NeRF This! Diffusion-based Visual Accumulation

Alex Lin[1], **Forrest Meng**[1]
Virginia Tech CS 5824
alexlin@vt.edu, forrestm@vt.edu

Code Repository: https://github.com/alxn3/NeRF-This

Abstract

We developed NeRF This!, a method of iteratively adding in synthetically created data from a neural radiance field (NeRF) to the NeRF's training set to improve the overall render's accumulation rates and provide more information on the scene. Previously, researchers have used diffusion to extrapolate 3D NeRF generations of objects given a few images and have modified NeRFs using diffusion models with prompts. As NeRFs require a lot of training data to develop the scenes, we looked into possible routes of using diffusion to add synthetic data to improve a NeRF generation's accumulation, which is the number of areas with high density in the 3D space. Using camera walking, masking, and diffusion inpainting, we created a pipeline model that can add synthetic data to the training set of a NeRF iteratively as the NeRF is training to increase the overall accumulation.

1 Introduction

Neural Radiance Fields (NeRFs) have emerged as a prominent novel way of synthesizing neural views of 3D scenes from image and video captures(1). Typical users can capture their own data and use prebuilt model pipelines to construct a NeRF for the continuous view-dependent 2D rendering of environments and objects. However, for many casual users who capture NeRFs and other instances related to limitations with cameras, the initial input data to the NeRF may be low quality or inadequate to have high accumulation and confidence in consensus. As NeRFs tend to overfit on input data for better accumulation and density, not having enough input views to provide high-density accumulation can lead to artifacts such as floaters, clouds, and general noise where an element of the NeRF is supposed to be present.

There are several methods currently to clean up generations with NeRFs, including methods to remove floaters and clouds by applying NeRF reconstruction on identified 3D surface patches to reinforce densities(2). However, while these methods are effective, they often require high compute power and a high-quality ground truth. Additionally, there are methods to generate NeRF renders with just a single view synthesis, providing solutions for unseen areas covered by occlusion or low data (4). However, while these methods effectively generate NeRFs off of low amounts of data, they require low-resolution input priors and have a costly fine-tuning stage. Given the rise in effectiveness of diffusion models, and that NeRFs generate consensus from noisy 3D spaces, we believe that diffusion-based models could aid in providing extra accumulation in NeRFs. In this project paper, we propose a novel method of extending accumulation and density for high-resolution NeRFs using an augmented pipeline with diffusion inpainting. Our method samples accumulation and information from a prior NeRF that contains areas of low accumulation and noise, performs a K-means and Laplacian-based noise detection and masking, in-paints the areas with a low prompt guidance diffusion model, and iterative replaces and retrains the NeRF for gradual accumulation and consensus. We demonstrated the overall effectiveness of adding accumulation and evaluated our results with comparisons of initial and resulting accumulation maps.

[1]signifies equal contribution.

1.1 Motivation

The use of NeRFs has become popular for generating high-quality 3D scene renderings from image and video captures. However, limitations with the input data can lead to artifacts such as floaters, clouds, and general noise that impact the quality of the NeRF. While there are methods to address these issues, they often require high computing power and high-quality ground truth. With the potential for NeRFs to be used in applications like 3D asset generation, mapping, and digitization of objects, it is integral to have robust methods of improving data density for casual users. Given the effectiveness of diffusion models and the need for extra accumulation in NeRFs, we propose a novel method of extending accumulation and density for high-resolution NeRFs using an augmented pipeline with diffusion inpainting. Our approach samples accumulation and information from a prior NeRF with low accumulation and noise, perform noise detection and masking, in-paints the areas with a low prompt guidance diffusion model, and iteratively replaces and retrains the NeRF for gradual accumulation and consensus. Our method improves the accumulation density of NeRFs and can be an effective solution for casual users with low-quality input data.

1.2 Related Works

Neural Radiance Fields Neural Radiance Fields (NeRF) is a powerful scene representation technique that enables the modeling of static scenes as a continuous 5D function(1). This function outputs the radiance emitted in each direction at every point in 3D space, along with a density value that acts like differential opacity. NeRF optimizes a multilayer perceptron (MLP) that regresses from a single 5D coordinate to a volume density and view-dependent RGB color. To render a scene from a particular viewpoint, camera rays are marched through the scene to generate a sampled set of 3D points, and these points are used as input to the neural network to produce an output set of colors and densities. To capture complex variations in the radiance field, NeRF employs positional encoding to transform the 5D coordinates into a higher dimensional space, and hierarchical sampling to reduce the number of queries required to adequately sample the high-frequency scene representation. However, for more complex and new views, there are instances of floating artifacts, noise, and low accumulation that cannot be addressed by the NeRF itself without adding more information. Our proposed method provides opportunities to create realistic synthetic data for consensus on lost data.

