KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs

Christian Reiser\textsuperscript{1,2} Songyou Peng\textsuperscript{1,3} Yiyi Liao\textsuperscript{1,2} Andreas Geiger\textsuperscript{1,2}
\textsuperscript{1}Max Planck Institute for Intelligent Systems, Tübingen \textsuperscript{2}University of Tübingen \textsuperscript{3}ETH Zurich
\{firstname.lastname\}@tue.mpg.de

Abstract

NeRF synthesizes novel views of a scene with unprecedented quality by fitting a neural radiance field to RGB images. However, NeRF requires querying a deep Multi-Layer Perceptron (MLP) millions of times, leading to slow rendering times, even on modern GPUs. In this paper, we demonstrate that real-time rendering is possible by utilizing thousands of tiny MLPs instead of one single large MLP. In our setting, each individual MLP only needs to represent parts of the scene, thus smaller and faster-to-evaluate MLPs can be used. By combining this divide-and-conquer strategy with further optimizations, rendering is accelerated by three orders of magnitude compared to the original NeRF model without incurring high storage costs. Further, using teacher-student distillation for training, we show that this speed-up can be achieved without sacrificing visual quality.

1. Introduction

Novel View Synthesis (NVS) addresses the problem of rendering a scene from unobserved viewpoints, given a number of RGB images and camera poses as input, e.g., for interactive exploration. Recently, NeRF \cite{mildenhall2020nerf} demonstrated state-of-the-art results on this problem using a neural radiance field representation for representing 3D scenes. NeRF produces geometrically consistent, high-quality novel views even when faced with challenges like thin structures, semi-transparent objects and reflections. Additionally, NeRF’s underlying representation requires only very little storage and thus can be easily streamed to users.

The biggest remaining drawbacks of NeRF are its long training and rendering times. While training can be sped up with a multi-GPU cluster, rendering must happen in real-time on the consumer’s device for interactive applications like virtual reality. This motivates us to focus on increasing NeRF’s rendering speed in this paper.

NeRF represents the scene’s geometry and appearance with a Multi-Layer Perceptron (MLP). During volumetric rendering, this network is sampled hundreds of times for millions of pixels. As the MLP used in NeRF is relatively deep and wide, this process is very slow. A natural idea is to decrease the depth and number of hidden units per layer in order to speed up the rendering process. However, without any further measures, a reduction in network size leads to an immediate loss in image quality due to the limited capacity for fitting complex scenes. We counteract this by using a large number of independent and small networks, and by letting each network represent only a fraction of the scene.

We find that training our KiloNeRF with thousands of networks, naively from scratch leads to noticeable artifacts. To overcome this problem, we first train a regular NeRF as teacher model. KiloNeRF is then trained such that its outputs (density and color) match those of the teacher model for any position and view direction. Finally, KiloNeRF is fine-tuned on the original training images. Thanks to this three-stage training strategy, our model reaches the same
visual fidelity as the original NeRF model, while being able to synthesize novel views three orders of magnitude faster as illustrated in Fig. 1. Crucial for achieving our speedup is an adequate implementation of the concurrent evaluation of many MLPs. Towards this goal, we published our efficient implementation using PyTorch, MAGMA, Thrust and custom CUDA kernels at https://github.com/creiser/kilonerf.

2. Related Work

Novel View Synthesis: NVS methods can be categorized according to the representations they use to model the underlying scene geometry. Mesh-based methods, including classical [5, 9, 65, 69] and learning-based [45, 46, 61] ones, typically require a preprocessing step, e.g., Structure from Motion (SfM) [49], to reconstruct the scene. Similarly, point cloud-based methods rely on SfM or RGB-D sensors to recover geometry [2]. In contrast, approaches using multi-plane images (MPIs) [11, 28, 55, 56, 64, 78] represent the scene as a stack of images or feature maps. While MPI approaches demonstrate photorealistic image synthesis, they only allow for small viewpoint changes during inference. Another line of methods considers voxel grids as scene representation [19, 25, 40, 51, 52, 58] which, however, are typically restricted in terms of their resolution.

