Photorealistic Reconstruction from First Principles

Sara Fridovich-Keil

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-63
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-63.html

May 3, 2023

Photorealistic Reconstruction from First Principles

by

Sara Fridovich-Keil

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering—Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Benjamin Recht, Chair
Professor Laura Waller
Assistant Professor Angjoo Kanazawa
Doctor Rebecca Roelofs

Spring 2023

Photorealistic Reconstruction from First Principles

Abstract

Photorealistic Reconstruction from First Principles

by

Sara Fridovich-Keil

Doctor of Philosophy in Engineering—Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Benjamin Recht, Chair

In computational imaging, inverse problems describe the general process of turning measurements into images using algorithms: images from sound waves in sonar, spin orientations in magnetic resonance imaging, or X-ray absorption in computed tomography. Today, the two dominant algorithmic approaches for solving inverse problems are compressed sensing and deep learning. Compressed sensing leverages convex optimization and comes with strong theoretical guarantees of correct reconstruction, but requires linear measurements and substantial processor memory, both of which limit its applicability to many imaging modalities. In contrast, deep learning methods leverage nonconvex optimization and neural networks, allowing them to use nonlinear measurements and limited memory. However, they can be unreliable, and are difficult to inspect, analyze, and predict when they will produce correct reconstructions.

In this dissertation, we focus on an inverse problem central to computer vision and graphics: given calibrated photographs of a scene, recover the optical density and view-dependent color of every point in the scene. For this problem, we take steps to bridge the best aspects of compressed sensing and deep learning: (i) combining an explicit, non-neural scene representation with optimization through a nonlinear forward model, (ii) reducing memory requirements through a compressed representation that retains aspects of interpretability, and extends to dynamic scenes, and (iii) presenting a preliminary convergence analysis that suggests faithful reconstruction under our modeling.

To my family.

Contents

Acknowledgments

My deepest thanks to the many mentors, collaborators, staff, friends, and family members who have inspired and supported me throughout my PhD. The work presented in this dissertation was joint with wonderful collaborators – acknowledged at the start of each chapter – who have been a pleasure to work with throughout my PhD. I've also greatly enjoyed working with and learning from many collaborators on diverse projects not included in this dissertation, but to whom I am no less grateful.

Many thanks to my advisor, Ben Recht, for your guidance. You've helped me to develop my taste in research and to broaden and deepen my technical expertise. You've also connected me with diverse collaborators and allowed me substantial academic freedom to work on varied projects and grow in my independence. You've supported me in both my academic and personal pursuits, including my (now-successful) quest to do an unassisted pull-up. Thank you for a wonderful five years at Berkeley!

I also wish to thank my full dissertation committee, including Laura Waller, Angjoo Kanazawa, and Rebecca Roelofs. Laura, I'm grateful for your support throughout my PhD, and for connecting me with your lab and enabling my exploration into computational imaging. Angjoo, thank you for welcoming me into computer vision, connecting me with collaborators, and teaching me how to make compelling figures and presentations. Becca, many thanks for helping me get started as a first-year PhD student, and for hosting me at Google Brain as an intern and student researcher.

Thank you to Ruzena Bajcsy, my temporary advisor at the start of my PhD, for being so generous with your time and mentorship as I settled into graduate school. Thanks also to Ludwig Schmidt for your unofficial advising and mentoring throughout my PhD.

I also wish to thank some fabulous staff members at Berkeley, who have had an outsized positive impact on my PhD experience. Shirley Salanio, thank you for your support at all stages of my PhD and through all manner of logistical difficulties. In the land of Modest Yachts, I have also been fortunate to interact with Naomi Yamasaki and Kostadin Ilov, who make the lab environment and compute systems welcoming and productive.

Thanks also to the wonderful lab-mates and office-mates at Berkeley who've made me want to come to the office every day, and to my friends across the department and at Hillel who have made this campus feel like home.

Finally, my deepest love and gratitude to my family – Mom, Dad, David, and extended family – for your unwavering support, inspiration, and love throughout my life. I can reassure you that I know Atlanta is still "home" even if I've come to feel at home in Berkeley too.

This work is partially supported by the National Science Foundation Graduate Research Fellowship Program.

Chapter 1

Introduction

Throughout history, scientific and medical revolutions have been precipitated by the invention of imaging tools. Observations made using a telescope lent credence to the Renaissance notion of heliocentrism. Microscopes enabled the discovery of cells and bacteria. X-ray crystallography revealed the structure of DNA. Though early imaging methods, like microscopes, telescopes, and medical X-rays, directly produce images, the modern use of computation in imaging enables us to visualize and study a far broader class of structures and processes. By combining computation with otherwise uninterpretable measurements, we can map the ocean floor with sonar, see the the brain in action through functional magnetic resonance imaging, and model the structure of a protein through cryo-electron microscopy.

At the core of every computational imaging method is an inverse problem. These problems are so named because their goal is to computationally invert a physical measurement process, producing meaningful images that visualize phenomena not visible to the naked eye. Inverse problems are often challenging for three reasons. The first is that the measurement process itself may not be truly invertible due to physical limitations. For example, in cryo-electron microscopy, measurements are corrupted by noise whose magnitude can surpass that of the signal itself. In phase retrieval, we wish to recover a complex-valued quantity from measurements of its magnitude only. In X-ray computed tomography, we often intentionally collect fewer measurements than strictly necessary for reconstruction, to limit a patient's exposure to harmful ionizing radiation. In these situations, the measurements may be inconsistent with each other, or consistent with multiple possible reconstructions. To distinguish between these possibilities, we must turn to prior knowledge and assumptions about the structure of interest; which assumptions to choose is often unclear.

The second challenge in many inverse problems is model mismatch, in which our computational model of the measurement process, or *forward model*, is an approximation of a more complex physical process that is only partially understood. In some cases, the measurement process depends on the object of study, making it impossible to model perfectly *a priori*. For example, this is the case in magnetic resonance imaging: a patient's body

creates small inhomogeneities in the magnetic field used to study it. Model mismatch may also result from manufacturing imperfections, such as lens aberrations in a computational microscope or telescope.

The third challenge that arises in certain inverse problems is computational. Even when we do have a physically accurate forward model and sufficient measurements to define a unique, correct reconstruction, we may not be able to compute that reconstruction efficiently. For example, though we may understand the way light propagates, reflecting and refracting when it interacts with different materials, and we may have access to unlimited photographs of a scene, the task of inverse rendering requires computational tradeoffs that limit reconstruction quality. Computational challenges arise when the forward model is highly complex to simulate, or when the forward model is nondifferentiable, such as the discrete bounce of a reflected ray, since computational methods often rely on gradient-based optimization.

In the face of these challenges, two computational paradigms have emerged in the field of inverse problems: compressed sensing and deep learning. Both of these methods rely on optimization to invert the measurement model and reconstruct the object or signal of interest. A typical optimization objective, or loss function, takes the form:

$$\mathcal{L}(\hat{\boldsymbol{x}}) = \|\boldsymbol{y} - f(\hat{\boldsymbol{x}})\|_{\ell_2}^2 \tag{1.0.1}$$

where $\boldsymbol{y} \in \mathbb{R}^m$ is a vector of measurements, f is our forward model, and $\hat{\boldsymbol{x}}$ is our current reconstruction, an approximation of the true signal $\boldsymbol{x} \in \mathbb{R}^n$. Compressed sensing is typically restricted to inverse problems with *linear* forward models f, in which the objective in Eq. (1.0.1) is convex in the optimization parameter $\hat{\boldsymbol{x}}$. In this linear setting, convex optimization is guaranteed to converge to a global minimizer of $\mathcal{L}$. Compressed sensing also gives theoretical guarantees of faithful reconstruction in the absence of complete measurements, *i.e.*, when $m < n$, as long as we can (1) make some sparsity assumptions on the true $\boldsymbol{x}$ (even if $\boldsymbol{x}$ is sparse under some change of basis) and (2) implement those assumptions through regularization or constraints added to Eq. (1.0.1). Compressed sensing through convex optimization is thus a powerful framework for solving the subset of inverse problems whose forward models are linear, for example when measurements are linear projections (in the case of tomography) or Fourier coefficients (in the case of magnetic resonance imaging). However, compressed sensing offers little practical or theoretical guidance on nonlinear inverse problems, such as the task of photorealistic reconstruction we consider in this dissertation.

Deep learning has emerged more recently as a powerful toolbox for solving these nonlinear inverse problems. In this paradigm, the parameters of the reconstruction $\hat{\boldsymbol{x}}$ may be represented implicitly through the weights of a neural network, and if the forward model f is not well-modeled explicitly it may also be represented implicitly through a neural network. This flexible modeling, combined with nonconvex gradient-based optimization, allows deep learning to tackle nonlinear inverse problems even with model mismatch, often with impressive experimental results. However, the cost of this flexibility is the

absence of theoretical guarantees. Nonconvex optimization may converge to local, rather than global, minimizers of $\mathcal{L}$, and we have little guidance towards appropriate regularizers to constrain our implicitly-modeled reconstructions $\hat{x}$ in the absence of sufficient measurements.

The goal of this dissertation is to bridge the best aspects of each of these computational paradigms, compressed sensing and deep learning, in the context of a particular inverse problem of interest in computer vision and graphics. We focus on the inverse problem defined by photorealistic reconstruction: our measurements are color photographs of a scene taken from known camera poses, and we wish to recover the optical density σ and (possibly view-dependent) color c of every location in the scene. We strive for the interpretable regularization and theoretically-guaranteed reconstruction fidelity of compressed sensing, married to the flexibility of deep learning to efficiently invert a nonlinear forward model.

1.0.1 Overview

This dissertation is organized into three technical chapters, each of which combines aspects of compressed sensing and deep learning to make practical or theoretical progress on the photorealistic reconstruction problem.

In Chapter 2, we introduce a practical scene representation, called "plenoxels" for "plenoptic volume elements," that uses a fully explicit, voxel-based model of the scene parameters $\hat{x}$, and a well-understood total variation regularizer, both common features of compressed sensing, but optimizes using stochastic gradient methods through a nonlinear forward model and its corresponding nonconvex loss $\mathcal{L}$, a common feature of deep learning. We show empirically that plenoxels achieves reconstructions rivaling the quality of deep learning alternatives, while offering substantial improvements in optimization speed and interpretability.

In Chapter 3, we extend the plenoxels framework to a compressed, yet still grid-based, representation of the scene parameters $\hat{x}$, essentially a low-rank approximation of a volume. This representation substantially reduces memory usage, enabling it to model both static (3D) and dynamic (4D) scenes, while retaining aspects of interpretability including well-understood regularizers and spatially-localized parameters.

In Chapter 4, we propose a theoretical model of the photorealistic reconstruction problem. We explore connections between this model and the simpler inverse problem of computed tomography, in which we can process the measurements to create a linear forward model in which compressed sensing results apply. For the nonlinear photorealistic reconstruction problem, we show correct recovery in the population regime, when we have access to unlimited measurements, and we give a sketch for how this analysis may extend to the limited-measurement regime to guide the number of measurements necessary for a given reconstruction quality. This analysis is a first step towards a more general extension of compressed sensing guarantees towards certain nonlinear forward models that arise in practice, promising greater trust in our reconstructions under these models.

We conclude in Chapter 5 with a discussion of exciting directions for future work towards this marriage of compressed sensing and deep learning, for inverse problems across computational imaging.

Chapter 2

Plenoxels: Static Scene Reconstruction from First Principles

This chapter is based on the paper "Plenoxels: Radiance Fields without Neural Networks" [FYT+22], written in collaboration with Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa.

Please see `alexyu.net/plenoxels` for videos and code associated to this chapter.

Many thanks to Utkarsh Singhal and Ren Ng for helpful discussions, and to Hang Gao for reviewing the chapter draft.

2.1 Introduction

A recent body of research has capitalized on implicit, coordinate-based neural networks as the 3D representation to optimize 3D volumes from calibrated 2D image supervision. In particular, Neural Radiance Fields (NeRF) [MST+20] demonstrated photorealistic novel viewpoint synthesis, capturing scene geometry as well as view-dependent effects. This impressive quality, however, requires extensive computation time for both training and rendering, with training lasting more than a day and rendering requiring 30 seconds per frame, on a single GPU. Multiple subsequent papers [YLT+21; RPLG21; RJY+20; LGLCT21; GKJSV21; HSMBD21] reduced this computational cost for rendering, but single GPU training still requires multiple hours, a bottleneck that limits the practical application of photorealistic volumetric reconstruction.

In this paper, we show that we can train a radiance field from scratch, without neural networks, while maintaining NeRF quality and reducing optimization time by two orders of magnitude. We provide a custom CUDA [NVF20] implementation that capitalizes on the model simplicity to achieve substantial speedups. Our typical optimization time on a single Titan RTX GPU is 11 minutes on bounded scenes (compared to roughly 1 day for NeRF, more than a 100× speedup) and 27 minutes on unbounded scenes (compared to roughly 4 days for NeRF++ [ZRSK20], again more than a 100× speedup). Although our implementation is not optimized for fast rendering, we can render novel viewpoints at interactive rates (15 fps). If faster rendering is desired, our optimized Plenoxel model can be converted into a PlenOctree [YLT+21].

Specifically, we propose an explicit volumetric representation, based on a view-dependent sparse voxel grid without any neural networks. Our model can render photorealistic novel viewpoints and be optimized end-to-end from calibrated 2D photographs, using the differentiable rendering loss on training views along with a total variation regularizer. We call our model Plenoxel for plenoptic volume elements, as it consists of a sparse voxel grid in which each voxel stores density and spherical harmonic coefficients, which model view dependence [AB91]. By interpolating these coefficients, Plenoxels achieve a continuous model of the plenoptic function [AB91]: the light at every position and in every direction inside a volume. To achieve high resolution on a single GPU, we prune empty voxels and follow a coarse to fine optimization strategy. Although our core model is a bounded voxel grid, we show that unbounded scenes can be modeled by using normalized device coordinates (for forward-facing scenes) or by surrounding our grid with multisphere images to encode the background (for 360° scenes).

Our method reveals that photorealistic volumetric reconstruction can be approached using standard tools from inverse problems: a data representation, a forward model, a regularization function, and an optimizer. Our method shows that each of these components can be simple and state of the art results can still be achieved. Our experiments suggest the key element of Neural Radiance Fields is not the neural network but the differentiable volumetric renderer.

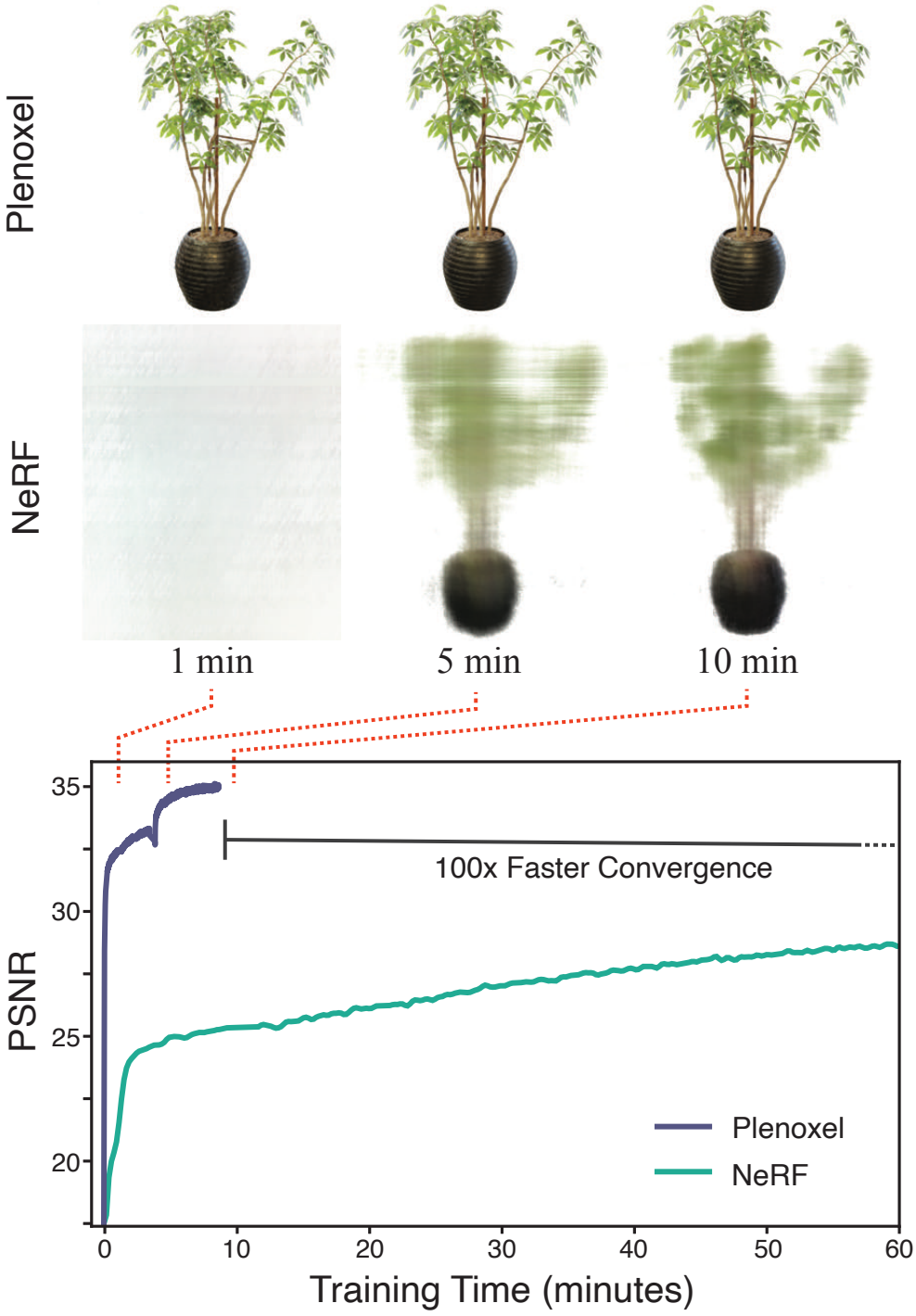


Figure 2.1: **Plenoxel: Plenoptic Volume Elements** for fast optimization of radiance fields. We show that direct optimization of a fully explicit 3D model can match the rendering quality of modern neural based approaches such as NeRF while optimizing over two orders of magnitude faster.

2.2 Related work

Classical Volume Reconstruction. We begin with a brief overview of classical methods for volume reconstruction, focusing on those which find application in our work. The most common classical methods for volume rendering are voxel grids [Lev88; UK88; CXGCS16; SD97; SSKK00; KHM17; TZEM17; STH+19; LSS+19a] and multi-plane images (MPIs) [SG04; PZ17; ZTFFS18; STB+19; MSO+19; WPYS21]. Voxel grids are capable of representing arbitrary topologies but can be memory limited at high resolution. One approach for reducing the memory requirement for voxel grids is to encode hierarchical structure, for instance using octrees [RUG17; HTM17; TDB17; WLGST17] (see [Kno06] for a survey); we use an even simpler sparse array structure. Using these grid-based representations combined with some form of interpolation [UK88] produces a continuous representation that can be arbitrarily resized using standard signal processing methods (see [OS09] for reference). We combine this classical sampling and interpolation paradigm with the forward volume rendering formula introduced by Max [Max95] (based on work from Kajiya and Von Herzen [KV84] and used in NeRF) to directly optimize a 3D grid from indirect 2D observations. We further extend these classical approaches by modeling view dependence, which we accomplish by optimizing spherical harmonic coefficients for each color channel at each voxel. Spherical harmonics are a standard basis for functions over the sphere, and have been used previously to represent view dependence [RH01; SKS02; BJ03; YLT+21; WPYS21].

Neural Volume Reconstruction. Recently, dramatic improvements in neural volume reconstruction have renewed interest in this direction. Neural representations were first used to model occupancy [MONNG19; CZ19; MLL+21] and signed distance to an object's surface [PFSNL19; TLY+21], and perform novel view synthesis from 3D point clouds [ASKUL20; RFS21; WGSJ20; LZ21]. Several papers extended this idea to model a 3D scene using only calibrated 2D image supervision via a differentiable volume rendering formulation [STH+19; SZW20; LSS+19a; MST+20]. NeRF [MST+20] in particular produces impressive results but requires more than a day for full training, and about half an minute to render a full 800×800 image, because every rendered pixel requires evaluating a coordinate-based MLP at hundreds of sample locations along the corresponding ray. Many papers have since extended the capabilities of NeRF, including modeling the background in 360° views [ZRSK20] and incorporating anti-aliasing for multiscale rendering [BMT+21a]. We extend our Plenoxel method to unbounded 360° scenes using a background model inspired by NeRF++ [ZRSK20].

Of these methods, Neural Volumes [LSS+19a] is the most similar to ours in that it uses a voxel grid with interpolation, but optimizes this grid through a convolutional neural network and applies a learned warping function to improve the effective resolution (of a 128^3 grid). We show that the voxel grid can be optimized directly and high resolution can be achieved by pruning and coarse to fine optimization, without any neural networks or warping functions.

Accelerating NeRF. In light of the substantial computational requirements of NeRF for

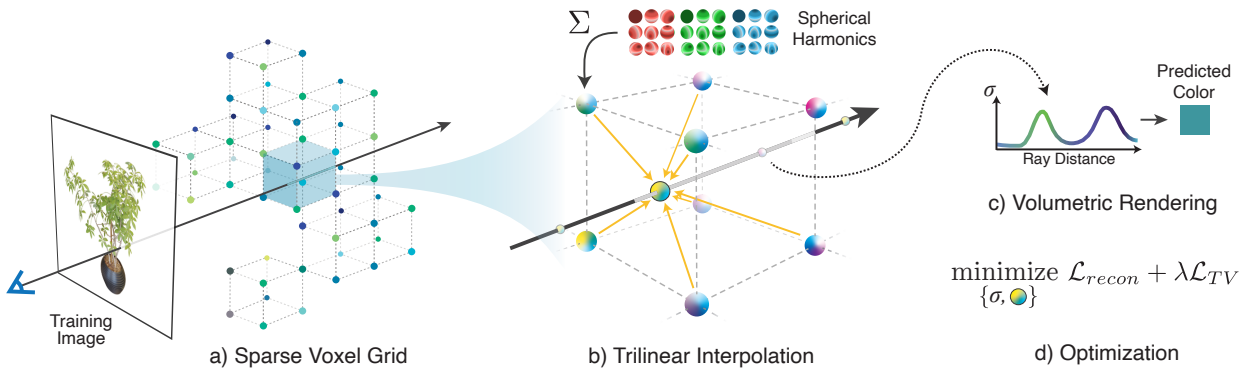


Figure 2.2: **Overview of our sparse Plenoxel model.** Given a set of images of an object or a scene, we optimize a (a) sparse voxel ("Plenoxel") grid with density and spherical harmonic coefficients at each voxel. To render a ray, we (b) compute the color and opacity of each sample point via trilinear interpolation of the neighboring voxel coefficients. We integrate the color and opacity of these samples using (c) differentiable volume rendering, following the recent success of NeRF [MST+20]. The voxel coefficients can then be (d) optimized using the standard MSE reconstruction loss relative to the training images, along with a total variation regularizer.

both training and rendering, many recent papers have proposed methods to improve efficiency, particularly for rendering. Among these methods are some that achieve speedup by subdividing the 3D volume into regions that can be processed more efficiently [RJY+20; LGLCT21]. Other speedup approaches have focused on a range of computational and pre- or post-processing methods to remove bottlenecks in the original NeRF formulation. JAXNeRF [DBS20], a JAX [BFH+18] reimplementation of NeRF, offers a speedup for both training and rendering via parallelization across many GPUs or TPUs. AutoInt [LMW21] restructures the coordinate-based MLP to compute ray integrals exactly, for more than 10× faster rendering with a small loss in quality. Learned Initializations [TMW+21a] employs meta-learning on many scenes to start from a better MLP initialization, for both $>$ 10× faster training and better priors when per-scene data is limited. Other methods [NSP+21; PC21; KJJ+21] achieve speedup by predicting a surface or sampling near the surface, reducing the number of samples necessary for rendering each ray.

Another approach is to pretrain a NeRF (or similar model) and then extract it into a different data structure that can support fast inference [GKJSV21; HSMBD21; RPLG21; YLT+21]. In particular, PlenOctrees [YLT+21] extracts a NeRF variant into a sparse voxel grid in which each voxel represents view-dependent color using spherical harmonic coefficients. Because the extracted PlenOctree can be further optimized, this method can speed up training by roughly 3×, and because it uses an efficient GPU octree implementation without any MLP evaluations, it achieves $>$ 3000× rendering speedup. Our method extends PlenOctrees to perform end-to-end optimization of a sparse voxel representa-

tion with spherical harmonics, offering much faster training (two orders of magnitude speedup compared to NeRF). Our Plenoxel model is a generalization of PlenOctrees to support sparse plenoptic voxel grids of arbitrary resolution (not necessary powers of two) with the ability to perform trilinear interpolation, which is easier to implement with this sparse voxel structure.

2.3 Method

Our model is a sparse voxel grid in which each occupied voxel corner stores a scalar density σ and a vector of spherical harmonic (SH) coefficients for each color channel. From here on we refer to this representation as *Plenoxels*. The density and color at an arbitrary position and viewing direction are determined by trilinearly interpolating the values stored at the neighboring voxels and evaluating the spherical harmonics at the appropriate viewing direction. Given a set of calibrated images, we optimize our model directly using the rendering loss on training rays. Our model is illustrated in Fig. 2.2 and described in detail below.

2.3.1 Volume Rendering

We use the same differentiable model for volume rendering as in NeRF, where the color of a ray is approximated by integrating over samples taken along the ray:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \big(1 - \exp(-\sigma_i \delta_i)\big) \mathbf{c}_i \tag{2.3.1}$$

$$\text{where} \qquad T_i = \exp\left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right) \tag{2.3.2}$$

T_i represents how much light is transmitted through ray $\mathbf{r}$ to sample i, $\big(1 - \exp(-\sigma_i \delta_i)\big)$ denotes how much light is contributed by sample i, σ_i denotes the density of sample i, and $\mathbf{c}_i$ denotes the color of sample i, with distance δ_i to the next sample. Although this formula is not exact (it assumes single-scattering [KV84] and constant values between samples [Max95]), it is differentiable and enables updating the 3D model based on the error of each training ray.

2.3.2 Voxel Grid with Spherical Harmonics

Similar to PlenOctrees [YLT+21], we use a sparse voxel grid for our geometry model. However, for simplicity and ease of implementing trilinear interpolation, we do not use an octree for our data structure. Instead, we store a dense 3D index array with pointers into a separate data array containing values for occupied voxels only. Like PlenOctrees, each

occupied voxel stores a scalar density σ and a vector of spherical harmonic coefficients for each color channel. Spherical harmonics form an orthogonal basis for functions defined over the sphere, with low degree harmonics encoding smooth (more Lambertian) changes in color and higher degree harmonics encoding higher-frequency (more specular) effects. The color of a sample $\mathbf{c}_i$ is simply the sum of these harmonic basis functions for each color channel, weighted by the corresponding optimized coefficients and evaluated at the appropriate viewing direction. We use spherical harmonics of degree 2, which requires 9 coefficients per color channel for a total of 27 harmonic coefficients per voxel. We use degree 2 harmonics because PlenOctrees found that higher order harmonics confer only minimal benefit.

Plenoxel grid uses trilinear interpolation to define a continuous plenoptic function throughout the volume. This is in contrast to PlenOctrees, which assumes that the density and spherical harmonic coefficients remain constant inside each voxel. This difference turns out to be an important factor in successfully optimizing the volume, as we discuss below. All coefficients (for density and spherical harmonics) are optimized directly, without any special initialization or pretraining with a neural network.

