6.8410: Shape Analysis, 2023
Instructor: Justin Solomon
TA: Theia Henderson

NeuStyle: A Cubic Stylization Approach for Reconstructed Signed Distance Fields

David Charatan[1] and Peter Werner[1]

[1]MIT Department of Electrical Engineering and Computer Science

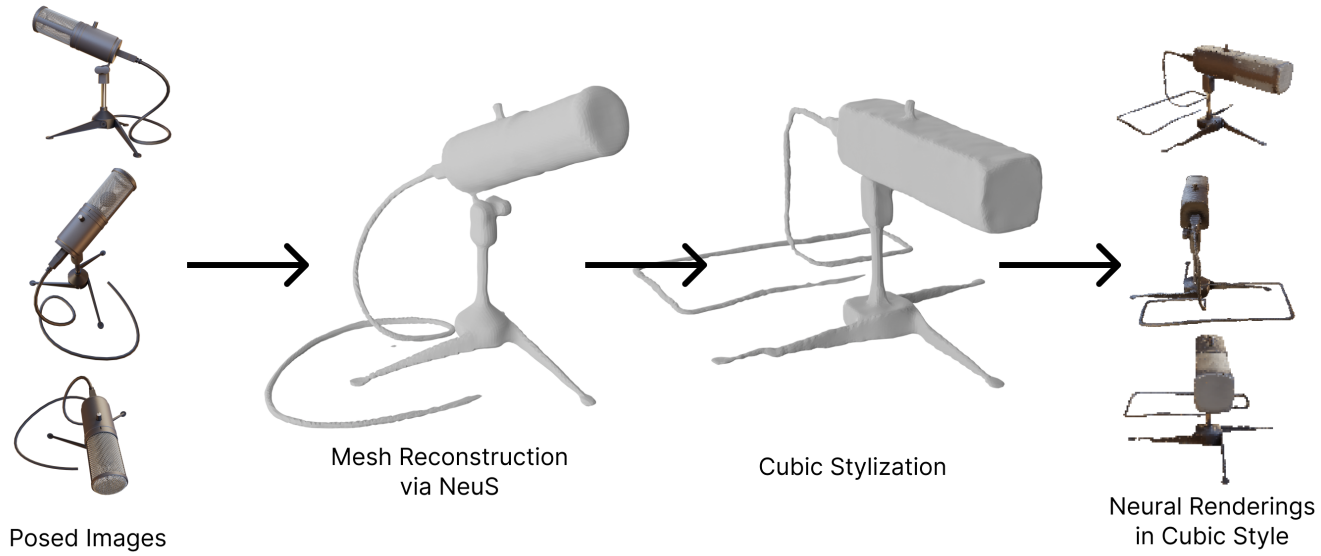


Figure 1: An overview of our method. Given posed input images, we generate a mesh reconstruction via NeuS. We then cubically stylize this reconstruction, extract the resulting deformation, and produce novel views of the scene via deformation-aware neural rendering.

Abstract

We present NeuStyle, a method for applying mesh processing operations to neural radiance fields (NeRFs). Given a set of posed input images, our method produces an undeformed mesh, a deformed mesh with mesh processing operations applied, and NeRF-style neural renderings in deformed space. As a proof of concept, we implement cubic stylization and apply it to radiance fields generated from the NeRF-Synthetic dataset.

1. Introduction

Since their introduction, neural radiance fields (NeRFs) have become the de facto standard for 3D reconstruction. While these methods are capable of photorealistic novel view synthesis, they produce a volumetric representation that cannot be converted into a high-quality mesh. This is inconvenient because it prevents a wide range of existing mesh processing algorithms and tools from being applied to NeRF's outputs. Fortunately, a variant of NeRF called NeuS [WLL*23], which is based on signed distance functions (SDFs), addresses this issue. NeuS is able to produce high-

quality meshes which can be used in standard geometry processing pipelines.

In this report, we leverage NeuS' ability to generate meshes to apply standard geometry processing algorithms to NeRF's volumetric representation. In particular, we reconstruct a set of images using NeuS, process the resulting mesh, and then apply the resulting deformation to NeuS' underlying volumetric formulation. As a proof of concept, we choose cubic stylization [LJ19] as our geometry processing operation, which allows us to effectively re-render NeRFs as if they were cubically stylized meshes.

