Neural Radiance Field (NeRF) as a Rendering Primitive

StreamNeRF - Adapting a NeRF Model for Progressive Decoding

Matei Galesanu

Supervisor(s): Prof. Dr. Elmar Eisemann, Petr Kellnhofer, Michael Weinmann

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Matei Galesanu
Final project course: CSE3000 Research Project
Thesis committee: Prof. Dr. Elmar Eisemann, Petr Kellnhofer, Michael Weinmann, Dr. Jan van Gemert

Abstract

Neural Radiance Fields (NeRF) and their adaptations are known to be computationally intensive during both the training and the evaluating stages. Despite being the end goal, directly rendering a full-resolution representation of the scene is not necessary and not very practical for scenarios like streamed applications. Our goal is to design a streamable adaptation for a model that can produce fast, rough estimates of 3D scenes, by only using a shallow part of the network. The quality is subsequently improved as more parts of the network are available, such that it can be used in online applications where the model needs to be transferred. Separate models can be trained at different resolutions, but this approach results in a large space overhead and also increases the evaluation time. This can be mitigated by reducing the depth of low-resolution models, but redundancy will still be high as each new model needs to re-evaluate the input data, rendering previous calculations obsolete. Our method combines key concepts from previous approaches to create a progressively trained model that is able to produce intermediate outputs of increasing quality while attempting to optimize the trade-off between overhead and quality. Our model is able to produce a recognizable representation of the scene with as little as one hidden layer from the original model. It also allows for division into streamable chunks which can be sent individually and, upon reconstruction, provide intermediate outputs that bring consistent improvement in quality. The newly streamed data uses the residual output from previous computations in order to reduce redundancy. We show that the final quality of our adaptation is within 2% of the original in terms of previously used quantitative metrics.

1 Introduction

View synthesis is a concept that focuses on estimating the appearance of an object or a scene from different viewing positions and directions, based on a limited number of input data or images. Extensive work has been done in this field in recent years, with the goal of achieving realistic representations efficiently and reliably [1; 2; 3]. NeRF or Neural Radiance Field represents a technique of encoding a 3D scene into the weights of a neural network, whose input is a 5D value describing the camera position and its view direction. The model outputs the estimated, view-dependent, RGB radiance, and volume density at a specified point [4]. The model became the starting point of many adaptations and implementations, meant to optimize and improve the performance of this technique for various uses and scenarios. Instant NGP [5], Zip-NeRF [6], Mip-NeRF [7] and BungeeNeRF [8] are only a few of the already-proposed alternatives to the original model. None, however, focuses on progressive training and decoding of the model.

As shown by available publications, neural radiance fields are arguably complex and can therefore be slow in producing images. Despite the fact that they are optimized to produce high-fidelity, high-resolution and realistic results, some use cases like online applications that need to transfer the model for rendering may benefit from faster, rough estimates that can give an overall impression about the geometry and texture of the final output. Models like MipNeRF [7] are already capable of producing outputs of various resolutions but obtaining a set of sequentially improved images translates into multiple independent evaluations of an entire model, increasing the overall time complexity linearly with the number of desired outputs. Smaller models that can produce lower-resolution outputs have also been previously proposed [9], but not in the context of a sequentially improved representation. Our conclusion was that a dedicated model with applications in streamed environments can further improve the domain of neural radiance fields. The main research question this paper answers is:

How can we divide a single adapted NeRF model into streamable chunks such that online applications can render a fast representation with the first chunk and improve it by progressively decoding additionally streamed data?

This was divided into two sub-questions, stated below:

- How can a model be partitioned in order to be streamed sequentially and reconstructed without compromising on the quality of the final output (deconstructing and rebuilding the model should not affect the results)?
- How can the resolution of an existing scene be improved by streaming additional data from the same model?

Two more questions were formulated to define the intended direction of the research:

- What is the lowest depth at which the output is recognizable?
- What is the quality/similarity of each intermediate render (i.e. what is the information gain for each data chunk that is streamed)?

