# Geometry in Style: 3D Stylization via Surface Normal Deformation

## Supplementary Material

These sections provide additional results and more information on our deformation method. Appendix A presents qualitative results and comparisons in addition to those from the paper. In Appendix B, we show an evaluation of CLIP similarity to the prompt. In Appendix C, we perform an ablation on the rotation-finding method and show the regularization significance of the Procrustes local step. In Appendix D, we explain in detail the Cascaded Score Distillation (CSD) semantic loss, introduced by Decatur et al. [2], and provide the hyperparameters and configuration we used. In Appendix E we provide extra clarifications on the quantitative evaluation from the main paper. Finally, in Appendix F, we show a running time comparison of the global step solves of dARAP and NJF [1].

## A. Additional Qualitative Results

In Fig. 12 we present additional qualitative comparisons of our method against MeshUp and TextDeformer. In the *racer bunny* prompt, our method achieves the target style with most fidelity. In the *pagoda* prompt, TextDeformer produces artifacts and achieves less satisfactory style; MeshUp achieves the target style but aggressively transforms the shape beyond recognition from the source, losing its identity. Our method achieves the *pagoda* style and retains the original vessel-and-column configuration. In the *cybernetic glove* example, our method achieves the target style cleanly and preserves the slender proportions of the source hand.

In Fig. 13, we add to Fig. 5 in the main paper with another example showing the same shape deformed with different style prompts, here a person shape. All three examples show salient features of the style but preserve the slender proportions of the original shape.

In Fig. 15 we present additional results of our method on select quadruped animal meshes from the SMAL model [8] (the image-fitting result meshes presented in their work) paired with diverse prompts.

## B. CLIP Similarity to Prompt

We computed the CLIP similarity between rendered images of the deformed mesh and the stylization text prompt. Our evaluation on the 20-shape set (the same set evaluated quantitatively in the main paper) using 16 views per mesh shows (Tab. 2) that we obtain better semantic similarity to the specified style prompt compared to MeshUp [4] and TextDeformer [3].

| | TextDeformer | MeshUp | **Ours** |
|---|---|---|---|
| ViT-B/16 CLIP sim. (↑) | 0.650 | 0.653 | **0.655** |

Table 2. Our method achieves better CLIP similarity to the prompt than TextDeformer and MeshUp.

## C. Ablations

**Rotation-finding method.** In Fig. 14, we compare choices of the local rotation-finding method. Our local step using a Procrustes solve, which finds a rotation matrix for each vertex given a target normal, is inherently regularized by virtue of finding a best-fit rotation for not only the source normal but also a neighborhood of edge vectors (Fig. 4).

Without this Procrustes solve, one might *directly optimize* per-vertex rotations (*e.g.* using the continuous $3 \times 2$ matrix representation described in Zhou et al. [7]). However, we show in Fig. 14 that this, without identity regularization loss, is far less restrictive than our target-normal-

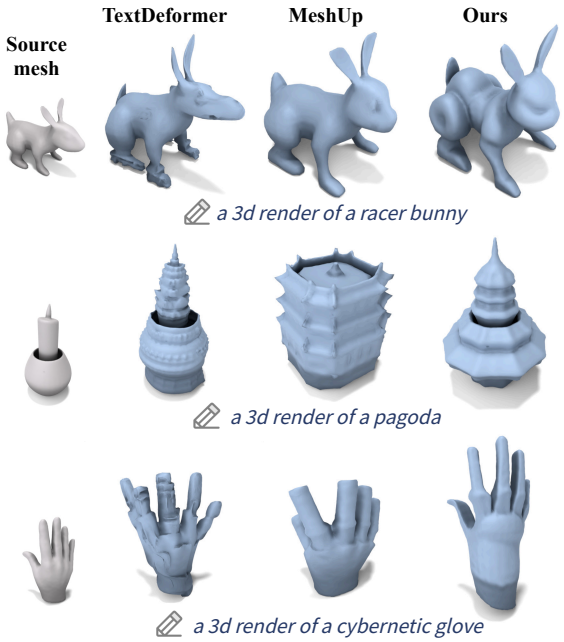


Figure 12. **More qualitative comparisons with MeshUp and TextDeformer.** Note that TextDeformer contains artifacts and noisy surface detail, and MeshUp either fails to attain the prompt style, or deforms the shape beyond recognition and loses the source shape's identity. Our method attains the target style and preserves the source shape's identity.

