A Modular Framework for Neural Radiance Field Development

Matthew Tancik* Ethan Weber* Evonne Ng* Ruilong Li
Brent Yi Justin Kerr Terrance Wang Alexander Kristoffersen Jake Austin
Kamyar Salahi Abhik Ahuja David McAllister Angjoo Kanazawa

UC Berkeley

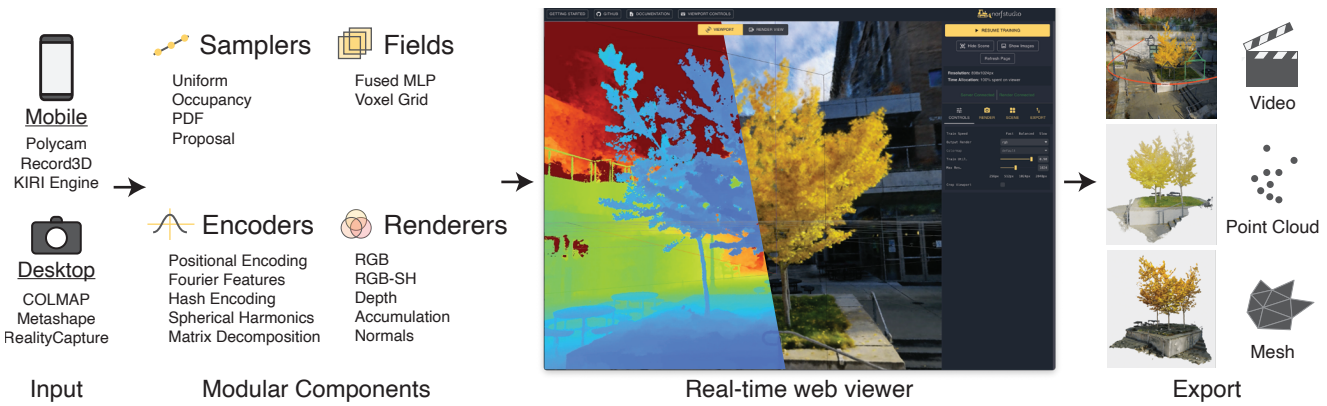

Figure 1. **Nerfstudio framework.** Nerfstudio is a Python framework for Neural Radiance Field (NeRF) development. Nerfstudio supports multiple input data pipelines, is built around multiple modular core NeRF components, integrates with a real-time web viewer, and supports multiple export modalities. The goal of the Nerfstudio framework is to simplify the development of custom NeRF methods, processing of real-world data, and interacting with reconstructions.

Abstract

Neural Radiance Fields (NeRF) are a rapidly growing area of research with wide-ranging applications in computer vision, graphics, robotics, and more. In order to streamline the development and deployment of NeRF research, we propose a modular PyTorch framework, Nerfstudio. Our framework includes plug-and-play components for implementing NeRF-based methods, which make it easy for researchers and practitioners to incorporate NeRF into their projects. Additionally, the modular design enables support for extensive real-time visualization tools, streamlined pipelines for importing captured in-the-wild data, and tools for exporting to video, point cloud and mesh representations. The modularity of Nerfstudio enables the development of Nerfacto, our method that combines components from recent papers to achieve a balance between speed and quality, while also remaining flexible to future modifications. To promote community-driven development, all associated code and data are made publicly available with open-source licensing at https://nerf.studio.

*Denotes equal contribution

1. Introduction

Neural Radiance Fields (NeRFs) [22] are gaining popularity for their ability to create 3D reconstructions in real-world settings, with rapid research in the area pushing the field forward. Since the introduction of NeRFs in 2020, there has been an influx of papers focusing on advancements to the core method including few-image training [47, 54], explicit features for editing [18, 45, 57], surface representations for high-quality 3D mesh exports [27, 46, 51], speed improvements for real-time rendering and training [25, 41, 53], 3D object generation [30], and more [50].

These research innovations have driven interests in a wide variety of disciplines in both academia and industry. Roboticists have explored using NeRFs for manipulation, motion planning, simulation, and mapping [2, 5, 8, 15, 39, 60]. NeRFs are also explored for tomography applications [35], as well as perceiving people in videos [29]. Visual effects and gaming studios are exploring the technology for production and digital asset creation. News outlets capture NeRF portraits to tell stories in new formats [49].

The potential applications are vast, and even startups [*] are emerging to focus on deploying this technology.

Despite the growing use of NeRFs, support for development is still rudimentary. Due to the influx of papers and lack of code consolidation, tracking progress is difficult. Many papers implement features in their own siloed repository. This complicates the process of transferring features and research contributions across different implementations. Additionally, few tools exist to easily run NeRFs on real-world data collected by users. To address these challenges, we present Nerfstudio (Fig. 1), a modular framework that consolidates NeRF research innovations and makes them easier to use in real-world applications.

Furthermore, while NeRFs solve an inherently visual task, there is a lack of comprehensive and extensible tools for visualizing and interacting with NeRFs trained on real-world data. Despite the availability of several NeRF repositories, existing implementations are often focused on achieving state-of-the-art results on metrics such as PSNR, SSIM, and LPIPS. These evaluations are typically based on held-out images along the capture trajectory that are similar to the training images. This often makes them misleading indicators of performance for many real-world applications when data is captured in unstructured environments and novel views are rendered with large baselines. Qualitative evaluations have historically been a challenge due to the computational demands of NeRF, which often resulted in rendering times up to multiple seconds per image. Recent developments such as Instant-NGP [25] significantly reduce computational overhead, enabling real-time training and rendering. However, Instant-NGP relies significantly on GPU accelerations with custom CUDA kernels, making development and quick prototyping a challenge. We present a framework that enables interactive visualizations while also being flexible and model-agnostic.

Nerfstudio is an extensible and versatile framework for neural radiance field development. Our design goals are the following:

1. Consolidating various NeRF techniques into reusable, modular components.
2. Enabling real-time visualization of NeRF scenes with a rich suite of controls.
3. Providing an end-to-end, easy-to-use workflow for creating NeRFs from user-captured data.

For modularity, we devise an organization among components across various NeRFs that allows abstracting away method-specific implementations. Our real-time visualizer is designed to work with any model during training or testing. Furthermore, the visualizer is hosted on the web, making it accessible without requiring a local GPU machine.

[*] https://lumalabs.ai/

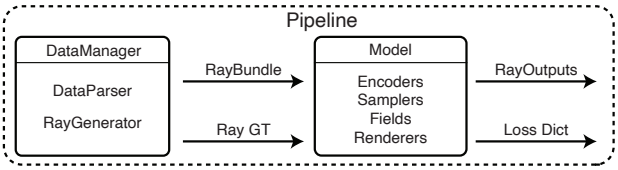

Figure 2. **Pipeline components.** Each NeRF method is implemented as a custom Pipeline. DataManagers process input images into bundles of rays (RayBundles) that get rendered by the Model to produce a set of NeRF outputs (RayOutputs). A dictionary of losses supervises the pipeline end-to-end.