2.3.3 Interpolation

The density and color at each sample point along each ray are computed by trilinear interpolation of density and harmonic coefficients stored at the nearest 8 voxels. We find that trilinear interpolation significantly outperforms a simpler nearest neighbor interpolation; see Tab. 2.1. The benefits of interpolation are twofold: interpolation increases the effective resolution by representing sub-voxel variations in color and density, and interpolation produces a continuous function approximation that is critical for successful optimization. Both of these effects are evident in Tab. 2.1: doubling the resolution of a nearest-neighbor-interpolating Plenoxel closes much of the gap between nearest neighbor and trilinear interpolation at a fixed resolution, yet some gap remains due to the difficulty of optimizing a discontinuous model. Indeed, we find that trilinear interpolation is more stable with respect to variations in learning rate compared to nearest neighbor interpolation (we tuned the learning rates separately for each interpolation method in Tab. 2.1, to provide close to the best number possible for each setup).

2.3.4 Coarse to Fine

We achieve high resolution via a coarse-to-fine strategy that begins with a dense grid at lower resolution, optimizes, prunes unnecessary voxels, refines the remaining voxels by subdividing each in half in each dimension, and continues optimizing. For example, in the synthetic case, we begin with 256^3 resolution and upsample to 512^3. We use trilinear interpolation to initialize the grid values after each voxel subdivision step. In fact, we can resize between arbitrary resolutions using trilinear interpolation. Voxel pruning is performed using the method from PlenOctrees [YLT+21], which applies a threshold to the

	PSNR ↑	SSIM ↑	LPIPS ↓
Trilinear, 256^3	30.57	0.950	0.065
Trilinear, 128^3	28.46	0.926	0.100
Nearest Neighbor, 256^3	27.17	0.914	0.119
Nearest Neighbor, 128^3	23.73	0.866	0.176

Table 2.1: **Ablation over interpolation method.** Results are averaged over the 8 NeRF synthetic scenes. We find that trilinear interpolation provides dual benefits of improving effective resolution and improving optimization, such that trilinear interpolation at resolution 128^3 outperforms nearest neighbor interpolation at 256^3.

maximum weight $T_i(1 - \exp(-\sigma_i \delta_i))$ of each voxel over all training rays (or, alternatively, to the density value in each voxel). Due to trilinear interpolation, naively pruning can adversely impact the the color and density near surfaces since values at these points interpolate with the voxels in the immediate exterior. To solve this issue, we perform a dilation operation so that a voxel is only pruned if both itself and its neighbors are deemed unoccupied.

2.3.5 Optimization

We optimize voxel densities and spherical harmonic coefficients with respect to the mean squared error (MSE) over rendered pixel colors, with total variation (TV) regularization [RO94]. Specifically, our base loss function is:

$$\mathcal{L} = \mathcal{L}_{recon} + \lambda_{TV}\, \mathcal{L}_{TV} \tag{2.3.3}$$

Where the MSE reconstruction loss $\mathcal{L}_{recon}$ and the total variation regularizer $\mathcal{L}_{TV}$ are:

$$\mathcal{L}_{recon} = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r} \in \mathcal{R}} \| C(\mathbf{r}) - \hat{C}(\mathbf{r}) \|_2^2$$

$$\mathcal{L}_{TV} = \frac{1}{|\mathcal{V}|} \sum_{\substack{\mathbf{v} \in \mathcal{V} \\ d \in [D]}} \sqrt{\Delta_x^2(\mathbf{v}, d) + \Delta_y^2(\mathbf{v}, d) + \Delta_z^2(\mathbf{v}, d)}$$

with $\Delta_x^2(\mathbf{v}, d)$ shorthand for the squared difference between the dth value in voxel $\mathbf{v} := (i, j, k)$ and the dth value in voxel $(i+1, j, k)$ normalized by the resolution, and analogously for $\Delta_y^2(\mathbf{v}, d)$ and $\Delta_z^2(\mathbf{v}, d)$, where D is the total number of density and spherical harmonic (SH) coefficients stored at each voxel. In practice we use different weights for SH coefficients and σ values. These weights are fixed for each scene type (bounded, forward-facing, and 360°).

For faster iteration, we use a stochastic sample of the rays $\mathcal{R}$ to evaluate the MSE term and a stochastic sample of the voxels $\mathcal{V}$ to evaluate the TV term in each optimization step. We use the same learning rate schedule as JAXNeRF and Mip-NeRF [DBS20; BMT+21a], but tune the initial learning rate separately for density and harmonic coefficients. The learning rate is fixed for all scenes in all datasets in the main experiments.

Directly optimizing voxel coefficients is a challenging problem for several reasons: there are many values to optimize (the problem is high-dimensional), the optimization objective is nonconvex due to the rendering formula, and the objective is poorly conditioned. Poor conditioning is typically best resolved by using a second order optimization algorithm (*e.g.* as recommended in [NW06b]), but this is practically challenging to implement for a high-dimensional optimization problem because the Hessian is too large to easily compute and invert in each step. Instead, we use RMSProp [Hin] to ease the ill-conditioning problem without the full computational complexity of a second-order method.

2.3.6 Unbounded Scenes

With minor modifications, Plenoxels extend to real, unbounded scenes, both forward-facing and 360°. For forward-facing scenes, we use normalized device coordinates, as defined in the original NeRF paper [MST+20].

Background model. For 360° scenes, we augment our sparse voxel grid foreground representation with a multi-sphere image (MSI) background model, which also uses learned voxel colors and densities with trilinear interpolation within and between spheres. Note that this is effectively the same as our foreground model, except the voxels are warped into spheres using the simple equirectangular projection (voxels index over sphere angles θ and ϕ). We place 64 spheres linearly in inverse radius from 1 to ∞ (we pre-scale the inner scene to be approximately contained in the unit sphere). To conserve memory, we store only rgb channels for the colors (only zero-order SH) and store all layers sparsely by using density thresholding as in our main model. This is similar to the background model in NeRF++ [ZRSK20].

2.3.7 Regularization

We illustrate the importance of TV regularization in Fig. 2.3. In addition to TV regularization, which encourages smoothness and is used on all scenes, for certain types of scenes we also use additional regularizers.

On the real, forward-facing and 360° scenes, we use a sparsity prior based on the Cauchy loss from SNeRG [HSMBD21]:

$$\mathcal{L}_s = \lambda_s \sum_{i,k} \log\left(1 + 2\sigma(\mathbf{r}_i(t_k))^2\right) \tag{2.3.4}$$

Full | No SH TV | No σ TV | No TV

Figure 2.3: **Ablation over TV regularization.** Clear artifacts are visible in the forward-facing scenes without TV on both σ and SH coefficients, although PSNR does not always reflect this.

where $\sigma(\mathbf{r}_i(t_k))$ denotes the density of sample k along training ray i. In each minibatch of optimization on forward-facing scenes, we evaluate this loss term at each sample on each active ray. This is also similar to the sparsity loss used in PlenOctrees [YLT+21] and encourages voxels to be empty, which helps to save memory.

On the real, 360° scenes, we also use a beta distribution regularizer on the accumulated foreground transmittance of each ray in each minibatch. This loss term, following Neural Volumes [LSS+19a], promotes a clear foreground-background decomposition by encouraging the foreground to be either fully opaque or empty. This beta loss is:

$$\mathcal{L}_\beta = \lambda_\beta \sum_{\mathbf{r}} \left(\log(T_{FG}(\mathbf{r})) + \log(1 - T_{FG}(\mathbf{r}))\right) \tag{2.3.5}$$

where $\mathbf{r}$ are the training rays and $T_{FG}(\mathbf{r})$ is the accumulated foreground transmittance (between 0 and 1) of ray $\mathbf{r}$.

2.3.8 Implementation

Since sparse voxel volume rendering is not well-supported in modern autodiff libraries, we created a custom PyTorch CUDA [NVF20] extension library to achieve fast differentiable volume rendering. We also provide a slower, higher-level JAX [BFH+18] implementation. The speed of our implementation is possible in large part because the gradient of our Plenoxel model becomes very sparse very quickly, as shown in Fig. 2.4. Within the first 1-2 minutes of optimization, fewer than 10% of the voxels have nonzero gradients.

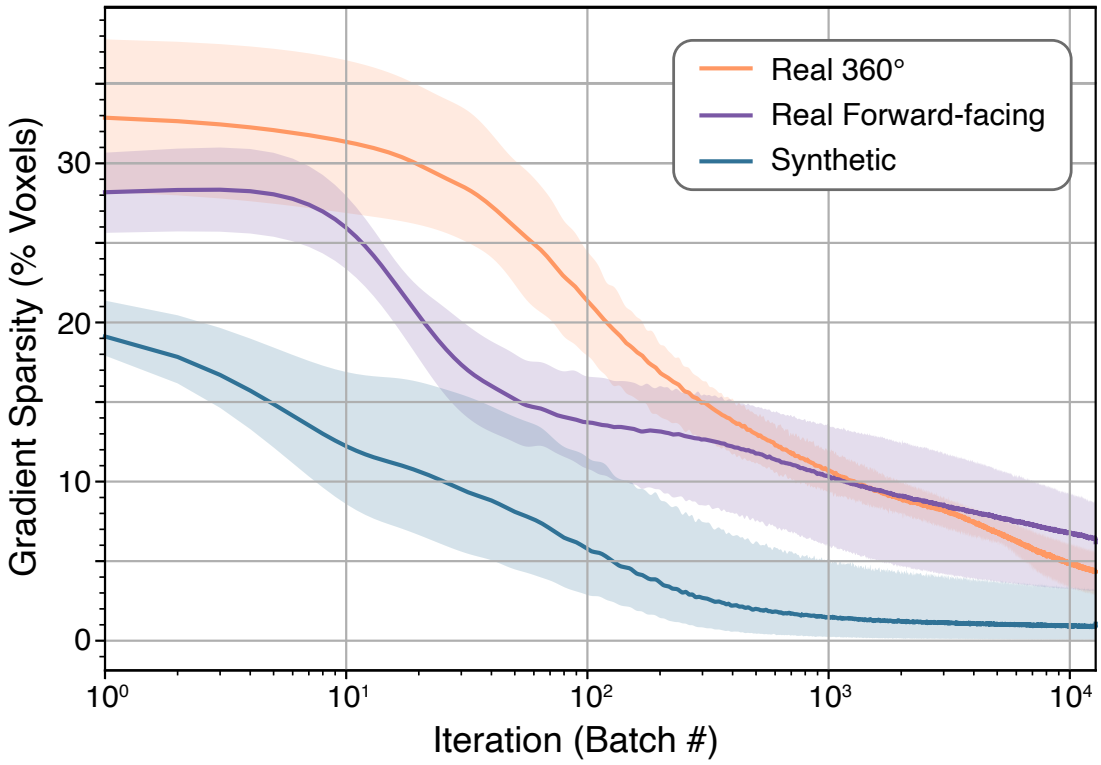


Figure 2.4: **Gradient sparsity.** The gradient becomes very sparse spatially within the first 12800 batches (one epoch for the synthetic scenes), with as few as 1% of the voxels updating per batch in the synthetic case. This enables efficient training via sparse parameter updates. The solid lines show the mean and the shaded regions show the full range of values among all scenes of each type.

Figure 2.5: **1 minute, 20 seconds.** Results on the synthetic scenes after 1 epoch of optimization, an average of 1 minute and 20 seconds.

2.4 Results

We present results on synthetic, bounded scenes; real, unbounded, forward-facing scenes; and real, unbounded, 360° scenes. We include time trial comparisons with prior work, showing dramatic speedup in training compared to all prior methods (alongside real-time rendering). Quantitative comparisons are presented in Tab. 2.2, and visual comparisons

	PSNR ↑	SSIM ↑	LPIPS ↓	Train Time
Ours	31.71	**0.958**	**0.049**	**11 mins**
NV [LSS+19a]	26.05	0.893	0.160	>1 day
JAXNeRF [MST+20; DBS20]	**31.85**	0.954	0.072	1.45 days
Ours	26.29	**0.839**	**0.210**	**24 mins**
LLFF [MSO+19]	24.13	0.798	0.212	—*
JAXNeRF [MST+20; DBS20]	**26.71**	0.820	0.235	1.62 days
Ours	20.40	**0.696**	**0.420**	**27 mins**
NeRF++ [ZRSK20]	**20.49**	0.648	0.478	~4 days

Table 2.2: **Results.** *Top:* average over the 8 synthetic scenes from NeRF; *Middle:* the 8 real, forward-facing scenes from NeRF; *Bottom:* the 4 real, 360° scenes from Tanks and Temples [KPZK17]. 4 of the synthetic scenes train in under 10 minutes. *LLFF requires pretraining a network to predict MPIs for each view, and then can render novel scenes without further training; this pretraining is amortized across all scenes so we do not include it in the table.

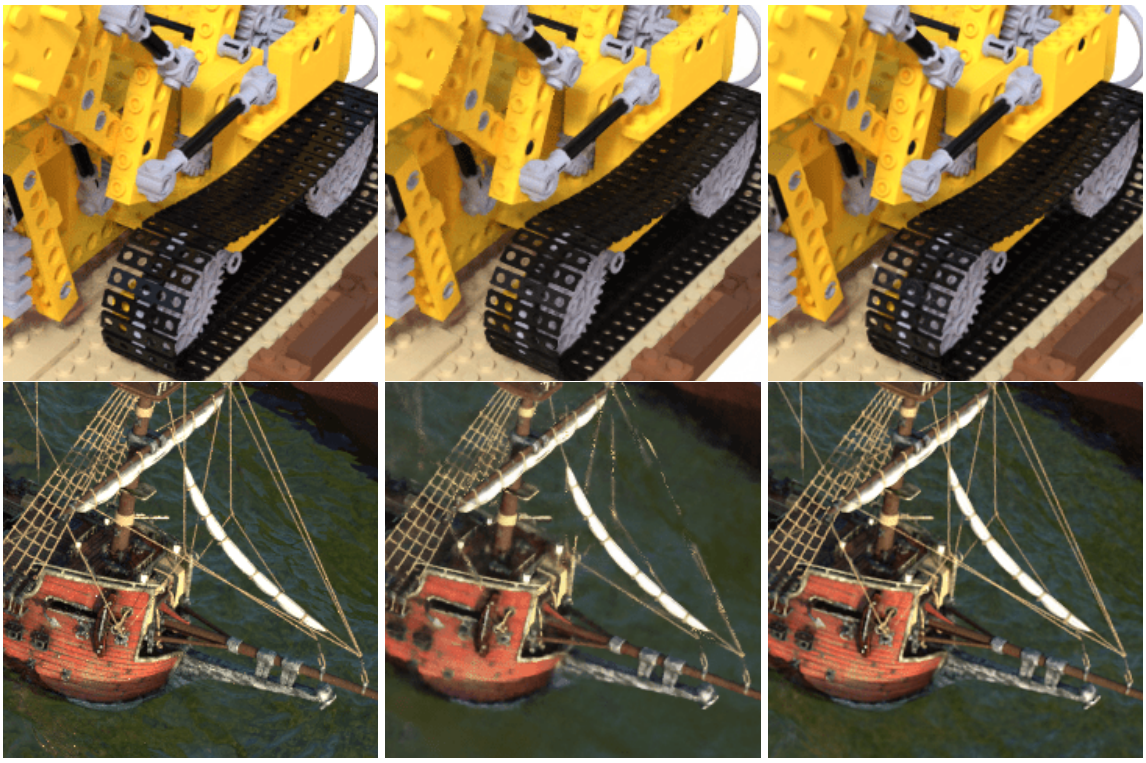
are shown in Fig. 2.1, Fig. 2.6, Fig. 2.7, and Fig. 2.8. Our method achieves quality results after even the first epoch of optimization, less than 1.5 minutes, as shown in Fig. 2.5.

We also present the results from various ablation studies of our method. In the main text we present average results (PSNR, SSIM [WBSS04], and VGG LPIPS [ZIESW18]) over all scenes of each type; full quantitative and visual results on each scene, and full experimental details (hyperparameters, etc.) are included in the appendix.

2.4.1 Synthetic Scenes

Our synthetic experiments use the 8 scenes from NeRF: chair, drums, ficus, hotdog, lego, materials, mic, and ship. Each scene includes 100 ground truth training views with 800×800 resolution, from known camera positions distributed randomly in the upper hemisphere facing the object. Each scene is evaluated on 200 test views, also with resolution 800×800 and known inward-facing camera positions in the upper hemisphere. We provide quantitative comparisons in Tab. 2.2 and visual comparisons in Fig. 2.6.

We compare our method to Neural Volumes (NV) [LSS+19a], a prior grid-based method with a 3D convolutional network, and JAXNeRF [MST+20; DBS20]. For Neural Volumes we use values reported in [MST+20]; for JAXNeRF we report results from our own rerunning, fixing its centered pixel bug [BMT+21a]. Our method achieves comparable quality compared to the best baseline, while training in an average of 11 minutes per scene on a single GPU and supporting interactive rendering.



Ground Truth JAXNeRF [DBS20; MST+20] Plenoxels

Figure 2.6: **Synthetic, bounded scenes.** Example results on the lego and ship synthetic scenes from NeRF [MST+20].

2.4.2 Real Forward-Facing Scenes

We extend our method to unbounded, forward-facing scenes by using normalized device coordinates (NDC), as derived in NeRF [MST+20]. Our method is otherwise identical to the version we use on bounded, synthetic scenes, except that we use TV regularization (with a stronger weight) throughout the optimization. This change is likely necessary because of the reduced number of training views for these scenes, as described in Sec. 2.4.4.

Our forward-facing experiments use the same 8 scenes as in NeRF, 5 of which are originally from LLFF [MSO+19]. Each scene consists of 20 to 60 forward-facing images captured by a handheld cell phone with resolution 1008×756, with $\frac{7}{8}$ of the images used for training and the remaining $\frac{1}{8}$ of the images reserved as a test set.

We compare our method to Local Light Field Fusion (LLFF) [MSO+19], a prior method that uses a 3D convolutional network to predict a grid for each input view, and JAXNeRF. We provide quantitative comparisons in Tab. 2.2 and visual comparisons in Fig. 2.7.

Ground Truth JAXNeRF [DBS20; MST+20] Plenoxels

Figure 2.7: **Real, forward-facing scenes.** Example results on the fern and orchid forward-facing scenes from NeRF.

2.4.3 Real 360° Scenes

We extend our method to real, unbounded 360° scenes by surrounding our sparse voxel grid with an multi-sphere image (MSI, based on multi-plane images introduced by [ZTFFS18]) background model, in which each background sphere is also a simple voxel grid with trilinear interpolation (both within each sphere and between adjacent layers).

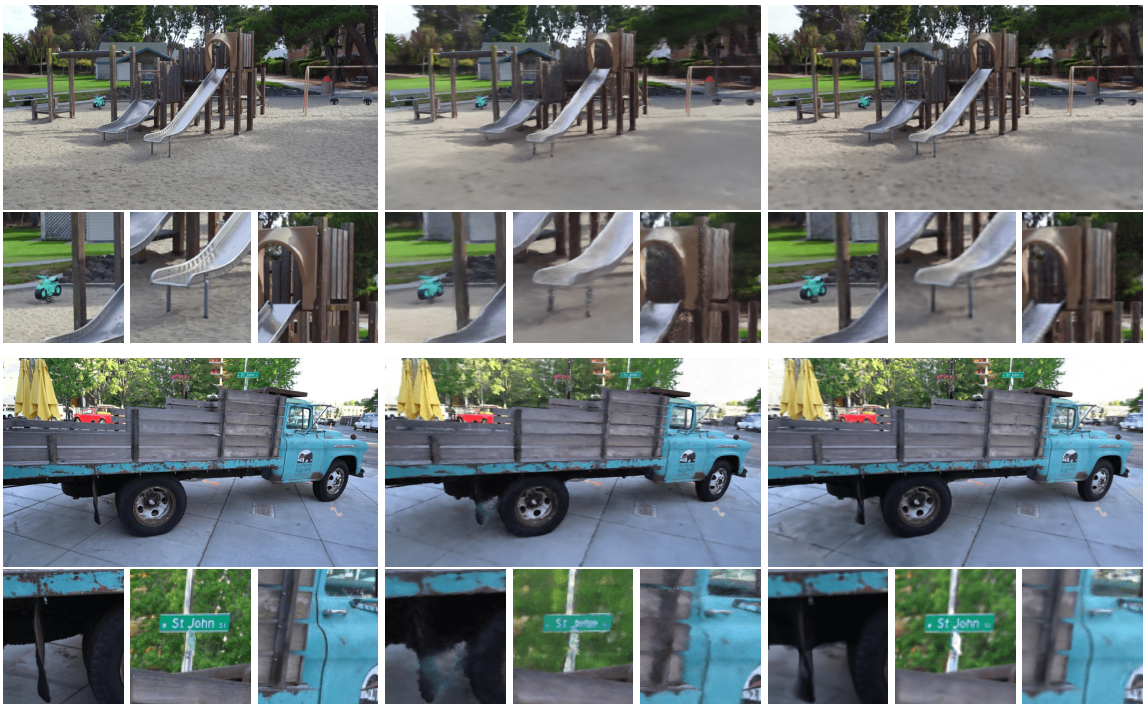
Our 360° experiments use 4 scenes from the Tanks and Temples dataset [KPZK17]: M60, playground, train, and truck. For each scene, we use the same train/test split as [RK20].

We compare our method to NeRF++ [ZRSK20], which augments NeRF with a background model to represent unbounded scenes. We present quantitative comparisons in Tab. 2.2 and visual comparisons in Fig. 2.8.

2.4.4 Ablation Studies

In this section, we perform extensive ablation studies of our method to understand which features are core to its success, with such a simple model. In Tab. 2.1, we show that continuous (in our case, trilinear) interpolation is responsible for dramatic improvement in fidelity compared to nearest neighbor interpolation (*i.e.*, constant within each voxel) [YLT+21].

In Tab. 2.3, we consider how our method handles a dramatic reduction in training data,



Ground Truth NeRF++ [ZRSK20] Plenoxels

Figure 2.8: **Real,** 360° **scenes.** Example results on the playground and truck 360° scenes from Tanks and Temples [KPZK17].

from 100 views to 25 views, on the 8 synthetic scenes. We compare our method to NeRF and find that, despite its lack of complex neural priors, by increasing TV regularization our method can outperform NeRF even in this limited data regime. This ablation also sheds light on why our model performs better with higher TV regularization on the real forward-facing scenes compared to the synthetic scenes: the real scenes have many fewer training images, and the stronger regularizer helps our optimization extend smoothly to sparsely-supervised regions.

We also ablate over the resolution of our Plenoxel grid in Tab. 2.4 and the rendering formula in Tab. 2.5. The rendering formula from Max [Max95] yields a substantial improvement compared to that of Neural Volumes [LSS+19a], perhaps because it is more physically accurate (as discussed further in the appendix). The appendix also includes ablations over the learning rate schedule and optimizer demonstrating Plenoxel optimization to be robust to these hyperparameters.

	PSNR ↑	SSIM ↑	LPIPS ↓
Ours: 100 images (low TV)	31.71	0.958	0.050
NeRF: 100 images [MST+20]	31.01	0.947	0.081
Ours: 25 images (low TV)	26.88	0.911	0.099
Ours: 25 images (high TV)	28.25	0.932	0.078
NeRF: 25 images [MST+20]	27.78	0.925	0.108

Table 2.3: **Ablation over the number of views.** By increasing our TV regularization, we exceed NeRF fidelity even when the number of training views is only a quarter of the full dataset. Results are averaged over the 8 synthetic scenes from NeRF.

Resolution	PSNR ↑	SSIM ↑	LPIPS ↓
512^3	31.71	0.958	0.050
256^3	30.57	0.950	0.065
128^3	28.46	0.926	0.100
64^3	26.11	0.892	0.139
32^3	23.49	0.859	0.174

Table 2.4: **Ablation over the Plenoxel grid resolution.** Results are averaged over the 8 synthetic scenes from NeRF.

Rendering Formula	PSNR ↑	SSIM ↑	LPIPS ↓
Max [Max95], used in NeRF [MST+20]	30.57	0.950	0.065
Neural Volumes [LSS+19a]	27.54	0.906	0.201

Table 2.5: **Comparison of different rendering formulas.** We compare the rendering formula from Max [Max95] (used in NeRF and our main method) to the one used in Neural Volumes [LSS+19a], which uses absolute instead of relative transmittance. Results are averaged over the 8 synthetic scenes from NeRF.

2.5 Discussion

We present a method for photorealistic scene modeling and novel viewpoint rendering that produces results with comparable fidelity to the state-of-the-art, while taking orders of magnitude less time to train. Our method is also strikingly straightforward, shedding light on the core elements that are necessary for solving 3D inverse problems: a differentiable forward model, a continuous representation (in our case, via trilinear interpolation), and appropriate regularization. We acknowledge that the ingredients for this method have been available for a long time, however nonlinear optimization with tens of millions of variables has only recently become accessible to the computer vision practitioner.

Limitations and Future Work. As with any underdetermined inverse problem, our method is susceptible to artifacts. Our method exhibits different artifacts than neural methods, as shown in Fig. 2.9, but both methods achieve similar quality in terms of standard metrics (as presented in Sec. 2.4). Future work may be able to adjust or mitigate these remaining artifacts by studying different regularization priors and/or more physically accurate differentiable rendering functions.

Ground Truth JAXNeRF [DBS20; MST+20] Plenoxels

Figure 2.9: **Artifacts.** JAXNeRF and Plenoxel exhibit slightly different artifacts, as shown here in the specularities in the synthetic drums scene. Note that some artifacts are unavoidable for any underdetermined inverse problem, but the specific artifacts vary depending on the priors induced by the model and regularizer.

Although we report all of our results for each dataset with a fixed set of hyperparameters, there is no optimal *a priori* setting of the TV weight λ_{TV}. Better results may be obtained by tuning this parameter on a scene-by-scene basis, which is possible due to our fast training time. This is expected because the scale, smoothness, and number of training views varies between scenes.

Our method should extend naturally to support multiscale rendering with proper anti-aliasing through voxel cone-tracing, similar to the modifications in Mip-NeRF [BMT+21a]. Another easy addition is tone-mapping to account for white balance and exposure changes

[RFS21], which we expect would help especially in the real 360° scenes. A hierarchical data structure (such as an octree) may provide additional speedup compared to our sparse array implementation, provided that differentiable interpolation is preserved. Likewise, a more compressed representation that takes greater advantage of the native scene sparsity and priors would reduce processor memory needed to optimize and store Plenoxel models, which currently require $\sim$ 10 GB per scene for optimization and $\sim$ 2 GB per scene for storage.

Since our method is two orders of magnitude faster than NeRF, we believe that it may enable downstream applications currently bottlenecked by the performance of NeRF–for example, multi-bounce lighting and 3D generative models across large databases of scenes. By combining our method with additional components such as camera optimization and large-scale voxel hashing, it may enable a practical pipeline for end-to-end photorealistic 3D reconstruction.

Chapter 3

K-Planes: A Compressed Representation for Dynamic Scenes

This chapter is based on the paper "K-Planes: Explicit Radiance Fields in Space, Time, and Appearane" [FMWRK23], written in collaboration with Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa.

Please see `sarafridov.github.io/K-Planes` for videos and code associated to this chapter.

Many thanks to Matthew Tancik, Ruilong Li, and other members of KAIR for helpful discussion and pointers. We also thank the DyNeRF authors for their response to our questions about their method.

3.1 Introduction

Recent interest in dynamic radiance fields demands representations of 4D volumes. However, storing a 4D volume directly is prohibitively expensive due to the curse of dimensionality. Several approaches have been proposed to factorize 3D volumes for static radiance fields, but these do not easily extend to higher dimensional volumes.

