Frustum Volume Caching for Accelerated NeRF Rendering

MICHAEL STEINER, Graz University of Technology, Austria
THOMAS KÖHLER and LUKAS RADL, Graz University of Technology, Austria
MARKUS STEINBERGER, Graz University of Technology, Austria and Huawei Technologies, Austria

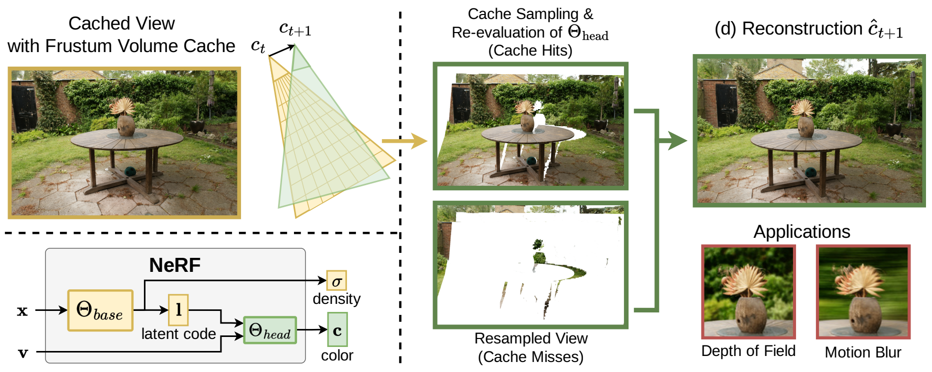

Cached View with Frustum Volume Cache
c_t
c_{t+1}
Cache Sampling & Re-evaluation of Θ_{head} (Cache Hits)
(d) Reconstruction $\hat{c}_{t+1}$
Resampled View (Cache Misses)
Applications
Depth of Field
Motion Blur
NeRF
$\mathbf{x}$
$\mathbf{v}$
Θ_{base}
$\mathbf{l}$
latent code
Θ_{head}
σ
density
$\mathbf{c}$
color

Fig. 1. We use a frustum volume cache to store the expensively computed view-independent output of Θ_{base}, allowing for efficient lookups via backward reprojection and trilinear interpolation, and re-evaluation of view-dependent effects via the smaller Θ_{head}. We utilize an occupancy grid to detect cache-misses and seamlessly combine cached samples with newly evaluated ones along view rays. Our approach accelerates real-time rendering, as well as offline rendering with expensive effects, *e.g.* motion blur.

Neural Radiance Fields (NeRFs) have revolutionized the field of inverse rendering due to their ability to synthesize high-quality novel views and applicability in practical contexts. NeRFs leverage volume rendering, evaluating view-dependent color at each sample with an expensive network, where a high computational burden is placed on extracting an informative, view-independent latent code. We propose a temporal coherence method to accelerate NeRF rendering by caching the latent codes of all samples in an initial viewpoint and reusing them in consecutive frames. By utilizing a sparse frustum volume grid for caching and performing lookups via backward reprojection, we enable temporal reuse of NeRF samples while maintaining the ability to re-evaluate view-dependent effects efficiently. To facilitate high-fidelity rendering from our cache with interactive framerates, we propose a novel cone encoding and explore a training scheme to induce local linearity into the latent information. Extensive experimental evaluation demonstrates that these choices enable high-quality real-time rendering from our cache, even when reducing latent code size significantly. Our proposed method scales exceptionally well for large networks, and our highly optimized real-time implementation allows for cache initialization at runtime. For offline rendering of high-quality video sequences with expensive supersampled effects like motion blur or depth of field, our approach provides speed-ups of up to 2×.

CCS Concepts: • **Computing methodologies → Rendering**; *Machine learning*.

Additional Key Words and Phrases: Neural Radiance Fields, Volume Rendering, Temporal Coherence

Authors' addresses: Michael Steiner, michael.steiner@tugraz.at, Graz University of Technology, Austria; Thomas Köhler, t.koehler@tugraz.at; Lukas Radl, lukas.radl@tugraz.at, Graz University of Technology, Austria; Markus Steinberger, steinberger@tugraz.at, Graz University of Technology, Austria and Huawei Technologies, Austria.

ACM Reference Format:
Michael Steiner, Thomas Köhler, Lukas Radl, and Markus Steinberger. 2024. Frustum Volume Caching for Accelerated NeRF Rendering. *Proc. ACM Comput. Graph. Interact. Tech.* 7, 3 (July 2024), 22 pages. https://doi.org/10.1145/3675370

1 INTRODUCTION

NeRFs [Mildenhall et al. 2020] have recently received considerable attention due to their ability to faithfully synthesize novel views from a learned 3D scene representation, leveraging differentiable volume rendering. Although explicit or hybrid representations [Fridovich-Keil et al. 2022; Kerbl et al. 2023] have emerged as competitors citing faster rendering times, NeRF-based methods, particularly Zip-NeRF [Barron et al. 2023], still achieve the highest view synthesis quality. The volumetric nature of NeRFs requires spatial sampling along view rays with high sample counts. Although common volume rendering techniques such as empty space skipping or early ray termination can alleviate this limitation, many of the best performing methods are not capable of real-time rendering due to large per-sample computational cost. Even in the context of offline-rendering, high-quality NeRF-based methods can be prohibitively slow. The most common way to accelerate NeRF rendering is to bake the underlying representation of a trained model into a more render-friendly representation, *e.g.* a mesh [Chen et al. 2023; Reiser et al. 2024; Yariv et al. 2023] or a voxel grid [Garbin et al. 2021; Hedman et al. 2021]. However, baking these models can be time-consuming, and often involves a trade-off between quality retention and memory requirements.

Intending to accelerate NeRF-based rendering, we first consider the typical network architecture employed across virtually all NeRF methods. Generally, the evaluation of each sample is split into two parts: an expensive *base* network predicts density and view-independent latent codes, while a lightweight *head* network predicts outgoing radiance conditioned on a view direction and the view-independent intermediate output. However, when rendering consecutive frames of a video sequence or during real-time rendering, the expensive *base* network is evaluated repeatedly at very similar 3D positions, making temporal reuse of view-independent information desirable.

We therefore propose a volumetric caching approach, storing view-independent latent information per sample in a view-aligned sparse frustum voxel (*froxel*) grid, which can be efficiently sampled via backward reprojection (see Fig. 1). For each sample position, we interpolate the latent code and density from our cache and re-evaluate view-dependent effects via the *head* network. While our approach works out-of-the-box with popular NeRF models like Instant-NGP [Müller et al. 2022], we find that both quality and performance are subpar. To address both limitations, we first propose a novel view-directional cone encoding, which allows for significantly smaller latent codes and an even more lightweight *head* network, resulting in increased performance. To remedy interpolation artifacts, we propose to induce spatially local linearity to the latent codes during training, resulting in higher image quality when rendering from our cache. When combined, both techniques enable high-fidelity rendering from our cache with interactive framerates.

We provide the source code for our training framework, and a real-time capable renderer that showcases our method's capabilities. Our highly optimized CUDA implementation features an adaptive, asynchronous cache initialization for the real-time viewer, supersampled motion blur and depth of field for offline rendering, as well as a custom fused kernel to speed up the *head* network. Our method effectively accelerates real-time rendering, being up to 4.8× faster than Instant-NGP while achieving equal quality - compared to non-cached variants, caching increases rendering speed by up to 2×. Consequently, our renderer performs particularly well for high-quality video sequences with expensive effects, where we do not observe any loss of rendering quality.

2 BACKGROUND & RELATED WORK

In the following, we give an overview of the most important NeRF methods, different acceleration techniques for neural rendering, as well as a general overview of temporal coherence methods.

2.1 Neural Radiance Fields

NeRFs [Mildenhall et al. 2020] tackle the inverse rendering problem, learning a 3D scene from a set of posed 2D images. Prior techniques mostly tried to solve this problem with image-based techniques *e.g.* with multi-layered images [Mildenhall et al. 2019; Srinivasan et al. 2019]. Instead, NeRF uses an MLP to learn and encode an implicit volumetric representation of a 3D scene as density and view-dependent outgoing radiance. By applying positional encoding [Tancik et al. 2020], their approach produced state-of-the-art results in novel view synthesis, however, expensive raymarching using a large MLP limited its applicability. Follow-up work aimed to speed up training and increase quality by explicitly subdividing the 3D space, employing either multiple smaller MLPs [Reiser et al. 2021] or learning per-voxel Spherical Harmonics (SH) coefficients [Fridovich-Keil et al. 2022]. Other works used more efficient explicit representations like octrees [Yu et al. 2021] or factorized the 5D space into lower-rank tensors [Chen et al. 2022; Tang et al. 2022]. Müller et al. [2022] introduced an efficient multi-resolution hash encoding, allowing for much smaller MLPs and, therefore, faster optimization and rendering.

To alleviate the issue of high sample counts, recent work explored depth oracles [Neff et al. 2021], occupancy grids [Liu et al. 2020; Müller et al. 2022], or automatic per-ray integration [Lindell et al. 2021]. Other methods investigated more efficient sample placement with proposal networks [Barron et al. 2022, 2023] or sparse sampling networks [Kurz et al. 2022]. To circumvent aliasing, Mip-NeRF [Barron et al. 2021] proposed an integrated positional encoding, allowing the model to reason about scale. Mip-NeRF 360 [Barron et al. 2022] extended this idea to unbounded scenes, and Zip-NeRF [Barron et al. 2023] combined multisampled anti-aliasing with iNGP's hash grid encoding [Müller et al. 2022]. In contrast to volumetric neural rendering, 3D Gaussian Splatting [Kerbl et al. 2023] recently demonstrated fast training and rendering with high visual quality, relying on differentiable rasterization of a mixture of anisotropic 3D Gaussians.

2.2 Baking NeRFs

One common way to accelerate NeRF rendering is to "bake" the neural representation into a render-friendly format, frequently resulting in a trade-off between quality retention and memory requirements. FastNeRF [Garbin et al. 2021] factorizes the NeRF network into separate position-dependent and view-dependent MLPs, whose intermediate outputs are cached in world-space and can be efficiently queried. Hedman et al. [2021] leverage a sparse voxel grid storing opacity, diffuse color and a neural feature vector for efficient rendering on commodity hardware. To further limit storage requirement, MERF [Reiser et al. 2023] reduces the voxel grid resolution and utilizes 2D feature planes. Duckworth et al. [2024] subsequently demonstrated real-time rendering of apartment-scale scenes with a set of MERFs. Other approaches bake NeRFs into a mesh [Chen et al. 2023; Reiser et al. 2024; Yariv et al. 2023] leveraging the efficient polygon rasterization pipeline: these approaches struggle with fine geometric structures and semi-transparent objects during meshing. Finally, other data structures for baking, such as duplex meshes with neural features [Wan et al. 2023] or view-dependent volumes [Yu et al. 2021], have also been explored. Our approach circumvents common drawbacks of related work by constructing a view-aligned representation in real-time from the original NeRF representation.

