SWIFT3D: END-TO-END ACCELERATION OF HIGH-RESOLUTION NATIVE 3D GENERATION

Anonymous authors
Paper under double-blind review



Figure 1: High-resolution geometry and textured 3D assets generated by Swift3D.

ABSTRACT

Native 3D generative models aim to synthesize detailed geometry and textures from a single image, but producing high-resolution textured meshes remains time-consuming, particularly during iterative diffusion sampling, geometry decoding, and UV parameterization. To address these bottlenecks, we present Swift3D, an end-to-end acceleration framework for high-resolution native 3D generation. We develop stage-aware distillation of diffusion transformers (DiTs), combining phased consistency model (PCM) initialization with teacher anchoring of coarse geometry during distributional and adversarial refinement, enabling generation with fewer sampling steps. Adaptive geometry decoding uses probes to guide subdivision and reconciles coarse and fine fields through consistent reconstruction, reducing redundant neural field queries for high-resolution surface extraction. Progressive UV atlasing aligns chart construction with projection directions and iteratively refines only failed charts, reducing UV parameterization overhead on dense meshes. Experiments on geometry reconstruction, geometry generation, texturing, and end-to-end generation demonstrate substantial acceleration while retaining high fidelity. Swift3D reduces average end-to-end latency from 520.56 to 47.39 seconds ($10.98\times$), outperforming the evaluated open-source baselines in asset quality. Further distilling the geometry-refinement and texturing DiTs to one step each enables generation in under 30 seconds with quality comparable to high-quality open-source state-of-the-art methods.

1 INTRODUCTION

3D assets are widely used in games, film production, and virtual environments, motivating efficient methods for asset creation and iteration. Recent native 3D generative models have made substantial progress in generating high-resolution geometry and textures from a single image (Lai et al., 2026a;

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Xiang et al., 2026; Lai et al., 2026b). However, high-resolution native 3D generation remains time-consuming, limiting rapid asset creation and iteration. The dominant computational cost arises from iterative model sampling, geometry decoding, and UV parameterization. Reducing end-to-end latency from an image to a textured mesh while preserving quality remains a key challenge.

Iterative evaluations of diffusion transformers (DiTs) make sampling costly. Few-step distillation reduces this cost (Wang et al., 2024), but preserving quality across geometry generation, geometric refinement, and texture generation is challenging because these stages serve different roles. Initial geometry establishes global structure and correspondence to the input, while later stages recover local details and material boundaries. Distribution matching can improve sample quality without explicitly preserving teacher–student correspondence (Yin et al., 2024). In cascaded 3D generation, the challenge is to retain conditional structure and fine detail under a substantially reduced sampling budget.

After sampling is accelerated, high-resolution geometry decoding becomes the dominant runtime cost in our pipeline (Figure 2). Decoding costs depend on the underlying representation, with common choices including sparse voxels (Xiang et al., 2026) and compact latent sets such as VecSet (Zhang et al., 2023). Our pipeline builds on LATTICE (Lai et al., 2026a), whose spatially anchored VoxSet representation supports high-fidelity reconstruction and detailed geometry generation. However, recovering high-resolution surfaces from these compact latents still requires many neural field evaluations. FlashVDM reduces this query cost through hierarchical volume decoding that skips regions far from the surface (Lai et al., 2025). Yet selected near-surface regions are still refined to the target resolution, even when a coarse field may already capture the local geometry. Moreover, predictions across refinement levels and query contexts may disagree, causing surface artifacts. The challenge is to reduce redundant queries while preserving geometric detail and surface continuity.

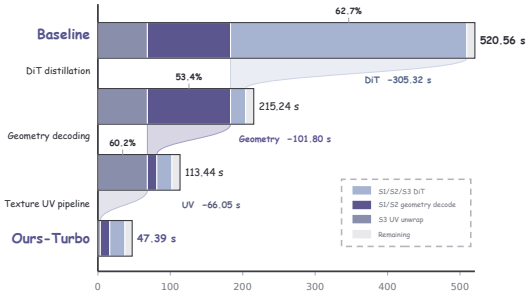

Figure 2: Step-by-step effects of Swift3D acceleration techniques on end-to-end runtime.

Accelerating sampling and geometry decoding shifts the runtime bottleneck to UV parameterization on dense meshes. Methods such as xatlas (Young, 2019) repeatedly revisit faces and validate their mappings during incremental chart growth and merging. SF3D (Boss et al., 2024) uses direct projection for fast UV construction, but placing triangles with unresolved overlaps separately can fragment connected surfaces. The challenge is therefore to reduce UV parameterization cost while avoiding overlaps and excessive chart fragmentation.

To address these challenges, we present Swift3D, an end-to-end acceleration framework for high-resolution native 3D generation that accelerates sampling, geometry decoding, and UV parameterization while preserving generation quality. For DiT sampling, we develop stage-aware distillation that initializes each student with phased consistency model (PCM) training (Wang et al., 2024) before distributional and adversarial refinement. A teacher anchor is applied only to coarse geometry to preserve the structure inherited by subsequent stages. Together, these designs accelerate total DiT sampling across the three stages by $15.60\times$. We further introduce adaptive geometry decoding. Probe-guided subdivision allocates neural queries according to local approximation error, while consistent field reconstruction reconciles predictions across query contexts and refinement levels. Our decoder achieves $4.41\times$ and $7.32\times$ reconstruction decoding speedups at resolutions of 1024 and 1536, respectively. For UV parameterization, progressive UV atlasing combines projection-aligned charting with iterative local refinement. It groups connected faces by projection direction and selectively subdivides and reprojects charts with unresolved overlaps, achieving a $20.16\times$ speedup.

Swift3D reduces average end-to-end latency from 520.56 to 47.39 seconds ($10.98\times$), retaining high geometry and texture fidelity while outperforming the evaluated state-of-the-art open-source methods on end-to-end quality metrics. Distilling the geometry-refinement and texturing DiTs to one step each (1 NFE) yields assets in under 30 seconds with quality comparable to these baselines.

Our contributions are fourfold:

- To the best of our knowledge, we present the first end-to-end acceleration framework for high-resolution textured asset generation with native 3D models. Swift3D delivers product-ready assets in under 50 seconds on average, achieving a $10.98\times$ speedup over our reference pipeline while retaining high fidelity.
- We develop stage-aware distillation with PCM initialization and teacher anchoring, achieving a $15.60\times$ speedup in total DiT sampling across the three stages.
- We propose adaptive geometry decoding that combines probe-guided subdivision with consistent field reconstruction, achieving $4.41\times$ and $7.32\times$ decoding speedups at reconstruction resolutions of 1024 and 1536, respectively.
- We introduce progressive UV atlasing that couples projection-aligned charting with iterative local refinement, achieving a $20.16\times$ speedup in UV parameterization.

2 Related work

3D Generation. 3D generation methods draw on either pretrained 2D priors or generative models learned directly from 3D data. DreamFusion (Poole et al., 2023) uses pretrained image diffusion to supervise per-instance 3D optimization. Native 3D representations include compact latent sets in 3DShape2VecSet (Zhang et al., 2023), spatially anchored VoxSet latents in LATTICE (Lai et al., 2026a), and sparse structured latents in TRELLIS.2 (Xiang et al., 2026). Texture generation spans multi-view material diffusion in MaterialMVP (He et al., 2025), UV-space diffusion in TEXGen (Yu et al., 2024), and latent 3D color diffusion in NaTex (Lai et al., 2026b). In this context, we accelerate native 3D generation end to end while preserving geometry and texture quality.

DiT Distillation. Few-step DiT generation builds on distillation techniques developed for diffusion models. Progressive distillation (Salimans & Ho, 2022) repeatedly halves sampling steps, while consistency models (Song et al., 2023) learn consistent mappings from noisy states to data. PCM (Wang et al., 2024) divides trajectories into phases for few-step generation. Complementing trajectory-based training, DMD2 (Yin et al., 2024) combines distribution matching with adversarial learning for image synthesis. CausVid (Yin et al., 2025) extends distribution matching to few-step autoregressive video generation. In 3D, FlashVDM (Lai et al., 2025) introduces progressive flow distillation, and MDT-dist (Zhou et al., 2026) distills both flow transformers in TRELLIS through marginal-data transport. In this work, we combine PCM initialization with distributional and adversarial refinement, using a teacher anchor on coarse geometry to preserve structure across stages.

3D Acceleration. Beyond diffusion sampling, constructing textured meshes requires efficient geometry decoding and UV unwrapping. FlashVDM (Lai et al., 2025) reduces decoding cost through hierarchical near-surface querying and adaptive key–value selection. Unlike conventional chart-based tools such as xatlas (Young, 2019), SF3D (Boss et al., 2024) accelerates UV unwrapping through parallelizable cube projection. However, its separate placement of residual triangles can fragment connected surfaces. Further gains lie in reducing redundant field queries and chart processing while preserving geometric detail and limiting UV fragmentation.

3 Method

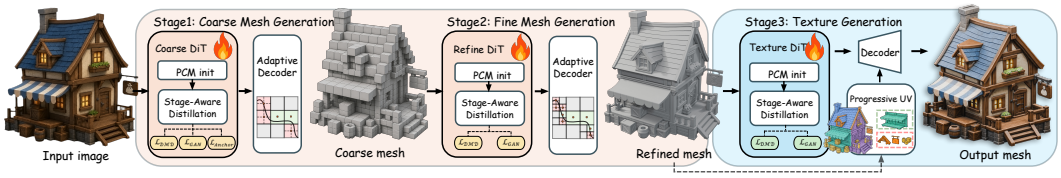

Figure 3: Overview of the Swift3D generation and acceleration pipeline.

In this section, we first outline the Swift3D pipeline (Figure 3). We then present stage-aware distillation for DiT sampling (Sec. 3.1), adaptive geometry decoding (Sec. 3.2), and progressive UV atlasing (Sec. 3.3). Together, they reduce end-to-end latency while preserving asset quality.

3.1 STAGE-AWARE DISTILLATION

The three DiTs in Figure 3 perform different conditional generation tasks, with target distributions over coarse shapes, refined geometry, and textures. These stages are also interdependent: the coarse shape provides the structural basis for refinement and texturing. Few-step distillation must therefore accommodate the different output distributions while preserving the conditional structure on which the cascade depends.

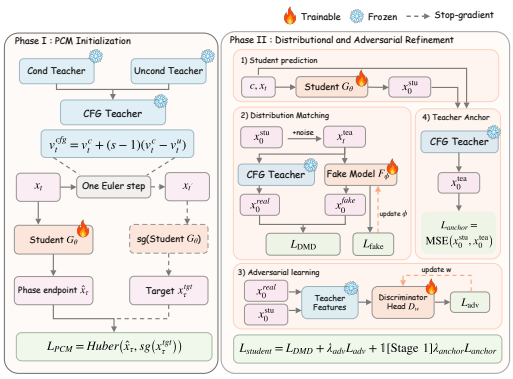

Figure 4: Training pipeline for stage-aware distillation.