The limitations of voxel grids can be alleviated by adopting neural function representations [29, 32, 53, 72]. Differentiable Volumetric Rendering (DVR) [32] and Implicit Differentiable Renderer (IDR) [72] adopt surface rendering and therefore rely on pixel-accurate object masks as input during training. Furthermore, they assume solid scenes and cannot handle semi-transparent objects or thin structures well. In contrast, NeRF [29] uses volumetric rendering [3] which enables training without masks and allows for recovering fine structures using alpha blending.

One reason for NeRF’s success lies in its representation, which utilizes a parametric function to map 3D coordinates and viewing directions to volumetric densities and color values. Such function representations have a long tradition [47, 48] and have recently resurfaced in geometric computer vision [7, 27, 35]. These recent methods use deep neural networks as function class and have advanced the underlying scene geometry. Mesh-based methods, including classical [5, 9, 65, 69] and learning-based [45, 46, 61] ones, typically require a preprocessing step, e.g., Structure from Motion (SfM) [49], to reconstruct the scene. Similarly, point cloud-based methods rely on SfM or RGB-D sensors to recover geometry [2]. In contrast, approaches using multi-plane images (MPIs) [11, 28, 55, 56, 64, 78] represent the scene as a stack of images or feature maps. While MPI approaches demonstrate photorealistic image synthesis, they only allow for small viewpoint changes during inference. Another line of methods considers voxel grids as scene representation [19, 25, 40, 51, 52, 58] which, however, are typically restricted in terms of their resolution.

The limitations of voxel grids can be alleviated by adopting neural function representations [29, 32, 53, 72]. Differentiable Volumetric Rendering (DVR) [32] and Implicit Differentiable Renderer (IDR) [72] adopt surface rendering and therefore rely on pixel-accurate object masks as input during training. Furthermore, they assume solid scenes and cannot handle semi-transparent objects or thin structures well. In contrast, NeRF [29] uses volumetric rendering [3] which enables training without masks and allows for recovering fine structures using alpha blending.

One reason for NeRF’s success lies in its representation, which utilizes a parametric function to map 3D coordinates and viewing directions to volumetric densities and color values. Such function representations have a long tradition [47, 48] and have recently resurfaced in geometric computer vision [7, 27, 35]. These recent methods use deep neural networks as function class and have advanced the state-of-the-art in tasks like 3D reconstruction, shape generation, point cloud completion or relighting [7, 27, 33, 35, 38].

Faster NeRF Rendering: Neural Sparse Voxel Fields (NSVF) [24] speed up NeRF’s rendering using classical techniques like empty space skipping and early ray termination. Additionally, NSVF’s network is conditioned on feature vectors located on a uniform 3D grid to increase the model’s capacity without increasing network depth or width. By applying empty space skipping already during training, NSVF can sample more densely around surfaces at the same computational budget. We also make use of empty space skipping during training and early ray termination during rendering. The key difference is that we use thousands of small networks, while NSVF uses a single feature-conditioned network. Since our networks are only responsible for representing a small region, they require lower capacity compared to NSVF’s single network that must represent the entire scene. As a consequence, KiloNeRF renders views two orders of magnitude faster than NSVF.

Concurrently to us, numerous works [14, 16, 23, 30, 43, 74] were developed with the purpose of speeding up NeRF. DeRF [43] also represents the scene by a number of independent networks. However, in DeRF the scene is decomposed into sixteen irregular Voronoi cells. In this paper, we demonstrate that a much simpler strategy of decomposing the scene into thousands of MLPs arranged on a regular 3D grid leads to significantly higher speedups. AutoInt replaces the need for numerical integration in NeRF by learning a closed form solution of the antiderivative [23]. By making use of the fundamental theorem of calculus, a pixel is rendered with a significantly smaller amount of network queries. DONeRF [30] demonstrates the feasibility to render a pixel using only 4 samples on the ray. This is achieved by placing samples more closely around the first surface that the ray intersects. Towards this goal, for each ray an additional network is queried, that directly predicts suitable sample locations. We note that the latter two approaches are orthogonal to KiloNeRF and the combination of KiloNeRF with either of these new techniques is promising. Other works show that real-time rendering can be achieved by converting the neural representation into a discrete one after training [14, 16, 74]. In comparison to KiloNeRF, these approaches consume significantly more GPU memory which might make them less suitable for larger scenes.

