Lagrangian Hashing for Compressed Neural Field Representations

Shrisudhan Govindarajan*[1], Zeno Sambugaro*[2], Akhmedkhan (Ahan) Shabanov[1], Towaki Takikawa[3], Daniel Rebain[4], Weiwei Sun[4], Nicola Conci[2], Kwang Moo Yi[4], and Andrea Tagliasacchi[1,3,5]

[1]Simon Fraser University, [2]University of Trento, [3]University of Toronto, [4]University of British Columbia, [5]Google DeepMind

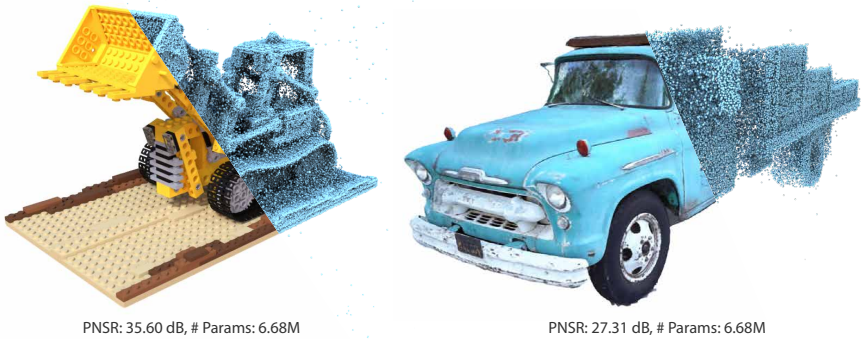


Fig. 1: We introduce a hybrid representation that is simultaneously Eulerian (grids) and Lagrangian (points), which realizes high-quality novel view synthesis as shown above, while at the same time being more memory efficient.

Abstract. We present Lagrangian Hashing, a representation for neural fields combining the characteristics of fast training NeRF methods that rely on Eulerian grids (i.e. InstantNGP), with those that employ points equipped with features as a way to represent information (e.g. 3D Gaussian Splatting or PointNeRF). We achieve this by incorporating a point-based representation into the high-resolution layers of the hierarchical hash tables of an InstantNGP representation. As our points are equipped with a field of influence, our representation can be interpreted as a mixture of Gaussians stored within the hash table. We propose a loss that encourages the movement of our Gaussians towards regions that require more representation budget to be sufficiently well represented. Our main finding is that our representation allows the reconstruction of signals using a more compact representation without compromising quality.

Keywords: Lagrangian · Point Cloud · Compact Representation · Neural Field

1 Introduction

As immersive mixed reality interfaces and volumetric content capture systems become popular, there is an increasing demand for a multimedia format that can compactly represent and transmit various types of multimedia. The diversity of multimedia formats (images, videos, volumetric 3D, radiance fields, etc) in mixed reality systems necessitates the codecs themselves to also be flexible to handle different types of formats.

Neural fields [46] have emerged as a general data format that can represent different multimedia formats with a unified codec based on model fitting. They have been popular in recent literature for representing all sorts of data, but in particular have been widely used for radiance field reconstruction [27]. In contrast to traditional multimedia formats that convert data into alternate formats through transformations, neural fields convert data by fitting a model to the data via optimization. This adds to the flexibility of the transformation by allowing the integration of additional objective functions and scenarios, like 3D reconstruction in an inverse problem setting.

A class of neural field models that have been particularly successful are *feature grids*, which uses a differentiable data structure holding features and a small multi-layer perceptron (MLP) as the model. These models inherit the strengths of highly performant data structure (such as an octree [39] or hash grid [28]) which allow the neural fields to fit complex data with large spatial extent without sacrificing performance. These feature grids methods, however, typically come at the cost of a larger memory footprint.

Although many data structure tricks like sparsity [26, 39], low-rank factorization [7], linear transforms [31], and hash probing [40] have been proposed to improve the *memory-quality tradeoff curve* of feature grids, these works do not fundamentally address the spatially non-homogeneous structure of 3D data: more features should be allocated for parts of the data with higher complexity. Achieving this objective would let the representation use the available memory footprint more efficiently.

These feature grid-based neural fields typically represent data in an *Eulerian* way, as a vector field over some coordinate system. Even if they use sparse or factorized data structures, they generally use a grid where vertices are laid out in uniform intervals which allows for simple implementations of indexing algorithms. *Lagrangian* ways of representing data, on the other hand, would allow the representation to flexibly allocate the grid points in space, but may suffer from more complex indexing algorithms that may involve techniques like approximate nearest neighbour searches. [1]

Our work, Lagrangian Hashing (LagHash), marries the simplicity and performance of *Eulerian* representations where features are laid out on uniform grids with a *Lagrangian* representation that employs a point-based representation in

[1] We refer to Eulerian and Lagrangian representations in numerical physics, where one can store the state of the system (*e.g.*, velocity of a fluid) on either grids (Eulerian) or on particles (Lagrangian).

which features can freely move around in space. In more details, our representation builds upon hierarchical hashes introduced by InstantNGP [28], but, in each hash bucket, rather than just storing a feature, we store a small set of points, each equipped with a feature.[2] While other point-based representations require acceleration data structures to access the points, our representation *reuses* the hash implementation. Most importantly, as point-clouds can adaptively allocate representation budget by increasing the density of the point cloud where needed, they are a more effective representation for storing high-frequency information, which results in smaller models that achieve similar visual performance.

2 Related Works

In what follows, we review the literature on traditional compression, neural compression method and point-based representation.

Traditional Compression. The dominant approach in lossy compression for traditional multimedia usually involve transform coding [15], quantization [16], and entropy coding [18]. Linear projections into a fixed basis space (like discrete cosine transform [1]) are often used in practice for image and video codecs like JPEG [44]. Transform coding methods usually treat the multimedia data as a collection of vectors, where each vector represents a local fixed-size patch of pixels. These *block-based* methods also exist for 3D volumetric data [4], where 3D patches of voxels are encoded using linear transformations [10, 42, 43]. These methods are not spatially adaptive, in that each equal-sized patch of an image or volume go through the same encoding regardless of the local resolution content. In contrast, our work uses a point-based, Lagrangian formulation where more feature vectors can be placed where there is more complexity of data. The spatial adaptivity allows our work to more efficiently allocate resources.

