Neural Harmonic Textures for High-Quality Primitive Based Neural Reconstruction

Jorge Condor[1,2], Nicolas Moënne-Loccoz[1], Merlin Nimier-David[1], Piotr Didyk[2], Zan Gojcic[1], and Qi Wu[1]

[1] NVIDIA
{nicolasm,mnimierdavid,zgojcic,qiwu}@nvidia.com
[2] Università della Svizzera italiana, Lugano, Switzerland
{jorge.condor,piotr.didyk}@usi.ch

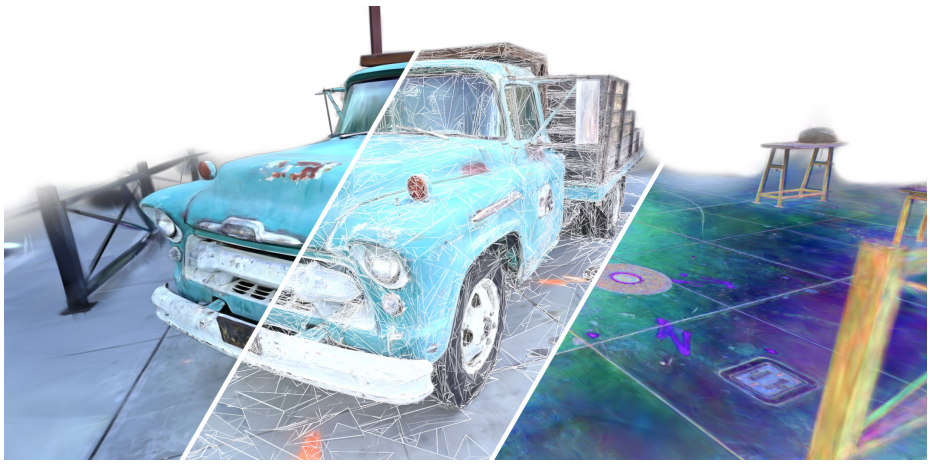


Fig. 1: Neural Harmonic Textures for novel view synthesis. We attach learnable feature vectors (right) to the virtual vertices of bounding tetrahedra encapsulating each primitive (center). After harmonic encoding and accumulation along the ray, a small neural network decodes the resulting signal into RGB color in a deferred manner (left). Source code and further results are available at `https://research.nvidia.com/labs/sil/projects/neural-harmonic-textures/`.

Abstract. Primitive-based methods such as 3D Gaussian Splatting have recently become the state-of-the-art for novel-view synthesis and related reconstruction tasks. Compared to neural fields, these representations are more flexible, adaptive, and scale better to large scenes. However, the limited expressivity of individual primitives makes modeling high-frequency detail challenging. We introduce *Neural Harmonic Textures*, a neural representation approach that anchors latent feature vectors on a virtual scaffold surrounding each primitive. These features are interpolated within the primitive at ray intersection points. Inspired by Fourier analysis, we apply periodic activations to the interpolated features, turning alpha blending into a weighted sum of harmonic components. The resulting signal is then decoded in a single deferred pass using a small

neural network, significantly reducing computational cost. Neural Harmonic Textures yield state-of-the-art results in real-time novel view synthesis while bridging the gap between primitive- and neural-field-based reconstruction. Our method integrates seamlessly into existing primitive-based pipelines such as 3DGUT, Triangle Splatting, and 2DGS. We further demonstrate its generality with applications to 2D image fitting and semantic reconstruction.

1 Introduction

Since the introduction of 3D Gaussian Splatting [24], (Lagrangian) primitive-based approaches have largely displaced (Eulerian) continuous neural radiance fields [44, 50, 56] for novel view synthesis [20, 24].

This shift is driven not only by significantly improved rendering speed, but also by the structural advantages of explicit representations. Primitive-based methods naturally adapt to scene detail, scale more gracefully, and readily support motion, deformation, and editing [5, 62]. Furthermore, they align well with feed-forward reconstruction pipelines [31,72] and closely resemble widely adopted point-map representations [59, 61].

Despite these advantages, the limited expressive power of individual primitives remains a fundamental bottleneck. Geometry and appearance are tightly coupled within each primitive, forcing high-frequency spatial detail to be represented by increasing the number of primitives, which directly increases memory consumption and hampers rendering speed. Directional appearance modeling is constrained in a similar manner. View-dependent effects are typically represented using low-order Spherical Harmonics, which are spectrally band-limited and poorly suited for high-frequency phenomena such as sharp specular highlights. Alternative angular parametric functions, such as spherical Gaussians, spherical Beta kernels [33], and spherical Voronoi [9] increase directional bandwidth, but leave the fundamental coupling between spatial support and appearance modeling unchanged. In contrast, neural radiance fields achieve high local representational capacity by combining positional encodings such as Fourier features [41,57] or multi-resolution hash grids [44] with a learned multi-layer perceptron (MLP) decoding. Recent works [37,73] attempted to bridge both paradigms by employing primitives as acceleration structures for querying a global neural field. However, these hybrid approaches inherit key limitations of global neural representations, particularly with respect to motion, deformation, editability and scalability.

In this work, we introduce *Neural Harmonic Textures*, a novel primitive-based neural representation that increases the expressive power of individual primitives while retaining the advantages of Lagrangian representations. To this end, we reinterpret primitives as both geometric carriers and local positional encodings. Specifically, we attach learnable feature vectors to a virtual scaffold encapsulating each primitive (e.g., Gaussians or triangles) and interpolate them locally at ray–primitive intersection points. Compared to globally anchored feature repre-

sentations, our formulation moves with the primitives and therefore naturally supports operations such as motion, deformation, and scene editing.

In prior work, such interpolated features are typically processed by a lightweight MLP to decode RGB values, followed by alpha compositing across intersected primitives to produce the final color, requiring numerous neural evaluations. Instead, we draw inspiration from the Fourier transform, where a complex signal is expressed as a sum of harmonic components (i.e., periodic functions with different amplitudes). We therefore apply periodic functions (sine and cosine) to the interpolated feature vectors before compositing them along the ray. Under this formulation, the activated features act as frequency components, while the primitive opacity modulates their amplitude (Fig. 2). The resulting implicit signal in image space is sufficiently rich to be directly decoded using a lightweight MLP in a single evaluation per pixel, in a deferred-shading manner.

We validate *Neural Harmonic Textures* on standard novel view synthesis benchmarks, achieving state-of-the-art performance among both real-time and offline methods (Sec. 5.1). Our proposed formulation is agnostic to the underlying primitive type: triangles, 2D or 3D Gaussians, tetrahedra, and other elements can be used interchangeably, enabling seamless integration into existing primitive-based pipelines while significantly increasing their representational capacity (Sec. 5.2). Finally, by abstracting the encoded signal, our approach naturally extends to higher-dimensional signals. Joint modeling of color and semantic features becomes possible within a unified local representation, allowing us to exploit both spatial and cross-modal correlations for improved efficiency and compactness (Sec. 5.2).

2 Related Work

2.1 Neural Fields

Neural fields have become a foundational model for representing complex, multi-scale signals across a wide range of domains [67], including audio [36], images [39, 49], 3D geometry [30, 46, 55, 60], radiance fields [41], scientific data [12, 63, 64], and volumetric video [11, 47]. These methods primarily differ in how they encode their inputs and parameterize their outputs. In most formulations, latent features are stored in a compressed representation and subsequently decoded by a neural network. This storage can be implicit within the network weights [34, 35, 41, 43, 48, 51, 57], or explicit through dedicated spatial data structures. Explicit representations typically offer improved scalability and performance, though often at increased memory cost. Common examples include voxel grids [39], triplanes [4, 11], point clouds [14], hash grids [44], and meshes [58]. In addition, explicit spatial structures are frequently used as acceleration mechanisms to skip empty regions and efficiently identify relevant query locations [32, 44, 73]. More recently, neural fields have been combined with primitive-based methods by turning the primitives into tiny neural fields with closed-form antiderivatives, enhancing their individual modelling power [77].

Beyond the choice of spatial representation, prior work has shown that positional encodings play a crucial role in enabling neural networks to represent high-frequency signals in low-dimensional spatial domains [41, 44, 51, 57]. In contrast to globally anchored spatial structures such as hierarchical voxel grids (hashgrid encodings), which follow an Eulerian formulation, our approach is Lagrangian. We anchor latent feature vectors relative to explicit particles, enabling adaptive spatial resolution and naturally supporting explicit motion, deformation, and scene editing.

2.2 Primitive-based Representations for Neural Reconstruction

Primitive-based methods have recently gained significant traction for 3D reconstruction and novel view synthesis, largely driven by the success of 3D Gaussian Splatting [24]. Numerous extensions have since explored variants of Gaussian primitives [7,8,21,28,33,42,62,65] as well as alternative representations, including Voronoi cells [13], triangle soups [20], and connected meshes [37]. While offering high rendering performance, editability, and interpretability, these approaches typically exhibit poor memory scaling when fitting high-frequency detail due to their joint modeling of geometry and appearance.

Neural field methods, in contrast, aggregate information across the entire scene, enabling compact representations. Concurrent work [38, 73] attempts to bridge this gap by using geometric primitives as acceleration structures for neural field evaluation. However, these approaches lose some of the advantages of purely primitive-based representations, as the learned features remain globally anchored (e.g. through hashgrids) and do not follow the primitives under motion or deformation. Furthermore, they inherit the limitations of Eulerian encodings, namely poor scaling in large scenes and high-frequency detail modelling. In contrast, we propose that geometric primitives themselves, when appropriately parameterized, can serve as an effective positional encoding mechanism, and enable more efficient signal decoding (i.e. deferred shading).

2.3 Deferred Shading and Texturing in Radiance fields

Alternatives to direct color evaluation in novel view synthesis have been explored to improve expressivity, rendering quality, and efficiency. Early work introduced image-based neural deferred shading conditioned on 3D meshes [58]. Later approaches extended this idea to neural radiance fields by baking appearance features into extracted geometry for real-time rendering [6, 19, 69], removing the necessity to run neural networks during runtime. However, these methods rely on multi-stage pipelines that first optimize a global field, then extract geometry and bake appearance features. In contrast, our method jointly optimizes primitives and neural appearance from scratch, preserving volumetric flexibility and eliminating the need for a separate baking stage, while retaining high quality and speed.

Deferred shading has also been explored in the context of radiance field rendering using neural networks as deferred shaders. In Han et al. [16], global feature grids are queried, with features accumulated and decoded once in a deferred manner. Similarly, point-based neural renderers have recently mixed local neural decoding with deferred shading. Hahlbohm et al. [15] uses point-clouds as query points for a global hash-grid encoded feature field, decoded locally using a shallow neural network into implicit features. These are splatted to the camera plane and finally rendered through a convolutional neural network. However, the necessity of large-bandwidth neural networks to uplift the poor scalability and detail afforded by the spatial encoding made them slow and difficult to train; the former requiring guiding networks with regular NeRF-style rendering to steer the optimization, while the latter required expensive perceptual-loss supervision.

To improve expressivity without global feature grids, several primitive-based methods introduce per-primitive textures [3, 22, 54, 66, 68, 75]. However, highly expressive primitives can complicate optimization and degrade reconstruction quality. Instead, we decouple geometry and appearance by pairing local per-particle features with a shared lightweight neural decoder, in turn enabling the reduction of queries to the neural field to a single deferred pass. This allows particle density to more closely follow geometric complexity rather than being artificially increased to reproduce high-frequency appearance.

3 Preliminaries

We review primitive-based radiance fields, primitive-based volume rendering, and positional encodings, focusing on the aspects most relevant to our method.

Primitive-based Radiance Fields. Primitive-based radiance fields represent complex scenes using an unstructured collection of 2D or 3D geometric primitives, such as anisotropic Gaussians [24], oriented disks [21], or triangles [20]. In Sec. 4, we present our formulation on the example of 3D Gaussian primitives [24, 65] whose spatial response function

$$\rho(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \tag{1}$$

is defined by the particle center $\boldsymbol{\mu} \in \mathbb{R}^3$ and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times3}$. To ensure positive semi-definiteness during optimization, the covariance is parameterized as $\boldsymbol{\Sigma} = \mathbf{RSS}^T\mathbf{R}^T$, where $\mathbf{R} \in SO(3)$ represents rotation and $\mathbf{S} \in \mathbb{R}^{3\times3}$ encodes scaling. In practice, these parameters are stored as a quaternion $\mathbf{q} \in \mathbb{R}^4$ and a scale vector $\mathbf{s} \in \mathbb{R}^3$. Each primitive additionally carries an opacity parameter $\sigma \in \mathbb{R}$ and a view-dependent radiance function $\phi(\mathbf{d})$, commonly represented using spherical harmonics. Note that in this case, $\phi(\mathbf{d})$ depends only on the ray direction and not on the spatial position of the intersection point.