NeRF Follow-ups: NeRF++ extends NeRF to unbounded scenes [76] while NeRF-W tackles unstructured photo collections [26]. GRAF [50], pi-GAN [6] and GIRAFFE [31] propose generative models of radiance fields. A series of works improve generalization from limited number of training views [13, 42, 44, 59, 63, 66, 75] while others [57, 68, 73] remove the requirement for pose estimation. [4, 15, 54] enable learning from images with varying light conditions. Finally, [10, 12, 21, 22, 34, 36, 39, 41, 62, 67, 70] extend NeRF to videos. Many of these methods would benefit from faster rendering and are compatible with KiloNeRF.

3. Method

Our method builds upon NeRF [29], which represents a single scene as a collection of volumetric density and color values. Crucially, in NeRF those densities and colors are not stored in a discrete voxel representation. Instead, NeRF encodes densities and colors at any continuous 3D position in the scene using an MLP. While NeRF uses a single net-
work to represent the entire scene, we are inspired by [8] and represent the scene with a large number of independent and small MLPs. More specifically, we subdivide the scene into a 3D grid. Each MLP is tasked to represent the part of the scene that falls within a particular 3D cell. Since each network only represents a small portion of the scene, a low network capacity is sufficient for photo-realistic results as demonstrated by our experiments. Additionally, we employ empty space skipping and early ray termination to speed up rendering further. We start with a brief review of the original NeRF model which forms the basis for KiloNeRF.

3.1. Background

In NeRF [29], the scene is represented by a neural network \( f_\theta \) with parameters \( \theta \). \( f_\theta \) takes as input a 3D position \( x \) and viewing direction \( d \) and maps them to a color \( c \) and density \( \sigma \). The architecture of \( f_\theta \) is chosen such that only the color \( c \) depends on the viewing direction \( d \). This allows modeling of view-dependent effects like specularities and reflections while also encouraging a consistent geometry to be learned. In a deterministic pre-processing step, \( x \) and \( d \) are transformed by a positional encoding \( \gamma \) which promotes learning of high-frequency details, see [29, 60] for details.

A pixel is rendered by shooting a ray from the camera’s eye through the pixel’s center and evaluating the network \( f_\theta \) for \( K \) sample points \( x_1, \ldots, x_K \) along the ray. For each sample \( x_i \), the network outputs a color \( c_i \) and a density value \( \sigma_i \). In other words, we compute a list of tuples:

\[
(c_i, \sigma_i) = f_\theta(x_i, d) \quad \text{with} \quad i = 1, 2, \ldots, K \tag{1}
\]

Here, the viewing direction \( d \) is computed by normalizing the vector from the camera center to the pixel. Subsequently, the final pixel color \( \hat{c} \) is calculated by \( \alpha \)-blending the color values \( c_1, \ldots, c_K \)

\[
\hat{c} = \sum_{i=1}^{K} T_i \alpha_i c_i \tag{2}
\]

\[
T_i = \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{3}
\]

\[
\alpha_i = 1 - \exp(\sigma_i \delta_i) \tag{4}
\]

where \( \alpha_i \) determines the alpha value used for blending the color values and the calculation of \( \alpha_i \) depends on the distance between adjacent sample points \( \delta_j = \|x_{j+1} - x_j\| \). Finally, \( T_i \) corresponds to the transmittance which accounts for occlusion along the ray. This procedure needs to be repeated for every pixel in the image. Consequently, \( W \times H \times K \) network evaluations are required for rendering an image of width \( W \) and height \( H \).