2.3 Temporal Coherence Methods

Reusing rendering information in consecutive frames has always been desirable for computer graphics applications. Early work used forward reprojection with mesh-based 2.5D reconstructions [Mark et al. 1997], or layered depth images [Shade et al. 1998]. On the contrary, backward reprojection for shading reuse is done by re-rendering geometry [Nehab et al. 2007] or through flow fields and fixed-point iteration [Bowles et al. 2012]. These surface-based methods mostly do not translate well to volume rendering. Greger et al. [1998] propose to use 3D probes to store irradiance volumetrically. Other works explored image-based techniques, rendering layers of slabs [Mueller et al. 1999], or perform point-based rendering [Zellmann et al. 2012]. More recently, neural networks were used to cache 3D radiance [Müller et al. 2021] for path-tracing, showing promising results for both surfaces and participating media. Wronski [2014] and Hillaire [2015] both use a frustum voxel (froxel) grid to temporally integrate scattered light for volumetric effects, *e.g.* fog. Lochmann et al. [2016] propose irregularly sized frustum segments to store a piecewise-analytic emission-absorption representation. Our temporal coherence method is entirely volumetric and uses a sparse regular froxel grid and backward reprojection to store and retrieve NeRF latent codes, enabling re-evaluation of view-dependent effects.

3 PRELIMINARIES

In this section, we first recite details about volumetric rendering with NeRFs, discuss extensions to handle unbounded scenes, and analyze the underlying architecture of common NeRF methods.

3.1 NeRF Volume Rendering

The goal of NeRF is to learn a function $\Theta : (\mathbf{x}, \mathbf{d}) \mapsto (\mathbf{c}, \sigma)$ that outputs outgoing RGB radiance $\mathbf{c} \in [0,1]^3$ and density $\sigma \in [0,1]$ at a 3D point $\mathbf{x} \in \mathbb{R}^3$ in direction $\mathbf{d} \in \mathbb{R}^3$. The color along a ray $\mathbf{r} = \mathbf{o} + t\mathbf{d}$ is then accumulated via raymarching, querying Θ at N_S sample positions $\{t_1, \ldots, t_{N_S}\}$

$$C(\mathbf{r}) = \sum_{i=1}^{N_S} T_i(1 - \exp(-\delta_i\sigma_i))\mathbf{c}_i, \quad \text{where} \quad T_i = \prod_{j=1}^{i-1} \exp(-\sigma_j\delta_j), \tag{1}$$

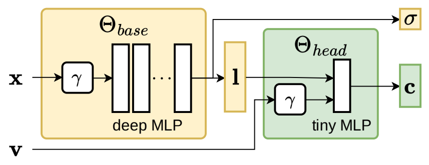
with step size δ_i and transmittance T_i, translating to alpha blending with $\alpha_i = (1 - \exp(-\delta_i\sigma_i))$.

A straightforward approach for sample placement is via an invertible function $g(\cdot)$, mapping an input from stepping-space $\mathcal{S}$ to a corresponding t value. iNGP [Müller et al. 2022] performs exponential stepping for unbounded scenes, with $t_{i+1} = t_i \cdot (1 + a)$ for a small cone angle $a = \frac{1}{256}$. This translates to a step size function $g(i) = t_0 \cdot (1 + a)^i = t_i$, and $g^{-1}(t_i) = \log(\frac{t_i}{t_0} - (1 + a)) = i$. To prevent under-/oversampling, δ_i is clamped between $[\Delta t_{\min}, \Delta t_{\max}]$, dependent on the size of the scene's bounding box (cf. Supplemental A for details). iNGP distills a binary occupancy grid O during optimization to enable empty space skipping, representing the expected density within discretized voxels; for large-scale scenes, a hierarchy of grids is employed.

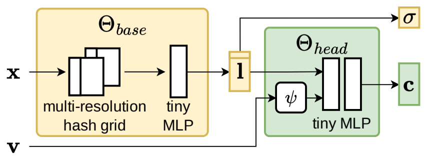
To ease the learning objective for large, unbounded scenes, multiple works utilize a scene contraction [Barron et al. 2022; Neff et al. 2021] to bound the input domain of Θ. Mip-NeRF 360 [Barron et al. 2022] contracts 3D points $\mathbf{x}$ that lie outside the unit norm ball:

$$\text{contract}(\mathbf{x}) = \begin{cases} \mathbf{x} & \text{if } ||\mathbf{x}|| \leq 1 \\ \left(2 - \frac{1}{||\mathbf{x}||}\right)\left(\frac{\mathbf{x}}{||\mathbf{x}||}\right) & \text{if } ||\mathbf{x}|| > 1 \end{cases}. \tag{2}$$

When using the infinity norm $||\cdot||_\infty$, this leads to a final contracted space that spans the entire domain in range $[-2, 2]^3$, thereby fitting the cubic hash grid geometrically [Tancik et al. 2023].



(a) NeRF



(b) iNGP

Fig. 2. Network architecture comparison between the original NeRF [Mildenhall et al. 2020] and iNGP [Müller et al. 2022]. NeRF uses frequency encoding γ for both 3D position $\mathbf{x}$ and view-direction $\mathbf{v}$, a deep *base* MLP, and a wide latent code $\mathbf{l}$ ($N_l = 256$). In contrast, iNGP uses a multi-resolution hash grid, a tiny *base* MLP, very narrow $\mathbf{l}$ ($N_l = 16$) and SH encoding ψ for the view-direction.

Mip-NeRF 360 [Barron et al. 2022] introduces a *distortion loss* $\mathcal{L}_{\text{dist}}$, that encourages weights $w_i = T_i(1 - \exp(-\delta_i\sigma_i))$ along a ray to form a Dirac distribution in stepping-space. Their loss also includes a second term, which forces intervals to become smaller, which can be omitted for a method with fixed step sizes. When using a stepping-space where $g^{-1}(t_i) = i$, the loss $\mathcal{L}_{\text{dist}}$ and its partial derivative can be rewritten as

$$\mathcal{L}_{\text{dist}}(w) = \sum_i^{N_S}\sum_j^{N_S} w_i w_j |i - j|, \qquad \frac{\partial \mathcal{L}_{\text{dist}}}{\partial w_i} = \sum_j^{N_S} w_j |i - j|. \tag{3}$$

3.2 NeRF Network Architecture

As our final approach leverages the underlying NeRF architecture for efficient caching, we examine current state-of-the-art networks. The most prominent NeRF variants split Θ into a view-independent *base* and a view-dependent *head* network

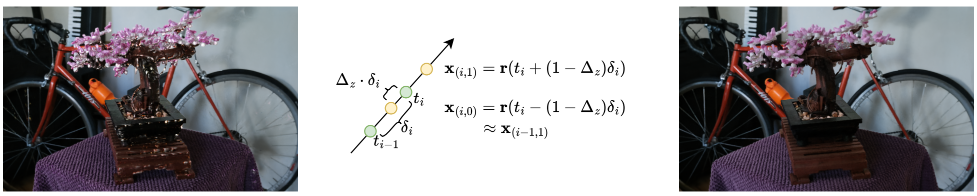
$$\Theta_{base} : \mathbb{R}^3 \longrightarrow \mathbb{R}^{(N_l+1)}, \quad (\mathbf{x}) \mapsto (\sigma, \mathbf{l}), \tag{4}$$

$$\Theta_{head} : \mathbb{R}^{(N_l+3)} \longrightarrow \mathbb{R}^3, \quad (\mathbf{l}, \mathbf{v}) \mapsto (\mathbf{c}), \tag{5}$$

where σ is view-independent by design and *head* is conditioned on the view-independent latent code $\mathbf{l} \in \mathbb{R}^{N_l}$. NeRF's *base* consists of a frequency encoding γ and a deep MLP, with $N_l = 256$. Mip-NeRF [Barron et al. 2021] and Mip-NeRF 360 [Barron et al. 2022] introduce a scale-aware positional encoding, with even wider MLP hidden layers and latent code for Mip-NeRF 360. On the contrary, iNGP [Müller et al. 2022] use their multi-resolution hash encoding and a tiny MLP as Θ_{base}, with a narrow $\mathbf{l}$ ($N_l = 16$; the first value of $\mathbf{l}$ is also log-space σ), and SH encoding ψ of the view-direction. Finally, Zip-NeRF [Barron et al. 2023] follows iNGP's network design, but with wider latent codes ($N_l = 256$) and a much larger view-dependent MLP for increased quality. We contrast the architectures of NeRF and iNGP in Fig. 2. All discussed architectures provide opportunity for accelerated rendering through caching as they use compute intensive Θ_{base} networks, however, large latent codes ultimately lead to excessive cache size requirements. In this work, we propose several techniques to produce smaller but more informative latent codes.

4 INTERPOLATING NERF SAMPLES

We propose a temporal coherence method for caching and temporal reuse of view-independent information to accelerate NeRF rendering. Our key idea is to cache the output of Θ_{base} for each evaluated sample in a view-aligned volumetric data structure, sample this cache with trilinear


$\Delta_z \cdot \delta_i$
t_i
δ_i
t_{i-1}
$\mathbf{x}_{(i,1)} = \mathbf{r}(t_i + (1 - \Delta_z)\delta_i)$
$\mathbf{x}_{(i,0)} = \mathbf{r}(t_i - (1 - \Delta_z)\delta_i)$
$\approx \mathbf{x}_{(i-1,1)}$

(a) w/o induced linearity (b) Training with interpolation (c) w/ induced linearity

Fig. 3. Depiction of our proposed training scheme to induce spatial linearity into the latent information $\mathbf{l}$ and σ. (a) Interpolation fails catastrophically at object boundaries if spatial linearity is not learned, which manifests as bright dots all over the object in the shown example. (b) We induce linearity during training by interpolating every actual sample at $\mathbf{x}_i$ from two artificial samples $\mathbf{x}_{(i,0)}, \mathbf{x}_{(i,1)}$, which are shifted along the view ray by a random per-ray stepping-space offset $\Delta_z \in [0, 1]$. (c) This enables our final model to correctly perform trilinear interpolation from cache.

interpolation from novel viewpoints, and efficiently re-evaluate view-dependent effects via Θ_{head}. This poses several challenges, addressed here.

