Instant Neural Graphics Primitives with a Multiresolution Hash Encoding

THOMAS MÜLLER, NVIDIA, Switzerland
ALEX EVANS, NVIDIA, United Kingdom
CHRISTOPH SCHIED, NVIDIA, USA
ALEXANDER KELLER, NVIDIA, Germany

https://nvlabs.github.io/instant-ngp

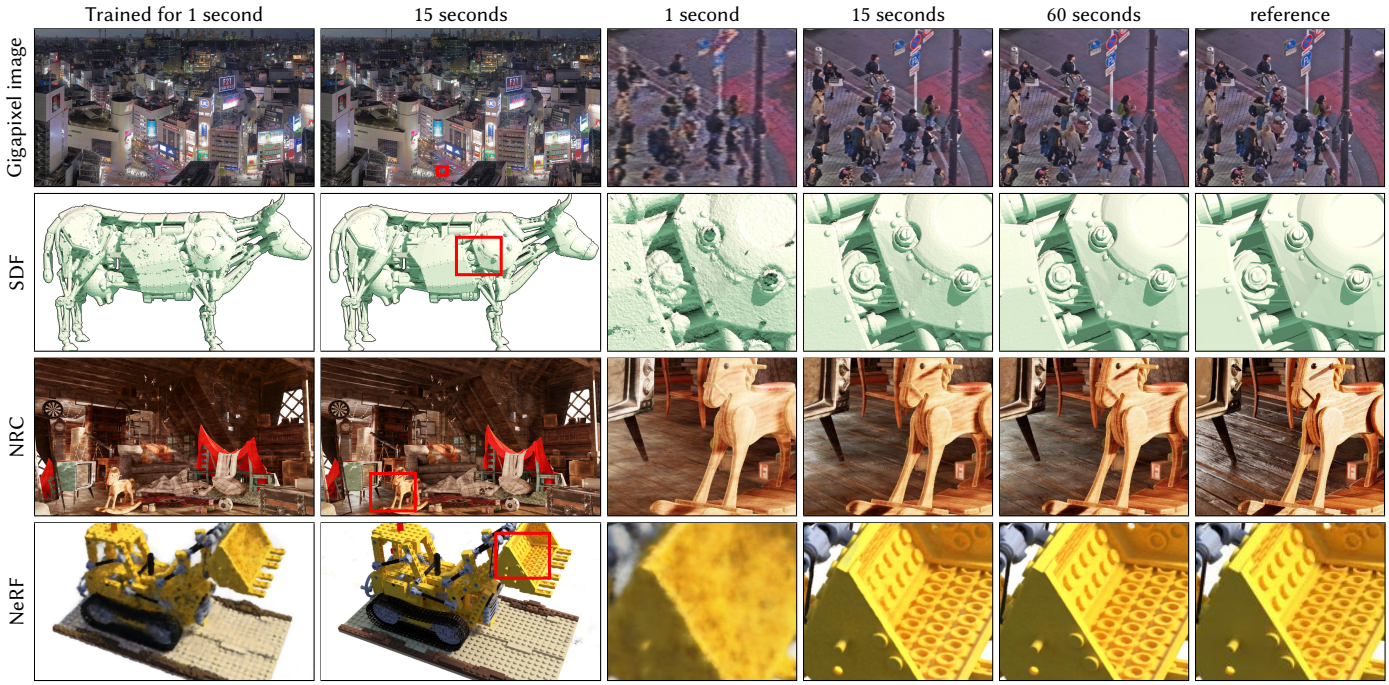

Fig. 1. We demonstrate instant training of neural graphics primitives on a single GPU for multiple tasks. In *Gigapixel image* we represent a gigapixel image by a neural network. *SDF* learns a signed distance function in 3D space whose zero level-set represents a 2D surface. Neural radiance caching (*NRC*) [Müller et al. 2021] employs a neural network that is trained in real-time to cache costly lighting calculations. Lastly, *NeRF* [Mildenhall et al. 2020] uses 2D images and their camera poses to reconstruct a volumetric radiance-and-density field that is visualized using ray marching. In all tasks, our encoding and its efficient implementation provide clear benefits: rapid training, high quality, and simplicity. Our encoding is task-agnostic: we use the same implementation and hyperparameters across all tasks and only vary the hash table size which trades off quality and performance. Tokyo gigapixel photograph ©Trevor Dobson (CC BY-NC-ND 2.0), Lego bulldozer 3D model ©Håvard Dalen (CC BY-NC 2.0)

Neural graphics primitives, parameterized by fully connected neural networks, can be costly to train and evaluate. We reduce this cost with a versatile new input encoding that permits the use of a smaller network without sacrificing quality, thus significantly reducing the number of floating point and memory access operations: a small neural network is augmented by a multiresolution hash table of trainable feature vectors whose values are optimized through stochastic gradient descent. The multiresolution structure allows the network to disambiguate hash collisions, making for a simple architecture that is trivial to parallelize on modern GPUs. We leverage this parallelism by implementing the whole system using fully-fused CUDA kernels with a focus on minimizing wasted bandwidth and compute operations. We achieve a combined speedup of several orders of magnitude, enabling training of high-quality neural graphics primitives in a matter of seconds, and rendering in tens of milliseconds at a resolution of 1920×1080.

CCS Concepts: • **Computing methodologies** → *Massively parallel algorithms*; *Vector / streaming algorithms*; **Neural networks**.

Additional Key Words and Phrases: Image Synthesis, Neural Networks, Encodings, Hashing, GPUs, Parallel Computation, Function Approximation.

ACM Reference Format:
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

Authors' addresses: Thomas Müller, NVIDIA, Zürich, Switzerland, tmueller@nvidia.com; Alex Evans, NVIDIA, London, United Kingdom, alexe@nvidia.com; Christoph Schied, NVIDIA, Seattle, USA, cschied@nvidia.com; Alexander Keller, NVIDIA, Berlin, Germany, akeller@nvidia.com.

1 INTRODUCTION

Computer graphics primitives are fundamentally represented by mathematical functions that parameterize appearance. The quality and performance characteristics of the mathematical representation are crucial for visual fidelity: we desire representations that remain fast and compact while capturing high-frequency, local detail. Functions represented by multi-layer perceptrons (MLPs), used as *neural graphics primitives*, have been shown to match these criteria (to varying degree), for example as representations of shape [Martel et al. 2021; Park et al. 2019] and radiance fields [Liu et al. 2020; Mildenhall et al. 2020; Müller et al. 2020, 2021].

The important commonality of the these approaches is an encoding that maps neural network inputs to a higher-dimensional space, which is key for extracting high approximation quality from compact models. Most successful among these encodings are trainable, task-specific data structures [Liu et al. 2020; Takikawa et al. 2021] that take on a large portion of the learning task. This enables the use of smaller, more efficient MLPs. However, such data structures rely on heuristics and structural modifications (such as pruning, splitting, or merging) that may complicate the training process, limit the method to a specific task, or limit performance on GPUs where control flow and pointer chasing is expensive.

We address these concerns with our multiresolution hash encoding, which is adaptive and efficient, independent of the task. It is configured by just two values—the number of parameters T and the desired finest resolution $N_{\max}$—yielding state-of-the-art quality on a variety of tasks (Figure 1) after a few seconds of training.

Key to both the task-independent adaptivity and efficiency is a multiresolution hierarchy of hash tables:

- **Adaptivity:** we map a cascade of grids to corresponding fixed-size arrays of feature vectors. At coarse resolutions, there is a 1:1 mapping from grid points to array entries. At fine resolutions, the array is treated as a hash table and indexed using a spatial hash function, where multiple grid points alias each array entry. Such hash collisions cause the colliding training gradients to average, meaning that the largest gradients—those most relevant to the loss function—will dominate. The hash tables thus *automatically* prioritize the sparse areas with the most important fine scale detail. Unlike prior work, no structural updates to the data structure are needed at any point during training.
- **Efficiency:** our hash table lookups are $O(1)$ and do not require control flow. This maps well to modern GPUs, avoiding execution divergence and serial pointer-chasing inherent in tree traversals. The hash tables for all resolutions may be queried in parallel.

We validate our multiresolution hash encoding in four representative tasks (see Figure 1):

(1) **Gigapixel image:** the MLP learns the mapping from 2D coordinates to RGB colors of a high-resolution image.
(2) **Neural signed distance functions (SDF):** the MLP learns the mapping from 3D coordinates to the distance to a surface.
(3) **Neural radiance caching (NRC):** the MLP learns the 5D light field of a given scene from a Monte Carlo path tracer.
(4) **Neural radiance and density fields (NeRF):** the MLP learns the 3D density and 5D light field of a given scene from image observations and corresponding perspective transforms.

In the following, we first review prior neural network encodings (Section 2), then we describe our encoding (Section 3) and its implementation (Section 4), followed lastly by our experiments (Section 5) and discussion thereof (Section 6).

2 BACKGROUND AND RELATED WORK

Early examples of encoding the inputs of a machine learning model into a higher-dimensional space include the one-hot encoding [Harris and Harris 2013] and the kernel trick [Theodoridis 2008] by which complex arrangements of data can be made linearly separable.

For neural networks, input encodings have proven useful in the attention components of recurrent architectures [Gehring et al. 2017] and, subsequently, transformers [Vaswani et al. 2017], where they help the neural network to identify the location it is currently processing. Vaswani et al. [2017] encode scalar positions $x \in \mathbb{R}$ as a multiresolution sequence of $L \in \mathbb{N}$ sine and cosine functions

$$\begin{aligned}\operatorname{enc}(x) = \big(&\sin(2^0 x), \sin(2^1 x), \ldots, \sin(2^{L-1} x),\\ &\cos(2^0 x), \cos(2^1 x), \ldots, \cos(2^{L-1} x)\big).\end{aligned} \tag{1}$$

This has been adopted in computer graphics to encode the spatio-directionally varying light field and volume density in the NeRF algorithm [Mildenhall et al. 2020]. The five dimensions of this light field are *independently* encoded using the above formula; this was later extended to randomly oriented parallel wavefronts [Tancik et al. 2020] and level-of-detail filtering [Barron et al. 2021a]. We will refer to this family of encodings as *frequency encodings*. Notably, frequency encodings followed by a linear transformation have been used in other computer graphics tasks, such as approximating the visibility function [Annen et al. 2007; Jansen and Bavoil 2010].

Müller et al. [2019; 2020] suggested a continuous variant of the one-hot encoding based on rasterizing a kernel, the *one-blob* encoding, which can achieve more accurate results than frequency encodings in bounded domains at the cost of being single-scale.

