SRMCV Project: Extending the Diffeomorphic Neural Reconstruction Approach

Alexander Fuchs Zehranaz Canfes Vasiliki Papadouli
Technical University Munich (TUM)
Boltzmannstraße 3, 85748 Garching
fuchsale@in.tum.de zehranaz.canfes@tum.de Vasiliki.Papadouli@cit.tum.de

Abstract

Mesh reconstruction is one of the key challenges in computer vision for applications such as novel view synthesis. A rather unique approach is presented in [2], where the mesh is not modeled directly, but indirectly via a velocity field that acts on an initial mesh and deforms it into the target shape.

A major drawback of this existing approach is the requirement of a trivial (black) background in the image dataset. In its current form, the method fails when given an object in the context of a more complex background. To see if this problem can be solved by modeling the mesh background separately, we perform a series of experiments.

For our approach, we simplify the original architecture, create a number of datasets for training and validation, and perform our experiments on them.

We find that reconstruction quality varies depending on the background chosen, but that modeling the background separately leads to only marginally improved results.

1. Introduction

In the dynamic landscape of 3D modeling and computer graphics, the integration of deep learning has paved the way for significant advances, including in the area of mesh reconstruction. Mesh reconstruction can be used for other computer vision-related tasks, such as novel view synthesis, or for downstream applications that require a mesh, such as 3D modeling or 3D printing, making it a very useful application.

A distinctive and noteworthy approach to mesh reconstruction using a deep learning approach, as discussed in [2], introduces a departure from direct mesh modeling. Instead of directly modeling the mesh using a deep learning model, a velocity field is modeled instead. This velocity field is then applied iteratively over a number of time steps to transform a sampled unit sphere into the desired target shape.

However, an inherent limitation of this approach is its dependence on a trivial (black) background. In its current state, the method fails when confronted with an object in front of a more complicated background. This limitation prompts an investigation into the potential solution of modeling the mesh background separately using an additional method such as a Neural Radiance Field (NeRF) [5]. Our project is motivated by this limitation and aims to determine whether isolating the mesh background could indeed help to address this issue.

To perform our experiments, we simplify the original architecture and generate a series of datasets for training and validation purposes. Our experiments take place within this framework. The results of our investigation show that a sufficiently different background compared to the mesh results in a good reconstruction with the base method. However, the effect of providing information about the background to the renderer is more subdued than expected.

2. Method

2.1. Prior Work

The goal of this project is to build on the previous deep learning-based approach of Cheng *et al.* (2022) [2], which aims to reconstruct 3D objects with arbitrary materials from a dataset of images from different viewpoints with trivial backgrounds. Our approach aims to refine the original authors' methodology for reconstructing 3D objects with arbitrary backgrounds by using their architecture with some simplifications and adaptations.

Architecture The key to the architecture of the previous work by Cheng *et al.* (2022) [2] is to deform an icosphere into the target mesh to be reconstructed through time by using a velocity field representation (see Figure 1). This makes the transition smooth and diffeomorphic as well as orientation preserving.

To achieve this, they make use of two MLP networks: A BRDF-Net, as well as a ShapeNet [1] to learn the material properties of the mesh in terms of *spatially-varying Bidirectional Reflectance Distribution Function* (svBRDF) and the shape of the mesh, respectively. The method is shown

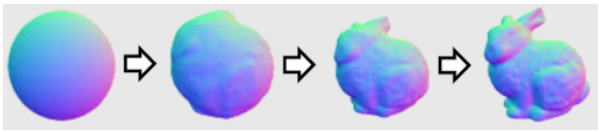

Figure 1. From left to right, the icosphere deforms over time into the target bunny mesh.

