

A APPENDIX

A.1 REMARKS ON SHAPE DIAMETER FUNCTION (SHDF)

Our semantic-aware parameterization, explained in Sec.3.3, relies on computing a per-vertex semantic partition of the input mesh using shape diameter function (Shapira et al., 2008). To estimate per-vertex semantic label, we design and implement a method based on Shape Diameter Function (Shapira et al., 2008) that involves the following key steps.

Compute a per face local thickness (shape-diameter) field. For a surface sample p (face centroid or vertex), the value $\text{ShDF}(p)$ estimates the local object diameter by probing the interior along a cone centered on the inward normal at p . Let $\mathcal{R}(p) = \{r_i\}_{i=1}^R$ be a set of rays radiating from p inside a cone of half-angle α around the inward normal. For each ray r_i we compute the length ℓ_i of the intersection segment between r_i and the mesh interior (i.e., the distance from p to the first opposite intersection). We robustly aggregate the ensemble $\{\ell_i\}$ using a median (or trimmed mean) to obtain: $\text{ShDF}(p) = \text{median}(\{\ell_i\}_{i=1}^R)$. This median-based aggregation reduces sensitivity to spurious long or short intersections caused by complex local geometry. In practice, to estimate a scalar per surface element that measures local object thickness (intuitively, it presents the diameter of the object in the neighborhood of each sample), we use ray casting (using PyMeshLab (Muntoni & Cignoni, 2021)) in multiple directions per sample and return a per-face scalar. Then, the raw scalars are normalized to $[0, 1]$ and re-scaled via a monotonic compression (e.g., a log-like transform) to reduce dynamic range and improve cluster separability. In our implementation we use per-face ShDF evaluated at face centroids with the following defaults: cone angle $\theta = 120^\circ$ (as in the original work (Shapira et al., 2008)), number of rays $R = 60$, and ray sampling distributed uniformly within the cone.

Smoothing and normalization. Similar to (Shapira et al., 2008), we smooth raw ShDF samples on the mesh to promote spatial coherence and reduce pose-dependent noise. We apply a small number (e.g., 1-2) of iterations of Laplacian smoothing on the ShDF scalar field with area-weighted averaging. To balance values across scales and to boost contrast of small parts, we perform a normalization:

$$\tilde{\text{ShDF}}(p) = \frac{\log(1 + \text{ShDF}(p)) - \min_p \log(1 + \text{ShDF}(p))}{\max_p \log(1 + \text{ShDF}(p)) - \min_p \log(1 + \text{ShDF}(p))}. \quad (9)$$

Fit a Gaussian Mixture Model (GMM) to ShDF values. Given a desired number of semantic components, to obtain soft class probabilities, we fit a one-dimensional Gaussian mixture model (GMM) to the per-face thickness scalars obtained from previous step. The probabilities from the GMM provide a data-driven per-face likelihood for each component and serve as the unary data term in the subsequent energy formulation.

Construct a boundary smoothness cost between adjacent faces. To bias the partition toward spatially coherent regions, we compute a smoothness cost on adjacent face pairs based on local geometry (for example, a function of the dihedral angle). Intuitively, nearly-coplanar adjacent faces incur a low boundary penalty, whereas faces separated by sharp edges incur a higher penalty for being assigned the same label.

Initial partition assignment via data likelihood. As an initial labeling, each face is assigned to the GMM component with maximum probability (equivalently, minimum negative log-likelihood of the probability). This produces a purely data-driven segmentation that is generally noisy but captures the major mode structure of the thickness field.

Refine partitioning via energy-based graph cuts. The initial assignment, obtained from the previous step, is then refined using an iterative expansion-based graph-cut procedure that minimizes a global energy

$$E(\text{labeling}) = \sum_{\text{faces}} \mathcal{C}_{\text{data}}(\text{face}, \text{label}) + \sum_{\text{adjacent faces}} \mathcal{C}_{\text{smooth}}(\text{pair}) \cdot \mathbf{1}[\text{labels differ}], \quad (10)$$

where the unary term is derived from the GMM negative log-likelihoods and the pairwise term is the smoothness cost described above. The expansion moves are solved by an optimal min-cut on an auxiliary graph for each label. Repeating this process across labels and iterating a few times yields a stable and low-energy partition.

Re-label connected components. After refinement, a single label index can correspond to multiple disconnected components. To solve this issue, we relabel each connected component of faces so that

every final label corresponds to a single connected sub-mesh. This guarantees that subsequent per-part processing operates on contiguous regions.

Post-processing. To produce clean 3D segmentation results, we apply simple post-processing heuristics: (i) remove tiny components below a face-count threshold, (ii) smooth labels by majority filtering on local neighborhoods, and (iii) (when appropriate) merge adjacent labels that are semantically similar. These operations stabilize the partition and remove spurious tiny regions that would complicate per-part UV parameterization. The output is then a set of connected semantic sub-meshes $\{\mathcal{M}_k = (V_k, F_k)\}_{k=1}^K$ suitable for the per-part UV parameterization stage described in Sec.3.2.

A.2 REMARKS ON BASE ARCHITECTURE FOR UV-PARAMETERIZATION LEARNING

Overview. Our backbone, based on bi-directional cycle mapping proposed in (Zhao et al., 2025), implements a learnable and interpretable pipeline that mimics the artist operations used to create a UV atlas: (i) *deforming network*, starting from a regular 2D grid of uniformly spaced coordinates, this network adaptively deforms the grid to produce a set of candidate UV coordinates that are potentially well-suited for mapping; (ii) *wrapping network*, this module maps 2D UV candidates, generated by the deforming network, to the 3D surface by smoothly bending and folding the deformed planar grid until it conforms to the target geometry (in this case the input 3D mesh) which leads to effectively bridge the 2D and 3D domains; (iii) *cutting network*, operating directly on the 3D surface, generated by the wrapping network, this network transforms the input closed 3D geometry, obtained from the wrapping network, into an open, more developable 3D surface manifold. By strategically determining seam locations, i.e. where to cut the input closed 3D surface, it facilitates distortion reduction in subsequent flattening steps; and (iv) *unwrapping network*, this final network projects the open 3D surface points back to the 2D domain, preserving smoothness and geometric coherence. In this design, each subnetwork is designed as a multilayer perceptron (MLP) applied pointwise to mesh vertices. The entire architecture is trained jointly in an end-to-end unsupervised fashion. The four subnetworks are assembled into two complementary branches that enforce both 2D-3D-2D and 3D-2D-3D cycle mappings, ensuring consistency across 2D and 3D domains. Coupled with differentiable loss functions designed to encourage bijectivity and minimize distortion, this backbone forms the base architecture of our method to achieve a candidate UV parameterization for the next stages. In the following, we detail on the loss designs and the interaction of the four sub-networks within the two-branch bi-directional cycle.

Notation. Let $\mathcal{M} = (V, F)$ be a triangle mesh with vertex positions $P \in \mathbb{R}^{V \times 3}$ and normals $N \in \mathbb{R}^{V \times 3}$. Let $G = \{g_i\}_{i=1}^V \subset \mathbb{R}^2$ denote a canonical planar lattice (the artist’s canvas). The learnable UV mapping is $u_\theta(\cdot)$; the four MLP sub-networks are *DeformNet*, *WrapNet*, *CutNet*, *UnwrapNet*.

Sub-network Definitions. Each module is implemented as a small pointwise MLP. We use additive residuals in the deform and cut modules to bias predictions toward small, interpretable offsets:

$$\hat{Q} = \text{DeformNet}(G) = G + \phi_d''([\phi_d'(G); G]), \quad (11)$$

$$[\hat{P}, \hat{N}] = \text{WrapNet}(\hat{Q}) = \phi_w''([\phi_w'(\hat{Q}); \hat{Q}]), \quad (12)$$

$$\hat{P}_{\text{cut}} = \text{CutNet}(\hat{P}) = \hat{P} + \phi_c''([\phi_c'(\hat{P}); \hat{P}]), \quad (13)$$

$$\hat{Q}_{\text{cycle}} = \text{UnwrapNet}(\hat{P}_{\text{cut}}) = \phi_u(\hat{P}_{\text{cut}}). \quad (14)$$

where $\hat{Q}$ are deformed 2D candidates, $\hat{P}$ are wrapped 3D points (with normals $\hat{N}$), $\hat{P}_{\text{cut}}$ is the opened 3D surface after the learned cut offsets, $\hat{Q}_{\text{cycle}}$ are the re-flattened coordinates produced by the UnwrapNet, $\phi_d'' : \mathbb{R}^{(h+2)} \rightarrow \mathbb{R}^2$, $\phi_d' : \mathbb{R}^2 \rightarrow \mathbb{R}^h$, $\phi_w' : \mathbb{R}^2 \rightarrow \mathbb{R}^h$, $\phi_w'' : \mathbb{R}^{h+2} \rightarrow \mathbb{R}^6$, $\phi_c'' : \mathbb{R}^{h+3} \rightarrow \mathbb{R}^3$, $\phi_c' : \mathbb{R}^3 \rightarrow \mathbb{R}^h$, and $\phi_u : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ are implemented as stacked MLP layers.

