Supplementary Material for 3D Paintbrush: Local Stylization of 3D Shapes with Cascaded Score Distillation

A. Additional Experiments

Variation. For a given mesh and prompt, there are often multiple valid interpretations for the local edit. With different seeds, our method produces various diverse, yet plausible, results (see Fig. 12) allowing users to choose the output that best matches their desired edit.

Qualitative comparisons. We show qualitative comparisons of our method to other localization and editing techniques. For our localizations, we compare to the mesh-native approaches 3D Highlighter [10] and SATR [2]. Our localizations are more accurate and more highly-detailed than 3D Highlighter and SATR (see Fig. 13). For example, the position and shape of our apron more closely adheres to that of an apron and our necklace contains the fine-grained chain links whereas both 3D Highlighter and SATR produce an overly smooth band. Since our method is the first approach to perform local edits on meshes, we evaluate our local editing capabilities by comparing to the mesh-native global editing method Latent Paint [38] and the voxel-based NeRF local editing method Vox-E [49]. In Fig. 14, we observe that our method produces local stylizations with higher resolution textures than Latent Paint [38] and Vox-E [49]. Furthermore, our texture edits are contained to a local semantic region corresponding to the target edit. In contrast, Vox-E often makes changes to parts of the texture that are unrelated to the target edit and Latent Paint consistently makes changes to the entire texture.

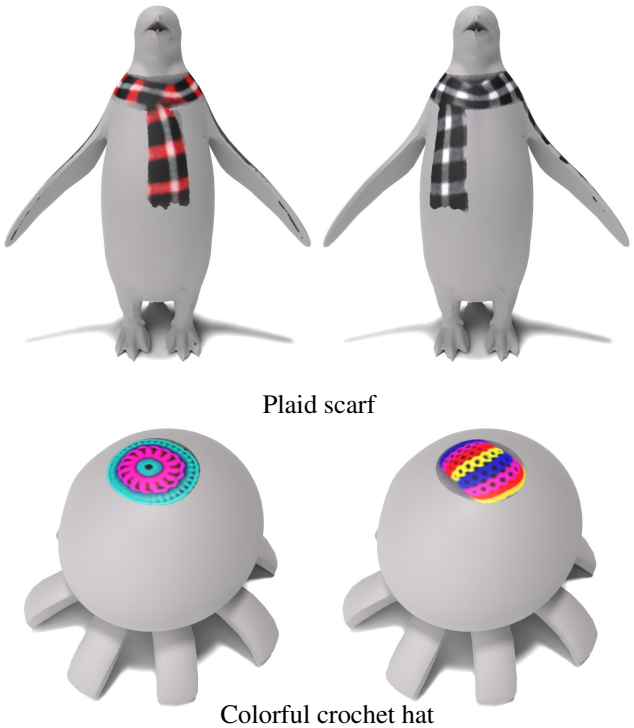


Figure 12. **Variation**. 3D Paintbrush can give multiple varied, yet plausible, outputs for a given local style depending on the input seed. We show results on two different seeds for each local style.

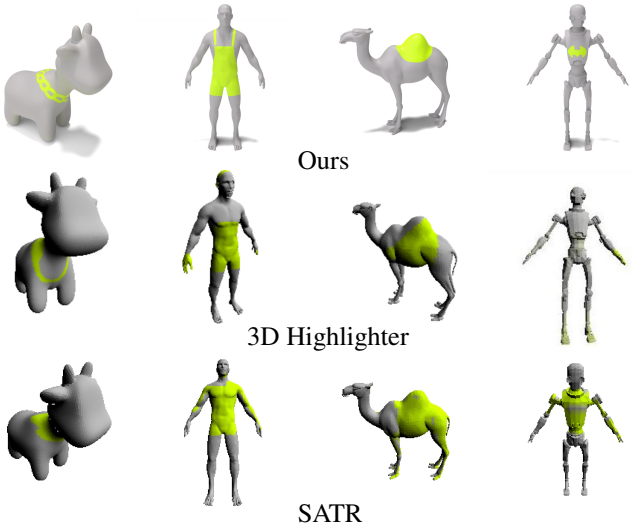


Figure 13. **Qualitative localization comparison**. Our localizations are more accurate and finer-grained than 3D Highlighter [10] and SATR [2]. Full input text prompts (from left to right): a 3D render of a gray cow with a yellow chain link necklace, a 3D render of a gray person with a yellow apron, a 3D render of a gray camel with a yellow turtle shell, a 3D render of a gray robot with a yellow batman chest emblem.

