

Texture Generation on 3D Meshes with Point-UV Diffusion

Xin Yu¹ Peng Dai¹ Wenbo Li² Lan Ma³ Zhengzhe Liu^{2†} Xiaojuan Qi^{1†}

¹The University of Hong Kong, ²The Chinese University of Hong Kong, ³TCL Corporate Research
 {yuxin, daipeng, xjqj}@eee.hku.hk {wenboli, zzliu}@cse.cuhk.edu.hk rubyma@tcl.com



Figure 1. A gallery of generated results by our Point-UV diffusion. Our method is capable of processing meshes of any genus, generating diversified, geometry-compatible, and high-fidelity textures.

Abstract

In this work, we focus on synthesizing high-quality textures on 3D meshes. We present Point-UV diffusion, a coarse-to-fine pipeline that marries the denoising diffusion model with UV mapping to generate 3D consistent and high-quality texture images in UV space. We start with introducing a point diffusion model to synthesize low-frequency texture components with our tailored style guidance to tackle the biased color distribution. The derived coarse texture offers global consistency and serves as a condition for the subsequent UV diffusion stage, aiding in regularizing the model to generate a 3D consistent UV texture image. Then, a UV diffusion model with hybrid conditions is developed to enhance the texture fidelity in the 2D UV space. Our method can process meshes of any genus, generating diversified, geometry-compatible, and high-fidelity textures. Code is available at <https://cvmi-lab.github.io/Point-UV-Diffusion>.

1. Introduction

Texturing 3D meshes is a fundamental task in computer vision and graphics. It enhances the visual richness of 3D objects, thereby facilitating their application in various fields such as video games, 3D movies, and AR/VR technologies. However, generating high-quality textures can be daunting and time-consuming, often requiring specialized knowledge and resources. As such, there is a pressing need for an efficient approach to automatically create high-quality textures on 3D meshes.

Despite the substantial progress the community has made in 2D image synthesis and 3D shape generation using GANs [14, 21, 49, 12] or diffusion models [39, 41, 40, 20], crafting realistic textures on mesh surfaces remains challenging. One major difficulty stems from the need for suitable 3D representations for texture synthesis. Early approaches investigate the use of voxels [5, 50, 6] or point clouds [11] and synthesize point/voxel colors. However, they can only afford to synthesize low-resolution results with low-fidelity textures due to memory and model complexity constraints. In response, Texture Fields [34] adopts an implicit representation with the potential to synthesize

[†]: Corresponding authors

high-resolution textures, but still could not yield satisfactory results as shown in Figure 2 (a): over-smoothed results. Most recently, Siddiqui *et al.* [43] propose to parameterize the shape as tetrahedral meshes and introduce tetrahedral mesh convolution to enhance local details. Albeit improving results, tetrahedral parameterization inevitably destroys geometric details of the input mesh and thus cannot faithfully preserve the original structure. As shown in Figure 2 (b), the delicate structures of the chair’s back are absent. Moreover, the generative models utilized in these methods are limited to GANs [34, 43] and VAEs [34, 13]. The more advanced diffusion model, which could potentially open up new avenues for high-quality texture generation, remains insufficiently explored.

In this paper, we delve into a novel texture representation based on UV maps and investigate the advanced diffusion model for texture generation. The 2D nature of the UV map enables it to circumvent the cost of high-resolution point/voxel representations. Besides, the UV map is compatible with arbitrary mesh topologies, thereby preserving the original geometric structures. However, while promising, direct integration of the UV map representation with a 2D diffusion model presents challenges in synthesizing seamless textures, leading to severe artifacts, as shown in Figure 2 (c). This occurs because the UV mapping process fragments the continuous texture on the 3D surface into isolated patches on the 2D UV plane (see Figure 3).

To this end, we introduce Point-UV diffusion, a two-stage coarse-to-fine framework consisting of point diffusion and UV diffusion. Specifically, we initially design a point diffusion model to generate color for sampled points that act as low-frequency texture components. This model is equipped with a style guidance mechanism that alleviates the impact of biased color distributions in the dataset and facilitates diversity during inference. Next, we project these colorized points onto the 2D UV space with 3D coordinate interpolation, thereby generating a coarse texture image that maintains 3D consistency and continuity. Given the coarse textured image, we develop a UV diffusion model with elaborately designed hybrid conditions to improve the quality of the textures (see Figure 2 (ours) and Figure 1).

In short, our contributions are as follows: 1) We propose a new framework for texture generation for given meshes. Our representation can handle meshes with arbitrary topology and is able to faithfully preserve geometric structures. 2) To the best of our knowledge, we are the first to train a diffusion model specifically for mesh texture generation. Our coarse-to-fine framework allows us to enjoy the efficiency of 2D representation while enhancing 3D consistency. 3) We compare our approach with multiple methods in unconditional generation and achieve state-of-the-art results. Furthermore, we demonstrate that our method can be easily extended to scenarios with text-conditioning and



Figure 2. **Comparisons with different methods.** Our generative results (a) possess high-quality details, (b) faithfully preserve the mesh structure, and (c) are better consistent with the given shape, compared with Texture Fields [34], Texturify [43] and 2D diffusion [17], respectively.

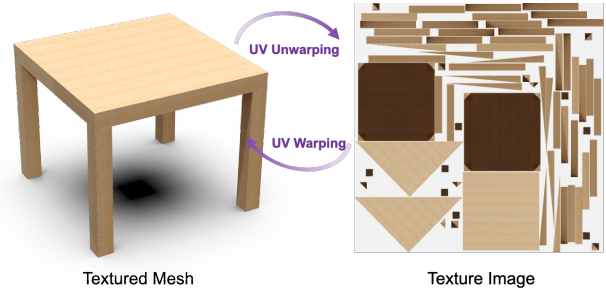


Figure 3. **Illustration of UV mapping process.** It establishes connections between the 2D texture map and the surface appearance of 3D shape.

image-conditioning.

2. Related Work

Texture generation. Early works [46, 48, 6] propose using voxels to represent colors. However, due to a cubic increase in memory usage and computational cost with rising resolution, these methods produce only coarse, low-resolution textures. For meshes, recent work in texture generation [43] predicts the color for each face, but the representation capacity remains largely constrained by the mesh’s resolution. Another set of works [35, 47, 31, 13] leverage UV mapping, but they require either spherical parameterization or part segmentation and, therefore, are confined to handling low-genus shapes. Other works [37, 15] rely on image exemplars, while our approach focuses on unconditional generation. Recently, implicit functions attract increasing attention for generative tasks [12, 36, 44, 3, 9, 42, 34, 25]. Most of this research centers on generating 3D-aware images [44, 3, 9, 42], rather than synthesizing textures for given 3D meshes. Texture Fields [34] represents

the most similar work to our task, which, however, tends to produce over-smoothed results.

Some methods focus on test-time optimization for generation tasks. For instance, Text2Mesh [29] employs CLIP [38] to design a loss function. This function stylizes a 3D mesh to align with a target text prompt, predicting both the color and displacement for each mesh vertex. More recently, DreamFusion [36] and Magic3D [24] leverage pre-trained stable diffusion [40] for score distillation, creating either NeRF [30] or a 3D mesh. However, these approaches often demand extensive optimization time or necessitate alterations to the existing mesh geometry [29]. In this paper, our objective is to generate textures for specified full-3D meshes with arbitrary topology. This goal entails two fundamental requirements: 1) preserving the original mesh’s geometric structure; 2) producing a 3D texture representation exportable as either vertex color or a uv-texture-image, instead of merely generating/rendering multi-view or 3D-aware images.