To answer these questions, we took inspiration from previous work, namely Mip-NeRF, which focuses on countering aliasing effects that appear when rendering at low resolutions [7], and BungeeNeRF, which proposes a progressive training technique for rendering large cityscapes at different scales, with different levels of detail [8]. We want to introduce a method of streaming NeRF models. To this extent, we encode them as a multi-leveled representation, with individually trained groups of layers, such that it can be easily deconstructed and reconstructed without loss of data. Smaller model chunks require less computations to produce an output, and can therefore decrease the time needed to display a rough estimate of the scene. The high modularity also allows additional parts of the model to be subsequently sent and used for improving the quality of the output. Sections 2 and 3 explain related work and past innovations in the area of neural radiance fields. Implementation details are provided in Section 4, while evaluation metrics, results to support our claims, and discussions can be inspected in Section 5. Section 6 describes responsible research concepts applicable to our process, while the last (Section 7) includes conclusions, as well as potential extensions of the project and intended future work.

2 Related Work

Progressive decoding is a technique used in scenarios where streaming large data is necessary and a fast, rough output that acts as a placeholder is more valuable than a slow render at full resolution. It represents the process of dividing the original data into chunks which can then be sent individually and used to reconstruct partial - and ultimately full - representations of the original information. A straightforward example is progressive decoding in images [10], as they are a common resource in streamed applications like websites, but other use cases can benefit from this technique of efficiently transmitting and interpreting large data [11; 12]. Neural radiance fields are also large in terms of storage, and online applications to which the model has to be transferred could, therefore, benefit from an adaptation in this direction to produce faster estimates before the final output.

KiloNeRF achieved real-time rendering by training a set of small MLPs rather than a single large one [9]. Each network is optimized to represent a small portion of the entire scene, and the outputs of each MLP are composed to obtain the final render. Despite the low render times, the model does not compromise on quality because it uses distillation for training, being supervised by a previously trained NeRF model. The real-time characteristic of KiloNeRF makes it a viable alternative to our approach, but the core of our research focused on adapting a slower model for progressive decoding rather than increasing the speed of the full-resolution render.

NeRFPlayer proposed an approach for progressively reconstructing a 4D scene that could be explored using virtual reality, in a streamable environment [13]. Their method focused on encoding the time dimension along with the 3-dimensional space coordinate as a method of adapting a model to account for dynamic scenes. A separate neural field is used for each category to which a 4D point can be assigned - ”static, deforming, or new”. This addresses the issue of evaluating neural fields in online applications, but our research is centered around dividing a model and sending it sequentially. More than this, the goal of NeRFPlayer was to correctly account for temporal changes in the scene, while we focus on static, discrete scenes from previously used datasets.

3 Background

NeRF shed new light on MLP-based realistic rendering by handling scenes with complex geometry and synthesizing outputs that are comparable with those produced by discrete techniques like voxel grids or triangle meshes [4]. The model is trained on images sampled around a 3D scene, for which the camera positions and viewing angles are known. The input is represented by a 5D coordinate, composed of a 3D space coordinate and two viewing angles. It outputs the estimated RGB color and density by integrating multiple sample points along the ray described by the input data. The 3D position is encoded as a sequence of increasingly higher-frequency functions since this type of network has been shown to favor lower frequencies during training [14]. The encoding function $\gamma(p)$ defined as

$$\left(sin(2^0\pi p), cos(2^0\pi p)...sin(2^{L-1}\pi p), cos(2^{L-1}\pi p)\right) \quad (1)$$

is applied to each coordinate within the 3D position $\mathbf{x}$ (L is a hyperparameter tuned by the original NeRF implementation).

The technique optimizes two networks simultaneously: the first one is used for stratified sampling and determines optimal sampling locations for querying the second, finer network. *Mildenhall et al.* interpret the volume density $\sigma(x)$ at location x as ”the differential probability of a ray terminating at an infinitesimal particle” at that location. They also define the expected color of a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ as

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt, \quad (2)$$

where

$$T(t) = exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right), \quad (3)$$

with t_n and t_f the near and far bounds. This innovation in the field of view synthesis has provided a strong basis for subsequent optimizations and adaptations. Our model does not directly improve on this particular approach since it is based on a more optimal implementation, but the concepts of NeRF lie at the core of our research.

MipNeRF extends upon the existing radiance fields approaches by exploring an anti-aliasing solution for low-resolution renders, inspired by the mipmapping principle [7]. It proposes sampling within a cone as opposed to the single ray that NeRF used and as a more efficient alternative to supersampling. Their ”Integrated Positional Encoding (IPE)” represents a method of approximating a conical frustum with the help of a multivariate Gaussian, as an adaptation of NeRF’s positional encoding that was used for points along a single ray. It is also faster and smaller than the original NeRF model, as shown by their results. Our approach takes advantage of this optimization for anti-aliasing for rendering smooth images. It also improves it by adapting the training process to allow intermediate outputs.

