Received XX Month, XXXX; revised XX Month, XXXX; accepted XX Month, XXXX; Date of publication XX Month, XXXX; date of current version XX Month, XXXX.

Digital Object Identifier 10.1109/XXXX.2022.1234567

LEMON: Localized Editing with Mesh Optimization and Neural Shaders

FURKAN MERT ALGAN, UMUT YAZGAN, DRITON SALIHU
CEM ETEKE, ECKEHARD STEINBACH

[1]Technical University of Munich, 80333 Munich, Germany

Corresponding author: Furkan Mert Algan (email: fmert.algan@tum.de).

This work was supported in part by the Bavarian Ministry of Economic Affairs, Regional Development and Energy under the project 6G Future Lab Bavaria.

ABSTRACT In practical use cases, polygonal mesh editing can be faster than generating new ones, but it can still be challenging and time-consuming for users. Existing solutions for this problem tend to focus on a single task, either geometry or novel view synthesis, which often leads to disjointed results between the mesh and view. In this work, we propose LEMON, a mesh editing pipeline that combines neural deferred shading with localized mesh optimization. Our approach starts by identifying the most important vertices in the mesh for editing, using a segmentation model to focus on these key regions. Given multi-view images of an object, we optimize a neural shader and a polygonal mesh while extracting the normal map and the rendered image from each view. Using these outputs as conditioning data, we edit the input images with a text-to-image diffusion model and iteratively update our dataset while deforming the mesh. This process results in a polygonal mesh that is edited according to the given text instruction, preserving the geometric characteristics of the initial mesh while focusing on the most significant areas. We evaluate our pipeline on the DTU dataset, demonstrating that it generates finely-edited meshes more rapidly than the current state-of-the-art methods. We include our code and additional results in the supplementary material.

INDEX TERMS 3D Reconstruction, Diffusion Models, Mesh Editing, Neural Rendering

I. INTRODUCTION

3D object representations can now be obtained with relative ease through neural rendering methods such as NeRFs [1] and Gaussian Splatting [2]. By using multi-view pictures from calibrated cameras and optimizing a neural network, 3D object representations can be generated efficiently. However, if we want to edit a representation while retaining its geometric characteristics instead of generating a new one, manual editing is often the only viable option.

Although manual editing methods offer greater control to the user, this approach can be cumbersome and time consuming because the user has to process each piece of data themselves. Instruct-NeRF2NeRF [3] and more recently GaussianEditor [4] deliver impressive results in editing 3D scenes from multi-view images based on user-provided text prompts, but they also carry the inherent limitations of novel view methods. A significant drawback is the challenge of extracting meshes from continuous volumetric functions and Gaussian distributions. This limits their use in many cases where polygonal meshes are needed, as it can be challenging to maintain the characteristics of the edited object while incorporating new details.

We propose LEMON, a localized mesh editing method that deforms polygonal meshes based on given text instructions while preserving 3D consistency. Given multi-view images of the mesh, we use CLIPSeg [5] to generate vertex scores to determine important vertices in the given context. Afterwards, we use normal maps and shaded images of the mesh obtained by neural deferred shading [6] as conditions of ControlNet [7] to achieve 3D consistent edited images. Based on vertex scores, we mask edited images to localize our modifications. We iteratively update our multi-view image dataset with modified pictures and deform meshes based on these edits and vertex scores. Through this process, we ensure that the resulting meshes reflect the text-based instructions while maintaining the initial 3D structure.

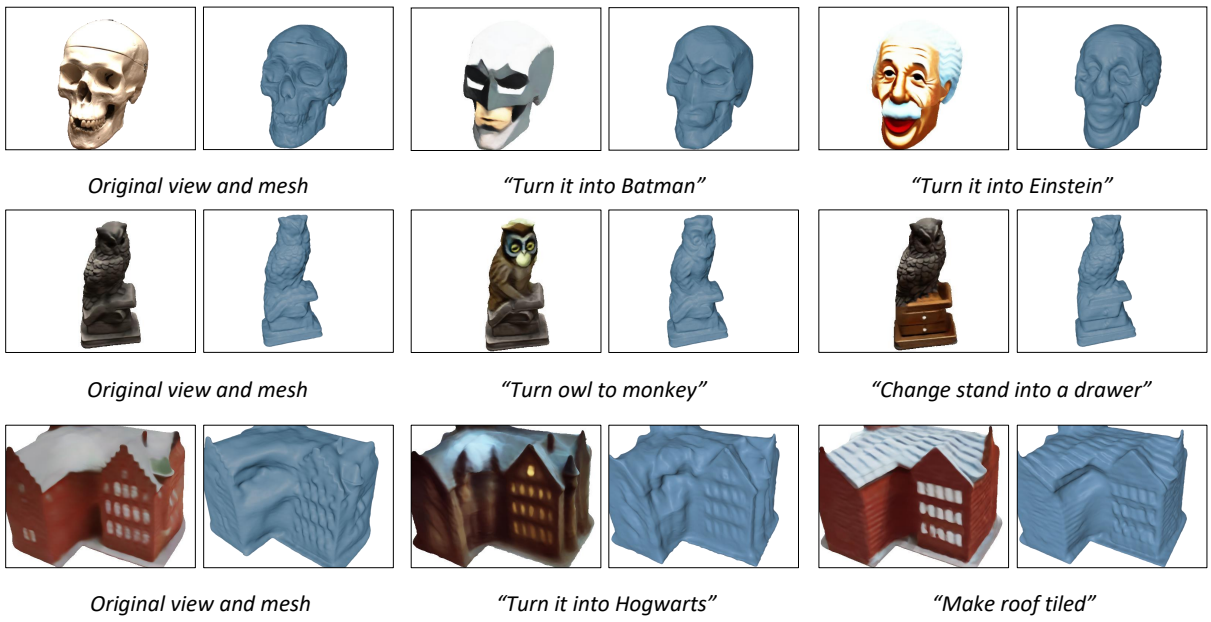

FIGURE 1. **We propose LEMON, a polygonal mesh editing method that takes multi-view images and user-provided text instructions as input and edits the mesh while preserving the geometric characteristics of the original mesh. Our method localizes accordingly to given instruction only changes the important parts of the mesh and provides a neural shader for the novel view.**