Neural Compression. Instead of transforming data, model-based compression methods compress data by fitting a model to the data. These models can be polynomials [12], partial differential equations [13], gaussian mixture models [9], or neural fields [32]. Works that use neural fields either use multi-layer perceptrons [11, 32, 35] or feature grids [28, 38, 40]. Most compression methods that use neural fields have still not adapted models that use point-based representations which have an advantage of spatial adaptivity. In contrast, our work uses a hybrid of Eulerian feature grids and a Lagrangian representation that enables spatial adaptivity.

Point-based Representations. Point-based representations have been widely explored as a representation in computer graphics [2, 17, 22, 33]. Polygon meshes, which are a dominant representation in computer graphics, can be seen as a form of a point-based representation, as the points lying on a polygonal primitive are represented as convex combinations of its vertices. Point-based representations have also been used in reconstruction settings in conjunction with neural

[2] Note this is done on a selection of levels, and therefore our representation is an Eulerian-Lagrangian hybrid.

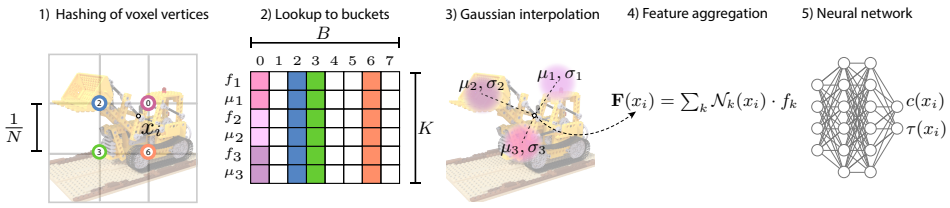


Fig. 2: *(1) Hashing of voxel vertices:* For any given input coordinate x_i, our method identifies surrounding voxels across L Levels of detail (Lods) (Only one Lod is showed for convenience). Indices are then assigned to the vertices of these voxels, through hashing procedure. *(2) Lookup to buckets:* for all resulting corner indices, we look up the corresponding B buckets, containing K feature vector and their corresponding μ_k position. *(3) Gaussian interpolation:* We compute Gaussian weights with respect to the input position for every feature vector in the bucket. *(4) Feature aggregation:* We multiply the Gaussian weights for the feature corresponding to the feature vector and aggregate them from every level of detail. *(5) Neural Network*: the resulting concatenated features are mapped to the input domain by the Neural Network.

fields, but they frequently either require careful initialization, such as points from depth [3,23,29,30,47], COLMAP point cloud [19] and LiDAR [6,36]. In contrast, our method does not require careful initialization, and guides point placement via a carefully designed guidance loss.

Recently, point-based neural rendering [19, 24, 51, 52] enabled the rendering of 3D point clouds onto images via differentiable rasterization. Particularly, Gaussian Splatting [19] and its follow-ups [45, 48–50] have achieved impressive results in rendering quality and test-time efficiency. Despite their quick adoption, these methods still rely on COLMAP for point initialization, and carefully tuned heuristics to grow and prune points. More importantly, these methods typically require *large storage*, with millions of points needed to accurately represent a scene, whereas our method naturally leads to a compact representation. For more details of recent advances on Gaussian Splatting, we refer the interested readers to a recent survey [8].

3 Method

We introduce a new representation that combines the Eulerian nature of fast-training NeRF methods [28] to the Lagrangian nature of emergent 3D Gaussian Splatting (3DGS) representations [19]. Building directly on top of the hierarchical Eulerian representation introduced by InstantNGP [28], we achieve this by incorporating a point-based representation in the high-resolution layers of its hash tables. We select the high-resolution layers as to let the model focus its representation power to precise locations in space, akin to how 3DGS [19] employs small Gaussians to capture fine-grain details of the scene. We (implicitly) equip each point with a standard deviation proportional to the grid resolution,

hence our representation can be interpreted as a mixture of Gaussian with non-trainable standard deviation and mixture weights. Differently from 3DGS, we employ *isotropic* Gaussians, and the standard deviation associated with each point describes the portion of space that the feature stored alongside the point position is meant to represent.

Overview A visual overview of our representation can be found in Figure 2. We reviewing the hashed multi-scale representation in Section 3.1, how features are interpolated within each level in Section 3.2, and then detail how to augment hash buckets with a mixture-of-Gaussians representation in Section 3.3. We discuss our training methodology in Section 3.4, including the introduction of a loss that is critical to guide the MoG towards regions of space that require additional representation power.

3.1 Multi-scale representation

Analogously to InstantNGP [28], our architecture produces a field value $\mathcal{F}(\mathbf{x})$ at an arbitrary position in space $\mathbf{x}$ by interpolating feature vectors evaluated at the vertices of a stack $l = \{1, \dots, L\}$ of regular grids. Their resolution follows the geometric progression $N_l = N_{\min} \cdot b^l$, where L, $N_{\min}$ and b are hyper-parameters. The field value is computed by concatenating ($\oplus$) the features across levels, and then passing this vector through a shared decoder with parameters $\boldsymbol{\theta}$:

$$\mathcal{F}(\mathbf{x}) = \text{MLP}(\mathbf{f}_1(\mathbf{x}) \oplus \mathbf{f}_2(\mathbf{x}) \oplus ... \oplus \mathbf{f}_L(\mathbf{x}); \boldsymbol{\theta}), \qquad \mathcal{F}(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}^F \tag{1}$$

3.2 Per-level feature – $\mathbf{f}_l(\mathbf{x})$

Let us now consider how each feature $\mathbf{f}_l(\mathbf{x})$ is computed. Denote with $\mathbf{F}$ a tensor of features in memory, and let $\mathcal{H}(\mathbf{x})$ be an indexing function that retrieves the *indices* of $\mathbf{F}$ corresponding to the features of the grid corners $\{\mathbf{v}_v\}$ of field query position $\mathbf{x}$. Denote the corresponding grid interpolation weights as $\{\alpha_v\}$, where $v = \{1, \dots, V{=}2^D\}$. We interpolate features at position $\mathbf{x}$ as:

$$\mathbf{f}_l(\mathbf{x}) = \sum_{v \in \mathcal{H}_l(\mathbf{x})} \alpha_v \cdot \mathbf{F}_l[v](\mathbf{x}) \tag{2}$$