Our key technical contribution is a method which allows us to extrapolate mesh deformations, which are defined on surfaces, to the surrounding 3D space. This is necessary because NeuS' rendering formulation requires sampling 3D space near (but outside) the surface. Our method is based on linear approximations of deformations near surfaces, since we observe that accurately deforming empty space is unnecessary for high-quality rendering. Overall, our contributions are:

- Implementing NeuS.
- Implementing cubic stylization.
- Fusing the above approaches using a simple ray-deformation approach that makes applying mesh processing algorithms to NeRFs possible.

2. Related Work

In this class project we recombine existing techniques to produce a novel pipeline for directly stylizing NeRFs. These techniques are summarized in the following paragraphs:

1. **Neural Radiance Fields (NeRF):** NeRFs have emerged as a popular method for 3D reconstruction and novel view synthesis. They excel in generating photorealistic images but suffer from the limitation of producing volumetric representations that are incompatible with traditional mesh processing algorithms [MST*21]. While NeRFs have achieved impressive results, their inability to generate meshes hinders their broader applicability.
2. **Neural Implicit Surfaces (NeuS):** NeuS, a variant of NeRF, addresses the limitation of volumetric representations by utilizing signed distance functions (SDFs) to generate high-quality meshes [WLL*23]. Unlike NeRF, which directly predicts a volumetric density at each point, NeuS predicts a signed distance at each point. This signed distance is then converted into a volumetric density via a function that ensures that renderings of the resulting density field are unbiased representations of the underlying surface. Thus, although NeuS' training procedure closely matches that of the original NeRF paper, NeuS can produce high-quality meshes via marching cubes [LC87]. The availability of meshes allows for the utilization of standard geometry processing algorithms and tools, enabling NeuS outputs to integrate into existing geometry processing pipelines.
3. **Cubic Stylization:** The work by Liu et al. introduces cubic stylization, a geometry processing procedure based off of as-rigid-as-possible mesh deformations [SA07], that produces a cubically stylized version of the mesh, that retains many of the original geometric features.

3. Technical Approach

In this section we describe our pipeline to training, stylizing, and rendering a NeuS. The following section is structured in three parts: In Section 3.1 we detail how we construct and train and extract a mesh from our NeuS, in Section 3.2 we describe our stylization approach, and in Section 3.3 our rendering approach for the stylized NeuS is described.

3.1. NeuS Training

We use the training procedure described in [WLL*23] to train our NeuS. In this section we give brief summary of the procedure.

Our NeuS $\mathcal{N}_\theta = (\mathcal{S}, \mathcal{C})$ is a tuple of two neural networks with parameters θ, which take points $p \in \mathbb{R}^3$ and directions $d \in R^3$ as inputs and output predictions $\mathcal{S}(p)$ and $\mathcal{C}(p)$ of the signed distance function and the color field of the scene at the point (p,d).

The training data is a set of posed images of a scene with both camera intrinsics and extrinsics. At training time, batches of rays are extracted from the training images by randomly selecting pixels, and subsequently computing ray origins $r_o \in \mathbb{R}^3$ and directions $r_d \in \mathbb{R}^3$ based on the camera intrinsics and extrinsics. We update the parameters θ by evaluating:

$$\mathcal{N}(r_o^i + \frac{j d_{\max}}{R} r_d^i, r_d^i) = (c_j^i, s_j^i) \text{ for } j = 0, 1, \ldots, R \tag{1}$$

$\mathcal{N}$ at R evenly spaced samples along the ray, reconstructing the pixels, and performing gradient descent on the reconstruction loss given in [WLL*23]. For a single ray r_o^i, r_d^i this reconstruction loss reads

$$L^i = \underbrace{\left|\left|C^i - \sum_{j=0}^{R} \alpha(s_j^i) c_j^i\right|\right|_2^2}_{\text{Reconstruction Loss}} + \lambda \underbrace{\sum_{j=0}^{R} \left(1 - \left|\left| \frac{\partial \mathcal{S}}{\partial p}\Big|_{p = r_o^i + \frac{j d_{\max}}{R} r_d^i} \right|\right|_2\right)^2}_{\text{Eikonal Loss}} \tag{2}$$

where C^i is the original color value of the pixel, and the function α is the Gaussian alpha compositing function from [WLL*23]. We perform stochastic gradient descent on L