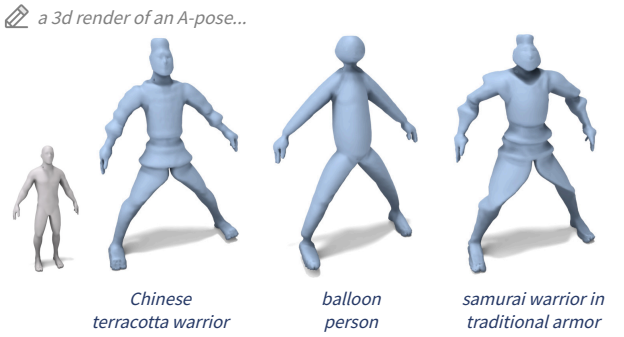


Figure 13. Another example of deforming the same mesh towards different text-specified styles.

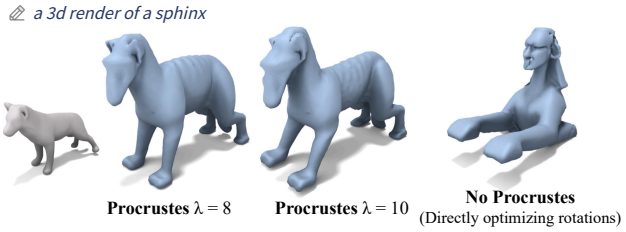


Figure 14. **Rotation ablation.** Our Procrustes solve preserves identity even at different values of $\lambda$ (tunable to taste). Directly predicting rotations is not restrictive enough and severely changes shape identity.

based local step and loses the identity of the source mesh, undesirable for our goal of stylization.

Further, our Procrustes solve hyperparameter $\lambda$ can be chosen (for optimization) to add additional detail yet still preserve identity to taste. Fig. 14 shows that the stylization is visible yet preserves the identity of the source dog mesh for both $\lambda = 8$ and $\lambda = 10$. In particular, the squared feet and sharper angle of the neck are prominent in both, with $\lambda = 8$ preserving the front leg pose better, while $\lambda = 10$ having stronger sphinx-like feet. We use $\lambda = 8$ in the main results as it strikes a good balance of identity preservation and strength for most meshes, but this can be tuned on a per-mesh basis. Additionally, as shown in Fig. 8, $\lambda$ can also be changed at inference/application time to further adjust a pre-optimized deformation's strength.

## D. Cascaded Score Distillation (CSD)

### D.1. Method

In this section, we briefly describe how Cascaded Score Distillation is used to optimize the target vertex normals $\hat{\mathcal{U}} = \{\hat{\mathrm{u}}_k \mid k \in \{1 \dots |\mathcal{V}|\}\}$. We first provide an overview of how Score Distillation Sampling (SDS) can be applied to optimize a deformation quantity (*e.g.* jacobians, or target normals in our case) using diffusion models, and extend the concept to show how we use Cascaded Score Distillation as our guidance. See Fig. 16 for an illustrative overview.