Sub-network architectures. Each subnetwork is implemented as a sequence of PWE blocks (point-wise 1x1 convolution with LeakyReLU activation functions), applied point-wisely to per-vertex features. In practice each MLP is split into an encoding stage (MLP_1) and a decoding stage (MLP_2) (e.g., Deform use $[2 \rightarrow 512 \rightarrow 512 \rightarrow 512 \rightarrow 64]$ then $[66 \rightarrow 512 \rightarrow 512 \rightarrow 512 \rightarrow 2]$ as encoder and decoder, respectively). DeformNet produces deformed UV candidates, WrapNet maps those

Table 4: Layer-wise summary of four sub-networks used in our base neural UV architecture (PWE = point-wise embedding conv1d). Each listed MLP is a sequence of PWE blocks; intermediate PWEs use LeakyReLU (with the negative slope of 0.01) while the final PWE is linear.

Subnetwork	Component	Layer channels	# PWE	Input / Output dim	Notes (activation)
DeformNet	MLP.1	[2 → 512 → 512 → 512 → 64]	4	2 / 64	LeakyReLU(0.01) on hidden PWEs; final PWE linear
	MLP.2	[66 → 512 → 512 → 512 → 2]	4	66 / 2	LeakyReLU(0.01) on hidden PWEs; final PWE linear
WrapNet	MLP.1	[2 → 512 → 512 → 512 → 64]	4	2 / 64	same as Deform MLP.1
	MLP.2	[66 → 512 → 512 → 512 → 6]	4	66 / 6	outputs 6D (3 coords + 3 normals)
CutNet	MLP.1	[3 → 512 → 512 → 64]	3	3 / 64	LeakyReLU(0.01) on hidden PWEs; final PWE linear
	MLP.2	[67 → 512 → 512 → 3]	3	67 / 3	produces 3D offsets X_o (then $X_c = X + X_o$)
UnwrapNet	MLP	[3 → 512 → 512 → 2]	3	3 / 2	maps cut 3D points → 2D unwrapped points
	CutNet	see above	—	3 / 3	Unwrapping uses Cutting before this MLP
PWE (block)	conv layer	Conv1d(kernel=1): $C_{in} \rightarrow C_{out}$	1 (per call)	-	point-wise 1D conv + bias; optional LeakyReLU(0.01)

2D candidates to 3D points and normals, CutNet predicts pointwise 3D offsets to open the surface (making it more developable), and UnwrapNet applies CutNet followed by a final MLP to project the opened surface back to 2D. All subnetworks operate pointwise (per-vertex), the final PWE in each MLP is linear, intermediate PWEs use LeakyReLU as activation functions, and the entire architecture is trained end-to-end in an unsupervised manner (see Table 4 for a layer-wise breakdown).

Bi-Directional Cycle Mapping. We follow (Zhang et al., 2024b; Srinivasan et al., 2024; Zhao et al., 2025) to build our base neural UV parameterization architecture with two-branch MLP architecture that leverages the four sub-networks, explained above.

- **2D-3D-2D:** $G \xrightarrow{DeformNet} \hat{Q} \xrightarrow{WrapNet} \hat{P} \xrightarrow{CutNet} \hat{P}_{cut} \xrightarrow{UnwrapNet} \hat{Q}_{cycle}$. Enforcing $\hat{Q}$ to be close to $\hat{Q}_{cycle}$ stabilizes deformations and ensures that the learned 2D deformations are actually realizable on the surface.
- **3D-2D-3D:** $P \xrightarrow{CutNet} P_{cut} \xrightarrow{UnwrapNet} Q \xrightarrow{WrapNet} \tilde{P}$. Enforcing $\tilde{P}$ to be close to P constrains the unwrapping map so that the predicted UV coordinates can be re-embedded to match the true geometry.

Together the two cycles expose inconsistencies including any seams, overlaps, stretches, and/or distortions as reconstruction losses that the network can reduce. We now detail all the reconstruction losses in detail.

Training losses. The backbone is trained with a weighted sum of unsupervised, geometry-aware losses:

$$\mathcal{L} = \lambda_{wrap}\mathcal{L}_{wrap} + \lambda_{cycle}\mathcal{L}_{cycle} + \lambda_{repel}\mathcal{L}_{repel} + \lambda_{dist}\mathcal{L}_{dist} \quad (15)$$

Below are the main terms.

Wrapping loss. Encourages $\hat{P}$ (Wrap-Net output) to approximate the input surface by computing the chamfer distance between the input mesh vertices P and reconstructed $\hat{P}$:

$$\mathcal{L}_{wrap} = \text{ChamferDistance}(\hat{P}, P) + \kappa_{norm} (1 - \text{CosineSim}(\hat{N}, N)) \quad (16)$$

where the second term enforces vertex normal consistency by computing point-wise cosine similarity between normal vectors.

Cycle-consistency loss. Penalizes reconstruction error on both cycles by computing the squared norm 2 of the pair of deformed 2D candidates $\hat{Q}$ and the output of UnwrapNet $\hat{Q}_{cycle}$ and the pair of input mesh vertices P and reconstructed vertices $\tilde{P}$ from the WrapNet:

$$\mathcal{L}_{cycle} = \|\hat{Q} - \hat{Q}_{cycle}\|_2^2 + \|P - \tilde{P}\|_2^2 \quad (17)$$

optionally augmented with normal/feature consistency terms.

Anti-overlap (repulsion). Discourages UV overlaps by repelling nearby vertex pairs in UV space:

$$\mathcal{L}_{\text{repe}} = \frac{1}{|\mathcal{N}|} \sum_{(i,j) \in \mathcal{N}} \phi(\|u_{\theta}(i) - u_{\theta}(j)\|_{\text{uv}}) \quad (18)$$

where $\mathcal{N}$ denotes local neighbor pairs, ϕ is a hinge-style repulsion, and $\|\cdot\|_{\text{uv}}$ includes chart-periodicity handling.

Differential and triangle distortion losses. Control conformal and area distortion using a Jacobian-level differential loss (DDL) and a triangle-based discrete loss (TDL):

$$\mathcal{L}_{\text{dist}} = \lambda_{\text{DDL}} \mathcal{L}_{\text{DDL}} + \lambda_{\text{TDL}} \mathcal{L}_{\text{TDL}} \quad (19)$$

The DDL is computed by applying automatic differentiation to the MLP outputs to approximate the local Jacobian, while the TDL explicitly measures changes in angles and areas between 3D triangles and their corresponding 2D counterparts. In practice, we adopt the same formulations of DDL and TDL as proposed in (Zhao et al., 2025), and refer readers to (Sheffer et al., 2007b; Hormann et al., 2007; Floater & Hormann, 2005) for further details on distortion metrics and differential geometry concepts.

A.3 ADDITIONAL EXPERIMENTS AND RESULTS

We show expanded results, ablation studies, and discussions from the main paper featuring more surface parameterization and more viewpoints of different 3D objects in Figs. 6- 15, and Tables. 5- 9.

Ablation studies on different UV packing (aggregator) methods. To further improve our semantic-aware UV parameterization pipeline’s output, a simple enhancement is to post-process the UV charts by jointly optimizing two key UV metrics: texel density and UV space utilization. Higher texel density improves the quality of the projected 3D texture, and maintaining consistent texel density across charts ensures uniform visual fidelity. At the same time, efficient space utilization maximizes the use of available texture memory. Hence, to analyze the effect of more advanced UV-chart packing (aggregator) strategies (different from the one proposed in Sec. 3.3), we perform an ablation study and evaluate several UV-packing techniques from three widely used commercial tools including, Autodesk Maya (Autodesk, Inc., 2025), Houdini (Toronto-based SideFX, 2025), and Blender (Blender Foundation, 2025). We use a 3D cow mesh shown in Fig. 6 to evaluate texel density and UV-space utilization of UV parameterizations packed using different UV packing strategies from the three commercial tools. Specifically, we apply our semantic-aware UV parameterization framework as follows: (i) the 3D cow is first partitioned by PartField (Liu et al., 2025) into seven semantic 3D parts, (ii) then the geometry-preserving neural backbone is applied separately to each part to learn UV parameterizations for each 3D part, (iii) instead of using our default UV packing (presented in Sec. 3.3), we apply different UV packing strategies from the three commercial tools, and (iv) lastly, for each packing strategy, we compute two UV metrics, namely texel density and UV-space utilization, for the UV parameterizations resulting from the three different UV packing methods. Table 5 reports the resulting values for both metrics to illustrate how each tool performs on a 3D cow mesh example. Also, in Fig. 6, we qualitatively compare the three commercial tools, applying their automatic packing and layout features to the same unwrapped mesh of 3D cow.