NeRF is trained by minimizing a photometric loss between rendered images and training images. For each parameter update step, we randomly sample a training image and \( B \) pixels \( c_1, \ldots, c_B \) inside that image. Subsequently, the corresponding pixels \( \hat{c}_1, \ldots, \hat{c}_B \) are rendered according to Eq. 2. The model parameters \( \theta \) are optimized by minimizing an \( L_2 \) reconstruction loss between the rendered pixels and their ground truth:

\[
\mathcal{L} = \frac{1}{B} \sum_b \|c_b - \hat{c}_b\|^2_2 \tag{5}
\]

3.2. KiloNeRF

KiloNeRF assumes knowledge of an axis aligned bounding box (AABB) enclosing the scene. Let \( b_{\text{min}} \) and \( b_{\text{max}} \) be the minimum and maximum bounds of this AABB. We subdivide the scene into a uniform grid of resolution \( r = (r_x, r_y, r_z) \). Each grid cell with 3D index \( i = (i_x, i_y, i_z) \) corresponds to a tiny MLP network with an independent set of parameters \( \theta(i) \). The mapping \( g \) from position \( x \) to index \( i \) of the tiny network assigned to position \( x \) is defined through spatial binning

\[
g(x) = \lfloor (x - b_{\text{min}})/(b_{\text{max}} - b_{\text{min}})/r \rfloor \tag{6}
\]

where all operations are element-wise. Querying KiloNeRF at a point \( x \) and direction \( d \) involves determining the network \( g(x) \) responsible for the respective grid cell:

\[
(c, \sigma) = f_{\theta(g(x))}(x, d) \tag{7}
\]

Network Architecture: As illustrated in Fig. 2, we use a downscaled version of the fully-connected architecture of NeRF which similarly to the NeRF architecture enforces that the predicted density is independent of the view direction. However, NeRF uses a total of 10 hidden layers where each of the first 9 hidden layers outputs a 256-dimensional feature vector and the last hidden layer outputs a 128-dimensional feature vector. In contrast, we use a much smaller MLP with only 4 hidden layers and 32 hidden units each. Due to the low depth of our network, a skip connection as in the original NeRF architecture is not required. Like in NeRF, we provide the viewing direction as an additional input to the last hidden layer. All affine layers are followed by a ReLU activation with two exceptions: The output layer that computes the RGB color \( c \) uses a sigmoid activation and no activation is applied to the feature vector that is given as input to the penultimate hidden layer.

3.3. Training with Distillation

Training a KiloNeRF from scratch can lead to artifacts in free space as visualized in Fig. 3. We observed that better results can be obtained by first training an ordinary NeRF model and distilling [17] the knowledge of this teacher into the KiloNeRF model. Here, the KiloNeRF model is trained such that its outputs match the outputs of the teacher model.
Figure 2: Model Architecture. KiloNeRF’s MLP architecture is a downscaled version of NeRF’s architecture. A forward pass through KiloNeRF’s network only requires 1/87th of the floating point operations (FLOPs) of the original architecture.

Figure 3: Distillation. (a) Training KiloNeRF from scratch can lead to artifacts in free space. (b) Distillation by imitating a pre-trained standard NeRF model mitigates this issue.

for all possible inputs. More specifically, for each of our networks, we randomly sample a batch of 3D points inside the 3D grid cell that corresponds to the respective network. These batches are augmented with viewing directions which are drawn randomly from the unit sphere. We query both the student and the teacher in order to obtain the respective densities $\sigma$ and color values $c$. The obtained $\sigma$-values are converted to $\alpha$-values according to Eq. 4, to put less emphasis on small differences between big density values.

We optimize the student’s parameters using an $L_2$ loss between the $\alpha$-values and color values predicted by the student and those obtained from the teacher. Note that for the distillation step we are neither performing volumetric rendering nor are we utilizing the training images. Therefore, distillation alone would imply that KiloNeRF’s rendering quality is upper bounded by the teacher method’s rendering quality. Hence, in a final step KiloNeRF is fine-tuned on the training images with the photometric loss from Eq. 5. From this perspective, distillation can be seen as a means to provide powerful weight initialization to KiloNeRF.

Regularization: The original NeRF architecture uses a small fraction of its total capacity to model the dependency of the color on the viewing direction as can be observed in Fig. 2 (left). Zhang et al. [76] postulate that this choice encourages that color is a simple function of the viewing direction. Further, they experimentally demonstrate how this inductive bias is crucial for avoiding artifacts in empty space [76]. Our first attempt was to replicate this strategy in KiloNeRF, i.e., we reduced the number of output features of the last hidden layer to get a similar regulatory effect. However, this lead to an overall loss in visual quality due to our small network width. We therefore instead apply $L_2$ regularization to the weights and biases of the last two layers of the network which are responsible for view-dependent modeling of the color. This strategy allows us to impose the same inductive bias as in NeRF without sacrificing visual quality. We refer to Fig. 6 in our experiments for an illustration.