Removing Floating Artifacts from NeRFs Floating artifacts are one of the main visible noise artifacts in a generated NeRF(2). Previous methods have focused on mitigating artifacts using various techniques such as camera pose optimization and robust loss functions. However, newer methods outlines an evaluation process that involves training a pseudo-ground truth model and using visibility masks to evaluate only co-observed regions. By training a diffusion model to learn the distribution of 3D surface patches, applying local 3D priors on NeRF reconstructions and the Density Score Distillation Score (DSDS) loss to regularize the sampled density can remove a majority of the floating artifacts. We aim to provide a novel method to reduce the noise from lack of visual accumulation, which could resolve some issues with floating noise as well.

Diffusion-based NeRF modification Other papers have also used 2D-based diffusion models to modify a NeRF environment's overall appearance(3). One paper takes in a reconstructed NeRF scene, natural language editing instruction, and corresponding source data as input. The method uses Instruct Pix2Pix, a diffusion model, to iteratively update image content at the captured viewpoints with the help of NeRF training. Through an alternating update scheme, where training dataset images are iteratively updated using a diffusion model, the images are subsequently consolidated into the globally consistent 3D representation. The results are generally good, but limitations include that sometimes the edit cannot be fully performed and sometimes edits fail to consolidate. We found that the general pipeline of this method was a promising way to introduce new data into a NeRF without corrupting the generation.

Diffusion-based NeRF synthesis from limited data Existing methods like NeRFs require many images to overfit a scene and lack generalization ability(4). A proposed method jointly trains a camera-space triplane-based NeRF with a 3D aware conditional diffusion model (CDM) on a collection of scenes. The method uses a naive finetuning strategy of optimizing NeRF parameters directly using CDM outputs and updates the NeRF representation and guides the multiview diffusion process in an alternating fashion to resolve the uncertainty in single-image view synthesis. The paper also

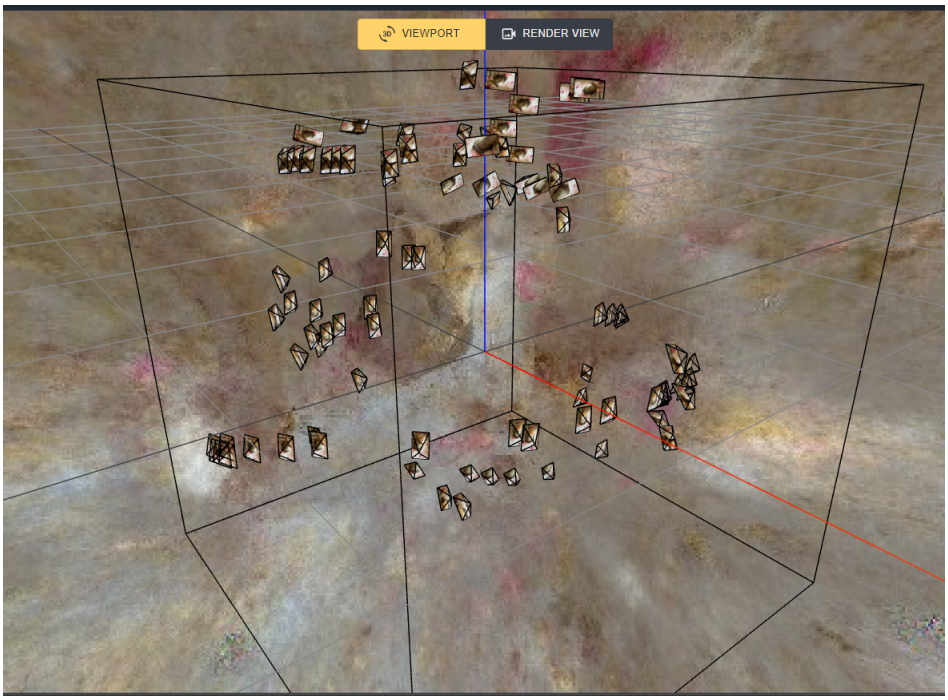


Figure 1: Generated new camera views created due to the camera walk method, in addition to the original dataset.

introduces a single image NeRF with Local Triplanes that allocates depth information without any additional positional encoding needed, and the MLP network for 3D representation can be made smaller. The proposed method has limitations, requiring at least two views and the finetuning process is expensive and time-consuming. We propose a method that can help with lack of training data for NeRFs that can be possibly less time-consuming due to reinforcement from our iterative data insertion method.