Ablation studies on different partitioning strategies (ShDF vs. SAMesh vs. PartField). We replace our default Shape Diameter Function (ShDF) partitioner (Shapira et al., 2008) with SAMesh (Tang et al., 2024) and PartField (Liu et al., 2025) while keeping the remainder of the partition-and-parameterize (semantic-aware UV parameterization) pipeline identical. Then, we train each pipeline separately on input 3D meshes and measure and report angle-preservation (conformality) and area-preservation (equiareality) properties in the generated UV parameterization. Figs. 13- 14 and Table 6 shows the qualitative and quantitative results of these experiments, respectively. To interpret the numbers in the table, we point out two points: (i) There is no obvious or general relationship between the choice of segmentation algorithm and the geometry-preservation metrics (conformality and equiareality) of the final UV parameterization. This is due to the fact that existing 3D segmentation methods were developed to produce semantically meaningful parts, not to optimize downstream UV parameterization. Consequently, they do not incorporate UV-specific information. Therefore,

Table 5: Quantitative ablations of different UV-packing strategies from three widely used commercial tools: Autodesk Maya, Blender, and Houdini. We use a 3D cow mesh shown in Fig. 6 to evaluate texel density and UV-space utilization of UV parameterizations packed using different UV packing strategies from the three commercial tools. Specifically, after applying our semantic-aware UV parameterization framework, we obtain UV parameterizations for each segmented 3D part, resulting in seven UV charts. Then, instead of using our default UV packing (presented in Sec. 3.3), we apply different UV packing strategies from the three commercial tools. Lastly, for each packing strategy, we compute two UV metrics, namely texel density and UV-space utilization, for the UV parameterizations resulting from the three different UV packing methods. Higher texel density improves the quality of the projected 3D texture, and maintaining consistent texel density across charts ensures uniform visual fidelity. At the same time, efficient space utilization maximizes the use of available texture memory.

Method	Texel Density							UV Space Utilization	
	Chart1	Chart2	Chart3	Chart4	Chart5	Chart6	Chart7	Avg	
Maya (Autodesk, Inc., 2025)	160.62	150.41	150.41	150.41	156.51	150.41	150.41	152.74	71%
Blender (Blender Foundation, 2025)	151.66	142.02	142.02	142.02	147.78	142.02	142.02	144.22	57%
Houdini (Toronto-based SideFX, 2025)	157.08	147.09	147.09	147.09	153.06	147.09	147.10	149.37	62%

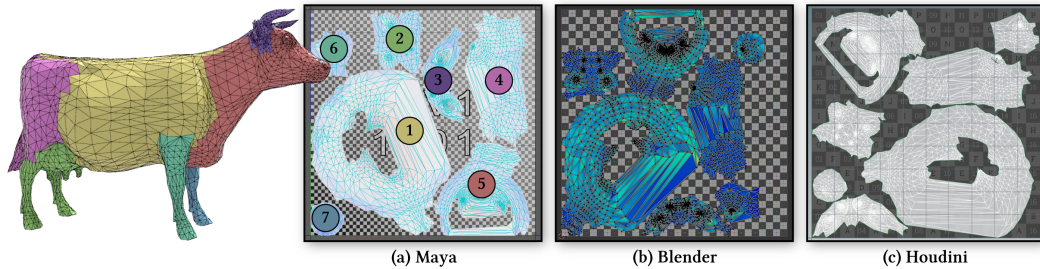


Figure 6: Qualitative ablations of different UV-packing strategies. To further improve our semantic-aware UV parameterization pipeline’s output, we present alternative more advanced UV packing results (different from the one proposed in Sec. 3.3) obtained from three widely used commercial tools: (a) Autodesk Maya, (b) Blender, and (c) Houdini. The left-most column shows the 3D cow mesh segmented by PartField (Liu et al., 2025). In the Autodesk Maya result (a), each UV chart corresponding to a semantic 3D part is colored to match its counterpart on the 3D mesh. The same applies to the Blender and Houdini results. Specifically, the 3D cow is first partitioned by PartField (Liu et al., 2025) into seven semantic 3D parts. Then, the geometry-preserving neural backbone (Sec. 3.2) is applied separately to each part to learn UV parameterizations for each 3D part leading to seven UV charts. Then, instead of using our default UV packing (presented in Sec. 3.3), we apply automatic packing strategies layout features of the three commercial tools to the unwrapped mesh of 3D cow.

from the numbers in Table 6, we cannot draw a unified, causal conclusion connecting segmentation method type to final UV geometry properties. Having that said, we include this segmentation ablation and Table 6 intentionally to inform practitioners that different partitioning methods, while varying in semantic quality, can also influence geometry-preservation in the resulting UVs. Reporting these results shows the important trade-off between semantic-awareness and geometric fidelity. Therefore, practitioners should try different segmentation approaches and choose the one that best fits their application needs. (ii) Of course, state-of-the-art, learning-based segmentation methods can produce more semantically coherent partitions than ShDF (Shapira et al., 2008) (as shown by works such as SAMesh (Tang et al., 2024)). Therefore, any improvements in that literature can directly improve the semantic-awareness of our pipeline because segmentation is the first stage of our semantic-aware parameterization (Fig.2). In other words, a better 3D partitioner is an immediate plug-in improvement for our method.

Ablation studies on geometry-preservation vs. semantic / visibility objectives. Optimizing multiple geometry-preservation metrics simultaneously (conformality, equiareality, stretch, bijectivity, etc.) is known to require trade-offs. Specifically, achieving the best possible value for every metric at once is generally impractical. Introducing higher-level objectives (semantic- and visibility-awareness), that we propose in this work, increases this tension and therefore can further degrade

some geometry metrics. To quantify these effects, we include an ablation in Table. 7 that compares our proposed pipelines to the baseline (labelled “Base Neural UV Arch” in Table. 7, i.e., FlexPara) which is state-of-the-art for geometry preservation. We compare three variants of our pipeline to the baseline: (i) Semantic-Aware Param, the pipeline of Sec. 3.3 (Fig. 2), which first partitions the mesh into semantic parts and then learns per-part parameterizations; (ii) Visibility-Aware Param, the pipeline of Sec. 3.4 (Algorithm. 1), which produces a global atlas while steering seams toward less-visible regions; and (iii) Visibility+Semantic-Aware Param, the combined pipeline that applies both sets of objectives. Three observations explain the quantitative trends: (i) when comparing conformality and equiareality, the Semantic-Aware and the combined Visibility+Semantic-Aware pipelines lose little equiareality relative to the baseline and even show slight improvement in conformality in some cases. Intuitively, this is because the semantic-aware stage first partitions the mesh into multiple semantically meaningful subparts. Intuitively, learning a geometry-preserving parameterization on smaller, simpler patches is easier. In other words, the network can better satisfy geometry-preservation while also enforcing semantic consistency. Therefore, semantic partitioning provides a natural way to reduce distortion without heavily sacrificing geometry metrics. (ii) By contrast, the visibility-aware pipeline does not partition the mesh. Instead, it steers seams to less-visible (more occluded) regions while producing a global atlas (a single large chart). Unwrapping a whole 3D mesh into a single UV chart to while simultaneously optimizing visibility and geometry is significantly harder. Consequently, the Visibility-Aware pipeline shows a larger drop in equiareality compared to the baseline and compared to the semantic-aware pipelines. (iii) The combined Visibility+Semantic-Aware pipeline aims to obtain the best of both worlds: it first partitions the mesh into smaller parts to retain geometry preservation, and then steers seams toward less-visible regions within those parts. This combined approach yields a more favorable trade-off: it loses far less equiareality (0.6369) than the pure visibility-aware pipeline (0.6093) while still improving seam visibility and semantic consistency.

Analysis on training and inference computation times. To provide users and practitioners with some insights about the computation time needed for the proposed method in this study, we report both inference and training time of the proposed method in Table 3, Table 1, Table 8, and Table 9, and compare them with the computation time of the state-of-the-art method (at the time of writing this manuscript), FlexPara (Zhao et al., 2025).