The modular nature of our framework facilitates the integration of various data input formats, thereby simplifying the workflow for incorporating user-captured real-world scenes. We provide support for images and videos with various camera types, as well as other mobile capture applications (Polycam, Record3D, KIRI Engine) and outputs from popular photogrammetry software like RealityCapture and Metashape. In particular, integration with these applications enable users to by-pass structure-from-motion tools like COLMAP [36], which can be time-consuming. Furthermore, we provide support for multiple export formats, including video, depth maps, point clouds, and meshes.

The modularity of Nerfstudio enables developing Nerfacto, our method that combines components from recent papers to achieve a balance between speed and quality. We show that this method is comparable to the other state-of-the-art methods such as MipNeRF-360 [4] while achieving an order of magnitude speedup. We also conduct an ablation study that demonstrates its flexibility on a new in-the-wild dataset consisting of 10 in-the-wild scenes. Our findings highlight the limitations of commonly used NeRF metrics and the importance of a real-time viewer for qualitative assessments. The potential of our framework as a consolidated codebase for NeRF research is reflected in the traction thus far with extensions such as SDFStudio [56]. Furthermore, Nerfstudio is an open-source project with active improvements from both academic and industry contributors.

2. Related Works

2.1. Frameworks and tools

Software frameworks have played a crucial role in consolidating and driving the advancement of various fields. In deep learning, Caffe [13], TensorFlow [1], and PyTorch [28] provide readily usable machine learning functionalities. Similarly, frameworks such as PyTorch3D [34] and Kornia [9] provide reusable components for 3D computer vision tasks. Other examples of frameworks include Mitsuba3 [11], Halide [33], Taichi [10], and Reyes [7] for graphics, Phototourism [40] and COLMAP [36–38]

Figure 3. **Nerfstudio Dataset.** Our Nerfstudio Dataset contains 10 scenes: 4 phone captures with pinhole lenses and 6 Mirrorless camera captures with a fisheye lens. We focus our efforts on real-world data, and these scenes can help benchmark progress.

for photogrammetry and visualization, and AverageExplorer [59] for data collection. Despite the diversity of topics covered, each of these frameworks originated from the need to provide reusability and reproducibility to a rapidly expanding field. In light of the fast-paced growth of NeRFs in both academia and industry, Nerfstudio aims to streamline advancements in neural rendering by offering a flexible and comprehensive framework for development.

2.2. NeRF codebases

In recent years, several codebases for NeRFs have gained popularity among the research community, including the original NeRF codebase [22], nerf-pytorch [26, 52], Nerf_pl [32], Instant NGP [25], torch-ngp, Ngp_pl, and MultiNeRF [23]. Due to the lack of consolidation, there exists a significant number of NeRF repositories that focus on improving specific components of specific algorithms. For example, Mip-NeRF [3] aims to address the anti-aliasing problem of NeRF [22] and Mip-NeRF 360 [4] addresses the limitations of Mip-NeRF. Additionally, Plenoxels [53], TensoRF [6] and InstantNGP [25] propose different approaches to address the problem of computational efficiency. Furthermore, RawNeRF [20], Ref-NeRF [44], and NeRF-W [19] each address distinct challenges related to NeRF, resulting in parallel, non-interacting implementations. Nerfstudio aims to address the lack of consolidated development in the field of NeRFs by consolidating critical techniques introduced in the existing literature. This allows for more efficient and effective experimentation with combining components from multiple solutions into a single, comprehensive method, and facilitates the ability of the community to build upon existing prior approaches.

2.3. Neural rendering frameworks

Concurrent efforts such as NeRF-Factory [12], NerfAcc [16], MultiNeRF [23], and Kaolin-Wisp [42] all make significant efforts in advancing the usability of NeRFs. While NeRF-Factory consolidates multiple prior works into a single repository, it places less emphasis on reusable modules shared across these prior works and focuses more on benchmarking. NerfAcc prioritizes pythonic modularity, but focuses primarily on the lower-level components rather than the entire pipeline. Kaolin-Wisp and Multi-NeRF each consolidate multiple paper implementations into a single repository. None of these repositories are as comprehensive as Nerfstudio in delivering our three design goals: modularity, real-time visualization, and end-to-end usability for user-captured data. Furthermore, Nerfstudio is released under an Apache2 license, which allows for its use by both researchers and companies.

3. Framework Design

The goals of Nerfstudio are to provide (1) modularity, (2) real-time visualization for development, and (3) ease of use with real data. In designing the framework, we consider trade-offs against designs that optimize for faster rendering or higher quality results on synthetic scenes. For instance, we prefer an implementation that allows for a modularized pythonic non-CUDA method over one that supports a faster, non-modularized CUDA method. Additionally, our design choices lead to simpler interfacing with an extensive visualization ecosystem which supports real-time rendering during test and train with custom camera paths. Finally, we focus on delivering results for real-world data rather than synthetic scenes to address audiences outside research including those in industry and non-technical users.

With these three goals, the design of Nerfstudio promotes collaborations by providing a consolidated platform on which people can request for or contribute to new features. The long-term goal is for Nerfstudio to continue improving through community-driven contributions.

3.1. Modularity

We propose an organization of components that is both intuitive and abstract, enabling the implementation of existing and novel NeRFs by swapping reusable components. Fig. 1 shows a subset of the components types and implementations we currently have available in Nerfstudio.

3.2. Visualization for development

The Nerfstudio real-time viewer offers an interactive and intuitive way to visualize Neural Radiance Fields (NeRFs) during both training and testing phases. To ensure ease of use, the visualizer is simple to install, works seamlessly across both local and remote GPU compute environments, supports different models, and offers a user interface for creating and rendering custom camera paths, shown in Fig. 6 (a).

Our real-time visualization interface is particularly useful for qualitatively evaluating a model, allowing for more informed decisions during method development. While metrics such as PSNR can provide some insight, they do not offer a comprehensive understanding of performance– especially for views that are far away from the capture trajectory. Qualitative evaluation with an interactive viewer addresses these limitations and allows developers to gain a more holistic understanding of the model performance.

3.3. Easy workflow for user-captured data

While we offer support for synthetic datasets (Blender [22], D-NeRF [31]), in Nerfstudio we focus primarily on "real world data" — images or videos from a physical phone or camera. To this end, we present a new Nerfstudio Dataset (shown in Fig. 3) composed of real-world scenes casually captured with mobile phones and a mirrorless camera. Our motivation is to provide a framework compatible with a diverse array of applications which requires supporting real data. For instance, a few use cases for Nerfstudio outside of research include VFX, gaming, and non-technical film-makers who create 3D and video art. To support this wide range of expertise in NeRFs, we ensure our codebase is easily installable and deployable.

