

LoopNeRF: Exploring Temporal Compression for 3D Video Textures

Alexander Kristoffersen



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-118

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-118.html>

May 11, 2023

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

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

Acknowledgement

Thank you to Professor Gonzalez and Sky Computing for the opportunity to do research at a graduate level here at Berkeley. Without this 5th year at Berkeley, many exciting opportunities I can see in my future would not have been available to me. I thank the Professor Kanazawa for the mentorship, and to KAIR for being so inviting. Thank you to the Nerfstudio team for the opportunity to build high-impact research OSS. Thank you to Matthew and Ozan for being the fellow grad students in my life, and for being great friends through it all. Finally, I thank my brother Matt and my parents for supporting me throughout my academic career.

LoopNeRF: Exploring Temporal Compression for 3D Video Textures

by

Alexander Kristoffersen

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

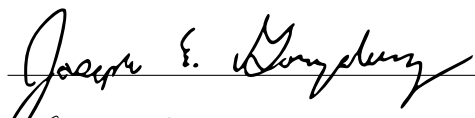

University of California, Berkeley

Committee in charge:

Professor Joseph Gonzalez, Chair
Professor Angjoo Kanazawa

Spring 2023

The dissertation of Alexander Kristoffersen, titled LoopNeRF: Exploring Temporal Compression for 3D Video Textures, is approved:

Chair		Date	<u>5/11/2023</u>
		Date	<u>5/11/2023</u>
	_____	Date	_____

University of California, Berkeley

LoopNeRF: Exploring Temporal Compression for 3D Video Textures

Copyright 2023
by
Alexander Kristoffersen

Abstract

LoopNeRF: Exploring Temporal Compression for 3D Video Textures

by

Alexander Kristoffersen

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Joseph Gonzalez, Chair

Neural Radiance Fields (NeRFs) have been shown to be effective representations for static scenes, with the newest methods able to generate photo-realistic renderings from novel views from casual captures. Dynamic view synthesis (DVS) via NeRF representations, however, have not seen similar results, with many SOTA methods requiring complicated capture setups synonymous with multi-view, synchronous captures. In this paper, we explore a related problem: given a trained dynamic NeRF, can we distill the dynamicism of the scene into a more compact representation?

To this end, we present LoopNeRF, a method to compress a dynamic NeRF trained on a long monocular video capture by significantly reducing the time-dependent parameters. The resulting model can then be used to generate infinitely looping dynamic renderings from novel views: 3D Video Textures.

Contents

Contents	i
List of Figures	iii
List of Tables	v
1 Introduction	1
2 Related Work	2
2.1 Video Textures and Cinemagraphy	2
2.2 Explicit Dynamic Representations	2
2.3 Dynamic NeRF Representations	3
2.4 3D Cinemagraphs	3
2.5 NeRF Compression and Distillation	4
3 LoopNeRF	5
3.1 Problem Statement	5
3.2 Preliminaries	5
3.3 Method	6
3.4 Temporal Distillation	8
3.5 Implementation Details	12
3.6 Comparison to VideoLoop3D	13
4 Experiments and Results	15
4.1 Implementation Details	15
4.2 Monocular Results	15
4.3 VideoLoop3D Comparison	16
4.4 Ablations	16
5 Discussion	23
5.1 Why Does This Work?	23
5.2 Limitations and Future Work	25
5.3 Conclusion	26

Bibliography

List of Figures

3.1	NeRF Rendering Equation: For a given ray r , we sample s_i across the ray. Colors c_i and densities σ_i are evaluated for each s_i and integrated into a final color $C(r)$.	6
3.2	Distillation Method: In distillation, the static planes, feature decoders, and appearance embeddings are shared and frozen, while the dynamic planes are replaced with a much smaller resolution proportional to the number of frames in the looping 3D video texture.	7
3.3	Distillation Video Selection: Regions of significant groundtruth supervision are selected for rendering from a fully trained dynamic k -Planes. These are regions in the camera trajectory where there is minimal camera rotation and translation.	9
3.4	VideoLoop3D Looping Loss: Right: Temporally overlapping 3D patches are extracted from both the groundtruth sampled video V_p and corresponding rendered video \hat{V}_0 . Left: For each source patch, a similarity score is calculated for every target patch. A loss is computed between every source patch and their nearest neighbor target patch. The looping effect is the result of re-use of patches across the temporal boundary, denoted by the blue arrow.	11
3.5	Multi-resolution Sampling and Training: Memory limitations require small 3D patches to be extracted and rendered during training. Coarse-to-fine training allows for the model to benefit from both the spatial consistency of large patches and the high-frequency detail reconstruction of small patches.	12
4.1	Monocular Capture Results: Left: For each scene, we render from the initial dynamic k -Planes model (left), the LoopNeRF model (middle), and a depth map from the LoopNeRF model (right). We sample a horizontal line at every timestep and visualize it over time. Top: <i>Berkeley Pond</i> , Bottom: <i>Berkeley Stream</i> . . .	18
4.2	Monocular Capture Results: <i>Berkeley Ledge</i>	19
4.3	Monocular Capture Results: Top: <i>Oakland Fire</i> , Bottom: <i>Berkeley Storefront</i> .	20
4.4	VideoLoop3D Comparison: When rendering scenes far away from training views, VideoLoop3D results in block-like artifacts (highlighted in red). Top: <i>Pillar720p</i> , Bottom: <i>Towerd720p</i>	21
4.5	Multi-resolution Training Ablation: Left: Coarse-to-fine patch sampling enabled. Right: Baseline. Note that without course-to-fine training, a hole inside the awning is rendered, showing the lack of spatial consistency.	22

5.1	Fixed Proposal Networks: The proposal networks learned in the initial k -Planes network learn to sample close to surface geometry. In distillation, this minimizes the generation of floaters, as areas far away from the learned scene surface are rarely sampled during training.	24
5.2	Failure Case: If the initial K -Planes has learned incorrect static geometry, it will not be corrected in the distilled LoopNeRF model, resulting in the same artifacting. See the discontinuities on the utility pole and the ghosting artifacts near the traffic cones.	25

List of Tables

3.1	Architecture Comparisons. LoopNeRF combines many of the features of VideoLoop3D, but with a analogous volumetric versions.	13
4.1	Model Parameters Comparison: LoopNeRF uses a comparable number of parameters to VideoLoop3D, but learns a volumetric representation that can learn arbitrary scene layouts.	17
4.2	Comparison and Ablation: We show better reconstruction qualities in most metrics compared to VideoLoop3D. When adding multi-resolution training, these metrics only marginally decline.	17

Acknowledgments

Thank you to Professor Gonzalez and Sky Computing for the opportunity to do research at a graduate level here at Berkeley. Without this 5th year at Berkeley, many exciting opportunities I can see in my future would not have been available to me. I thank the Professor Kanazawa for the mentorship, and to KAIR for being so inviting. Thank you to the Nerfstudio team for the opportunity to build high-impact research OSS. Thank you to Matthew and Ozan for being the fellow grad students in my life, and for being great friends through it all. Finally, I thank my brother Matt and my parents for supporting me throughout my academic career.

Chapter 1

Introduction

When training static Neural Radiance Fields (NeRFs), the underlying assumption is that the scene being captured remains static while the camera moves around it. The real world, however, is constantly in motion. For instance, a scene with a tree-lined sidewalk contains numerous branches and leaves swaying in the breeze, and a pond scene features ripples on the water surface. These are the details that truly bring a scene to life, yet they are not accurately represented in static NeRF reconstructions.

Dynamic NeRFs face different problems. In a monocular video capture of a dynamic scene, both the camera and the scene can move independently in every frame, making the reconstruction objective in NeRF training under-constrained. Although various techniques have been developed to address this issue, such as introducing regularizers or physics-based priors on motion and deformations, the core problem persists - each timestep has only a single groundtruth view for supervision [8]. This presents a significant challenge in accurately capturing the complex dynamics of a scene.

Drawing insights from the video texture literature, we argue that compelling and plausible dynamicism can be represented in dynamic NeRFs without reconstructing the true movement within the scene exactly. In the tree or pond scene examples, branches sway back and forth and ripples repeat continuously. The movement in the natural world is often cyclical, allowing for strong priors on what future motion may look like given some finite information of the previous motion.

This report makes the following contributions:

1. We re-frame the problem of training compelling dynamic NeRFs on long-form monocular videos as a distillation problem.
2. We introduce LoopNeRF, a method for compressing a trained dynamic NeRF model into one with significantly fewer time-dependent parameters. The resulting model can generate looping dynamic renderings from novel viewpoints, i.e. 3D video textures.

Chapter 2

Related Work

2.1 Video Textures and Cinemagraphy

The popularity of looping, semi-static videos or cinemagraphs has soared with the advent of the GIF format. Unlike still photos, cinemagraphs capture sparse motion, making the subject come alive and catching the viewer’s eye.

There are many existing methods for creating these cinemagraphs, or video textures, from either video or static images. Initial methods relied on careful editing between short video segments and photographs by selecting candidate frames from a video and using user-defined masks for static and moving parts of the scene. Automatic methods aim to generate these cinemagraphs from a single input video or image without user input. Classical approaches such as Video Textures [18] detect repeatable loops of frames from an input video and create seamless loop animations from them. Deep learning methods such as Eulerian Motion Fields [9] learn from a dataset of similar movements to predict a motion field that can be forward and backward integrated to generate reasonable frames of a looping video.

Although visual textures offer a compelling visual effect, they work solely in the image plane and lack a concept of depth within the scene, restricting the viewer’s experience to a 2D image. For use in 3D representations, additional 3D consistency techniques are necessary.