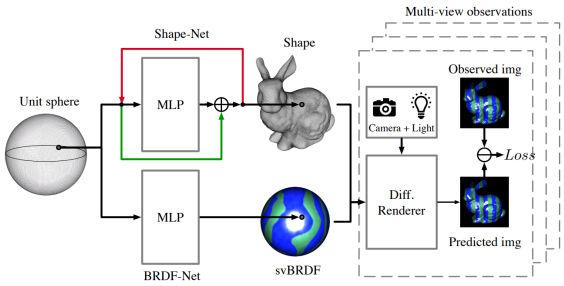

Figure 2. Pipeline for the 'Diffeomorphic Neural Surface Parameterization for 3D and Reflectance Acquisition" [2] approach. The shape and svBRDF of the mesh are predicted and fed into the differentiable renderer to obtain the training loss and train the networks.

in Figure 2.

After predicting the shape and svBRDF of the mesh, a differentiable renderer allows the predicted mesh to be rendered with texture and a rendering loss and velocity field regularization term to be computed. The rendering loss, computed by taking the absolute differences between the predicted (I^{prd}) and ground truth images (I^{obs}), is given for each rendering view k by:

$$L^{\text{render}} = \sum_k ||I_k^{prd} - I_k^{obs}|| \tag{1}$$

and the velocity field regularization term is given by:

$$L^{\text{velocity}} = \sum_x ||(I - \alpha\mathcal{L})V(x)||^2 \tag{2}$$

where I is identity and $\mathcal{L}$ is the Laplacian operator, and $\alpha = 0.01$.

The velocity loss ensures spatial smoothness and geometric correctness such as isometry, conformality, and surface preservation via regularization. Note that the original implementation also makes use of other regularization terms such as silhouette loss, normal consistency, and Laplacian smoothing. However, these are not mentioned in the original paper and did not have a noticeable positive effect on our training results.

Advantages and Disadvantages An important assumption of the previous [2] approach, on which our project is based, is that the deformation assumes that the surface to be reconstructed is bounded and therefore has no holes. This ensures that the deformation mapping, i.e. the spherical parameterization, is well defined and the reconstructed shape is therefore guaranteed to be closed.

However, this restricts the topology of the mesh, since the deformation is only possible under the assumption of a bounded surface. Thus, if the object to be reconstructed has holes, it will not learn a correct mapping from the icosphere to the target mesh.

Another disadvantage of this method is that it always assumes images with a trivial black background and fails to reconstruct objects with non-trivial backgrounds. This is addressed in our approach and the architecture is further improved to also model the background of the objects to be reconstructed.

Challenges One of the major challenges of this project is the limited compute resources related to memory constraints. Therefore, it is crucial to first simplify the architecture by addressing these constraints before moving on to further implementations.

Since our goal is to extend the method to work on non-trivial backgrounds, it is also important to make sure that the existing implementation works. The logical way to do this is to start with a very simple setup and then reintroduce the complexity to ensure that there are no errors at any step. Many adaptations are made for this purpose.

2.2. Adaptations

Dataset In the original approach on which our architecture is based, the authors use the Diligent MV dataset [4], which contains different views of an array of objects with specular illumination applied.

For our experiments, we create our own dataset based on the Stanford Bunny. Since the body contains a lot of detail, and to keep the process simpler at first, we use only the head of the bunny (see Figure 3). Also, since we have limited access to the GPU, we have reduced the number of views from 50 to 30 to reduce memory requirements. Finally, to keep the rendering process simple, the dataset consists only of the rabbit's mesh without texture, while the original approach also uses meshes with textures (reflectances) (see Figure 4).

More details about the dataset creation can be found in subsection 2.3.

Rendering As explained in subsection 2.2, our method reconstructs objects without any texture. This is to simplify the architecture before moving on to the more complex parts. Therefore, we don't use specular reflections in the rendering process. The diffuse RGB color of the mesh is set to white by default. This eliminates the need for the

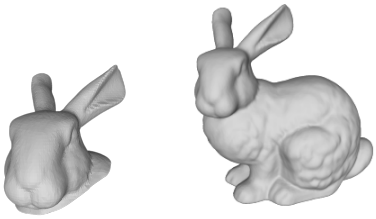

Figure 3. On the left, our ground truth mesh. On the right, the ground truth mesh of the previous approach [2].

