Compact Neural Graphics Primitives with Learned Hash Probing

TOWAKI TAKIKAWA, NVIDIA, Canada and University of Toronto, Canada
THOMAS MÜLLER, NVIDIA, Switzerland
MERLIN NIMIER-DAVID, NVIDIA, Switzerland
ALEX EVANS, NVIDIA, United Kingdom
SANJA FIDLER, NVIDIA, Canada and University of Toronto, Canada
ALEC JACOBSON, University of Toronto, Canada and Adobe, Canada
ALEXANDER KELLER, NVIDIA, Germany

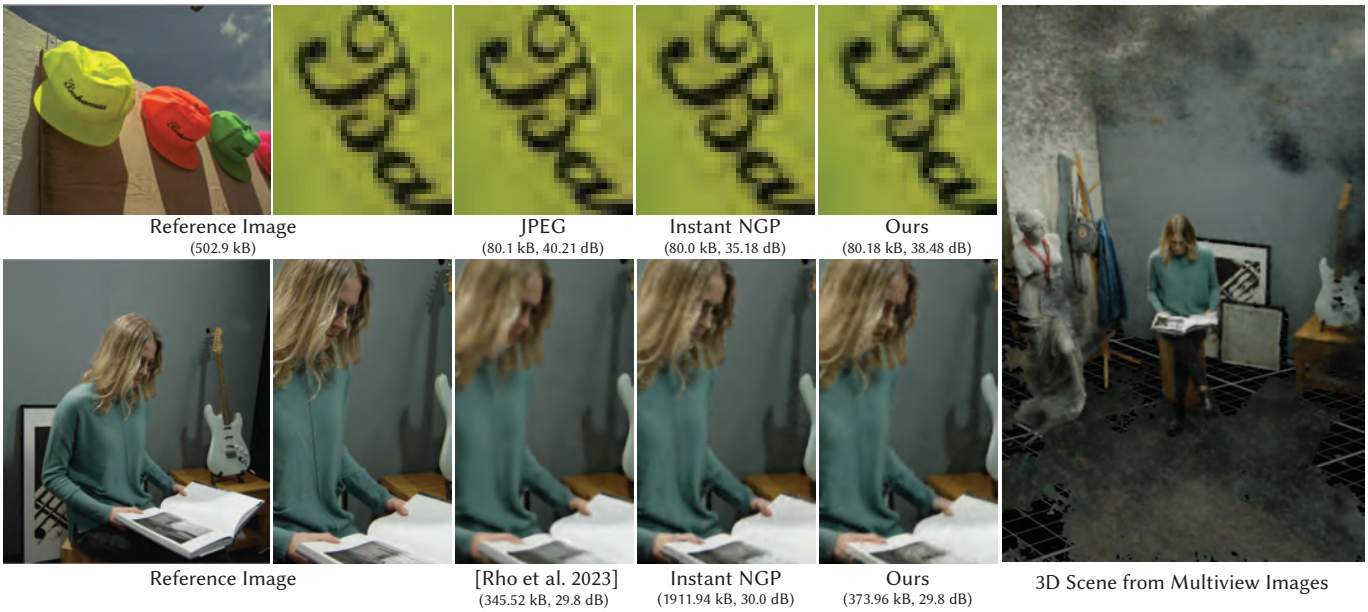

Fig. 1. *Compact neural graphics primitives* (Ours) have an inherently small size across a variety of use cases with automatically chosen hyperparameters. In contrast to similarly compressed representations like JPEG for images (top) and masked wavelet representations [Rho et al. 2023] for NeRFs [Mildenhall et al. 2020] (bottom), our representation neither uses quantization nor coding, and hence can be queried without a dedicated decompression step. This is essential for level of detail streaming and working-memory-constrained environments such as video game texture compression. The compression artifacts of our method are easy on the eye: there is less ringing than in JPEG and less blur than in Rho et al. [2023] (though more noise). Compact neural graphics primitives are also fast: training is only 1.2–2.6× slower (depending on compression settings) and inference is faster than Instant NGP [Müller et al. 2022] because our significantly reduced file size fits better into caches.

Neural graphics primitives are faster and achieve higher quality when their neural networks are augmented by spatial data structures that hold trainable features arranged in a grid. However, existing feature grids either come with a large memory footprint (dense or factorized grids, trees, and hash tables) or slow performance (index learning and vector quantization). In this paper, we show that a hash table with learned probes has neither disadvantage, resulting in a favorable combination of size and speed. Inference is faster than unprobed hash tables at equal quality while training is only 1.2–2.6× slower, significantly outperforming prior index learning approaches. We arrive at this formulation by casting all feature grids into a common framework: they each correspond to a lookup function that indexes into a table of feature vectors. In this framework, the lookup functions of existing data structures can be combined by simple arithmetic combinations of their indices, resulting in Pareto optimal compression and speed.

CCS Concepts: • **Computing methodologies** → **Graphics file formats**; **Image compression**; **Ray tracing**.

Additional Key Words and Phrases: Neural graphics primitives, compression.

Authors' addresses: Towaki Takikawa, NVIDIA, Canada and University of Toronto, Canada, tovacinni@gmail.com; Thomas Müller, NVIDIA, Zürich, Switzerland, tmueller@nvidia.com; Merlin Nimier-David, NVIDIA, Zürich, Switzerland, mnimierdavid@nvidia.com; Alex Evans, NVIDIA, London, United Kingdom, bluespoon@gmail.com; Sanja Fidler, NVIDIA, Toronto, Canada and University of Toronto, Canada, sfidler.com; Alec Jacobson, University of Toronto, Toronto, Canada and Adobe, Toronto, Canada, jacobson@cs.toronto.edu; Alexander Keller, NVIDIA, Berlin, Germany, akeller@nvidia.com.

0730-0301/2023/12-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Towaki Takikawa, Thomas Müller, Merlin Nimier-David, Alex Evans, Sanja Fidler, Alec Jacobson, and Alexander Keller. 2023. Compact Neural Graphics Primitives with Learned Hash Probing. *ACM Trans. Graph.* 1, 1 (December 2023), 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

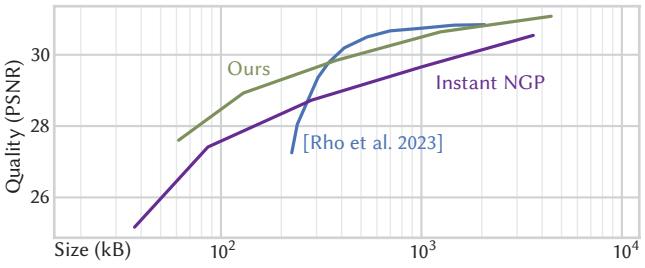

Fig. 2. Size vs. PSNR Pareto curves on the NeRF scene from Figure 1. Our work is able to outperform Instant NGP across the board and performs competitively with masked wavelet representations [Rho et al. 2023].

1 INTRODUCTION

The ever increasing demand for higher fidelity immersive experiences not only adds to the bandwidth requirements of existing multimedia formats (images, video, etc.), but also fosters in the use of higher-dimensional assets such as volumetric video and light field representations. This proliferation can be addressed by a unified compression scheme that efficiently represents both traditional and emerging multimedia content.