Similarly to InstantNGP [28], we implement $\mathcal{H}_l$ as an injective map whenever $N_l < B$, and as a hash function otherwise. In other words, $\mathcal{H}_l : \mathbb{R}^D \rightarrow [1, N_l - 1]$ if $N_l < B$, and $\mathcal{H}_l : \mathbb{R}^D \rightarrow [0, B - 1]$ otherwise, and respectively the feature tensor $\mathbf{F} \in \mathbb{R}^{N_l \times F}$ or $\mathbf{F} \in \mathbb{R}^{B \times F}$. As we closely follow InstantNGP [28] to enable fair comparisons, we refer the reader to this paper for additional details regarding the implementation of $\mathcal{H}$. As at finer levels v indexes the buckets of a hashing operation, we refer to $\mathbf{F}_l[v]$ as a "per-bucket" feature.

Eulerian vs. Lagrangian features. Note that, differently from InstantNGP [28], the notation in (2) hints at the fact that the feature grid is *evaluated* at the position $\mathbf{x}$. The feature evaluation depends on the depth of the corresponding

level. Consider a hierarchical hash of L levels where the last $\tilde{L}$ are Lagrangian. At shallow levels, that is $l<(\mathrm{L}-\tilde{L})$, the feature $\mathbf{F}$ follows InstantNGP [28], that is, the feature is retrieved from the hashed Eulerian grid. In other words, $\mathbf{F}_l[v](\mathbf{x}) \equiv \mathbf{F}_l[v]$. At deeper levels, we retrieve the features from a Lagrangian representation that takes the form of a *Gaussian mixture*, which we will detail next in Section 3.3.

3.3 Per-bucket feature (Lagrangian) – $\mathbf{F}_l[v](\mathbf{x})$

To simplify notation, and without loss of generality, let us drop the bucket index $[v]$ and the level index l. Within each bucket, we store an isotropic Gaussian mixture consisting of $k = \{1, \ldots, K\}$ elements, parameterized by mean $\boldsymbol{\mu}_k$, standard deviation σ_k^2, and corresponding feature vector $\mathbf{f}_k$. The feature is computed by evaluating the Gaussian mixture at position $\mathbf{x}$:

$$\mathbf{F}(\mathbf{x}) = \sum_k \mathcal{N}_k(\mathbf{x}) \cdot \mathbf{f}_k, \qquad \mathcal{N}_k(\mathbf{x}) = \frac{1}{(2\pi)^{1/2}\sigma_k} \exp\left(-\frac{\|\mathbf{x}-\boldsymbol{\mu}_k\|_2^2}{2\sigma_k^2}\right). \tag{3}$$

3.4 Training (novel view synthesis)

During training, we jointly optimize the shared decoder parameters, the Eulerian representation for levels $l<(\mathrm{L}-\tilde{L})$, and the Lagrangian representation for the last $\tilde{L}$ levels. The latter includes the mean and feature, and (optionally) the standard deviation of each Gaussian. To train our representation, we optimize the loss:

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda_{\text{dist}}\mathcal{L}_{\text{dist}} + \lambda_{\text{guide}}\mathcal{L}_{\text{guide}} \tag{4}$$

where $\mathcal{L}_{\text{recon}}$ is the pixel reconstruction loss computed by volume rendering [27], evaluated via a Huber loss analogously to InstantNGP [28], $\mathcal{L}_{\text{dist}}$ is the distortion loss proposed in [5] to promote the formation of surfaces within the volume, and $\mathcal{L}_{\text{guide}}$ avoids vanishing gradients in the optimization of the Lagrangian portion of our representation by guiding the movement of Gaussians towards surfaces.

Guidance loss. During training, whenever we back-propagate a position $\mathbf{x}$ with respect to a Gaussian whose mean $\boldsymbol{\mu}$ that is too far from $\mathbf{x}$ (scaled by the standard deviation σ^2), the computed gradients become very small, which interferes with effective optimization of our representation. Note that a very similar problem is found in the training of Gaussian Mixture Models, for which Expectation-Maximization (EM) is typically employed to address this issue [34]. In defining our loss, we take *inspiration* from EM training of GMMs. For the moment being, consider having Gaussians at a *single* level, as we will extend this later to the multi-level setting. In the *E-step*, given a query point $\mathbf{x}$ we identify the Gaussian (across buckets *and* Gaussians therein) whose PDF (scaled by the mixture weights) is maximal:

$$G(\mathbf{x}) = \alpha_{v^*} \cdot \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{k^*,v^*}, \sigma^2_{k^*,v^*}), \quad k^*, v^* = \arg\max_{k,v} \ \alpha_v \cdot \mathcal{N}_{k,v}(\mathbf{x}) \tag{5}$$

Then, in the *M-step*, we optimize the parameters so to minimize the discrepancy between $G(\mathbf{x})$ and the NeRF integration weights $W(\mathbf{x}) = T(\mathbf{x}) \cdot \tau(\mathbf{x})$, which is a PDF along the ray, as derived in [37, Eq. (29)]. [3] We measure the discrepancy between the two PDFs via the KL-divergence, and after dropping the subscripts (k^*, v^*) to simplify notation we note that the equality:

$$\arg\max_{\boldsymbol{\mu},\sigma^2} \ \mathrm{KL}(W(\mathbf{x})||G(\mathbf{x})) \equiv \arg\max_{\boldsymbol{\mu},\sigma^2} \ -W(\mathbf{x}) \cdot \log(G(\mathbf{x})) \tag{6}$$