We propose a factorization of 4D volumes that is simple, interpretable, compact, and yields fast training and rendering. Specifically, we use six planes to represent a 4D volume, where the first three represent space and the last three represent space-time changes, as illustrated in Fig. 3.1(d). This decomposition of space and space-time makes our model interpretable, *i.e.*, dynamic objects are clearly visible in the space-time planes, whereas static objects only appear in the space planes. This interpretability enables dimension-specific priors in time and space.

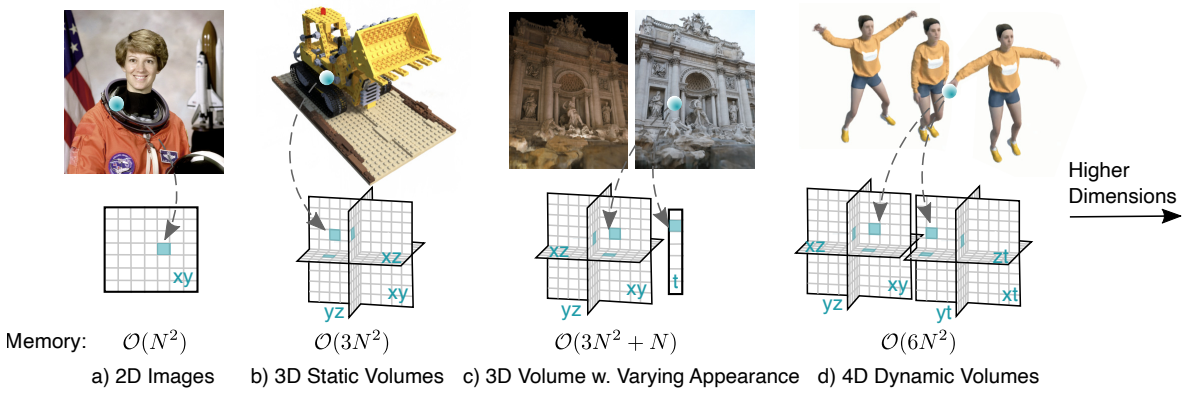


Figure 3.1: **Planar factorization of d-dimensional spaces.** We propose a simple planar factorization for volumetric rendering that naturally extends to arbitrary-dimensional spaces, and that scales gracefully with dimension in both optimization time and model size. We show the advantages of our approach on 3D static volumes, 3D photo collections with varying appearances, and 4D dynamic videos.

More generally, our approach yields a straightforward, prescriptive way to select a factorization of any dimension with 2D planes. For a d-dimensional space, we use $k = \binom{d}{2}$ ("d-*choose*-2") *k-planes*, which represent every pair of dimensions — for example, our model uses $\binom{4}{2} = 6$ *hex-planes* in 4D and reduces to $\binom{3}{2} = 3$ *tri-planes* in 3D. Choosing any other set of planes would entail either using more than k planes and thus occupying unnecessary memory, or using fewer planes and thereby forfeiting the ability to represent some potential interaction between two of the d dimensions. We call our model k-planes; Fig. 3.1 illustrates its natural application to both static and dynamic scenes. Though we do not test this capability, k-planes is naturally suited to higher-dimensional scenes as well – for example, consider hyperspectral imaging of a dynamic scene, with five dimensions

x, y, z, t, λ, or a direct parameterization of view-dependent color in a dynamic scene, with six dimensions x, y, z, t, θ, ϕ.

Most radiance field models entail some black-box components with their use of MLPs. Instead, we seek a simple model whose functioning can be inspected and understood. We find two design choices to be fundamental in allowing k-planes to be a white-box model while maintaining reconstruction quality competitive with or better than previous black-box models [LSZ+22; PCPM21]: (1) Features from our k-planes are *multiplied* together rather than added, as was done in prior work [CLC+22; CXGYS22], and (2) our linear feature decoder uses a learned basis for view-dependent color, enabling greater adaptivity including the ability to model scenes with variable appearance. We show that an MLP decoder can be replaced with this linear feature decoder only when the planes are multiplied, suggesting that the former is involved in both view-dependent color and determining spatial structure.

Our factorization of 4D volumes into 2D planes leads to high compression without relying on MLPs, using 200 MB to represent a 4D volume whose direct representation at the same resolution would require more than 300 GB, a compression rate of three orders of magnitude. Furthermore, despite not using any custom CUDA kernels, k-planes trains orders of magnitude faster than prior implicit models and on par with concurrent hybrid models.

In summary, we present the first white-box, interpretable model capable of representing radiance fields in arbitrary dimensions, including static scenes, dynamic scenes, and scenes with variable appearance. Our k-planes model achieves competitive performance across reconstruction quality, model size, and optimization time across these varied tasks, using a pure PyTorch implementation without any custom CUDA kernels.

3.2 Related work

K-planes is an interpretable, explicit model applicable to static scenes, scenes with varying appearances, and dynamic scenes, with compact model size and fast optimization time. Our model is the first to yield all of these attributes, as illustrated in Tab. 3.1. We further highlight that k-planes satisfies this in a simple framework that naturally extends to arbitrary dimensions.

Spatial decomposition. NeRF[MST+20] proposed a fully implicit model with a large neural network queried many times during optimization, making it slow and essentially a black-box. Several works have used geometric representations to reduce the optimization time. Plenoxels [FYT+22] proposed a fully explicit model with trilinear interpolation in a 3D grid, which reduced the optimization time from hours to a few minutes. However, their explicit grid representation of 3D volumes, and that of DVGO [SSC22], grows exponentially with dimension, making it challenging to scale to high resolution and completely intractable for 4D dynamic volumes.

	Static	Appearance	Dynamic	Fast	Compact	Explicit
NeRF	✓	✗	✗	✗	✓	✗
NeRF-W	✓	✓	✗	✗	✓	✗
DVGO	✓	✗	✗	✓	✗	✗
Plenoxels	✓	✗	✗	✓	✗	✓
Instant-NGP, TensoRF	✓	✗	✗	✓	✓	✗[1]
DyNeRF, D-NeRF	–	✗	✓	✗	✓	✗
TiNeuVox, Tensor4D	–	✗	✓	✓	✓	✗
MixVoxels, V4D	–	✗	✓	✓	✗	✗
NeRFPlayer	–	✗	✓	✓	✓[2]	✗
K-planes hybrid (Ours)	✓	✓	✓	✓	✓	✗
K-planes explicit (Ours)	✓	✓	✓	✓	✓	✓

[1] TensoRF offers both hybrid and explicit versions, with a small quality gap [2] NerfPlayer offers models at different sizes, the smallest of which has < 100 million parameters but the largest of which has > 300 million parameters

Table 3.1: **Related work overview.** The k-planes model works for a diverse set of scenes and tasks (static, varying appearance, and dynamic). It has a low memory usage (compact) and fast training and inference time (fast). Here "fast" includes any model that can optimize within a few (< 6) hours on a single GPU, and "compact" denotes models that use less than roughly 100 million parameters. "Explicit" denotes white-box models that do not rely on MLPs as a representation or nonlinear feature decoder.

Hybrid methods [SSC22; MESK22; CXGYS22] retain some explicit geometric structure, often compressed by a spatial decomposition, alongside a small MLP feature decoder. Instant-NGP [MESK22] proposed a multiresolution voxel grid encoded implicitly via a hash function, allowing fast optimization and rendering with a compact model. TensoRF [CXGYS22] achieved similar model compression and speed by replacing the voxel grid with a tensor decomposition into planes and vectors. In a generative setting, EG3D [CLC+22] proposed a similar spatial decomposition into three planes, whose values are added together to represent a 3D volume.

Our work is inspired by the explicit modeling of Plenoxels as well as these spatial decompositions, particularly the triplane model of [CLC+22], the tensor decomposition of [CXGYS22], and the multiscale grid model of [MESK22]. We also draw inspiration from Extreme MRI [OZC+20], which uses a multiscale low-rank decomposition to represent 4D

dynamic volumes in magnetic resonance imaging. These spatial decomposition methods have been shown to offer a favorable balance of memory efficiency and optimization time for static scenes. However, it is not obvious how to extend these factorizations to 4D volumes in a memory-efficient way. *K*-planes defines a unified framework that enables efficient and interpretable factorizations of 3D and 4D volumes and trivially extends to even higher dimensional volumes.

Dynamic volumes. Applications such as Virtual Reality (VR) and Computed Tomography (CT) often require the ability to reconstruct 4D volumes. Several works have proposed extensions of NeRF to dynamic scenes. The two most common schemes are (1) modeling a deformation field on top of a static *canonical* field [PCPM21; TTG+21; PSB+21; DZYTW21; YLSL21; FYW+22; LNSW21], or (2) directly learning a radiance field conditioned on time[XHKK21; LNSW21; GSKH21; LSZ+22; PSH+21]. The former makes decomposing static and dynamic components easy [YLSL21; WZTCO22], but struggles with changes in scene topology (e.g. when a new object appears), while the latter makes disentangling static and dynamic objects hard. A third strategy is to choose a representation of 3D space and repeat it at each timestep (e.g. NeRFPlayer [SCL+22]), resulting in a model that ignores space-time interactions and can become impractically large for long videos.

Further, some of these models are fully implicit [PCPM21; LSZ+22] and thus suffer from extremely long training times (e.g. DyNeRF used 8 GPUs for 1 week to train a single scene), as well as being completely black-box. Others use partially explicit decompositions for video [FYW+22; GCD+22; WTLTL22; GXHCY22; SZT+22; LCM+22; LSS+19b; SCL+22], usually combining some voxel or spatially decomposed feature grid with one or more MLP components for feature decoding and/or representing scene dynamics. Most closely related to *k*-planes is Tensor4D [SZT+22], which uses 9 planes to decompose 4D volumes. *K*-planes is less redundant (*e.g.* Tensor4D includes two yt planes), does not rely on multiple MLPs, and offers a simpler factorization that naturally generalizes to static and dynamic scenes. Our method combines a fully explicit representation with a built-in decomposition of static and dynamic components, the ability to handle arbitrary topology and lighting changes over time, fast optimization, and compactness.

Appearance embedding. Reconstructing large environments from photographs taken with varying illumination is another domain in which implicit methods have shown appealing results, but hybrid and explicit approaches have not yet gained a foothold. NeRF-W [MRS+21] was the first to demonstrate photorealistic view synthesis in such environments. They augment a NeRF-based model with a learned global appearance code per frame, enabling it to explain away changes in appearance, such as time of day. With Generative Latent Optimization (GLO) [BJLS17], these appearance codes can further be used to manipulate the scene appearance by interpolation in the latent appearance space. Block-NeRF [TCY+22] employs similar appearance codes.

We show that our *k*-planes representation can also effectively reconstruct these unbounded environments with varying appearance. We similarly extend our model – either the learned color basis in the fully explicit version, or the MLP decoder in the hybrid

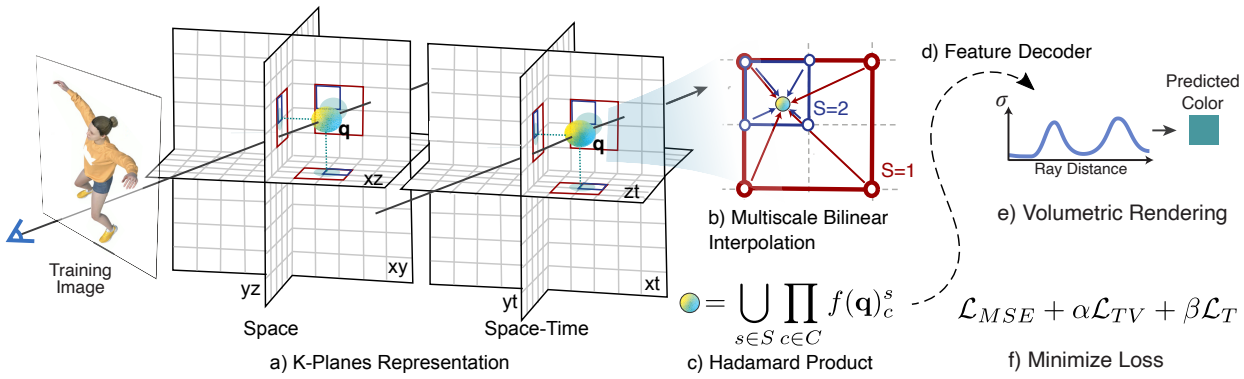


Figure 3.2: **Method overview.** (a) Our k-planes representation factorizes 4D dynamic volumes into six planes, three for space and three for spatiotemporal variations. To obtain the value of a 4D point $\mathbf{q} = (x, y, z, t)$, we first project the point into each plane, in which we (b) do multiscale bilinear interpolation. (c) The interpolated values are multiplied and then concatenated over the S scales. (d) These features are decoded either with a small MLP or our explicit linear decoder. (e) We follow the standard volumetric rendering formula to predict ray color and density. The model is optimized by (f) minimizing the reconstruction loss with simple regularization in space and time.

version – with a global appearance code to disentangle global appearance from a scene without affecting geometry. To the best of our knowledge, ours is both the first fully explicit and the first hybrid method to successfully reconstruct these challenging scenes.

3.3 K-planes model

We propose a simple and interpretable model for representing scenes in arbitrary dimensions. Our representation yields low memory usage and fast training and rendering. The k-planes factorization, illustrated in Fig. 3.2, models a d-dimensional scene using $k = \binom{d}{2}$ planes representing every combination of two dimensions. For example, for static 3D scenes, this results in *tri-planes* with $\binom{3}{2} = 3$ planes representing xy, xz, and yz. For dynamic 4D scenes, this results in *hex-planes*, with $\binom{4}{2} = 6$ planes including the three space-only planes and three space-time planes xt, yt, and zt. Should we wish to represent a 5D space, we could use $\binom{5}{2} = 10$ *deca-planes*. In the following section, we describe the 4D instantiation of our k-planes factorization.

3.3.1 Hex-planes

The hex-planes factorization uses six planes. We refer to the space-only planes as $\mathbf{P}_{xy}$, $\mathbf{P}_{xz}$, and $\mathbf{P}_{yz}$, and the space-time planes as $\mathbf{P}_{xt}$, $\mathbf{P}_{yt}$, and $\mathbf{P}_{zt}$. Assuming symmetric spatial

and temporal resolution N for simplicity of illustration, each of these planes has shape $N\boldsymbol{x}N\boldsymbol{x}M$, where M is the size of stored features that capture the density and view-dependent color of the scene.

We obtain the features of a 4D coordinate $\boldsymbol{q} = (i, j, k, \tau)$ by normalizing its entries between $[0, N)$ and projecting it onto these six planes

$$f(\boldsymbol{q})_c = \psi\big(\mathbf{P}_c, \pi_c(\boldsymbol{q})\big), \tag{3.3.1}$$

where π_c projects $\boldsymbol{q}$ onto the c'th plane and ψ denotes bilinear interpolation of a point into a regularly spaced 2D grid. We repeat Eq. (3.3.1) for each plane $c \in C$ to obtain feature vectors $f(\boldsymbol{q})_c$. We combine these features over the six planes using the Hadamard product (elementwise multiplication) to produce a final feature vector of length M

$$f(\boldsymbol{q}) = \prod_{c \in C} f(\boldsymbol{q})_c. \tag{3.3.2}$$

These features will be decoded into color and density using either a linear decoder or an MLP, described in Sec. 3.3.3.

Why Hadamard product? In 3D, k-planes reduces to the tri-plane factorization, which is similar to [CLC+22] except that the elements are multiplied. A natural question is why we multiply rather than add, as has been used in prior work with tri-plane models [CLC+22; PNMPG20]. Fig. 3.3 illustrates that combining the planes by multiplication allows k-planes to produce spatially localized signals, which is not possible with addition.

This selection ability of the Hadamard product produces substantial rendering improvements for linear decoders and modest improvement for MLP decoders, as shown in Tab. 3.2. This suggests that the MLP decoder is involved in both view-dependent color and determining spatial structure. The Hadamard product relieves the feature decoder of this extra task and makes it possible to reach similar performance using a linear decoder solely responsible for view-dependent color.

Plane Combination	Explicit	Hybrid	# params ↓
Multiplication	35.29	35.75	33M
Addition	28.78	34.80	33M

Table 3.2: **Ablation study over Hadamard product.** Multiplication of plane features yields a large improvement in PSNR ↑ for our explicit model, whereas our hybrid model can use its MLP decoder to partially compensate for the less expressive addition of planes. This experiment uses the static *Lego* scene [MST+20] with 3 scales: 128, 256, and 512, and 32 features per scale.

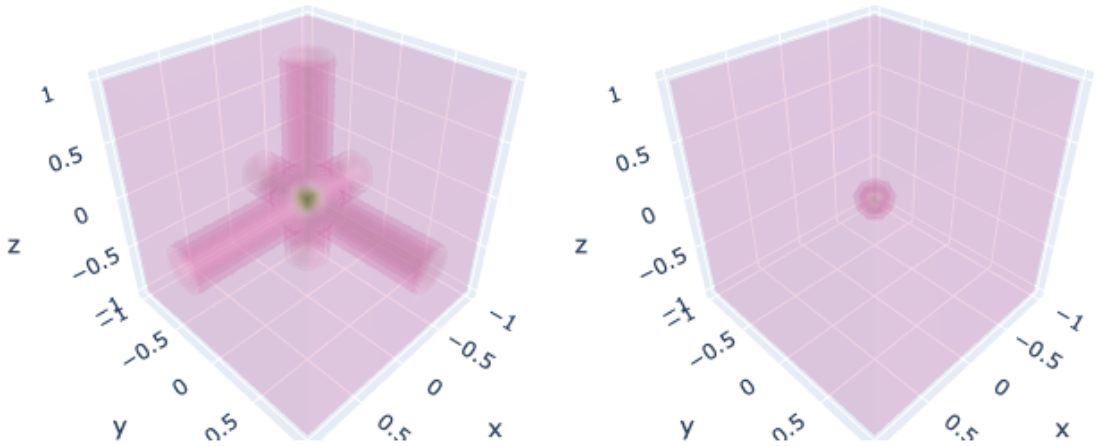


Figure 3.3: **Addition versus Hadamard product.** Elementwise addition of plane features (left) compared to multiplication (right), in a triplane example. A single entry in each plane is positive and the rest are zero, selecting a single 3D point by multiplication but producing intersecting lines by addition. This selection ability of multiplication improves the expressivity of our explicit model.

3.3.2 Interpretability

The separation of space-only and space-time planes makes the model interpretable and enables us to incorporate dimension-specific priors. For example, if a region of the scene never moves, its temporal component will always be 1 (the multiplicative identity), thereby just using the features from the space planes. This offers compression benefits since a static region can easily be identified and compactly represented. Furthermore, the space-time separation improves interpretability, *i.e.*, we can track the changes in time by visualizing the elements in the time-space planes that are not 1. This simplicity, separation, and interpretability make adding priors straightforward.

Multiscale planes. To encourage spatial smoothness and coherence, our model contains multiple copies at different spatial resolutions, for example 64, 128, 256, and 512. Models at each scale are treated separately, and the M-dimensional feature vectors from different scales are concatenated together before being passed to the decoder. The red and blue squares in Fig. 3.2a-b illustrate bilinear interpolation with multiscale planes. Inspired by the multiscale hash mapping of Instant-NGP[MESK22], this representation efficiently encodes spatial features at different scales, allowing us to reduce the number of features stored at the highest resolution and thereby further compressing our model. Empirically, we do not find it necessary to represent our time dimension at multiple scales.

Total variation in space. Spatial total variation regularization encourages sparse gradients (with L1 norm) or smooth gradients (with L2 norm), encoding priors over

edges being either sparse or smooth in space. We encourage this in 1D over the spatial dimensions of each of our space-time planes and in 2D over our space-only planes:

$$\mathcal{L}_{TV}(\mathbf{P}) = \frac{1}{|C|n^2} \sum_{c,i,j} \left(\|\mathbf{P}_c^{i,j} - \mathbf{P}_c^{i-1,j}\|_2^2 + \|\mathbf{P}_c^{i,j} - \mathbf{P}_c^{i,j-1}\|_2^2 \right), \tag{3.3.3}$$

where i, j are indices on the plane's resolution. Total variation is a common regularizer in inverse problems and was used in Plenoxels [FYT+22] and TensoRF [CXGYS22]. We use the L2 version in our results, though we find that either L2 or L1 produces similar quality.

Smoothness in time. We encourage smooth motion with a 1D Laplacian (second derivative) filter

$$\mathcal{L}_{smooth}(\mathbf{P}) = \frac{1}{|C|n^2} \sum_{c,i,t} \|\mathbf{P}_c^{i,t-1} - 2\mathbf{P}_c^{i,t} + \mathbf{P}_c^{i,t+1}\|_2^2, \tag{3.3.4}$$

to penalize sharp "acceleration" over time. We only apply this regularizer on the time dimension of our space-time planes. Please see the appendix for an ablation study.

Sparse transients. We want the static part of the scene to be modeled by the space-only planes. We encourage this separation of space and time by initializing the features in the space-time planes as 1 (the multiplicative identity) and using an ℓ_1 regularizer on these planes during training:

$$\mathcal{L}_{sep}(\mathbf{P}) = \sum_c \|\mathbf{1} - \mathbf{P}_c\|_1, \qquad c \in \{xt, yt, zt\}. \tag{3.3.5}$$

In this way, the space-time plane features of the k-planes decomposition will remain fixed at 1 if the corresponding spatial content does not change over time.

3.3.3 Feature decoders

We offer two methods to decode the M-dimensional temporally- and spatially-localized feature vector $f(\boldsymbol{q})$ from Eq. (3.3.2) into density, σ, and view-dependent color, $\boldsymbol{c}$.

Learned color basis: a linear decoder and explicit model. Plenoxels [FYT+22], Plenoctrees [YLT+21], and TensoRF [CXGYS22] proposed models where spatially-localized features are used as coefficients of the spherical harmonic (SH) basis, to describe view-dependent color. Such SH decoders can give both high-fidelity reconstructions and enhanced interpretability compared to MLP decoders. However, SH coefficients are difficult to optimize, and their expressivity is limited by the number of SH basis functions used (often limited 2nd degree harmonics, which produce blurry specular reflections).

Instead, we replace the SH functions with a learned basis, retaining the interpretability of treating features as coefficients for a linear decoder yet increasing the expressivity of the basis and allowing it to adapt to each scene, as was proposed in NeX [WPYS21]. We represent the basis using a small MLP that maps each view direction $\boldsymbol{d}$ to red $b_R(\boldsymbol{d}) \in \mathbb{R}^M$, green $b_G(\boldsymbol{d}) \in \mathbb{R}^M$, and blue $b_B(\boldsymbol{d}) \in \mathbb{R}^M$ *basis vectors*. The MLP serves as an adaptive

drop-in replacement for the spherical harmonic basis functions repeated over the three color channels. We obtain the color values

$$\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{d}) = \bigcup_{i \in \{R,G,B\}} f(\boldsymbol{q}) \cdot b_i(\boldsymbol{d}), \tag{3.3.6}$$

where $\cdot$ denotes the dot product and $\cup$ denotes concatenation. Similarly, we use a learned basis $b_\sigma \in \mathbb{R}^M$, independent of the view direction, as a linear decoder for density:

$$\sigma(\boldsymbol{q}) = f(\boldsymbol{q}) \cdot b_\sigma. \tag{3.3.7}$$

Predicted color and density values are finally forced to be in their valid range by applying the sigmoid to $\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{d})$, and the exponential (with truncated gradient) to $\sigma(\boldsymbol{q})$.

MLP decoder: a hybrid model. Our model can also be used with an MLP decoder like that of Instant-NGP [MESK22] and DVGO [SSC22], turning it into a hybrid model. In this version, features are decoded by two small MLPs, one g_σ that maps the spatially-localized features into density σ and additional features $\hat{f}$, and another g_{RGB} that maps $\hat{f}$ and the embedded view direction $\gamma(\boldsymbol{d})$ into RGB color

$$\begin{aligned} \sigma(\boldsymbol{q}), \hat{f}(\boldsymbol{q}) &= g_\sigma(f(\boldsymbol{q})) \\ \boldsymbol{c}(\boldsymbol{q}, \boldsymbol{d}) &= g_{RGB}(\hat{f}(\boldsymbol{q}), \gamma(\boldsymbol{d})). \end{aligned} \tag{3.3.8}$$

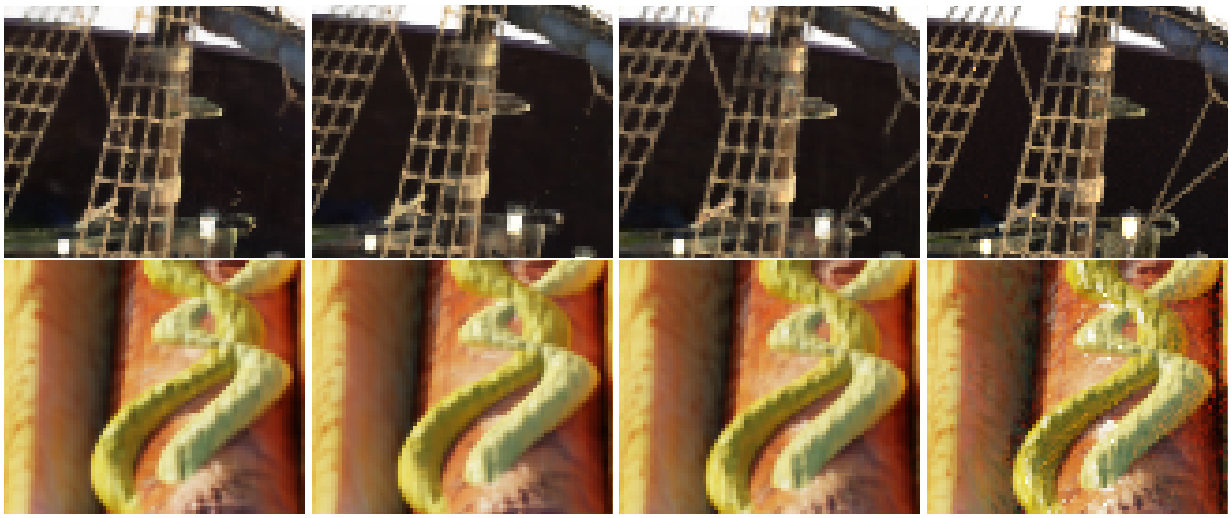
As in the linear decoder case, the predicted density and color values are finally normalized via exponential and sigmoid, respectively.

Global appearance. We also show a simple extension of our k-planes model that enables it to represent scenes with consistent, static geometry viewed under varying lighting or appearance conditions. Such scenes appear in the Phototourism [JMM+21] dataset of famous landmarks photographed at different times of day and in different weather. To model this variable appearance, we augment k-planes with an M-dimensional vector for each training image $1, \ldots, T$. Similar to NeRF-W [MRS+21], we optimize this per-image feature vector and pass it as an additional input to either the MLP learned color basis b_R, b_G, b_B, in our explicit version, or to the MLP color decoder g_{RGB}, in our hybrid version, so that it can affect color but not geometry.

3.3.4 Optimization details

Contraction and normalized device coordinates. For forward-facing scenes, we apply normalized device coordinates (NDC) [MST+20] to better allocate our resolution while enabling unbounded depth. We also implement an ℓ_∞ version (rather than ℓ_2) of the scene contraction proposed in Mip-NeRF 360 [BMVSH22], which we use on the unbounded Phototourism scenes.

