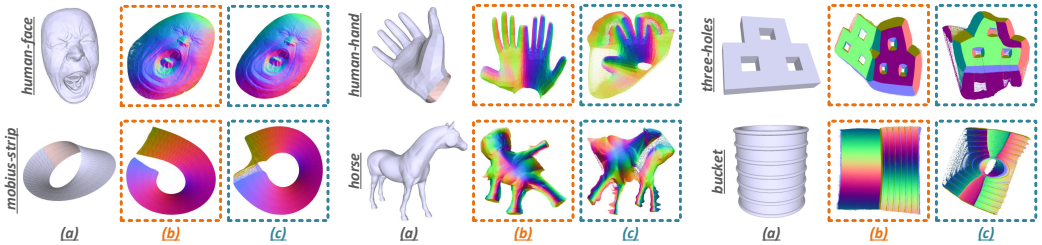


Figure 7: Ablation study on the necessity of Cut-Net: (a) input meshes; (b) results obtained from our full FAM architecture; (c) results obtained by removing the Cut-Net component.

Testing Models	FAM	SLIM
Open-Surface Models (as in Figure 3)	0.156%	0.133%
Higher-Genus Models (as in Figure 4)	0.204%	N/A

Table 4: Quantitative self-intersection metrics of our parameterization results.

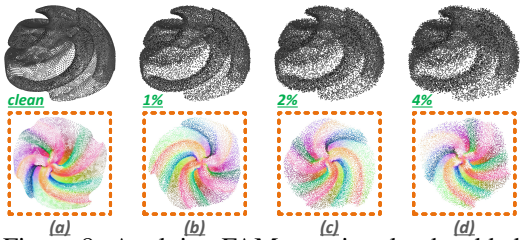
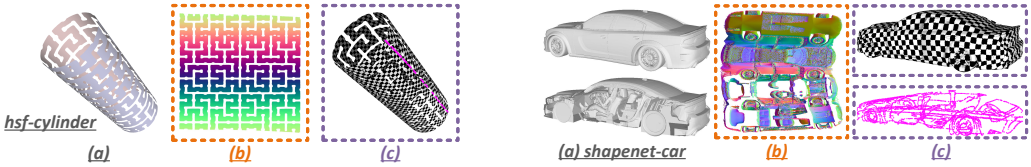


Figure 8: Applying FAM to point clouds added with different levels of Gaussian ***noises***.



(a) Highly challenging stress tests of our FAM. (b) Failure cases of complicated CAD models.

Figure 9: (a) input meshes; (b) UV maps; (c): texture mappings and learned cutting seams.

5 Conclusion

We proposed FAM, the first neural surface parameterization approach targeted at global free-boundary parameterization. Our approach is universal for dealing with different geometric/topological complexities regardless of mesh triangulation quality and even applicable to unstructured point clouds. More importantly, our approach automatically learns appropriate cutting seams along the target 3D surface, and adaptively deforms the 2D UV parameter domain. As an unsupervised learning framework, our FAM shows significant potentials and practical values. The current technical implementations still have several aspects of limitations. Our working mode of per-model overfitting cannot exploit existing UV unwrapping results (despite their limited amounts) as ground-truths for training generalizable neural models. Besides, a series of more advanced properties, such as shape symmetry, cutting seam visibility, seamless parameterization, have not yet been considered.

References

[1] P. Azariadis and N. Sapidis. Product design using point-cloud surfaces: A recursive subdivision technique for point parameterization. *Computers in Industry*, 58(8-9):832–843, 2007.

[2] J. Bednarik, S. Parashar, E. Gundogdu, M. Salzmann, and P. Fua. Shape reconstruction by learning differentiable surface representations. In *Proc. CVPR*, pages 4716–4725, 2020.

[3] S. Chen, C. Duan, Y. Yang, D. Li, C. Feng, and D. Tian. Deep unsupervised learning of 3d point clouds via graph topology inference and filtering. *IEEE Transactions on Image Processing*, 29:3183–3198, 2019.

[4] G. P. Choi, Y. Liu, and L. M. Lui. Free-boundary conformal parameterization of point clouds. *Journal of Scientific Computing*, 90(1):14, 2022.

[5] G. P.-T. Choi, K. T. Ho, and L. M. Lui. Spherical conformal parameterization of genus-0 point clouds for meshing. *SIAM Journal on Imaging Sciences*, 9(4):1582–1618, 2016.

[6] T. Deprelle, T. Groueix, M. Fisher, V. Kim, B. Russell, and M. Aubry. Learning elementary structures for 3d shape generation and matching. *Proc. NeurIPS*, 32, 2019.

[7] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proc. SIGGRAPH*, pages 173–182, 1995.

[8] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. In *Proc. SCG*, pages 244–253, 2002.

[9] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.

