Lafite: A Generative Latent Field for 3D Native Texturing

Supplementary Material

A. Implementation details

A.1. Network Architectures

Variational Encoder. Our VAE, designed to compress 3D surface appearance into a sparse latent field, comprises four key components with a total of 188.9M parameters:

- **Local point cloud encoder $\mathcal{E}_{\mathbf{P}}$**: A lightweight point cloud encoder with 1.8M parameters. The 9-channel point features (position, normal, color) are mapped into 256-channel hidden features, and an intra-voxel self-attention is applied to points within the same voxel with varlen flash attention [6]. Then, these per-point features are aggregated into their located voxel by a cross-attention between the points and the latent embedding v of their located voxels, where v is a learnable embedding shared by all voxels initially. Then the obtained color latent of the voxel $\tilde{v}$ is processed by the color latent encoder $\mathcal{E}_{\mathrm{S}}$.
- **Color latent encoder $\mathcal{E}_{\mathbf{S}}$**: A color latent encoder with 85.8M parameters. The output 1024-channel $\tilde{v}$ features are then processed by a structured latent encoder [21] with 12 layers of 3D sparse shifted window attention blocks, and a hidden latent channels of 768, and obtain the bottleneck 32-channel latents z. The overall encoding process is represented by:

$$\mathcal{E} = \mathcal{E}_{\mathrm{S}} \circ \mathcal{E}_{\mathrm{P}} \tag{1}$$

- **Latent decoder $\mathcal{D}_{\mathbf{S}}$**: A structured latent decoder with 100.4M parameters. The 32-channel latents z are decoded into 768-channel latents z_{mid} through 12 layers of 3D sparse shifted window attention blocks. $z_{\mathbf{mid}}$ are then decoded into color field latents z_c through 2 upsampling blocks, which finally results in a $4\times$ denser grid ($64^3 \rightarrow 256^3$, $128^3 \rightarrow 512^3$).
- **Color field decoder $\mathcal{D}_{\mathbf{C}}$**: A lightweight MLP decoder with 0.8M parameters, shared by all color field latents. We directly concatenate the coordinate of the query point and the z_c of their corresponding voxel, and pass the concatenated vector into a 4-layer MLP and output a 3-channel RGB color. The overall decoding process is represented by:

$$\mathcal{D} = \mathcal{D}_{\mathrm{C}} \circ \mathcal{D}_{\mathrm{S}} \tag{2}$$

Rectified Flow Generator. The color latent rectified flow transformer contains 1.4B parameters, with 30 sparse grid attention blocks. It takes a 64-channel structure latent as input, which is the concatenation of the geometry latent (or albedo color latent if trained for PBR material generation) and a noisy latent with the same shape as input for denoising, and outputs a 32-channel denoised latent, which will be re-concatenated with the geometry latent for the next denoising step. The input latent for attention blocks is first patchified and downsampled ($128^3 \rightarrow 64^3$) for more efficient attention processing.

We apply two image encoders, DINOv2 [16] and SigLIP2 [20], to extract the image features conditioning on the generator, as DINO failed to faithfully capture all fine-grained details [18] while causing color shifting.

A.2. Training Details

For the generative model, we perform two-stage training: In the first stage, we train a 64-resolution model using latent representations generated by a 64-resolution VAE as supervision. In the second stage, we fine-tune the model trained in the first stage. We continue using the VAE trained at 64 resolution, but adjust the input to 128 resolution and double the frequency of positional encoding to obtain 128-resolution color latent as supervision.

A.3. Inference and Application

We employ a first-order Euler solver for rectified flow sampling. During sampling, the classifier-free guidance [9] scale for the albedo generator and the MR generator is 5.