Removing objects from NeRFs A few papers propose a method to remove objects from a given RGB-D sequence, including one by using 2D inpainting models(5). The method uses a confidence-based view selection scheme to iteratively remove inconsistent inpaintings from the optimization. The proposed method consists of four main steps: 2D single image inpainting, confidence-based view selection, view direction and multiview consistency, and iterative updates of the NeRF. The paper also suggests using a 3D mask that uses a 2D object segmentation method to mask the objects. The proposed method has shown promising results in removing objects from scenes while maintaining the consistency of the remaining parts of the scene. This provides validity to the idea that painting can be used to modify a scene.

2 Methods

Our proposed method involves two main steps as additions to the general NeRF pipeline, using a pre-trained NeRF as an input with the original data set:

1. Camera walking along the NeRF to find new, relevant views to perform diffusion on and add back to the original set of views.
2. Performing masking and inpainting on the images according to accumulation maps.
3. Reinserting the image back into the training data iteratively and retraining the NeRF.

2.1 Camera Walking

The high-level overview of camera walking is to take the normal vector of the geometry a camera is pointing towards, the directional vector of the camera in world coordinates, and transforming the camera along the gradient of the depth stochastically, so that the camera "walks" across the 3D NeRF generation and finds new views that can also be relevant.

2.1.1 Calculating Normals

A challenge to extracting 3D data and understanding the geometry from NeRFs is that without using complex 3D surface and voxel analysis techniques, which can be also computationally expensive, the next best option is using 2D images and maps. To calculate the normal vector, assuming the Y axis is vertical, the X axis is horizontal, and the Z axis as being outwards away from the image plane, we take the center pixels of the image plane, calculate the depth gradient ∇D by averaging the depth change across pixels n+1 and n-1, for axis n, and augment it into a normalized 3-dimensional vector.

2.1.2 Augmenting Camera Transformation Matrix

To find the camera's principle axis vector, we need to utilize the camera transformation matrix, which has the structure of $P = KR[I|-C]$, where P is the transformation matrix, K is the intrinsic feature matrix, R is the rotational matrix, I is the identity matrix, and C is the transformation matrix. The resultant matrix that is given in camera coordinates is as follows:

$$\begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix}$$

We find the normal vector of the new camera position in the world view by comparing the difference in position to the depth map position and transposing a normalized vector. To get the rotation matrix, we take the rotational axis, calculated between the two normals, and apply them in a skew-symmetric cross-product matrix.

$$\begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}$$

Finally, we add the transformation vector to the fourth column, multiplied by a movement distance constant. Overall, the algorithm is shown in Algorithm 1.

Algorithm 1 Camera Walking Algorithm

let D = depth map
let P = camera matrix
let I = image dataset
for $iinI$ **do**
    $d_0 \leftarrow \nabla D$
    $x_{world} \leftarrow P - D$
    $P^* \leftarrow x_{world}, d_0$
    $I^* \leftarrow I, P^*$
end for

2.2 Accumulation-based Inpainting

2.2.1 Masking

Careful selection of areas for inpainting is extremely important so as to not cloud up or remove important landmarks from the ground truth and original NeRF renders. Therefore, we have a two-step process of making sure our masks for inpainting cover the correct areas of low accumulation and high noise.