Diffusion models for 3D generation. In addition to 2D image generation, diffusion models recently gain significant attention in 3D generation, leading to many related works [54, 19, 20, 33, 36, 52, 1]. For instance, Zhou et al. [54] introduce point-voxel diffusion for point cloud generation and completion. Hui et al. [20] and Hu et al. [19] suggest a compact wavelet domain representation for shapes, enabling higher-quality shape creation via diffusion models. Nichol et al. [33] present Point-E, a system that uses a text-conditional strategy to produce colored 3D point clouds. This approach enables an efficient synthesis of intricate 3D shapes from textual prompts. Yet, this system yields lower-resolution point clouds, often missing detailed shapes and textures. In this work, we develop a brand new 3D diffusion model for texture image synthesis, which allows high-fidelity and 3D-consistent texture generation.

3. Preliminaries

3.1. Denoising diffusion model.

Diffusion model [45, 17] is a kind of likelihood-based generative model that has gained significant attention recently. It learns the data distribution $q(x_0)$ by progressive denoising from a prior Gaussian distribution. Given a sample from the data distribution $x_0 \sim q(x_0)$, a fixed forward process $q(\mathbf{x}_{1:T} | \mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t | \mathbf{x}_{t-1})$ is used to perturb the data with Gaussian kernels $q(\mathbf{x}_t | \mathbf{x}_{t-1}) := \mathcal{N}(\sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I})$, producing increasingly noisy latent variables $\{x_1, x_2, \dots, x_T\}$. Then, a parameterized Markov process $p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$ with transition kernel $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) := \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})$ is optimized through maximizing a variational lower bound of log data likelihood, which essentially targets to match the joint

distribution $q(\mathbf{x}_{0:T})$:

$$\mathbb{E}_{q(\mathbf{x}_0)} [\log p_\theta(\mathbf{x}_0)] \geq \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} \right]. \quad (1)$$

After training, novel samples can then be generated via iterative sampling from $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$ following:

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sqrt{\beta_t} \mathbf{z}, \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}).$$

Diffusion models have shown impressive capabilities in generating high-quality and diverse content. This paper proposes a texture generation framework based on diffusion model.

3.2. UV mapping and challenges.

UV mapping is a method of surface parameterization that translates a 3D surface into a 2D image, effectively creating a 2D coordinate system known as a UV map for a polygonal mesh S . This is achieved by explicitly assigning UV coordinates to each vertex of the mesh. Further, an arbitrary surface coordinate on the mesh can be mapped to its 2D coordinate through barycentric interpolation:

$$(u, v) = f(p) \quad f : S \rightarrow \Omega, \quad (2)$$

where f represents the UV mapping process.

As illustrated in Figure 3 “UV warping”, we can apply textures to the mesh surface by generating a high-resolution texture image in the UV space. Besides texture, we can also create this kind of 2D map for surface normals and point coordinates. Our objective is to generate a high-quality 2D UV texture image of the given mesh. However, the mapping process requires cutting the continuous texture on the 3D shape into a series of individual patches in the 2D UV plane, as depicted in Figure 3. This fragmentation makes it challenging for the generative model to directly learn the 3D adjacency relationships of the patches within the 2D texture UV map. Consequently, this can lead to discontinuity and inconsistency issues when the generated texture map is applied back to the 3D mesh surface. As shown in Figure 2 “2D Diffusion”, the diffusion model generates inconsistent textures and suffers from discontinuity issues.

4. Method

The challenge mentioned in Section 3.2 motivates us to propose a coarse-to-fine framework for texture image synthesis, namely Point-UV diffusion, illustrated in Figure 4. To start, we design a 3D point diffusion model to colorize a set of sampled points on the mesh surface, as shown in Figure 4 (Top). This stage leverages the 3D topology for predicting low-frequency colors on the mesh, without being affected by the discontinuity of the UV map. Based on the color components generated by the coarse stage, we then

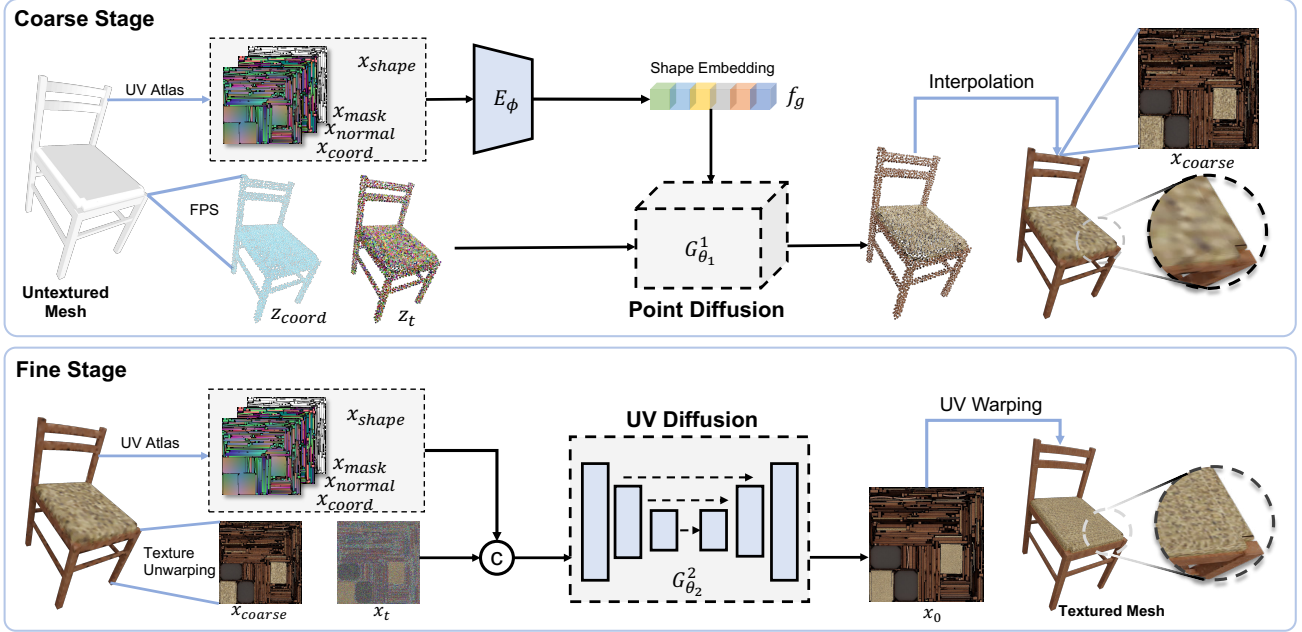


Figure 4. **The overview of our Point-UV-diffusion framework.** (Top) The coarse stage first samples a point cloud on the mesh surface, and then predicts the color for each point using a 3D diffusion model conditioned on shape features including surface normal, mask, and coordinates. Then the points are mapped to the 2D UV map and the remaining uncolored points are filled up by tri-linear interpolation based on the 3D coordinates. (Bottom) The fine stage predicts high-quality textures with a 2D diffusion model conditioned on the shape attributes and the coarse texture image.

establish a 2D diffusion model in the UV space. This enhances the fidelity of the generated texture, as depicted in Figure 4 (Bottom).

4.1. Coarse Stage: Point Diffusion

In the coarse stage, we begin by executing farthest point sampling (FPS) on the mesh surface, deriving a point set consisting of K points. These points are defined by their coordinates \mathbf{z}_{coord} and colors \mathbf{z}_0 . During training, a forward diffusion process degrades the clean colors \mathbf{z}_0 , transforming them into a noisy state \mathbf{z}_t . The noise level is dictated by the time step t , where $t \in \{0, 1, \dots, T\}$.