4.1 Interpolation

An integral part of our approach is interpolating latent information $\{\sigma, \mathbf{l}\}$ of NeRF samples from a volumetric data structure. The cache is initialized from a camera at position $\mathbf{o}_t$, with corresponding view-projection matrix M_t. We store the information for a view frustum in 3D froxel grids $Z = \{Z_\sigma, Z_\mathbf{l}\}$ with indices in froxel-space $\mathcal{Z} = (x, y, z)$ consisting of (x, y) in image pixel space, and z in stepping space $\mathcal{S}$. Any world-space point $\mathbf{x} \in \mathbb{R}^3$ can be transformed into a $\mathbf{z} \in \mathcal{Z}$ via the transformation F:

$$\mathbf{z} = F(\mathbf{x}) = \begin{pmatrix} (M_t^{-1}\mathbf{x})_x \\ (M_t^{-1}\mathbf{x})_y \\ g^{-1}\left(||\mathbf{x} - \mathbf{o}_t||_2\right) \end{pmatrix}. \tag{6}$$

Due to the L_2 norm, the frustum appears curved along the z-dimension when transformed back into view-space. Notably, Z is sparsely populated, with a cell (x, y, z) only being set if the ray at pixel (x, y) placed a sample at $t = g(z)$ during cache initialization. To detect cache-hits and handle missing information, we store occupancy information in an additional binary froxel grid Z_o. We obtain our interpolated values $\{\hat{\sigma}, \hat{\mathbf{l}}\}$ via trilinear interpolation from Z_σ and $Z_\mathbf{l}$, where we replace any value with zero if the corresponding cell in Z_o is unoccupied. This introduces a zero-shift into the interpolation results, effectively assuming that values must be zero in unoccupied space, which is a reasonable assumption for density but not for latent codes. Hence, we re-normalize $\hat{\mathbf{l}}$ by considering adjacent occupancy information, effectively disregarding unoccupied cells:

$$\hat{\sigma} = \mathrm{trilerp}(\mathbf{z}, Z_\sigma), \qquad \hat{\mathbf{l}} = \frac{\mathrm{trilerp}(\mathbf{z}, Z_\mathbf{l})}{\mathrm{trilerp}(\mathbf{z}, Z_o)}. \tag{7}$$

4.2 Learning Spatial Linearity

Each latent code $\mathbf{l}$ is an intermediate output of a neural network, approximating a highly non-linear function. Naïve linear interpolation of $\mathbf{l}$ is not well behaved, even for spatially close 3D samples. We can induce linearity by performing the interpolation from Eqn. (7) with randomly shifted samples during training, however, performing full trilinear interpolation from eight samples is expensive. Notably, for large-scale unbounded scenes and when using exponentially increasing step size,

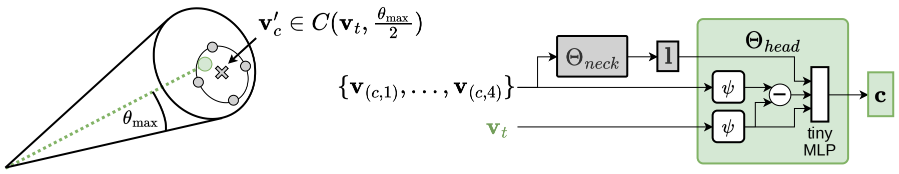

$\mathbf{v}'_c \in C(\mathbf{v}_t, \frac{\theta_{\max}}{2})$
$\theta_{\max}$
$\{\mathbf{v}_{(c,1)}, \dots, \mathbf{v}_{(c,4)}\}$
$\mathbf{v}_t$
Θ_{neck}
$\mathbf{l}$
Θ_{head}
ψ
ψ
tiny MLP
$\mathbf{c}$

Fig. 4. Our proposed cone encoding learns to produce latent codes $\mathbf{l}$, which were generated from a view-direction $\mathbf{v}_c$, but can be re-evaluated from a similar view-direction $\mathbf{v}_t$, lying in the same cone $C(\mathbf{v}_c, \theta_{\max})$. By randomly generating a number of view-directions $\{\mathbf{v}_{(c,1)}, \dots, \mathbf{v}_{(c,N_c)}\} \in C(\mathbf{v}_t, \theta_{\max})$ during training, providing the difference in viewing-angle to Θ_{head}, and averaging the loss of all N_c samples, the model learns to treat $\mathbf{l}$ as a cone encoding around $\mathbf{v}_c$. To guarantee variety during each training iteration, we randomly choose a center point $\mathbf{v}'_c \in C(\mathbf{v}_t, \frac{\theta_{\max}}{2})$, and uniformly space the N_c samples on a circle with radius $\frac{\theta_{\max}}{2}$.

adjacent samples in a view frustum exhibit the largest spatial differences along the view rays, *i.e.* froxel-space z-dimension. Therefore, we propose to perform interpolation from just two samples, shifted along the view ray.

During training, we generate a single random offset in stepping-space $\Delta_z \in [0, 1]$ per ray $\mathbf{r}$ to interpolate every actual sample $\mathbf{x} = \mathbf{r}(t_i)$ from two artificial samples at $\mathbf{x}_{0,1} = \mathbf{r}(t_i \pm (1 - \Delta_z)\delta_i)$, cf. Fig. 3 for a visualization. We then evaluate O at these positions, set all interpolation values to zero if the corresponding position is unoccupied, and perform the same normalization as in Eqn. (7) with

$$\hat{\sigma} = \mathrm{lerp}(\Delta_z, o_0\sigma_{\mathbf{x}_0}, o_1\sigma_{\mathbf{x}_1}), \qquad \hat{\mathbf{l}} = \frac{\mathrm{lerp}(\Delta_z, o_0\mathbf{l}_{\mathbf{x}_0}, o_1\mathbf{l}_{\mathbf{x}_1})}{\mathrm{lerp}(\Delta_z, o_0, o_1)}. \tag{8}$$

where $\left\{\sigma_{\mathbf{x}_j}, \mathbf{l}_{\mathbf{x}_j}\right\} = \Theta_{base}(\mathbf{x}_j)$, and $o_j = O(\mathbf{x}_j) \in \{0, 1\}$. We then continue the evaluation of the network with $\hat{\sigma}$ and $\hat{\mathbf{l}}$.

As can be seen in Fig. 3, rendering novel views from our frustum volume cache can produce disturbing artifacts when interpolating naïvely. Inducing linearity along view rays during optimization eliminates these artifacts. Our experiments suggest that this linearity constraint leads to slightly degraded image quality metrics. However, considering the increased performance with our temporal coherence method, inducing spatial linearity proves worthwhile.

4.3 View-dependent Cone Encoding

The performance of our caching approach is highly dependent on the size of $\mathbf{l}$, as this directly influences cache size and lookup speed. The second crucial factor is the performance of Θ_{head}, as it is always re-evaluated for cached samples. Notably, $\mathbf{l}$ is fully view-independent, meaning the whole 360° viewing information is encoded into $\mathbf{l}$, and Θ_{head} is able to produce outgoing radiance for every possible view-direction. This is wasteful for a temporal coherence approach, where cached samples will only ever be viewed from viewpoints similar to the cached viewpoint.

We introduce a view-dependent cone encoding, which produces view-dependent latent codes that encode the viewing information for a cone $C(\mathbf{v}_c, \theta_{\max})$ of angle $\theta_{\max}$ around view-direction $\mathbf{v}_c$:

$$C(\mathbf{v}_c, \theta_{\max}) = \{\mathbf{v} : \mathbf{v}^T\mathbf{v}_c < \cos(\theta_{\max})\}. \tag{9}$$

This change allows us to shift computational load from Θ_{head} to a new cone encoding network Θ_{neck}, which takes the view-independent output of Θ_{base} and $\mathbf{v}_c$ as input, and outputs the cone encoding $\mathbf{l}$. The outgoing color for the actual view-direction $\mathbf{v}_t$ can be recovered from $\mathbf{l}$ by providing the difference of encoded view-directions to Θ_{head}.

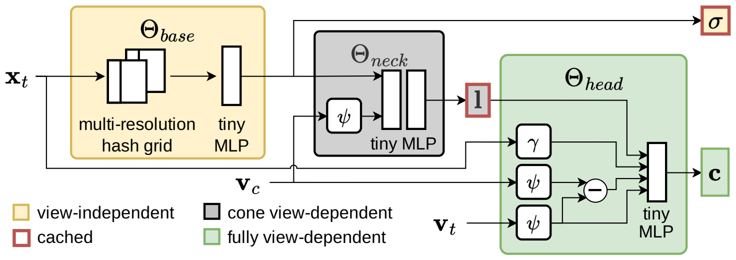


Fig. 5. Our proposed model utilizes a novel *cone encoding*, to shrink the size of latent codes $\mathbf{l}$, and minimize the cost for re-evaluating $\mathbf{l}$ from a different view-direction $\mathbf{v}_t$. The newly introduced intermediate network Θ_{neck}, receives a SH encoded initial view-direction $\psi(\mathbf{v}_c)$, which is allowed to differ up to $\theta_{\max}$ degrees from $\mathbf{v}_t$. The resulting $\mathbf{l}$ is now view-dependent, however, our proposed training scheme enables re-evaluation from $\mathbf{v}_t$ by providing the difference in viewing-angle to Θ_{head}. Forwarding the frequency encoded 3D sample position $\gamma(\mathbf{x_t})$ to Θ_{head} helps to counteract a possible information loss between Θ_{base} and Θ_{head}.

During optimization, Θ_{head} needs to be provided with examples to learn the relationship between $\mathbf{l}$, $\mathbf{v}_c$, and $\mathbf{v}_t$. For this purpose, we generate N_c view-directions $\{\mathbf{v}_{(c,1)}, \ldots, \mathbf{v}_{(c,N_c)}\}$ for every actual view-direction $\mathbf{v}_t$, such that $\mathbf{v}_{(c,j)} \in C(\mathbf{v}_t, \theta_{\max})$. For each sample along $\mathbf{v}_t$, each of those shifted view-directions computes $\mathbf{l}_j$ from Θ_{neck}, evaluates the sample color via $\Theta_{head}(\mathbf{l}_j, \mathbf{v}_{(c,j)}, \mathbf{v}_t)$, and accumulates them along the ray. The total loss is then computed as the average loss of all N_c view-directions.

Randomly sampling $\mathbf{v}_{(c,j)}$ from $C(\mathbf{v}_t, \theta_{\max})$ does not result in a good enough variety of view-directions during training. Therefore, we propose a slightly more sophisticated sampling scheme, depicted in Fig. 4. We first uniformly random sample a smaller cone $C(\mathbf{v}_t, \frac{\theta_{\max}}{2})$ to obtain a center view-direction $\mathbf{v}'_c$. Next, we place the N_c samples uniformly on a circle of radius $\frac{\theta_{\max}}{2}$ around $\mathbf{v}'_c$, with a random radial offset ϕ'. By effectively encircling $\mathbf{v}_t$, we ensure a good distribution of view-directions while guaranteeing that no sample lies outside the encoding cone. Note that $\mathbf{v}_c = \mathbf{v}_t$ when rendering without a cache, and Θ_{head} does not need to be executed during cache initialization.

4.4 A Model for Efficient Rendering From Cache

To maximize quality and performance, we propose a NeRF model that is particularly well suited for our caching approach in Fig. 5. The model is based on iNGP (big) [Müller et al. 2022], but with several adaptions. Most importantly, we leverage our view-dependent cone encoding and spatial linearity training recipe, detailed previously. These modifications enable a significantly smaller $\mathbf{l}$ and Θ_{head}. To counter the capacity loss when shrinking $\mathbf{l}$ further, we provide the encoded sample position $\mathbf{x}$ to Θ_{head}, where we use NeRFs positional encoding [Mildenhall et al. 2020], which can be re-evaluated efficiently and does not need to be cached.