2.2 Explicit Dynamic Representations

There has been significant research in extending image-based rendering techniques to represent video sequences through a variety of parameterizations. Explicit representations are compact and generally quick to render, and are therefore especially useful for real-time use-cases. Multi-planar images (MPIs), and their dynamic counterpart multi-planar videos (MPVs), parameterize the world as semi-transparent planes discretizing depths within the scene. While they are an efficient representation, they are limited in the camera views they can synthesis successfully and are generally limited to forward facing scenes. In Deep Multi Depth Panoramas [13], MPIs are extended to panoramic images to represent a larger diver-

sity of scenes, such as those that could be represented with most modern NeRF techniques. MatryODShka [2] proposes multi-spherical images (MSIs) to further increase the effectiveness of depth-discretized representations. For dynamic scenes, Broxton et. al. [4] generates a dynamic mesh from a learned dynamic MSI, allowing for dynamic 6DOF view synthesis at real-time framerates. The training of such a representation is expensive and requires a custom synchronous multi-camera setup. VideoLoop3D [14] applies MPVs to synchronous static videos, producing similar photo-realistic real-time results to Broxton, but with a simple training method and more casual capture requirements. LoopNeRF can be trained from casual captures of dynamic scenes, and maintains a compact, explicit backbone.

2.3 Dynamic NeRF Representations

NeRF has been extended to represent dynamic scenes. Early dynamic NeRF methods simply added time as an extra input with 3D position and viewing angle, or used time to transform a ray into a learned canonical space [17, 16]. These methods rely on the ability for the underlying MLP to memorize the scene, but therefore may not generalize to unseen viewing angles or times. HyperNeRF [15] modifies these deformation fields to deal with topology changes more effectively. DyNeRF [11] learns a time-dependent latent code across frames of synchronized videos. NeRFPlayer [19] learns both a deformation field as well as additional newness and decomposition fields to represent a dynamic scene.

Despite these advances, dynamic scene reconstruction is still under-constrained without multi-view video inputs. To address this, additional constraints have been added to the reconstruction objective, such as using learned splines [10] or restricting the domain to scenes of known objects such as humans [22]. k -Planes restricts the magnitude of the time-dependent parameters of the network with an L1 regularization for sparsely dynamic scenes. However, no existing method can reconstruct arbitrary dynamic scenes from a single casual monocular video. Recognizing this as a data limitation rather than an architectural hurdle, LoopNeRF tackles an easier problem: representing dynamic scenes where the majority of the scene is static and the dynamic motion is natural and reasonably loopable.

2.4 3D Cinemagraphs

Video loop generation has been previously extended to allow for varying input views by approximating a 3D structure. Panoramic Video Textures [1] allow for rotations of the view camera by applying Video Textures to a stitched panoramic video. 3D Cinemagraphy [12] uses similar techniques to Holynski et. al., but with an additional depth map input to allow for slight amounts of camera motion in rendered loops. VideoLoop3D [14] is the most similar method to LoopNeRF, approximating 3D video view synthesis by decomposing forward-facing scenes into static and looping dynamic tiles at discrete depths.

Unlike LoopNeRF, which uses a true 3D volumetric representation, these methods can only approximate the geometry of a scene. This limits their ability to produce photo-realistic results and the range of viable baseline input. LoopNeRF, on the other hand, can work with casual data from any real-world monocular capture, making it capable of capturing a much wider variety of captures while maintaining all the benefits that come with volumetric rendering techniques.

2.5 NeRF Compression and Distillation

NeRF models can require a significant number of parameters compared to other NVS techniques. TensorRF [6] learns a compressed representation of a scene by factorizing a 3D volume with a collection of 2D and 1D vectors. However, such methods have decreased quality outside of synthetic scenes. Similarly, k -Planes [7] learns a compressed representation of Hex-Planes to represent any dimension of space, and has been shown to generate efficient representations for both synthetic and real scenes. [21] parameterise dynamic scenes in the frequency domain through use of the Fourier transform. While these methods are effective at learning compressed representations of scenes, they may still be limited in the frequencies within the input data that they can represent. Further, the dynamic version of these representations may still require multi-view or effective multi-view data to produce reasonable reconstructions.

Chapter 3

LoopNeRF

3.1 Problem Statement

LoopNeRF aims to distill a dynamic NeRF trained on long monocular video of a scene containing natural motion into a representation with significantly less time-dependent parameters. This distilled model will be able to render 3D video textures that contain looping and representative motion found in the original video capture.

We assume that candidate captures for this method contain some natural, loopable motion. This motion, such as the flowing of water or swaying of trees, are both visually interesting and semi-periodic, making them perfect candidates for 3D video texture creation.

3.2 Preliminaries

LoopNeRF builds on top of extensive research in neural radiance fields, or NeRFs. We will briefly outline the NeRF representation and rendering equations. A NeRF model F_{Θ} takes in a point $\mathbf{p} \in \mathbb{R}^3$ in space and viewing direction $\mathbf{d} = (\theta, \phi)$ as input, and returns two values: a color c and volumetric density σ .

$$F_{\Theta}(\mathbf{p}, \mathbf{d}) = \mathbf{c}, \sigma$$

A NeRF model is primarily supervised by camera rays extracted from posed input images. To render the color of ray \mathbf{r} , F_{θ} is sampled repeatedly at multiple sampled positions across the ray. We accumulate the samples $\mathbf{s}_i = (\mathbf{p}_i, \mathbf{d})$ across the ray from nearest to farthest following this formula:

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

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

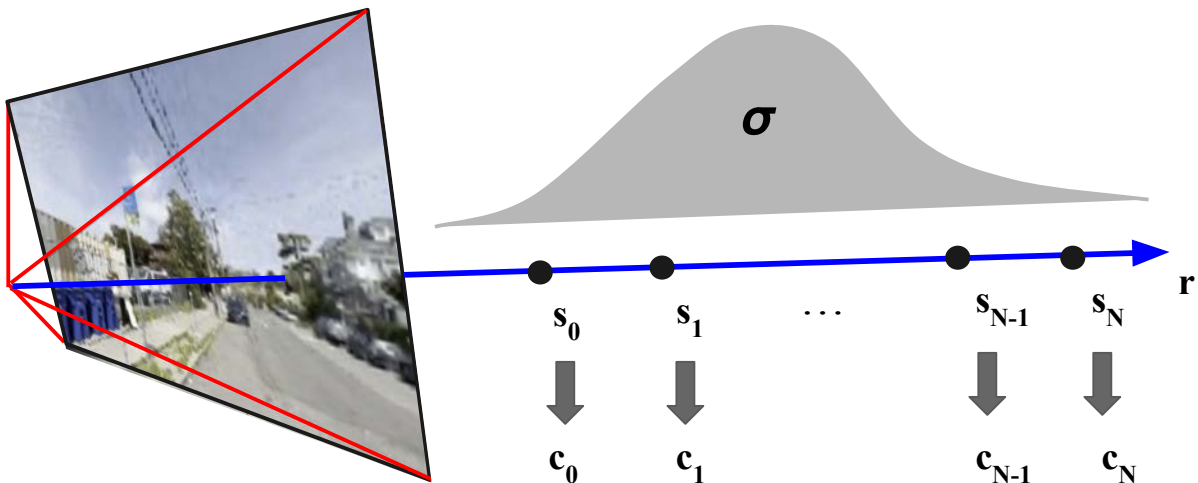


Figure 3.1: NeRF Rendering Equation: For a given ray r , we sample s_i across the ray. Colors c_i and densities σ_i are evaluated for each s_i and integrated into a final color $C(r)$.

where $\mathbf{c}_i, \sigma_i = F_{\Theta}(\mathbf{p}_i, \mathbf{d})$ and δ_i is the distance on the ray \mathbf{r} between consecutive samples \mathbf{s}_i and \mathbf{s}_{i+1} . An L2 reconstruction loss is applied between the rendered ray color $C(\mathbf{r})$ and the ground-truth color from the image pixel corresponding to \mathbf{r} .

As NeRFs are generally parameterized by a neural network, rendering a single ray results in multiple expensive network evaluations. To alleviate this, sampling methods have been developed to both minimize the required number of samples and optimize their locations. Sampling methods aim to place samples at locations in space where the ray intersects a surface and away from free space, as samples within free space have minimal densities and therefore will not contribute to the final ray color. With proposal sampling as described in Mip-NeRF 360 [3], an additional network is used to learn where to sample, and is supervised by the histogram of density weights generated from main NeRF network.

3.3 Method

LoopNeRF can be separated into two sections: initial training and distillation. In the initial model training, the LoopNeRF model is trained on the entire dataset with the aim to capture and memorize as much information about the scene as possible. We then distill the temporal parameters with the aim to represent the dynamicism of the scene into a looping 3D video texture.

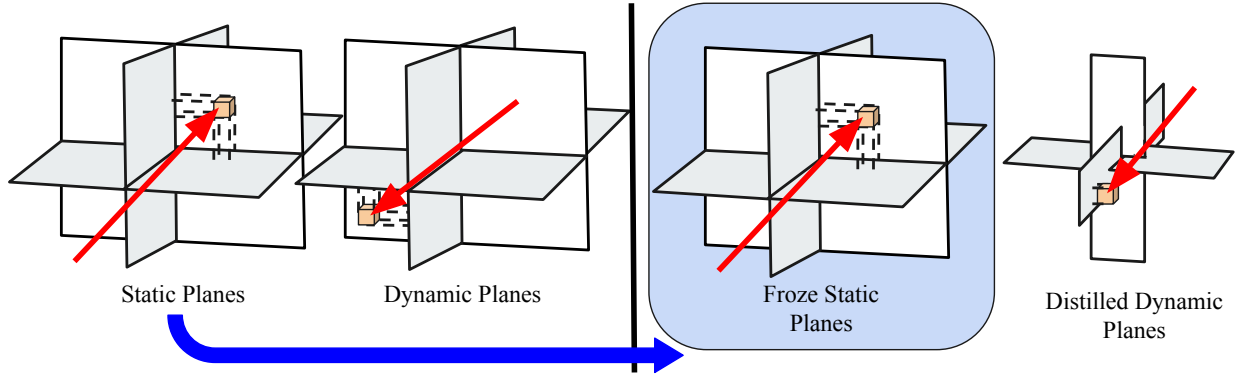


Figure 3.2: Distillation Method: In distillation, the static planes, feature decoders, and appearance embeddings are shared and frozen, while the dynamic planes are replaced with a much smaller resolution proportional to the number of frames in the looping 3D video texture.