holds due to the definition of KL divergence, and that the term $W(\mathbf{x}) \cdot \log(W(\mathbf{x}))$ is constant with respect to the Gaussian means, and hence can be dropped. Similarly to the EM algorithm, our approach involves alternating between two key steps: 1) determining the posterior distribution of latent variables by assigning each point to a Gaussian, and 2) maximizing the KL divergence based on the defined correspondence. This two-step process can be written as a loss: [4]

$$\mathcal{L}^l_{\text{guide}}(\mathbf{x}) = -W(\mathbf{x}) \cdot \ \log\left(\max_{k,v} \ \alpha_{v,l} \cdot \mathcal{N}_{k,v,l}(\mathbf{x})\right) \tag{7}$$

and as the max commutes with the (monotonic) log operator, after some straightforward algebraic manipulations, we define our loss to its final form:

$$\mathcal{L}_{\text{guide}}(\mathbf{x}) = \sum_l \mathcal{L}^l_{\text{guide}}(\mathbf{x}), \mathcal{L}^l_{\text{guide}}(\mathbf{x}) = W(\mathbf{x}) \left(\min_{k,v} \ -\log\left(\alpha_{v,l}\right) + \frac{\|\mathbf{x} - \boldsymbol{\mu}_{k,v,l}\|_2^2}{2\sigma^2_{k,v,l}}\right) \tag{8}$$

Note that this loss has a very intuitive interpretation. It states that, while integrating along a ray, if we find a position $\mathbf{x}$ that is likely to lie on a surface (i.e. $W(\mathbf{x}) \approx 1$), then there should be *one* Gaussian nearby (i.e. $\boldsymbol{\mu} \approx \mathbf{x}$). Further, if we assume constant α and σ^2, note that amongst all possible Gaussians the loss will select the *closest* Gaussian. In other words, our loss is a (scaled) one-sided *Chamfer* loss between Eulerian and Lagrangian representations.

3.5 Implementation

We now provide an overview of the key implementation details of our method. [5] We based our NeRF implementation on the *nerfacc* framework [25] and our 2D image fitting on the Kaolin-wisp library [41]. For our experiments, we chose a feature dimension, F=2, and L=16 levels of detail (LoDs) for all the experiments in the paper as proposed in InstantNGP. As the decoder in our architecture, we employ a Multi-layer Perceptron (MLP) with one hidden layer, containing 64 neurons for all the tasks.

[3] We denote density with $\tau(\mathbf{x})$ to avoid confusion with the Gaussian's variance σ^2.

[4] We draw *inspiration* from EM algorithm, but there are distinctions. Our implementation does not incorporate optimizable mixture weights, and we employ a *hard* assignment between points and Gaussians. Finally, each point in our model is weighted, contrasting with the EM algorithm in GMM where all points have equal weights.

[5] We provide a copy of our source code in the supplementary materials.

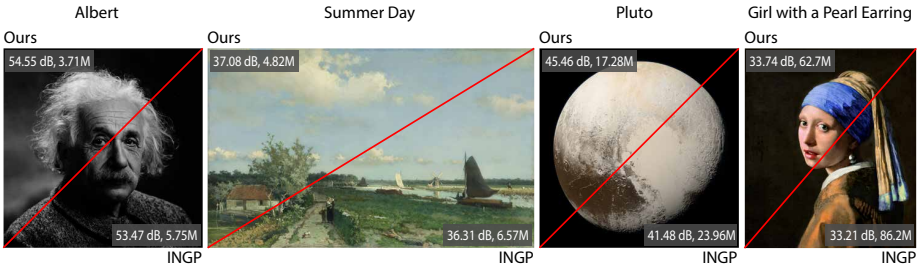


Fig. 3: Qualitative comparisons on the giga-pixel images. On each image, we show the reconstruction quality (PSNR) together with the number of parameters.

For initialization of our decoder, we employ a classical Xavier uniform distribution [14]. The hash table features are initialized using a normal distribution with zero mean and standard deviation of $1e^{-3}$. We parameterize the Gaussians with a learnable mean and a *fixed* standard deviation. The initialization of Gaussian means for the image tasks is randomized within the image space, while for Neural Radiance Field (NeRF) reconstruction, they are uniformly distributed within a sphere of radius 0.75. In our framework, the standard deviation of each Gaussian defines a field of influence for each feature vector. The standard deviations are directly related to the spatial domain of the grid vertices. We initialize the standard deviations as $50\times$ the measure of the grid cells for each LoD, and then exponentially decay their size during training to $5\times$ the measure of the grid cell for each LoD. This approach proved to be beneficial, particularly at the initial stages of training; allowing for smooth convergence of the Gaussians to regions with high surface density weights.

4 Experiments

We demonstrate the efficiency of our method in building compact representations with two distinct applications: 2D image fitting (Section 4.1), and 3D radiance reconstruction from inverse rendering (Section 4.2, Section 4.3). We evaluate image reconstruction on four complex high-resolution images, and NeRF reconstruction on the Synthetic blender dataset [27] and the Tanks & Temples dataset [21]. We conclude by ablating our design choices (Section 4.4).

4.1 2D image fitting (gigapixel)

The image fitting task involves learning a mapping between 2D coordinates and image colors, and is a popular benchmark for evaluating neural field methods capabilities in representing high-frequency signals. We train with an L2 reconstruction loss, and parameterize our models with codebook containing $B = 2^{17}$ and $B = 2^{18}$ buckets, utilize a maximum of K=4 Gaussian feature per bucket, and set the maximum resolution N_{max} to half the width of the image. For this

Table 1: We quantitatively compare our method with InstantNGP [28] on four gigapixel images: Girl with a Pearl Earring (Girl), Pluto, Summer Day(Summer), and Albert. We compare average PSNR↑ and average # paramaters↓.

Method	# Params	Girl	Pluto	Summer	Albert	Avg.
InstantNGP($B = 2^{19}$)	11.91M	28.73	38.59	37.26	53.81	39.60
Ours($B = 2^{17}$)	4.56M	27.60	37.53	37.08	54.55	39.19
Ours($B = 2^{18}$)	8.41M	28.83	39.72	38.93	55.35	40.71

task, the spatial gradient norm of pixel values $\|\nabla I(x_i)\|$ is employed for $W(\mathbf{x})$ in $\mathcal{L}_{\text{guide}}$ (8), hence prioritizing representation of areas within the image containing high-frequency details.