BungeeNeRF focuses on multi-scale scenes and optimizing renders at different distances or zoom levels, with a relevant application in representing real-world scenes, using satellite imagery as an example [8]. The main concept behind its efficiency is the progressively trained model, which is supervised at different levels of detail to account for multiple viewing distances. The network is composed of a base group of layers, to which other layers are added throughout the training process, corresponding to more detailed input images. We adapted this idea for progressive representations at the same distance from the scene, but at different levels of detail. The model is trained as if the view zooms in on the model, but output images are always rendered at the initial camera position.

4 Method

We identified the strengths of MipNeRF [7] and BungeeNeRF [8] and designed an adaptation that combines their concepts to achieve an efficient method of rendering intermediate outputs while still providing a structure that allows for seamless partitioning and reconstructing to help in a potential streaming process. The resulting model is comprised of multiple blocks (groups of layers) that can be individually stored and evaluated to produce outputs.

The starting point of our implementation is represented by the MipNeRF code provided by their repository [15]. This was chosen because of its publicly available code, as well as its permissive license. One relevant hyperparameter that was evaluated during experiments is the batch size (b), which determines the number of rays cast or each pixel. With the anti-aliasing logic already in place and training and evaluation pipelines set up, we could direct our focus on adapting the model to support intermediate outputs.

The first step was to integrate the progressive training principle of BungeeNeRF [8] into the existing pipeline. This implied defining an initial group of layers to serve as the starting model, and then adding more blocks as the training progressed, depending on the number of desired outputs or detail levels (hyperparameter L, defined as a tuple that specifies the depth of each block, thus also inferring the total number of detail levels). Whenever a new group of layers is added, it is supervised in combination with all previous groups, but this supervision happens on the same ground truth image, as opposed to the different zoom levels that *Xiangli et al.* proposed [8]. To achieve this joint supervision, the loss of each output is computed and their sum is used to update the weights of the entire network for the following iteration. This ensures that each intermediate output is trained to converge towards the same result, with deeper levels achieving higher quality due to the increased number of parameters that are optimized before producing an output. Because new blocks are added at certain checkpoints in the training process, a trade-off is established between a layer's depth and the number of training iterations it benefits from. The first parts of the model are shallow and are therefore less capable of estimating an accurate representation of the scene, thus requiring more iterations before converging toward a useful output (without the need for further processing like downsampling and filtering). Although receiving less training, deeper blocks have the advantage of a larger background of optimized parameters and are thus able to converge much quicker and produce higher-quality estimations.

Each block corresponds to one detail level and is composed of one or more 256-wide layers. We decided to fix the depth of the model's main structure to 8 layers in order to avoid extended training times and to keep the model size consistent with the original. As results confirmed (Section 5), the quality gain of extending the model is not justifiable, especially considering the added training time. As in the original network, the last layer before the output structure is used to estimate the volume density and is concatenated with the encoded view direction when that feature is enabled. Every group is connected to an output structure of a 128-wide layer and is designed to output an RGB value. Since the resulting blocks are quite shallow, the original input to the network is re-appended to the first layer of each group (skipping the first one in the network), providing additional data for processing and improving the quality of each intermediate output. A visual representation of such a block is visible in Figure 1.

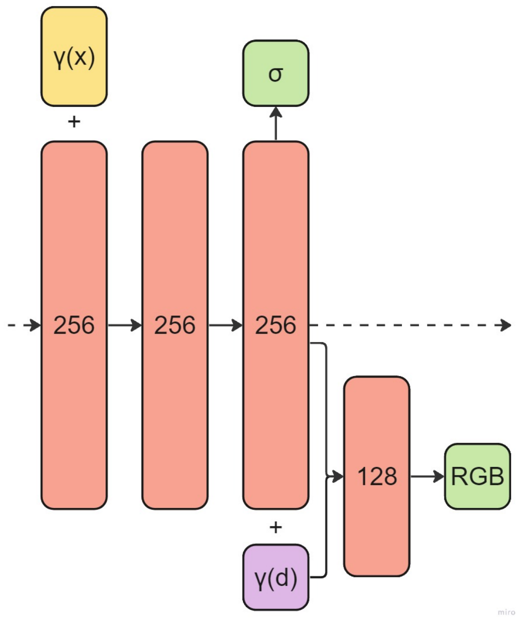

Figure 1: Conceptual representation of a block corresponding to one level of detail. Red shapes represent hidden layers and green shapes represent output layers. Yellow and purple blocks represent concatenation with the positionally encoded location and view direction, respectively. Dashed arrows represent residual input/output from other blocks in the network.