LoopNeRF Architecture

LoopNeRF uses a dynamic k -Planes (Hex-Plane) [7, 5] model as the underlying backbone. The k -Planes representation factorizes a 4D volume as 6 multi-scale feature grids, each representing a unique combination of two input variables. To evaluate the model at point $\mathbf{p} = (x, y, z, t)$ and viewing direction $\mathbf{d} = (\theta, \phi)$, we sample each of the planes with bilinear interpolation. These features are combined via a hadamard product (3.1), and concatenated with the features from all scales and encoded viewing direction (3.3). This final feature is then decoded into output RGB and volumetric density through an MLP (3.4).

$$f(\mathbf{p}) = \prod_{i=1}^6 \text{Plane}_i(\mathbf{p}) \quad (3.1)$$

$$\text{MLP}_{\text{geo}}(f(\mathbf{p})) = f_{\text{geo}}, \sigma \quad (3.2)$$

$$f(\mathbf{p}, \mathbf{d}) = \text{CONCAT}(f_{\text{geo}}, \text{Enc}(\mathbf{d})) \quad (3.3)$$

$$F_{\Theta}(\mathbf{p}, \mathbf{d}) = \{\text{MLP}_{\text{decoder}}(f(\mathbf{p}, \mathbf{d})), \exp(\sigma(\mathbf{p}))\} \quad (3.4)$$

The k -Planes architecture is a compact representation of a d -dimensional space. However, due to its explicit feature grid backbone, the maximum frequencies a Hex-Plane model can store and represent is determined by the maximum resolution for each plane dimension. As will be described later, this may be a limitation when learning scenes of significant length in time.

3.4 Temporal Distillation

Given a long monocular video of T frames, LoopNeRF first learns a dynamic NeRF f_{Θ} . This is then distilled into a representation \hat{f}_{Θ^*} that has significantly fewer time-dependent parameters. To achieve this, we first recognize that the amount of temporal information that can be represented within a dynamic NeRF scales with the temporal resolution of its time-dependent parameters. Taking k -Planes as an example, a model that was able to lossless represent T frames of a monocular video, where each frame has a unique time associated with it, would require feature grids with a temporal resolution at least T . Further, if the input video recorded at z frames per second, then per the Nyquist Sampling Theorem, such a k -Planes and video could only represent frequencies smaller than $z/2$. This means that for long-form videos with high frequency dynamicism, explicit-backed representations such as k -Planes would require large temporal resolutions to accurately represent the highest frequencies.

Naive Solution

There are immediate difficulties in distilling f_{Θ} into \hat{f}_{Θ^*} . For a reasonable distillation, we argue that \hat{f}_{Θ^*} must maintain the ability to represent both the static objects and the high frequency motion within a scene. A naive solution would be to simply reduce the temporal resolution to F , where $T \gg F$. The new frame timestamps are then the original timestamps modulo the new temporal resolution F :

$$t' = t \pmod{F}$$

When training with this naive distillation method, the resulting model exhibits both spatial blurriness and floater artifacts. This is due to an incorrect assumption that the monocular video can be treated as multiple synchronized videos, each containing F frames captured at the same times. Since the objects in the scene may not be in the same spatial location every F frames, this inconsistency leads to the appearance of artifacts during training. To avoid these issues, a correct distillation method must account for the asynchronous nature of the input data.

Our Solution

Re-initialization

In order to capture the sparse high frequency movement within the scene while maintaining its static parts during distillation, we reinitialize the time-dependent parameters of a trained dynamic NeRF and freeze its static parameters. Freezing only the time-independent parameters within a dynamic NeRF is generally non-trivial, but is easy with the k -Planes architecture due to its static-dynamic decomposition. We freeze the 3 spatial planes of f_{Θ} (xy , yz , and xz), along with the feature decoder MLPs and direction encoder. We find that

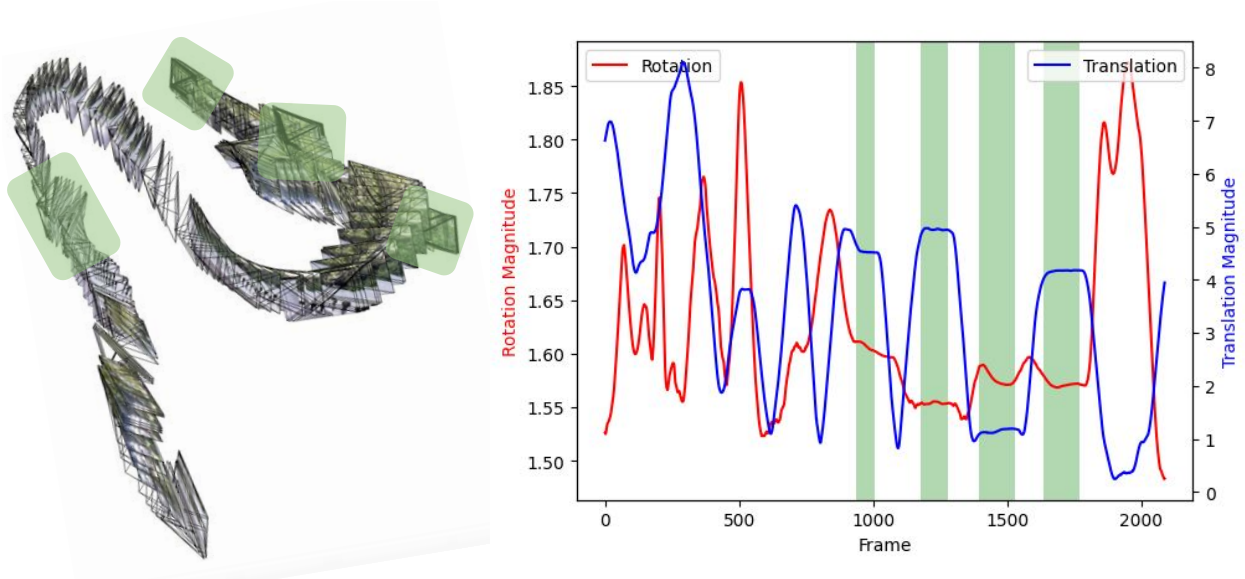


Figure 3.3: Distillation Video Selection: Regions of significant groundtruth supervision are selected for rendering from a fully trained dynamic k -Planes. These are regions in the camera trajectory where there is minimal camera rotation and translation.

freezing the decoders are especially important in distillation training as it guarantees that the rendered results in fully static parts of the scene are unchanged. Re-initialization of the dynamic parameters of \hat{f}_{Θ^*} by replacing the dynamic planes with ones of a much smaller resolution in the time axis. This step is visualized in Figure 3.2.

The original dynamic k -Planes also uses an additional set of dynamic k -Planes to back a proposal sampler. This allows for the proposal sampler to vary sample placement over time. We discard this feature in the distilled k -Planes by re-initializing the time-dependent planes into all 1s and freezing the entire proposal model. As the k -Planes combined plane features via a Hadamard product, this 1s initialization forces the proposal sampler to only rely on the static information of the scene for sample placement. We discuss the importance of this design choice in 5.1.

Distillation View Selection

We supervise the distilled dynamic k -Planes \hat{f}_{Θ^*} with rendered videos from the original model f_{Θ} . As noted in Hang et. al [8], dynamic NeRF models have difficulty interpolating views and times far from ground truth captures. To sidestep this issue, we select for views within the scene that should be reasonably memorized by f_{Θ} . Following the path of the input monocular camera, locations where both camera rotation and translation are small will lead to sharp results in the trained NeRF, as these are views and times in which there

is significant supervision. In this way, we use f_{Θ} more as a video stabilizer, evaluating from poses that have minimal baselines from many input camera poses.

As seen in Figure 3.3, we use a simple heuristic to determine what frame segments of the monocular capture are suitable for distillation. We find that thresholding the magnitude of the change in translation and Euler angle rotations between frames gives suitable candidate frame segments. For more consistent results, we also employ a rolling average to these changes and remove segments that are smaller than some minimum segment length. The threshold value, rolling average window, and minimum segment lengths are all hyperparameters tuned for every capture.

For every selected segment s_i , we determine a average pose p_i by averaging the poses across every encompassing frame. We re-normalize the averaged rotation matrix via QR decomposition. We render each segment and corresponding pose from the full k -Planes model f_{Θ} , resulting in multiple asynchronous static videos.

The success of view selection, as with most dynamic NeRF methods, wholly depends on the data. A capture where camera is constantly in motion would result in no selected segments. If not enough segments are detected, the selection hyperparameters must be relaxed. This tends to result in selection renders that are of less quality, as the segment poses and times are further from groundtruth video frames. This method also assumes that the dynamic motion of the scene is reasonably captured within the selected segments. In practice, we find that many of these limitations are not an issue with casual monocular video captures that LoopNeRF is tailored to work on: many scenes have some segments of minimal camera movement, and dynamic parts of the scene are usually captured while the camera is fairly still.

