– Independent Work Report Fall 2023 –

SNeRF: Using segmentation to refine transient object removal in NeRFs

Jack Stanley
Adviser: Felix Heide

Abstract

In this paper, we attempt to improve upon existing models meant to manage and remove moving, or transient, objects that are present in NeRF training data by designing a new model: SNeRF, the Segmentation NeRF. We propose the addition of a segmentation model to identify objects in transient object masks and make them cleaner and more opaque. We also experiment with hyperparameters in existing models in order to obtain more accurate renderings. Ultimately, we were unable to successfully train our model to a point where its advantages could be properly tested, however, the preliminary results show promising potential.

1. Introduction

In the evolving landscape of computer vision and 3D rendering, Neural Radiance Fields (NeRFs) have emerged as a groundbreaking technology. By transforming a collection of 2D images into detailed, photorealistic 3D scenes, NeRFs are not just a fascinating academic exercise but a potent tool with wide-ranging practical applications. However, in order to make the most user-friendly and adaptable networks, NeRF models need to perform accurately in a wide range of situations. Most NeRF sample data is collected from controlled environments, where everything is stationary and capture is done slowly and carefully. Outside of these environments, NeRF outputs can get messy due to objects unexpectedly moving through the scene, often leaving ugly artifacts in final outputs.

Some existing papers have addressed this issue, semi-successfully removing these moving—or "transient"—objects, however they sometimes still leave behind shadows and other unwanted artifacts. To make an attempt to solve these problems, I propose SNeRF, the Segmentation NeRF, the goal of which is to more seamlessly remove transient objects from NeRF outputs. While previous

projects have demonstrated success with learning a transient object mask separately from the static mask, SNeRF extends this idea by introducing a pretrained segmentation model to sharpen and clarify the transient mask. I implemented this solution in PyTorch, extending an existing model meant to deal with transient objects. Unfortunately, due to necessary changes in the batching process during training, I was unable to come up with conclusive results, however the data collected still gave some interesting insights into the idea as a whole.

2. Background

2.1. NeRF Models

The principal function of NeRF models is to generate novel views of 3-dimensional scenes. Given a set of images from a scene along with data about the camera's position and direction, the model can generate photorealistic images of the scene as if they were taken from a point of view that the model has never seen before. NeRF's ability to effectively create a 3d scene from 2d images makes it a promising technology for virtual and augmented reality, visual effects, animation, and countless other areas.

At the heart of a NeRF model is a fully-connected neural network trained on a set of images. This network often contains multiple branches for processing separate pieces of the data, however, this can be thought of as one larger multilayer perceptron (MLP). Figure 1 shows the basic layout of a NeRF network. Despite having a relatively standard network structure, NeRFs are atypical of machine learning models in that they don't rely on prior training from a large, diverse set of scenes. Instead, all the training data comes from a single set of images from the one scene that needs to be modeled.

A NeRF generally trains by taking in a set of two input spaces and using them to predict two output spaces. As discussed earlier, the input to the model consists of a position in 3D space, defined by (x, y, z), and the camera ray direction, defined by (θ, ϕ). Note that the camera position is not important since the colors of objects will not change based on their distance from the camera, and therefore only encoding direction is sufficient. After passing these inputs through the various

multilayer perceptrons, the model outputs an RGB color, $\hat{\mathbb{C}}$, and a radiance σ. The color represents the color predicted for that point in space, while the radiance represents how bright or dense this color is, ie. how much light it should add to the scene. A point with a very low radiance is treated as transparent, while a higher radiance is more opaque and will add more of its color to the scene.

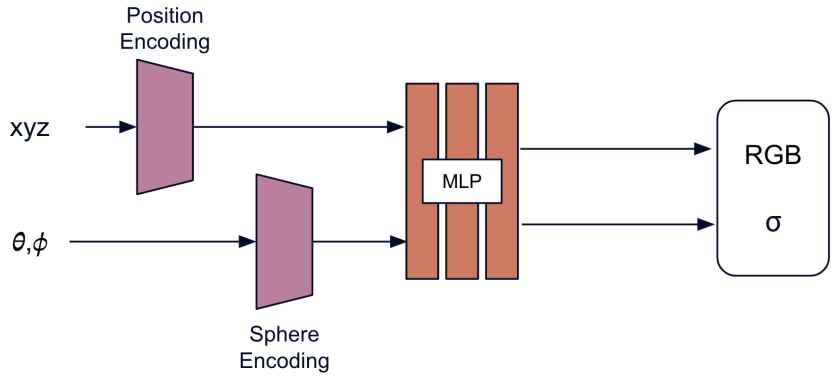


Figure 1: Typical NeRF Architecture. Transform data from images is encoded and enters the main MLP, which then outputs the RGB value and the radiance at that point.