4. Core components

The proposed framework of Nerfstudio, illustrated in Fig. 2, is based on the conceptual grouping of NeRF methods into a series of basic building blocks. Nerfstudio takes a set of posed images and optimizes for a 3D representation of the scene, which is defined by radiance (color), density (structure), and possibly other quantities (semantics, normals, features, etc.). We ingest these inputs into the framework which comprises of a DataManager and a Model, where the DataManager is responsible for (1) parsing image formats via a DataParser and (2) generating rays as RayBundles. These rays are then passed into a Model, which will query Fields and render quantities. Finally, the whole Pipeline is supervised end-to-end with a loss.

4.1. DataManagers and DataParsers

The first step of the Pipeline is the DataManager which is responsible for turning posed images into RayBundles, which are slices of 3D space that start at a camera origin. Within the DataManager, the DataParser first loads the input images and camera data. The DataParser is designed to be compatible with arbitrary data formats such as COLMAP. Previous research codebases primarily utilize COLMAP with helper scripts [25], however, COLMAP can be challenging to install and use for non-technical users. To make the framework more accessible to a wider range of users, including scientists, artists, photographers, hobbyists and journalists, we have implemented DataParsers for mobile apps (Record3D, Polycam, KIRI Engine) and 3D tools such as Metashape and Reality Capture. Once the images are properly loaded and formatted, the DataManager iterates through the data, generating RayBundles and ground truth supervision. It can also optimize camera poses during training.

4.2. RayBundles, RaySamples, and Frustums

NeRFs operate on regions of 3D space, which can be parametrized in many different ways. We have adopted a more generic representation of 3D space through the use of Frustum for both point-based and volume-based samples. The RayBundles, which are primitives that represent a slice through 3D space, are parameterized with an origin, direction, and other meta-information such as camera indices and time. By specifying the interval bin spacing, the RayBundles generate RaySamples, which represent sampled chunks of 3D space along each ray. These chunks, represented as Frustums, can be encoded either as point samples [22] or as Gaussians with mean and covariance [3], which have been shown to help with anti-aliasing. This abstraction allows for flexibility in representation, as the user can decide which representation to use with a simple function call. A visualization of this abstraction can be found in Fig. 4.

4.3. Models and Fields

The RayBundles are sent to Models as input, which samples them into RaySamples. The RaySamples are consumed by Fields to turn regions of space (i.e., Frustums) into quantities such as color or density. The Nerfstudio framework contains various implementations of models and fields. We've implemented various samplers and renderers including fourier features, hash encodings [25], spherical harmonics, and matrix decompositions [6]. Field components include fused MLPs, voxel grids, and normal MLPs,

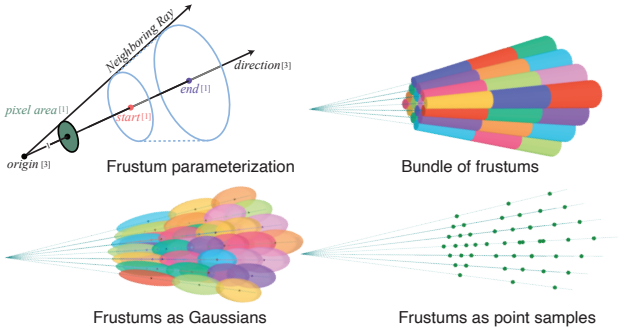

Figure 4. **Sample representations.** (Top) We define a frustum as a cone with a start and end. This region of space can be converted into Gaussians (bottom left) or point samples (bottom right) depending on the field input format.

activation functions, encodings, spatial distortions, and temporal distortions.

4.4. Real-time web viewer

We draw inspiration from the real-time viewer presented in Instant NGP [25], which facilitates real-time rendering during training. However, the viewer in Instant NGP is designed to work on local compute, which can be cumbersome to setup in remote settings. To address this issue, we have developed a ReactJS-based web viewer packaged as a publicly hosted website at https://viewer.nerf.studio.

The viewer is designed to be accessible to a wide range of users, including those utilizing both local and remote GPUs. The process of utilizing remote compute is streamlined, requiring only the forwarding of a port locally via SSH. Once training begins, the web interface renders the NeRF in real-time as training progresses (See Fig. 10). Users can pan, zoom and rotate around the scene as the optimization runs or while evaluating a trained model. The design of the viewer is illustrated in Fig. 5.

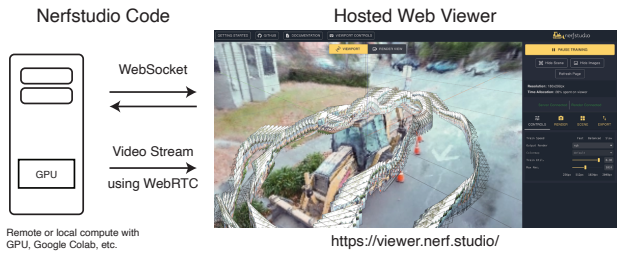

Figure 5. **Web viewer design.** A machine with a GPU (left) starts a NeRF training session. When a user navigates to to the hosted web viewer (right), the viewer client will establish WebSocket and WebRTC connections with the training session.

4.4.1 Implementation

Real-time training visualization utilizes WebSockets and WebRTC to establish a connection between the NeRF training session and the web client. This approach eliminates the need to install local screens and other GUI software. Upon opening the web viewer, a WebSocket connection is established with the training session, which subsequently populates the scene with training images as illustrated in Fig. 5 (right). The web viewer continuously streams the viewport camera pose to the training session during the training process. The training session utilizes this camera pose to render images and transmits them via a WebRTC video stream. Additionally, the viewer camera controls and UI are implemented using ThreeJS, allowing us to overlay 3D assets such as images, splines, and cropping boxes in front of the NeRF renderings. For instance, the viewer displays training images at their capture locations, letting users intuitively compare performance at seen and novel viewpoints.

4.4.2 Viewer features

Our viewer is compatible with different models of varying rendering speeds. We accomplish this by balancing the computation of training and viewer rendering on a single GPU. Similar to Instant-NGP [25], we adjust the rendering resolution based on the speed of the camera movement. When the camera moves quickly, the rendering resolution will be smaller to maintain a frame rate and prevent lag in the user experience. We can also reduce the time spent on training and allocate more resources for rendering in the viewer. Some of the features of our viewer include:

- Switching between various model outputs (e.g., rgb, depth, normals, semantics).
- Creating custom camera paths composed of keyframes with position and focal length interpolation (Fig. 6).
- Visualizing the captured training images in 3D.
- Crop and export options for point clouds and meshes.
- Mouse and keyboard controls to easily navigate in the scene.

The viewer played an instrumental role in providing qualitative assessments that informed design choices in our default method Nerfacto. Other codebases have integrated our viewer into their own codebases, including Arc-Nerf [55] and SDFStudio [56].

4.5. Geometry export

Many creators and artists have workflows that require exporting to point clouds or meshes for further processing and incorporation in downstream tools such as game engines.