Parametric encodings. Recently, state-of-the-art results have been achieved by parametric encodings which blur the line between classical data structures and neural approaches. The idea is to arrange additional trainable parameters (beyond weights and biases) in an auxiliary data structure, such as a grid [Chabra et al. 2020; Jiang et al. 2020; Liu et al. 2020; Mehta et al. 2021; Peng et al. 2020a; Sun et al. 2021; Tang et al. 2018; Yu et al. 2021a] or a tree [Takikawa et al. 2021], and to look-up and (optionally) interpolate these parameters depending on the input vector $\mathbf{x} \in \mathbb{R}^d$. This arrangement trades a larger memory footprint for a smaller computational cost: whereas for each gradient propagated backwards through the network, every weight in the fully connected MLP network must be updated, for the trainable input encoding parameters ("feature vectors"), only a very small number are affected. For example, with a trilinearly interpolated 3D grid of feature vectors, only 8 such grid points need to be updated for each sample back-propagated to the encoding. In this way, although the total number of parameters is much higher for a parametric encoding than a fixed input encoding, the number of FLOPs and memory accesses required for the update during training is not increased significantly. By reducing the size of the MLP, such parametric models can typically be trained to convergence much faster without sacrificing approximation quality.

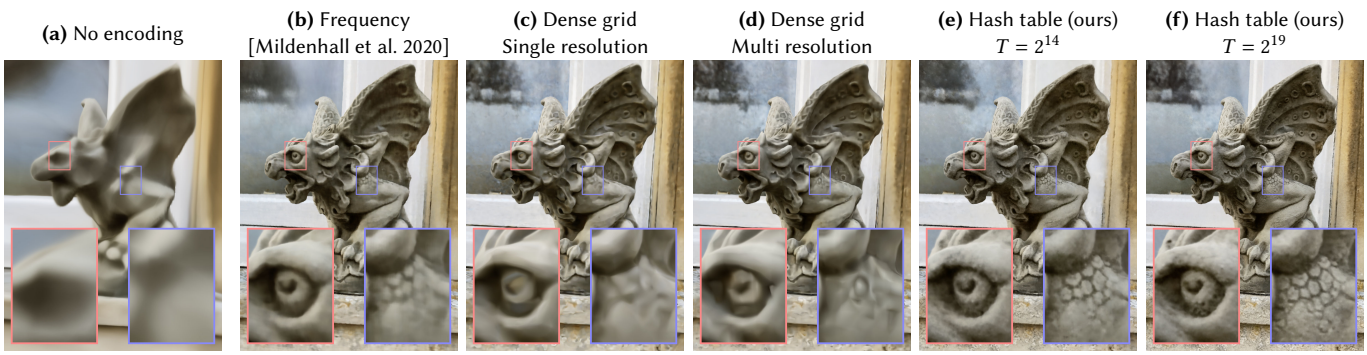

Fig. 2. A demonstration of the reconstruction quality of different encodings and parametric data structures for storing trainable feature embeddings. Each configuration was trained for 11 000 steps using our fast NeRF implementation (Section 5.4), varying only the input encoding and MLP size. The number of trainable parameters (MLP weights + encoding parameters), training time and reconstruction accuracy (PSNR) are shown below each image. Our encoding **(e)** with a similar total number of trainable parameters as the frequency encoding configuration **(b)** trains over 8× faster, due to the sparsity of updates to the parameters and smaller MLP. Increasing the number of parameters **(f)** further improves reconstruction accuracy without significantly increasing training time.

Another parametric approach uses a tree subdivision of the domain $\mathbb{R}^d$, wherein a large auxiliary *coordinate encoder* neural network (ACORN) [Martel et al. 2021] is trained to output dense feature grids in the leaf node around $\mathbf{x}$. These dense feature grids, which have on the order of 10 000 entries, are then linearly interpolated, as in Liu et al. [2020]. This approach tends to yield a larger degree of adaptivity compared with the previous parametric encodings, albeit at greater computational cost which can only be amortized when sufficiently many inputs $\mathbf{x}$ fall into each leaf node.

Sparse parametric encodings. While existing parametric encodings tend to yield much greater accuracy than their non-parametric predecessors, they also come with downsides in efficiency and versatility. Dense grids of trainable features consume much more memory than the neural network weights. To illustrate the trade-offs and to motivate our method, Figure 2 shows the effect on reconstruction quality of a neural radiance field for several different encodings. Without any input encoding at all **(a)**, the network is only able to learn a fairly smooth function of position, resulting in a poor approximation of the light field. The frequency encoding **(b)** allows the same moderately sized network (8 hidden layers, each 256 wide) to represent the scene much more accurately. The middle image **(c)** pairs a smaller network with a dense grid of 128^3 trilinearly interpolated, 16-dimensional feature vectors, for a total of 33.6 million trainable parameters. The large number of trainable parameters can be efficiently updated, as each sample only affects 8 grid points.

However, the dense grid is wasteful in two ways. First, it allocates as many features to areas of empty space as it does to those areas near the surface. The number of parameters grows as $O(N^3)$, while the visible surface of interest has surface area that grows only as $O(N^2)$. In this example, the grid has resolution 128^3, but only 53 807 (2.57%) of its cells touch the visible surface.

Second, natural scenes exhibit smoothness, motivating the use of a multi-resolution decomposition [Chibane et al. 2020; Hadadan et al. 2021]. Figure 2 **(d)** shows the result of using an encoding in which interpolated features are stored in eight co-located grids with resolutions from 16^3 to 173^3, each containing 2-dimensional feature vectors. These are concatenated to form a 16-dimensional (same as **(c)**) input to the network. Despite having less than half the number of parameters as **(c)**, the reconstruction quality is similar.

If the surface of interest is known a priori, a data structure such as an octree [Takikawa et al. 2021] or sparse grid [Chabra et al. 2020; Chibane et al. 2020; Hadadan et al. 2021; Jiang et al. 2020; Liu et al. 2020; Peng et al. 2020a] can be used to cull away the unused features in the dense grid. However, in the NeRF setting, surfaces only emerge during training. NSVF [Liu et al. 2020] and several concurrent works [Sun et al. 2021; Yu et al. 2021a] adopt a multi-stage, coarse to fine strategy in which regions of the feature grid are progressively refined and culled away as necessary. While effective, this leads to a more complex training process in which the sparse data structure must be periodically updated.

Our method—Figure 2 **(e,f)**—combines both ideas to reduce waste. We store the trainable feature vectors in a compact spatial hash table, whose size is a hyper-parameter T which can be tuned to trade the number of parameters for reconstruction quality. It neither relies on progressive pruning during training nor on a priori knowledge of the geometry of the scene. Analogous to the multi-resolution grid in **(d)**, we use multiple separate hash tables indexed at different resolutions, whose interpolated outputs are concatenated before being passed through the MLP. The reconstruction quality is comparable to the dense grid encoding, despite having 20× fewer parameters.

Unlike prior work that used spatial hashing [Teschner et al. 2003] for 3D reconstruction [Nießner et al. 2013], we do not explicitly handle collisions of the hash functions by typical means like probing, bucketing, or chaining. Instead, we rely on the neural network to learn to disambiguate hash collisions itself, avoiding control flow divergence, reducing implementation complexity and improving performance. Another performance benefit is the predictable memory layout of the hash tables that is independent of the data that is represented. While good caching behavior is often hard to achieve with tree-like data structures, our hash tables can be fine-tuned for low-level architectural details such as cache size.

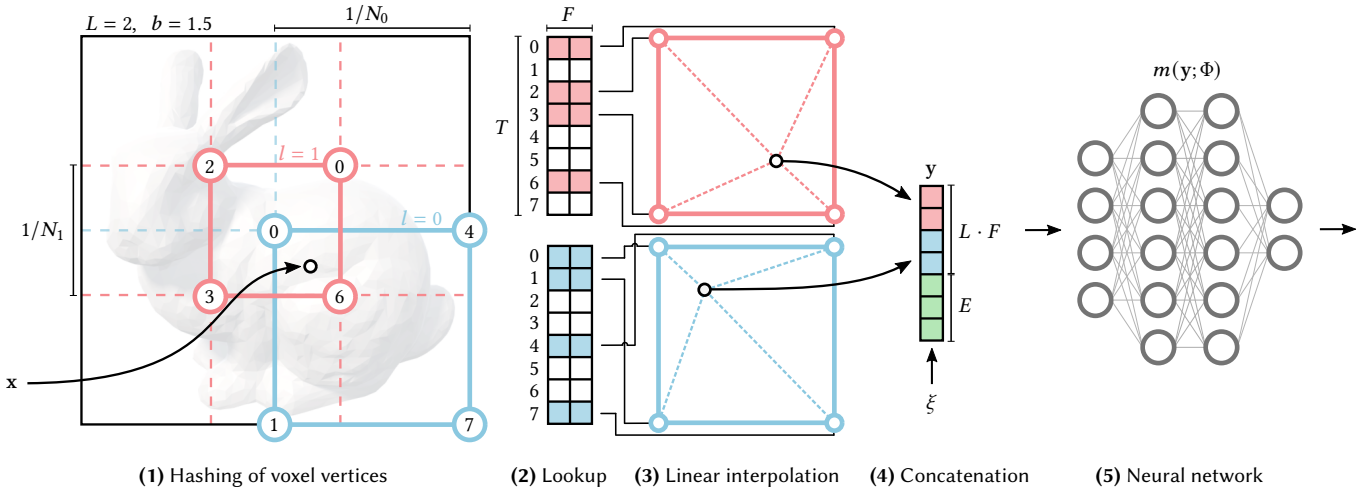

Fig. 3. Illustration of the multiresolution hash encoding in 2D. **(1)** for a given input coordinate $\mathbf{x}$, we find the surrounding voxels at L resolution levels and assign indices to their corners by hashing their integer coordinates. **(2)** for all resulting corner indices, we look up the corresponding F-dimensional feature vectors from the hash tables θ_l and **(3)** linearly interpolate them according to the relative position of $\mathbf{x}$ within the respective l-th voxel. **(4)** we concatenate the result of each level, as well as auxiliary inputs $\xi \in \mathbb{R}^E$, producing the encoded MLP input $y \in \mathbb{R}^{LF+E}$, which **(5)** is evaluated last. To train the encoding, loss gradients are backpropagated through the MLP **(5)**, the concatenation **(4)**, the linear interpolation **(3)**, and then accumulated in the looked-up feature vectors.

Table 1. Hash encoding parameters and their ranges in our results. Only the hash table size T and max. resolution $N_{\max}$ need to be tuned to the task.

Parameter	Symbol	Value
Number of levels	L	16
Max. entries per level (hash table size)	T	2^{14} to 2^{24}
Number of feature dimensions per entry	F	2
Coarsest resolution	$N_{\min}$	16
Finest resolution	$N_{\max}$	512 to 524288

3 MULTIRESOLUTION HASH ENCODING

Given a fully connected neural network $m(\mathbf{y};\Phi)$, we are interested in an *encoding* of its inputs $\mathbf{y} = \text{enc}(\mathbf{x};\theta)$ that improves the approximation quality and training speed across a wide range of applications without incurring a notable performance overhead. Our neural network not only has trainable weight parameters Φ, but also trainable encoding parameters θ. These are arranged into L levels, each containing up to T feature vectors with dimensionality F. Typical values for these hyperparameters are shown in Table 1. Figure 3 illustrates the steps performed in our multiresolution hash encoding. Each level (two of which are shown as red and blue in the figure) is independent and conceptually stores feature vectors at the vertices of a grid, the resolution of which is chosen to be a geometric progression between the coarsest and finest resolutions $[N_{\min}, N_{\max}]$:

$$N_l := \left\lfloor N_{\min} \cdot b^l \right\rfloor, \tag{2}$$

$$b := \exp\left(\frac{\ln N_{\max} - \ln N_{\min}}{L-1}\right). \tag{3}$$

$N_{\max}$ is chosen to match the finest detail in the training data. Due to the large number of levels L, the growth factor is usually small. Our use cases have $b \in [1.26, 2]$.