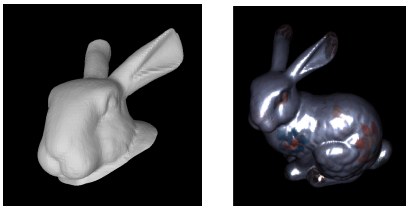

Figure 4. On the left, one of our input views of bunny without texture. On the right, the original approach's [2] input image of bunny with texture.

BRDF-Net in our implementation, which initially reduces memory requirements since we don't have to train this MLP.

Furthermore, the original approach [2] uses directional lights in their implementation and also explains that they use a point light source that moves freely between different viewpoints. In our methodology, we use only one point light source whose location is constant throughout the rendering of different viewpoints.

Finally, to be able to separately model the background in the future, a background parameter is passed to the rendering method which will then be used to model the background separately, while reconstructing the foreground object. For our experiments, we can provide the ground truth background to simulate an external modeling process. After integrating a method such as NeRF [5] into our implementation, the background will be learned simultaneously with the reconstruction of the shape in the foreground.

Training Setup The original approach of Cheng *et al.* (2022) [2], trains their network for 4000 epochs, which would take about 8 hours on our modified architecture, which is too much in our case, due to restricted access to compute resources. Therefore, we started training our network with 1000 iterations and later make use of early stopping, as described in subsection 3.2, to make sure we use our GPU resources efficiently. More details on early stopping can be found in subsection 3.2.

In addition, to reduce the memory requirements of the GPU, we also trained our network with different sizes of the ShapeNet. In the original approach [2], the ShapeNet consists of layers of sizes $(256, 256, 256, 256, 3)$. In our implementation, it is possible to set the ShapeNet size from the configuration file (see subsection 2.4), and we experimented with different setups to see if we could reduce the memory requirements by reducing the size of the ShapeNet network. More details on how the layer sizes of ShapeNet affect our results can be found in subsection 3.5 where we show that we can achieve successful results with a smaller ShapeNet.

Another step associated with making use of a NeRF architecture in the future is reducing the size of our trained images. As the NeRF training relies on calculating a ray for each pixel of the image, using a high resolution would lead to running into resource constraints. [5]

To solve this issue, we looked into whether we would be able to use cropped images during training and still achieve good results. At each iteration, a random portion of the image is rendered to ensure that the resolution is reduced from 100x100 to 50x50. The cropping method is explained in detail in subsection 2.5.

2.3. Dataset Creation

Process We created a simple dataset with the help of the widely used mesh of the bunny by using the Blender software [3]. For this, we considered only the head of the bunny, so we cut off the head with the bisect tool and removed the part of the body. We then re-centered the head on the origin and scaled it until it occupied a large part of the unit sphere, while still remaining strictly inside it as shown in Figure 5. Convergence should be better this way, since in the primary code, the geometry is initialized as a unit sphere. After creating our new mesh, we created 30 different keyframes, by having the camera rotate around the object but focusing only on the front part of the bunny's head as can be seen in Figure 6. We used these camera settings to render 30 different images with a soft rasterizer, and these images form our training dataset, shown in Figure 7.

We modified the soft rasterizer used in the original work of the paper by Cheng *et al.* (2022) [2], excluding every

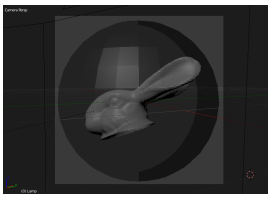

Figure 5. Mesh within a unit sphere.

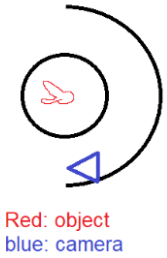

Figure 6. Camera focusing on the front part of the head.

reference to specularities and replacing the diffuse albedo with a tensor full of ones for the white color.

For the lighting, we used a Point light source model, and we assumed that exactly the same light is used for all the pictures, so when we generated all the images we used the same L0 value (L0 $= 10.0$).