Inpainting. Following Repaint [14], a mask m and an initial latent z^* is required when performing inpainting, where in each step:

$$z_{t-1} = m \odot \mathcal{G}(z_t, t) + (1 - m) \odot \text{add\_noise}(z^*, t) \tag{3}$$

where $\mathcal{G}$ is the rectified flow generator that predicts the latent z_{t-1} at the next timestep by current latent z_t, $\text{add\_noise}(z^*, t)$ indicate adding noise to the original latent z^* so that during denoising, they will all be in the same noise level.

Refinement. Suppose we are given an object with a coarse texture, where surface colors can be queried using a color function $\mathcal{C}(\cdot)$. Our goal is to refine this texture through two steps: resampling and inpainting. For a selected region Ω to be refined, we first compute its axis-aligned bounding box and normalize it while preserving its aspect ratio. The mesh faces within this bounding box are then uniformly rescaled, producing an enlarged local surface patch. We sample points on these enlarged triangles to obtain higher-density geometric and color observations. The sampled colored points are encoded into a color latent, and we apply a mask to retain only the valid interior region while preserving partial contextual color around it. Conditioned on this

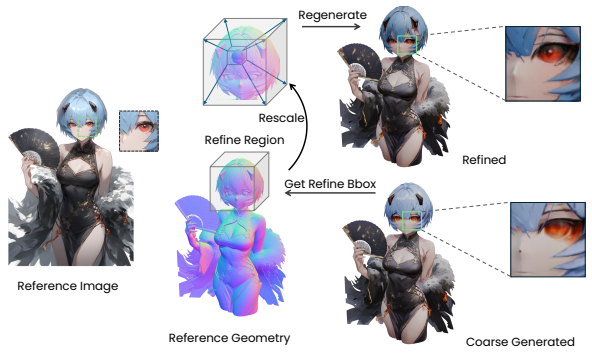

Figure 1. **Refinement process.** We select a region for refinement on a mesh that already has a coarse texture, rescale that region, and then regenerate the texture for that area.

masked representation and a higher-resolution partial image corresponding to Ω, the generator performs inpainting, yielding a refined and more detailed texture. Fig. 1 shows an example of how we refine a partial region of a mesh.

B. Data Pre-processing

B.1. Rendering Settings

We use Blender's default color management to render input images. These images serve as the condition for both the base color generator and the metallic/roughness generator. For the lighting used during rendering, we strive to ensure that the base colors and tonal rendering of objects after lighting match the rendered images.

B.2. Materials Processing

Transparency. Common transparent materials include glass, acrylic sheets, liquids, and other refractive substances. In practice, however, for rendering or storage efficiency, transparency is also used to approximate geometrically complex structures like leaves on a tree. We treat these two cases differently.

For physically transparent materials, rendering involves refractive effects governed by Fresnel terms, where rays may traverse multiple layers of geometry before reaching the sensor. As a result, pixels at the same screen position may accumulate contributions from different surfaces along the viewing direction. To handle such materials in the rendering, we reinterpret these surfaces as opaque and perform normal sampling directly on their surface geometry, avoiding unintended multi-layer color mixing.

In contrast, many assets use transparency purely as a representational shortcut: regions where the texture's alpha channel is zero are intended to denote missing geometry, even though triangles still exist in those locations. For these cases, we treat fully transparent texels as geometry holes. During sampling, we discard points whose corresponding texture alpha value is zero, keep all partially or fully opaque samples, and preserve the alpha-mask semantics during rendering. This ensures that geometry meant to be visually absent remains absent, while valid surface regions are faithfully retained.

Emission. As discussed in the main text, problematic emission effects arise from two sources.

First, an object may be fully emissive—for example, when using an Emission BSDF rather than a Principled BSDF in Blender. In this case, the base color is determined entirely by the emission texture. Ambient lighting no longer influences the appearance, and attributes such as roughness and metallic become ineffective. Consequently, we exclude these samples from roughness and metallic estimation to prevent the generator from learning invalid supervision signals.