Proposal sampling. We use a variant of the proposal sampling strategy from Mip-NeRF 360 [BMVSH22], with a small instance of k-planes as density model. Proposal sampling



(a) Ours-explicit **(b)** Ours-hybrid **(c)** TensoRF **(d)** Ground truth

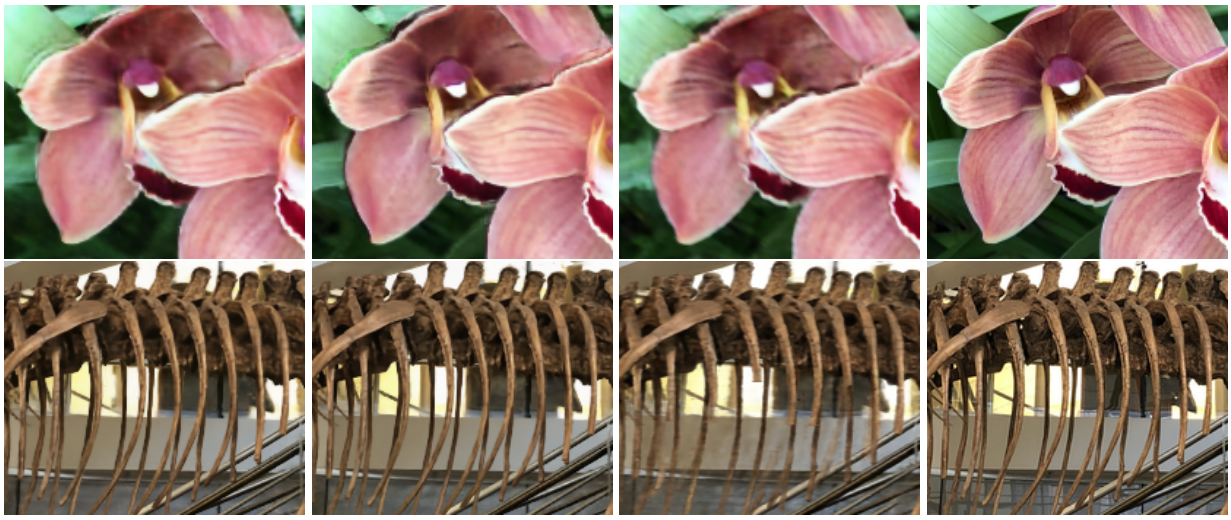
Figure 3.4: **Zoomed qualitative results on static NeRF scenes.** Visual comparison of k-planes, TensoRF [CXGYS22], and the ground truth, on *ship* (top) and *hotdog* (bottom).

works by iteratively refining density estimates along a ray, to allocate more points in the regions of higher density. We use a two-stage sampler, resulting in fewer samples that must be evaluated in the full model and in sharper details by placing those samples closer to object surfaces. The density models used for proposal sampling are trained with the histogram loss [BMVSH22].

Importance sampling. For multiview dynamic scenes, we implement a version of the importance sampling based on temporal difference (IST) strategy from DyNeRF [LSZ+22]. During the last portion of optimization, we sample training rays proportionally to the maximum variation in their color within 25 frames before or after. This results in higher sampling probabilities in the dynamic region. We apply this strategy after the static scene has converged with uniformly sampled rays. In our experiments, IST has only a modest impact on full-frame metrics but improves visual quality in the small dynamic region. Note that importance sampling cannot be used for monocular videos or datasets with moving cameras.

3.4 Results

We demonstrate the broad applicability of our planar decomposition via experiments in three domains: static scenes (both bounded 360° and unbounded forward-facing), dynamic scenes (forward-facing multi-view and bounded 360° monocular), and Phototourism scenes with variable appearance. For all experiments, we report the metrics PSNR



(a) Ours-explicit **(b)** Ours-hybrid **(c)** TensoRF **(d)** Ground truth

Figure 3.5: **Zoomed qualitative results on static LLFF scenes.** Visual comparison of k-planes, TensoRF [CXGYS22], and the ground truth, on *orchids* (top) and *T-rex* (bottom).

(pixel-level similarity) and SSIM[1] [WBSS04] (structural similarity), as well as approximate training time and number of parameters (in millions), in Tab. 3.3. Blank entries in Tab. 3.3 denote baseline methods for which the corresponding information is not readily available. Full per-scene results may be found in the appendix.

3.4.1 Static scenes

We first demonstrate our triplane model on the bounded, 360°, synthetic scenes from NeRF [MST+20]. We use a model with three symmetric spatial resolutions $N \in \{128, 256, 512\}$ and feature length $M = 32$ at each scale; please see the appendix for ablation studies over these hyperparameters. The explicit and hybrid versions of our model perform similarly, within the range of recent results on this benchmark. Fig. 3.4 shows zoomed-in visual results on a small sampling of scenes. We also present results of our triplane model on the unbounded, forward-facing, real scenes from LLFF [MSC+19]. Our results on this dataset are similar to the synthetic static scenes; both versions of our model match or exceed the prior state-of-the-art, with the hybrid version achieving slightly higher metrics than the fully explicit version. Fig. 3.5 shows zoomed-in visual results on a small sampling of scenes.

[1]Note that among prior work, some evaluate using an implementation of SSIM from MipNeRF [BMT+21b] whereas others use the scikit-image implementation, which tends to produce higher values. For fair comparison on each dataset we make a best effort to use the same SSIM implementation as the relevant prior work.

	PSNR ↑	SSIM ↑	Train Time ↓	# Params ↓
NeRF [MST+20] (static, synthetic)				
Ours-explicit	32.21	0.960	38 min	33M
Ours-hybrid	32.36	0.962	38 min	33M
Plenoxels [FYT+22]	31.71	0.958	11 min	∼500M
TensoRF [CXGYS22]	33.14	0.963	17 min	18M
I-NGP [MESK22]	33.18	-	5 min	∼ 16M
LLFF [MSC+19] (static, real)				
Ours-explicit	26.78	0.841	33 min	19M
Ours-hybrid	26.92	0.847	33 min	19M
Plenoxels	26.29	0.839	24 min	∼500M
TensoRF	26.73	0.839	25 min	45M
D-NeRF [PCPM21] (dynamic, synthetic)				
Ours-explicit	31.05	0.97	52 min	37M
Ours-hybrid	31.61	0.97	52 min	37M
D-NeRF	29.67	0.95	48 hrs	1-3M
TiNeuVox[FYW+22]	32.67	0.97	30 min	∼12M
V4D[GXHCY22]	33.72	0.98	4.9 hrs	275M
DyNeRF [LSZ+22] (dynamic, real)				
Ours-explicit	30.88	0.960	3.7 hrs	51M
Ours-hybrid	31.63	0.964	1.8 hrs	27M
DyNeRF [LSZ+22]	[1]29.58	-	1344 hrs	7M
LLFF [MSC+19]	[1]23.24	-	-	-
MixVoxels-L[WTLTL22]	30.80	0.960	1.3 hrs	125M
Phototourism [JMM+21] (variable appearance)				
Ours-explicit	22.25	0.859	35 min	36M
Ours-hybrid	22.92	0.877	35 min	36M
NeRF-W [MRS+21]	27.00	0.962	384 hrs	∼2M
NeRF-W (public)[2]	19.70	0.764	164 hrs	∼2M
LearnIt [TMW+21b]	19.26	-	-	-

[1] DyNeRF and LLFF only report metrics on the *flame salmon* video (the first 10 seconds); average performance may be higher as this is one of the more challenging videos. [2] Open-source version `https://github.com/kwea123/nerf_pl/tree/nerfw` where we re-implemented test-time optimization as for *k*-planes.

Table 3.3: **Results.** Averaged metrics over all scenes in the respective datasets. Note that Phototourism scenes use MS-SSIM (multiscale structural similarity) instead of SSIM. *K*-planes timings are based on a single NVIDIA A30 GPU. Please see the appendix for per-scene results and the website for video reconstructions.

Figure 3.6: **Qualitative video results.** Our hexplane model rivals the rendering quality of state-of-the-art neural rendering methods. Our renderings were obtained after at most 4 hours of optimization on a single GPU whereas DyNeRF trained for a week on 8 GPUs. MixVoxels frame comes from a slightly different video rendering, and is thus slightly shifted.

3.4.2 Dynamic scenes

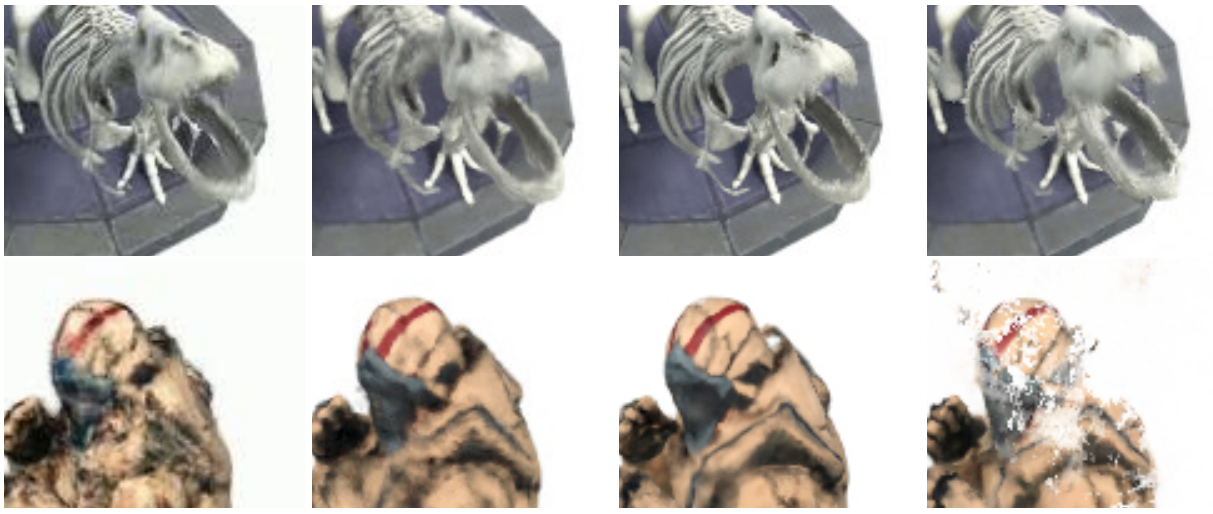
We evaluate our hexplane model on two dynamic scene datasets: a set of synthetic, bounded, 360°, monocular videos from D-NeRF [PCPM21] and a set of real, unbounded, forward-facing, multiview videos from DyNeRF [LSZ+22].

The D-NeRF dataset contains eight videos of varying duration, from 50 frames to 200 frames per video. Each timestep has a single training image from a different viewpoint; the camera "teleports" between adjacent timestamps [GLTRK22]. Standardized test views are from novel camera positions at a range of timestamps throughout the video. Both our explicit and hybrid models outperform D-NeRF in both quality metrics and training time, though they do not surpass very recent hybrid methods TiNeuVox [FYW+22] and V4D [GXHCY22], as shown in Fig. 3.7.

The DyNeRF dataset contains six 10-second videos recorded at 30 fps simultaneously by 15-20 cameras from a range of forward-facing view directions; the exact number of cameras varies per scene because a few cameras produced miscalibrated videos. A central camera is reserved for evaluation, and training uses frames from the remaining cameras. Both our methods again produce similar quality metrics to prior state-of-the-art, including recent hybrid method MixVoxels [WTLTL22], with our hybrid method achieving higher quality metrics. See Fig. 3.6 for a visual comparison.

3.4.2.1 Decomposing time and space

One neat consequence of our planar decomposition of time and space is that it naturally disentangles dynamic and static portions of the scene. The static-only part of the scene can be obtained by setting the three time planes to one (the multiplicative identity). Subtracting the static-only rendered image from the full rendering (i.e. with the time plane parameters not set to 1), we can reveal the dynamic part of the scene. Fig. 3.9 shows



(a) Ours-explicit **(b)** TiNeuVox **(c)** V4D **(d)** D-NeRF

Figure 3.7: **Zoomed qualitative results on scenes from D-NeRF [PCPM21].** Visual comparison of k-planes, D-NeRF [PCPM21], TiNeuVox [FYW+22] and V4D [GXHCY22], on *t-rex* (top) and *hook* (bottom).

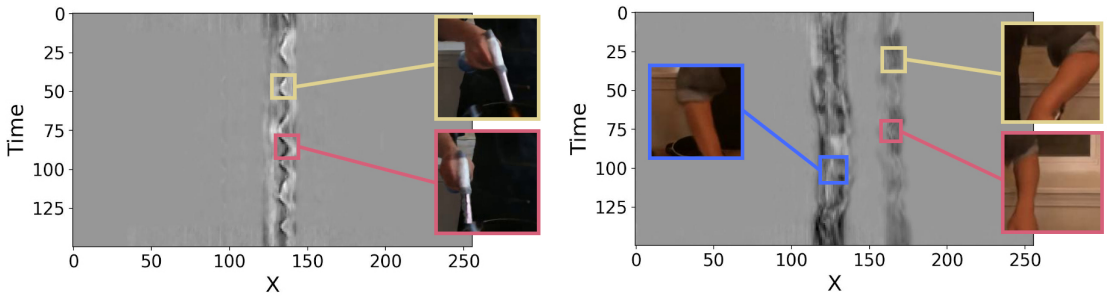


Figure 3.8: **Visualization of a time plane.** The xt plane highlights the dynamic regions in the scene. The wiggly patterns across time correspond to the motion of the person's hands and cooking tools, in the *flame salmon* scene (left) where only one hand moves and the *cut beef* scene (right) where both hands move.

this decomposition of time and space. This natural volumetric disentanglement of a scene into static and dynamic regions may enable many applications across augmented and virtual reality [BWCB22].

We can also visualize the time planes to better understand where motion occurs in a video. Fig. 3.8 shows the averaged features learned by the xt plane in our model for the *flame salmon* and *cut beef* DyNeRF videos, in which we can identify the motions of the hands in both space and time. The xt plane learns to be sparse, with most entries equal to the multiplicative identity, due to a combination of our sparse transients prior and the

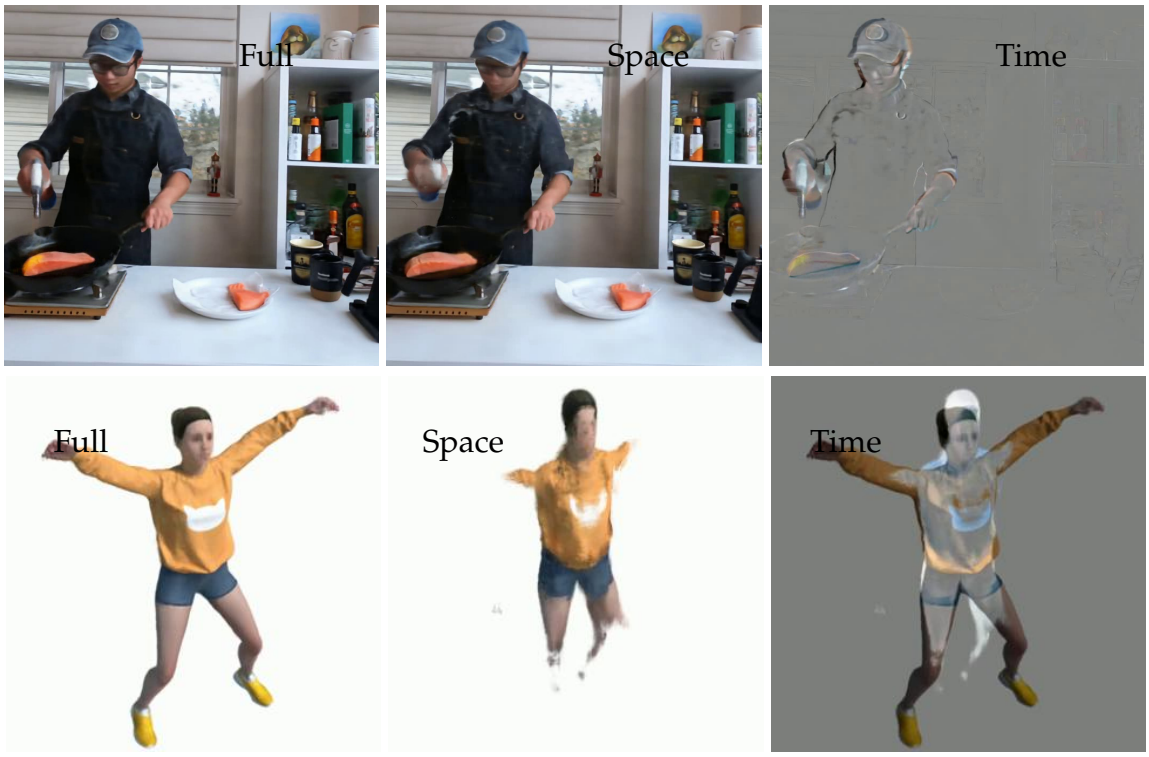


Figure 3.9: **Decomposition of space and time.** *K*-planes (left) naturally decomposes a 3D video into static and dynamic components. We render the static part (middle) by setting the time planes to the identity, and the remainder (right) is the dynamic part. Top shows the *flame salmon* multiview video [LSZ+22] and bottom shows the *jumping jacks* monocular video [PCPM21].

true sparsity of motion in the video. For example, in the left side of Fig. 3.8 one of the cook's arms contains most of the motion, while in the right side both arms move. Having access to such an explicit representation of time allows us to add time-specific priors.

3.4.3 Variable appearance

Our variable appearance experiments use the Phototourism dataset [JMM+21], which includes photos of well-known landmarks taken by tourists with arbitrary view directions, lighting conditions, and transient occluders, mostly other tourists. Our experimental conditions parallel those of NeRF-W [MRS+21]: we train on more than a thousand tourist photographs and test on a standard set that is free of transient occluders. Like NeRF-W, we evaluate on test images by optimizing our per-image appearance feature on the left half of the image and computing metrics on the right half. Visual comparison to prior work is shown in the appendix.

Figure 3.10: **Appearance interpolation**. Like NeRF-W [MRS+21], we can interpolate our appearance code to alter the visual appearance of landmarks. We show three test views from the *Trevi fountain* with appearance codes corresponding to day and night.

Also similar to NeRF-W [MRS+21; BJLS17], we can interpolate in the appearance code space. Since only the color decoder (and not the density decoder) takes the appearance code as input, our approach is guaranteed not to change the geometry, regardless of whether we use our explicit or our hybrid model. Fig. 3.10 shows that our planar decomposition with a 32-dimensional appearance code is sufficient to accurately capture global appearance changes in the scene.

3.5 Conclusions

We introduced a simple yet versatile method to decompose a d-dimensional space into $\binom{d}{2}$ planes, which can be optimized directly from indirect measurements and scales gracefully in model size and optimization time with increasing dimension, without any custom CUDA kernels. We demonstrated that the proposed k-planes decomposition applies naturally to reconstruction of static 3D scenes as well as dynamic 4D videos, and with the addition of a global appearance code can also extend to the more challenging task of unconstrained scene reconstruction. K-planes is the first explicit, simple model to demonstrate competitive performance across such varied tasks.

Chapter 4

A Theory of Photorealistic Reconstruction

This chapter was written in collaboration with Mahdi Soltanokotabi and Benjamin Recht.

4.1 Introduction

In this chapter, we study the same task – recovering a volume given images of it – from a theoretical perspective. We study the measurement model itself, and the extent to which it may be inverted to recover the underlying volume. We seek to understand to what fidelity, and at what rate, iterative gradient-based optimization with a pure grid representation, like that used in Chapter 2, succeeds in volume reconstruction. Analysis to address these questions can help guide future algorithmic and modeling choices, as well as provide guidance about data collection, such as how many views are necessary and how to best leverage regularization.

By no means does this chapter fully resolve these questions. However, we take first steps suggesting that the optimization in Chapter 2 is successful despite the nonconvexity of its data-fidelity objective. We also draw parallels between the photorealistic volume recovery problem at the focus of this dissertation, and the well-studied inverse problem of computed tomography. We show that the tomography problem can be viewed as a special case of the photorealistic reconstruction problem, and in this special case stronger results can be shown regarding both global convergence of the optimization and local confidence bounds over the recovered density parameters.

4.2 Related work

Our analysis is inspired by literature in two domains: optimization and compressed sensing. From optimization we draw tools for convergence analysis over our nonconvex objective, and from compressed sensing we work towards an understanding of volume recovery in the absence of full measurements, given some sparsity assumptions of the true volume.

We draw on a rich literature of convex and nonconvex optimization [BV11; NW06a; WR22], focusing on two questions: (1) does the optimization converge to the global minimum of the objective, and (2) how quickly? We are specifically interested in high-dimensional optimization, because we wish to operate at high resolution and in 3 or 4 dimensions, leading to millions of optimizable parameters. This high dimensionality makes more efficient second-order optimization algorithms intractable; in our analysis here we consider purely first-order stochastic gradient descent, though in practice in Chapter 2 and Chapter 3 we use slightly more expensive methods RMSProp [Hin] and Adam [KB17], respectively. These adaptive methods require storing memory equivalent to double (for RMSProp) or triple (for Adam) the size of the gradient, and are two of many available schemes for accelerated first-order optimization [Yu 83; NW06a; Noc80].

We are similarly inspired by literature in compressed sensing [CRT06; Vid20; WM22], which studies optimization in the specific context of recovering a signal from insufficient measurements, given some prior knowledge of the signal. Compressed sensing typically requires linear and incoherent measurements [DH01], such as Fourier-domain samples

[CRT06], and common priors include sparsity of the signal itself, its gradient, or its representation in some known domain (*e.g.* wavelets). These priors are well-motivated for several inverse problems, notably the linear tomography problem and the magnetic resonance imaging reconstruction problem, which both involve under-sampling in the Fourier domain (either directly for MRI, or via the Fourier slice theorem in tomography [Nis10]). Some results exist for certain classes of nonlinear measurements, as long as these measurements are not *too* nonlinear [OS17]; our contribution continues this line of research.

4.3 Problem formulation

In Chapter 2 and Chapter 3, we optimize the following objective function:

$$\mathcal{L}(\boldsymbol{c}, \sigma) = \frac{1}{2}\mathbb{E}_{\boldsymbol{r}}\left[\left\|\hat{C}(\boldsymbol{r}; \boldsymbol{c}, \sigma) - C(\boldsymbol{r}; \boldsymbol{c}_*, \sigma_*)\right\|_{\ell_2}^2\right] + \mathcal{R}(\boldsymbol{c}, \sigma), \tag{4.3.1}$$

where C denotes color of ray $\boldsymbol{r}$, and $\mathcal{R}$ is a regularizer applied to the grid of color and density values $\boldsymbol{c}$ and σ. Here we use $\hat{C}$ to denote our predicted ray color using the current model of density and color, and we use C to denote the true color as measured in the training images, based on the optimal color and density values $\boldsymbol{c}_*$ and σ_*.

This model is faithful to the one used in practice in the preceding chapters, with the following modifications:

1. In Chapter 2, $\mathcal{R}$ is total variation applied in 3D over the color and density parameters.
2. In Chapter 3, $\mathcal{R}$ includes both total variation in the spatial dimensions of each plane, and smoothness (norm of second derivative) over the time dimension of the space-time planes.
3. In Chapter 2 we model σ directly, but we model $\boldsymbol{c}$ using spherical harmonics so that color may be view-dependent.
4. In Chapter 3, we model latent features that are decoded into $\boldsymbol{c}$ and σ using either a linear (dot product) decoder or an MLP decoder. We also represent these features in planes that are multiplied together into the full volume.
5. There are additional practical constraints that $\sigma \geq 0$ and $\boldsymbol{c} \in [0, 1]$; these are implemented using nonlinear activations (exp for density and either clipping or sigmoid for color). For simplicity, these constraints are not modeled at present; in future they could be treated as optimization constraints, or the nonlinear activations could be incorporated into the measurement model.

An ideal theoretical analysis would use exactly the same model that is used experimentally in practice. However, as a first step of analysis we assume that $\boldsymbol{c}$ and σ are

represented directly, which matches the experimental reality of Chapter 2 in the case of using zero-order spherical harmonics for color (*i.e.*, no view-dependence). We begin by ignoring the regularization term $\mathcal{R}$ and assuming access to infinite measurement rays (except in Sec. 4.6).

4.4 Measurement models

In Chapter 2 and Chapter 3, we use the following measurement model for the color $\hat{C}(\mathbf{r})$ of a ray $\mathbf{r}$:

$$\hat{C}(\mathbf{r}) = \sum_{i=0}^{N-1} \exp\left(-\sum_{j=0}^{i-1} \sigma_j \delta_j\right) \left(1 - \exp(-\sigma_i \delta_i)\right) \mathbf{c}_i, \tag{4.4.1}$$

where σ_i is the optical density of a sample i along the ray, $\mathbf{c}_i$ is its color vector, and δ_i denotes the distance between sample i and the sample behind it, with N total samples distributed along the ray through the volume of interest. This model is due to Max [Max95] as applied in Neural Radiance Fields (NeRF) [MST+20]; it is derived as a discretized approximation of the Beer-Lambert Law, which governs the attenuation of light through absorptive media. Note that this model is nonlinear in the parameters σ_i and linear in the parameters $\mathbf{c}_i$.

It will also be convenient to consider an alternate formulation of the same measurement model, using matrix notation. We begin by rewriting the sum in a more convenient manner:

$$\begin{aligned}
\hat{C}(\mathbf{r}) &= \sum_{i=0}^{N-1} \exp\left(-\sum_{j=0}^{i-1} \sigma_j \delta_j\right) \left(1 - \exp(-\sigma_i \delta_i)\right) \mathbf{c}_i && (4.4.2)\\
&= (1 - \exp(-\sigma_0\delta_0))\mathbf{c}_0 + \exp(-\sigma_0\delta_0)(1 - \exp(-\sigma_1\delta_1))\mathbf{c}_1 && (4.4.3)\\
&\quad + \exp(-\sigma_0\delta_0 - \sigma_1\delta_1)(1 - \exp(-\sigma_2\delta_2))\mathbf{c}_2 && (4.4.4)\\
&\quad + \exp(-\sigma_0\delta_0 - \sigma_1\delta_1 - \sigma_2\delta_2)(1 - \exp(-\sigma_3\delta_3))\mathbf{c}_3 + \ldots && (4.4.5)\\
&= \mathbf{c}_0 + \exp(-\sigma_0\delta_0)(\mathbf{c}_1 - \mathbf{c}_0) + \exp(-\sigma_0\delta_0 - \sigma_1\delta_1)(\mathbf{c}_2 - \mathbf{c}_1) + \ldots && (4.4.6)\\
&= \mathbf{c}_0 - \exp\left(-\sum_{i=0}^{N-1} \sigma_i \delta_i\right) \mathbf{c}_{N-1} + \sum_{i=0}^{N-2} \exp\left(-\sum_{j=0}^{i} \sigma_j \delta_j\right) (\mathbf{c}_{i+1} - \mathbf{c}_i). && (4.4.7)
\end{aligned}$$

We can now rewrite this measurement model using matrices:

$$\hat{C}(\boldsymbol{r}) = \langle \boldsymbol{\Delta}\mathbf{c}(\boldsymbol{r}), \exp(-\mathbf{S}\boldsymbol{\sigma}(\boldsymbol{r}) \rangle \tag{4.4.8}$$

where the matrices $\boldsymbol{\Delta}$, $\mathbf{c}(\boldsymbol{r})$, and $\mathbf{S}$, and the vector $\boldsymbol{\sigma}(\boldsymbol{r})$ are defined as follows, using an example where we only take 4 samples along the ray $\boldsymbol{r}$ (numbered 0 through 3).

$$\boldsymbol{\Delta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad \mathbf{c}(\boldsymbol{r}) = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \delta_0 & 0 & 0 & 0 \\ \delta_0 & \delta_1 & 0 & 0 \\ \delta_0 & \delta_1 & \delta_2 & 0 \\ \delta_0 & \delta_1 & \delta_2 & \delta_3 \end{bmatrix}, \quad \boldsymbol{\sigma}(\boldsymbol{r}) = \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \sigma_3 \end{bmatrix}$$

We use $\boldsymbol{\Delta}$ to denote a discrete difference convolution of the color samples $\mathbf{c}(\boldsymbol{r})$, and $\mathbf{S}$ to denote a cumsum of the density samples $\boldsymbol{\sigma}(\boldsymbol{r})$.

