Geometry-Aware Preference Learning for 3D Texture Generation

AmirHossein Zamani [1 2] **Tianhao Xie** [2] **Amir G. Aghdam** [2] **Tiberiu Popa** [2] **Eugene Belilovsky** [1 2]

Abstract

Recent advances in 3D generative models have achieved impressive results but 3D contents generated by these models may not align with subjective human preferences or task-specific criteria. Moreover, a core challenge in the 3D texture generation domain remains: most existing approaches rely on repeated calls to 2D text-to-image generative models, which lack an inherent understanding of the 3D structure of the input 3D mesh object. To address this, we propose an end-to-end differentiable preference learning framework that back-propagates human preferences, represented by differentiable reward functions, through the entire 3D generative pipeline, making the process inherently geometry-aware. We demonstrate the effectiveness of our framework using four proposed novel geometry-aware reward functions, offering a more controllable and interpretable pathway for high-quality 3D content creation from natural language.

1. Introduction

While large-scale generative computer vision models learn broad knowledge and some reasoning skills to generate images (Black et al., 2024; Fan et al., 2023), videos (Wu et al., 2023), and 3D objects (Poole et al., 2022; Lin et al., 2023), achieving precise control of their behavior is difficult due to the completely unsupervised nature of their training. Among these generative models, 3D generative ones often rely on adapting text-to-image models at inference (Poole et al., 2022; Lin et al., 2023; Zeng et al., 2024), but the resulting 3D content may not align with human preferences or task-specific needs. This challenge highlights the need for a preference learning frameworks adapted to 3D content creation. One general solution proposed mainly in the large language models' literature (Ouyang et al., 2022; Christiano et al., 2017; Dong et al., 2023; Rafailov et al., 2023) is to incorporate human or automated feedback as a supervisory signal to guide the generative models toward desired behavior. This is usually done by leveraging reinforcement learning from human feedback (RLHF) (Christiano et al., 2017). Prior work has shown that RLHF can significantly improve results in some domains, such as text generation (Ouyang et al., 2022; Dong et al., 2023) and image synthesis (Black et al., 2024; Fan et al., 2023). However, to the best of our knowledge, DreamReward (Ye et al., 2024) is the only prior work that incorporates human feedback into 3D generative model training. It applies differentiable rewards to optimize NeRF-based geometry editing, without training the generative model. In contrast, we directly fine-tune the diffusion model for texture editing within a 3D generation pipeline, enabling precise control aligned with user preferences. Moreover, unlike DreamReward's aesthetic rewards that ignore geometry, our geometry-aware rewards ensure textures are both perceptually rich and structurally aligned with the geometry of the input 3D mesh. Despite recent advances, a core challenge in 3D texture generation remains: most existing approaches rely on repeated calls to 2D text-to-image models, which lack an inherent understanding of 3D structure. Therefore, applying preference learning directly to these 2D components often leads to solutions that neglect crucial 3D constraints. To address this, we propose an end-to-end differentiable preference learning framework that back-propagates through the entire 3D generative pipeline, making the process inherently geometry-aware. Our method is reinforcement learning–free, easy to implement, and computationally efficient. By directly fine-tuning the 3D texture generation model through differentiable geometry-aware rewards, our framework supports interactive and user-aligned texture editing, producing results that are both visually compelling and geometrically coherent. By proposing such an approach, we offer a main advantage over existing methods in other generative domains: it is computationally more efficient as it does not necessitate too many expensive sampling steps from the models, which is typically required in reinforcement learning pipelines. The main contributions of this study are: (i) design and development of the first preference learning framework for high-quality 3D geometry-aware content (texture) creation from text, and (ii) proposing novel geometry-aware reward functions for aligning 3D generation models.

2. Methodology

We propose a simple, reinforcement-learning-free, end-to-end differentiable preference learning framework for generating texture images aligned with the geometric features of

[1]Mila - Quebec AI Institute, Montreal, Canada [2]Concordia University, Montreal, Canada. Correspondence to: AmirHossein Zamani <amirhossein.zamani@mila.quebec>.

ICML 2025 Workshop on Models of Human Feedback for AI Alignment, Vancouver, Canada. 2025.

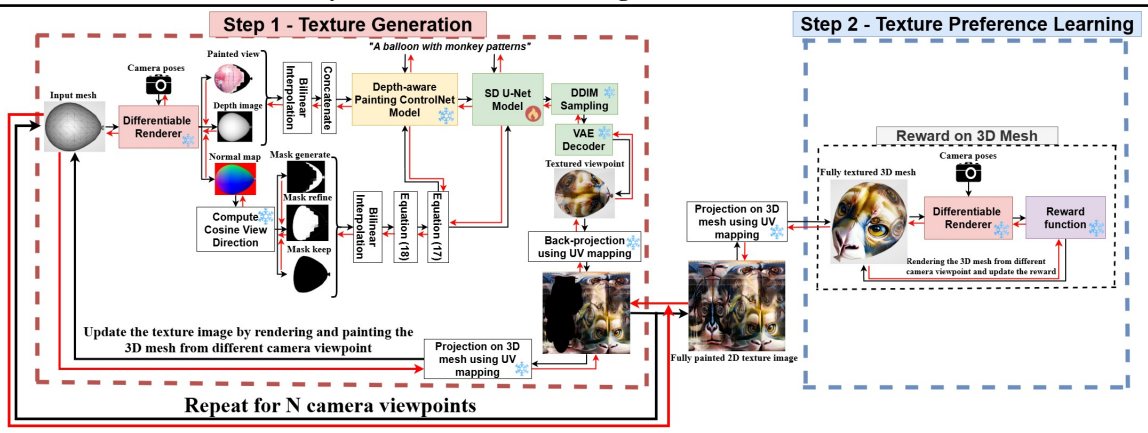

Figure 1. An overview of the proposed training process, consisting of two main stages: (i) *texture generation* (Section 2.1), where a latent diffusion model generates high-quality images from textual prompts. Combined with differentiable rendering and 3D vision techniques, this step produces realistic textures for 3D objects. (ii) *texture preference learning* (Section 2.2), where an end-to-end differentiable pipeline fine-tunes the pre-trained diffusion model ϵ_θ by maximizing a differentiable reward function r. To demonstrate the method's effectiveness in producing textures aligned with 3D geometry, we introduce four novel geometry-aware reward functions, detailed in Section 2.3 and Appendix B.

a 3D mesh. The training consists of two stages: (i) *texture generation*, where a diffusion model (Rombach et al., 2022) combined with 3D vision techniques like differentiable rendering (Laine et al., 2020) produces realistic textures from text prompts (Section 2.1); and (ii) *texture preference learning*, where the diffusion model ϵ_θ is fine-tuned via a differentiable pipeline that maximizes a reward function r, enabling gradients to flow through the full 3D generation process and making it geometry-aware (Section 2.2). We introduce four novel geometry-aware reward functions to demonstrate the method's effectiveness, each detailed in Section 2.3 and Appendix B. An overview of the training procedure of the the proposed end-to-end differentiable framework is shown in Figure 1 and Algorithm 1.

2.1. Texture Generation

Our depth-aware text-to-texture pipeline builds on two established methods, (Richardson et al., 2023; Tang et al., 2024). Using a differentiable renderer (Laine et al., 2020), we render the object from multiple viewpoints to extract inpainted images, depth maps, normal maps, and UV coordinates. Given a text prompt, each view is painted using a depth-aware text-to-image diffusion model (Rombach et al., 2022), guided by a pre-trained ControlNet (Zhang et al., 2023) that provides depth information This ensures the generated textures align with both the text and depth (geometry) information. We repeat this process iteratively across all viewpoints until the full 3D surface is painted. Due to the stochastic nature of diffusion models, this can result in inconsistent textures and visible seams. To address this, we adopt a dynamic partitioning strategy inspired by (Richardson et al., 2023; Chen et al., 2023), using the *view direction cosine* (see Appendix C) to segment each view into three regions: (i) *generate* – newly visible and unpainted areas, (ii) *refine* – previously painted areas now seen from a better angle (higher cosine), and (iii) *keep* – already well-painted regions that require no update. This partitioning guides the rendering and repainting process to maintain visual consistency. However, this partitioning and certain other operations in the texture generation step are not inherently differentiable, so we introduce mathematical modifications to ensure differentiability. These modifications are essential not only for making the pipeline differentiable, enabling the training of the diffusion model, but also for incorporating gradient information from each component of the texture-generation process. This ultimately makes the entire pipeline aware of the geometry of the mesh being textured. For more on the differentiable mathematical formulation, we refer the reader to Appendix C.1 and Appendix C.2.