To reduce the overall sample count and improve quality for unbounded scenes, we incorporate different techniques from recent NeRF methods. We use the scene contraction and an adapted version of the distortion loss of Mip-NeRF 360 [Barron et al. 2022], as described in Eqns. (2) and (3). Our scenes are still bounded by the occupancy grid O, however, the input domain of Θ is contracted. Finally, we downweigh gradients close to the camera to reduce floaters, as suggested by Philip and Deschaintre [2023].

Notably, combining our view-dependent cone encoding with interpolation along the view ray means that each sample is evaluated twice for each of the N_c view-directions during optimization. A naïve approach would execute Θ for each of those combinations, leading to large memory

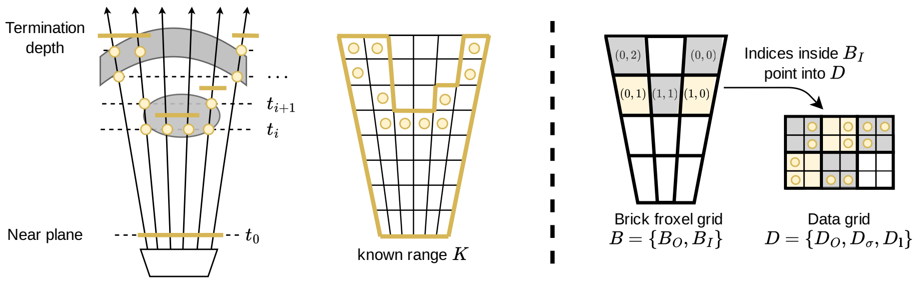

Termination depth
t_{i+1}
t_i
Near plane
t_0
known range K
(0, 2)
(0, 0)
(0, 1)
(1, 1)
(1, 0)
Indices inside B_I point into D
Brick froxel grid $B = \{B_O, B_I\}$
Data grid $D = \{D_O, D_\sigma, D_l\}$

Fig. 6. Visualization of the cache datastructures for a 2D toy example with a center object and an opaque background. During cache initialization, each ray places a number of samples at the same depths t_i, and performs early ray termination after reaching full opacity. The near plane and termination depths form a 2.5D frustum volume, representing the known range K. Memory is sparsely allocated for fixed-size bricks inside a froxel grid B, storing occupancy B_O, and indices B_I per brick. The indices B_I point into voxel grid D, which contains the actual per-sample cached data. Note that this is only a simplified visualization and our froxel grids are actually curved along the z-dimension when transformed to view-space.

requirements and computational overhead. However, the two interpolation samples are joined when computing $\hat{\mathbf{I}}$, and the N_c view-directions of the cone encoding can share the same Θ_{base} outputs. Assuming $N_c = 4$, this reduces the number of evaluations of Θ to $2\times\Theta_{base}$, $8\times\Theta_{neck}$, and $4\times\Theta_{head}$ per sample, ensuring fast optimization for our method.

5 NERF FRUSTUM VOLUME CACHING & REPROJECTION

We propose a caching and reprojection approach to retrieve interpolated NeRF latent information from a view-aligned sparse froxel grid and efficiently re-evaluate view-dependent effects. Our proposed datastructures and methodologies are designed to fulfill the following objectives: we aim to maximize cache lookup speed, minimize the cache size, and perform fast cache initialization during runtime. Our design decisions were guided by quality and performance evaluations on our highly optimized CUDA/C++ implementation.

5.1 Caching Datastructure

Due to its design, our NeRF model produces a low number of samples per ray, which are sparsely allocated inside the view frustum. We utilize a sparse froxel grid with fixed-size cubic bricks of side length N_B, enabling efficient parallel initialization and providing a favorable compromise between sparsity and lookup speed. We employ a *brick froxel grid* B that stores per-brick binary occupancy information B_O and indices B_I, pointing into the *data voxel grid* D where the brick's actual data resides. D stores per-sample latent information in three separate voxel grids: density D_σ, latent code D_l, and binary sample occupancy D_O. As neighboring bricks in B are not necessarily adjacent in D, prohibiting the use of efficient hardware interpolation, we opt to pad bricks inside D to store additional information of neighboring bricks.

We maintain an additional datastructure to accelerate occupancy checks by spanning a 2.5D frustum volume, representing the known range K of our cache. Essentially, each sample outside K is guaranteed to be unoccupied in B. We can easily construct this volume during cache initialization by using the current camera position, its corresponding near plane and each rays termination depth. For a visualization of our datastructures, cf. Fig. 6.

5.2 Cache Initialization

During cache initialization, all rays place their i-th sample at the same distance t_i, allowing us to use a *regular* froxel grid - note that all values are explicitly initialized as zero. A naïve initialization strategy would let all rays perform sample placement and early ray termination independently, only synchronizing for the allocation of bricks. Regrettably, this can leave bricks underutilized if most rays already terminated, but a few distant rays remain and keep placing samples.

We mitigate this limitation with *brick-wise initialization*, operating with brick-sized ray bundles of dimension $N_B \times N_B$ instead of individual rays. Ray bundles always perform stepping collectively, only terminate as a whole, and never terminate before reaching the end of the currently sampled brick in B. Additionally, bricks are only allocated if any ray of the bundle places a valid sample, leading to much higher utilization of bricks and better memory coherence. This strategy maximizes brick occupancy and lookup performance, but leads to slower cache initialization as collective ray bundle termination inevitably leads to more evaluated samples. Note that we differentiate between *inner* and *outer* rays when using padded bricks. Only *inner* rays are considered for ray bundle termination and can initiate brick initialization, whereas *outer* rays merely place samples when bricks are initialized.

5.3 Reprojection & Sampling

With our caching approach established, we show our algorithm for sampling in Alg. (1). We transform samples $\mathbf{x}$ into froxel-space $\mathcal{Z}$ via Eqn. (6). A sample $\mathbf{z} \in \mathcal{Z}$ is inside the known range K, if $\mathbf{z}_z$ lies between the cache's near plane and the termination depth at $T[\mathbf{z}_x, \mathbf{z}_y]$. If B_O is occupied, we determine the position $\mathbf{d}$ inside D by *expanding* the brick index from B_I and adding $\mathbf{z}$'s offset inside the current brick. Finally, we count this sample as a cache-hit if D_O is occupied at $\mathbf{d}$ and opacity exceeds a threshold τ_α, to perform our normalized trilinear interpolation as proposed in Eqn. (7). If a sample is outside K but occupied in O, we treat it as a cache-miss and evaluate Θ.

5.4 Implementation Details

Our implementation consists of a *pytorch* training framework, based on NerfAcc [Li et al. 2023], and an optimized CUDA/C++ real-time viewer and offline renderer. Both applications leverage *tiny-cuda-nn* [Müller 2021] for efficient MLP and input encoding inference. We provide our source code in the supplementary material.

Datastructures. To allow for better memory access patterns and hardware interpolation, we store the data of B and D in 3D textures. All values are stored in full-precision, except for $\mathbf{l}$, which is natively output by *tiny-cuda-nn* in *half*-precision. To further speed up cache retrieval, we group the entries of $D_\mathbf{l}$ into batches of four (*half4*), a format supported by CUDA and GPU hardware.

Sampling. Following iNGP [Müller et al. 2022], we perform sampling rounds with a fixed number of samples per ray, terminating and compacting the remaining rays after each round. In addition, we reject samples with opacity below $\tau_\alpha = 10^{-5}$ and perform early ray termination at transmittance $T < 10^{-4}$. We split our implementation of Alg. (1) across multiple kernels: the sampling kernel only performs the cache-hit detection and writes the t-values of each ray's cached and new samples to separate t-buffers. Individual rays might exclusively place cached or new samples during a round, leading to sparse sample buffers and unnecessary evaluations of Θ and Θ_{head}. Therefore, we compact the t-buffers after sampling, and perform the interpolation of $\hat{\mathbf{l}}$ and generation of network inputs in a separate per-sample kernel. Finally, to accelerate empty space skipping during ray initialization, we mesh and ray-trace the occupancy grid following Wald et al. [2021] to place the first sample.

ALGORITHM 1: Sampling, cache-hit detection & interpolation
NeRF model: Network $\Theta = \{\Theta_{base}, \Theta_{neck}, \Theta_{head}\}$; Occupancy grid O
Cache data: Brick froxel grid $B = \{B_O, B_I\}$; Data voxel grid $D = \{D_O, D_\sigma, D_\mathbf{l}\}$; Known range K

```
Data: Sampled 3D world position $\mathbf{x}$, view-direction $\mathbf{v}_t$, cache view-direction $\mathbf{v}_c$, and step size $\delta$
Result: Sample color and density $(\mathbf{c}, \sigma)$:
        Interpolated from cache (cache-hit), resampled (cache-miss), or NULL if rejected
if $O[\mathbf{x}]$ occupied then
    $\mathbf{z} \leftarrow F(\mathbf{x})$ ;                // Transform x into froxel-space $\mathcal{Z}$, following Eqn. (6)
    if $\mathbf{x} \in K$ then
        if $B_O[\mathbf{z}]$ occupied then
            $\mathbf{d} \leftarrow \text{expand}(B_I[\mathbf{z}], \mathbf{z})$ ;        // Retrieve position inside $D$ via z and $B_I$
            $\hat{\sigma} \leftarrow \text{trilerp}(\mathbf{d}, D_\sigma)$ ;
            if $D_O[\mathbf{d}]$ occupied and $(1 - \exp(\delta\hat{\sigma})) > \tau_\alpha$ then
                $\hat{\mathbf{l}} \leftarrow \frac{\text{trilerp}(\mathbf{d}, D_\mathbf{l})}{\text{trilerp}(\mathbf{d}, D_o)}$ ;      // Cache-Hit $\Rightarrow$ Interpolate $\{\hat{\sigma}, \hat{\mathbf{l}}\}$ from $D$ ...
                return $(\mathbf{c} \leftarrow \Theta_{head}(\hat{\mathbf{l}}, \mathbf{x}, \mathbf{v}_t, \mathbf{v}_c), \hat{\sigma})$ ;     // ... and re-evaluate only $\Theta_{head}$
            end
        end
        return NULL ;                                   // Rejected by $B_O$, $D_O$ or $\tau_\alpha$
    else
        return $(\mathbf{c}, \sigma) \leftarrow \Theta(\mathbf{x}, \mathbf{v}_t)$ ;          // Cache-Miss $\Rightarrow$ evaluate $\Theta$ entirely
    end
end
return NULL ;                                           // Rejected by $O$
```

Fused Head Network. Our potential speedup is clearly dependent on the runtime of Θ_{head}, which is evaluated for each cached sample. Our model employs two additional input encodings $\psi(\mathbf{x})$ and $\gamma(\mathbf{v}_c)$, which need to be communicated via slow global memory, if implemented naïvely. We thus opt for a custom fused kernel that performs the full evaluation of Θ_{head}, thereby only requiring communication of $(\mathbf{x}, \mathbf{v}_t, \mathbf{v}_c)$ instead of their much larger encodings. Each thread is responsible for loading and encoding one input sample, performing cooperative tensor-core matrix multiplications for the input layer, and matrix-vector multiplication in registers for the output layer.