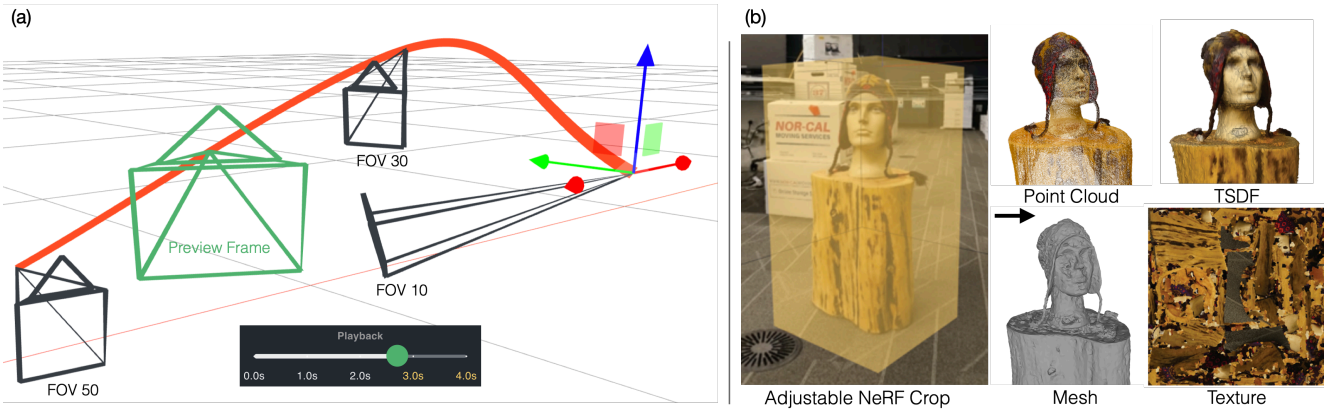

Figure 6. **Exporting videos and geometry.** We make exporting videos (a) and geometry (b) easy with real-data captures. The left side shows the interactive camera trajectory editor, which allows animatable poses, FOVs, and speed, to eventually render videos of NeRF's outputs. On the right we show the cropping interface in the viewer and resulting export formats including point clouds, TSDFs, and textured meshes.

Hence, our framework accommodates various export methods and facilitates the easy addition of new export methods. Fig. 6b illustrates our export interface, as well as some of the supported formats, including point clouds, a truncated signed distance function (TSDF) to mesh, and Poisson surface reconstruction [14]. We apply texture to the mesh by densely sampling the texture image, utilizing barycentric interpolation to determine corresponding 3D point locations, and rendering short rays to obtain RGB values.

5. Nerfacto Method

We leverage our modular design to integrate ideas from multiple research papers into our default and recommended method, Nerfacto. This method is heavily influenced by the structure of MipNeRF-360 [4], but certain parts of the original design are replaced to improve performance. We reference papers such as NeRF-- [48], Instant-NGP [25], NeRF-W [19], and Ref-NeRF [44] in Nerfacto. Fig. 7 illustrates how these papers are used.

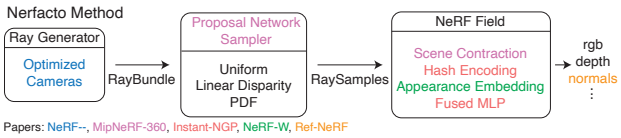

Figure 7. **Nerfacto method.** Diagram of the Nerfacto method. It combines features from many papers (bottom left). The method will evolve over time as new papers and features are added to the Nerfstudio codebase.

5.1. Ray generation and sampling

The Nerfacto method first optimizes camera views using an optimized SE(3) transformation [17, 43, 48]. These camera views are then used to generate RayBundles. To improve the efficiency and effectiveness of the sampling process, we employ a piece-wise sampler. This sampler samples uniformly up to a fixed distance from the camera, followed by samples that are distributed such that the step size increases with each sample. This allows efficient sampling of distant objects while still maintaining a dense set of samples for nearby objects. These samples are then fed into a proposal network sampler, proposed in the MipNeRF-360 method [4]. The proposal sampler consolidates the sample locations into regions of the scene that contribute most to the final render, typically the first surface intersection. This importance sampling greatly improves reconstruction quality. Furthermore, we use a small fused MLP with a hash encoding [25] for the scene's density function as it has been found to have sufficient accuracy and is computationally efficient. To further reduce the number of samples along rays, the proposal network sampler can contain multiple density fields. These density fields iteratively reduce the number of samples. Empirically, using two density fields works well. In our base Nerfacto configuration, we generate 256 samples from the piece-wise sampler, which gets resampled into 96 samples in the first iteration of the proposal sampler followed by 48 samples in the second.

5.2. Scene contraction and NeRF field

Many real-world scenes are unbounded, meaning they could extend indefinitely. This poses a challenge for processing as input samples could have position values that vary across many scales of magnitude. To overcome this issue, we utilize *scene contraction*, which compresses the infinite space into a fixed-size bounding box. Our method of contraction is based on the one proposed in MipNeRF-360 [4], but we use L^∞ norm contraction instead of L^2

norm, which contracts to a cube rather than a sphere. The cube better aligns with voxel-based hash encodings. Fig. 8 illustrates how L^∞ contraction maps samples into the range with minimum values of -2,-2,-2 and maximum values of 2,2,2. These samples can then be used with the hash encoding introduced by Instant-NGP and is available via the tiny-cuda-nn [24] Python bindings.

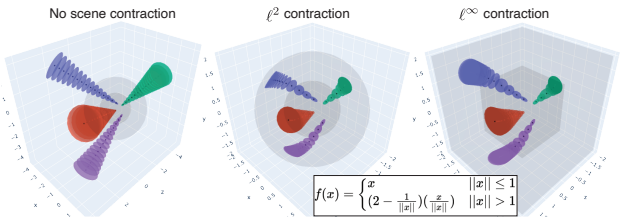

Figure 8. **Scene contraction.** Here we show cameras contained in an inner sphere with Gaussian samples along rays. Scene contraction warps the unbounded samples into bounded space before querying a NeRF field. We use L^∞ contraction rather than MipNeRF-360's L^2 contraction to better accommodate the geometry/capacity of the hash grid.

Nerfacto's field incorporates per-image appearance embeddings to account for differences in exposure among training cameras [19]. Additionally, we use techniques from Ref-NeRF [44] to compute and predict normals. Nerfacto is implemented using PyTorch, which allows for easy customization and eliminates the need for complex and custom CUDA code. We will incorporate new papers into Nerfacto as the field progresses.

6. Nerfstudio Dataset

Our "Nerfstudio Dataset" includes 10 in-the-wild captures obtained using either a mobile phone or a mirror-less camera with a fisheye lens. We processed the data using either COLMAP or the Polycam app to obtain camera poses and intrinsic parameters. Our goal is to provide researchers with more 360 real-world captures that are not limited to forward-facing scenes [21]. Our dataset is similar to MipNeRF-360 [4] but does not focus on a central object and includes captures with varying degrees quality. We have used this dataset to select the default settings for our proposed NeRF-based method, Nerfacto, and we encourage other researchers to similarly employ real-world data in the development and evaluation of NeRF methods.