3.4. Sampling

A maximum of $K$ equidistant points are sampled along the ray. To reduce the number of network queries, we make use of empty space skipping (ESS) and early ray termination (ERT). Consequently, the number of sampled points per ray is variable and the hyperparameter $K$ effectively controls the distance $\delta$ between sampled points. Thanks to the combination of ESS and ERT, only points in the vicinity of the first surface that a ray intersects are evaluated, given that the intersection is with an opaque object. We do not make use of the hierarchical sampling scheme introduced in NeRF, which also enables dense sampling in occupied space, but is costlier than ESS. Similar to NeRF [29], we use stratified sampling to avoid bias.
Empty Space Skipping: For empty space skipping, we instantiate a second uniform grid with a higher grid resolution in the given AABB. Each cell in this grid is populated with a binary value which indicates whether the scene has any content within that cell. This occupancy grid can be used to avoid querying the network in empty space [3]. Only if the cell, in which a sample point \( x \) lies, is occupied, the network gets evaluated at \( x \). We extract the occupancy grid from the trained teacher and use it for more efficient fine-tuning and inference. To populate the occupancy grid, we sample densities on a \( 3 \times 3 \times 3 \) subgrid from the teacher model for each cell of the occupancy grid. A cell is marked as occupied, if any of the evaluated densities is above a threshold \( \tau \). No dataset-specific tuning of \( \tau \) was necessary.

Early Ray Termination: If the transmittance value \( T_i \) becomes close to 0 during volumetric rendering, evaluation of subsequent sample points \( x_{i+1}, x_{i+2}, \ldots \) is unnecessary, since the contribution of corresponding color values \( c_{i+1}, c_{i+2}, \ldots \) to the final pixel value becomes vanishingly small. Therefore, once transmittance \( T_i \) falls below a threshold \( \epsilon \), sampling further points along the ray can be avoided. ERT is only used during inference and is implemented by grouping network queries according to their distance from the camera, i.e., points close to the camera are evaluated first and then successively more distant points are queried for those rays that are not terminated yet.

3.5. Implementation

Our prototype implementation is based on PyTorch [37], MAGMA [1] and Thrust. Additionally, we developed custom CUDA kernels for sampling, empty space skipping, early ray termination, positional encoding, network evaluation and alpha blending. Crucial for high performance is the proper handling of the simultaneous query of thousands of networks. To illustrate this, let us consider a single linear layer of a fully connected network which requires the computation of a single matrix multiplication. More specifically, a \( B \times I \) matrix is multiplied with an \( I \times O \) matrix, where \( B \) is the batch size and \( I/O \) is the number of input/output features. In KiloNeRF, for each network a separate matrix multiplication is required. Assuming \( N \) networks in total, we need to multiply a \( B_i \times I \) matrix with an \( I \times O \) matrix for each \( i \in 1, \ldots, N \). Since \( B_i \) varies per network, traditional batched matrix multiplications cannot be used. To tackle this problem, we exploit the HPC library MAGMA which provides a CUDA routine for this situation [1].

4. Experimental Evaluation

4.1. Setup

Datasets: Both NSVF [24] and KiloNeRF assume scenes to be bounded. We can therefore directly use the four datasets provided by NSVF, together with the given bounding boxes. Consequently, KiloNeRF is tested on both synthetic datasets (Synthetic-NeRF, Synthetic-NSVF) as well as datasets comprising real scenes (NSVF’s variants of BlendedMVS [71] and Tanks&Temples [18]). In total, we evaluate on 25 scenes.

Baselines: We compare KiloNeRF to NeRF [29] and NSVF [24]. Note that we compare to the original NeRF model which adopts the hierarchical sampling strategy despite it is not used for our teacher. NSVF is particularly interesting as a baseline, because NSVF also makes use of ESS and ERT. Hence speedups over NSVF can be largely attributed to the smaller network size used in KiloNeRF. For comparisons to other NVS techniques [25,53] we refer the reader to [24].