Consider a single level l. The input coordinate $\mathbf{x} \in \mathbb{R}^d$ is scaled by that level's grid resolution before rounding down and up $\lfloor \mathbf{x}_l \rfloor := \lfloor \mathbf{x} \cdot N_l \rfloor$, $\lceil \mathbf{x}_l \rceil := \lceil \mathbf{x} \cdot N_l \rceil$.

$\lfloor \mathbf{x}_l \rfloor$ and $\lceil \mathbf{x}_l \rceil$ span a voxel with 2^d integer vertices in $\mathbb{Z}^d$. We map each corner to an entry in the level's respective feature vector array, which has fixed size of at most T. For coarse levels where a dense grid requires fewer than T parameters, i.e. $(N_l + 1)^d \leq T$, this mapping is 1:1. At finer levels, we use a hash function $h : \mathbb{Z}^d \to \mathbb{Z}_T$ to index into the array, effectively treating it as a hash table, although there is no explicit collision handling. We rely instead on the gradient-based optimization to store appropriate sparse detail in the array, and the subsequent neural network $m(\mathbf{y};\Phi)$ for collision resolution. The number of trainable encoding parameters θ is therefore $O(T)$ and bounded by $T \cdot L \cdot F$ which in our case is always $T \cdot 16 \cdot 2$ (Table 1).

We use a spatial hash function [Teschner et al. 2003] of the form

$$h(\mathbf{x}) = \left(\bigoplus_{i=1}^{d} x_i \pi_i\right) \mod T, \tag{4}$$

where $\oplus$ denotes the bit-wise XOR operation and π_i are unique, large prime numbers. Effectively, this formula XORs the results of a per-dimension linear congruential (pseudo-random) permutation [Lehmer 1951], *decorrelating* the effect of the dimensions on the hashed value. Notably, to achieve (pseudo-)independence, only $d-1$ of the d dimensions must be permuted, so we choose $\pi_1 := 1$ for better cache coherence, $\pi_2 = 2\,654\,435\,761$, and $\pi_3 = 805\,459\,861$.

Lastly, the feature vectors at each corner are d-linearly interpolated according to the relative position of $\mathbf{x}$ within its hypercube, i.e. the interpolation weight is $\mathbf{w}_l := \mathbf{x}_l - \lfloor \mathbf{x}_l \rfloor$.

Recall that this process takes place independently for each of the L levels. The interpolated feature vectors of each level, as well as auxiliary inputs $\xi \in \mathbb{R}^E$ (such as the encoded view direction and textures in neural radiance caching), are concatenated to produce $\mathbf{y} \in \mathbb{R}^{LF+E}$, which is the encoded input $\text{enc}(\mathbf{x};\theta)$ to the MLP $m(\mathbf{y};\Phi)$.

Performance vs. quality. Choosing the hash table size T provides a trade-off between performance, memory and quality. Higher values of T result in higher quality and lower performance. The memory

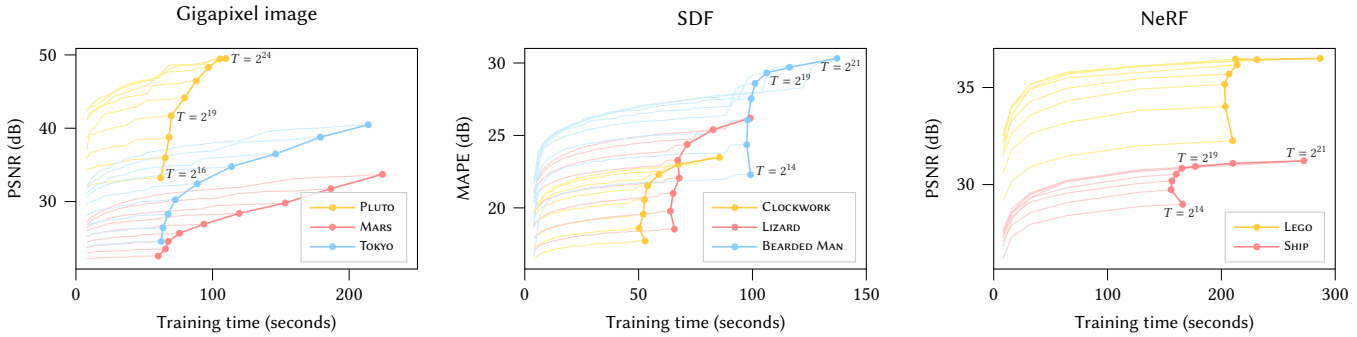

Fig. 4. The main curves plot test error over training time for varying hash table size T which determines the number of trainable encoding parameters. Increasing T improves reconstruction, at the cost of higher memory usage and slower training and inference. A performance cliff is visible at $T > 2^{19}$ where the cache of our RTX 3090 GPU becomes oversubscribed (particularly visible for SDF and NeRF). The plot also shows model convergence over time leading up to the final state. This highlights how high quality results are already obtained after only a few seconds. Jumps in the convergence (most visible towards the end of SDF training) are caused by learning rate decay. For NeRF and Gigapixel image, training finishes after 31 000 steps and for SDF after 11 000 steps.

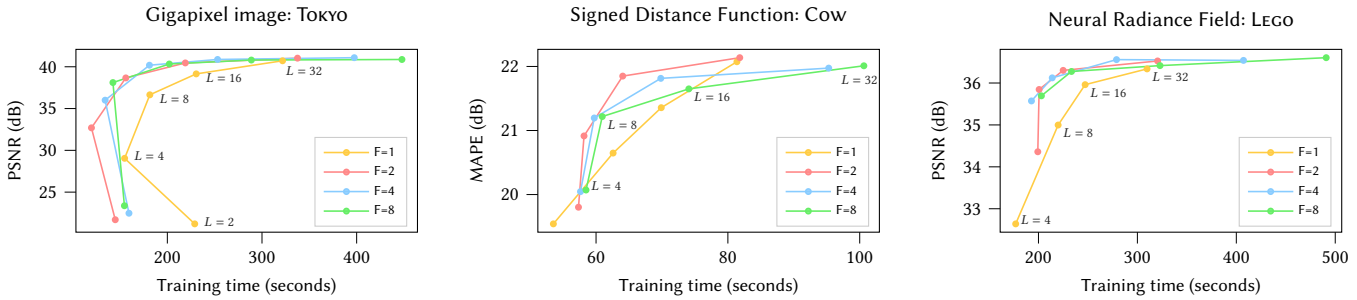

Fig. 5. Test error over training time for fixed values of feature dimensionality F as the number of hash table levels L is varied. To maintain a roughly equal trainable parameter count, the hash table size T is set according to $F \cdot T \cdot L = 2^{24}$ for SDF and NeRF, whereas gigapixel image uses 2^{28}. Since $(F = 2, L = 16)$ is near the best-case performance and quality (top-left) for all applications, we use this configuration in all results. $F = 1$ is slow on our RTX 3090 GPU since atomic half-precision accumulation is only efficient for 2D vectors but not for scalars. For NeRF and Gigapixel image, training finishes after 31 000 steps whereas SDF completes at 11 000 steps.

footprint is linear in T, whereas quality and performance tend to scale sub-linearly. We analyze the impact of T in Figure 4, where we report test error vs. training time for a wide range of T-values for three neural graphics primitives. We recommend practitioners to use T to tweak the encoding to their desired performance characteristics.

The hyperparameters L (number of levels) and F (number of feature dimensions) also trade off quality and performance, which we analyze for an approximately constant number of trainable encoding parameters θ in Figure 5. In this analysis, we found $(F = 2, L = 16)$ to be a favorable Pareto optimum in all our applications, so we use these values in all other results and recommend them as the default.

Implicit hash collision resolution. It may appear counter-intuitive that this encoding is able to reconstruct scenes faithfully in the presence of hash collisions. Key to its success is that the different resolution levels have different strengths that complement each other. The coarser levels, and thus the encoding as a whole, are injective—that is, they suffer from no collisions at all. However, they can only represent a low-resolution version of the scene, since they offer features which are linearly interpolated from a widely spaced grid of points. Conversely, fine levels can capture small features due to their fine grid resolution, but suffer from many collisions—that is, disparate points which hash to the same table entry. Nearby inputs with equal integer coordinates $\lfloor \mathbf{x}_l \rfloor$ are not considered a collision; a collision occurs when *different* integer coordinates hash to the same index. Luckily, such collisions are pseudo-randomly scattered across space, and statistically unlikely to occur *simultaneously* at every level for a given pair of points.

When training samples collide in this way, their gradients average. Consider that the importance to the final reconstruction of such samples is rarely equal. For example, a point on a visible surface of a radiance field will contribute strongly to the reconstructed image (having high visibility and high density, both multiplicatively affecting the magnitude of gradients) causing large changes to its table entries, while a point in empty space that happens to refer to the same entry will have a much smaller weight. As a result, the gradients of the more important samples dominate the collision average and the aliased table entry will naturally be optimized in such a way that it reflects the needs of the higher-weighted point.

The multiresolution aspect of the hash encoding covers the full range from a coarse resolution $N_{\min}$ that is guaranteed to be collision-free to the finest resolution $N_{\max}$ that the task requires. Thereby, it guarantees that all scales at which meaningful learning *could* take place are included, regardless of sparsity. Geometric scaling allows covering these scales with only $O\big(\log\left(N_{\max}/N_{\min}\right)\big)$ many levels, which allows picking a conservatively large value for $N_{\max}$.

Online adaptivity. Note that if the distribution of inputs $\mathbf{x}$ changes over time during training, for example if they become concentrated in a small region, then finer grid levels will experience fewer collisions and a more accurate function can be learned. In other words, the multiresolution hash encoding *automatically* adapts to the training data distribution, inheriting the benefits of tree-based encodings [Takikawa et al. 2021] without task-specific data structure maintenance that might cause discrete jumps during training. One of our applications, neural radiance caching in Section 5.3, continually adapts to animated viewpoints and 3D content, greatly benefitting from this feature.

d-linear interpolation. Interpolating the queried hash table entries ensures that the encoding $\text{enc}(\mathbf{x};\theta)$, and by the chain rule its composition with the neural network $m(\text{enc}(\mathbf{x};\theta);\Phi)$, are continuous. Without interpolation, grid-aligned discontinuities would be present in the network output, which would result in an undesirable blocky appearance. One may desire higher-order smoothness, for example when approximating partial differential equations. A concrete example from computer graphics are signed distance functions, in which case the gradient $\partial m(\text{enc}(\mathbf{x};\theta);\Phi)/\partial\mathbf{x}$, i.e. the surface normal, would ideally also be continuous. If higher-order smoothness must be guaranteed, we describe a low-cost approach in Appendix A, which we however do *not* employ in any of our results due to a small decrease in reconstruction quality.