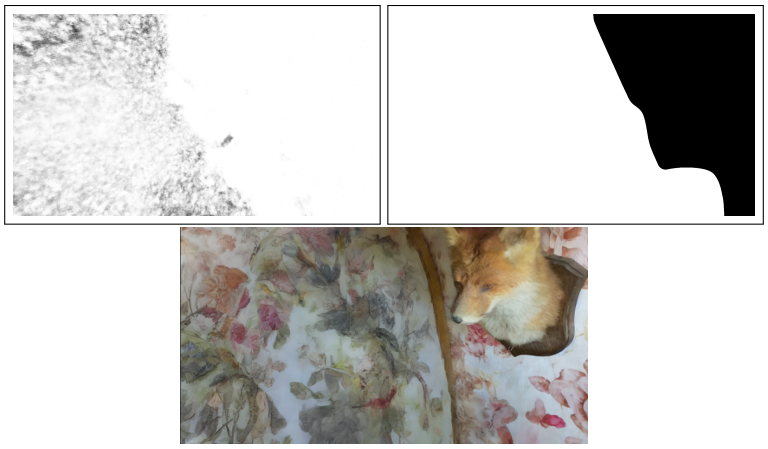


Figure 2: The accumulation map, the mask generated from the accumulation map using k-means, and the final inpainted result using the mask and the original image.

The first pass uses a K-means algorithm to generate the mask. Given a binary image of the accumulation map at a given view, we invert the bytes so that the areas of low accumulation are shown with whiter shades. We then apply a convex hull to draw the overall border and connect regions on the image. Afterward, through repeated feathering and applying k-means clustering on the image to create a mask over the areas with high noise and low accumulation. We need to feather the image since to be able to effectively cover the areas of visual noise near low-accumulation areas, but may not actually have low accumulation, we need to expand the overall masked area.

However, due to the fact that some views have regions of high noise but not much variation is shown in the rendered accumulation view, just k-means clustering would not suffice for effective masking. Therefore, if we detect that the accumulation ratio is too low or if the mask covers the entire image (or does not cover at all), we will first apply a Laplacian Filter on the image and mark high-frequency edges with white. We make the assumption that areas with actual convergence would have high-frequency edges, while noise would not have any above a certain threshold. By performing the same k-means filtering and feathering process as previously, and then inverting the result, we get a mask that covers low-resolution and noisy areas.

2.2.2 Diffusion Inpainting

Once we finish with the masking, both the mask and the original rendered RGB image are converted to PIL and inserted into the StableDiffusionInpaintPipeline. We render two images at a time with zero prompt guidance. This is to help manage the inevitable volatile generations from the diffusion process. Since we are not operating with geometries or depth as prior, our generations will not have persistence without outside methods; therefore, we average the image results before sending them back into the data loader for the NeRF to train on until after 100 steps later, where another view would have its view ground truth changed by inpainting. This iterative process allows for the NeRF to not change drastically and not generate too much noise at one time.

3 Implementation & Experiment Specifications

For the implementation of our method, we used Nerfstudio(6) as it provides an existing pipeline for training NeRFs that we can build on top on. The existing model that we are building on top of is "Instant-NGP" as it provided more consistent accumulation renders and outputs compared to other existing models present in Nerfstudio. Additionally, "Instant-NGP" is based on hashing during the training process, which allows for much faster and cheaper training. A heavy modification was required of the original Nerfstudio pipeline as it does not readily support the insertion of new views and images at runtime.

The dataset we used was the default Fox bust dataset provided by the NVIDIA team. We modified this dataset during the training process with extra views and inpainting. For inpainting, we used images

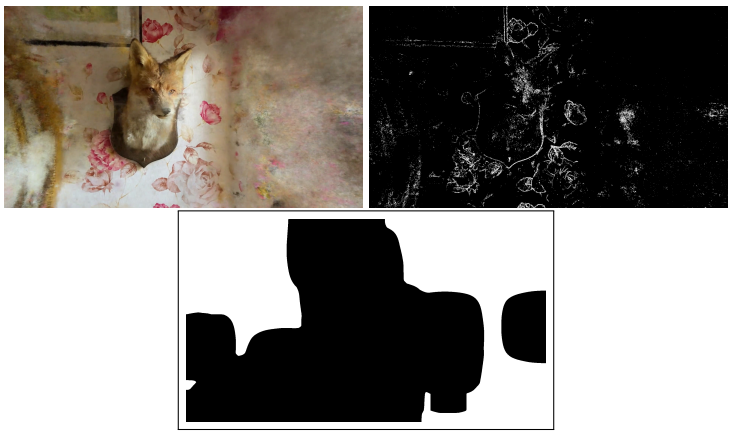


Figure 3: The original image has areas of high noise but low accumulation. Therefore, we use the Laplace filter to find areas of high frequency and apply a map according to those values.