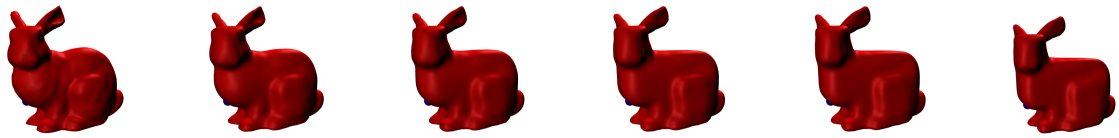
$$L = \frac{1}{B} \sum_{i=1}^{B} L^i \tag{3}$$

for batches of rays using ADAM [KB14]. A figure with NeuS renderings shown throughout the training process on the *chair* scene from the NeRF-Synthetic dataset is shown in figure 3.

3.2. Stylization

We base our cubic stylization approach on the technique introduced in [LJ19]. Given a triangle mesh $\mathcal{T} = (\mathcal{V}, \mathcal{F})$ they describe a procedure to minimize the following cost in (4).

$$J(\mathcal{V}, \mathcal{F}, R, \hat{\mathcal{V}}) = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}(i)} w_{ij} \underbrace{||R_i d_{ij} - \hat{d_{ij}}||_2^2}_{\text{ARAP}} + \underbrace{\lambda a_i ||R_i n_i||_1}_{\text{Cube target}} \tag{4}$$



(a) Original (b) Iteration 1 (c) Iteration 2 (d) Iteration 3 (e) Iteration 4 (f) Iteration 5

Figure 2: Progression of cubic stylization with $\lambda = 0.35$ on the Stanford Bunny mesh. The initial mesh on the left progressively becomes more axis-aligned in five completed global local alternations. Typically the resulting deformation converged after 5 to 10 alternations.

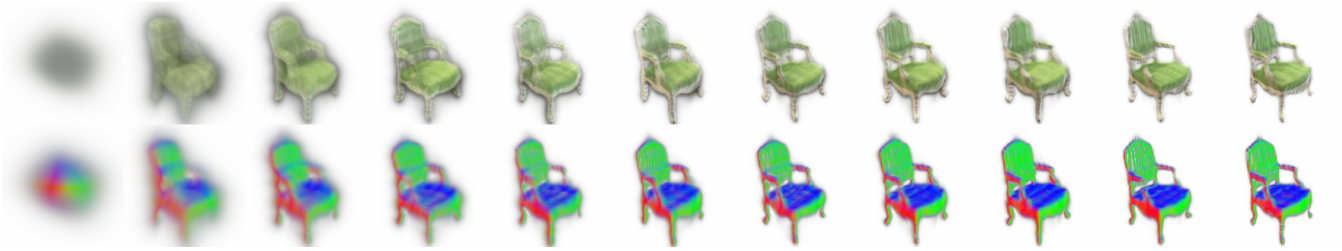
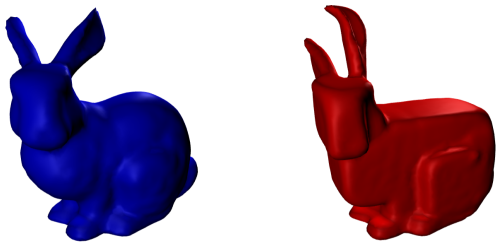


Figure 3: Evolution of rendered (color) output and estimated surface normals during the beginning of NeuS training on the *chair* scene from the NeRF-Synthetic dataset.