Dataset and baselines. We evaluate our method qualitatively and quantitatively on publicly available gigapixel images, where the total number of pixels ranges from 4 M to 213 M, with InstantNGP as a representative baseline. We train both our network and InstantNGP with Adam [20] for 350 epochs with a learning rate of $1e^{-2}$ and parameters β_1=0.9, β_2=0.99, ϵ=10^{-15}, while the learning rate for Gaussian positions is set to $1e^{-3}$. We use a batch size of 2^{16}.

Discussion. As shown in Figure 3 and Table 1 our method reconstructs high-fidelity gigapixel images in comparison to InstantNGP while also being a 2.6× more compact representation. Note the superior performance on images characterized by localized high-frequency features (e.g. Pluto) This aspect of our method is also highlighted in the pareto plot in Figure 6, where our method consistently outperforms InstantNGP on different parameters counts.

4.2 Novel view synthesis

We now demonstrate the applicability of our method in a NeRF [27] setup. Differently from the 2D image fitting task, in the NeRF setting, a volumetric shape is parameterized by a spatial (3D) density function and a spatio-directional (5D) radiance. We demonstrate that our method is capable of solving this problem better than baselines while requiring fewer parameters.

Dataset and baselines. We evaluate our method qualitatively and quantitatively with InstantNGP on two widely used benchmarks: the NeRF synthetic dataset [27] and the Tanks & Temples real-world dataset [21]. For Tanks & Temples, for both our method and InstantNGP, we use mask supervision and train without background modeling. We evaluate each method both qualitatively (Figure 4), and quantitatively (Tables 2 and 3), using peak signal-to-noise ratio (PSNR) as the metric.

Implementation. For this task, we train our models with codebook containing B=2^{17} and B=$2^{17.9}$ (i.e. we match the number of parameters of InstantNGP) buckets. We employ K=4 Gaussian features per bucket for both datasets. We

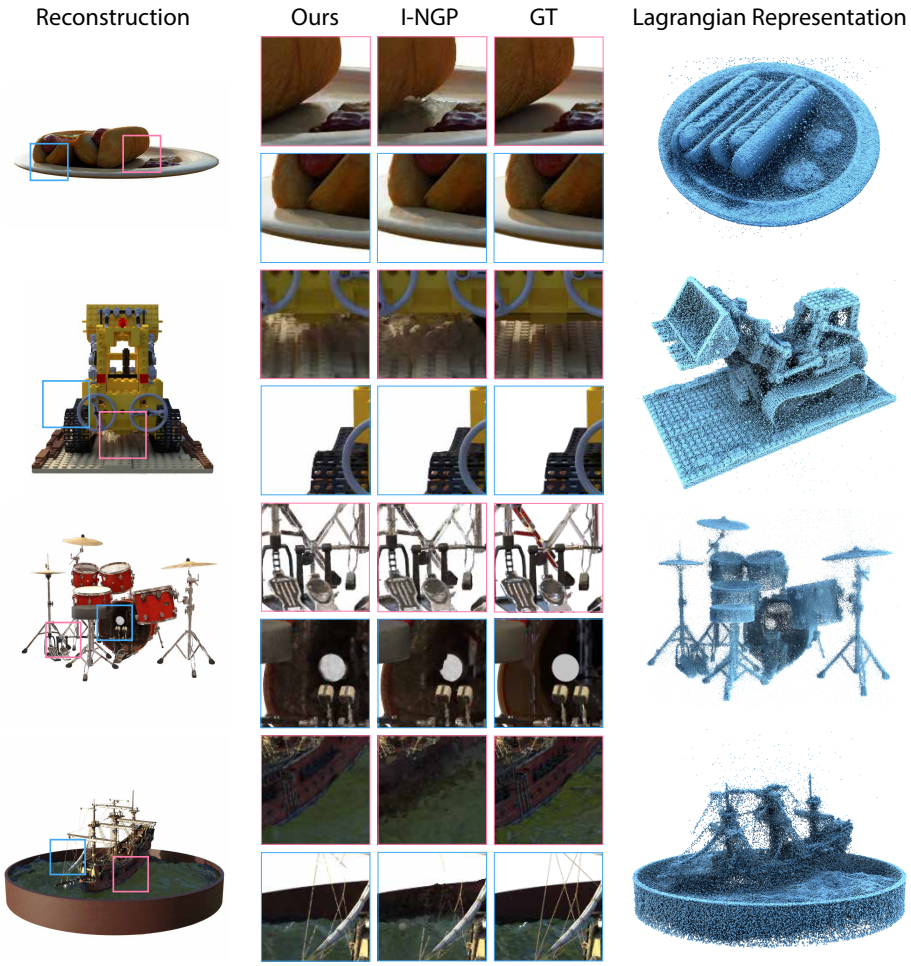


Fig. 4: Qualitative comparisons on the Synthetic NeRF Dataset [27]. The leftmost column (reconstruction) shows the full-image results of our method and the rightmost column shows the Lagrangian Representation which is learned by our model for the particular scene.

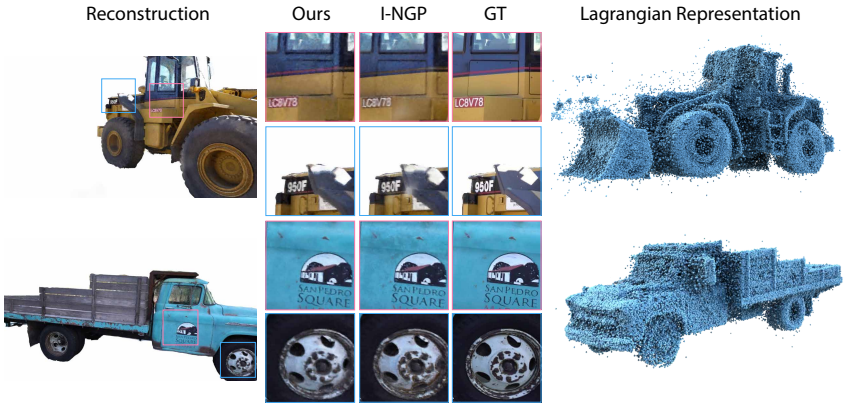


Fig. 5: Qualitative comparisons on the Tanks and Temples dataset [21]. The leftmost column (reconstruction) shows the full-image results of our method and the Rightmost column shows the Lagrangian Representation which is learned by our model.