Algorithm 2 Masking and Inpainting

let img = RGB Image
let acc = Accumulation map
let I = image dataset
if $acc_{ratio} > threshold$ **then**
    $m \leftarrow ConvexHull(img)$
else
    $m \leftarrow !\nabla^2(img)$
end if
for $nsteps$ **do**
    $m \leftarrow N_{filter}(c)$
    $m \leftarrow Kmeans(c)$
end for
$resArr \leftarrow SDPipeline(img, m)$
$res \leftarrow$ average of $resArr$
$I \leftarrow res$

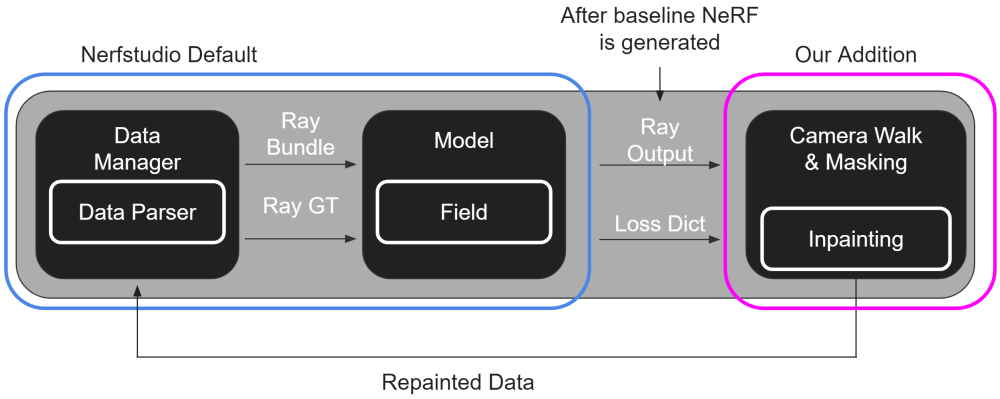


Figure 4: The overview of the pipeline and how we included it into the NeRF Studio Pipeline.

size 1980x1080 in PIL format with a guidance score of 0 and 10 inference steps. We trained the model for an entire 200k steps, rendering both the RGB and accumulation results roughly every 40k steps at 7 defined camera views. To evaluate the effectiveness of adding accumulation, we took the accumulation map at each of these camera steps at every 40k steps and calculated the accumulation ratio, which was the amount of space white bytes took up in the image. The higher the accumulation ratio, the better the result, as the accumulation increases. Qualitatively, we looked at images from the rendered videos and determined if they were any clearer if we could see any noise, and if there was an increase in accumulation.

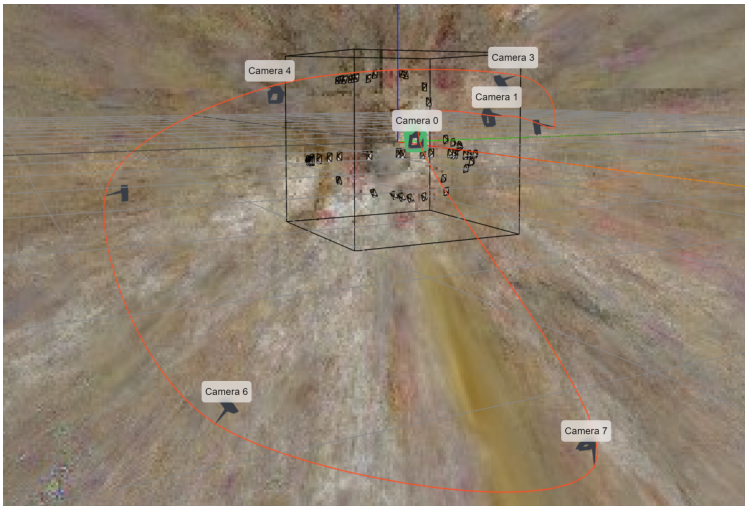


Figure 5: The path of camera views we used for rendering and evaluation.