Real-time Viewer & Offline Renderer. Our real-time viewer is designed for high interactivity, providing automatic cache updates based on the cache-hit ratio (CHR). We hide latency with double-buffered caches and run initialization asynchronously in a lower priority stream, enhancing user experience. Our offline renderer can be used to render high-quality images and video sequences with supersampled motion blur and depth of field, adapted from iNGP [Müller et al. 2021].

6 RESULTS

We evaluate our proposed model and caching approach on challenging, real-world scenes from the Mip-NeRF 360 dataset [Barron et al. 2022]. First, we demonstrate the effectiveness of our proposed model components and training schemes by comparing against other popular NeRF models. Additionally, we evaluate the performance and quality of our caching approach, when applied to our trained models. We provide additional implementation details in Supplemental B.

Models. We refer to our proposed model from Sec. 4.4 as *Ours*, using $N_c = 4$ samples and degree $\theta_{\max} = 25°$ for optimizing the cone encoding, and 4 degrees for the frequency encoding $\gamma_4(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}^{24}$. To evaluate the effectiveness of our proposed model, we compare against a

Table 1. Standard image quality metrics and average number of samples N_S per ray for different models on the Mip-NeRF 360 dataset [Barron et al. 2022]. Results with (‡) are reproduced from Kerbl et al. [2023].

Dataset	Mip-NeRF 360 Indoor					Mip-NeRF 360 Outdoor				
Method	PSNR↑	SSIM↑	LPIPS↓	FLIP↓	mean(N_S)↓	PSNR↑	SSIM↑	LPIPS↓	FLIP↓	mean(N_S)↓
Plenoxels‡	24.84	0.765	0.366	0.182	-	21.69	0.513	0.467	0.229	-
Mip-NeRF 360‡	31.57	0.914	0.182	0.088	-	24.42	0.691	0.286	0.170	-
3DGS‡	30.41	0.920	0.190	0.099	-	24.64	0.732	0.233	0.167	-
iNGP (big)†	29.44	0.866	0.257	0.108	35.02	23.07	0.547	0.425	0.197	57.20
iNGP Ours	30.32	0.889	0.219	0.101	12.94	24.18	0.653	0.317	0.174	21.73
Ours	30.14	0.891	0.220	0.104	12.92	24.23	0.659	0.312	0.173	20.18
Ours 5°	30.38	0.891	0.218	0.101	12.80	24.25	0.659	0.311	0.173	20.69
Ours (huge)	30.58	0.901	0.203	0.101	12.53	24.52	0.682	0.289	0.168	21.15

standard variant of iNGP (big) with scene contraction, distortion loss, gradient scaling and our interpolation training, which we dub *iNGP Ours*. Both models use MLP layers of width 128, and a hash-grid encoding with 8 levels, 4 features, and 2^{21} hash entries, growing from 16^3 to 4096^3 in resolution. *Ours*, in contrast to iNGP, uses MLP layer counts for $(\Theta_{base}/\Theta_{neck}/\Theta_{head})$ of $(1/2/1)$, and smaller latent codes with $N_l = 8$. To showcase the scalability of our caching approach, we add a version of our model with a larger hash grid (10 levels, 2^{22} entries, and max. resolution of 8192^3), called *Ours* (huge).

6.1 Model Evaluation

For the quantitative evaluation of our trained models, we report PSNR, SSIM [Wang et al. 2004], LPIPS [Zhang et al. 2018] and ꟻLIP [Andersson et al. 2020] in Tab. 1. We also include numbers for our re-implementation of iNGP (big)† [Müller et al. 2022], where we use the same hash grid configuration as for *Ours*. To facilitate cross-method comparisons, we also include metrics for 3DGS [Kerbl et al. 2023], Mip-NeRF 360 [Barron et al. 2022] and Plenoxels [Fridovich-Keil et al. 2022], which are reproduced from Kerbl et al. [2023] and marked with ‡.

Our evaluations show that *Ours* is able to maintain competitive quality and reduce sample counts, even though it uses smaller latent codes and a smaller Θ_{head}. Reducing $\theta_{\max}$ of the cone encoding to 5° improves quality slightly, but struggles when re-evaluating the latent codes for larger view-changes (see Supplemental D for details). *Ours* (huge) performs best out of our methods and rivals even Mip-NeRF 360 and 3DGS, but leads to slower render times and considerably larger model size compared to the (big) variant. Contrasting *iNGP Ours* with the base iNGP model showcases the effect of the distortion loss and scene contraction, reducing the sample count drastically and improving image quality.

On average, training time increases slightly from 20 to 24 minutes when enabling interpolation training in *iNGP Ours*, and to 47 minutes with our cone encoding (*Ours*). Although each sample is evaluated 8×, training times only increase by 2.5×, which is achieved by sharing common computations (cf. Sec. 4.4). For comparison, 3DGS trains for 30–40 minutes on our system, and Mip-NeRF 360 for 12 hours on 4 A100 GPUs[1].

6.2 Caching & Reprojection

To evaluate our cache-based rendering for different view changes, we utilize the test set views (at position $\mathbf{o}$ with view-direction $\mathbf{v}$) from Mip-NeRF 360 [Barron et al. 2022] and initialize our

[1]As reported by Kerbl et al. [2023]

Table 2. Qualitative evaluation based on PSNR image metrics when rendering from our cache for different rotational/translational viewpoint changes. Models trained without induced linearity deliver significantly degraded image quality during cache rendering. *Ours* can faithfully render from cache even for small latent code sizes, if trained for a large enough cone angle.

Dataset	Mip-NeRF 360 Indoor						Mip-NeRF 360 Outdoor					
		Rotation			Translation			Rotation			Translation	
Method	No Cache	5°	10°	15°	+	−	No Cache	5°	10°	15°	+	−
Ours	30.12	30.25	30.18	30.05	30.29	30.14	24.22	24.31	24.28	24.23	24.35	24.23
Ours w/o linearity	30.27	24.52	24.65	24.73	25.73	23.53	24.28	20.58	20.67	20.75	21.41	19.94
Ours 5°	30.35	30.29	29.60	28.34	30.49	30.33	24.28	24.31	24.04	23.40	24.39	24.27
Ours 45°	30.02	30.15	30.09	29.99	30.19	30.04	24.17	24.26	24.24	24.20	24.30	24.19
Ours $N_l = 4$	30.03	30.16	30.08	29.95	30.20	30.07	24.18	24.27	24.23	24.17	24.31	24.20
iNGP Ours	30.30	30.43	30.37	30.28	30.47	30.31	24.17	24.25	24.23	24.19	24.28	24.17
iNGP Ours w/o linearity	30.56	29.84	29.83	29.81	30.10	29.67	24.28	23.93	23.93	23.93	24.06	23.83
iNGP Ours $N_l = 8$	30.15	29.70	29.68	29.64	29.89	29.54	24.02	23.80	23.79	23.78	23.90	23.71

cache from different locations in the vicinity of $\mathbf{o}$. To quantify the effect of small/large translation, we move the cache initialization camera's position $\mathbf{o}_{\text{trans}}$ along $\mathbf{v}$, *i.e.* $\mathbf{o}_{\text{trans}} = \mathbf{o} + t \cdot \mathbf{v}$, with $t \in \{\pm \frac{\Delta t_{\min}}{2}, \pm 0.1, \pm 0.25\}$. Similarly, we create rotated initialization views $\mathbf{o}_{\text{rot}}$ by rotating $\mathbf{o}$ around a reference point along $\mathbf{v}$, with rotation angles θ and ϕ (see Supplemental E for details). Following [Lochmann et al. 2016], we shift the viewing-angle by $\theta \in \{5°, 10°, 15°\}$, and sample each θ at six different angles $\phi \in \{0°, 60°, 120°, 180°, 240°, 300°\}$. We utilize this setup for both performance and qualitative evaluation, reporting average results for θ and positive/negative t. All evaluations use padded bricks of size $N_B = 6$ for our cache datastructure.

Image Metrics. We provide a comparison based on PSNR for positive/negative translation and different rotation angles θ in Tab. 2. Both *Ours* and *iNGP Ours* are able to faithfully render the test set views from cache, even for larger rotational movements. When linearity is not induced during training, quality degrades significantly. Contrary to *iNGP Ours*, *Ours* is able to faithfully render from cache, even when latent code size is further reduced. Our cone encoding delivers higher initial quality when trained for smaller cone angles, however, quality degrades heavily for larger rotational view changes. Example images for a single test view can be seen in Fig. 7.

Performance & Cache Size. We evaluate the average performance over all scenes of the Mip-NeRF 360 dataset [Barron et al. 2022] for different configurations of *Ours* and *iNGP Ours* in Tab. 3. The timings are measured on an NVIDIA RTX 4090, and averaged over 10 runs in FullHD resolution. *iNGP Ours* has the fastest baseline performance but only experiences slight speed-ups when rendering from the cache, as evaluation of Θ_{head} is expensive for this model. Reducing N_l for *iNGP Ours* from 16 to 8 leads to a decreased memory footprint, but increases performance only slightly. *Ours* experiences large speedups when employing caching, especially for smaller rotational movements where the CHR is high. Scaling up Θ_{base} in *Ours* (huge) leads to slower overall render times, however, we can take full advantage of caching if CHR is high enough. Finally, changing N_l for *Ours* has a considerable impact on cache size and speedup.

Cache initialization times and cache size are also highly dependent on brick size N_B and padding choice, with initialization for *Ours* and $N_B = 6$ (w/ padding) taking 161ms and using 3.41 GiB of memory on average, while $N_B = 8$ (w/o padding) requires 75ms and 1.64 GiB (cf. Supplemental G). Our double-buffered initialization allows us to hide this latency over multiple frames.

