Investigating realtime capabilities of DSNeRF

Maarten Bussler 03696710 Philipp Eckart-Weibäcker 03688221 David Drothler 03695467

Abstract

Depth-supervised NeRF (DSNeRF) [3] is a state of the art deep neural network method for implicit 3D scenes representation from multi-view inputs.
Recent research has show that passing input points through high-frequency functions before feeding the data to the network enables the network to accurately depict high-frequency regions of a scene. We investigate the performance of different such embeddings on the quality of the final output. Specifically, we analyse gaussian fourier feature mappings [11], and approaches that leverage periodic activation functions like SIREN [10] and SINONE [2]. Furthermore, we want to examine the possibility to enhance DSNeRF's real time capabilities by implementing the concepts of FastNeRF [4]. We achieve a GPU accelerated realtime interactive implementation. The overall goal is to produce a DSNeRF based application that is more applicable to real world use by trying to get closer to realtime rendering performance, achieving higher quality reconstruction of difficult scenes and needing less viewpoints.

1. Introduction

Neural networks that learn to implicitly capture complex 3D scenes with the weights of the network are a powerful tool to significantly lower memory requirements of such scene representations and to synthesize novel views of the scene from a sparse set of training images. NeRF [6] is a state-of-the-art deep learning approach for representing static scenes as a 5D function learned by a neural network. The network is trained with a set of input images with known camera poses and outputs the volume density and view dependent emitted radiance at each position $p = (x, y, z)$ for each viewing direction $d = (\theta, \Phi)$. Novel views are synthesized by evaluating the network along rays for each pixel. With the help of volume rendering techniques the generated density and radiance values along the rays are then combined to a final 2D image. While the NeRF architecture produces impressive results, it's usefulness for real life applications is held back by it's dependency on a large amount of input images for training the network, as well as the long training time required to produce high quality renderings. DSNeRF [3] is an extension to the original NeRF network that seeks to address these shortcomings and results in a network architecture that trains 2-3 times faster and produces better rendering results for fewer input views than the original NeRF. The authors of DSNeRF reason that the standard volume rendering used by NeRF to produce the novel views does not respect the general sparseness of real life scenes. As a result the network is prone to fitting to incorrect geometries and does not evaluate the rays at meaningful positions, given too few input images. As a solution they introduce an additional loss that incorporates depth-data into the network. As NeRF already requires input images with known poses, this additional depth data can easily be generated by SFM approaches. The additional depth loss thereby already solves the 3D correspondence matching problem that NeRF would otherwise implicitly have to solve and encourages the ray termination distributions to respect the given surface priors.
We are interested in investigating approaches that can further improve the rendering quality of DSNerf. Recent work by Tancik et al [11] suggests that different approaches of embedding the input of the network to a higher dimensional domain enables the network to better fit to higher frequency components of the training data, which results in sharper and higher quality renderings. Furthermore, periodic activation functions, coupled with careful initialization for stable optimization, have been recently shown to outperform more advanced deep learning architectures on image reconstruction tasks. To this end, we are also interested in analysing possible benefits from combining different input embedding techniques, as well as the SIREN [10] and SINONE [2] architectures with DSNeRF.
Another problem of the NeRF method is it's high computational requirement and as such slow rendering time, which makes it infeasible to use the network for interactive realtime applications. The cause for NeRF's long render times lies in it's volumetric scene representation. In order to render a single image, each ray has to be evaluated multiple times at different positions, which amounts to 100s of network calls for a single pixel. Several approaches [12], [5], [8] solve this problem, but introduce bigger changes on the existing architecture and provide openly available implementations. FastNeRF [4] tackles this is-

sue by introducing a method to efficiently cache and query scene radiances with view directions, which omits the need for constant network calls and produces a rendering algorithm that is three magnitudes faster than the original NeRF, without sacrificing quality. As to our knowledge no open source implementation of FastNeRF exists, we want to implement and analyse the proposed network architecture first hand in regard to it's realtime capabilities when combined with DSNeRF.