4 Results

4.1 Quantitative Results

Looking at Figure 8, we can see that after 200k steps of the model, there has been clear growth in accumulation throughout the NeRF. The biggest improvement in accumulation was in Frame 4, as shown in Table 1, from having an accumulation ratio of around 0.819 at 0 steps to 0.971 at 200k steps. However, in the earlier steps, we have seen that accumulation has decreased in some frames. We believe that this is because of inconsistent updated views that our inpainting method provided. This causes the existing NeRF to introduce noise to try to (over)fit the data over. However, the general

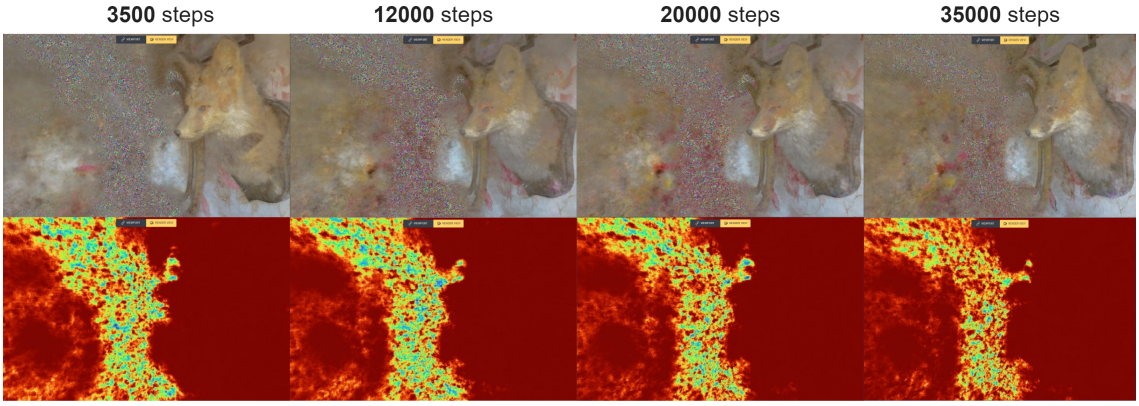
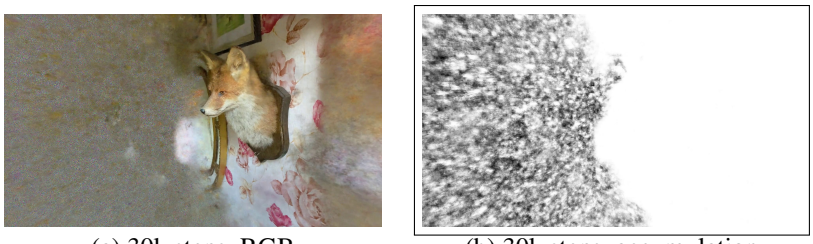
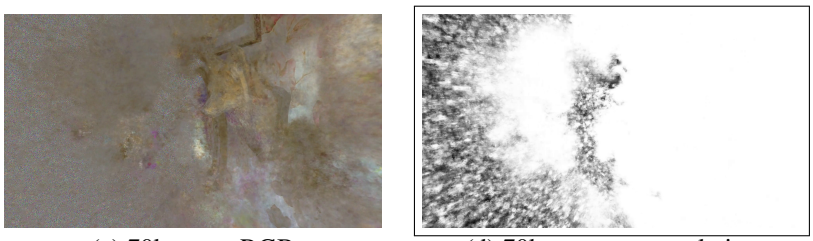


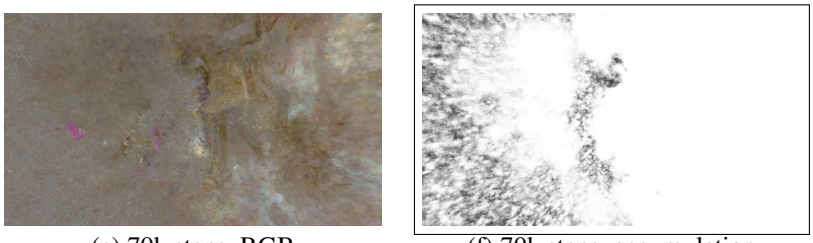
Figure 6: Progress of the pipeline between 3.5k and 35k steps. Even though the model was trained for over 200k steps, there is a clear growth in the accumulation map at the early stages of the pipeline.