The cost is composed of two terms. The first term is the as-rigid-as-possible cost (ARAP) from [SA07]. Minimizing this cost subject to constraints such as pinned vertices, tries to find close to rigid deformation of the mesh that matches the constraints. In practice, this results in the mesh retaining a lot of its shape features, while also behaving in the user-imposed way. Minimizing this cost searches for the best possible a per-vertex rotation matrix and vertex locations, such that the distance of the neighbours $j \in \mathcal{N}(i)$ mapped by a rigid rotation to their deformed counterparts is minimized. The weights are the cotangent weights as derived in [SA07] to make the energy as mesh independent as possible. The stylized cost $\mathcal{J}$ adds an additional term that penalizes the L1-norm of the mapped surface normal. The L1-norm encourages sparse solutions [HTFF09] and hence encourages the mapped neighbourhoods to be axis-aligned, giving the mesh a cubic look. See Figure 4 for example.



(a) (b)

Figure 4: (a) Undeformed Stanford Bunny. (b) Cubicly stylized Stanford Bunny using $\lambda = 0.25$ with the left ear and base pinned to pre-fixed locations.

The authors in [LJ19] adapt a similar local-global approach to [SA07] to minimizing $\mathcal{J}$, which we employ in this project. First, all the per-vertex rotation matrices are fitted using ADMM [BPC*11]. This maps to the following iteration procedure until convergence,

$$R_i^{k+1} \leftarrow \underset{R_i \in SO(3)}{\textbf{argmin}} \frac{1}{2}||R_i D_i - D_i||_{W_i}^2 + \frac{\rho_k}{2}||R_i n_i - z_i^k + u^k||_2^2 \quad (5)$$

$$z_i^{k+1} \leftarrow \underset{z \in \mathbb{R}^3}{\textbf{argmin}}\, \lambda a_i ||z_i||_1 + \frac{\rho_k}{2}||R_i n_i - z^k + u^k||_2^2 \quad (6)$$

$$\tilde{u}^{k+1} \leftarrow u_k + R_i^{k+1} n_i - z^{k+1} \quad (7)$$

$$\rho^{k+1}, u^{k+1} \leftarrow \text{rescale}(\rho^k, \tilde{u}^{k+1}) \quad (8)$$

where the first problem is an *orthogonal procrustes* problem, and can be solved via singular value decomposition (see [LJ19]). The second step is an instance of the lasso shrinkage problem, and can be solved with the *shrinkage step* in [BPC*11]. Furthermore, in the last step the multipliers are rescaled according to [BPC*11].

The global step updates the vertex locations after suitable rotation matrices have been found. Since the cube target cost is constant with respect to the deformed vertex locations, we can apply the same approach as in [SA07]. This means we need to solve the following

$$\sum_{j \in \mathcal{N}(i)} w_{ij}(\tilde{v}_i - \tilde{v}_j) = \sum_{j \in \mathcal{N}(i)} \frac{w_{ij}}{2}(R_j + R_i)(v_i - v_j) \quad (9)$$

system of equations with additional constraints on vertices. This results in the linear system,

$$\mathbf{L}\tilde{V} = b \quad (10)$$

where $\mathbf{L}$ is the Laplace-Beltrami operator. Note that pinning a specific vertex just means we need to update the right hand side ac-

cordingly and then remove the corresponding rows and columns of the system.

We implement this algorithm in Python3 using the python bindings for the igl library [JP*18] for select mesh computations. See Figure 2 for a typical progression of the alternations.

3.3. Deformed NeuS Rendering

In order to accomplish this we cast rays $\mathcal{R} = (\mathcal{O}, \mathcal{D})$, given in origin and unit-vector direction tuples, onto the deformed mesh $\mathcal{T}_{\text{cube}} = (\mathcal{U}, \mathcal{F})$. For every ray $r_i = (o_i, d_i)$ we only detect the first intersection p_i. About the intersection point we construct a small sub-ray, as a collection of samples $\mathcal{S}$ that protrude equally into and out of the surface. In order to map these samples into the original radiance field, we rotate the samples about the nearest vertex's inverse rotation obtained from the stylization and shift the subray out to the vertex. Formally

$$s_j^{\text{render}} = R_{i^*}^T s_j \; \forall s_j \in \mathcal{S} \tag{11}$$

where i^* denotes the nearest vertex $u_{i^*} \in \mathcal{U}$. This procedure is illustrated in Figure 5. In order to determine the pixel value of the rendered frame, the NeuS queried on the transformed subrays s_j^{render}. Note that this approach ensures that the NeuS is sampled with a consistent incidence angle around the point of intersection with the mesh, which is empirically where most of the color information is stored.

