An Exploration into 3D Generative Models with Nerfstudio

Terrance Wang
Ren Ng, Ed.
Matthew Tancik, Ed.

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-132
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-132.html

May 12, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

An Exploration into 3D Generative Models with Nerfstudio

Terrance Wang

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee

Ren Ng
Research Advisor

May 9 2023

(Date)

★ ★ ★ ★ ★ ★ ★

Angjoo Kanazawa
Second Reader

May 11 2023

(Date)

An Exploration into 3D Generative Models with Nerfstudio

by

Terrance Wang

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ren Ng, Chair
Professor Angjoo Kanazawa

Spring 2023

An Exploration into 3D Generative Models with Nerfstudio

Copyright 2023
by
Terrance Wang

Abstract

An Exploration into 3D Generative Models with Nerfstudio

by

Terrance Wang

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Ren Ng, Chair

Techniques combining Neural Radiance Fields (NeRFs) and diffusion models have shown great promise for generating novel scenes and objects from text input. Most advancements in the generative 3D space have focused on fine tuning diffusion models to improve performance, but have neglected to investigate other aspects, such as the underlying NeRF architecture or noise sampling schedule. Because this is a novel approach, there are still many open questions in this research space that have not been properly explored. In this work, we explore some of these questions using the open-source platform Nerfstudio. By implementing these models and conducting experiments in this library, we are able to easily conduct training and evaluation, while also providing code for others to build on for their own experiments in this space. The following work shows that the choice of underlying NeRF model and noise sampling schedule does have an impact on generated outputs, and lays the groundwork for further experiments in this line of investigation.

Contents

List of Figures

Acknowledgments

I would like to thank Professor Ren Ng, for his continued support and feedback throughout this program. I am grateful for Professor Angjoo Kanazawa and the Nerfstudio team. It has been a pleasure working with them over the last year, and I have learned so much from them through this experience. I would also like to thank my mentor, Matthew Tancik, who has offered me so much kindness and patience since the start of my research journey. I could not have asked for a better mentor, and I am endlessly grateful for all the advice and guidance he's given me over the years. Finally, I would like to thank my friends and family, whose support is the reason why I am here today.

Chapter 1

Introduction

3D reconstruction and novel view synthesis are two fields that have received a lot of research focus in the last few years, largely due to the introduction of Neural Radiance Fields (NeRFs). Since the release of the first NeRF paper [15], these models have seen great popularity due to their ability to reconstruct detailed real world scenes with high accuracy. A trained NeRF model can not only render out new viewpoints of a given scene, but also extract useful 3D information from the training data, such as depth maps, surface normals, and meshes. This means these models are a way to accurately convert from 2D images to 3D information, which is necessary for many downstream applications. Some common examples of these applications include visual effects editing, creating 3D assets, and robotics training simulations. The active and fast-paced research in this space has led to many key improvements in memory [4], speed [16][4][27], and model architecture [3][2][11], making these models more accessible for the average user.

Within the last year, a new application of NeRFs has emerged, focusing on generating new objects and scenes rather than reconstructing real world data. This approach combines NeRFs with image generation models, such as Stable Diffusion [21] and Imagen [24], to create text-prompted 3D generative models. For the first time, users could describe an object or scene, and see their ideas come to life in 3D. This area of research saw a lot of interest due to the potential value of a high quality 3D generative model. 3D modeling was originally a time consuming and highly technical task, and companies that needed 3D models, such as video game developers and VFX studios, had to hire experts to create them. Similar to how image generation models lowered the barrier to entry for digital art and photo editing, 3D generative models are poised to make 3D modeling more accessible as well.

Because this is a new and fast-moving area of research, there are still many open questions regarding best practices for training these models. While there have been many papers [12] [20] [9] investigating how to optimize diffusion models for generative 3D, there has been little research exploring better NeRF representations. The goal of this thesis is to investigate how certain training and model choices affect the quality of 3D generative models. Specifically, we will analyze how different NeRF architectures and different noise sampling schedules affect the resulting generated 3D models. In the process of answering these questions, these models

were implemented in the open-source library Nerfstudio [28], which is a modular framework designed for NeRF development. This framework made it easy to swap between different NeRF feature representations, model architectures, and ray sampling methods, minimizing the code that needed to be reimplemented while also providing a foundation for others to experiment with 3D generative models. The Nerfstudio library also provided a real-time viewer, which was essential for accurately evaluating the performance of these models.

This work focuses on bringing two existing text to 3D methods, Dreamfusion [18] and Latent-NeRF [14] into Nerfstudio, and running experiments to explore these potential improvements in the 3D generative space. With these contributions, we show that diffusion model optimization is not the only way to improve 3D generation, and dig into how different NeRF architectures and noise sampling schedules might provide more stable training and higher quality results.

Chapter 2

Background and Related Work

2.1 Diffusion models

Diffusion models [7] are a class of generative models used for image generation. These types of models utilize two processes: a forward process where noise is iteratively added to remove structure from input images, and a reverse process that learns to recover structure from noisy input to reconstruct the original input. The goal of the diffusion model is to accurately learn this reverse process, so that they can generate new images by taking in random noise and iteratively applying the denoising process. These models have seen popularity since release because they were straightforward to train and produce a wide distribution of high quality results.

Diffusion models replaced Generative Adversarial Networks (GANs) [6] as the go to image generation model because they did not suffer from the same issues that GANs often faced, such as training instability and mode collapse. Furthermore, these models were straightforward to modify for additional functionality, such as including CLIP [19] text embeddings as additional input to create a text to image model [24], or conditioning the input noise on an input image to create an image to image model [23].

One issue with the original diffusion model was that it was computationally expensive to train and run because it operated directly in high dimensional image space. Diffusion models were made more accessible for regular users with the introduction of latent diffusion models [21]. These models first use an encoder network to convert images to a lower dimensional latent space and perform the noising and denoising steps in this latent space.