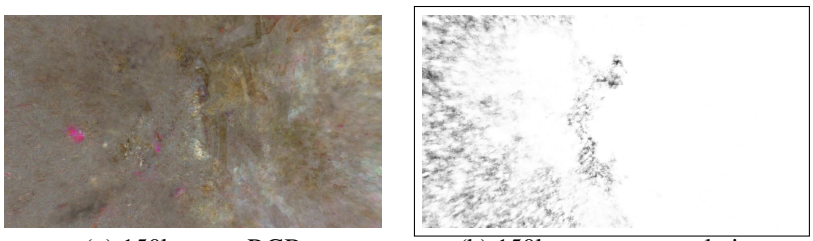
(a) 30k steps, RGB (b) 30k steps, accumulation



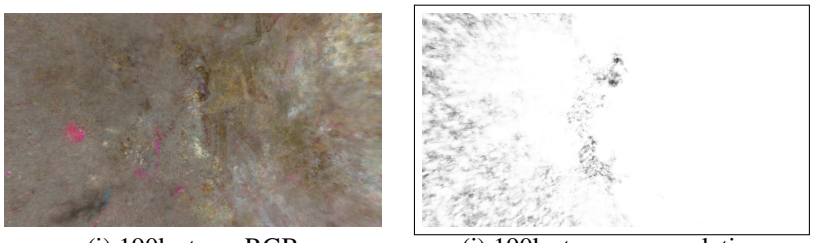
(c) 70k steps, RGB (d) 70k steps, accumulation



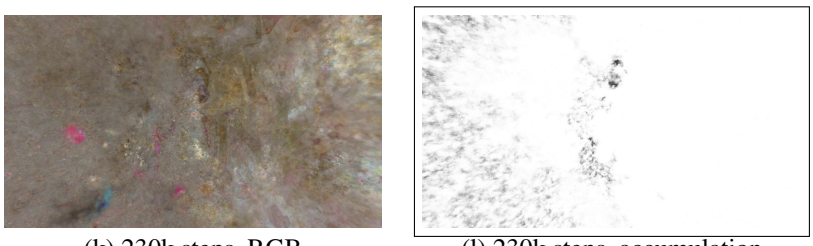
(e) 70k steps, RGB (f) 70k steps, accumulation



(g) 150k steps, RGB (h) 150k steps, accumulation



(i) 190k steps, RGB (j) 190k steps, accumulation



(k) 230k steps, RGB (l) 230k steps, accumulation

Figure 7: RGB and accumulation maps throughout the training steps. It is clear that although accumulation increased overall as the training progressed, and that empty noise was reduced, there was more RGB noise as the steps increased.

	Frame 1	Frame 2	Frame 3	Frame 4	Frame 5
0 steps	0.9984042775	0.9258773867	0.8396701654	0.8189229076	0.854721583
40k steps	0.9990543244	0.9115876131	0.8836223259	0.8530127409	0.8478540381
80k steps	0.9996100823	0.9466045638	0.9301940586	0.91001519	0.9097427057
120k steps	0.9997372439	0.9722696192	0.9602422178	0.948551516	0.9516367821
160k steps	0.9996165256	0.9811022452	0.9730917832	0.964852544	0.9688390655
200k steps	0.9996466919	0.9824022899	0.9778065503	0.9710040982	0.975287247
	Frame 6	Frame 7	Frame 8	Frame 9	Frame 10
0 steps	0.9942359901	0.8917537748	0.8741603293	0.9048120896	0.992426845
40k steps	0.9929435764	0.873431157	0.8548266593	0.9009335493	0.9948763598
80k steps	0.9946238653	0.9290584812	0.920241206	0.9430880273	0.9957631343
120k steps	0.9951738664	0.9608939724	0.9610316037	0.9687028735	0.9974759308
160k steps	0.9964693892	0.9762513919	0.9782552745	0.9812614909	0.9976967593
200k steps	0.9967618615	0.9823199305	0.9817764993	0.9840288482	0.9981753037