Neural graphics primitives (NGP) are a promising candidate to enable the seamless integration of old and new assets across applications. Representing images, shapes, volumetric and spatio-directional data, they facilitate novel view synthesis (NeRFs) [Mildenhall et al. 2020], generative modeling [Lin et al. 2023; Poole et al. 2023], and light caching [Müller et al. 2021], among more applications [Xie et al. 2022]. Particularly successful are those primitives that represent data by a *feature grid* that contains trained latent embeddings to be decoded by a multi-layer perceptron (MLP). Various such feature grids have been proposed, but they usually come with a substantial memory footprint [Chabra et al. 2020], even when factorized into low-rank representations [Chen et al. 2022] or represented in terms of sparse data structures [Fridovich-Keil et al. 2022; Liu et al. 2020; Müller et al. 2022; Takikawa et al. 2021; Yu et al. 2021]. In part, this limitation has been addressed by methods that *learn* to index feature vectors [Li et al. 2023; Takikawa et al. 2022a] and leverage sparse tree structures to avoid storing feature vectors in empty space. However, in these methods, index learning causes long training time and maintenance of sparse tree structures reduces flexibility.

Our work, Compact NGP, combines the speed of hash tables and the compactness of index learning by employing the latter as a means of collision detection by *learned probing*. We arrive at this combination by casting all feature grids into a common framework: they all correspond to indexing functions that map into a table of feature vectors. By simple arithmetic combinations of their indices, the data structures can be combined in novel ways that yield state-of-the-art compression vs. quality trade-offs. Mathematically, such arithmetic combinations amount to assigning the various data structures to *subsets* of the bits of the indexing function—thereby drastically reducing the cost of learned indexing that scales exponentially in the number of bits.

Our approach inherits the speed of hash tables while compressing much better—coming close to JPEG when representing images (Figure 1)—while remaining differentiable and without relying on a dedicated decompression scheme such as an entropy code. Compact NGP works across a wide range of user-controllable compression rates and provides streaming capabilities where partial results can be loaded in particularly bandwidth-constrained environments.

The paper is organized as follows: we review related work and its relation to indexing schemes in Section 2 before we introduce Compact NGP in Section 3. We demonstrate our method in Section 4 and discuss extensions, alternatives, and limitations in Section 5 ahead of the conclusion in the last section.

2 RELATED WORK AND PRELIMINARIES

In this article, we focus on lossy compression as it enables the highest compression rates for the multimedia under consideration. We begin by reviewing traditional techniques before studying the connection between (neural) feature grids and indexing functions.

2.1 Compression

Traditional compression. Lossy compression techniques typically employ transform coding [Goyal 2001] and quantization [Gray and Neuhoff 1998] followed by lossless entropy coding such as Huffman codes [1952]. On image and video content, linear transforms such as the discrete cosine [Ahmed et al. 1974] and wavelet [Haar 1909] transforms are applied to exploit coefficient sparsity and reduce the visual impact of quantization errors. Rather than transform coding, our work learns indices into a feature codebook, which is a form of vector quantization [Gray 1984; Wei and Levoy 2000], to find patterns in the data.

Texture compression relies on efficient random access to any part of the image without having to first decode the entire compressed representation. Most methods perform block-wise compression, packing small tiles of the texture into codes of fixed size [Beers et al. 1996; Reed 2012; Ström and Akenine-Möller 2005]. Although our approach is different, it similarly allows for random access queries without a decompression step, enabling its potential use for texture compression in real-time renderers where feature grids have already shown promise [Müller et al. 2022; Vaidyanathan et al. 2023].

Volumetric compression in computer graphics [Balsa Rodríguez et al. 2014] similarly uses block-based coding schemes [De Queiroz and Chou 2016; Tang et al. 2018, 2020; Wang et al. 2021]. Taking into account the often hierarchical structure of sparsity, subdivision-based spatial data structures such as trees additionally improve compression such as in OpenVDB [Museth 2021; Museth et al. 2019]. By contrast, our work combines index learning and hash tables that both do not rely on a subdivision scheme for sparsity.

Neural compression. In neural image compression, *auto-encoder approaches* use a neural network for transform coding [Ballé et al. 2020, 2018; Theis et al. 2017]. Other works use coordinate-based neural representations to fit and compress images as continuous

vector fields, some without feature grids [Dupont et al. 2022; Lindell et al. 2022; Song et al. 2015; Strümpler et al. 2022] and some with feature grids [Martel et al. 2021; Müller et al. 2022; Saragadam et al. 2022]. Although many of these works achieve a better equal-quality compression rate than JPEG [Wallace 1992] at low parameter counts, high parameter counts remain challenging. Our method is also a feature-grid and coordinate-based representation, yet performs competitively with JPEG across a wider range of qualities; see Figure 7.

Coordinate-based neural representations are additionally applicable to volumetric and spatio-directional data; most commonly NeRFs [Mildenhall et al. 2020]. Without feature grids, Bird et al. [2021] minimize the entropy of a multi-layer perceptron (MLP) and Lu et al. [2021] apply vector quantization directly to the MLP parameters. Such pure MLP methods usually have high computational cost and poor quality as compared to MLPs augmented by feature grids, so our work instead focuses on compressing feature grids while keeping the MLP sufficiently small to be fast.

2.2 Feature Grids in the Framework of Lookup Functions

Let us formalize feature grid methods in the following framework: they train a *feature codebook* $D_f \in \mathbb{R}^{N_f \times F}$ of N_f F-dimensional feature vectors that are associated with a conceptual grid in the d-dimensional application domain. The mapping from grid vertices $\mathbf{v} = (v_0, v_1, \ldots) \in \mathbb{Z}^d$ to feature vectors is established by a lookup function $f(\mathbf{v})$ that *indexes* into the codebook, denoted by $D_f[\cdot]$.[1]

Dense grids. The canonical feature grid is a dense Cartesian grid[2], visualized in Figure 3 **(a)**, that establishes a one-to-one correspondence of grid vertices to feature vectors, given for $d = 3$ as

$$f(\mathbf{v}) = D_f[v_0 + s_0 \cdot (v_1 + s_1 \cdot v_2)]\,, \tag{1}$$

where the scale $\mathbf{s} = (s_0, s_1, \ldots)$ defines the resolution of the grid. Dense grids cannot adapt to sparsity in the data which makes them undesirable in practice. For example, in 3D surface reconstruction the number of dense grid vertices is $O(n^3)$ while the surfaces to be reconstructed only intersect $O(n^2)$ cells. Therefore, practitioners either combine dense grids with classic sparsification methods such as transform coding [Isik et al. 2022] or they choose more sophisticated indexing schemes that will be discussed next.

k-plane methods. [Chan et al. 2022; Chen et al. 2022; Fridovich-Keil et al. 2023; Peng et al. 2020] project the dense grid along k sets of one or more axes as shown in Figure 3 **(b)**, and combine the resulting lower-dimensional (but still dense, usually planar) lookups arithmetically, e.g.

$$f(\mathbf{v}) = D_f[v_0 + s_0 \cdot v_1] \cdot D_f[s_0 \cdot s_1 + v_2] \cdot D_f[\ldots] + \ldots\,. \tag{2}$$

Special cases of this scheme are equivalent to tensor decompositions of the dense grid [Chen et al. 2022]. While k-planes ensure fewer than $O(n^d)$ parameters, they makes the strong assumption that sparsity in the data can be well explained by axis aligned projections

[1] Many methods maintain multiple codebooks at different resolutions, each with its own lookup function [Müller et al. 2022; Takikawa et al. 2022a, 2021], the values of which are combined before being fed to the MLP. Furthermore, most methods invoke the lookup functions at several grid vertices to compute continuous outputs by interpolation [Liu et al. 2020; Takikawa et al. 2021].

[2] Other tilings, such as permutohedral lattices [Rosu and Behnke 2023], are also possible.