Hyperparameters: For each scene, we choose the network grid resolution such that the largest dimension equals 16 and the other dimensions are chosen such that the resulting cells are cubic, hence the maximally possible resolution is \( 16 \times 16 \times 16 = 4096 \). To arrive at the occupancy grid’s resolution we multiply the network grid resolution by 16, which implies that the occupancy grid resolution is upper bounded by \( 256 \times 256 \times 256 \) cells. We find that setting the threshold \( \tau \) for occupancy grid extraction to 10 works well on all datasets. We use the Adam optimizer with a learning rate of \( 5 \times 4 \), a batch size of 8192 pixels and the same learning rate decay schedule as NeRF. For \( L_2 \) regularization we use a weight of \( 1 \times 4 \). For early ray termination, we set the threshold to \( \epsilon = 0.01 \). As found in NSVF, early ray termination with this choice of \( \epsilon \) does not lead to any loss of quality [24]. Further, using the same value for \( \epsilon \) as NSVF increases comparability. The hyperparameter \( K \), which indirectly controls the distance between sample points, is set to 384. We use the same number of frequencies for the positional encoding as NeRF.

Training Schedule: We train both the NeRF baseline and KiloNeRF for a high number of iterations to find the limits of the representation capabilities of the respective architectures. In fact, we found that on some scenes NeRF takes up to 600k iterations to converge. When comparing our qualitative and quantitative results to those reported by Liu et al. [24], one might notice that we obtained much better results for the NeRF baseline, which might be explained by our longer training schedule. Training the NeRF baseline on an NVIDIA GTX 1080 Ti takes approximately 2 days. For our teacher, we train another NeRF model but without hierarchical sampling for 600k iterations as NeRF’s coarse model.
is too inaccurate for distillation and NeRF’s fine model is biased towards surfaces. This model is distilled for 150k iterations into a KiloNeRF model, which takes up to 8 hours. Finally, the KiloNeRF model is fine-tuned for 1000k iterations, which requires only 17 hours thanks to ESS and the reduced size of our MLPs.

**Measurements:** Our test system consists of an NVIDIA GTX 1080 Ti consumer GPU, an Intel i7-3770k CPU and 32GB of RAM. For each scene, inference time is measured by computing the average time over all test images.

### 4.2. Results

Table 1 shows our main results. KiloNeRF attains similar visual quality scores as the baselines NeRF and NSVF. In terms of the perceptual metric LPIPS [77], KiloNeRF even slightly outperforms the other two methods on three out of four datasets. However, on all scenes, KiloNeRF renders three orders of magnitude faster than NeRF and two orders of magnitude faster than NSVF. In Fig. 4, we show novel view synthesis results for all three methods. More qualitative results can be found in the supplementary.

### 4.3. Ablations

In Fig. 5, we conduct an ablation study on KiloNeRF using the Lego bulldozer scene as an example.

**Single Tiny MLP:** For this ablation, KiloNeRF’s network hyperparameters (number of layers/hidden units) are used, but only a single network gets instantiated. Unsurprisingly, quality suffers dramatically since a single MLP with only 6k parameters cannot accurately represent the entire scene.

**No Fine-Tuning:** For this experiment, the ultimate fine-tuning phase is omitted. Our results show that fine-tuning is crucial for achieving high quality and that the prior distillation phase only provides a good parameter initialization.

