SRMCV Project: Extending the Diffeomorphic Neural Reconstruction Approach - Week 3 Report

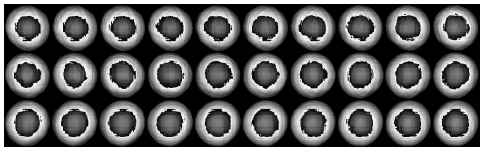
1 Overview

Our main goal for the third week was to validate our results using only the losses proposed in the original paper, as well as to ensure that training using a cropped version of the input works as intended. Alongside this, we also looked at the results of the network when training on an image set with a background. As expected the network struggles to accurately model this kind of input which is why we will add the NeRF representation to enable the network to properly handle backgrounds.

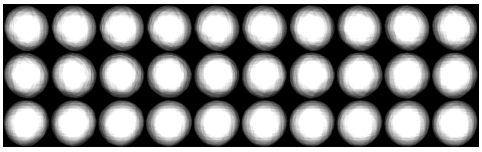
Finally, to enable easier hyperparameter tuning we added Tensorboard support, which enables us to more easily track the performance of different network configurations.

1.1 Holes in the Mesh

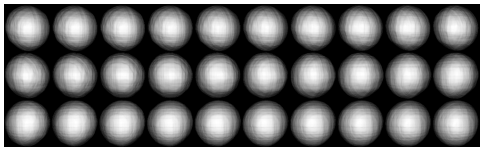
In the following experiments we noticed some black holes in the mesh, particularly when first starting the training as shown in Figure 1(a), top. Sometimes these holes also appear later into training.



(a) Holes appear in the rendered sphere due to brightness values out of bounds.



(b) Clamping the values results in overexposed patches.



(c) Scaling to the maximum brightness results in a smooth output.

Figure 1: Render of initial unit sphere during training

We initially thought this was caused by the camera plane clipping into the mesh as it always happens on parts of the mesh closest to the camera. However these points are actually vertices with a brightness value > 1.0, which causes them to overflow when transforming into a 16bit Integer. The highest brightness in our dataset is normalized to 1, so it makes sense the network might overshoot near these locations. We can fix this issue by clamping the final output of the image to a range $[0, 1]$ or scaling using the maximum value, shown in Figure 1(b) and Figure 1(c) respectively. Ideally however, the network should not overshoot significantly. This issue should not affect training, as a brightness value of > 1 is perfectly fine for the differentiable renderer to output.

1.2 Training with Randomly Cropped Images

Previously, this method was implemented in a way that updates the translation matrix T of the camera. For each iteration, it randomly assigned a value to change the camera's 3D position that helped the camera move in the 3D space and render the from different positions. This way, the renders were cropped and no processing of the whole mesh was needed. However, this approach fails with our training pipeline because of two reasons:

1. Required to render the ground truth mesh along with the predicted one with the same T at each iteration, which both increases the computational workload and also requires our method to have access to a 3D mesh, which contradicts with the project's aim.
2. Another way would be to crop the ground truth images by slicing the image arrays directly and recreate this 2D crop offset using specific different 3D camera settings. However due to using a point light source, lighting would vary if the camera is closer/further from the mesh. Because of this, we also ruled this out.

Because of these reasons, we implemented the training with cropped images in a more direct way. The cropped images in an iteration can be found in Fig. 2. At each iteration, a random 2D offset value for cropping i.e. slicing the rendered ground truth and predicted meshes are assigned. Using these values, we crop after the rendering on the whole mesh is complete. The image loss is computed on the cropped images and compare the prediction with the cropped ground truth. This way, we only work with ground truth images without the need of a 3D mesh and we decrease the resolution of the rendered images to be able to work more efficiently with NeRF in the future.

As can be seen in Fig. 4, although the method constructs the ears and the side view of the mesh successfully, it fails to reconstruct the center of the face accurately. This may be due to insufficient number of iterations, high learning rate, or insufficient image loss weight. Until now, we have tested with the following configurations for 50x50 resolution:

- The same set up with the paper with 1200 iterations
- Smaller learning rate of 0.0005 with 1000 iterations

You can compare the results (50x50) of the 1000 iterations with 0.001 learning rate and 1000 iterations with 0.0005 learning rate, by looking at the Fig. 3. More experiments to improve the results are currently in the process with the following configurations for 50x50 resolution:

- The same set up with the paper 1000 iterations and a larger image loss of value 1.5
- Smaller learning rate of 0.0005 with 4000 iterations
- Smaller learning rate of 0.0001 with 1000 iterations

Our tests so far seem to indicate the learning rate needs to be lowered further when training with a cropped image size.