Looping Loss

Supervising \hat{f}_{Θ^*} directly from the selected static video segments requires a different loss than the standard L2 loss used in standard NeRF training. We rely on the looping loss proposed by VideoLoop3D to combine the motion expressed in each of the rendered video segments.

In every iteration, we select at random a video patch from one of the video segments, $V_p \in \mathbb{R}^{T \times h \times w \times 3}$, where T is the number of frames in the video segment. We render a corresponding video segment from \hat{f}_{Θ^*} , $\hat{V}_0 \in \mathbb{R}^{F \times h \times w \times 3}$, where T corresponds to the number of frames in the loop we wish to train for, and is usually significantly smaller than F . We extract temporally overlapping 3D patches from both the groundtruth video V_p and \hat{V}_0 . To extract 3D patches that extend past the length of \hat{V}_0 , we circularly pad the end of \hat{V}_0 . This ensures that \hat{f}_{Θ^*} will be temporally loopable and have no sharp visual boundaries at the end of the loop. Using VideoLoop3D’s notation, we extract patches $\{Q_i; i = 1, \dots, n\}$ from \hat{V}_0 and $\{K_j; j = 1, \dots, m\}$ from V_p , where each patch contains d frames and have stride s .

We match source patches $\{Q_i\}$ to target patches within $\{K_j\}$ by minimizing their bidirectional similarities. This is done by first calculating a normalized similarity score for every (Q_i, K_j) . The nearest neighbor for every source patch Q_i is the target patch K_j with the

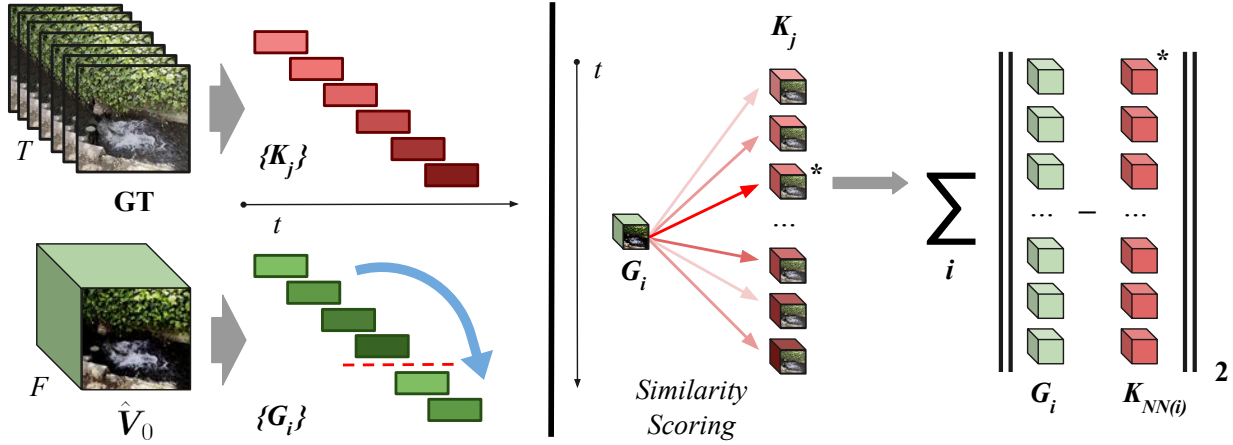


Figure 3.4: VideoLoop3D Looping Loss: Right: Temporally overlapping 3D patches are extracted from both the groundtruth sampled video V_p and corresponding rendered video \hat{V}_0 . Left: For each source patch, a similarity score is calculated for every target patch. A loss is computed between every source patch and their nearest neighbor target patch. The looping effect is the result of re-use of patches across the temporal boundary, denoted by the blue arrow.

minimal normalized similarity score. The final output of the looping loss is the MSE between every source patch and its nearest neighbor $K_{NN(i)}$ (Eq. 3.5).

$$\mathcal{L}_{\text{loop}} = \frac{1}{nhw} \sum_{\text{pixel}} \sum_{i=1}^n \|Q_i - K_{NN(i)}\|_2^2 \quad (3.5)$$

Multi-resolution Sampling and Training

Memory constraints limit the size of patches that can be used in the looping loss, as it requires the rendering and gradient storage for backpropagation of $n \times h \times w$ pixels from \hat{f}_{Θ^*} . This grows quadratically in the length of a square patch, thus the memory requirements are quickly met when increasing patch sizes. This is a major roadblock for this method, as large patches tend to result in cleaner and more spatially-consistent results. Small patch sizes can result in spatial discontinuities in the trained 3D video texture, as the nearest neighbor patches selected for some small patch may not align with the target patch selections chosen elsewhere. A different problem arises when using large patches downsampled to h_{max} and w_{max} : large patches will result in spatial consistency, but the final video texture will be blurry and lack high frequency details.

We get the best of both worlds by training in a coarse-to-fine manner. Early in training, we use large patch sizes that are then downsampled with bilinear interpolation to

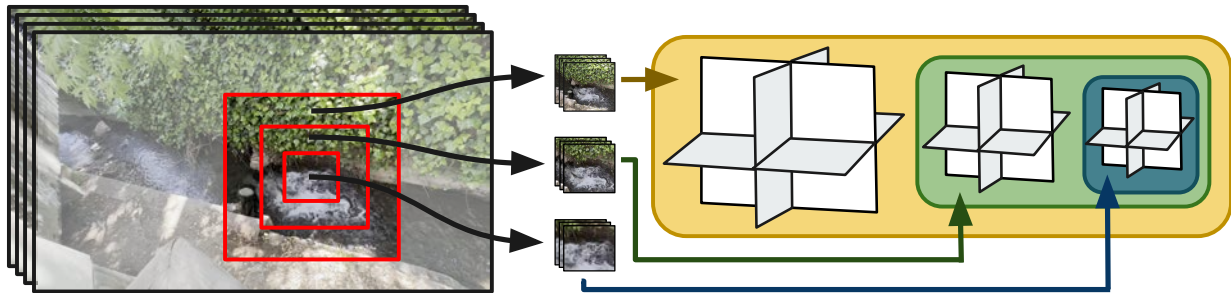


Figure 3.5: Multi-resolution Sampling and Training: Memory limitations require small 3D patches to be extracted and rendered during training. Coarse-to-fine training allows for the model to benefit from both the spatial consistency of large patches and the high-frequency detail reconstruction of small patches.

(h_{\max}, w_{\max}) . As training continues, we reduce the patch size by a factor of 2 every 1000 iterations until $(h, w) = (h_{\max}, w_{\max})$. We take advantage of the multi-resolution architecture of k -Planes by varying which planes are supervised over training. In the first 1000 iterations, all spatial resolutions of the dynamic planes are supervised. Every patch size update, the lowest-resolution of unfrozen dynamic planes are frozen. This occurs until all but the highest frequency planes are being trained. Intuitively, this freezes the low-frequency shapes within the scene to learn their lowest frequency loopable motion early in training, and leaves the higher frequency details to be optimized later in training.

LoopNeRF can both represent the lowest frequencies of a scene as stored in the initial dynamic k -Planes, as well as represent the sparse high frequency information within the scene. Unlike the naive distillation method, LoopNeRF does not incorrectly assume synchronization of the input data. Rather, it works by merging the dynamic motion found in multiple asynchronous videos rendered from a pseudo-groundtruth trained dynamic k -Planes. With view selection and distillation, casual monocular captures can be used to create compelling DVS renders, unlike previous methods which require multi-view setups or static captures.

3.5 Implementation Details

LoopNeRF is built on top of a k -Planes implementation built within the Nerfstudio framework [20], with the looping loss code used from VideoLoop3D [14]. We use the standard hyperparameters for training used in k -Planes Dynamic. We increase the base resolutions of both the static and dynamic planes to 128 and 1536 respectively to maximize the detail maintained in the NeRF before distillation.

For the dynamic k -Planes initial training, we rely on all regularizers and loss weights as described in the k -Planes paper [7]. Equation 3.6 shows the total loss calculation when

training f_{θ} . During distillation, we also employ the same original losses, but replace the MSE loss with the looping loss as described above (3.7).

$$\mathcal{L}_{\text{init}} = \mathcal{L}_{\text{MSE}} + \alpha\mathcal{L}_{\text{TV}} + \beta\mathcal{L}_{\text{T}} \quad (3.6)$$

$$\mathcal{L}_{\text{distill}} = \mathcal{L}_{\text{loop}} + \alpha'\mathcal{L}_{\text{TV}} + \beta'\mathcal{L}_{\text{T}} \quad (3.7)$$

3.6 Comparison to VideoLoop3D

In comparison to VideoLoop3D, which employs multi-planar images (MPIs) as its underlying representation, LoopNeRF utilizes volumetric rendering techniques to generate photo-realistic and loopable dynamic scenes. MPIs and the analogous video method of MPVs methods have many properties that are beneficial to creating 3D video textures. Due to their lightweight architecture, they can be rendered at high resolutions and real-time frame-rates, even on mobile devices. And as the parameters of MPVs are RGBA images and video sprites, they can be easily supported with rendering libraries with minimal custom rendering code.

MPIs, and therefore VideoLoop3D, are limited to the scenes and views that they can accurately reconstruct. Because MPIs discretize depth of a scene with each layer, they may not accurately represent objects in the scene that exist at depths between layers, resulting in artifacts. As a 2.5D representation, MPIs can only render views within a small baseline away from training views without rendering with significant artifacts (Fig. 3.1). VideoLoop3D by default only represent forward facing scenes, though extending the method to use multi-spherical images as done in MatryODShka [2] may allow the method to represent outward-facing 360° scenes as well. This modification, however, would still limit the effective baseline that could be rendered without artifacts.