Globally, we have a shared $\boldsymbol{\sigma}$ and a shared $\mathbf{c}$ for the volume, and each ray is measuring a subset of these values $\boldsymbol{\sigma}(\boldsymbol{r})$, $\mathbf{c}(\boldsymbol{r})$. Using $\boldsymbol{M_r}$ to denote a mask matrix associated to ray $\boldsymbol{r}$, we can write $\mathbf{c}(\boldsymbol{r}) = \boldsymbol{M_r}\mathbf{c}$ and $\boldsymbol{\sigma}(\boldsymbol{r}) = \boldsymbol{M_r}\boldsymbol{\sigma}$, leading us to the measurement model:

$$\hat{C}(\boldsymbol{r}) = \langle \boldsymbol{\Delta}\boldsymbol{M_r}\mathbf{c}, \exp(-\mathbf{S}\boldsymbol{M_r}\boldsymbol{\sigma}) \rangle, \tag{4.4.9}$$

where the optimization variables $\mathbf{c}$ and $\boldsymbol{\sigma}$ are very tall (*e.g.* d^3, where $d \geq 256$ is the resolution of a volume in each dimension). The mask $\boldsymbol{M_r}$ is a short matrix, with number of rows no more than a small constant times d, because only a small portion of the volume contributes to the color of a given ray. The masks are also very structured, always choosing entries that are contiguous (following a ray) in the un-vectorized volume represented by $\mathbf{c}$ and $\boldsymbol{\sigma}$.

The randomness in this measurement model comes from the camera positioning – we have a set of random camera coordinates and viewing directions, and each camera captures a set of measurement rays (order d^2). Rays (and their corresponding masks) from the same camera are highly correlated, but rays from different cameras are independent. In practice we would like to use as few rays as necessary, and we can assume that the volume – the $\boldsymbol{c}_*$ and $\boldsymbol{\sigma}_*$ values of the ground truth – is somewhat sparse and very gradient sparse.

The only nonlinearities in the model are the elementwise exponential around $\mathbf{S}\boldsymbol{M_r}\boldsymbol{\sigma}$, and the inner product involving $\boldsymbol{c}$ and $\boldsymbol{\sigma}$. These nonlinearities result in the loss function $\mathcal{L}(\boldsymbol{c}, \boldsymbol{\sigma})$ being nonconvex.

4.4.1 Nonlinear tomography

The tomographic setting is nearly identical to our primary problem of photorealistic recovery, except with two key changes: (1) There is no notion of color, as a single wavelength is typically used, and (2) The distribution of optical densities σ_i is shifted to smaller values, since typically a higher-energy wavelength (such as X-ray) is chosen so that most materials are at least somewhat transmissive. With color removed, the measurement model

simplifies to:

$$\hat{\alpha}(\mathbf{r}) = \sum_{i=0}^{N-1} \exp\left(-\sum_{j=0}^{i-1} \sigma_j \delta_j\right)\left(1 - \exp(-\sigma_i \delta_i)\right) \tag{4.4.10}$$

$$= 1 - \exp(-\sigma_0\delta_0) + \exp(-\sigma_0\delta_0)(1 - \exp(-\sigma_1\delta_1)) \tag{4.4.11}$$

$$+ \exp(-\sigma_0\delta_0 - \sigma_1\delta_1)(1 - \exp(-\sigma_2\delta_2)) \tag{4.4.12}$$

$$+ \exp(-\sigma_0\delta_0 - \sigma_1\delta_1 - \sigma_2\delta_2)(1 - \exp(-\sigma_3\delta_3)) + \ldots \tag{4.4.13}$$

$$= 1 - \exp\left(-\sum_{i=0}^{N-1} \sigma_i \delta_i\right), \tag{4.4.14}$$

where $\hat{\alpha}(\mathbf{r})$ denotes the fraction of incident light absorbed along ray $\mathbf{r}$. This formula may be derived either by this method, in which we simplify Eq. (4.4.1) by treating color $\mathbf{c}_i$ as constant unity and telescoping, or alternatively by treating the volume of interest as purely absorptive and directly applying the corresponding component of the discretized Beer-Lambert law [Max95].

We observe that this tomographic measurement model is still nonlinear in the remaining parameters σ_i. However, in this form it is a composition of a nonlinear function $f(\cdot) = 1 - \exp(-\cdot)$ and a linear function (the weighted sum of sample densities along the ray). We can even simplify things further by assuming that $\delta_i = \delta \ \ \forall i$, and thereby wrap δ_i into σ_i, if we choose regularly spaced samples. Following this simplification, we arrive at the tomographic measurement model:

$$\hat{\alpha}(\mathbf{r}) = 1 - \exp\left(-\sum_{i=0}^{N-1} \sigma_i\right). \tag{4.4.15}$$

Experimentally, when we are optimizing to fit this density-only objective, we can invert the nonlinearity and produce a linear problem. This linearization step is common in practice for tomographic reconstruction, although numerical stability issues arise for values of $\alpha \approx 1$, corresponding to optically dense materials. Practically, these appear as artifacts around metal objects in X-ray CT, often taking the form of spurious streaks or highlights around the perimeter of the metallic or absorptive region. Alternatively, we propose nonconvex optimization directly through Eq. (4.4.15), which avoids the numerical instability of linearization in the presence of absorptive materials.

We can also write this tomographic measurement model in matrix form, using now a mask vector $\boldsymbol{m}_r$ associated to each ray $\boldsymbol{r}$:

$$\hat{\alpha}(\mathbf{r}) = 1 - \exp\left(-\boldsymbol{m}_r^T \boldsymbol{\sigma}\right). \tag{4.4.16}$$

4.4.2 Gaussian approximations

For ease of analysis, we approximate the measurement models in both the photorealistic reconstruction problem and the nonlinear tomography problem using random Gaussian measurement matrices, as described below. These approximations are motivated by three steps of simplification: (1) linear projection measurements are equivalent to radial samples of the Fourier domain, according to the Fourier slice theorem [Nis10], (2) structured, multidimensional Fourier domain measurements may be well-approximated by random one-dimensional Fourier domain measurements [CRT06], and (3) random Fourier measurements may be well-approximated by random Gaussian measurements [ORS18].

For the photorealistic reconstruction problem, we approximate the measurement model in Eq. (4.4.9) using standard Gaussian matrices $\boldsymbol{B}$ and $\boldsymbol{A}$ to approximate $\boldsymbol{\Delta M_r}$ and $-\mathbf{S}\boldsymbol{M_r}$:

$$\hat{C}(\boldsymbol{r}) = \langle \boldsymbol{Bc}, \exp(\boldsymbol{A\sigma}) \rangle \tag{4.4.17}$$

where $\boldsymbol{B}$ and $\boldsymbol{A}$ are $\in \mathbb{R}^{N \times n}$, where N is the number of samples per ray and $n \gg N$ is the number of voxels we want to recover, so $\boldsymbol{c}, \boldsymbol{\sigma} \in \mathbb{R}^n$. Note that this entails a simplification of color to be scalar per voxel, corresponding to a single wavelength of light; in practice the analysis could be repeated over multiple wavelengths *e.g.* for the typical red, green, and blue (RGB) color channels. We assume that $\boldsymbol{B}$ and $\boldsymbol{A}$ are standard Gaussian matrices sampled independently of each other and independently for each ray $\boldsymbol{r}$.

For the tomography (no-color) problem, we make a similar approximation by replacing the negated mask vector $-\boldsymbol{m_r}$ in Eq. (4.4.16) with a standard Gaussian vector $\boldsymbol{a} \in \mathbb{R}^n$:

$$\hat{\alpha}(\mathbf{r}) = 1 - \exp\left(\boldsymbol{a}^T \boldsymbol{\sigma}\right) \tag{4.4.18}$$

where $\boldsymbol{a}$ is again sampled independently for each ray $\boldsymbol{r}$.

4.5 Convergence analysis

In this section, we take first steps toward theoretical analysis of the photorealistic reconstruction problem defined by Eq. (4.4.9) and the nonlinear tomographic reconstruction problem defined by Eq. (4.4.16). To make analysis more tractable, we use the Gaussian approximations defined by Eq. (4.4.17) and Eq. (4.4.18) as our measurement models for these problems, respectively.

In this section, we analyze the optimization of $\mathcal{L}$ in the *population* setting, where we have access to unlimited ray measurements and we optimize data fidelity alone, without regularization. In this setting, we show that a special case of gradient descent converges almost immediately to the exact solution, for both the photorealistic reconstruction problem and the nonlinear tomography problem. These population convergence results are encouraging and nontrivial given the nonconvex objective function, though in practice we wish to optimize using as few measurements as possible. We therefore provide a proof

sketch of the global convergence argument in the more practically-relevant finite-sample regime, though the full proof in this case is left to future work.

4.5.1 Photorealistic reconstruction: population convergence

Using the Gaussian measurement model of Eq. (4.4.17), we can analyze the first step of gradient descent on $\mathcal{L}(\boldsymbol{c}, \boldsymbol{\sigma})$, without regularization, in expectation. Let $\boldsymbol{c}$ be our reconstruction of $\boldsymbol{c}_*$ and $\boldsymbol{\sigma}$ be our reconstruction of $\boldsymbol{\sigma}_*$. We begin by initializing $\boldsymbol{c} = \mathbf{0}_n$ and $\boldsymbol{\sigma} = \mathbf{0}_n$. Our first gradient descent step will be to update $\boldsymbol{c} = \mathbf{0}_n - t_c \mathcal{L}(\boldsymbol{c} = \mathbf{0}_n, \boldsymbol{\sigma} = \mathbf{0}_n)$:

$$\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\nabla_{\boldsymbol{c}} \mathcal{L}(\boldsymbol{\sigma} = \mathbf{0}_n, \boldsymbol{c} = \mathbf{0}_n)] \overset{(a)}{=} \mathbb{E}_{\boldsymbol{A},\boldsymbol{B}} \left[\left(\boldsymbol{c}^T \boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}) - \boldsymbol{c}_*^T \boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}_*) \right) \boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}) \right] \tag{4.5.1}$$

$$\overset{(b)}{=} -\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}_*) \boldsymbol{B}^T \mathbf{1}_N] \tag{4.5.2}$$

$$\overset{(c)}{=} -\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \, \mathbb{E}_{\boldsymbol{A}}[\exp(\boldsymbol{A}\boldsymbol{\sigma}_*)] \boldsymbol{B}^T \mathbf{1}_N] \tag{4.5.3}$$

$$\overset{(d)}{=} -\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \, \mathbb{E}_{\boldsymbol{g}_N}[\exp(\boldsymbol{g}_N \, \|\boldsymbol{\sigma}_*\|_{\ell_2})] \boldsymbol{B}^T \mathbf{1}_N] \tag{4.5.4}$$

$$\overset{(e)}{=} -\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \mathbf{1}_N \, \mathbb{E}_{g}[\exp(g \, \|\boldsymbol{\sigma}_*\|_{\ell_2})] \boldsymbol{B}^T \mathbf{1}_N] \tag{4.5.5}$$

$$\overset{(f)}{=} -\exp\left(\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2 \right) \mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \mathbf{1}_N \boldsymbol{B}^T \mathbf{1}_N] \tag{4.5.6}$$

$$\overset{(g)}{=} -N \exp\left(\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2 \right) \mathbb{E}_{\boldsymbol{g}_n}[\boldsymbol{c}_*^T \boldsymbol{g}_n \boldsymbol{g}_n] \tag{4.5.7}$$

$$\overset{(h)}{=} -N \exp\left(\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2 \right) \mathbb{E}_{\boldsymbol{g}_n}[\boldsymbol{g}_n \boldsymbol{g}_n^T] \boldsymbol{c}_* \tag{4.5.8}$$

$$\overset{(i)}{=} -N \exp\left(\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2 \right) \boldsymbol{c}_*. \tag{4.5.9}$$

In (a) we evaluate the gradient of the loss with respect to $\boldsymbol{c}$. In (b) we plug in the initialization values $\boldsymbol{c} = \mathbf{0}_n$ and $\boldsymbol{\sigma} = \mathbf{0}_n$. In (c) we pull the expectation over $\boldsymbol{A}$ inside. In (d) we simplify the inner expectation by replacing $\boldsymbol{A}\boldsymbol{\sigma}_*$ with $\boldsymbol{g}_N \, \|\boldsymbol{\sigma}_*\|_{\ell_2}$, where $\boldsymbol{g}_N \in \mathbb{R}^N$ is a standard Gaussian vector. In (e) we further simplify the inner expectation to one over a scalar Guassian g, since each entry in the original vector has the same expectation. In (f) we evaluate this inner Gaussian expectation. In (g) we simplify the remaining expectation by replacing $\boldsymbol{B}^T \mathbf{1}_N$ with $\boldsymbol{g}_n \sqrt{N}$, where $\boldsymbol{g}_n \in \mathbb{R}^n$ is a standard Gaussian vector. In (h) we note that $\boldsymbol{c}_*^T \boldsymbol{g}_n$ is a scalar, so we can transpose it and move it to the right side of the expectation, allowing us to pull $\boldsymbol{c}_*$ outside the expectation. In (i) we evaluate the final expectation to the identity (of size n).

Using gradient descent from this all-zeros initialization, the expected first iterate of $\boldsymbol{c}$ is

$$\mathbb{E}[\boldsymbol{c}_1] = \mathbf{0}_n - t_c \, \mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\nabla_{\boldsymbol{c}} \mathcal{L}(\boldsymbol{\sigma} = \mathbf{0}_n, \boldsymbol{c} = \mathbf{0}_n)] \tag{4.5.10}$$

$$= t_c N \exp\left(\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2\right) \boldsymbol{c}_*, \tag{4.5.11}$$

where t_c is the step size in $\boldsymbol{c}$, to be selected carefully. From this expression, we can see that our expected first iterate of $\boldsymbol{c}$ is well aligned with the true value $\boldsymbol{c}_*$, and can be set exactly equal by careful choice of t_c with knowledge of $\|\boldsymbol{\sigma}_*\|_{\ell_2}$. In particular, we should choose:

$$t_c = N^{-1} \exp\left(-\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2\right), \tag{4.5.12}$$

so that $\mathbb{E}[\boldsymbol{c}_1] = \boldsymbol{c}_*$.

Using this expected first iterate $\boldsymbol{c}_1$, we now compute the expected first step for $\boldsymbol{\sigma}_1 = \mathbf{0}_n - t_\sigma \, \mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\nabla_{\boldsymbol{\sigma}} \mathcal{L}(\boldsymbol{\sigma} = \mathbf{0}_n, \boldsymbol{c} = \mathbb{E}[\boldsymbol{c}_1])]$.

$$\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\nabla_{\boldsymbol{\sigma}} \mathcal{L}(\boldsymbol{\sigma} = \mathbf{0}_n, \boldsymbol{c} = \mathbb{E}[\boldsymbol{c}_1])] \tag{4.5.13}$$

$$\overset{(a)}{=} \mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}\left[\left(\mathbb{E}[\boldsymbol{c}_1]^T \boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}) - \boldsymbol{c}_*^T \boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}_*)\right) \boldsymbol{A}^T \operatorname{diag}\left(\exp(\boldsymbol{A}\boldsymbol{\sigma})\right) \boldsymbol{B}\, \mathbb{E}[\boldsymbol{c}_1]\right] \tag{4.5.14}$$

$$\overset{(b)}{=} \mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}\left[\left(\mathbb{E}[\boldsymbol{c}_1]^T \boldsymbol{B}^T \mathbf{1}_N - \boldsymbol{c}_*^T \boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}_*)\right) \boldsymbol{A}^T \boldsymbol{B}\, \mathbb{E}[\boldsymbol{c}_1]\right] \tag{4.5.15}$$

$$\overset{(c)}{=} -\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}_*) \boldsymbol{A}^T \boldsymbol{B}\, \mathbb{E}[\boldsymbol{c}_1]] \tag{4.5.16}$$

$$\overset{(d)}{=} -\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \, \mathbb{E}_{\boldsymbol{A}}[\exp(\boldsymbol{A}\boldsymbol{\sigma}_*) \frac{\boldsymbol{\sigma}_* \boldsymbol{\sigma}_*^T}{\|\boldsymbol{\sigma}_*\|_{\ell_2}^2} \boldsymbol{A}^T] \boldsymbol{B}\, \mathbb{E}[\boldsymbol{c}_1]] \tag{4.5.17}$$

$$\overset{(e)}{=} -\mathbb{E}_{\boldsymbol{B},\boldsymbol{g}_N}[\boldsymbol{c}_*^T \boldsymbol{B}^T \exp(\boldsymbol{g}_N \|\boldsymbol{\sigma}_*\|_{\ell_2}) \frac{\boldsymbol{\sigma}_*}{\|\boldsymbol{\sigma}_*\|_{\ell_2}} \boldsymbol{g}_N^T \boldsymbol{B}\, \mathbb{E}[\boldsymbol{c}_1]] \tag{4.5.18}$$

$$\overset{(f)}{=} -\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \, \mathbb{E}_{\boldsymbol{g}_N}[\exp(\boldsymbol{g}_N \|\boldsymbol{\sigma}_*\|_{\ell_2}) \boldsymbol{g}_N^T] \boldsymbol{B}\, \mathbb{E}[\boldsymbol{c}_1]] \frac{\boldsymbol{\sigma}_*}{\|\boldsymbol{\sigma}_*\|_{\ell_2}} \tag{4.5.19}$$

$$\overset{(g)}{=} -\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \, \mathbb{E}_{g}[\exp(g \|\boldsymbol{\sigma}_*\|_{\ell_2}) g] \boldsymbol{B}\, \mathbb{E}[\boldsymbol{c}_1]] \frac{\boldsymbol{\sigma}_*}{\|\boldsymbol{\sigma}_*\|_{\ell_2}} \tag{4.5.20}$$

$$\overset{(h)}{=} -\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \, \|\boldsymbol{\sigma}_*\|_{\ell_2} \, \mathbb{E}_{g}[\exp(g \|\boldsymbol{\sigma}_*\|_{\ell_2})] \boldsymbol{B}\, \mathbb{E}[\boldsymbol{c}_1]] \frac{\boldsymbol{\sigma}_*}{\|\boldsymbol{\sigma}_*\|_{\ell_2}} \tag{4.5.21}$$

$$\overset{(i)}{=} -\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T \boldsymbol{B}^T \boldsymbol{B}\, \mathbb{E}[\boldsymbol{c}_1]] \exp\left(\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2\right) \boldsymbol{\sigma}_* \tag{4.5.22}$$

$$\overset{(j)}{=} -N \boldsymbol{c}_*^T \, \mathbb{E}[\boldsymbol{c}_1] \exp\left(\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2\right) \boldsymbol{\sigma}_* \tag{4.5.23}$$

$$\overset{(k)}{=} -t_c N^2 \|\boldsymbol{c}_*\|_{\ell_2}^2 \exp(\|\boldsymbol{\sigma}_*\|_{\ell_2}^2) \boldsymbol{\sigma}_*. \tag{4.5.24}$$

In (a) we evaluate the gradient of the loss with respect to $\boldsymbol{\sigma}$. In (b) we plug in the initialization value $\boldsymbol{\sigma} = \mathbf{0}_n$. In (c) we use linearity of expectation and evaluate the first term to zero because $\boldsymbol{A}$ only appears once in it and $\boldsymbol{A}$ has mean zero. In (d) we separate $\boldsymbol{A}$ into components parallel and orthogonal to $\boldsymbol{\sigma}_*$, and evaluate the expectation of the orthogonal component to zero. In (e) we replace $\boldsymbol{A}\boldsymbol{\sigma}_*$ with $\boldsymbol{g}_N \|\boldsymbol{\sigma}_*\|_{\ell_2}$, where $\boldsymbol{g}_N \in \mathbb{R}^N$ is a standard Gaussian vector, as these have the same distribution. In (f) we pull out $\frac{\boldsymbol{\sigma}_*}{\|\boldsymbol{\sigma}_*\|_{\ell_2}}$ by noticing that the terms before and after it are each scalar; we then pull the expectation over $\boldsymbol{g}_N$ inside. In (g) we notice that the inner expectation is a multiple of the identity matrix, so we can rewrite it as an expectation over a scalar Gaussian g. In (h) we simplify this scalar inner expectation using Stein. In (i) we evaluate the inner expectation. In (k) we plug in the value of $\mathbb{E}[\boldsymbol{c}_1]$.

Using gradient descent with this expected loss for $\boldsymbol{\sigma}$, the expected first iterate of $\boldsymbol{\sigma}$ (after taking an expected step in $\boldsymbol{c}$) is:

$$\mathbb{E}[\boldsymbol{\sigma}_1] = \mathbf{0}_n - t_\sigma \, \mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\nabla_{\boldsymbol{\sigma}} \mathcal{L}(\boldsymbol{\sigma} = \mathbf{0}_n, \boldsymbol{c} = \mathbb{E}[\boldsymbol{c}_1])] \tag{4.5.25}$$

$$= t_\sigma t_c N^2 \, \|\boldsymbol{c}_*\|_{\ell_2}^2 \exp(\|\boldsymbol{\sigma}_*\|_{\ell_2}^2) \boldsymbol{\sigma}_*, \tag{4.5.26}$$

where t_σ is the step size in $\boldsymbol{\sigma}$, to be chosen carefully. From this expression, we can see that our expected first iterate of $\boldsymbol{\sigma}$ is well aligned with the true value $\boldsymbol{\sigma}_*$, and can be set exactly equal by careful choice of t_σ with knowledge of $\|\boldsymbol{\sigma}_*\|_{\ell_2}$ and $\|\boldsymbol{c}_*\|_{\ell_2}$. In particular, we should choose:

$$t_\sigma = N^{-1} \exp\left(-\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2\right) \|\boldsymbol{c}_*\|_{\ell_2}^{-2}, \tag{4.5.27}$$

so that $\mathbb{E}[\boldsymbol{\sigma}_1] = \boldsymbol{\sigma}_*$.

Now that we know how to select appropriate step sizes, we must first recover $\|\boldsymbol{\sigma}_*\|_{\ell_2}$ and $\|\boldsymbol{c}_*\|_{\ell_2}$ to be able to compute them. We do this by separately evaluating $\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\boldsymbol{B}\hat{C}]$ and $\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\hat{C}\boldsymbol{A}^T\boldsymbol{B}]$ to produce two equations with two unknowns $\|\boldsymbol{\sigma}_*\|_{\ell_2}$ and $\|\boldsymbol{c}_*\|_{\ell_2}$, from which

we can solve for these values.

$$\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\boldsymbol{B}\hat{\boldsymbol{C}}] \overset{(a)}{=} \mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\boldsymbol{B}\boldsymbol{c}_*^T\boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}_*)] \tag{4.5.28}$$

$$\overset{(b)}{=} \mathbb{E}_{\boldsymbol{B}}[\boldsymbol{B}\boldsymbol{c}_*^T\boldsymbol{B}^T]\,\mathbb{E}_{\boldsymbol{A}}[\exp(\boldsymbol{A}\boldsymbol{\sigma}_*)] \tag{4.5.29}$$

$$\overset{(c)}{=} \mathbb{E}_{\boldsymbol{B}}[\boldsymbol{B}\boldsymbol{c}_*^T\boldsymbol{B}^T]\mathbf{1}_N \exp\left(\frac{1}{2}\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}^2\right) \tag{4.5.30}$$

$$\overset{(d)}{=} \mathbb{E}_{\boldsymbol{B}}[\boldsymbol{B}\frac{\boldsymbol{c}_*\boldsymbol{c}_*^T}{\left\|\boldsymbol{c}_*\right\|_{\ell_2}^2}\boldsymbol{c}_*^T\boldsymbol{B}^T]\mathbf{1}_N \exp\left(\frac{1}{2}\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}^2\right) \tag{4.5.31}$$

$$\overset{(e)}{=} \mathbb{E}_{\boldsymbol{g}_N}[\boldsymbol{g}_N\boldsymbol{c}_*^T\boldsymbol{g}_N^T\mathbf{1}_N] \exp\left(\frac{1}{2}\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}^2\right) \tag{4.5.32}$$

$$\overset{(f)}{=} \mathbb{E}_{\boldsymbol{g}_N}[\boldsymbol{g}_N\boldsymbol{g}_N^T\mathbf{1}_N]\boldsymbol{c}_*^T \exp\left(\frac{1}{2}\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}^2\right) \tag{4.5.33}$$

$$\overset{(g)}{=} \mathbf{1}_N\boldsymbol{c}_*^T \exp\left(\frac{1}{2}\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}^2\right). \tag{4.5.34}$$

In (a) we plug in our measurement model for $\hat{\boldsymbol{C}}$. In (b) we separate the independent expectations. In (c) we evaluate the second expectation, by rewriting it as a Gaussian vector. In (d) we separate $\boldsymbol{B}$ into components parallel and orthogonal to $\boldsymbol{c}_*$, and evaluate the expectation of the orthogonal component to zero. In (e) we replace $\boldsymbol{B}\boldsymbol{c}_*$ with $\boldsymbol{g}_N \left\|\boldsymbol{c}_*\right\|_{\ell_2}$, where $\boldsymbol{g}_N \in \mathbb{R}^N$ is a standard Gaussian vector, as these have the same distribution. We also bring the $\mathbf{1}_N$ inside the expectation. In (f), we note that $\boldsymbol{g}_N^T\mathbf{1}_N$ is scalar, so we move it before the $\boldsymbol{c}_*^T$ and pull that outside the expectation. In (g) we evaluate the expectation over $\boldsymbol{g}_N$. The result is an $N \times n$ matrix whose Frobenius norm is $\sqrt{N}\left\|\boldsymbol{c}_*\right\|_{\ell_2} \exp\left(\frac{1}{2}\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}^2\right)$.

$$\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\hat{\boldsymbol{C}}\boldsymbol{A}^T\boldsymbol{B}] \overset{(a)}{=} \mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\boldsymbol{c}_*^T\boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}_*)\boldsymbol{A}^T\boldsymbol{B}] \tag{4.5.35}$$

$$\overset{(b)}{=} \mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\boldsymbol{c}_*^T\boldsymbol{B}^T \exp(\boldsymbol{A}\boldsymbol{\sigma}_*)\frac{\boldsymbol{\sigma}_*\boldsymbol{\sigma}_*^T}{\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}^2}\boldsymbol{A}^T\boldsymbol{B}] \tag{4.5.36}$$

$$\overset{(c)}{=} \mathbb{E}_{\boldsymbol{g}_N,\boldsymbol{B}}[\boldsymbol{c}_*^T\boldsymbol{B}^T \exp(\boldsymbol{g}_N\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2})\frac{\boldsymbol{\sigma}_*}{\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}}\boldsymbol{g}_N^T\boldsymbol{B}] \tag{4.5.37}$$

$$\overset{(d)}{=} \frac{\boldsymbol{\sigma}_*}{\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}}\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T\boldsymbol{B}^T\,\mathbb{E}_{\boldsymbol{g}_N}[\exp(\boldsymbol{g}_N\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2})\boldsymbol{g}_N^T]\boldsymbol{B}] \tag{4.5.38}$$

$$\overset{(e)}{=} \boldsymbol{\sigma}_*\,\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{c}_*^T\boldsymbol{B}^T\boldsymbol{B}] \exp\left(\frac{1}{2}\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}^2\right) \tag{4.5.39}$$

$$\overset{(f)}{=} \boldsymbol{\sigma}_*\boldsymbol{c}_*^T N \exp\left(\frac{1}{2}\left\|\boldsymbol{\sigma}_*\right\|_{\ell_2}^2\right). \tag{4.5.40}$$

In (a) we plug in our measurement model for $\hat{C}$. In (b) we separate $\boldsymbol{A}$ into components parallel and orthogonal to $\boldsymbol{\sigma}_*$, and evaluate the expectation of the orthogonal component to zero. In (c) we replace $\boldsymbol{A}\boldsymbol{\sigma}_*$ with $\boldsymbol{g}_N \|\boldsymbol{\sigma}_*\|_{\ell_2}$, where $\boldsymbol{g}_N \in \mathbb{R}^N$ is a standard Gaussian vector, as these have the same distribution. In (d) we notice that the term in front of $\frac{\boldsymbol{\sigma}_*}{\|\boldsymbol{\sigma}_*\|_{\ell_2}}$ is scalar, so we can pull this vector outside the expectation, and move the expectation over $\boldsymbol{g}_N$ inside. In (e) we evaluate this inner expectation with respect to $\boldsymbol{g}_N$ by first noticing that the result is a multiple of the identity matrix, so we can rewrite it as an expectation over a scalar Gaussian g, which we evaluate using Stein. In (f) we evaluate the remaining expectation over $\boldsymbol{B}$. The result is an $n \times n$ matrix whose Frobenius norm is $\|\boldsymbol{\sigma}_*\|_{\ell_2} \|\boldsymbol{c}_*\|_{\ell_2} N \exp\left(\frac{1}{2}\|\boldsymbol{\sigma}_*\|_{\ell_2}^2\right)$.