2.2 3D reconstruction

The objective of this problem is to recover a 3D representation of an object or scene given 2D input images. Existing methods involve using point clouds [10], voxel grids [13], or signed distance functions [17] to represent the target, and often require a large set of training images as well as corresponding camera pose estimates in order to obtain accurate results. There

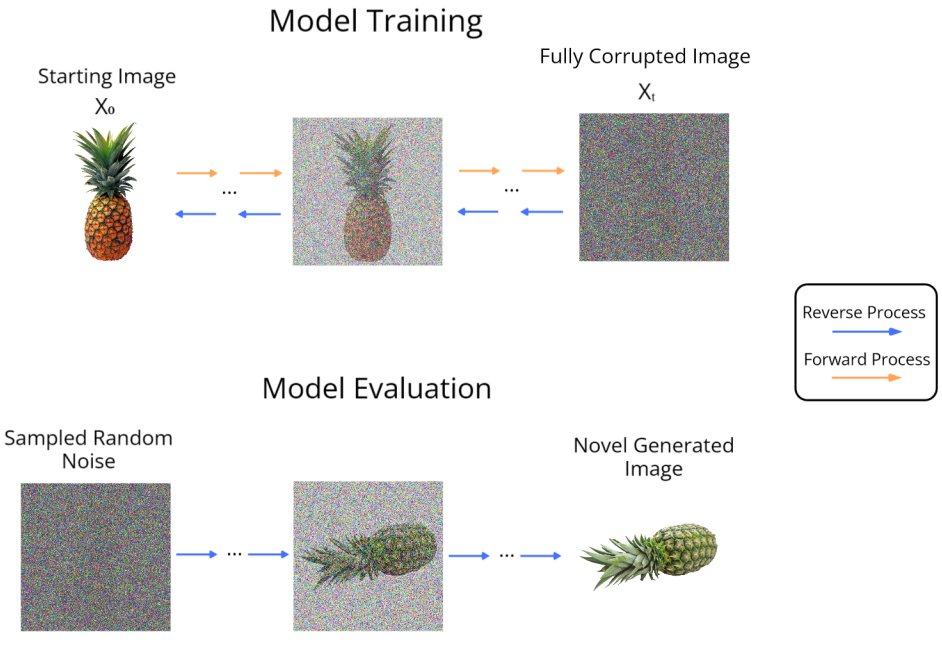

Figure 2.1: Diffusion models are trained to reconstruct images from noisy images by predicting the parts of the image that are noise. The noising and denoising are gradually applied iteratively to help these models train better and produce high quality results.

are many practical applications for recovering good solutions to this problem. An accurate 3D representation can be used to generate 3D models for assets or animation, depth maps for scene editing, or training environments for self-driving and robotics simulations. 2D data is also easier to capture, and there are many existing sources of data on the internet, making it valuable to be able to accurately convert from 2D data to 3D information.

2.3 Neural Radiance Fields

Many recent methods have used Neural Radiance Fields (NeRFs) [15], as an approach to solve 3D reconstruction. NeRFs model appearance and geometry with radiance fields, which map spatial coordinates and view direction to density and color values. Many papers have shown that a small MLP with positionally encoded input coordinates can learn to represent a target scene with a high degree of accuracy [26] [17]. In NeRFs, a small MLP learns to represent the target radiance field. Through standard volume rendering procedures, rays can be sampled, evaluated, and converted to image pixels, and the model optimizes over mean squared error between the outputted RGB and the training images. The radiance field

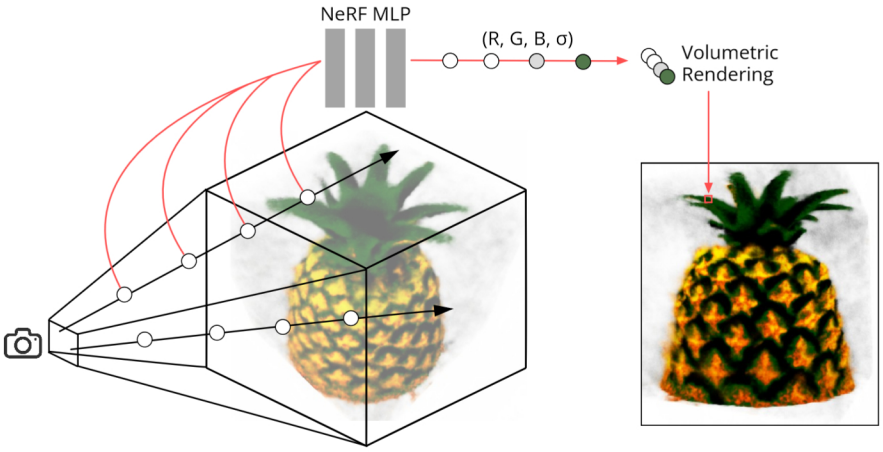

Figure 2.2: How images are rendered from NeRFs. Rays are shot through the scene based on the chosen camera position and intrinsics. Samples along the ray are passed through the NeRF MLP, which outputs the corresponding RGB and density for each point. The ray samples are accumulated using volumetric rendering to convert each ray to a pixel's RGB value.

can then be rendered out as images, depth maps, or converted to a mesh for downstream applications.

While NeRFs achieved state of the art reconstruction quality, the model described in the original paper had a couple serious limitations for practical use. It assumed that the target object or scene was completely stationary and unchanging throughout all training images. It was reliant on structure from motion algorithms such as COLMAP [25] to provide accurate camera pose estimates for the training data. It was also slow and expensive to train each NeRF, taking a couple hours for a single scene.

However, as research in this area progressed, many papers came out that addressed these original flaws. Nerf-W [11] enabled training NeRFs on in-the-wild datasets, which consist of images with inconsistent lighting conditions and transient objects, by training a separate feature vector and MLP head to model these inconsistencies. TensoRF [4] and Instant NGP [16] both drastically improved training speeds by using trainable feature vectors as input to the MLP. MipNeRF 360 [2] introduced another way to speed up training by training a small proposal network to supervise where points should be sampled along a ray, reducing the number of samples queried on the larger NeRF MLP. NeRFs can now train 100 times faster, are more robust to data inconsistencies, and create more accurate reconstructions than before. As a whole, these improvements caused people to consider NeRFs as a tool

that could be practically used in more situations.