To address these requirements, we use a two-phase distillation scheme with stage-specific teacher anchoring (Figure 4). High-resolution 3D generation requires modeling both global structure and fine geometric and appearance details. Simply reducing the sampling steps of a pretrained multi-step model can substantially degrade output quality. Inspired by the importance of few-step initialization in Causal Forcing++ (Zhao et al., 2026), we first establish each student's few-step generation capability through phased consistency model (PCM) initialization (Wang et al., 2024). All three students then undergo distribution matching and adversarial training (Yin et al., 2024). During this second phase, we additionally anchor the coarse-geometry student's prediction to its teacher at the same noisy state and condition, providing an explicit reference for the geometry passed to later stages. We describe one stage below, omitting its index until the combined objective.

PCM initialization. Our PCM initialization learns phase-endpoint predictions from local teacher transitions computed online, without precomputing full teacher trajectories. We use normalized flow time, where $t = 1$ denotes noise and $t = 0$ a clean latent. Let v_T and v_θ denote the frozen teacher and trainable student velocity fields, respectively, and c the condition for the current generation stage. The teacher uses classifier-free guidance (CFG), $v_T^{\text{cfg}} = v_T^c + (s-1)(v_T^c - v_T^\emptyset)$, where s is the guidance scale and $\emptyset$ denotes the unconditional branch.

Given a noisy latent $\mathbf{x}_t$, we take a local teacher step to a nearby time t'. This step is clipped at the endpoint τ of the current consistency phase, ensuring $\tau \le t' < t$. The teacher transition and the student's phase-endpoint prediction are

$$\mathbf{x}_{t'} = \mathbf{x}_t + (t'-t)v_T^{\text{cfg}}(\mathbf{x}_t, t, c), \qquad G_\theta(\mathbf{x}_t, t, c; \tau) = \mathbf{x}_t + (\tau - t)v_\theta(\mathbf{x}_t, t, c). \tag{1}$$

We train the student to predict the same phase endpoint from $\mathbf{x}_t$ and $\mathbf{x}_{t'}$ by minimizing

$$\mathcal{L}_{\text{PCM}} = \mathbb{E}[\rho_\delta(G_\theta(\mathbf{x}_t, t, c; \tau) - \text{sg}[G_\theta(\mathbf{x}_{t'}, t', c; \tau)])], \tag{2}$$

where sg denotes stop-gradient and $\rho_\delta(u) = \sqrt{u^2 + \delta^2} - \delta$ is a smooth Huber loss, averaged over latent elements. Both predictions use the same student, with gradients stopped through the target evaluated at $\mathbf{x}_{t'}$. Because teacher transitions use CFG, PCM transfers guided behavior to the student without unconditional evaluations at inference.

Distributional and adversarial refinement. PCM provides a trajectory-based initialization, but matching phase endpoints alone does not directly optimize the realism of generated assets. We therefore refine the initialized students with DMD2 (Yin et al., 2024). For a student prediction $\mathbf{x}_0^{\text{stu}}$, we add noise at an independently sampled time and obtain two denoised estimates: one from the frozen CFG teacher and one from an online fake-score model F_ϕ. Their normalized difference supplies the distribution-matching update $\mathcal{L}_{\text{DMD}}$; F_ϕ is trained by denoising regression on perturbed student outputs. To further improve local structure and appearance, we train a discriminator head D_w on frozen teacher features of real and generated latents, yielding an adversarial generator loss $\mathcal{L}_{\text{adv}}$. This supervision operates in latent space, avoiding mesh or texture decoding for discrimination. Student updates alternate with fake-score and discriminator updates.

Teacher anchoring for coarse geometry. PCM initializes the student through consistency training along the teacher's guided trajectories. During subsequent refinement, however, distribution

matching aligns output distributions without explicitly preserving teacher–student correspondence at the same noisy state and condition (Yin et al., 2024). Consequently, refinement may alter the geometry generated for a given condition even when local detail improves. In our coarse-geometry distillation runs, we observed orientation and proportion drift. We query the frozen CFG teacher at the student's input $(\mathbf{x}_t, t, c_1)$ and penalize their clean-latent discrepancy:

$$\mathbf{x}_0^{\text{tea}} = \mathbf{x}_t - t\, v_{T_1}^{\text{cfg}}(\mathbf{x}_t, t, c_1), \qquad \mathcal{L}_{\text{anchor}} = \mathbb{E}\big[\text{MSE}\big(G_{\theta_1}(\mathbf{x}_t, t, c_1; 0), \text{sg}[\mathbf{x}_0^{\text{tea}}]\big)\big]\,. \tag{3}$$

The online target uses the same noisy latent, time, and image condition as the student. We apply this anchor only to Stage 1: coarse geometry establishes the structure inherited by the cascade, whereas refinement and texturing must remain free to improve fine structure and appearance. The second-phase objective for stage $k \in \{1, 2, 3\}$ is

$$\mathcal{L}_{\text{student}}^{(k)} = \mathcal{L}_{\text{DMD}}^{(k)} + \lambda_{\text{adv}}^{(k)} \mathcal{L}_{\text{adv}}^{(k)} + \mathbf{1}_{\{k=1\}} \lambda_{\text{anchor}} \mathcal{L}_{\text{anchor}}, \tag{4}$$

with stage-specific adversarial weights and a positive anchor weight for coarse geometry. At inference, only the distilled students are evaluated: each uses one conditional forward pass per sampling step, reducing both trajectory length and the cost of CFG across the three-stage pipeline.

3.2 Adaptive Geometry Decoding

After DiT distillation, high-resolution geometry decoding becomes the next major bottleneck (Figure 2). To recover a mesh from the generated shape latents, we query a neural decoder for signed-distance field (SDF) values and then extract the iso-surface. The cost of these neural field evaluations depends on the spatial sampling strategy. A vanilla implementation queries every point on the target grid, so the query count grows cubically with resolution. FlashVDM (Lai et al., 2025) uses hierarchical volume decoding to skip regions far from the surface and adaptive KV selection to reduce the number of latent tokens involved in each query. However, two issues remain. First, selected near-surface regions are still queried up to the finest resolution. For example, a coarse field can already capture the geometry of a planar region, making further queries at finer resolutions redundant. Second, the latent key–value subset depends on the local query group. As the selected subset changes, predictions at the same position may differ, causing newly queried values to disagree with cached coarse values.

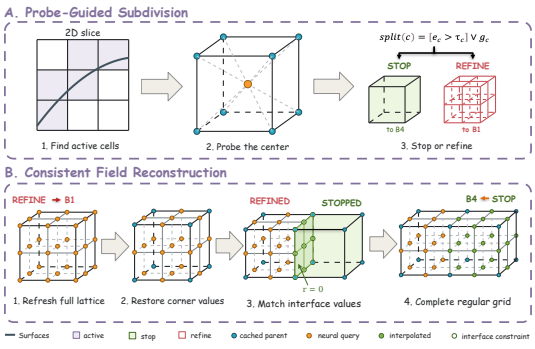

Figure 5: Adaptive geometry decoding pipeline.

To address these issues, we couple adaptive query allocation with consistent field reconstruction (Figure 5). We allocate neural queries according to the local need for refinement. Probe-guided subdivision retains coarse fields where sufficient and concentrates finer queries on regions requiring refinement. We then anchor fine samples to cached values at parent corners and align field values across boundaries with regions that retain coarse fields. These corrections reconcile predictions across query contexts and refinement levels before surface extraction.

Probe-Guided Subdivision. For each cell c whose corner values straddle the surface, we probe the decoder at the cell center and compare its prediction s_c with a coarse estimate $\hat{s}_c$ interpolated from its eight cached corners. Their normalized discrepancy guides subdivision:

$$e_c = \frac{|s_c - \hat{s}_c|}{\Delta_c}, \qquad \text{split}(c) = [e_c > \tau_c] \vee g_c, \tag{5}$$

where Δ_c measures local field variation from the cached corners and τ_c is the refinement tolerance for cell c. The guard g_c is true when the queried and interpolated center values lie on different sides of the extraction iso-level ℓ, triggering refinement even for a small residual. Cells that do not split retain their coarse field; the others receive finer queries.

Consistent Field Reconstruction. Let $q_c(\mathbf{x})$ denote the newly queried SDF value at position $\mathbf{x}$ within cell c. Let $\mathbf{s}_c^{\text{corner}}$ and $\mathbf{q}_c^{\text{corner}}$ denote the vectors of cached and newly queried SDF values

at the same eight parent corners, respectively. We interpolate their differences to correct the newly queried field:

$$q_c^{\text{anchor}}(\mathbf{x}) = q_c(\mathbf{x}) + I_8(\mathbf{s}_c^{\text{corner}} - \mathbf{q}_c^{\text{corner}})(\mathbf{x}), \tag{6}$$

where I_8 denotes trilinear interpolation from eight corners and q_c^{anchor} is the corrected field. This restores the cached corner values while retaining the new samples' fine residuals. Let $r_c(\mathbf{x}) = q_c^{\text{anchor}}(\mathbf{x}) - I_8(\mathbf{s}_c^{\text{corner}})(\mathbf{x})$ denote the remaining fine residual. At samples on each shared face with a stopped region, we set $r_c(\mathbf{x}) = 0$, so both sides use the same coarse interpolant. Interior residuals are retained for geometric detail, while scalar field values agree across the interface. Both corrections use existing samples without additional neural queries. Corrected samples and interpolation of stopped regions then complete the grid for surface extraction.

3.3 Progressive UV Atlasing

With geometry decoding and texture synthesis accelerated, UV unwrapping becomes a major remaining cost on dense meshes (Figure 2). Many 3D generation methods use xatlas (Young, 2019), which directly projects planar regions to simplify unwrapping. In our tests, its strict planar criteria cover few faces on complex meshes. Most faces therefore undergo chart growth, merging, and parameterization, with repeated checks and recomputation when mappings fail. SF3D (Boss et al., 2024) offers fast projection along six fixed directions, but placing triangles with unresolved overlaps individually can fragment connected surfaces. PartUV (Wang et al., 2025) uses learned part information to reduce fragmentation but remains costly on complex meshes.

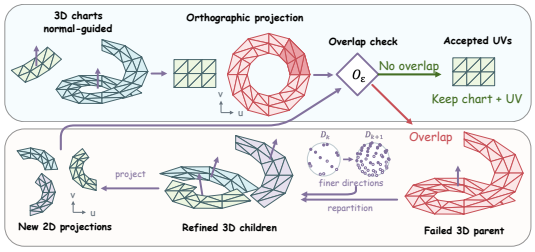

Figure 6: Progressive UV atlasing pipeline.

Our method uses face orientations and connectivity as geometric priors to construct charts for direct projection, then progressively refines regions with overlaps (Figure 6). Projection-Aligned Charting and Iterative Local Refinement work together to reduce repeated computation and unnecessary cuts.

Projection-Aligned Charting. A chart is a connected surface patch mapped separately to 2D. Since projection near a face normal causes less compression, our method groups faces with similar orientations around a shared projection direction. It starts from a small set of unit directions $\mathcal{D}$ and adapts them once to the mesh with a modified spherical k-means update (Hornik et al., 2012). Face areas provide the weights, while a small step and a rotation limit keep the directions close to their initial values. The update brings projection directions closer to the mesh's main surface orientations.