4 IMPLEMENTATION

To demonstrate the speed of the multiresolution hash encoding, we implemented it in CUDA and integrated it with the fast fully-fused MLPs of the *tiny-cuda-nn* framework [Müller 2021].[1] We release the source code of the multiresolution hash encoding as an update to Müller [2021] and the source code pertaining to the neural graphics primitives at `https://github.com/nvlabs/instant-ngp`.

Performance considerations. In order to optimize inference and backpropagation performance, we store hash table entries at half precision (2 bytes per entry). We additionally maintain a master copy of the parameters in full precision for stable mixed-precision parameter updates, following Micikevicius et al. [2018].

To optimally use the GPU's caches, we evaluate the hash tables level by level: when processing a batch of input positions, we schedule the computation to look up the first level of the multiresolution hash encoding for *all* inputs, followed by the second level for all inputs, and so on. Thus, only a small number of consecutive hash tables have to reside in caches at any given time, depending on how much parallelism is available on the GPU. Importantly, this structure of computation *automatically* makes good use of the available caches and parallelism for a wide range of hash table sizes T.

On our hardware, the performance of the encoding remains roughly constant as long as the hash table size stays below $T \leq 2^{19}$. Beyond this threshold, performance starts to drop significantly; see Figure 4. This is explained by the 6 MB L2 cache of our NVIDIA RTX 3090 GPU, which becomes too small for individual levels when $2 \cdot T \cdot F > 6 \cdot 2^{20}$, with 2 being the size of a half-precision entry.

[1] We observe speed-ups on the order of 10× compared to a naïve Python implementation. We therefore also release PyTorch bindings around our hash encoding and fully fused MLPs to permit their use in existing projects with little overhead.

The optimal number of feature dimensions F per lookup depends on the GPU architecture. On one hand, a small number favors cache locality in the previously mentioned streaming approach, but on the other hand, a large F favors memory coherence by allowing for F-wide vector load instructions. $F = 2$ gave us the best cost-quality trade-off (see Figure 5) and we use it in all experiments.

Architecture. In all tasks, except for NeRF which we will describe later, we use an MLP with two hidden layers that have a width of 64 neurons, rectified linear unit (ReLU) activation functions on their hidden layers, and a linear output layer. The maximum resolution N_{max} is set to 2048 × scene size for NeRF and signed distance functions, to half of the gigapixel image width, and 2^{19} in radiance caching (large value to support close-by objects in expansive scenes).

Initialization. We initialize neural network weights according to Glorot and Bengio [2010] to provide a reasonable scaling of activations and their gradients throughout the layers of the neural network. We initialize the hash table entries using the uniform distribution $\mathcal{U}(-10^{-4}, 10^{-4})$ to provide a small amount of randomness while encouraging initial predictions close to zero. We also tried a variety of different distributions, including zero-initialization, which all resulted in a very slightly worse initial convergence speed. The hash table appears to be robust to the initialization scheme.

Training. We jointly train the neural network weights and the hash table entries by applying Adam [Kingma and Ba 2014], where we set $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-15}$, The choice of β_1 and β_2 makes only a small difference, but the small value of $\epsilon = 10^{-15}$ can significantly accelerate the convergence of the hash table entries when their gradients are sparse and weak. To prevent divergence after long training periods, we apply a weak L2 regularization (factor 10^{-6}) to the neural network weights, but not to the hash table entries.

When fitting gigapixel images or NeRFs, we use the $\mathcal{L}^2$ loss. For signed distance functions, we use the mean absolute percentage error (MAPE), defined as $\frac{|\text{prediction} - \text{target}|}{|\text{target}| + 0.01}$, and for neural radiance caching we use a luminance-relative $\mathcal{L}^2$ loss [Müller et al. 2021].

We observed fastest convergence with a learning rate of 10^{-4} for signed distance functions and 10^{-2} otherwise, as well a a batch size of 2^{14} for neural radiance caching and 2^{18} otherwise.

Lastly, we skip Adam steps for hash table entries whose gradient is exactly 0. This saves ∼10% performance when gradients are sparse, which is a common occurrence with $T \gg \text{BatchSize}$. Even though this heuristic violates some of the assumptions behind Adam, we observe no degradation in convergence.

Non-spatial input dimensions $\xi \in \mathbb{R}^E$. The multiresolution hash encoding targets spatial coordinates with relatively low dimensionality. All our experiments operate either in 2D or 3D. However, it is frequently useful to input auxiliary dimensions $\xi \in \mathbb{R}^E$ to the neural network, such as the view direction and material parameters when learning a light field. In such cases, the auxiliary dimensions can be encoded with established techniques whose cost does not scale superlinearly with dimensionality; we use the one-blob encoding [Müller et al. 2019] in neural radiance caching [Müller et al. 2021] and the spherical harmonics basis in NeRF, similar to concurrent work [Verbin et al. 2021; Yu et al. 2021a].

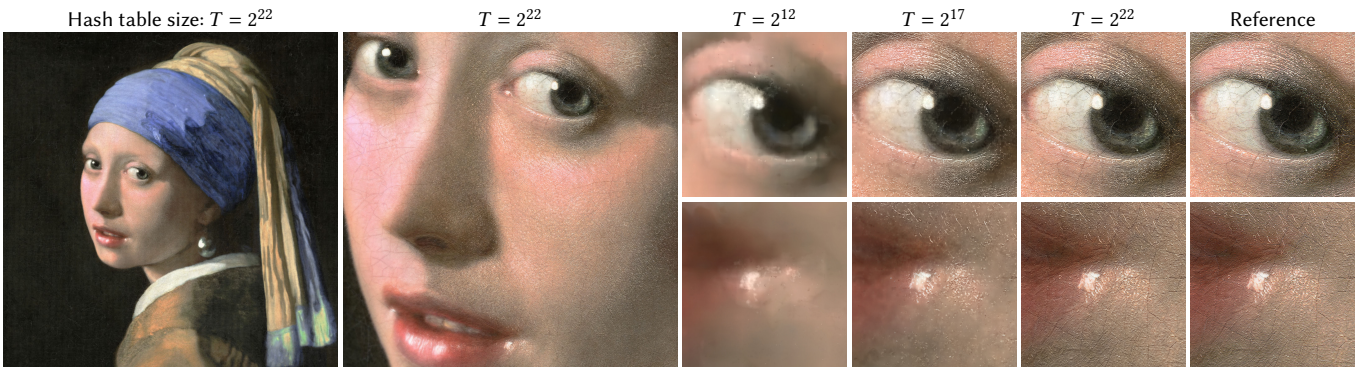

Fig. 6. Approximating an RGB image of resolution 20 000 × 23 466 (469 M RGB pixels) with our multiresolution hash encoding. With hash table sizes T of 2^{12}, 2^{17}, and 2^{22} the models shown have 117 k, 2.7 M, and 47.5 M trainable parameters respectively. With only 3.4% of the degrees of freedom of the input, the last model achieves a reconstruction PSNR of 29.8 dB. "Girl With a Pearl Earring" renovation ©Koorosh Orooj (CC BY-SA 4.0)

5 EXPERIMENTS

To highlight the versatility and high quality of the encoding, we compare it with previous encodings in four distinct computer graphics primitives that benefit from encoding spatial coordinates.

5.1 Gigapixel Image Approximation

Learning the 2D to RGB mapping of image coordinates to colors has become a popular benchmark for testing a model's ability to represent high-frequency detail [Martel et al. 2021; Müller et al. 2019; Sitzmann et al. 2020; Tancik et al. 2020]. Recent breakthroughs in adaptive coordinate networks (ACORN) [Martel et al. 2021] have shown impressive results when fitting very large images—up to a billion pixels—with high fidelity at even the smallest scales. We target our multiresolution hash encoding at the same task and converge to high-fidelity images in seconds to minutes (Figure 4).

For comparison, on the Tokyo panorama from Figure 1, ACORN achieves a PSNR of 38.59 dB after 36.9 h of training. With a similar number of parameters ($T = 2^{24}$), our method achieves the same PSNR after 2.5 *minutes* of training, peaking at 41.9 dB after 4 min. Figure 6 showcases the level of detail contained in our model for a variety of hash table sizes T on another image.

It is difficult to directly compare the performance of our encoding to ACORN; a factor of ∼10 stems from our use of fully fused CUDA kernels, provided by the tiny-cuda-nn framework [Müller 2021]. The input encoding allows for the use of a much smaller MLP than with ACORN, which accounts for much of the remaining 10×–100× speedup. That said, we believe that the biggest value-add of the multiresolution hash encoding is its simplicity. ACORN relies on an adaptive subdivision of the scene as part of a learning curriculum, none of which is necessary with our encoding.

5.2 Signed Distance Functions

Signed distance functions (SDFs), in which a 3D shape is represented as the zero level-set of a function of position $\mathbf{x}$, are used in many applications including simulation, path planning, 3D modeling, and video games. DeepSDF [Park et al. 2019] uses a large MLP to represent one or more SDFs at a time. In contrast, when just a single SDF needs to be fit, a spatially learned encoding, such as ours can be employed and the MLP shrunk significantly. This is the application we investigate in this section. As baseline, we compare with NGLOD [Takikawa et al. 2021], which achieves state-of-the-art results in both quality and speed by prefixing its small MLP with a lookup from an octree of trainable feature vectors. Lookups along the hierarchy of this octree act similarly to our multiresolution cascade of grids: they are a collision-free analog to our technique, with a fixed growth factor $b = 2$. To allow meaningful comparisons in terms of both performance and quality, we implemented an optimized version of NGLOD in our framework, details of which we describe in Appendix B. Details pertaining to real-time training of SDFs are described in Appendix C.

In Figure 7, we compare NGLOD with our multiresolution hash encoding at roughly equal parameter count. We also show a straightforward application of the frequency encoding [Mildenhall et al. 2020] to provide a baseline, details of which are found in Appendix D. By using a data structure tailored to the reference shape, NGLOD achieves the highest visual reconstruction quality. However, even without such a dedicated data structure, our encoding approaches a similar fidelity to NGLOD in terms of the intersection-over-union metric (IoU[2]) with similar performance and memory cost. Furthermore, the SDF is defined everywhere within the training volume, as opposed to NGLOD, which is only defined within the octree (i.e. close to the surface). This permits the use of certain SDF rendering techniques such as approximate soft shadows from a small number of off-surface distance samples [Evans 2006], as shown in the adjacent figure.

To emphasize differences between the compared methods, we visualize the SDF using a shading model. The resulting colors are sensitive to even slight changes in the surface normal, which emphasizes small fluctuations in the prediction more strongly than in other graphics primitives where color is predicted directly. This sensitivity reveals undesired microstructure in our hash encoding on the scale

[2]IoU is the ratio of volumes of the interiors of the intersection and union of the pair of shapes being compared. IoU is always ≤ 1 with a perfect fit corresponding to = 1. We measure IoU by comparing the signs of the SDFs at 128 million points uniformly distributed within the bounding box of the scene.

