2D-3D Style Transfer for Reusable Assets

Khang Tran
Princeton University
khang.tran@princeton.edu

Sakshyam Pokharel
Princeton University
sakshyam@princeton.edu

Abstract

While stylized graphics are becoming increasingly fashionable in modern media, current stylization techniques are rarely optimized for creating reusable 3D assets. To address this, we present a comprehensive 2D-to-3D style transfer pipeline that transforms a standard 3D mesh using a single style image to produce a fully stylized, reusable mesh. Our method begins with Multiview-consistent Style Transfer (MVST), utilizing ConsisLoRA alongside Depth and Canny ControlNets to stylize rendered views, while ensuring multiview consistency via nearest-neighbor seam healing, histogram matching, and RePaint inpainting. These stylized views are then backprojected into a UV map to initialize physically based rendering (PBR) textures. We employ a differentiable rendering framework with an ADAM optimizer to generate accurate normal, roughness, and albedo maps, governed by style, total variation, and color losses. Finally, to resolve reprojection artifacts and missing data, we integrate a Stable Diffusion inpainting step using RunwayML, an IP-Adapter, and Paint3D ControlNet, culminating in high-fidelity, production-ready stylized meshes. https://github.com/khangtranduc/23st

1. Introduction

2D image stylization has seen massive leaps in quality and accessibility, extending these capabilities into the 3D has introduced significant challenges. Currently, there is a notable lack of robust stylization methods specifically targeted at creating reusable 3D assets. Recent explorations into 3D stylization, such as StyleGaussian and techniques for learning to stylize novel views, have made considerable strides in novel view synthesis. However, producing a standard mesh with physically based rendering (PBR) textures that consistently retains a specific artistic style across all unconstrained camera angles remains a significant hurdle for downstream asset integration. To bridge this gap, we present a novel 2D-to-3D style transfer pipeline designed to transform a standard, untextured 3D mesh into a stylized, production-ready asset using only a single reference style image. Our methodology overcomes the challenges of multiview consistency and texture baking through a multi-stage approach. First, we employ a Multiview-consistent Style Transfer (MVST) technique to stylize individual rendered views of the mesh. These views are subsequently backprojected into a global UV space. To translate these discrete stylized views into cohesive, reusable material properties, we utilize a differentiable rendering framework to optimize PBR textures—specifically normal, roughness, and albedo maps. Finally, because backprojection inherently leaves unobserved or poorly sampled regions on the UV map, we apply a Stable Diffusion-based inpainting (based on Paint3D) step to heal seams and complete the texture. The resulting output is a high-fidelity, stylized mesh that can be seamlessly dropped into traditional rendering engines and animation pipelines.

2. Related Work

2.1. 3D Style Transfer and Representation

A foundational work in texturing explicit geometry is StyleMesh [2], which jointly optimizes an explicit RGB texture over a reconstructed 3D scene using a single reference image. By incorporating depth and surface normal data into a multi-view Neural Style Transfer (NST) optimization process, StyleMesh mitigates view-dependent artifacts and ensures that the final output can be rendered in real-time using traditional graphics pipelines. While our approach shares the ultimate goal of producing a standard, engine-ready 3D asset, we advance beyond traditional VGG-based NST. We incorporate powerful generative diffusion priors (via ConsisLoRA and RePaint) to capture far more complex semantic styles, and we employ differentiable rendering to bake full physically based rendering (PBR) material maps—including normal, roughness, and albedo—rather than relying on a flat RGB texture.

Recent advancements in novel view synthesis, particularly 3D Gaussian Splatting (3DGS), have opened new avenues for 3D stylization with one of them being StyleGaussian [4]. StyleGaussian enables instant, zero-shot 3D style transfer at high frame rates while preserving strict

multi-view consistency. StyleGaussian achieves this by embedding 2D VGG scene features into reconstructed 3D Gaussians, transforming these features according to a reference style image, and decoding them into stylized RGB using a novel K-nearest-neighbor-based 3D CNN. While StyleGaussian achieves real-time rendering capabilities, the resulting assets are represented exclusively as Gaussian splats. As we explore later, reconstructing standard, editable geometry from 3DGS representations remains highly challenging. Our work diverges from StyleGaussian by explicitly targeting the generation of standard 3D meshes with reusable PBR textures.