Each face f initially selects the direction closest to its unit normal $\mathbf{n}_f$ through $\ell_f = \arg\max_j \mathbf{n}_f^\top \mathbf{d}_j$, where $\mathbf{d}_j \in \mathcal{D}$. Faces with the same direction index may lie in disconnected regions, so our method joins adjacent faces across shared edges when their labels match and their normals satisfy alignment constraints. Each connected component forms a chart C. Given orthonormal axes $\mathbf{t}_C, \mathbf{b}_C$ perpendicular to its direction $\mathbf{d}_C$, a vertex $\mathbf{x}_v$ maps directly to

$$\mathbf{u}_v = \left(\mathbf{t}_C^\top \mathbf{x}_v,\, \mathbf{b}_C^\top \mathbf{x}_v\right). \tag{7}$$

This gives both chart membership and initial UVs without repeated grouping and projection trials.

Iterative Local Refinement. These initial projections may still overlap, so our method checks each chart and retains those that pass. Failed charts are subdivided by reusing the direction assignment and connectivity grouping with denser directions and tighter alignment constraints. This lets faces that shared one direction form smaller patches. The finer partition is applied only within failed charts, while passing charts keep their grouping and UVs throughout the refinement loop:

$$C \longrightarrow \begin{cases} C, & \text{passes the overlap check,} \\ \{C_i\}, & \text{overlap remains: subdivide.} \end{cases} \tag{8}$$

Each child C_i is projected from the original 3D vertices using Equation 7 and checked again, so different regions stop at different levels according to their projection results. A finite budget limits refinement; remaining failures use piecewise unfolding along shared edges to retain connected patches where possible. Finally, the charts are packed into a UV atlas.

Comparisons with existing UV methods show that our approach averages under 2 seconds for UV processing. Implementation details and additional comparison results are provided in Appendix C.

4 EXPERIMENTS

Setup. We evaluate Swift3D on geometry reconstruction and image-conditioned geometry, texture, and end-to-end generation. The three-stage reference pipeline (Ours) combines LATTICE (Lai et al., 2026a) for coarse geometry generation and refinement with NaTex (Lai et al., 2026b) for texture generation. Ours-Turbo denotes this pipeline accelerated by Swift3D and refers to the default 47-second configuration unless otherwise specified. Our generation pipeline uses a target geometry decoding resolution of 1536. Texture comparisons use input meshes with a 500K-face budget and 2048×2048 texture maps. Runtimes are measured on GPUs with approximately 140 TFLOPS of peak dense BF16 throughput per device (comparable to an NVIDIA H20) and reported in seconds. Detailed evaluation and timing protocols are provided in Appendix D.

Comparison and Metrics. We compare two groups of open-source 3D generators: *fast methods*, including TripoSR (Tochilkin et al., 2024), SPAR3D (Huang et al., 2025), and Hunyuan3D-2 Turbo (Lai et al., 2025); and *high-quality methods*, including Hunyuan3D-2.1 (Team Hunyuan3D et al., 2025), TRELLIS.2 (Xiang et al., 2026), and UltraShape 1.0 (Jia et al., 2025). For reconstruction, we compare with 3DShape2VecSet (Zhang et al., 2023), Dora (Chen et al., 2025), Michelangelo (Zhao et al., 2023), and Hunyuan3D-2.1. We include TEXGen (Yu et al., 2024) for texturing. We report mean runtime per asset (seconds). Reconstruction is evaluated using volume and surface IoU (V-IoU and S-IoU) and PSNR. CLIP-N, ULIP-I, and Uni3D-I measure geometry–image alignment. FID, c-FID, LPIPS, CMMD, and CLIP-I assess appearance quality and image alignment for texture and end-to-end generation. In Tables 1–4, bold and underlined entries mark the best and second-best results, excluding Ours (reference).

4.1 RECONSTRUCTION

Table 1: Geometry reconstruction on Toys4K at resolutions of 1024 and 1536.

	Resolution 1024				*Resolution 1536*			
Method	Time (s) ↓	V-IoU ↑	S-IoU ↑	PSNR ↑	Time (s) ↓	V-IoU ↑	S-IoU ↑	PSNR ↑
3DShape2VecSet	33.90	0.7239	0.2742	25.740	115.89	0.7274	0.2704	25.847
Dora	42.07	0.8434	0.7219	34.602	137.20	0.8474	0.7285	34.700
Michelangelo	<u>15.86</u>	0.8111	0.4553	29.172	<u>52.72</u>	0.8171	0.4673	29.413
Hunyuan3D-2.1	26.56	<u>0.8507</u>	<u>0.7777</u>	<u>36.479</u>	89.63	<u>0.8531</u>	<u>0.7847</u>	<u>36.824</u>
Ours-Turbo	**4.74**	**0.8556**	**0.7924**	**38.367**	**8.35**	**0.8584**	**0.7919**	**38.705**
Ours (reference)	20.89	0.8623	0.7989	38.583	61.13	0.8628	0.8010	38.911

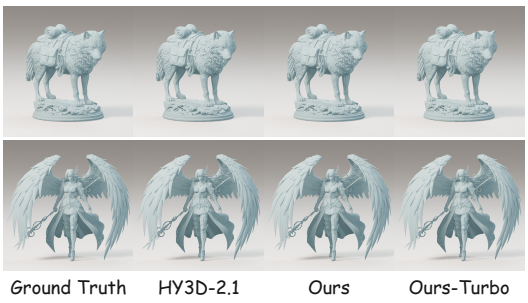

Figure 7: Qualitative comparison of geometry reconstruction.

We first evaluate geometry decoding on the Toys4K benchmark (Stojanov et al., 2021) at resolutions of 1024 and 1536. As shown in Table 1, Ours-Turbo is the fastest method and surpasses all external baselines in V-IoU, S-IoU, and PSNR at both resolutions. Relative to Ours, it achieves $4.41\times$ and $7.32\times$ speedups, respectively, with minimal changes in geometric overlap: V-IoU and S-IoU retain over 99% and 98% of the reference scores at both resolutions. The greater speedup at 1536 highlights the benefit of adaptive decoding at higher resolutions. Figure 7 shows that fine geometric structures, including fur and wing feathers, remain well resolved.

4.2 GENERATION

Geometry Generation. We evaluate geometry generation on Hi3DBench (Zhang et al., 2025). Ours-Turbo achieves an $11.84\times$ speedup (26.13 s) with less than 1.3% relative degradation per alignment metric (Table 2). It surpasses fast baselines in speed and alignment and high-quality baselines in ULIP-I and Uni3D-I.

Table 2: Image-conditioned geometry generation on Hi3DBench.

Method	Time (s) ↓	CLIP-N ↑	ULIP-I ↑	Uni3D-I ↑
TripoSR	37.74	0.6548	0.1727	0.1903
SPAR3D	<u>30.39</u>	0.6411	0.1722	0.2798
Hunyuan3D-2 Turbo	84.52	0.6844	0.2120	<u>0.3232</u>
Hunyuan3D-2.1	129.03	0.6926	<u>0.2123</u>	0.3211
TRELLIS.2	122.46	**0.7094**	0.2077	0.2318
UltraShape 1.0	572.20	0.6899	0.2070	0.3149
Ours-Turbo	**26.13**	<u>0.7032</u>	**0.2209**	**0.3287**
Ours (reference)	309.37	0.7057	0.2238	0.3313

Texture Generation. Following NaTex (Lai et al., 2026b) and MaterialMVP (He et al., 2025), we evaluate texture and end-to-end generation on 200 high-quality assets. With identical input meshes, Ours-Turbo is fastest at 23.63 s ($9.25\times$ speedup), with lower c-FID, LPIPS, and CMMD than all external baselines, FID comparable to TRELLIS.2, and CLIP-I close to Ours (Table 3).

Table 3: Texture generation using the same input meshes across methods.

Method	Time (s) ↓	FID ↓	c-FID ↓	LPIPS ↓	CMMD ↓	CLIP-I ↑
Hunyuan3D-2 Turbo	208.51	32.959	4.508	0.1719	0.1210	<u>0.7818</u>
TEXGen	261.86	105.595	21.341	0.3081	0.7460	0.7072
TRELLIS.2	<u>63.55</u>	**27.102**	<u>3.493</u>	<u>0.1456</u>	<u>0.0858</u>	0.7798
Hunyuan3D-2.1	235.54	30.515	3.774	0.1751	0.1216	0.7770
Ours-Turbo	**23.63**	<u>27.140</u>	**2.867**	**0.1396**	**0.0453**	**0.7844**
Ours (reference)	218.62	26.451	2.522	0.1354	0.0327	0.7853

End-to-End Generation. Ours-Turbo achieves a $10.98\times$ end-to-end speedup (47.39 s), largely preserving reference quality and outperforming all external baselines on the five quality metrics (Table 4). Figure 8 compares results with open-source methods, Hunyuan3D-3.1, and Tripo 3.1.

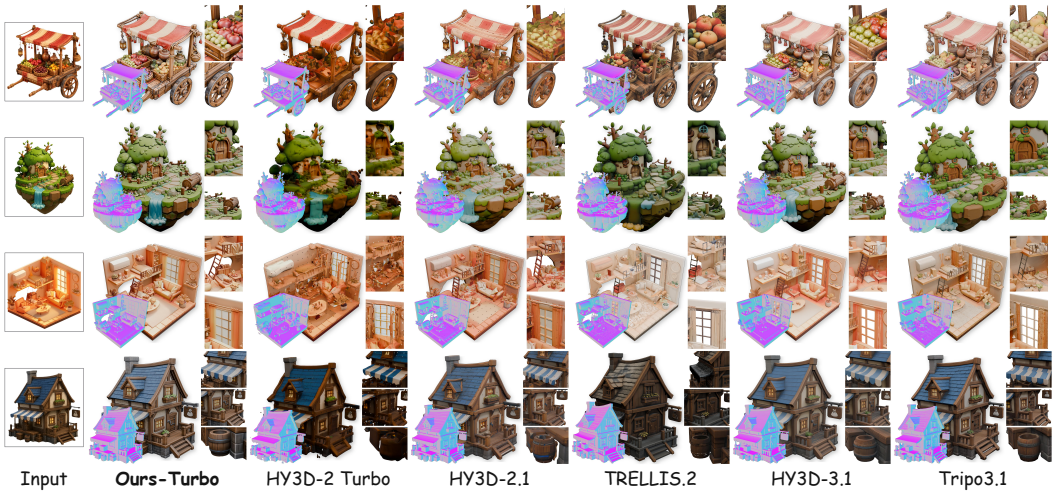

Figure 8: Qualitative comparison of end-to-end generation with open-source and commercial methods. Swift3D generates detailed geometry and textures in under 50 seconds on average.

Table 4: End-to-end generation of textured meshes from single images.