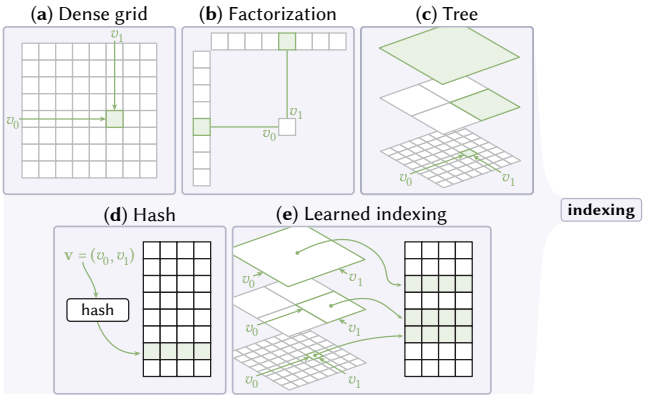

Fig. 3. Various indexing schemes mapping integer grid coordinates $\mathbf{v} = (v_0, v_1, \ldots)$ to feature vectors have been proposed, including **(a)** dense grids, **(b)** k-planes, **(c)** sparse grids and trees, **(d)** spatial hashing, and **(e)** learned indexing. Since each scheme ultimately computes an index into a codebook of feature vectors, the schemes can be combined by arithmetic operations on the indices they produce. Our method combines deterministic hashing and a learned indexing as visualized in Figure 4.

that are decoded by the MLP. In practice, this is not always the case, necessitating application-specific tricks such as bounding box cropping [Chen et al. 2022] or transform coding of the projected grids [Rho et al. 2023] for better compression.

Spatial hashing. Contrasting with the axis aligned parameter collisions of k-planes, spatial hashing [Teschner et al. 2003] distributes its collisions uniformly across lookups

$$f(\mathbf{v}) = D_f[\mathsf{hash}(\mathbf{v}) \bmod N_f]\,, \qquad \mathsf{hash}(\mathbf{v}) = \bigoplus_{i=0}^{d-1} v_i \cdot \pi_i\,, \tag{3}$$

where $\oplus$ is the binary XOR operation and π_i are large prime numbers (optionally, $\pi_0 = 1$). Well designed hash functions have the benefit that the lookups *always* uniformly cover the codebook D_f, regardless of the underlying shape of the data, permitting sparsity to be learned independently of the data and thus application [Müller et al. 2022]. But hashing also comes with the significant downside of "scrambling" the entries of the learned codebook D_f (now a hash table), precluding structure-dependent post processing such as generative modelling or transform coding.

Subdivision. Some applications [Chabra et al. 2020; Kim et al. 2022; Martel et al. 2021; Takikawa et al. 2021] construct a sparse hierarchical data structure such as a tree whose nodes hold indices into the feature codebook:

$$f(\mathbf{v}) = D_f[\mathtt{tree\_index}(\mathbf{v})]\,. \tag{4}$$

Unfortunately, many tasks are ill-suited to such a subdivision scheme, for example image compression where subdivision heuristics are difficult to design or 3D reconstruction where sparsity is unknown a priori and only emerges during optimization [Fridovich-Keil et al. 2022; Liu et al. 2020]. Furthermore, unlike the indexing schemes above, tree traversal involves cache-unfriendly pointer chasing and therefore incurs a non-negligible performance overhead.

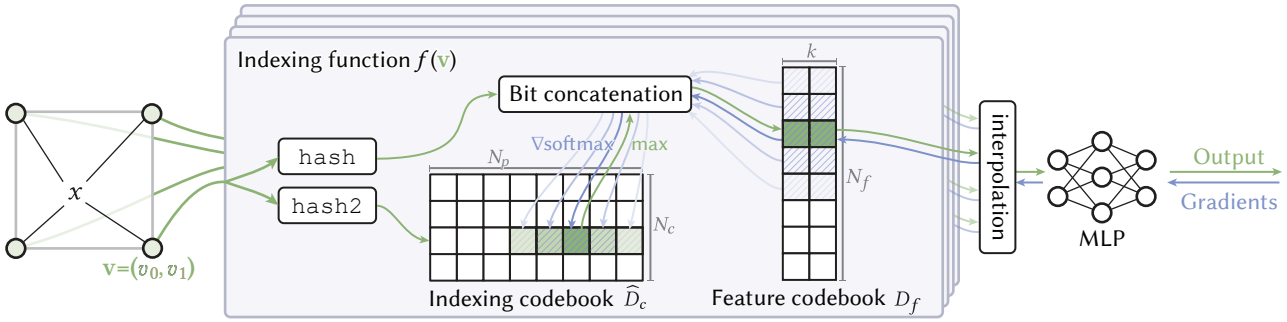

Fig. 4. Overview of Compact NGP. For a given input coordinate $\mathbf{x} \in \mathbb{R}^d$ (far left), we find its enclosing integer grid vertices $\mathbf{v} \in \mathbb{Z}^d$ and apply our indexing function $f(\mathbf{v})$ to each one. The most significant bits of the index are computed by a spatial hash (hash) and the least significant bits by looking up a row of N_p confidence values from an indexing codebook $\widehat{D}_c$ that is in turn indexed by an auxiliary spatial hash (hash2), and then picking the index with maximal confidence (green arrow). Bitwise concatenation of the two indices yields an index for looking up from the feature codebook D_f, which is subsequently d-linearly interpolated per $\mathbf{x}$ and fed into an MLP. For optimization, we propagate gradients as if the indexing codebook used a softmax instead of a hard maximum, i.e. we use a "straight-through" estimator [Bengio et al. 2013]. In practice, after each training step, we bake this $\log_2 N_p$-bit indices of the maximal values in each row of $\widehat{D}_c$ into an auxiliary indexing codebook D_c that is both compact and allows for more efficient forward evaluation of the model.

Learning the indexing function. Rather than designing the indexing function by hand, it can also be learned from data [Li et al. 2023; Takikawa et al. 2022a]. In these methods, an *index codebook* $D_c \in \mathbb{N}^{N_c}$ holds the lookup indices into the feature codebook and is in turn indexed by one of the methods above. For example, VQAD [Takikawa et al. 2022a] has the lookup function

$$f(\mathbf{v}) = D_f\big[D_c[\texttt{tree\_index}(\mathbf{v})]\big], \tag{5}$$

where D_c is trained by softmax-weighted[3] indexing into all entries of D_f. This is expensive even for moderately sized feature codebooks (and prohibitive for large ones) but has no inference overhead and results in over 10× better compression than spatial hashing. The compression is not quite as effective as a combination of k-plane methods with transform coding [Rho et al. 2023] but has the advantage that it can be cheaply queried without in-memory decompression to a larger representation.

Combining methods. Using the framework of lookup functions we can relate our method to previous work: we combine learned indexing with spatial hashing by arithmetically combining their indices. The most significant bits of our index come from Instant NGP's hash encoding [Müller et al. 2022] and the least significant bits are learned by a variation of VQAD [Takikawa et al. 2022a]. Thus, our method performs *learned probing* for collision resolution and information reuse in analogy to classic hash table probing methods [Knuth 1963]. This will be motivated and explained in the next section.

3 METHOD

Our goal is to minimize the number of parameters θ and Φ of a multi-layer perceptron $m(y;\Phi)$ and its corresponding input encoding $y = \psi(x;\theta)$ without incurring a significant speed penalty. Furthermore, we want to remain application agnostic and therefore avoid structural modifications such as tree subdivision and transform codings that may depend on application-specific heuristics.