Architecture Features	VideoLoop3D	LoopNeRF
Explicit Representation	MTV	Hex-Planes
Sparsity Optimization	Tile Culling	Proposal Sampling
Dynamic Decomposition	Static / Dynamic Texture Atlas	Static/Dynamic Planes
Multi-scale Training	Video Pyramid Training	Multi-resolution Grids

Table 3.1: **Architecture Comparisons.** LoopNeRF combines many of the features of VideoLoop3D, but with a analogous volumetric versions.

Volumetric methods like NeRF are capable of representing arbitrary scenes, including unbounded 360-degree views as seen in Mip-NeRF 360 [3]. Implicit volumetric representations can also be sampled at continuous values, removing the depth-discretization artifacts

found in MPIs. This expressivity is at the expense of rendering speeds, and while significant research has been accomplished to reduce training and rendering times of NeRFs, real-time rendering of dynamic scenes still requires a GPU.

LoopNeRF addresses this challenge by leveraging the expressivity and flexibility of NeRF-based methods, while also incorporating compressed and lightweight architecture features found in VideoLoop3D. We show the analogous model components in Table 3.1. Instead of a compact explicit representation of MTVs / MPIs, LoopNeRF uses an explicit Hex-Planes to parameterize the scene. VideoLoop3D further optimizes the model representation by culling unused tiles. We replace this functionality with a learned proposal sampler [3, 20], which optimizes the sample locations across array during training. The static-dynamic decomposition found with VideoLoop3D’s separate static and dynamic texture atlases are also found in LoopNeRF between the static and dynamic grids. VideoLoop3D also increases the resolution of the model overtime, which is similar to the multi-resolution plane scheme used in LoopNeRF.

Chapter 4

Experiments and Results

4.1 Implementation Details

For training the initial k -Planes model, we follow the standard training procedure described in [7]. We train for 30K iterations on every frame extracted from the input video on a 24GB Nvidia RTX A5000. We save all time-independent model components every 1K iterations for use in distillation. As described in 3.5, we significantly increase the time resolutions in the dynamic planes of the k -Planes model in order to reconstruct higher frequencies of dynamic movement in the input data.

During distillation, we tune view selection hyperparameters for each scene depending on each capture’s camera paths. Generally, we select segments within the camera path where the normalized translational difference is less than 0.02 and the normalized Euler angle rotational difference is less than 0.2. We remove segments that are smaller than a minimum segment length of 50-60 frames. This results in 2-5 candidate asynchronous static renders of resolution $1080\text{px} \times 1920\text{px}$ that will be used to supervise the distillation process of LoopNeRF.

We train the distilled LoopNeRF model for 8K iterations on a 24GB Nvidia RTX A5000. In every iteration, we sample a patch of 50-100 frames from the static renders for supervision. In the first 1K iterations, the size of the patch is $192\text{px} \times 192\text{px}$. This is downsampled to the maximum patch size of 48px. The patch size is halved every 1K until 48px is reached.

4.2 Monocular Results

Numerical evaluation is difficult in this problem, as there is no groundtruth 3D video texture in which to compare to. Further, LoopNeRF’s results are temporal, and are not easily visualized in a static paper format. Instead of significant numerical results, we will show the effectiveness of the method by showing numerous visual results.

We visualize the results of several casual monocular captures of varying subjects. Each capture is around 45-60 seconds long, and aim to capture the dynamic motion well. We

show the rendered result of the same pose from both the initial k -Planes and the LoopNeRF. These poses were taken directly from the view selection process, therefore we can assume that the renders from the initial k -Planes are a reasonably good reconstruction of the scene’s underlying motion. We visualize the looping effect by sampling a horizontal line in all rendered videos and stacking the results. In Fig. 4.1, Fig. 4.2, and Fig. 4.3 we can see that the motion captured in the dynamic render is reasonably reconstructed and smoothly looped in the LoopNeRF rendering. We also note that the static parts of the scene— locations where the stacked times show no vertical change in the k -Planes rendering— are also maintained in the LoopNeRF results. We show that there is reasonable loopable geometry change captured in the LoopNeRF results by rendering the depth results as well.

4.3 VideoLoop3D Comparison

We evaluate on multiple scenes from the VideoLoop3D dataset. Each scene contains 7-9 calibrated asynchronous static videos. That is, each video is assumed to be from a static camera pose for its entirety. As VideoLoop3D works only on forward-facing scenes, each pose is roughly in the same orientation and translations varying within a small baseline.

Because of the minimal baselines and sparse views, many scenes in the VideoLoop3D dataset are unsuitable for NeRF reconstructions. LoopNeRF is aimed to work on more casual captures, where objects of interest in the scene are viewed from many different viewpoints, rather than a sparse set of static asynchronous videos. Therefore, we will only comparison experiments on those scenes that did not result in a degenerate solution in the original dynamic NeRF training.

We use all default parameters as described for each evaluated scene when training and rendering with VideoLoop3D. For LoopNeRF evaluation, we disable appearance embeddings. Instead of selecting and rendering static views from a monocular capture, we use the static asynchronous views as provided. We note that this is an easier problem than the casual monocular video case, as groundtruth views are of higher quality than those rendered from a dynamic NeRF. For each scene, we train a dynamic k -Planes on all views, holding out one for evaluation. Unlike VideoLoop3D, which first trains a static MPI representation before learning the final MTV, LoopNeRF first trains a full dynamic k -Planes model to minimize floating artifacts found in static NeRFs of dynamic scenes. We stitch each view’s video back to back into a single monocular capture, giving each frame F_i a time $t_i = i/N$, where N is the total number of recorded frames. This assures that the dynamic NeRF is not forced to assume the views are synchronously captured, resulting in less floater artifacts.

4.4 Ablations

We show the effectiveness of multi-resolution training through an ablation. In general, we find that smaller patch sizes result in higher local detail quality, but a lack of spatial consistency.

Model Metrics			
Model	# Total Parameters	# Trainable Parameters	Render Speed
VideoLoop3D	33M-184M	33M-184M	100-140fps
<i>k</i> -Planes Dynamic	315M	313M	1.3fps
LoopNeRF	106M-140M	30M-40M	1.3fps

Table 4.1: **Model Parameters Comparison:** LoopNeRF uses a comparable number of parameters to VideoLoop3D, but learns a volumetric representation that can learn arbitrary scene layouts.

	<i>LPIPS</i> ↓	<i>PSNR</i> ↑	<i>SSIM</i> ↑
VideoLoop3D	0.169877	26.3038	0.7625
LoopNeRF (Ours)	<u>0.175811</u>	29.1661	0.833048
LoopNeRF-Multires	0.182237	<u>28.5571</u>	<u>0.814608</u>

Table 4.2: Comparison and Ablation: We show better reconstruction qualities in most metrics compared to VideoLoop3D. When adding multi-resolution training, these metrics only marginally decline.

This is most visible in large object movements, such as a scene of an awning blowing in the wind. With smaller patch sizes, the awning, parts of the fabric of the awning will be rendered as disconnected, as the patch size does not guarantee consistency across the whole of the fabric. This can be seen in Fig. 4.5, where without coarse-to-fine training, the looping loss results in the store awning, which can be seen both in the rendered RGB and depth. Because coarse-to-fine freezing of dynamic planes limits the expressivity of the model during training, it does result in worse quantitative metrics as shown in Table 4.2. However, the more realistic results generated from multi-resolution training may not show up in metrics such as PSNR or SSIM.