2.4 3D Generative Models

Recent papers have explored alternative ways of training NeRF models with language and diffusion models, enabling the creation of text to 3D models. For example, DreamFields [8] trains a NeRF by directly minimizing the CLIP loss between an input caption and randomly rendered viewpoints from the NeRF. Other approaches, such as Dreamfusion [18] and Score Jacobian Chaining [29], use losses that directly leverage the denoising step of a diffusion model to predict NeRF weight updates. Following Dreamfusion, approaches such as Dreambooth3D [20] and RealFusion [12] emerged, enhancing the controllability of these models by incorporating example images along with text as input. These methods rely on fine tuning parts of the diffusion model, either with Dreambooth [22] or textual inversion [5], to generate 3D models from even a single image. While it is uncertain if the resulting 3D models from these methods are good enough for use in downstream applications yet, this is an extremely fast paced area of research, with new ideas and papers being published every few weeks.

2.5 Nerfstudio

Nerfstudio [28] is an open-source library that was created in response to the rapid pace of advancements in the field of NeRF research, as well as the lack of a centralized repository that implemented them in a clean and easy-to-use format. Many of these advancements only involve modifications to one component of the NeRF model, such as the feature representation, loss function, or MLP output, meaning they could potentially be in conjunction. However, these improvements are often spread out across multiple code bases, making it difficult for researchers and users to use them in their own projects.

Nerfstudio aims to solve this problem by providing a modular open-source library that incorporates improvements from various major NeRF-related papers. This modular approach allows for easy integration of new ideas and methods and for mixing and matching different methods to suit the user's needs.

The library is designed to be accessible for regular users, with simple installation and setup instructions. Additionally, it includes a real-time viewer that enables the user to traverse the scene as it is training and monitor their model's progress in real-time. The viewer is crucial for accurately evaluating model performance, as it allows the user to zoom in and examine the reconstruction in a dynamic way, which can reveal previously hard to notice artifacts. With the viewer, users can also avoid separately rendering out images or videos for evaluation, thereby saving time and making the training process more efficient.

Chapter 3

Methods

In order to run experiments on how NeRF architecture and noise sampling schedules impact 3D generative models, I first brought these methods into Nerfstudio. This chapter will describe the two novel 3D generative methods that were added to the library, and further implementation details and modifications.

3.1 Dreamfusion

Problem Setup

Dreamfusion [18] is a generative approach that utilizes priors learnt by a diffusion model to train a NeRF from text inputs. Unlike text to image models, which have benefited from an abundance of text-image pair data, no such datasets exist for 3D. Therefore, Dreamfusion leverages pre-trained 2D diffusion models to provide supervision for training in 3D. This approach is similar to Dream Fields [8], which employs a frozen CLIP model that is also trained on text-image pairs. Dream Fields trains a NeRF to maximize the similarity score between rendered images and an input caption for all viewpoints. Dreamfusion and its followup works use a diffusion model as the score generator instead, which is used to provide updates for the weights of the NeRF model.

SDS loss

The loss function used by these diffusion-based text to 3D methods is called the SDS loss, or Score Distillation Sampling loss, and is first introduced in the Dreamfusion paper. The intuition behind this loss function is that the NeRF model should look like a valid diffusion model output from all view directions.

The loss function for optimizing a typical NeRF is:

$$\mathcal{L} = \sum_{i=1}^{N} |F_\theta(r_i) - x_i|_2^2 \tag{3.1}$$

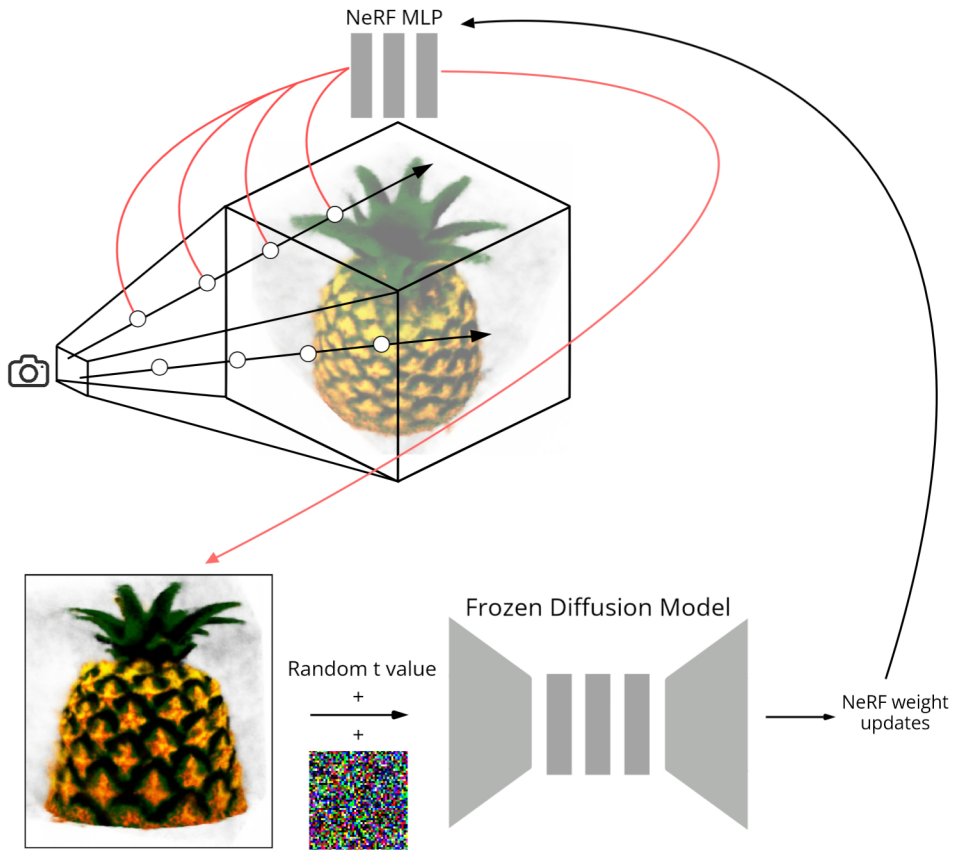

Figure 3.1: Instead of iteratively applying noise to the NeRF output, we sample one t value that determines how the noise strength for each training step. The predicted noise from the diffusion model is then directly used to update and train the NeRF.