7. Experiments

We benchmark Nerfacto against a state-of-the-art method MipNeRF-360 and emphasize the modularity of our repository by conducting ablation studies. Furthermore, we highlight the limitations of commonly used evaluation metrics such as PSNR, SSIM, and LPIPS when applied to subsampled evaluation images.

7.1. Mip-NeRF 360 dataset comparison

Here we compare Nerfacto with numbers reported in the MipNeRF-360 [4] paper. We evaluate on their 7 publicly available scenes. We train our method for up to 30K iterations (30 minutes) on an NVIDIA RTX A5000, but we also report results at 5K iterations (5 minutes).

Evaluation protocol. The evaluation protocol followed is similar to that of MipNeRF360, but we process their data using our COLMAP pipeline to recover poses. The original images were downsampled by a factor of 4x. We used 7/8 of the images for training and the remaining 1/8 images were evenly spaced and used for evaluation. Note that this protocol does not include camera pose optimization as it is not an option implemented in MipNeRF360.

Findings. Table 1 presents the averages of the results across the 7 captures in the MipNeRF-360 dataset. The complete table can be found in the appendix. In as little as 5K iterations (∼5 minutes), our Nerfacto method achieves reasonable quality in contrast to MipNeRF-360 which takes several hours on a TPU with 32 cores. Training for up to 30K (∼30 minutes) iterations further improves quality. While Nerfacto falls short of metric results obtained by MipNeRF-360, we prioritize efficiency and general usability over optimizing quantitative metrics on this particular benchmark. Fig. 10 shows qualitative results on the "garden" scene in our viewer after only a few minutes.

Method	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
MipNeRF-360	29.23	0.844	0.207
Nerfacto (ours)	26.75 / 25.38	0.748 / 0.688	0.307 / 0.390

Table 1. **Average metrics on the MipNeRF360 dataset.** Our methods are without pose & appearance optimization. MipNeRF-360 takes several hours to train. Our metrics reported as { after 30K iterations (∼30min) / after 5k iterations (∼5min) }.

It is worth emphasizing that our Nerfacto method is optimized for qualitative novel-view quality by using the web viewer, rather than solely relying on common metrics. For further illustration, we refer the reader to the appendix where we provide rendered videos from our Nerfacto method.

7.2. Nerfacto component ablations

Given the modularity of our codebase, we can easily conduct ablation studies on our method Nerfacto, a unified approach that combines important components from various papers to achieve a fast, high-quality method. We experiment with disabling the pose optimization, appearance embeddings, scene contraction, and variations of the proposal networks, and more. The modularity of our codebase allows

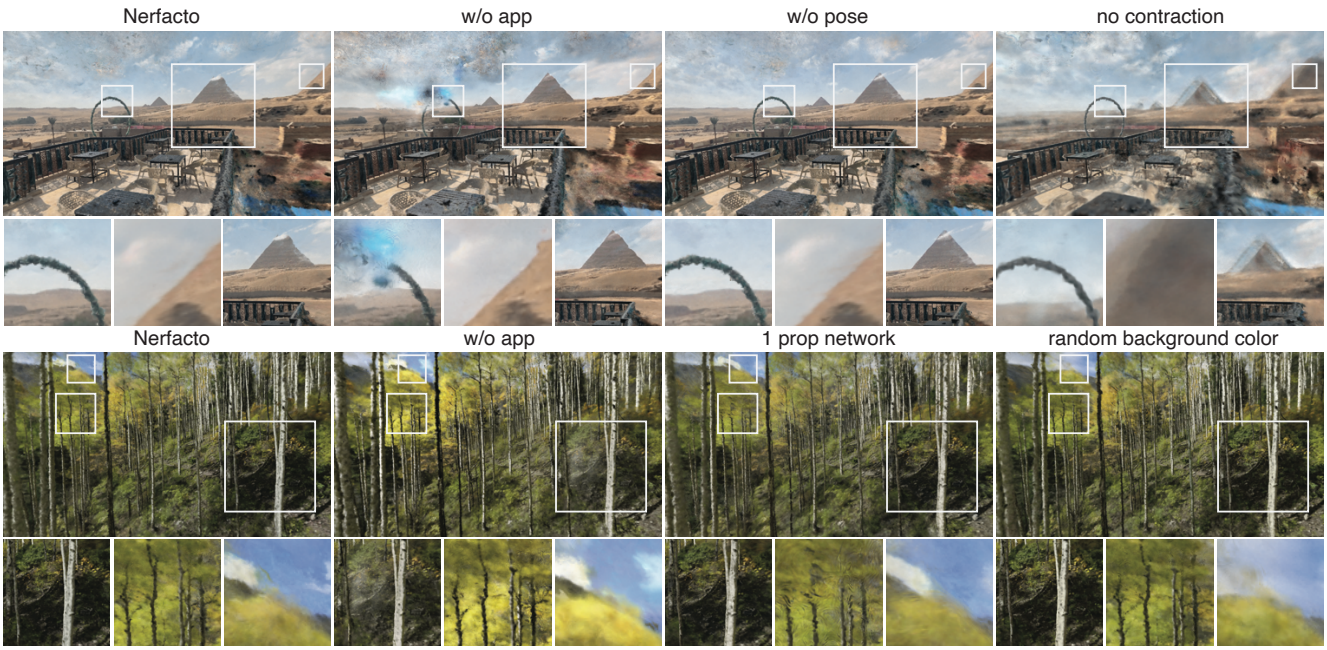

Figure 9. **Nerfstudio ablation qualitative examples.** Here we show renderings from different Nerfacto ablation variants. (Top) is the "Egypt" capture and (bottom) is the "aspen" capture from the Nerfstudio Dataset. These novel views are far from the training images to get a sense of how well these methods perform qualitatively. We zoom in on crops to highlight differences in the rendered images.

for easy implementation of these modifications through the use of different flags with the command line interface.

Evaluation protocol. In our ablation study, we utilize the Nerfstudio Dataset for evaluation. Due to the complexity of the appearance embeddings and pose optimization modules, we adopt a test-time optimization procedure for the evaluation. Specifically, we employ Adam optimizers to optimize the evaluation camera poses. Once the camera poses are fixed, we randomly select either the left or right side of the evaluation image and optimize the appearance code as done in Martin et al. [19]. Finally, with the optimized camera pose and appearance embedding, we compute PSNR, SSIM, and LPIPS. For these experiments, 10% of the images chosen at equal intervals were used for evaluation. We will release this evaluation protocol so future work can run similar experiments.