In order to create an image from a model that predicts color and radiance, we cast a ray through the scene for each desired pixel. At each point on the ray t_k, we calculate the color $c(t_k)$ and the radiance $\sigma(t_k)$ from the model, then composite them based on equation 1, where δ_k is the distance between t_k and t_{k-1}. After compositing along the ray, we are left with the predicted color for that one pixel. This means that the neural network is queried numerous times for every pixel in the desired output. Once all pixel values are found, the output image can be used to train the network or as the final output for a wide range of applications.

$$\hat{\mathbb{C}} = \sum_{k=1}^{K} T(t_k)(1 - e^{-\sigma(t_k)\delta_k})c(t_k) \tag{1}$$

$$T(t_k) = exp(-\sum_{m=1}^{k-1} \sigma(t_{k'})\delta_{k'})$$

2.2. Current Limitations

NeRF data can come from a wide variety of environments, each with its own caveats and limitations.

Since NeRF has the potential to generate very accurate representations of 3D objects, even at large scales, statues, sculptures, and architecture are popular targets for NeRF testing and usage. However, when collecting image data to train a NeRF, it's important that the objects being captured are consistent and stationary. Large scenes in populated areas tend to involve vehicles and pedestrians that can block camera views, resulting in significant artifacts in NeRF output. These artifacts usually manifest themselves in gray, cloudy artifacts in the image foreground, an example of which is shown in Figure 2. We call these moving obstructions "transient objects," and aim to mitigate the problems that they cause during rendering.

Figure 2: Cloudy Artifact. An artifact caused by a basketball rolling past the pumpkin during capture.

3. Related Work

As Neural Radiance Fields (NeRFs) continue to gain traction, the past few years have witnessed an upsurge in scholarly interest, leading to a multitude of research papers. These papers predominantly focus on enhancing NeRFs by either elevating their accuracy, speeding up the training and rendering phases, or introducing new features. Among these advancements, a notable area of exploration involves managing transient objects within the 3D scenes. Some research efforts are directed toward recognizing transient objects and offering some way to control their changing position over time, such as adding time as a tunable parameter in the output. Typical of these models is the use of segmentation to recognize and track the transient objects through time, as is the case in [6, 7]. These papers however rely on recognizing specific objects such as humans rather than looking for arbitrary transient objects. Although they have the added feature of video playback, they aren't designed to cope with all situations.

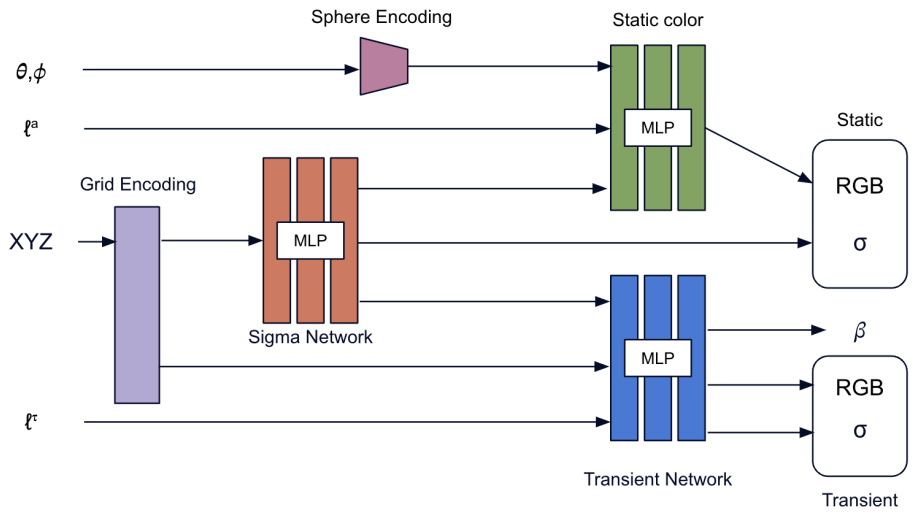
Other papers focus instead on removing transient objects entirely from the NeRF output. This solution is ideal for public situations since there is no added value in being able to play back the movements of obstructions. Additionally, without the need to build context around these transient

objects, object removal is more straightforward and theoretically more accurate.

3.1. NeRF in the Wild

The most central work to this paper's design is NeRF-W [4], a NeRF model aimed to be trained on sparse image datasets, such as a series of images from an internet search of a monument. Since these images are potentially taken years apart and in all sorts of situations, NeRF-W needs to account for a wide variety of variables within these images.

The first main challenge is that each picture can be taken at different times of day or with different lightings or cameras. For example, the wall of a white building may appear a darker gray on a cloudy day, or even orange if an image is taken at sunset. NeRF-W must adjust for these differences and be able to model an accurate color for an average setting and camera. NeRF-W approaches this issue by adding an additional embedding, ℓ^a, called the appearance embedding. The appearance embedding gets piped into the model's color MLP after being encoded using a method of generative latent optimization [3]. This results in a latent space that can interpolate between various lightings and camera effects. Once this has been trained alongside the rest of the model, the model can recreate the scene with consistent lighting and coloration at test time just by setting one constant value for the image embedding.