To stochastically optimize parameters (target normals in our case) with respect to a pre-trained 2D diffusion model, Poole et al. [5] proposed Score Distillation Sampling (SDS), where given a rendered image $\mathbf{z}$ and a text condition $y$, the objective is to minimize the L2 loss

$$\mathcal{L}_{\text{Diff}}(\omega, \mathbf{z}, y, \epsilon, t) = w(t) \left\| \epsilon_\omega \left( \mathbf{z}_t, y, t \right) - \epsilon \right\|_2^2, \quad (8)$$

which is the L2 difference between the sampled noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ added to the image, and the noise $\epsilon_\omega$ predicted by a denoising U-Net $\omega$ at some timestep $t$, sampled from a uniform distribution $t \sim U(0, 1)$. Here, $w(t)$ is a weighting term, and $\mathbf{z}_t$ is the rendered image, noised according to the sampled timestep $t$. To compute the gradient of the optimizable parameters, which in our case is the set of all target normals $\mathrm{u}_k$ with respect to the loss $L_{\text{Diff}}$, it has been shown that the gradients of the U-Net can be omitted for efficient computation [5, 6], giving

$$\nabla_{\mathrm{u}_k} \mathcal{L}_{\text{SDS}}(\phi, \mathbf{z}, y, \epsilon, t) = w(t) \left( \epsilon_\omega(\mathbf{z}_t, y, t) - \epsilon \right) \frac{\partial \mathbf{z}_t(\mathrm{u}_k)}{\partial \mathrm{u}_k}. \quad (9)$$

$\frac{\partial \mathbf{z}_t(\mathrm{u}_k)}{\partial \mathrm{u}_k}$ can be obtained by backpropagating the gradient from the rendered images through our fully differentiable pipeline, and using $\nabla_{\mathrm{u}_k}$ we can optimize the target normals with text-to-image diffusion models.

Cascaded Score Distillation is an extension of SDS that allows our parameters to be optimized with additional guidance from super-resolution models. While SDS only approximates the gradients from the first stage, base diffusion model, CSD utilizes the super-resolution models (usually 2nd or 3rd stage models), which are additional models used to upsample and fine-tune low-resolution images generated by the base model. This additional guidance from high-resolution modules allows for a more fine-grained generation of stylistic details in our results, making it an appropriate choice for our

To use Cascaded Score Distillation as our guidance, we add the gradient,

$$\nabla_{\mathrm{u}_k} \mathcal{L}_{\text{CSD}_i}(\phi^i, z^i, z^{i-1}, y) = w(t) \left( \epsilon_{\phi^i}(z_t^i, t, z_s^{i-1}, s, \mathbf{y}) - \epsilon^i \right) \frac{\partial \mathbf{z}_t(\mathrm{u}_k)}{\partial \mathrm{u}_k}, \quad (10)$$

where the idea is to use an image upsampled at the resolution of the $i^{th}$ stage module, $z^i$, along with the image of the resolution of the previous $i-1^{th}$ stage module, $z^{i-1}$, and noise them respectively at timestep $t$ and $s$ to use them as inputs to the $i^{th}$ stage high-resolution module. Likewise for SDS, we omit the expensive computation of the gradient through the U-Net and calculate the gradient with respect to