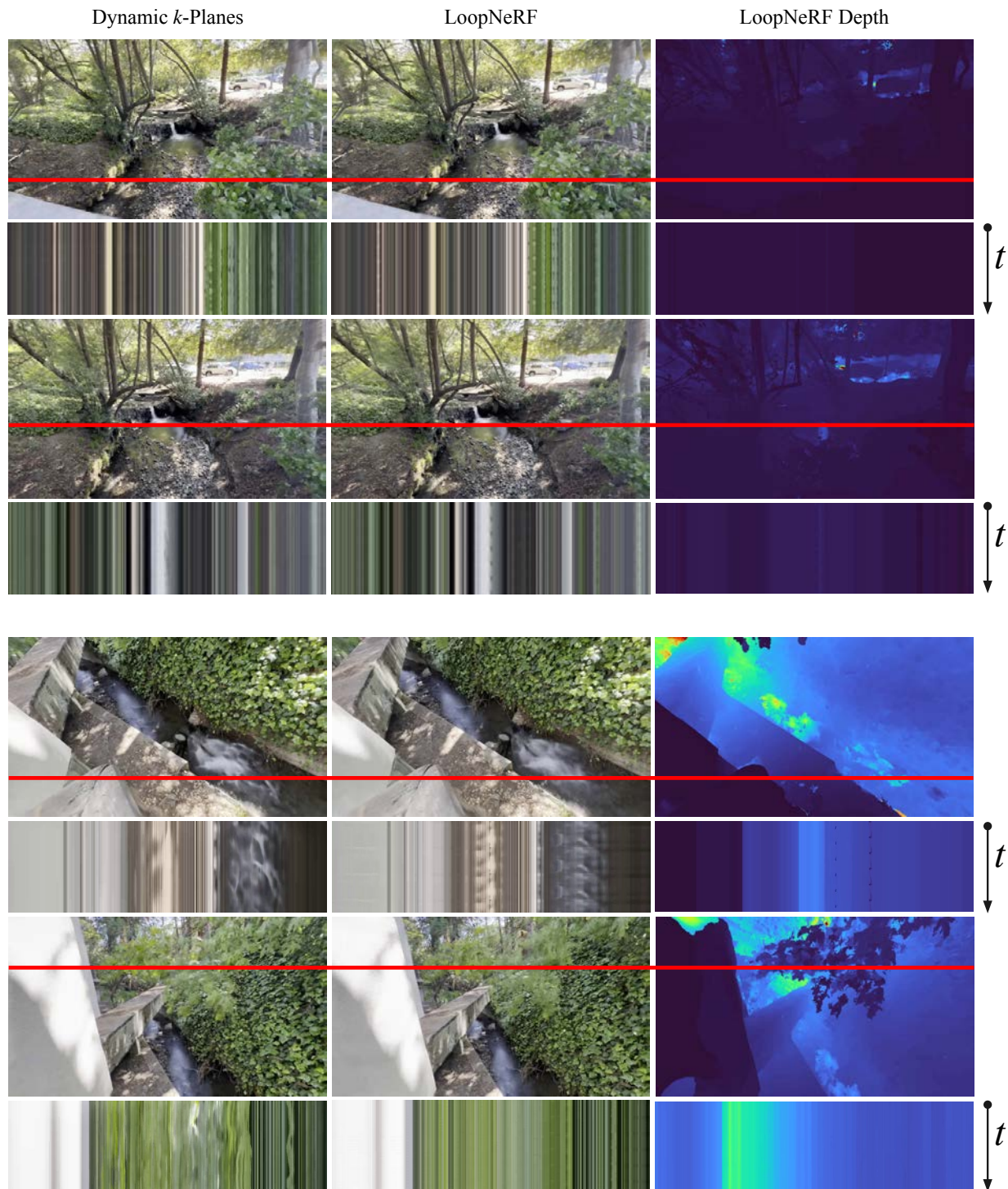


Figure 4.1: Monocular Capture Results: Left: For each scene, we render from the initial dynamic k -Planes model (left), the LoopNeRF model (middle), and a depth map from the LoopNeRF model (right). We sample a horizontal line at every timestep and visualize it over time. Top: *Berkeley Pond*, Bottom: *Berkeley Stream*

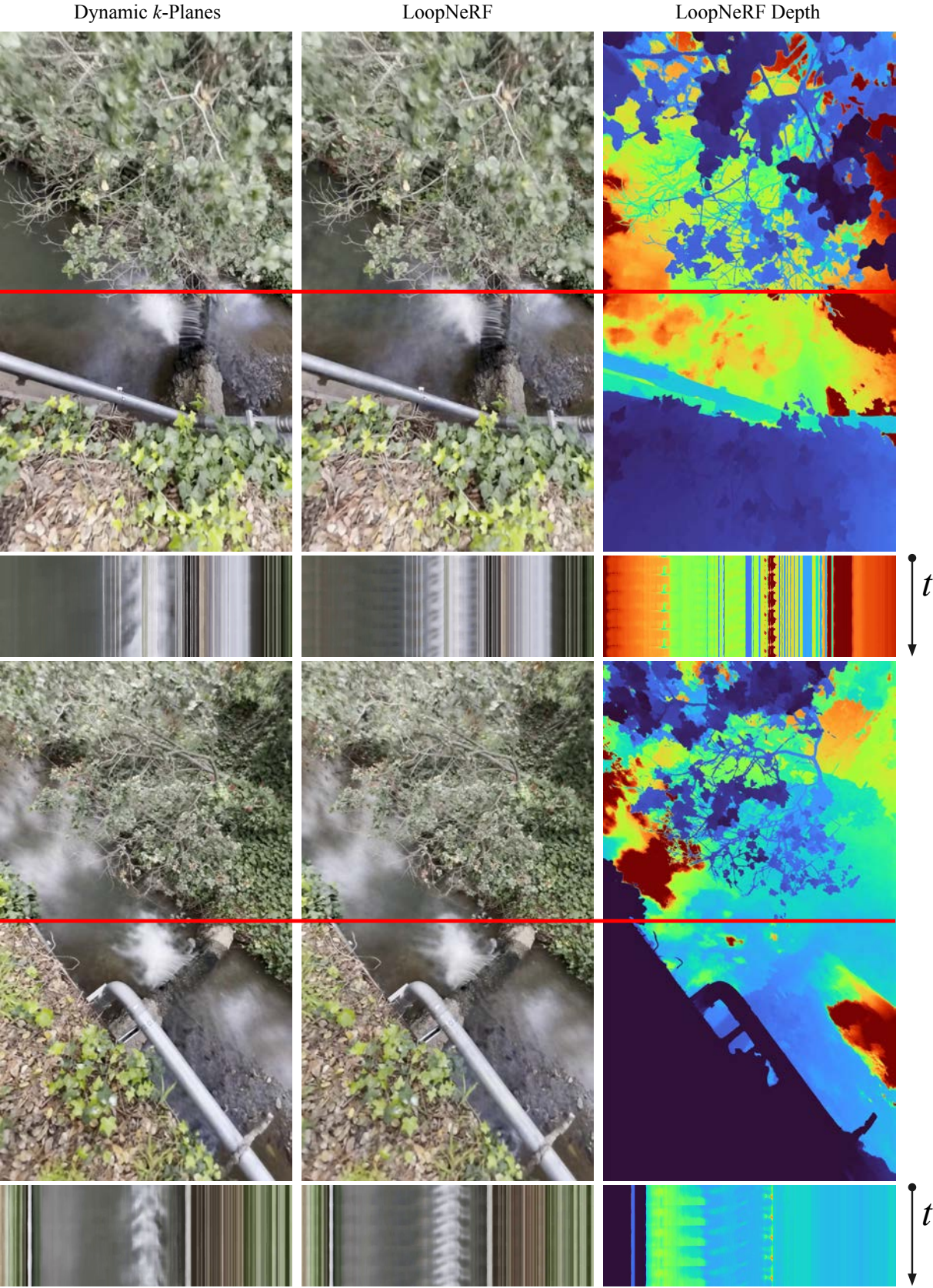


Figure 4.2: Monocular Capture Results: *Berkeley Ledge*

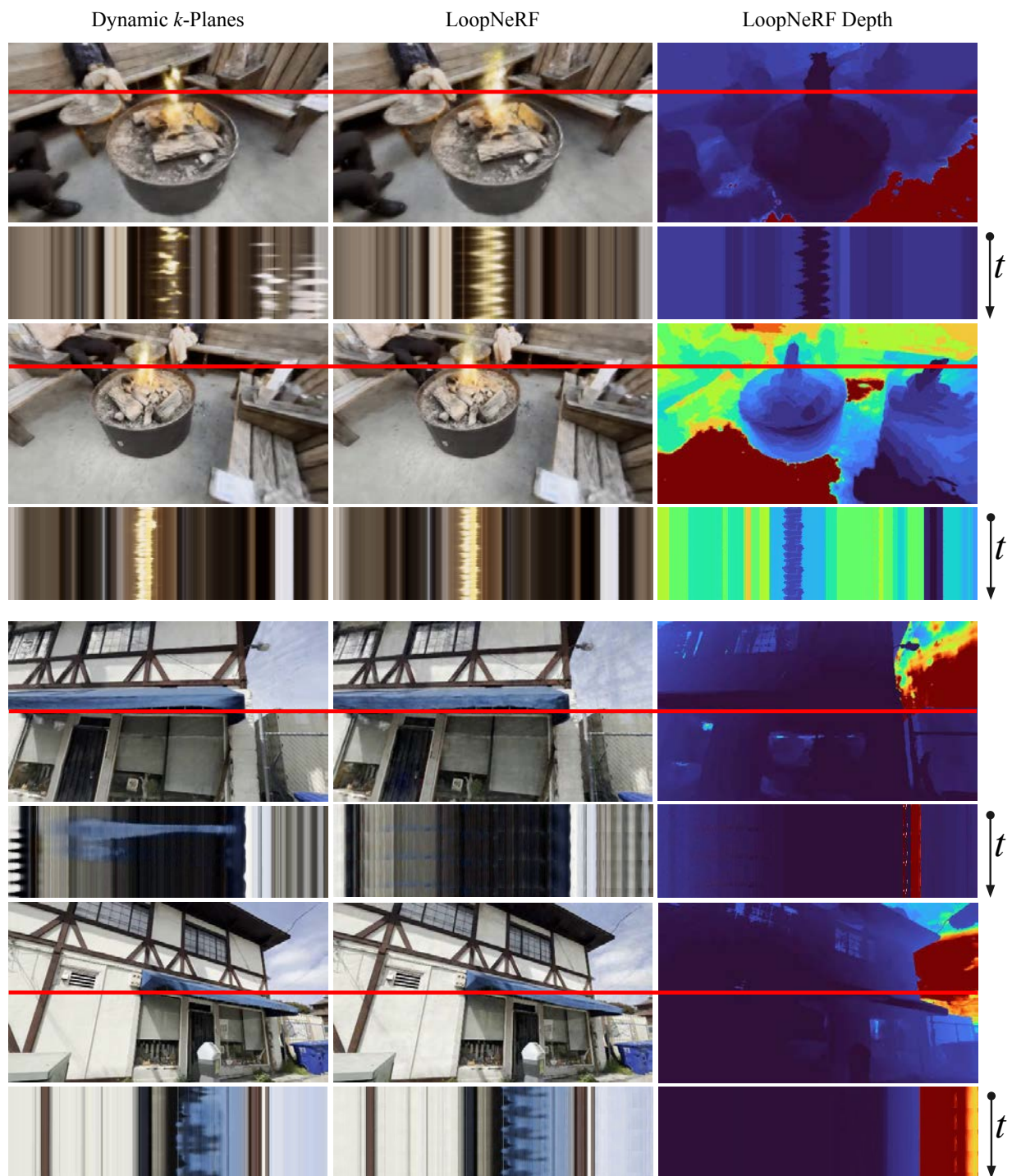


Figure 4.3: Monocular Capture Results: Top: *Oakland Fire*, Bottom: *Berkeley Storefront*