where each x_i is a training image in the dataset, each r_i is the batch of rays that correspond with the camera intrinsics of x_i, and the loss is the L2 error between the pixel values of x_i and the RGB output from the NeRF model F_θ.

In the text to 3D setup, the diffusion model predictions take the place of ground truth images. At each training step, a random view direction and random t value is sampled evaluated by the NeRF model. The t value affects the strength of the noise ϵ that is applied to the NeRF output. The loss function then optimizes NeRF weights θ to minimize the difference between the denoiser UNet's predicted noise vs. the actual noise added.

$$x_{noisy} = w(t)F_\theta(r) + (1 - w(t))\epsilon \tag{3.2}$$

$$\mathcal{L} = (U(x_{noisy}, t, text) - \epsilon) \tag{3.3}$$

Directly taking the gradient of this loss with respect to θ would involve backpropogating through the diffusion model UNet, which is expensive to compute due to the UNet's size. Instead, we skip this and directly use the difference between the predicted and actual noise to get the update direction for θ.

$$\nabla_\theta \mathcal{L} = w(t)(U(x_{noisy}, t, text) - \epsilon)\frac{\partial x_{noisy}}{\partial \theta} \tag{3.4}$$

Implementation Details

Dreamfusion employs various techniques and regularizers to help with training and enhance the performance of their model. One such method involves location-based sampling of the target text caption. During each training step, the input caption is modified based on the location of the sampled training camera. Four possible modifications are used, which involve appending the strings ", front view", ", side view", ", back view", and ", top view" to the caption. These location-based prompts are crucial for generating coherent results. The original paper illustrates that neglecting these prompts can lead to multiple faces or incoherent geometry, as the diffusion model tends to default to the front-facing view of an object given a regular target caption.

Dreamfusion also employs shading-based regularizers to improve model performance. After the model has partially converged, the RGB image passed to the loss function will sometimes be randomly replaced with a textureless shaded version of the output. This shaded output is calculated by randomly sampling a light source position, replacing the output RGB with white, and rendering using Lambertian shading. This regularization encourages the model to produce objects with better surfaces, and prevents the degenerate solution of creating a flat surface that looks correct from most viewing directions.

Changes and Improvements

One significant difference between our Dreamfusion implementation and the original paper is the use of the open-source model Stable Diffusion as the underlying diffusion model. This is Dreamfusion used Imagen as its diffusion model, which does not have publicly available weights. Unlike Imagen, Stable Diffusion is a latent diffusion model, meaning that its diffusion process operates on an image's latent space, while Imagen operates directly on image pixels. This change in diffusion model has a few key consequences for our model training and results.

Although we render our NeRF at 64x64 per training step, which is the same as Dreamfusion, we must upsample our rendered image to 512x512 because the image must first be passed through an autoencoder to encode it into the latent space. The autoencoder transforms our 512x512x3 dimensional input into a 64x64x4 dimensional latent embedding. However, the encoding and decoding steps may result in a slight loss of information from the input images. Additionally, embedding each NeRF render into the latent space at each training step slows down the training process, especially because we must also backpropogate through the autoencoder in order to get our weight updates.

The underlying NeRF model in our implementation is a simplified version of the Nerfacto model, which combines proposal sampling from Mip-NeRF 360 [2] and feature hash grids from Instant-NGP [16]. This simplification enables faster model training with comparable quality.

3.2 Latent NeRF

Problem Setup

This paper's [14] key insight is to avoid running the forward pass and backpropogating through the autoencoder, by training the NeRF to directly output in the 4-dimensional latent space of the diffusion model instead of RGB. The same volume rendering procedures can be applied to render the latent input to the diffusion model U-Net, resulting in comparable quality to Dreamfusion results with a shorter training time. At evaluation, these latents are passed through Stable Diffusion's decoder to convert back to image space.

Besides this change in output representation, the method is largely similar to the one proposed by Dreamfusion. An SDS loss is applied to use the diffusion model as a score generator for guiding NeRF training. Furthermore, the directional text prompts are used to produce coherent geometry.

3.3 Nerfstudio Viewer

The real-time viewer in NerfStudio is a valuable asset for experimenting with these generative 3D models by streamlining the training and evaluation process. Traditional methods of

tracking and assessing model performance during training involve periodically calculating metrics, rendering images, and visualizing them using tools such as Tensorboard or Weights and Biases. While this approach works fine for monitoring conventional NeRF training, it proves to be more problematic when evaluating generative 3D models.

With traditional NeRFs, model accuracy can be determined by calculating the PSNR between the model's RGB and ground truth. However, with generative models, there are no easy metrics to gauge model performance. Furthermore, rendering out single images is not sufficient for evaluating the model either. This is because a common failure mode for these types of models is generating a multi-headed output for non-rotationally symmetric objects, such as animals. The model may appear to produce coherent results when viewed from a single image, but when rendered as a video or viewed from multiple angles, it becomes apparent that the model does not represent the target object. Evaluating the model by rendering videos during training is impractical because it is significantly slower than rendering a single image. For these reasons, the real-time viewer was invaluable for parameter tuning and model evaluation.

Chapter 4

Experiments and Results

4.1 Open questions in 3D generation

Does the underlying NeRF model matter?