Table 2: We compare our method with InstantNGP on the Nerf Synthetic dataset [27]. We report PSNR↑ and the parameter count ↓. Our method matches the performances of this state-of-the-art method with considerably fewer parameters.

Method	# Params	Lego	Mic	Materials	Chair	Hotdog	Ficus	Drums	Ship	Avg.
InstantNGP($B = 2^{19}$)	12.10M	35.67	36.85	29.60	35.71	37.37	33.95	25.44	30.29	33.11
Ours($B = 2^{17}$)	6.68M	35.60	36.45	29.63	35.61	37.23	33.89	25.67	30.84	33.12
Ours($B = 2^{17.9}$)	12.13M	35.74	36.78	29.66	35.76	37.30	34.02	25.75	31.01	33.25

evaluate our method's performance with a maximum grid resolution N_{max}=1024. We fix the weights of distortion loss (λ_{dist}) to $1e^{-3}$ for the NeRF Synthetic dataset and $1e^{-}2$ for the Tanks & Temples dataset, for both InstantNGP and our model. Additionally, we train our network with a guidance loss weight (λ_{guide}) of $1e^{-1}$ and a warm-up schedule to allow for the coarse structure of the scene to be learnt first. We train both the models with an Adam optimizer for $20K$ iterations with a learning rate of $1e^{-2}$ and parameters β_1=0.9, β_2=0.99, ϵ=10^{-15}. The learning rate of Gaussian positions is set to $1e^{-3}$.

Discussion. As shown in Tables 2 and 3 our method matches the quality of reconstruction of InstantNGP while achieving a 1.8× more compact representation. In Figure 5, we qualitatively evaluate our method on real data containing complex structures and reflections. These evaluations highlight our method's superior performance in handling complex scenes. Notably, our approach demonstrates its capability to resolve collisions that, within the InstantNGP framework, lead to the formation of micro-structures on smooth surfaces (as observed on the truck model, and wheels of the truck). In Figure 4, we present a visual comparison that highlights the improved rendering of thin structures, such as the legs of

Table 3: We quantitatively compare our method with InstantNGP on the Tanks and Temple dataset [21]. We compare the PSNR↑ and the parameter count ↓. Notably, we are able to match the performances of this state-of-the-art method with considerably fewer parameters.

Method	# Params	Truck	Barn	Family	Caterpillar	Avg.
InstantNGP($B = 2^{19}$)	12.10M	27.42	27.10	33.23	26.27	28.51
Ours($B = 2^{17}$)	6.68M	27.31	27.36	33.22	26.31	28.55
Ours($B = 2^{17.9}$)	12.13M	27.38	27.66	33.36	26.33	28.68

Table 4: We quantitatively compare our method with CompactNGP [40] on the Nerf Synthetic Dataset [27]. We compare the PSNR↑ and the parameter count ↓. We outperform the recently published CompactNGP while matching in parameter count.

Method	# Params	Lego	Mic	Materials	Chair	Hotdog	Ficus	Drums	Ship	Avg.
InstantNGP($B = 2^{14}$)	0.50M	32.03	35.08	28.73	32.59	34.99	30.99	25.36	27.71	30.94
CompactNGP(from [40])	0.18M	32.31	33.88	28.32	32.05	34.26	32.05	24.71	27.71	30.66
Ours($B = 2^{11}$)	0.18M	31.15	32.65	28.52	32.44	35.67	31.98	25.07	28.26	30.72

a drum instrument, the mast of a ship, and the structure of a hotdog. Figure 4 also illustrates our Lagrangian representation, where we demonstrate how the Gaussian in our model converge to regions of high surface density in 3D space.

4.3 Compact representation

We now demonstrate the efficacy of our model in developing compact representation in both Image fitting and NeRF reconstruction applications. In this experiment, we study decreasing the number of parameters to demonstrate the capability of our method to focus the limited capacity of the codebook towards capturing the high-frequency details of the signal.

Dataset and baselines. We evaluate our method quantitatively with CompactNGP [40], a compressed variant of InstantNGP, on the NeRF synthetic dataset. Since the source code for CompactNGP is not publicly available, we compare our framework with the scores directly taken from the paper. We also include in our analysis a comparison with InstantNGP, focusing on the quality of the reconstruction and compactness of the representation, across a range of hyperparameter configurations on NeRF reconstruction and 2D image fitting. For plotting our compression graph, we evaluate both our model and InstantNGP on Tanks & Temples for NeRF reconstruction and gigapixel images fitting.

Implementation. For these experiments, we train our models with codebook containing $B = 2^{11}$ buckets, and we employ a maximum of K=6 Gaussian features per bucket and evaluate the model at a maximum grid resolution N_{max}=256. We disable the distortion loss (λ_{dist}=0) for our network to enable a fair comparison with CompactNGP, that did not use this loss. Additionally, we train our network with a guidance loss weight (λ_{guide}) of $1e^{-1}$ and a warm-up scheduler. Following CompactNGP, we train our model with an Adam optimizer for