Method	Time (s) ↓	FID ↓	c-FID ↓	LPIPS ↓	CMMD ↓	CLIP-I ↑
Hunyuan3D-2 Turbo	324.80	147.719	15.497	0.3296	0.1957	0.7516
Hunyuan3D-2.1	378.44	134.376	14.195	0.3039	0.1614	0.7596
TRELLIS.2	187.24	133.669	13.658	0.3793	0.1034	0.7680
Ours-Turbo (47s)	47.39	**129.218**	**11.996**	**0.2936**	**0.0861**	**0.7762**
Ours-Turbo (27s)	**27.67**	130.083	13.016	0.2964	0.0923	0.7695
Ours (reference)	520.56	128.654	11.465	0.2993	0.0770	0.7786

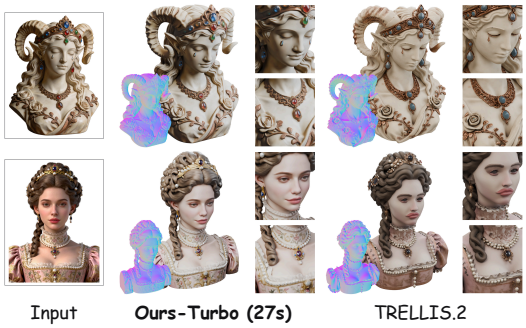

Figure 9: Qualitative comparison of Ours-Turbo (27s) with TRELLIS.2.

We further distill the geometry-refinement and texturing DiTs (Stages 2 and 3) to one step each (1 NFE). The resulting Ours-Turbo (27s) variant produces a textured mesh in 27.67 seconds on average. This yields a $6.77\times$ speedup over TRELLIS.2 while retaining asset quality comparable to high-quality open-source methods (Table 4). Figure 9 compares Ours-Turbo (27s) with TRELLIS.2, illustrating preserved facial features and fine geometric and texture details in these examples. Together, these results demonstrate detailed textured asset generation in under 30 seconds, even when refinement and texturing each use one sampling step.

4.3 Ablation Studies

We evaluate cumulative acceleration on the end-to-end evaluation data, with each row of Table 5 retaining the components added previously.

Geometry DiT distillation yields a $2.40\times$ speedup with largely preserved alignment (Table 5(a)). Adaptive decoding adds a further $5.28\times$ speedup, reducing runtime to 23.76 seconds, with similar CLIP-N and modest decreases in ULIP-I and Uni3D-I. Texture DiT distillation yields a $2.44\times$ speedup, keeping FID and CLIP-I close to the baseline (Table 5(b)). Progressive UV atlasing further accelerates texturing by $3.80\times$ to 23.63 seconds, with nearly unchanged quality.

Table 5: Ablation study of geometry and texture acceleration.

(a) Geometry acceleration	Time (s) ↓	CLIP-N ↑	ULIP-I ↑	Uni3D-I ↑
Geometry baseline	301.94	0.7239	0.2478	0.3303
+ Geometry DiT distillation	125.56	0.7200	0.2415	0.3239
+ Adaptive geometry decoding	23.76	0.7206	0.2329	0.3223

(b) Texture acceleration	Time (s) ↓	FID ↓	c-FID ↓	LPIPS ↓	CMMD ↓	CLIP-I ↑
Texture baseline	218.62	26.451	2.522	0.1354	0.0327	0.7853
+ Texture DiT distillation	89.68	27.080	2.850	0.1395	0.0452	0.7856
+ Progressive UV atlasing	23.63	27.140	2.867	0.1396	0.0453	0.7844

5 Conclusion

We presented Swift3D, which accelerates high-resolution textured 3D generation from single images by addressing shifting bottlenecks in DiT sampling, geometry decoding, and UV parameterization. Reconstruction and generation experiments show substantial acceleration while retaining high fidelity. Swift3D reduces average end-to-end latency from 520.56 to 47.39 seconds ($10.98\times$), outperforming the evaluated open-source baselines in asset quality. Distilling the geometry-refinement and texturing DiTs to one step each further enables generation in under 30 seconds with quality comparable to high-quality open-source methods.

REFERENCES

Mark Boss, Zixuan Huang, Aaryaman Vasishta, and Varun Jampani. SF3D: Stable fast 3D mesh reconstruction with UV-unwrapping and illumination disentanglement. *arXiv preprint arXiv:2408.00653*, 2024. doi: 10.48550/arXiv.2408.00653.

Rui Chen, Jianfeng Zhang, Yixun Liang, Guan Luo, Weiyu Li, Jiarui Liu, Xiu Li, Xiaoxiao Long, Jiashi Feng, and Ping Tan. Dora: Sampling and benchmarking for 3D shape variational auto-encoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16251–16261, 2025.

Zebin He, Mingxin Yang, Shuhui Yang, Yixuan Tang, Tao Wang, Kaihao Zhang, Guanying Chen, Yuhong Liu, Jie Jiang, Chunchao Guo, and Wenhan Luo. MaterialMVP: Illumination-invariant material generation via multi-view PBR diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 26294–26305, 2025.

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. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Kurt Hornik, Ingo Feinerer, Martin Kober, and Christian Buchta. Spherical k-means clustering. *Journal of Statistical Software*, 50(10):1–22, 2012. doi: 10.18637/jss.v050.i10.

Zixuan Huang, Mark Boss, Aaryaman Vasishta, James M. Rehg, and Varun Jampani. SPAR3D: Stable point-aware reconstruction of 3D objects from single images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16860–16870, 2025.

Sadeep Jayasumana, Srikumar Ramalingam, Andreas Veit, Daniel Glasner, Ayan Chakrabarti, and Sanjiv Kumar. Rethinking FID: Towards a better evaluation metric for image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9307–9315, 2024.

Tanghui Jia, Dongyu Yan, Dehao Hao, Yang Li, Kaiyi Zhang, Xianyi He, Lanjiong Li, Yuhan Wang, Jinnan Chen, Lutao Jiang, Qishen Yin, Long Quan, Ying-Cong Chen, and Li Yuan. UltraShape 1.0: High-fidelity 3D shape generation via scalable geometric refinement. *arXiv preprint arXiv:2512.21185*, 2025. doi: 10.48550/arXiv.2512.21185.

Zeqiang Lai, Yunfei Zhao, Zibo Zhao, Haolin Liu, Fuyun Wang, Huiwen Shi, Xianghui Yang, Qingxiang Lin, Jingwei Huang, Yuhong Liu, Jie Jiang, Chunchao Guo, and Xiangyu Yue. Unleashing VecSet diffusion model for fast shape generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2523–2533, 2025.

Zeqiang Lai, Yunfei Zhao, Zibo Zhao, Haolin Liu, Qingxiang Lin, Jingwei Huang, Chunchao Guo, and Xiangyu Yue. LATTICE: Democratize high-fidelity 3D generation at scale. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19982–19992, 2026a.

Zeqiang Lai, Yunfei Zhao, Zibo Zhao, Xin Yang, Xin Huang, Jingwei Huang, Xiangyu Yue, and Chunchao Guo. NaTex: Seamless texture generation as latent color diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18619–18629, 2026b.

Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. On aliased resizing and surprising subtleties in GAN evaluation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11410–11420, 2022.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. In *International Conference on Learning Representations*, 2023.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 2021.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 32211–32252. PMLR, 2023.

Stefan Stojanov, Anh Thai, and James M. Rehg. Using shape to categorize: Low-shot learning with an explicit shape bias. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1798–1808, 2021.

Team Hunyuan3D, Shuhui Yang, Mingxin Yang, Yifei Feng, Xin Huang, Sheng Zhang, Zebin He, Di Luo, Haolin Liu, Yunfei Zhao, Qingxiang Lin, Zeqiang Lai, Xianghui Yang, Huiwen Shi, Zibo Zhao, Bowen Zhang, Hongyu Yan, Lifu Wang, Sicong Liu, Jihong Zhang, Meng Chen, Liang Dong, Yiwen Jia, Yulin Cai, Jiaao Yu, Yixuan Tang, Dongyuan Guo, Junlin Yu, Hao Zhang, Zheng Ye, Peng He, Runzhou Wu, Shida Wei, Chao Zhang, Yonghao Tan, Yifu Sun, Lin Niu, Shirui Huang, Bojian Zheng, Shu Liu, Shilin Chen, Xiang Yuan, Xiaofeng Yang, Kai Liu, Jianchen Zhu, Peng Chen, Tian Liu, Di Wang, Yuhong Liu, Linus, Jie Jiang, Jingwei Huang, and Chunchao Guo. Hunyuan3D 2.1: From images to high-fidelity 3D assets with production-ready PBR material. *arXiv preprint arXiv:2506.15442*, 2025. doi: 10.48550/arXiv.2506.15442.

Dmitry Tochilkin, David Pankratz, Zexiang Liu, Zixuan Huang, Adam Letts, Yangguang Li, Ding Liang, Christian Laforte, Varun Jampani, and Yan-Pei Cao. TripoSR: Fast 3D object reconstruction from a single image. *arXiv preprint arXiv:2403.02151*, 2024. doi: 10.48550/arXiv.2403.02151.

Fu-Yun Wang, Zhaoyang Huang, Alexander William Bergman, Dazhong Shen, Peng Gao, Michael Lingelbach, Keqiang Sun, Weikang Bian, Guanglu Song, Yu Liu, Xiaogang Wang, and Hongsheng Li. Phased consistency models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2668.

Zhaoning Wang, Xinyue Wei, Ruoxi Shi, Xiaoshuai Zhang, Hao Su, and Minghua Liu. PartUV: Part-based UV unwrapping of 3D meshes. In *SIGGRAPH Asia 2025 Conference Papers*, 2025. doi: 10.1145/3757377.3763843.

Jianfeng Xiang. CuMesh: High-performance geometry processing for PyTorch. `https://github.com/JeffreyXiang/CuMesh`, 2025.

Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, and Jiaolong Yang. Native and compact structured latents for 3D generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14419–14429, 2026.

Le Xue, Ning Yu, Shu Zhang, Artemis Panagopoulou, Junnan Li, Roberto Martín-Martín, Jiajun Wu, Caiming Xiong, Ran Xu, Juan Carlos Niebles, and Silvio Savarese. ULIP-2: Towards scalable multimodal pre-training for 3D understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 27091–27101, 2024.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Frédo Durand, and William T. Freeman. Improved distribution matching distillation for fast image synthesis. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1505.

Tianwei Yin, Qiang Zhang, Richard Zhang, William T. Freeman, Frédo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast autoregressive video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22963–22974, 2025.

Jonathan Young. xatlas. `https://github.com/jpcy/xatlas`, 2019.

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*, 43(6):213:1–213:14, 2024. doi: 10.1145/3687909.

Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3DShape2VecSet: A 3D shape representation for neural fields and generative diffusion models. *ACM Transactions on Graphics*, 42(4), 2023. doi: 10.1145/3592442.

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*, pp. 586–595, 2018.

Yuhan Zhang, Long Zhuo, Ziyang Chu, Tong Wu, Zhibing Li, Liang Pan, Dahua Lin, and Ziwei Liu. Hi3DEval: Advancing 3D generation evaluation with hierarchical validity. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1564.

Min Zhao, Hongzhou Zhu, Kaiwen Zheng, Zihan Zhou, Bokai Yan, Xinyuan Li, Xiao Yang, Chongxuan Li, and Jun Zhu. Causal Forcing++: Scalable few-step autoregressive diffusion distillation for real-time interactive video generation. *arXiv preprint arXiv:2605.15141*, 2026. doi: 10.48550/arXiv.2605.15141.