Primitive-based Volume Rendering. Given a camera ray $\mathbf{r}(\tau) = \mathbf{o} + \tau\mathbf{d}$ with origin $\mathbf{o} \in \mathbb{R}^3$ and direction $\mathbf{d} \in \mathbb{R}^3$, the rendered ray color $\mathbf{c} \in \mathbb{R}^3$ is obtained via volume rendering over the primitives P_c intersecting the ray:

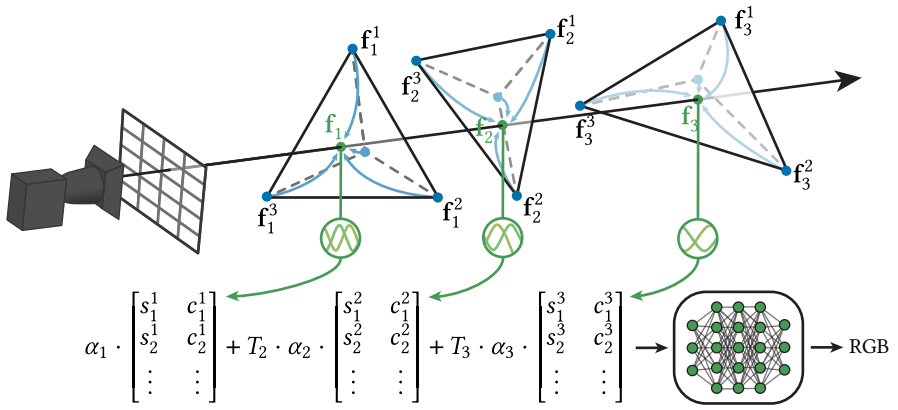


Fig. 2: Neural Harmonic Textures applied to novel-view synthesis. We virtually attach feature vectors $\mathbf{f}_i$ to the vertices of tetrahedra inscribing the Gaussian primitives[3]. Following 3DGUT [65], we evaluate the point along the ray where the projected Gaussian has maximum response. We barycentrically interpolate vertex features at that point, and encode them with sine and cosine functions into different channels. These are then alpha blended along the rest of the ray, until the resulting sum of harmonics is decoded by a shallow MLP in a single image-space pass.

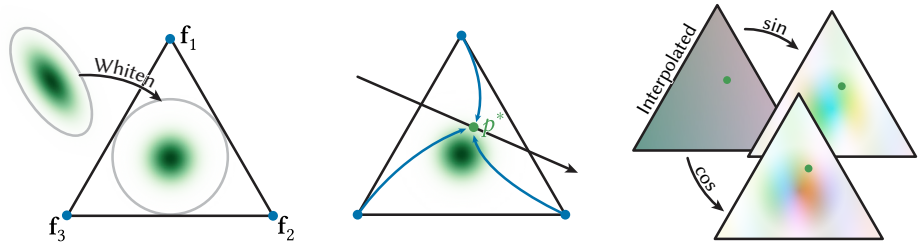
$$\mathbf{c}(\mathbf{o},\mathbf{d}) = \sum_{i \in P_c} \phi(\mathbf{d}) \cdot T_i \cdot \alpha_i, \quad T_i = \prod_{j=1}^{i-1} 1 - \alpha_j, \tag{2}$$

where the opacity α_i is defined as $\alpha_i = \sigma_i \, \rho_i(\mathbf{o} + \tau\mathbf{d})$ and T_i is the transmittance.

Positional Encoding: Neural networks exhibit a natural spectral bias that limits their ability to learn high-frequency functions in low-dimensional domains [57].

Positional encodings mitigate this limitation by mapping low-dimensional spatial and directional coordinates into higher-dimensional representations before feeding them into the network [57]. While early approaches relied on fixed *frequency encodings* [17, 45], modern methods often use *parametric encodings* that store learnable feature vectors in auxiliary spatial data structures, such as grids or trees [44, 55]. Query coordinates retrieve and interpolate these features, effectively shifting much of the representational burden from the neural network to the encoding itself. This design trades increased memory usage for reduced computation: only a small subset of encoding parameters is updated per sample, allowing the neural network to remain compact and efficient while significantly accelerating convergence and maintaining high reconstruction quality.

[3] Gaussian particles are bounded by ellipsoids in world space, which become spheres after whitening. Our virtual bounding tetrahedra are defined in this canonical space.



(a) Latent vectors **(b)** Interpolation **(c)** Harmonic decomposition

Fig. 3: Illustrating our method in 2D. Each primitive is bounded by an ellipsoid in world space, which becomes a sphere in whitened canonical space **(a)**. Considering a virtual bounding tetrahedron in this canonical space, we attach one N-dimensional feature vector $\mathbf{f}^j$ to each vertex. The primitive's contribution is evaluated at the point of maximum response $\mathbf{p}^*$ of the projected Gaussian along the intersecting ray **(b)**. The feature vectors are barycentrically interpolated at $\mathbf{p}^*$ and encoded with sine and cosine periodic functions **(c)**.

4 Method

We present the core components of our approach using 3D Gaussian primitives in the context of novel view synthesis, following the formulation of 3DGUT [65]. We begin by introducing our *primitive-bound feature embedding* in Sec. 4.1. Next, Sec. 4.2 presents *harmonic texturing*, which increases per-primitive expressive power while preserving the locality and explicit form of the representation. Finally, Sec. 4.3 describes a *neural deferred decoding* scheme that enables efficient signal reconstruction in a single image-space pass. Generalization to other primitive types and additional applications will be discussed in Sec. 5.2. Fig. 2 provides a schematic overview of our method.

4.1 Primitive-bound feature embedding

Spatial structures such as triplanes [10], voxel grids [53], and multi-resolution hash grids [44] are commonly used in neural fields to hold latent feature vectors that embed query positions into a higher-dimensional space before decoding the signal. Despite their effectiveness, these representations rely on globally defined regular grids, which limits their scalability to large scenes or high-frequency detail. Moreover, because these structures are fixed in space, they struggle to represent scenes undergoing motion, deformation, or editing.

Instead, we exploit the Lagrangian nature and adaptivity of primitive-based representations by anchoring latent features to a virtual scaffold that encapsulates each primitive. Specifically, consider the isosurface ellipsoid defined by an anisotropic 3D Gaussian (Eq. (1)) with bounded support, which becomes a sphere in canonical space after whitening. We define the virtual tetrahedron bounding this sphere and assign a feature vector $\mathbf{f}^j \in \mathbb{R}^{N_f}$ to each of its four vertices ($j \in \{0..3\}$) (Fig. 3a). For each primitive intersected by a ray, we determine the point $\mathbf{p}^*$ corresponding to the maximum Gaussian response along

the ray [65]. The feature vector at this location, $\mathbf{f}$, is obtained via barycentric interpolation of the vertex features $\mathbf{f}^j$. A detailed derivation is provided in the supplementary material.

Compared to globally anchored feature representations, our formulation moves with the primitives and therefore naturally supports operations such as motion, deformation, or deletion during scene editing.

4.2 Harmonic Texturing

In previous work, feature vectors are typically passed through a lightweight neural network to decode the primitive appearance *prior* to volume rendering, requiring dozens of MLP evaluations per ray to produce the final pixel color [37, 44, 74]. In 3DGS, color decoding overhead (without neural networks) is reduced by approximating view-dependent color emission once per primitive, thereby reducing the complexity from the number of ray–primitive intersections to the number of primitives. However, this approximation assumes that the signal does not vary spatially within each primitive, which makes it unsuitable for our representation.

An alternative strategy is to blend the features along each ray and decode the signal in image space, in the spirit of deferred shading [19]. Inspired by the Fourier transform, where a complex signal is expressed as a sum of harmonic components (i.e., periodic functions of different amplitudes), we instead encode the interpolated features using periodic functions before blending them along the ray. This transforms the features into frequencies and amplitudes, yielding a harmonic decomposition of the signal, which we term *Harmonic Textures* (Fig. 3c).

In this formulation, the interpolation function effectively acts as a frequency modulator: large differences between vertex features within a primitive produce rapidly oscillating, spatially varying textures. Each primitive additionally has a kernel-weighted opacity, which acts as the harmonic amplitude. This behavior is illustrated in the inset Fig. 4 (without the amplitude weighting, to better visualize the harmonics).

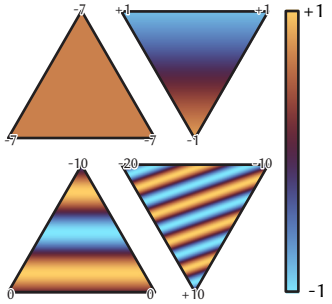


Fig. 4: Harmonic textures.

4.3 Neural Deferred Shading

The rich high-dimensional signal yielded by the volume rendering of *Harmonic Textures* enables a more efficient decoding strategy. We concatenate the accumulated harmonics with the ray direction $\mathbf{d}$ encoded using second-degree spherical harmonics $\mathrm{SH}_2(\mathbf{d}) \in \mathbb{R}^9$ as done in [44], and decode the final pixel color $\mathbf{c}$ using a shallow MLP, without further positional encoding. The resulting rendering equation is

$$\mathbf{c} = \mathrm{MLP}_\theta \left(\sum_{i \in \mathcal{G}} \alpha_i \, T_i \begin{bmatrix} \sin(\mathbf{f}_i) \\ \cos(\mathbf{f}_i) \end{bmatrix}, \; k \cdot \mathrm{SH}_2(\mathbf{d}) \right), \tag{3}$$

where $\mathbf{f}_i = \text{interpolate}\big(\big[\mathbf{f}_i^0, \mathbf{f}_i^1, \mathbf{f}_i^2, \mathbf{f}_i^3\big], \mathbf{p}_i^*\big)$ denotes the primitive feature interpolated at the point $\mathbf{p}_i^*$ of maximum response, $\mathcal{G}$ is the set of primitives intersected by the ray, and α_i and T_i correspond to the primitive opacity and accumulated transmittance defined in Eq. (2).

5 Results

We present our results for radiance-field novel-view synthesis. Additionally, we demonstrate our method on two other reconstruction tasks: semantic scene reconstruction and high-resolution image fitting.

5.1 Radiance Field Reconstruction

Optimization We directly adopt the densification strategy, loss function and regularization terms of 3DGS-MCMC [25]. Specifically, our loss function is a combination of L_1, D-SSIM, and regularizers on primitive opacity and scale:

$$\mathcal{R}_\alpha = \frac{1}{P}\sum_{i=1}^{P}\alpha_i\,, \qquad \mathcal{R}_s = \frac{1}{P}\sum_{i=1}^{P}\|\mathbf{s}_i\|_1\,, \tag{4}$$

where P is the number of primitives and $\mathbf{s}_i$ is the scale vector of primitive i. Our final loss function is:

$$\mathcal{L} = (1-\lambda)\,\mathcal{L}_{\text{L}_1} + \lambda\,\mathcal{L}_{\text{D-SSIM}} + \lambda_\alpha\,\mathcal{R}_\alpha + \lambda_s\,\mathcal{R}_s\,. \tag{5}$$

Similarly to the exponential scheduling applied to primitive positions [24], we use a cosine annealing scheduler for the feature and MLP learning rates.

We also apply an Exponential Moving Average (EMA) filter [44] to the MLP weights θ. This enhances robustness to noise and avoids overfitting to individual frames:

$$\bar{\theta}_t \leftarrow \gamma\,\bar{\theta}_{t-1} + (1-\gamma)\,\theta_t\,, \tag{6}$$

where $\bar{\theta}_t$ denotes the filtered weights at step t and γ is the decay factor.

Finally, we dedicate the last 3000 training iterations to optimize only the feature and MLP weights, disabling all regularization terms and freezing all other parameters. We found that this refinement step slightly improves color fidelity, particularly in large-scale scenes. All hyperparameters and other smaller details will be included in the supplementary.

Implementation details We implement our method in *gsplat* [70], using the 3DGUT [65] formulation, to which we add custom CUDA kernels for the forward and backward logic described in Sec. 4. To reduce register pressure, we use half-precision memory fetches on feature vectors during rasterization, for both forward and backward kernels. Our neural network uses tiny-cuda-nn's [44] JIT-compiled cooperative vector MLPs, trained and evaluated in half precision (FP16). We apply an automatic scaling factor to the loss values in order to improve half-precision training stability [40].