II. RELATED WORK

Neural Rendering: As deep learning techniques gain prominence in both computer vision and graphics, neural rendering [8] plays an important role in 3D reconstruction. One of the most popular techniques, neural radiance fields(NeRF) [1], are a type of volumetric scene representation based on a continuous volumetric function parameterized by a multilayer perceptron(MLP). NeRF can produce photorealistic renderings but is computationally expensive and slow. Additionally, most NeRF methods focus on view synthesis and rely on other techniques for surface extraction [9], which often results in less accurate meshes. In recent years, Gaussian Splatting [2] has emerged as an efficient alternative, offering faster rendering by representing scenes with a collection of Gaussian kernels, though it also struggles with precise surface reconstruction. Recent works, such as [10], show that it is possible to extract meshes from Gaussians. However, rather than being direct polygonal mesh editing, it is more of a post-processing step that adds extra time to the editing process. In contrast, we use neural deferred shading(NDS) [6], which integrates traditional mesh deformation with a neural shading pipeline, giving more precise and detailed 3D reconstructions from multi-view images.

Diffusion Models: Recent breakthroughs in diffusion models provide a more flexible approach to creating images from text or other conditioning data. Denoising Diffusion Implicit Models (DDIM) [11] iteratively remove noise from an initial noisy input to generate samples, while Contrastive Language–Image Pre-training (CLIP) [12] guides image generation based on text prompts. Stable Diffusion [13] is known for its high-quality output while InstructPix2Pix [14] extends this concept by using text prompts to edit existing images. However, expressing complex layouts through text prompts alone can be challenging in text-to-image models. ControlNet [7] because it addresses this issue by incorporating spatial conditioning controls, such as normal maps, which helps maintain the geometric characteristics of the object.

Mesh Editing: Despite the extensive research in mesh generation [15]–[17], there are relatively few studies focusing on 3D model editing, and these typically do not involve polygonal mesh editing. Instruct-NeRF2NeRF [3] introduced a text-based editing technique for NeRF scenes, where an image-conditioned diffusion model [14] generates new images based on a NeRF representation of a scene and the images used to construct it, which are then used to iteratively update the training dataset, guiding the NeRF to converge to the edited version. However, this technique relies on an implicit representation rather than directly manipulating the surface geometry, which limits its applicability to polygonal mesh editing. GaussianEditor [4] adopts a similar approach to Instruct-NeRF2NeRF but applies it to Gaussian splatting, allowing for text-based editing of Gaussian representations. However, for precise mesh extraction and reconstruction, additional pipelines such as SuGaR [10] are necessary to efficiently convert Gaussian splats into accurate 3D meshes. Inspired by [18], TextDeformer [19] is capable of creating global deformations on an input mesh based on a text prompt. However, direct manipulation of meshes through Jacobians is computationally costly and they do not provide any color information about the edited mesh. In contrast, our

method deforms a polygonal mesh using a fast neural shader combined with a diffusion model, giving the deformable mesh additional context about color and specularity.

III. METHOD

We propose LEMON, a fast and lightweight method for editing polygonal meshes by combining neural deferred shading and text-to-image models. We take a set of multi-view m images, $\mathcal{I} = \{I^1, \ldots, I^m\}$, from calibrated cameras, along with the corresponding masks of the interest region, $\mathcal{M} = \{M^1, \ldots, M^m\}$, and the camera parameters, as input. As in [6], we represent the surfaces as a triangle mesh, $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, consisting of vertices $\mathcal{V}$, edges $\mathcal{E}$, and faces $\mathcal{F}$. Additionally, we use a text prompt T to specify the editing instructions. The output of our method is an edited version of the initial mesh and a neural shader, based on the user-provided instruction. In this section, we first provide background information on the pipelines we used and explain how our method integrates them.

A. BACKGROUND

Neural Deferred Shading: NDS [6] is an analysis-by-synthesis mesh reconstruction method that optimizes a mesh and a neural shader simultaneously by using calibrated images and their corresponding masks as its input. If the initial mesh is not provided, the optimization process begins with a mesh, which is derived from the masks and resembles a visual hull [20]. Starting with a coarsely triangulated mesh, its resolution is gradually increased as optimization proceeds. In each upsampling iteration the surface is remeshed reducing the average edge length by half [21].

In the first step, the mesh is rasterized using a differentiable renderer [22], which provides triangle indices and barycentric coordinates for each pixel. By interpolating this information, a geometry buffer (g-buffer) is generated, which contains per-pixel positions, normals, and mask information. In second step g-buffer is processed by a learned shader, a MLP which resulting in an RGB color:

$$f_\theta(x, n, \omega_o) \in [0, 1]^3 \tag{1}$$

where θ represents the learnable parameters, $x \in \mathbb{R}^3$ is the position, $n \in \mathbb{R}^3$ denotes the normal and $\omega_o \in \mathbb{R}^3$ is the view direction relative to the center of the camera.

The neural shader and the initial mesh is optimized based on an objective function that balances the rendered appearance of mesh and geometric characteristics of the mesh:

$$\arg\min_{V,\theta} L_{\text{appearance}}\left(\mathcal{G}, \theta; \mathcal{I}, \mathcal{M}\right) + L_{\text{geometry}}\left(\mathcal{G}\right) \tag{2}$$

where $L_{\text{appearance}}$ consists of a shading loss, which computes distance between rendered image $\tilde{\mathcal{I}}$ and input image $\mathcal{I}$, and a mask loss, which computes distance between rendered mask $\tilde{\mathcal{M}}$ and input mask $\mathcal{M}$. L_{geometry} ensures to avoid undesired vertex configurations while deforming the mesh by minimizing the distance between a vertex and the average position of the neighbors and computing cosine similarity between neighboring face normals. Because mesh deformation and shader optimization happen simultaneously, NDS can preserve the geometric characteristics of opaque objects, even with varying materials and illumination, making it an ideal candidate for use in mesh editing.