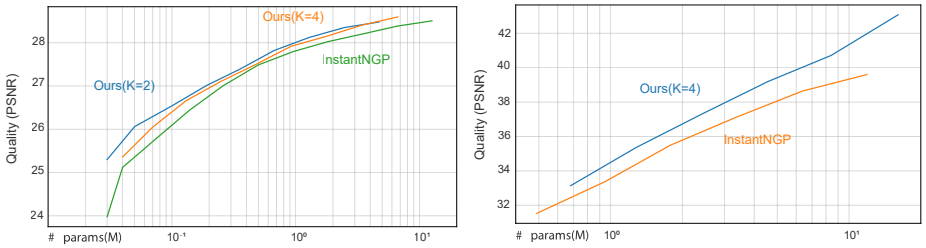
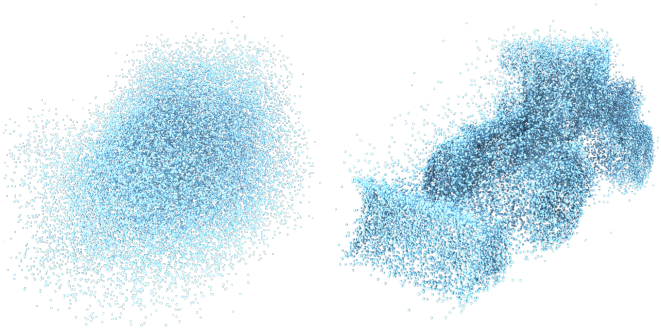


Fig. 6: Pareto plot: Tanks and temples(left), Gigapixel images(right). We demonstrate that our method consistently outperforms InstantNGP in terms of quality vs number of parameters.

	PSNR
Full	**27.94**
w/o $\mathcal{L}_{\text{dist}}$	27.70
w/o $\mathcal{L}_{\text{guide}}$	27.75

(a) Impact on PSNR



(b) w/o $\mathcal{L}_{\text{guide}}$ **(c)** w/ $\mathcal{L}_{\text{guide}}$

Fig. 7: Impact of losses. We show the proposed losses are essential to have the optimal PSNR. We also show that $\mathcal{L}_{\text{guide}}$ is critical to place points onto the surface (i.e., locations in space that need more capacity to be represented), which is the key to achieve a good compression rate.

$35K$ iterations with a learning rate of $1e^{-2}$ and parameters β_1=0.9, β_2=0.99, ϵ=10^{-15}. The learning rate of Gaussians is set to $1e^{-3}$.

Discussion. As shown in Table 4, our method quantitatively outperforms the recently proposed CompactNGP method on its primary task, while achieving the same level of compression, a 2.8× more compact representation when compared to InstantNGP. In Figure 6, we show that our pareto front lies ahead of InstantNGP in both NeRF application and image fitting, meaning that our model *consistently* outperforms InstantNGP in terms of quality vs. number of parameters.

4.4 Ablations

We validate our method in the NeRF reconstruction on the real-world scenes from Tanks & Temples. We investigate the importance of the losses, the number of Gaussian features per codebook, and the number of Lagrangian levels.

Table 5: Num. of Gaussian (K). We report the storage (the number of parameters) and PSNR with the different number of Gaussian in each bucket. We observe that 4 Gaussians achieve a better trade-off between performance and storage.

# of Gauss.	No mixture	2	**4**	8
# Params. (M)	0.50	0.67	**0.92**	1.41
PSNR	27.49	27.82	**27.94**	27.99

Table 6: Num. of levels ($\tilde{L}$). We report the storage (the number of parameters) and PSNR with the different number of Lagrangian levels. We observe that the model of 2 Lagrangian levels at the finest level Gaussians achieves the better trade-off between performance and storage.

# of levels	0	**2**	4	8
# Params. (M)	0.5	**0.92**	1.34	2.2
PSNR	27.49	**27.94**	28.00	28.03

Loss function – Fig. 7. We analyze the importance of each loss term except for $\mathcal{L}_{\text{recon}}$. Note both losses beneficially contribute to the final performance. We also qualitatively examine how losses impact point learning, which is a key to learning a compact representation – by design, we achieve a high compression rate by allocating more points near the object's surface. We observe that both losses are essential for points, and therefore features, to focus on these surfaces. Particularly, $\mathcal{L}_{\text{guide}}$ as shown in Fig. 7, guide random points towards the surface. We also observe that $\mathcal{L}_{\text{dist}}$, as discussed in MipNeRF360 [5], eliminates floaters.

Number of Gaussian features per bucket (K) – Table 5. We study the impact of the number of Gaussians in each bucket on reconstruction performance. And we show that 4 Gaussians achieve a good trade-off between the number of parameters and performance. As shown in Table 5, 8 or even more Gaussians in each bucket, while leading to better performance, start to introduce redundant parameters since the spatial collision is already well addressed with only 4 Gaussians.

Number of Lagrangian levels ($\tilde{L}$). We further investigate the impact of the number of Lagrangian levels. We observe that two Lagrangian levels in the finest levels where the majority of spatial collisions happen achieve the best trade-off between reconstruction and storage.

5 Conclusions

We introduce a neural representation that unifies Eulerian with Lagrangian schools of thought. We take advantage of the Eulerian grids that allow the use of hash tables, which we extend with the Lagrangian point clouds, imbuing them with the flexibility of point representations. Taking the best of both worlds, we outperform the state of the art, and provide a better PSNR - parameter count ratio than InstantNGP across various neural field workloads, including real-world scenes.