Zibo Zhao, Wen Liu, Xin Chen, Xianfang Zeng, Rui Wang, Pei Cheng, Bin Fu, Tao Chen, Gang Yu, and Shenghua Gao. Michelangelo: Conditional 3D shape generation based on shape-image-text aligned latent representation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-3236.

Junsheng Zhou, Jinsheng Wang, Baorui Ma, Yu-Shen Liu, Tiejun Huang, and Xinlong Wang. Uni3D: Exploring unified 3D representation at scale. In *International Conference on Learning Representations*, 2024.

Zanwei Zhou, Taoran Yi, Jiemin Fang, Chen Yang, Lingxi Xie, Xinggang Wang, Wei Shen, and Qi Tian. Few-step flow for 3D generation via marginal-data transport distillation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(16):13853–13861, 2026. doi: 10.1609/aaai.v40i16.38394.

A Details of Stage-Aware Distillation

We report the distillation setup for the main 47-second configuration. The 27-second variant targets one sampling step for both geometry refinement and texturing.

A.1 Training Inputs

Geometry. Each sample pairs one conditioning view with 65,536 random surface points and 65,536 sharp-feature points. Coarse geometry uses 768×768 images and 4,096 latent tokens; geometry refinement uses 756×756 images and 16,384 tokens. Both latent spaces have 64 channels.

Texture. Each sample uses one of 42 conditioning views at 1024×1024, 2048×2048 UV-space supervision, and 128-channel latents with aligned geometry features as spatial conditions. All VAEs and image encoders remain frozen during distillation.

A.2 Distillation Training

Each DiT receives 200 optimization steps of PCM initialization (Wang et al., 2024), followed by DMD2 and adversarial refinement (Yin et al., 2024). Geometry and texture distillation require approximately 256 and 768 GPU-hours, respectively. Distillation is performed without gradient accumulation, with fully sharded data parallelism used for texture training. Table 6 separates the settings for PCM initialization and subsequent refinement. Global batch sizes are shared across the two training phases. Teacher anchoring is applied only to coarse geometry during refinement.

Table 6: Training settings for the three DiTs, grouped by distillation phase. Each PCM initialization runs for 200 optimization steps.

Setting	Coarse geometry	Geometry refinement	Texture
Global batch size	32	32	64
PCM initialization			
Teacher schedule steps	50	50	50
Teacher CFG scale	5	5	3.5
Smooth Huber δ	10^{-3}	10^{-3}	10^{-3}
AdamW learning rate	10^{-5}	10^{-5}	10^{-5}
AdamW (β_1, β_2)	$(0.9, 0.95)$	$(0.9, 0.95)$	$(0.9, 0.95)$
Weight decay	0.05	0.05	0.05
DMD2 and adversarial refinement			
Teacher CFG scale	5	5	3.5
Generator / fake-score LR	10^{-5}	10^{-6}	10^{-6}
Discriminator learning rate	10^{-5}	10^{-6}	10^{-5}
AdamW (β_1, β_2)	$(0.9, 0.95)$	$(0.9, 0.99)$	$(0.9, 0.95)$
Weight decay	0.05	0.01	0.05
Generator update period m	5	5	8
Adversarial weight	10^{-2}	5×10^{-3}	5×10^{-3}
Teacher-anchor weight	1.0	0	0

During PCM initialization, the student is updated at every iteration. During refinement, the generator is updated once every m iterations, with fake-score and discriminator updates in the remaining $m-1$ iterations, where m is the period listed in the table.

A.3 Teacher Anchoring

Structural drift can arise during coarse-geometry distillation. In the illustrated example, the unanchored result contains intersecting table frames and displaced balls, whereas the anchored result more closely preserves the teacher's structure and spatial layout (Figure 10).

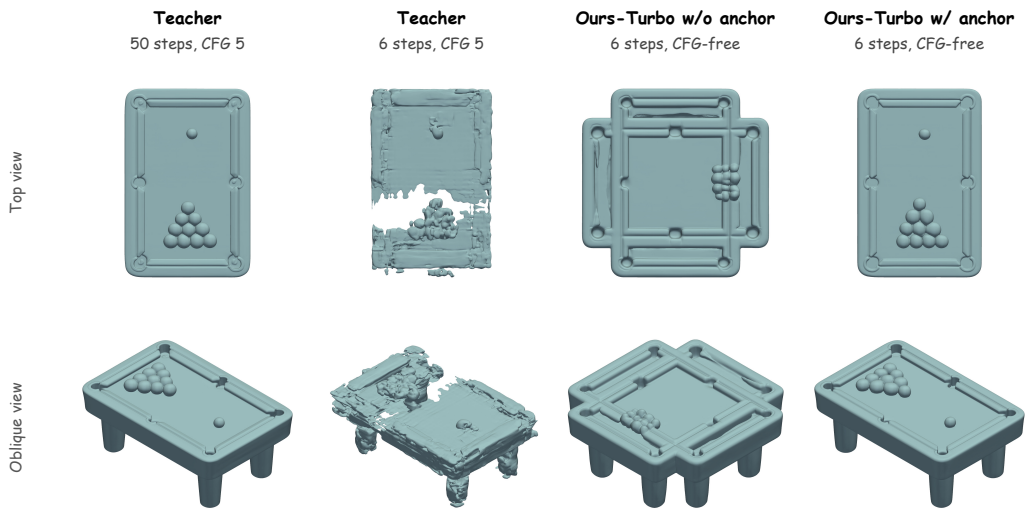

Figure 10: **Coarse-geometry structure and layout under distillation.** The columns show the high-quality teacher, the reduced-step teacher, and selected distilled results without and with teacher anchoring. Top and oblique views expose missing geometry in the reduced-step teacher and intersecting structures in the unanchored result. The anchored example more closely preserves the reference layout. All four use the same input, seed 42, 4,096 latents, image resolution 768×768, and extraction resolution 512. Cameras, shading, crop bounds, and display scales are shared.

A.4 Analysis of Two-Phase Distillation

Figure 11 examines the two-phase design in Section 3.1 through texture generation. High-resolution 3D appearance combines broad color patterns with fine local detail. Our two-phase scheme first establishes few-step generation through PCM, providing a starting point for subsequent distributional and adversarial refinement. PCM only corresponds to Phase 1; w/o PCM applies Phase 2 without PCM initialization; Ours-Turbo completes both phases.

Reducing the teacher from 50 to four steps noticeably simplifies the painted patterns and tassel details. PCM only produces coherent color regions and ornament contours, but the eye region has a flatter appearance. The w/o PCM result captures the overall ornamentation, while its eye contours and tassel details remain softer than those of Ours-Turbo. Our four-step result combines well-defined eye contours with distinct tassel strands and richer local appearance. These observations are consistent with the complementary roles of the two phases: PCM supplies a few-step initialization, and subsequent distributional and adversarial refinement improves local appearance beyond this initialization.



Figure 11: **Qualitative results of two-phase texture distillation.** From left: teacher (50 steps), teacher (4 steps), w/o PCM, PCM only, and Ours-Turbo. All students use four sampling steps. PCM only is the Phase 1 checkpoint; Ours-Turbo completes both phases. Phase 2 combines distribution matching and adversarial training; w/o PCM omits PCM initialization. Orange and blue boxes mark the eye ornament and chest tassels. Input mesh, UV layout, seed, rendering, and crops are shared. All configurations use 16,384 latent tokens for inference.

B Details of Consistent Field Reconstruction

Sources of field inconsistency. Hierarchical reconstruction requires field consistency at both reused sample locations and coarse–fine interfaces. FlashVDM (Lai et al., 2025) accelerates field evaluation by selecting latent key–value tokens from aggregate scores over a local query group. Changing the group's members can change the selected tokens and hence the prediction at a fixed position. During hierarchical decoding, these query-context changes can shift refreshed corner predictions away from cached values. Mixing these inconsistent values across levels can introduce artificial field variations that appear as surface ripples. A second mismatch arises where refined cells meet regions that retain their coarse fields. Even when shared corner values agree, fine samples at face edge midpoints and centers need not match the neighboring coarse interpolant. These

boundary discrepancies can disrupt surface continuity, producing jagged or crack-like gaps in the extracted mesh. Coarse-corner anchoring corrects the context-induced corner shifts, while coarse–fine interface alignment reconciles the remaining boundary residuals with the coarse field.

Coarse-corner anchoring. Coarse-corner anchoring addresses query-context shifts by aligning refreshed predictions with the cached coarse field. Each selected cell receives a complete 27-point refresh: eight corners, twelve edge midpoints, six face centers, and the cell center. The repeated corner queries measure the prediction shifts between the coarse and fine query contexts. We interpolate the cached-minus-refreshed corner differences throughout the cell and add this correction to the refreshed samples. The corrected corners recover their cached values, while the other samples retain the fine residuals of the refreshed field, now expressed relative to the cached corner interpolant. This removes context-induced corner shifts without discarding the fine residual, addressing a source of surface ripples during hierarchical reconstruction. Once the refresh is available, the correction requires only interpolation and no additional decoder call. Figure 12 provides a qualitative comparison of surface ripples and their reduction.

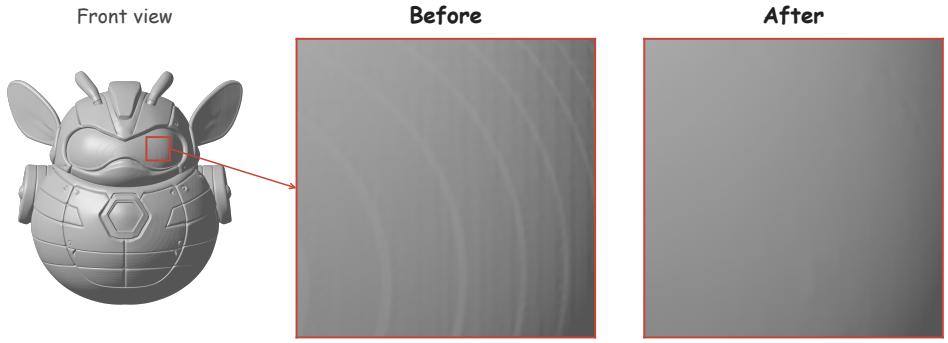

Figure 12: **Surface ripples before and after field correction.** The highlighted region shows pronounced ripples before correction and a smoother surface afterwards. Coarse-corner anchoring addresses the query-context mismatch described above by restoring cached corner values while retaining fine residuals.

Coarse–fine interface alignment. After corner anchoring, fine residuals at a face's edge midpoints and center can still make the refined face disagree with the neighboring coarse interpolant. We inspect the six face neighbors of each refined cell and replace samples on faces with unrefined neighbors by the parent interpolant. Faces between refined cells are not subject to this constraint, and interior residuals are retained. This operation follows corner anchoring and precedes classification of the next level's active cells. The final corrected samples and interpolation of stopped regions populate the extraction grid. Figure 13 provides a qualitative comparison of junction continuity.