In addition to the appearance embedding, NeRF-W must also cope with transient objects within

the sparse image collection. Since many images are taken by tourists during busy times, it's quite common to have people or vehicles obstructing the camera view and the object being targeted. To make a clearer, more consistent view, NeRF-W attempts to remove all of these transient objects from the output, leaving only whatever was stationary during capture. In order to achieve this, NeRF-W learns two almost entirely separate models: one that learns the stationary or static features of the scene, and a second that only learns to predict the transient objects. This allows the model to accurately learn how to reproduce every image while still maintaining the separation between static and transient. At test time, the model can simply disregard the transient output, leaving just the static features. This functionality is achieved through two main augmentations to the existing NeRF architecture, both of which are detailed below.

First, the model splits the MLP at a certain point, creating one path for the static outputs and a similar one for the transient outputs. Aside from the added appearance embedding, the static MLP behaves similarly to that of a typical NeRF model. The transient MLP is similar as well but contains a few key changes as well. In addition to the color and radiance outputs, the transient network also learns an uncertainty mask, β, which models color differences due to transient objects as input-dependent Bayesian variables. The addition of this uncertainty output and the second set of weights is what allows the model to eventually predict the static portions of the scene completely separately from the static portion.

Second, similar to the appearance embedding, NeRF-W includes a transient image embedding, ℓ^{τ}. Just like the appearance embedding, the transient embedding uses generative latent optimization to assign a unique vector to each image. The point of this embedding is to allow the output of the transient network to vary based on the input image. This is necessary because the concept of transient objects is that they can be totally different across each image, thus the model needs a way to base its outputs on the image that the current inputs are based on. The full model architecture is diagrammed in Figure **??**.

Because of the added MLP, NeRF-W also uses a modified equation to calculate pixel colors during training time. In addition to the parameters outputted by a typical NeRF, NeRF-W also

includes the transient radiance σ_τ and the transient color c_τ. The transient and static components must be rendered side-by-side in order to correctly composite any overlapping radiance, as shown in equation 2.

$$\hat{\mathbb{C}} = \sum_{k=1}^{K} T(t_k)((1 - e^{-\sigma(t_k)\delta_k})c(t_k) + (1 - e^{-\sigma_\tau(t_k)\delta_k})c_\tau(t_k)) \tag{2}$$

$$T(t_k) = exp(-\sum_{m=1}^{k-1} \sigma(t_{k'})\delta_{k'} + \sigma_\tau(t_{k'})\delta_{k'})$$

Finally, once a ray's final pixel value is calculated, it gets passed into NeRF-W's loss function (Equation 3). The loss function is similar to that of a typical NeRF, but contains some extra terms to deal with the transient pieces of the model and the uncertainty component.

$$L_i(r) = \frac{||\mathbb{C}_i - \hat{\mathbb{C}}||_2^2}{2\beta_i(r)^2} + \frac{ln(\beta_i(r)^2)}{2} + \frac{\lambda_u}{K}\sum_{k=1}^{K}\sigma_\tau(t_k) \tag{3}$$

The first two terms in the loss model each pixel in the output as part of a normal distribution with variance decided by the uncertainty output of the model. The last term, which is more fundamental to this paper, is a regularizer that keeps the transient mask from becoming too large. Without this regularizer, the model could learn to reproduce every image through an entirely transient solution, eliminating the static context needed to generate images in between. This final term punishes the model for having too much weight in the transient components keeping them smaller and more accurate. This term is scaled by λ_u, a hyperparameter that adjusts the model's affinity for larger transient masks.

In terms of performance, NeRF-W is very impressive in its ability to remove transient objects. The vast majority of transient objects are removed with much success and the static objects that were blocked are typically modeled quite well. However, in some cases, the model can leave behind unwanted shadows and markings where a transient object originally blocked it. Figure 4 shows a comparison of the input and some of the outputs of NeRF-W, showcasing the dark shadows sometimes left behind where a transient object was. This doesn't happen in all cases, however it's

obvious that it could be improved.

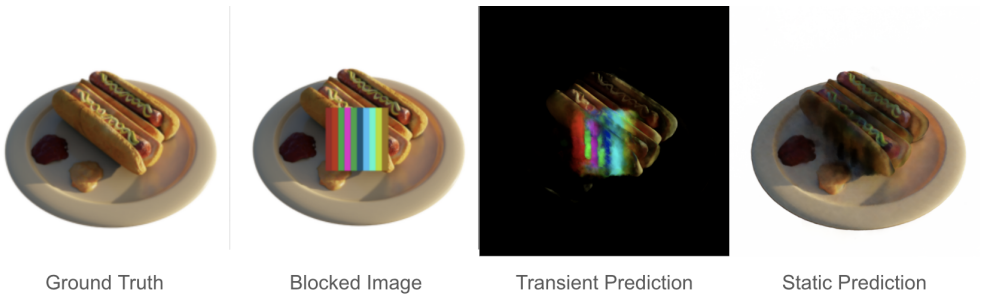


Figure 3: NeRF-W Shortcomings. In certain situations, blurry, transparent, and oversized masks can cause static predictions to have dark, shadowy artifacts behind the transient object.