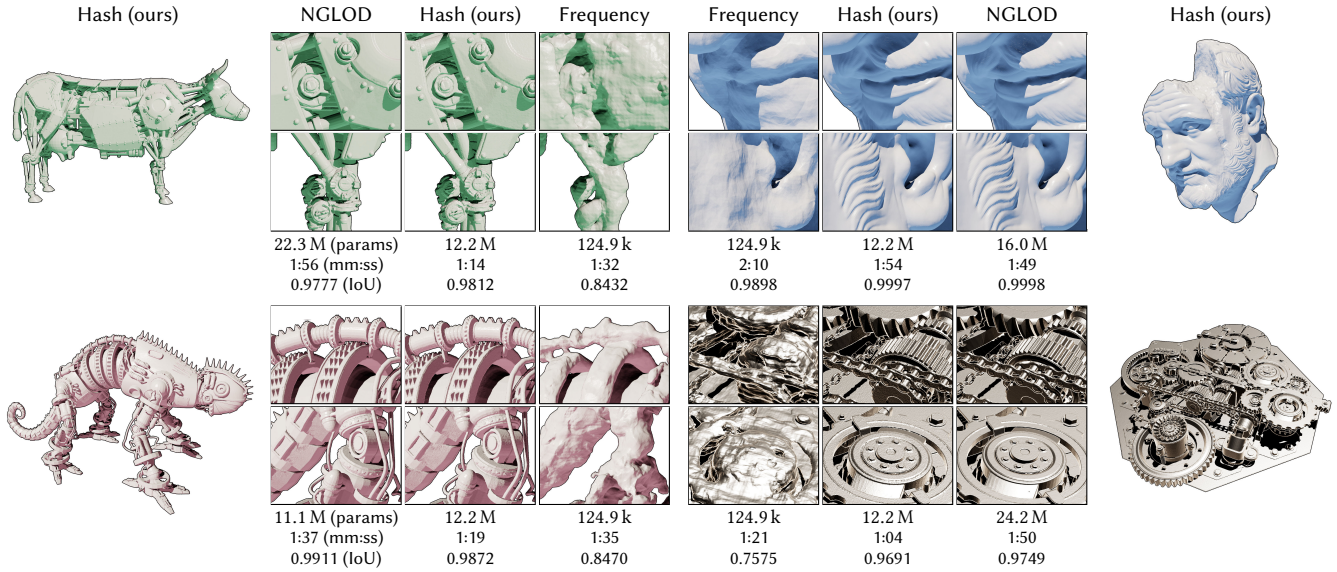

Fig. 7. Neural signed distance functions trained for 11 000 steps. The frequency encoding [Mildenhall et al. 2020] struggles to capture the sharp details on these intricate models. NGLOD [Takikawa et al. 2021] achieves the highest visual quality, at the cost of only training the SDF inside the cells of a close-fitting octree. Our hash encoding exhibits similar numeric quality in terms of intersection over union (IoU) and can be evaluated anywhere in the scene. However, it also exhibits visually undesirable surface roughness that we attribute to randomly distributed hash collisions. Bearded Man ©Oliver Laric (CC BY-NC-SA 2.0)

Fig. 8. Summary of the neural radiance caching application [Müller et al. 2021]. The MLP $m(\mathrm{enc}(x;\theta);\Phi)$ is tasked with predicting photorealistic pixel colors from feature buffers *independently for each pixel*. The feature buffers contain, among other variables, the world-space position $\mathbf{x}$, which we propose to encode with our method. Neural radiance caching is a challenging application, because it is supervised *online during real-time rendering*. The training data are a sparse set of light paths that are continually spawned from the camera view. As such, the neural network and encoding do *not* learn a general mapping from features to color, but rather they *continually overfit* to the current shape and lighting. To support animated content, training has a budget of *one* millisecond per frame.

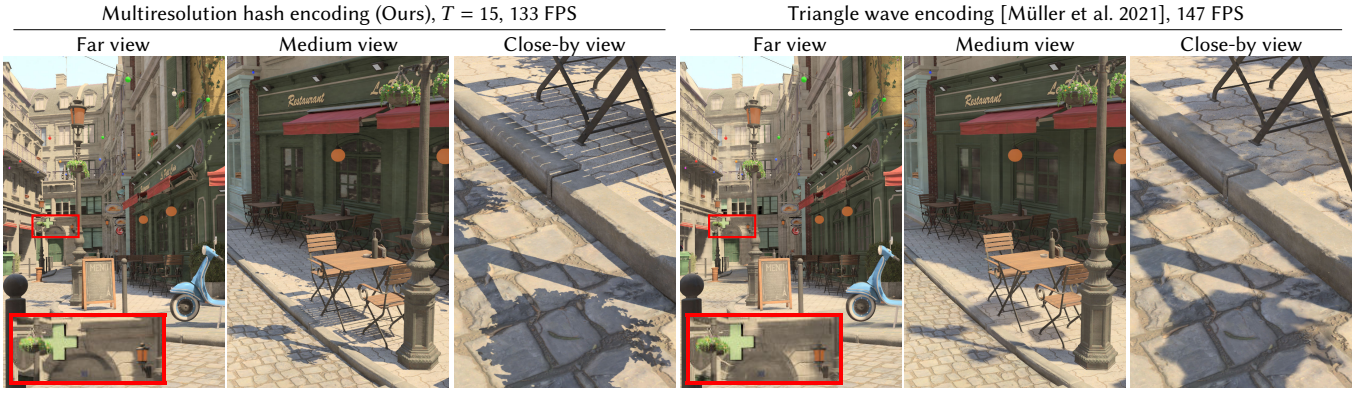

Fig. 9. Neural radiance caching [Müller et al. 2021] gains much improved quality from the multiresolution hash encoding with only a mild performance penalty: 133 versus 147 frames per second at a resolution of 1920×1080px. To demonstrate the online adaptivity of the multiple hash resolutions vs. the prior triangle wave encoding, we show screenshots from a smooth camera motion that starts with a far-away view of the scene (left) and zooms onto a close-by view of an intricate shadow (right). Throughout the camera motion, which takes just a few seconds, the neural radiance cache continually learns from sparse camera paths, enabling the cache to learn ("overfit") intricate detail at the scale of the content that the camera is momentarily observing.

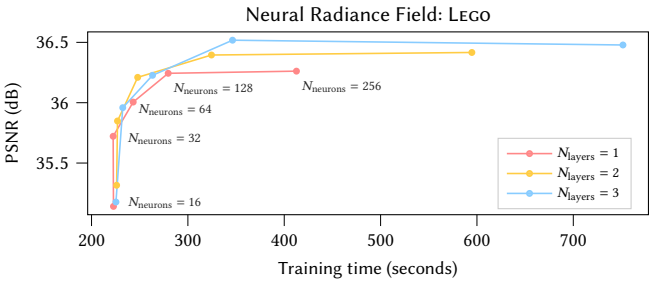

Fig. 10. The effect of the MLP size on test error vs. training time (31 000 training steps) on the Lego scene. Other scenes behave almost identically. Each curve represents a different MLP depth, where the color MLP has N_{layers} hidden layers and the density MLP has 1 hidden layer; we do not observe an improvement with deeper density MLPs. The curves sweep the number of neurons the hidden layers of the density and color MLPs from 16 to 256. Informed by this analysis, we choose $N_{\text{layers}} = 2$ and $N_{\text{neurons}} = 64$.

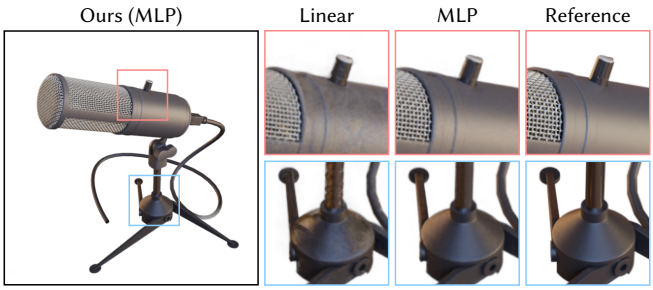

Fig. 11. Feeding the result of our encoding through a linear transformation (no neural network) versus an MLP when learning a NeRF. The models were trained for 1 min. The MLP allows for resolving specular details and reduces the amount of background noise caused by hash collisions. Due to the small size and efficient implementation of the MLP, it is only 15% more expensive—well worth the significantly improved quality.

of the finest grid resolution, which is absent in NGLOD and does not disappear with longer training times. Since NGLOD is essentially a collision-free analog to our hash encoding, we attribute this artifact to hash collisions. Upon close inspection, similar microstructure can be seen in other neural graphics primitives, although with significantly lower magnitude.

5.3 Neural Radiance Caching

In neural radiance caching [Müller et al. 2021], the task of the MLP is to predict photorealistic pixel colors from feature buffers; see Figure 8. The MLP is run independently for each pixel (i.e. the model is not convolutional), so the feature buffers can be treated as per-pixel feature vectors that contain the 3D coordinate $\mathbf{x}$ as well as additional features. We can therefore directly apply our multiresolution hash encoding to $\mathbf{x}$ while treating all additional features as auxiliary encoded dimensions ξ to be concatenated with the encoded position, using the same encoding as Müller et al. [2021]. We integrated our work into Müller et al.'s implementation of neural radiance caching and therefore refer to their paper for implementation details.

For photorealistic rendering, the neural radiance cache is typically queried only for *indirect* path contributions, which masks its reconstruction error behind the first reflection. In contrast, we would like to *emphasize* the neural radiance cache's error, and thus the improvement that can be obtained by using our multiresolution hash encoding, so we directly visualize the neural radiance cache at the first path vertex.

Figure 9 shows that—compared to the triangle wave encoding of Müller et al. [2021]—our encoding results in sharper reconstruction while incurring only a mild performance overhead of 0.7 ms that reduces the frame rate from 147 to 133 FPS at a resolution of 1920 × 1080px. Notably, the neural radiance cache is trained online—during rendering—from a path tracer that runs in the background, which means that the 0.7 ms overhead includes *both* training and runtime costs of our encoding.

5.4 Neural Radiance and Density Fields (NeRF)

In the NeRF setting, a volumetric shape is represented in terms of a spatial (3D) density function and a spatiodirectional (5D) emission function, which we represent by a similar neural network architecture as Mildenhall et al. [2020]. We train the model in the same ways as Mildenhall et al.: by backpropagating through a differentiable ray marcher driven by 2D RGB images from known camera poses.

Model Architecture. Unlike the other three applications, our NeRF model consists of two concatenated MLPs: a density MLP followed by a color MLP [Mildenhall et al. 2020]. The density MLP maps the hash encoded position $\mathbf{y} = \text{enc}(\mathbf{x}; \theta)$ to 16 output values, the first of which we treat as log-space density. The color MLP adds view-dependent color variation. Its input is the concatenation of

- the 16 output values of the density MLP, and
- the view direction projected onto the first 16 coefficients of the spherical harmonics basis (i.e. up to degree 4). This is a natural frequency encoding over unit vectors.

Its output is an RGB color triplet, for which we use either a sigmoid activation when the training data has low dynamic-range (sRGB) or an exponential activation when it has high dynamic range (linear HDR). We prefer HDR training data due to the closer resemblance to physical light transport. This brings numerous advantages as has also been noted in concurrent work [Mildenhall et al. 2021].