The second case is more challenging: the object is partially emissive. Emission here may serve multiple purposes. It may indicate genuinely self-luminous regions (e.g., light bulbs, LEDs, glowworms), in which case the emitted radiance contributes to illumination and is not part of the intrinsic base color. Alternatively, emission may be used stylistically in mixed PBR-NPR pipelines to enforce "unchanged" or artist-controlled colors. In the latter scenario, the emissive component contributes to the perceived color but cannot be cleanly transformed into a physically meaningful base color.

To handle such cases, we approximate how the emissive contribution blends with the underlying base color to produce the rendered appearance. Since the true illumination is unknown, recovering an exact base color is inherently ambiguous. Nonetheless, by ensuring that the predicted color preserves the hue observed in the rendered image, we maintain sufficient fidelity for training and still allow the model to learn the correct underlying material information. A simple color blending with tone mapping could be a Reinhard Tonemap:

$$\text{blend\_color} = \text{albedo} + \text{emission} \times \text{strength} \quad (4)$$

$$\text{albedo\_convert} = \text{blend\_color}/(1 + \text{blend\_color}) \quad (5)$$

Normal Maps. Normal maps typically influence geometric perception far more than color appearance. Conceptually, they serve as a geometric approximation—much like fully transparent regions in color maps—allowing fine-scale surface detail to be represented without increasing mesh complexity, which is crucial for real-time rendering efficiency. However, in scenes with lighting and on relatively smooth surfaces, discrepancies between the perturbed normals from the normal map and the true geometric surface

can introduce shading artifacts. These non-orthogonal deviations may create highlights that do not correspond to the underlying geometry, or suppress highlights that should be present. Such inconsistencies lead to tonal variations that no longer align with the surface's intrinsic color sampling. To avoid propagating these shading-induced biases, we disable the use of normal maps during rendering.

Other Materials. Unlike the base-color-roughness-metallic workflow, the diffuse-specular workflow is typically designed for stylized or non-physically-based rendering. While it is sometimes possible to approximate a conversion from diffuse-specular to base-color-roughness-metallic using heuristic rules, significant discrepancies often remain, particularly under varying lighting conditions. In effect, the range of appearance captured by diffuse-specular constitutes a superset of what is representable by the physically-based base color-roughness-metallic workflow. To avoid introducing inconsistencies, we therefore only retain data that is high specular with low diffuse (metals) or low specular with high diffuse (non-metals) during dataset construction.

B.3. Geometry Processing

Interior Removal. A clean geometric structure is essential for both geometry learning and texture learning. Although our framework is capable of assigning colors to any point in the 3D space, learning a stable and expressive color latent space requires restricting supervision to points that truly lie on the visible surface. To enforce this, we retain only surface samples and remove all points that fall inside the mesh volume.

We achieve this using a flood-fill–based procedure. Specifically, we first voxelize the mesh and compute connected components representing empty space. By identifying the outermost empty connected component, we can determine which voxels lie on or near the actual surface. We then keep only the sampled points located in voxels directly adjacent to this exterior region, while discarding all samples in interior voxels. This ensures that the training supervision reflects the true surface geometry, avoids color ambiguity from internal samples, and leads to more consistent and reliable texture learning.

Outlined Objects. "Outline" meshes are commonly used in stylized rendering: a slightly expanded duplicate mesh, rendered with backface culling and flipped normals, produces a black contour around the object. However, during surface sampling, this shell introduces misleading proximity cues. Points sampled on the outline mesh may be incorrectly associated with voxels of the primary surface, causing the encoder to learn blurred or corrupted textures.

Figure 2. **Outlined model and outline-removed model.** The model with stylized outline meshes and the model after their removal.

To prevent this, our pipeline automatically detects and removes such non-essential geometry. We use a set of heuristics to identify meshes exhibiting characteristics of outline shells (e.g., inverted normals, exclusive backface visibility, or emissive black materials) and exclude them from the sampling stage, ensuring that learning is guided only by the true visible surface.