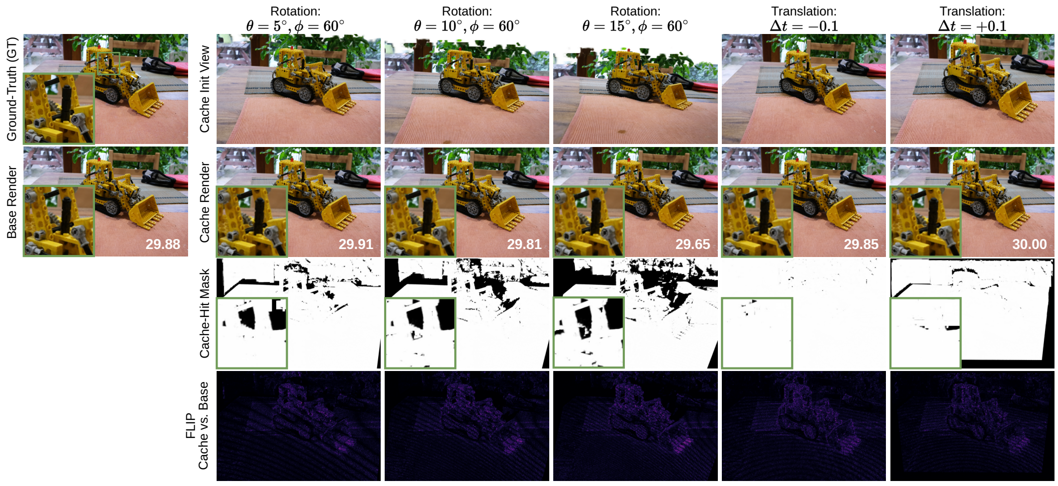

Rotation: $\theta = 5°, \phi = 60°$
Rotation: $\theta = 10°, \phi = 60°$
Rotation: $\theta = 15°, \phi = 60°$
Translation: $\Delta t = -0.1$
Translation: $\Delta t = +0.1$
Ground-Truth (GT)
Base Render
Cache Init View
Cache Render
Cache-Hit Mask
FLIP Cache vs. Base
29.88
29.91
29.81
29.65
29.85
30.00

Fig. 7. Qualitative comparison of our cache-based rendering approach for an example view of the Kitchen scene, rendered with *Ours*. The cache is rendered from the test set viewpoint, but initialized from a slightly rotated/translated viewpoint in its vicinity. We visualize the contribution of cache-hit vs. cache-miss samples and provide a ꟻLIP comparison to the baseline render (= without caching). As can be seen in the inset PSNR scores (compared to the ground truth image), we maintain the baseline quality and even exceed it in some configurations. Note that the stripes in the ꟻLIP images appear due to deterministic sample placement during rendering, and smooth out when rendering multiple "jittered" rays per pixel.

Table 3. Performance and cache size (w/o double-buffering) ablation of our proposed methods. We report speedup of each method compared to its baseline performance ("No Cache"). Times in ms for FullHD resolution.

Type Method	No Cache	$t = -\frac{\Delta t_{\min}}{2}$ CHR ~97%	$\theta = 5°$ CHR ~83%	$\theta = 10°$ CHR ~72%	$\theta = 15°$ CHR ~63%	Avg. Cache Size (GiB)
Ours	48.82	25.68 (1.90×)	28.38 (1.72×)	30.64 (1.59×)	32.63 (1.50×)	3.41
Ours $N_l = 16$	51.09	32.05 (1.59×)	34.16 (1.50×)	35.79 (1.43×)	37.55 (1.36×)	5.73
Ours $N_l = 4$	51.36	24.62 (2.09×)	27.99 (1.83×)	30.47 (1.69×)	32.71 (1.57×)	2.34
Ours (huge)	60.33	26.38 (2.29×)	31.61 (1.91×)	35.53 (1.70×)	38.84 (1.55×)	3.39
Ours w/o fused Θ_{head}	64.70	43.62 (1.48×)	44.94 (1.44×)	46.42 (1.39×)	48.08 (1.35×)	3.41
iNGP (big)†	124.30	-	-	-	-	-
iNGP Ours	46.79	43.32 (1.08×)	42.69 (1.10×)	42.35 (1.10×)	42.46 (1.10×)	5.69
iNGP Ours $N_l = 8$	49.44	42.46 (1.16×)	42.50 (1.16×)	42.56 (1.16×)	43.01 (1.15×)	3.58

The speedups are heavily dependent on the efficiency of Θ_{head}, as can be seen when we disable our custom fused Θ_{head} implementation. We include render times for iNGP (big)† to put the other results into perspective, indicating that *Ours* can achieve speedups of 3.8× to 4.8× compared to iNGP (big). We show detailed per-stage timings of the cache rendering pipeline in Supplemental F.

Video Sequence. To showcase scalability of our approach for high-quality offline rendering, we measure performance for a 300 frame video sequence of the Stump and Bonsai scene. We initialize the cache with double the sampling rate along the view ray to prevent any possible undersampling, reaching a maximum cache size of 5.15 GiB for the Stump scene, and 3.95 GiB for Bonsai, with only ~2.5% of bricks occupied in the sparse brick froxel grid. The cache is re-initialized automatically after the CHR drops below 85%, which occurs around 50 times in both scenes and takes around 200ms per initialization. Considering overall render times, including the time spent for cache initialization, we achieve an average speed-up of ~1.84× through our caching approach, without any noticeable loss in quality (see supplementary videos and Supplemental H for details).

7 LIMITATIONS & FUTURE WORK

Even though our work shows promising results, there are limitations to its application. The most obvious limitation is cache size, which can become prohibitively large for lower-end graphics cards, especially on higher resolutions. Secondly, even though cache initialization can be performed efficiently at runtime, it can still lead to a worse user experience during real-time rendering, if performed naïvely. Our double-buffered initialization hides latency to a high extent, however, more sophisticated asynchronous approaches or prediction mechanisms would be required to completely mitigate this drawback. This is not an issue for offline-rendering, where initialization times are amortized by faster rendering and longer cache reuse. Furthermore, our approach currently relies on a deterministic step size function to perform cache lookups, which prohibits the usage of proposal sampling techniques. Finally, choosing the stepping intervals too large during cache initialization can lead to undersampling, which could be solved by using anti-aliased sampling techniques, as proposed by Zip-NeRF [Barron et al. 2023]. Applying our method to more expensive models could also lead to more significant speedups, but would only be feasible for offline-rendering. Another promising application is virtual reality, where a single cache can be used for both viewpoints and reused across multiple frames.

While our method is currently outperformed by 3DGS [Kerbl et al. 2023] and mesh-based methods [Chen et al. 2023] in terms of performance, volumetric NeRF approaches do not exhibit some of their drawbacks. 3DGS suffers from popping artifacts (discussed in detail in Radl et al. [2024]), slight inaccuracies due to their projection approximation, and comparatively large models. Mesh-based methods struggle to faithfully represent thin structures and semi-transparent surfaces, and often require an additional baking step.

Reducing cache size through compression, *e.g.* by consolidating similar latent codes hierarchically, is a first logical next step. Additionally, updating the caching datastructure continuously with the information of cache-miss samples could help improve performance and postpone the next cache initialization. Finally, although we only tested our method on static NeRF scenes, it would be intriguing to apply it to dynamic NeRFs by using a bi-directional deformation field.

8 CONCLUSION

In this paper, we examined common NeRF network architectures in detail and proposed a caching and reprojection approach to exploit their underlying structure. Our method allows for temporal reuse of NeRF samples via a view-aligned, fully volumetric cache representation, while enabling re-evaluation of view-dependent effects. We are able to accelerate rendering by caching expensively computed view-independent latent codes, and proposed efficient cache sampling algorithms and training schemes, to further improve quality and performance. Additionally, we introduced a novel view-dependent cone encoding that allows for much smaller latent codes, thereby decreasing memory requirements and further improving performance when rendering from the cache. Our approach scales exceptionally well with larger models and can speed up offline rendering with expensive effects by up to 2×, without requiring any baking of the underlying NeRF model. Furthermore, our insights into interpolation and caching of volumetric sample latent codes have the potential to inspire further developments in classical volume rendering. We think that temporal coherence methods will be essential going forward, especially for expensive, high-quality NeRF models. The source code of our training framework and renderer are publicly available (after review).

REFERENCES

Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. 2020. FLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3, 2, Article 15 (2020), 23 pages.

Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. 2021. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.*

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2023. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.*

Huw Bowles, Kenny Mitchell, Robert W. Sumner, Jeremy Moore, and Markus Gross. 2012. Iterative Image Warping. *Computer Graphics Forum* 31, 2 (2012), 237–246.

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *Proceedings of the European Conference on Computer Vision.*

Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. 2023. MobileNeRF: Exploiting the Polygon Rasterization Pipeline for Efficient Neural Field Rendering on Mobile Architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

Daniel Duckworth, Peter Hedman, Christian Reiser, Peter Zhizhin, Jean-François Thibert, Mario Lučić, Richard Szeliski, and Jonathan T. Barron. 2024. SMERF: Streamable Memory Efficient Radiance Fields for Real-Time Large-Scene Exploration. *ACM Transactions on Graphics* 43, 4, Article 63 (2024).

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields without Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. 2021. FastNeRF: High-Fidelity Neural Rendering at 200FPS. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.*

Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. 1998. The Irradiance Volume. *IEEE Computer Graphics and Applications* 18, 2 (1998), 32–43.

Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. 2021. Baking Neural Radiance Fields for Real-Time View Synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.*

Sebastien Hillaire. 2015. Towards Unified and Physically-Based Volumetric Lighting in Frostbite. (2015). In SIGGRAPH Advances in Real-Time Rendering in Games course.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4, Article 139 (2023).

Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations.*

Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. 2022. AdaNeRF: Adaptive Sampling for Real-time Rendering of Neural Radiance Fields. In *Proceedings of the European Conference on Computer Vision.*

Ruilong Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. 2023. NerfAcc: Efficient Sampling Accelerates NeRFs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.*

David B Lindell, Julien NP Martel, and Gordon Wetzstein. 2021. AutoInt: Automatic Integration for Fast Neural Volume Rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields.

Gerrit Lochmann, Bernhard Reinert, Arend Buchacher, and Tobias Ritschel. 2016. Real-time Novel-view Synthesis for Volume Rendering Using a Piecewise-analytic Representation. In *Proceedings of the Conference on Vision, Modeling and Visualization.*

William R. Mark, Leonard McMillan, and Gary Bishop. 1997. Post-Rendering 3D warping. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.*

Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. 2019. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *ACM Transactions on Graphics* 38, 4 (2019), 1–14.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Proceedings of the European Conference on Computer Vision.*

Klaus Mueller, Naeem Shareef, Jian Huang, and Roger Crawfis. 1999. IBR-Assisted Volume Renderin. In *Proceedings of IEEE Visualization Conference.*

Thomas Müller. 2021. *tiny-cuda-nn.* https://github.com/NVlabs/tiny-cuda-nn

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Transactions on Graphics* 41, 4, Article 102 (2022), 15 pages.

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time Neural Radiance Caching for Path Tracing. *ACM Transactions on Graphics* 40, 4, Article 36 (2021), 16 pages.

Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. 2021. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 45–59.

Diego Nehab, Pedro V. Sander, Jason Lawrence, Natalya Tatarchuk, and John R. Isidoro. 2007. Accelerating Real-Time Shading with Reverse Reprojection Caching. In *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware.*

Julien Philip and Valentin Deschaintre. 2023. Floaters No More: Radiance Field Gradient Scaling for Improved Near-Camera Training. In *Eurographics Symposium on Rendering.*

Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. 2024. StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering. *ACM Transactions on Graphics* 43, 4, Article 64 (2024).

Christian Reiser, Stephan Garbin, Pratul P. Srinivasan, Dor Verbin, Richard Szeliski, Ben Mildenhall, Jonathan T. Barron, Peter Hedman, and Andreas Geiger. 2024. Binary Opacity Grids: Capturing Fine Geometric Detail for Mesh-Based View Synthesis. *ACM Transactions on Graphics* 43, 4, Article 149 (2024).

Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. 2021. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.*

Christian Reiser, Rick Szeliski, Dor Verbin, Pratul P. Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. 2023. MERF: Memory-Efficient Radiance Fields for Real-time View Synthesis in Unbounded Scenes. *ACM Transactions on Graphics* 42, 4, Article 89 (2023), 12 pages.

Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. 1998. Layered Depth Images. In *Proceedings of the ACM on Computer Graphics and Interactive Techniques.*

Pratul P. Srinivasan, Richard Tucker, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. 2019. Pushing the Boundaries of View Extrapolation with Multiplane Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. In *Advances in Neural Information Processing Systems.*

Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, et al. 2023. Nerfstudio: A Modular Framework for Neural Radiance Field Development. In *SIGGRAPH.*

Jiaxiang Tang, Xiaokang Chen, Jingbo Wang, and Gang Zeng. 2022. Compressible-composable NeRF via Rank-residual Decomposition. In *Advances in Neural Information Processing Systems.*

Ingo Wald, Stefan Zellmann, and Nate Morrical. 2021. Faster RTX-Accelerated Empty Space Skipping using Triangulated Active Region Boundary Geometry. In *Eurographics Symposium on Parallel Graphics and Visualization.*

Ziyu Wan, Christian Richardt, Aljaž Božič, Chao Li, Vijay Rengarajan, Seonghyeon Nam, Xiaoyu Xiang, Tuotuo Li, Bo Zhu, Rakesh Ranjan, and Jing Liao. 2023. Learning Neural Duplex Radiance Fields for Real-Time View Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.

Bart Wronski. 2014. Volumetric Fog: Unified Compute Shader-Based Solution to Atmospheric Scattering. (2014). In SIGGRAPH Advances in Real-Time Rendering in Games course.

Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. 2023. BakedSDF: Meshing Neural SDFs for Real-Time View Synthesis. In *SIGGRAPH.*

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.*

Stefan Zellmann, Martin Aumüller, and Ulrich Lang. 2012. Image-Based Remote Real-Time Volume Rendering: Decoupling Rendering From View Point Updates. In *Proceedings of the International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.*

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

A CLAMPED EXPONENTIAL STEP SIZE

Instant-NGP [Müller et al. 2022] performs clamped exponential stepping for unbounded scenes, with $t_{i+1} = \min(\max(t_i \cdot (1+a), \Delta t_{\min}), \Delta t_{\max})$ for a small cone angle $a \in \mathbb{R}^+$ and a min/max step size $\Delta t_{\min}, \Delta t_{\max}$. We can derive an invertible stepping function $g(\cdot)$ as

$$g(i) = \begin{cases} t_0 + i \cdot \Delta t_{\min} & \text{if } i < i_{\min} \\ t_{i_{\min}} \cdot (1+a)^{1-i_{\min}} & \text{if } i_{\min} \leq i < i_{\max} \\ t_{i_{\max}} + (i - i_{\max}) \cdot \Delta t_{\max} & \text{if } i \geq i_{\max} \end{cases}, \tag{10}$$

$$g^{-1}(t) = \begin{cases} \frac{t-t_0}{\Delta t_{\min}} & \text{if } t < t_{i_{\min}} \\ i_{\min} + \log\left(\frac{t}{t_{i_{\min}}} - (1+a)\right) & \text{if } t_{i_{\min}} \leq t < t_{i_{\max}} \\ i_{\max} + \frac{t - t_{i_{\max}}}{\Delta t_{\max}} & \text{if } t \geq t_{i_{\max}} \end{cases}, \tag{11}$$

with boundaries in stepping space $i_{\min} = \frac{1}{a} - \frac{t_0}{\Delta t_{\min}}$, and $i_{\max} = \log\left(\frac{\Delta t_{i_{\max}}}{a \cdot t_{i_{\min}}} - (1+c)\right) + i_{\min}$.

B ADDITIONAL IMPLEMENTATION DETAILS

Dataset & Scaling. We use the 2× downsampled images for indoor scenes and 4× for outdoor scenes, following related work. The multi-resolution occupancy grid O grows exponentially with factor 2 from $[-1, 1]$ to $[-32, 32]$ for outdoor scenes (6 levels), and $[-16, 16]$ for indoor scenes (5 levels). Scene contraction is applied such that $[-2, 2]$ becomes the unwarped area.

Training. We optimize our model using Adam [Kingma and Ba 2015], with parameters $\epsilon = 10^{-15}, \beta_1 = 0.9, \beta_2 = 0.999$, and L_2 weight decay with $\lambda = 10^{-6}$. All our models are trained for 60k iterations, with learning rate η increasing linearly from 10^{-4} to 10^{-2} for the first 6k warmup iterations, followed by a cosine annealing schedule towards $\eta = 10^{-4}$. We use a target sample batch size of 2^{18}, and distill O as detailed in Müller et al. [2022]. The distortion loss $\mathcal{L}_{\text{dist}}$ is scaled by $\lambda_{\text{dist}} = 10^{-2}$, and gradient scaling is performed linearly up to a distance of 1 to the camera.

C MODEL COMPONENTS ABLATION

We perform an in-detail ablation for different model components and latent code sizes N_l for *Ours* and *iNGP Ours* in Tab. 4. Evidently, inducing spatial linearity during training decreases quality slightly, as this prevents the model from producing locally non-linear $\mathbf{l}$ and σ. Adding the frequency encoded sample position as input to Θ_{head} helps to mitigate the spatial information loss caused by interpolation and the smaller latent codes. Decreasing N_l leads to lower overall image metrics for both models, however, *Ours* is able to retain more quality. The distortion loss $\mathcal{L}_{\text{dist}}$ encourages the model to form proper surfaces, leading to better quality and fewer samples per ray.

D VIEW-CONSISTENCY OF CONE ENCODING

Our proposed view-dependent cone encoding needs to be trained for a maximum cone angle $\theta_{\max}$, up to which the latent code $\mathbf{l}$ can be re-evaluated. The default cone angle for *Ours* is $\theta_{\max}$=25°. We evaluate view-consistency for different view-changes and $\theta_{\max}$ through the following process: for all test views, we generate four view-directions $\mathbf{v}_c$, uniformly placed on a circle of radius θ on the unit sphere around the actual view-directions $\mathbf{v}_t$, such that $\mathbf{v}_c^T \mathbf{v}_t = \cos(\theta)$. We then evaluate $\mathbf{l}$ from $\mathbf{v}_t$ via Θ_{neck}, re-evaluate it through Θ_{head} for all four $\mathbf{v}_c$, and average the resulting image metrics. The quantitative results can be seen in Tab. 5, with an example visualization in Fig. 8.

Table 4. Ablation of different model configurations for *Ours* and *iNGP Ours*.

Dataset	Mip-NeRF 360 Indoor				Mip-NeRF 360 Outdoor			
Method	PSNR↑	SSIM↑	LPIPS↓	FLIP↓	PSNR↑	SSIM↑	LPIPS↓	FLIP↓
iNGP Ours	30.32	0.889	0.219	0.101	24.18	0.653	0.317	0.174
iNGP Ours w/o linearity	30.58	0.893	0.215	0.099	24.29	0.658	0.314	0.172
iNGP Ours w/o linearity w/o $\mathcal{L}_{\text{dist}}$	29.99	0.886	0.231	0.103	23.98	0.629	0.348	0.179
iNGP Ours w/ freq. enc.	30.47	0.890	0.218	0.100	24.25	0.657	0.313	0.172
iNGP Ours $N_l = 8$	30.10	0.885	0.223	0.104	24.03	0.646	0.326	0.178
iNGP Ours $N_l = 4$	29.45	0.872	0.238	0.112	23.75	0.631	0.342	0.186
Ours	30.14	0.891	0.220	0.104	24.23	0.659	0.312	0.173
Ours w/o linearity	30.29	0.891	0.222	0.102	24.28	0.660	0.311	0.172
Ours w/o freq. enc.	30.04	0.889	0.219	0.106	24.18	0.655	0.317	0.175
Ours $N_l = 16$	30.27	0.891	0.218	0.103	24.18	0.656	0.316	0.175
Ours $N_l = 4$	30.06	0.889	0.223	0.107	24.19	0.656	0.315	0.175
Ours 5°	30.38	0.891	0.218	0.101	24.25	0.659	0.311	0.173
Ours 45°	30.04	0.889	0.222	0.106	24.18	0.654	0.316	0.174

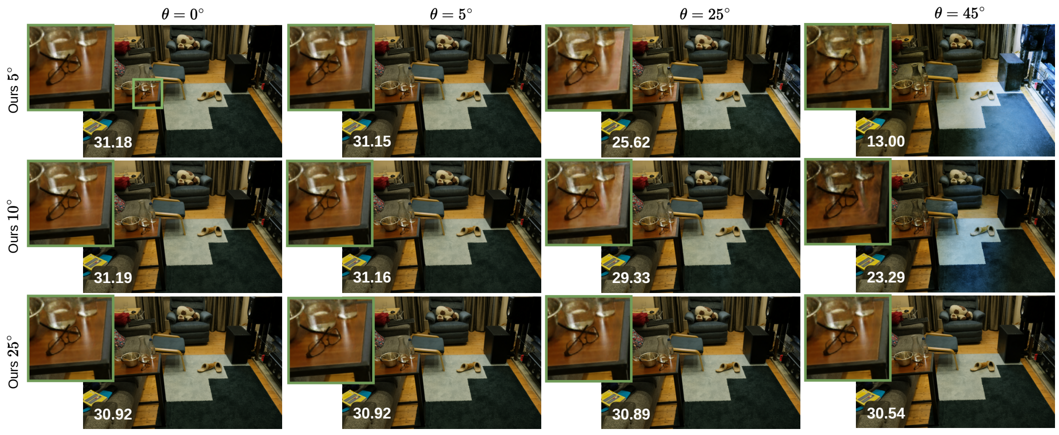


Fig. 8. View-consistency evaluation for our proposed model and cone encoding with different training degrees θ_{max} (*Ours* 5°–25°) and actual view-direction changes θ for the Room scene of the Mip-NeRF 360 dataset.

As expected, a smaller θ_{max} leads to a higher baseline quality, as Θ_{neck} can produce a more exact view-dependent encoding, and $\mathbf{l}$ needs to be less informative. However, re-evaluating $\mathbf{l}$ for larger view-direction changes fails catastrophically, as the model has not seen such large view-changes during training. Ultimately, there is a trade-off between quality and adaptability to view-changes, which can be chosen to accommodate the application's usecase.

E GENERATING OFFSET VIEW-DIRECTIONS

In order to evaluate our cache rendering after rotational view-changes, we aim to generate rotated views $\mathbf{o}_{\text{rot}}$ with view-direction $\mathbf{v}_{\text{rot}}$ for each test view at position $\mathbf{o}$ with view-direction $\mathbf{v}$. Therefore, we require some reference point $\mathbf{p}_{\text{ref}}$ along $\mathbf{v}$ to act as our rotation anchor. As Mip-NeRF 360 views are all inward facing towards the scene origin, we choose $\mathbf{p}_{\text{ref}}$ as the closest point to the scene origin along $\mathbf{v}$, determined via the projection of $-\mathbf{o}$ onto $\mathbf{v}$. We then use the scene's up-vector and

Table 5. View-consistency evaluation of our poposed model and cone encoding for different training degrees $\theta_{\max}$ on the Mip-NeRF 360 dataset.