The original Dreamfusion paper uses Mip-NeRF 360 as its NeRF representation. This paper has two main contributions: scene contraction, which helps model performance on unbounded scenes, and proposal networks, which provide better predictions of where to sample points along a ray. While Mip-NeRF 360 achieves high quality reconstruction results, these contributions are less useful in the 3D generative setting. 3D generative scenes are not unbounded, and proposal network sampling matters less when sampling in a small bounding box that is mostly taken up by the generated object. In fact, models such as Instant NGP and TensoRF can achieve similar reconstruction quality and faster training times on these small bounded scenes. In these experiments, we will compare Latent-NeRF along with Dreamfusion using three different NeRF backends: Instant-NGP, TensoRF, and Nerfacto, which combines proposal networks with Instant-NGP hash encodings. We will analyze if these NeRF representations can achieve similar generation quality, and if there are any advantages to choosing one representation over another.

Is there a better way to sample noise?

As described previously, during training random t values are sampled for the SDS loss [18]. Random sampling is chosen primarily for convenience, because it is difficult to manually find a sampling schedule for t that will work well for a variety of prompts. However, as the model converges, intuitively we would want to use smaller t values and add less noise to keep the existing structure intact and focus on sharpening fine detail. In this experiment, we will explore different t sampling procedures and what effect they have on model training and convergence.

4.2 Different NeRF Representations

Experiment Details

The four methods being compared in this experiment are: Dreamfusion, Dreamfusion with Instant-NGP, Dreamfusion with TensoRF, and Latent-NeRF. As mentioned previously, this Dreamfusion implementation uses the Nerfacto [28] NeRF model, which combines the ray sampling method described in Mip-NeRF 360 with the feature hash encoding from Instant-NGP. Dreamfusion with Instant-NGP replaces this Mip-NeRF 360 ray sampling method with the occupancy grid based sampling described in Instant-NGP. Dreamfusion with TensoRF samples rays based on the learned density tensor components and learns RGB with the color tensor components. Lastly, this Latent-NeRF implementation also uses Nerfacto as its underlying NeRF representation.

Each model uses the hyperparameter values that were used in their original papers, and each model was trained for 15k steps on a 12GB GPU. The Dreamfusion models took approximately an hour to train, while Latent-NeRF trained twice as fast, which is consistent with the training time described in the Latent-NeRF paper. In the figures below, four views roughly in a circle around the object are rendered for each prompt and model architecture.

Analysis

As seen in Figure 4.2 below, it is not a straightforward task to compare these different methods. Unlike in 3D reconstruction, there is no ground truth to compare with or compute metrics on. There is also an infinite number of prompts that could be given as input, with some being easier to generate than others. As seen in the pineapple example, all four methods are able to generate a plausible and detailed pineapple. However, even with this simple prompt, we can start to observe some of the differences between each method.

Dreamfusion and Latent-NeRF both display a tendency to place clouds of density around an object, which might be attributed to the proposal network overfitting to the SDS loss. Dreamfusion with Instant-NGP has a tendency to produce oversaturated and unrealistic colors. Dreamfusion with TensoRF produces good color and geometry, but closer examination in Figure 4.3 reveals high frequency artifacts along the surface of the model, likely due to the underlying tensor decomposition that represents the scene. Latent-NeRF's outputs can appear smudged with some prompts because it learns the scene directly in latent space, but it generally also reconstructs good color and geometry.

The frog scene is a more complicated prompt because it is not rotationally symmetric. In this scene, we can see an example of the multi-headed failure case. In the second and third columns of the Dreamfusion output, we can see the model start to generate three eyes and multiple mouths. We can also start to see this effect in the Latent-NeRF frog. This effect is even more prominent when evaluating in the viewer because the multi-headed effect can only be seen from certain view directions, as shown in Figure 4.1. This failure mode occurs because diffusion models are unable to correctly render an object from various viewpoints,

Figure 4.1: Images from frog prompt rendered in a circle from the viewer. Most view directions look good, but there are certain angles where the multi-headed issue can been seen.

and will tend to produce a front-facing viewpoint, even if it was prompted for a side view or back view. The multi-head issue is explained by the NeRF overfitting to the front-biased SDS loss.

The banana prompt was included to test the ability of these models to capture fine surface texture and detail. The Dreamfusion approaches are able to accurately capture the crochet texture of the banana prompt, while Latent-NeRF only generates a fuzzy texture. Latent-NeRF performs a lot better on the less object-centric castle scene, while the Dreamfusion approaches fail to converge.

While no model was definitively superior across all prompts, each produced good results on at least one input. This speaks to the difficulty of training these generative models, in large part because the problem of text to 3D generation is heavily under-constrained. With more hyperparameter tuning and better regularization terms, all of these methods have the potential to perform well.