Our network is trained to reverse this diffusion process, aiming to denoise \mathbf{z}_t back to its original clean colors. This denoising network is informed by three pre-computed maps: a coordinate map \mathbf{x}_{coord} , a normal map \mathbf{x}_{normal} , and a mask map \mathbf{x}_{mask} (details in Section 3.2). Unless stated otherwise, these conditions concatenate along the channel dimension, culminating in what we term the shape map \mathbf{x}_{shape} :

$$\mathbf{x}_{shape} = ([\mathbf{x}_{normal}, \mathbf{x}_{mask}, \mathbf{x}_{coord}]). \quad (3)$$

Our network architecture is constructed upon PVCNN [26], drawing similarities to point-voxel diffusion [54]. However, we introduce slight modifications to amplify the integration of global shape information, contrasting with the approach in [54] which primarily relies on point coordinates. Ini-

tially, we employ a lightweight shape encoder E_ϕ to extract a global shape embedding f_g from \mathbf{x}_{shape} . This embedding is subsequently fed into the 3D network $G_{\theta_1}^1$ along with \mathbf{z}_t , \mathbf{z}_{coord} , and t to predict the color $\hat{\mathbf{z}}_0$:

$$\begin{aligned} f_g &= E_\phi(\mathbf{x}_{shape}), \\ \hat{\mathbf{z}}_0 &= G_{\theta_1}^1([\mathbf{z}_{coord}, \mathbf{z}_t, f_g, t]). \end{aligned} \quad (4)$$

Style guidance. We observe that the synthesized colors are largely influenced by the predominant colors within the dataset (for instance, textures in the ShapeNet “chair” category are typically white, pure magenta, or wood-colored), leading to a lack of diversity in the outputs. To address this bias and promote color diversity, we introduce a style guidance mechanism. This is achieved by flattening each \mathbf{z}_0 into a unidimensional vector, followed by employing PCA to extract the principal component coefficients, thus reducing the dimensionality of \mathbf{z}_0 . Subsequently, K-means clustering is utilized to assign a style label to each \mathbf{z}_0 . As depicted in Figure 5, shapes within a particular cluster exhibit similar color styles. During training, we provide the network with an additional style label z_{style} as a condition, which is referred to as style guidance:

$$\hat{\mathbf{z}}_0 = G_{\theta_1}^1([\mathbf{z}_{coord}, \mathbf{z}_t, f_g, t, z_{style}]). \quad (5)$$

In this way, we can guide the network to predict the desired color during inference by providing a certain style condi-

tion, thereby alleviating the biased color issue.

Coarse texture image. With the colored point clouds, we then project them onto the 2D UV space based on the pre-calculated UV mapping. Following this, we perform KNN interpolation based on the 3D coordinates to assign colors to the remaining uncolored pixels, thereby generating a *coarse texture image* $\mathbf{x}_{\text{coarse}}$ for the given mesh. This coarse texture image offers a coherent base color initialization and functions as a conditional element to guide the high-fidelity texture generation in the second stage, helping to avoid discontinuity caused by UV mapping.

4.2. Fine Stage: UV Diffusion

To further refine the coarse texture image, we design a fine stage using 2D diffusion in the UV space, as depicted in Figure 4 (Bottom). In addition to the conditions used in the coarse stage, we incorporate the coarse texture image $\mathbf{x}_{\text{coarse}}$ as an additional condition. We apply a 2D U-Net $G_{\theta_2}^2$ combined with self-attention modules to learn the high-quality texture image $\hat{\mathbf{x}}_0$, from a noisy texture image \mathbf{x}_t :

$$\hat{\mathbf{x}}_0 = G_{\theta_2}^2([\mathbf{x}_{\text{shape}}, \mathbf{x}_t, \mathbf{x}_{\text{coarse}}, t]). \quad (6)$$

Hybrid condition. During training, we employ FPS on the ground-truth texture image, \mathbf{x}_0 , followed by interpolation to simulate the coarse texture image, $\mathbf{x}_{\text{coarse}}$. However, in joint cascaded testing involving both stages, we note that the output quality of $\hat{\mathbf{x}}_{\text{coarse}}$ from the first stage doesn’t always align flawlessly with the quality of $\mathbf{x}_{\text{coarse}}$. Such mismatches can influence the performance of the subsequent stage. To address this, we introduce a hybrid conditioning method aimed at narrowing this discrepancy. Firstly, we create a smooth texture image $\mathbf{x}_{\text{smooth}}$ (see Figure 6), inspired by the blur augmentation described in [18] for cascaded diffusion models. In particular, we segment the mask map into multiple discrete regions using four-connectivity detection, then perform average color pooling within each region based on its connectivity. After this procedure, $\mathbf{x}_{\text{smooth}}$ maintains merely the regional color, ensuring more consistent alignment across both training and testing phases. Then, we combine $\mathbf{x}_{\text{coarse}}$ with $\mathbf{x}_{\text{smooth}}$, using a certain probability p_{hybrid} during training. Thus, the network is forced to be capable of generating textures even when adopting a weaker condition $\mathbf{x}_{\text{smooth}}$ in the fine stage during inference. Considering that the diffusion model first generates low-frequency information and then higher one [8], we also explore a condition-truncated sampling, as detailed discussed in Section 5.4, where we condition both maps for generation during the initial time steps and then exclusively utilize the smooth map for the remainder of the generation.

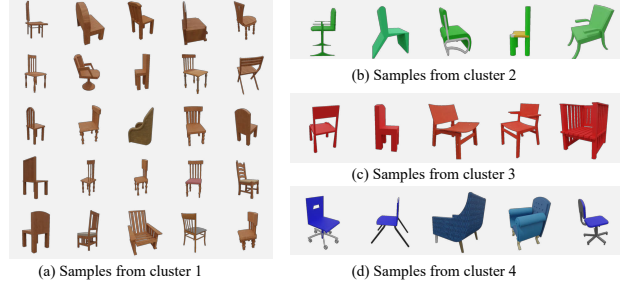


Figure 5. **Illustration of samples across various clusters.** Shapes within a particular cluster exhibit analogous color styles. However, there exists an imbalance in the quantity of shapes among different clusters, leading to challenges in unbiased synthesis.

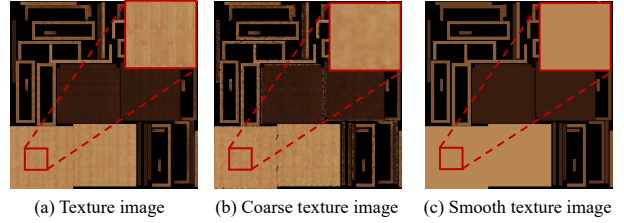


Figure 6. **Varieties of texture images.** (a) Original texture image enriched with high-frequency details, (b) Coarse texture image, and (c) Smooth texture image.

5. Experiments

5.1. Datasets and Implementation Details

We conduct experiments across four categories of the ShapeNet dataset [4], namely, chair, table, car, and bench. Before training, we use an open-source UV-Atlas tool [51] to generate the UV map and pre-process the dataset to obtain the shape maps and ground-truth texture images. We sample $K = 4096$ points in our coarse stage and synthesize a texture image in the UV space with a resolution of 512×512 for the fine stage. For training the diffusion models, akin to [1], we predict the clean signals. This approach provides more stable training than predicting the noise component as recommended by [17]. We employ the cosine noise scheduling [32] ranging from 0.0001 to 0.02 over 1,024 time steps for both stages. In the fine stage, we leverage the noise scaling strategy from [7] and also incorporate a rendering loss. Specifically, we randomly select four views and render the mesh using the predicted UV map and the ground-truth UV map to produce 1024×1024 images. Subsequently, we crop 224×224 patches from these images and compute the corresponding L_1 loss. For the hybrid condition in the fine stage, we use $p_{\text{hybrid}} = 0.3$ and sweep over condition-truncated time t_c (see Figure 12).