- **Inference time.** (i) Visibility-aware pipeline: although training is performed per-mesh (as is common in this literature: Nuvo (Srinivasan et al., 2024), FlexPara (Zhao et al., 2025), OptCuts (Li et al., 2018)), the visibility objective is applied only during training (not the inference stage). Specifically, at inference the trained model uses the same neural architecture as FlexPara (Zhao et al., 2025) and therefore incurs no additional runtime overhead. This is why the inference times for our Visibility-Aware Param and FlexPara are the same in Table 1 (≈ 2 seconds). (ii) Semantic-aware pipeline: the semantic-aware pipeline does introduce extra inference cost because it performs three stages at test time: (a) 3D semantic partitioning, (b) per-part parameterization (running the backbone once per part), and (c) aggregation/UV packing. These additional steps (segmentation, multiple backbone invocations, and packing) explain the larger inference time reported in Table 2 and for the combined pipeline (semantic+visibility-aware) in Table 1. A simple and effective mitigation is to parallelize the per-part backbone calls by using multi-threading during the inference, which would significantly reduce inference time.
- **Training time.** (i) Semantic-aware pipeline (Table 8): counterintuitively, our semantic-aware pipeline achieves shorter training time than FlexPara (Zhao et al., 2025) across different 3D meshes. The reason is that the mesh is partitioned into many smaller submeshes: although our semantic-aware pipeline calls the backbone multiple times (once per semantic part), which might suggest longer training, this is compensated by the fact that the 3D segmented parts are much smaller than the original input mesh used by FlexPara. Hence, training on many small submeshes is faster than training a single model on a large, complex mesh with many vertices and faces. This empirical observation is consistent with our careful time measurement of each stage in the semantic-aware pipeline which is: learning UV parameterization for small 3D parts is much faster than doing the same for large meshes. More importantly, the reduction in per-part training cost outweighs the multiplicative overhead of multiple backbone runs, which results in shorter overall training time for the semantic-aware pipeline in our experiments. This is why Table 8 shows shorter train-

ing times for our semantic-aware pipeline compared to FlexPara in our experiments. (ii) Visibility-aware pipeline (Table 9): The visibility-aware pipeline trains slower than FlexPara (Zhao et al., 2025) across different meshes. This is expected because, compared to the base neural backbone (i.e., FlexPara), our visibility-aware pipeline introduces per-iteration overheads: (a) per-vertex ambient occlusion (AO) computation, (b) candidate UV generation via the backbone, (c) seam/boundary extraction in UV space, and (d) AO-weighted loss computation. The most expensive stage is seam extraction which accounts for approximately 90% of the additional training time. Specifically, for each vertex we inspect its 3–5 neighbors, compare mapped UV positions, and evaluate Eq. 5 to detect UV discontinuities. This neighbor-checking and distance computation scales with vertex count and dominates the additional training cost (90% of the overhead), making visibility-aware training significantly heavier for complex meshes.

- **Potential attempts to overcome computation burdens.** To mitigate the computational burdens discussed above, we propose two practical solutions: (i) parallelizing per-part backbone calls in semantic-aware pipeline. In our current implementation, the neural backbone is executed sequentially for each semantic part. Switching to a multi-threaded or multi-GPU implementation to run per-part backbone calls in parallel would significantly reduce both inference and training times. (ii) Optimizing seam detection in visibility-aware pipeline. The seam-detection stage is the main training-time bottleneck. Algorithmic and engineering improvements, e.g., spatial hashing, cached neighborhood lookups, and GPU-accelerated neighbor/distance checks, can significantly reduce this overhead. We note these optimizations as straightforward improvements and sensible directions for future work.

Table 6: Ablation studies on the proposed partitioning strategy (ShDF vs. SAMesh vs. PartField). We replace our default Shape Diameter Function (ShDF) partitioner (Shapira et al., 2008) with SAMesh (Tang et al., 2024) and PartField (Liu et al., 2025) while keeping the remainder of the *partition-and-parameterize* pipeline identical. Quantitatively, using ShDF in our pipeline yields UV parameterizations that are more conformal (better angle preservation) and more equiareal (better area preservation) than those produced when using SAMesh.

Method	Conformality $\uparrow$	Equiareality $\uparrow$
Ours - Semantic-Aware Param (PartField)	0.8999	0.6921
Ours - Semantic-Aware Param (SAMesh)	0.8694	0.6270
Ours - Semantic-Aware Param (ShDF)	0.9123	0.6707

Table 7: Ablation: the effect of semantic and visibility objectives on geometry-preserving metrics. Reported are mean Conformality and Equiareality (higher is better). The Semantic-Aware and the combined Visibility+Semantic-Aware pipelines lose little equiareality relative to the baseline and even show slight improvement in conformality in some cases. Intuitively, this is because the semantic-aware stage first partitions the mesh into multiple semantically meaningful subparts and so the network can better satisfy geometry-preservation while also enforcing semantic consistency. However, the visibility-aware pipeline does not partition the mesh. Instead, it steers seams to less-visible (more occluded) regions while producing one global atlas (a single large chart). Unwrapping and optimizing a single global chart to satisfy both geometry-preservation and visibility objectives is significantly harder. Consequently, the Visibility-Aware pipeline shows a larger drop in equiareality compared to the baseline. The combined Visibility+Semantic-Aware pipeline aims to obtain the best of both worlds: (i) partition the mesh into smaller parts to retain geometry preservation, and (ii) steer seams toward less-visible regions within those parts. This combined approach yields a favorable trade-off: it loses far less equiareality (0.6369) than the pure visibility-aware pipeline (0.6093) while still improving seam visibility and semantic consistency.

Method	Conformality $\uparrow$	Equiareality $\uparrow$
Base Neural UV Arch.	0.9097	0.6759
Ours - Semantic-Aware Param	0.9123	0.6707
Ours - Visibility-Aware Param	0.9175	0.6093
Ours - Visibility+Semantic-Aware Param	0.9153	0.6369

Table 8: Comparison of training time between our proposed semantic-aware parameterization pipeline and FlexPara’s multi-chart parameterization (Zhao et al., 2025). Total times are shown outside the parentheses, and per-iteration times are shown inside.

Method	Training Time (Seconds)			
	Rabbit	Potion	Squidward House	Average
FlexPara (multi-Chart) (Zhao et al., 2025)	900 (0.0900)	16377 (1.6377)	13784 (1.3784)	10353 (1.0353)
Ours - Semantic-Aware Param	676 (0.1352)	666 (0.1332)	602 (0.1205)	648 (0.1296)

Table 9: Comparison of training time between our proposed visibility-aware parameterization pipeline and FlexPara’s global (single-chart) parameterization (Zhao et al., 2025). Total times are shown outside the parentheses, and per-iteration times are shown inside.

Method	Training Time (Seconds)					
	Duckie	Fish	FlipFlop	Rabbit	Spot	Average
FlexPara (single-Chart) (Zhao et al., 2025)	1306 (0.1306)	1323 (0.1323)	1275 (0.1275)	1293 (0.1293)	1230 (0.1230)	1285 (0.1285)
Ours - Visibility-Aware Param	36642 (3.6642)	47460 (4.7360)	16864 (3.3729)	35232 (2.3488)	19605 (1.9605)	31160 (3.1160)

Table 10: To evaluate semantic- and visibility-awareness of the proposed method, we conducted a user-study with 70 general participants (including graduate students with computer science and engineering backgrounds) performing 11 comparisons between textured 3D shapes and UV parameterizations produced by our method and baselines. We report the percentage of general participant preferences for each method. Our proposed method is strongly preferred by the general users over the baselines.

Method	General User Preference Percentage
Visibility-Awareness Evaluation	
OptCuts (Li et al., 2018)	1.43 %
FlexPara (single-Chart) (Zhao et al., 2025)	7.14 %
Ours - Visibility-Aware Param	91.42 %
Semantic-Awareness Evaluation	
xatlas (jpcy, 2025)	4.28 %
Blender (Blender Foundation, 2025)	4.28 %
Autodesk Maya (Autodesk, Inc., 2025)	10 %
FlexPara (multi-Chart) (Zhao et al., 2025)	7.14 %
Ours - Semantic-Aware Param	74.29 %