Informed by the analysis in Figure 10, our results were generated with a 1-hidden-layer density MLP and a 2-hidden-layer color MLP, both 64 neurons wide.

Accelerated ray marching. When marching along rays for both training and rendering, we would like to place samples such that they contribute somewhat uniformly to the image, minimizing wasted computation. Thus, we concentrate samples near surfaces by maintaining an occupancy grid that coarsely marks empty vs. non-empty space. In large scenes, we additionally cascade the occupancy grid and distribute samples exponentially rather than uniformly along the ray. Appendix E describes these procedures in detail.

At HD resolutions, synthetic and even real-world scenes can be trained in seconds and rendered at 60 FPS, without the need of caching of the MLP outputs [Garbin et al. 2021; Wizadwongsa et al. 2021; Yu et al. 2021b]. This high performance makes it tractable to add effects such as anti-aliasing, motion blur and depth of field by brute-force tracing of multiple rays per pixel, as shown in Figure 12.

Table 2. Peak signal to noise ratio (PSNR) of our NeRF implementation with multiresolution hash encoding ("Ours: Hash") vs. NeRF [Mildenhall et al. 2020], mip-NeRF [Barron et al. 2021a], and NSVF [Liu et al. 2020], which require ~hours to train (values taken from the respective papers). To demonstrate the comparatively rapid training of our method, we list its results after training for 1 s to 5 min. For each scene, we mark the methods with least error using gold ●, silver ●, and bronze ● medals. To analyze the degree to which our speedup originates from our optimized implementation vs. from our hash encoding, we also report PSNR for a nearly identical version of our implementation, in which the hash encoding has been replaced by the frequency encoding and the MLP correspondingly enlarged to match Mildenhall et al. [2020] ("Ours: Frequency"; details in Appendix D). It approaches NeRF's quality after training for just ~5 min, yet is outperformed by our full method after training for 5 s–15 s, amounting to a 20–60× improvement that can be attributed to the hash encoding.

	Mic	Ficus	Chair	Hotdog	Materials	Drums	Ship	Lego	avg.
Ours: Hash (1 s)	26.09	21.30	21.55	21.63	22.07	17.76	20.38	18.83	21.202
Ours: Hash (5 s)	32.60	30.35	30.77	33.42	26.60	23.84	26.38	30.13	29.261
Ours: Hash (15 s)	34.76	32.26	32.95	35.56	28.25	25.23	28.56	33.68	31.407
Ours: Hash (1 min)	35.92 ●	33.05 ●	34.34 ●	36.78	29.33	25.82	30.20 ●	35.63 ●	32.635 ●
Ours: Hash (5 min)	36.22	33.51 ●	35.00	37.40	29.78 ●	26.02 ●	31.10 ●	36.39 ●	33.176 ●
mip-NeRF (~hours)	36.51 ●	33.29	35.14 ●	37.48 ●	30.71	25.48 ●	30.41	35.70	33.090
NSVF (~hours)	34.27	31.23	33.19	37.14 ●	32.68 ●	25.18	27.93	32.29	31.739
NeRF (~hours)	32.91	30.13	33.00	36.18	29.62	25.01	28.65	32.54	31.005
Ours: Frequency (5 min)	31.89	28.74	31.02	34.86	28.93	24.18	28.06	32.77	30.056
Ours: Frequency (1 min)	26.62	24.72	28.51	32.61	26.36	21.33	24.32	28.88	26.669

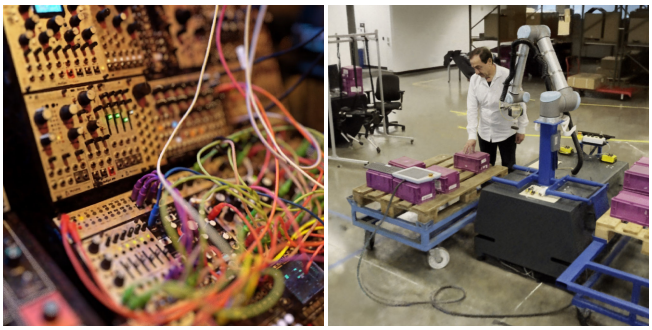

Fig. 12. NeRF reconstruction of a modular synthesizer and large natural 360 scene. The left image took 5 seconds to accumulate 128 samples at 1080p on a single RTX 3090 GPU, allowing for brute force defocus effects. The right image was taken from an interactive session running at 10 frames per second on the same GPU.

Comparison with direct voxel lookups. Figure 11 shows an ablation where we replace the entire neural network with a single linear matrix multiplication, in the spirit of (although not identical to) concurrent direct voxel-based NeRF [Sun et al. 2021; Yu et al. 2021a]. While the linear layer is capable of reproducing view-dependent effects, the quality is significantly compromised as compared to the MLP, which is better able to capture specular effects and to resolve hash collisions across the interpolated multiresolution hash tables (which manifest as high-frequency artifacts). Fortunately, the MLP is only 15% more expensive than the linear layer, thanks to its small size and efficient implementation.

Comparison with high-quality offline NeRF. In Table 2, we compare the peak signal to noise ratio (PSNR) our NeRF implementation with multiresolution hash encoding ("Ours: Hash") with that of NeRF [Mildenhall et al. 2020], mip-NeRF [Barron et al. 2021a], and NSVF [Liu et al. 2020], which all require on the order of hours to train. In contrast, we list results of our method after training for 1 s to 5 min. Our PSNR is competitive with NeRF and NSVF after just 15 s of training, and competitive with mip-NeRF after 1 min to 5 min of training.

On one hand, our method performs best on scenes with high geometric detail, such as Ficus, Drums, Ship and Lego, achieving the best PSNR of all methods. On the other hand, mip-NeRF and NSVF outperform our method on scenes with complex, view-dependent reflections, such as Materials; we attribute this to the much smaller MLP that we necessarily employ to obtain our speedup of several orders of magnitude over these competing implementations.

Next, we analyze the degree to which our speedup originates from our efficient implementation versus from our encoding. To this end, we additionally report PSNR for a nearly identical version of our implementation: we replace the hash encoding by the frequency encoding and enlarge the MLP to approximately match the architecture of Mildenhall et al. [2020] ("Ours: Frequency"); see Appendix D for details. This version of our algorithm approaches NeRF's quality after training for just ~5 min, yet is outperformed by our full method after training for a much shorter duration (5 s–15 s), amounting to a 20–60× improvement caused by the hash encoding and smaller MLP.

For "Ours: Hash", the cost of each training step is roughly constant at ~6 ms per step. This amounts to 50 k steps after 5 min at which point the model is well converged. We decay the learning rate after 20 k steps by a factor of 0.33, which we repeat every further 10 k steps. In contrast, the larger MLP used in "Ours: Frequency" requires ~30 ms per training step, meaning that the PSNR listed after 5 min corresponds to about 10 k steps. It could thus keep improving slightly if trained for extended periods of time, as in the offline NeRF variants that are often trained for several 100 k steps.

While we isolated the performance and convergence impact of our hash encoding and its small MLP, we believe an additional study is required to quantify the impact of advanced ray marching schemes (such as ours, coarse-fine [Mildenhall et al. 2020], or DONeRF [Neff et al. 2021]) independently from the encoding and network architecture. We report additional information in Section E.3 to aid in such an analysis.

(a) Offline rendered reference

(b) Hash (ours), trained for 10 s
Rendered in 32 ms (2 samples per pixel)

(c) Path tracer
Rendered in 32 ms (16 samples per pixel)

Fig. 13. Preliminary results of training a NeRF cloud model **(b)** from real-time path tracing data. Within 32 ms, a 1024×1024 image of our model convincingly approximates the offline rendered ground truth **(a)**. Our model exhibits less noise than a GPU path tracer that ran for an equal amount of time **(c)**. The cloud data is ©Walt Disney Animation Studios (CC BY-SA 3.0)

6 DISCUSSION AND FUTURE WORK

Concatenation vs. reduction. At the end of the encoding, we *concatenate* rather than reduce (for example, by summing) the F-dimensional feature vectors obtained from each resolution. We prefer concatenation for two reasons. First, it allows for independent, fully parallel processing of each resolution. Second, a reduction of the dimensionality of the encoded result $\mathbf{y}$ from LF to F may be too small to encode useful information. While F could be increased proportionally, it would make the encoding much more expensive.

However, we recognize that there may be applications in which reduction is favorable, such as when the neural network is significantly more expensive than the encoding, in which case the added computational cost of increasing F could be insignificant. We thus argue for concatenation *by default* and not as a hard-and-fast rule. In our applications, concatenation, coupled with $F = 2$ always yielded by far the best results.

Choice of hash function. A good hash function is efficient to compute, leads to coherent look-ups, and uniformly covers the feature vector array regardless of the structure of query points. We chose our hash function for its good mixture of these properties and also experimented with three others:

(1) The PCG32 [O'Neill 2014] RNG, which has superior statistical properties. Unfortunately, it did not yield a higher-quality reconstruction, making its higher cost not worthwhile.
(2) Ordering the least significant bits of $\mathbb{Z}^d$ by a space-filling curve and only hashing the higher bits. This leads to better look-up coherence at the cost of worse reconstruction quality. However, the speed-up is only marginally better than setting $\pi_1 := 1$ as done in our hash, and is thus not worth the reduced quality.
(3) Even better coherence can be achieved by treating the hash function as a tiling of space into dense grids. Like (2), the speed-up is small in practice with significant detriment to quality.

Alternatively to hand-crafted hash functions, it is conceivable to *optimize* the hash function in future work, turning the method into a dictionary-learning approach. Two possible avenues are **(1)** developing a continuous formulation of indexing that is amenable to analytic differentiation or **(2)** applying an evolutionary optimization algorithm that can efficiently explore the discrete function space.

Microstructure due to hash collisions. The salient artifact of our encoding is a small amount of "grainy" microstructure, most visible on the learned signed distance functions (Figure 1 and Figure 7). The graininess is a result of hash collisions that the MLP is unable to fully compensate for. We believe that the key to achieving state-of-the-art quality on SDFs with our encoding will be to find a way to overcome this microstructure, for example by filtering hash table lookups or by imposing an additional smoothness prior on the loss.

Generative setting. Parametric input encodings, when used in a generative setting, typically arrange their features in a dense grid which can then be populated by a separate generator network, typically a CNN such as StyleGAN [Chan et al. 2021; DeVries et al. 2021; Peng et al. 2020b]. Our hash encoding adds an additional layer of complexity, as the features are not arranged in a regular pattern through the input domain; that is, the features are not bijective with a regular grid of points. We leave it to future work to determine how best to overcome this difficulty.

Other applications. We are interested in applying the multiresolution hash encoding to other low-dimensional tasks that require accurate, high-frequency fits. The frequency encoding originated from the attention mechanism of transformer networks [Vaswani et al. 2017]. We hope that parametric encodings such as ours can lead to a meaningful improvement in general, attention-based tasks.