3.2. MetaAI's Segment Anything Model

One of the most widely researched problems in image processing is image segmentation. Given an image containing multiple objects and a background, a segmentation algorithm or model is tasked with identifying objects with the image and marking which pixels depict part of each object. This has proven to be a very difficult problem to solve, and countless neural networks have been designed for exactly this task. One of the most popular models for out-of-the-box performance is Meta AI's Segment Anything Model [5] (SAM), a transformer-based model that can be used with or without prompting to pick objects out of an image. Segmentation models have an extremely diverse set of applications, and SAM's extensive features make it ideal for utilizing in a project.

In addition to being great on its own as a segmentation model, SAM can be utilized within another neural network as part of the process of training. As will be discussed later in the paper, this can be used to identify transient objects during NeRF training and testing.

4. Approach

The main problem identified with existing solutions for transient object removal is the learning of blurry, partially transparent masks during training. This is what ultimately leads to unwanted artifacts in the static outputs. In order to mitigate this, I propose the addition of image segmentation into a pipeline similar to that of NeRF-W.

Figure 4: Segmentation Output. An example output of Meta AI's SAM model. Each colored mask is an identified object in the image.

As the model learns the transient mask that is later left unrendered during testing, instead of allowing the mask to remain unclear, I employ SAM to identify what part of the mask constitutes an object. SAM will output a mask with well-defined edges, highlighting exactly what is and isn't part of the transient object. With a clearer transient mask, I expect less leakage of transient weight into the static model and fewer artifacts behind the transient object.

5. Implementation

The implementation for this project went through many iterations and phases, most of which will not be explained in detail.

5.1. Model Building Blocks

Rather than write a new model from scratch, I chose to start from an existing NeRF model. NeRF-W was the obvious place to start, however, I was unable to find an official implementation of NeRF-W as a starting point. This resulted in first starting out with a PyTorch/CUDA implementation of instant NGP, but it quickly proved to be an overly complex task to work with the CUDA architecture in the short time frame. Instead, I opted to use an unofficial implementation of NeRF-W that utilized PyTorch Lightning found on GitHub [1].

Outside of a few adjustments to MLP sizes and activations, the NeRF_PL model structure is identical to that described in the NeRF-W paper. The appearance embedding and transient portions of the model are toggleable as well, enabling testing with and without transient embeddings.

For the purposes of this paper, the appearance embeddings were left disabled as they have no bearings on the removal of transient objects. The datasets used for testing were all computer-generated in an environment with consistent lighting and coloring, meaning the appearance embedding was not necessary to reproduce accurate results.

5.2. Model Structure

The structure of SNeRF is quite similar to that of NeRF-W with the addition of SAM. I chose to remove the appearance embedding that was included in NeRF-W, but kept the transient embedding for obvious reasons. Figure 5 diagrams the model structure.

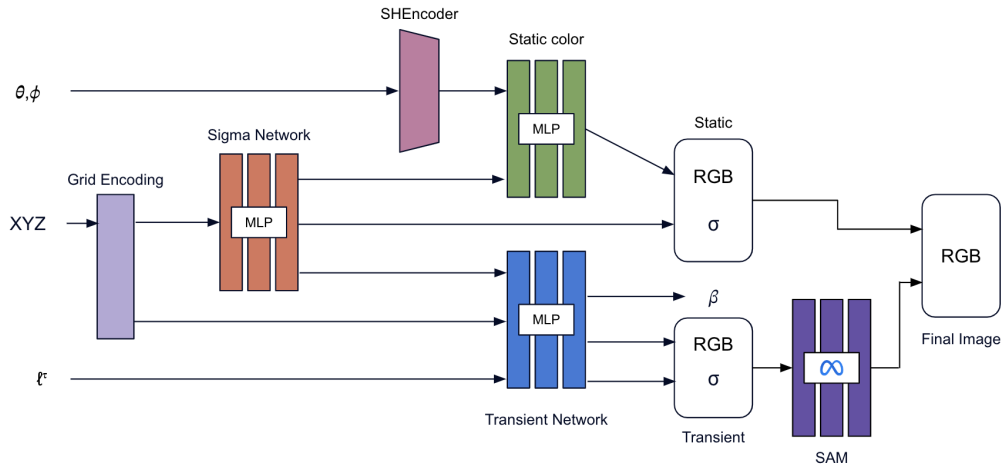


Figure 5: SNeRF Model Structure

5.3. SAM Setup and Performance

One of the most important aspects of the design turned out to be the format of the data passed into SAM. Sam is trained on real-world scenes with commonplace objects, and not necessarily the abstract, colorful obstructions used in testing. Additionally, the transient mask already begins to get separated from the background without segmentation, leaving the image of the transient

object against a mostly black background, quite far from most of the scenes that SAM was trained on. Due to these realizations, I first tested SAM on some of the transient masks produced by the unedited NeRF_PL model. These masks were chosen for testing since the SAM model would be inserted into the pipeline after the transient prediction was made for an entire image, and the masks created during training with the segmentation in place were expected to be similar to those produced without segmentation.