2.2. Diffusion Models and Consistent Stylization

Denoising Diffusion Probabilistic Models (DDPMs) and Latent Diffusion Models such as SDXL, 2D image stylization has reached unprecedented levels. LDMs achieve this by utilizing an autoencoder to compress images into a lower-dimensional latent space, applying the diffusion process there, and utilizing cross-attention mechanisms for complex conditioning. To adapt these powerful 2D priors for consistent generation across multiple views, our pipeline relies on ConsisLoRA [1]. Standard diffusion parameterization typically learns to predict noise (ϵ-prediction), which can lead to content leakage and style misalignment during style transfer. ConsisLoRA mitigates this by optimizing the Low-Rank Adaptation (LoRA) weights to predict the original image (x_0-prediction). Furthermore, it employs a two-step training strategy with a stepwise loss transition to decouple the learning of global structure and local stylistic details from the reference image.

2.3. Image Inpainting and Texture Completion

Because projecting 2D stylized views back onto 3D geometry inherently leaves unobserved regions due to occlusions, sophisticated inpainting is a critical component of 3D asset generation. We leverage RePaint, which introduces a robust method for free-form inpainting using pre-trained, unconditional DDPMs. Instead of requiring mask-specific training, RePaint alters the reverse diffusion iterations by sampling the known (unmasked) regions directly from the forward-noised original image and combining them with the model's prediction for the unknown regions. To harmonize boundaries, it employs a resampling strategy that repeatedly moves forward and backward in diffusion time.

To finalize the global UV texture atlas, we utilize techniques inspired by Paint3D [8]. Paint3D generates high-resolution, lighting-less 2K UV texture maps by employing a coarse-to-fine generative framework. Crucially, to resolve reprojection artifacts in UV space while maintaining geometric fidelity, it utilizes a shape-aware UV inpainting diffusion model conditioned heavily on a position map (via a position encoder) rather than relying solely on 2D image priors that suffer from baked-in illumination bias.

3. Basic Theory

3.1. Denoising Diffusion Probabilistic Models (DDPMs

DDPMs learn a data distribution by reversing a Markovian forward process that gradually adds Gaussian noise to an image. Given an initial data point $x_0 \sim q(x)$, the forward process over T steps is defined by a variance schedule β_t:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

The model learns to reverse this process by predicting the parameters of a Gaussian distribution $\mu_\theta(x_t, t)$ and $\Sigma_\theta(x_t, t)$. In Latent Diffusion Models (such as SDXL), this entire Markov chain operates within a perceptually compressed latent space z, drastically reducing the computational bottleneck of pixel-space diffusion while maintaining high-frequency details via the decoder.

3.2. LoRA Tuning and ControlNets

Fine-tuning large-scale LDMs directly is computationally prohibitive. Low-Rank Adaptation (LoRA) addresses this by freezing the pre-trained model weights $W \in \mathbb{R}^{d\times k}$ and injecting trainable rank decomposition matrices. The weight update is parameterized as $\Delta W = BA$, where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, and the rank $r \ll \min(d, k)$. This allows for the efficient injection of specific styles or subjects without catastrophic forgetting.To provide spatial guidance during generation, ControlNets create a trainable copy of the LDM's encoding layers, locking the original weights. This trainable copy processes task-specific spatial conditions (such as depth maps or Canny edge maps) and adds the resulting feature maps back to the locked UNet decoder via zero-convolutions. This ensures the structural integrity of the 3D mesh's perspective is perfectly preserved during the multi-view stylization process.

3.3. Birefnet

Accurate foreground extraction is critical in our pipeline to prevent background pixels from bleeding into our view-warping and UV backprojection steps. For this, we employ the Bilateral Reference Network (BiRefNet), a framework explicitly designed for high-resolution dichotomous image segmentation (DIS). Standard segmentation models often resize input images to lower resolutions during processing, which destroys the high-frequency, non-salient details necessary for crisp boundary matting.

To overcome this, BiRefNet purposefully decomposes the segmentation task into two distinct modules: a Localization Module (LM) and a Reconstruction Module (RM). The LM utilizes a vision transformer backbone to extract hierarchical semantic features, squeezing them to obtain

coarse object localization. The RM subsequently refines these coarse predictions using a novel Bilateral Reference (BiRef) mechanism, which operates via a dual-reference approach. First, an inward reference feeds patches of the original, un-resized source image directly into the decoder stages, ensuring intact spatial details are preserved and bypassing the resolution bottleneck. Second, an outward reference introduces explicit gradient map supervision. By computing derivative operations on the original image, this outward reference forces the network's attention toward areas with rich, complex edge information. This architecture allows for the extraction of highly precise foreground masks, accommodating the complex, slender geometric structures often found in our rendered meshes.

4. Methodology

Our pipeline takes a single style image and a 3D mesh as inputs to produce a fully stylized mesh. We first render multiple views of the mesh and then stylize those views in a multiview-consistent manner. After that, we backproject those stylized views onto the original mesh to obtain a stylized albedo map. Finally, we use a differentiable renderer to tune the normal and roughness map to further reduce style loss. Further details on how we incorporate style loss into our pipeline can be found in the differentiable renderer section.