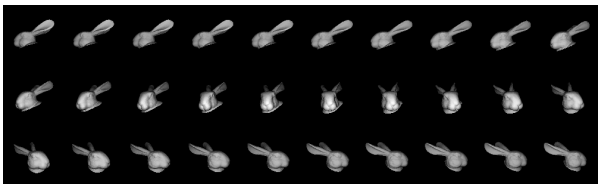

Figure 7. The 30 viewpoints for our base training dataset. For this dataset, the camera only rotates around the front of the bunny's head.

Our final goal is to reconstruct the bunny mesh with a non-trivial background, so we created a new dataset (see Figure 8) for this purpose. We added some white cube meshes to the background of our object, and as a result, the background is either white because of the meshes or black because of the default background. We placed the white cubes as close to the unit sphere as possible so that the light would reach them properly. However, due to the fact that the camera rotates around the front of the bunny, we could not place the cubes too close to the unit sphere in front of the bunny, as the cubes would obstruct the mesh for some of the viewpoints.

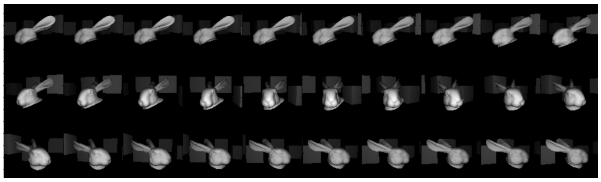

Figure 8. The 30 viewpoints for our training dataset with non-trivial background. For this dataset, white cubes were placed as a background behind the bunny.

Separation of Datasets (training, validation) To test how our reconstructed mesh compares from unseen angles, we created validation datasets. We used the Blender software to generate new keyframes. We set the camera from a higher angle and took a set of viewpoints looking at the object from above, another set of viewpoints looking at the object from a much lower height, and finally a set of viewpoints rotating around the whole head of the bunny and not just the front part (see Figure 9).

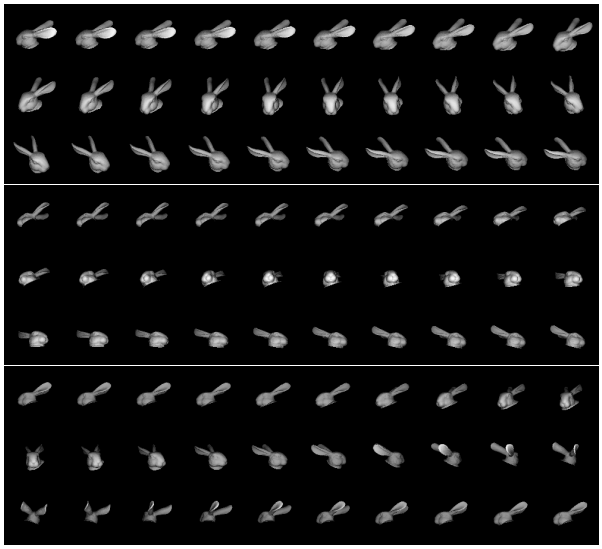

Figure 9. The 30 viewpoints for each validation dataset. The viewpoints are depicted in the following order: viewpoints looking from above, from below, and lastly viewpoints from the full rotation around the head.

2.4. Further Implementation Details

To make the training and experimentation process as time-efficient and straightforward as possible, we provide some additional tools. These include convenient configuration and batch training, as well as the feedback provided via Tensorboard logging.

Configuration Files As we provide a number of different datasets for both training and validation as described in subsection 2.3, as well as a number of variations for the network itself we decided on an external configuration file as a way of setting up our experiments.

The losses used for training as described in subsection 2.1 may be weighted. Similarly, hyperparameters such as total iterations and learning rate as well as the network size may also be configured. This also includes checkpoint and render intervals.

Lastly, multiple different configuration files may be provided during a single run, allowing for multiple training sessions to be queued and executed.

Tensorboard To better track our training results and make more informed decisions in regards to hyperparameter choice, we decided to track the data of our runs using Tensorboard logs.

This allows us to understand how losses evolve for both our training as well as validation datasets at a glance, as shown in the loss curves in Figure 13.

We also store the mesh as well as renders from different

viewpoints at an interval specified in the configuration file. The renders are formatted as shown in Figure 7.