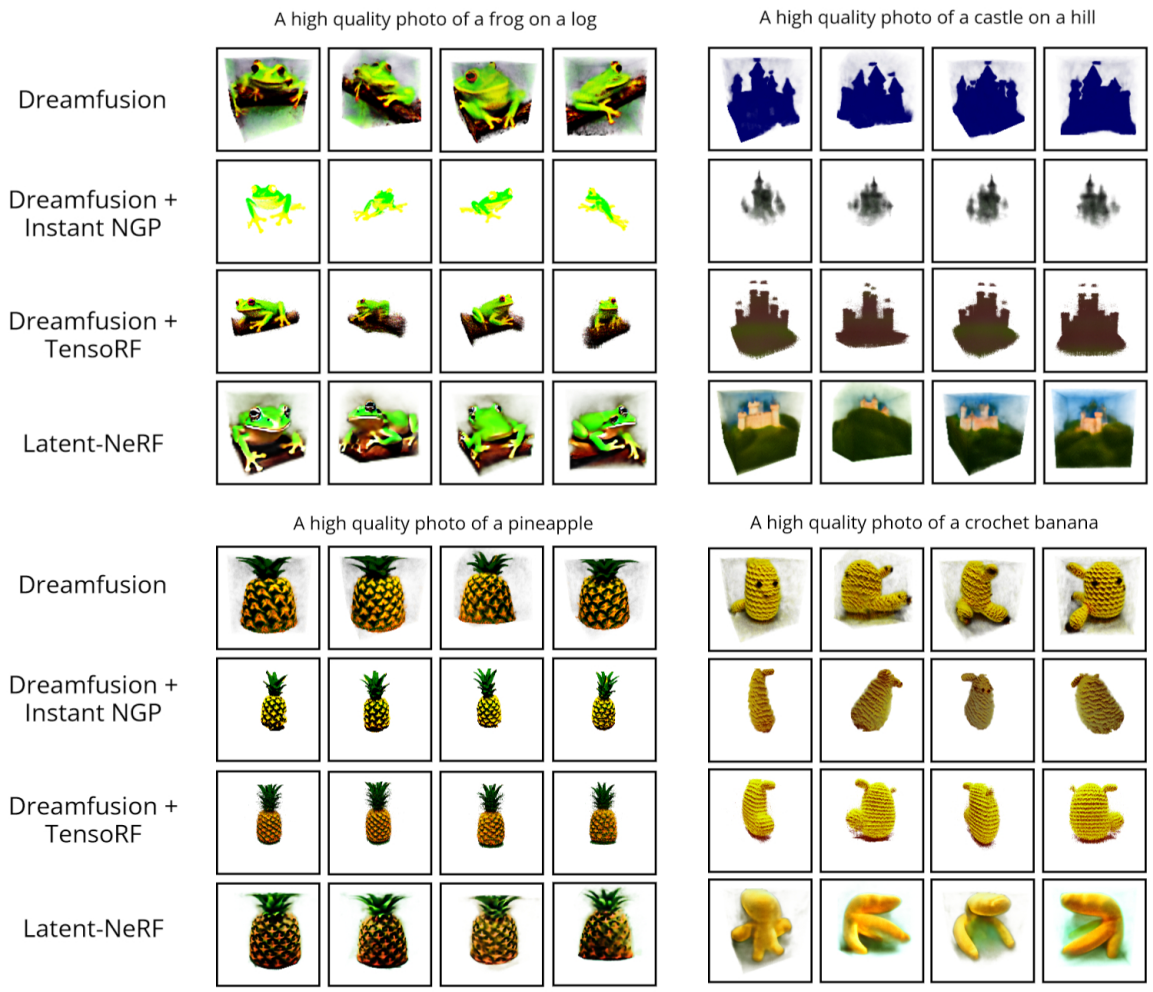

Figure 4.2: Rendered images from different 3D generative architectures with different prompts. The rotationally symmetric pineapple is the easiest prompt to reconstruct, while the other three are more challenging due to having facial features and fine details.

Figure 4.3: Enlarged version of above images

4.3 Scheduling noise sampling

Experiment Details

This experiment focused on using Dreamfusion to determine how t value sampling affects model performance. In typical Dreamfusion training, a random t value between 0.02 and 0.98 is sampled at each train step, with larger values of t corresponding with more added noise when calculating the SDS loss. In this set of experiments, the upper bound of random t values is linearly interpolated as training progresses, from 0.98 to either 0.75, 0.5, or 0.25. We evaluate on two prompts: a crochet banana and a blue jay. These prompts were chosen to see if the surface details of the banana became sharper, and if the multi-headed issue would be alleviated when training the blue jay.

Analysis

The intuition behind this experiment is that as the NeRF model gets closer to learning our target scene, we want to apply less drastic changes with the SDS loss. One theory explaining the multi-headed issue is that it occurs from applying lots of noise to the SDS loss after the model geometry has converged. This would encourage the NeRF to start generating multiple faces to overfit to the SDS loss.

In reality, it seems that this naive way of adjusting t sampling does not significantly sharpen fine detail or help model convergence. In figure 4.4, there seems to be little difference between the four experiments. As we interpolated to smaller max t values, there does seem to be a slight improvement in surface texture quality, but there are still blurry and incoherent areas as well. Overall, it seems that t value interpolation did not seem to hurt model performance, but did not significantly improve the output quality either.

In figure 4.5, we see another example of how these models can be unstable while training. The control experiment and the max t = 0.25 experiment both converged to coherent geometry, though they both still slightly suffer from the multi-headed issue. The t = 0.75 and 0.5 experiments failed to converge, and display the failure mode of learning multiple 2D images rather than a coherent 3D structure. These results seem to suggest that for harder scenes, max t interpolation can prevent the model from converging, and may require further tuning.

4.4 Future Work

These experiments show many promising lines of further exploration into 3D generation. Models such as the Dreamfusion + TensoRF model can likely see even better performance with more hyperparameter tuning, and learning the tensor decomposition for a scene seems to provide more stable training than learning a hash grid of features. Adjusting the learning rate and resolution for these tensor components may solve the high frequency artifacts and may help produce a better object surface.

Another avenue of investigation would be using the real-time viewer in Nerfstudio as a way to monitor model progress and interactively adjust the max t value accordingly. Because each prompt may train at a different speed, the viewer would be useful for decreasing the sampled t values only after the model geometry has converged.

Finally, there are many recently released techniques that improve on the diffusion model to increase alleviate the multi-headed issue [12] [1]. Implementing these methods into Nerfstudio and running similar ablations on NeRF model architecture might also lead to more interesting results.