The modular structure allows each level to independently estimate an output and simultaneously provide residual input for the next block in the network. This means that each block can be stored separately and queried in isolation, as long as the appropriate input is provided. When such a model is streamed, the first block can be sent and the evaluation can already begin while the next chunk is being sent. When the evaluation is completed, the intermediate result can be stored, and the output can be displayed visually. As soon as the next group of layers is received, evaluation can restart and the new output can replace its predecessor as soon as this process is finished. This can continue until all blocks have been sent, resulting in a progressive representation of the scene which is ultimately displayed at full resolution. As expected, the performance and impact of this approach are directly dependent on the levels of detail, with more levels allowing for quicker and more efficient parallelization of the network evaluation.

5 Evaluation and Results

The model has been trained and evaluated on 4 scenes from the Blender dataset used by the original NeRF implementation [4], as well as MipNeRF [7]. All experiments were run on an Nvidia RTX 2070 and an Nvidia RTX 3060 OC, as using two different machines highly increased the time efficiency of the training process. However, the same GPU has been used for comparisons where computation time was relevant.

5.1 Experimenting with modularity

The first experiment had the goal of determining the extent to which high modularity affects the relevance of output data. For this scenario, we trained models in $L = (8 \times 1)^1$ configurations, meaning that each layer in the network was connected to an output structure. This allowed us to determine which layers are the most suitable for producing intermediate outputs with respect to computation time and model size. To quantitatively measure the quality of each intermediate output, we used metrics like PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index Measure) [16], which are popular in many other NeRF papers. For this particular experiment, inspecting the plots for each metric provided enough information for optimizing the model for the subsequent evaluation phase.

Figure 2 clearly shows an upward trend. However, a closer inspection shows that deeper levels do not provide constant quality improvement. Certain levels seem to have a similar output to the next one in the network, so an appropriate optimization would be to reduce the total number of levels. The first and last levels were kept to ensure the fastest render possible and the full-resolution output, respectively. Out of the remaining 6 levels, only half were kept, since similarly-performing layers were observed to appear in pairs. Experiments showed that output structures were optimally attached to layers 0, 2, 3, 6, and 7 in the network, leading to a $L = (1, 2, 1, 3, 1)$ configuration. Optimizing the structure on a per-scene basis could lead to better results, but a general architecture was selected as a proof of concept.

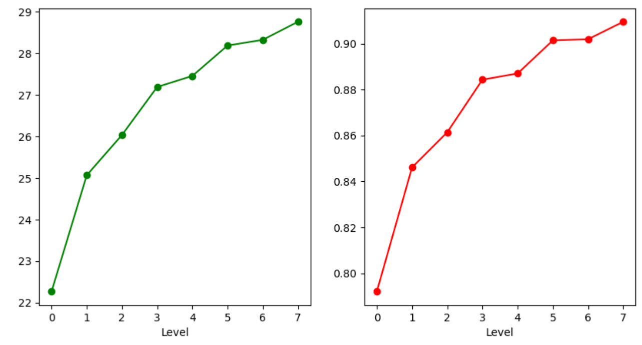

Figure 2: PSNR (left) and SSIM (right) for each level in the $L = (8 \times 1)$ model.

An unexpected side result of the first experiment was the quality of the first layer's output, exemplified in Figure 3. The scene is recognizable even after a single layer, which is the fastest representation that could be achieved using this approach.

5.2 Optimizing the model

Having determined ideal locations for intermediate outputs, the structure of the model was adapted and trained on all four selected scenes. The results matched expectations, as each output brought a clear improvement over the previous. Reducing the number of output structures also reduced

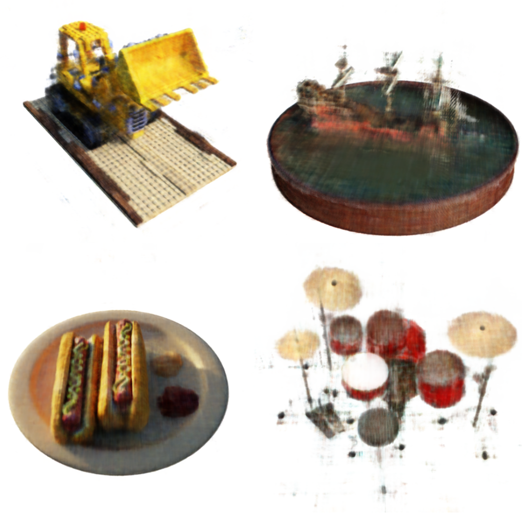

Figure 3: Output of Level 0 for the four scenes in the dataset. The objects are already recognizable in this state.