Compared methods. We compare our method with state-of-the-art approaches, including Texturify [43], Texture



Figure 7. **Our texture synthesis results in different categories.** The results on chairs, tables, cars, and benches demonstrate that our approach can generate natural, vivid, and diverse textures, even though the given 3D shapes are challenging, such as cutouts on the bench.

Fields [34], and PVD [54], in the context of unconditional texture generation. PVD was originally designed for point cloud generation instead of texture generation, so it cannot be directly compared with our approach. To facilitate comparison, we modify this framework to learn RGB values from the sampled point cloud. This extension, referred to as “PVD-Tex”, serves as a baseline for directly learning texture in the point space using the diffusion model.

5.2. Unconditional Texture Generation

Gallery. First, we showcase our texture generation results for each category without additional conditions. The results in Figure 7 and Figure 1 demonstrate the remarkable performance of our method, which is able to generate appealing textures with fine details and preserve the geometric structures. Note that our method is compatible with meshes of diverse topology and intricate geometric details.

Qualitative comparisons. As depicted in Figure 8, our approach excels at generating high-quality textures while maintaining the geometric intricacies of the input mesh. Firstly, the results of Texture Fields [34] appear deficient in high-frequency details. Additionally, while PVD-Tex [54] is able to produce spatially varying colors to some extent (e.g., the car in Figure 8), it falls short in synthesizing intricate high-frequency details. Lastly, even though Texturify [43] demonstrates an ability to generate finer textures, it compromises on preserving slender structures (*i.e.* chair).

Quantitative comparisons. To quantitatively compare with existing works, we follow [43] and assess the genera-

tion quality using Frechet Inception Distance (FID)[16] and Kernel Inception Distance (KID)[2], metrics that are widely used for evaluating image generation models. To this end, we render 512×512 images from each generated textured mesh and ground-truth textured mesh using four distinct camera views. Table 1 presents the quantitative comparisons with current approaches, revealing that our method surpasses existing works.

5.3. Conditional Texture Generation

In addition, we demonstrate the capability of our framework to synthesize textures conditioned on either text prompts or a single-view image. We conduct our experiments on the chair and table categories. For the text condition, we utilize text-shape pairs as provided in [6] (with additional corrections for text accuracy). For the image condition, we randomly render a view from the ground-truth mesh. To infuse the network with condition-specific information, we use the pre-trained vision-language model CLIP [38] to extract the corresponding embedding from either the image or the text. This embedding is then fed into a simple MLP to incorporate the information into the diffusion model as a condition for both training and inference. As depicted in Figure 9, our method succeeds in generating textures that align well with the given text descriptions or images. We also compare our approach with Text2Mesh [29], a test-time optimization method. Text2Mesh takes around 10 minutes per instance, while ours only requires 30 seconds. Importantly, to adapt to our task which requires preserving the mesh geometry, we freeze the geometry deformation branch of Text2Mesh to generate only colors. As shown in Figure 10, Text2Mesh

	Chair		Car		Table		Bench	
Methods	FID ↓	KID ↓	FID ↓	KID ↓	FID ↓	KID ↓	FID ↓	KID ↓
Texture Fields [34]	24.24	1.07	156.38	13.64	68.96	4.20	62.71	2.96
Texturify [43]	27.80	1.32	73.16	4.71	-	-	-	-
PVD-Tex [54]	15.52	0.62	59.47	3.74	16.12	0.55	28.94	0.39
Ours	9.88	0.22	26.89	0.68	9.63	0.15	23.09	0.15

Table 1. **Quantitative comparisons with existing works.** Ours outperforms other approaches on both FID [16] and KID ($\times 10^2$) [2].

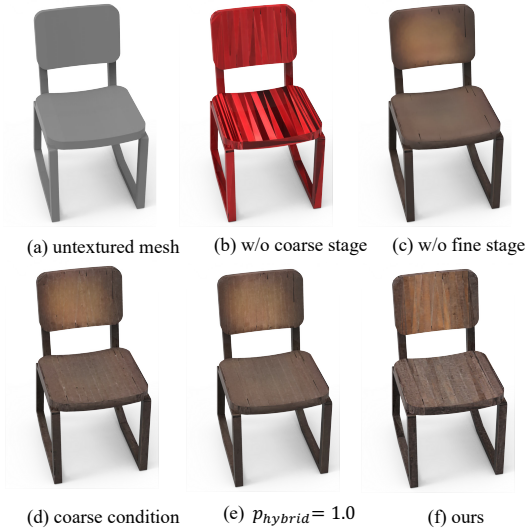


Figure 11. **Ablation studies.** Without our two-stage generation pipeline or hybrid condition designs, the generated results are inconsistent with the 3D shape or lack high-frequency details.

	FID ↓	KID ↓
w/o fine stage	17.88	0.76
w/o coarse stage	15.11	0.49
coarse condition	14.93	0.56
$p_{\text{hybrid}} = 1.0$	15.25	0.59
Ours	9.88	0.22

Table 2. **Ablation studies.** This table shows the effectiveness of each component in our proposed method.

qualitatively and quantitatively, as presented in Table 2 and Figure 11 “ours”. Furthermore, we also sweep over the effect of condition-truncated time t_c for inference. As Figure 12 shows, $t_c = 0.4$ strikes a sweet point. The model utilizes information from both conditions maps to better generate low-frequency components during the first 40% of the sampling timesteps. After that, the coarse map $\mathbf{x}_{\text{coarse}}$ is dropped, and the model further focuses on fine-grained detail generation without relying on the coarse map.

Style guidance. The style guidance is aimed at addressing the issue of insufficient diversity due to color distribution bias in the dataset. As a result, we do not utilize FID or KID for evaluation since they evaluate the distribution sim-

Methods	Preference↑	LPIPS↑
Ours	49.8%	0.083
Texturify [43]	15.9%	0.086
PVD-Tex [54]	29.2%	0.029
Texture Fields [34]	5.1%	0.005

Table 3. **LPIPS and user study.** This table shows the average LPIPS and preference via user study in the chair category.

ilarity between generated results and ground truth. However, the ground-truth distribution shows a strong bias towards particular colors, making these metrics inappropriate for assessing generative texture diversity. In contrast, we adopt LPIPS [53] to measure the pairwise similarity among five textures generated by our method given the same input mesh, where a larger diversity in textures will lead to a higher LPIPS value. In Figure 13, we report the quantitative results for 500 shapes. Our approach achieves a higher LPIPS in most of the evaluated cases, indicating better diversity. Besides, as shown in Figure 14, we present the results of generating three textures randomly for the shapes without and with style guidance. In the latter case, we uniformly sample three style labels as style inputs. It is evident that without style guidance, the generated textures are nearly identical, and the color styles of different shapes tend to be similar. With the introduction of style guidance, however, the diversity of colors has significantly increased. We also conduct a comparison with other methods in terms of diversity, as well as a quality assessment through user studies, as shown in Table 3.

6. Discussion, Limitations, and Conclusions

This paper presents Point-UV diffusion, a brand-new framework that employs a coarse-to-fine pipeline to generate textures for 3D meshes. We begin with a 3D diffusion model to synthesize low-frequency texture components from point clouds, which maintain 3D consistency. We then refine the textures using a 2D UV-space diffusion model. Our method is compatible with meshes with arbitrary topology and can faithfully preserve the geometry structure. We further demonstrate the flexibility of our framework by extending it to conditional generative models.