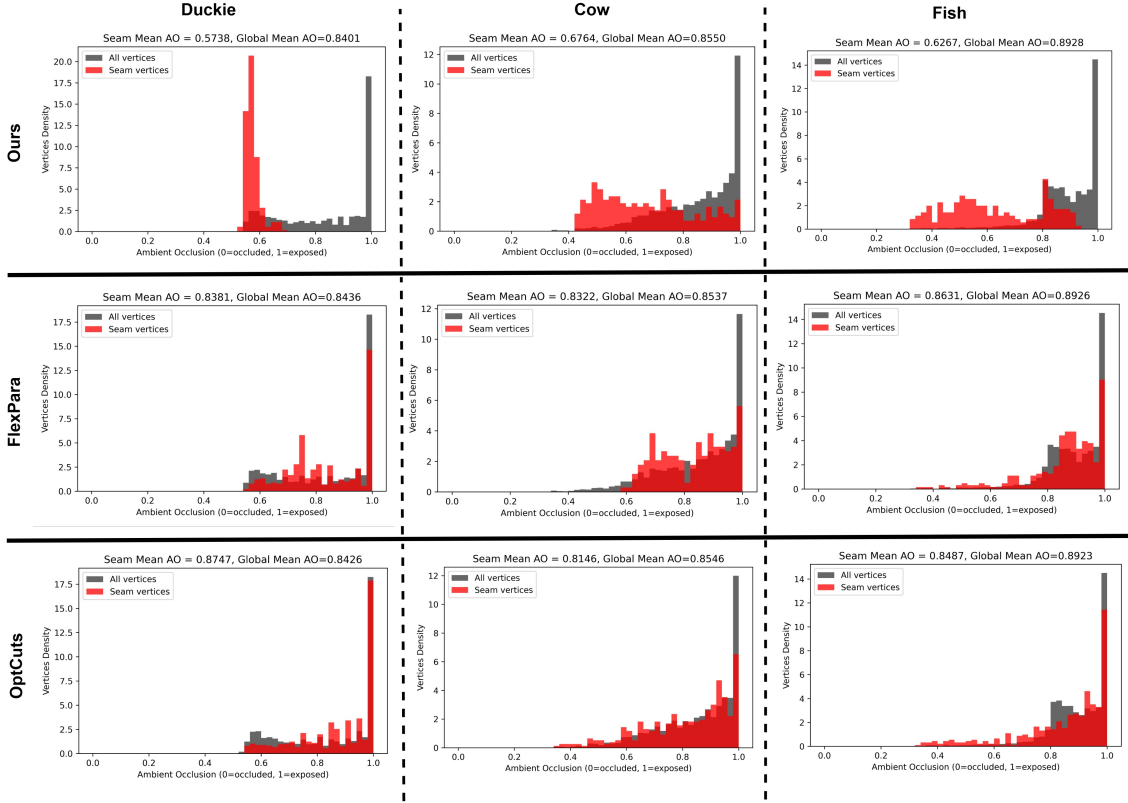


Figure 7: Quantitative distribution of seam-vertex ambient occlusion (AO) for three meshes parameterized with our method, FlexPara (Zhao et al., 2025), and OptCuts (Li et al., 2018). Red bars indicate AO distributions of seam vertices, while black bars represent all vertices. The y -axis denotes vertex density and the x -axis AO values. Our method shifts seam-vertex AO distributions toward lower values (left side), showing that the visibility-aware pipeline relocates seams to less visible (more occluded) regions, unlike FlexPara and OptCuts.

B THE USE OF LARGE LANGUAGE MODELS

We used a large language model (LLM) to assist with editing, refining, and polishing the manuscript text (for example: abstract and captions). The LLM was used only for language refinement, improving grammar, wording, concision, and overall readability, and did not contribute to the technical content, experiments, results, or scientific claims. All suggestions generated by LLM were manually and carefully reviewed and, where appropriate, revised by the authors. The authors have full responsibility for the final text and for the accuracy of the work.

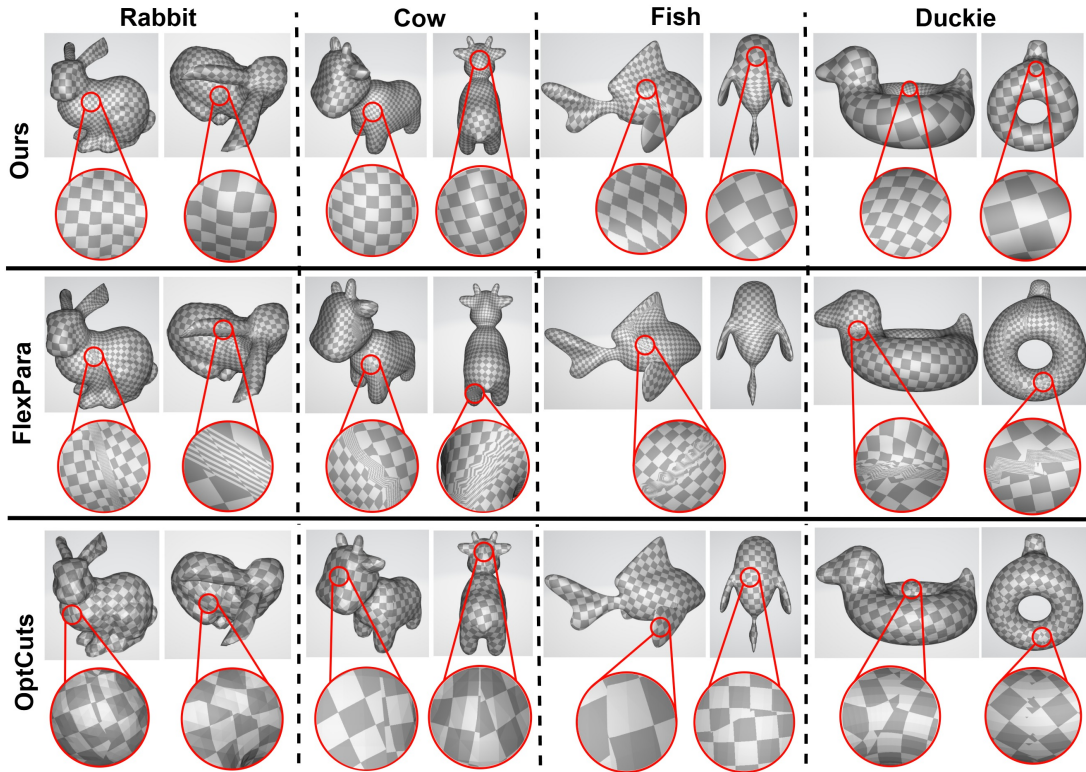


Figure 8: Extended version of Fig. 5 in the main paper. Checkerboard texturing comparison using UV parameterizations produced by our visibility-aware method, FlexPara (Zhao et al., 2025), and OptCuts (Li et al., 2018). Each row shows rendered views of different meshes textured with a checkerboard and a magnified inset of a visually important region near seams (red circles). Because our method steers seams toward occluded regions, the checkerboard pattern appears substantially more continuous from typical camera viewpoints. By contrast, baselines exhibit visible seam artifacts in the zoomed-in insets.

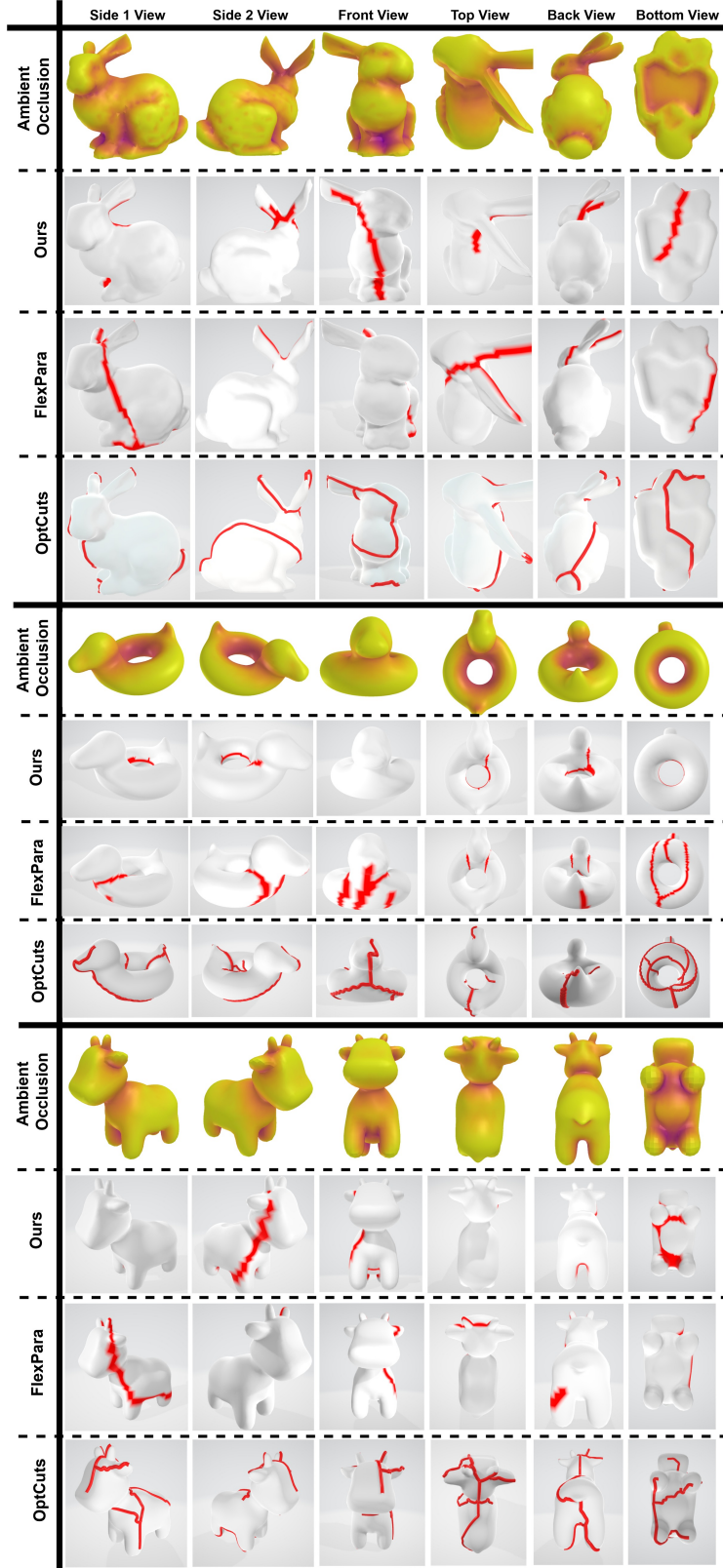


Figure 9: Extended version of Fig. 4 in the main paper. Qualitative results for visibility-aware seam placement and UV parameterization on three representative meshes. For each mesh, the top row shows per-vertex ambient occlusion (yellow = exposed, purple = occluded). Beneath are the visualizations of cutting seams (red) from our method, FlexPara (Zhao et al., 2025), and OptCuts (Li et al., 2018) (top to bottom). Our method places a larger fraction of seam geometry in less-exposed regions, reducing the likelihood of visible seam artifacts under typical viewpoints.