For preliminary testing, we utilized the high-level SamPredictor interface from the Python segment_anything module because of the simplicity it offered for fast and efficient testing. The results from this model were far from the desired results, as the model seemed to prefer choosing the black background instead of the actual object in the foreground. This could be fixed by giving the model a point on the object as an input, however, this wouldn't be practical for use in an actual model without having the user manually input points on the transient objects in any image.

In order to improve the recognition of transient objects, the first modification was to instead render the transient mask on a white background rather than black. This was done through modifying the compositing equation used to sum the colors and radiance along the ray. With the white background, SAM was able to find a mask for the transient object much more consistently, perhaps because most training images tended to have lighter backgrounds if they were taken outdoors or a well-lit area. Although there was much improvement in the consistency, the mask found by SAM still contained artifacts from other spots in the background and still selected the background in some situations.

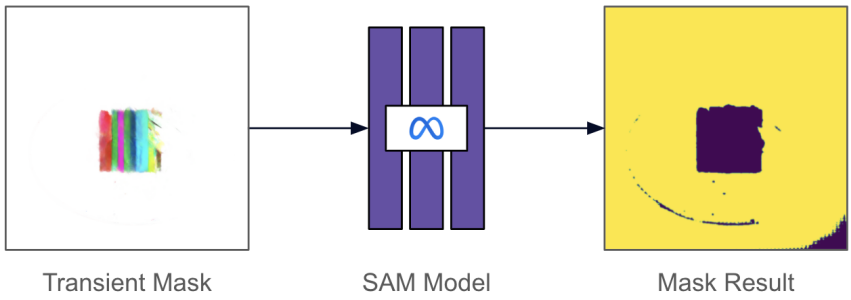


Figure 6: SAM workflow. The transient mask is rendered on a white background and gets passed into the SAM model, resulting in a clear boolean mask.

The final change made in the SAM testing was to switch from the SamPredictor interface to using the actual SAM model from the transformers Python module. The main reason for this switch was the fact that SamPredictor was implemented partially in numpy rather than pure PyTorch, and therefore lacked the ability to track the gradients of the input tensors and couldn't be used within the model's training. However, once this switch was made, I saw significant improvements in the accuracy and consistency of masking in our specific case.

5.4. Incorporating SAM Output

Once the boolean mask is returned by SAM, the mask needs to be utilized in the pipeline before the loss is calculated. The point of the mask is to more accurately separate between the static and transient components of the scene, so naturally, the areas identified by SAM retain their transient pieces, while the rest of the image is taken directly from the static output. The hope is that this will eliminate the partially transparent boundary pieces of the transparent mask.

The following steps achieve this using PyTorch:

- Render the entire image for both static and transient components
- Run segmentation on the transient mask, receiving the SAM mask
- Multiply the transient mask by the SAM mask to receive the segmented transient mask
- Place the segmented mask transient mask on top of the static scene

The resulting image is then treated as the final predicted image, matching the original image (including transient objects) as closely as possible. This image passed directly into the NeRF-W loss function, equation 3.

5.5. Resolution and Batch Size

One key difference in the training of the model as compared to NeRF-W is the way that batches are handled. In NeRF-W, each pixel can be treated as a single unit since the neighboring pixels don't change their calculation in any way. This means that any subset of pixels from any input image can be mixed together during training without hurting performance. As a result, training can be done

in arbitrarily sized batches and with randomized groups, as was done in the NeRF-W paper and implemented in the NeRF-PL repository.

In my case, choosing random subsets of pixels to train on is not an option due to our use of a segmentation model. In a segmentation model, the context of the surrounding pixels is imperative to the model's functioning, and as such the model must have the entire image rendered before it can produce any meaningful output. However, since the gradients need to be tracked throughout the entire training pipeline, all the gradients for every pixel in an image currently being trained on need to be stored in memory. Although the NeRF-W model is relatively small, the huge number of calls to render an entire image makes the total number of gradients to be tracked monumental. This leads to extensive GPU memory usage and makes it next to impossible to train on full image resolutions. Experiments were done with gradient aggregation and other memory-saving strategies, however these proved unuseful due to the necessity of the whole image before any loss can be calculated.