Method	MipNeRF360			Tanks & Temples			Deep Blending		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Instant NGP-Big [44]	25.59	0.695	0.375	21.92	0.740	0.342	24.96	0.815	0.459
Mip-NeRF 360 [1]	27.60	0.788	0.275	22.22	0.754	0.290	29.40	0.899	0.306
ZipNeRF [2]	28.55	0.829	0.218	23.64	0.836	0.179	–	–	–
2DGS [21]	27.22	0.804	0.275	22.85	0.827	0.244	29.56	0.904	0.325
3DGS-MCMC [25]	27.99	0.830	0.229	24.46	0.866	0.174	29.49	0.912	0.306
3DGUT-MCMC [25]	27.82	0.826	0.233	24.20	0.861	0.180	29.87	0.913	0.309
Beta Splatting-MCMC [33]	28.12	0.831	0.238	24.54	0.866	0.196	29.56	0.907	0.316
Spherical Voronoi [9]	28.56	**0.835**	0.228	24.80	0.871	0.172	30.34	0.914	**0.299**
Triangle Splatting [20]	27.00	0.808	0.231	23.05	0.843	0.191	28.92	0.891	0.308
Textured Gaussians [3]	27.35	0.827	–	24.26	0.854	–	28.33	0.891	–
NeST [74]	26.54	0.776	0.260	–	–	–	–	–	–
Radiance Meshes [37]	27.15	0.810	0.274	23.13	0.851	0.200	29.39	0.901	0.362
Neural Harmonic Textures (Ours)	**28.74**	0.834	**0.216**	**25.68**	**0.882**	**0.141**	**30.94**	**0.919**	0.302

Table 1: Comparison on MipNeRF360 [1], Tanks & Temples [27], and Deep Blending [18], on Neural Field-style methods, primitive-based methods, and mixed neural field/primitive-based methods. For the MipNeRF360 datasets, we disable *gsplat*'s default downscaling and instead train and evaluate directly on the provided JPEG-compressed reference images following prior work. We measure the effect in Tab. 9. For our method, we use 64 features per primitive (16 per vertex), with a 128-wide × 3 hidden layers MLP, which results in a roughly similar number of total parameters as previous primitive-based approaches on average. For indoor scenes, which typically span a smaller spatial volume, we use 2M primitives, while for larger outdoor scenes we increase the primitive count to 5M. To regularize training in high-parallax regimes (e.g., indoor datasets), we encode the central camera ray rather than the individual ray direction. Finally, to ensure similar training epochs, we train indoor datasets longer (45k iterations for $\sim$ 290 images) than outdoor (25k for $\sim$ 175 images), following Spherical Voronoi [9]. Unlike Spherical Voronoi, we do not fine tune learning rates per dataset. Note that convergence of an MLP-based appearance model can be slower than SH-based ones.

Results We present the results of our method against a number of previous works using pure neural fields [1], hashgrid-encoded neural fields [2, 44], pure primitive-based methods [20,21,25,33,65] with regular Spherical Harmonics (SH) appearance or more complex appearance models [3,9], and closer to us, methods mixing primitives (e.g. for acceleration) and neural fields [37, 74] (Tab. 1). In this table, we use the originally provided JPEG images for proper comparison with previous works in MipNeRF360; every other result presented in this section, unless otherwise stated, uses *gsplat*'s default downscaling for all methods tested. We measure the performance delta of this choice in the supplementary (Tab. 9). We consistently outperform all previous works in a varied set of real datasets [1, 18,27]. We include a selection of views in Fig. 6. Our method particularly excels at high-frequency detail modelling, capturing specular highlights and reflections with superior fidelity.

In order to completely isolate the impact of our method, we include a controlled experiment in Table 2. We compare between 3DGS-MCMC, 3DGUT-MCMC, and different appearance models for the latter: regular SH, current

Method (w/ MCMC)	MipNeRF360				Tanks & Temples				Deep Blending			
	PSNR↑	SSIM↑	LPIPS↓	FPS↑	PSNR↑	SSIM↑	LPIPS↓	FPS↑	PSNR↑	SSIM↑	LPIPS↓	FPS↑
3DGS + SH	27.94	0.829	0.246	**251**	24.25	0.861	0.188	**294**	29.98	0.912	0.317	**331**
3DGUT + SH	27.93	0.828	0.247	201	23.99	0.859	0.192	245	30.21	0.913	0.318	282
3DGUT + SV	28.15	0.823	0.248	202	24.18	0.861	0.187	242	30.29	0.912	0.320	267
3DGUT + NHT (Ours)	**28.46**	**0.830**	**0.232**	140	**24.79**	**0.875**	**0.169**	226	**30.88**	**0.918**	**0.311**	240

Table 2: Quantitative results of our approach and baselines on the MipNeRF360 [1], Tanks & Temples [27], and Deep Blending [18] datasets, comparing our method against Spherical Voronoi [9] (SV) and regular SH models. We isolate the effect of our approach by implementing all methods in the same framework (*gsplat* [71] with its default downsampling), using the same number of primitives (1M), training for the same number of iterations (30k), allocating the same number of parameters per primitive for appearance (48) and using the same hyperparameters for all scenes. Our method uses a 128×3 hidden MLP. We improve reconstruction quality while still managing real-time performance (measured ona an RTX A6000 Ada GPU).

Table 3: Generality of our approach: NHT applied to different primitive-based representations, evaluated on MipNeRF360 [1]. Triangle Splatting methods use JPEG references as in its source implementation. We would like to note that our results for Triangle Splatting are just a proof-of-concept and were not extensively optimized.

Method	PSNR↑	SSIM↑	LPIPS↓
Triangle Splatting	27.00	**0.808**	0.231
Triangle Splatting + NHT	**27.52**	0.807	**0.227**
2DGS	27.48	0.816	0.263
2DGS + NHT	**28.27**	**0.820**	**0.238**
3DGUT-MCMC	27.93	0.828	0.247
3DGUT-MCMC + NHT	**28.46**	**0.830**	**0.232**

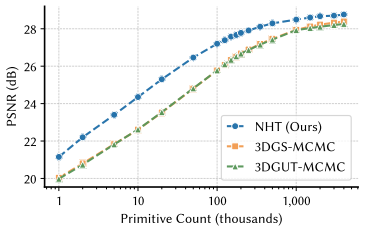


Fig. 5: Our method outperforms 3DGS and 3DGUT at all primitive counts. The improvement is particularly pronounced in the low-primitive regime ($\leq 100k$), with deltas upwards of 2dB of PSNR.

state-of-the-art Spherical Voronoi functions [9] (SV) and ours (NHT). We implement all 4 under the same framework (*gsplat*), under strictly controlled conditions to ensure the only difference between them is the choice of appearance model itself. Table 2 shows that our method consistently outperforms both SH and SV across all benchmarks, without sacrificing real-time performance.

Per-scene breakdowns and more details are provided in the appendix (Appendix Tab. 10 and Tab. 11).

Ablations We ablate the key design decisions of our method on the MipNeRF360 dataset.

Optimization choices. Appendix Tab. 14 isolates the individual contributions of the training strategies detailed in Sec. 5.1: learning rate scheduling, EMA on

Fig. 6: Comparison between our and previous works on radiance field reconstruction on scenes from MipNeRF360 [1], Tanks and Temples [27]. Our method models high frequency detail and view dependent effects to a higher degree than previous works.

the MLP weights, loss scaling, the color-refinement phase, and opacity and scale regularization.

Primitive count. Fig. 5 illustrates how reconstruction quality scales with the number of primitives, ranging from 1K to 4M. Our method improves over previous works across this entire range, excelling particularly in highly compact regimes with low primitive counts. For instance, we achieve performance comparable to previous methods using 1M primitives with only a third of that amount, offering more flexibility in the memory-speed-quality trade-off.

Tab. 16 and Tab. 17 in the appendix explore the impact of varying the per-primitive feature dimension and the MLP architecture, respectively, to illustrate the resulting quality-speed trade-offs.

Feature encoding. Tab. 4 compares different encoding functions applied to the interpolated features.

5.2 Other Applications

Alternative primitive-based methods We demonstrate the generality of our approach by incorporating Neural Harmonic Textures into 2DGS [21] (as implemented in *gsplat*) and Triangle Splatting [20]. Our method is readily adapted

Encoding	PSNR↑	SSIM↑	LPIPS↓
None (identity)	28.21	0.819	0.255
ReLU	28.19	0.819	0.255
Cosine	28.34	0.826	0.234
Cosine & Sine	**28.46**	**0.830**	**0.232**

Table 4: Ablating the choice of feature encoding function (MipNeRF360 dataset).

to these 2D primitives by simply replacing the bounding tetrahedra with virtual triangles (2DGS) or directly anchoring feature vectors to explicit triangle geometry (Triangle Splatting), therefore using three feature vectors per primitive instead of four. The results are shown in Tab. 3, with further details in the Supplementary.

Semantic Field Reconstruction Our method can easily model higher dimensional data as well. By simply expanding the decoding head of our MLP, we can fit signals of arbitrary dimension. As the rest of the MLP layers remain small, and the size of feature vectors remains the same, the model is forced to exploit potential correlations between dimensions.

A particularly challenging case is semantic scene reconstruction, where fields are upwards of 512-dimensional. 3DGS-based methods relying on high-dimensional explicit features generally struggle with such scale, even when paired with feature upsampling networks.

We thus test our method on joint novel-view synthesis of RGB radiance and LSEG [29] 512-wide semantic features. Due to the limited resolution of LSEG feature maps, we render and predict at the RGB resolution, but bilinearly downsample the spatial dimension of the resulting feature maps in order to supervise on ground truth features. We compare against Feature 3DGS [76], a hybrid method relying on 128-wide semantic feature vector per primitive plus a CNN to jointly downsample the spatial dimension and upsample the feature dimension of the resulting rasterized image up to the full 512 dimensions, and another set of 48 SH basis weights for regular RGB color. Results for the aggregated MipNeRF360 dataset can be seen in Tab. 5. We afford Feature-3DGS twice our training time budget, up to 7K iterations, slightly more than in the original paper (5K), to accommodate for the larger-scale datasets. We further ablate the per-primitive feature dimensionality for this task in Appendix Tab. 12. Our results show substantial improvement over Feature 3DGS, and the capability to easily scale upwards or downwards in performance or memory, in a completely detached manner from the nature of the data being encoded. It is particularly interesting to note how small the drop in RGB quality is, which points to either high correlation among feature vectors or excessive free bandwidth in our RGB experiments. Our features can then be used in downstream tasks such as scene editing or semantic segmentation.

Method	$\text{RGB}_{\text{PSNR}}\uparrow$	$\text{RGB}_{\text{SSIM}}\uparrow$	$\text{RGB}_{\text{LPIPS}}\downarrow$	$\text{LSEG}_{\text{P}}\uparrow$	$\text{LSEG}_{\text{C}}\uparrow$	FPS↑
Feature 3DGS [76]	26.27	0.785	0.262	43.68	0.985	3.02
Neural Harmonic Textures (Ours)	**28.16**	**0.824**	**0.228**	**46.90**	**0.993**	**28.11**

Table 5: Joint RGB and LSEG semantic feature reconstruction on MipNeRF360 [1], measured on an RTX A6000 Ada. LSEG_{P} and LSEG_{C} denote PSNR and cosine similarity of the 512-dim semantic features, respectively. We use 80 features per primitive and a 128×3 MLP, while Feature 3DGS uses 176 features per primitive, 512×128 CNN and 1.5× the number of primitives we use, resulting in an approximately 3× higher average memory footprint.

2D Image Reconstruction Finally, we show results on 2D image fitting. To adapt our method to this application, we make two key changes with respect to the primitive-based approaches. First, we employ a connected 2D triangle mesh topology. This mesh is fully opaque: there is no alpha blending, no Gaussian or window kernel, simply interpolation of local features for a given pixel. Second, we change our barycentric interpolation to Clough-Tocher cubic interpolation. This makes interpolated features continuous over the triangle edges, which removes high frequency artifacts. The rest of the architecture is the same as described in Sec. 4, and we include further details on the optimization strategy in the Supplementary.

We evaluate image reconstruction performance on a new curated dataset of high-resolution (45.7 MP) 14-bit HDR RAW images, and compare it against Instant NGP [44] and the JPEG-XL encoder [23], which supports high-dynamic range. We measure reconstruction quality at two compression ratios (10× and 100×), computing PSNR in μ-law transformed space (PSNR_{μ}), standard in HDR imaging pipelines. We also report PSNR, SSIM, and LPIPS on tonemapped images (subscript "tm").

Results are shown in Tab. 6. Our method achieves comparable pixel-error quality to Instant NGP in both HDR and linear spaces, while substantially reducing perceptual error: at 100× compression, NHT achieves an LPIPS of 0.048 compared to 0.078 for Instant NGP, a 38% relative improvement. JPEG-XL however still comes out on top, particularly in perceptual metrics, which is to be expected given its human-perception-based inductive biases. We can further compress our representation by a factor of 3 without noticeable loss in quality by aggressively reducing floating point precision of the implicit features, among other strategies. We include more details on this in the supplementary.

6 Discussion

Limitations and future work Our method focuses on increasing per-primitive expressivity, which has inherent downsides. In particular, our method tends to overfit more to individual views in scenarios with very sparse supervision, leading to lower novel view synthesis performance. Furthermore, while our method consistently achieves 140+ frames per second at inference time, the additional neural decoder results in slightly slower rendering than pure 3DGS methods(Tab. 2).