2.2. Texture Preference Learning

The texture image, obtained from the *texture generation* step, serves as an input to the next step, *texture preference learning*, to be evaluated and then enhanced according to a differentiable reward which represents human preferences or task-specific objectives. The goal is to fine-tune the pre-trained stable diffusion model's parameters θ such that the texture images maximize a differentiable reward function r. The maximization problem can be then formulated as:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{c}\sim p_c}\left[r\left(\text{TexGen}\left(\boldsymbol{\theta}, \mathbf{c}, \mathbf{v_{gen}}\right), \mathbf{c}\right)\right] \quad (1)$$

where r is a differentiable reward function, $\text{TexGen}(\boldsymbol{\theta}, \mathbf{c}, \mathbf{v_{gen}})$ represents the iterative texturing process from different viewpoints v_{gen} (explained in Section 2.1 and Appendix C), and c is the text prompt randomly sampled from the text dataset p_c. To solve Equation (1), instead of maximizing the reward function, we consider the negative of it as a loss function $L(\theta) = -J(\theta)$. Then, we compute $-\nabla_{\boldsymbol{\theta}} r\left(\text{TexGen}\left(\boldsymbol{\theta}, \mathbf{c}, \mathbf{v_{gen}}\right), \mathbf{c}\right)$ and update θ using gradient-based optimization algorithms and the following update rule: $\theta \leftarrow \theta + \eta \nabla_{\boldsymbol{\theta}} r\left(\text{TexGen}\left(\boldsymbol{\theta}, \mathbf{c}, \mathbf{v_{gen}}\right), \mathbf{c}\right)$.

However, naively optimizing the θ would results in requiring a massive amount of memory which makes the fine-tuning process impractical. This is because of the fact that computing the gradients $\nabla_{\boldsymbol{\theta}} r$ requires backpropagation through multiple diffusion steps and multiple camera viewpoints which stems from the texturing process $\mathrm{TexGen}(\boldsymbol{\theta}, \mathbf{c}, \mathbf{v_{gen}})$. Consequently, this necessitates storing all the intermediate variables during the multi-view texturing process. More specifically, each step in the texturing process involves at least 10 diffusion steps, and with 8–10 total texturing steps (corresponding to the number of camera viewpoints), this results in approximately 100 diffusion steps overall. Each step requires around 3 GB of GPU memory, leading to a total memory demand of roughly 300 GB to train the texture-generation pipeline. Such hardware requirements are nearly infeasible, especially in typical academic environments. To alleviate this issue, we leverage two main approaches to reduce the memory used by our method during the training stage: 1) *low-rank adaptation (LoRA)* (Hu et al., 2022) and 2) *gradient checkpointing* (Chen et al., 2016). Applying LoRA to the U-Net in the latent diffusion model reduces trainable parameters by roughly a factor of 1000 compared to full fine-tuning. Additionally, two levels of activation checkpointing keep memory usage constant across views and diffusion steps, enabling higher-resolution renders and more diffusion steps. Further memory-saving details are provided in Appendix D.

2.3. Geometry-Aware Reward Design

Our differentiable framework enables the design and implementation of task-specific differentiable reward functions (which represent human preferences), defined either in the 2D texture image space or directly on the 3D mesh surface, to guide the texture generation toward the desired outcome. Hence, we propose four novel geometry-aware reward functions to demonstrate the effectiveness of the proposed differentiable framework, each detailed in the following subsections.

Geometry-Texture Alignment Reward. To encourage texture image features (e.g. edges) to be aligned with the geometry of a 3D object, we introduce a differentiable reward function that aligns texture gradient vectors with principal curvature directions on the mesh surface (see Appendix E.1 for details). These curvature vectors encode the surface's bending behavior, and aligning texture gradients with them results in textures that are both visually and semantically consistent with the underlying geomtery of the 3D input mesh. More specifically, for each point in the UV space (texture), obtained by projecting 3D mesh vertices into 2D texture space, we compute the alignment as the squared dot product (cosine similarity) between the normalized texture gradient and corresponding normalized curvature direction vector:

$$R_1 = \frac{1}{N}\sum_{i=1}^{N}(\vec{\mathbf{g}}_i \cdot \vec{\mathbf{c}}_i)^2 \tag{2}$$

where N is the number of UV coordinates, $\vec{\mathbf{g}}_i$ and $\vec{\mathbf{c}}_i$ are the texture gradient and minimum principal curvature at each UV coordinate, respectively.

Geometry-Guided Texture Colorization Reward. To encode surface curvature into texture color, we design a reward function that encourages warm colors (e.g., red, yellow) in regions of high curvature and cool colors (e.g., blue, green) in regions of low curvature. Given a per-pixel scalar curvature map $C(x, y) \in [-1, 1]$, and a curvature threshold $T \in [-1, 1]$, let $I_r(x, y)$ and $I_b(x, y)$ denote the red and blue channels of the predicted texture at pixel (x, y), respectively. For pixels where $C(x, y) > T$, we encourage warm colors by rewarding a higher red-blue channel difference $\Delta_{rb}(x, y) = I_r(x, y) - I_b(x, y)$. Conversely, for $C(x, y) < T$, we encourage cool colors by penalizing this difference. The final reward is the average Δ_{rb} across all pixels:

$$R_2 = \frac{1}{N}\sum_{x,y}\begin{cases}\Delta_{rb}(x, y), & \text{if } C(x, y) > T\\ -\Delta_{rb}(x, y), & \text{if } C(x, y) \le T\end{cases} \tag{3}$$

where N is the total number of pixels. This formulation captures the desired color-geometry correlation. However, its hard conditional logic makes it non-differentiable and unsuitable for gradient-based training. To resolve this, we replace hard thresholds with smooth, sigmoid-based approximations, enabling end-to-end differentiability.

Texture Features Emphasis Reward. This reward encourages texture patterns that emphasize 3D surface structure while maintaining perceptual richness through color variation. It consists of two components: (i) *texture-curvature magnitude alignment* which strengthens texture variations in high-curvature regions to enhance capturing geometric structures, and (ii) *colorfulness* inspired by (Du et al., 2024; Hasler & Suesstrunk, 2003), which discourages desaturated textures by encouraging vibrant color use, measured through the standard deviation and mean distance of opponent color channels from neutral gray. Let $T(x, y) \in [0, 1]^3$ denote the RGB texture at pixel (x, y) and $C(x, y) \in [0, 1]$ be the normalized curvature map in UV space. The *texture-curvature alignment* term is mathematically defined as the negative mean squared error (MSE):

$$R_{magnitude} = -\frac{1}{N}\sum_{x,y}(\nabla\mathbf{T}(x, y) - C(x, y))^2 \tag{4}$$

where $\nabla\mathrm{T}(x, y) = \sqrt{\left\|\frac{\partial \mathrm{T}}{\partial x}(x, y)\right\|^2 + \left\|\frac{\partial \mathrm{T}}{\partial y}(x, y)\right\|^2}$ is the texture gradient magnitude and N is the total number of pixels in the texture image. Then, we augment this reward with a colorfulness term based on opponent color channels (Hasler & Suesstrunk, 2003). Let $R(x, y), G(x, y), B(x, y) \in \mathbb{R}^{H\times W}$ denote the red, green, and blue channels of the texture. We compute the opponent axes:

$$rg = R - G, \quad yb = \frac{1}{2}(R + G) - B \tag{5}$$

the *colorfulness* term is then defined as:

$$R_{color} = \sigma_{rg} + \sigma_{yb} + 0.3\left(|\mu_{rg}| + |\mu_{yb}|\right) \quad (6)$$

where σ_{rg}, σ_{yb}, μ_{rg}, and μ_{yb} are the standard deviations and mean colors offsets corresponding to color opponent components $rg(x,y) = R(x,y) - G(x,y)$ and $yb(x,y) = \frac{R(x,y)+G(x,y)}{2} - B(x,y)$, respectively. The full *texture features emphasis* reward combines both components:

$$R_3 = \alpha_m R_{magnitude} + \alpha_c R_{color} \quad (7)$$

where α_m and α_c are weights for the *texture-curvature magnitude alignment* and *colorfulness* terms, respectively.

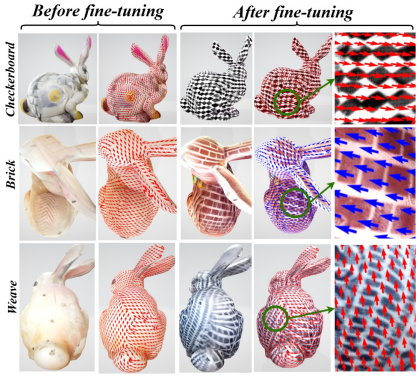

Figure 2. Qualitative results of the geometry-texture alignment experiment on a rabbit (bunny) mesh.

3. Experiments and Results

To show the effectiveness of our approach, we perform four training experiments using the four proposed geomatry-aware reward functions in Section 2.3, and Appendix B, each detailed in the following and Appendix F. We qualitatively and quantitatively compare our results against the the automated version of InTeX's (Tang et al., 2024) method, an state-of-the-art method in the field of texture generation. We report the qualitative results for each experiment in Figure 2, Figure 3, and quantitative results in Table 1 (more results in Appendix F).

Geometry-Texture Alignment. The objective of this task is to generate texture images whose patterns are aligned with the surface geometry of a 3D mesh object, represented by its minimal curvature directions. Figure 2 presents qualitative results on a rabbit mesh textured using different textual prompts. Each row corresponds to a different description, with the main texture pattern written on the left and overlaid vectors indicating the minimal curvature directions on the surface. As illustrated, our method successfully guides texture patterns to follow the underlying geometric cues, a capability that InTex (Tang et al., 2024) lacks (more results in Figure 8). Compared to InTeX, which uses a pre-trained model without fine-tuning, our approach exhibits an interesting capability: the generation of repetitive texture patterns that align with the curvature structure. Additional analysis of this effect are provided in Appendix E.3 and Appendix E.4.