Heterogenous volumetric density fields, such as cloud and smoke stored in a VDB [Museth 2013, 2021] data structure, often include empty space on the outside, a solid core on the inside, and sparse detail on the volumetric surface. This makes them a good fit for our encoding. In the code released alongside this paper, we have included a preliminary implementation that fits a radiance and density field directly from the noisy output of a volumetric path tracer. The initial results are promising, as shown in Figure 13, and we intend to pursue this direction further in future work.

7 CONCLUSION

Many graphics problems rely on task specific data structures to exploit the sparsity or smoothness of the problem at hand. Our multi-resolution hash encoding provides a practical learning-based

alternative that automatically focuses on relevant detail, independent of the task. Its low overhead allows it to be used even in time-constrained settings like online training and inference. In the context of neural network input encodings, it is a drop-in replacement, for example speeding up NeRF by several orders of magnitude and matching the performance of concurrent non-neural 3D reconstruction techniques.

Slow computational processes in any setting, from lightmap baking to the training of neural networks, can lead to frustrating workflows due to long iteration times [Enderton and Wexler 2011]. We have demonstrated that single-GPU training times measured in seconds are within reach for many graphics applications, allowing neural approaches to be applied where previously they may have been discounted.

ACKNOWLEDGMENTS

We are grateful to Andrew Tao, Andrew Webb, Anjul Patney, David Luebke, Fabrice Rousselle, Jacob Munkberg, James Lucas, Jonathan Granskog, Jonathan Tremblay, Koki Nagano, Marco Salvi, Nikolaus Binder, and Towaki Takikawa for profound discussions, proofreading, feedback, and early testing. We also thank Arman Toorians and Saurabh Jain for the factory robot dataset in Figure 12 (right).

REFERENCES

Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. 2007. Convolution Shadow Maps. In *Rendering Techniques*, Jan Kautz and Sumanta Pattanaik (Eds.). The Eurographics Association. https://doi.org/10.2312/EGWR/EGSR07/051-060

Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. 2021a. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. *arXiv* (2021). https://jonbarron.info/mipnerf/

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2021b. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *arXiv:2111.12077* (Nov. 2021).

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 *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 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. 2021. Efficient Geometry-aware 3D Generative Adversarial Networks. *arXiv:2112.07945* (2021). arXiv:2112.07945 [cs.CV]

Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. 2020. Implicit Functions in Feature Space for 3D Shape Reconstruction and Completion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Terrance DeVries, Miguel Angel Bautista, Nitish Srivastava, Graham W. Taylor, and Joshua M. Susskind. 2021. Unconstrained Scene Generation with Locally Conditioned Radiance Fields. *arXiv* (2021).

Eric Enderton and Daniel Wexler. 2011. The Workflow Scale. In *Computer Graphics International Workshop on VFX, Computer Animation, and Stereo Movies*.

Alex Evans. 2006. Fast Approximations for Global Illumination on Dynamic Scenes. In *ACM SIGGRAPH 2006 Courses* (Boston, Massachusetts) *(SIGGRAPH '06)*. Association for Computing Machinery, New York, NY, USA, 153–171. https://doi.org/10.1145/1185657.1185834

Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. 2021. FastNeRF: High-Fidelity Neural Rendering at 200FPS. *arXiv:2103.10380* (March 2021).

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 1243—-1252.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proc. 13th International Conference on Artificial Intelligence and Statistics* (Sardinia, Italy, May 13–15). JMLR.org, 249–256.

Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. 2021. Neural radiosity. *ACM Transactions on Graphics* 40, 6 (Dec. 2021), 1—-11. https://doi.org/10.1145/3478513.3480569

David Money Harris and Sarah L. Harris. 2013. 3.4.2 - State Encodings. In *Digital Design and Computer Architecture* (second ed.). Morgan Kaufmann, Boston, 129–131. https://doi.org/10.1016/B978-0-12-394424-5.00002-1

Jon Jansen and Louis Bavoil. 2010. Fourier Opacity Mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Washington, D.C.) *(I3D '10)*. Association for Computing Machinery, New York, NY, USA, 165—-172. https://doi.org/10.1145/1730804.1730831

Chiyu Max Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. 2020. Local Implicit Grid Representations for 3D Scenes. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* (June 2014).

Derrick H. Lehmer. 1951. Mathematical Methods in Large-scale Computing Units. In *Proceedings of the Second Symposium on Large Scale Digital Computing Machinery*. Harvard University Press, Cambridge, United Kingdom, 141–146.

Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. 2018. Noise2Noise: Learning Image Restoration without Clean Data. *arXiv:1803.04189* (March 2018).

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. *NeurIPS* (2020). https://lingjie0206.github.io/papers/NSVF/

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).

Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. 2021. Modulated Periodic Activations for Generalizable Local Functional Representations. In *IEEE International Conference on Computer Vision*. IEEE.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. *arXiv:1710.03740* (Oct. 2018).

Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul Srinivasan, and Jonathan T. Barron. 2021. NeRF in the Dark: High Dynamic Range View Synthesis from Noisy Raw Images. *arXiv:2111.13679* (Nov. 2021).

Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. 2019. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *ACM Trans. Graph.* 38, 4, Article 29 (July 2019), 14 pages. https://doi.org/10.1145/3306346.3322980

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 *ECCV*.

Thomas Müller. 2021. Tiny CUDA Neural Network Framework. https://github.com/nvlabs/tiny-cuda-nn.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. *ACM Trans. Graph.* 38, 5, Article 145 (Oct. 2019), 19 pages. https://doi.org/10.1145/3341156

Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. 2020. Neural Control Variates. *ACM Trans. Graph.* 39, 6, Article 243 (Nov. 2020), 19 pages. https://doi.org/10.1145/3414685.3417804

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. 2013. VDB: High-Resolution Sparse Volumes with Dynamic Topology. *ACM Trans. Graph.* 32, 3, Article 27 (July 2013), 22 pages. https://doi.org/10.1145/2487228.2487235

Ken Museth. 2021. NanoVDB: A GPU-Friendly and Portable VDB Data Structure For Real-Time Rendering And Simulation. In *ACM SIGGRAPH 2021 Talks* (Virtual Event, USA) *(SIGGRAPH '21)*. Association for Computing Machinery, New York, NY, USA, Article 1, 2 pages. https://doi.org/10.1145/3450623.3464653

Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. 2021. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *Computer Graphics Forum* 40, 4 (2021). https://doi.org/10.1111/cgf.14340

Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. 2013. Real-Time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Trans. Graph.* 32, 6, Article 169 (nov 2013), 11 pages. https://doi.org/10.1145/2508363.2508374

Fakir S. Nooruddin and Greg Turk. 2003. Simplification and Repair of Polygonal Models Using Volumetric Techniques. *IEEE Transactions on Visualization and Computer Graphics* 9, 2 (apr 2003), 191––205. https://doi.org/10.1109/TVCG.2003.1196006

Melissa E. O'Neill. 2014. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*. Technical Report HMC-CS-2014-0905. Harvey Mudd College, Claremont, CA.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape

Representation. *arXiv:1901.05103* (Jan. 2019).
Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. 2020a. Convolutional Occupancy Networks. In *European Conference on Computer Vision (ECCV)*.
Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. 2020b. Convolutional Occupancy Networks. (2020). arXiv:2003.04618 [cs.CV]
Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation (3rd ed.)* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 1266 pages.
Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Proc. NeurIPS*.
Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2021. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. *arXiv:2111.11215* (Nov. 2021).
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. (2021).
Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. *NeurIPS* (2020). https://bmild.github.io/fourfeat/index.html
Danhang Tang, Mingsong Dou, Peter Lincoln, Philip Davidson, Kaiwen Guo, Jonathan Taylor, Sean Fanello, Cem Keskin, Adarsh Kowdle, Sofien Bouaziz, Shahram Izadi, and Andrea Tagliasacchi. 2018. Real-Time Compression and Streaming of 4D Performances. *ACM Trans. Graph.* 37, 6, Article 256 (dec 2018), 11 pages. https://doi.org/10.1145/3272127.3275096
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.
Sergios Theodoridis. 2008. *Pattern Recognition*. Elsevier.
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *arXiv:1706.03762* (June 2017).
Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. 2021. Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields. *arXiv:2112.03907* (Dec. 2021).
Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. 2021. NeX: Real-time View Synthesis with Neural Basis Expansion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2021a. Plenoxels: Radiance Fields without Neural Networks. *arXiv:2112.05131* (Dec. 2021).
Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021b. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *ICCV*.

A SMOOTH INTERPOLATION

One may desire smoother interpolation than the d-linear interpolation that our multiresolution hash encoding uses by default.

In this case, the obvious solution would be using a d-quadratic or d-cubic interpolation, both of which are however very expensive due to requiring the lookup of 3^d and 4^d instead of 2^d vertices, respectively. As a low-cost alternative, we recommend applying the smoothstep function,

$$S_1(x) = x^2(3 - 2x)\,, \tag{5}$$

to the d-linear interpolation weights. Crucially, the derivative of the smoothstep,

$$S_1'(x) = 6x(1 - x)\,, \tag{6}$$

vanishes at 0 and at 1, causing the discontinuity in the derivatives of the encoding to vanish by the chain rule. The encoding thus becomes C^1-smooth.

However, by this trick, we have merely traded discontinuities for zero-points in the individual levels which are not necessarily more desirable. So, we offset each level by half of its voxel size $1/(2N_l)$, which prevents the zero derivatives from aligning across all levels. The encoding is thus able to learn smooth, non-zero derivatives for all spatial locations $\mathbf{x}$.

For higher-order smoothness, higher-order smoothstep functions S_n can be used at small additional cost. In practice, the computational cost of the 1st order smoothstep function S_1 is hidden by memory bottlenecks, making it essentially free. However, the reconstruction quality tends to decrease as higher-order interpolation is used. This is why we do not use it by default. Future research is needed to explain the loss of quality.

B IMPLEMENTATION DETAILS OF NGLOD

We designed our implementation of NGLOD [Takikawa et al. 2021] such that it closely resembles that of our hash encoding, only differing in the underlying data structure; i.e. using the vertices of an octree around ground-truth triangle mesh to store collision-free feature vectors, rather than relying on hash tables. This results in a notable difference to the original NGLOD: the looked-up feature vectors are concatenated rather than summed, which in our implementation serendipitously resulted in higher reconstruction quality compared to the summation of an equal number of trainable parameters.

The octree implies a fixed growth factor $b = 2$, which leads to a smaller number of levels than our hash encoding. We obtained the most favorable performance vs. quality trade-off at a roughly equal number of trainable parameters as our method, through the following configuration:

(1) the number of feature dimensions per entry is $F = 8$,
(2) the number of levels is $L = 10$, and
(3) look-ups start at level $l = 4$.

The last point is important for two reasons: first, it matches the coarsest resolution of our hash tables $2^4 = 16 = N_{\text{min}}$, and second, it prevents a performance bottleneck that would arise when all threads of the GPU atomically accumulate gradients in few, coarse entries. We experimentally verified that this does not lead to reduced quality, compared to looking up the entire hierarchy.

C REAL-TIME SDF TRAINING DATA GENERATION

In order to not bottleneck our SDF training, we must be able to generate a large number of ground truth signed distances to high-resolution meshes very quickly (~millions per second).