2.5. Cropped Training

Our future plans were to add a background to our mesh and model this background with a NeRF [5]. Due to the fact that NeRF is implemented by shooting a ray at every single pixel of the rendered image, we want to reduce the computational cost of our implementation. Therefore, we add to our code the possibility to train by rendering a random part of the image at each iteration. For example, if the full resolution of the image is 100x100, we train by rendering a batch of randomly cropped 50x50 sub-images (see Figure 10).

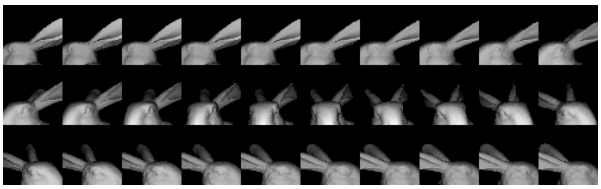

Figure 10. Images are cropped from 100x100 to 50x50 to save on computational resources during the training process. Cropping is performed with random offsets for each sample.

At each iteration, we crop a random 2D offset value on the rendered ground truth and predicted mesh. We compute the image loss on the cropped image and compare the prediction to the cropped ground truth. In this way, we reduce the resolution of the rendered images to be able to work more efficiently with NeRF in the future.

3. Results

3.1. Baseline Results

Training our network only with image loss and velocity loss as mentioned in subsection 2.1, the deformation of the mesh is successfully completed after 1000 iterations. It is worth mentioning again that the training process only uses viewpoints from the front part of the bunny's head, and viewpoints from different angles are used for validation. In Figure 11 we can see the ground truth mesh, the predicted one, and then from different viewpoints (looking at the object from above, lower height, and rotating around the whole head) that were used for validation. The loss curves for the corresponding validation viewpoints are shown in Figure 12. Using the viewpoints around the the hole head, the results are quite good, with a loss of 0.01483. In the cases where the viewpoints are taken at a much higher or lower height, the loss increases to 0.01895 and 0.01959 respectively, since the ears, which are not well separated, are more obvious from these points of view. However, in some cases, the loss curve begins to rise again after a few epochs, likely due to overfitting to the seen viewpoints, so it is worthwhile to work with early stopping for the rest of our experiments as presented in subsection 3.2.

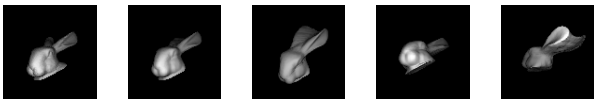

Figure 11. On the left, the ground truth mesh. The second image shows the reconstructed mesh used for training loss. The rest of the images show the reconstructed mesh again but from different viewpoints that are used for validation loss.

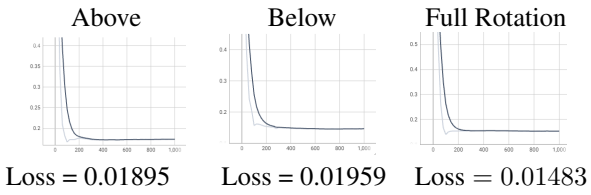

Figure 12. The loss curves from the corresponding viewpoints

3.2. Early Convergence

It is important to observe the validation loss curves of our network to make sure that there is no overfitting or to get a better understanding of how many iterations we need to train our model. For this purpose, the early stopping technique is implemented and used.

As can be seen in Figure 13, we trained our network with different early stopping patiences to see the behavior of our model over time. If the patience is too low, our network stops training after 120 iterations in about 12 minutes. The reason for this is that there is a small increase in the loss curve after 60 iterations. The too low stopping patience causes the training to stop because it misinterprets this slope as overfitting. At this point, however, the training is not complete and therefore the results are not accurate. Therefore, we need a higher early stopping patience.

When the early stopping patience is increased, the training stops after 620 iterations in about 1 hour. The results at this point look more promising and accurate, and this stopping point is considered acceptable.