Roughly, the process is divided into four main stages: Multiview-consistent Style Transfer, Backprojection, and Differentiable Rendering.

4.1. Multiview-consistent Style Transfer (MVST)

4.1.1 Rendering Views

We use Kaolin to render 16 views of the target mesh: 12 views at 30° azimuth increments $(30^\circ, \dots, 360^\circ)$ at zero elevation, plus 4 polar views at azimuth 30° with elevations $\{\pm 45^\circ, \pm 89^\circ\}$, giving full azimuthal and elevation coverage. Adjacent views are kept close in appearance to improve RePaint inpainting quality.

Each camera is placed at distance 4.0 in normalized mesh space with a 30° FoV perspective at 1024×1024. Illumination uses a spherical Gaussian model with three directional lights. Per view we render an RGB image, a foreground-normalized depth map, and a Canny edge map (thresholds $100/200$). Camera intrinsics (f_x, f_y, c_x, c_y), extrinsics $(R, \mathbf{t})$, and the raw depth range $[z_{\min}, z_{\max}]$ are serialized to `camera.json`.

4.1.2 ConsisLoRA Training

We use ConsisLoRA [1] to learn one content LoRA per view and a single shared style LoRA.

Content LoRAs. Each content LoRA is trained with prompt `"A [c]"` for 1500 steps at rank 64. A learning rate of 2×10^{-4} and ϵ-prediction loss are used for the first 500 steps; thereafter x_0 loss is added and the rate drops to 1×10^{-4}.

Style LoRA. Training is two-stage. *Stage 1* trains a content LoRA on the style reference image with prompt `"A [v]"` using the same schedule, providing a structural anchor. *Stage 2* trains the style LoRA proper for 1000 steps conditioned on the Stage-1 LoRA, with prompt `"An image in the style of [v]"`, learning rate 1×10^{-4}, x_0 loss from the start, and a noise offset of 0.03 for improved contrast.

4.1.3 Inference Pipeline

Views are processed in ascending order of |azimuth| + |elevation|, so view $i = 0$ (azimuth 30°, elevation 0°) is the canonical frontal pose. All cameras are converted from Kaolin's OpenGL convention (looks along $-Z$) to OpenCV convention (looks along $+Z$) by flipping the Y and Z basis vectors before any transform.

First view ($i = 0$). We run SDXL [7] with prompt `"a [c] in the style of [v]"`, loading both the view-specific content LoRA and style LoRA at unit scale. Structural conditioning is provided by a depth ControlNet (scale 0.8) and a Canny ControlNet (scale 0.4). Because no warped prior exists, RePaint resampling is disabled (jump length and resample count both 1). The result serves as the *anchor view*: all subsequent views are histogram-matched to it to prevent color drift. A foreground mask is extracted using BiRefNet [9].

Cumulative view warping ($i > 0$). Before diffusion for view i, every prior stylized view $j < i$ is forward-warped into the current frame to construct a color prior. Background pixels are zeroed using the per-view BiRefNet mask before warping.

For each foreground pixel (u, v) in source view j, metric depth is recovered from the stored OpenGL Z range as

$$d = -\big(z_{\min} + \tfrac{p}{255}(z_{\max} - z_{\min})\big)\,,$$

where $p \in [0, 255]$ is the rendered depth value (the negation converts from OpenGL's $-Z$ convention to metric depth). The pixel is unprojected and reprojected into the destination