C.1 Efficient Sampling of 3D Training Positions

Similar to prior work [Takikawa et al. 2021], we distribute some (1/8th) of our training positions uniformly in the unit cube, some (4/8ths) uniformly on the surface of the mesh, and the remainder (3/8ths) *perturbed* from the surface of the mesh.

The uniform samples in the unit cube are trivial to generate using any pseudorandom number generator; we use a GPU implementation of PCG32 [O'Neill 2014].

To generate the uniform samples *on the surface of the mesh*, we compute the area of each triangle in a preprocessing step, normalize the areas to represent a probability distribution, and store the corresponding cumulative distribution function (CDF) in an array. Then, for each sample, we select a triangle proportional to its area by the inversion method—a binary search of a uniform random number

over the CDF array—and sample a uniformly random position *on that triangle* by standard sample warping [Pharr et al. 2016].

Lastly, for those surface samples that must be perturbed, we add a random 3D vector, each dimension independently drawn from a logistic distribution (similar shape to a Gaussian, but cheaper to compute) with standard deviation $r/1024$, where r is the bounding radius of the mesh.

Octree sampling for NGLOD. When training our implementation of Takikawa et al. [2021], we must be careful to rarely generate training positions outside of octree leaf nodes. To this end, we replace the uniform unit cube sampling routine with one that creates uniform 3D positions in the leaf nodes of the octree by first rejection sampling a uniformly random leaf node from the array of all nodes and then generating a uniform random position within the node's voxel. Fortunately, the standard deviation $r/1024$ of our logistic perturbation is small enough to almost never leave the octree, so we do not need to modify the surface sampling routine.

C.2 Efficient Signed Distances to the Triangle Mesh

For each sampled 3D position $\mathbf{x}$, we must compute the signed distance to the triangle mesh. To this end, we first construct a triangle bounding volume hierarchy (BVH) with which we perform efficient *unsigned* distance queries; $O\left(\log N_{\text{triangles}}\right)$ on average.

Next, we *sign* these distances by tracing 32 "stab rays" [Nooruddin and Turk 2003], which we distribute uniformly over the sphere using a Fibonacci lattice that is pseudorandomly and independently offset for every training position. If any of these rays reaches infinity, the corresponding position $\mathbf{x}$ is deemed "outside" of the object and the distance is marked positive. Otherwise, it is marked negative.[3]

For maximum efficiency, we use NVIDIA ray tracing hardware through the OptiX 7 framework, which is over an order of magnitude faster than using the previously mentioned triangle BVH for ray-shape intersections on our RTX 3090 GPU.

D BASELINE MLPS WITH FREQUENCY ENCODING

In our signed distance function (SDF), neural radiance caching (NRC), and neural radiance and density fields (NeRF) experiments, we use an MLP prefixed by a frequency encoding as baseline. The respective architectures are equal to those in the main text, except that the MLPs are larger and that the hash encoding is replaced by sine and cosine waves (SDF and NeRF) or triangle waves (NRC). The following table lists the number of hidden layers, neurons per hidden layer, frequency cascades (each scaled by a factor of 2 as per Vaswani et al. [2017]), and adjusted learning rates.

Primitive	Hidden layers	Neurons	Frequencies	Learning rate
SDF	8	128	10	$3 \cdot 10^{-4}$
NRC	3	64	10	10^{-2}
NeRF	7 / 1	256 / 256	16 / 4	10^{-3}

For NeRF, the first listed number corresponds to the density MLP and the second number to the color MLP. For SDFs, we make two additional changes: **(1)** we optimize against the relative $\mathcal{L}^2$ loss [Lehtinen et al. 2018] instead of the MAPE described in the main text, and **(2)** we perturb training samples with a standard deviation of $r/128$ as opposed to the value of $r/1024$ from Appendix C.1. Both changes smooth the loss landscape, resulting in a better reconstruction with the above configuration.

Notably, even though the above configurations have fewer parameters *and* are slower than our configurations with hash encoding, they represent favorable performance vs. quality trade-offs. An equal parameter count comparison would make pure MLPs too expensive due to their scaling with $O(n^2)$ as opposed to the sub-linear scaling of trainable encodings. On the other hand, an equal throughput comparison would require prohibitively small MLPs, thus underselling the reconstruction quality that pure MLPs are capable of.

We also experimented with Fourier features [Tancik et al. 2020] but did not obtain better results compared to the axis-aligned frequency encodings mentioned previously.

E ACCELERATED NERF RAY MARCHING

The performance of ray marching algorithms such as NeRF strongly depends on the marching scheme. We utilize three techniques with imperceivable error to optimize our implementation:

(1) exponential stepping for large scenes,
(2) skipping of empty space and occluded regions, and
(3) compaction of samples into dense buffers for efficient execution.

E.1 Ray Marching Step Size and Stopping

In synthetic NeRF scenes, which we bound to the unit cube $[0, 1]^3$, we use a fixed ray marching step size equal to $\Delta t := \sqrt{3}/1024$; $\sqrt{3}$ represents the diagonal of the unit cube.

In all other scenes, based on the intercept theorem[4], we set the step size *proportional* to the distance t along the ray $\Delta t := t/256$, clamped to the interval $\left[\sqrt{3}/1024, s \cdot \sqrt{3}/1024\right]$, where s is size of the largest axis of the scene's bounding box. This choice of step size exhibits exponential growth in t, which means that the computation cost grows only logarithmically in scene diameter, with no perceivable loss of quality.

Lastly, we stop ray marching and set the remaining contribution to zero as soon as the transmittance of the ray drops below a threshold; in our case $\epsilon = 10^{-4}$.

Related work. Mildenhall et al. [2019] already identified a non-linear step size as benefitial: they recommend sampling uniformly in the disparity-space of the average camera frame, which is more aggressive than our exponential stepping, requiring on one hand only a constant number of steps, but on the other hand can lead to a loss of fidelity compared to exponential stepping [Neff et al. 2021].

In addition to non-linear stepping, some prior methods propose to warp the 3D domain of the scene towards the origin, thereby improving the numerical properties of their input encodings [Barron et al. 2021b; Mildenhall et al. 2020; Neff et al. 2021]. This causes rays to curve, which leads to a worse reconstruction in our implementation.

[3] If the mesh is watertight, it is cheaper to sign the distance based on the normal(s) of the closest triangle(s) from the previous step. We also implemented this procedure, but disable it by default due to its incompatibility with typical meshes in the wild.

[4] The appearance of objects stays the same as long as their size and distance from the observer remain proportional.

In contrast, we *linearly* map input coordinates into the unit cube before feeding them to our hash encoding, relying on its exponential multiresolution growth to reach a proportionally scaled maximum resolution $N_{\max}$ with a constant number of levels (variable b as in Equation (3)) or logarithmically many levels L (constant b).

E.2 Occupancy Grids

To skip ray marching steps in empty space, we maintain a cascade of K multiscale occupancy grids, where $K = 1$ for all synthetic NeRF scenes (single grid) and $K \in [1, 5]$ for larger real-world scenes (up to 5 grids, depending on scene size). Each grid has a resolution of 128^3, spanning a geometrically growing domain $[-2^{k-1} + 0.5, 2^{k-1} + 0.5]^3$ that is centered around $(0.5, 0.5, 0.5)$.

Each grid cell stores occupancy as a single bit. The cells are laid out in Morton (z-curve) order to facilitate memory-coherent traversal by a digital differential analyzer (DDA). During ray marching, whenever a sample is to be placed according to the step size from the previous section, the sample is skipped if its grid cell's bit is low.

Which one of the K grids is queried is determined by both the sample position $\mathbf{x}$ and the step size Δt: among the grids covering $\mathbf{x}$, the finest one with cell side-length larger than Δt is queried.

Updating the occupancy grids. To continually update the occupancy grids while training, we maintain a second set of grids that have the same layout, except that they store full-precision floating point density values rather than single bits.

We update the grids after every 16 training iterations by performing the following steps. We

(1) decay the density value in each grid cell by a factor of 0.95,
(2) randomly sample M candidate cells, and set their value to the maximum of their current value and the density component of the NeRF model at a random location within the cell, and
(3) update the occupancy bits by thresholding each cell's density with $t = 0.01 \cdot 1024/\sqrt{3}$, which corresponds to thresholding the opacity of a minimal ray marching step by $1 - \exp(-0.01) \approx 0.01$.

The sampling strategy of the M candidate cells depends on the training progress since the occupancy grid does not store reliable information in early iterations. During the first 256 training steps, we sample $M = K \cdot 128^3$ cells uniformly without repetition. For subsequent training steps we set $M = K \cdot 128^3/2$ which we partition into two sets. The first $M/2$ cells are sampled uniformly among all cells. Rejection sampling is used for the remaining samples to restrict selection to cells that are currently occupied.

Related work. The idea of constraining the MLP evaluation to occupied cells has already been exploited in prior work on trainable, cell-based encodings [Liu et al. 2020; Sun et al. 2021; Yu et al. 2021a,b]. In contrast to these papers, our occupancy grid is independent from the learned encoding, allowing us to represent it more compactly as a bitfield (and thereby at a resolution that is decoupled from that of the encoding) and to utilize it when comparing against other methods that do not have a trained spatial encoding, e.g. "Ours: Frequency" in Table 2.

Empty space can also be skipped by importance sampling the depth distribution, such as by resampling the result of a coarse prediction [Mildenhall et al. 2020] or through neural importance sampling [Müller et al. 2019] as done in DONeRF [Neff et al. 2021].

Table 3. Batch size, number of rays per batch, and number of samples per ray for our full method ("Ours: Hash"), our implementation of frequency encoding NeRF ("Ours: Freq.") and mip-NeRF. Since the values corresponding to our method vary by scene, we report minimum and maximum values over the synthetic scenes from Table 2.

Method	Batch size	=	Samples per ray	×	Rays per batch
Ours: Hash	256 Ki		3.1 to 25.7		10 Ki to 85 Ki
Ours: Freq.	256 Ki		2.5 to 9		29 Ki to 105 Ki
mip-NeRF	1 Mi		128 coarse + 128 fine		4 Ki

E.3 Number of Rays Versus Batch Size

The batch size has a significant effect on the quality and speed of NeRF convergence. We found that training from a larger number of rays, i.e. incorporating more viewpoint variation into the batch, converged to lower error in fewer steps. In our implementation where the number of samples per ray is variable due to occupancy, we therefore include as many rays as possible in batches of fixed size rather than building variable-size batches from a fixed ray count.

In Table 3, we list ranges of the resulting number of rays per batch and corresponding samples per ray. We use a batch size of 256 Ki, which resulted in the fastest wall-clock convergence in our experiments. This is 4× smaller than the batch size chosen in mip-NeRF, likely due to the larger number of samples each of their rays requires. However, due to the myriad other differences across implementations, a more detailed study must be carried out to draw a definitive conclusion.

Lastly, we note that the occupancy grid in our frequency-encoding baseline ("Ours: Freq."; Appendix D) produces even *fewer* samples than when used alongside our hash encoding. This can be explained by the slightly more detailed reconstruction of the hash encoding: when the extra detail is finer than the occupancy grid resolution, its surrounding empty space can not be effectively culled away and must be traversed by extra steps.