ControlNet: Denoising diffusion models [23] generate data by incrementally removing noise, using the U-Net architecture [24] to provide both local and global context during the denoising process. Text-to-image diffusion models like Instruct-Pix2Pix [14] produce high-quality images from text-based instructions, encoding text into latent vectors using pretrained language models like CLIP [12]. However, these models often lack control over specific conditions and rely on broad assumptions about user preferences. Moreover, training new models can be time-consuming and burdensome, particularly with large datasets.

ControlNet [7] freezes the original parameters of a diffusion model and creates a trainable copy of its encoding layers, which are then trained on specific conditions. This method preserves the original diffusion model's quality and functionality while enabling greater control through defined conditions. In our case, extra conditions, such as normal maps, provide valuable supervision on geometry of the mesh, which is why we chose ControlNet as our diffusion model in our mesh editing pipeline.

B. LEMON

Starting with an initial polygonal mesh (along with a corresponding dataset of calibrated images and their camera parameters), our method combines a diffusion model with a neural deferred shading pipeline to deform the mesh based on given textual instruction. If the initial mesh is not provided, it is generated using standard neural deferred shading pipeline. [6]

We closely follow Instruct-NeRF2NeRF [3]'s editing process albeit with some differences. First we perform a pre-processing step to determine important regions of the mesh based on given instruction. During our editing process, we store another set of ground truth multi-view images, which we denote as $\mathcal{I}^v$. After every$d$-th iteration, we update our training dataset by replacing the images with their modified versions. Sometime later our mesh deforms into the desired edited version. The overview of our pipeline is given in Figure 2.

Vertex and Image Scores: For the first step of our pre-processing, we generate segmentation masks for every calibrated image using the provided text prompt. To keep it simple for the user and ensure compatibility with our image editing process using CLIP [12], we have selected CLIPSeg [5] to generate segmentation scores. These scores highlight the regions that are relevant to the given prompt as seen in Figure 3.

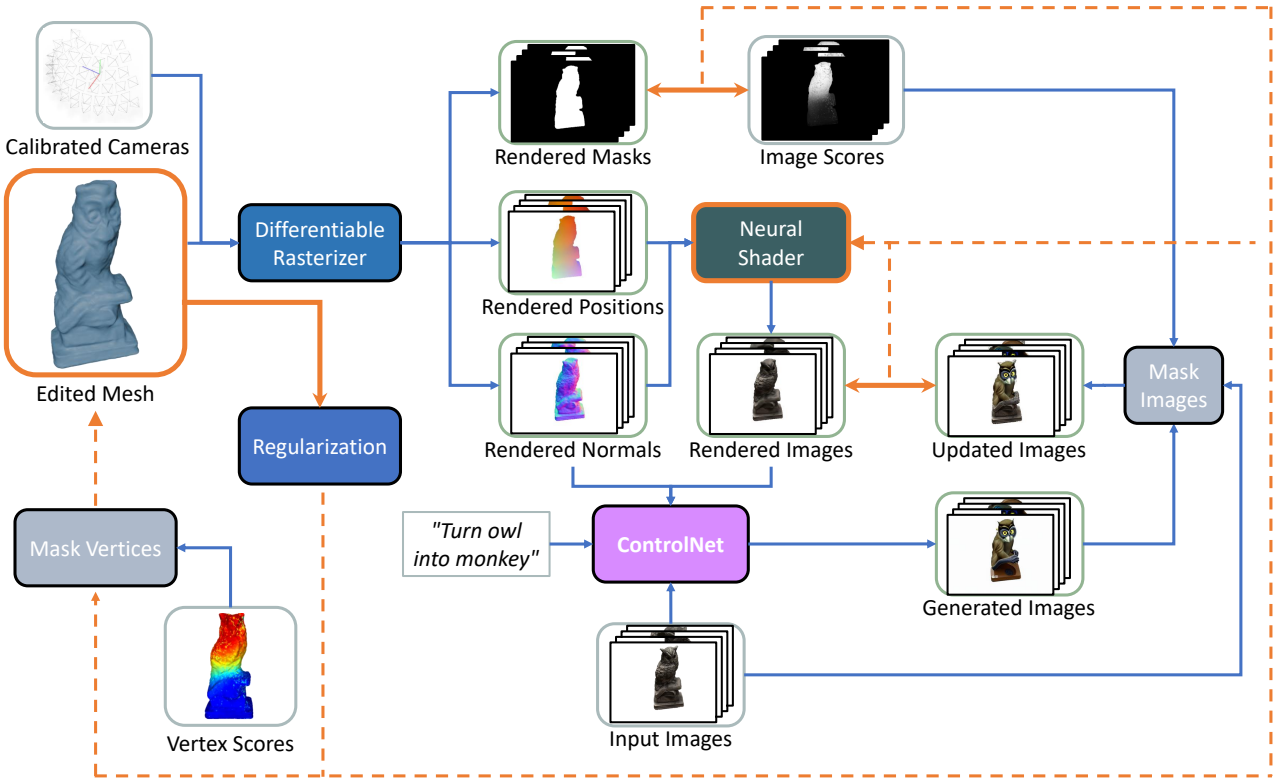

FIGURE 2. Pipeline of LEMON: We complement a multi-view mesh reconstruction model with a text-to-image model using localized features. After gathering vertex scores in our pre-processing step, we begin our editing process. Every d iterations new images are generated by the ControlNet [7], based on the prompt. The initial noise calculation of diffusion model is derived from a weighted sum of input images and rendered images, while it is conditioned on rendered normals and images of the mesh. The generated images are masked, and the masked regions are overlaid onto the original images, creating modified versions that are then used to update the dataset. Using vertex scores as a mask on the mesh, we update only the subset of vertices that is relevant to the prompt. By continuously updating the dataset with edited images, we deform the mesh to align with the user's request.