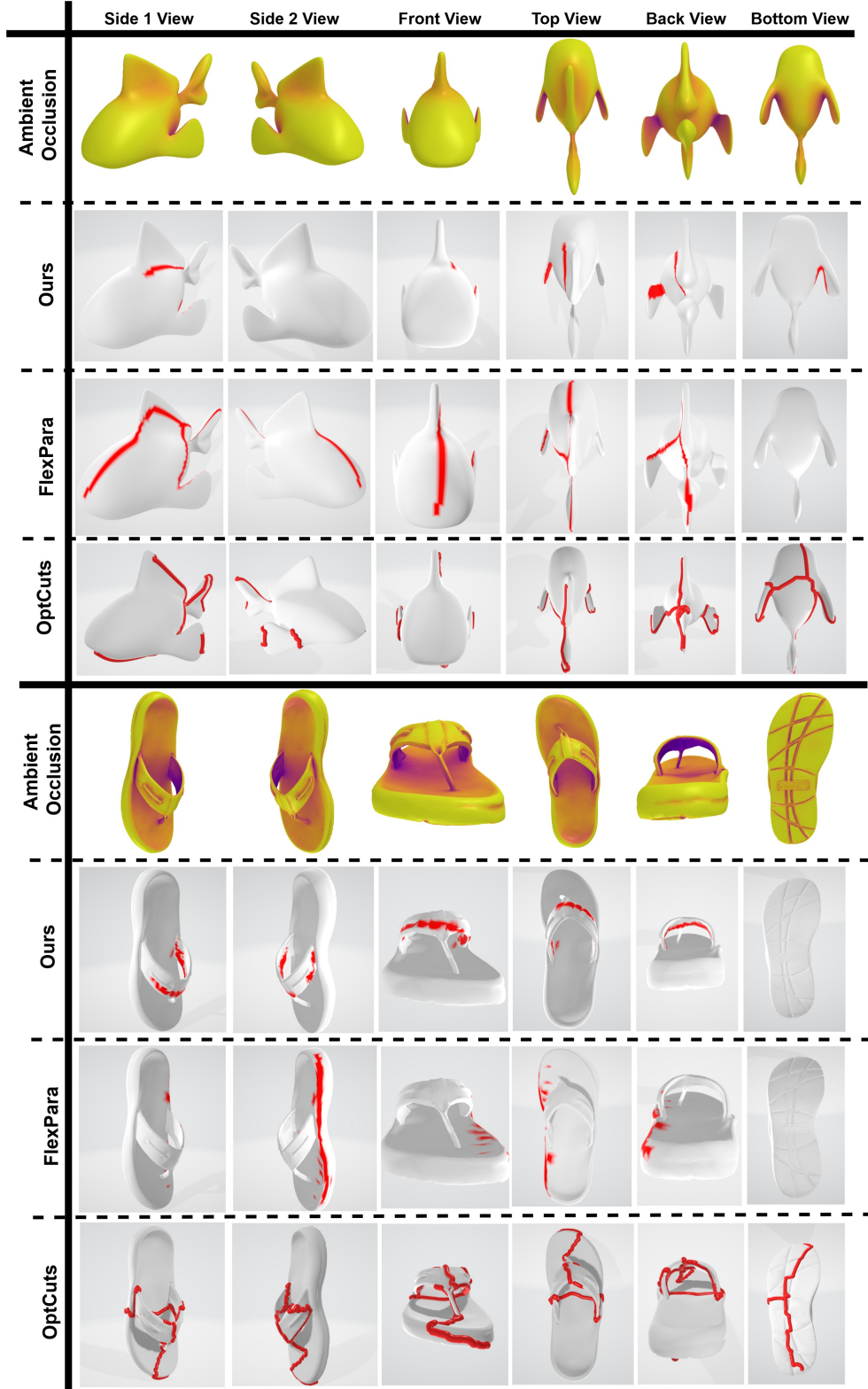


Figure 10: Extended version of Fig. 4 in the main paper. Qualitative results for visibility-aware seam placement and UV parameterization on three representative meshes. For each mesh, the top row shows per-vertex ambient occlusion (yellow = exposed, purple = occluded). Beneath are the visualizations of cutting seams (red) from our method, FlexPara (Zhao et al., 2025), and OptCuts (Li et al., 2018) (top to bottom). Our method places a larger fraction of seam geometry in less-exposed regions, reducing the likelihood of visible seam artifacts under typical viewpoints.

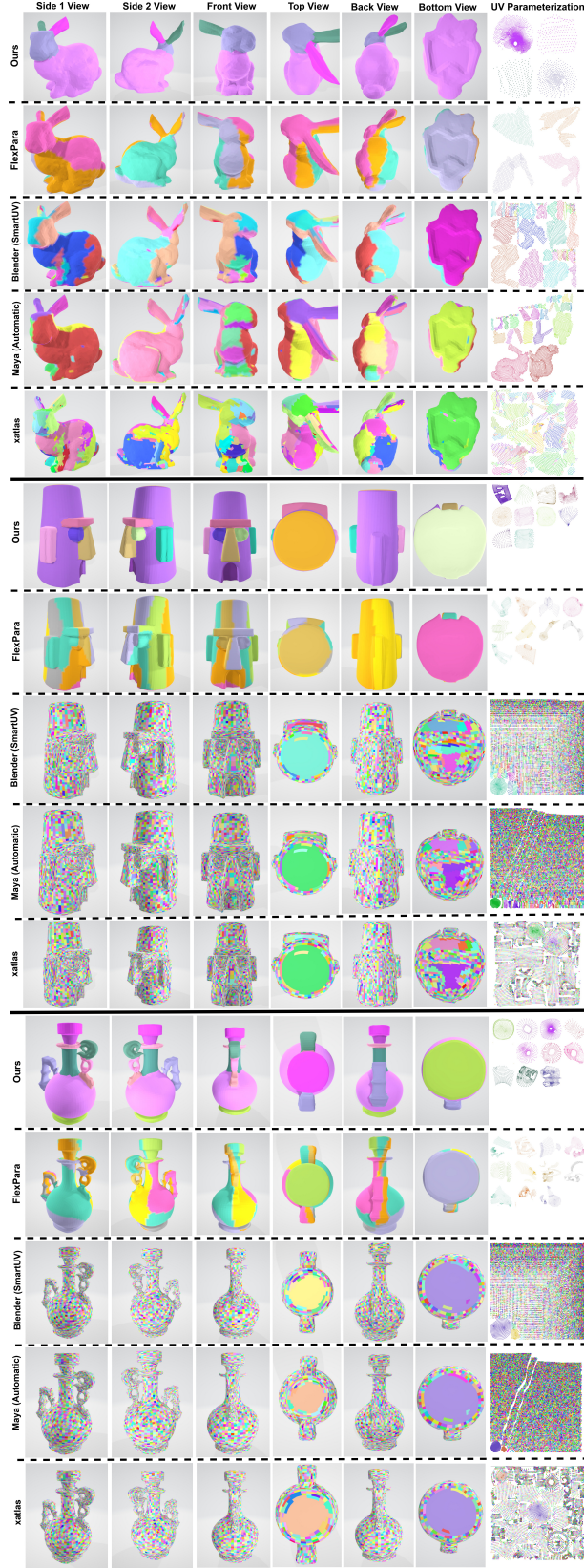


Figure 11: Extended version of Fig. 3 in the main paper. Qualitative results of the proposed semantic-aware UV parameterization method on a Rabbit mesh. For each rendered 3D objects from multiple viewpoints, with the corresponding UV atlas in the rightmost column. As shown, our method produces UV charts that align more semantically with the mesh’s 3D semantic parts, unlike the baselines.