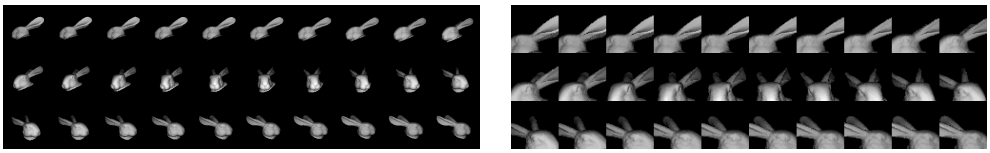


Figure 2: On the left, the ground truth images without cropping with size 100x100. On the right, the ground truth images cropped to 50x50.

1.3 Creation of a Dataset with a Non-Trivial Background

In this part of our project, we added some white cube meshes at 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, in order for the light to reach them properly. However, due to the fact that the camera is rotating around the front part of

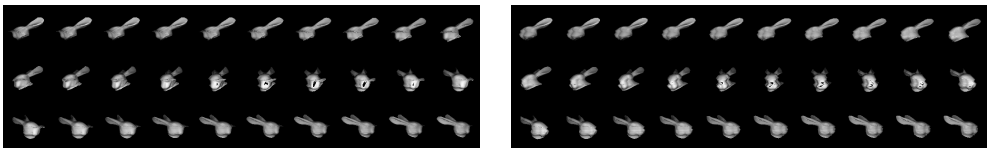


Figure 3: On the left, the results with 1000 iterations and 0.001 learning rate. On the right, the results with 1000 iterations and 0.0005 learning rate. The deformations and horizontal line artifacts decrease slightly in the configuration with a lower learning rate.

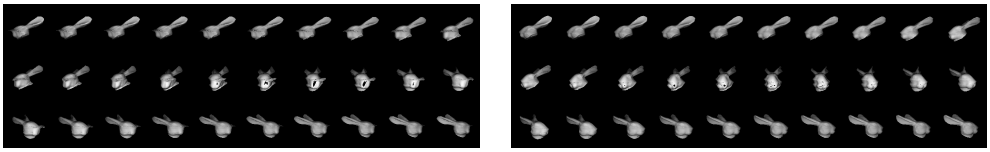


Figure 4: On the left, the results after 1000 iterations with a 50x50 resolution crop. On the right, the results after 1000 iterations with an 80x80 resolution crop. As the resolution decreases, the reconstruction errors around the center of the bunny increase.

the bunny, we could not put the cubes too close to the unit sphere in front of the bunny, as the cubes would then obstruct the mesh for some of the viewpoints. When training the network with the new mesh including the cubes and the face of the bunny, the result we get is depicted in figure 5. It is obvious that the training process does not perform well, as the model fails to reconstruct the ears of the bunny, since the background is also white as the object. Cubes which are located bit farther from the mesh do not affect the training process as they are not well illuminated by the point light source.

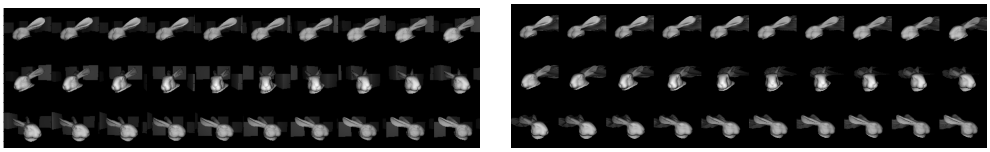


Figure 5: On the left, the dataset of the new mesh is depicted, and on the right the results after the training process.

1.4 Creation of a Validation Dataset

In order to test how our model performs from different viewpoints, we created validation datasets. Using Blender software we generated new keyframes. Setting the camera from a higher angle we 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 lastly, a set of viewpoints rotating around the whole head of the bunny and not just the front part. In the first row of the figure 1.4, we see the results from the training process with a loss of 0.00727. In the second row, we check the model from viewpoints taken around the hole head and not just the front part and the results are pretty good, getting a loss of 0.01483. In the cases where the viewpoints are taken at a much higher or lower height, the loss is raised to 0.01895 and 0.01959 respectively as the ears which are joint and not well separated can be seen more obviously from these viewpoints.