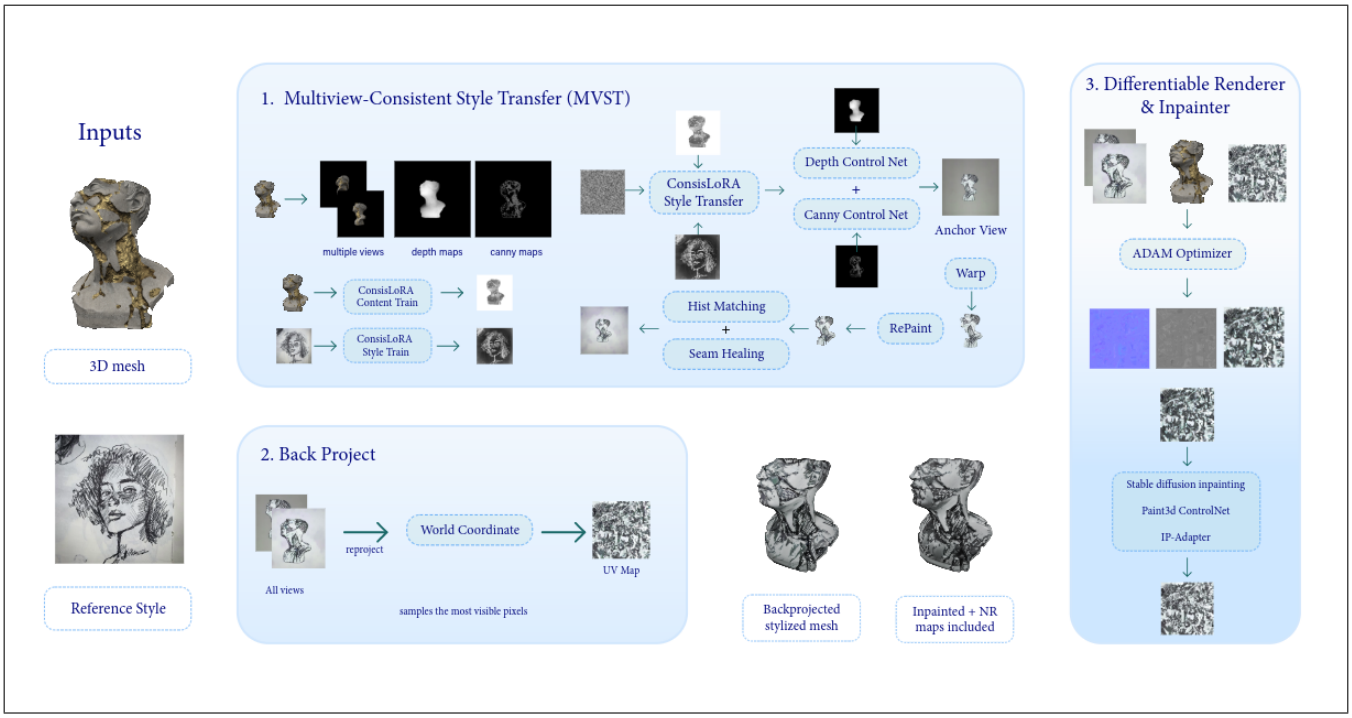

Figure 1. Methodology

view:

$$\mathbf{P}^{s}_{\text{cam}} = \left(\tfrac{u-W/2}{f^{s}_{x}}\,d,\ \tfrac{v-H/2}{f^{s}_{y}}\,d,\ d\right)^{\top},$$
$$\mathbf{P}_{\text{world}} = R_s^{\top}\left(\mathbf{P}^{s}_{\text{cam}} - \mathbf{t}_s\right),$$
$$\mathbf{P}^{d}_{\text{cam}} = R_d\,\mathbf{P}_{\text{world}} + \mathbf{t}_d,$$
$$u_d = f^{d}_{x}\,\tfrac{X_d}{Z_d} + \tfrac{W}{2}, \qquad v_d = f^{d}_{y}\,\tfrac{Y_d}{Z_d} + \tfrac{H}{2}.$$

Colors are scattered far-to-near so nearer surfaces overwrite farther ones. Two artifact-suppression steps follow: (1) small seam holes are filled by nearest-neighbor propagation within 3 pixels of a valid warped pixel and inside the destination mask; (2) the valid alpha mask is eroded by 20 pixels toward the occlusion side (opposite the projected source camera direction) to prevent dark boundary pixels from bleeding into inpainted regions.

Warped views are composited by a *nearest-view-wins* policy: each destination pixel takes the color from the source j maximizing $w_j = \exp(-\angle(\mathbf{c}_j, \mathbf{c}_i))$, where $\mathbf{c} = -R^{\top}\mathbf{t}$ is the camera position. The fusion also yields a binary validity mask $\mathbf{m}$ marking every destination pixel covered by at least one warped view.

RePaint inpainting. SDXL is run with the same LoRAs and ControlNets, using an adapted RePaint [5] schedule: denoising proceeds in blocks of $L = 10$ steps, each block repeated $r = 10$ times. At each step k, after the model updates the latent, the known region is pinned by compositing the prior noised to the next-timestep level:

$$\tilde{\mathbf{z}}_{\text{prior}} = \mathbf{z}_{\text{prior}} + \sigma_{t_{k+1}}\,\boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{1}$$
$$\mathbf{z}_{k+1} \leftarrow \mathbf{m} \odot \tilde{\mathbf{z}}_{\text{prior}} + (\mathbf{1} - \mathbf{m}) \odot \mathbf{z}^{\text{model}}_{k+1}. \tag{2}$$

Before the final resample of each block, latents are renoised back to the block-entry level to allow the model to harmonize the boundary:

$$\mathbf{z} \leftarrow \mathbf{z} + \sqrt{\sigma^2_{t_{\text{start}}} - \sigma^2_{t_{\text{end}}}}\;\boldsymbol{\varepsilon}. \tag{3}$$

After the main pass, a single lightweight refinement (`strength` $= 0.1$, $\approx$3 steps, no prior) subtly harmonizes the pinned and inpainted regions without substantially altering the result.