[10] M. S. Floater and M. Reimers. Meshless parameterization and surface reconstruction. *Computer Aided Geometric Design*, 18(2):77–92, 2001.

[11] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. A papier-mâché approach to learning 3d surface generation. In *Proc. CVPR*, pages 216–224, 2018.

[12] X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. In *Proc. SIGGRAPH*, pages 355–361, 2002.

[13] X. Guo, X. Li, Y. Bao, X. Gu, and H. Qin. Meshless thin-shell simulation based on global conformal parameterization. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):375–385, 2006.

[14] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023.

[15] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21(3):362–371, 2002.

[16] M. Li, D. M. Kaufman, V. G. Kim, J. Solomon, and A. Sheffer. Optcuts: Joint optimization of surface cuts and parameterization. *ACM Transactions on Graphics*, 37(6):1–13, 2018.

[17] J. Liang, R. Lai, T. W. Wong, and H. Zhao. Geometric understanding of point clouds using laplace-beltrami operator. In *Proc. CVPR*, pages 214–221, 2012.

[18] C.-H. Lin, J. Gao, L. Tang, T. Takikawa, X. Zeng, X. Huang, K. Kreis, S. Fidler, M.-Y. Liu, and T.-Y. Lin. Magic3d: High-resolution text-to-3d content creation. In *Proc. CVPR*, pages 300–309, 2023.

[19] L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. J. Gortler. A local/global approach to mesh parameterization. *Computer Graphics Forum*, 27(5):1495–1504, 2008.

[20] R. Liu, N. Aigerman, V. G. Kim, and R. Hanocka. Da wand: Distortion-aware selection using neural mesh parameterization. In *Proc. CVPR*, pages 16739–16749, 2023.

[21] F. Losasso, H. Hoppe, S. Schaefer, and J. Warren. Smooth geometry images. In *Symposium on Geometry Processing*, volume 43, pages 138–145, 2003.

[22] Q. Meng, B. Li, H. Holstein, and Y. Liu. Parameterization of point-cloud freeform surfaces using adaptive sequential learning rbfnetworks. *Pattern Recognition*, 46(8):2361–2375, 2013.

[23] T. W. Meng, G. P.-T. Choi, and L. M. Lui. Tempo: feature-endowed teichmuller extremal mappings of point clouds. *SIAM Journal on Imaging Sciences*, 9(4):1922–1962, 2016.

[24] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proc. ECCV*, pages 405–421, 2020.

[25] L. Morreale, N. Aigerman, V. G. Kim, and N. J. Mitra. Neural surface maps. In *Proc. CVPR*, pages 4639–4648, 2021.

[26] A. Nichol, H. Jun, P. Dhariwal, P. Mishkin, and M. Chen. Point-e: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022.

[27] J. Pang, D. Li, and D. Tian. Tearingnet: Point cloud autoencoder to learn topology-friendly representations. In *Proc. CVPR*, pages 7453–7462, 2021.

[28] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *Proc. ICLR*, 2023.

[29] M. Rabinovich, R. Poranne, D. Panozzo, and O. Sorkine-Hornung. Scalable locally injective mappings. *ACM Transactions on Graphics*, 36(4):1, 2017.

[30] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In *Proc. ICML*, pages 5301–5310, 2019.

[31] A. Sheffer and E. de Sturler. Parameterization of faceted surfaces for meshing using angle-based flattening. *Engineering with Computers*, 17:326–337, 2001.

[32] A. Sheffer and J. C. Hart. Seamster: inconspicuous low-distortion texture seam layout. In *IEEE Visualization*, pages 291–298, 2002.

[33] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov. Abf++: fast and robust angle based flattening. *ACM Transactions on Graphics*, 24(2):311–330, 2005.

[34] Y. Siddiqui, A. Alliegro, A. Artemov, T. Tommasi, D. Sirigatti, V. Rosov, A. Dai, and M. Nießner. Meshgpt: Generating triangle meshes with decoder-only transformers. *arXiv preprint arXiv:2311.15475*, 2023.

[35] J. Smith and S. Schaefer. Bijective parameterization with free boundaries. *ACM Transactions on Graphics*, 34(4):70:1–70:9, 2015.

[36] P. P. Srinivasan, S. J. Garbin, D. Verbin, J. T. Barron, and B. Mildenhall. Nuvo: Neural uv mapping for unruly 3d representations. In *Proc. ECCV*, 2024.

[37] G. Tewari, C. Gotsman, and S. J. Gortler. Meshing genus-1 point clouds using discrete one-forms. *Computers & Graphics*, 30(6):917–926, 2006.

[38] W. T. Tutte. How to draw a graph. *Proc. LMS*, 3(1):743–767, 1963.

[39] Y. Yang, C. Feng, Y. Shen, and D. Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proc. CVPR*, pages 206–215, 2018.