Method	Clip R-Precision ↑					
	CLIP B/32		CLIP B/16		CLIP L/14	
	Loc	Style	Loc	Style	Loc	Style
3DH	13.3	n/a	7.0	n/a	23.7	n/a
LatentPaint	n/a	20.0	n/a	20.0	n/a	20.0
Ours	**26.7**	**60.0**	**13.0**	**40.0**	**46.7**	**73.3**

Table 2. Quantitative evaluation. We compare our localizations to 3D Highlighter [10] (3DH) and our local textures to Latent Paint [38] and report CLIP R-Precision. Note, 3DH does not produce stylizations and Latent Paint does not produce localizations.

Quantitative evaluation with CLIP R-Precision. We compare our method to mesh-native baselines using CLIP R-Precision, an automated method for measuring alignment between generations and text prompts commonly used in text-to-3D [10, 25, 41, 44]. Given a set of text prompts

Figure 14. **Qualitative local stylization comparison**. Our 3D Paintbrush gives more highly detailed and local edits than Latent Paint [38] and Vox-E [49]. Full input text prompts (from left to right): a 3D render of a gray hand with a fancy gold watch, a 3D render of a gray Lego minifigure with a Barcelona jersey, a 3D render of a gray desk chair with a colorful doily seat cushion, a 3D render of a gray person with a tie-dye apron.

and corresponding generated textured meshes, this metric reports the percentage with which CLIP is able to retrieve the correct text prompt used to generate each given textured mesh. 3D Paintbrush consistently scores the highest for both localization and local texturing (Tab. 2). We give further details on CLIP R-Precision in Appendix B.

MLP and direct optimization ablation. We show an ablation of using MLPs by replacing them with direct optimization for the localization and texture maps (Fig. 15). Our full method (far left) using MLPs for both the localization and texture maps gives an accurate and contiguous localization with a clean, detailed local texture. Directly optimizing the texture map values instead of using an MLP (center left) results in a noisier, grainy texture. Using an MLP for the localization map is especially important since we want the localizations to be mostly contiguous and the smoothness of the MLPs gives us this. Directly optimizing the localization map values instead of using an MLP gives a non-contiguous, speckled localization. This localization detrimentally affects the texture as well, whether the texture is optimized either with an MLP (center right) or directly (far right).

Figure 15. **MLP ablation**. We ablate the use of MLPs by using direct optimization for both the localization and texture maps. Using MLPs provides smooth localization and texture while using direct optimization leads to noisy and less coherent results.

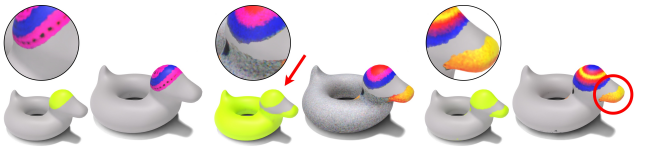
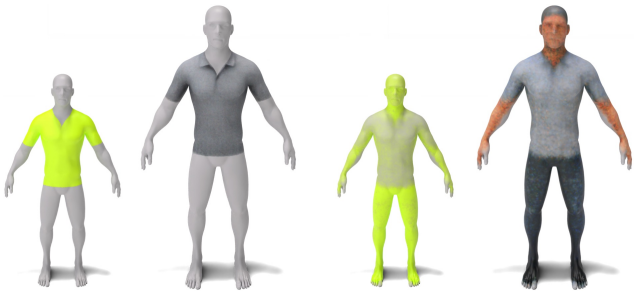


Figure 16. **Background loss ablation**. Our method (left) produces an accurate localization and texture that are highly detailed and tightly coupled. Removing the background MLP $\mathcal{F}_\psi$ and instead learning the background through the main texture map T_{map} leads to poor localization which also degrades the texture (middle). Removing the background loss completely leads to the incorporation of superfluous elements from the input model (*i.e*. a bill on a duck) into the localization (right).

Background loss ablation. We opt to learn the background texture in a separate map from the desired texture map (middle and bottom branches of Fig. 3) enabling our method to produce accurate localizations with tightly coupled and detailed textures (Fig. 16, left). If we instead remove the explicit background texture map (and MLP) and allow the background to be predicted using the same texture map as the main edit texture T_{map}, (by applying background loss to T_{map} masked with the inverse of the localization mask), then the localization and texture become misaligned (Fig. 16, middle). In this case, when the edit and background share the same texture map, if the localization region expands during training to include features that had been considered background, the edit texture may retain these features of the object (rather than expand the edit texture features to fill the localization) and the localization may continue to expand. If we instead remove the background loss entirely, this also produces undesirable results (Fig. 16, right). Specifically, we observe extra elements incorporated in additional localization regions that contain characteristics of the input shape (*e.g*. bill on a duck).