If we compare our results with early stopping, i.e. 620 iterations, with the results without early stopping, i.e. 1500 iterations (2.5 hours), we see that our model starts to converge after some time and that there is not much improvement in either the validation loss curves or the reconstructed mesh. Therefore, training with this many iterations is considered unnecessary and it is crucial to use early stopping in our experiments to keep the use of computational resources to a minimum.

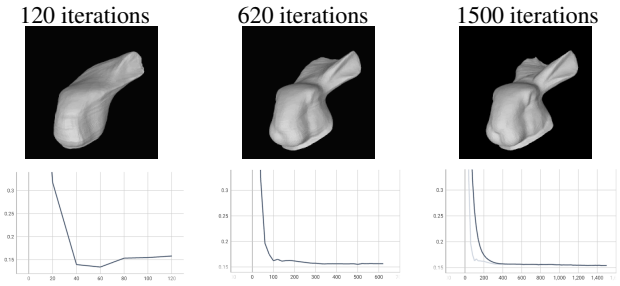

Figure 13. The first row consists of renderings of the reconstructed mesh with increasing early stopping patiences from left to right. The second row shows the validation loss curves from the 'above' viewpoint with increasing stopping patiences from left to right. As the patience increases, the number of iterations increases and convergence is observed.

3.3. Cropped Inputs Results

Here we compare the results between the training with and without the cropping method. In Figure 14, the ground truth mesh of the bunny, the predicted mesh, and finally the predicted mesh using the cropping method are shown in order. As we can see from the results, the ears of the bunny are not as well formed as in the traditional training.

Figure 14. The second and the third images show the difference in the reconstructed mesh without and with using the cropping method.

3.4. Non-Trivial Background

As mentioned before, our final goal is to reconstruct our mesh with a background, so we created a new dataset as described in subsection 2.3, with some white cube meshes on the background of our object. When we train our current network with the new mesh including the cubes and the bunny's face, the result we get is shown in Figure 15. The training process does not work well, the model fails to reconstruct the ears of the bunny, and this is because the background is also white like the object.

3.4.1 Results with Background Provided

Our next step is to model the background using NeRF. However, before we continue with this process, we first checked if this will work by using the ground truth rendered background as input to our rendering function. In this way, the

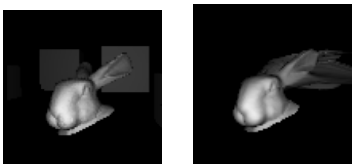

Figure 15. On the left, the ground truth image of the full scene consisting of meshes of cubes as the background and the bunny mesh. On the right, the reconstructed mesh after the training process

background is provided by our renderer and the Loss should only try to reconstruct the bunny mesh itself. We see that the results in Figure 16 are not what we expected, and the training process still fails to reconstruct the mesh with the background.

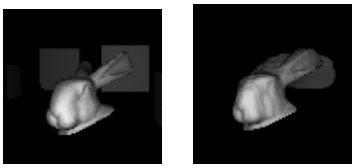

Figure 16. On the left, the ground truth image of the full scene consisting of meshes of cubes as the background and the bunny mesh. On the right, the reconstructed mesh using rendered background images of the cubes.

3.4.2 Background Variation: Rotation

One possible way to improve the reconstructed ears is to add viewpoints behind the bunny's head and perform a full rotation. To do this, we placed the cubes a little further away from the bunny to allow the camera to make a full rotation around the head (see Figure 17). This time we see that with a full rotation around the head, there is a significant improvement in the reconstruction of all parts of the bunny's head, including the ears Figure 18. However, the fact that the cubes are also further away and thus less strongly visible, can be the reason why we have a good reconstruction.

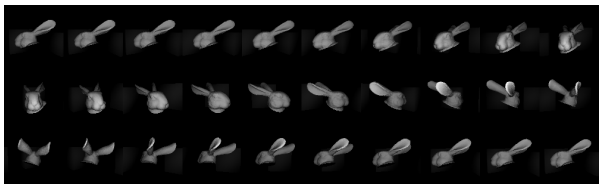

Figure 17. The 30 viewpoints for our training dataset performing a full-rotation of the head. For this dataset, viewpoints were taken around the whole head of the bunny, and not just the front part.