2. Embedding

Basri et al. [9] as well as Rahaman et al. [7] show that neural networks are biased to learn lower frequency functions easier than higher frequency functions. This leads the network to first fit to the low frequency components of the target function, and then to the higher frequency parts. If data is missing, the Network as such interpolates with a low-frequency function instead of a more straightforward and smoother curve that would introduce higher frequency components. Consequently, most scene representation networks, such as NeRF, result in renderings that only poorly capture high frequency variation in color and geometry. To counteract this issue, the authors of the original NeRF algorithm [6] propose to leverage high frequency functions to map the original input to a higher dimensional space before passing it to the network. The authors point out that this mapping increases computation time, but also enables the network to better fit to higher frequency components of the data and significantly increases rendering quality (See Figure 1).
In the following paragraph we want to compare different approaches to such embeddings and their influence on the output quality of DSNeRF.

2.1. Theoretical Background of Embeddings

Fourier Features. Tancik et al. [11] present the idea to pass the raw network input through a Fourier feature mapping γ to featurize the input and enhance convergence speed and generalization ability of the neural network. The function γ enables the network to learn high frequency functions in low dimensional space by applying a set of sinusoids to the input points.
We focus on comparing the following Fourier feature mappings:

1. Positional encoding of the original NeRF algorithm to be compared as a baseline: $\gamma(v) = [\sin(2^0\pi v), \cos(2^0\pi v), ..., \sin(2^{L-1}\pi v), \cos(2^{L-1}\pi v)]$, with the hyperparameter L describing the number of featurizing frequencies.

2. Gaussian Mapping by Tancik et al. [11]: $\gamma(v) = [\sin(2\pi Bv), \cos(2\pi Bv)]$, where $B \in \mathbb{R}^{m\times d}$ is sampled from $\mathcal{N}(0, \sigma^2)$ with a Gaussian distribution.

SIREN. Sitzmann and Martel [10] discuss limitations of implicit scene representation networks that build on traditional ReLU-based multilayer perceptrons (MLPs). They show that these architectures fail to represent a signal's higher order derivatives and lack the capacity to capture fine details. The authors instead propose to leverage periodic activation functions and create the new SIREN network architecture. The usage of periodic activation functions enables this network to fit images significantly faster and with more detail than ReLU-based networks without preprocessing of the data. The proposed SIREN architecture utilizes sine activation functions together with MLPs:

$$\Phi_i(v_i) = \sin(\omega_i W_i v_i + b_i) \tag{1}$$

, where Φ_i is the i-th layer, the weights and biases W_i and b_i are initialized with a standard normal distribution and ω_i serves as a scale hyperparameter that increases the spatial frequency of the first layer to better match the frequency spectrum of the signal.

SINONE. SINONE [2] is a neural network for implicit scene representations that can be understood as an extension to the SIREN architecture. The authors show that this alternative network architecture offers a more numerically stable version with similar convergence properties as the ReLU-based MLP with Fourier features for image reconstruction tasks, while omitting preprocessing of the data and thereby saving computational costs. The SINONE architecture consists of a combination of ReLU- and Sigmoid-based MLPs. The first layer copies a sine-based periodic activation function as in SIREN, in order to enable the network to reliably capture the high frequency components of the input and produce outputs with fine detail:

$$\begin{aligned} \Phi_1(v_1) &= \sin(\omega_1 W_1 v_1 + b_1) \\ \Phi_i(v_i) &= \text{ReLU}(W_i v_i + b_i) \\ \Phi_k(v_k) &= \text{Sigmoid}(W_k v_k + b_k) \end{aligned} \tag{2}$$

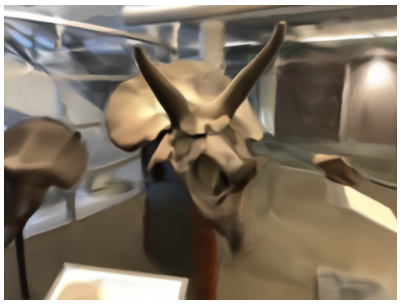
, with $i = 2, ..., k-1$ for k layers.