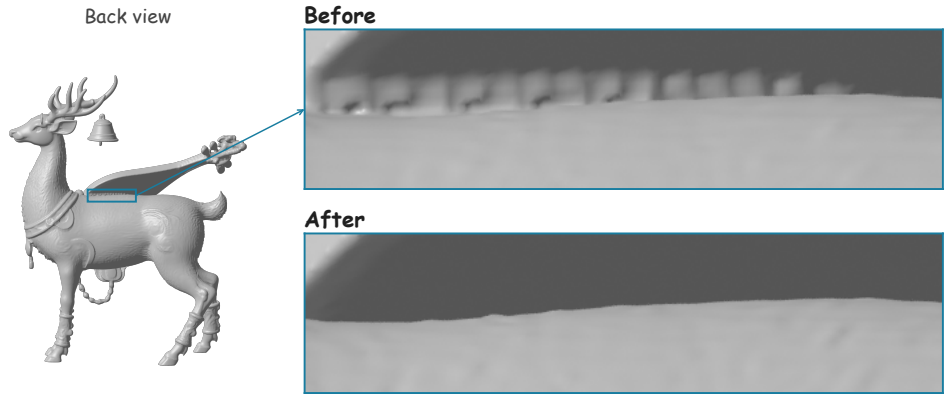

Figure 13: **Coarse–fine interface alignment.** The Before reconstruction exhibits repeated jagged, crack-like gaps along the junction, whereas the After result shows a smoother, more continuous connection. Coarse–fine interface alignment enforces field consistency at refinement boundaries by matching refined face samples to the neighboring coarse interpolant.

C Details of Progressive UV Atlasing

This appendix gives the implementation details of Section 3.3 and additional comparisons with xatlas, SF3D, and PartUV. Algorithm 1 summarizes chart construction, selective refinement, and atlas construction.

C.1 Algorithm and Implementation

Projection-aligned charting. Let $\mathcal{D}_k$ denote the unit projection directions at level k. ADAPT adjusts $\mathcal{D}_0$ to the mesh with one modified spherical k-means update (Hornik et al., 2012). Faces select the closest directions by their normals, and each direction receives a target given by the normalized, area-weighted sum of its assigned face normals. The target is blended with the original direction using weight α; the result is normalized and its rotation capped at θ. Directions without a valid target remain unchanged. Only $\mathcal{D}_0$ is adapted; finer levels use fixed directions.

At each level, initial direction labels and projected coordinates are computed as

$$\ell_f = \arg\max_j \mathbf{n}_f^\top \mathbf{d}_j, \qquad \mathbf{u}_v = \big(\mathbf{t}_C^\top \mathbf{x}_v, \mathbf{b}_C^\top \mathbf{x}_v\big), \tag{9}$$

where $\mathbf{t}_C, \mathbf{b}_C$ are orthonormal axes perpendicular to the direction of chart C. In CHART, labels are smoothed for s_k passes. A label changes only when at least two manifold neighbors agree and the face normal has a dot product of at least τ_k with the proposed direction. Adjacent faces are joined across shared edges when their labels match, both normals meet the threshold for their shared direction, and their mutual dot product is nonnegative. Faces with area at most 10^{-9} of the total mesh area remain singletons. Each connected component forms a chart with its faces and projection direction. PROJECT adds UVs from the original vertices; CHECK identifies charts with and without triangle overlaps.

Algorithm 1 Progressive UV Atlasing

Input: Mesh $\mathcal{M} = (V, F)$, atlas resolution R
Output: Atlas $\mathcal{A}$ and UV coordinates $\mathbf{U}$

▷ *Initialize projection-aligned charts*
$\mathcal{D}_0 \leftarrow \textsc{Adapt}(\mathcal{M}, \mathcal{D}_0, \alpha, \theta)$
$\mathcal{C} \leftarrow \textsc{Chart}(\mathcal{M}, \mathcal{D}_0, \tau_0, s_0)$
$\mathcal{P} \leftarrow \emptyset$
▷ *Keep passing charts; refine charts with overlaps*
for $k = 0, \ldots, L$ **do**
  $\mathcal{C} \leftarrow \textsc{Project}(\mathcal{C}, V)$
  $(\mathcal{G}, \mathcal{B}) \leftarrow \textsc{Check}(\mathcal{C})$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{G}$ ▷ Retain charts and their UVs
  if $\mathcal{B} = \emptyset$ **or** $k = L$ **then**
    break
  if $k > 0$ **and** $|\mathcal{B}| \geq N$ **and** $\sum_{C \in \mathcal{B}} |C| \geq \rho |F|$ **then**
    break ▷ Switch to piecewise unfolding
  $\mathcal{Q} \leftarrow \textsc{Chart}(\mathcal{M}, \mathcal{D}_{k+1}, \tau_{k+1}, s_{k+1})$
  $\mathcal{C} \leftarrow \textsc{Split}(\mathcal{B}, \mathcal{Q})$ ▷ Only regions with overlaps continue
▷ *Finish residual regions and pack*
$\mathcal{P} \leftarrow \mathcal{P} \cup \textsc{Unfold}(\mathcal{B})$
$\mathcal{P} \leftarrow \textsc{Merge}(\mathcal{P})$
return $\textsc{Pack}(\mathcal{P}, R, K)$

Iterative local refinement. At each finer level, CHART uses denser directions and a tighter alignment threshold to partition the whole mesh. SPLIT intersects these finer charts $\mathcal{Q}$ with overlapping parent charts $\mathcal{B}$. Each nonempty intersection inherits its direction from its source chart in $\mathcal{Q}$ and becomes a candidate for the next iteration. Only these candidates are rebuilt and reprojected. Passing charts accumulate in $\mathcal{P}$ and keep their grouping and UVs throughout the loop; input triangles are never subdivided. Refinement ends when all charts pass or the available levels are exhausted. After the first finer level, it also stops early for piecewise unfolding when at least N charts with overlaps cover a fraction ρ or more of the input faces. In the algorithm, $|C|$ counts the faces in chart C.

Residual regions and atlas construction. UNFOLD handles remaining charts with overlaps piecewise along shared edges. MERGE attempts local merges through reprojection, shared-edge alignment, and bounded conformal solves, accepting only those that pass overlap and quality checks. PACK packs the charts, checks float32 UVs and six-decimal serialization, and isolates affected faces for up to K repair and repacking attempts.

Parameter settings. We use $L = 2$ refinement levels, with direction counts $(|\mathcal{D}_0|, |\mathcal{D}_1|, |\mathcal{D}_2|) = (14, 26, 50)$, alignment thresholds $(\tau_0, \tau_1, \tau_2) = (0.78, 0.85, 0.90)$, and smoothing counts $(s_0, s_1, s_2) = (4, 1, 1)$. The initial directions are the six coordinate axes and eight cube diagonals. We set $\alpha = 0.25$, $\theta = 2^\circ$, $N = 64$, $\rho = 0.05$, and $K = 12$.

C.2 ADDITIONAL COMPARISONS

Experimental setup. We compare our method with xatlas (Young, 2019), SF3D (Boss et al., 2024), PartUV (Wang et al., 2025), and CuMesh (Xiang, 2025). We select 149 samples from the official Hunyuan3D and Tripo galleries and generate their geometry using the geometry settings of our 47-second Ours-Turbo configuration. We simplify input meshes to a target of 500K faces. Within each comparison, all methods receive identical geometry and use 2048×2048 textures, with each method retaining its native packer. UV time includes unwrapping and packing, but excludes model loading, texture generation, rendering, and export.

Metrics. UV islands are connected components across shared edges with matching positions and UVs. Area distortion is a scale-normalized area-ratio score (ideal value 1), evaluated on nondegenerate faces; the worst-chart score takes the maximum chart mean per case. Overlapping faces participate in positive-area UV intersections; boundary contacts are excluded. UV time, both distortion scores, and overlapping-face counts are averaged over the 149 samples.

Table 7: **Additional UV comparison.** Arrows indicate the preferred direction; bold marks the best value for each scored metric.

Metric	xatlas	SF3D	PartUV	CuMesh	Ours
UV time (s) ↓	279.89	2.50	300.61	36.96	**1.81**
UV islands	16,802	356,703	756	15,751	6,651
Overall area distortion ↓	1.0847	1.7778	1.1660	1.0866	**1.0404**
Worst-chart area distortion ↓	2.5888	5.0333	1.9915	2.9003	**1.1854**
Overlapping faces ↓	64.43	888.07	17.26	72.85	**0**

Results. We compare different UV methods by integrating each into our 3D asset generation pipeline. As shown in Table 7, our method achieves the lowest mean UV unwrapping and packing time of 1.81 s among the evaluated methods. It also achieves the lowest overall and worst-chart area-distortion scores (1.0404 and 1.1854, respectively), with no overlapping faces detected across all 149 evaluated samples. The close-ups in Figure 14 further illustrate the clean textured appearance produced by our method.

D DETAILS OF EXPERIMENTS

We describe the test data, metric computation, and runtime accounting used in our experiments. Each quality comparison uses a shared set of inputs and the same evaluation procedure for all methods.

D.1 EVALUATION PROTOCOL

D.1.1 GEOMETRY RECONSTRUCTION

Test data and setup. We evaluate geometry reconstruction on Toys4K (Stojanov et al., 2021), which contains objects with varied shapes and surface structures. Each method reconstructs the same reference meshes at requested decoding resolutions of 1024 and 1536. Predictions are evaluated in the reference coordinate system, with a shared normalization of the input assets. The evaluation grid and rendering settings are fixed across methods and decoding resolutions.

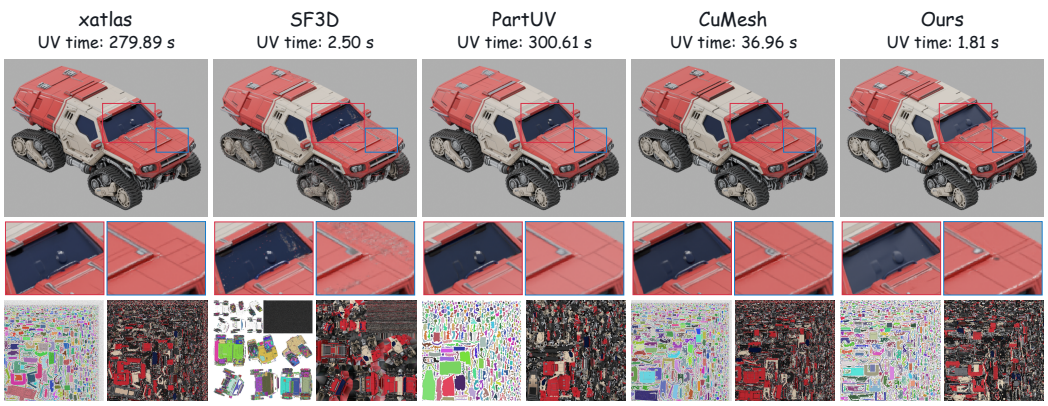

Figure 14: **Textured appearance and UV layouts.** Matched views and close-ups compare texture quality and visible artifacts across methods. Bottom panels show UV islands and base-color atlases. Ours denotes our progressive UV atlasing method.