Geometry-Guided Texture Colorization. This task aims to colorize textures based on surface bending intensity, represented by mean curvature, an average of the minimal and maximal curvature directions, on the 3D mesh. Specifically, the model is encouraged to apply warm colors (e.g., red, yellow) in high-curvature regions and cool colors (e.g., blue, green) in low-curvature areas. Figure 9 shows qualitative results on rabbit and cow mesh objects colorirzed with different textual prompts. As illustrated, our method consistently adapts texture colors, regardless of initial patterns, according to local curvature and successfully maps warmth and coolness to geometric variation.

Texrure Features Emphasis. This task aims to learn texture images with salient features (e.g., edges) emphasized at regions of high surface bending, represented by the magnitude of mean curvature. This promotes texture patterns that highlight 3D surface structure while preserving perceptual richness through color variation. Figure 3 shows qualitative results on a rabbit mesh with brick patterns. As illustrated, our method enhances texture features, such as edges and mortar, in proportion to local curvature, a capability InTex (Tang et al., 2024) lacks, often resulting in pattern-less (white) areas, particularly on the back and head of the rabbit (more results in Figure 10).

Figure 3. Qualitative results of the texture features emphasis experiment on a rabbit (bunny) object.

4. Conclusions

This study presents an end-to-end differentiable preference learning framework for 3D texture generation, enabling human preferences, expressed as differentiable reward functions, to be back-propagated through the entire 3D generative pipeline. To demonstrate its effectiveness, we introduce four geometry-aware reward functions that guide texture generation to align closely with the input mesh geometry. The framework is broadly applicable, supporting any differentiable feedback defined in either 2D texture space or directly on the 3D surface, and is extensible to tasks beyond texture generation. As future work, we envision applying this approach to general 3D content creation, including joint optimization of geometry and texture.

Acknowledgements

We acknowledge funding from Mila Tech Transfer Grant with Silicolabs. Compute resources provided by Calcul Quebec and the digital researcher alliance.

References

Abdi, H. and Williams, L. J. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

Avrahami, O., Fried, O., and Lischinski, D. Blended latent diffusion. *ACM transactions on graphics (TOG)*, 42(4): 1–11, 2023.

Black, K., Janner, M., Du, Y., Kostrikov, I., and Levine, S. Training diffusion models with reinforcement learning. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Chen, D. Z., Siddiqui, Y., Lee, H.-Y., Tulyakov, S., and Nießner, M. Text2tex: Text-driven texture synthesis via diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 18558–18568, 2023.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Dong, H., Xiong, W., Goyal, D., Zhang, Y., Chow, W., Pan, R., Diao, S., Zhang, J., Shum, K., and Zhang, T. Raft: Reward ranked finetuning for generative foundation model alignment. *Trans. Mach. Learn. Res.*, 2023.

Du, X., Zhou, Z., Wu, X., Wang, Y., Wang, Z., Zheng, Y., and Jin, C. Multicolor: Image colorization by learning from multiple color spaces. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 6784–6792, 2024.

Fan, Y., Watkins, O., Du, Y., Liu, H., Ryu, M., Boutilier, C., Abbeel, P., Ghavamzadeh, M., Lee, K., and Lee, K. Dpok: Reinforcement learning for fine-tuning text-to-image diffusion models. *Advances in Neural Information Processing Systems*, 36:79858–79885, 2023.

Hasler, D. and Suesstrunk, S. E. Measuring colorfulness in natural images. In *Human vision and electronic imaging VIII*, volume 5007, pp. 87–95. SPIE, 2003.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Jacobson, A., Panozzo, D., et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/.

Laine, S., Hellsten, J., Karras, T., Seol, Y., Lehtinen, J., and Aila, T. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (ToG)*, 39(6):1–14, 2020.

Lin, C.-H., Gao, J., Tang, L., Takikawa, T., Zeng, X., Huang, X., Kreis, K., Fidler, S., Liu, M.-Y., and Lin, T.-Y. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 300–309, 2023.

Liu, S., Li, T., Chen, W., and Li, H. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 7708–7717, 2019.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Paszke, A. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Pearson, K. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11): 559–572, 1901.

Poole, B., Jain, A., Barron, J. T., and Mildenhall, B. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36: 53728–53741, 2023.

Richardson, E., Metzer, G., Alaluf, Y., Giryes, R., and Cohen-Or, D. Texture: Text-guided texturing of 3d shapes. In *ACM SIGGRAPH 2023 conference proceedings*, pp. 1–11, 2023.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Tang, J., Lu, R., Chen, X., Wen, X., Zeng, G., and Liu, Z. Intex: Interactive text-to-texture synthesis via unified depth-aware inpainting. *arXiv preprint arXiv:2403.11878*, 2024.

Wu, J. Z., Ge, Y., Wang, X., Lei, S. W., Gu, Y., Shi, Y., Hsu, W., Shan, Y., Qie, X., and Shou, M. Z. Tune-a-video: One-shot tuning of image diffusion models for text-to-video generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7623–7633, 2023.

Ye, J., Liu, F., Li, Q., Wang, Z., Wang, Y., Wang, X., Duan, Y., and Zhu, J. Dreamreward: Text-to-3d generation with human preference. In *European Conference on Computer Vision*, pp. 259–276. Springer, 2024.

Zeng, X., Chen, X., Qi, Z., Liu, W., Zhao, Z., Wang, Z., Fu, B., Liu, Y., and Yu, G. Paint3d: Paint anything 3d with lighting-less texture diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4252–4262, 2024.

Zhang, L., Rao, A., and Agrawala, M. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 3836–3847, 2023.

A. Differentiable 3D Texture Preference Learning Algorithm

We outline the training procedure of the proposed end-toend differentiable framework in Algorithm 1. The algorithm supports any form of human preference that can be formulated as a differentiable feedback signal, either in the 2D texture space or directly on the 3D surface. This flexibility makes the approach broadly applicable and extensible to a wide range of 3D tasks beyond texture generation.

Algorithm 1 Differentiable 3D Texture Preference Learning

Require: Inference timesteps T, training iterations E, prompt dataset p_c, camera viewpoints for texture generation V_{gen}, camera viewpoints for texture evaluation V_{eval}, pre-trained diffusion model ϵ_θ conditioned on control commands generated by the ControlNet model, and diffusion model weights θ

for $e = 1$ **to** E **do**
  `# Step 1: Texture Generation`
  Sample a prompt $c \sim p_c$, and sample $x_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
  Initialize the 3D asset's parameters a including texture image $albedo$, weight image cnt, and the view cosine cache image $viewcos$.
  for v_{gen} **in** V_{gen} **do**
    $h^i = DiffRender(a, v_{gen})$
    for $t = T$ **to** 1 **do**
      $y^i_{t-1} = ControlNet(h^i, t, c)$
      $x^i_{t-1} = \mu(x^i_t, y^i_{t-1}, t, c) + \sigma_t z, \quad z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    end for
    $Inpaint3DAsset(a, v_{gen})$
    $albedo \leftarrow Update3DAsset(a, v_{gen})$
  end for
  `# Step 2: Texture Preference Learning`
  if 3D Reward **then**
    for v_{eval} **in** V_{eval} **do**
      $h^i = DiffRender(a, v_{eval})$
      $reward = reward + r(h^i)$
    end for
    $reward = reward/len(V_{eval})$
    $g = -\nabla_\theta reward$
  else if Texture Reward **then**
    $g = -\nabla_\theta r(albedo)$
  end if
  $\theta \leftarrow \theta - \eta g$
end for
return θ

B. Symmetry-Aware Reward Design

We propose a differentiable reward function that encourages texture patterns to be symmetric across mirrored UV coordinates while maintaining visual richness through color

variation. We first identify the mesh's intrinsic plane of symmetry using Principal Component Analysis (PCA) (Abdi & Williams, 2010; Pearson, 1901), then compute mirror pairs of surface points across this plane, project these pairs into the 2D UV domain for use in a symmetry-aware reward function, and finally compute the symmetry reward. The four steps are outlined in the following.

Symmetry plane estimation via PCA. To estimate the mesh's dominant symmetry plane, we apply principal component analysis (PCA) to the 3D vertext coordinates. Let $\{v_i \in \mathbb{R}^3\}_{i=1}^N$ denoted the 3D vertext positions of the mesh. We then compute the centroid of the mesh as $\mathbf{c} = \frac{1}{N}\sum_{i=1}^N \mathbf{v}_i$. Next, we construct the covariance matrix of the centered vertices: $\Sigma = \frac{1}{N}\sum_{i=1}^N (\mathbf{v}_i - \mathbf{c})(\mathbf{v}_i - \mathbf{c})^\top$. We then perform eigen-decomposition of Σ, yielding eigenvalues $\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq$ and corresponding orthonormal eigenvectors e_1, e_2, e_3. Teh eigenvector $n = e_1$ associated with teh smallest eigenvalue is to taken to define the normal vector of the symmetry plane, as it corresponds to the direction of least variance in the point cloud (vertices). The symmetry plane Π is then defined as the set of points:

$$\Pi = \left\{ \mathrm{x} \in \mathbb{R}^3 \mid (\mathrm{x} - \mathrm{c}) \cdot \mathrm{n} = 0 \right\} \tag{8}$$

Reflecting vertices across the symmetry plane. To identify symmetric correspondences, we first compute the signed distance of each vertex to the plane: $d_i = (\mathbf{v}_i - \mathbf{c}) \cdot \mathbf{n}$. We keep only the vertices where $d_i > 0$, i.e., those lying on one side of the plane, to avoid duplication. Each selected v_i is then reflected across the plane to compute its mirrored position $v_i^{mirror} \in \mathbb{R}^3$ using: $\mathbf{v}_i^{\text{mirror}} = \mathbf{v}_i - 2\left[(\mathbf{v}_i - \mathbf{c}) \cdot \mathbf{n}\right]\mathbf{n}$.