4.1.4 Backprojection

Stylized view colors are baked into a 4096×4096 UV texture atlas.

UV rasterization. Each texel (c, r) is assigned a world-space position by rasterizing mesh triangles in UV space via barycentric interpolation of vertices and normals:

$$\mathbf{P}_{\text{world}} = \textstyle\sum_i \lambda_i\,\mathbf{v}_i, \qquad \hat{\mathbf{n}} = \textstyle\sum_i \lambda_i\,\mathbf{n}_i \Big/ \left\|\sum_i \lambda_i\,\mathbf{n}_i\right\|.$$

The GLTF UV convention places $(0, 0)$ at the bottom-left, so rows are mapped as $r = (1 - v)H$.

Projection and visibility. Each texel is projected into every view (OpenCV convention). A view's contribution is accepted if: (1) the projected pixel is in-bounds with $Z_{\text{cam}} > 0$; (2) the depth map (max-pooled over 5×5 to handle silhouette boundaries) matches Z_{cam} within $\delta = 0.05$; and (3) the pixel lies inside the BiRefNet foreground mask.

View selection and color sampling. Among passing views, we select the one maximizing the face-on weight

$$w = \max\left(0,\ \hat{\mathbf{n}} \cdot \frac{\mathbf{c} - \mathbf{P}_{\text{world}}}{\|\mathbf{c} - \mathbf{P}_{\text{world}}\|}\right), \qquad \mathbf{c} = -R^{\top}\mathbf{t},$$

down-weighting grazing-angle views where perspective distortion is large. Color is read via bilinear interpolation. Unpainted texels at UV seams are filled by nearest-neighbor EDT propagation from the nearest painted texel.

4.2. Differentiable Rendering

To create reusable PBR materials, we optimize normal, roughness, and albedo maps using a Differentiable Rendering (DR) framework powered by an ADAM optimizer . We compare reference renders against our DR renders in a two-phase process:

Phase 1 (Normal and Roughness Maps): We optimize the normal and roughness maps using grayscale renders under directional lighting. $Loss = style\_loss + tv\_loss$. To capture high-frequency stylistic details on high-resolution 1024×1024 renders, our style loss extracts multiple random 256×256 cropped patches (with spatial and scaling jitter) from both the rendered and target images before passing them through the VGG network. The total loss minimizes this cropped style loss and Total Variation (TV) loss:

$$\mathcal{L}_{TV}(X) = \sum_{u,v}(||X_{u+1,v} - X_{u,v}||_1 + ||X_{u,v+1} - X_{u,v}||_1)$$

The style loss utilizes a VGG network to match feature maps (Φ_l) between rendered (R) and target (T) images, ignoring backgrounds via masks (M_R, M_T):

$$\mathcal{L}_{style} = \sum_{l} w_l ||\Phi_l(R \odot M_R) - \Phi_l(T \odot M_T)||_2^2$$

Phase 2 (Albedo Map): We reduce the learning rate for the normal and roughness maps and optimize the albedo map under ambient lighting. Because different styles require different levels of smoothness, we calculate an adaptive TV weight based on the mean image gradients of the target style reference, dynamically lowering the TV penalty for highly textured styles. Furthermore, to identify areas requiring subsequent inpainting, we track the optimizer's struggle by accumulating the absolute gradients of the albedo map over the final 100 iterations. The final loss incorporates Sliced Wasserstein Distance (SWD) for color matching: $Loss = 15.0\mathcal{L}_{style} + \lambda_{adapt}\mathcal{L}_{TV}(A) + 5.0\mathcal{L}_{color}$. The loss incorporates the previous terms along with a color loss: $Loss = style\_loss + tv\_loss + color\ loss$. The color loss enforces global color distribution matching using random 3D projections (θ):

$$\mathcal{L}_{color} = E_{\theta \sim \mathcal{S}^2}[\frac{1}{N}\sum_{i=1}^{N}(sort(R_{pixels} \cdot \theta)_i - sort(T_{pixels} \cdot \theta)_i)^2]$$

where R_{pixels} and T_{pixels} are sampled RGB values, and $sort()$ orders the projected values .