[40] L. Zhang, L. Liu, C. Gotsman, and H. Huang. Mesh reconstruction by meshless denoising and parameterization. *Computers & Graphics*, 34(3):198–208, 2010.

[41] Q. Zhang, J. Hou, Y. Qian, A. B. Chan, J. Zhang, and Y. He. Reggeonet: Learning regular representations for large-scale 3d point clouds. *International Journal of Computer Vision*, 130(12):3100–3122, 2022.

[42] Q. Zhang, J. Hou, Y. Qian, Y. Zeng, J. Zhang, and Y. He. Flattening-net: Deep regular 2d representation for 3d point cloud analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[43] Z. Zhu, A. Iglesias, L. You, and J. J. Zhang. A review of 3d point clouds parameterization methods. In *Proc. ICCS*, pages 690–703, 2022.

[44] M. Zwicker and C. Gotsman. Meshing point clouds using spherical parameterization. In *Proc. PBG*, pages 173–180, 2004.

A Appendix

A.1 Implementation Details

In our bi-directional cycle mapping framework, all sub-networks are architecturally built upon stacked MLPs without batch normalization. We uniformly configured LeakyReLU non-linearities with the negative slope of 0.01, except for the output layer. Within the Deform-Net, ξ_d' and ξ_d'' are four-layer MLPs with channels $[2, 512, 512, 512, 64]$ and $[66, 512, 512, 512, 2]$. Within the Wrap-Net, ξ_w' and ξ_w'' are four-layer MLPs with channels $[2, 512, 512, 512, 64]$ and $[66, 512, 512, 512, 6]$. Within the Cut-Net, ξ_c' and ξ_c'' are three-layer MLPs with channels $[3, 512, 512, 64]$ and $[67, 512, 512, 3]$. Within the Unwrap-Net, ξ_u is implemented as three-layer MLPs with channels $[3, 512, 512, 2]$. In addition to network structures, there is also a set of hyperparameters to be configured and tuned. As presented in Eqn. (8) for cutting seam extraction, we chose $K_{\text{cut}} = 3$. Suppose that, at the current training iteration, the side length of the square bounding box of 2D UV coordinates $\mathbf{Q}$ is denoted as $L(\mathbf{Q})$. Then we set the threshold T_{cut} to be 2% of $L(\mathbf{Q})$. Besides, as presented in Eqn. (9) for computing the unwrapping loss, we choose the threshold as $\epsilon = 0.2 \cdot L(\mathbf{Q})/\sqrt{N}$, and $K_{\text{u}} = 8$. When formulating the overall training objective, the weights for ℓ_{unwrap}, ℓ_{wrap}, ℓ_{cycle}, and ℓ_{conf} are set as 0.01, 1.0, 0.01, and 0.01. All our experiments are conducted on a single NVIDIA GeForce RTX 3090 GPU.

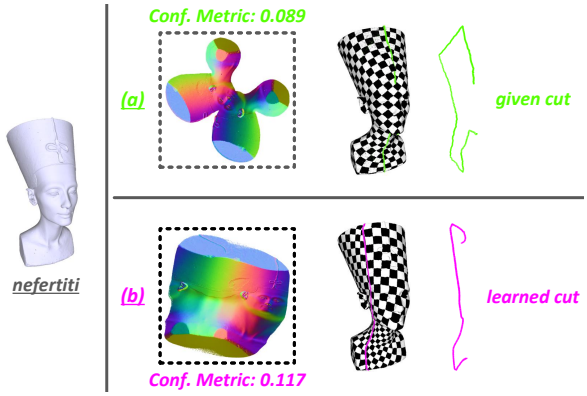


Figure 10: Experiments on the *nefertiti* testing model whose topology is not homeomorphic to a disk. (a) The parameterization results of SLIM are obtained by ***manually-specifying*** a high-quality cutting seam, with the conformality metric of 0.089. (b) The parameterization results of our FAM, in which the cutting seam is automatically learned, with the conformality metric of 0.117.

Table 5: Optimization time costs (minutes) of our FAM and SLIM.

Model	*human-face*	*human-head*	*car-shell*	*spiral*	*human-hand*	*shirt*	*three-men*	*camel-head*
SLIM	39 min	38 min	19 min	25 min	28 min	31 min	17 min	40 min
FAM	around 18 min (basically unchanged for different models)							

A.2 Discussions about Global and Multi-Chart Surface Parameterization

Over the years, global surface parameterization has continuously been the mainstream direction of research, since it potentially achieves smooth transitions and uniform distribution of mapping distortion across the entire surface, while multi-chart parameterization typically introduces discontinuities along patch boundaries, causing more obvious visual artifacts for texture mapping (perhaps the most important application scenario in the graphics field) and bringing additional difficulties in many other shape analysis tasks (e.g., remeshing, morphing, compression). For multi-chart parameterization, it is worth emphasizing that only obtaining chart-wise UV maps is not the complete workflow. We need to further perform chart packing to compose the multiple UV domains, without which the actual usefulness can be largely weakened. However, packing is known as a highly non-trivial problem, which is basically skipped in recent neural learning approaches, such as DiffSR and Nuvo. Still, local parameterization does have its suitable application scenarios for processing highly-complicated surfaces such as typical ShapeNet-style CAD meshes with rich interior structures and severe multi-layer