Finally, we can use these two quantities to recover $\|\boldsymbol{\sigma}_*\|_{\ell_2}$ and $\|\boldsymbol{c}_*\|_{\ell_2}$. Let $p = \left\|\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\hat{C}\boldsymbol{A}^T\boldsymbol{B}]\right\|_F$ and $q = \left\|E_{\boldsymbol{A},\boldsymbol{B}}[\boldsymbol{B}\hat{C}]\right\|_F$. Then:

$$\|\boldsymbol{\sigma}_*\|_{\ell_2} = \frac{p}{q\sqrt{N}} \tag{4.5.41}$$

$$\|\boldsymbol{c}_*\|_{\ell_2} = \frac{q}{\sqrt{N}} \exp\left(-\frac{1}{2}\|\boldsymbol{\sigma}_*\|_{\ell_2}^2\right). \tag{4.5.42}$$

In the population setting, our optimization procedure is: (1) recover $\|\boldsymbol{c}_*\|_{\ell_2}$ and $\|\boldsymbol{\sigma}_*\|_{\ell_2}$ by measuring $\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\boldsymbol{B}\hat{C}]$ and $\mathbb{E}_{\boldsymbol{A},\boldsymbol{B}}[\hat{C}\boldsymbol{A}^T\boldsymbol{B}]$ and following Eq. (4.5.41), (2) Starting from the all-zeros initialization for both $\boldsymbol{c}$ and $\boldsymbol{\sigma}$, update $\boldsymbol{c}$ following the negative expected gradient in $\boldsymbol{c}$, with the step size defined by Eq. (4.5.12), (3) From this updated value for $\boldsymbol{c}$, and the initialization $\boldsymbol{\sigma} = \boldsymbol{0}_n$, update $\boldsymbol{\sigma}$ following the negative expected gradient in $\boldsymbol{\sigma}$, with the step size defined by Eq. (4.5.27). After these steps, we arrive at $\boldsymbol{c} = \boldsymbol{c}_*$ and $\boldsymbol{\sigma} = \boldsymbol{\sigma}_*$.

Note that numerical issues may arise in the special cases when $\|\boldsymbol{\sigma}_*\|_{\ell_2} \to \infty$ or $\|\boldsymbol{c}_*\|_{\ell_2} \to 0$; these cases correspond respectively to scenes with infinite optical density (full occlusion), and no illumination (total darkness). It is not surprising that reconstruction quality suffers under these circumstances.

4.5.2 Nonlinear tomography: population convergence

Using the Gaussian measurement model of Eq. (4.4.18), we can analyze the first step of gradient descent on $\mathcal{L}(\boldsymbol{\sigma})$, without regularization, in expectation.

The gradient of our loss function evaluated at a random ray $\boldsymbol{r}$ with true density $\alpha(\boldsymbol{r})$ is:

$$\nabla\mathcal{L}(\boldsymbol{\sigma}) = \boldsymbol{a} f'(\boldsymbol{a}^T\boldsymbol{\sigma})(f(\boldsymbol{a}^T\boldsymbol{\sigma}) - \alpha(\boldsymbol{r})) \tag{4.5.43}$$

where $f(\cdot) = 1 - \exp(\cdot)$ is our nonlinearity such that $\hat{\alpha}(\boldsymbol{r}) = f(\boldsymbol{a}^T\boldsymbol{\sigma})$ as in Eq. (4.4.18). We consider what happens when we take our first gradient step starting from an initialization at $\boldsymbol{\sigma} = \boldsymbol{0}$. The expectation is over the randomness in the Gaussian measurement vector $\boldsymbol{a}$

associated to the random ray $\boldsymbol{r}$.

$$\mathbb{E}_{\boldsymbol{a}}[\nabla \mathcal{L}(\boldsymbol{\sigma})|_{\sigma=\mathbf{0}}] = \mathbb{E}_{\boldsymbol{a}}[\boldsymbol{a} f'(0)(f(0) - \alpha(\boldsymbol{r}))] \tag{4.5.44}$$

$$\overset{(a)}{=} \mathbb{E}_{\boldsymbol{a}}[-\boldsymbol{a}\alpha(\boldsymbol{r})] \tag{4.5.45}$$

$$\overset{(b)}{=} \mathbb{E}_{\boldsymbol{a}}[-\boldsymbol{a}(1 - \exp(\boldsymbol{a}^T \boldsymbol{\sigma}_*))] \tag{4.5.46}$$

$$\overset{(c)}{=} \mathbb{E}_{\boldsymbol{a}}[\boldsymbol{a} \exp(-\boldsymbol{a}^T \boldsymbol{\sigma}_*))] \tag{4.5.47}$$

$$\overset{(d)}{=} \mathbb{E}_{\boldsymbol{a}}[\frac{\boldsymbol{\sigma}_* \boldsymbol{\sigma}_*^T}{\|\boldsymbol{\sigma}_*\|_{\ell_2}^2} \boldsymbol{a} \exp(-\boldsymbol{a}^T \boldsymbol{\sigma}_*))] \tag{4.5.48}$$

$$\overset{(e)}{=} \frac{\boldsymbol{\sigma}_*}{\|\boldsymbol{\sigma}_*\|_{\ell_2}} \mathbb{E}_g[g \exp(-g \|\boldsymbol{\sigma}_*\|_{\ell_2}))] \tag{4.5.49}$$

$$\overset{(f)}{=} -\boldsymbol{\sigma}_* \mathbb{E}_g[\exp(-g \|\boldsymbol{\sigma}_*\|_{\ell_2}))] \tag{4.5.50}$$

$$\overset{(g)}{=} -\boldsymbol{\sigma}_* \exp\left(\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2\right). \tag{4.5.51}$$

In (a) we evaluate $f(0) = 0$ and $f'(0) = 1$. In (b) we plug in the value of the measurement $\alpha(\boldsymbol{r})$ using the true, optimal density vector $\boldsymbol{\sigma}_*$. In (c) we use linearity of expectation and evaluate the first term, $\mathbb{E}_{\boldsymbol{a}}[\boldsymbol{a}] = \mathbf{0}_n$. In (d) we separate the leading $\boldsymbol{a}$ into components parallel and orthogonal to $\boldsymbol{\sigma}_*$, and evaluate the expectation of the orthogonal term to zero. In (e) we replace $\boldsymbol{a}^T \boldsymbol{\sigma}_*$ with $g \|\boldsymbol{\sigma}_*\|_{\ell_2}$ for a scalar Gaussian g, as these have the same distribution. In (f) we use Stein. In (g) we evaluate the remaining expectation. From this expectation, we can already see that things look pretty good if $\|\boldsymbol{x}\|_{\ell_2}$ is small, but the gradient blows up as $\|\boldsymbol{x}\|_{\ell_2}$ gets large.

We can choose a step size t so that in expectation, our first step $\boldsymbol{\sigma}_1 = \mathbf{0}_n - t\, \mathbb{E}_{\boldsymbol{a}}[\nabla \mathcal{L}(\boldsymbol{\sigma})|_{\sigma=\mathbf{0}}] = \boldsymbol{\sigma}_*$. Specifically, we should choose

$$t = \exp\left(-\frac{\|\boldsymbol{\sigma}_*\|_{\ell_2}^2}{2}\right). \tag{4.5.52}$$

To use this step size, we must estimate $\|\boldsymbol{\sigma}_*\|_{\ell_2}$, which we can do using $\mathbb{E}_{\boldsymbol{r}}[\alpha(\boldsymbol{r})]$.

$$\mathbb{E}_{\boldsymbol{r}}[\alpha(\boldsymbol{r})] = \mathbb{E}_{\boldsymbol{a}}[1 - \exp(-\boldsymbol{a}^T \boldsymbol{\sigma}_*)] \tag{4.5.53}$$

$$= \mathbb{E}_g[1 - \exp(g \|\boldsymbol{\sigma}_*\|_{\ell_2})] \tag{4.5.54}$$

$$= 1 - \exp\left(\frac{1}{2} \|\boldsymbol{\sigma}_*\|_{\ell_2}^2\right). \tag{4.5.55}$$

In the population setting we have direct access to this expectation and can therefore compute the step size t exactly, to achieve global optimality in one step of the expected gradient. Note that numerical issues may arise in the special case when $\|\boldsymbol{\sigma}_*\|_{\ell_2} \to \infty$,

corresponding to scenes with infinite optical density (full occlusion). It is not surprising that reconstruction quality suffers in this situation.

In the finite-measurement setting for this nonlinear tomography problem, we also note that the exponential nonlinearity is of the form discussed in [OS17], in which linear convergence is shown for projected gradient descent with nearly minimal samples, provided appropriate regularization.

4.5.3 Limited-data convergence: proof sketch

In Sec. 4.5.1 and Sec. 4.5.2, we show one-step convergence of gradient descent on both the photorealistic and tomographic reconstruction problems in the population setting, when we are given unlimited access to measurements. We now sketch a strategy to extend this result to the finite-sample regime, in which we take stochastic gradient steps and apply priors through regularization or optimization constraints, to "fill in" for otherwise insufficient measurements. Note that this proof sketch is relevant primarily for the photorealistic reconstruction task; finite-sample convergence for the tomography task may be shown using a special case of [OS17], though the method described here may yield slightly tighter analysis.

Let $\boldsymbol{x}$ denote the general optimization variable; in the photorealistic reconstruction problem $\boldsymbol{x} \in \mathbb{R}^{2n}$ is the concatenation of $\boldsymbol{c}$ and $\boldsymbol{\sigma}$, and in the nonlinear tomography problem $\boldsymbol{x} \in \mathbb{R}^n$ is just an alias for $\boldsymbol{\sigma}$. We wish to show that, with high probability, $\|\boldsymbol{x} - \boldsymbol{x}_*\|_{\ell_2}$ decreases by at least a constant fraction with each step of stochastic gradient descent.

The first step is to compute the finite-sample Hessian matrix of the loss $\mathcal{L}$, and bound its minimum and maximum eigenvalues by values α and β: $\alpha\,\mathbb{I} \preceq \nabla^2\mathcal{L} \preceq \beta\,\mathbb{I}$. Then we can analyze step s of stochastic gradient descent:

$$\boldsymbol{x}_{s+1} = \boldsymbol{x}_s - t\nabla\mathcal{L}(\boldsymbol{x}_s) \tag{4.5.56}$$

$$\boldsymbol{x}_{s+1} - \boldsymbol{x}_* \overset{(a)}{=} \boldsymbol{x}_s - \boldsymbol{x}_* - t(\nabla\mathcal{L}(\boldsymbol{x}_s) - \nabla\mathcal{L}(\boldsymbol{x}_*)) \tag{4.5.57}$$

$$\overset{(b)}{=} \boldsymbol{x}_s - \boldsymbol{x}_* - t\nabla^2\mathcal{L}(\bar{\boldsymbol{x}}_s)(\boldsymbol{x}_s - \boldsymbol{x}_*) \tag{4.5.58}$$

$$= (\mathbb{I} - t\nabla^2\mathcal{L}(\bar{\boldsymbol{x}}_s))(\boldsymbol{x}_s - \boldsymbol{x}_*). \tag{4.5.59}$$

In (a) we subtract $\boldsymbol{x}_*$ from each side, and add zero in the form of $\nabla\mathcal{L}(\boldsymbol{x}_*)$. In (b) we use the mean value theorem with $\bar{\boldsymbol{x}}_s \in [\boldsymbol{x}_s, \boldsymbol{x}_*]$. Then we have

$$\|\boldsymbol{x}_{s+1} - \boldsymbol{x}_*\|_{\ell_2} \leq \left\|\mathbb{I} - t\nabla^2\mathcal{L}\right\|_{\ell_2} \|\boldsymbol{x}_s - \boldsymbol{x}_*\|_{\ell_2}\,, \tag{4.5.60}$$

and what remains is to show that we can choose step sizes t so that $\left\|\mathbb{I} - t\nabla^2\mathcal{L}\right\|_{\ell_2} < 1$ and the distance to the optimum shrinks by a geometric factor with each step.

This in turn requires two pieces: (1) showing that, with high probability, the finite-sample Hessian matrix $\nabla^2\mathcal{L}$ is positive definite in some radius Δ around the optimum $\boldsymbol{x}_*$,

so that there exists a positive step size t in this radius that results in convergence, and (2) showing that, with high probability, our first stochastic gradient step lands at an iterate $\boldsymbol{x}$ with $\|\boldsymbol{x} - \boldsymbol{x}_*\|_{\ell_2} < \Delta$. Each of these pieces may be done using the eigenvalue bounds α, β on $\nabla^2 \mathcal{L}$.

Finally, the finite-sample convergence proof may be strengthened by the addition of a prior, such as sparsity or low total variation. The prior is implemented in practice as a regularization term in the objective, but for analysis we consider it as a constraint set during optimization. In this setting, we replace gradient or stochastic gradient descent with its projected version to maintain constraint satisfaction. This projection step then reduces the complexity of the optimization space by tightening the eigenvalue bounds on the relevant subspace of the Hessian, allowing faster convergence with fewer samples as long as the true signal satisfies the prior constraint [Sol19; Sol17].

4.6 Sensitivity analysis: Voxel-wise confidence for tomographic reconstruction

In this section, we explore a different type of error bound: voxel-wise confidence intervals for the nonlinear tomography problem. One motivation for this type of bound comes from medical imaging, in which tomographic reconstruction is used to identify and diagnose localized features of medical importance, such as dental cavities, tumors, and arterial blockages. For these detail-critical tasks, global error bounds are of limited use without additional information about how errors are distributed throughout the reconstructed volume.

We therefore pose, and take a step towards answering, the following question: assuming we can optimize our volume to achieve zero loss on training rays, what is our confidence interval over the density of each voxel? We can compute a range $[\underline{\sigma}_i, \bar{\sigma}_i]$ such that the total training loss changes by no more than ε as long as the density σ_i at voxel i remains within this range, and all other voxel densities remain unchanged.

For convenience, consider the case of nearest neighbor interpolation, where all samples within a voxel are considered as having the exact same density, and all samples are spaced equally along each ray (so we can drop the subscript on δ). Let J_i denote the set of rays j passing through voxel i and T_j denote the final transmission of ray j, such that $T_j = 0$ if the ray is completely opaque and $T_j = 1$ if the ray is completely transparent. We can formalize our desired range as follows.

$$\sum_{j \in J_i} \left(T_j - \exp\left(- \sum_{k \in j} \sigma_k \delta \right) \right)^2 \leq \varepsilon \tag{4.6.1}$$

$$\sum_{j \in J_i} \left(T_j - \exp(-\sigma_i \delta) \exp\left(- \sum_{k \in j; k \neq i} \sigma_k \delta \right) \right)^2 \leq \varepsilon \tag{4.6.2}$$

$$\sum_{j \in J_i} T_j^2 - 2\exp(-\sigma_i \delta) \sum_{j \in J_i} T_j \exp\left(- \sum_{k \in j; k \neq i} \sigma_k \delta \right) + \exp(2\sigma_i \delta) \sum_{j \in J_i} \exp\left(- 2 \sum_{k \in j; k \neq i} \sigma_k \delta \right) \leq \varepsilon \tag{4.6.3}$$

$$\sum_{j \in J_i} T_j^2 - 2\exp(-\sigma_i \delta) \sum_{j \in J_i} T_j z_{ji} + \exp(2\sigma_i \delta) \sum_{j \in J_i} z_{ji}^2 \leq \varepsilon. \tag{4.6.4}$$

In the last line for notational convenience we let $z_{ji} = \exp\left(- \sum_{k \in j; k \neq i} \sigma_k \delta \right)$, the transmission of ray j excluding the contribution of the sample in voxel i. Then we can solve the quadratic:

$$\exp(-\sigma_i \delta) \in \frac{\sum_{j \in J_i} T_j z_{ji} \pm \sqrt{\left(\sum_{j \in J_i} T_j z_{ji} \right)^2 - \left(\sum_{j \in J_i} z_{ji}^2 \right) \left(-\varepsilon + \sum_{j \in J_i} T_j^2 \right)}}{\sum_{j \in J_i} z_{ji}^2}. \tag{4.6.5}$$

As a sanity check, we can see that as the rest of a ray becomes more transparent (as z_{ji} increases towards 1), the tighter the confidence interval around σ_i becomes. Likewise, as more training rays pass through a voxel i, $\sum_{j \in J_i} z_{ji}^2$ can only increase and the confidence interval around σ_i can only shrink.

This bound has the expected behavior and accurately captures our uncertainty in the value of σ_i assuming that the training loss is zero, that sample values are from nearest neighbor interpolation (all samples in the same voxel have the same density as that stored at the voxel), that samples are evenly spaced along rays, and that each ray passes through any given voxel no more than once (this might be violated if samples are more densely spaced along the ray).

In particular, we do not use nearest neighbor interpolation; we use trilinear interpolation (or could use other continuous interpolators). These continuous interpolations preclude the factoring out of the σ_i term from the ray sum, because the same value σ_i contributes with varying weights to the sample densities of multiple samples along each ray in its vicinity. Nonetheless, this bound may be interpreted as a best-case range of σ_i values for which, all else held fixed, the training loss changes by no more than ε.

Future work may make these per-voxel error bounds more conservative and complete by incorporating error due to nonzero training loss, generalization error if training measurements are insufficient, as well as any measurement noise (which we assume to be zero in this analysis).

Chapter 5

Next Steps

This dissertation represents a first foray into the rich interaction between compressed sensing and deep learning, for the particular inverse problem of photorealistic reconstruction. For this task, we have explored practical reconstruction algorithms and scene representations in Chapter 2 and Chapter 3, showing that grid-based representations and explicit regularization combine fruitfully with nonlinear, gradient-based optimization for high-quality, fast reconstruction. We have also taken a theoretical perspective in Chapter 4, in which we show that, at least in the presence of unlimited measurements, the forward model of photorealistic reconstruction is perfectly invertible by gradient descent despite its nonlinearity. In this chapter, we take a step back and consider the many exciting avenues for future research that may build upon this work.

In particular, Chapter 4 is but the beginning of a full understanding of the photorealistic reconstruction problem, and its many cousins in other inverse problems. Immediate next steps for the two inverse problems discussed here, photorealistic reconstruction and nonlinear tomography, include (1) completing the convergence analysis for the finite-sample case with regularization, and (2) experimental validation of this analysis, including scaling of error with number of training rays, and use of the theoretically-motivated step size for SGD. For the best-case local error bounds in Sec. 4.6, next steps include extending the confidence interval to account for any measurement noise in the training data, as well as generalization error.

Another class of future work involves studying the bias induced by compressive models, like the "low-rank" method of Chapter 3. More broadly, future work may develop a comprehensive theory of regularization for inverse problems. The work presented here focuses on prior-regularizer pairs that are well-understood and tend to apply widely, namely total variation in space and smoothness in time. Stronger, data-driven priors have shown great promise empirically on controlled datasets, but remain poorly understood and therefore difficult to deploy with confidence on safety-critical tasks. Given a type of data, how should we construct the best regularizer, and what priors should it be based on? How can we verify if our priors are still applicable on new data as it arises? How can we visualize and understand the effects of our regularizers, especially when they are

data-driven?

Finally, the broad topics discussed in this dissertation may find parallels and application in other inverse problems of interest across scientific and medical imaging systems. For example, what is the most efficient representation for different types of domain-specific signals, *e.g*. what basis tends to produce the most sparse coefficients? How can we optimize that sparse representation efficiently? Under what conditions can we guarantee correct reconstruction, for inverse problems with different nonlinear forward models? It is hoped that this dissertation may serve as a useful building block towards a full exploration of these questions, for the future benefit of scientific and medical imaging and inquiry.

Bibliography

[AB91] Edward H. Adelson and James R. Bergen. "The Plenoptic Function and the Elements of Early Vision". In: *Computational Models of Visual Processing*. MIT Press, 1991.

[ASKUL20] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. "Neural point-based graphics". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16*. Springer. 2020.

[BFH+18] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.2.5. 2018. URL: http://github.com/google/jax.

[BJ03] R. Basri and D.W. Jacobs. "Lambertian reflectance and linear subspaces". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.2 (2003). DOI: 10.1109/TPAMI.2003.1177153.

[BJLS17] Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. *Optimizing the Latent Space of Generative Networks*. 2017. DOI: 10.48550/ARXIV.1707.05776. URL: https://arxiv.org/abs/1707.05776.

[BMT+21a] 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*. 2021. arXiv: 2103.13415 [cs.CV].

[BMT+21b] 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: *Int. Conf. Comput. Vis.* IEEE, 2021. DOI: 10.1109/ICCV48922.2021.00580. URL: https://doi.org/10.1109/ICCV48922.2021.00580.

[BMVSH22] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields". In: *IEEE Conf. Comput. Vis. Pattern Recog.* IEEE, 2022. DOI: 10.1109/CVPR52688.2022.00539. URL: https://doi.org/10.1109/CVPR52688.2022.00539.

[BV11] Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge Univ. Pr., 2011.

[BWCB22] Sagie Benaim, Frederik Warburg, Peter Ebert Christensen, and Serge Belongie. *Volumetric Disentanglement for 3D Scene Manipulation*. 2022. DOI: 10.48550/ARXIV.2206.02776. URL: https://arxiv.org/abs/2206.02776.

[CLC+22] Eric R. Chan et al. "Efficient Geometry-aware 3D Generative Adversarial Networks". In: *IEEE Conf. Comput. Vis. Pattern Recog.* 2022. DOI: 10.1109/CVPR52688.2022.01565.

[CRT06] E.J. Candes, J. Romberg, and T. Tao. "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information". In: *IEEE Transactions on Information Theory* 52.2 (2006). DOI: 10.1109/TIT.2005.862083.

[CXGCS16] Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. *3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction*. 2016. arXiv: 1604.00449 [cs.CV].

[CXGYS22] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. "TensoRF: Tensorial Radiance Fields". In: *Eur. Conf. Comput. Vis.* 2022.

[CZ19] Zhiqin Chen and Hao Zhang. *Learning Implicit Fields for Generative Shape Modeling*. 2019. arXiv: 1812.02822 [cs.GR].

[DBS20] Boyang Deng, Jonathan T. Barron, and Pratul P. Srinivasan. *JaxNeRF: an efficient JAX implementation of NeRF*. Version 0.0. 2020. URL: https://github.com/google-research/google-research/tree/master/jaxnerf.

[DH01] D.L. Donoho and X. Huo. "Uncertainty principles and ideal atomic decomposition". In: *IEEE Transactions on Information Theory* 47.7 (2001). DOI: 10.1109/18.959265.

[DZYTW21] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B. Tenenbaum, and Jiajun Wu. "Neural Radiance Flow for 4D View Synthesis and Video Processing". In: *Int. Conf. Comput. Vis.* 2021. DOI: 10.1109/ICCV48922.2021.01406.

[FMWRK23] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. "K-Planes: Explicit Radiance Fields in Space, Time, and Appearance". In: *CVPR*. 2023.

[FYT+22] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. "Plenoxels: Radiance Fields without Neural Networks". In: *IEEE Conf. Comput. Vis. Pattern Recog.* IEEE, 2022. DOI: 10.1109/CVPR52688.2022.00542.

[FYW+22] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. "Fast Dynamic Radiance Fields with Time-Aware Neural Voxels". In: *SIGGRAPH Asia 2022 Conference Papers*. ACM, 2022. DOI: 10.1145/3550469.3555383. URL: https://doi.org/10.1145%2F3550469.3555383.

[GCD+22] Xiang Guo, Guanying Chen, Yuchao Dai, Xiaoqing Ye, Jiadai Sun, Xiao Tan, and Errui Ding. "Neural Deformable Voxel Grid for Fast Optimization of Dynamic View Synthesis". In: *ACCV*. 2022.

[GKJSV21] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. *FastNeRF: High-Fidelity Neural Rendering at 200FPS*. 2021. arXiv: 2103.10380 [cs.CV].

[GLTRK22] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. "Monocular Dynamic View Synthesis: A Reality Check". In: *Adv. Neural Inform. Process. Syst.* 2022.

[GSKH21] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. "Dynamic View Synthesis From Dynamic Monocular Video". In: *Int. Conf. Comput. Vis.* 2021.

[GXHCY22] Wanshui Gan, Hongbin Xu, Yi Huang, Shifeng Chen, and Naoto Yokoya. *V4D: Voxel for 4D Novel View Synthesis*. 2022. DOI: 10.48550/ARXIV.2205.14332. URL: https://arxiv.org/abs/2205.14332.

[Hin] Geoffrey Hinton. *RMSProp*. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[HSMBD21] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. *Baking Neural Radiance Fields for Real-Time View Synthesis*. 2021. arXiv: 2103.14645 [cs.CV].

[HTM17] Christian Häne, Shubham Tulsiani, and Jitendra Malik. *Hierarchical Surface Prediction for 3D Object Reconstruction*. 2017. arXiv: 1704.00710 [cs.CV].

[JMM+21] Yuhe Jin, Dmytro Mishkin, Anastasiia Mishchuk, Jiri Matas, Pascal Fua, Kwang Moo Yi, and Eduard Trulls. "Image Matching Across Wide Baselines: From Paper to Practice". In: *Int. J. Comput. Vis.* 129.2 (2021). DOI: 10.1007/s11263-020-01385-0. URL: https://doi.org/10.1007/s11263-020-01385-0.

[KB17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[KHM17] Abhishek Kar, Christian Häne, and Jitendra Malik. *Learning a Multi-View Stereo Machine*. 2017. arXiv: 1708.05375 [cs.CV].

[KJJ+21] Petr Kellnhofer, Lars Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. *Neural Lumigraph Rendering*. 2021. arXiv: 2103.11571 [cs.CV].

[Kno06] Aaron Knoll. *A Survey of Octree Volume Rendering Methods*. 2006.

[KPZK17] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. "Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction". In: *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073599. URL: https://doi.org/10.1145/3072959.3073599.

[KV84] James T. Kajiya and Brian P Von Herzen. "Ray Tracing Volume Densities". In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '84. New York, NY, USA: Association for Computing Machinery, 1984. ISBN: 0897911385. DOI: 10.1145/800031.808594. URL: https://doi.org/10.1145/800031.808594.

[LCM+22] Jia-Wei Liu, Yan-Pei Cao, Weijia Mao, Wenqiao Zhang, David Junhao Zhang, Jussi Keppo, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. "DeVRF: Fast Deformable Voxel Radiance Fields for Dynamic Scenes". In: *arxiv:2205.15723* (2022). URL: https://arxiv.org/abs/2205.15723.

[Lev88] M. Levoy. "Display of surfaces from volume data". In: *IEEE Computer Graphics and Applications* 8.3 (1988). DOI: 10.1109/38.511.

[LGLCT21] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. *Neural Sparse Voxel Fields*. 2021. arXiv: 2007.11571 [cs.CV].