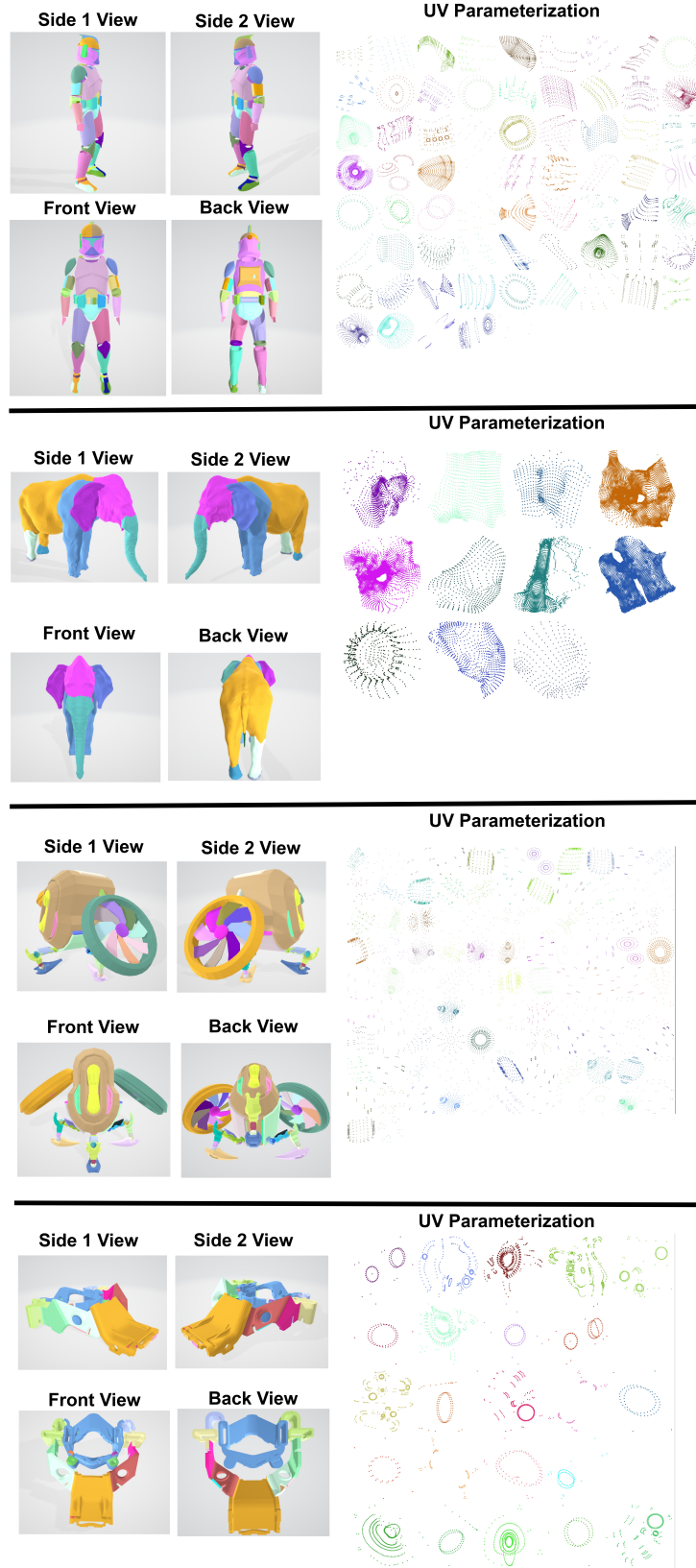


Figure 12: Qualitative results of the proposed semantic-aware UV parameterization method on complex mesh objects with high number of triangle faces. For each method, we show the rendered 3D objects from multiple viewpoints, with the corresponding UV atlas in the rightmost column. As shown, our method is able to produce UV charts, for complex meshes, that are align more semantically with the mesh’s 3D semantic parts

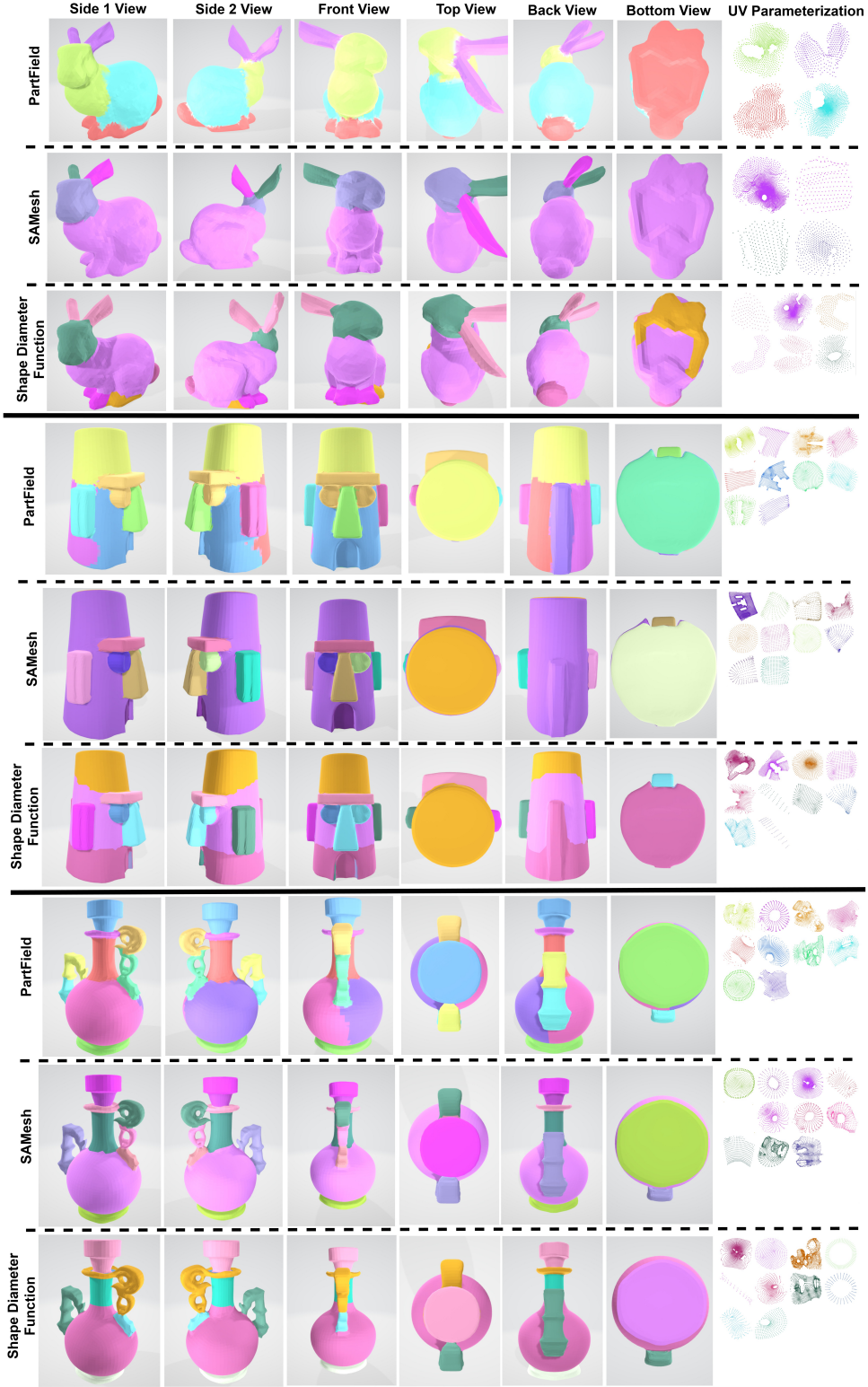


Figure 13: Ablation studies on the proposed partitioning strategy (ShDF vs. SAMesh vs. PartField) of simple meshes. We replace our default Shape Diameter Function (ShDF) partitioner (Shapira et al., 2008) with SAMesh (Tang et al., 2024) and PartField (Liu et al., 2025) while keeping the remainder of the *partition-and-parameterize* pipeline identical. Qualitatively all three partitioners can produce part decompositions that align with geometric structure across simple meshes, resulting compact, low-distortion per-part UV charts. However, it seems that PartField produces more accurate and more detailed semantic segmentations than ShDF and SAMesh in some cases (e.g., the feet and the tail are correctly detected, whereas they are not detected properly in SAMesh).