Despite its merits, our method has inherent limitations. Similar to other methodologies relying on 3D data for train-

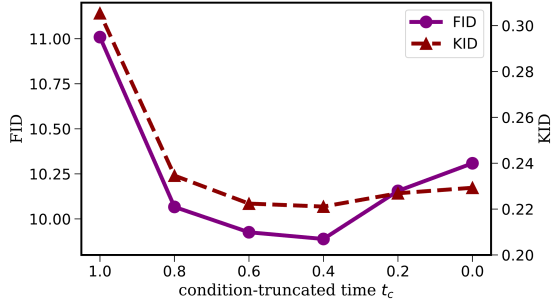


Figure 12. **Examination of condition-truncated time.** The horizontal axis denotes the percentage of time points when the coarse texture image is conditioned during inference. The left vertical axis indicates the FID value, while the right vertical axis shows the KID value ($\times 10^2$).

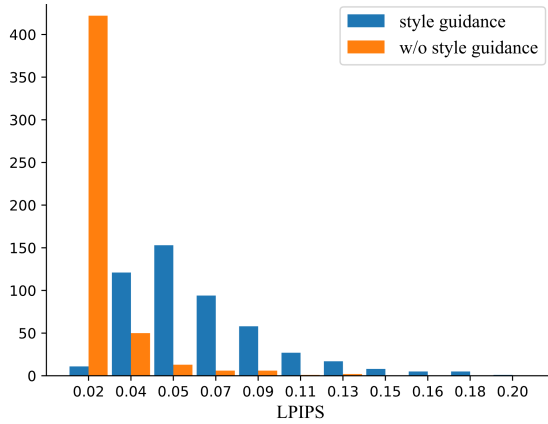


Figure 13. **Diversity measurement.** Utilizing style guidance leads to larger LPIPS scores, indicating enhanced generation diversity.

ing, our technique is upper-bounded by the scope and diversity of current 3D datasets. This restriction poses challenges in generating textures that parallel the depth of effects seen in 2D image synthesis. Moreover, our method’s efficacy relies on the quality of UV mapping. Our approach faces difficulties in rendering high-quality results for meshes where the UV mapping produces excessive fragmented cuts, resulting in fragmented artifacts. This phenomenon is commonly observed in the car category, as shown in the Figure 15. We believe the emergence of larger and more diverse 3D datasets would be helpful for generating superior-quality textures. Further advancements in UV parameterization would also be beneficial in augmenting our method’s consistency.

7. Acknowledgements

This work has been supported by Hong Kong Research Grant Council - Early Career Scheme (Grant No. 27209621), General Research Fund Scheme (Grant No. 17202422), and RGC matching fund scheme (RMGS). Part

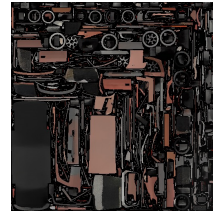
(a) w/o style guidance



(b) w/ style guidance



Figure 14. **Efficacy of style guidance.** The model without style guidance produces consistent colors across different random seeds (rows 1-2). In contrast, with style guidance, diverse outcomes emerge (rows 3-4).



(a) Generated Texture Image



(b) Textured Mesh

Figure 15. **Failure case.** Our approach still struggles to generate seamless results when there are too many fragmented cuts of the UV map.

of the described research work is conducted in the JC STEM Lab of Robotics for Soft Materials funded by The Hong Kong Jockey Club Charities Trust.

References

- [1] Titas Anciukevičius, Zexiang Xu, Matthew Fisher, Paul Henderson, Hakan Bilen, Niloy J Mitra, and Paul Guerrero. Rendering: Image diffusion for 3d reconstruction, inpainting and generation. *arXiv preprint arXiv:2211.09869*, 2022. 3, 5
- [2] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. *arXiv preprint arXiv:1801.01401*, 2018. 6, 8
- [3] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5799–5809, 2021. 2
- [4] Angel X. Chang, Thomas Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], 2015. 5
- [5] Bindita Chaudhuri, Nikolaos Sarafianos, Linda Shapiro, and Tony Tung. Semi-supervised synthesis of high-resolution editable textures for 3d humans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7991–8000, 2021. 1
- [6] Kevin Chen, Christopher B Choy, Manolis Savva, Angel X Chang, Thomas Funkhouser, and Silvio Savarese. Text2shape: Generating shapes from natural language by learning joint embeddings. In *Computer Vision—ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*, pages 100–116. Springer, 2019. 1, 2, 6
- [7] Ting Chen. On the importance of noise scheduling for diffusion models. *arXiv preprint arXiv:2301.10972*, 2023. 5
- [8] Jooyoung Choi, Jungbeom Lee, Chaehun Shin, Sungwon Kim, Hyunwoo Kim, and Sungroh Yoon. Perception prioritized training of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11472–11481, 2022. 5
- [9] Yu Deng, Jialong Yang, Jianfeng Xiang, and Xin Tong. Gram: Generative radiance manifolds for 3d-aware image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10673–10683, 2022. 2
- [10] Prafulla Dhariwal and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis, 2021. 15
- [11] Jun Gao, Wenzheng Chen, Tommy Xiang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. *Advances In Neural Information Processing Systems*, 33:9936–9947, 2020. 1
- [12] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. In *Advances In Neural Information Processing Systems*, 2022. 1, 2
- [13] Lin Gao, Tong Wu, Yu-Jie Yuan, Ming-Xian Lin, Yu-Kun Lai, and Hao Zhang. Tm-net: Deep generative networks for textured meshes. *ACM Transactions on Graphics (TOG)*, 40(6):1–15, 2021. 2
- [14] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017. 1
- [15] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Learning a neural 3d texture space from 2d exemplars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8356–8364, 2020. 2
- [16] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. *NIPS*, 2017. 6, 8
- [17] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, 2020. 2, 3, 5, 12
- [18] Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *J. Mach. Learn. Res.*, 23(47):1–33, 2022. 5
- [19] Jingyu Hu, Ka-Hei Hui, Zhengzhe Liu, Ruihui Li, and Chi-Wing Fu. Neural wavelet-domain diffusion for 3d shape generation, inversion, and manipulation. *arXiv preprint arXiv:2302.00190*, 2023. 3
- [20] Ka-Hei Hui, Ruihui Li, Jingyu Hu, and Chi-Wing Fu. Neural wavelet-domain diffusion for 3d shape generation. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022. 1, 3
- [21] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 1
- [22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 12
- [23] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6):1–14, 2020. 13
- [24] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 300–309, 2023. 3
- [25] Zhengzhe Liu, Peng Dai, Ruihui Li, Xiaojuan Qi, and Chi-Wing Fu. Iss: Image as setting stone for text-guided 3d shape generation. *International Conference on Learning Representations*, 2023. 2
- [26] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. *Advances in Neural Information Processing Systems*, 32, 2019. 4
- [27] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 15
- [28] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 15