[LMW21] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. *AutoInt: Automatic Integration for Fast Neural Volume Rendering*. 2021. arXiv: 2012.01714 [cs.CV].

[LNSW21] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. "Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes". In: *IEEE Conf. Comput. Vis. Pattern Recog.* 2021.

[LSS+19a] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. "Neural volumes". In: *ACM Transactions on Graphics* 38.4 (July 2019). ISSN: 1557-7368. DOI: 10.1145/3306346.3323020. URL: http://dx.doi.org/10.1145/3306346.3323020.

[LSS+19b] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. "Neural Volumes: Learning Dynamic Renderable Volumes from Images". In: *ACM Trans. Graph.* 38.4 (July 2019).

[LSZ+22] Tianye Li et al. "Neural 3D Video Synthesis from Multi-view Video". In: *IEEE Conf. Comput. Vis. Pattern Recog.* 2022. DOI: 10.1109/CVPR52688.2022.00544.

[LZ21] Christoph Lassner and Michael Zollhöfer. "Pulsar: Efficient Sphere-based Neural Rendering". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[Max95] N. Max. "Optical models for direct volume rendering". In: *IEEE Trans. Vis. Comput. Graph.* 1.2 (1995). DOI: 10.1109/2945.468400.

[MC21] Duane Merrill and NVIDIA Corporation. *CUB: Cooperative primitives for CUDA C++*. Version 1.9.9. 2021. URL: https://github.com/NVIDIA/cub.

[MESK22] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding". In: *ACM Trans. Graph.* 41.4 (2022).

[MGK+19] Moustafa Meshry, Dan B. Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. "Neural Rerendering in the Wild". In: *IEEE Conf. Comput. Vis. Pattern Recog.* 2019. DOI: 10.1109/CVPR.2019.00704.

[MLL+21] Julien N. P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. *ACORN: Adaptive Coordinate Networks for Neural Scene Representation*. 2021. arXiv: 2105.02788 [cs.CV].

[MONNG19] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. *Occupancy Networks: Learning 3D Reconstruction in Function Space*. 2019. arXiv: 1812.03828 [cs.CV].

[MRS+21] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. "NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections". In: *IEEE Conf. Comput. Vis. Pattern Recog.* 2021.

[MSC+19] 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". In: *ACM Trans. Graph.* 38.4 (2019). DOI: 10.1145/3306346.3322980. URL: https://doi.org/10.1145/3306346.3322980.

[MSO+19] 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*. 2019. arXiv: 1905.00889 [cs.CV].

[MST+20] 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". In: *Eur. Conf. Comput. Vis.* Springer. 2020.

[Nis10] Dwight George Nishimura. *Principles of Magnetic Resonance Imaging*. 2010.

[Noc80] Jorge Nocedal. "Updating Quasi-Newton Matrices with Limited Storage". In: *Mathematics of Computation* 35.151 (1980). ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2006193 (visited on 04/17/2023).

[NSP+21] T. Neff, P. Stadlbauer, M. Parger, A. Kurz, J. H. Mueller, C. R. A. Chaitanya, A. Kaplanyan, and M. Steinberger. "DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks". In: *Computer Graphics Forum* 40.4 (July 2021). ISSN: 1467-8659. DOI: 10.1111/cgf.14340. URL: http://dx.doi.org/10.1111/cgf.14340.

[NVF20] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. *CUDA, release: 10.2.89*. 2020. URL: https://developer.nvidia.com/cuda-toolkit.

[NW06a] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer, 2006.

[NW06b] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006.

[ORS18] Samet Oymak, Benjamin Recht, and Mahdi Soltanolkotabi. "Isometric sketching of any set via the Restricted Isometry Property". In: *Information and Inference: A Journal of the IMA* 7.4 (Mar. 2018). ISSN: 2049-8764. DOI: 10.1093/imaiai/iax019. eprint: https://academic.oup.com/imaiai/article-pdf/7/4/707/32468519/iax019.pdf. URL: https://doi.org/10.1093/imaiai/iax019.

[OS09] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0131988425.

[OS17] Samet Oymak and Mahdi Soltanolkotabi. "Fast and Reliable Parameter Estimation from Nonlinear Observations". In: *SIAM Journal on Optimization* 27.4 (2017). DOI: 10.1137/17M1113874. eprint: https://doi.org/10.1137/17M1113874. URL: https://doi.org/10.1137/17M1113874.

[OZC+20] Frank Ong, Xucheng Zhu, Joseph Y. Cheng, Kevin M. Johnson, Peder E. Z. Larson, Shreyas S. Vasanawala, and Michael Lustig. "Extreme MRI: Large-scale volumetric dynamic imaging from continuous non-gated acquisitions". In: *Magnetic Resonance in Medicine* 84.4 (2020). DOI: https://doi.org/10.1002/mrm.28235. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/mrm.28235. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/mrm.28235.

[PC21] Martin Piala and Ronald Clark. *TermiNeRF: Ray Termination Prediction for Efficient Neural Rendering*. 2021. arXiv: 2111.03643 [cs.CV].

[PCPM21] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. "D-NeRF: Neural Radiance Fields for Dynamic Scenes". In: *IEEE Conf. Comput. Vis. Pattern Recog.* 2021.

[PFSNL19] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation*. 2019. arXiv: 1901.05103 [cs.CV].

[PNMPG20] Songyou Peng, Michael Niemeyer, Lars M. Mescheder, Marc Pollefeys, and Andreas Geiger. "Convolutional Occupancy Networks". In: *Eur. Conf. Comput. Vis.* Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Lecture Notes in Computer Science. 2020. DOI: 10.1007/978-3-030-58580-8_31.

[PSB+21] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. "Nerfies: Deformable Neural Radiance Fields". In: *Int. Conf. Comput. Vis.* 2021.

[PSH+21] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. "HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields". In: *ACM Trans. Graph.* 40.6 (Dec. 2021).

[PZ17] Eric Penner and Li Zhang. "Soft 3D reconstruction for view synthesis". In: *ACM Transactions on Graphics (TOG)* 36 (2017).

[RFS21] Darius Rückert, Linus Franke, and Marc Stamminger. *ADOP: Approximate Differentiable One-Pixel Point Rendering*. 2021. arXiv: 2110.06635 [cs.CV].

[RH01] Ravi Ramamoorthi and Pat Hanrahan. "On the relationship between radiance and irradiance: determining the illumination from images of a convex Lambertian object". In: *JOSA A* 18.10 (2001).

[RJY+20] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. *DeRF: Decomposed Radiance Fields*. 2020. arXiv: 2011.12490 [cs.CV].

[RK20] Gernot Riegler and Vladlen Koltun. *Free View Synthesis*. 2020. arXiv: 2008.05511 [cs.CV].

[RO94] Leonid I Rudin and Stanley Osher. "Total variation based image restoration with free local constraints". In: *Proceedings of 1st International Conference on Image Processing*. Vol. 1. IEEE. 1994.

[RPLG21] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. *KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs*. 2021. arXiv: 2103.13744 [cs.CV].

[RUG17] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. *OctNet: Learning Deep 3D Representations at High Resolutions*. 2017. arXiv: 1611.05009 [cs.CV].

[SCL+22] Liangchen Song, Anpei Chen, Zhong Li, Zhang Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. *NeRFPlayer: A Streamable Dynamic Scene Representation with Decomposed Neural Radiance Fields*. 2022. URL: https://arxiv.org/abs/2210.15947.

[SD97] S.M. Seitz and C.R. Dyer. "Photorealistic scene reconstruction by voxel coloring". In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1997. DOI: 10.1109/CVPR.1997.609462.

[SG04] Richard Szeliski and Polina Golland. "Stereo Matching with Transparency and Matting". In: *International Journal of Computer Vision* 32 (2004).

[SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. "Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments". In: *ACM Trans. Graph.* 21.3 (July 2002). ISSN: 0730-0301. DOI: 10.1145/566654.566612. URL: https://doi.org/10.1145/566654.566612.

[Sol17] Mahdi Soltanolkotabi. "Learning ReLUs via Gradient Descent". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017. ISBN: 9781510860964.

[Sol19] Mahdi Soltanolkotabi. "Structured Signal Recovery From Quadratic Measurements: Breaking Sample Complexity Barriers via Nonconvex Optimization". In: *IEEE Transactions on Information Theory* 65.4 (2019). DOI: 10.1109/TIT.2019.2891653.

[SSC22] Cheng Sun, Min Sun, and Hwann-Tzong Chen. "Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction". In: *IEEE Conf. Comput. Vis. Pattern Recog.* 2022.

[SSKK00] Seitz, Steven, Kutulakos, and Kiriakos. "A Theory of Shape by Space Carving". In: (Jan. 2000).

[STB+19] Pratul P. Srinivasan, Richard Tucker, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. *Pushing the Boundaries of View Extrapolation with Multiplane Images*. 2019. arXiv: 1905.00413 [cs.CV].

[STH+19] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. *DeepVoxels: Learning Persistent 3D Feature Embeddings*. 2019. arXiv: 1812.01024 [cs.CV].

[SZT+22] Ruizhi Shao, Zerong Zheng, Hanzhang Tu, Boning Liu, Hongwen Zhang, and Yebin Liu. *Tensor4D : Efficient Neural 4D Decomposition for High-fidelity Dynamic Reconstruction and Rendering*. 2022. DOI: 10.48550/ARXIV.2211.11610. URL: https://arxiv.org/abs/2211.11610.

[SZW20] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. *Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations*. 2020. arXiv: 1906.01618 [cs.CV].

[TCY+22] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. *Block-NeRF: Scalable Large Scene Neural View Synthesis*. 2022. DOI: 10.48550/ARXIV.2202.05263. URL: https://arxiv.org/abs/2202.05263.

[TDB17] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. *Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs*. 2017. arXiv: 1703.09438 [cs.CV].

[TLY+21] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. *Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes*. 2021. arXiv: 2101.10994 [cs.CV].

[TMW+21a] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. *Learned Initializations for Optimizing Coordinate-Based Neural Representations*. 2021. arXiv: 2012.02189 [cs.CV].

[TMW+21b] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. "Learned Initializations for Optimizing Coordinate-Based Neural Representations". In: *IEEE Conf. Comput. Vis. Pattern Recog.* Computer Vision Foundation / IEEE, 2021. DOI: 10.1109/CVPR46437.2021.00287.

[TTG+21] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. "Non-Rigid Neural Radiance Fields: Reconstruction and Novel View Synthesis of a Dynamic Scene From Monocular Video". In: *Int. Conf. Comput. Vis.* IEEE. 2021.

[TZEM17] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. *Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency*. 2017. arXiv: 1704.06254 [cs.CV].

[UK88] Craig Upson and Michael Keeler. "V-Buffer: Visible Volume Rendering". In: *SIGGRAPH Comput. Graph.* 22.4 (June 1988). ISSN: 0097-8930. DOI: 10.1145/378456.378482. URL: https://doi.org/10.1145/378456.378482.

[Vid20] Mathukumalli Vidyasagar. *An introduction to compressed sensing*. SIAM, Society for Industrial and Applied Mathematics, 2020.

[WBSS04] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Trans. Image Process.* 13.4 (2004). DOI: 10.1109/TIP.2003.819861.

[WGSJ20] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. *SynSin: End-to-end View Synthesis from a Single Image*. 2020. arXiv: 1912.08804 [cs.CV].

[WLGST17] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. "O-CNN". In: *ACM Transactions on Graphics* 36.4 (July 2017). ISSN: 1557-7368. DOI: 10.1145/3072959.3073608. URL: http://dx.doi.org/10.1145/3072959.3073608.

[WM22] John Wright and Yi Ma. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2022.

[WPYS21] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. "NeX: Real-Time View Synthesis With Neural Basis Expansion". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021. DOI: 10.1109/CVPR46437.2021.00843. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Wizadwongsa%5C_NeX%5C_Real-Time%5C_View%5C_Synthesis%5C_With%5C_Neural%5C_Basis%5C_Expansion%5C_CVPR%5C_2021%5C_paper.html.

[WR22] Stephen J. Wright and Benjamin Recht. *Optimization for Data Analysis*. Cambridge University Press, 2022. DOI: 10.1017/9781009004282.

[WTLTL22] Feng Wang, Sinan Tan, Xinghang Li, Zeyue Tian, and Huaping Liu. *Mixed Neural Voxels for Fast Multi-view Video Synthesis*. 2022. DOI: 10.48550/ARXIV.2212.00190. URL: https://arxiv.org/abs/2212.00190.

[WZTCO22] Tianhao Wu, Fangcheng Zhong, Andrea Tagliasacchi, Forrester Cole, and Cengiz Oztireli. *D^2NeRF: Self-Supervised Decoupling of Dynamic and Static Objects from a Monocular Video*. 2022. DOI: 10.48550/ARXIV.2205.15838. URL: https://arxiv.org/abs/2205.15838.

[XHKK21] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. "Space-time Neural Irradiance Fields for Free-Viewpoint Video". In: *IEEE Conf. Comput. Vis. Pattern Recog.* 2021.

[YLSL21] Wentao Yuan, Zhaoyang Lv, Tanner Schmidt, and Steven Lovegrove. "STaR: Self-supervised Tracking and Reconstruction of Rigid Objects in Motion with Neural Rendering". In: *IEEE Conf. Comput. Vis. Pattern Recog.* 2021.

[YLT+21] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. "PlenOctrees for Real-time Rendering of Neural Radiance Fields". In: *Int. Conf. Comput. Vis.* 2021. DOI: 10.1109/ICCV48922.2021.00570.

[Yu 83] Yu. E. Nesterov. "A method of solving a convex programming problem with convergence rate $O\left(\frac{1}{k^2}\right)$". In: *Dokl. Akad. Nauk SSSR* 269 (3 1983).

[ZIESW18] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". In: *CVPR*. 2018.

[ZRSK20] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. *NeRF++: Analyzing and Improving Neural Radiance Fields*. 2020. arXiv: 2010.07492 [cs.CV].

[ZTFFS18] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. *Stereo Magnification: Learning View Synthesis using Multiplane Images*. 2018. arXiv: 1805.09817 [cs.CV].

Appendix: Static Scene Reconstruction from First Principles

Overview

In the appendix, we include additional experimental details and present results and visualizations of further ablation studies. We also present full, per-scene quantitative and visual comparisons between our method and prior work. We encourage the reader to see the video at `alexyu.net/plenoxels` for results of our method on a wide range of scenes.

Experimental details

Implementation details

As briefly discussed in Sec. 2.3.2, we use a simple data structure which consists of a data table in addition to a dense grid, where each cell is either NULL or a pointer into the data table. Each entry in the data table consists of the density value and the SH coefficients for each of the RGB color channels. NULL cells are considered to have all 0 values. This data structure allows for reasonably efficient trilinear interpolation both in the forward and backward passes while maintaining sparsity; due to the relatively large memory requirements to store the SH coefficients, gradients, and RMSProp running averages, the dense pointer grid is usually not dominant in size. Nevertheless, reading the pointers currently appears to take a significant amount of rendering time, and optimizations are likely possible.

Our main CUDA rendering and gradient kernels simultaneously parallelize across rays, colors, and SH coefficients. Each CUDA warp (32 threads) handles one ray, with threads processing one SH coefficient each; since coefficients are stored contiguously, this means access to global memory is highly coalesced. The SH coefficients are combined into colors using warp-level operations from NVIDIA CUB [MC21]. These features are particularly significant in the case of trilinear interpolation.

Note that in order to correctly perform trilinear color interpolation, instead of using the sigmoid function to ensure that predicted sample colors are always between 0 and 1

as in NeRF [MST+20], we simply clip negative color values to 0 with a ReLU to preserve linearity as much as possible.

We use weight-based thresholding (as in PlenOctrees [YLT+21]) for the synthetic and real, 360° scenes, and density-based thresholding for the forward-facing scenes. The reason for this is that some content (especially at the edges) in the forward-facing scenes is not visible in most of the training views, so weight-based thresholding tends to prune these sparsely-supervised features.

We use a batch size of 5000 rays and optimize with RMSProp [Hin]. For σ we use the same delayed exponential learning rate schedule as Mip-NeRF [BMT+21a], where the exponential is scaled by a learning rate of 30 (this is where the exponential would start, if not for the delay) and decays to 0.05 at step 250000, with an initial delay period of 15000 steps. For SH we use a pure exponential decay learning rate schedule, with an initial learning rate of 0.01 that decays to 5×10^{-6} at step 250000.

The TV losses are evaluated stochastically; they are applied only to 1% of all voxels in the grid in each step. Note that empty voxels can be selected, as their neighbors may not be empty. In practice, for performance reasons, we always apply the TV regularization on random contiguous segments of voxels (in the order that the pointer grid is stored). This is much faster to evaluate on the GPU due to locality. In all cases, the voxel differences in the TV loss defined below Eq. (2.3.3) is in practice normalized by the voxel resolution in each dimension, relative to 256 (for historical reasons):

$$\Delta_x((i,j,k),d) = \frac{|V_d(i+1,j,k) - V_d(i,j,k)|}{256/D_x} \tag{5.0.1}$$

Where D_x is the grid resolution in the x dimension, and $V_d(i,j,k)$ is the dth value of voxel (i,j,k) (either density or a SH coefficient). We scale Δ_y, Δ_z analogously. Note that the same loss is applied in NDC and to the background model, except in the background model, the TV also wraps around the edges of the equirectangular image. For SH, empty grid cells and edges are considered to have the same value as the current cell (instead of 0) for purposes of TV.

Synthetic experiments

On the synthetic scenes, we found that our method performs nearly identically when TV regularization is present only in the first stage of optimization; turning off the regularization after pruning voxels and increasing resolution reduces our training time modestly. We suspect (see Tab. 2.3) this is due to the large number of training views (100) available for these scenes as well as the low level of noise; for the other datasets we retain TV regularization throughout optimization.

We start at resolution 256^3, prune and upsample to resolution 512^3 after 38400 steps (the equivalent of 3 epochs), and optimize for a total of 128000 steps (the equivalent of 10 epochs). We prune using a weight threshold of 0.256, and use λ_{TV} of 1×10^{-5} for σ and

1×10^{-3} for SH, only during the initial 38400 steps (and then turn off regularization after pruning and upsampling, for faster optimization).

Forward-facing experiments

For the forward-facing scenes we start at resolution $256\times256\times128$, prune and upsample to resolution $512\times512\times128$ at step 38400, prune and upsample to resolution $1408\times1156\times128$ at step 76800, and optimize for a total of 128000 steps. The final grid resolution is derived from the image resolution of the dataset, with some padding added on each side. We prune using a σ threshold of 5, use λ_{TV} of 5×10^{-4} for σ and 5×10^{-3} for SH, and use a sparsity penalty λ_s of 1×10^{-12} to encourage empty voxels.

While these TV parameters work well for the forward-facing NeRF scenes, more generally, we find that sometimes it is preferrable to use λ_{TV} 5×10^{-3} for density and 5×10^{-2} for SH, which reduces artifacts while blurring the scene more. This is used for some of the examples in the video, for example the piano. In general, since scenes differ significantly in content, camera noise, and actual scale, a hyperparameter sweep of the TV weights can be helpful, and using different TV values across the scenes would improve the metrics for the NeRF scenes as well.

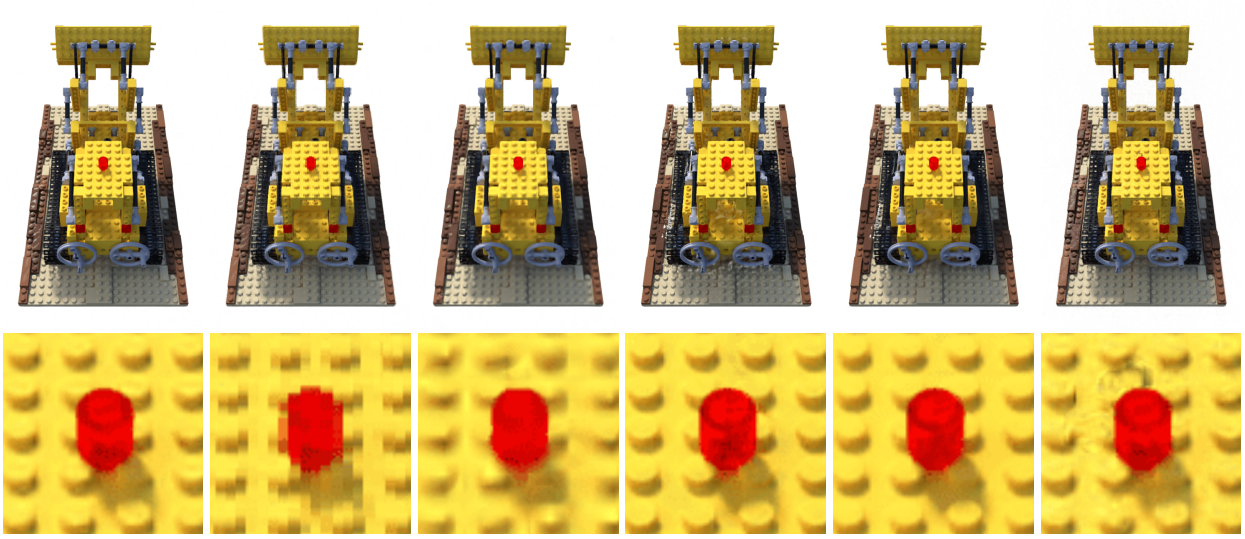
360° experiments

For the 360° scenes our foreground Plenoxel grid starts at resolution 128^3; we prune and upsample to 256^3, 512^3, and 640^3 with 25600 steps in between each upsampling. We optimize for a total of 102400 steps. We prune using a weight threshold of 1.28, and use λ_{TV} of 5×10^{-5} for σ and 5×10^{-3} for SH for the inner grid and λ_{TV} of 1×10^{-3} for both σ and SH for the 64 background grid layers of resolution 2048×1024. We use λ_s of 1×10^{-11} and λ_β of 1×10^{-5}. For simplicity of implementation, we did not use coarse-to-fine for the background and only use σ thresholding. We also do not use the delayed learning rate function for the background, opting instead to use an exponential decay to allow the background to optimize faster than the foreground at the beginning.

While the TV weights were fixed for these scenes, in general, a hyperparameter sweep of the TV weights can be helpful. For more general scenes, it is sometimes useful to use a near-bound on the camera rays (as in NeRF) to prevent floaters very close to the camera, or to only begin optimizing the foreground after, say, 1000 iterations. Further sparsity losses to encourage the weight distribution to be a delta function may also help.

Ablation studies

We visualize ablations on the synthetic lego scene in Fig. 1. In addition to comparing nearest neighbor and trilinear interpolation, we also experimented with tricubic interpolation, which produces a function approximation that is both continuous (like trilinear



(a) Trilinear, 256^3 **(b)** Nearest, 256^3 **(c)** Trilinear, 128^3 **(d)** 25 Views, Low TV **(e)** 25 Views, High TV **(f)** Neural Vol. Formula

Figure 1: **Visual results of ablation studies** on the synthetic lego scene. Trilinear interpolation at resolution 256^3 is quite similar to our full model at resolution 512^3. Nearest neighbor interpolation shows clear voxel artifacts. Trilinear interpolation at lower resolution appears less detailed. Reducing the number of training views produces visual artifacts that are mostly resolved by increasing the TV regularization. Optimizing and rendering with the Neural Volumes [LSS+19a] formula produces different visual artifacts.

interpolation) and smooth. However, we found tricubic interpolation offered negligible improvements compared to trilinear, in exchange for a substantial increase in computation (this increase in computation is why we do not include a full ablation table for tricubic interpolation).

Tab. 1 and Tab. 2 show ablations over learning rate schedule and optimizer, respectively. We find that Plenoxel optimization is reasonably robust to both of these hyperparameters, although there is a noticeable improvement from using RMSProp compared to SGD, particularly for the spherical harmonic coefficients. Note that when comparing different learning rate schedules and optimizers, we tune the initial learning rate separately for each row to provide the best results possible for each configuration.

Tab. 3 shows ablation over regularization, for the forward-facing scenes. We find that TV regularization is important for these scenes, likely due to their low number of training images. Regularization on density has a quantitatively larger effect than regularization on spherical harmonics, but both are important for avoiding visual artifacts (see Fig. 2.3).

Tab. 2.5 compares the performance of Plenoxels when trained with the rendering formula used in NeRF (originally from Max [Max95]) and when trained with the rendering

LR Schedule	PSNR ↑	SSIM ↑	LPIPS ↓
Exp for SH, Delayed for σ [BMT+21a]	30.57	0.950	0.065
Exp for SH and σ	30.58	0.950	0.066
Exp for SH, Constant for σ	30.37	0.948	0.068
Constant for SH and σ	30.13	0.945	0.075

Table 1: **Comparison of different learning rate schedules** for σ (voxel density) and spherical harmonics (SH), with fixed resolution 256^3 and RMSProp [Hin]. Results are averaged over the 8 synthetic scenes from NeRF [MST+20]. Our method is robust to variations in learning rate schedule.

Optimizer	PSNR ↑	SSIM ↑	LPIPS ↓
RMSProp [Hin] for SH and σ	30.57	0.950	0.065
RMSProp for SH, SGD for σ	30.20	0.946	0.072
SGD for SH, RMSProp for σ	29.82	0.940	0.076
SGD for SH and σ	29.35	0.932	0.087

Table 2: **Comparison of different optimizers** for σ and SH, with fixed resolution 256^3. Results are averaged over the 8 synthetic scenes from NeRF [MST+20]. Our method is robust to variations in optimizer, although there is a benefit to RMSProp particularly for optimizing the spherical harmonic coefficients.

formula used in Neural Volumes [LSS+19a]. The Max formula is defined in Eq. (5.0.6) and rewritten here in a slightly more convenient format:

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right) \tag{5.0.2}$$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} (T_i - T_{i+1})\mathbf{c}_i \tag{5.0.3}$$

Regularizer	PSNR ↑	SSIM ↑	LPIPS ↓
TV SH, TV σ, Sparsity	26.29	0.839	0.210
- Sparsity	26.31	0.839	0.210
- TV σ	25.25	0.807	0.226
- TV SH	25.80	0.814	0.234

Table 3: **Ablation over regularization.** Results are averaged over the 8 forward-facing scenes from NeRF, which are particularly sensitive to regularization due to the low number of training views. We find that the sparsity regularizer is not necessary for quality, but we retain it to reduce memory footprint. TV regularization is essential for σ but also important for spherical harmonics, as visualized in Fig. 2.3, even though this effect is not as pronounced in the PSNR metric. Without any TV regularization (on SH or σ), three of the eight scenes run out of memory on our GPU.

The Neural Volumes formula can be written as:

$$T_i = \min\left\{1,\ \sum_{j=1}^{i-1} \exp(-\delta_i \sigma_i)\right\} \tag{5.0.4}$$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} (T_i - T_{i+1})\mathbf{c}_i \tag{5.0.5}$$

where $\exp(-\sigma_i)$ is modeled directly rather than modeling σ_i and then exponentiating (we write it in this format to make the comparison to the Max formula more clear).

These formulas only differ in their definition of the transmittance T_i. In particular, the Neural Volumes formula treats the fraction of the ray contributed by sample i as a function of the density and sampling distance of sample i only, unless the ray is already fully occluded before it exits sample i. In contrast, the contribution of sample i in the Max formula depends on the density of sample i as well as the densities of all preceding samples along the ray. In essence, opacity in the Neural Volumes formula is absolute and ray-independent (except for clipping the total contribution to 1), whereas opacity in the Max formula denotes the fraction of incoming light that each sample absorbs, a ray-dependent quantity. As we show in Tab. 2.5, the Max formula results in substantially better performance; we suspect this difference is due to its more physically-accurate modeling of transmittance.