4.3. Inpainting

Direct backprojection causes artifacts and empty areas in occluded regions. To correct this, we utilize the accumulated gradients from Phase 2 to generate a binary "struggle mask." By thresholding the average gradients at $mean + 0.5 \times std$ and applying a max-pooling dilation, we accurately isolate regions where the differentiable renderer failed to converge. We input the UV map, a generated UV position map, and the struggle mask into a Stable Diffusion Inpainting pipeline via Runway ML. This process is guided by a semantic text prompt, an IP-Adapter for style reference, and Paint3D ControlNet to maintain spatial awareness. To prevent the diffusion model from bleeding colors across distinct UV islands, we implement a post-processing edge preservation step. By calculating the difference between a dilated and eroded boundary of the mask, we seamlessly blend the original baked pixels with the newly generated pixels exclusively along the geometric boundaries.

5. Results

5.1. Albedo Style Transfer

Can be observed in Figure 2.

5.2. NR Maps

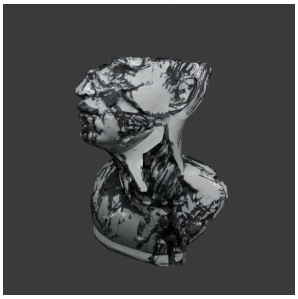

Figure 3. Mesh with DR + Inpainter Pipeline.

Figure 4. Mesh with the backprojected texture + original NR

We can observe here that the backprojected texture with the original Normal and Roughness maps, doesn't account

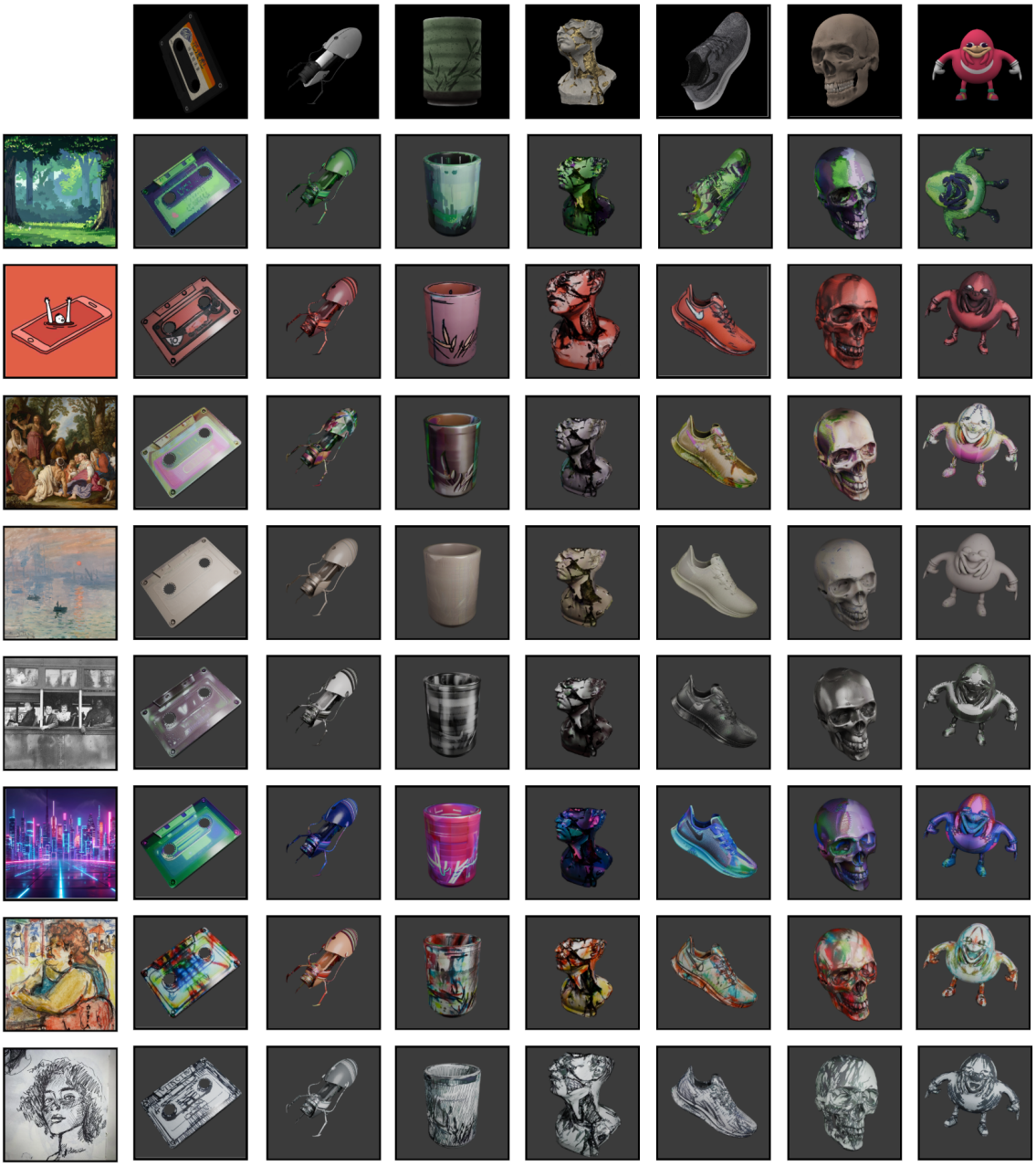

Figure 2. Results (note that these don't have normal or roughness maps)