Dataset	Mip-NeRF 360 Indoor				Mip-NeRF 360 Outdoor			
Method	PSNR	PSNR 5°	PSNR 10°	PSNR 25°	PSNR	PSNR 5°	PSNR 10°	PSNR 25°
Ours 5°	30.38	30.10	29.08	21.52	24.29	24.24	23.99	20.07
Ours 10°	30.29	30.23	30.01	26.99	24.26	24.25	24.20	22.72
Ours	30.14	30.14	30.12	29.94	24.23	24.23	24.22	24.17
Ours 45°	30.13	30.13	30.12	30.04	24.19	24.19	24.19	24.16

Table 6. Detailed timings for the individual stages of the cache rendering pipeline, for rotational and minimal translational view-changes. "Sample" refers to raymarching and sample placement with occupancy grid checks. "Misc" contains the color accumulation and compaction stages. Times in ms for FullHD resolution for *Ours*. Note: Summed up stage timings can differ slightly from total timings in other tables.

		Detailed Per-Stage Timing (ms)					
Type	CHR	Sample	Inference New	Cache Lookup	Inference Cache	Misc	Sum
$t = -\frac{\Delta t_{\min}}{2}$	97%	5.85	1.34	8.48	3.46	5.54	24.67 (1.94×)
$\theta = 5°$	83%	5.77	5.63	7.36	2.96	5.63	27.36 (1.75×)
$\theta = 10°$	72%	5.77	9.13	6.41	2.57	5.73	29.61 (1.62×)
$\theta = 15°$	63%	5.78	12.18	5.61	2.26	5.77	31.61 (1.51×)
w/o cache	-	5.13	37.02	-	-	5.73	47.88 (1.00×)

the vector $\mathbf{p}_{\text{ref}} - \mathbf{o}$ as rotation axes for our rotation angles (θ, ϕ) to receive the rotated view position $\mathbf{o}_{\text{rot}}$ and view-direction $\mathbf{v}_{\text{rot}} = \frac{\mathbf{p}_{\text{ref}} - \mathbf{o}_{\text{rot}}}{\|\mathbf{p}_{\text{ref}} - \mathbf{o}_{\text{rot}}\|}$.

F DETAILED PER-STAGE TIMINGS

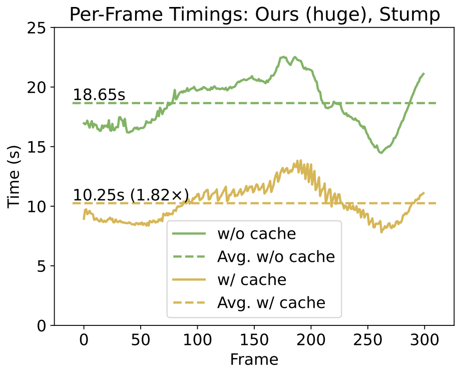
We report detailed timings for the individual stages of the rendering pipeline in Tab. 6. In the case of slight translation, where the CHR is close to 100%, *Ours* achieves speedups of almost 2×. Notably, already for rotational movements of $\theta = 10°$ (CHR =~72%), the inference of new samples (cache-misses) exceeds the combined cost of cache lookup and inference. Unfortunately, the rendering pipeline also spends a considerable amount of time with stages whose performance is independent of caching ("Misc" and "Sample"), thereby limiting the speedup potential of a caching approach. If we only consider the inference and cache lookup stages, we actually achieve speedups of 2.7×.

G PERFORMANCE ABLATION: BRICK SIZES

We evaluate the performance of our caching approach for *Ours* with different brick sizes N_B, and show their impact on cache size and cache initialization times. In theory, padding should allow for faster memory access, since it eliminates the need for interpolation across brick borders. We see this effect for larger brick sizes, however, this does not manifest for small bricks. One possible reason is the larger memory overhead, as padded bricks with $N_B = 6$ already use 57% of their values to store neighboring information, leading to larger cache sizes and cache initialization times. In general, larger bricks allow for less sparsity in the datastructure, therefore also increasing cache size. Note: we use a custom interpolation kernel for the no-padding case, where we launch eight threads per

Table 7. Ablation of performance and cache sizes/initialization times for different brick size N_B and padding configurations. Times in ms for FullHD resolution on *Ours*.

		5°		10°		15°		Avg. Cache Size (GiB)	Cache Init. Time
N_B	Pad?	Time	CHR	Time	CHR	Time	CHR		
6	yes	28.38	82%	30.64	71%	32.63	62%	3.41	161.30
8	no	28.08	84%	30.32	74%	32.40	65%	1.64	74.63
14	yes	26.96	86%	29.16	75%	31.30	66%	3.57	133.36
16	no	27.79	87%	29.92	76%	31.97	68%	2.54	92.31



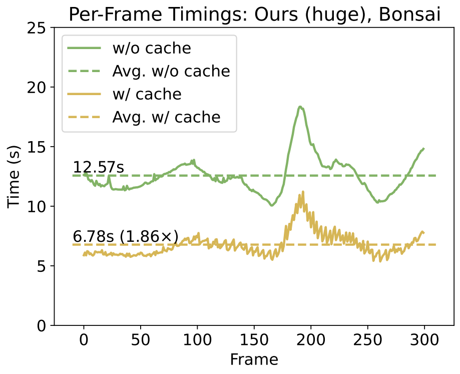


Fig. 9. Per-frame render timings (including cache initialization) for high-quality video sequences, rendering the Stump and Bonsai scene with 256 samples per ray in FullHD on *Ours* (huge).

sample and perform trilinear interpolation via shuffle operations. A naïve implementation might show a different runtime behavior.

H VIDEO SEQUENCE RENDERING

To showcase the capabilities of our caching approach for accelerating offline-rendering of high-quality videos, we render a 300 frame video sequence for the Stump and Bonsai scenes in FullHD, using 256× supersampled motion blur and depth of field. We use *Ours* (huge) for both video sequences, reporting average rendering time and per-frame timings in Fig. 9. Note that cache initialization is already accounted for in the render times.

For caching, we use $N_B = 16$ without padding and initialize the cache with twice the resolution along the view ray to prevent any undersampling. Note that this leads to larger cache sizes and longer cache initialization times, as well as slightly slower rendering from the cache. The cache is re-initialized around 50 times in both video sequences, taking around 200ms per initialization. Evidently, this overhead is quickly amortized through the increased rendering performance, especially since this cache is then sampled 256× per frame and reused across multiple frames. The small spikes in the plot signalize how render times increase as the CHR decreases, dropping back down as soon as the cache is re-initialized. Independent of caching, the performance is heavily influenced by the overall number of samples per ray, which is dependent on the currently rendered scene content.

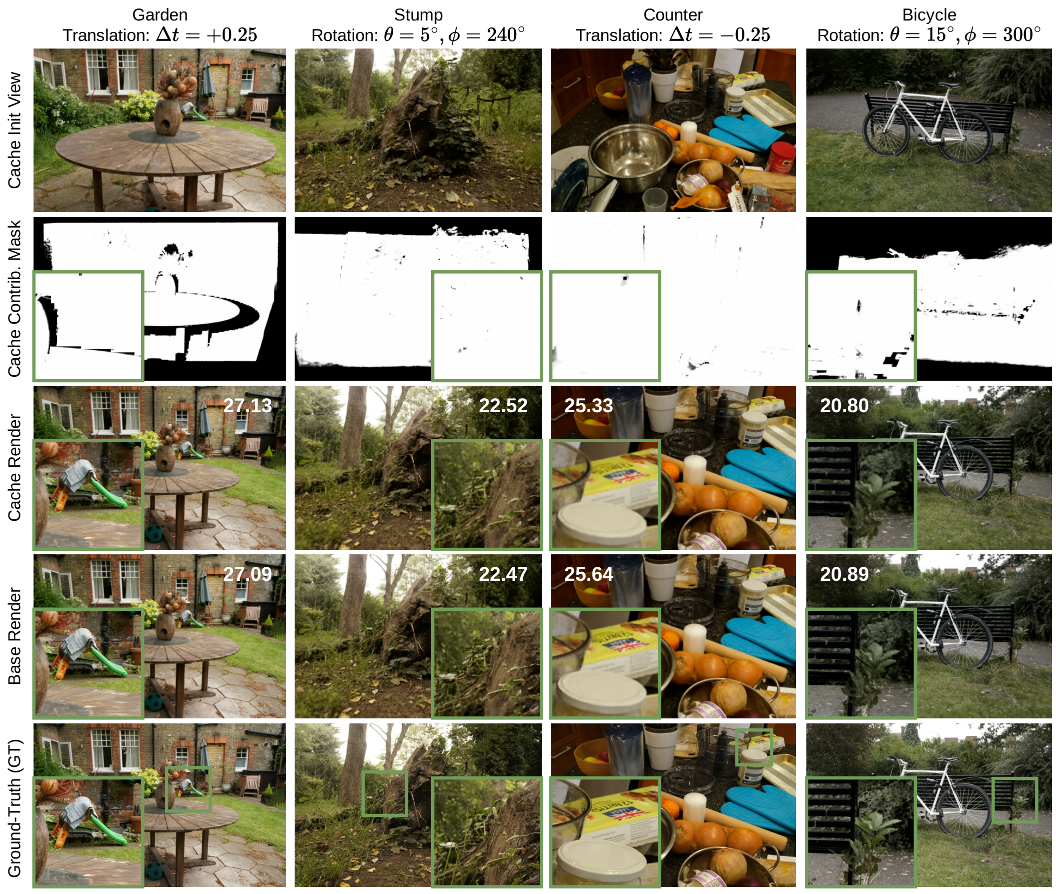


Fig. 10. Example images from our caching evaluation for different scenes of the Mip-NeRF 360 dataset, rendered *Ours*. We shift the cache initialization view away from our test set views via rotation or translation, and render this cache from the test set view. We also visualize the contribution of cache-hit samples vs. cache-miss samples. For comparison, we also show the base render (without caching) and the ground-truth images, as well as PSNR image metrics between base/cache renderings and ground-truth images.

I RENDERING FROM CACHE: QUALITATIVE IMAGE COMPARISON

We show additional examples from our qualitative cache rendering evaluation in Fig. 10. Our cache rendering approach is able to seamlessly blend the contributing samples from cache with the new samples. We also report PSNR for our cache renderings and the baseline images, compared to the ground-truth test set images. Notably, the cache rendered images exceed the baseline images in PSNR on several occasions. This is generally the case if the cache is initialized closer to the scene content, *i.e.* translation "+", as this effectively increases the cache's resolution when sampled from further back. The drawback is that it produces more cache-misses. The opposite effect can be observed when initializing the cache from further away, *i.e.* translation "-".