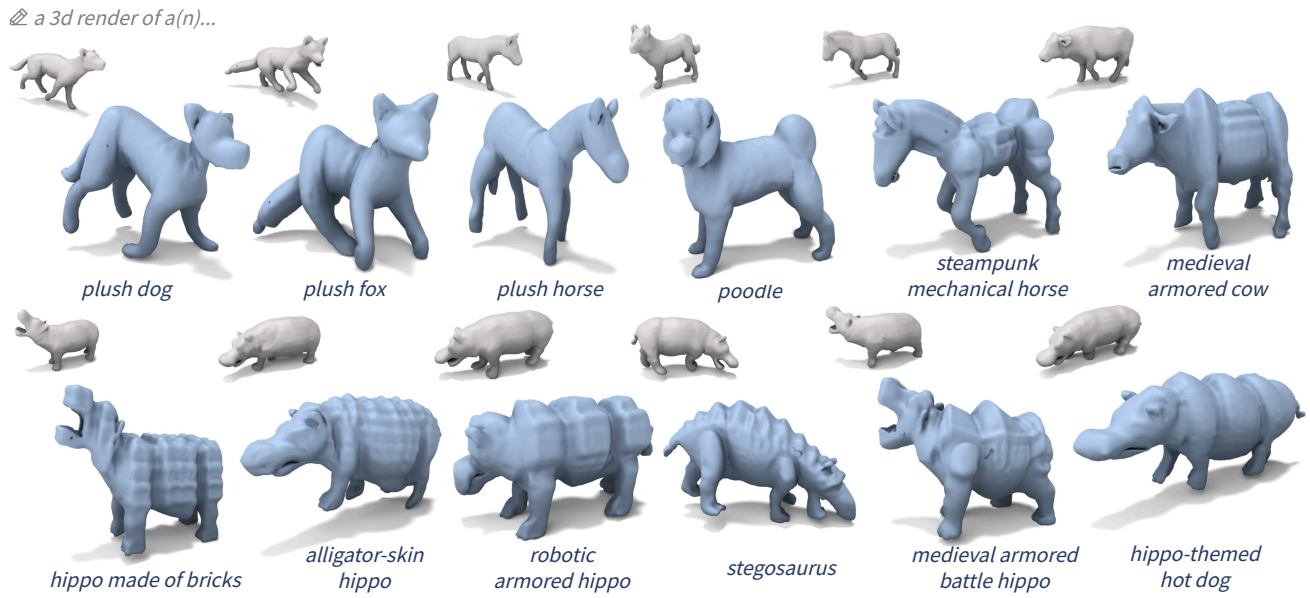


Figure 15. **Diverse prompts on SMAL [8] quadruped animals.** For a variety of animals, the deformation imparts the requested style but preserves the general pose. Even in cases where the prompt's identity is different (*e.g. poodle, stegosaurus, hot dog*), our method works with the existing geometry and keeps its identity, effectively stylizing the original shape to look like the target object rather than overriding the source shape and growing a distinct body.

our target normals by backpropagating through our differentiable pipeline.

Combining the gradient from SDS with CSD gives us the following final gradient,

$$\nabla_{\mathbf{u}_k}\mathcal{L}_{\text{CSD}}(\phi,\mathbf{z},y,\epsilon,t) = \alpha^1\nabla_{\mathbf{u}_k}\mathcal{L}_{\text{SDS}} + \sum_{i=2}^{N}\alpha^i\nabla_{\mathbf{u}_k}\mathcal{L}_{\text{CSD}_\text{i}}(\phi^i,z^i,z^{i-1},\mathbf{y}) \tag{11}$$

where $\alpha^i$ are the user-defined weights for each gradient.

### D.2. Configuration Details

**CSD settings.** Recall that an epoch in our optimization pipeline consists of a batch of randomly sampled views (we use a view batch size of 8) fed to CSD. Apart from the base model for SDS, we only use one upscaling stage of DeepFloyd IF (*i.e.* $N = 2$ in Eq. (11)) The weight $\alpha^2$ of the CSD second-stage is linearly ramped up from 0 to 0.2 over the course of 1000 epochs, then 0.2 to 0.3 over 750 epochs, remaining at 0.3 for the rest of the epochs. The weight $\alpha^1$ for SDS is a constant 1.0. We use a classifier-free guidance weight of 100 as in MeshUp [4].

For each batch of rendered views we compute the CSD loss, backpropagate, and perform a gradient descent update on $\hat{\mathcal{U}}$ *twice* before recomputing the deformed shape and re-rendering it for the next epoch/batch of views. This technique was also used in the official MeshUp implementation and empirically leads to sharper deformations.

**View sampling settings.** The renders sample from a range of views around the mesh: a full azimuth range of $0°$ to $360°$, an elevation range of $0°$ to $60°$ (or $30°$ for tall and slim shapes such as humans), a distance range of $2.5$ to $3.0$ for most shapes (or $1.4$ to $2.6$ for tall and slim shapes such as humans), and a fixed FOV of $60°$.