- [29] Oscar Michel, Roi Bar-On, Richard Liu, Sagie Benaim, and Rana Hanocka. Text2mesh: Text-driven neural stylization for meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13492–13502, 2022. 3, 6
- [30] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 3
- [31] Nasir Mohammad Khalid, Tianhao Xie, Eugene Belilovsky, and Tiberiu Popa. Clip-mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–8, 2022. 2
- [32] Alex Nichol and Prafulla Dhariwal. Improved Denoising Diffusion Probabilistic Models, 2021. 5, 12
- [33] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022. 3
- [34] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4531–4540, 2019. 1, 2, 6, 7, 8
- [35] Dario Pavllo, Graham Spinks, Thomas Hofmann, Marie-Francine Moens, and Aurelien Lucchi. Convolutional generation of textured 3d meshes. *Advances in Neural Information Processing Systems*, 33:870–882, 2020. 2
- [36] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *International Conference on Learning Representations*, 2023. 2, 3
- [37] Tiziano Portenier, Siavash Arjomand Bigdeli, and Orcun Goksel. Gramgan: Deep 3d texture synthesis from 2d exemplars. *Advances in Neural Information Processing Systems*, 33:6994–7004, 2020. 2
- [38] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 3, 6
- [39] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*, 2022. 1
- [40] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 1, 3
- [41] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *NeurIPS*, 2022. 1
- [42] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *Advances in Neural Information Processing Systems*, 33:20154–20166, 2020. 2
- [43] Yawar Siddiqui, Justus Thies, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Texturify: Generating textures on 3d shape surfaces. *arXiv preprint arXiv:2204.02411*, 2022. 2, 5, 6, 7, 8
- [44] Ivan Skorokhodov, Sergey Tulyakov, Yiqun Wang, and Peter Wonka. Epigraf: Rethinking training of 3d gans. *Advances in Neural Information Processing Systems*, 35:24487–24501, 2022. 2
- [45] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015. 3
- [46] Yongbin Sun, Ziwei Liu, Yue Wang, and Sanjay E Sarma. Im2avatar: Colorful 3d reconstruction from a single image. *arXiv preprint arXiv:1804.06375*, 2018. 2
- [47] Shubham Tulsiani, Nilesch Kulkarni, and Abhinav Gupta. Implicit mesh reconstruction from unannotated image collections. *arXiv preprint arXiv:2007.08504*, 2020. 2
- [48] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2626–2634, 2017. 2
- [49] Hao Wang, Guosheng Lin, Steven Hoi, and Chunyan Miao. Cycle-consistent inverse GAN for text-to-image synthesis. *ACM MM*, 2021. 1
- [50] Kangxue Yin, Jun Gao, Maria Shugrina, Sameh Khamis, and Sanja Fidler. 3dstylenet: Creating 3d shapes with geometric and texture style variations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12456–12465, 2021. 1
- [51] Jonathan Young. xatlas, 2018. <https://github.com/jpcy/xatlas>. 5, 14
- [52] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. Lion: Latent point diffusion models for 3d shape generation. *arXiv preprint arXiv:2210.06978*, 2022. 3
- [53] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 8
- [54] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5826–5835, 2021. 3, 4, 6, 7, 8, 15

Appendix

A. Denoising Diffusion Model

Forward process. Denoising diffusion probabilistic model (DDPM) [32] learns the data distribution by introducing a series of latent variables and matching the joint distribution. The model starts with a sample from the data distribution, denoted as $x_0 \sim q(\mathbf{x}_0)$, and a forward process $q(\mathbf{x}_{1:T} | \mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t | \mathbf{x}_{t-1})$ progressively perturbs the data with Gaussian kernels $q(\mathbf{x}_t | \mathbf{x}_{t-1}) := \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$, producing increasingly noisy latent variables $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$. Notably, x_t can be directly sampled from x_0 thanks to the closed form:

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}), \quad (7)$$

where $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^t \alpha_s$. In general, the forward process variances β_t are fixed and increased linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$. Besides, T should be large (e.g., 1000) enough to ensure $q(\mathbf{x}_T | \mathbf{x}_0) \approx \mathcal{N}(0, \mathbf{I})$. The diffusion model aims to model the joint distribution $q(\mathbf{x}_{0:T})$, which includes a tractable sampling path for the marginal distribution $q(\mathbf{x}_0)$.

Reverse process and optimization. To learn how to reverse the forward process, the diffusion model defines a parameterized Markov chain with parameterized transition kernels:

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t), \quad (8)$$

$$p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)),$$

where $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$ are optimized. The training is performed by optimizing a variational bound of negative log likelihood:

$$\mathbb{E}_{q(\mathbf{x}_0)} [-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_{q(\mathbf{x}_{0:T})} \left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} \right] =: L. \quad (9)$$

The loss term L can be rewritten as:

$$L = \mathbb{E}_q \left[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T | \mathbf{x}_0) \| p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t))}_{L_{t-1}} - \log p_\theta(\mathbf{x}_0 | \mathbf{x}_1) \right]. \quad (10)$$

In practice, the core optimization terms are $L_{t-1}(t > 1)$ that can be analytically calculated since both two terms compared in the KL divergence are Gaussians, i.e.,:

$$q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}), \quad (11)$$

$$p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)),$$

where $\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\bar{\alpha}_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$. DDPM [17] fix $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$ during training, where σ_t^2 is set to be β_t or $\tilde{\beta}_t$. In our experiments, we set $\sigma_t^2 = \beta_t$.

Loss function. Models can be trained in the reverse process to predict the mean value of \mathbf{x}_{t-1} , i.e. $\boldsymbol{\mu}_t$. Alternatively, by modifying the parameterization, we can also train it to predict \mathbf{x}_0 or ϵ , as illustrated in [17]. The original DDPM [17] predicts ϵ . In this case, through reparameterization trick [22] and empirical simplification [17], the final training term is performed as follows:

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t, \mathbf{x}_0, \epsilon} \left[\left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t) \right\|_2^2 \right], \quad (12)$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ and t is uniformly sampled between 1 and T . However, in our experiments, we predict \mathbf{x}_0 since we find it results in more stable training. The loss function is modified as:

$$L_{\mathbf{x}_0}(\theta) := \mathbb{E}_{t, \mathbf{x}_0, \epsilon} \left[\left\| \mathbf{x}_0 - G_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t) \right\|_2^2 \right], \quad (13)$$

where G_θ is the network to be optimized.

Sampling. After training, started from an initial noise map $x_T \sim p(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$, new images can be then generated via iteratively sampling from $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$, using the following equation:

$$\mathbf{x}_{t-1} = \tilde{\boldsymbol{\mu}}_t + \sigma_t \mathbf{z}, \quad (14)$$

where $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$. In particular, we predict the x_0 , so that the sampling process can be modified as:

$$\begin{aligned} \mathbf{x}_{t-1} &= \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{\mathbf{x}}_0 + \frac{\sqrt{\bar{\alpha}_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \sigma_t \mathbf{z} \\ &= \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}G_\theta(\mathbf{x}_t, t) + \frac{\sqrt{\bar{\alpha}_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \sigma_t \mathbf{z}. \end{aligned} \quad (15)$$

B. Method

B.1. Basic Point-UV Diffusion

In this section, we introduce the basic training process of our model, which *does not include style guidance and hybrid conditioning*. Our nets are denoted by $G_{\theta_1}^1$ and $G_{\theta_2}^2$ for the coarse stage and fine stage. As we have mentioned in Section A, we train the models to predict clean signals instead of noise components.

We summarize our training procedure in Algorithm 1. In this algorithm flowchart, q represents the true texture distribution that we aim to estimate, while \mathbf{z}_0 and \mathbf{x}_0 represent

the ground-truth point cloud color and texture image, respectively. L_{coarse} and L_{fine} represent the loss functions for the two stages, which will be described in detail below. It should be noted that the training of the fine stage can be parallel to the coarse stage, as it does not depend on the training of the coarse stage.

Loss functions. For the coarse stage, we simply adopt the loss function we described in Equation 13, *i.e.*:

$$L_{\text{coarse}} := \mathbb{E}_{t, \mathbf{z}_0, \epsilon} \left[\left\| \mathbf{z}_0 - G_{\theta_1}^1(\mathbf{z}_t, t, \mathbf{x}_{\text{shape}}, \mathbf{z}_{\text{coord}}) \right\|_2^2 \right], \quad (16)$$

where $G_{\theta_1}^1$ is the coarse network and $\mathbf{z}_t = \sqrt{\bar{\alpha}_t} \mathbf{z}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$ is the noisy point cloud color.