Projecting 3D mirror points into UV space. To obtain the UV coordinates corresponding to v_i^{mirror}, we perform a closest-point query on the mesh surface to find the nearest triangle containing the surface projection of v_i^{mirror}. Given the triangle's 3D vertices v_1, v_2, v_3, and corresponding UV coordinates $u_1, u_2, u_3 \in [0,1]^2$, we compute barycentric coordinates $(\beta_1, \beta_2, \beta_3)$ of the projected point and obtain the UV coordinates of the mirror as: $\mathbf{u}_i^{\text{mirror}} = \beta_1 \mathbf{u}_1 + \beta_2 \mathbf{u}_2 + \beta_3 \mathbf{u}_3$. We store the UV coordinate u_i of the original vertex along with u_i^{mirror} and form a pair (u_i, u_i^{mirror}) used in our symmetry-awared reward.

Symmetry-aware texture reward design Given a set of UV coordinate pairs $\{(\mathbf{u}_i, \mathbf{u}_i^{\text{mirror}})\}_{i=1}^M$, obtained by projecting symmetric 3D vertex pairs into the UV domain, we define a differentiable reward function that encourages symmetry in the generated texture. Let $\mathcal{T} \in \mathbb{R}^{3\times H\times W}$ denote the RGB texture defined in UV space. For each UV pair, we sample RGB values via bilinear interpolation:

$$\mathcal{T}_i = \text{Sample}(\mathcal{T}, \mathbf{u}_i), \quad \mathcal{T}_i^{\text{mirror}} = \text{Sample}(\mathcal{T}, \mathbf{u}_i^{\text{mirror}}) \tag{9}$$

where $\mathcal{T}_i, \mathcal{T}_i^{\text{mirror}} \in \mathbb{R}^3$ represent the RGB values at each location. Sampling is implemented using the differentiable `grid_sample` operation in Pytorch (Paszke, 2019) to ensure gradient flow during training. To encourage symmetry, we minimize the mean squared error (MSE) between original and mirrored samples:

$$\mathcal{L}_{\text{symmetry}} = \frac{1}{M}\sum_{i=1}^{M} \left\| \mathcal{T}_i - \mathcal{T}_i^{\text{mirror}} \right\|_2^2 \tag{10}$$

We define the symmetry reward as the negative of this loss:

$$\mathcal{R}_{\text{symmetry}} = -\mathcal{L}_{\text{symmetry}} \tag{11}$$

To discourage trivial (e.g., grayscale) solutions, similar to *texture features emphasis* reward (Section 2.3), we augment the symmetry reward with a colorfulness term using Equation (6). The total reward is a weighted combination of the symmetry alignment and colorfulness terms:

$$\mathcal{R}_4 = \alpha_{\text{sym}} \cdot \mathcal{R}_{\text{symmetry}} + \alpha_{\text{color}} \cdot \mathcal{R}_{\text{color}} \tag{12}$$

We use $\alpha_{\text{sym}} = 1.0$ and $\alpha_{\text{color}} = 0.05$ in our experiments.

C. Remarks on The Texture Generation Step

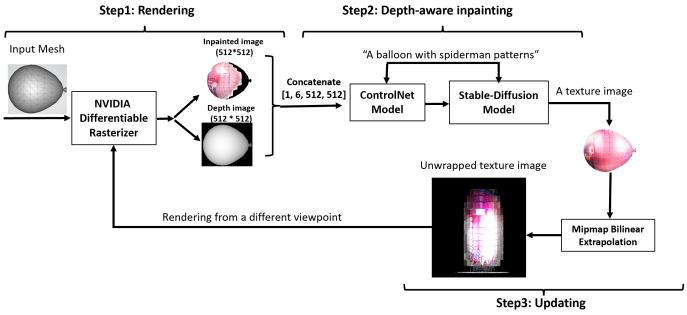

Figure 4. Visualization of three main stages in the *texture generation*: (i) rendering, render the object from a camera viewpoint using a differentiable renderer and extract a rendering buffer, (ii) depth-aware painting, given a text prompt, each viewpoint is painted using a depth-aware text-to-image diffusion model, guided by a pre-trained ControlNet that provides depth information. This ensures the generated textures align with both the text and depth (geometry) information, and (iii) update the final texture. We repeat this process iteratively across all camera viewpoints until the full 3D surface is painted.

Due to the stochastic nature of the diffusion models, applying the iterative painting process for texturing process, explained in Section 2.1, would leads to having inconsistent textures with noticeable seams on the appearance of the object. To resolve this issue, we take a dynamic partitioning approach similar to (Richardson et al., 2023; Chen et al., 2023), where each rendered viewpoint is divided to three regions based on a concept called *view direction cosine*. The

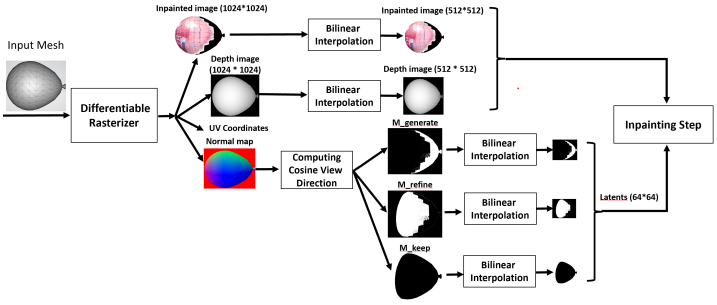

Figure 5. Visualization of *rendering* in the *texture generation* step. For each camera viewpoint, we render the object using a differentiable renderer and extract rendering buffer including painted viewpoint image, depth maps, normal maps, and UV coordinates. Then, using the normal map, obtained from the differentiable renderer, we compute the *view direction cosine*, and then generate three regions (masks), explained above, for each viewpoint: $M_{generate}, M_{refine}$ and M_{keep}. These three regions will serve as input to the next step of the texturing process to enforce the consistency in the output texture.

view direction cosine is utilized to determine how the surface is facing the viewer. Mathematically, the view direction cosine is represented as the cosine of the angle between two vectors: the *view direction vector* which is the direction of the viewer's (in our case, the camera) line of sight, and *mesh's surface normal vectors* associated with each face or vertex of the mesh. These normals indicate the direction perpendicular to the surface at that point (vertex):

$$\text{viewcos } = \cos(\phi) = \frac{\vec{V}.\vec{N}}{|\vec{V}| \cdot |\vec{N}|} \tag{13}$$

where $\vec{V}$ is the view vector, a vector starting from the camera position points toward a point on the surface of the 3D mesh object, $\vec{N}$ is the surface normal, vector perpendicular to the mesh surface, and ϕ is the angle between the view vector and surface normal. During the rendering process, this helps determine how to project and render 3D objects onto a 2D screen, ensuring that objects are viewed from the correct perspective. More specifically, the *view direction cosine* is computed for each rendered viewpoint, and the rendered viewpoint is partitioned into three regions based on the value of *view direction cosine*: 1) *generate* - the regions have been viewed for the first time by the camera and have never been painted during the texturing process, 2) *refine* - the regions that have already been viewed and painted from a viewpoint in previous iterations, but is now seen from a better camera angle (i.e. higher *view direction cosine* values) and should be painted again, and 3) *keep* - the regions that have already been viewed and pained from a good camera angle and should not be painted again. Appendix C.1 presents further details and the differentiable mathematical representation of our *texture generation* step.