2.2. Experiments

We compare the output quality of DSNeRF when coupled with different embedding strategies. To this end we analyse the raw visual output as well as the calculated peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) of each experimental run. Table 1 compares the different mappings on a real world (Horns) and 2 synthetic (Lego, Hotdog) data sets. Each experiment was trained for 50000 iterations and the number and size of the layers of the DSNeRF network was not changed between runs. Though with different mappings the embedded size of the input and overall network architecture varies between runs. The optimal choice of the scale hyperparameter for the scaled Gauss

(a) Ground Truth

(b) Positional Encoding



(c) No Embedding

Figure 1. Comparison of rendering quality of DSNeRF for Horns scene when embedding the input with Positional Encoding or using no input embedding. In both cases DSNeRF was trained for 50000 iterations with five training images. As demonstrated by Tancik et al [11], the network produces blurry images without the use of prior input embeddings, whereas the network is able to produce sharp renderings that accurately depict even higher frequency details of the scene when coupled with an embedding technique. .

Mapping differs for varying data sets and was determined with the help of hyperparameter search.

Fourier Features. One can notice that for both the Positional- and Gauss Encoding a larger embedding and as such a larger input feature vector size correspond to a better rendering output, as indicated by a larger PSNR and SSIM. However, this quality increase comes at a higher computational and memory cost when applying the embedding.

The Gauss Mapping achieves the best results for each data set among the tested embedding methods. The quality of the Gauss Mapping can be further increased by introducing an additional scale hyperparameter that is applied to the gaussian matrix B. Compared to Positional Encoding with similar embedding size, the scaled version of the Gauss Mapping increases the achieved PSNR by 0.4 and the SSIM by 0.25 points for the horns data set. Although the Positional Encoding and Gauss Mapping are only marginally different with respect to the PSNR and SSIM metrics, both embedding methods produce distinct rendering outputs (Figure 2). While the output of the Positional Encoding generates sharp images that depict even the finer details of the training data, the Gauss Mapping constructs smoothed renderings, that contain less artifacts and better preserve the shape of objects, as can be seen in Figure 3.

SIREN. The SIREN network performs the worst from all experimental embeddings. The reached PSNR and SSIM are considerable lower than those achieved by the Fourier features embedding or by the SINONE network. Moreover, the rendered 2D image does not resemble the training scene, like the outputs of the SINONE or Fourier features embeddings (Figure 2). The strong loss-oscillations of SIREN during training (Figure 4) suggest that the SIREN network is highly unstable in combination with DSNeRF. Although the SIREN network performs well for the image reconstruction experiments done in the paper of Sitzmann and Martel and possesses good image memorizing capabilities, we thus conclude that the SIREN architecture is poorly suited for the task of inducing novel views of learned scenes.

SINONE. The SINONE architecture performs better than SIREN and achieves comparable results in PSNR and SSIM to the basic Positional Encoding and Gauss Mapping. However, the rendered output of the SINONE network is blurrier than the outputs of the Gauss- and Positional Encoding (Figure 2), which results in overall worse PSNR and SSIM scores. The reason for this drop in quality might lie in the approach of the SINONE network to mimic the high frequency embedding of the Fourier feature mappings with just a singular SIN-layer. The weights of the SIN-layer are supposed to map the input to a higher dimensional feature vector, similar to the different frequencies in the Fourier features embedding. But, compared to the fixed frequency bands in the Fourier features, the SIN-layer is trained at the same time with the rest of the network and is as such harder to control when tasked to produce high quality renderings with DSNeRF's volume rendering. Although the output quality is enhanced compared to the experiment with no embedding techniques at all, the SINONE network cannot achieve a similar embedding quality as the Fourier feature mapping techniques, which leads to a blurrier rendering and lower PSNR and SSIM scores.

3. FastNeRF

FastNeRF [4] fits well together with other changes to the original NeRF network, such as DSNeRF [3], which we utilize as an implementation base. We modify the existing network by separating the direction and position dependent MLPs.

The traditional NeRF network architecture is split into two parts, position- and viewing direction-dependent net-

work as described in Figure 5. Instead of five-dimensional input, a pair of 3D (positional) and 2D (view-dependent) inputs is required. This network architecture simply replaces the baseline DSNeRF network while achieving identical results. In order to obtain the desired speedup at render time, the network needs to be cached. A cache resolution s is specified and the positional input is stored in a three-dimensional cache, whereas the viewdirection input is stored via polar-coordinates in a two-dimensional cache. The latter is generated by sampling the two angles $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$ and converting them to unit-vectors, which are then used as input for the view-dependent network. The network creates four float32 scalars per viewdirection input, one for each of the four cache layers. The output is then saved in a binary file. For the purpose of sampling the positional data, a bounding box in x-, y- and z-direction is created. Along each dimension of the bounding box, s samples are taken after which all possible combinations of those three dimensions are queried in the positional network. Each positional input creates $4*3$ outputs, each cache layer thus has one value for u, v, and w. The output is then saved into three files, each containing four float32 values per input. Additionally the density output is also stored in a single cache file, here one float32 value is mapped to each positional input.