Ratio	Method	$\text{PSNR}_\mu\uparrow$	$\text{PSNR}_{\text{tm}}\uparrow$	$\text{SSIM}_{\text{tm}}\uparrow$	$\text{LPIPS}_{\text{tm}}\downarrow$
10×	JPEG-XL [23]	**45.31**	**44.66**	**0.994**	**0.001**
	Instant NGP [44]	37.63	38.91	0.949	0.061
	NHT (Ours)	37.16	39.87	0.963	0.023
100×	JPEG-XL [23]	**38.53**	35.54	**0.971**	**0.017**
	Instant NGP [44]	35.17	36.39	0.923	0.078
	NHT (Ours)	35.06	**36.43**	0.927	0.048

Table 6: High-resolution (45.7 MP) 14-bit HDR RAW image fitting, averaged over 15 images. We report PSNR in μ-law transformed space (PSNR_μ), common in HDR pipelines. PSNR, SSIM, and LPIPS are also computed on tonemapped images (subscript "tm"). All methods use the same memory budget per compression ratio.

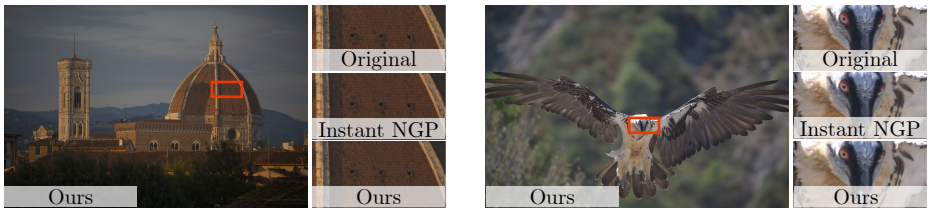


Fig. 7: Comparison of our work vs Instant NGP on a 100× compression task. Original images are 45.7MP 14-bit HDR RAW files. We achieve substantially superior perceptual quality at equal compression and similar training times.

In future work, we would like to investigate the feasibility of extracting automatic Level of Detail from a finer-grained harmonic decomposition. Given the generality of Neural Harmonic Textures, applications beyond novel view synthesis such as radiance caching, neural physically-based rendering and geometric reconstruction are also of great interest. Finally, our work opens the possibility of removing kernel functions entirely, as we can still propagate derivatives to explicit geometry without the spatially decaying opacity function, potentially enabling drastic performance gains.

Conclusion We presented Neural Harmonic Textures, which combine particle-anchored feature vectors, harmonic activations, and a single image-space decoding pass to achieve state-of-the-art results in novel view synthesis. Our method consistently outperforms baselines across different particle counts, trains rapidly, and has high inference performance.

Moreover, it is compatible with existing deferred-shading rendering pipelines and supports a wide range of applications, from lifting semantic fields to high-resolution image reconstruction and beyond.

Acknowledgements We would like to thank Ruilong Li and Martin Bisson for helpful discussions. Jorge Condor and Piotr Didyk acknowledge funding from

the Swiss National Science Foundation (SNSF, Grant 200502) and an academic gift from Meta.

References

1. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2022) 10, 11, 12, 14, 22, 24, 30
2. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Zip-NeRF: Anti-aliased grid-based neural radiance fields. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2023) 10
3. Chao, B., Tseng, H.Y., Porzi, L., Gao, C., Li, T., Li, Q., Saraf, A., Huang, J.B., Kopf, J., Wetzstein, G., Kim, C.: Textured gaussians for enhanced 3d scene appearance modeling. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2025) 5, 10
4. Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H.: Tensorf: Tensorial radiance fields. In: European conference on computer vision. pp. 333–350. Springer (2022) 3
5. Chen, Y., Chen, Z., Zhang, C., Wang, F., Yang, X., Wang, Y., Cai, Z., Yang, L., Liu, H., Lin, G.: Gaussianeditor: Swift and controllable 3d editing with gaussian splatting. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 21476–21485 (2024) 2
6. Chen, Z., Funkhouser, T., Hedman, P., Tagliasacchi, A.: MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In: The Conference on Computer Vision and Pattern Recognition (CVPR) (2023) 4
7. Condor, J., Hermann, N., Yurtsever, M.A., Didyk, P.: Gabor fields: Orientation-selective level-of-detail for volume rendering (2026), `https://arxiv.org/abs/2602.05081` 4
8. Condor, J., Speierer, S., Bode, L., Bozic, A., Green, S., Didyk, P., Jarabo, A.: Don't splat your gaussians: Volumetric ray-traced primitives for modeling and rendering scattering and emissive media. ACM Transactions on Graphics **44**(1) (2025) 4
9. Di Sario, F., Rebain, D., Verbin, D., Grangetto, M., Tagliasacchi, A.: Spherical voronoi: Directional appearance as a differentiable partition of the sphere. arXiv preprint arXiv:2512.14180 (2025) 2, 10, 11
10. Duckworth, D., Hedman, P., Reiser, C., Zhizhin, P., Thibert, J.F., Lučić, M., Szeliski, R., Barron, J.T.: Smerf: Streamable memory efficient radiance fields for real-time large-scene exploration (2023) 7
11. Fridovich-Keil, S., Meanti, G., Warburg, F.R., Recht, B., Kanazawa, A.: K-planes: Explicit radiance fields in space, time, and appearance. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12479–12488 (2023) 3
12. Gadirov, H., Wu, Q., Bauer, D., Ma, K.L., Roerdink, J.B., Frey, S.: Hyperflint: Hypernetwork-based flow estimation and temporal interpolation for scientific ensemble visualization. Computer Graphics Forum **44**(3), e70134 (2025). `https://doi.org/https://doi.org/10.1111/cgf.70134`, `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.70134` 3
13. Govindarajan, S., Rebain, D., Yi, K.M., Tagliasacchi, A.: Radiant foam: Real-time differentiable ray tracing. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4135–4145 (2025) 4

14. Govindarajan, S., Sambugaro, Z., Shabanov, A., Takikawa, T., Rebain, D., Sun, W., Conci, N., Yi, K.M., Tagliasacchi, A.: Lagrangian hashing for compressed neural field representations. In: European Conference on Computer Vision. pp. 183–199. Springer (2024) 3
15. Hahlbohm, F., Franke, L., Kappel, M., Castillo, S., Eisemann, M., Stamminger, M., Magnor, M.: INPC: Implicit Neural Point Clouds for Radiance Field Rendering . In: 2025 International Conference on 3D Vision (3DV). pp. 168–178. IEEE Computer Society, Los Alamitos, CA, USA (Mar 2025). https://doi.org/10.1109/3DV66043.2025.00021, https://doi.ieeecomputersociety.org/10.1109/3DV66043.2025.00021 5
16. Han, K., Xiang, W., Yu, L.: Volume feature rendering for fast neural radiance field reconstruction. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems. vol. 36, pp. 65416–65427. Curran Associates, Inc. (2023), https://proceedings.neurips.cc/paper_files/paper/2023/file/ce182e31662883d4decc84a0255335b6-Paper-Conference.pdf 5
17. Harris, D., Harris, S.L.: Digital design and computer architecture. Morgan Kaufmann (2010) 6
18. Hedman, P., Philip, J., Price, T., Frahm, J.M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering. ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia) **37**(6), 257:1–257:15 (2018) 10, 11, 22, 24, 30
19. Hedman, P., Srinivasan, P.P., Mildenhall, B., Barron, J.T., Debevec, P.: Baking neural radiance fields for real-time view synthesis. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2021) 4, 8
20. Held, J., Vandeghen, R., Deliege, A., Hamdi, A., Rebain, D., Giancola, S., Cioppa, A., Vedaldi, A., Ghanem, B., Tagliasacchi, A., et al.: Triangle splatting for real-time radiance field rendering. In: Thirteenth International Conference on 3D Vision (3DV) (2025) 2, 4, 5, 10, 12
21. Huang, B., Yu, Z., Chen, A., Geiger, A., Gao, S.: 2D Gaussian splatting for geometrically accurate radiance fields. In: ACM SIGGRAPH 2024 Conference Papers (2024). https://doi.org/10.1145/3641519.3657428 4, 5, 10, 12
22. Huang, Z., Gong, M.: Textured-gs: Gaussian splatting with spatially defined color and opacity. arXiv preprint arXiv:2407.09733 (2024) 5
23. Joint Photographic Experts Group: JPEG XL image coding system. https://jpeg.org/jpegxl/ (2024), accessed: 2024-05-24 14, 15, 33
24. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3D gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics (Proceedings of SIGGRAPH) **42**(4) (2023) 2, 4, 5, 9
25. Kheradmand, S., Rebain, D., Sharma, G., Sun, W., Tseng, Y.C., Isack, H., Kar, A., Tagliasacchi, A., Yi, K.M.: 3D gaussian splatting as markov chain monte carlo. In: Advances in Neural Information Processing Systems (NeurIPS) (2024) 9, 10
26. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2017) 28
27. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction. ACM Transactions on Graphics (Proceedings of SIGGRAPH) **36**(4) (2017) 10, 11, 12, 22, 24, 30
28. Kulhanek, J., Rakotosaona, M.J., Manhardt, F., Tsalicoglou, C., Niemeyer, M., Sattler, T., Peng, S., Tombari, F.: Lodge: Level-of-detail large-scale gaussian splatting with efficient rendering. arXiv preprint arXiv:2505.23158 (2025) 4
29. Li, B., Weinberger, K.Q., Belongie, S., Koltun, V., Ranftl, R.: Language-driven semantic segmentation. arXiv preprint arXiv:2201.03546 (2022) 13

30. Li, Z., Müller, T., Evans, A., Taylor, R.H., Unberath, M., Liu, M.Y., Lin, C.H.: Neuralangelo: High-fidelity neural surface reconstruction. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 8456–8465 (2023) 3
31. Liang, H., Ren, J., Mirzaei, A., Torralba, A., Liu, Z., Gilitschenski, I., Fidler, S., Oztireli, C., Ling, H., Gojcic, Z., Huang, J.: Feed-forward bullet-time reconstruction of dynamic scenes from monocular videos. arXiv preprint arXiv:2412.03526 (2024) 2
32. Liu, L., Gu, J., Lin, K.Z., Chua, T.S., Theobalt, C.: Neural sparse voxel fields. In: Advances in Neural Information Processing Systems (NeurIPS) (2020) 3
33. Liu, R., Sun, D., Chen, M., Wang, Y., Feng, A.: Deformable beta splatting. In: Proceedings of SIGGRAPH Conference Papers (2025) 2, 4, 10
34. Lombardi, S., Simon, T., Saragih, J., Schwartz, G., Lehrmann, A., Sheikh, Y.: Neural volumes: Learning dynamic renderable volumes from images. ACM Transactions on Graphics (Proceedings of SIGGRAPH) **38**(4) (2019) 3
35. Lombardi, S., Simon, T., Schwartz, G., Zollhoefer, M., Sheikh, Y., Saragih, J.: Mixture of volumetric primitives for efficient neural rendering. ACM Transactions on Graphics (Proceedings of SIGGRAPH) **40**(4) (2021) 3
36. Luo, A., Du, Y., Tarr, M., Tenenbaum, J., Torralba, A., Gan, C.: Learning neural acoustic fields. Advances in Neural Information Processing Systems **35**, 3165–3177 (2022) 3
37. Mai, A., Hedstrom, T., Kopanas, G., Kontkanen, J., Kuester, F., Barron, J.T.: Radiance meshes for volumetric reconstruction. arXiv preprint arXiv:2512.04076 (2025) 2, 4, 8, 10
38. Mai, A., Hedstrom, T., Kopanas, G., Kontkanen, J., Kuester, F., Barron, J.T.: Radiance meshes for volumetric reconstruction (2025), `https://arxiv.org/abs/2512.04076` 4
39. Martel, J.N.P., Lindell, D.B., Lin, C.Z., Chan, E.R., Monteiro, M., Wetzstein, G.: ACORN adaptive coordinate networks for neural scene representation. ACM Transactions on Graphics (Proceedings of SIGGRAPH) **40**(4) (2021). `https://doi.org/10.1145/3450626.3459785`, `https://doi.org/10.1145/3450626.3459785` 3
40. Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al.: Mixed precision training. arXiv preprint arXiv:1710.03740 (2017) 9
41. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing scenes as neural radiance fields for view synthesis. In: Proceedings of ECCV (2020) 2, 3, 4
42. Moenne-Loccoz, N., Mirzaei, A., Perel, O., de Lutio, R., Esturo, J.M., State, G., Fidler, S., Sharp, N., Gojcic, Z.: 3d gaussian ray tracing: Fast tracing of particle scenes. ACM Transactions on Graphics and SIGGRAPH Asia (2024) 4
43. Mujkanovic, F., Nsampi, N.E., Theobalt, C., Seidel, H.P., Leimkühler, T.: Neural gaussian scale-space fields. ACM Trans. Graph. **43**(4) (Jul 2024). `https://doi.org/10.1145/3658163`, `https://doi.org/10.1145/3658163` 3
44. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Trans. Graph. **41**(4) (2022) 2, 3, 4, 6, 7, 8, 9, 10, 14, 15, 28, 33
45. Müller, T., McWilliams, B., Rousselle, F., Gross, M., Novák, J.: Neural importance sampling. ACM Transactions on Graphics (TOG) **38**(5), 1–19 (2019) 6

46. Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 165–174 (2019) 3
47. Pumarola, A., Corona, E., Pons-Moll, G., Moreno-Noguer, F.: D-nerf: Neural radiance fields for dynamic scenes. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10318–10327 (2021) 3
48. Reiser, C., Peng, S., Liao, Y., Geiger, A.: KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In: Proceedings of ICCV (2021) 3
49. Saragadam, V., LeJeune, D., Tan, J., Balakrishnan, G., Veeraraghavan, A., Baraniuk, R.G.: Wire: Wavelet implicit neural representations. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 18507–18516 (2023) 3
50. Sitzmann, V., Martel, J., Bergman, A., Lindell, D., Wetzstein, G.: Implicit neural representations with periodic activation functions. Advances in neural information processing systems **33**, 7462–7473 (2020) 2
51. Sitzmann, V., Martel, J.N., Bergman, A.W., Lindell, D.B., Wetzstein, G.: Implicit neural representations with periodic activation functions. In: Proc. NeurIPS (2020) 3, 4
52. Su, R., Dong, H., Jin, H., Chen, Y., Wang, G., Li, S.: Vertex features for neural global illumination. In: Proceedings of the SIGGRAPH Asia 2025 Conference Papers. pp. 1–11 (2025) 26
53. Sun, C., Sun, M., Chen, H.: Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In: CVPR (2022) 7
54. Svitov, D., Morerio, P., Agapito, L., Del Bue, A.: Billboard splatting (bbsplat): Learnable textured primitives for novel view synthesis. arXiv preprint arXiv:2411.08508 (2024) 5
55. Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M., Fidler, S.: Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11358–11367 (2021) 3, 6
56. Takikawa, T., Müller, T., Nimier-David, M., Evans, A., Fidler, S., Jacobson, A., Keller, A.: Compact neural graphics primitives with learned hash probing. In: SIGGRAPH Asia 2023 Conference Papers. SA '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3610548.3618167, https://doi.org/10.1145/3610548.3618167 2
57. Tancik, M., Srinivasan, P.P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J.T., Ng, R.: Fourier features let networks learn high frequency functions in low dimensional domains. NeurIPS (2020) 2, 3, 4, 6
58. Thies, J., Zollhöfer, M., Nießner, M.: Deferred neural rendering: Image synthesis using neural textures. Acm Transactions on Graphics (Proceedings of SIGGRAPH) **38**(4) (2019) 3, 4
59. Wang, J., Chen, M., Karaev, N., Vedaldi, A., Rupprecht, C., Novotny, D.: Vggt: Visual geometry grounded transformer. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 5294–5306 (2025) 2
60. Wang, P., Liu, L., Liu, Y., Theobalt, C., Komura, T., Wang, W.: Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. arXiv preprint arXiv:2106.10689 (2021) 3
61. Wang, Y., Zhou, J., Zhu, H., Chang, W., Zhou, Y., Li, Z., Chen, J., Pang, J., Shen, C., He, T.: π^3: Permutation-equivariant visual geometry learning (2025), https://arxiv.org/abs/2507.13347 2

62. Wu, G., Yi, T., Fang, J., Xie, L., Zhang, X., Wei, W., Liu, W., Tian, Q., Wang, X.: 4D gaussian splatting for real-time dynamic scene rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2024) 2, 4
63. Wu, Q., Bauer, D., Doyle, M.J., Ma, K.L.: Interactive volume visualization via multi-resolution hash encoding based neural representation. IEEE Transactions on Visualization and Computer Graphics pp. 1–14 (2023). https://doi.org/10.1109/TVCG.2023.3293121 3
64. Wu, Q., Insley, J.A., Mateevitsi, V.A., Rizzi, S., Papka, M.E., Ma, K.L.: Distributed neural representation for reactive in situ visualization. IEEE Transactions on Visualization and Computer Graphics **31**(9), 5199–5214 (2025). https://doi.org/10.1109/TVCG.2024.3432710 3
65. Wu, Q., Martinez Esturo, J., Mirzaei, A., Moenne-Loccoz, N., Gojcic, Z.: 3dgut: Enabling distorted cameras and secondary rays in gaussian splatting. Conference on Computer Vision and Pattern Recognition (CVPR) (2025) 4, 5, 6, 7, 8, 9, 10
66. Wurster, S., Zhang, R., Zheng, C.: Gabor splatting for high-quality gigapixel image representations. In: ACM SIGGRAPH 2024 Posters. SIGGRAPH '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3641234.3671081, https://doi.org/10.1145/3641234.3671081 5
67. Xie, Y., Takikawa, T., Saito, S., Litany, O., Yan, S., Khan, N., Tombari, F., Tompkin, J., Sitzmann, V., Sridhar, S.: Neural fields in visual computing and beyond. In: Computer graphics forum. vol. 41, pp. 641–676. Wiley Online Library (2022) 3
68. Xu, T.X., Hu, W., Lai, Y.K., Shan, Y., Zhang, S.H.: Texture-gs: Disentangling the geometry and texture for 3d gaussian splatting editing. In: European Conference on Computer Vision. pp. 37–53. Springer (2024) 5
69. Yariv, L., Hedman, P., Reiser, C., Verbin, D., Srinivasan, P.P., Szeliski, R., Barron, J.T., Mildenhall, B.: BakedSDF: Meshing neural SDFs for real-time view synthesis. In: ACM SIGGRAPH 2023 Conference Proceedings (2023) 4
70. Ye, V., Li, R., Kerr, J., Turkulainen, M., Yi, B., Pan, Z., Seiskari, O., Ye, J., Hu, J., Tancik, M., Kanazawa, A.: gsplat: An open-source library for gaussian splatting. Journal of Machine Learning Research **26**(34), 1–17 (2025) 9, 32
71. Ye, V., Li, R., Kerr, J., Turkulainen, M., Yi, B., Pan, Z., Seiskari, O., Ye, J., Hu, J., Tancik, M., Kanazawa, A.: gsplat: An open-source library for gaussian splatting. Journal of Machine Learning Research **26**(34), 1–17 (2025) 11
72. Zhang, K., Bi, S., Tan, H., Xiangli, Y., Zhao, N., Sunkavalli, K., Xu, Z.: Gs-lrm: Large reconstruction model for 3d gaussian splatting. European Conference on Computer Vision (2024) 2
73. Zhang, X., Chen, A., Xiong, J., Dai, P., Shen, Y., Xu, W.: Neural shell texture splatting: More details and fewer primitives. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 25229–25238 (2025) 2, 3, 4
74. Zhang, X., Chen, A., Xiong, J., Dai, P., Shen, Y., Xu, W.: Neural shell texture splatting: More details and fewer primitives. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2025) 8, 10
75. Zhou, J., Huang, Y., Dai, W., Zou, J., Zheng, Z., Kan, N., Li, C., Xiong, H.: 3dgabsplat: 3d gabor splatting for frequency-adaptive radiance field rendering. arXiv preprint arXiv:2508.05343 (2025) 5
76. Zhou, S., Chang, H., Jiang, S., Fan, Z., Zhu, Z., Xu, D., Chari, P., You, S., Wang, Z., Kadambi, A.: Feature 3dgs: Supercharging 3d gaussian splatting to enable distilled feature fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 21676–21685 (2024) 13, 14, 23

77. Zhou, X., Nguyen, B.H., Magne, L., Golyanik, V., Leimkühler, T., Theobalt, C.: Splat the net: Radiance fields with splattable neural primitives. arXiv preprint arXiv:2510.08491 (2025) 3

Method	MipNeRF360			Tanks & Temples			Deep Blending		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
3DGS-MCMC + SH	28.21	**0.841**	0.214	24.46	0.866	0.174	29.49	0.912	0.306
3DGUT-MCMC + SH	28.08	0.837	0.218	24.20	0.861	0.180	29.87	0.913	0.309
3DGUT-MCMC + NHT (Ours, 48F)	28.65	0.834	0.212	25.41	0.878	0.156	30.67	**0.918**	0.302
3DGUT-MCMC + NHT (Ours, 64F)	**28.68**	0.832	**0.211**	**25.64**	**0.879**	**0.154**	**30.69**	**0.918**	**0.301**

Table 7: Standardized comparison on MipNeRF360 [1], Tanks & Temples [27], and Deep Blending [18]: 3DGS-MCMC, 3DGUT-MCMC, and ours with 48 and 64 features per primitive (48F, 64F). All methods use the same primitive count (matching the high-quality preset in 3DGS-MCMC, which replicates the number of primitives in each scene from original 3DGS), 30k iterations, and the same hyperparameters across all scenes. Extended version of Tab. 2 of the main paper. While our work particularly excels at extracting higher quality from less primitives, we still outperform previous works at high primitive counts across the board.

Supplementary Material

A Additional Results

In Tab. 10 we provide a detailed breakdown of Tab. 2 from the main paper, with per-scene results on MipNeRF360, Tanks and Temples and Deep Blending. For this comparison, we isolate completely the effect of our method by ensuring an even setup for all methods. All four approaches are implemented under the same framework (*gsplat*), trained for 30k iterations, using 1M primitives for all scenes, sharing training parameters among all scenes, and limiting to 48 features per primitive reserved for appearance. All results presented in this paper compute the LPIPs metric using the VGG model with normalized inputs to the range of $[-1, 1]$. Furthermore, in Tab. 7 we provide another standardized comparison on all three benchmarks, only this time we match the same primitive count in each scene as the high-quality reported baselines in their respective papers, training for the same amount of iterations. We include our method with both 48 and 64 features per primitive (48F, 64F).

Tab. 11 details the per-scene results of main Tab. 1, which uses our split indoor/outdoor training configuration. This is our higher quality setup, which adapts to the different scene scale and dataset sizes of outdoor and indoor datasets.

Effect of reference image format. As noted in Tab. 1, MipNeRF360 images are distributed as pre-downscaled, re-compressed JPEG files. The default *gsplat* pipeline downscales and converts these to lossless PNGs before training and evaluation, which removes JPEG compression artifacts from the ground truth. In Tab. 9 we compare NHT results under both protocols. Training and evaluating directly on the original JPEG references yields slightly lower PSNR (-0.28 dB), SSIM (-0.011), and higher LPIPS ($+0.014$) on average, as JPEG compression artifacts in the ground truth lower the effective quality ceiling.

Scene	3DGS-MCMC + SH			3DGUT-MCMC + SH			NHT (Ours, 48F)			NHT (Ours, 64F)		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
MipNeRF360 – Outdoor												
bicycle	**25.87**	**0.801**	**0.183**	25.69	0.790	0.193	25.53	0.779	0.202	25.62	0.776	0.202
garden	28.11	**0.880**	0.107	27.84	0.874	0.111	**28.33**	0.878	**0.102**	28.23	0.876	0.103
stump	**27.32**	**0.808**	**0.203**	27.15	0.802	0.215	27.06	0.788	0.215	27.02	0.786	0.215
treehill	**23.29**	**0.670**	0.310	23.13	0.664	0.313	21.96	0.639	0.311	22.27	0.640	**0.308**
flowers	**22.25**	**0.652**	**0.312**	**22.25**	0.648	0.318	21.90	0.637	0.315	21.77	0.631	**0.312**
MipNeRF360 – Indoor												
room	32.56	0.939	0.240	32.43	0.939	0.241	**33.82**	0.945	0.229	33.75	**0.946**	**0.227**
counter	29.53	0.925	0.219	29.55	0.925	0.220	30.80	**0.931**	0.205	**30.92**	**0.931**	**0.203**
kitchen	32.10	0.938	0.136	31.96	0.938	0.136	33.43	**0.944**	0.127	**33.44**	**0.944**	**0.124**
bonsai	32.84	0.955	0.216	32.76	0.955	0.215	34.99	**0.963**	0.202	**35.07**	0.962	**0.201**
Tanks & Temples												
train	22.48	0.832	0.221	22.19	0.826	0.229	23.88	0.856	0.198	**24.33**	**0.858**	**0.194**
truck	26.43	0.899	0.126	26.21	0.896	0.131	26.94	**0.901**	0.115	**26.96**	**0.901**	**0.114**
Deep Blending												
drjohnson	29.25	0.910	0.310	29.13	0.910	0.315	30.06	**0.916**	0.308	**30.09**	**0.916**	**0.306**
playroom	29.72	0.913	0.301	30.60	0.916	0.302	**31.28**	**0.920**	0.297	**31.28**	**0.920**	**0.296**
M360 Outdoor Avg	**25.37**	**0.762**	**0.223**	25.21	0.756	0.230	24.96	0.744	0.229	24.98	0.742	0.228
M360 Indoor Avg	31.76	0.939	0.203	31.67	0.939	0.203	33.26	**0.946**	0.191	**33.30**	**0.946**	**0.189**
M360 Total Avg	28.21	**0.841**	0.214	28.08	0.837	0.218	28.65	0.834	0.212	**28.68**	0.832	**0.211**
T&T Avg	24.46	0.866	0.174	24.20	0.861	0.180	25.41	0.879	0.157	**25.65**	**0.879**	**0.154**
DB Avg	29.49	0.912	0.306	29.87	0.913	0.309	30.67	**0.918**	0.302	**30.69**	**0.918**	**0.301**

Table 8: Per-scene results for the standardized comparison (Tab. 7). All methods use the same primitive count (matching the baseline default), 30k iterations, and the same hyperparameters across all scenes.