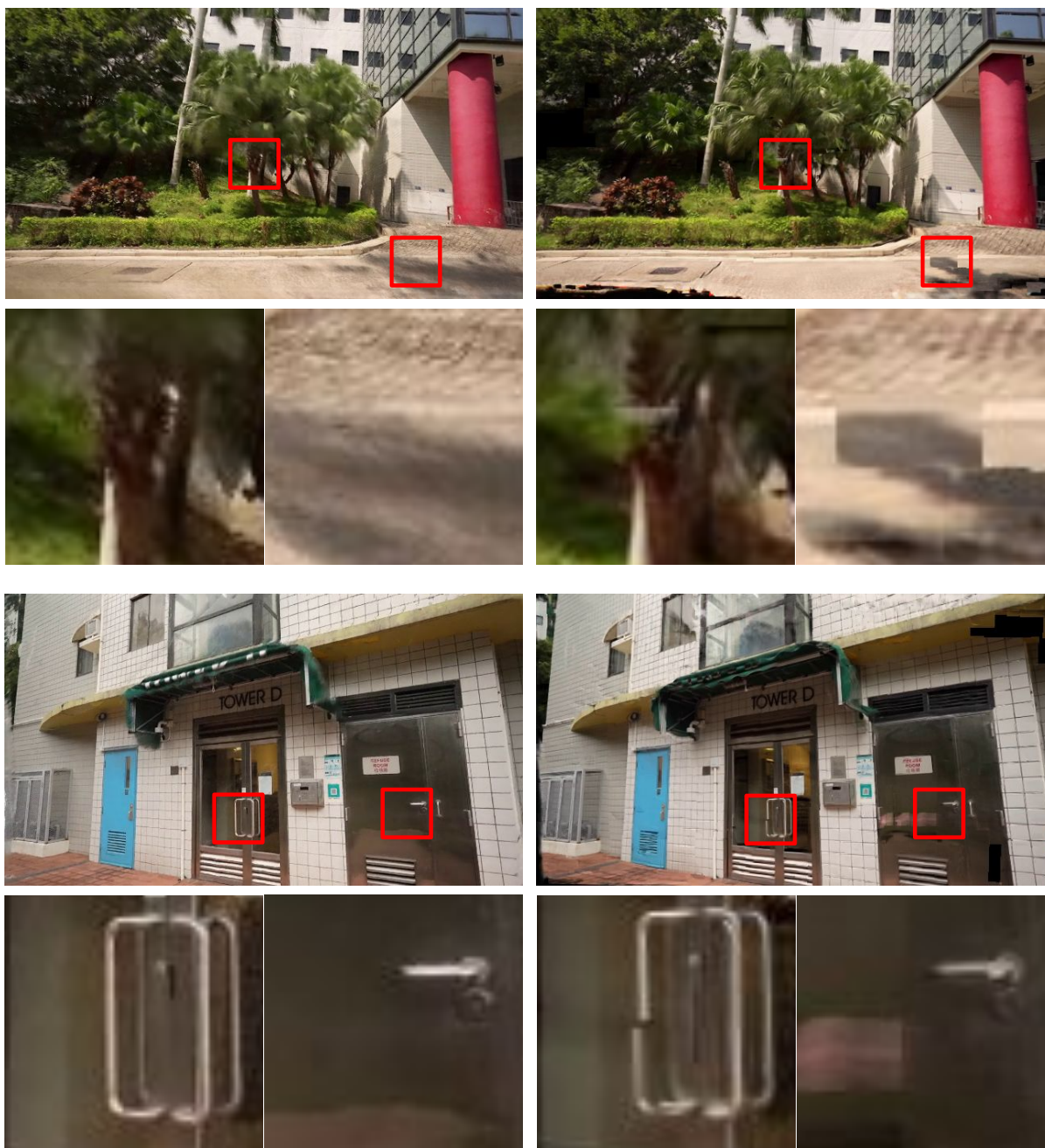


Figure 4.4: VideoLoop3D Comparison: When rendering scenes far away from training views, VideoLoop3D results in block-like artifacts (highlighted in red). Top: *Pillar720p*, Bottom: *Towerd720p*

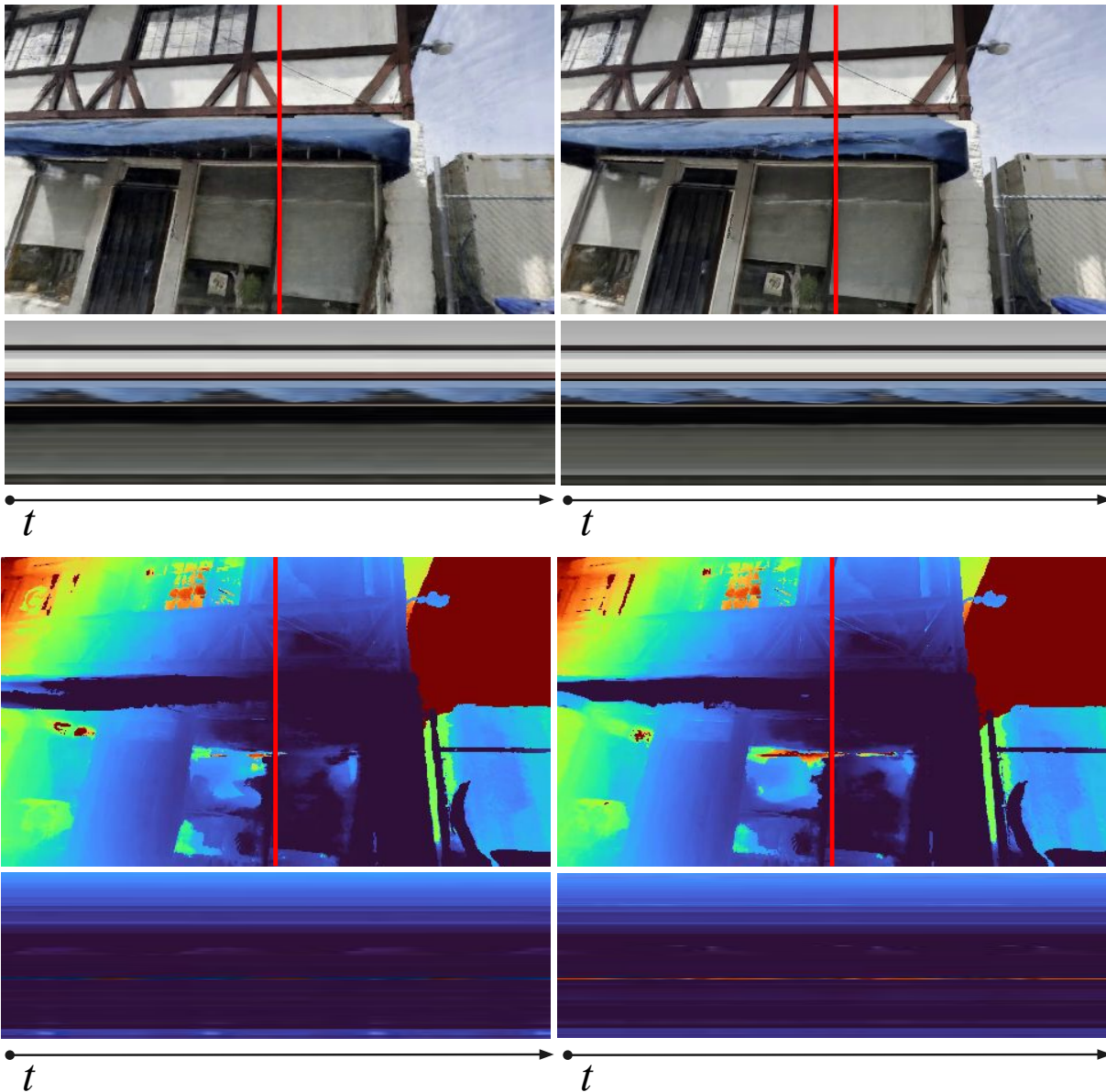


Figure 4.5: Multi-resolution Training Ablation: Left: Coarse-to-fine patch sampling enabled. Right: Baseline. Note that without course-to-fine training, a hole inside the awning is rendered, showing the lack of spatial consistency.

Chapter 5

Discussion

5.1 Why Does This Work?

Given the simple distillation method, it is not obvious why LoopNeRF does not immediately degrade during distillation. In this section, we will briefly mention the reasons for why this method is effective.

Strong Dynamic Sparsity Loss Prior

Because k -Planes is able to effectively decompose a scene into its static and dynamic parts, it is able to apply time-dependent losses unlike other dynamic NeRF methods. Part of why the LoopNeRF method is effective on dynamic scenes trained on monocular capture is due to a strong prior on the sparseness of dynamicism within the scene.

The scenes we evaluate LoopNeRF on are mostly static. That is, the scene is almost completely static, with only high frequency motion located sparsely within the scene. This is perfectly suited for a strong temporal sparsity regularization term during training, implemented as a L1 loss between the features of the dynamic planes and the 1s plane. This promotes the initial k -Planes model to learn as much of the scene within its static planes, and only sparsely populate its dynamic planes where it is absolutely necessary. Because of this, the re-initialized k -Planes, containing only static information of the scene, is already a reasonable reconstruction of the scene before any distillation training has occurred.