For the fine stage, in addition to using the basic diffusion loss, *i.e.*:

$$L_{\text{basic}} := \mathbb{E}_{t, \mathbf{x}_0, \epsilon} \left[\left\| \mathbf{x}_0 - G_{\theta_2}^2(\mathbf{x}_t, t, \mathbf{x}_{\text{shape}}, \mathbf{x}_{\text{coarse}}) \right\|_2^2 \right], \quad (17)$$

where $G_{\theta_2}^2$ is the fine network and $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$ is the noisy texture image. we also utilize differentiable rendering to render the generated textured mesh and the ground-truth mesh, then calculate the L_1 rendering loss between rendered images. Specifically, during training, by using Nvdiffrast [23], we randomly select four views and render the mesh based on the predicted UV map and ground-truth UV map to obtain four 1024×1024 images. We then crop patches of size 224×224 from each of these images to obtain $\hat{\mathbf{x}}_{\text{render}} \in \mathbb{R}^{4 \times 3 \times 224 \times 224}$ and $\mathbf{x}_{\text{render}} \in \mathbb{R}^{4 \times 3 \times 224 \times 224}$. The rendering loss is thus defined as:

$$L_{\text{render}} := \mathbb{E}_{t, \mathbf{x}_0, \epsilon} [\| \mathbf{x}_{\text{render}} - \hat{\mathbf{x}}_{\text{render}} \|_1]. \quad (18)$$

The final loss function for the fine stage is:

$$L_{\text{fine}} := L_{\text{render}} + L_{\text{basic}}. \quad (19)$$

B.2. Style Guidance

The motivation of style guidance is to address the color bias in the training dataset. To this end, we use the traditional clustering technique to get a style label for each $\mathbf{z}_0^i \in \mathbb{R}^{3 \times 4096}$ (note that \mathbf{z}_0^i is defined as the color of the point cloud and not contains the coordinates).

Style label. We commence by transforming each \mathbf{z}_0^i in the training dataset into a feature vector. This transformation is realized by flattening the matrix into a one-dimensional array and normalizing its pixel values to attain zero mean and unit variance. The resultant feature vectors, represented as $\mathbf{z}_{\text{flat}}^i$, act as inputs for the subsequent principal component analysis (PCA). PCA is employed to reduce the dimensionality of the feature vectors. This process involves

Algorithm 1: Train Basic Point-UV Diffusion

```

1 do
2    $\mathbf{z}_0 \sim q(\mathbf{z} \mid \mathbf{x}_{\text{shape}}, \mathbf{z}_{\text{coord}})$ 
3    $t \sim \mathcal{U}(\{1, \dots, T\})$ 
4    $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
5    $\mathbf{z}_t = \sqrt{\bar{\alpha}_t} \mathbf{z}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$ 
6    $\theta_1 \leftarrow \theta_1 - \eta \nabla_{\theta_1} L_{\text{coarse}}(\mathbf{z}_t, \mathbf{z}_0, \mathbf{x}_{\text{shape}}, \mathbf{z}_{\text{coord}})$ 
7 until converged;
8 do
9    $(\mathbf{x}_0, \mathbf{z}_0) \sim q(\mathbf{x}, \mathbf{z} \mid \mathbf{x}_{\text{shape}}, \mathbf{z}_{\text{coord}})$ 
10  get  $\mathbf{x}_{\text{coarse}}$  by interpolating from  $\mathbf{z}_0$ 
11   $t \sim \mathcal{U}(\{1, \dots, T\})$ 
12   $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
13   $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$ 
14   $\theta_2 \leftarrow \theta_2 - \eta \nabla_{\theta_2} L_{\text{fine}}(\mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_{\text{shape}}, \mathbf{x}_{\text{coarse}})$ 
15 until converged;

```

computing the eigenvectors and eigenvalues of the covariance matrix associated with the feature vectors. By retaining only the top- n eigenvectors (where n signifies the intended dimensionality of the trimmed dataset), we can project the feature vectors into a subspace with fewer dimensions. Given the global and low-frequency nature of the color style, we fix $n = 5$ for our experiments. Once the reduced feature vectors are acquired, the K-means clustering algorithm is employed to cluster similarly colored point clouds. For our experiments, we define $k = 40$.

B.3. Hybrid Condition

Algorithm 2: Training of Fine Stage with Hybrid Condition

```

1 do
2    $(\mathbf{x}_0, \mathbf{z}_0) \sim q(\mathbf{x}, \mathbf{z} \mid \mathbf{x}_{\text{shape}}, \mathbf{z}_{\text{coord}})$ 
3   get  $\mathbf{x}_{\text{coarse}}$  by interpolating from  $\mathbf{z}_0$ 
4   get  $\mathbf{x}_{\text{smooth}}$  by smoothing  $\mathbf{x}_{\text{coarse}}$ 
5   if  $\text{rand}(0, 1) < p_{\text{hybrid}}$  then
6      $\mathbf{x}_{\text{hybrid}} = (\mathbf{x}_{\text{smooth}}, \mathbf{x}_{\text{coarse}})$ 
7   else
8      $\mathbf{x}_{\text{hybrid}} = (\mathbf{x}_{\text{smooth}}, \emptyset)$ 
9    $t \sim \mathcal{U}(\{1, \dots, T\})$ 
10   $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
11   $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$ 
12   $\theta_2 \leftarrow \theta_2 - \eta \nabla_{\theta_2} L_{\text{fine}}(\mathbf{x}_t, \mathbf{x}_0, \mathbf{x}_{\text{shape}}, \mathbf{x}_{\text{hybrid}})$ 
13 until converged;

```

Smooth map. The smooth map $\mathbf{x}_{\text{smooth}}$ is obtained by smoothing the coarse UV map $\mathbf{x}_{\text{coarse}}$ by regions. Specifi-

Algorithm 3: Sampling of Fine Stage with Condition Truncation

```

1  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
2 for  $t = T$  to 1 do
3   if  $t/T > t_c$  then
4      $\mathbf{x}_{\text{hybrid}} = (\mathbf{x}_{\text{smooth}}, \mathbf{x}_{\text{coarse}})$ 
5   else
6      $\mathbf{x}_{\text{hybrid}} = (\mathbf{x}_{\text{smooth}}, \emptyset)$ 
7    $\hat{\mathbf{x}}_0 = G_{\theta_2}^2(\mathbf{x}_t, t, \mathbf{x}_{\text{shape}}, \mathbf{x}_{\text{hybrid}})$ 
8    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
9    $\mathbf{x}_{t-1} = \frac{\sqrt{\alpha_t-1}\beta_t}{1-\alpha_t}\hat{\mathbf{x}}_0 + \frac{\sqrt{\alpha_t}(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{x}_t + \sigma_t\mathbf{z}$ 
10 return  $\hat{\mathbf{x}}_0$ 

```

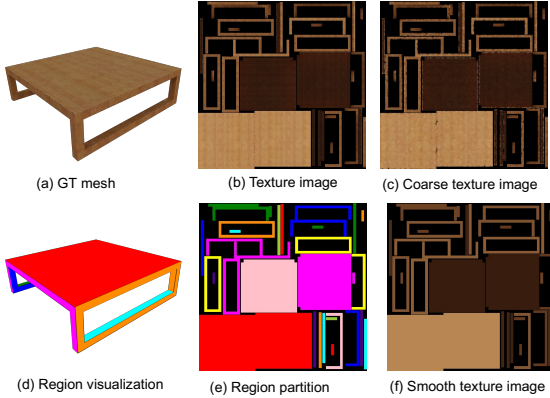


Figure 16. **Smooth map.** Different components for generating the smooth map.

cally, we partition the mask map into multiple distinct regions based on the detection of four-connectivity, and then apply average color pooling on each region based on its connectivity, as shown in Figure 16. After this operation, the smooth map retains only the region-based color style compared to the coarse map.