A.1 Semantic Reconstruction

Semantic Reconstruction Details For our semantic reconstruction task, following a similar setup to Feature 3DGS [76], we train a joint RGB and LSEG semantic feature field. Joint RGB and LSEG training uses the same approach, densification, and regularization as standard radiance-field training, with the differences being in the addition of semantic supervision losses and an expanded decoding head in our MLP. The deferred MLP is extended to predict both RGB (3 channels) and LSEG features (512 dimensions) via addition of an extra Pytorch linear layer. Rendering speed becomes bottlenecked by this linear layer, and more efficient fused alternatives or upsampling strategies could be explored in the future—we consider this a proof-of-concept implementation.

The total loss adds an LSEG term to the usual L_1, D-SSIM, opacity and scale regularizers terms:

$$\mathcal{L}_{\text{LSEG}} = \lambda_{\text{LSEG}}\, \mathcal{L}_{\text{L}_1}\left(\hat{f}, f^*\right) + \lambda_{\cos}\left(1 - \cos(\hat{f}, f^*)\right), \tag{7}$$

where $\hat{f}$ and f^* are predicted and ground-truth LSEG feature maps, and *cos* their cosine similarity. We compute these losses at the native LSEG resolution: the model renders at RGB resolution (high); then, we bilinearly downsample the predicted feature map to match the (lower-resolution) LSEG targets. Tab. 13 lists the LSEG-specific hyperparameters used in our final training. We largely kept

References	PSNR↑	SSIM↑	LPIPS↓
Downscaled PNGs	**29.022**	**0.8447**	**0.2026**
JPEG (original)	28.738	0.8341	0.2164
Δ	−0.284	−0.0106	+0.0138

Table 9: Effect of reference image format on MipNeRF360 evaluation for NHT (Ours). *Downscaled PNGs* uses *gsplat*'s default strategy, which downscales and converts the provided JPEG references to lossless PNGs before training and evaluation. *JPEG (original)* trains and evaluates directly on the provided JPEG-compressed references without any preprocessing. JPEG compression artifacts in the ground truth lower the effective quality ceiling, slightly degrading all three metrics.

	3DGS-MCMC			3DGUT-MCMC + SH			3DGUT-MCMC + SV			3DGUT-MCMC + NHT (Ours)		
Scene	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
MipNeRF360 – Outdoor												
bicycle	**25.61**	**0.774**	0.246	25.51	0.771	0.250	25.21	0.758	0.255	25.43	0.771	**0.243**
garden	27.29	0.855	0.155	27.21	0.854	0.156	27.48	0.856	0.156	**28.04**	**0.872**	**0.122**
stump	**26.93**	0.788	0.250	**26.93**	**0.789**	0.251	25.95	0.757	0.270	26.58	0.772	**0.249**
treehill	**23.45**	**0.665**	0.366	23.30	0.663	0.368	23.16	0.653	0.371	23.13	0.660	**0.340**
flowers	22.04	**0.628**	0.362	**22.07**	0.627	0.363	21.76	0.616	0.367	21.60	0.615	**0.351**
MipNeRF360 – Indoor												
room	32.33	0.938	0.245	32.36	0.937	0.246	32.67	0.939	0.243	**33.03**	**0.943**	**0.234**
counter	29.48	0.924	0.223	29.47	0.923	0.224	30.69	**0.932**	0.213	**30.83**	0.931	**0.208**
kitchen	31.67	0.935	0.144	31.72	0.935	0.144	32.38	0.938	0.140	**32.98**	**0.942**	**0.132**
bonsai	32.69	0.953	0.220	32.76	0.953	0.220	34.07	0.959	0.215	**34.55**	**0.961**	**0.208**
Tanks & Temples												
train	22.50	0.832	0.224	22.10	0.827	0.231	22.32	0.831	0.223	**23.11**	**0.851**	**0.202**
truck	26.00	0.891	0.151	25.88	0.890	0.153	26.04	0.891	0.151	**26.48**	**0.898**	**0.135**
Deep Blending												
drjohnson	29.50	0.909	0.320	29.60	0.909	0.321	29.69	0.909	0.321	**30.24**	**0.915**	**0.317**
playroom	30.45	0.915	0.313	30.82	0.916	0.315	30.89	0.915	0.318	**31.52**	**0.920**	**0.306**
M360 Outdoor Avg	**25.06**	**0.742**	0.276	25.00	0.741	0.278	24.71	0.728	0.284	24.96	0.738	**0.261**
M360 Indoor Avg	31.54	0.938	0.208	31.58	0.937	0.209	32.45	0.942	0.203	**32.85**	**0.944**	**0.196**
M360 Total Avg	27.94	0.829	0.246	27.93	0.828	0.247	28.15	0.823	0.248	**28.46**	**0.830**	**0.232**
T&T Avg	24.25	0.861	0.188	23.99	0.859	0.192	24.18	0.861	0.187	**24.79**	**0.875**	**0.169**
DB Avg	29.98	0.912	0.317	30.21	0.913	0.318	30.29	0.912	0.320	**30.88**	**0.918**	**0.311**

Table 10: Per-scene results on MipNeRF360 [1], Tanks & Temples [27], and Deep Blending [18], from Tab. 2. This is the standardized setup limited to 1M primitives, 30k iterations, same number of appearance features per primitive (48) and same hyperparameters for all scenes, isolating entirely the effect of our approach from other variables.

the same configuration from our pure RGB radiance field experiments; further tuning for this specific use case may render higher performance.

Differences to RGB-only training. Aside from the extra output head and loss terms, training proceeds as in the main paper: same optimizer, learning-rate schedules, EMA on the MLP, and color-refinement phase. The only architectural change is the increased output dimension (3 + 512 when fused, or a separate 512-dim linear layer). We do not use a separate training stage for LSEG; RGB and LSEG are optimized jointly from the start.

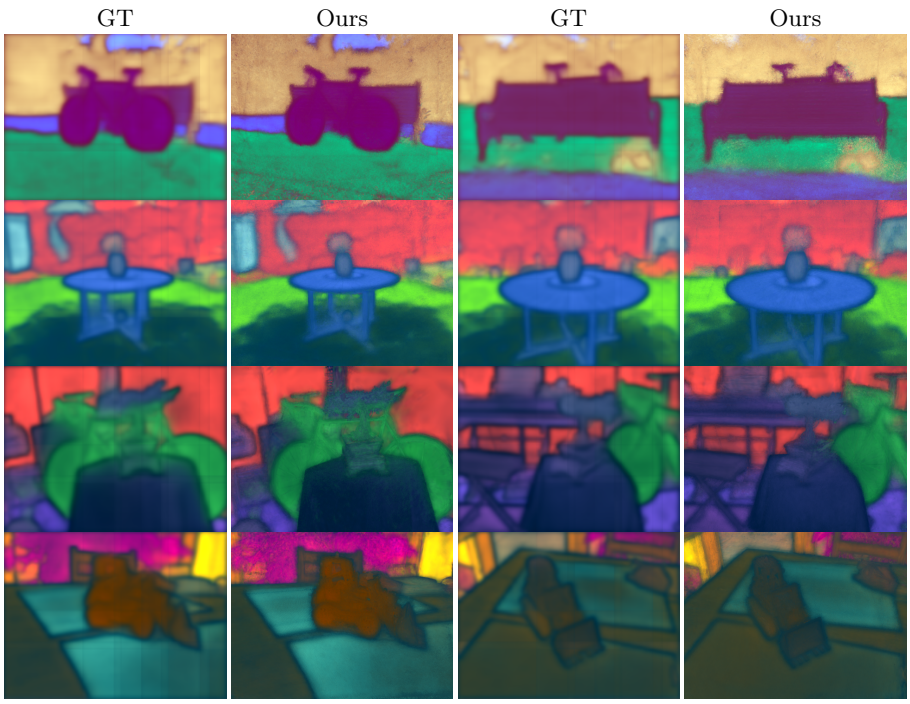


Fig. 8: LSEG feature PCA visualizations on test views from MipNeRF360. Rows from top to bottom: *bicycle*, *garden*, *bonsai*, *kitchen*. Each pair shows the ground-truth LSEG features (GT) alongside our rendered features (Ours). Our method faithfully reconstructs semantic feature maps while preserving sharp boundaries, at higher resolutions, and real-time.

PCA visualization. LSEG feature maps are 512-dimensional and therefore not directly viewable. For qualitative inspection, we project both ground-truth and predicted features to RGB via PCA. The PCA basis is computed from the first image, then applied the to all frames. Fig. 8 shows example PCA visualizations. They allow a direct visual comparison of semantic structure and show that the predicted features align well with the target, while being substantially faster to generate and higher resolution than inferring the RGB frame only and running the LSEG model to obtain its feature map. Semantic scene reconstruction extracts detail from the multiple training views, improving the granularity of the semantic information available for downstream applications.

Finally, in Tab. 12 we ablate the feature dimensionality for the joint RGB and LSEG semantic reconstruction task.

B Feature interpolation

For a spatial coordinate $\mathbf{p}$ lying on the surface of our topology, or inside of its volume, we can compute the feature vector by barycentrically interpolating the feature vectors at the vertices of the shape. For example, for a 2D triangle with vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$, the feature vector $f(\mathbf{p})$ is computed as

$$f(\mathbf{p}) = \sum_{i=1}^{n} \alpha_i f_i \tag{8}$$

where f_i are the features at the triangle vertices, and α_i are the barycentric coordinates at the evaluated point. In the case of 2D triangles, barycentric weights are computed as

$$\alpha_i = \frac{A_i}{A}, \quad A = \sum_{j=1}^{n} A_j \tag{9}$$

where A is the total area of the triangle and A_i is the area of the sub-triangle formed by the evaluation point $\mathbf{p}$ and the edge opposite to vertex i. For a triangle with vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$:

$$\begin{aligned} A_1 &= \frac{1}{2} \left\| (\mathbf{v}_2 - \mathbf{p}) \times (\mathbf{v}_3 - \mathbf{p}) \right\|, \\ A_2 &= \frac{1}{2} \left\| (\mathbf{v}_3 - \mathbf{p}) \times (\mathbf{v}_1 - \mathbf{p}) \right\|, \\ A_3 &= \frac{1}{2} \left\| (\mathbf{v}_1 - \mathbf{p}) \times (\mathbf{v}_2 - \mathbf{p}) \right\|. \end{aligned} \tag{10}$$

For 2D connected triangles, this is similar to vertex features recently introduced in the context of neural global illumination [52]. Our formulation however is more general and extends to higher order topologies and disconnected meshes. For tetrahedra, barycentric weights are similarly computed as

$$\alpha_i = \frac{V_i}{V}, \quad V = \sum_{j=1}^{n} V_j \tag{11}$$

where V is the total volume of the tetrahedron and V_i is the volume of the sub-tetrahedron formed by the evaluation point $\mathbf{p}$ and the face opposite to vertex i. For a tetrahedron with vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$:

$$\begin{aligned} V_1 &= \frac{1}{6} \left\| (\mathbf{v}_2 - \mathbf{p}) \times (\mathbf{v}_3 - \mathbf{p}) \cdot (\mathbf{v}_4 - \mathbf{p}) \right\|, \\ V_2 &= \frac{1}{6} \left\| (\mathbf{v}_3 - \mathbf{p}) \times (\mathbf{v}_4 - \mathbf{p}) \cdot (\mathbf{v}_1 - \mathbf{p}) \right\|, \\ V_3 &= \frac{1}{6} \left\| (\mathbf{v}_4 - \mathbf{p}) \times (\mathbf{v}_1 - \mathbf{p}) \cdot (\mathbf{v}_2 - \mathbf{p}) \right\|, \\ V_4 &= \frac{1}{6} \left\| (\mathbf{v}_1 - \mathbf{p}) \times (\mathbf{v}_2 - \mathbf{p}) \cdot (\mathbf{v}_3 - \mathbf{p}) \right\|. \end{aligned} \tag{12}$$

B.1 Optimization choices and hyperparameters

Tab. 14 ablates the individual contributions of the training strategies described in the main paper (learning rate scheduling, EMA on the MLP weights, loss scaling, color-refinement phase, and opacity and scale regularization). Tab. 15 lists the full set of hyperparameters and optimization details used for radiance field reconstruction.