[3]The softmax function $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ is defined as $\sigma_i(\mathbf{x}) = e^{x_i} / \sum_j e^{x_j}$.

Table 1. Hyperparameters of our method and recommended ranges. We inherit most parameters from Instant NGP [Müller et al. 2022] and introduce two additional ones pertaining to the index codebook. Gray parameters are unaffected by our method and therefore set to the same values as in Instant NGP; the choice of remaining parameters is explained in Section 3.

Source	*Parameter*	*Symbol*	*Value*
new in our method	Index probing range	N_p	2^1 to 2^4
	Index codebook size	N_c	2^{10} to 2^{24}
inherited from Instant NGP	Feature codebook size	N_f	2^6 to 2^{12}
	Feature dimensionality	F	2
	Number of levels	L	16
	Coarsest resolution	$N_{\min}$	16
	Finest resolution	$N_{\max}$	512 to 524288
	Num. hidden neurons	N_{neurons}	64

Hence, we base our method on Instant NGP's multi-resolution hash encoding [Müller et al. 2022] and generalize its indexing function, Eq. (3), by introducing learned probing. In our lookup function, the spatial hash produces the most significant bits of the index, while the remaining user-configurable $\log_2 N_p$ least significant bits are learned within an auxiliary index codebook $D_c \in \{0, 1, \ldots, N_p - 1\}^{N_c}$ that is in turn indexed by a second spatial hash (one that uses different prime numbers from the first). The lookup function is illustrated in Figure 4 and given for a single grid vertex by

$$f(\mathbf{v}) = D_f\big[\big(N_p \cdot \texttt{hash}(\mathbf{v})\big) \bmod N_f + D_c[\texttt{hash2}(\mathbf{v})]\big]. \tag{6}$$

Intuitively, the index codebook D_c, sparsified by the second spatial hash, learns to *probe* the feature codebook over N_p values for collision resolution and information re-use. The index codebook's size N_c as well as its probing range N_p are hyperparameters of our method that extend those inherited from Instant NGP; see Table 1.

Following Takikawa et al. [2022a], we maintain two versions of the index codebook: one for training $\widehat{D}_c \in \mathbb{R}^{N_c \times N_p}$ that holds confidence values for each of the N_p features in the probing range, and one for inference $D_c \in \{0, 1, \ldots, N_p - 1\}^{N_c}$ that holds $\log_2 N_p$-bit

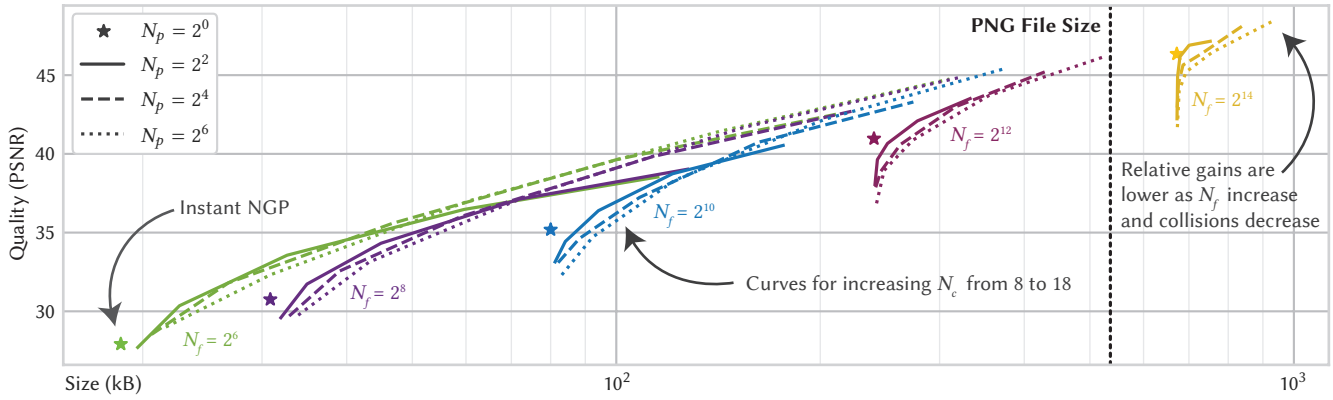

Fig. 5. PSNR vs. file size for varying hyperparameters in compressing the Kodak image dataset. We sweep three parameters: the feature codebook size N_f (colors), the index codebook size N_c (curves ranging from 2^{12} to 2^{20}), and the probing range N_p (dashing and dotting). A value of $N_p = 1$ corresponds to Instant NGP (shown as ★) and has no curve because it is invariant under N_c. We see that the optimal curve at a given file size N has a feature codebook size (same-colored ★) of roughly $N_f = 1/3N$ and index codebook size $N_c = 2/3N$. Small probing ranges (solid curves) are sufficient for good compression—in-fact optimal for small values of N_c (left side of curves)—but larger probing ranges (dashed and dotted curves) yield further small improvements for large values of N_c (right side of curves) at the cost of increased training time.

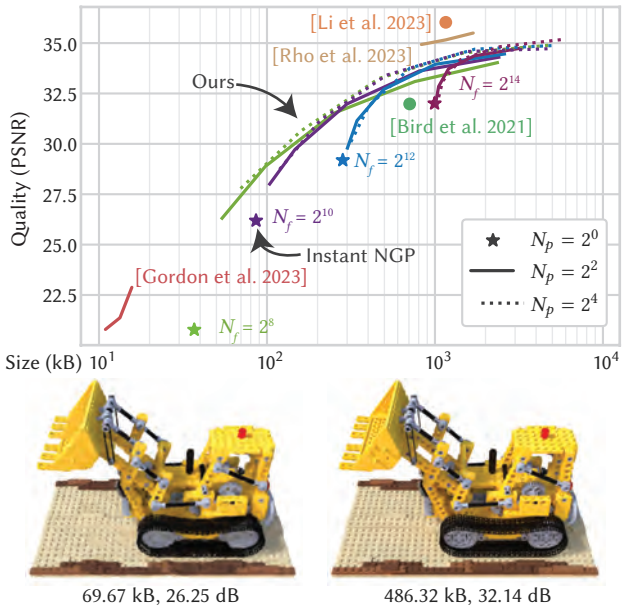

Fig. 6. PSNR vs. file size for varying hyperparameters in compressing the NeRF Lego digger. The layout is the same as Figure 5. We also show rendered images of our compressed representation at two quality settings.

integer indices corresponding to the probe offset with largest confidence. Compared to Instant NGP, the only inference-time overhead is the index lookup from D_c. Furthermore, our smaller parameter count leads to improved cache utilization; we hence achieve similar and in some cases better inference performance as shown in Table 2.

Training. In the forward pass we use D_c to look up the feature with largest confidence and in the backward pass we distribute gradients into *all* features within the probing range, weighted by the softmax of their confidence values from $\widehat{D}_c$ (see Figure 4). This strategy of combining a discrete decision in the forward pass with continuous gradients in the backward pass is also known as a "straight-through" estimator that helps to learn hard non-linearities [Bengio et al. 2013].

Table 2. Training and inference time overheads of Compact NGP. Training times are measured for an iteration of training on the NeRF Lego digger dataset. Inference times are for 2^{18} lookups on a single multiresolution level. The relative training overhead (denoted with $n\times$) is measured with respect to Instant NGP ($N_f = 2^{16}$), ranging from 1.2–2.6×. The largest impact on speed has the probing range N_p, whereas N_c (shown) and N_f (see Müller et al. [2022]) only have a weak effect.