Ultimately, I opted for the simplest solution: to reduce the resolution of the input images until the weights and gradients for one entire image could fit into the 80GB of GPU memory available to me. Unfortunately, this required lowering the resolution to just 160x160 pixels, which eliminates much of the detail from the images but still provides an acceptable benchmark for testing. This results in a batch size of 25600, so that the entire 160x160 image can be rendered before being segmented and finally passed into the loss function. This also means that there is no shuffling of data within the batches, which could potentially lead to negative performance impacts.

6. Results

6.1. Methods

For consistency, all models were trained for 20 epochs. As explained in the previous section, the batch size for all segmentation experiments was set to 25600 and resolution to 160x160. Even though this isn't necessary for the control network that doesn't use the segmentation model, to make an unbiased comparison, I used the same resolution and batch size. The data presented in this paper used the "hotdog" scene provided in the synthetic dataset. To add transient objects to the scene,

I utilized the perturbation methods provided in NeRF_PL, which adds deterministically random obstructions to the scene. The obstructions are squares made of 10 randomly-colored bars placed at random positions in the images, and all colors and positions are the same for each experiment.

In order to quantitatively test the accuracy of the reproduced image, I turned to Peak Signal-to-Noise Ratio (PSNR), a quantity that measures the log of the inverse of an object's mean squared error (equation 4).

$$PSNR = 10 \cdot \log_{10} \frac{1}{\frac{1}{K}\sum_{k=1}^{K} ||\mathbb{C}_k - \hat{\mathbb{C}}_k||^2} \tag{4}$$

where K is the number of pixels in the image.

6.2. Dataset and Testing

Ideally, SNeRF would have been trained on a manually collected dataset of scenes including human- or vehicle-sized transient objects. However, due to time constraints, I was not able to collect and encode a dataset for this purpose. Instead, I chose to use the synthetic datasets referenced in NeRF-PL [2]. These scenes are generated in Blender and contain single 3D objects against a white background. This eliminates many extraneous variables since the lighting, object positions, and camera position data are guaranteed to be accurate and consistent. Each scene in the dataset contains 100 images for training, 100 for testing, and 100 for validation, along with camera transform data for all images. The vast number of training images

6.3. Hyperparameter Adjustments

Before training with the segmentation model, I experimented with tuning the hyperparameters pertaining to the model's loss at full resolution. I noticed that in many cases, such as the transient prediction in figure 2, the transient mask would contain remnants of the static image, in this case, the ghostly remnants of the two static hotdogs. Then, because this weight was in the transient mask, these areas of the static prediction are left dark and shadowy (see figure 2). In order to incentivize

the model to get rid of these portions, I raised the value of λ_u, increasing the proportion of the loss based on the total radiance of the transient mask.

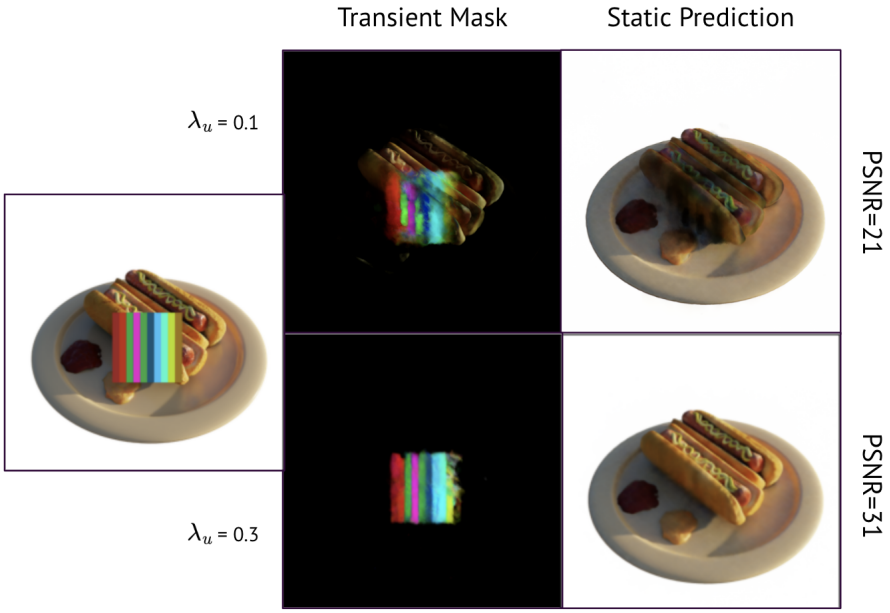


Figure 7: Hyperparameter Adjustments. Adjusting λ_u results in the removal of most of the shadows in the static output and results in a significant increase in PSNR. Note that these values were found with resolution 800x800, which explains the higher PSNR compared to other data.