C 2D Image Reconstruction Details

We provide additional details on the high-resolution HDR image fitting experiment of Sec. 5.2. We adapt Neural Harmonic Textures to this domain while preserving the core idea: learnable per-vertex features, interpolated per pixel and activated with periodic functions, and decoded by a shallow MLP. However, there are a number of differences to the 3D and semantic reconstruction tasks, which we detail below.

C.1 Representation

Triangle mesh. Unlike the volumetric disconnected primitives used in the scene reconstruction tasks, the 2D image fitting experiment employs a *connected Delaunay triangulation* that tiles the image plane without overlap. Every pixel belongs to exactly one triangle; there is no alpha blending, no Gaussian kernel, and no depth ordering. Vertices can be freely learned and positioned, and carry individual learnable feature vectors. Given a query pixel, its containing triangle is found via a tile-accelerated rasterizer, and the features at the triangle's three vertices are interpolated to produce a per-pixel feature vector.

Clough–Tocher C^1 interpolation. Standard barycentric (linear) interpolation produces a C^0 feature field: while continuous, the feature gradient is discontinuous across triangle edges. This turns out to be problematic when meshes are fully opaque, as it creates high frequency artifacts. Despite being followed by a frequency encoding and a nonlinear MLP, these gradient discontinuities manifest as visible seam artifacts along every edge.

To eliminate this, we replace linear interpolation with *Clough–Tocher cubic interpolation*, which constructs a degree-3 Bézier patch on each triangle from vertex values and vertex gradients, guaranteeing C^1 continuity everywhere (and C^∞ within each triangle). The 10 Bézier control points are:

$$
\begin{aligned}
&c_{300} = f_0, \quad c_{030} = f_1, \quad c_{003} = f_2, \\
&c_{210} = f_0 + \tfrac{1}{3}\,\nabla f_0 \cdot \mathbf{e}_{01}, \quad c_{120} = f_1 - \tfrac{1}{3}\,\nabla f_1 \cdot \mathbf{e}_{01}, \\
&c_{021} = f_1 + \tfrac{1}{3}\,\nabla f_1 \cdot \mathbf{e}_{12}, \quad c_{012} = f_2 - \tfrac{1}{3}\,\nabla f_2 \cdot \mathbf{e}_{12}, \\
&c_{102} = f_2 + \tfrac{1}{3}\,\nabla f_2 \cdot \mathbf{e}_{20}, \quad c_{201} = f_0 - \tfrac{1}{3}\,\nabla f_0 \cdot \mathbf{e}_{20}, \\
&c_{111} = \tfrac{1}{6}\big(c_{210}{+}c_{120}{+}c_{021}{+}c_{012}{+}c_{102}{+}c_{201}\big) \\
&\qquad - \tfrac{1}{6}\big(c_{300}{+}c_{030}{+}c_{003}\big),
\end{aligned}
\tag{13}
$$

where f_i and ∇f_i are the feature value and gradient at vertex i, and $\mathbf{e}_{ij} = \mathbf{v}_j - \mathbf{v}_i$.

Vertex gradient estimation. The Clough–Tocher scheme requires per-vertex gradients. We estimate these via a per-vertex least-squares fit over the one-ring neighborhood: for each vertex v, we gather all neighbor vertices $\{v_j\}_{j \in \mathcal{N}(v)}$ and solve

$$\nabla f_v = \mathbf{A}_v^{-1}\, \mathbf{b}_v, \quad \mathbf{A}_v = \sum_j \Delta\mathbf{p}_j\, \Delta\mathbf{p}_j^\top, \quad \mathbf{b}_v = \sum_j \Delta f_j\, \Delta\mathbf{p}_j, \tag{14}$$

where $\Delta\mathbf{p}_j = \mathbf{p}_j - \mathbf{p}_v$ and $\Delta f_j = f_j - f_v$. The 2×2 system is solved analytically per vertex in a CUDA kernel. This operation is differentiable and computed at every forward pass, enabling gradients to flow through the vertex topology.

Feature encoding and MLP. Feature encoding and MLP decoding are the same as in the main scene reconstruction task.

C.2 Training

Pixel sampling. At each iteration, we sample a batch of pixels via stratified sampling. Ground-truth values at sub-pixel coordinates are obtained via bilinear interpolation on the target image, implemented as a CUDA texture lookup for efficiency.

Optimizers and schedules. We use Adam [26] with separate parameter groups for positions ($\mathrm{lr} = 10^{-4}$), features ($\mathrm{lr} = 5 \times 10^{-3}$), and MLP weights ($\mathrm{lr} = 5 \times 10^{-5}$). An exponential decay schedule is applied after 20 000 iterations with a factor of 0.33 every 10 000 steps, following Instant NGP [44].

Loss function. We train with a pixel-wise MSE loss in μ-law encoded space. We found this to produce better results than using the relative L2 loss suggested in the Instant NGP paper. Raw 14-bit sensor values x are transformed as $f(x) = \log(1 + \mu\, x/w)/\log(1 + \mu)$ with $\mu = 5000$ and $w = 2^{14} - 1 = 16{,}383$ (the white level of 14-bit data). This compresses the dynamic range before supervision, placing relatively more emphasis on shadow detail, which is standard in HDR imaging pipelines.

C.3 Coarse-to-Fine Densification

Training begins from a sparse mesh initialized via *edge-aware sampling*: we compute Sobel gradient magnitudes on a downsampled version of the target image and sample initial vertex positions proportionally to edge strength, with a small uniform floor for coverage. The result is a Delaunay triangulation of the sampled points, with boundary vertices ensuring full image coverage.

The mesh is progressively refined through densification. Every 500 iterations (starting at iteration 1 500 and ending at 15 000), we score each triangle by

$$\mathrm{score}_t = \overline{\mathrm{DSSIM}}_t \cdot |\mathrm{pixels}_t|^\alpha, \tag{15}$$

where $\overline{\text{DSSIM}}_t$ is the mean DSSIM (per-pixel structural dissimilarity) of triangle t and $\alpha = 0.75$ is the area-weighting exponent. The area weighting prevents repeated splitting of tiny, high-error triangles and distributes vertices more evenly in proportion to their spatial impact. Triangles with fewer than 3 pixels are excluded from scoring.

At each densification step, we insert centroid vertices for the top-scoring triangles (up to $0.35 \times V$ new vertices, where V is the current count), initialize their features by averaging the parent triangle's vertex features, and re-run Delaunay triangulation on the full vertex set. We also detect when the mesh becomes degenerate (e.g. due to vertices becoming colinear) and re-mesh, although this rarely occurs in practice.

C.4 CUDA Rasterizer

All rasterization and interpolation operations are implemented as custom forward and backward CUDA kernels, supporting both full-image rendering and batched pixel-query modes. Training generally takes between 1-3 minutes in an RTXA6000 Ada.

C.5 Post-Training Compression

We also briefly explore the use of post-training compression techniques to further reduce storage requirements.

We compress vertex positions to `UInt16`, vertex features to uniform `Int8` (with a per-channel scale and offset), and MLP weights to `FP16`. An entropy coder (Zstandard, level 19) is applied to the serialized payload. Combining these techniques, the resulting images can be compressed by an additional factor of approximately 3× beyond the training result, with negligible quality loss. Tab. 18 summarizes the results of the post-training compression techniques.

C.6 Per-Image Results

Tabs. 19 and 20 provide per-image results for the 2D image fitting experiment at 10× and 100× compression ratios, respectively. We provide visual comparisons in Fig. 7 and Fig. 9. Our method achieves competitive PSNR in both μ-law and tonemapped spaces while substantially outperforming Instant NGP in perceptual quality (LPIPS). The advantage is most pronounced at high compression ratios, where the non-overlapping mesh topology and C^1 feature interpolation avoid the ringing and block artifacts that plague hash-grid methods.

Tab. 21 provides the full set of hyperparameters for the 2D image fitting experiment.

Scene	PSNR↑	SSIM↑	LPIPS↓
M360 Outdoor (5M, 25k iters, per-ray)			
garden	28.06	0.875	0.108
bicycle	25.53	0.782	0.203
stump	27.10	0.797	0.206
treehill	23.09	0.652	0.298
flowers	22.08	0.642	0.304
M360 Indoor (2M, 45k iters, center ray)			
bonsai	35.09	0.957	0.222
counter	30.83	0.925	0.220
kitchen	33.40	0.940	0.138
room	33.43	0.936	0.250
Tanks & Temples (2.5M, 40k iters, center ray)			
truck	26.91	0.900	0.112
train	24.45	0.865	0.169
Deep Blending (2M, 30k iters, center ray)			
drjohnson	30.43	0.918	0.309
playroom	31.45	0.921	0.296
M360 Outdoor Avg	25.17	0.749	0.224
M360 Indoor Avg	33.19	0.940	0.207
M360 Total	**28.74**	**0.834**	**0.216**
T&T Avg	**25.68**	**0.882**	**0.141**
DB Avg	**30.94**	**0.919**	**0.302**

Table 11: Per-scene results of NHT (Ours) with split training configurations on MipNeRF360 [1], Tanks & Temples [27], and Deep Blending [18]. We adapt primitive count and training length per dataset to accommodate different dataset sizes and spatial scales, but keep the same learning rates and other hyperparameters. Indoor datasets benefit from encoding the central camera ray, instead of per-ray directions, due to their high parallax, acting as a regularization on ray orientation. Trained and evaluated on the original JPEG-compressed reference images.

Dim	PSNR↑	SSIM↑	LPIPS↓	LSEG_P↑	LSEG_C↑	FPS↑
4	23.81	0.853	0.254	44.33	0.988	76
8	25.40	0.876	0.200	47.10	0.994	75
16	26.06	0.885	0.178	48.23	0.995	72
32	26.34	0.887	0.166	48.68	0.996	72
64	26.42	0.887	0.163	48.79	0.996	70

Table 12: Feature dimensionality ablation for joint RGB + LSEG reconstruction on the *truck* scene (Tanks & Temples). LSEG_P and LSEG_C denote PSNR and cosine similarity of the 512-dim semantic features, respectively. Measured on an RTX A6000 Ada.

Parameter	Value
LSEG output feature dimension	512
LSEG L_1 loss weight λ_{LSEG}	1.0
LSEG cosine similarity weight $\lambda_{\cos}$	0.1
Per-primitive feature dim. (shared with RGB)	80

Table 13: LSEG-specific hyperparameters used for joint RGB and semantic feature training.

	Configuration	PSNR↑	SSIM↑	LPIPS (Alex)↓
	Our Model	28.49	0.828	0.153
A)	No LR Scheduling	28.37	0.829	0.147
B)	No EMA on MLP Weights	28.46	0.828	0.152
D)	No Color Refinement Phase	28.35	0.827	0.154
E)	No Opacity Regularization	27.85	0.806	0.181
F)	No Scale Regularization	28.50	0.828	0.153
G)	No Direction Encoding	28.11	0.816	0.155
I)	No Direction Scaling	28.35	0.822	0.158

Table 14: Ablation study on training strategies. The rows are *not* additive, i.e. we only test one strategy at a time. All experiments on the MipNeRF360 dataset, using 64 features per primitive and a 128×3 MLP. Note that the scale regularization (F) does not significantly affect reconstruction quality, but does improve render time.

Training	
Color refinement phase	Last 3 000 steps (geometry frozen, no reg.)
Loss & regularization	
SSIM weight λ (D-SSIM vs L_1)	0.1
Opacity regularization λ_α	0.02
Scale regularization λ_s	0.005
Learning rates (initial)	
Positions (means)	1.6×10^{-4}
Scales	5×10^{-3}
Opacities	5×10^{-2}
Quaternions	1×10^{-3}
Deferred features	1.5×10^{-2}
Deferred MLP	6.8×10^{-4}
LR schedule	
Positions	Exponential decay (final factor 0.01)
Features & MLP	Cosine annealing (final factor 0.1)
EMA (deferred MLP)	
Decay γ	0.95
Start step	0
Appearance (NHT)	
Feature dim. per primitive	16-64
MLP hidden dim. $\times$ layers	$64 - 128 \times 2 - 3$
View encoding	Spherical harmonics (2nd degree)

Table 15: Hyperparameters and optimization details for radiance field reconstruction (MipNeRF360, Tanks & Temples, Deep Blending). Values follow our *gsplat* implementation [70].

Table 16: Ablating feature count N. Measured on an RTXA6000 Ada.

Dim	PSNR↑	SSIM↑	LPIPS↓	FPS↑
4	27.18	0.809	0.288	165
8	27.83	0.818	0.261	156
16	28.22	0.826	0.246	142
24	28.31	0.828	0.240	134
32	28.39	0.829	0.237	133
48	28.43	0.829	0.231	116
64	28.48	0.828	0.230	105
80	28.47	0.828	0.227	98

Table 17: Ablating MLP architecture (layer width $\times$ hidden layer count). Measured on an RTXA6000 Ada.