C.1. Mathematical Representation

Mathematically, the texture-generation process can be formulated as steps below (see Figure 4): (i) *rendering*, render the object using a differentiable renderer (Laine et al., 2020) and extract a rendering buffer h^i corresponding to the camera viewpoint i, including painted viewpoint image, depth maps, normal maps, and UV coordinates:

$$h^i = f_{render}(a, v_{gen}) \tag{14}$$

where a is the 3D asset's parameters including texture image, the *view direction cosine* for the current view, *view direction cosine* for previous iterations, v_{gen} is the camera viewpoints for texture-generation process, and $i \in v_{gen}$ is the current viewpoint. Using the normal map, obtained from the differentiable renderer, we then compute the *view direction cosine*, and then generate three regions (masks), explained above, for each viewpoint: $M_{generate}, M_{refine}$ and M_{keep}. These three regions will serve as input to the next step of the texturing process to enforce the consistency in the output texture. Figure 5 visually demonstrates the details of the output generated by the differentiable renderer. (ii) *Depth-aware painting*, at each denoising step t, the depth-aware diffusion process can be formulated as below:

$$\begin{aligned} y^i_{t-1} &= f_{controlnet}(h^i, t, c) \\ x^i_{t-1} &= \mu(x^i_t, y^i_{t-1}, t, c) + \sigma_t z, \quad z \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \end{aligned} \tag{15}$$

where $f_{controlnet}$ represents the ControlNet model whose parameters are fixed during the training, h^i is the differentiable renderer output, y^i_{t-1} is the control signal generated by the ControlNet model at timestep t for viewpoint i, and c is the prompt embdedding obtained from the CLIP encoder model (Radford et al., 2021). To enforce the consistency during the painting process, we follow (Avrahami et al., 2023) to incorporate latent mask blending in the latent space of the diffusion process. Specifically, by referring to the part that we wish to modify as $m_{blended}$ and to the remaining part as $m_{keep} = 1 - m_{blended}$, we blend the two parts in the latent space, as the diffusion progresses. This modifies the sampling process in a way that the diffusion model does not change the *keep* regions of the rendered image. Therefore, the latent variable at the current diffusion timestep t is computed as:

$$z^i_t \leftarrow z^i_t \odot m_{\text{blended}} + z^i_{O_t} \odot (1 - m_{\text{blended}}) \tag{16}$$

where $z^i_{O_t}$ is the latent code of the original rendered image with added noise at timestep t and for the viewpoint i and $m_{blended}$ is the painting mask defined below:

$$m_{blended} = \begin{cases} M_{\text{generate}}\,, & t \le (1-\alpha) \times T \\ M_{\text{generate}} \cup M_{\text{refine}}\,, & t > (1-\alpha) \times T \end{cases} \tag{17}$$

where T is the total number of diffusion denoising steps and α is the refining strength which controls the level of refinement over the viewpoints - the larger it is, the less strong refinement we will have during the multi-viewpoint

texturing process. Lastly, when the blending latent diffusion process is done, we output the image by decoding the resultant latent using the pre-trained variational autoencoder (VAE) existing in the stable diffusion pipeline.

C.2. Modifications To Make Texture Generation Step Differentiable

Differentiable Camera Pose Computation. Instead of using traditional camera positioning, to enable end-to-end training with backpropagation through the camera positioning steps, we provide a re-implementation of this procedure using PyTorch (Paszke, 2019) operations, preserving gradient flow. More specifically, camera poses are computed using statically generated camera-to-world transformation matrices from spherical coordinates (elevation and azimuth). This process involved the following steps: (i) spherical to cartesian conversion - given azimuth ϕ, elevation θ, and radius r, the camera position $c \in \mathbb{R}^3$ is computed as:

$$\begin{aligned} x &= r\cos(\theta)\sin(\phi), \\ y &= -r\sin(\theta), \\ z &= r\cos(\theta)\cos(\phi), \\ \mathbf{c} &= [x, y, z]^\top + \mathbf{t}, \end{aligned} \tag{18}$$

where $\mathbf{t}$ is the target point. (ii) Look-at matrix construction - the rotation matrix $R \in \mathbb{R}^{3\times3}$ is computed using a right-handed coordinate system by constructing orthonormal basis vectors:

$$\begin{aligned} \mathbf{f} &= \text{ normalize } (\mathbf{c} - \mathbf{t}) \quad \text{(forward vector)}, \\ \mathbf{r} &= \text{ normalize } (\mathbf{up} \times \mathbf{f}) \quad \text{(right vector)}, \\ \mathbf{u} &= \text{ normalize } (\mathbf{f} \times \mathbf{r}) \quad \text{(up vector)}, \end{aligned} \tag{19}$$

where $\mathbf{up} = [0, 1, 0]^T$. The camera pose matrix $T \in \mathbb{R}^{4\times4}$ is then presented as:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{c} \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{20}$$

This modification transforms a previously non-trainable preprocessing step into a fully differentiable component, allowing camera positions and orientations to participate in gradient-based learning, particularly during texture preference learning where the texture is optimized to match geometry-aware reward signals.

Differentiable Viewpoint Dynamic Partitioning (Masking). In the original implementation of the texture generation step (Tang et al., 2024), dynamic viewpoint partitioning for multi-view texturing was performed using hard, non-differentiable binary masks based on thresholding and bitwise operations. Specifically, three disjoint masks were used to guide different behaviors across image regions during training: (i) a generation mask $M_{generate}$, (ii) a refinement mask M_{refine}, and (iii) a keep mask M_{keep}. Let $c \in \mathbb{R}^{H\times W}$ denotes the count map (indicating coverage across viewpoints), and $v, v_{old} \in \mathbb{R}^{H\times W}$ be the view-consistency scores for the current and cached viewpoints, respectively. The original masks were defined as:

$$\begin{aligned} M_{\text{generate}} &= 1[c < 0.1] \\ M_{\text{refine}} &= 1\left[v > v_{\text{old}}\right] \cdot \neg M_{\text{generate}} \\ M_{\text{keep}} &= \neg M_{\text{generate}} \cdot \neg M_{\text{refine}} \end{aligned} \tag{21}$$

where $1[\cdot]$ denotes the indicator function and logical operators like, $\neg$ produce discrete masks, making them non-differentiable and unsuitable for gradient-based optimization. To enable end-to-end differentiability, we replace these hard decisions with soft, continuous approximations using the sigmoid function. Let $\sigma_k(x) = \frac{1}{1+\exp(-kx)}$ denote the sigmoid function with steepness k. We define soft masks as follows: (i) soft generation mask -

$$M_{\text{generate}}^{\text{soft}} = 1 - \sigma_k(c - 0.1) \tag{22}$$

This approaximates $1[c < 0.1]$, where $k = 100$ ensures a sharp transition.(ii) soft refinement mask -

$$M_{\text{refine}}^{\text{soft}} = \sigma_k\left(v - v_{\text{old}}\right) \cdot \left(1 - M_{\text{generate}}^{\text{soft}}\right) \tag{23}$$

This replaces the hard bitwise AND with elementwise multiplication and soft comparison. (iii) soft keep mask -

$$M_{\text{keep}}^{\text{soft}} = \left(1 - M_{\text{generate}}^{\text{soft}}\right) \cdot \left(1 - M_{\text{refine}}^{\text{soft}}\right) \tag{24}$$

These soft masks retain the semantics of the original binary partitioning but allow gradients to propagate through all mask operations, enabling joint optimization with the diffusion model during our texture preference learning step. We also apply bilinear interpolation in the zooming operation to maintain spatial smoothness across render resolutions.

D. Remarks on Memory-saving Details in The Texture Preference Learning Step

Activation Checkpointing. Gradient or activation checkpointing (Chen et al., 2016) is a technique that trades compute for memory. Instead of keeping tensors needed for backward alive until they are used in gradient computation during backward, forward computation in checkpointed regions omits saving tensors for backward and recomputes them during the backward pass. In our framework, we apply two levels of gradient checkpointing to our pipeline. First, *low-level checkpointing*, where we apply the checkpointing technique to the sampling process in the diffusion model by only storing the input latent for each denoising step, and re-compute the UNet activations during the backpropagation. Second, *high-level checkpointing*, where we apply the checkpointing technique to every single-view texturing process in our multi-view texturing algorithm. Practically, what

it means is that, for each viewpoint during the multi-view iterative texturing process, intermediate tensors inside the per-view texturing are not saved. Instead, only the inputs (camera pose, prompt embeddings, etc.) are saved. Then, during the backward pass, we recompute the forward pass of the per-view texturing again so gradients can flow backward. Overall, this technique allows to keep memory usage constant even across many viewpoints and diffusion steps which consequently leads to scaling to more views and using higher-resolution renders or more diffusion steps to have a more realistic and visually appealing texture image without any memory issues.

Low-Rank Adaptation (LoRA). LoRA is a technique, originally introduced in (Hu et al., 2022) for large language models, that keeps the pre-trained model weights fixed and adds new trainable low-rank matrices into each layer of the neural architecture which highly reduces the number of trainable parameters during the training process. Mathematically, for a layer with base parameters W whose forward pass is $h = Wx$, the LoRA adapted layer is $h = Wx + BAx$, where A and B are the low-rank trainable matrices and the W remains unchanged during the training process. In our case, we apply the LoRA techniques to the U-Net architecture inside the latent diffusion model (LDM) (Liu et al., 2019) which lies at the heart of our texture generation step. This allows us to fine-tune the LDM using a parameter set that is smaller by several orders of magnitude, around 1000 times fewer, than the full set of LDM parameters.

E. Remarks on Geometry-Aware Reward Design

E.1. Principal Curvature Directions

Principal curvature directions at a point on a surface describe how the surface bends in different directions. Curvature is defined as $k = \frac{1}{R}$ where R is the radius of the circle that best fits the curve in a given direction. Among all possible directions, two orthogonal directions, called the principal curvature directions, are of special interest: the maximum and minimum curvature directions. The minimum curvature direction corresponds to the direction in which the surface bends the least (i.e., the largest fitting circle), while the maximum curvature direction corresponds to the direction in which it bends the most (i.e., the smallest fitting circle). See Figure 6 for an illustration. Additionally, the mean curvature at a point is defined as the average of the two principal curvatures: $H = \frac{1}{2}(k_1 + k_2)$. We use the Libigl library (Jacobson et al., 2018) to compute both the mean curvature values and the principal curvature directions on our 3D mesh surfaces.