To assign these scores to the vertices, we use NvDiffrast [22] to render the mesh from given viewpoints and project segmentation scores onto the mesh surface. Each vertex's score is calculated by averaging the projected scores across all viewpoints. This aggregated scoring system makes the segmentation scores more accurate based on the geometry, effectively representing each vertex's importance in depicting the given prompt's feature distribution across the 3D mesh. After calculating the segmentation scores for each vertex, we project these scores back onto the image viewpoints. This process ensures that the segmentation scores are accurately reflected in the 2D image space.

In order to provide the user with more control over how much the mesh changes based on the context, the threshold $\tau \in [0, 1]$ is also taken as an input. By keeping scores larger than τ, we gather new image masks $\widetilde{\mathcal{M}} = \{\widetilde{M}^1, \ldots, \widetilde{M}^m\}$ and a subset of vertices $\widetilde{\mathcal{V}} \subset \mathcal{V}$ whose scores are higher than τ. We optimize only this subset while editing the mesh. We are doing these processes to ensure that changes occur only in the most important regions, localized according to the context of the instruction.

Editing Image: For the editing process, we use a pre-trained variant of ControlNet [7] with two modules, one

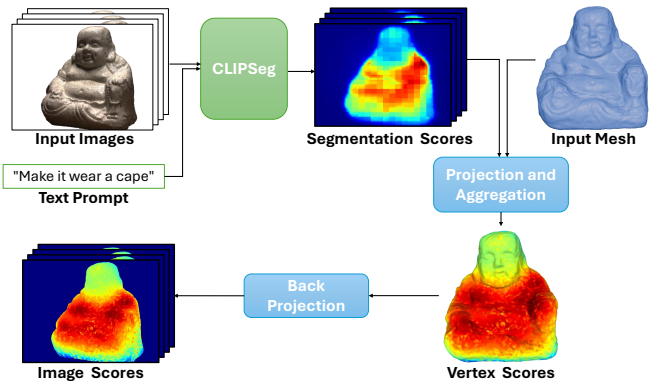

FIGURE 3. Image and vertex scoring process. Using CLIPSeg [5] we segment most important parts of the mesh given instruction

fine-tuned on Instruct-Pix2Pix [14] images and the other on normal maps. For the calculation of the initial noise z_0, Instruct-NeRF2NeRF follows the approach of SDEdit [25] where the rendered image of the current global 3D model plays a role in the output of the diffusion model. Instead of using only the rendered image, we use a weighted sum of

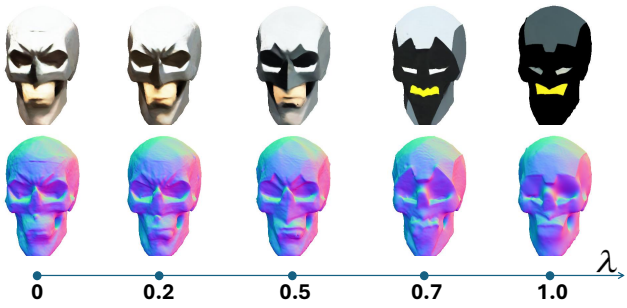

FIGURE 4. The effect of the latent weight hyperparameter λ on editing of the skull object from [26] for the "Turn it into Batman" prompt. Top of the skull has very bright shading, but the prompt requires the object to be darker. When λ is set to 0, only the ground truth image is used for the initial noise calculation, resulting in the lines in the skull to stay. If λ is too high, the rendered image may diverge to darker tones, leading to unintended edits.

the rendered image $\tilde{\mathcal{I}}^v$ and the original input image $\mathcal{I}^v$ from a given viewpoint v.

$$z_0 = \sqrt{\hat{\alpha_0}}(\lambda\mathcal{E}(\tilde{\mathcal{I}}^v) + (1-\lambda)\mathcal{E}(\mathcal{I}^v)) + \sqrt{1-\hat{\alpha_0}}\epsilon \quad (3)$$

where $\epsilon \sim \mathcal{N}(0,1)$, $\hat{\alpha_0}$ is the noise scheduling factor at timestep 0 and $\mathcal{E}$ is the VAE encoder of the latent diffusion model. Hyperparameter λ is the latent weight used to control the proportion of $\tilde{\mathcal{I}}^v$ and $\mathcal{I}^v$ latents. In the case of $\lambda = 1$, the initial noise calculation is the same as Instruct-NeRF2NeRF. Since our focus is on editing rather than mesh generation, we want our edited mesh to be constrained on the initial input pictures as well. However, in order to be restrained not too much on the initial mesh we also add rendered images to add some variance to the input noise. This prevents the diffusion model from diverging toward overly dark or bright images, as shown in Figure 4.

As conditions for the diffusion model, we include the encoded text prompt c_T, generated by [12] to guide our pipeline for editing. For the normal map module, we provide the normal map n_i of the g-buffer from a given viewpoint v as conditioning input. This leads to generated images that are more geometrically consistent, directly influenced by the optimized mesh $\mathcal{G}$. Instead of using ground-truth images for conditions as in Instruct-NeRF2NeRF, our Instruct-Pix2Pix module uses rendered images $\tilde{\mathcal{I}}^v$ generated by f_θ as its conditioning input. This results in generated images with greater variability, directly influenced by the optimized neural shader f_θ. In a sense our neural deferred shading pipeline and our conditioning inputs optimize each other, resulting in more consistent image generation.