As an alternative, we can directly eliminate the "outline triangles" themselves. We cast rays from the outside of the object toward the mesh and examine the triangles they intersect. If a hit triangle is a backface and its material is configured to be backface-culled, we classify it as part of the outline mesh and mark it for removal. To obtain reliable exterior ray origins, we apply a flood-fill on the voxel grid to locate a voxel center that belongs to an exterior connected component adjacent to the mesh surface. From this seed, we densely project rays onto the nearby mesh and analyze the intersected triangles to determine which should be discarded. Fig. 2 illustrates several models containing outline meshes alongside their outline-removed counterparts. Although outline meshes are often rendered as a uniform solid color, they may also appear with per-vertex colors or as UV-mapped textures. Therefore, simply checking for backface-culling attributes or verifying whether the material is a solid color is insufficient. Instead, the determination must rely on whether the geometry would actually be rendered, although this approach can still be problematic for some non-watertight surfaces.

Overlapped Faces. For overlapping or near-overlapping surfaces, two cases may occur. In the first case, the surfaces share identical colors, which does not affect sampling or training since the duplicated geometry yields consistent supervision. In the second—and more problematic—case, the overlapping surfaces carry different colors. Such configurations cause mutual interference during sampling: points drawn from adjacent but distinct layers produce mixed or

Table 1. **Traits of different commercial models.** Number of triangular faces, UV fragmentation, and geometric quality in different mesh groups.

Model Group	#Faces	Fragmentation	Geometry Quality
#1	0.5M	middle	middle
#2	0.5M	high	middle
#3	1.0M	middle	high
#4	1.5M	high	high

averaged color signals, leading to undesirable blurring and smoothing in the learned representation. To avoid introducing these ambiguities into the dataset, we detect meshes containing overlapping or tightly coupled surfaces and exclude them from our data.

B.4. Scale of Training Data

We note that differences in training data scale can significantly affect model performance and should be carefully considered when making comparisons. Our model is trained on 1M samples; however, it may still exhibit strong generalization under reduced data regimes, which we leave for future investigation. Moreover, comparing against prior methods trained on smaller datasets offers a more informative and fair assessment of the intrinsic characteristics and applicability of different 3D representations, as it helps disentangle the effects of representation design from those of data scale.

C. Experiment Evaluation

C.1. Data Sources

We collected 4 groups of meshes with different traits from different 3D commercial AI generation tools resource [1–3, 8, 19], prompted with about 200 images in same for each tool. Tab. 1 shows the number of triangle faces, the UV fragmentation (density of UV islands), and the quality of the geometry (alignment with prompt images) of the mesh.

C.2. Evaluation Metrics

For our evaluation, we test 800 meshes and compare our approach against eight baseline methods by computing the Fréchet Distance (FD) [4] and Kernel Distance (KD) [17] between the prompt images and the rendered outputs. The baselines cover text-conditioned, image-conditioned, and mixed image-plus-text-conditioned generation methods. Because the generators are generally pose-agnostic, we render multiple views of each mesh and compute the metrics on four different model groups (#1−#4 in Tab. 1). Since several baselines do not support PBR rendering, we evaluate both shaded and unshaded versions to ensure fair comparison.

The table in the main paper reports the metric computed between the prompt images and three canonical renders views of each mesh: a front view, a view at $45°$ azimuth and $30°$ elevation, and a view at $-45°$ azimuth and $30°$ elevation. Tab. 2-5 provides a more exhaustive evaluation using eight render views: azimuth angles of $0°$, $90°$, $-90°$, and $180°$ at $0°$ elevation, and azimuth angles of $45°$, $135°$, $-45°$, and $-135°$ at $30°$ elevation, correspond to each model group in Tab. 1.