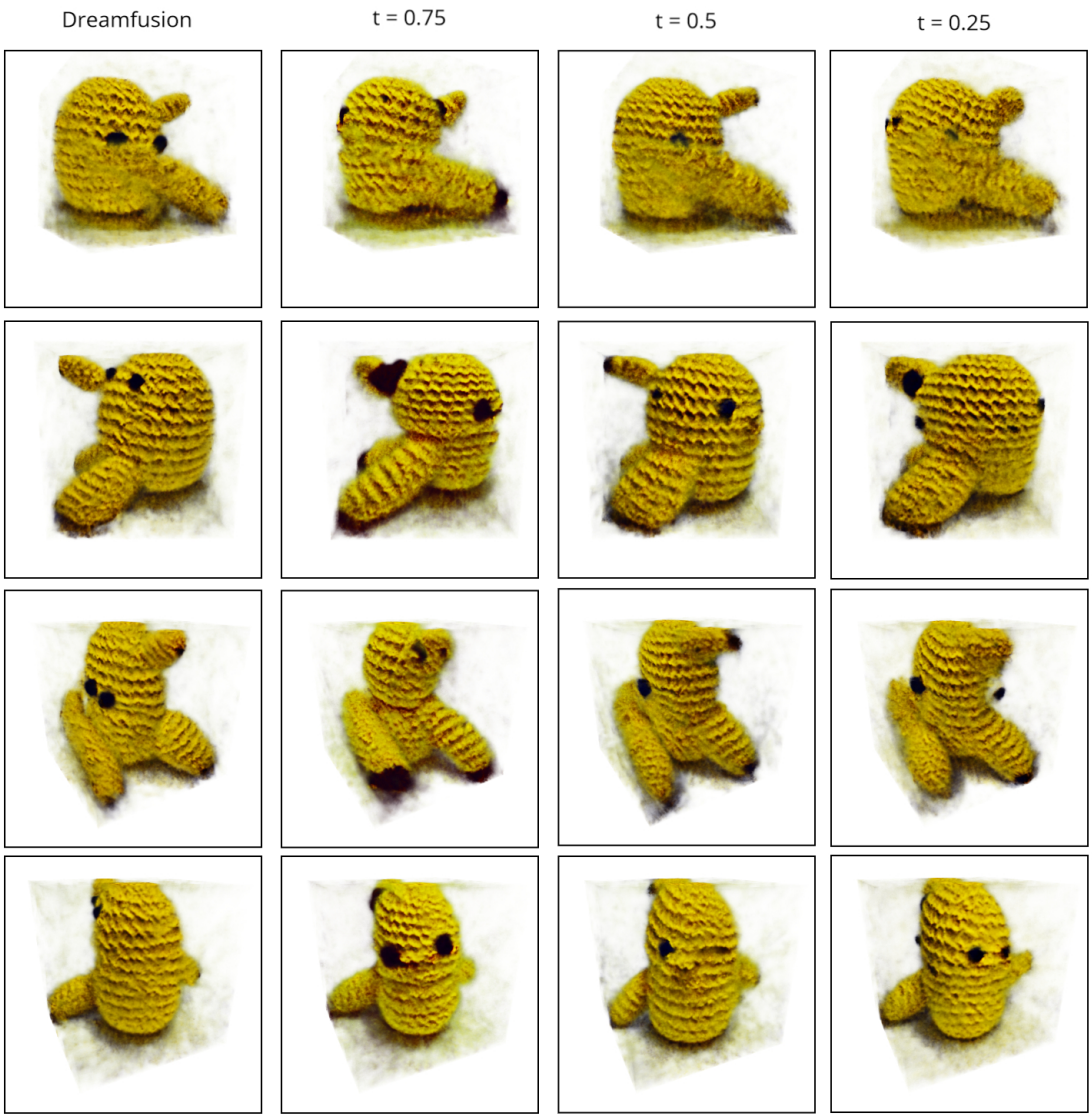

Figure 4.4: The given text caption is “a high quality photo of a crochet banana”.