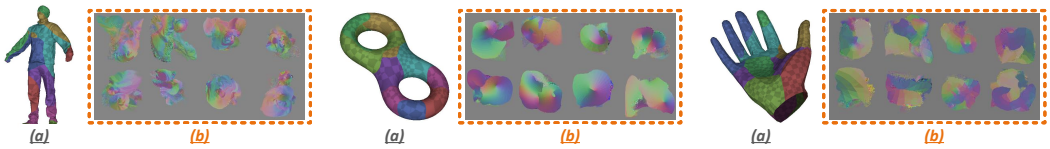


Figure 11: Results of Nuvo: (a) input meshes (different colors denote chart assignment); (b) chart-wise UV maps.

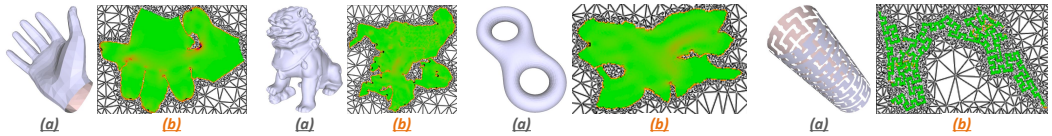


Figure 12: Results of OptCuts: (a) input meshes; (b) UV maps.

issues, and per-chart distortion can be reduced since the geometry and topology of cropped surface patches become simpler.

A.3 Comparison with SLIM on Non-Disk-Type Surfaces with Manually-Specified Cuts

In the main manuscript, our experimental comparisons with SLIM focus on 3D models with open boundaries and/or disk topologies, on which SLIM can be applied. Here, we made additional efforts to conduct comparisons with SLIM on the non-disk-type 3D surface model of *nefertiti* by manually specifying an optimal cutting seam for SLIM to run. Note that this is a quite **unfair** experimental setting, because finding optimal cuts is known to be a critical yet complicated task. As illustrated in Figure 10, our approach still achieves highly competitive performances compared with SLIM even under such an unfair comparison setup.

A.4 Running Efficiency

In addition to parameterization quality, we further compared the time costs of our FAM and SLIM for training and optimization. Here, the official code of SLIM runs on the Intel(R) Core(TM) i7-9700 CPU. Table 5 lists the time costs for different testing models. Note that, since our FAM operates on surface-sampled points and we uniformly used the same number of input points (i.e., N), the time costs of our FAM basically maintain unchanged for different testing models. It turns out that our approach achieves satisfactory learning efficiency.

A.5 Comparison with Nuvo [36]

We provided typical parameterization results of Nuvo using a third-party implementation available at `https://github.com/ruiqixu37/Nuvo`, as presented in Figure 11. We can observe that its chart assignment capability is still not stable. It often occurs that some spatially-disconnected surface patches are assigned to the same chart. Moreover, the critical procedure of chart packing is actually ignored in Nuvo. Directly merging the rectangular UV domains is not a valid packing. A real-sense packing should adaptively adjust the positions, poses, and sizes of each UV domain via combinations of translation, rotation, and scaling.

A.6 Comparison with OptCuts [16]

We provided typical parameterization results of OptCuts, as presented in Figure 12. Comparatively, our neural parameterization paradigm still shows advantages in terms of flexibility (not limited to well-behaved meshes; applicable to point clouds), convenience (exploiting GPU parallelism; much easier for implementing and tuning), and parameterization smoothness (not limited to mesh vertices). Although OptCuts is able to jointly obtain reasonable surface cuts and UV unwrapping results, its performances are generally sub-optimal compared with ours.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract clearly indicates the contribution and advantages of our proposed novel neural surface parameterization approach.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations of our current technical implementations are analyzed in the end of Section 5. Efficiency is also compared in the Appendix.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: This paper is not a theoretical work.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The experimental setups and detailed technical implementations have been presented in the main manuscript and the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Although the code is not uploaded for now, we have explicitly claimed in the Abstract that our code will be publicly available.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We have detailedly introduced our technical implementations in the Appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Not applicable to our surface parameterization setup, which overfits for each given 3D model.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The specific types of GPU and CPU are reported in the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We conform all the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: There is no societal impact of our technical exploration of surface parameterization as a classic geometry processing task.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our work has no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [NA]

    Justification: We did not use existing assets that need to be particularly credited/mentioned.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: Our paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.