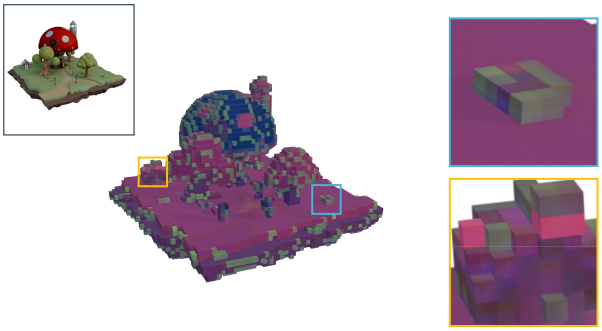

Figure 3. **Locality.** Our approach exhibits locality, which enables it to generalize more effectively to higher-resolution inference.

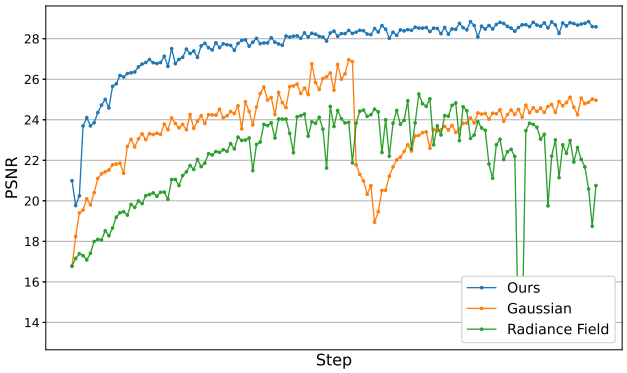

Figure 4. **Stability.** Our method demonstrates better stability during larger-scale training.

C.3. Additional Analysis

Locality. During inference, our method exhibits strong scalability. Unlike multi-view feature projection inputs or outputs like GS [11] or NeRF [15], our approach can directly infer a 128^3-resolution color latent from a 64^3 model. Fig. 3 visualizes the $4\times$ denser voxel grid obtained after decoding the color latent and applying upsampling (the voxel colors are visualized using PCA for dimensionality reduction from 512-channel color field features to 3-channel RGB value). Notably, even after convolutional upsampling, the voxels retain sharp boundaries at the unsampled resolution scale, highlighting the inherently local nature of our representation. In contrast to the mesh branch of TRELLIS and other octree-based frameworks, our color-field decoder

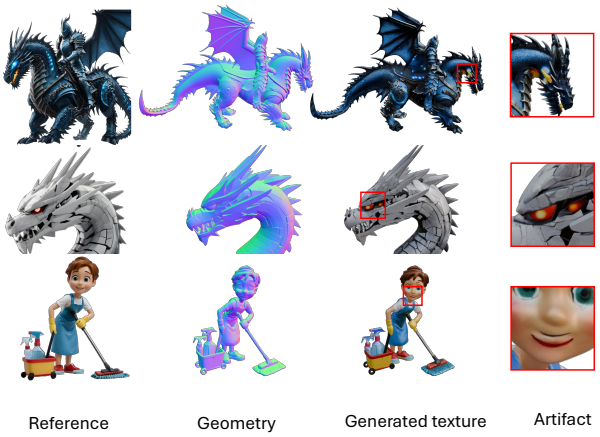

Figure 5. **Failure cases.** Several failure cases of our generation model, due to the lack of 2D priors.

predicts the color of a point using only the latent of the voxel it belongs to, without interpolating across neighboring voxels. This design leads to a more localized and structurally coherent representation.

Stability. Using the same point cloud input, we demonstrate the superior fitting capability of our method by comparing the fitting efficiency of Gaussian Splatting and Radiance Fields against our Color Field outputs. In addition to the comparisons presented in the main text, we further evaluate generalization by scaling up the dataset size. As shown in Fig. 4, our method exhibits significantly more stable training behavior. For fairness, both GS and NeRF follow the parameter settings used in TRELLIS [21], including rendering supervision with LPIPS and SSIM, as well as volume and opacity regularization.