Method	N_f	N_c	N_p	*Training time per iteration*	*Inference time for 2^{18} lookups*	*Quality (PSNR dB)*
I NGP	2^{16}	n/a	2^0	5.4 ms	28.7μs	33.60 dB
	2^{14}	n/a	2^0	5.1 ms	13.7μs	32.00 dB
	2^8	n/a	2^0	4.5 ms	9.8μs	19.04 dB
Ours	2^8	2^{12}	2^2	6.8ms (1.26×)	10.1μs	26.25 dB
	2^8	2^{16}	2^2	6.8 ms (1.26×)	10.1μs	31.58 dB
	2^8	2^{12}	2^3	8.3 ms (1.53×)	10.1μs	27.13 dB
	2^8	2^{16}	2^3	8.5 ms (1.57×)	10.2μs	32.58 dB
	2^8	2^{12}	2^4	12.7 ms (2.35×)	10.2μs	27.67 dB
	2^8	2^{16}	2^4	14.1 ms (2.61×)	10.2μs	33.24 dB

By keeping the learned number of bits $\log_2 N_p$ small, we limit the number of features and confidence values that need to be loaded in the backward pass. And since the learned bits are the least significant ones, their corresponding features lie adjacent in memory, usually located in the same cache line and thereby incurring only a moderate training overhead of 1.2–2.6× (see Table 2) while realizing compression rates on par with the orders of magnitude slower VQAD [Takikawa et al. 2022a].

Selecting hyperparameters. Recall that our method inherits its hyperparameters from Instant NGP and introduces two new ones: the index codebook size N_c and its probing range N_p; see Table 1 for a complete list. To find quality-maximizing parameters, we recommend the following scheme inspired by Figures 5 and 6, which we use in all our following results. First, set $N_c = 1$ and $N_p = 1$, turning the method into Instant NGP as indicated by ★ in the figure. Second,

set the feature codebook size N_f according to the desired lower bound on the compressed size. Third, double N_c until a reasonable maximum value (usually $N_c = 2^{16}$). Lastly, if even higher quality is desired, double N_f. The remaining parameter N_p can be tuned to taste, as this parameter governs how expensive the training is, but a higher value tends to produce slightly better Pareto tradeoffs between size and quality.

4 RESULTS

We have implemented our algorithm on top of the version of Instant NGP in the PyTorch-based Kaolin Wisp library [Takikawa et al. 2022b]. Computationally expensive operations like sparse grid ray tracing and feature grid lookups of both Instant NGP and our method are accelerated by custom CUDA kernels called from PyTorch. All results are measured on an NVIDIA RTX 6000 Ada GPU.

Performance. Table 2 lists inference and training times of our method on the NeRF Lego digger from Figure 6. Compared to Instant NGP, our 1.2–2.6× training overhead scales with the probing range N_p, confirming the analysis in Section 3 and exposing a trade-off between training speed and compression to the user. Since the compression benefit of larger probing ranges quickly falls off, we cap $N_p \leq 2^4$ in all our experiments, manifesting the worst-case overhead of 2.6×. An important performance consideration for training is the accumulation of gradients into the feature codebook D_f. Since our method uses very small codebooks $N_f \in [2^6, 2^{12}]$, special care must be taken on massively parallel processors, such as GPUs, to first accumulate gradients in threadblock-local memory before broadcasting them into RAM. This avoids contention that would otherwise make training ~7× slower.

Table 2 also demonstrates that Compact NGP has *faster* inference than Instant NGP at roughly equal quality settings. This is because our method has a much smaller size ($N_f = 2^{16}$ vs. $N_f = 2^8, N_c = 2^{16}$) and thereby fits better into caches. The only inference overhead of our method is the additional index lookup from D_c, which we find negligible (0.4μs at $N_f = 2^8$).

Image compression. Figure 7 shows the quality vs. size tradeoff of our method on the Kodak image dataset, which consists of 24 images of 768×512 pixels. The figure also shows JPEG as well as prior coordinate MLP methods. On this dataset, our method performs close to JPEG at small file sizes and worse at larger ones. At small file sizes, our representation is dominated by floating point parameters like the MLP and the feature codebook, causing competing methods that apply quantization on top of pure MLPs [Dupont et al. 2021; Strümpler et al. 2022] to compress better. However, these methods do not scale to higher quality targets (~35dB and above) as it is difficult to train pure MLPs to such qualities. To demonstrate the better scaling of our method, we investigate a much larger 8000×8000 image of Pluto in Figure 8 on which we outperform both JPEG on most practical sizes (~megabyte) and prior neural large-scale methods (Instant NGP [Müller et al. 2022] and ACORN [Martel et al. 2021]) at high quality settings. Our method is also evaluated against texture compression methods in Table 6.

NeRF compression. We evaluate NeRF compression on a real-world scene in Figures 1 and 2 as well as synthetic scenes [Mildenhall et al. 2020] in Figure 6 (Lego) and Table 5 (full dataset). We compare with several contemporary NeRF compression techniques that are mostly based on TensoRF [Chen et al. 2022]. We report numbers from the original papers where available. For the real world scene, we ran masked wavelets [Rho et al. 2023] as a strong and recent baseline. In both scenes, we outperform Instant NGP in terms of quality vs.

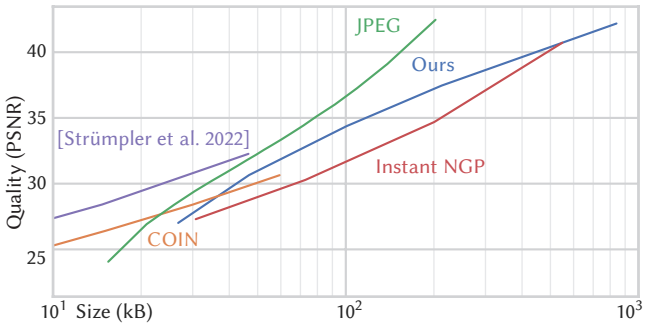

Fig. 7. PSNR vs. file size on the Kodak image dataset using parameters $N_f = 2^6$ and $N_p = 2^4$ and varying N_c (blue curve ranging from 2^{12} to 2^{20}). On this dataset, our method performs close to JPEG at small file sizes and worse at larger ones. At small file sizes, our representation is dominated by floating point parameters like the MLP and the feature codebook. Competing methods that quantize pure MLPs perform better in this regime [Dupont et al. 2021; Strümpler et al. 2022], whereas we omit quantization for simplicity and flexibility. At visually pleasant targets (~35dB and above) these prior works do not scale as it is difficult to train pure MLPs to such qualities.

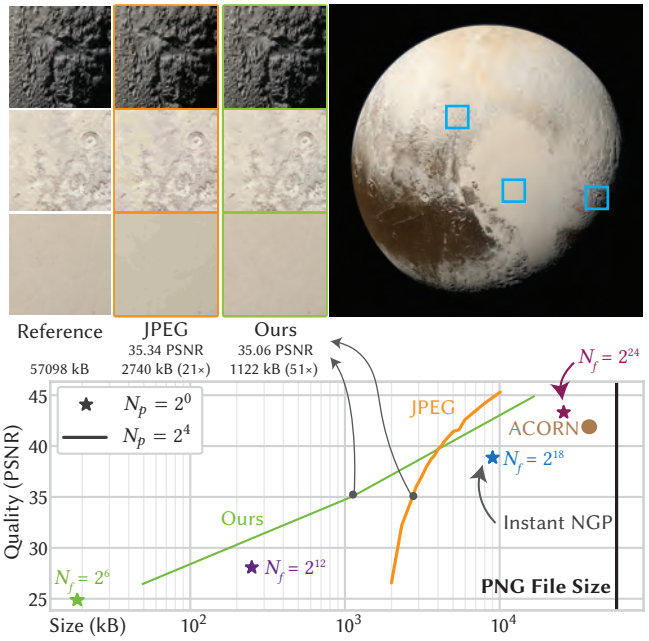

Fig. 8. We fit Compact NGP to the 8000×8000px Pluto image using parameters $N_f = 2^6$ and $N_p = 2^4$ and varying N_c (green curve ranging from 2^{12} to 2^{24}). We show that we are able to outperform JPEG on a wide range of quality levels. The qualitative comparisons at *equal size* (insets) show the visual artifacts exhibited by different methods: while JPEG has color quantization arfitacts, ours appears slightly blurred.