4. Results

We applied our method to various models in the Nerf-Synthetic dataset, plus an additional scene of a teddy bear that we rendered. We show mesh results from NeuS, which are extracted using marching cubes, in figure 8. We show the corresponding mesh results from cubic stylization in figure 9. For the scenes with multiple objects, we find that our approach fails if not every single object is pinned, because the ARAP deformation step results in solving an ill conditioned or even rank deficient linear system. This is an obvious consequence of the ARAP cost not constraining relative distances between disjoint meshes. As a result, we only apply cubic stylization to the largest connected component (by number of vertices) in each mesh. This works well for most scenes, but fails on scenes with thin structures (e.g., *ficus*, *drums*) or with multiple objects (e.g., *materials*).

We show neural renderings from NeuS in figure 6. In particular, we observe visually consistent renderings of the deformed objects using our method on the scenes *teddy bear*, *mic*, and *chair*. We note that specular effects (e.g., the glint on the microphone) are convincingly reproduced by our method, although this is difficult to ascertain from static images.

We show failure cases of neural rendering in figure 7. These failure cases occur because the underlying deformed meshes are incomplete. However, we also noticed color artifacts at occluded locations when viewing from the wrong angles. This makes sense, as the NeuS does not get any meaningful training data from those perspectives.

5. Conclusion

In this class project we presented a method for stylizing scenes given as a series of posed images. Our approach trains a scene specific NeuS, extracts a mesh using marching cubes, stylizes it by finding an as-rigid-as-possible deformation that attempts to make every surface axis-aligned, and renders the resulting stylized mesh by casting rays in the deformed space and querying the color field at corresponding locations in the undeformed space. As result, we get consistently colored stylized images as shown in Section 4. In principle this can be applied to any general set of posed images, although, manual selection of what part of the mesh to stylize will still be necessary in the general setting.

6. Future Directions

While our method is based on cubic stylization, one could in principle use our method to render NeRFs with a variety of deformations. For example, one could capture a mesh of a human using NeuS, apply off-the-shelf animation tools to create animated mesh frames, and then render a NeRF with the corresponding mesh deformations. We thus see great potential for our method in shape-based stylization and animation tasks.

7. Contributions

Peter worked on the cubic stylization and ray deformation code. David worked on the NeuS training and deformed rendering code. Both Peter and David contributed to the report and presentation.

References

[BPC*11] BOYD S., PARIKH N., CHU E., PELEATO B., ECKSTEIN J., ET AL.: Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning 3*, 1 (2011), 1–122. 3

[HTFF09] HASTIE T., TIBSHIRANI R., FRIEDMAN J. H., FRIEDMAN J. H.: *The elements of statistical learning: data mining, inference, and prediction*, vol. 2. Springer, 2009. 3

[JP*18] JACOBSON A., PANOZZO D., ET AL.: libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/. 4

[KB14] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). 2

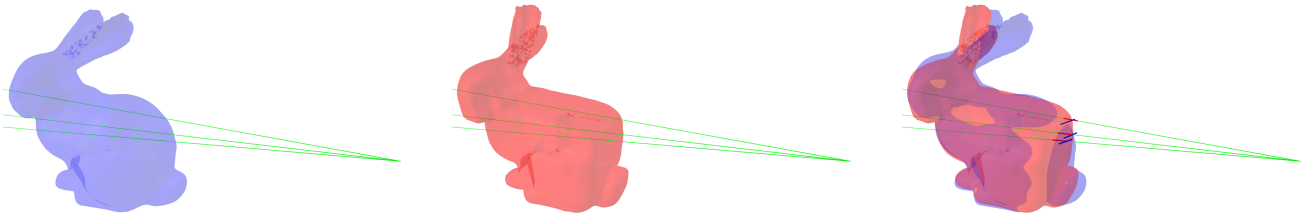
[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1987), SIGGRAPH '87, Association for Computing Machinery, p. 163–169. URL: https://doi.org/10.1145/37401.37422, doi:10.1145/37401.37422. 2

[LJ19] LIU H.-T. D., JACOBSON A.: Cubic stylization. *arXiv preprint arXiv:1910.02926* (2019). 2, 3