Optimizing Process: We adapt the iterative dataset update of Instruct-NeRF2NeRF to our neural deferred shading pipeline. After generating the initial mesh, we create a modified input image using our diffusion model every d-th iteration, replacing the current iteration's input image with the modified image. We gradually optimize vertices of the mesh and neural shader based on new input images. The equation below demonstrates the image update process.

$$\mathcal{I}^v_{i+1} \leftarrow U_\theta\left(\mathcal{I}^v_0, \tilde{\mathcal{I}}^v_i, t; \tilde{\mathcal{I}}^v_i, {n_i}^v, c_T\right) \odot \widetilde{M}^v + \mathcal{I}^v_0 \odot (1 - \widetilde{M}^v) \quad (4)$$

where t is the noise level, randomly selected between $[t_{min}, t_{max}]$ and U_θ is the DDIM [11] sampling process with a set number of intermediate steps s between the initial timestep t and 0. As in NeRFs, this approach ensures consistent mesh transformations, providing desired modifications while preserving the mesh's structural integrity.

To localize our editing, we use $\widetilde{\mathcal{M}}$ to mask the important regions of the generated images. By overlaying the masked generated images onto the original input images, we ensure that modifications are consistently applied only to the relevant areas during the dataset update process.

Unlike NDS, which focuses on all vertices of the mesh, we optimize only the editing subset $\widetilde{\mathcal{V}}$. This approach allows the mesh to focus on editing only the parts that are relevant to the text input, avoiding unnecessary changes and extra computation. By updating only the relevant vertices, we maintain the overall structure and integrity of the original mesh, resulting in more contextually appropriate edits in geometry.

IV. EXPERIMENTS

We evaluate our method both qualitatively and quantitatively on nine objects from the DTU multi-view dataset [26], using materials from earlier works [6], [27], based on three text prompts.

For our method, we first reconstruct the object by following the approach in [6], then proceed with editing. Since there isn't any specific mesh editing method with a shader for direct comparison, we have selected a mesh deformation and novel view synthesis techniques for comparison. For rendering, we selected the Instruct-NeRF2NeRF [3] model from Nerfstudio [28] and GSEditor [4]. Since NeRF and Gaussian Splatting are designed for novel view synthesis rather than surface reconstruction, they are less suitable for mesh-based comparisons. However, recent works [10] show that high-quality meshes can be extracted from Gaussian Splatting. Therefore, we compared the renderings of Instruct-NeRF2NeRF and GSEditor, as well as the meshes of GSEditor generated by SuGaR [10], with our neural shader and mesh results. To qualitatively evaluate our meshes, we also used another mesh deformation-based editing method, TextDeformer [19], by applying an adjusted prompt to our initial mesh.

We also captured object scans using commodity hardware and edited them using our method to demonstrate its effectiveness on real-life data. In addition to those, we also used our method on the ShapeNet dataset [29] to test its ability to transform everyday objects. We provide our ShapeNet results and further qualitative comparison of models in the DTU dataset in our supplementary materials.