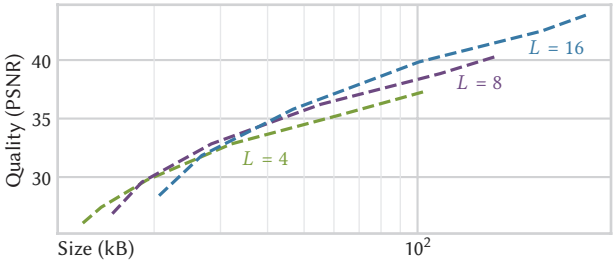

Fig. 9. Impact of the number of multiresolution levels L on PSNR vs. size. We use the parameters $N_f = 2^6$ and $N_p = 2^4$ while varying N_c (curve ranging from 2^{12} to 2^{20}) and L on the image compression task from Figure 1. The default value $L = 16$ (inherited from Instant NGP) performs well for a large range of sizes, particularly in the hundreds of kB range that is most practical. Yet, a lower number of levels results in a better Pareto curve at smaller sizes that could be used if one wanted to compete with MLP based image compression techniques; cf. Figure 7.

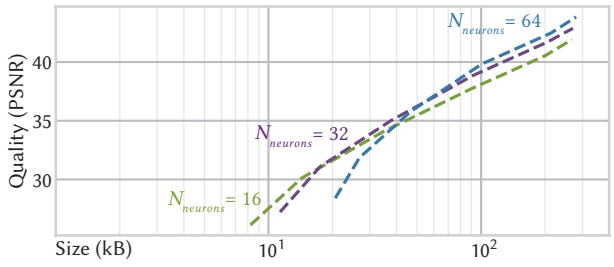

Fig. 10. Impact of the MLP width N_{neurons} on PSNR vs. size. The parameter sweeps over N_f, N_p, and N_c are the same as Figure 9. A similar conclusion can be drawn: the default value $N_{\text{neurons}} = 64$ (inherited from Instant NGP) performs well at practical sizes, whereas a better Pareto front can be achieved at smaller sizes.

size. On the synthetic scene (Figure 6), our Pareto front lies slightly below the specialized baselines that use scalar quantization and coding, and in the real-world scene our Pareto front is competitive (Figure 2) despite our work requiring neither.

The zoom-ins in Figure 1 reveal distinct visual artifacts of the different methods, even though their PSNR is the same. Masked wavelets [Rho et al. 2023] produce blurrier results whereas Compact NGP yields a sharper reconstruction with high frequency noise similar to that of Instant NGP.

Additional hyperparameter ablations. Aside from the feature codebook size N_f, we inherit the default hyperparameters of Instant NGP for a apples-to-apples comparisons. To verify that these defaults are reasonable, we sweep the number of multiresolution levels L in Figure 9 and the number of hidden neurons N_{neurons} in Figure 10. The default values $L = 16$ and $N_{\text{neurons}} = 64$ perform well for a large range of sizes, particularly in the hundreds of kB range that is most practical. Yet, lower values produce better Pareto frontiers at very small file sizes that could be used if one wanted to compete with MLP based image compression techniques; cf. Figure 7. However, we believe that the hundreds of kB range is more relevant in practice and we therefore stick to the default values for simplicity.

5 DISCUSSION AND FUTURE WORK

Compact NGP has been designed with content distribution in mind where the compression overhead is amortized and decoding on user equipment must be low cost, low power, and multi-scale for graceful degradation in bandwidth-constrained environments. As an example, NeRFs may be broadcasted and decoded on large numbers of end-user devices, possibly in real-time to enable live streaming video NeRFs. More generally, (learnable) compression codecs will enable the next generation of immersive content of which live streaming of NeRFs are just an example and other applications, like video game texture compression and volumetric video, being right around the corner.

Quality and compression artifacts. Beyond measuring PSNR, it is worth studying the qualitative appearance of compression artifacts with our method. Compared to JPEG, our method appears to produce less ringing at the cost of a small amount of additional blur, whereas in NeRF our methods looks similar to Instant NGP: sharp, but with high-frequency noise. This is in contrast to Rho et al. [2023], who produce a smoother yet blurry reconstruction; see Figure 1. Since we measure error in terms of PSNR, which is based on the $\mathcal{L}_2$ error, blurry results yield lower error than the human visual system might expect [Zhao et al. 2016].

From `float` *to* `int`. Our method shifts the storage cost from being `float`-dominated to `int`-dominated. In the settings we test in, we see that this tradeoff is favorable, particularly because our integers have only $\log_2 N_p$ bits—many fewer than than even 16-bit half precision floats. We have additionally investigated several methods that reduce the entropy of our learned indices (e.g. through additional terms in the loss), coupled to entropy coding, but so far with little success that does not warrant forfeiture of random access lookups. Alternatively, data-adaptive quantization of floats may reduce the bit count further than using an index codebook, but better training strategies are required to this end. We believe that further research into data-adaptive `float` quantization as well as `int` entropy minimization will be fruitful.

Entropy coding. Our method was inspired by a method that has spatial locality built-in [Takikawa et al. 2022a] (i.e. the index codebook represented by a tree). Such spatial locality could be exploited by an entropy coder much better than the spatial hash table that we use. We chose spatial hashing for being agnostic of the application [Müller et al. 2022]—and it performs competitively with transform and entropy coded prior work nonetheless—but if future research could devise local data structures that have the same flexibility and performance as hash tables, it will likely be worthwhile to utilize those instead of hashing.

Alternatives to straight-through estimators. In our work we use the softmax function along with the straight-through estimator to learn indexing. While effective, this can be computationally expensive for large indexing ranges as this requires backpropagation on all possible indices. As such, it may be worthwhile to explore the various sparse [Laha et al. 2018; Martins and Astudillo 2016; Peters et al. 2019] and stochastic [Lee et al. 2018; Paulus et al. 2020] variants have been proposed in the literature. Proximity-based indexing such

Table 3. Quantiative results on the full synthetic dataset from Mildenhall et al. [2020], showing a near-quality (PSNR) comparison between Instant NGP and our work. We see that we are able to achieve similar quality across the entire dataset with a 2.8× more compact representation.