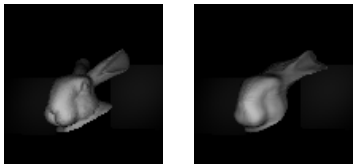

Figure 18. On the left, the ground truth image of the full scene consisting of meshes of cubes as the background and the bunny mesh. On the right, the reconstructed mesh.

3.4.3 Background Variation: Color

So far, the bunny and the background cubes had the same color, white. This can make it difficult for the optimization to distinguish between the background and the bunny itself as the foreground. So we decided to try rendering the scene with colored cubes in the background and the white bunny mesh in the foreground (see Figure 19). This should give the optimizer a better chance to distinguish between foreground and background.

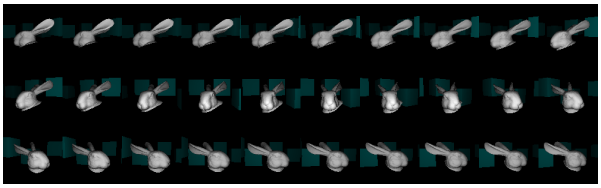

Figure 19. The training dataset with a colored background. For this dataset, cubes were colored and the bunny remained white.

After setting up the dataset, we perform training by again providing the ground truth background to the rendering function. As the results in Figure 20 show, the colored background greatly improves the reconstruction results. The model is now able to clearly distinguish between the background and the bunny, resulting in a better reconstruction of the ears. At this point, the bunny is successfully reconstructed using the ground truth background on the renderer, and we can continue our work with a NeRF to model the background.

However, as in subsubsection 3.4.1, we find that providing the ground truth background to the render function does not seem have any noticeable positive impact on the reconstruction of the mesh. This can be seen when comparing the second and the third image in Figure 20, as there is no visible difference between the two results. This implies that, at least with the current architecture, simply supplying a background through an external method such as a NeRF would not lead to an improved mesh reconstruction.

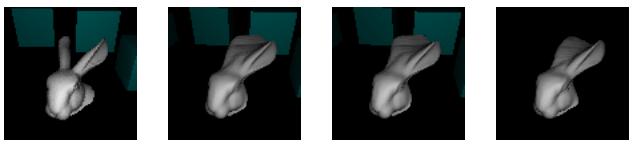

Figure 20. Left to right: 1. Ground truth with colored background, 2. Reconstructed mesh with background, 3. Reconstructed mesh with ground truth background provided during training, 4. Reconstruction when training with a trivial black background.

3.5. Effect of Shape-Net's Size on the Results

As mentioned in subsection 2.1, one of the challenges of our approach is the high GPU memory requirements. Therefore, it is important to find ways to reduce the memory requirements as much as possible. One way to do this is to reduce the size of the ShapeNet network. Therefore, we trained our network with different ShapeNet sizes to see if we could get successful results with a smaller ShapeNet. In Figure 21 it is shown that with a too small ShapeNet of size $(64, 64, 64, 3)$ the reconstructed mesh loses accuracy and artifacts appear around the bunny's ears. When the size is increased to $(128, 128, 128, 3)$, the results become much better in terms of details around the ears and eyes of the bunny's mesh. When the size is further increased to $(256, 256, 256, 3)$, we see that there is no significant difference between the model trained with a ShapeNet of size $(128, 128, 128, 3)$. Therefore, using a ShapeNet of size $(128, 128, 128, 3)$ should be preferred in terms of accurate reconstructions and reduced memory usage.

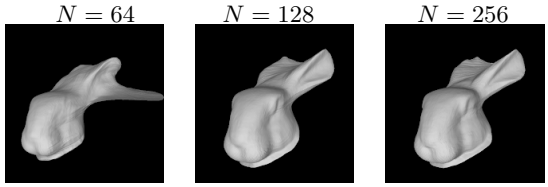

Figure 21. From left to right, results with ShapeNet layer size $N = 64, 128, 256$, where ShapeNet has layers of shape $(N, N, N, 3)$. As the size decreases, the results become less accurate and undesired deformations increase. However, the differences between the results with $N = 128$ and $N = 256$ are not visible to the eye.