Localization loss ablation. We show an ablation on the localization loss (see Fig. 17). Using our full method, we obtain a precise localization that accurately depicts the local edit region "polo shirt" (left). Without our localization loss,



Ours w/o localization loss

Figure 17. **Localization loss ablation**. We show an ablation of the localization loss using the local edit "polo shirt" on a person mesh. Without this loss, there is no explicit incentive for the localization to be visually meaningful in isolation which can lead to inaccurate localization regions (right).

the only supervision signal for the localization region comes from the style loss as that depends on the localization region to obtain the masked texture. From just this signal, there is no incentive for the localization to have visual meaning and thus we often end up with inaccurate localizations (right localization). These poor localizations can lead to worse textures as well (right texture).

B. Implementation and Further Details

MLP architecture. Our MLPs consist of a positional encoding layer followed by 6 linear layers. Both texture MLPs and the localization MLP take a dimension 3 input corresponding to a 3D coordinate. All linear layers have width 256 and each linear layer is followed by a ReLU activation and subsequently a layer norm. The localization MLP outputs a single probability between 0 and 1, while the texture MLPs output 3 values each also within the range 0 to 1 corresponding to RBG values.

Optimization and supervision details. We use PyTorch to implement our method. For our optimization, we use an Adam optimizer with a constant learning rate of $1e^{-4}$. We use the Huggingface Diffusers implementation of DeepFloyd IF for our diffusion supervision. For all experiments, we use a classifier free guidance weight of 20. We train for 4 hours on a single A40 GPU which typically equates to 4000 iterations.

Text prompt formulation. To create the text prompts, our method takes as input the object class (*i.e*. "cow"), the desired style term (*i.e*. "colorful crochet") and the desired local edit (*i.e*. "hat"). To create y_l, we formulate the prompt "a 3D render of a gray [object class] with a yellow [local edit]" or specifically, "a 3D render of a gray cow with a yellow hat." To get y_t, we use "a 3D render of a gray [object class] with a [style term] [local edit]" or "a 3D render of a gray cow with a colorful crochet hat." To obtain y_b, we use "a 3D render of a [object class] with a yellow [local edit]" or "a 3D render of a cow with a yellow hat."

View selection. At each iteration of our optimization we render our mesh from multiple views. To select these views, we randomly sample camera elevation, azimuth, and radius from predefined ranges that can be specified by the user. In all of our experiments, we render 4 views each iteration. For all experiments, we sampled azimuths ranging from 0 to 360 degrees. By default, we sample elevations from the range $0 - 150$ degrees, however for some meshes, we narrow this range to $0 - 100$ degrees. For all experiments, our camera radius is sampled uniformly from the range $1 - 1.5$.

Quantitative evaluation with Clip R-Precision details. To quantitatively evaluate our method, we use a CLIP R-Precision metric tailored to evaluating either localizations or local styles. We create 15 local edits and record the both the target localization y_l and target local style y_t for each edit as well as their corresponding generated localizations and local styles. To create our localizations for 3D Highlighter [10], we run 3D Highlighter on each of those 15 y_l's to get 15 localizations. To create our local styles for Latent Paint [38], we run Latent Paint on each of the 15 y_t's. To compute the CLIP R-Precision score for localizations for a given method, we do the following. For each localization result L_i of the 15 total from this method, we compute the CLIP similarity between renders of L_i and each of the 15 y_l's to get 15 similarity scores. If the y_l with the highest CLIP similarity to L_i is the y_l that was used to generate L_i then this L_i is awarded a score of 1, otherwise it is awarded a score of 0. To compute the CLIP R-Precision for this method, we take the average score of all 15 L_i's and multiply by 100 to get a percentage. The CLIP R-Precision for the local styles is computed similarly, except using local styles instead of localizations. Specifically to compute the CLIP R-Precision of the local style for a given method, we do the following. For each local style result S_i of the 15 total from this method, we compute the CLIP similarity between renders of S_i and each of the 15 y_t's to get 15 similarity scores. If the y_t with the highest similarity score to S_i is the y_t that was used to generate S_i then this S_i is awarded a score of 1, otherwise it is awarded a score of 0. To compute the CLIP R-Precision for this method, we take the average score of all 15 S_i's and multiple by 100 to get a percentage.