Method	N_f	N_c	N_p	*Mic*	*Ficus*	*Chair*	*Hotdog*	*Materials*	*Drums*	*Ship*	*Lego*	*avg.*	*Size (kB)*
I NGP	2^{14}	n/a	2^0	35.08	30.99	32.59	34.99	28.73	25.36	27.71	32.03	30.93	1000 kB
Ours	2^8	2^3	2^{16}	33.88	32.08	32.05	34.26	28.32	24.71	27.71	32.31	30.66	357 kB

Table 4. Quantiative results on texture compression on the *Paving Stones* texture set, retrieved from `https://ambientcg.com`, showing the tradeoff between quality (PSNR) and size (kB) for different methods. We compare against traditional texture compression baselines (BC) as well as recent neural baselines (NTC [Vaidyanathan et al. 2023]). We borrow the results from Vaidyanathan et al. [2023]. Although our work does not outperform NTC, which uses a specialized architecture for textures with quantization, we are still able to outperform BC and Instant NGP at similar size. We only report average across all channels for BC as that was the only data available, and compare against the NTC results without mipmaps (which increase quality) for fair comparison.

Method	*Quantization*	N_f	N_c	N_p	*Diffuse*	*Normal*	*Roughness*	*AO*	*Displacement*	*avg.*	*Size (kB)*
I NGP		2^{16}	n/a	n/a	21.58	22.32	26.79	27.72	35.62	24.75	3761 kB
I NGP		2^{14}	n/a	n/a	19.91	20.51	26.61	25.56	30.07	22.61	1049 kB
BC	✓	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	23.25	3500 kB
NTC	✓	n/a	n/a	n/a	26.10	27.17	29.37	31.22	40.59	29.00	3360 kB
Ours		2^{10}	2^{20}	2^3	24.02	25.00	27.90	29.94	36.18	26.69	3494 kB
Ours		2^8	2^{18}	2^3	21.55	22.61	26.94	27.43	33.74	24.51	1173 kB

as locality-sensitive hashing and the nearest-neighbour queries used in VQ-VAE [Van Den Oord et al. 2017] may be relevant as well.

6 CONCLUSION

We propose to view feature grids and their associated neural graphics primitives through a common lens: a unifying framework of lookup functions. Within this framework it becomes simple to mix methods in novel ways, such as our Compact NGP that augments efficient hash table lookups with low-overhead learned probing. The result is a state-of-the-art combination of compression and performance while remaining agnostic to the graphics application in question. Compact NGP has been designed with real-world use cases in mind where random access decompression, level of detail streaming, and high performance are all crucial (both in training and inference). As such, we are eager to investigate its future use in streaming applications, video game texture compression, live-training as in radiance caching, and many more.

ACKNOWLEDGMENTS

The Lego Bulldozer scene of Figure 6 was created by Blendswap user Heinzelnisse. The Pluto image of Figure 8 was created by NASA/Johns Hopkins University Applied Physics Laboratory/Southwest Research Institute/Alex Parker. We thank David Luebke, Karthik Vaidyanathan, and Marco Salvi for useful discussions throughout the project.

REFERENCES

Nasir Ahmed, T. Natarajan, and Kamisetty R. Rao. 1974. Discrete cosine transform. *IEEE transactions on Computers* 100, 1 (1974), 90–93.

Johannes Ballé, Philip A. Chou, David Minnen, Saurabh Singh, Nick Johnston, Eirikur Agustsson, Sung Jin Hwang, and George Toderici. 2020. Nonlinear transform coding. *IEEE Journal of Selected Topics in Signal Processing* 15, 2 (2020), 339–353.

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. 2018. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rkcQFMZRb

Marcos Balsa Rodríguez, Enrico Gobbetti, Jose Antonio Iglesias Guitian, Maxim Makhinya, Fabio Marton, Renato Pajarola, and Susanne K. Suter. 2014. State-of-the-art in compressed GPU-based direct volume rendering. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 77–100.

Andrew C. Beers, Maneesh Agrawala, and Navin Chaddha. 1996. Rendering from compressed textures. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. 373–378.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

Thomas Bird, Johannes Ballé, Saurabh Singh, and Philip A. Chou. 2021. 3D Scene Compression through Entropy Penalized Neural Representation Functions. In *2021 Picture Coding Symposium (PCS)*. 1–5. https://doi.org/10.1109/PCS50896.2021.9477505

Rohan Chabra, Jan E. Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. 2020. Deep local shapes: Learning local SDF priors for detailed 3D reconstruction. In *ECCV*. Springer, 608–625.

Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. 2022. Efficient Geometry-aware 3D Generative Adversarial Networks. In *CVPR*.

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

Ricardo L. De Queiroz and Philip A. Chou. 2016. Compression of 3D point clouds using a region-adaptive hierarchical transform. *IEEE Transactions on Image Processing* 25, 8 (2016), 3947–3956.

Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. 2021. COIN: COmpression with Implicit Neural representations. *ICLR 2021 Neural Compression Workshop Spotlight, arXiv preprint arXiv:2103.03123* (2021).

Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Golinski, Yee Whye Teh, and Arnaud Doucet. 2022. COIN++: Neural compression across modalities. *Transactions on Machine Learning Research* 2022, 11 (2022).

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

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields without Neural Networks. In *CVPR*.

Cameron Gordon, Shin-Fang Chng, Lachlan MacDonald, and Simon Lucey. 2023. On Quantizing Implicit Neural Representations. In *Proceedings of the IEEE/CVF Winter*

Conference on Applications of Computer Vision. 341–350.

Vivek K. Goyal. 2001. Theoretical foundations of transform coding. *IEEE Signal Processing Magazine* 18, 5 (2001), 9–21.

Robert M. Gray. 1984. Vector quantization. *IEEE ASSP Magazine* 1, 2 (1984), 4–29.

Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE Transactions on Information Theory* 44, 6 (1998), 2325–2383.

Alfred Haar. 1909. *Zur Theorie der orthogonalen Funktionensysteme*. Georg-August-Universität, Göttingen.

David A. Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.

Berivan Isik, Philip A. Chou, Sung Jin Hwang, Nick Johnston, and George Toderici. 2022. LVAC: Learned Volumetric Attribute Compression for Point Clouds using Coordinate Based Networks. *Frontiers in Signal Processing* 2 (2022). https://doi.org/10.3389/frsip.2022.1008812

Doyub Kim, Minjae Lee, and Ken Museth. 2022. NeuralVDB: High-resolution Sparse Volume Representation using Hierarchical Neural Networks. (2022). https://doi.org/10.48550/arXiv.2208.04448

Donald Knuth. 1963. Notes on "Open" Addressing. https://web.archive.org/web/20160303225949/http://algo.inria.fr/AofA/Research/11-97.html.

Anirban Laha, Saneem Ahmed Chemmengath, Priyanka Agrawal, Mitesh Khapra, Karthik Sankaranarayanan, and Harish G. Ramaswamy. 2018. On Controllable Sparse Alternatives to Softmax. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/6a4d5952d4c018a1c1af9fa590a10dda-Paper.pdf

Hae Beom Lee, Juho Lee, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. 2018. DropMax: Adaptive Variational Softmax. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/389bc7bb1e1c2a5e7e147703232a88f6-Paper.pdf

Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Liefeng Bo. 2023. Compressing Volumetric Radiance Fields to 1 MB. (June 2023), 4222–4231.

Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. 2023. Magic3D: High-Resolution Text-to-3D Content Creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 300–309.

David B. Lindell, Dave Van Veen, Jeong Joon Park, and Gordon Wetzstein. 2022. Bacon: Band-limited Coordinate Networks for Multiscale Scene Representation. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 16231–16241. https://doi.org/10.1109/CVPR52688.2022.01577

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 15651–15663. https://proceedings.neurips.cc/paper_files/paper/2020/file/b4b758962f17808746e9bb832a6fa4b8-Paper.pdf

Yuzhe Lu, Kairong Jiang, Joshua A. Levine, and Matthew Berger. 2021. Compressive Neural Representations of Volumetric Scalar Fields. *Computer Graphics Forum* 40, 3 (2021), 135–146. https://doi.org/10.1111/cgf.14295

Julien N.P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. 2021. ACORN: Adaptive Coordinate Networks for Neural Representation. *ACM Trans. Graph. (SIGGRAPH)* (2021).