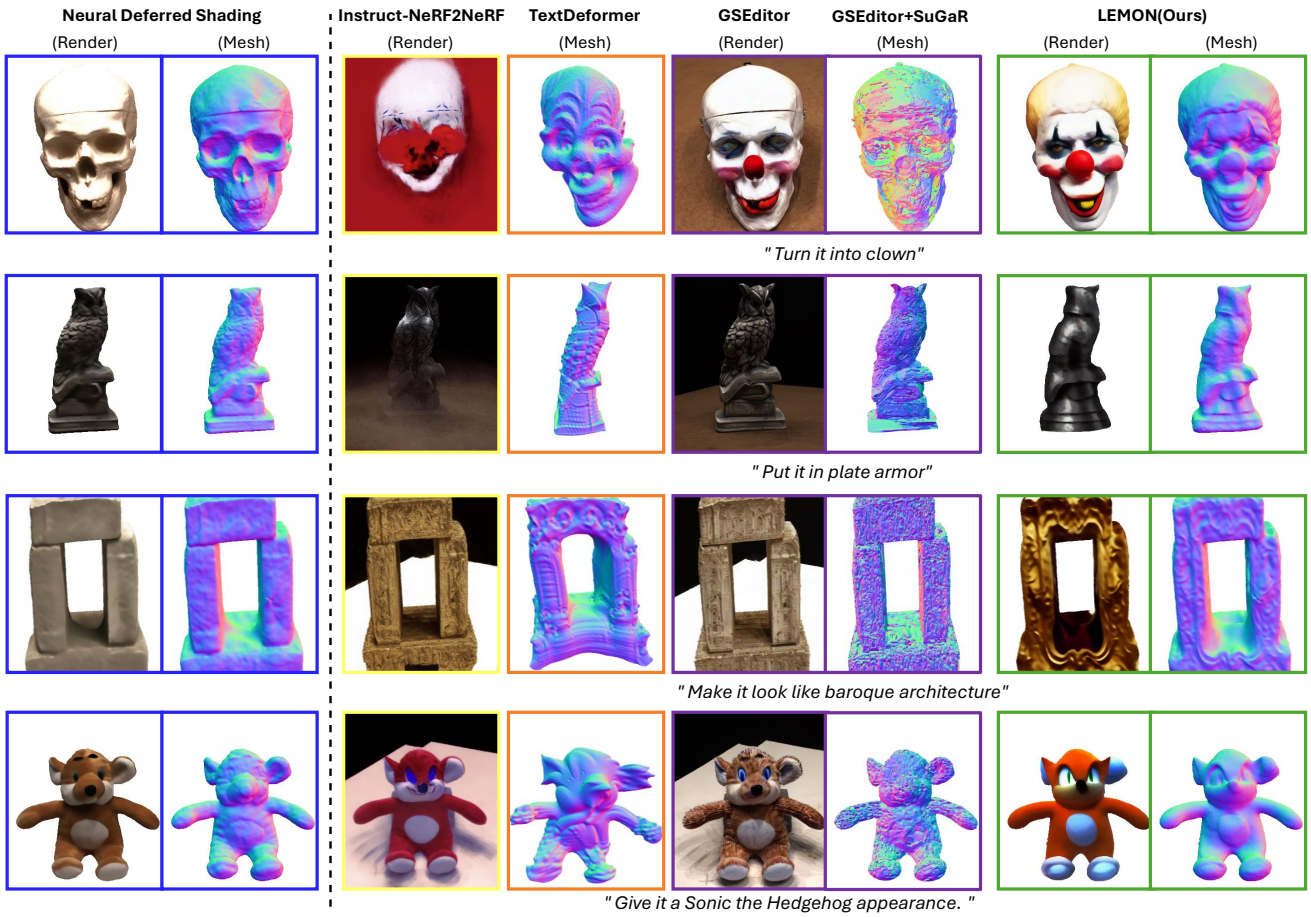

FIGURE 5. Editing results on the DTU dataset [26]. Blue boxes represent the initial mesh and shader reconstructed by neural deferred shading [6], providing a baseline. Orange boxes show the edited mesh results from TextDeformer while yellow boxes represent the edited views from Instruct-NeRFNeRF. Violet boxes represent renderings from GaussianEditor and their meshes extracted by SuGaR. LEMON achieves great results in both rendering and polygonal mesh quality.

A. IMPLEMENTATION DETAILS

For initial mesh reconstruction and subsequent optimization, we use the differentiable rendering pipeline from [22] on PyTorch [30], following the hyperparameters and loss functions from NDS to optimize our editing process.

The diffusion model's effectiveness and the consistency of its updates depend on several hyperparameters. For the initial noise calculation we set our latent weight $\lambda = 0.5$ and $[t_{min}, t_{max}] = [0.02, 0.98]$. Our denoising process is always done in 10 steps. The guidance scale for the text prompt is set at $s_T = 7.5$, while the conditioning scales for normal maps and rendered images are $s_I = 0.8$ and $s_n = 0.2$, respectively. These parameters determine the influence of each component in the denoising process. We generate an image and update our dataset with the modified image every 10 iterations. Most of the results shown in the paper follow these initial parameters, changing these parameters can add variation to the result. We train our method for a maximum of 8k iterations which takes around 15 minutes on a single NVIDIA A40. Further training could result in additional shape alteration and the inclusion of extra features.

B. QUALITATIVE RESULTS

In Figure 5, we present our main comparison on the DTU dataset [26]. Each row represents a specific editing case with it's corresponding text prompt listed below, while each column shows the output from a different method. In the case of mesh editing, our method retains the original mesh's geometric features while incorporating new refined details based on the text prompt. While TextDeformer achieves decent results, it struggles to preserve the original structure, often leading to distortions, such as the horizontal compression of the skull shown in the first row. Even in seemingly successful cases, as in the second row, TextDeformer tends to overedit the mesh, whereas our method maintains the basic structure and adds specific details in line with the prompt.

In the case of rendering, while Instruct-NeRF2NeRF achieves decent rendering results for simple edits such as "Put in plate armor," it may fall short when dealing with editing a geometrically complex figure such as the skull in the first row. Even in its most successful outcomes, Instruct-NeRF2NeRF seems to "color" the object rather than adding new geometrical features. GaussianEditor produces