Our interactive NeRF renderer is an extension of a standard volume renderer [1]. First, the opacity, u, v, w and direction sample caches are saved as data buffers from the binary output files of the cache generation routine. The opacity data is transferred to the GPU as CUDA 3D float texture. We choose 4 as a constant for the component size of the deep radiance map, as this is a good trade off of quality and cache size, according to the original FastNeRF paper. This also has the advantage that we are able to utilize CUDA supported 3D float4 textures; the unweighted rgb equivalents uvw are thus loaded in one texture each. The view cache is loaded as 2D float4 texture, similarly to rgba textures.

Rendering is handled via a CUDA kernel of the camera per pixel rays. Each ray steps through the dense volume grid and accumulates color and opacity along the ray in a front-to-back fashion. In order to sample the rgb value of a NeRF cache grid position the uvw 3D texures are accessed and weighted component-wise with the 2D texture view dependent weights. This is achieved by utilizing dot products. To move into the right color space a sigmoid is applied to the rgb values.

We compare performance for two different resolutions. Both fit into our test GPU's RTX2080 8GB VRAM. Generally performance is above 30 fps, with res 256 we manage to reach around 175 fps in the lego scene.

Visual quality is according to the chosen grid resolution, as in [4] higher resolutions are closer to the ground truth. Our implementation also supports optional hardware trilinear interpolation for all texture sampling. Figure 6 and figure 7 show examples of the application.

4. Discussion

We compare the quality of different embedding techniques on the DSNeRF network. On our experiments the Gauss Mapping surpasses the Positional Encoding that was used in the original DSNeRF network. The multiplication of the random sampled Gauss matrix with the input points produces a smooth rendering result that contains less artifacts and is able to contain the general shape of the scene objects better than the Positional Encoding. The SIREN and SINONE network architectures try to mimic the embedding of the Fourier feature mappings by introducing periodic activation functions to the network. Consequently, these techniques safe computational power by not having to map the input to a higher frequency domain prior to feeding the input to the neural network. While this approach works well on 2D image-completion tasks, our experiments show that the inserted sin-activation layers behave unstable during training when coupled with DSNeRF's 3D volume rendering to generate novel image views and do not fit to the training scene's higher frequency components as well as the Fourier Feature mappings.
We implement the FastNeRF architecture and build an interactive NeRF rendering application with GPU acceleration. It could be extended further to also support octree structures to make higher res cache rendering more feasible.

References

[1] Basic volume renderer. https://gitlab.com/shaman42/basic-volume-renderer. Accessed: 2022-07-22. 4

[2] Sinone. https://github.com/alextrevithick/Periodic-Activations. Accessed: 2022-07-20. 1, 2

[3] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised nerf: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12882–12891, 2022. 1, 3

[4] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021. 1, 3, 4

[5] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021. 1

[6] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf:

Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020. 1, 2

[7] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019. 2

[8] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 1

[9] Basri Ronen, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[10] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020. 1, 2

[11] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020. 1, 2, 3

[12] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 1

Table 1. Experiment runs to compare the quality of different embedding techniques on the DSNeRF network. The experiments were run on three different data sets, with 50000 training iterations each. The Horns data set is captured from real world objects, while the Lego and Hotdog data sets are synthetic. The Embedded Size consists of the summation of the size of the position and view feature vector after applying an embedding. Across all data sets, the Scaled Gauss Embedding performs the best in regard to the PSNR and SSIM metrics.