[MST*21] MILDENHALL B., SRINIVASAN P. P., TANCIK M., BARRON J. T., RAMAMOORTHI R., NG R.: Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM 65*, 1 (2021), 99–106. 2

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Symposium on Geometry processing* (2007), vol. 4, pp. 109–116. 2, 3

[WLL*23] WANG P., LIU L., LIU Y., THEOBALT C., KOMURA T., WANG W.: Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction, 2023. arXiv:2106.10689. 1, 2



(a) (b) (c)

Figure 5: The subrays around the surface of the deformed mesh are mapped to subrays on the undeformed mesh using the closest rigid deformations obtained from the stylization step and subsequently used for sampling the NeuS. (a) Rays cast onto undeformed mesh. (b) Rays cast onto stylized mesh. (c) Subrays mapped from the deformed (red) to the undeformed mesh (blue). Note the consistent incidence angle.

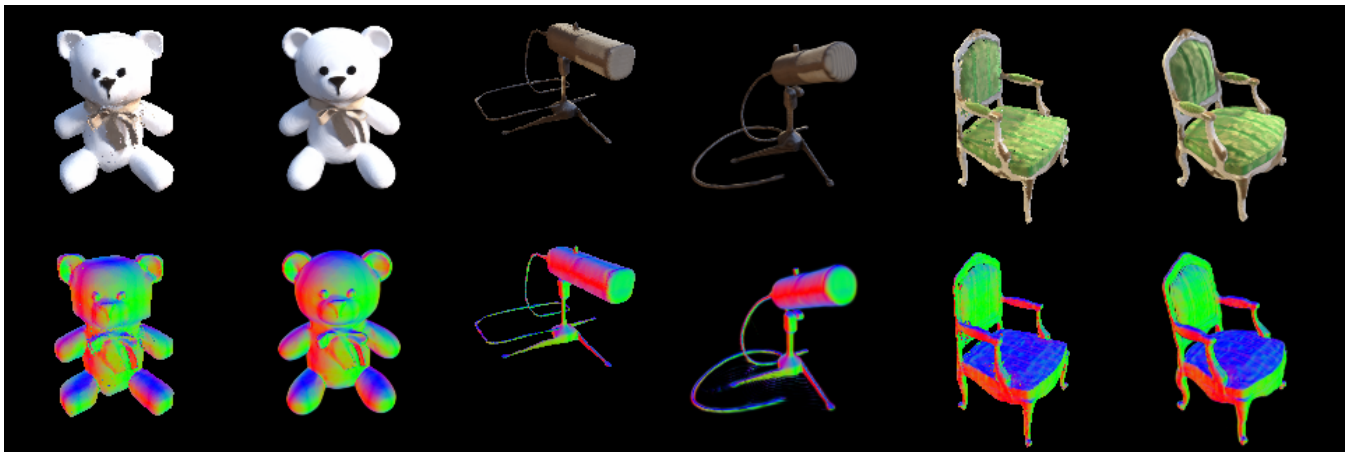


Figure 6: Examples of successful renderings of the NeRF-Synthetic dataset (*teddy bear*, *mic*, and *chair*) with cubic stylization applied. The top row shows RGB renderings, while the bottom row shows surface normals. For each object, the left column shows the deformed (cube-stylzed) rendering, while the right column shows the original NeuS rendering.