We will be using the validation dataset to track our actual improvement during training in addition to the prior train loss.

1.5 Improved tracking during Training

To make future training and hyperparameter tuning more straight-forward, we made some further modifications to the training outputs. The training loss over time, as well as renders at specified checkpoints and the mesh itself are now tracked using Tensorboard. This will should make it a lot easier to compare the effectiveness of different hyperparameter combinations.

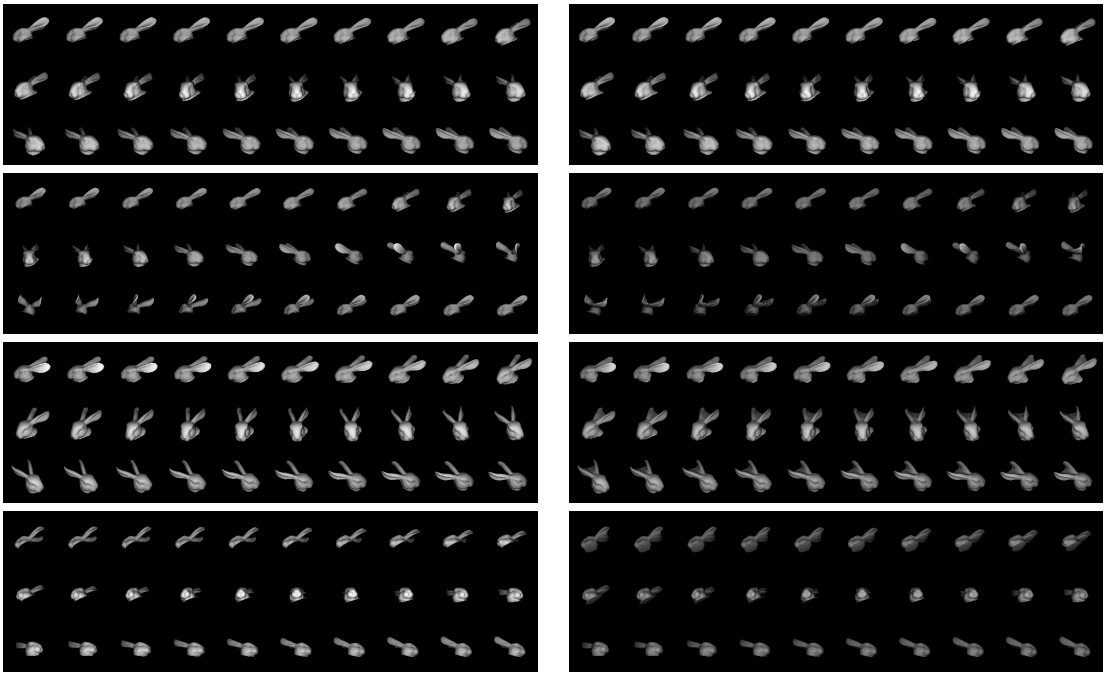


Figure 6: The grid depicts the ground truth sets on the left and the images rendered from the deformed mesh on the right using viewpoints in the following order: initial viewpoints used for training, viewpoints from the full rotation around the head, viewpoints looking from above, and lastly from below.

2 Next Steps

With the network verified for the existing datasets, as well as the ability to train on cropped versions of the data, we can now move on to modeling the background. The existing implementation cannot accurately recreate a non-trivial background, so we will be modeling it using a NeRF.

3 Task Assignment

Vasiliki Papadouli:

- Extracted new .npz files with keyframes taken from different viewpoints using blender software and created new validation datasets. Incorporated validation.py file and tracked losses with the new viewpoints.
- Added non-trivial background by creating a joint new mesh including white cubes and the head of the bunny by using blender. Trained the network with the new mesh and observed the results.

Alexander Fuchs:

- Added Tensorboard support
- Debugging of image generation
- Hyperparameter Tuning

Zehranaz Canfes:

- Fixed bugs in the train.py file that prevented the crop method to work
- Hyperparameter tuning, debugging, running multiple experiments for cropped training
- Tried to adapt the cropping method to our training pipeline and implemented it another way that works successfully