Findings. Table 2 presents the average results of our ablation studies. The complete table for all 10 scenes can be found in the appendix. This study highlights the challenge in extracting meaningful insights from quantitative metrics alone (Table 2), due to the fact that held-out evaluation images are close to the training images. For instance, disabling the appearance embeddings ("w/o app") leads to an improvement in PSNR and SSIM. However, Fig. 9 illustrates that the "w/o app" method results in the production of blurry "floater" artifacts in regions where the training images are located in 3D (bottom row, bottom left crop). Furthermore, ablations such as "1 prop network" result in subtle changes in the metrics but are more evident in visualizations of the novel views. The use of "1 prop network" as opposed to "Nerfacto (default)" with 2 prop networks leads to aliasing artifacts as can be seen around the small tree branches (bottom row, middle crop). While these artifacts are visible to the eye especially in the interactive viewer, such temporal discontinuity caused by aliasing is not captured by the quantitative metrics. Furthermore, scene contraction is necessary to correctly recover far objects (top row, right crop).

Overall, the real-time viewer proves to be useful for viewing out-of-distribution renders. The crops in Fig. 9 aid

Nerfacto method	**PSNR** ↑	**SSIM** ↑	**LPIPS** ↓
Nerfacto (default)	20.99	0.663	0.389
w/o pose	20.93	0.659	0.393
w/o app	22.65	0.672	0.406
w/o pose & app	22.53	0.671	0.411
1 prop network	21.07	0.669	0.396
l2 contraction	20.98	0.664	0.388
shared prop network	20.95	0.661	0.391
random backg. color	21.00	0.663	0.392
no contraction	18.59	0.534	0.506
synthetic on real	20.09	0.542	0.509

Table 2. **Average metrics for ablations on the Nerfstudio Dataset.** We remove and change various components of the Nerfacto method and report { PSNR, SSIM, LPIPS } on the Nerfstudio Dataset.

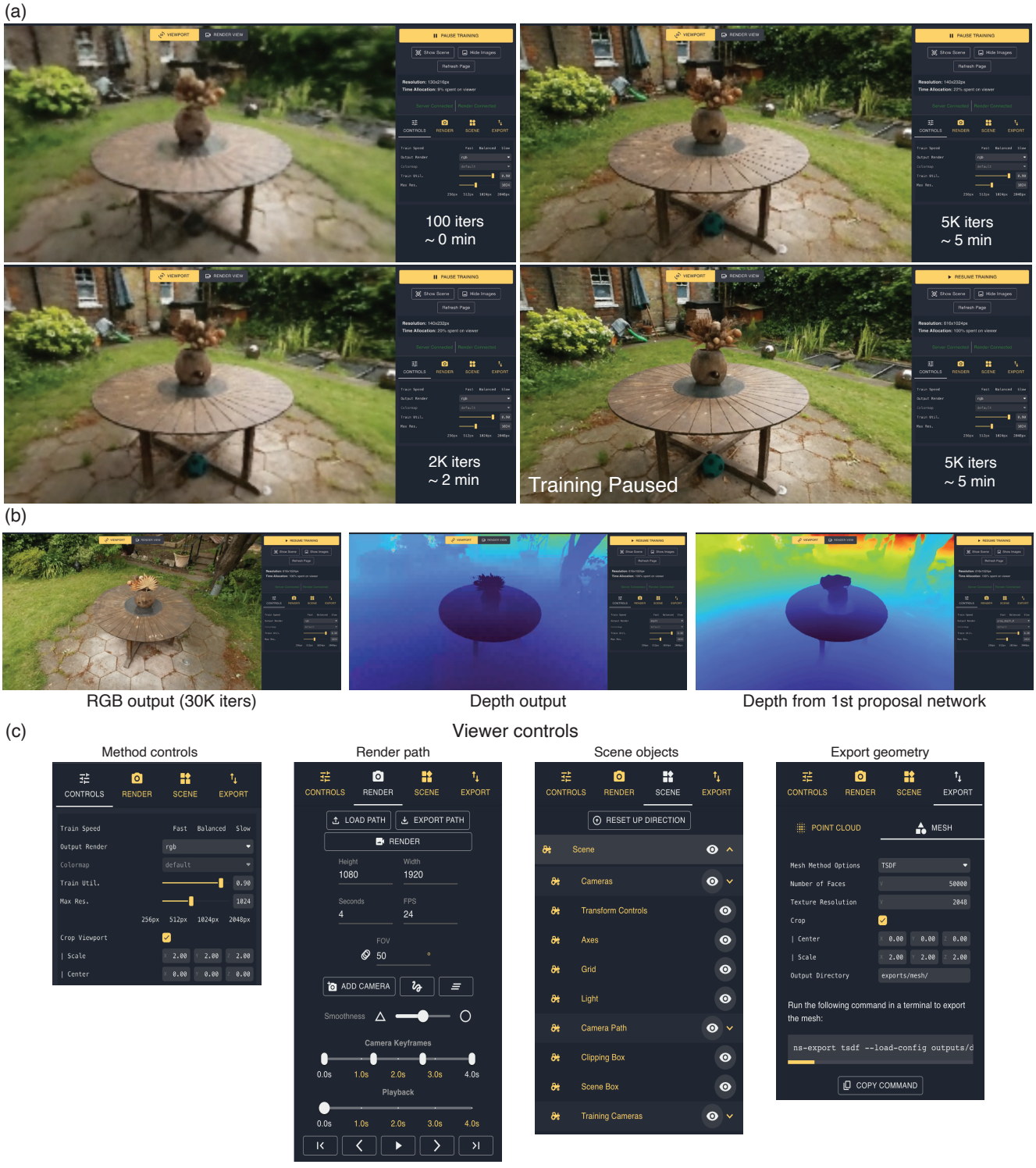

Figure 10. **Real-time viewer use.** (a) Training Nerfacto on the MipNeRF-360 garden scene. Good quality can be achieved after a few minutes. Pausing the training increases the rendered resolution. (b) Visualizing different model outputs with the viewer. (c) Viewer controls and settings available in the viewer.

in illustrating where certain methods excel over others, regardless of the metrics on the evaluation images. Developing more appropriate evaluation metrics is an important avenue for future research.

8. Open-source Contributions

One of the key strengths of our proposed framework is its versatility and ease of use, as demonstrated by our open-source contributions. Our GitHub repository has grown to include over 60 contributors and over 3K stars, reflecting a strong and active community. Additionally, two new libraries, SDFStudio [56] and ArcNerf [55], have been built on top of our framework. Since the release of Nerfstudio in October 2022, our contributors have enhanced and expanded Nerfstudio by addressing various GitHub issues and feature requests including improved camera paths, colab support, additional camera models, reconstruction of dynamic objects. In the future, we plan on supporting 3D generative pipelines, NeRF compositing, and more.

9. Conclusion and Future Work

We draw upon existing techniques and propose a framework that supports a more modularized approach to NeRF development, allows for real-time visualization, and is readily usable with real-world data. We emphasize the importance of utilizing the interactive real-time viewer during training to compensate for imperfect quantitative metrics in model design decisions. We hope the consolidation brought about by this new framework will facilitate the development of NeRF-based methods, thereby accelerating advances in the neural rendering community. Future research directions include the development of more appropriate evaluation metrics and integration of the framework with other fields such as computer vision, computer graphics, and machine learning.