Volume and surface overlap. We voxelize reference and reconstructed meshes on the same grid, using a core resolution of 380 over the unit cube and 48 padding voxels on each side. The voxel spacing is therefore $1/379$, independently of the decoding resolution. Let S and $\widehat{S}$ denote the sets of voxels occupied by the reference and reconstructed surfaces. We obtain the corresponding volume sets V and $\widehat{V}$ by filling enclosed interiors with six-connected voxel filling. We compute

$$\text{V-IoU} = \frac{|V \cap \widehat{V}|}{|V \cup \widehat{V}|}, \qquad \text{S-IoU} = \frac{|S \cap \widehat{S}|}{|S \cup \widehat{S}|}. \tag{10}$$

V-IoU measures volumetric agreement, while S-IoU measures agreement of the voxelized surfaces. We use the surface sets directly, without dilation. V-IoU is averaged over the common set of cases with valid volume filling across the compared methods; S-IoU is averaged over the shared reconstruction evaluation set. Higher scores indicate better agreement.

Normal-map fidelity. Following the rendering setup of TRELLIS.2 (Xiang et al., 2026), we compare camera-space normal maps from four fixed views. The camera radius is 10, elevation is $30°$, field of view is $6°$, and azimuths are $30°$, $120°$, $210°$, and $300°$. Normal vectors are mapped to RGB by $(\boldsymbol{n}+1)/2$ and rendered at 512×512 with $2\times$ supersampling and a gray background. For a reference normal map N and its reconstruction $\widehat{N}$, with RGB values in $[0,1]$, we compute

$$\text{PSNR}(N, \widehat{N}) = -10 \log_{10}\left(\frac{1}{3HW}\|N - \widehat{N}\|_F^2\right). \tag{11}$$

The mean squared error covers the full image. PSNR is first averaged over the four views of each asset and then across assets; higher is better.

D.1.2 Geometry Generation

Test data and setup. We evaluate image-conditioned geometry generation on Hi3DBench (Zhang et al., 2025). All methods receive the same input images. The evaluation measures the correspondence between each generated shape and its conditioning image using CLIP-N, ULIP-I, and Uni3D-I. The three scores capture alignment through rendered normals and native 3D features, respectively.

CLIP-N. We render four normal maps using the camera positions specified above, with a white background for CLIP evaluation. Meshes are centered and scaled to a unit cube, and fixed coordinate conversions account for the output conventions of each method. Let I_i be the conditioning image for asset i, N_{iv} its generated normal map at view v, and f the CLIP ViT-L/14 image encoder (Radford et al., 2021). With $c(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}/(\|\boldsymbol{a}\|_2\|\boldsymbol{b}\|_2)$, we report

$$\text{CLIP-N} = \frac{1}{4M}\sum_{i=1}^{M}\sum_{v=1}^{4} c\big(f(I_i), f(N_{iv})\big), \tag{12}$$

where M is the number of evaluated assets.

ULIP-I and Uni3D-I. We center each mesh and scale its longest bounding-box side to 1.96, sample 300K points uniformly by surface area, and apply farthest-point sampling to obtain 8,192 points. The point-sampling procedure and per-asset seeds are shared across methods. ULIP-I uses the ULIP-2 PointBERT model (Xue et al., 2024) with XYZ inputs. Uni3D-I uses Uni3D-g (Zhou et al., 2024) with its paired EVA02 image encoder; point colors are fixed to a constant gray so that the score evaluates geometry. For each model, we compute the cosine similarity between its point-cloud and conditioning-image embeddings and average over assets. Higher CLIP-N, ULIP-I, and Uni3D-I indicate stronger correspondence to the input image.

D.1.3 Texture and End-to-End Generation

Test data and setup. Following the benchmark design and evaluation protocols of NaTex (Lai et al., 2026b) and MaterialMVP (He et al., 2025), we use a benchmark of 200 high-quality 3D assets for texture and end-to-end generation. The benchmark covers varied object shapes and appearances; each asset provides a conditioning RGB image and a textured reference mesh. Texture generation uses the same input geometry and conditioning image for every method. End-to-end generation uses the conditioning image to produce both geometry and texture. Both tasks are evaluated against the corresponding reference assets.

Shared rendering protocol. We evaluate base-color appearance using albedo-only renders, following NaTex. Reference and generated assets are rendered with the same Blender setup at 512×512, with an orthographic camera and white background. Base color is rendered through an emission shader to avoid lighting-dependent shading. Each asset has 14 views: six axis-aligned views, four equatorial diagonal views, and four views at elevations of $\pm 45^\circ$ and azimuths of 0° or 180°. These images are shared by all five appearance metrics.

FID and c-FID. FID (Heusel et al., 2017) and c-FID compare the feature distributions of all generated renders and all reference renders. For feature means μ_g, μ_r and covariance matrices Σ_g, Σ_r, the Fréchet distance is

$$d_F = \|\mu_g - \mu_r\|_2^2 + \operatorname{tr}\left[\Sigma_g + \Sigma_r - 2\left(\Sigma_g^{1/2}\Sigma_r\Sigma_g^{1/2}\right)^{1/2}\right]. \tag{13}$$

FID uses Inception-v3 features with Clean-FID preprocessing (Parmar et al., 2022); c-FID uses CLIP ViT-B/32 image features (Radford et al., 2021). Both are computed once over the full collection of rendered views, and lower values indicate closer distributions.

CMMD. CMMD (Jayasumana et al., 2024) measures the maximum mean discrepancy between generated and reference renders in CLIP feature space. We use ℓ_2-normalized CLIP ViT-L/14@336 embeddings and a Gaussian kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|_2^2/(2\sigma^2))$ with $\sigma = 10$. For generated features $\{\boldsymbol{x}_a\}_{a=1}^n$ and reference features $\{\boldsymbol{y}_b\}_{b=1}^m$, we compute

$$\mathrm{CMMD} = 1000\left[\frac{1}{n^2}\sum_{a,a'} k(\boldsymbol{x}_a, \boldsymbol{x}_{a'}) + \frac{1}{m^2}\sum_{b,b'} k(\boldsymbol{y}_b, \boldsymbol{y}_{b'}) - \frac{2}{nm}\sum_{a,b} k(\boldsymbol{x}_a, \boldsymbol{y}_b)\right]. \tag{14}$$

As with FID and c-FID, the calculation pools views across the entire evaluation set. Lower CMMD indicates closer feature distributions.

LPIPS and CLIP-I. LPIPS (Zhang et al., 2018) measures perceptual differences between each generated render and the reference render of the same asset from the same camera. We use the AlexNet-based LPIPS model, which aggregates learned, weighted distances between normalized intermediate feature maps, with input images scaled to $[-1, 1]$. CLIP-I measures cosine similarity between each generated render and its conditioning image using CLIP ViT-B/32 (Radford et al., 2021). Both scores are averaged over the 14 views and then over assets. Lower LPIPS indicates better agreement with the reference appearance; higher CLIP-I indicates stronger alignment with the conditioning image. In end-to-end evaluation, rendered-image metrics reflect the combined effects of geometry and texture generation.

D.2 Inference Settings and Runtime Measurement

Comparison settings. Runtime benchmarks use GPUs with approximately 140 TFLOPS of peak dense BF16 throughput per device, comparable to an NVIDIA H20. We process one asset at a time per GPU and use BF16 precision for Swift3D training and neural inference. Compared methods share the input workload within each timing benchmark. Reconstruction is evaluated at both requested resolutions; the generation configuration requests geometry decoding at 1536. Texture timing uses meshes with a 500K-face budget and 2048×2048 texture maps, including each method's UV parameterization procedure. These shared settings control the principal workload differences that affect decoding and texturing cost. The default accelerated model uses 6, 6, and 4 sampling steps for coarse geometry, geometry refinement, and texturing, respectively. The faster variant uses one step for each of the latter two stages. Distillation settings are given in Appendix A.

Timing protocol. To support fair comparisons, methods are evaluated on the same GPU model and input workload within each timing benchmark, using the task-specific settings described above. We report mean per-asset computation time in seconds, excluding process startup, model loading, file export, and offline evaluation. For every method, recorded per-asset times are aggregated using the arithmetic mean, and speedups are calculated as ratios of the corresponding mean runtimes.

E More Results

E.1 Generalization

We further evaluate the transferability of adaptive geometry decoding and progressive UV atlasing beyond the model components used in the main experiments. Specifically, we apply our decoding method to the pretrained Hunyuan3D 2.1 VAE and evaluate our UV method on meshes generated by Hunyuan3D 2.1, without retuning either module.

E.1.1 Geometry Decoding

Experimental setup. To evaluate whether our geometry decoding method transfers beyond the backbone used in the main experiments, we apply it to the pretrained Hunyuan3D 2.1 VAE (Team Hunyuan3D et al., 2025), without retraining or retuning the adaptive decoding parameters. We retain its encoder, model weights, and latent representation, and transfer the adaptive query allocation and field reconstruction of Section 3.2. We evaluate on the same Toys4K (Stojanov et al., 2021) dataset as in the main experiments, at a decoding resolution of 1024.

Table 8: Geometry decoding generalization to Hunyuan3D 2.1 at resolution 1024.

Method	Time (s) ↓	V-IoU ↑	S-IoU ↑	PSNR ↑
Hunyuan3D-2.1	26.56	0.8507	0.7777	36.479
Hunyuan3D-2.1 + Ours	4.95	0.8485	0.7721	36.432

Results. As shown in Table 8, our method reduces the decoding time of Hunyuan3D 2.1 from 26.56 to 4.95 s while maintaining comparable reconstruction quality. The relative reductions in V-IoU and S-IoU are only 0.26% and 0.72%, respectively, while normal-map PSNR drops by 0.047 dB. These results demonstrate the transferability of our geometry decoding method to Hunyuan3D 2.1.

E.1.2 UV Atlasing

Experimental setup. To evaluate the generalization of progressive UV atlasing across geometry generators, we reuse cases from the evaluation set in Appendix C.2, drawn from the official Hunyuan3D and Tripo galleries. For the same input images, we apply our UV method to geometry generated by Hunyuan3D 2.1 (Team Hunyuan3D et al., 2025). Both methods use input meshes simplified to a target of 500,000 faces and 2048×2048 UV atlases. Starting from the original full-resolution outputs, we exactly weld duplicate vertex positions before simplification. We remove zero-area faces from the shared input and inspect its topology. Each mesh has one connected component and no open boundary edges. On identical geometry, we compare our method with the original

UV method used by Hunyuan3D 2.1, xatlas. We retain the UV implementations, parameters, timing boundaries, and evaluation procedures of Appendix C.2, without retuning the UV module. Both methods are measured on the same host.

Table 9: **UV generalization to Hunyuan3D 2.1 geometry.** Metrics follow Appendix C.2 and are averaged over the evaluated meshes, including island counts. Arrows indicate the preferred direction; bold marks the best value for each scored metric.

Metric	xatlas	Ours
UV time (s) ↓	250.37	**2.56**
UV islands	10,258.5	3,724.5
Overall area distortion ↓	1.0523	**1.0293**
Worst-chart area distortion ↓	1.4111	**1.0876**
Overlapping faces ↓	96.00	**0**

Results. Table 9 shows that our method reduces mean UV time from $250.37\,\mathrm{s}$ to $2.56\,\mathrm{s}$. It also produces fewer UV islands, lower area distortion, and no detected overlapping faces. These generalization results on geometry generated by Hunyuan3D 2.1 demonstrate that our UV acceleration transfers to this architecture's outputs without parameter retuning, retaining fast UV atlas construction and low area distortion.