the total number of parameters and therefore the time needed to train the network. Figure 4 shows that performance is not equal for each scene, but the upward trend is clearly visible. To emphasize this, Figure 5 displays the same results, but normalized. This proves that the sequential improvement is consistent throughout the different scenes, and the difference in performance is only given by the complexity of the scene (the *ship* scene is arguably more complex and therefore has lower similarity scores).

Another design choice that was validated by experiments was the number of main layers in the network. Although the goal was to match the depth of the original model, increasing it could still yield improved results. The 8-layer model was, therefore, compared to one with two extra layers and one additional output structure. To measure the efficiency of the extended model, the quantitative improvement (PSNR and SSIM) was compared to the increase in runtime and space needed to complete the training. Table 1 shows the results of this comparison. We concluded that the increase in size and especially computation time constitutes a reason to not expand the model.

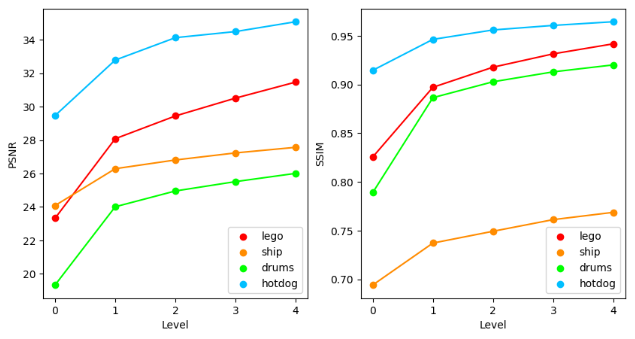

Figure 4: PSNR (left) and SSIM (right) measurements for the optimized model. The upward trend is more consistent than previously.

[1]Short notation for (1, 1, 1, 1, 1, 1, 1, 1)

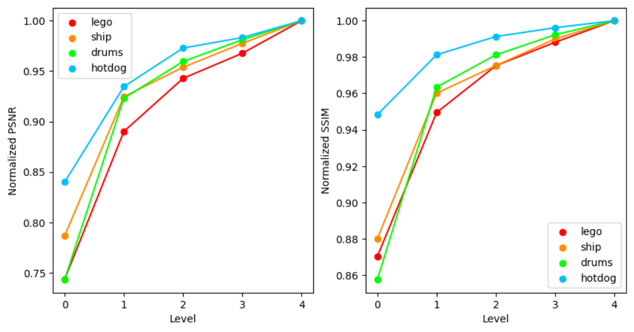

Figure 5: Normalized PSNR and SSIM measurements. Consistency in improvement throughout the scenes is clearly visible.

	8 layers	10 layers	Increase
Parameters	1097620	1356568	23.59%
Storage (KB)	12869	15904	23.58%
Training time (m)	560	791	41.25%
Best PSNR	31.4859	31.4866	0.002%
Best SSIM	0.9414	0.9418	0.04%

Table 1: Comparison between standard and extended model. The trade-off between time and space overhead and information gain does not justify a model extension.

5.3 Final evaluation

Now that the model was in its optimal configuration (Figure 6), it was trained on all 4 scenes within the dataset. The performance of this final model has been measured as PSNR and SSIM, and also compared with the original (MipNeRF), in order to highlight the advantages and drawbacks of our proposed adaptation. Qualitatively, the performance of our model can be analyzed in Figure 7, while a side-by-side comparison with the baseline is present in Table 2. This clearly shows the increase in model size as well as training time, but it also proves that our adaptation is capable of a faster initial output without sacrificing on the final quality. The complete results can be inspected in the Appendix.

	MipNeRF	Ours	Ratio
Parameters	612740	1097620	79.13%
Storage (KB)	7183	12869	79.16%
Training time (m)	375	560	49.33%
Best PSNR	32.0160	31.4859	-1.66%
Best SSIM	0.9449	0.9414	-0.37%
First output after	612740	127620	-79.17%

Table 2: Side-by-side comparison of performance between the original model and our adaptation. *First output after* refers to the minimum required parameters for estimating an output.