Training with the hybrid condition. For training the fine stage, we utilize the smooth map as the primary condition and combine the original coarse map with a certain probability p_{hybrid} and get a hybrid condition $\mathbf{x}_{\text{hybrid}}$. Thus, the network is forced to be capable of generating textures even when adopting a weaker condition $\mathbf{x}_{\text{smooth}}$ in the fine stage during inference. Formally, apart from the shape conditions $\mathbf{x}_{\text{shape}}$, the condition here consists of six channels, with the first three channels being the smooth map and the latter three channels set to zero values, which are randomly replaced by the coarse map with a probability of p_{hybrid} . The training with the hybrid condition is summarized in Algorithm 2, wherein we set the $p_{\text{hybrid}} = 0.3$ empirically.

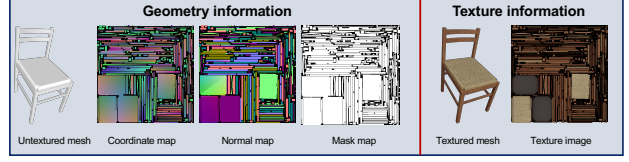


Figure 17. **Different information encoded in UV space.**

Denoising with condition truncation. After training with the hybrid condition strategy, the network is able to generate high-frequency information under weaker conditions, making it more robust to the generated results from the coarse stage during inference. Since the diffusion model is effective in recovering low-frequency components during the initial stages of sampling, followed by generating high-frequency components in the later stages, we employ two maps to jointly guide the generation process in the early stages and discard the coarse map in the later stages. In this way, the network will not be constrained by the inferior generated coarse map and has more flexibility for producing diversified high-frequency details. The sampling procedure with the condition truncation is summarized in Algorithm 3.

Analysis. The hybrid condition and condition truncation are proposed for the purpose of generating high-frequency details. The reason why the network trained and tested directly using the coarse map cannot generate texture might be due to that the coarse stage cannot be optimized perfectly, *i.e.*, there is a domain gap between the generated coarse map and the coarse map used to train the fine stage. Specifically, the second stage is trained to generate high-frequency details using coarse maps with high-frequency cues, which cannot be provided by generated coarse maps from the first-stage model during testing, leading to texture-less outputs.

C. Implementation Details

C.1. UV Map Process

We now describe how we process the raw data from ShapeNet to obtain our data format. Firstly, we utilize xatlas [51] to automatically generate UV mapping for each mesh. This allows us to establish a correspondence between the surface points and 2D planes, denoted by function f . Subsequently, we select a resolution $H \times W$ for the 2D plane. In our case, we use a resolution of 512×512 . With this resolution, we discretize the 2D plane to obtain the coordinates of each pixel in UV space. As shown in Figure 17, by applying function f , we are able to calculate the corresponding 3D coordinates for each pixel, which results in a coordinate map $\mathbf{x}_{\text{coord}} \in \mathbb{R}^{3 \times H \times W}$. It is worth noting that due to topological reasons, some pixels do not have corresponding 3D coordinates. This occurs when there are no surface points that are mapped to these pixels. These pixels are marked as invalid and we can obtain a binary mask map,

Level	Blocks Type	Number of Blocks	Block Channel	Dropout	Self Attention
1	Encoder	1	32	0.0	False
2	Encoder	2	64	0.0	False
3	Encoder	2	128	0.0	False
4	Encoder	2	192	0.0	False
5	Encoder	4	384	0.2	True
5	Decoder	5	384	0.2	True
4	Decoder	3	192	0.0	False
3	Decoder	3	128	0.0	False
2	Decoder	3	64	0.0	False
1	Decoder	2	32	0.0	False

Table 4. Details of our U-Net network.

as shown in Figure 17, to which we refer to the mask map. Similarly, we can compute the surface normal for each point and record them in their corresponding UV coordinates and get a normal map. Note that, these information only depends on the geometry of the mesh, which we use as shape conditions in our method.

To obtain the texture image, we use Blender to render each mesh from multiple viewpoints (*i.e.*, 50 views), and then we back-project the resulting multi-view images into 3D space using the camera matrices to obtain a corresponding dense colored point cloud P_{src} . For creating the ground-truth texture image with RGB colors, we query the 3D coordinates of each pixel and use KNN (in our case, we use 3-nearest neighbors) interpolation to calculate its corresponding color from P_{src} .

C.2. Training Configurations

We implement all the experiments using PyTorch on four NVIDIA RTX 3090 GPU cards with batch size 8. The learning rate is initially set to 0.0002 and scheduled by cyclic cosine annealing [27], and models are optimized by AdamW [28] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For the coarse stage, we train 500 epochs for bench category and 400 epochs for other categories. For the fine stage, we train 150 epochs for the Chair and Car categories and 250 epochs for other categories. The ablation studies are conducted in the same settings. We use an EMA rate of 0.9995 for all experiments. For training PVD-Tex, we sample a point cloud with 512×512 points on the surface, where each point corresponds to a pixel of the UV map with resolution 512×512 , and train a diffusion network to learn the RGB value for each point using the same network architecture and training scheme as in [54]. Then, we map the corresponding point colors onto the UV map, such that we can derive the textured mesh with this UV map.

C.3. Network Architecture

Coarse stage. The configurations of the network in our coarse stage are based on those in PVD [54], except for some modifications. In particular, PVD has two types of

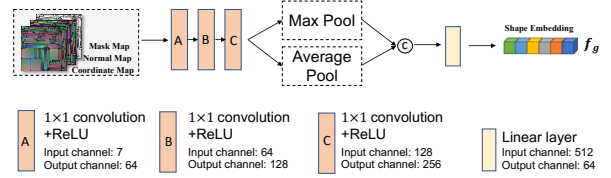


Figure 18. Details of our shape encoder.

inputs, *i.e.* noisy point cloud coordinates and time step. In contrast, we have five types of inputs, noisy point cloud colors $\mathbf{z}_t \in \mathbb{R}^{3 \times 4096}$, clean point cloud coordinates $\mathbf{z}_{coord} \in \mathbb{R}^{6 \times 4096}$ (attached with its normal), shape map $\mathbf{x}_{shape} \in \mathbb{R}^{7 \times 512 \times 512}$, style label z_{style} and time step t . To align with PVD, we first use a shape encoder E_ϕ to extract a global feature $f_g \in \mathbb{R}^{64 \times 1}$ from \mathbf{x}_{shape} . Then, we expand f_g in the last dimension to align with the point number of \mathbf{z}_t , *i.e.* we get $f_g^{expand} \in \mathbb{R}^{64 \times 4096}$. Next, we concatenate $f_g^{expand}, \mathbf{z}_t, \mathbf{z}_{coord}$ through the channel dimension, thereby aligning the point cloud input in PVD. To handle z_{style} , note that the network uses *nn.embedding* and linear layer to extract a time embedding from timestep t . Therefore, we perform the same operations on z_{style} and add its extracted embedding to the time embedding before feeding it into the network. The architecture of E_ϕ is shown in Figure 18.

Fine stage. We build our fine stage on a 2D U-Net structure based on guided diffusion [10]. In our case, we concatenate the coarse map, smooth map, shape map, and the noisy UV texture image along the channel dimension to obtain the input tensor $\mathbf{x}_{input} \in \mathbb{R}^{16 \times 512 \times 512}$. We utilize five levels for the U-Net architecture in both the encoder and decoder, incorporating dropout and self-attention at the lowest-resolution level. Each level’s basic block employs the residual block built from [10], with the number of blocks and convolutional channel numbers specified in Table 4.