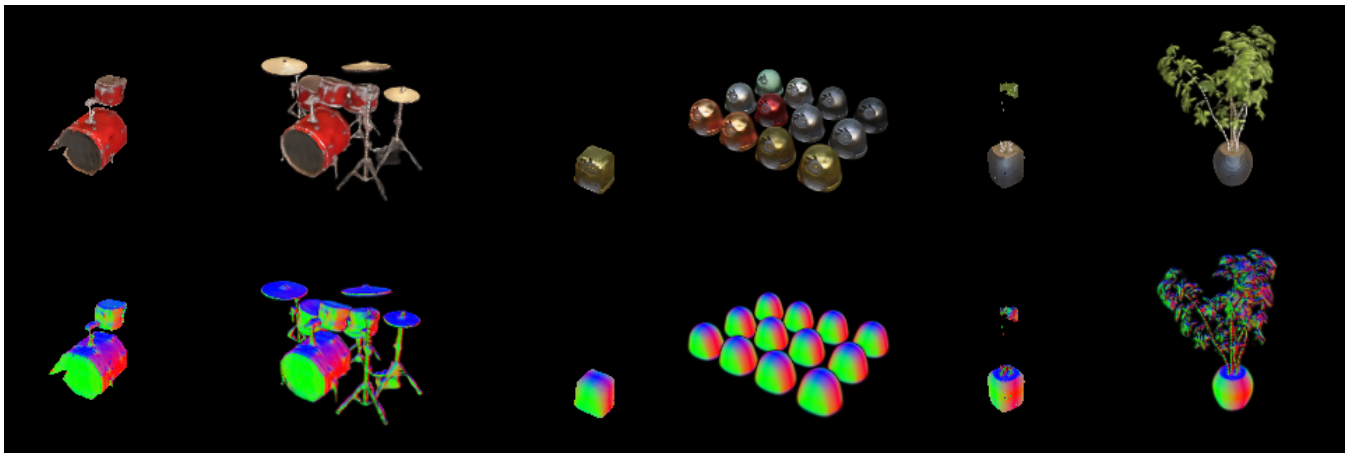


Figure 7: Examples of failed renderings of the NeRF-Synthetic dataset (*drums*, *materials*, and *ficus*) with cubic stylization applied. The top row shows RGB renderings, while the bottom row shows surface normals. For each object, the left column shows the deformed (cube-stylzed) rendering, while the right column shows the original NeuS rendering. Since cubic stylization cannot handle meshes with disconnected components, the meshes used for deformed rendering for these scenes are incomplete, causing the renderings to also be incomplete. Note, however, that the components of the mesh that are rendered still have reasonable cubic stylization applied.

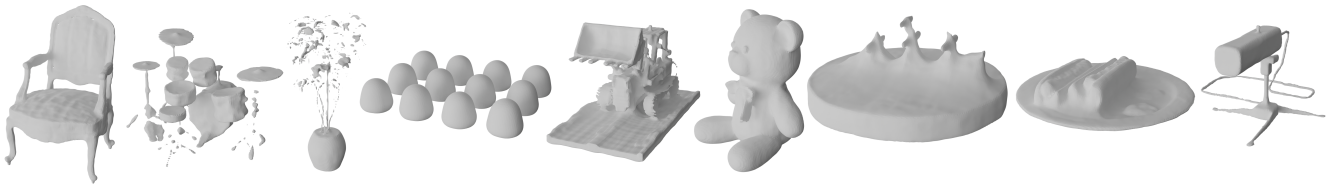


Figure 8: Mesh outputs from NeuS for the scenes in the NeRF-Synthetic dataset (plus our custom teddy bear scene). We apply cubic stylization to these meshes as part of our styilization and rendering pipeline.

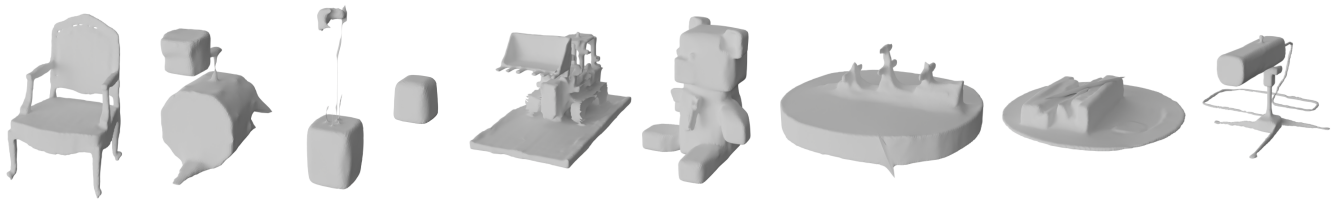


Figure 9: The scenes in the NeRF-Synthetic dataset (plus our custom teddy bear scene) with cubic stylization applied. Some scenes, like *ficus* and *materials*, are incomplete because their original meshes have disconnected components. Our cubic stylization algorithm discards all but the largest connected component.