**No Distillation:** Fig. 3 compares training KiloNeRF from

<table>
<thead>
<tr>
<th>Method</th>
<th>Render time ↓</th>
<th>Speedup ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>NeRF</td>
<td>56185 ms</td>
<td>–</td>
</tr>
<tr>
<td>NeRF + ESS + ERT</td>
<td>788 ms</td>
<td>71</td>
</tr>
<tr>
<td>KiloNeRF</td>
<td>22 ms</td>
<td>2548</td>
</tr>
</tbody>
</table>

Table 2: **Speedup Breakdown.** The original NeRF model combined with KiloNeRF’s implementation of ESS and ERT is compared against the full KiloNeRF technique.
Figure 4: **Qualitative Comparison.** Novel views synthesized by NeRF, NSVF and KiloNeRF. Despite being significantly faster, KiloNeRF attains the visual quality of the baselines. The numbers in the top-right corner correspond to the average render time of the respective technique on that scene. The rendered image resolution (in pixels) is specified on the left.
Single Tiny MLP | Half Network Res. | Half #Units | No Fine-Tuning | KiloNeRF | Ground Truth

0.1624  0.0495  0.0340  0.0809  0.0284

Figure 5: **Ablation Study.** Closeups of KiloNeRF on the Lego bulldozer scene, varying different parameters of the model. The numbers in the bottom-right corner correspond to perceptual similarity (LPIPS) wrt. the ground truth, lower is better.

![Ablation Study](image)

Figure 6: **Regularization.** Without weight regularization, visible artifacts in free space regions emerge in the rendered images as the view-dependent part of the MLP has too much capacity. Adding $L_2$ regularization alleviates this problem.

(a) No Regularization  | (b) Regularization  | (c) Ground Truth

scratch against the proposed training pipeline. Both alternatives yield the same level of detail, but only when using distillation artifacts in empty space are avoided.

**No Weight Regularization:** Fig. 6 demonstrates that artifacts in free space can also emerge when $L_2$ regularization on the last two layers of the network is omitted. Consequently, both distillation and $L_2$ regularization play important roles in avoiding these artifacts.

**ESS and ERT:** Finally, we are interested in determining to which extent the combination of ESS and ERT and the reduction of network size contribute to the final speedup. Towards this goal, we run the original-size NeRF with our optimized ESS/ERT implementation and compare to the full KiloNeRF technique on the Lego bulldozer scene. As can be deduced from Table 2, the reduction in network size contributes significantly to KiloNeRF’s overall speedup.

### 5. Discussion and Future Work

While KiloNeRF is able to render medium resolution images (800 × 800) at interactive frame rates, the speedups are not sufficient yet for real-time rendering of full HD images. Further speedups might be possible by scaling our approach to a higher number of smaller networks. Naïvely doing this, however, would lead to a higher storage impact. This might be mitigated by using memory-efficient data structures [8, 20] that allow networks to be exclusively instantiated in the vicinity of surfaces.

Furthermore, in this work, we mainly focused on making network queries faster by using many small MLPs. In contrast, the concurrently developed techniques AutoInt [23] and DONeRF [30] speed up inference by reducing the number of required samples along the ray. Therefore, the combination of KiloNeRF with either of these techniques constitutes a promising solution for achieving real-time rendering of high-resolution images in the near future.

**Limitations:** NSVF [24] and KiloNeRF share the assumption of a bounded scene which is a limitation that should be addressed in future work. Representing unbounded scenes requires a higher number of networks, leading to larger memory consumption. Again, efficient data structures could help with addressing this issue and therefore allow for scaling to larger (e.g., outdoor) scenes. On the other hand, the scenes considered in this paper are encoded with less than 100 MB of memory, respectively. This shows that – at least for medium-sized scenes – storage is not an issue.

### 6. Conclusion

In this paper, we demonstrated that real-time rendering of NeRFs can be achieved by spatially decomposing the scene into a regular grid and assigning a small-capacity network to each grid cell. Together with the other benefits inherited from NeRF – excellent render quality and a low storage impact – we believe that this constitutes an important step towards practical NVS. Moreover, the presented acceleration strategy might also apply more broadly to other methods relying on neural function representations including implicit surface models.

**Acknowledgements:** This work was supported by an NVIDIA research gift, the ERC Starting Grant LEGO-3D (850533) and the DFG EXC number 2064/1 - project number 390727645. Songyou Peng is supported by the Max Planck ETH Center for Learning Systems. Yiyi Liao is supported by the BMBF Tübingen AI Center, FKZ: 01IS18039A.
References


[30] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Chakravarty R. Alla Chaitanya, Anton Kaplanyan,


[74] Alex Yu, Ruiling Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. arXiv.org, 2021. 2