As demonstrated in figure 7, a change from $\lambda_u = 0.01$ to $\lambda_u = 0.03$ got rid of virtually all the remnants of the hotdogs in the transient mask and even made the mask of the actual transient object appear much more accurate. As a result, almost all of the shadowy regions in the static prediction were removed as well, and the PSNR increased by a significant margin.

Because of these findings, for the remainder of testing, the value of λ_u was kept at 0.03 in order to produce the most accurate results possible. Future research could explore changes in the other hyperparameters involved in this model and how well their improvements are reflected in real-world data.

6.4. Segmentation Testing

Testing of the models proved to be mostly unsuccessful, not necessarily due to the addition of segmentation, but more likely due to the changed batching technique that was necessary for the

segmentation model's sake. Both the NeRF-W model and SNeRF obtained significantly lower accuracies than the versions trained with shuffled weights and smaller batch sizes. The rendered predictions are quite blurry and have a gray shadow across where the transient object covered, and there seems to be no prediction of a transient object in the transient mask or the segmented mask.

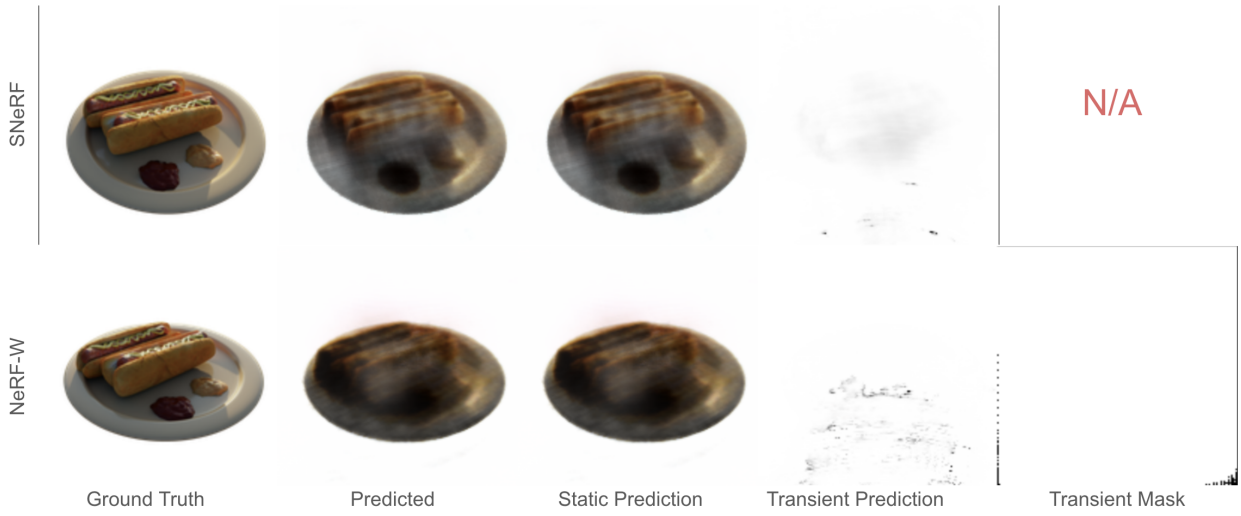


Figure 8: SNeRF Train Results

The test dataset was not much different, although this shows that both models were learning the geometry of the scene despite the transient objects in the way, perhaps suggesting some level of success. The test PSNR values were 23.47 and 23.05 for NeRF-W and SNeRF respectively, meaning that there was no significant difference in their performances.

In later experiments, increasing the training time to 100 epochs seemed to show no significant improvement in PSNR, suggesting that there needs to be a significant change in the way that the data is batched or in the structure of the model. There was a slight increase of about 3 in the PSNR after about 60 epochs, however, this didn't seem to be a continuing trend.

7. Conclusion

7.1. Future Research

Because of a few tradeoffs that had to be made for the sake of time, this paper leaves the door open for future research in multiple areas.