E.2 Additional End-to-End Results

We supplement the qualitative results in the main text with additional end-to-end examples from our 47-second configuration (Figures 15 and 16) and 27-second variant (Figure 17). Each example shows the input image, the generated geometry, and the textured asset.



Figure 15: **Additional end-to-end qualitative results.** Each example shows the input image (upper left), generated geometry (lower left), and textured asset (right).

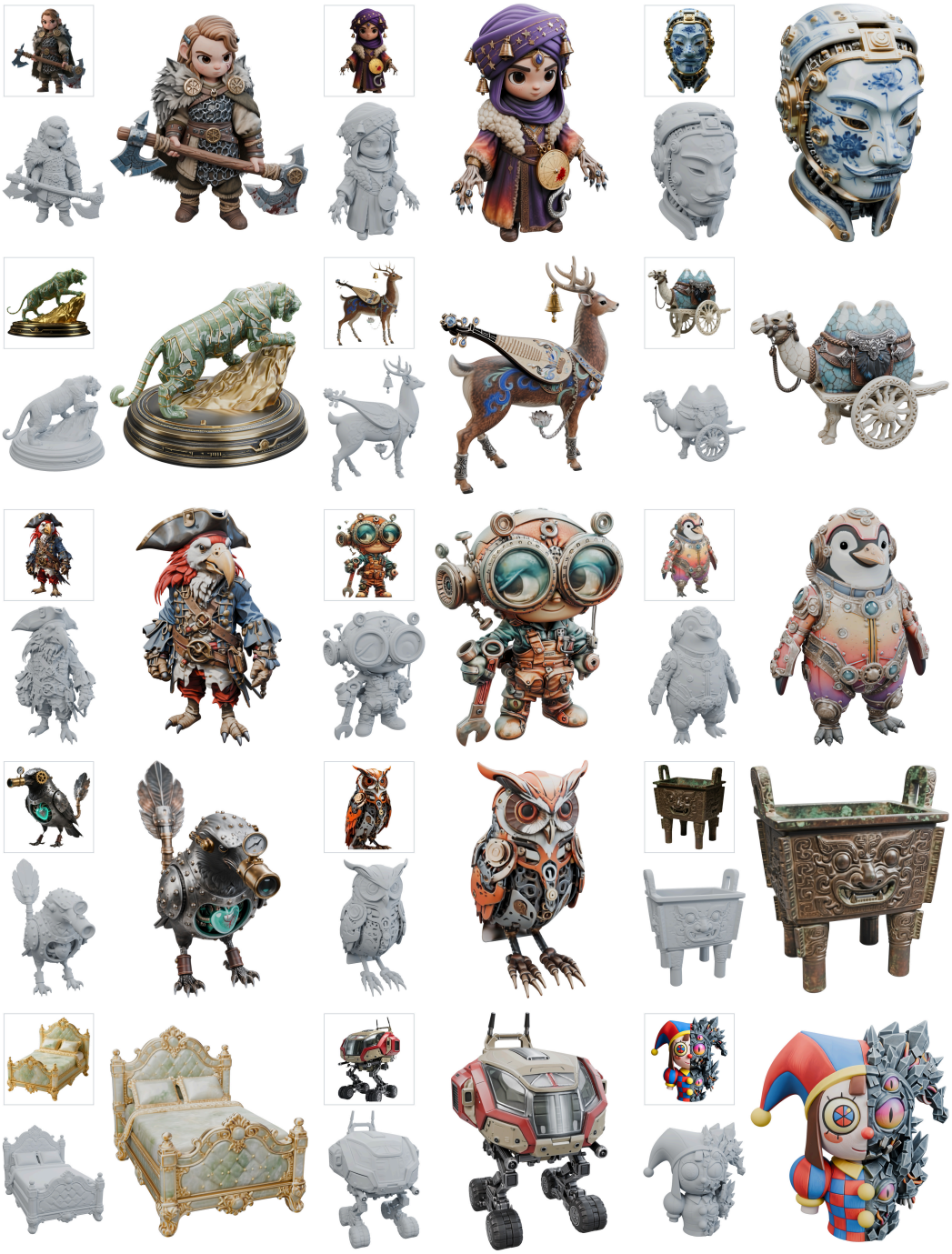

Figure 16: **Additional end-to-end qualitative results (continued).** The layout follows Figure 15.

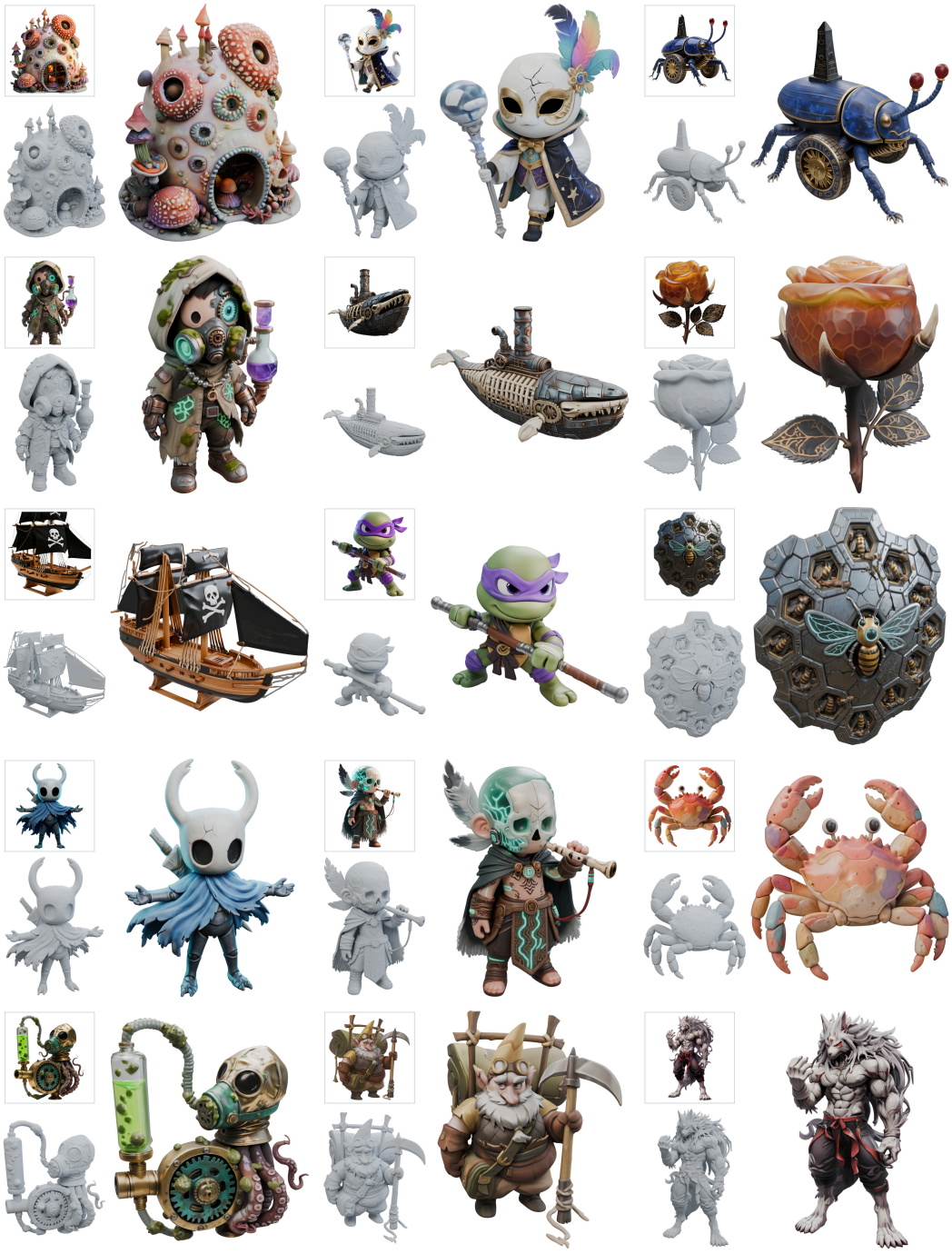

Figure 17: **Additional end-to-end qualitative results.** Each example shows the input image (upper left), generated geometry (lower left), and textured asset (right).

E.3 Additional Applications

Swift3D assets can also serve as visual references for video generation. We use rendered views of the generated textured meshes to synthesize scene keyframes, which are then animated with an image-to-video model. Figure 18 pairs three assets with sampled frames from the resulting videos. The supplementary material provides the corresponding video demonstrations.

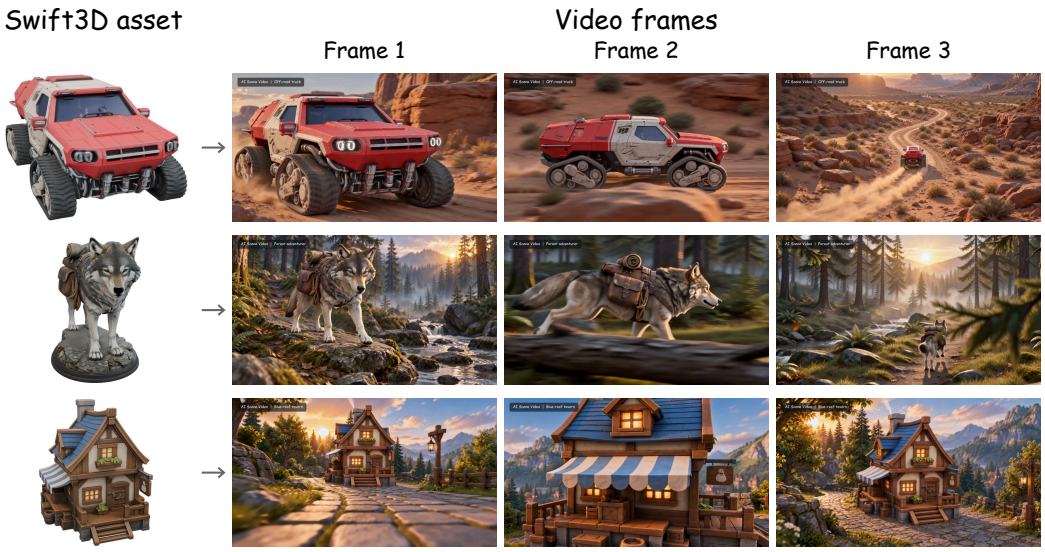

Figure 18: **3D assets for video generation.** Each row pairs a rendered Swift3D asset (left) with three frames from its corresponding video (right), shown in temporal order.

F Limitations and Future Work

Swift3D substantially accelerates end-to-end 3D asset generation, but opportunities for further optimization remain. For UV parameterization, future work could further improve the regularity of UV layouts while maintaining fast atlas construction. For diffusion sampling, extending one-step generation to coarse geometry is a promising direction, although preserving global structure remains challenging.

G AI Use Statement

LLMs were used to assist with translation, language polishing, and LaTeX formatting of this manuscript. This assistance included rephrasing sentences, suggesting alternative wording, correcting grammatical errors, and improving clarity and readability. The authors take responsibility for the final content.