5.4 Discussion

Adapting the NeRF model by inserting more output structure was expected to increase the model size as well as the

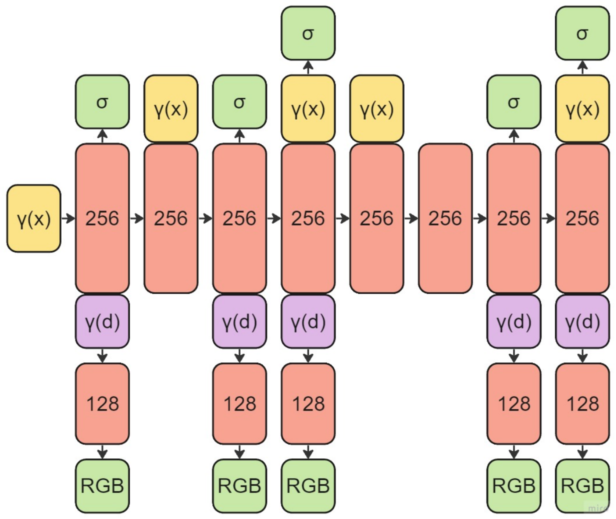

Figure 6: Optimized model architecture corresponding to $L = (1, 2, 1, 3, 1)$. This diagram has been simplified to avoid unnecessary complexity. For an in-depth explanation of the inner logic of each block, see Section 4 and Figure 1.

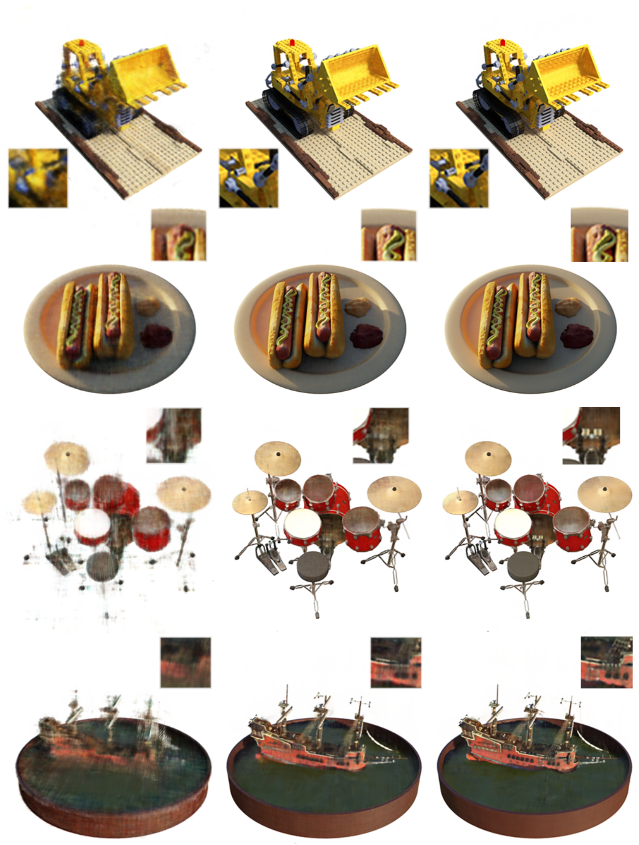

Figure 7: Example of progressively improved output quality (left to right). Only outputs of levels 0, 2, and 4 are displayed for clearer differences.

training time. This was confirmed by the results in the previous section and constitutes the main disadvantage of our proposed method. However, looking from a different perspective, our model is capable of producing five individual outputs with less than double the storage and training time. More than this, the quality of the final output is within 2% of the original model, so no compromise is made in that direction. Finally, our approach is 4.8 times faster in rendering a preview due to having an output structure after the first layer. The goal of our adaptation was to provide the option of progressive decoding, without compromising on the overall quality and results show that this was achieved.

6 Responsible Research

This study adheres to responsible research guidelines by allowing the described experiments to be fully replicated. All used code is publicly available on this project's GitHub repository [17] which is an official fork of the MipNeRF repository, to ensure compliance with their license. The repository contains the original instructions as well as additional information for running the adapted version.

All images, plots, and tables contain actual data obtained through real experiments which can be reproduced for similar results.

7 Conclusions and Future Work

Our goal was to design an adaptation for an existing NeRF model, such that it can be separated in individual blocks which can then be used for streamed applications or similar use cases.