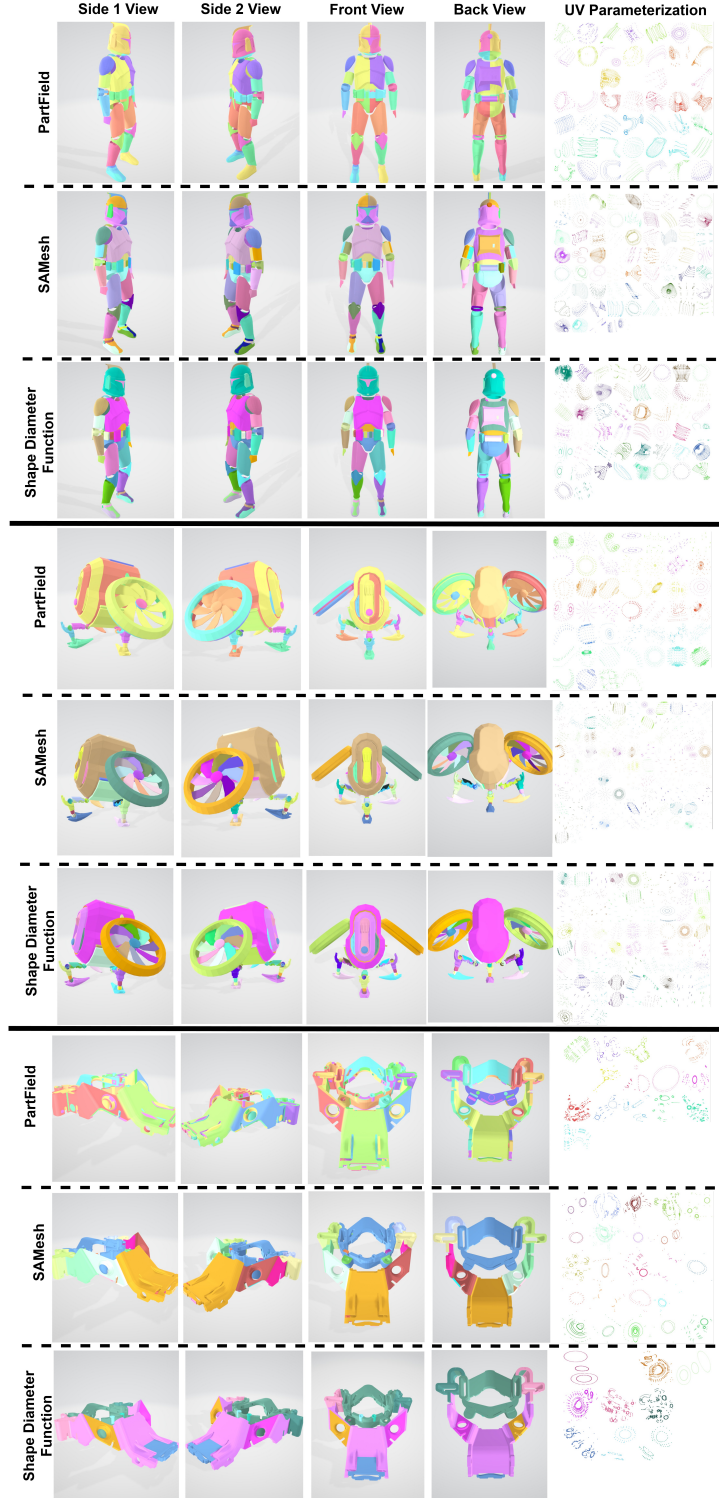


Figure 14: Ablation studies on the proposed partitioning strategy (ShDF vs. SAMesh vs. PartField) of complex meshes with high number of triangle faces. We replace our default Shape Diameter Function (ShDF) partitioner (Shapira et al., 2008) with SAMesh (Tang et al., 2024) and PartField (Liu et al., 2025) while keeping the remainder of the *partition-and-parameterize* pipeline identical. Qualitatively all partitioners produce part decompositions that align with geometric structure across complex models. However, PartField appears to produce more accurate and more compact semantic segmentations than ShDF and SAMesh in some cases. For example, PartField correctly groups the entire set of fan blades into a single segmented part, whereas SAMesh and ShDF split each blade into a separate part. This is not completely wrong, but it results in too many small charts after per-part UV parameterization, which may be undesirable for some downstream tasks.

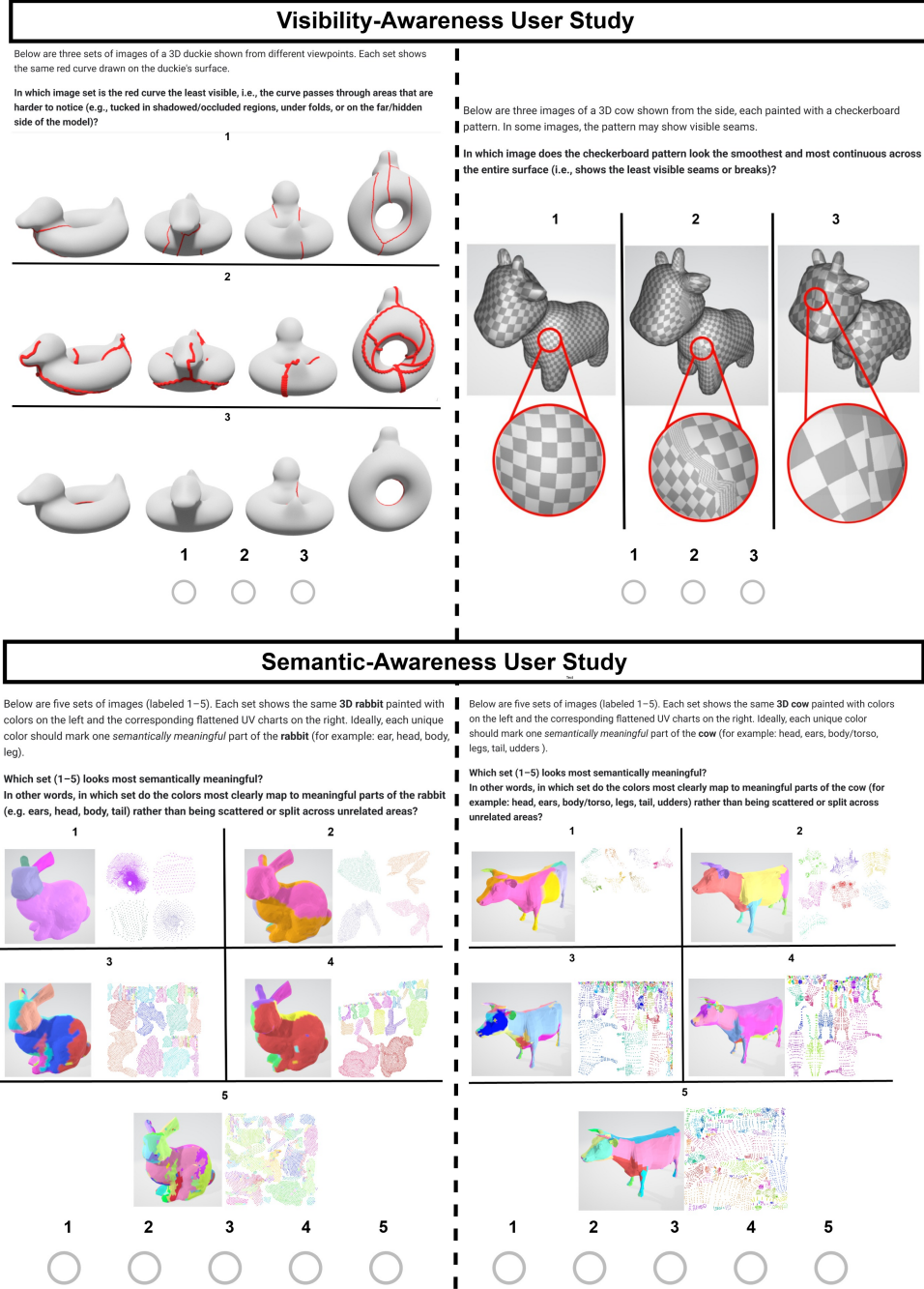


Figure 15: Examples of four of the eleven questions in our user-study questionnaire. To evaluate semantic- and visibility-awareness of the proposed method, we conducted a user-preference study with 115 participants. We distributed the questionnaire to two groups: experts and general participants. Of the 115 participants, 70 are general participants (including graduate students with computer science and engineering backgrounds) and 45 are experts: 31 software engineers, 3 project managers, 2 product owners, 5 UV/layout artists, and 4 modeling artists, all working in the Media and Entertainment industry for film and games. Each participant completed 11 comparisons between textured 3D shapes and UV parameterizations produced by our method, FlexPara (Zhao et al., 2025), OptCuts (Li et al., 2018), Autodesk Maya (Autodesk, Inc., 2025), Blender (Blender Foundation, 2025), and xatlas (jpcy, 2025). For each comparison, participants were rated each result according to three visual criteria: (i) texture-pattern smoothness/continuity, (ii) semantic alignment (how well colors correspond to meaningful parts), and (iii) seam visibility (how well seams are placed in occluded/less-obvious regions). The semantic-aware and visibility-aware evaluations comprised 5 and 6 questions, respectively, for a total of 11 comparisons per participant. Tables 2 and 10 report the percentage of expert and general participant preferences for each method, respectively. As the tables show, our proposed method is strongly preferred over the baselines by both expert industry-level users and general users.