for the fact that there's no gold anymore in the scar and is still reflective in that area. The inpainted albedo retains that information and is able to also account for the change in material that we observe through the style transfer. Furthermore, we are also able to notice the white areas of the marble bust are more reflective, and the scar is darkened as per the style transfer while still retaining the details of the backprojected texture.

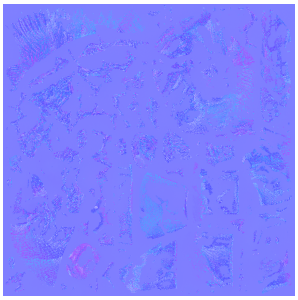

Figure 5. Normal Map

Figure 6. Roughness Map

6. Discussion and Conclusions

We can see that our final stylized meshes show great and accurate style transfer and are seamlessly integrable into traditional rendering engines. We are able to observe multiview consistency with great style transfer in the albedo map, accurate roughness and normal maps, and high quality reusable textures out of the box, but the process of projecting 2D generative hallucinations onto unyielding 3D geometry revealed several inherent limitations. We see that in style transfers for the pixelated drawing, and the robot arm which doesn't add detail to the arm and concentrates on the claw. Throughout the development of this pipeline, we encountered numerous challenges related to geometric mismatches, domain translation, and texture blurring, which provide valuable insights into the current limitations of 2D-to-3D asset generation.

6.1. Failed Experiments/Work In Progress/Future Work Directions

6.1.1 MVAdapter Integration

For multiview-consistent generation, we initially explored an adapter-based approach such as MV-Adapter [3]. Training the adapters themselves was out of the question due to time and compute constraints. However, we did attempt to see if integrating the pretrained MV-Adapter weights into the ConsisLoRA style transfer pipeline was possible. However, this did not work. It would appear that a retraining of the adapters is necessary, perhaps if we were to do this multiview consistency might improve compared to what we have now.

6.1.2 Optical Flow Warping of Depth Maps

As mentioned earlier in Section 4.1.3, the original depth maps are subtly inaccurate with respect to the corresponding stylized views. In our final pipeline, we chose to deal with it through a simple directional erosion. However, this doesn't fully solve the problem. We therefore investigated a more principled approach using DINOv2 [6] to estimate a dense warp that realigns the depth map to the stylized view's silhouette by computing landmark correspondences. However, we found that DINOv2 features exhibited poor cross-domain correspondence across the style gap causing incorrect correspondences. The resulting flow fields were noisy and the warped depth maps showed visible artifacts.

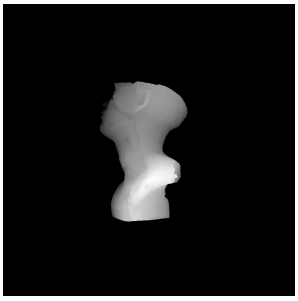

Figure 7. Original Depth Map.

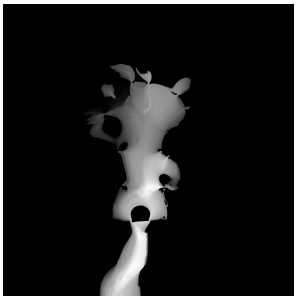

Figure 8. Warped Depth Map.

6.1.3 2DGS Mesh Generation and Editing

We noticed the depending on the style, the geometry of the stylized mesh (as inferred from the stylized views) will appear to be slightly different than the original mesh. As a result, a simple backprojection onto the original mesh will lose mesh-based stylistic features. To combat this, we attempted to use 2D Gaussian Splatting and then reconstruct a mesh from the Gaussian Splat. The 2DGS looks really good; however, reconstructing a good mesh from it proved to be quite difficult:

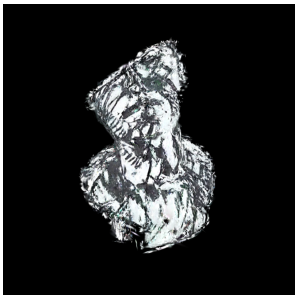

Figure 9. 2DGS point cloud (SuperSplat).