Acknowledgements

We want to thank the many open-source contributors who have helped create Nerfstudio, including Cyrus Vachha and Rohan Mathur from UC Berkeley and the following Github users: machenmusik, kevinddchen (Kevin Chen), dozeri83, nikmo33 (Nikhil Mohan), lsongx (Liangchen Song), zmurez (Zak Murez), JulianKnodt (Julian Knodt), katrinbschmid (Katrin Schmid), mpmisko (Michal Pandy), RandomPrototypes, ManuConcepBrito , Zunhammer (Nicolas Zunhammer), nlml (Liam Schoneveld), jkulhanek (Jonáš Kulhánek), mackopes (Martin Piala), cnsumner, devernay (Frédéric Devernay), matsuren (Ren Komatsu), Mason-McGough (Mason McGough), hturki, decrispell (Daniel Crispell), dkorolov (Dmytro Korolov), gilureta (Francisca T. Gil Ureta).

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[2] Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager. Vision-only robot navigation in a neural radiance world. *CoRR*, abs/2110.00168, 2021.

[3] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

[4] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.

[5] Arunkumar Byravan, Jan Humplik, Leonard Hasenclever, Arthur Brussee, Francesco Nori, Tuomas Haarnoja, Ben Moran, Steven Bohez, Fereshteh Sadeghi, Bojan Vujatovic, and Nicolas Heess. Nerf2real: Sim2real transfer of vision-guided bipedal motion skills using neural radiance fields, 2022.

[6] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. *arXiv preprint arXiv:2203.09517*, 2022.

[7] Robert L Cook, Loren Carpenter, and Edwin Catmull. The reyes image rendering architecture. *ACM SIGGRAPH Computer Graphics*, 21(4):95–102, 1987.

[8] Danny Driess, Zhiao Huang, Yunzhu Li, Russ Tedrake, and Marc Toussaint. Learning multi-object dynamics with compositional neural radiance fields. *arXiv preprint arXiv:2202.11855*, 2022.

[9] D. Ponsa E. Rublee E. Riba, D. Mishkin and G. Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2020.

[10] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

[11] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022. https://mitsuba-renderer.org.

[12] Yoonwoo Jeong, Seungjoo Shin, and Kibaek Park. Nerf-factory: An awesome pytorch nerf collection, 2022.

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast

feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[14] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[15] Justin Kerr, Letian Fu, Huang Huang, Yahav Avigal, Matthew Tancik, Jeffrey Ichnowski, Angjoo Kanazawa, and Ken Goldberg. Evo-nerf: Evolving nerf for sequential robot grasping of transparent objects. In *6th Annual Conference on Robot Learning*, 2022.

[16] Ruilong Li, Matthew Tancik, and Angjoo Kanazawa. Nerfacc: A general nerf accleration toolbox. *arXiv preprint arXiv:2210.04847*, 2022.

[17] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5741–5751, 2021.

[18] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020.

[19] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021.

[20] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P Srinivasan, and Jonathan T Barron. Nerf in the dark: High dynamic range view synthesis from noisy raw images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16190–16199, 2022.

[21] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019.

[22] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[23] Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, Peter Hedman, Ricardo Martin-Brualla, and Jonathan T. Barron. MultiNeRF: A Code Release for Mip-NeRF 360, Ref-NeRF, and RawNeRF, 2022.

[24] Thomas Müller. tiny-cuda-nn, 4 2021.

[25] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022.

[26] Krishna Murthy. nerf-pytorch: A pytorch re-implementation, 2020.

[27] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5589–5599, 2021.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[29] Georgios Pavlakos, Ethan Weber, Matthew Tancik, and Angjoo Kanazawa. The one where they reconstructed 3d humans and environments in tv shows. In *European Conference on Computer Vision*, pages 732–749. Springer, 2022.

[30] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

[31] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021.

[32] Chen Quei-An. Nerf_pl: a pytorchlightning implementation of nerf, 2020.

[33] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices*, 48(6):519–530, 2013.

[34] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020.

[35] Darius Rückert, Yuanhao Wang, Rui Li, Ramzi Idoughi, and Wolfgang Heidrich. Neat: Neural adaptive tomography. *arXiv preprint arXiv:2202.02171*, 2022.

[36] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[37] Johannes Lutz Schönberger, True Price, Torsten Sattler, Jan-Michael Frahm, and Marc Pollefeys. A vote-and-verify strategy for fast spatial verification in image retrieval. In *Asian Conference on Computer Vision (ACCV)*, 2016.

[38] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[39] Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann. Neural descriptor fields: Se (3)-equivariant object representations for manipulation. *arXiv preprint arXiv:2112.05124*, 2021.

[40] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM siggraph 2006 papers*, pages 835–846. 2006.

[41] 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*, pages 5459–5469, 2022.

[42] Towaki Takikawa, Or Perel, Clement Fuji Tsang, Charles Loop, Joey Litalien, Jonathan Tremblay, Sanja Fidler, and

Maria Shugrina. Kaolin wisp: A pytorch library and engine for neural fields research. https://github.com/NVIDIAGameWorks/kaolin-wisp, 2022.

[43] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-NeRF: Scalable large scene neural view synthesis. *arXiv*, 2022.

[44] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-nerf: Structured view-dependent appearance for neural radiance fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5481–5490. IEEE, 2022.

[45] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3835–3844, 2022.

[46] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021.

[47] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2021.

[48] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf–: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021.

[49] Katherine Watson, Alexandre Devaux, Niko Koppel, A.J. Chavar, and Peter Whidden. Creating workflows for nerf portraiture, 2022.

[50] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, volume 41, pages 641–676. Wiley Online Library, 2022.

[51] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021.

[52] Lin Yen-Chen. Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/, 2020.

[53] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021.

[54] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021.

[55] Yan-Pei Cao Yue Luo. Arcnerf: Nerf-based object/scene rendering and extraction framework, 2022.

[56] Bozidar Antic Songyou Peng Apratim Bhattacharyya Michael Niemeyer Siyu Tang Torsten Sattler Zehao Yu, Anpei Chen and Andreas Geiger. Sdfstudio: A unified framework for surface reconstruction, 2022.

[57] Kai Zhang, Nick Kolkin, Sai Bi, Fujun Luan, Zexiang Xu, Eli Shechtman, and Noah Snavely. Arf: Artistic radiance fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXI*, pages 717–733. Springer, 2022.

[58] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew J Davison. In-place scene labelling and understanding with implicit scene representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15838–15847, 2021.

[59] Jun-Yan Zhu, Yong Jae Lee, and Alexei A Efros. Averageexplorer: Interactive exploration and alignment of visual data collections. *ACM Transactions on Graphics (TOG)*, 33(4):1–11, 2014.

[60] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R. Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

A. Appendix

This appendix contains the full tables for both the MipNeRF-360 [4] comparison (Table 3) and the Nerfacto ablations (Table 4). Please note that in addition to Nerfacto, we've implemented our own variants of Instant NGP [25], MipNeRF [3], Semantic NeRF [58], the original NeRF [22], TensoRF [6], and D-NeRF [31].