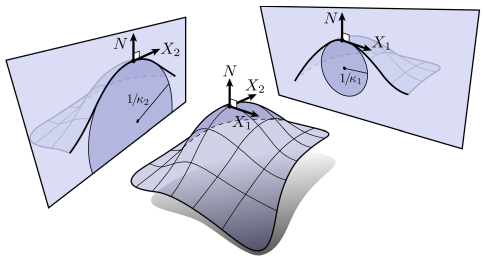

Figure 6. Visualization of principal curvatures (Image Credit: Keenan Crane). The minimum curvature direction corresponds to the direction in which the surface bends the least (i.e., the largest fitting circle), while the maximum curvature direction corresponds to the direction in which it bends the most (i.e., the smallest fitting circle).

E.2. Remarks on Geometry-Guided Colorization and Texture Features Emphasis Rewards

Note that to compute the curvature-quided colorization reward, we need the mean curvature 2D map as a way to incorporate mesh's geometry information into texture learning objectives (Equation (3)). To this end, we construct a mean curvature 2D map, a 2D image in UV space that encodes the mean curvature of the underlying 3D mesh surface and then compute the rewards in the 2D UV space by comparing the texure image and the 2D curvature map. This process involves the following key steps.

UV-Pixel mapping. Each mesh vertex is associated with a UV coordinate $(u, v) \in [0, 1]^2$. These UV coordinates are scaled to a discrete $H \times W$ pixel grid representing the output texture space.

Per-face barycentric interpolation. For each triangular face of the mesh, we project its UV coordinates to the texture grid, forming a triangle in 2D space. We then compute a barycentric coordinate transform that expresses any point within the triangle as a convex combination of its three corners.

Compute barycentric coordinates. We iterate over each pixel inside the UV triangle's bounding box and compute its barycentric coordinates. If a pixel lies within the triangle, its mean curvature value is interpolated from the curvature values at the triangle's corresponding 3D vertices using:

$$\begin{aligned} \text{val} = \ & \text{bary}[0] \cdot \text{curv}_v[\text{tri\_v}[0]] + \\ & \text{bary}[1] \cdot \text{curv}_v[\text{tri\_v}[1]] + \\ & \text{bary}[2] \cdot \text{curv}_v[\text{tri\_v}[2]] \end{aligned} \tag{25}$$

where $\text{bary}[i]$ is the i-th barycentric coordinate of the pixel relative to the UV triangle, $\text{tri\_v}[\text{i}]$ is the index of the i-th 3D vertex corresponding to the UV triangle corner, and curv_v is

the mean curvature array, containing one scalar curvature per 3D vertex. This produces a smoothly interpolated curvature value for each pixel within the UV triangle.

Accumulation and averaging. Each pixel may be covered by multiple UV triangles. Therefore, we accumulate curvature values and record the number of contributions per pixel. After processing all triangles, we normalize the curvature value at each pixel by the number of contributions, producing a final per-pixel average curvature. The output is then a dense 2D curvature texture defined over the UV domain. This texture captures the surface's geometric structure and is suitable for use in differentiable training objectives or geometric regularization.

E.3. Remarks on Geometry-Texture Alignment Reward

Note that curvature is a vertex-level attribute defined per vertex on the 3D mesh surface, whereas texture gradients are defined per pixel in the 2D UV space. Therefore, to guide texture generation using surface-aware vector fields and to compute cosine similarity between curvature vectors and texture gradients (Equation (2)), both must be in the same space. To address this, we project the minimum principal curvature direction vectors, defined at each vertex on the 3D mesh, into the corresponding 2D UV space. Unlike scalar quantities like mean curvature, vector fields require special process to preserve both direction and tangency. This process involves the following steps.

Compute principal curvature directions. We use the method from libigl (Jacobson et al., 2018) to compute the minimum principal curvature direction $d_i \in \mathbb{R}^3$ at each vertex $v_i \in \mathbb{R}^3$ of the mesh. Each direction vector lies in the tangent plane of the surface at its vertex.

Project to the tangent plane. Although the direction vectors are already tangent to the surface by construction, we explicitly project them onto the local tangent plane to remove residual normal components (due to numerical noise). For each vertex: $\mathbf{d}_i^{\tan} = \mathbf{d}_i - (\mathbf{d}_i \cdot \mathbf{n}_i)\,\mathbf{n}_i$ where $\mathbf{n}_i$ is the surface normal at vertex i.

Trace vector to find triangle containment. For each projected vector $\mathbf{d}_i^{\tan}$, we trace it forward from the verex v_i to obtain a new point: $\mathbf{p}_i = \mathbf{v}_i + \frac{1}{\lambda_i} \cdot \mathbf{d}_i^{\tan}$ where λ_i is a normalization constant to ensure the traced vector remains within a local triangle. We then search among the projected triangles incident to v_i to find one in which the point $\mathbf{p}_i$ lies. If found, we compute the barycentric coordinates $\beta_i = [\beta_1, \beta_2, \beta_3]$ of $\mathbf{p}_i$ relative to the triangle's projected vertices.

Convert to UV coordinates. Using the barycentric weights and the corresponding UV triangle $(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$, we map the end of the vector to UV space: $\mathbf{u}_{end} = \beta_1\mathbf{u}_1 + \beta_2\mathbf{u}_2 + \beta_3\mathbf{u}_3$. The UV origin $\mathbf{u}_{orig}$ is determined from the texture coordinate associated with vertex v_i. The 2D texture-space direction vector is then:

$$\mathbf{d}_i^{\mathbf{uv}} = \frac{\mathbf{u}_{\text{end}} - \mathbf{u}_{\text{orig}}}{\|\mathbf{u}_{\text{end}} - \mathbf{u}_{\text{orig}}\|} \tag{26}$$

Populate texture coordinates. Since a single vertex may be associated with multiple UV coordinates (due to UV seams), the final $\mathbf{d}_i^{\mathbf{uv}}$ is assigned to all associated texel indices. This step builds a complete UV-space vector field aligned with curvature directions.

Post-processing and smoothing. Some vertices may not yield a valid projection due to degenerate geometry, occlusion, or sharp creases. For these cases, we apply Laplacian smoothing, and iterative neighbor averaging technique using a texture-space adjacency graph. After resolving missing values, we apply several iterations of vector field smoothing to reduce visual noise while preserving directionality: $\mathbf{d}_i^{\text{uv}} \leftarrow$ mean(neighbors of i). This vector field in UV space is then used to supervise or visualize alignment of learned texture patterns with surface geometry using Equation (2).

E.4. Emergence of Repetitive Patterns through Geometry-Aligned Sampling.

An interesting observation in our results (see Figure 2 and Figure 8) is the emergence of repetitive texture patterns after fine-tuning with the geometry-texture alignment reward. This behavior stems from the mismatch between the number of curvature vectors and the number of texture gradients. Specifically, curvature is a vertex-level attribute defined per mesh vertex, whereas texture gradients are defined per pixel in the 2D UV space. For example, in the bunny mesh example, we have 4487 curvature vectors (one per vertex), but over one million texture gradients from a 1024×1024 texture image. To compute cosine similarity between curvature vectors and texture gradients, both must be in the same space and of the same dimension. We address this by sampling the texture gradients at the UV coordinates corresponding to mesh vertices, the same positions where curvature vectors are defined. This results in two aligned tensors (both of size 4487 in the bunny case), enabling the reward function to compare directional alignment directly. Over time, this differentiable sampling mechanism guides the model to place strong texture features (e.g., edges) at those specific UV coordinates. As these positions remain fixed (being tied to the mesh vertices), the model learns to reinforce edge features at the same UV locations across views and iterations. This alignment results in visually repetitive patterns in the generated textures, especially in regions of high curvature.

Table 1. A quantitative comparison between our proposed method (after fine-tuning) and InTeX (Tang et al., 2024) (before fine-tuning) across four experiments. We repeat each experiment five times and report the mean and standard deviation values of rewards. GEO-TEX-ALIGN, GEO-TEX-COLOR, TEX-EMPHASIS, and SYM-AWARE stand for geometry-texture alignment reward (Section 2.3), geometry-guided texture colorization reward (Section 2.3) , texture features emphasis reward (Section 2.3), and symmetry-aware texture generation reward (Appendix B), respectively.

REWARD	OURS	INTEX
GEO-TEX-ALIGN	**0.3347** $\pm$ 0.0039	0.2915 $\pm$ 0.0019
GEO-TEX-COLOR	**0.6006** $\pm$ 0.0003	-0.0256 $\pm$ 0.0195
TEX-EMPHASIS	**-0.1106** $\pm$ 0.0002	-0.1423 $\pm$ 0.0007
SYM-AWARE	**0.0114** $\pm$ 0.0093	-0.0635 $\pm$ 0.0039

F. Additional Experiments and Results

We show expanded results from the main paper featuring more texture image and more viewpoints of different 3D objects. Figure 7, Figure 8, Figure 9, and Figure 10 shows the qualitative results of the symmetry-aware, geometry-texture alignment, geometry-guided texture colorization, and texture features emphasis experiments, respectively.

Symmetry-Aware Texture Generation. The goal of this experiment is to encourages texture consistency across symmetric regions of a 3D object (see Appendix B). We evaluate this by training our pipeline using the symmetry reward on a balloon mesh object. Figure 7 presents qualitative results before and after fine-tuning with the symmetry reward. We show the rendered 3D object from multiple viewpoints, alongside the corresponding texture images (the last row), which highlight the symmetric regions. A vertical dashed line marks the symmetry axis in each texture image. The purple plane passing through the center of the balloon in each viewpoint indicates the estimated symmetry plane of the object. As shown, compared to the pre-trained model (Tang et al., 2024), our method generates textures that are more consistent across symmetric parts of the mesh, demonstrating the reward's effectiveness in enforcing symmetry consistency. Without symmetry supervision, patterns often differ noticeably between sides (more results in Figure 7).