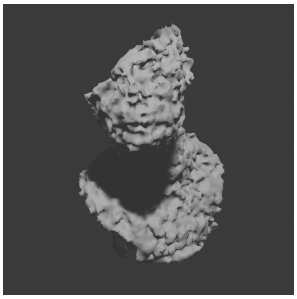

Figure 10. Generated Mesh.

Perhaps in the future, we can attempt to do a differential rendering process on the original mesh to edit it slightly so that it better matches the goemetry of the Gaussian Splat.

6.2. More Future Work

6.2.1 Optimal Camera Angle Selection

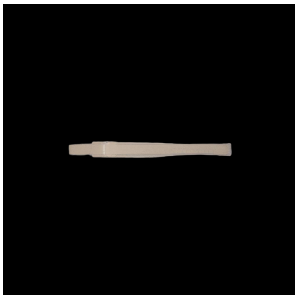

Figure 11. Az = 30, el = 0

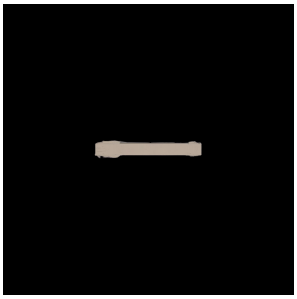

Figure 12. Az = 90, el = 0

Currently, our Multiview-consistent Style Transfer (MVST) phase relies on a hardcoded, uniform camera rig comprising 16 views (12 azimuthal views at 30° increments and 4 polar views). While this ensures baseline coverage for standard objects, it is computationally inefficient for simple, highly symmetric meshes and insufficient for complex meshes with deep concavities or severe self-occlusions. Or in these instances like we see with the cassettes (fig 8,9), the elevation being set at 0, makes it such that many references carry little to no detail. Future iterations of this pipeline should integrate an adaptive, geometry-aware camera placement algorithm. By analyzing the base mesh's surface normal variance and depth complexity prior to rendering, we could employ Next-Best-View (NBV) algorithms to dynamically calculate the minimum number of camera angles required to guarantee 100% surface visibility while still having enough overlap between views for the style transfer to work properly. This would reduce the computational overhead of ConsisLoRA and RePaint for simple objects while simultaneously eliminating the reprojection artifacts caused by grazing angles on complex geometry.

6.2.2 Mesh Editing

As noted during our experiments with 2D Gaussian Splatting (2DGS), 2D image stylization frequently implies physical changes to the underlying geometry—such as adding tufts of hair, exaggerated proportions, or jagged sketch outlines. Because our current pipeline bakes these stylized 2D views back onto the rigid, unaltered geometry of the original mesh, these geometric stylistic features are lost, sometimes causing severe texture stretching along the original silhouettes. While reconstructing a clean topology from a 2DGS point cloud proved infeasible, future work could circumvent this issue by extending our existing differentiable rendering framework. Instead of only optimizing texture maps (normal, roughness, and albedo) using the ADAM optimizer, we could introduce vertex displacement as a trainable parameter. By jointly optimizing the mesh vertices against the multi-view silhouettes and depth maps of the stylized views (or a 2DGS proxy), we could subtly deform the base mesh to physically match the implied geometry of the style. This would also fix errors that we can observe with styles like pixel art and such where the style transfer is unable to accurately reflecte pixelated structures.

7. Acknowledgements

We used Claude Code and Gemini to help with implementing the codebase.

References

[1] B. Chen, B. Zhao, H. Xie, Y. Cai, Q. Li, and X. Mao. ConsisLoRA: Enhancing content and style consistency for LoRA-based style transfer, Mar. 2025. arXiv:2503.10614.

[2] Lukas Höllein, Justin Johnson, and Matthias Nießner. Stylemesh: Style transfer for indoor 3d scene reconstructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6198–6208, 2022.

[3] Z. Huang et al. MV-Adapter: Multi-view consistent image generation made easy, Dec. 2024. arXiv:2412.03632.

[4] Kunhao Liu, Fangneng Zhan, Muyu Xu, Christian Theobalt, Ling Shao, and Shijian Lu. Stylegaussian: Instant 3d style transfer with gaussian splatting. *arXiv preprint arXiv:2403.07807*, 2024.

[5] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. V. Gool. RePaint: Inpainting using denoising diffusion probabilistic models, Aug. 2022. arXiv:2201.09865.

[6] M. Oquab et al. DINOv2: Learning robust visual features without supervision, Feb. 2024. arXiv:2304.07193.

[7] D. Podell et al. SDXL: Improving latent diffusion models for high-resolution image synthesis, July 2023. arXiv:2307.01952.

[8] 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, 2023.

[9] P. Zheng et al. Bilateral reference for high-resolution dichotomous image segmentation, 2024. arXiv:2401.03407, accessed May 12, 2026.