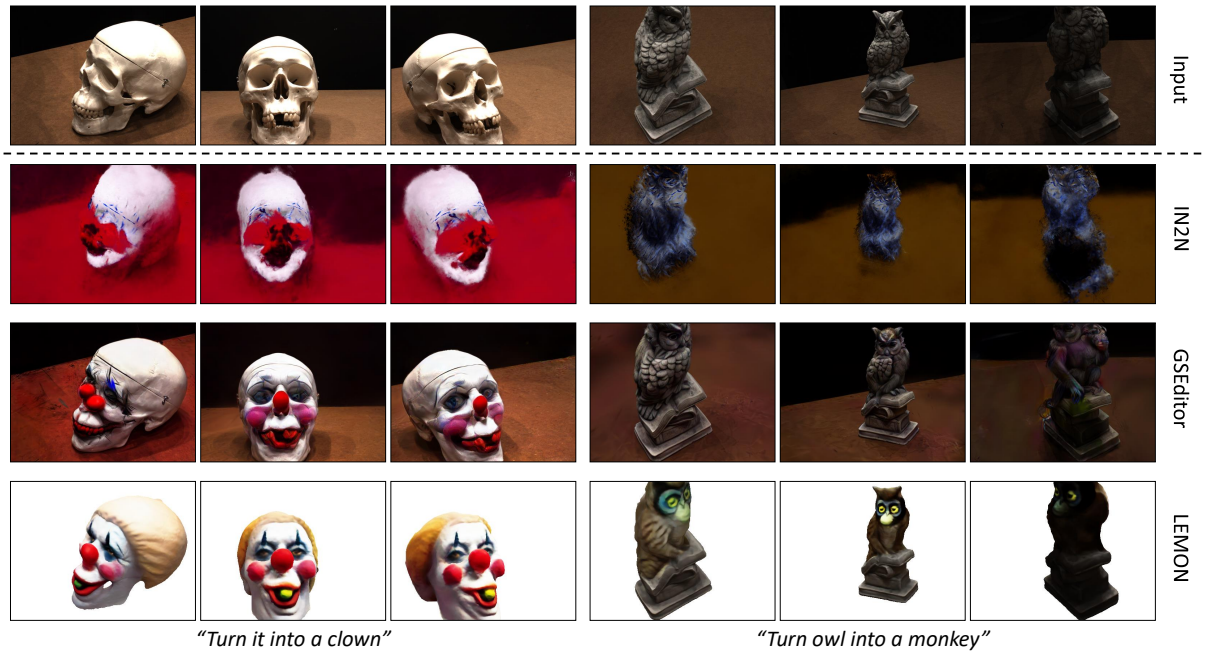

FIGURE 6. **Qualitative consistency results.**

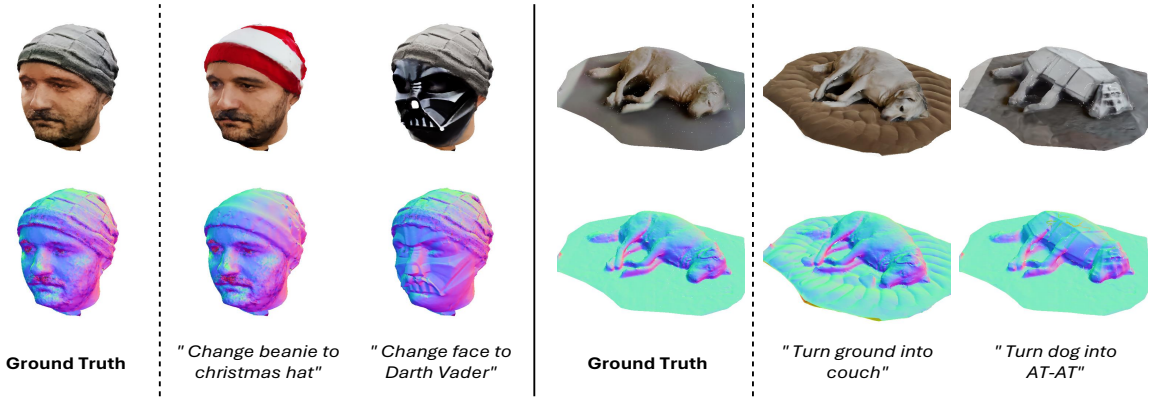

FIGURE 7. **Our local editing results on the real life scans.**

high-quality renderings of the edited object. However, it sometimes fails to generate the requested edits and shows less variation in the results. As opposed to the other two methods, our method maintains a concurrent relationship between mesh deformation and neural shader. This allows LEMON to adapt to significant structural changes while preserving the geometrical and shading characteristics of the object.

We also compare our shader's consistency with other rendering methods in Figure 6. As seen in the figure, other methods tend to show more inconsistency when there is a drift in camera motion. Although our shader may not produce the highest quality renderings, it is more consistent than the other methods. We believe this is because the mesh and neural shader optimization processes are intertwined, providing geometric consistency to each other.

Method	Time (Mins.) ↓	Memory (Peak GBs) ↓	CLIP Similarity ↑	User Study ↑
Instruct-NeRF2NeRF [3]	∼42	10.7	0.1118	6.94%
+Poisson Reconstruction [9]	∼3	6.0		2.2%
GaussianEditor [4]	∼10	9.7	0.1262	7.77%
+SuGaR [10]	∼45	6.5		8.88%
LEMON (Ours)	**∼15**	**6.7**	**0.2044**	**85.27%** **88.88%**

TABLE 1. **Quantitative results of our method.**

As final qualitative results, in Figure 7, we demonstrate the results of our object editing method on real-life scans captured by commodity hardware. We achieve results that are adequate in terms of localization, which shows the potential of our approach in handling objects captured from diverse environments.

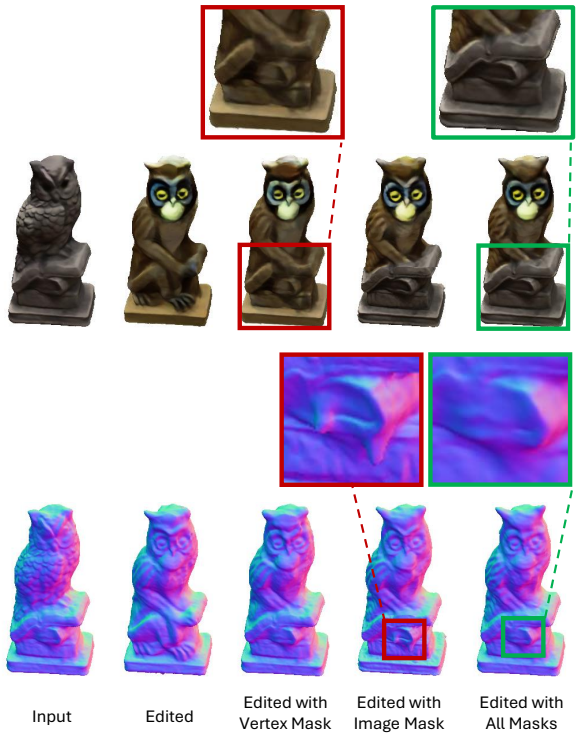

FIGURE 8. **Ablation studies of the masking.**

C. QUANTITATIVE RESULTS

Since editing is a subjective task, we rely on our qualitative evaluation more. We also use CLIP Directional Similarity [31] to measure the cosine similarity between the distances between pairs of images and their accompanying captions, evaluating this in all views of the object dataset. For the user study, we asked 20 participants to evaluate which editing outputs of the methods align best with the ground truth and the instruction text. Informed consent was obtained from all participants involved in the user study, ensuring their voluntary participation and understanding of the purpose and procedures of the study. In Table 1, the first line of each row represents the rendered images, while the second line corresponds to their normal maps for the methods, alongside the time spent and the consumption of GPU memory during the editing process.

Table 1 shows that LEMON is consistently preferred in the user study and outperforms other methods in CLIP directional similarity, indicating that our method more accurately follows the editing instructions. Since our pipeline simultaneously extracts the mesh and the neural shader, we also consider the memory and time consumed in mesh extraction for the other methods. We outperform Instruct-NeRF2NeRF even without considering the mesh extraction time. Although GaussianEditor is faster, extracting meshes with SuGar takes considerably more time. Therefore, we believe that our method strikes a good balance, achieving mesh and view synthesis together faster than all the other methods.