Frozen Proposal Sampling

A reasonable worry in distillation is that the scene will overfit to the supervised views: placing significant density directly in front of the sparse set of distillation view cameras, rather than actually modifying the underlying scene geometry in a view-consistent way. One of the reasons this does not occur in the LoopNeRF method is due to the freezing of the proposal sampler model components during distillation. As seen in Fig. 5.1, the proposal sampler learned during the initial k -Planes training learns a reasonably accurate surface of

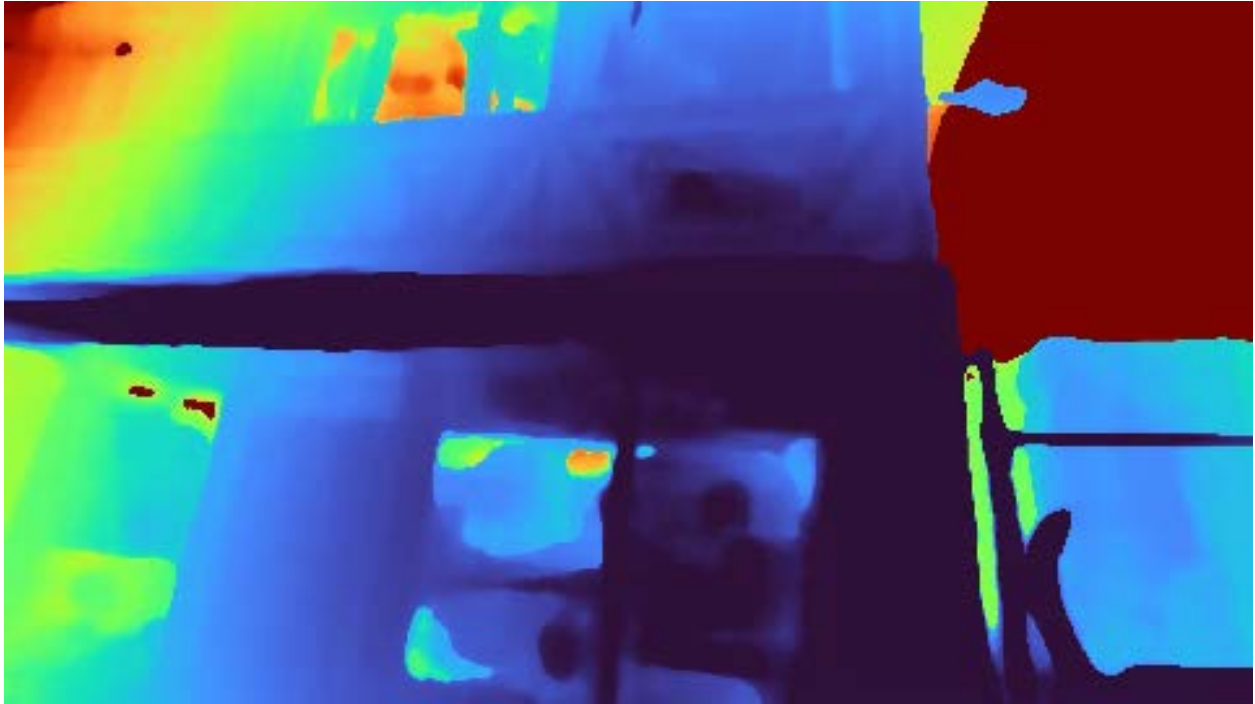


Figure 5.1: Fixed Proposal Networks: The proposal networks learned in the initial k -Planes network learn to sample close to surface geometry. In distillation, this minimizes the generation of floaters, as areas far away from the learned scene surface are rarely sampled during training.

the scene. For a successful distillation, LoopNeRF wants to modify the underlying densities of the scene where there is dynamic motion, but without overfitting to a single view. By fixing the proposal sampler, regions of the scene where there previously was no significant density are never sampled, and therefore minimizes the generation of such floaters from being generated.

Hybrid Representation

Unlike implicit methods, k -Planes has an underlying explicit grid that stores spatiotemporal information of the scene. The underlying explicit structure of LoopNeRF allow for locations in the scene where there is no distillation supervision to be relatively unaffected. This is because those spatiotemporal locations in the scene will never be sampled during distillation, and therefore will never see significant gradients during backpropagation. This is unlike vanilla NeRF or other implicit methods, where modification of some weights with respect to a training ray may drastically change other spatiotemporal locations in the scene not seen from that ray.



Figure 5.2: Failure Case: If the initial K -Planes has learned incorrect static geometry, it will not be corrected in the distilled LoopNeRF model, resulting in the same artifacting. See the discontinuities on the utility pole and the ghosting artifacts near the traffic cones.

5.2 Limitations and Future Work

With all distillation methods, the quality of the distillation is limited by the quality of the original model. The success of LoopNeRF distillation depends on both the quality of the original K -Planes and the input monocular video data. Our method struggles to work on all casual captures. Our distillation method assumes that there are some places in the capture camera patch in which there is minimal camera movement, resulting in areas of high supervision in the trained dynamic K -Planes. This is not true of all captures, and when hyperparameters are relaxed to extract static view captures, they may be noisy and not true to groundtruth. This noise will then be baked into the final LoopNeRF model.

Failures of the initial K -Planes model will also result in failure of this method. If the static parts of the scene are not correctly represented in the static planes of the initial K -Planes, they will stay incorrect in the final LoopNeRF as well. Other model components, such as the proposal sampler, can also fail in the initial model and result in the same failures in the distilled LoopNeRF, as seen in 5.2.

LoopNeRF is also expensive to evaluate, requiring a 24GB GPU at least for training. This is because video patch rendering and backpropagation is quite expensive, which limits

the patch size that can be rendered. We also note that LoopNeRF is significantly slower in inference than other methods, such as VideoLoop3D (4.1).

Future work in this method could aim to make LoopNeRF more robust and efficient. Given recent advances in generative models and video generation, we will explore how using generated data may further relax the requirements of training data. This may lead to a replacement of the view selection step of this method, which seems to be a large variable in model quality. We aim to also explore using extracted frequencies found in the input data to further compress the distilled K -Planes model.

5.3 Conclusion

In this work, we present LoopNeRF, a method for generating 3D loopable view synthesis videos from a trained dynamic K -Planes model and a casual monocular capture. In doing so, we argue for why naive methods would fail on such a problem, and give intuition for why a distillation method is reasonable. We evaluate LoopNeRF on multiple casual monocular captures. Finally, we compare against VideoLoop3D, and show that LoopNeRF creates less artifacting than MPI / MPV methods.

Bibliography

- [1] Aseem Agarwala et al. “Panoramic Video Textures”. In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 821–827. ISSN: 0730-0301. DOI: 10.1145/1073204.1073268. URL: <https://doi.org/10.1145/1073204.1073268>.
- [2] Benjamin Attal et al. “MatryODShka: Real-time 6DoF Video View Synthesis using Multi-Sphere Images”. In: *European Conference on Computer Vision (ECCV)*. Aug. 2020. URL: <https://visual.cs.brown.edu/matryodshka>.
- [3] Jonathan T. Barron et al. “Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields”. In: *CVPR* (2022).
- [4] Michael Broxton et al. “Immersive Light Field Video with a Layered Mesh Representation”. In: 39.4 (2020), 86:1–86:15.
- [5] Ang Cao and Justin Johnson. “HexPlane: A Fast Representation for Dynamic Scenes”. In: *arXiv:2301.09632* (2023).
- [6] Anpei Chen et al. “TensorRF: Tensorial Radiance Fields”. In: *European Conference on Computer Vision (ECCV)*. 2022.
- [7] Sara Fridovich-Keil et al. *K-Planes: Explicit Radiance Fields in Space, Time, and Appearance*. 2023. arXiv: 2301.10241 [cs.CV].
- [8] Hang Gao et al. *Monocular Dynamic View Synthesis: A Reality Check*. 2022. DOI: 10.48550/ARXIV.2210.13445. URL: <https://arxiv.org/abs/2210.13445>.
- [9] Aleksander Holynski et al. “Animating Pictures With Eulerian Motion Fields”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 5810–5819.
- [10] Julian Knodt. *Continuous Dynamic-NeRF: Spline-NeRF*. 2022. arXiv: 2203.13800 [cs.CV].
- [11] Tianye Li et al. *Neural 3D Video Synthesis from Multi-view Video*. 2021. DOI: 10.48550/ARXIV.2103.02597. URL: <https://arxiv.org/abs/2103.02597>.
- [12] Xingyi Li et al. *3D Cinemagraphy from a Single Image*. 2023. arXiv: 2303.05724 [cs.CV].
- [13] Kai-En Lin et al. *Deep Multi Depth Panoramas for View Synthesis*. 2020. arXiv: 2008.01815 [cs.CV].

- [14] Li Ma et al. “3D Video Loops from Asynchronous Input”. In: *arXiv preprint arXiv:2303.05312* (2023).
- [15] Keunhong Park et al. “HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields”. In: *ACM Trans. Graph.* 40.6 (Dec. 2021).
- [16] Keunhong Park et al. “Nerfies: Deformable Neural Radiance Fields”. In: *ICCV* (2021).
- [17] Albert Pumarola et al. “D-NeRF: Neural Radiance Fields for Dynamic Scenes”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.
- [18] Arno Schödl et al. “Video Textures”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’00. USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 489–498. ISBN: 1581132085. DOI: 10.1145/344779.345012. URL: <https://doi.org/10.1145/344779.345012>.
- [19] Liangchen Song et al. *NeRFPlayer: A Streamable Dynamic Scene Representation with Decomposed Neural Radiance Fields*. 2023. arXiv: 2210.15947 [cs.CV].
- [20] Matthew Tancik et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development”. In: *arXiv preprint arXiv:2302.04264* (2023).
- [21] Liao Wang et al. *Fourier PlenOctrees for Dynamic Radiance Field Rendering in Real-time*. 2022. arXiv: 2202.08614 [cs.CV].
- [22] Chung-Yi Weng et al. “HumanNeRF: Free-Viewpoint Rendering of Moving People From Monocular Video”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 16210–16220.