	PSNR ↑ for Mip-NeRF 360 Dataset						
Capture name	**bicycle**	**garden**	**stump**	**room**	**counter**	**kitchen**	**bonsai**
MipNeRF-360	24.37	26.98	26.4	31.63	29.55	32.23	33.46
Nerfacto (ours) w/o pose & app	23.44 / 22.36	25.58 / 24.05	19.02 / 18.94	30.74 / 29.36	27.09 / 25.92	30.09 / 28.17	31.31 / 28.98
	SSIM ↑ for Mip-NeRF 360 Dataset						
Capture name	**bicycle**	**garden**	**stump**	**room**	**counter**	**kitchen**	**bonsai**
MipNeRF-360	0.685	0.813	0.744	0.913	0.894	0.920	0.941
Nerfacto (ours) w/o pose & app	0.563 / 0.474	0.732 / 0.617	0.387 / 0.364	0.901 / 0.866	0.832 / 0.776	0.895 / 0.838	0.927 / 0.880
	LPIPS ↓ for Mip-NeRF 360 Dataset						
Capture name	**bicycle**	**garden**	**stump**	**room**	**counter**	**kitchen**	**bonsai**
MipNeRF-360	0.301	0.170	0.261	0.211	0.204	0.127	0.176
Nerfacto (ours) w/o pose & app	0.457 / 0.551	0.279 / 0.385	0.630 / 0.669	0.218 / 0.302	0.262 / 0.346	0.149 / 0.223	0.155 / 0.252

Table 3. **PSNR, SSIM, and LPIPS on Mip-NeRF 360 Dataset.** Metrics reported as { after 30K iterations (∼30min) / after 5k iterations (∼5min) }.

	PSNR ↑ / SSIM ↑ / LPIPS ↓ for Nerfstudio Dataset				
Capture name	**Egypt**	**person**	**kitchen**	**plane**	**dozer**
Capture device	phone	phone	phone	phone	fisheye
Camera est. method	colmap	colmap	polycam	colmap	colmap
nerfacto	21.67 / 0.689 / 0.375	25.17 / 0.692 / 0.320	20.55 / 0.807 / 0.389	22.11 / 0.649 / 0.419	20.20 / 0.743 / 0.391
w/o pose	21.56 / 0.689 / 0.374	25.52 / 0.716 / 0.323	20.47 / 0.798 / 0.402	21.83 / 0.645 / 0.421	20.38 / 0.747 / 0.390
w/o app	22.38 / 0.689 / 0.396	27.74 / 0.744 / 0.316	23.80 / 0.819 / 0.397	22.44 / 0.641 / 0.440	23.58 / 0.750 / 0.393
w/o pose & app	22.31 / 0.692 / 0.399	27.76 / 0.749 / 0.328	23.39 / 0.808 / 0.409	22.22 / 0.642 / 0.443	23.38 / 0.748 / 0.394
1 prop network	21.65 / 0.682 / 0.387	25.80 / 0.730 / 0.324	20.57 / 0.805 / 0.394	22.08 / 0.648 / 0.422	20.00 / 0.738 / 0.403
l2 contraction	21.71 / 0.691 / 0.375	25.05 / 0.690 / 0.322	20.62 / 0.808 / 0.386	22.07 / 0.649 / 0.416	20.47 / 0.745 / 0.386
shared prop network	21.64 / 0.687 / 0.382	25.11 / 0.691 / 0.322	20.56 / 0.805 / 0.392	22.11 / 0.650 / 0.416	20.46 / 0.746 / 0.385
random backg. color	21.56 / 0.678 / 0.389	25.54 / 0.709 / 0.328	20.64 / 0.809 / 0.390	21.95 / 0.642 / 0.423	20.29 / 0.746 / 0.387
no contraction	17.53 / 0.467 / 0.585	21.64 / 0.515 / 0.511	19.12 / 0.724 / 0.440	17.78 / 0.492 / 0.582	18.97 / 0.659 / 0.433
synthetic on real	18.07 / 0.471 / 0.574	23.68 / 0.555 / 0.494	21.21 / 0.727 / 0.453	18.37 / 0.511 / 0.553	21.17 / 0.657 / 0.441
Capture name	**floating tree**	**aspen**	**stump**	**sculpture**	**Giannini Hall**
Capture device	fisheye	fisheye	fisheye	fisheye	fisheye
Camera est. method	colmap	colmap	colmap	colmap	colmap
nerfacto	20.03 / 0.740 / 0.312	16.95 / 0.347 / 0.560	22.04 / 0.698 / 0.295	21.88 / 0.676 / 0.368	19.26 / 0.593 / 0.462
w/o pose	19.80 / 0.721 / 0.321	16.96 / 0.345 / 0.560	21.65 / 0.667 / 0.305	21.82 / 0.670 / 0.368	19.28 / 0.591 / 0.461
w/o app	21.84 / 0.727 / 0.329	16.92 / 0.333 / 0.567	25.11 / 0.777 / 0.308	22.16 / 0.657 / 0.422	20.54 / 0.587 / 0.495
w/o pose & app	21.77 / 0.719 / 0.341	16.91 / 0.334 / 0.567	24.87 / 0.778 / 0.307	22.08 / 0.650 / 0.428	20.63 / 0.587 / 0.494
1 prop network	19.96 / 0.736 / 0.318	16.79 / 0.325 / 0.587	22.70 / 0.751 / 0.283	21.99 / 0.685 / 0.365	19.20 / 0.585 / 0.475
l2 contraction	19.81 / 0.741 / 0.311	16.89 / 0.346 / 0.562	22.16 / 0.708 / 0.296	21.84 / 0.673 / 0.369	19.21 / 0.592 / 0.457
shared prop network	19.91 / 0.734 / 0.315	16.95 / 0.344 / 0.561	21.73 / 0.692 / 0.301	21.75 / 0.670 / 0.374	19.25 / 0.590 / 0.462
random backg. color	19.85 / 0.729 / 0.316	16.95 / 0.343 / 0.561	22.20 / 0.711 / 0.294	21.84 / 0.677 / 0.370	19.23 / 0.590 / 0.457
no contraction	18.85 / 0.588 / 0.410	14.13 / 0.170 / 0.717	20.27 / 0.630 / 0.392	20.71 / 0.619 / 0.436	16.92 / 0.472 / 0.557
synthetic on real	20.64 / 0.565 / 0.418	14.76 / 0.186 / 0.690	23.56 / 0.675 / 0.400	21.71 / 0.609 / 0.485	17.74 / 0.468 / 0.579

Table 4. **Ablations of Nerfacto method on Nerfstudio Dataset.** We remove and change various components of the Nerfacto method and report { PSNR, SSIM, LPIPS } on the Nerfstudio Dataset. "synthetic on real" is our synthetic settings adjusted to work on a real-world capture (i.e., a bounded scene and no scene contraction).