Baking efficiency. Our approach requires UV unwrapping and is compatible with varying UV map resolutions. By default, we adopt a 2048×2048 UV map for texture baking. Furthermore, texture querying can be seamlessly integrated into the rendering pipeline. Owing to the lightweight color decoding MLP, our method supports real-time rendering of 360° surround videos at 512×512 resolution and 60 FPS. In terms of efficiency, diffusion-based latent generation takes approximately 20 seconds on average, depending on the voxel count, while the baking process requires only about 3 seconds.

C.4. More Visualizations

Failure cases. Fig. 5 show some failure cases that is unlikely to occur due to inconsistency or occlusion, but lack of 2D priors, as mentioned in the main text. These issues will be addressed in the future through feature alignment and knowledge distillation by 2D image generation models.

More comparisons. Fig. 6 provide additional comparisons of image-conditioned generation with other baselines.

Table 2. **Quantitative comparison for conditional texture generation on 8 views use model group #1.**

Method	Condition	Unshaded					Shaded				
		FID↓	FD_{CLIP} ↓	FD_{DINO} ↓	KD_{CLIP} ↓	KD_{DINO} ↓	FID↓	FD_{CLIP} ↓	FD_{DINO} ↓	KD_{CLIP} ↓	KD_{DINO} ↓
MVPaint [5]	Text	145.31	88.79	124.39	0.136	0.147	128.58	81.59	102.30	0.122	0.079
MaterialAnything [10]	Text	176.88	112.46	133.69	0.236	0.159	137.46	87.79	129.50	0.140	0.163
SyncMVD [13]	Text	124.70	78.01	91.09	0.082	0.049	123.72	73.83	88.67	0.087	0.050
TEXGen [22]	Text+Image	140.60	100.56	124.92	0.178	0.123	136.64	88.54	112.80	0.138	0.102
SeqTex [23]	Text+Image	127.52	70.49	89.79	0.073	0.042	125.30	73.50	90.96	0.093	0.052
Paint3D [24]	Text/Image	126.37	76.19	94.17	0.112	0.061	122.07	71.09	91.52	0.100	0.058
MaterialMVP [7]	Image	120.27	65.42	80.63	0.075	0.036	110.11	59.18	81.08	0.060	0.041
UniTEX [12]	Image	110.79	59.26	81.24	0.056	0.043	111.66	61.09	82.34	0.061	0.046
Ours	Image	111.66	60.44	77.58	0.054	0.036	106.49	54.65	77.08	0.045	0.032

Table 3. **Quantitative comparison for conditional texture generation on 8 views use model group #2.**

Method	Condition	Unshaded					Shaded				
		FID↓	FD_{CLIP} ↓	FD_{DINO} ↓	KD_{CLIP} ↓	KD_{DINO} ↓	FID↓	FD_{CLIP} ↓	FD_{DINO} ↓	KD_{CLIP} ↓	KD_{DINO} ↓
MVPaint [5]	Text	147.90	90.24	133.81	0.142	0.166	122.50	77.11	102.91	0.098	0.080
MaterialAnything [10]	Text	161.57	99.68	128.18	0.184	0.139	126.97	80.23	124.06	0.104	0.138
SyncMVD [13]	Text	123.11	77.03	95.23	0.078	0.057	118.51	69.74	87.89	0.070	0.034
TEXGen [22]	Text+Image	135.80	89.71	113.74	0.134	0.101	132.24	81.64	105.25	0.108	0.076
SeqTex [23]	Text+Image	118.88	65.31	86.91	0.051	0.032	119.03	68.55	89.92	0.069	0.052
Paint3D [24]	Text/Image	123.02	71.62	93.60	0.090	0.059	116.38	64.89	87.75	0.072	0.048
MaterialMVP [7]	Image	118.60	64.82	85.49	0.067	0.042	107.72	57.27	82.80	0.052	0.038
UniTEX [12]	Image	110.39	58.43	85.16	0.050	0.046	110.66	60.14	88.04	0.057	0.056
Ours	Image	113.35	60.94	83.07	0.051	0.046	107.84	54.03	79.35	0.039	0.034