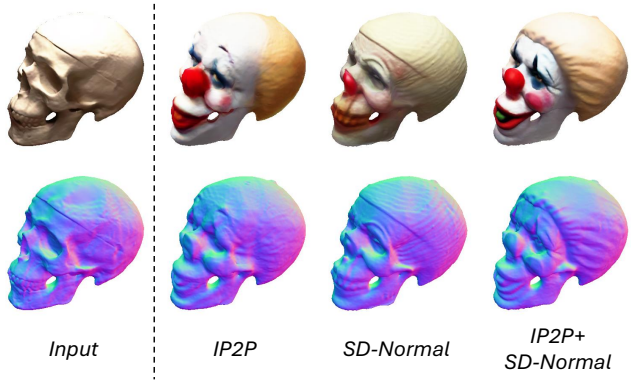

FIGURE 9. **Ablation studies of impact of text-to-image diffusion models on the mesh and neural shader.**

D. ABLATION STUDIES

Diffusion Model: The diffusion model plays a key role in our editing process by generating our modifications. As shown in Figure 9, while Instruct-Pix2Pix introduces variation to the meshes, it also loses key details, such as the line on the skull. In contrast, ControlNet's normal model preserves the geometric details of the shape but provides less variety, making fewer changes. To achieve the best of both worlds, we connect the two models through a Multi-ControlNet pipeline.

Vertice and Image Masking: Vertex and image masking processes are important contributions to our localized mesh editing. In Figure 8, we present a qualitative comparison of our masks in the example of the owl object from the DTU dataset. Our vertex masking allows us to avoid unnecessary topological changes in the mesh, while image masking during the editing process avoids redundant coloring, preserving original shading of the input in the process.

V. CONCLUSION

Even though our method is effective for editing meshes, it inherits some of the limitations from its underlying methods. Neural deferred shading uses object masks, limiting appearance loss to the masked regions. This restricts the diffusion model, making it more challenging to add new objects to the mesh. Additionally, we lack support for topology changes, which means that holes are only reconstructed if present in the initial geometry.

In this work, by concurrently optimizing our conditionings for the diffusion model and the learnable parameters of the neural shading, we achieved edited meshes guided by instructions on a localized region. Our results demonstrate that our pipeline delivers finely-edited meshes in a timelier manner compared to existing methods. We look forward to future work that further explores this problem, as mesh editing remains an area in need of advancement within 3D generative methods.

REFERENCES

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 1–14, 2023.

[3] A. Haque, M. Tancik, A. A. Efros, A. Holynski, and A. Kanazawa, "Instruct-nerf2nerf: Editing 3d scenes with instructions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 19 740–19 750.

[4] Y. Chen, Z. Chen, C. Zhang, F. Wang, X. Yang, Y. Wang, Z. Cai, L. Yang, H. Liu, and G. Lin, "Gaussianeditor: Swift and controllable 3d editing with gaussian splatting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 21 476–21 485.

[5] T. Lüddecke and A. Ecker, "Image segmentation using text and image prompts," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 7086–7096.

[6] M. Worchel, R. Diaz, W. Hu, O. Schreer, I. Feldmann, and P. Eisert, "Multi-view mesh reconstruction with neural deferred shading," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 6187–6197.

[7] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3836–3847.

[8] A. Tewari, J. Thies, B. Mildenhall, P. Srinivasan, E. Tretschk, W. Yifan, C. Lassner, V. Sitzmann, R. Martin-Brualla, S. Lombardi *et al.*, "Advances in neural rendering," in *Computer Graphics Forum*, vol. 41, no. 2. Wiley Online Library, 2022, pp. 703–735.

[9] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, no. 4, 2006.

[10] A. Guédon and V. Lepetit, "Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 5354–5363.

[11] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.

[12] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[13] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[14] T. Brooks, A. Holynski, and A. A. Efros, "Instructpix2pix: Learning to follow image editing instructions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 18 392–18 402.

[15] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall, "Dreamfusion: Text-to-3d using 2d diffusion," *arXiv preprint arXiv:2209.14988*, 2022.

[16] N. Mohammad Khalid, T. Xie, E. Belilovsky, and T. Popa, "Clip-mesh: Generating textured meshes from text using pretrained image-text models," in *SIGGRAPH Asia 2022 conference papers*, 2022, pp. 1–8.

[17] C.-H. Lin, J. Gao, L. Tang, T. Takikawa, X. Zeng, X. Huang, K. Kreis, S. Fidler, M.-Y. Liu, and T.-Y. Lin, "Magic3d: High-resolution text-to-3d content creation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 300–309.

[18] N. Aigerman, K. Gupta, V. G. Kim, S. Chaudhuri, J. Saito, and T. Groueix, "Neural jacobian fields: Learning intrinsic mappings of arbitrary meshes," *arXiv preprint arXiv:2205.02904*, 2022.

[19] W. Gao, N. Aigerman, T. Groueix, V. Kim, and R. Hanocka, "Textdeformer: Geometry manipulation using text guidance," in *ACM SIGGRAPH 2023 Conference Proceedings*, 2023, pp. 1–11.

[20] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, no. 2, pp. 150–162, 1994.

[21] M. Botsch and L. Kobbelt, "A remeshing approach to multiresolution modeling," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 2004, pp. 185–192.

[22] S. Laine, J. Hellsten, T. Karras, Y. Seol, J. Lehtinen, and T. Aila, "Modular primitives for high-performance differentiable rendering," *ACM Transactions on Graphics (ToG)*, vol. 39, no. 6, pp. 1–14, 2020.

[23] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, "Diffusion models in vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 2015, pp. 234–241.

[25] C. Meng, Y. He, Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon, "Sdedit: Guided image synthesis and editing with stochastic differential equations," *arXiv preprint arXiv:2108.01073*, 2021.

[26] R. Jensen, A. Dahl, G. Vogiatzis, E. Tola, and H. Aanæs, "Large scale multi-view stereopsis evaluation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2014, pp. 406–413.

[27] L. Yariv, Y. Kasten, D. Moran, M. Galun, M. Atzmon, B. Ronen, and Y. Lipman, "Multiview neural surface reconstruction by disentangling geometry and appearance," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[28] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja *et al.*, "Nerfstudio: A modular framework for neural radiance field development," in *ACM SIGGRAPH 2023 Conference Proceedings*, 2023, pp. 1–12.

[29] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[31] R. Gal, O. Patashnik, H. Maron, G. Chechik, and D. Cohen-Or, "Stylegan-nada: Clip-guided domain adaptation of image generators," 2021.