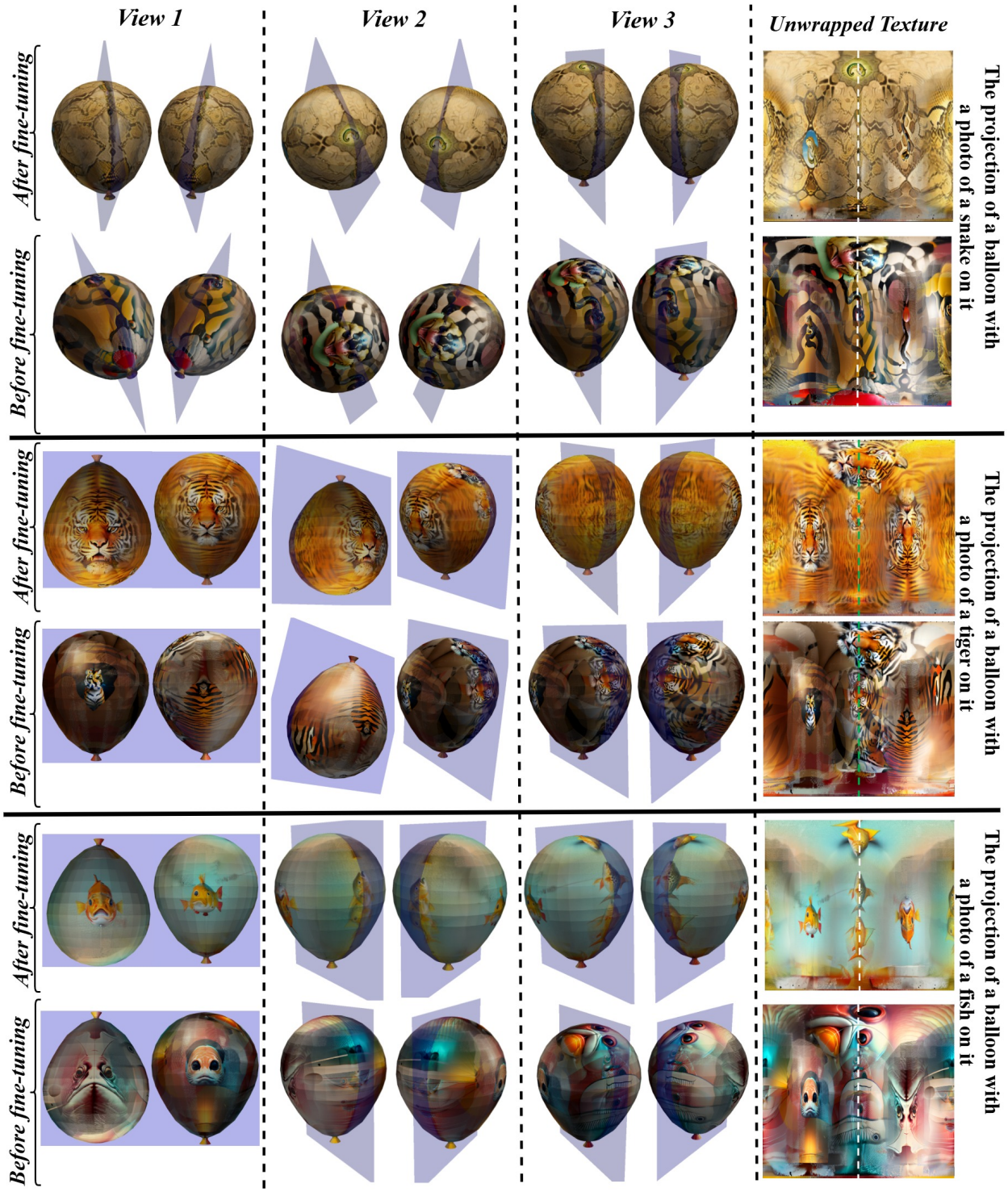

Figure 7. Qualitative results of symmetry-aware experiment for different examples on a balloon mesh object. For each example (each rwo), we show the rendered 3D object from multiple viewpoints, alongside the corresponding texture images (rightmost column), which highlight the symmetric regions. A vertical dashed line marks the symmetry axis in each texture image. The purple plane passing through the center of the balloon in each viewpoint indicates the estimated symmetry plane of the object. As shown, compared to the pre-trained model (Tang et al., 2024), our method generates textures that are more consistent across symmetric parts of the mesh. Without symmetry supervision, patterns often differ noticeably between sides. In contrast, textures trained with the proposed symmetry reward exhibit visually coherent features across symmetric regions, demonstrating the reward's effectiveness in enforcing symmetry consistency.

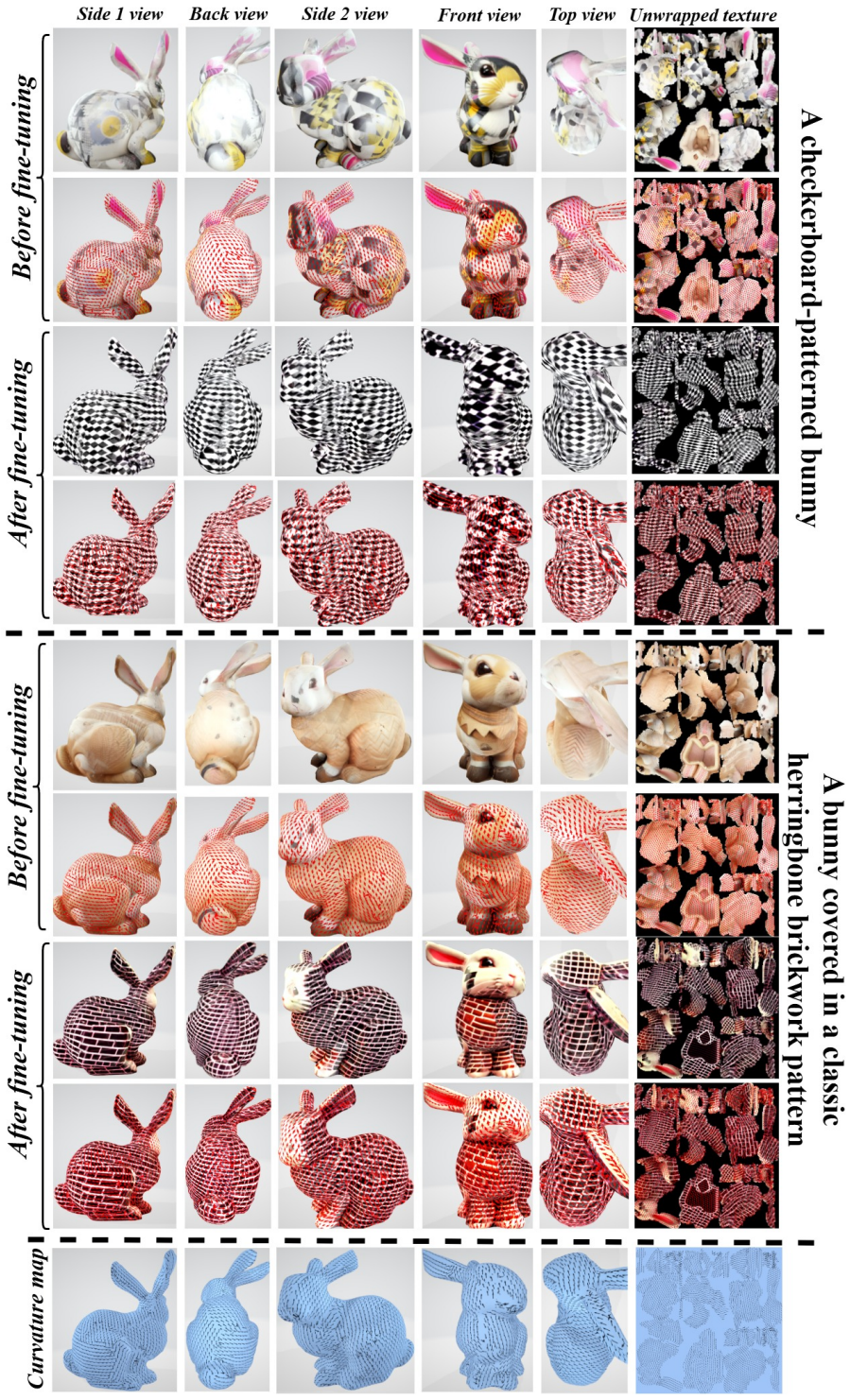

Figure 8. Qualitative results of the geometry-texture alignment experiment on a rabbit (bunny) mesh. For each example, we show the rendered 3D object from multiple viewpoints, with the corresponding texture image in the rightmost column. Minimum curvature vectors, representing the underlying surface geometry, are visualized in the bottom row and overlaid on the textured objects for comparison. As shown, our method produces textures whose patterns align more closely with the mesh's curvature directions, unlike InTex (Tang et al., 2024). Moreover, a notable outcome in our results is the emergence of repetitive texture patterns after fine-tuning with the geometry-texture alignment reward. This behavior arises from the differentiable sampling strategy used during reward computation. Specifically, it encourages the model to place edge features at specific UV coordinates which ultimately results in structured and repeated patterns in the texture (see Appendix E.4).