André F. T. Martins and Ramón F. Astudillo. 2016. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (New York, NY, USA) *(ICML'16)*. JMLR.org, 1614–1623.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*. Springer, 405–421.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time Neural Radiance Caching for Path Tracing. *ACM Trans. Graph.* 40, 4, Article 36 (Aug. 2021), 16 pages. https://doi.org/10.1145/3450626.3459812

Ken Museth. 2021. NanoVDB: A GPU-friendly and portable VDB data structure for real-time rendering and simulation. In *ACM SIGGRAPH 2021 Talks*. 1–2.

Ken Museth, Nick Avramoussis, and Dan Bailey. 2019. OpenVDB. In *ACM SIGGRAPH 2019 Courses*. 1–56.

Max Paulus, Dami Choi, Daniel Tarlow, Andreas Krause, and Chris J. Maddison. 2020. Gradient Estimation with Stochastic Softmax Tricks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 5691–5704. https://proceedings.neurips.cc/paper_files/paper/2020/file/3df80af53dce8435cf9ad6c3e7a403fd-Paper.pdf

Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. 2020. Convolutional Occupancy Networks. In *European Conference on Computer Vision (ECCV)*.

Ben Peters, Vlad Niculae, and André F. T. Martins. 2019. Sparse Sequence-to-Sequence Models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 1504–1519. https://doi.org/10.18653/v1/P19-1146

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. 2023. DreamFusion: Text-to-3D using 2D Diffusion. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=FjNys5c7VyY

Nathan Reed. 2012. Understanding BCn Texture Compression Formats. https://www.reedbeta.com/blog/understanding-bcn-texture-compression-formats/. Online; accessed 24 January 2023.

Daniel Rho, Byeonghyeon Lee, Seungtae Nam, Joo Chan Lee, Jong Hwan Ko, and Eunbyung Park. 2023. Masked Wavelet Representation for Compact Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 20680–20690.

Radu Alexandru Rosu and Sven Behnke. 2023. PermutoSDF: Fast Multi-View Reconstruction with Implicit Surfaces using Permutohedral Lattices. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Vishwanath Saragadam, Jasper Tan, Guha Balakrishnan, Richard G. Baraniuk, and Ashok Veeraraghavan. 2022. MINER: Multiscale Implicit Neural Representation. In *Computer Vision – ECCV 2022*, Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (Eds.). Springer Nature Switzerland, Cham, 318–333.

Ying Song, Jiaping Wang, Li-Yi Wei, and Wencheng Wang. 2015. Vector regression functions for texture compression. *ACM Transactions on Graphics (TOG)* 35, 1 (2015), 1–10.

Jacob Ström and Tomas Akenine-Möller. 2005. iPACKMAN: High-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. 63–70.

Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. 2022. 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, 74–91.

Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. 2022a. Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings*. 1–9.

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *CVPR*. 11358–11367.

Towaki Takikawa, Or Perel, Clement Fuji Tsang, Charles Loop, Joey Litalien, Jonathan Tremblay, Sanja Fidler, and Maria Shugrina. 2022b. Kaolin Wisp: A PyTorch library and engine for neural fields research.

Danhang Tang, Mingsong Dou, Peter Lincoln, Philip Davidson, Kaiwen Guo, Jonathan Taylor, Sean Fanello, Cem Keskin, Adarsh Kowdle, Sofien Bouaziz, et al. 2018. Real-time compression and streaming of 4D performances. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–11.

Danhang Tang, Saurabh Singh, Philip A. Chou, Christian Hane, Mingsong Dou, Sean Fanello, Jonathan Taylor, Philip Davidson, Onur G. Guleryuz, Yinda Zhang, et al. 2020. Deep implicit volume compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1293–1303.

Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomeranets, and Markus Gross. 2003. Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Proceedings of VMV'03, Munich, Germany*. 47–54.

Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. 2017. Lossy Image Compression with Compressive Autoencoders. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJiNwv9gg

Karthik Vaidyanathan, Marco Salvi, Bartlomiej Wronski, Tomas Akenine-Möller, Pontus Ebelin, and Aaron Lefohn. 2023. Random-Access Neural Compression of Material Textures. In *Proceedings of SIGGRAPH*.

Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. *Advances in neural information processing systems* 30 (2017).

Gregory K. Wallace. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics* 38, 1 (1992), xviii–xxxiv.

Jianqiang Wang, Hao Zhu, Haojie Liu, and Zhan Ma. 2021. Lossy Point Cloud Geometry Compression via End-to-End Learning. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 12 (2021), 4909–4923. https://doi.org/10.1109/TCSVT.2021.3051377

Li-Yi Wei and Marc Levoy. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 479–488.

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent sitzmann, and Srinath Sridhar. 2022. Neural Fields in Visual Computing and Beyond. *Computer Graphics Forum* 41, 2 (2022), 641–676. https://doi.org/10.1111/cgf.14505

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *ICCV*.

Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. 2016. Loss Functions for Image Restoration With Neural Networks. *IEEE Transactions on Computational Imaging* PP (12 2016), 1–1. https://doi.org/10.1109/TCI.2016.2644865

Table 5. Quantiative results on the full synthetic dataset from Mildenhall et al. [2020], showing a near-quality (PSNR) comparison between Instant NGP and our work. We see that we are able to achieve similar quality across the entire dataset with a 2.8× more compact representation.

Method	N_f	N_c	N_p	*Mic*	*Ficus*	*Chair*	*Hotdog*	*Materials*	*Drums*	*Ship*	*Lego*	*avg.*	*Size (kB)*
I NGP	2^{14}	n/a	2^0	35.08	30.99	32.59	34.99	28.73	25.36	27.71	32.03	30.93	1000 kB
Ours	2^8	2^3	2^{16}	33.88	32.08	32.05	34.26	28.32	24.71	27.71	32.31	30.66	357 kB

Table 6. Quantiative results on texture compression on the *Paving Stones* texture set, retrieved from `https://ambientcg.com`, showing the tradeoff between quality (PSNR) and size (kB) for different methods. We compare against traditional texture compression baselines (BC) as well as recent neural baselines (NTC [Vaidyanathan et al. 2023]). We borrow the results from Vaidyanathan et al. [2023]. Although our work does not outperform NTC, which uses a specialized architecture for textures with quantization, we are still able to outperform BC and Instant NGP at similar size. We only report average across all channels for BC as that was the only data available, and compare against the NTC results without mipmaps (which increase quality) for fair comparison.

Method	*Quantization*	N_f	N_c	N_p	*Diffuse*	*Normal*	*Roughness*	*AO*	*Displacement*	*avg.*	*Size (kB)*
I NGP		2^{16}	n/a	n/a	21.58	22.32	26.79	27.72	35.62	24.75	3761 kB
I NGP		2^{14}	n/a	n/a	19.91	20.51	26.61	25.56	30.07	22.61	1049 kB
BC	✓	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	23.25	3500 kB
NTC	✓	n/a	n/a	n/a	26.10	27.17	29.37	31.22	40.59	29.00	3360 kB
Ours		2^{10}	2^{20}	2^3	24.02	25.00	27.90	29.94	36.18	26.69	3494 kB
Ours		2^8	2^{18}	2^3	21.55	22.61	26.94	27.43	33.74	24.51	1173 kB