Dataset	**Run**	**PSNR**	**SSIM**	**Embedded Size**
Horns	No Embedding	20,5	0,517	131 + 131
	Positional Encoding	23,6	0,587	63 + 27
	Positional Encoding	23,6	0,620	131 + 131
	Positional Encoding	23,9	0,627	387 + 387
	Gauss	23,5	0,608	67 + 67
	Gauss	23,7	0,614	131 + 131
	Gauss	23,8	0,616	259 + 259
	Gauss Scaled	**24,0**	**0,645**	131 + 131
	SIREN	14,4	0,141	3 + 3
	SINONE	21,9	0,576	3 + 3
Lego	Positional Encoding	24,9	0,861	131 + 131
	Gauss Scaled	**25,6**	**0,881**	131 + 131
Hotdog	Positional Encoding	29,4	0,940	131 + 131
	Gauss Scaled	**30,1**	**0,945**	131 + 131

(a) Positional Encoding

(b) Gauss Mapping

(c) SIREN

(d) SINONE

Figure 2. Comparison of rendering outputs for the real world Horns scene when using different embedding approaches on DSNeRF. In all cases DSNeRF was trained for 50000 iterations with five images. For the Positional Encoding and Gauss Mapping, 131 features are used for the input position and views after applying the mapping. The SIREN and SINONE networks mimic the embedding with their use of a periodic activation function and do not use an input embedding. The basic Positional Encoding produces sharp renderings, while the Gauss Mapping creates a smoothed version that contains less artifacts. The SIREN network is unable to render the details of the test scene with DSNeRF's volume rendering. In contrast, SINONE renders a comparable, but blurrier version of the Positional- and Gauss Encoding output.

(a) Positional Encoding

(b) Gauss Mapping

Figure 3. Comparison of rendering outputs for the synthetic Hotdog scene when using standard Positional Encoding and Gauss Mapping on DSNeRF. In both cases DSNeRF was trained for 50000 iterations with five images and uses 131 features for the input position and views after applying the mapping. Especially when looking at the edge of the plate, one can see that the the Gauss Mapping produces smoothed renderings that contain less artifacts than the Positional Encoding.

Figure 4. Comparing the training of the original DSNeRF with Positional Encoding (green) and DSNeRF with SIREN (gray) on the Horns data set for 50000 iterations, with the number of iterations on the x-axis and the training loss on the y-axis. The Positional Encoding approach follows a smooth training curve that suggests correct fitting to the training data. In contrast the SIREN network behaves unstable during training with a training curve that has large oscillations and larger loss values.

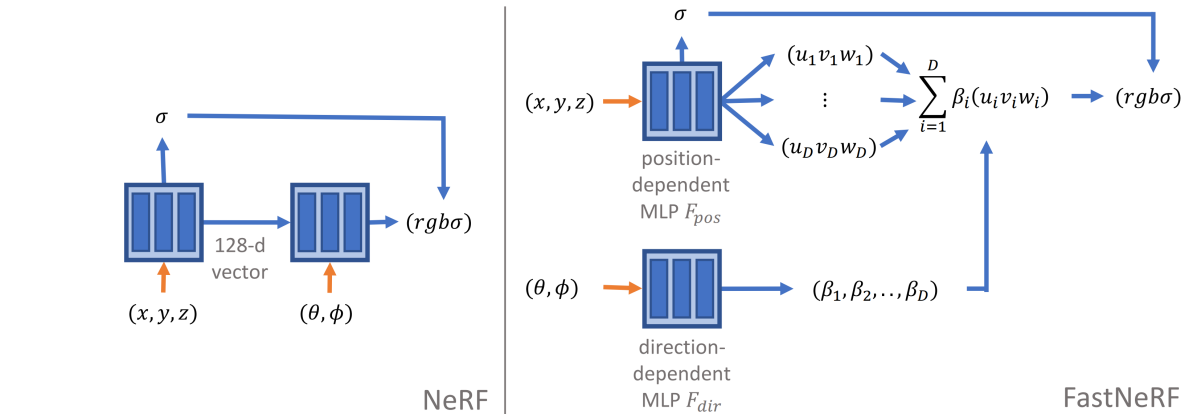


Figure 5. The main difference between the traditional NeRF architecture and the one of FastNeRF lies in the separation of positional and view-dependent network layers.

Figure 6. The Horns scene at res 256, note that the volume is small thus you can't see the stairs and background.

Figure 7. The Lego scene at res 256, note that this model has not been trained very long.