References

1. Ahmed, N., Natarajan, T., Rao, K.R.: Discrete cosine transform. IEEE transactions on Computers (1974)
2. Alexa, M., Gross, M., Pauly, M., Pfister, H., Stamminger, M., Zwicker, M.: Point-based computer graphics. In: ACM SIGGRAPH 2004 Course Notes (2004)
3. Aliev, K.A., Sevastopolsky, A., Kolos, M., Ulyanov, D., Lempitsky, V.: Neural point-based graphics. In: ECCV. Springer (2020)
4. Balsa Rodríguez, M., Gobbetti, E., Iglesias Guitian, J.A., Makhinya, M., Marton, F., Pajarola, R., Suter, S.K.: State-of-the-art in compressed gpu-based direct volume rendering. In: Computer Graphics Forum. Wiley Online Library (2014)
5. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. CVPR (2022)
6. Chang, M., Sharma, A., Kaess, M., Lucey, S.: Neural radiance field with lidar maps. In: CVPR (2023)
7. Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H.: Tensorf: Tensorial radiance fields. ARXIV (2022)
8. Chen, G., Wang, W.: A survey on 3d gaussian splatting. ARXIV (2024)
9. Cheng, Z., Sun, H., Takeuchi, M., Katto, J.: Learned image compression with discretized gaussian mixture likelihoods and attention modules. In: CVPR (2020)
10. De Queiroz, R.L., Chou, P.A.: Compression of 3d point clouds using a region-adaptive hierarchical transform. IEEE TIP (2016)
11. Dupont, E., Goliński, A., Alizadeh, M., Teh, Y.W., Doucet, A.: Coin: Compression with implicit neural representations. ICLR (2021)
12. Eden, M., Unser, M., Leonardi, R.: Polynomial representation of pictures. Signal Processing (1986)
13. Galić, I., Weickert, J., Welk, M., Bruhn, A., Belyaev, A., Seidel, H.P.: Image compression with anisotropic diffusion. Journal of Mathematical Imaging and Vision (2008)
14. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings (2010)
15. Goyal, V.K.: Theoretical foundations of transform coding. IEEE Signal Processing Magazine (2001)
16. Gray, R.M., Neuhoff, D.L.: Quantization. IEEE TIP (1998)
17. Gross, M., Pfister, H.: Point-based graphics. Elsevier (2011)
18. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE (1952)
19. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. TOG (Proc. of SIGGRAPH) (2023)
20. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. ARXIV (2014)
21. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction. TOG (2017)
22. Kobbelt, L., Botsch, M.: A survey of point-based techniques in computer graphics. Computers & Graphics (2004)
23. Kopanas, G., Philip, J., Leimkühler, T., Drettakis, G.: Point-based neural rendering with per-view optimization. In: Computer Graphics Forum. Wiley Online Library (2021)

24. Lassner, C., Zollhofer, M.: Pulsar: Efficient sphere-based neural rendering. In: CVPR (2021)
25. Li, R., Gao, H., Tancik, M., Kanazawa, A.: Nerfacc: Efficient sampling accelerates nerfs. ARXIV (2023)
26. Martel, J.N., Lindell, D.B., Lin, C.Z., Chan, E.R., Monteiro, M., Wetzstein, G.: Acorn: Adaptive coordinate networks for neural scene representation. ARXIV (2021)
27. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM (2021)
28. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. TOG (2022)
29. Ost, J., Laradji, I., Newell, A., Bahat, Y., Heide, F.: Neural point light fields. In: CVPR (2022)
30. Rakhimov, R., Ardelean, A.T., Lempitsky, V., Burnaev, E.: Npbg++: Accelerating neural point-based graphics. In: CVPR (2022)
31. Rho, D., Lee, B., Nam, S., Lee, J.C., Ko, J.H., Park, E.: Masked wavelet representation for compact neural radiance fields. In: CVPR (2023)
32. Song, Y., Wang, J., Wei, L.Y., Wang, W.: Vector regression functions for texture compression. TOG (2015)
33. Sridhar, S., Rhodin, H., Seidel, H.P., Oulasvirta, A., Theobalt, C.: Real-time hand tracking using a sum of anisotropic gaussians model. In: 2014 2nd International Conference on 3D Vision. IEEE (2014)
34. Stauffer, C., Grimson, W.E.L.: Adaptive background mixture models for real-time tracking. In: CVPR (1999)
35. Strümpler, Y., Postels, J., Yang, R., Gool, L.V., Tombari, F.: Implicit neural representations for image compression. In: Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVI. Springer (2022)
36. Sun, W., Trulls, E., Tseng, Y.C., Sambandam, S., Sharma, G., Tagliasacchi, A., Yi, K.M.: Pointnerf++: A multi-scale, point-based neural radiance field. ARXIV (2023)
37. Tagliasacchi, A., Mildenhall, B.: Volume rendering digest (for nerf). ARXIV (2022)
38. Takikawa, T., Evans, A., Tremblay, J., Müller, T., McGuire, M., Jacobson, A., Fidler, S.: Variable bitrate neural fields. In: ACM SIGGRAPH 2022 Conference Proceedings (2022)
39. 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: CVPR (2021)
40. 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 (2023)
41. Takikawa, T., Perel, O., Tsang, C.F., Loop, C., Litalien, J., Tremblay, J., Fidler, S., Shugrina, M.: Kaolin wisp: A pytorch library and engine for neural fields research. https://github.com/NVIDIAGameWorks/kaolin-wisp (2022)
42. Tang, D., Dou, M., Lincoln, P., Davidson, P., Guo, K., Taylor, J., Fanello, S., Keskin, C., Kowdle, A., Bouaziz, S., et al.: Real-time compression and streaming of 4D performances. TOG (2018)
43. Tang, D., Singh, S., Chou, P.A., Hane, C., Dou, M., Fanello, S., Taylor, J., Davidson, P., Guleryuz, O.G., Zhang, Y., et al.: Deep implicit volume compression. In: CVPR (2020)

44. Wallace, G.K.: The jpeg still picture compression standard. Commun. ACM (1991)
45. 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. ARXIV (2023)
46. 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. Computer Graphics Forum (2022)
47. Xu, Q., Xu, Z., Philip, J., Bi, S., Shu, Z., Sunkavalli, K., Neumann, U.: Point-nerf: Point-based neural radiance fields. In: CVPR (2022)
48. Yan, Z., Low, W.F., Chen, Y., Lee, G.H.: Multi-scale 3d gaussian splatting for anti-aliased rendering. ARXIV (2023)
49. Yang, Z., Gao, X., Zhou, W., Jiao, S., Zhang, Y., Jin, X.: Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. ARXIV (2023)
50. Yu, Z., Chen, A., Huang, B., Sattler, T., Geiger, A.: Mip-splatting: Alias-free 3d gaussian splatting. ARXIV (2023)
51. Zhang, Q., Baek, S.H., Rusinkiewicz, S., Heide, F.: Differentiable point-based radiance fields for efficient view synthesis. In: SIGGRAPH Asia 2022 Conference Papers (2022)
52. Zhang, Y., Huang, X., Ni, B., Li, T., Zhang, W.: Frequency-modulated point cloud rendering with easy editing. In: CVPR (2023)