The structure of a NeRF model can be modified such that it supports intermediate outputs, using a progressive training technique. Partitioning the new model is, therefore, trivial, as the entire network can be fully represented through its parameters. Instead of storing them in the same file, the data can be split between multiple resources and reconstructed without losing information.

With the new adaptation, a part of the model can be reliably evaluated even in the absence of the final layers. What this means is that a small component can be queried faster for a rough output, while storing the output of its last hidden layer. As soon as the next component is available, it has access to that residual data and can be directly evaluated to obtain an improved representation. Despite some extra computations for the additional output structures, the total evaluation time of the network does not increase by much.

As our experiments proved, even a single layer is capable of producing a recognizable representation of the scene, due to the modified training procedure. Concretely, evaluating a single layer as opposed to the full model is expected to take less time, resulting in a substantial speed-up of the first render.

Even though the output of the first layer is remarkably good, each new block further increases the render quality, as supported by the measurements in the results section. The information gain does decrease with deeper layers, but it is prevented from reaching very low values by the structure of the model, which limits the options for the number of detail levels. The final quality is, also, very close to the original, showing that very little information is lost with this adaptation.

The process of adapting a NeRF model for streamed applications has shown that the updated structure is highly reliant on the original architecture and is therefore not very specific to a certain model. The main possible extension of this study would be adapting other NeRF implementations that can benefit from a streamable representation, by modifying their inner structure in a very similar way as we did for MipNeRF. Another point of improvement would be experimenting with different sizes for the hidden layers since our adaptation was already capable of outputting a qualitative image after the first one, indicating that further optimizations are possible.

References

[1] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, "View synthesis by appearance flow," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 286–301, Springer International Publishing, 2016.

[2] S. Avidan and A. Shashua, "Novel view synthesis in tensor space," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1034–1040, 1997.

[3] N. K. Kalantari, T.-C. Wang, and R. Ramamoorthi, "Learning-based view synthesis for light field cameras," *ACM Trans. Graph.*, vol. 35, dec 2016.

[4] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.

[5] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, pp. 102:1–102:15, July 2022.

[6] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Zip-nerf: Anti-aliased grid-based neural radiance fields," *arXiv*, 2023.

[7] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," *ICCV*, 2021.

[8] Y. Xiangli, L. Xu, X. Pan, N. Zhao, A. Rao, C. Theobalt, B. Dai, and D. Lin, "Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering," in *The European Conference on Computer Vision (ECCV)*, 2022.

[9] C. Reiser, S. Peng, Y. Liao, and A. Geiger, "Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps," in *International Conference on Computer Vision (ICCV)*, 2021.

[10] L. Pu, M. W. Marcellin, B. Vasic, and A. Bilgin, "Unequal error protection and progressive decoding for jpeg2000," *Signal Processing: Image Communication*,

vol. 22, no. 3, pp. 340–346, 2007. Special issue on Mobile Video.

[11] Y. S. Han, S. Omiwade, and R. Zheng, “Survivable distributed storage with progressive decoding,” in *2010 Proceedings IEEE INFOCOM*, pp. 1–5, 2010.

[12] D. Hoang, H. Bhatia, P. Lindstrom, and V. Pascucci, “High-quality and low-memory-footprint progressive decoding of large-scale particle data,” in *2021 IEEE 11th Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 32–42, 2021.

[13] L. Song, A. Chen, Z. Li, Z. Chen, L. Chen, J. Yuan, Y. Xu, and A. Geiger, “Nerfplayer: A streamable dynamic scene representation with decomposed neural radiance fields,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 5, pp. 2732–2742, 2023.

[14] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, “On the spectral bias of neural networks,” in *International Conference on Machine Learning*, pp. 5301–5310, PMLR, 2019.

[15] J. Barron and M. Tancik, “mip-nerf.” https://github.com/google/mipnerf, 2021.

[16] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[17] M. Galesanu, “Streamnerf.” https://github.com/janelu44/streamnerf, 2023.