## E. Additional Quantitative Evaluation Details

We provide additional details and explanations about the quantitative evaluation reported in Tab. 1 in the main paper. For an informative comparison, we normalize the source shape to fit a side-2 cube centered at origin, and rescale the deformed shape to share the same axis-aligned bounding box diagonal length as that of the normalized source shape. We use this pre- and post-normalization as it is the same scheme used before and after deformation in MeshUp and our method. This normalization means the face area ratio will measure *two effects*: the first is any localized face area distortion to accommodate a deformation, and the second is if the deformation (before normalizing) significantly enlarges the shape's bounding box even if there is no significant localized distortion (*e.g.* when a limb is rotated to spread out further). When the latter case happens, after the normalization, the area ratios will be smaller than 1 due to a global rescaling down to fit the original bounding box extent, and vice versa.

In addition to the average and standard deviation of the area ratio reported in the main body, we further show the

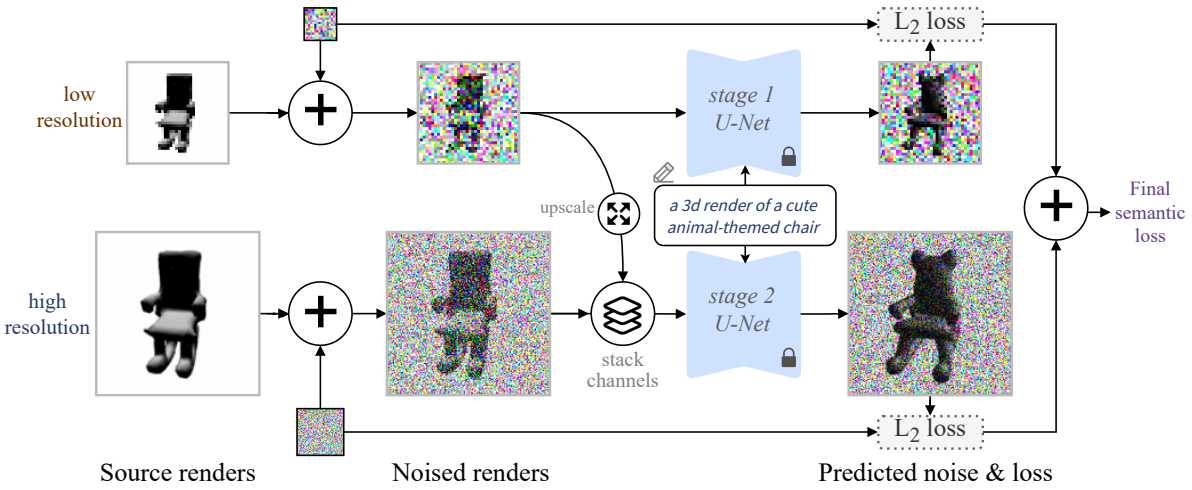


Figure 16. **Overview illustration of the Cascaded Score Distillation semantic loss.** Two-stage CSD uses two pretrained denoising U-Nets to predict the noise for the noised renders at their corresponding resolutions. The higher resolution stage incorporates an upscaled noised render from the low-resolution branch.

distribution of the area ratio in Fig. 17. Interestingly, we see that the distribution for TextDeformer [3] is mainly centered around values smaller than 1, suggesting that TextDeformer tends to enlarge the source shape's extents as a whole (resulting in a global shrinkage upon normalizing), while for MeshUp [4] the behavior is the other way around. In contrast, the distribution for our method is cleanly located around a ratio of value 1, indicating better triangle area and bounding box preservation compared to the other methods.

## F. Running Time

We compare the running time of dARAP against the equivalent in NJF [1]. Since faces approximately outnumber vertices 2:1 on simplicial surfaces, our vertex-based global solve is faster than NJF's face jacobian-based solve (0.0507s *vs*. 0.0755s average). Even combined with our local step, the two methods are comparable in run time; see the supplementary material for measurement details.

We compare in Tab. 3 the running time of dARAP against the equivalent in NJF [1]. Averaged over 10 runs on a mesh with 20708 faces and 10356 vertices on a GTX 1660Ti GPU, excluding the precomputations of both methods, our Poisson system construction and solution is faster than that of NJF, owing to the fact that faces usually outnumber vertices 2:1 on simplicial surfaces. While NJF does not involve a local step, even coupled with the local step, our local-global dARAP altogether has a comparable running time to an NJF Poisson solve.