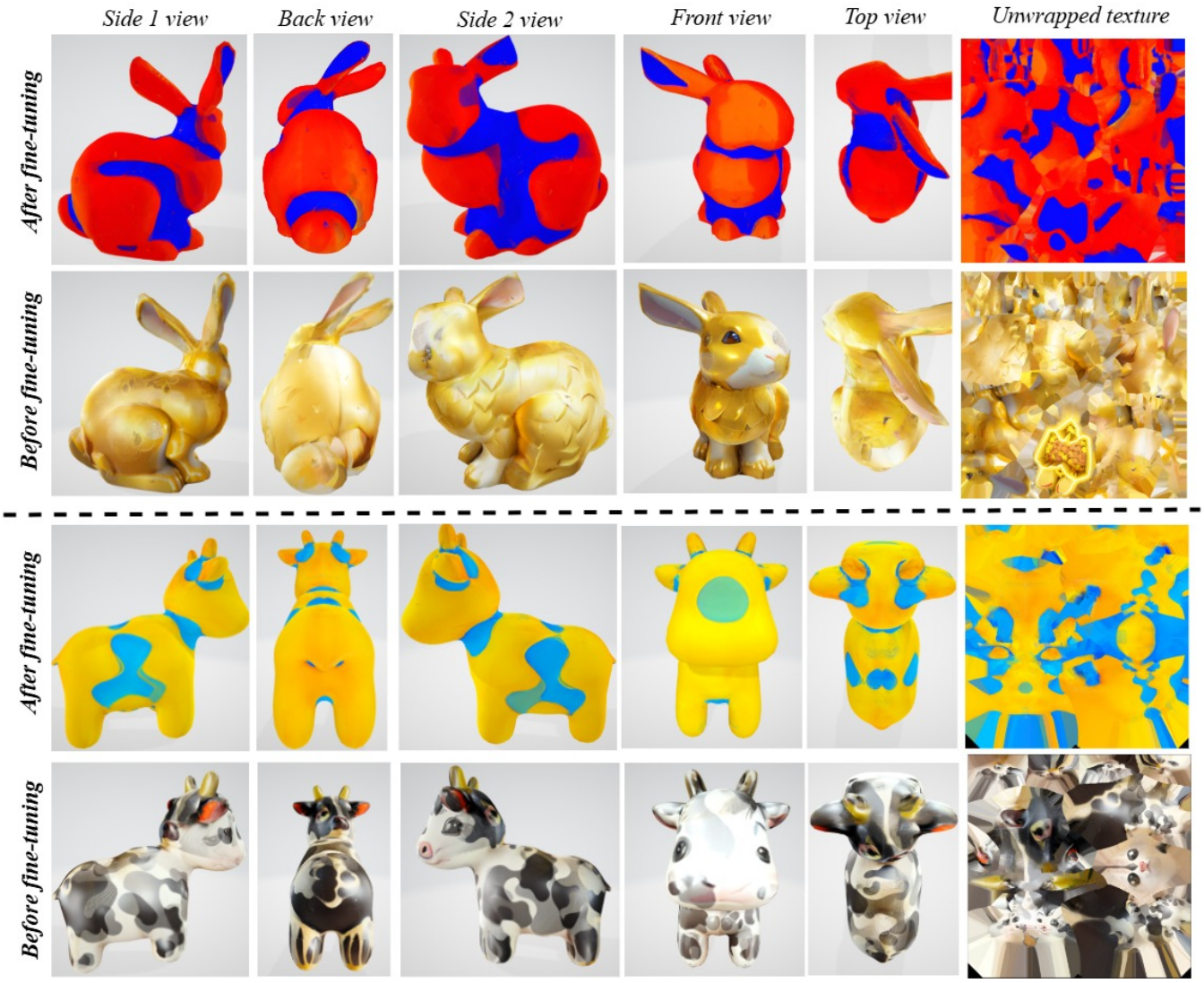

Figure 9. Qualitative results of the geometry-guided texture colorization experiment on rabbit (bunny) a cow mesh objects. For each example, we show the rendered 3D object from multiple viewpoints, with the corresponding texture image in the rightmost column. The goal is to colorize textures based on surface bending intensity, represented by mean curvature, an average of the minimal and maximal curvature directions, on the 3D mesh. Specifically, the model is encouraged to apply warm colors (e.g., red, yellow) in high-curvature regions and cool colors (e.g., blue, green) in low-curvature areas. As illustrated, our method consistently adapts texture colors, regardless of initial patterns, according to local curvature and successfully maps warmth and coolness to geometric variation.

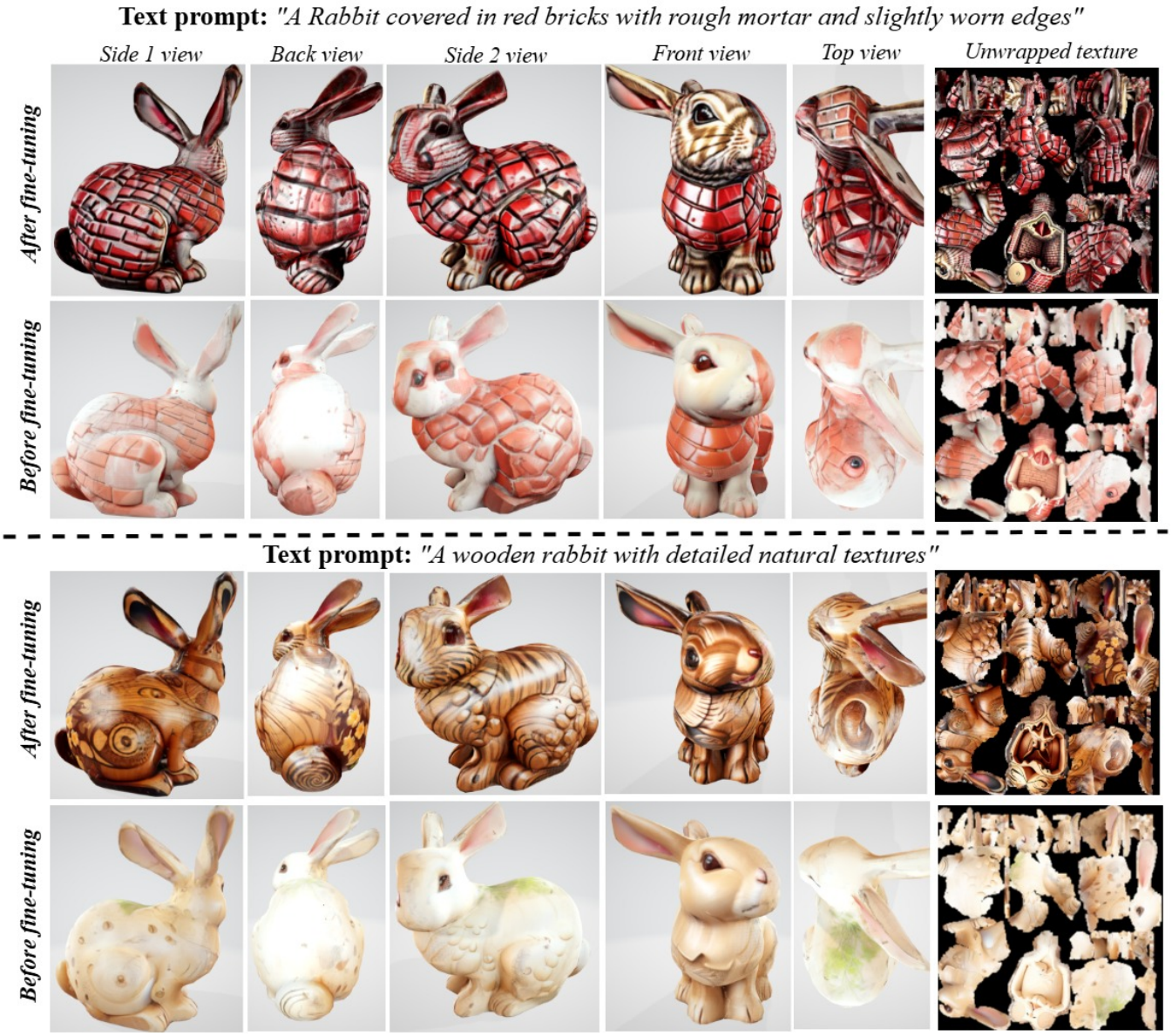

Figure 10. Qualitative results of the texture features emphasis experiment on a rabbit (bunny) objects with different text prompts. For each example, we show the rendered 3D object from multiple viewpoints, with the corresponding texture image in the rightmost column. This goal is to learn texture images with salient features (e.g., edges) emphasized at regions of high surface bending, represented by the magnitude of mean curvature. This encourages texture patterns that highlight 3D surface structure while preserving perceptual richness through color variation. As illustrated, our method enhances texture features, such as edges and mortar, in proportion to local curvature, a capability InTex (Tang et al., 2024) lacks, often resulting in pattern-less (white) areas, particularly on the back and head of the rabbit.