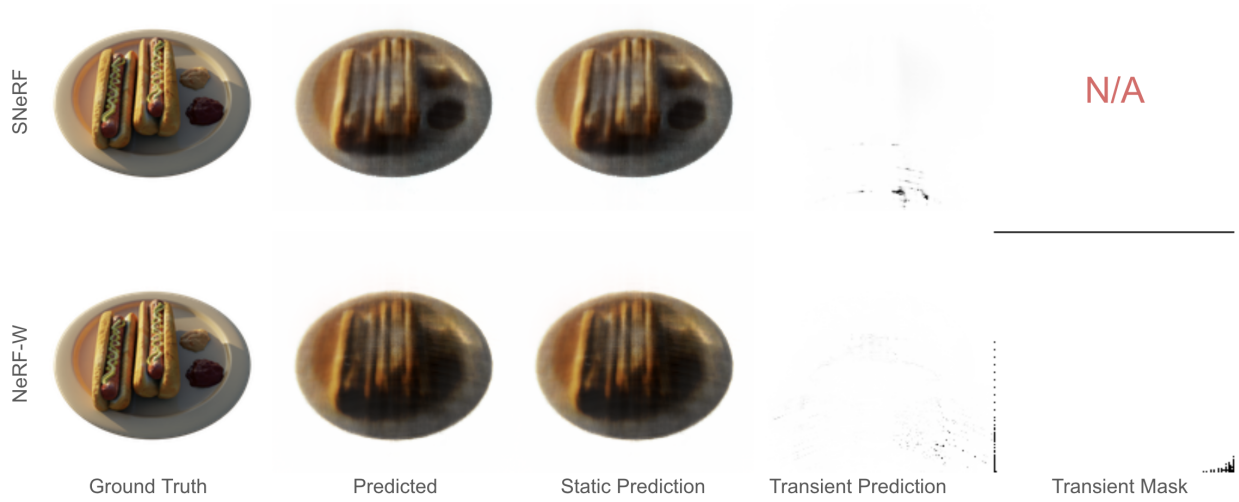


Figure 9: SNeRF Test Results. Results comparison for NeRF-W and SNeRF after 20 epochs on the test dataset. Results are very comparable with similar PSNR values.

The foremost aspect of the experimentation would be to attempt to solve the problems related to the batching technique. Unforeseen until the last minute, it seems that training on entire images at once makes a drastic difference in the training time and success. Future research could experiment with breaking the images into smaller pieces, allowing smaller batches and higher image resolution while still maintaining some of the context within each piece of the image. Another option would be to experiment with the original technique, but at a higher resolution, perhaps in a setup where more video memory is available.

Another interesting idea that I almost moved to as a backup would be to not use the segmentation at all, and instead combat the blurry, transparent masks using more traditional image processing techniques. For example, forcing the transparency of the learned mask to be entirely binary would eliminate the problem of partially transparent masks, and would still have potential to be learned accurately. Most real-world objects that block the scene are going to be totally opaque, so there shouldn't be much need to allow transparency within the mask. To take this a step further, it may be beneficial to not bother learning color at all in the transient portions of the mask. Since they don't need to be reproduced, if a binary mask is learned just on the spots where the transient object blocks, then these areas can be disregarded from training outside of their regularization term. Although these ideas don't have the benefit of the segmentation model, their pixels are context-independent,

meaning that the training batches can be shuffled and the batch size is tunable, most likely mitigating the problems seen in SNeRF training.

On the other hand, if training SNeRF could be achieved through some avenue, one of the most exciting and promising next steps would be to test performance on real-world datasets. While the segmentation model works on the abstract obstructions created in the synthetic dataset, it's much more likely that it will thrive in natural environments where the transient objects are people or vehicles. Once it gets over the hump of synthetic data, I see exciting potential for the segmentation model.

7.2. Closing Remarks

Ultimately, although my results weren't up to the caliber that I wanted, the idea behind SNeRF still proves to be somewhat promising. Compared to the normal transient NeRF with the same batch size, it achieved a similar (if underwhelming) performance in its first epochs of training. While this doesn't necessarily mean that it's even possible to train this model in any realistic time frame, it does imply that the problem doesn't lie within the model's design.

The outputs achieved by NeRF-W are already quite impressive, and with minor adjustments and improvements, it holds potential for groundbreaking changes in the accessibility of training data for NeRFs. The discoveries I made related to tuning hyperparameters also make it clear that there is still huge potential inside some of the existing architectures like NeRF-W. I achieved drastic improvement in both qualitative and quantitative aspects of the output, achieving nearly satisfactory results in the synthetic datasets.

8. Acknowledgements

First, I'd like to thank Professor Heide for his incredible guidance and positivity. His teaching and suggestions led me through my entire project from start to end, and I learned more than I could have imagined along the way. I'd also like to thank the rest of the Generative AI seminar for their continued support and insights throughout the semester.

References

[1] "nerf pl." [Online]. Available: https://github.com/kwea123/nerf_pl/tree/nerfw

[2] "Nerf synthetic dataset." [Online]. Available: https://drive.google.com/drive/folders/128yBriW1IG_3NJ5Rp7APSTZsJqdJdfc1

[3] P. Bojanowski *et al.*, "Optimizing the latent space of generative networks," in *ICML*, 2018.

[4] N. in the wild: Neural radiance fields for unconstrained photo collections, "Novel view synthesis of human interactions from sparse multi-view videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[5] A. Kirillov *et al.*, "Segment anything," 2023.

[6] J.-W. Liu *et al.*, "Hosnerf: Dynamic human-object-scene neural radiance fields from a single video," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[7] Q. Shuai *et al.*, "Novel view synthesis of human interactions from sparse multi-view videos," in *ACM SIGGRAPH 2022*, 2022.

9. Appendix

Code for the project will be available on Github under the profile jackpton14235, however the repository does not yet exist.