Table 1: Caption

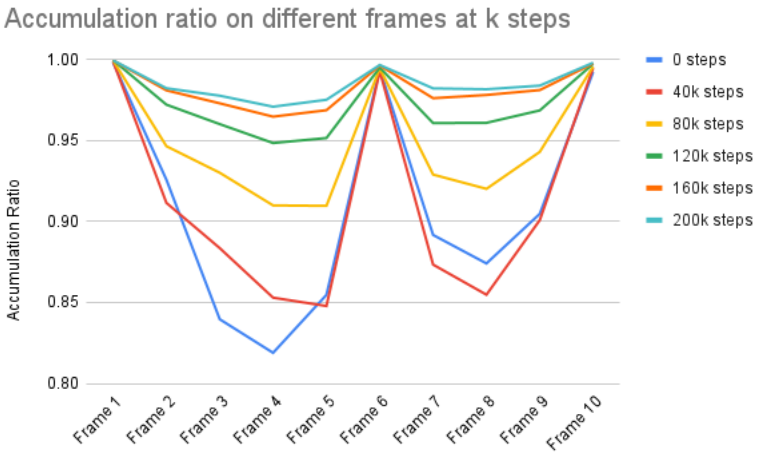


Figure 8: Accumulation differences between each step checkpoint throughout the camera path. Since the starting camera path and the camera path at Frame 6 were directly at the original view of the NeRF, the accumulation would not change much due to the already high accumulation.

features of the original NeRF, relative shapes, and colors, were still preserved after 200k iterations which are attributed to preserving the original images for the NeRF to retrain on to prevent the model from going too astray from the original as we are aiming to only add more data, not corrupt it.

4.2 Qualitative Results

Although accumulation has increased all frames, it is also apparent that floating artifacts and noise were introduced into the NeRF. As more and more accumulation gets added, the accumulation visibly increased and improved. As shown in Figure 7, it is clear that the accumulation at 230k steps was more dense and higher than the accumulation at 30k steps. However, the RGB render was a lot more noisy; in the end, it was hard to capture the visual geometry of the NeRF environment. The amount of empty noise was reduced a lot, however, and the colors generated in the accumulation matched the expected environment of pink and camel colored floral wallpaper.

4.3 Limitations

The main limitation of our approach is that we were not able to implement an effective way to do controlled diffusion image generation. Since the areas we are masking out for inpainting are areas of high noise, there is no geometric data as to what the scene should look like. Therefore, most of the

images just become hallucinations and estimations of the general wall and color. Therefore, in the end, we saw that the artifacts had a lot of the right colors, but lacked the structure due to not having the persistence each generation.

5 Conclusion

We propose a new pipeline method to increase accumulation in generations created by neural radiance fields. Our approach uses camera walking to find new views in a 3D environment and performs inpainting to generate synthetic data about the scene for the NeRF to process. Our approach successfully increases the accumulation over time without major changes to the geometry and original volumetric data of the NeRF, which has not been done before as a part of a pipeline. Additionally, it is clear that the added accumulation were corresponding to the scene color averages. However, the overall convergence of the scene was noisy and inconsistent due to the nature of volatile and noisy generations from the diffusion model we used. Future work could include experimenting with methods of improving persistence and consistency across generations such as fine-tuning a diffusion model, prompt guidance, and image generation pooling and filtering. Those methods have had literature on them to back the idea that they may help with persistence. Additionally, improving the camera walk algorithm could give us more geometric information despite novel views. Finally, testing and improving on more different types of datasets may provide better insight on the effectiveness of this method, as we only used one interior dataset.

6 Contributions

FM proposed a project in the area of NeRFs and worked with AL to narrow it down to a topic. More specific contributions are listed in the table below.

Alex Lin	Forrest Meng
• Presentation & Paper writing • Experimentation and data collection • Camera walk implementation • Pipeline integration and implementation • Code documentation • Long-term compute power	• Presentation & Paper writing • Camera walk implementation • Masking implementation • Diffusion inpainting implementation • Literature review

References

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, 'NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis', arXiv [cs.CV]. 2020.

[2] F. Warburg, E. Weber, M. Tancik, A. Holynski, and A. Kanazawa, 'Nerfbusters: Removing Ghostly Artifacts from Casually Captured NeRFs', arXiv [cs.CV]. 2023.

[3] A. Haque, M. Tancik, A. A. Efros, A. Holynski, and A. Kanazawa, 'Instruct-NeRF2NeRF: Editing 3D Scenes with Instructions', arXiv [cs.CV]. 2023.

[4] J. Gu et al., 'NerfDiff: Single-image View Synthesis with NeRF-guided Distillation from 3D-aware Diffusion', arXiv [cs.CV]. 2023.

[5] S. Weder et al., 'Removing Objects From Neural Radiance Fields', arXiv [cs.CV]. 2022.

[6] M. Tancik et al., 'Nerfstudio: A Modular Framework for Neural Radiance Field Development', arXiv [cs.CV]. 2023.