Table 4. **Quantitative comparison for conditional texture generation on 8 views use model group #3.**

Method	Condition	Unshaded					Shaded				
		FID↓	FD_{CLIP} ↓	FD_{DINO} ↓	KD_{CLIP} ↓	KD_{DINO} ↓	FID↓	FD_{CLIP} ↓	FD_{DINO} ↓	KD_{CLIP} ↓	KD_{DINO} ↓
MVPaint [5]	Text	148.50	91.26	133.14	0.144	0.177	127.88	78.51	107.79	0.108	0.093
MaterialAnything [10]	Text	171.87	106.51	125.94	0.212	0.145	134.94	81.25	126.21	0.117	0.169
SyncMVD [13]	Text	127.57	79.20	97.24	0.086	0.069	122.67	71.73	90.36	0.076	0.055
TEXGen [22]	Text+Image	142.97	101.65	129.79	0.179	0.139	137.72	89.32	113.39	0.136	0.097
SeqTex [23]	Text+Image	122.39	64.41	83.43	0.055	0.039	119.85	66.64	84.21	0.072	0.044
Paint3D [24]	Text/Image	129.09	78.63	98.15	0.119	0.074	120.44	69.61	90.49	0.090	0.056
MaterialMVP [7]	Image	118.22	63.24	78.33	0.071	0.039	108.00	55.71	78.10	0.051	0.039
UniTEX [12]	Image	108.49	55.17	77.46	0.047	0.038	108.11	55.47	77.12	0.050	0.038
Ours	Image	109.92	58.95	77.95	0.051	0.035	104.43	51.67	75.54	0.037	0.033

Table 5. **Quantitative comparison for conditional texture generation on 8 views use model group #4.**

Method	Condition	Unshaded					Shaded				
		FID↓	FD_{CLIP} ↓	FD_{DINO} ↓	KD_{CLIP} ↓	KD_{DINO} ↓	FID↓	FD_{CLIP} ↓	FD_{DINO} ↓	KD_{CLIP} ↓	KD_{DINO} ↓
MVPaint [5]	Text	158.94	101.20	142.38	0.180	0.185	129.65	81.88	112.06	0.112	0.102
MaterialAnything [10]	Text	166.35	102.82	131.09	0.190	0.140	128.96	80.90	125.00	0.109	0.137
SyncMVD [13]	Text	130.98	82.00	104.14	0.093	0.072	124.76	72.73	93.48	0.080	0.055
TEXGen [22]	Text+Image	136.99	92.36	115.38	0.149	0.093	134.93	83.81	107.17	0.118	0.083
SeqTex [23]	Text+Image	119.49	65.65	89.80	0.051	0.037	118.71	67.99	89.16	0.070	0.044
Paint3D [24]	Text/Image	128.34	76.30	101.67	0.110	0.072	119.20	68.01	92.94	0.082	0.054
MaterialMVP [7]	Image	116.92	64.19	84.32	0.070	0.039	106.77	56.45	81.01	0.047	0.037
UniTEX [12]	Image	115.54	65.15	88.66	0.068	0.048	113.11	63.24	88.08	0.063	0.048
Ours	Image	111.38	59.40	81.09	0.050	0.037	105.86	52.43	76.95	0.037	0.029

Figure 6. **Image-conditioned texture generation.**

References

[1] Hyper3D / Rodin AI. Rodin – free ai 3d model generator for everyone. https://hyper3d.ai/, 2025. Accessed: 2025-10-30. 4

[2] Meshy AI. Meshy ai – meet the world's most popular and intuitive free ai 3d model generator. https://www.meshy.ai/, 2025. Accessed: 2025-10-30.