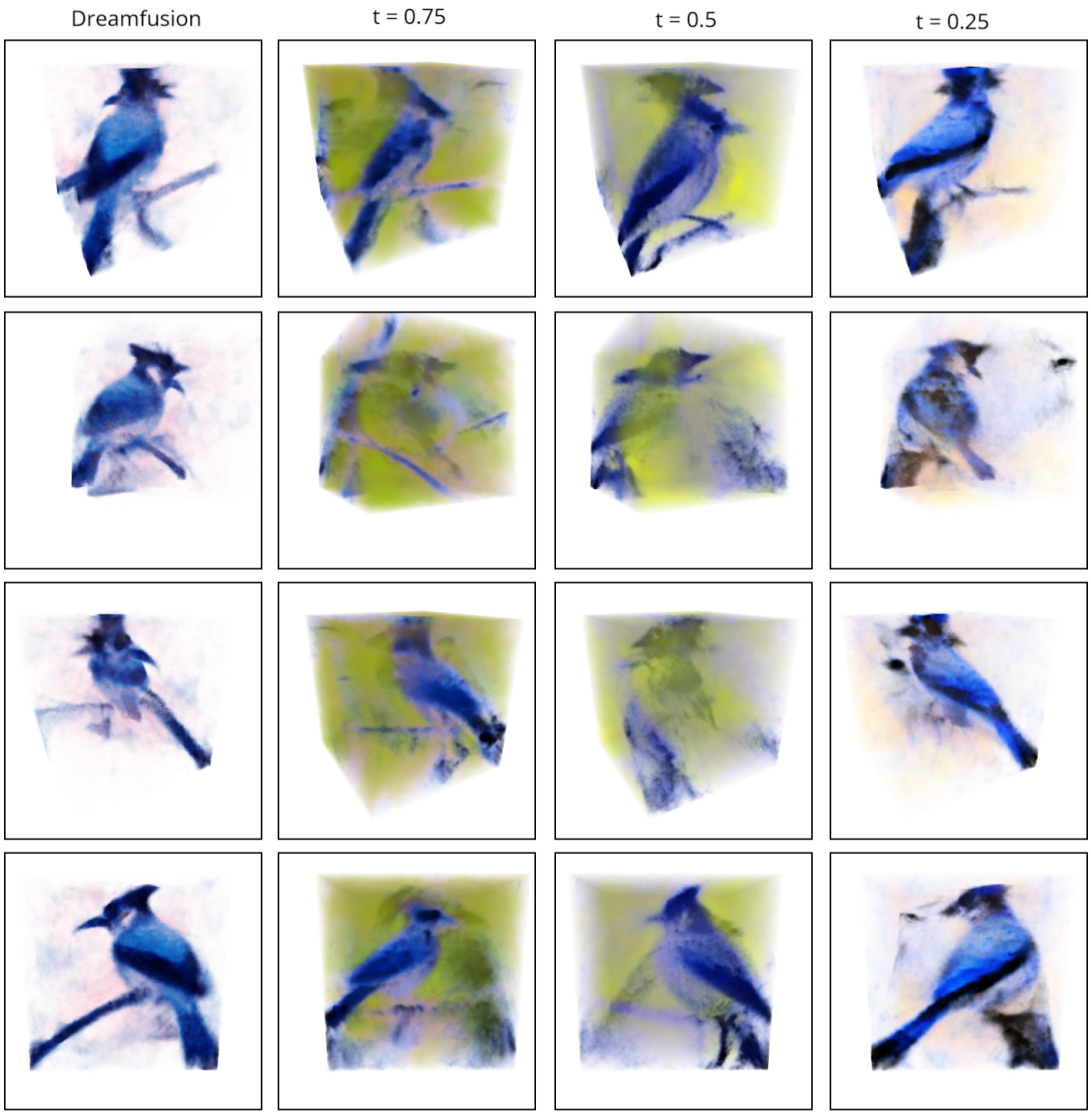

Figure 4.5: The given text caption is “a high quality photo of a perched blue jay bird, highly detailed”.

Chapter 5

Conclusion

These experiments show that 3D generative research should consider modifications not only in the diffusion model, but in the entire 3D generative pipeline. Switching between different underlying NeRF representations can have significant effects on the resulting outputs, and different approaches to sampling t values can effect how the model converges on fine details in the scene. It is still unclear which combination of approaches will lead to the ultimate goal of a model that consistently produces high quality results on a wide variety of prompts.

Although there are many more experiments to be done, implementing these 3D generative models in Nerfstudio is a good starting point to accelerate further research in this field. Nerfacto, TensoRF, and Instant-NGP are all promising NeRF models for use with 3D generation, and as the field of NeRF research progresses even further, it will be crucial for 3D generation to investigate and incorporate these improved NeRF models as well.

Bibliography

[1] Mohammadreza Armandpour et al. *Re-imagine the Negative Prompt Algorithm: Transform 2D Diffusion into 3D, alleviate Janus problem and Beyond*. 2023. arXiv: 2304.04968 [cs.CV].

[2] Jonathan T Barron et al. "Mip-nerf 360: Unbounded anti-aliased neural radiance fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5470–5479.

[3] Jonathan T Barron et al. "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5855–5864.

[4] Anpei Chen et al. "TensoRF: Tensorial Radiance Fields". In: *European Conference on Computer Vision (ECCV)*. 2022.

[5] Rinon Gal et al. *An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion*. 2022. arXiv: 2208.01618 [cs.CV].

[6] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG].

[8] Ajay Jain et al. *Zero-Shot Text-Guided Object Generation with Dream Fields*. 2022. arXiv: 2112.01455 [cs.CV].

[9] Ruoshi Liu et al. *Zero-1-to-3: Zero-shot One Image to 3D Object*. 2023. arXiv: 2303.11328 [cs.CV].

[10] Priyanka Mandikal and R. Venkatesh Babu. *Dense 3D Point Cloud Reconstruction Using a Deep Pyramid Network*. 2019. arXiv: 1901.08906 [cs.CV].

[11] Ricardo Martin-Brualla et al. "Nerf in the wild: Neural radiance fields for unconstrained photo collections". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 7210–7219.

[12] Luke Melas-Kyriazi et al. *RealFusion: 360*deg *Reconstruction of Any Object from a Single Image*. 2023. arXiv: 2302.10663 [cs.CV].

[13] Lars Mescheder et al. *Occupancy Networks: Learning 3D Reconstruction in Function Space*. 2019. arXiv: 1812.03828 [cs.CV].

[14] Gal Metzer et al. *Latent-NeRF for Shape-Guided Generation of 3D Shapes and Textures*. 2022. arXiv: 2211.07600 [cs.CV].

[15] Ben Mildenhall et al. "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *Communications of the ACM* 65.1 (2021), pp. 99–106.

[16] Thomas Müller et al. "Instant neural graphics primitives with a multiresolution hash encoding". In: *arXiv preprint arXiv:2201.05989* (2022).

[17] Jeong Joon Park et al. *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation*. 2019. arXiv: 1901.05103 [cs.CV].

[18] Ben Poole et al. "DreamFusion: Text-to-3D using 2D Diffusion". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=FjNys5c7VyY.

[19] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].

[20] Amit Raj et al. *DreamBooth3D: Subject-Driven Text-to-3D Generation*. 2023. arXiv: 2303.13508 [cs.CV].

[21] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV].

[22] Nataniel Ruiz et al. *DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation*. 2023. arXiv: 2208.12242 [cs.CV].

[23] Chitwan Saharia et al. *Palette: Image-to-Image Diffusion Models*. 2022. arXiv: 2111.05826 [cs.CV].

[24] Chitwan Saharia et al. *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. 2022. arXiv: 2205.11487 [cs.CV].

[25] Johannes Lutz Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[26] Vincent Sitzmann et al. *Implicit Neural Representations with Periodic Activation Functions*. 2020. arXiv: 2006.09661 [cs.CV].

[27] Cheng Sun, Min Sun, and Hwann-Tzong Chen. "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5459–5469.

[28] Matthew Tancik et al. *Nerfstudio: A Modular Framework for Neural Radiance Field Development*. 2023. arXiv: 2302.04264 [cs.CV].

[29] Haochen Wang et al. *Score Jacobian Chaining: Lifting Pretrained 2D Diffusion Models for 3D Generation*. 2022. arXiv: 2212.00774 [cs.CV].