Size	PSNR↑	SSIM↑	LPIPS↓	FPS↑
16×2	28.16	0.824	0.240	97
16×4	28.03	0.820	0.244	98
32×2	28.25	0.825	0.238	97
32×4	28.23	0.824	0.237	101
64×2	28.38	0.827	0.234	102
64×4	28.36	0.826	0.233	102
128×2	28.47	0.828	0.230	106
128×4	28.50	0.828	0.229	103

Scheme	Bytes	BPP	Ratio	PSNR↑ (μ)	PSNR↑ (lin)	PSNR↑ (tm)	SSIM↑ (μ)	SSIM↑ (tm)
Baseline	4,809,236	0.8418	100×	33.89	38.45	34.78	0.8830	0.9163
int8 features	2,418,189	0.4233	198.4×	33.85	38.33	34.71	0.8827	0.9160
int16 positions	4,011,138	0.7021	119.6×	33.89	38.45	34.78	0.8830	0.9163
int8+int16+fp16mlp	1,607,793	0.2814	298.4×	33.85	38.32	34.70	0.8827	0.9160
+ Zstandard	1,472,273	0.2577	326.0×	33.85	38.32	34.70	0.8827	0.9160
+ Brotli	1,450,542	0.2539	331.0×	33.85	38.32	34.70	0.8827	0.9160

Table 18: Impact of post-training compression on NHT at 100×. Averages over 15 images. *Baseline*: uncompressed fp32 features/parameters + fp16 MLP weights. *int8 features*: uniform 8-bit quantization with per-channel scale/offset. *int16 positions*: fixed-point uint16 vertex positions. *int8+int16+fp16*: combined quantization (features int8, positions int16, MLP fp16). *+ Zstandard/Brotli*: entropy coding on the serialized payload.

	PSNR↑ (μ-law)			PSNR↑ (tonemapped)			SSIM↑ (tonemapped)			LPIPS↓ (tonemapped)		
Image	NHT	NGP	JXL	NHT	NGP	JXL	NHT	NGP	JXL	NHT	NGP	JXL
DSC_1012	33.98	34.04	**37.70**	36.92	**37.59**	30.38	0.97	0.96	**0.99**	0.0090	0.0137	**0.0004**
DSC_1764	40.06	39.04	**49.03**	**43.61**	41.73	32.02	0.98	0.97	**0.99**	0.0401	0.1149	**0.0012**
DSC_1967	33.60	35.01	**41.48**	**36.50**	36.19	27.92	0.95	0.93	**0.99**	0.0108	0.0178	**0.0004**
DSC_2172	35.83	36.59	**41.74**	40.23	38.95	**40.31**	0.97	0.95	**0.99**	0.0182	0.0317	**0.0008**
DSC_2655	41.80	40.91	**50.26**	44.38	42.73	**55.95**	0.98	0.98	**1.00**	0.0292	0.0850	**0.0008**
DSC_3095	38.07	38.78	**46.05**	39.83	38.43	**52.47**	0.96	0.94	**1.00**	0.0184	0.0450	**0.0009**
DSC_3351	40.00	42.20	**48.40**	44.53	42.97	**55.37**	0.98	0.98	**1.00**	0.0046	0.0179	**0.0002**
DSC_4748	33.39	34.68	**37.97**	**37.53**	37.08	31.97	0.95	0.94	**0.99**	0.0218	0.0544	**0.0010**
DSC_5007	36.09	36.80	**47.75**	37.46	36.54	**49.40**	0.93	0.90	**0.99**	0.0250	0.0430	**0.0009**
DSC_6718	41.30	40.15	**51.27**	44.57	41.64	**55.28**	0.98	0.97	**1.00**	0.0464	0.1345	**0.0010**
DSC_6926	31.24	33.15	**34.93**	33.27	**34.28**	27.42	0.93	0.91	**0.98**	0.0246	0.0449	**0.0022**
DSC_9008	40.78	40.10	**49.73**	41.93	39.98	**53.74**	0.98	0.96	**1.00**	0.0111	0.0493	**0.0006**
DSC_1658	39.05	38.51	**50.39**	41.68	39.57	**53.53**	0.97	0.95	**1.00**	0.0471	0.1445	**0.0018**
DSC_4176	35.17	36.89	**48.17**	36.50	36.64	**50.77**	0.95	0.93	**1.00**	0.0215	0.0597	**0.0006**
DSC_7685	37.11	37.54	**44.83**	39.08	39.33	**53.30**	0.97	0.96	**1.00**	0.0184	0.0550	**0.0004**
Average	37.16	37.63	**45.31**	39.87	38.91	**44.66**	0.96	0.95	**0.99**	0.0231	0.0608	**0.0009**

Table 19: Per-image comparison at 10× compression on our 14-bit HDR RAW dataset (45.7 MP). We compare NHT (Ours), Instant NGP [44], and JPEG-XL [23]. All neural methods use the same parameter budget. Metrics on tonemapped sRGB images unless noted.

	PSNR↑ (μ-law)			PSNR↑ (tonemapped)			SSIM↑ (tonemapped)			LPIPS↓ (tonemapped)		
Image	NHT	NGP	JXL	NHT	NGP	JXL	NHT	NGP	JXL	NHT	NGP	JXL
DSC_1012	31.49	31.65	**32.58**	33.34	**33.51**	30.18	0.93	0.93	**0.98**	0.0243	0.0255	**0.0065**
DSC_1764	37.41	37.22	**41.89**	**39.95**	39.70	31.89	0.96	0.96	**0.98**	0.0552	0.1341	**0.0344**
DSC_1967	30.65	31.05	**34.25**	32.41	**32.61**	27.66	0.89	0.88	**0.96**	0.0328	0.0341	**0.0057**
DSC_2172	32.96	33.31	**36.09**	35.59	35.90	**38.59**	0.92	0.92	**0.97**	0.0419	0.0436	**0.0182**
DSC_2655	40.30	40.16	**43.56**	**41.74**	41.36	26.05	0.98	0.98	**0.99**	0.0396	0.0942	**0.0121**
DSC_3095	36.52	35.93	**39.03**	36.74	35.82	**43.61**	0.93	0.92	**0.98**	0.0459	0.0663	**0.0184**
DSC_3351	40.79	41.23	**43.53**	42.12	**42.51**	29.01	0.98	0.98	**0.99**	0.0099	0.0193	**0.0020**
DSC_4748	31.05	31.26	**32.83**	34.23	**34.38**	31.47	0.91	0.91	**0.96**	0.0516	0.0724	**0.0142**
DSC_5007	33.88	34.00	**38.84**	33.65	33.68	**36.35**	0.85	0.84	**0.95**	0.0692	0.0729	**0.0143**
DSC_6718	38.55	38.68	**43.99**	40.74	40.19	**47.59**	0.96	0.96	**0.99**	0.0847	0.1571	**0.0362**
DSC_6926	29.48	29.72	**30.01**	29.91	**30.54**	27.02	0.87	0.86	**0.93**	0.0587	0.0629	**0.0331**
DSC_9008	38.38	38.39	**43.15**	37.90	**37.92**	34.54	0.95	0.95	**0.98**	0.0275	0.0514	**0.0096**
DSC_1658	36.56	36.45	**41.84**	37.87	37.29	**44.60**	0.93	0.93	**0.98**	0.0890	0.1768	**0.0489**
DSC_4176	33.49	33.56	**38.28**	33.60	33.63	**40.03**	0.90	0.89	**0.97**	0.0521	0.0880	**0.0123**
DSC_7685	34.42	34.91	**38.10**	36.63	36.86	**44.49**	0.94	0.94	**0.98**	0.0345	0.0681	**0.0081**
Average	35.06	35.17	**38.53**	**36.43**	36.39	35.54	0.93	0.92	**0.97**	0.0478	0.0778	**0.0183**

Table 20: Per-image comparison at 100× compression (same dataset and setup as Tab. 19).

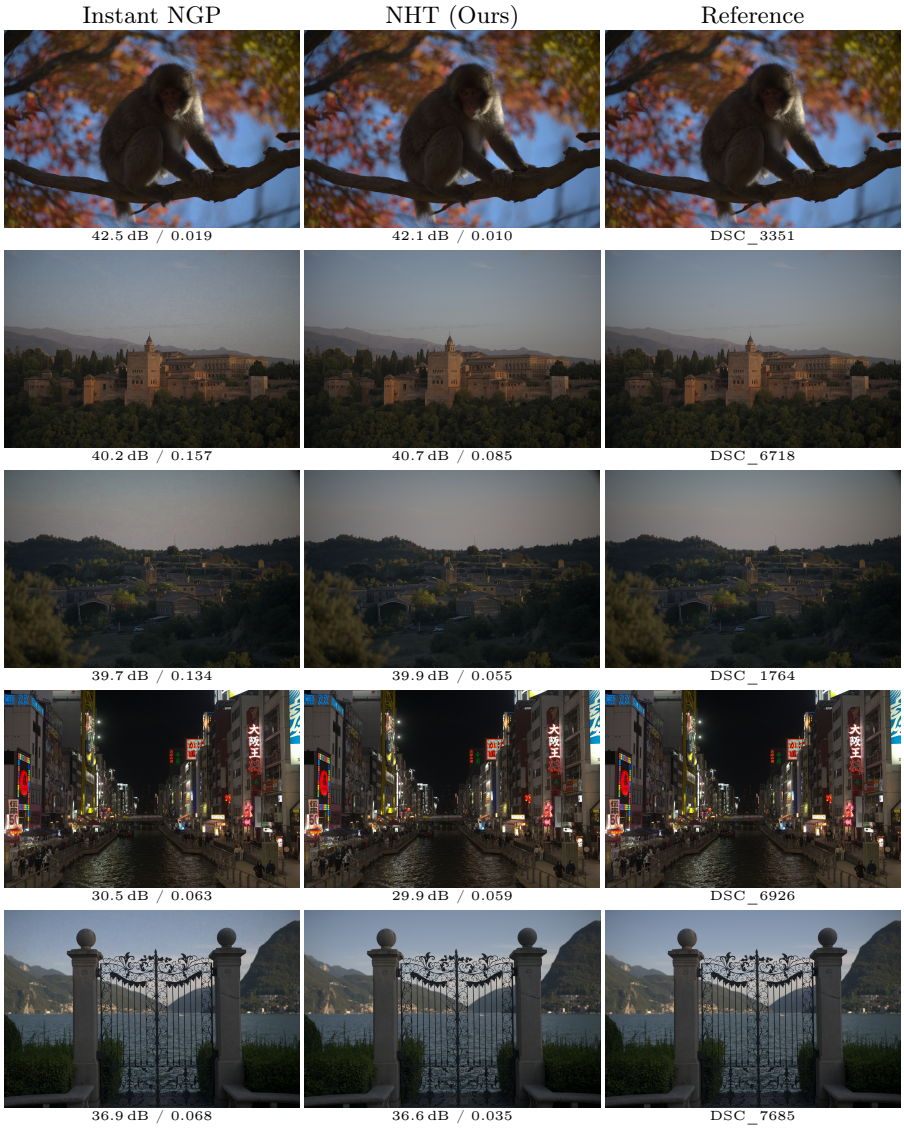


Fig. 9: Visual comparison at 100× compression. Each cell shows tonemapped PSNR (dB) and LPIPS. Our method (NHT) consistently achieves lower LPIPS (better perceptual quality) than Instant NGP at comparable or higher PSNR. The images are taken from the dataset we curated.

Representation	
Mesh topology	Delaunay triangulation (shared vertices)
Interpolation	Clough–Tocher C^1 cubic
Feature activation	SinCos: $[\sin(f), \cos(f)]$
Feature dim. per vertex	4-16
MLP hidden dim. $64 - 128 \times 2 - 4$ layers	
Training	
Total iterations	25 000
Pixel sampling	Stratified, 160 000 pixels/batch
Loss function	MSE
Learning rates (initial)	
Vertex positions	1×10^{-4}
Vertex features	5×10^{-3}
MLP weights	5×10^{-5}
LR schedule	
All parameters	Exponential decay: $\eta(t) = \eta_0 \cdot 0.33^{\max(0,\,(t-20\text{k})\,/\,10\text{k})}$
Densification	
Strategy	Coarse-to-fine Delaunay re-meshing
Schedule	Steps 1 500–15 000, every 500 steps
Growth rate	0.35× current vertex count per step
Scoring	DSSIM-based, area-weighted: $\text{score}_t = \overline{\text{DSSIM}}_t \cdot \lvert\text{pixels}_t\rvert^{0.75}$
Min. triangle size	3 pixels (skip smaller triangles)
Initialization	Edge-aware sampling (Sobel)
HDR pipeline	
Input	14-bit RAW (Nikon NEF), 45.7 MP
Encoding space	μ-law: $f(x) = \log(1 + \mu\, x/w)/\log(1 + \mu)$
μ parameter	5 000
White level w	$2^{14} - 1 = 16\,383$
GT pixel lookup	Bilinear interpolation

Table 21: Hyperparameters and optimization details for the 2D image fitting experiment. The parameter budget (maximum vertices, MLP size) is derived from the target compression ratio and image dimensions and bitrate.