[3] Tripo AI. Tripo ai – create your first 3d model with text and image. https://www.tripo3d.ai/, 2025. Accessed: 2025-10-30. 4

[4] Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 327–338. Springer, 1995. 4

[5] Wei Cheng, Juncheng Mu, Xianfang Zeng, Xin Chen, Anqi Pang, Chi Zhang, Zhibin Wang, Bin Fu, Gang Yu, Ziwei Liu, et al. Mvpaint: Synchronized multi-view diffusion for painting anything 3d. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 585–594, 2025. 6

[6] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. 2023. 1

[7] Zebin He, Mingxin Yang, Shuhui Yang, Yixuan Tang, Tao Wang, Kaihao Zhang, Guanying Chen, Yuhong Liu, Jie Jiang, Chunchao Guo, et al. Materialmvp: Illumination-invariant material generation via multi-view pbr diffusion. *arXiv preprint arXiv:2503.10289*, 2025. 6

[8] HiTem3D. Hitem3d: Next-gen ai 3d model generator. https://hitem3d.ai/, 2025. Accessed: 2025-10-30. 4

[9] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *CoRR*, abs/2207.12598, 2022. 1

[10] Xin Huang, Tengfei Wang, Ziwei Liu, and Qing Wang. Material anything: Generating materials for any 3d object via diffusion. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 26556–26565, 2025. 6

[11] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 4

[12] Yixun Liang, Kunming Luo, Xiao Chen, Rui Chen, Hongyu Yan, Weiyu Li, Jiarui Liu, and Ping Tan. Unitex: Universal high fidelity generative texturing for 3d shapes. *arXiv preprint arXiv:2505.23253*, 2025. 6

[13] Yuxin Liu, Minshan Xie, Hanyuan Liu, and Tien-Tsin Wong. Text-guided texturing by synchronized multi-view diffusion. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11, 2024. 6

[14] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471, 2022. 1

[15] 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. In *ECCV*, 2020. 4

[16] Maxime Oquab, Timothée Darcet, Theo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Russell Howes, Po-Yao Huang, Hu Xu, Vasu Sharma, Shang-Wen Li, Wojciech Galuba, Mike Rabbat, Mido Assran, Nicolas Ballas, Gabriel Synnaeve, Ishan Misra, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2023. 1

[17] Jeff M. Phillips and Suresh Venkatasubramanian. A geometric approach to comparing datasets via kernel distance. In *Proceedings of the 27th Annual Symposium on Computational Geometry (SoCG)*, pages 317–326. ACM, 2011. 4

[18] Minglei Shi, Haolin Wang, Wenzhao Zheng, Ziyang Yuan, Xiaoshi Wu, Xintao Wang, Pengfei Wan, Jie Zhou, and Jiwen Lu. Latent diffusion model without variational autoencoder. *arXiv preprint arXiv:2510.15301*, 2025. 1

[19] Tencent Hunyuan3D Team. Tencent hunyuan3d. https://3d.hunyuan.tencent.com/, 2025. Accessed: 2025-10-30. 4

[20] Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, et al. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. *arXiv preprint arXiv:2502.14786*, 2025. 1

[21] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 21469–21480, 2025. 1, 5

[22] Xin Yu, Ze Yuan, Yuan-Chen Guo, Ying-Tian Liu, Jianhui Liu, Yangguang Li, Yan-Pei Cao, Ding Liang, and Xiaojuan Qi. Texgen: a generative diffusion model for mesh textures. *ACM Transactions on Graphics (TOG)*, 43(6):1–14, 2024. 6

[23] Ze Yuan, Xin Yu, Yangtian Sun, Yuan-Chen Guo, Yan-Pei Cao, Ding Liang, and Xiaojuan Qi. Seqtex: Generate mesh textures in video sequence. *arXiv preprint arXiv:2507.04285*, 2025. 6

[24] Xianfang Zeng, Xin Chen, Zhongqi Qi, Wen Liu, Zibo Zhao, Zhibin Wang, Bin Fu, Yong Liu, and Gang Yu. Paint3d: Paint anything 3d with lighting-less texture diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4252–4262, 2024. 6