Per-scene results

Synthetic, bounded scenes

Full, per-scene results for the 8 synthetic scenes from NeRF are presented in Tab. 5 and Fig. 2. Note that the values for JAXNeRF are from our own rerunning with centered pixels (we ran JAXNeRF in parallel across 4 GPUs and multiplied the times by 4 to account for this parallelization).

Real, forward-facing scenes

Full, per-scene results for the 8 forward-facing scenes from NeRF are presented in Tab. 6. Note that the values for JAXNeRF are from our own rerunning with centered pixels (we ran JAXNeRF in parallel across 4 GPUs and multiplied the times by 4 to account for this parallelization).

Real, 360° scenes

Full, per-scene results for the four 360° scenes from Tanks and Temples [KPZK17] are presented in Tab. 4. Note that the values for NeRF++ appear slightly different from the paper; we re-evaluated the metrics independently using VGG LPIPS and standard SSIM, from rendered images shared by the original authors.

PSNR ↑

	M60	Playground	Train	Truck	Mean
Ours	17.93	23.03	17.97	22.67	20.40
NeRF++ [ZRSK20]	18.49	22.93	17.77	22.77	20.49

SSIM ↑

	M60	Playground	Train	Truck	Mean
Ours	0.687	0.712	0.629	0.758	0.696
NeRF++	0.650	0.672	0.558	0.712	0.648

LPIPS ↓

	M60	Playground	Train	Truck	Mean
Ours	0.439	0.435	0.443	0.364	0.420
NeRF++	0.481	0.477	0.531	0.424	0.478

Optimization Time ↓

	M60	Playground	Train	Truck	Mean
Ours	25.5m	26.3m	29.5m	28.0m	27.3m

Table 4: **Full results on** 360° **scenes.**

PSNR ↑

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours	33.98	25.35	31.83	36.43	34.10	29.14	33.26	29.62	31.71
NV	28.33	22.58	24.79	30.71	26.08	24.22	27.78	23.93	26.05
JAXNeRF	34.20	25.27	31.15	36.81	34.02	30.30	33.72	29.33	31.85

SSIM ↑

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours	0.977	0.933	0.976	0.980	0.975	0.949	0.985	0.890	0.958
NV	0.916	0.873	0.910	0.944	0.880	0.888	0.946	0.784	0.893
JAXNeRF	0.975	0.929	0.970	0.978	0.970	0.955	0.983	0.868	0.954

LPIPS ↓

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours	0.031	0.067	0.026	0.037	0.028	0.057	0.015	0.134	0.049
NV	0.109	0.214	0.162	0.109	0.175	0.130	0.107	0.276	0.160
JAXNeRF	0.036	0.085	0.037	0.074	0.068	0.057	0.023	0.192	0.072

Optimization Time ↓

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours	9.6m	9.8m	8.8m	12.5m	10.8m	11.0m	8.2m	18.0m	11.1m
JAXNeRF	37.8h	37.8h	37.7h	38.0h	26.0h	38.1h	37.8h	26.0h	34.9h

Table 5: **Full results on synthetic scenes.** We compare to Neural Volumes [LSS+19a] and JAXNeRF [DBS20; MST+20].

(a) Ground Truth **(b)** Neural Volumes **(c)** JAXNeRF **(d)** Plenoxels

Figure 2: **Synthetic scenes.** We show a random view from each of the synthetic scenes, comparing the ground truth, Neural Volumes [LSS+19a], JAXNeRF [MST+20; DBS20], and our Plenoxels.

(e) Ground Truth **(f)** Neural Volumes **(g)** JAXNeRF **(h)** Plenoxels

Figure 2: **Synthetic scenes.** We show a random view from each of the synthetic scenes, comparing the ground truth, Neural Volumes [LSS+19a], JAXNeRF [MST+20; DBS20], and our Plenoxels.

PSNR ↑

	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Mean
Ours	25.46	27.83	31.09	27.58	21.41	20.24	30.22	26.48	26.29
LLFF	28.42	22.85	19.52	29.40	18.52	25.46	24.15	24.70	24.13
JAXNeRF	25.20	27.80	31.57	27.70	21.10	20.37	32.81	27.12	26.71

SSIM ↑

	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Mean
Ours	0.832	0.862	0.885	0.857	0.760	0.687	0.937	0.890	0.839
LLFF	0.932	0.753	0.697	0.872	0.588	0.844	0.857	0.840	0.798
JAXNeRF	0.798	0.840	0.890	0.840	0.703	0.649	0.952	0.890	0.820

LPIPS ↓

	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Mean
Ours	0.224	0.179	0.180	0.231	0.198	0.242	0.192	0.238	0.210
LLFF	0.155	0.247	0.216	0.173	0.313	0.174	0.222	0.193	0.212
JAXNeRF	0.272	0.198	0.151	0.249	0.305	0.307	0.164	0.235	0.235

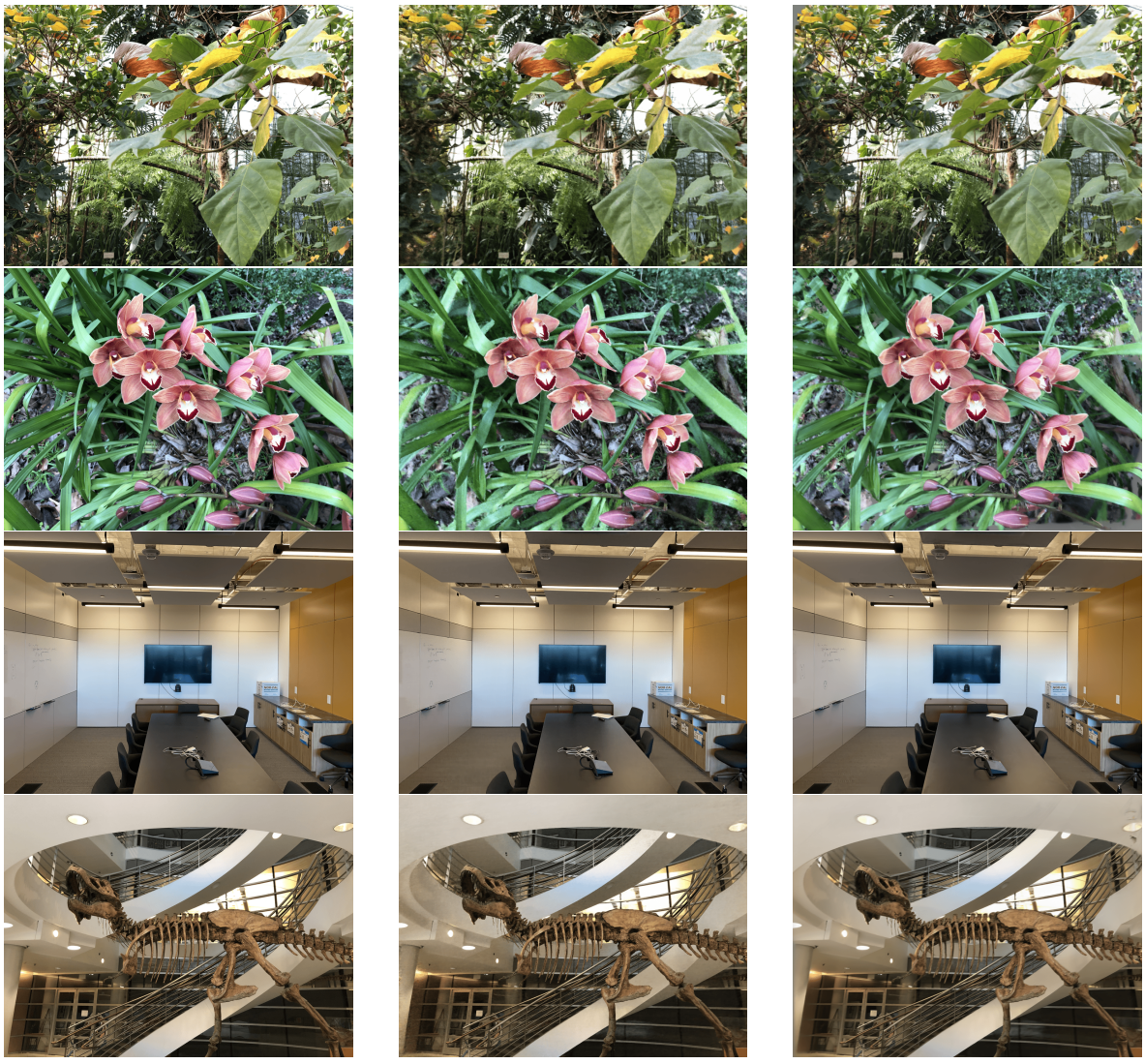
Optimization Time ↓

	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex	Mean
Ours	23.7m	22.0m	31.2m	26.3m	13.3m	23.4m	28.8m	24.8m	24.2m
JAXNeRF	38.9h	38.8h	38.6h	38.7h	38.8h	38.7h	39.1h	38.6h	38.8h

Table 6: **Full results on forward-facing scenes.** We compare to Local Light Field Fusion [MSO+19] and JAXNeRF [DBS20; MST+20].

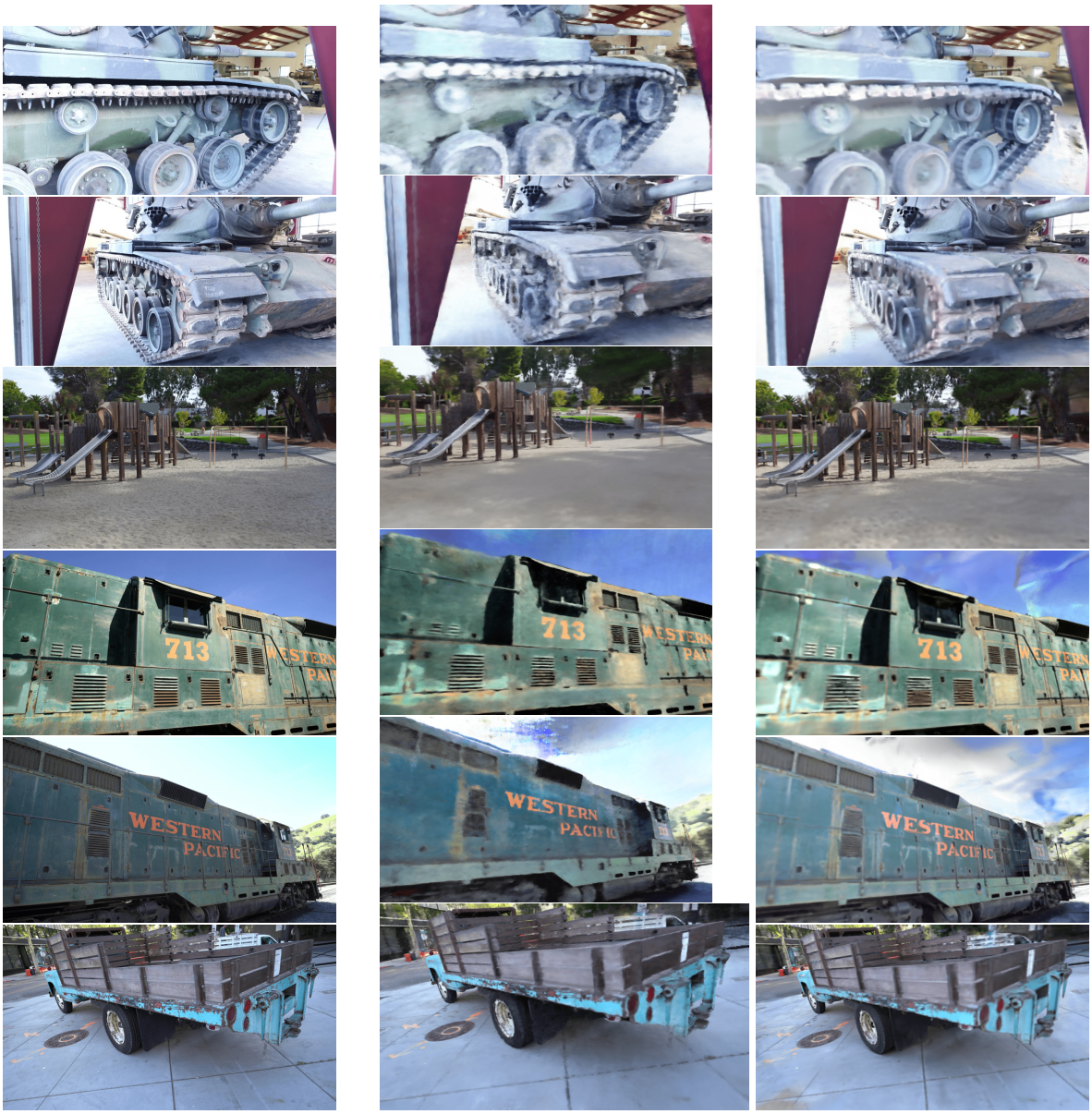
(a) Ground Truth **(b)** JAXNeRF **(c)** Plenoxels

Figure 3: **Forward-facing scenes.** We show a random view from each of the forward-facing scenes, comparing the ground truth, JAXNeRF [MST+20; DBS20], and our Plenoxels.



(d) Ground Truth **(e)** JAXNeRF **(f)** Plenoxels

Figure 3: **Forward-facing scenes.** We show a random view from each of the forward-facing scenes, comparing the ground truth, JAXNeRF [MST+20; DBS20], and our Plenoxels. Note that these two methods have different behaviors in unsupervised regions (*e.g.* the bottom right in the orchids view): JAXNeRF fills in plausible textures whereas Plenoxels default to gray.



(a) Ground Truth **(b)** NeRF++ **(c)** Plenoxels

Figure 4: 360° **scenes.** We show a random view from each of the Tanks and Temples scenes, comparing the ground truth, NeRF++ [ZRSK20], and our Plenoxels. We include two random views each for the M60 and train scenes, since the playground and truck scenes were shown in the main text.

Appendix: A Compressed Representation for Dynamic Scenes

Volumetric rendering

We use the same volume rendering formula as NeRF [MST+20], originally from [Max95], where the color of a pixel is represented as a sum over samples taken along the corresponding ray through the volume:

$$\sum_{i=1}^{N} \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)\left(1 - \exp(-\sigma_i \delta_i)\right)\mathbf{c}_i \tag{5.0.6}$$

where the first exp represents ray transmission to sample i, $1-\exp(-\sigma_i\delta_i)$ is the absorption by sample i, σ_i is the (post-activation) density of sample i, and $\mathbf{c}_i$ is the color of sample i, with distance δ_i to the next sample.

Per-scene results

Fig. 5 provides a qualitative comparison of methods for the Phototourism dataset, on the *Trevi fountain* scene. We also provide quantitative metrics for each of the three tasks we study, for each scene individually. Tab. 10 reports metrics on the static synthetic scenes, Tab. 11 reports metrics on the static real forward-facing scenes, Tab. 12 reports metrics on the dynamic synthetic monocular "teleporting camera" scenes, Tab. 13 reports metrics on the dynamic real forward-facing multiview scenes, and Tab. 14 reports metrics on the Phototourism scenes.

Ablation studies

Multiscale. In Tab. 7, we ablate our model on the static *Lego* scene [MST+20] with respect to our multiscale planes, to assess the value of including copies of our model at different scales.

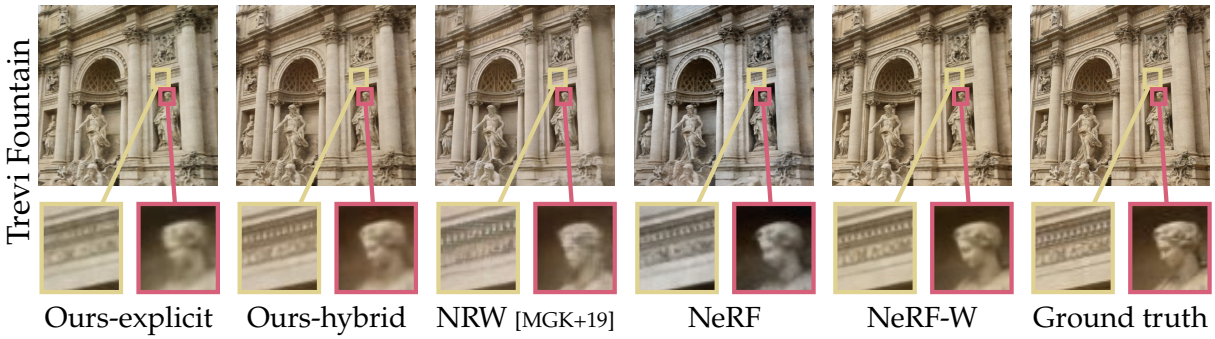


Figure 5: **Qualitative results from Phototourism dataset**. We compare our model with strong baselines. Our method captures the geometry and appearance of the scene, but produces slightly lower resolution results than NeRF-W. Note that our model optimizes in just 35 minutes on a single GPU compared to NeRF-W, which takes 2 days on 8 GPUs.

Feature length. In Tab. 8, we ablate our model on the static *Lego* scene with respect to the feature dimension M learned at each scale.

Time smoothness regularizer. Sec. 3.3.2 describes our temporal smoothness regularizer based on penalizing the norm of the second derivative over the time dimension, to encourage smooth motion and discourage acceleration. Tab. 9 illustrates an ablation study of this regularizer on the *Jumping Jacks* scene from D-NeRF [PCPM21].

Model hyperparameters

Our full hyperparameter settings are available in the config files in our released code, at `github.com/sarafridov/K-Planes`.

Scales (32 Feat. Each)	Explicit PSNR ↑	Hybrid PSNR ↑	# params ↓
64, 128, 256, 512	35.26	35.79	34M
128, 256, 512	35.29	35.75	33M
256, 512	34.52	35.37	32M
512	32.93	33.60	25M
64, 128, 256	34.26	35.07	8M

Scales (96 Feat. Total)	Explicit PSNR ↑	Hybrid PSNR ↑	# params ↓
64, 128, 256, 512	35.16	35.67	25M
128, 256, 512	35.29	35.75	33M
256, 512	34.50	35.16	47M
512	33.12	34.09	76M
64, 128, 256	34.26	35.07	8M

Table 7: **Ablation study over scales.** Including even a single lower scale improves performance, for both our explicit and hybrid models, even when holding the total feature dimension constant. Using lower scales only (excluding resolution 512^3) substantially reduces model size and yields quality much better than using high resolution alone, though slightly worse than including both low and high resolutions. This experiment uses the static *Lego* scene; in the top table each scale is allocated 32 features and in the bottom table a total of 96 features are allocated evenly among all scales.

Feature Length (M)	Explicit PSNR ↑	Hybrid PSNR ↑	# params ↓
2	30.66	32.05	2M
4	32.27	34.18	4M
8	33.80	35.12	8M
16	34.80	35.44	17M
32	35.29	35.75	33M
64	35.38	35.88	66M
128	35.45	35.99	132M

Table 8: **Ablation study over feature length M.** Increasing the feature length M learned at each scale consistently improves quality for both our models, with a corresponding linear increase in model size and optimization time. Our experiments in the main text use a mixture of $M = 16$ and $M = 32$; for specific applications it may be beneficial to vary M along this tradeoff between quality and model size. This experiment uses the static *Lego* scene with 3 scales: 128, 256, and 512.

Time Smoothness Weight (λ)	Explicit PSNR ↑	Hybrid PSNR ↑
0.0	30.45	30.86
0.001	31.61	32.23
0.01	32.00	32.64
0.1	31.96	32.58
1.0	31.36	32.22
10.0	30.45	31.63

Table 9: **Ablation study over temporal smoothness regularization.** For both models, a temporal smoothness weight of 0.01 is best, with PSNR degrading gradually with over- or under-regularization. This experiment uses the *Jumping Jacks* scene with 4 scales: 64, 128, 256, and 512, and 32 features per scale.

PSNR ↑

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours-explicit	34.82	25.72	31.2	36.65	35.29	29.49	34.00	30.51	32.21
Ours-hybrid	34.99	25.66	31.41	36.78	35.75	29.48	34.05	30.74	32.36
INGP	35.00	26.02	33.51	37.40	36.39	29.78	36.22	31.10	33.18
TensoRF	35.76	26.01	33.99	37.41	36.46	30.12	34.61	30.77	33.14
Plenoxels	33.98	25.35	31.83	36.43	34.10	29.14	33.26	29.62	31.71
JAXNeRF	34.20	25.27	31.15	36.81	34.02	30.30	33.72	29.33	31.85

SSIM ↑

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours-explicit	0.981	0.937	0.975	0.982	0.978	0.949	0.988	0.892	0.960
Ours-hybrid	0.983	0.938	0.975	0.982	0.982	0.950	0.988	0.897	0.962
INGP	-	-	-	-	-	-	-	-	-
TensoRF	0.985	0.937	0.982	0.982	0.983	0.952	0.988	0.895	0.963
Plenoxels	0.977	0.933	0.976	0.980	0.975	0.949	0.985	0.890	0.958
JAXNeRF	0.975	0.929	0.970	0.978	0.970	0.955	0.983	0.868	0.954

Table 10: **Full results on static synthetic scenes [MST+20].** We compare with Instant-NGP [MESK22], TensoRF [CXGYS22], Plenoxels [FYT+22], and JAXNeRF [DBS20; MST+20]. Dashes denote values that were not reported in prior work.

PSNR ↑

	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Mean
Ours-explicit	32.72	24.87	21.07	31.34	19.89	28.37	27.54	28.40	26.78
Ours-hybrid	32.64	25.38	21.30	30.44	20.26	28.67	28.01	28.64	26.92
NeRF	32.70	25.17	20.92	31.16	20.36	27.40	26.80	27.45	26.50
Plenoxels	30.22	25.46	21.41	31.09	20.24	27.83	26.48	27.58	26.29
TensoRF (L)	32.35	25.27	21.30	31.36	19.87	28.60	26.97	28.14	26.73
DVGOv2	-	-	-	-	-	-	-	-	26.34

SSIM ↑

	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Mean
Ours-explicit	0.955	0.809	0.738	0.898	0.665	0.867	0.909	0.884	0.841
Ours-hybrid	0.957	0.828	0.746	0.890	0.676	0.872	0.915	0.892	0.847
NeRF	0.948	0.792	0.690	0.881	0.641	0.827	0.880	0.828	0.811
Plenoxels	0.937	0.832	0.760	0.885	0.687	0.862	0.890	0.857	0.839
TensoRF (L)	0.952	0.814	0.752	0.897	0.649	0.871	0.900	0.877	0.839
DVGOv2	-	-	-	-	-	-	-	-	0.838

Table 11: **Full results on static forward-facing scenes [MSC+19].** We compare with NeRF [MST+20], Plenoxels [FYT+22], TensoRF [CXGYS22], and DVGO [SSC22]. Dashes denote values that were not reported in prior work.

PSNR ↑

	Hell Warrior	Mutant	Hook	Balls	Lego	T-Rex	Stand Up	Jumping Jacks	Mean
Ours-explicit	25.60	33.56	28.21	38.99	25.46	31.28	33.27	32.00	31.05
Ours-hybrid	25.70	33.79	28.50	41.22	25.48	31.79	33.72	32.64	31.61
D-NeRF	25.02	31.29	29.25	32.80	21.64	31.75	32.79	32.80	29.67
T-NeRF	23.19	30.56	27.21	32.01	23.82	30.19	31.24	32.01	28.78
Tensor4D	-	-	-	-	26.71	-	36.32	34.43	-
TiNeuVox	28.17	33.61	31.45	40.73	25.02	32.70	35.43	34.23	32.67
V4D	27.03	36.27	31.04	42.67	25.62	34.53	37.20	35.36	33.72

SSIM ↑

	Hell Warrior	Mutant	Hook	Balls	Lego	T-Rex	Stand Up	Jumping Jacks	Mean
Ours-explicit	0.951	0.982	0.951	0.989	0.947	0.980	0.980	0.974	0.969
Ours-hybrid	0.952	0.983	0.954	0.992	0.948	0.981	0.983	0.977	0.971
D-NeRF	0.95	0.97	0.96	0.98	0.83	0.97	0.98	0.98	0.95
T-NeRF	0.93	0.96	0.94	0.97	0.90	0.96	0.97	0.97	0.95
Tensor4D	-	-	-	-	0.953	-	0.983	0.982	-
TiNeuVox	0.97	0.98	0.97	0.99	0.92	0.98	0.99	0.98	0.97
V4D	0.96	0.99	0.97	0.99	0.95	0.99	0.99	0.99	0.98

Table 12: **Full results on monocular "teleporting-camera" dynamic scenes.** We use the synthetic scenes from D-NeRF [PCPM21], which we refer to as monocular "teleporting-camera" because although there is a single training view per timestep, the camera can move arbitrarily between adjacent timesteps. We compare with D-NeRF [PCPM21], T-NeRF [PCPM21], Tensor4D [SZT+22], TiNeuVox [FYW+22], and V4D [GXHCY22]. Dashes denote unreported values. TiNeuVox trains in 30 minutes, V4D in 4.9 hours, D-NeRF in 2 days, and Tensor4D for an unspecified duration (Tensor4D reports iterations rather than time). Our reported results were obtained after roughly 1 hour of optimization on a single GPU. Like D-NeRF and TiNeuVox, we train and evaluate using half-resolution images (400 by 400 pixels).

PSNR ↑							
	Coffee Martini	Cook Spinach	Cut Beef	Flame Salmon[1]	Flame Steak	Sear Steak	Mean
Ours-explicit	28.74	32.19	31.93	28.71	31.80	31.89	30.88
Ours-hybrid	29.99	32.60	31.82	30.44	32.38	32.52	31.63
LLFF	-	-	-	23.24	-	-	-
DyNeRF	-	-	-	29.58	-	-	-
MixVoxels-L[†]	29.36	31.61	31.30	29.92	31.21	31.43	30.80
SSIM ↑							
	Coffee Martini	Cook Spinach	Cut Beef	Flame Salmon[1]	Flame Steak	Sear Steak	Mean
Ours-explicit	0.943	0.968	0.965	0.942	0.970	0.971	0.960
Ours-hybrid	0.953	0.966	0.966	0.953	0.970	0.974	0.964
LLFF	-	-	-	0.848	-	-	-
DyNeRF	-	-	-	0.961	-	-	-
MixVoxels-L	0.946	0.965	0.965	0.945	0.970	0.971	0.960

[†] Very recent/concurrent work. MixVoxels was released in December 2022. [1]Using the first 10 seconds of the 30 second long video.

Table 13: **Full results on multiview dynamic scenes [LSZ+22].** We compare with Local Light Field Fusion [MSC+19], DyNeRF [LSZ+22], and MixVoxels [WTLTL22]. Dashes denote unreported values. Note that our method optimizes in less than 4 GPU hours, whereas DyNeRF trains on 8 GPUs for a week, approximately 1344 GPU hours.

PSNR ↑				
	Brandenburg Gate	Sacre Coeur	Trevi Fountain	Mean
Ours-explicit	24.85	19.90	22.00	22.25
Ours-hybrid	25.49	20.61	22.67	22.92
NeRF-W [MRS+21]	29.08	25.34	26.58	27.00
NeRF-W (public)†	21.32	19.17	18.61	19.70
LearnIt [TMW+21b]	19.11	19.33	19.35	19.26
MS-SSIM ↑				
	Brandenburg Gate	Sacre Coeur	Trevi Fountain	Mean
Ours-explicit	0.912	0.821	0.845	0.859
Ours-hybrid	0.924	0.852	0.856	0.877
NeRF-W	0.962	0.939	0.934	0.945
Nerf-W (public)†	0.845	0.752	0.694	0.764
LearnIt	-	-	-	-

† Open-source version `https://github.com/kwea123/nerf_pl/tree/nerfw` where we implement the test-time optimization ourselves exactly as for k-planes. NeRF-W code is not public.

Table 14: **Full results on phototourism scenes.** Note that our results were obtained after about 35 GPU minutes, whereas NeRF-W trains with 8 GPUs for two days, approximately 384 GPU hours.