Appendix

Detail level	Lego	Ship	Drums	Hotdog
0	23.4100	24.0840	19.3302	29.4661
1	28.0268	26.2884	24.0106	32.7874
2	29.6846	25.8165	24.9565	34.1285
3	30.4682	26.4577	25.5162	34.4935
4	31.4441	27.0672	26.0100	35.0820
Avg gain (%)	7.87	3.04	8.08	4.53

Table 3: PSNR values for each level on all 4 scenes

Detail level	Lego	Ship	Drums	Hotdog
0	0.8192	0.6941	0.7889	0.9144
1	0.8937	0.7372	0.8864	0.9462
2	0.9180	0.8136	0.9027	0.9559
3	0.9302	0.8259	0.9129	0.9606
4	0.9412	0.8343	0.9200	0.9643
Avg gain (%)	3.58	4.77	4.03	1.34

Table 4: SSIM values for each level on all 4 scenes

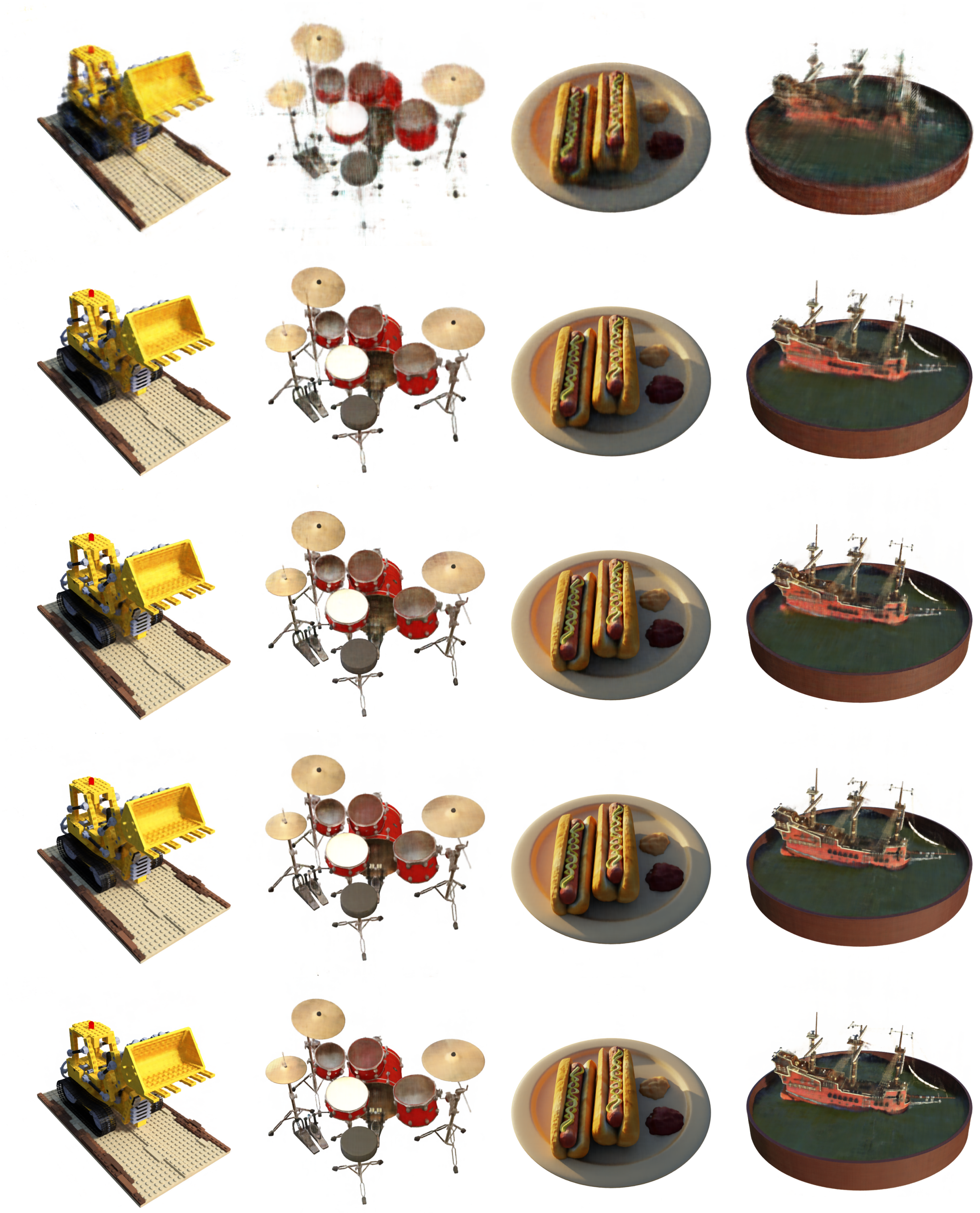

Figure 8: Progressive improvement of all four scenes. Detail levels increase from top to bottom.