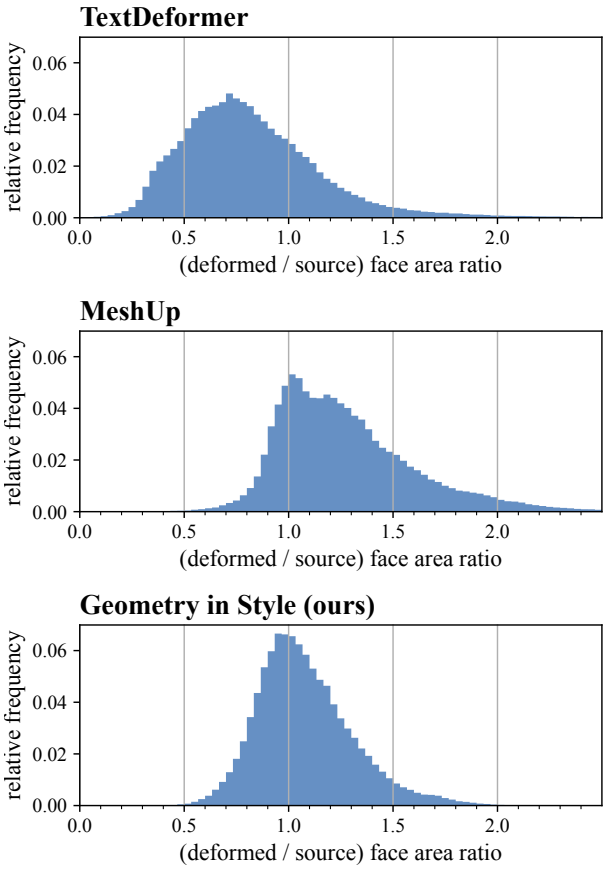


Figure 17. **Distribution of (deformed / source) face area ratios.** $n = 306300$ faces across 20 source-deformed mesh pairs.

| Method | Local step | Global step | Total |
|---|---|---|---|
| NJF [1] Poisson | — | 0.0755s | 0.0755s |
| dARAP (**ours**) | 0.0412s | 0.0507s | 0.0918s |

Table 3. **Running time comparison against NJF.** Since faces outnumber vertices 2:1 on simplicial surfaces, our vertex-based global solve is faster than NJF's face jacobian-based solve. Combined with our local step, the two methods are on par in run time.

## References


[1] Noam Aigerman, Kunal Gupta, Vladimir G. Kim, Siddhartha Chaudhuri, Jun Saito, and Thibault Groueix. Neural jacobian fields: learning intrinsic mappings of arbitrary meshes. *ACM Trans. Graph.*, 41(4), 2022. 1, 4, 5

[2] Dale Decatur, Itai Lang, Kfir Aberman, and Rana Hanocka. 3d paintbrush: Local stylization of 3d shapes with cascaded score distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4473–4483, 2024. 1

[3] William Gao, Noam Aigerman, Thibault Groueix, Vova Kim, and Rana Hanocka. Textdeformer: Geometry manipulation using text guidance. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–11, 2023. 1, 4

[4] Hyunwoo Kim, Itai Lang, Noam Aigerman, Thibault Groueix, Vladimir G Kim, and Rana Hanocka. Meshup: Multi-target mesh deformation via blended score distillation. *arXiv preprint arXiv:2408.14899*, 2024. 1, 3, 4

[5] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion, 2022. 2

[6] Haochen Wang, Xiaodan Du, Jiahao Li, Raymond A. Yeh, and Greg Shakhnarovich. Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation, 2022. 2

[7] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5745–5753, 2019. 1

[8] Silvia Zuffi, Angjoo Kanazawa, David Jacobs, and Michael J. Black. 3D menagerie: Modeling the 3D shape and pose of animals. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1, 3