4. Conclusion

In this project, we examined how to reconstruct meshes using a deep learning approach, based on the methodology discussed in the Cheng *et al.*(2022) paper [2]. Our primary goal was to investigate if this method can handle more complex backgrounds and whether modeling the background externally can enhance the approach's performance under these conditions.

We simplified some parts of the architecture and utilized custom datasets to demonstrate the method's potential for reconstructing objects with simple backgrounds. Thanks to our simplifications, we were able to work with limited GPU

access and compute resources. Our technique assumes a simple lighting model that consists of a point light placed at the camera origin, similar to a smartphone camera. This should help to generalize the method into real data.

Our experiments demonstrated how the quality of the reconstruction decreases depending on the background used. We discovered that providing the model with background data from a distinct and noticeable background, either due to brightness or color, improved reconstruction results a lot. Because of this, we believe that introducing an external method for background modeling would be beneficial.

However we found that providing the ground truth background to the rendering function did not result in an immediately noticeable improvement in the final reconstruction as described in subsubsection 3.4.1. Further exploration of this is required before adding a method for modeling the background separately.

We explored the idea of using an additional method for background modeling such as Neural Radiance Fields (NeRFs). Doing so should increase the accuracy of the mesh reconstruction like providing a background. However, this is mainly left for future work.

Overall, we believe that this provides some good insight for further experiments in working towards making this reconstruction approach work with arbitrary backgrounds.

4.1. Future Work

The results of this project lay the groundwork for future exploration and refinement but more investigation is needed. In particular, introducing background modeling through an external method during rendering should improve reconstruction quality, but this effect could not be shown in our experiments so far. As such, in order to achieve better results through e.g. training a NeRF simultaneously with the foreground more research is needed.

In addition to building directly on this approach, there are other challenges to the previously presented method that could be improved in the future.

Topology Constraints Improving adaptability to a wider range of objects could be achieved by addressing constraints related to mesh topology. The current technique assumes a mesh that is topologically equivalent to a sphere with no openings. As a result, it is impossible to reconstruct a mesh with through-holes using this method. A feasible solution to this problem would be to determine the number of holes present in the mesh to be reconstructed prior to the training phase. This can be done by a traditional approach or by a deep learning approach that determines the number of holes according to the visual hull of the mesh. To produce the final reconstruction, a different initial mesh with the appropriate number of holes can be used to initialize the training procedure. For example, if the mesh contains one hole, a torus shape could be used instead of a unit sphere.

BRDF modeling Due to limited computational resources and the need to create a more controlled testing environment, the rendering process was simplified for this project. In particular, we did not reconstruct specular reflections as we assume diffuse lighting conditions. The original work uses a more complex lighting model including taking the material of the mesh into account. Therefore, another reasonable next step would be to combine the external background reconstruction with the more complex one to see if the improvements remain in this scenario.

Comparability An important question is whether reconstructing the background separately using the diffeomorphic transformation approach is competitive with other modern mesh reconstruction methods. For example, NeRF variants [5] [6] have been shown to be capable of modeling both foreground and background through architectural adaptations.

Objective metrics, such as structural similarity indices, could provide better performance evaluation. This would improve the understanding of the impact of adjustments and refinements.

In conclusion, more research is needed to confirm the effectiveness of the velocity field-based approach when dealing with complex backgrounds.

References

[1] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 1

[2] Ziang Cheng, Hongdong Li, Richard Hartley, Yinqiang Zheng, and Imari Sato. Diffeomorphic neural surface parameterization for 3d and reflectance acquisition. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–10, 2022. 1, 2, 3, 7

[3] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 3

[4] Min Li, Zhenglong Zhou, Zhe Wu, Boxin Shi, Changyu Diao, and Ping Tan. Multi-view photometric stereo: A robust solution and benchmark dataset for spatially varying isotropic materials, 2020. 2

[5] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. 1, 3, 5, 8

[6] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields, 2020. 8