City-on-Web: Real-time Neural Rendering of Large-scale Scenes on the Web

Kaiwen Song[1], Xiaoyi Zeng[1], Chenqu Ren[2], and Juyong Zhang[1]*

[1] University of Science and Technology of China
[2] East China Normal University

1 Derivation of Block-based Volume Rendering

Traditional volume rendering methods, like those used in Block-NeRF [9] and Mega-NeRF [11], necessitate accessing radiance and opacity for all sampling points, even if the points belong to different blocks. This requirement means that when the rendering resources of multiple shaders cannot communicate with each other, their volume rendering algorithm cannot work. On the contrary, our block-based volume rendering algorithm is tailored for such resource-independent settings, allowing for the independent rendering of each block before blending their outputs. In this section, we will prove the equivalence between this resource-independent volume rendering and the traditional volume rendering under Lambertian conditions.

In traditional volume rendering, consider a ray $\boldsymbol{r}$ with M sampling points. For each sampling point p_i, the diffuse color, feature, and opacity are represented as $\boldsymbol{c}_i$, $\boldsymbol{f}_i$, and α_i respectively. The final diffuse color $\boldsymbol{C}_d$ of the ray $\boldsymbol{r}$ can be obtained by

$$\hat{\boldsymbol{C}}_d(\boldsymbol{r}) = \sum_{i=1}^{M} \prod_{j=1}^{i-1} (1-\alpha_j)\alpha_i \boldsymbol{c}_i \tag{1}$$

In block-based volume rendering, let's assume the ray, with M sampling points, traverses through K blocks. Within block $\mathcal{B}_k$, there are N_k sampling points. For sampling point p_i^k in block $\mathcal{B}_k$, the diffuse color, feature, and opacity output by the shader or submodel of block $\mathcal{B}_k$ are represented as $\boldsymbol{c}_i^k$, $\boldsymbol{f}_i^k$, and α_i^k, respectively.

$$\begin{aligned} \boldsymbol{c}_i^k &= \boldsymbol{c}_{N_1+\cdots+N_{k-1}+i} \\ \boldsymbol{f}_i^k &= \boldsymbol{f}_{N_1+\cdots+N_{k-1}+i} \\ \alpha_i^k &= \alpha_{N_1+\cdots+N_{k-1}+i} \end{aligned} \tag{2}$$

* Corresponding Author

Then, for this ray, diffuse color $\boldsymbol{C}_d^k$, specular feature $\boldsymbol{F}^k$ and opacity α^k of block $\mathcal{B}_k$ are caculated as follows:

$$\begin{aligned}\boldsymbol{C}_d^k &= \sum_{i=1}^{N_k}\prod_{j=1}^{i-1}(1-\alpha_j^k)\cdot\alpha_i^k\boldsymbol{c}_i^k \\ \boldsymbol{F}^k &= \sum_{i=1}^{N_k}\prod_{j=1}^{i-1}(1-\alpha_j^k)\cdot\alpha_i^k\boldsymbol{f}_i^k \\ \alpha^k &= \sum_{i=1}^{N_k}\prod_{j=1}^{i-1}(1-\alpha_j^k)\cdot\alpha_i^k\end{aligned} \tag{3}$$

By blending the rendering results of each block using Eq. (8), the final diffuse color $\boldsymbol{C}_d$ and specular feature $\boldsymbol{F}$ of the ray $\boldsymbol{r}$ can be obtained.

$$\begin{aligned}\boldsymbol{C}_d(\boldsymbol{r}) &= \sum_{k=1}^{K}\prod_{j=1}^{k-1}(1-\alpha^j)\cdot\boldsymbol{C}_d^k \\ \boldsymbol{F}(\boldsymbol{r}) &= \sum_{k=1}^{K}\prod_{j=1}^{k-1}(1-\alpha^j)\cdot\boldsymbol{F}^k\end{aligned} \tag{4}$$

From Eq. (3), we get the following equation:

$$\begin{aligned}1-\alpha^k &= 1-\sum_{i=1}^{N_k}\prod_{j=1}^{i-1}(1-\alpha_j^k)\cdot\alpha_i^k \\ &= 1-\alpha_1^k-(1-\alpha_1^k)\alpha_2^k-(1-\alpha_1^k)(1-\alpha_2^k)\alpha_3^k-\cdots \\ &= (1-\alpha_1^k)(1-\alpha_2^k-(1-\alpha_2^k)\alpha_3^k-\cdots) \\ &= (1-\alpha_1^k)(1-\alpha_2^k)(1-\alpha_3^k-\cdots) \\ &= \cdots \\ &= \prod_{i=1}^{N_k}(1-\alpha_i^k)\end{aligned} \tag{5}$$

Consequently, we can obtain the following by substituting Eq. (5) into Eq. (4).

$$\begin{aligned}\boldsymbol{C}_d(\boldsymbol{r}) &= \sum_{k=1}^{K}\prod_{j=1}^{k-1}(1-\alpha^j)\cdot\boldsymbol{C}_d^k \\ &= \sum_{k=1}^{K}\prod_{j=1}^{k-1}\prod_{i=1}^{N_j}(1-\alpha_i^j)\cdot\boldsymbol{C}_d^k \\ &= \sum_{k=1}^{K}\prod_{i=1}^{N_1+\cdots+N_{k-1}}(1-\alpha_i)\cdot\boldsymbol{C}_d^k\end{aligned} \tag{6}$$

By substituting $\boldsymbol{C}_d^k$ from Eq. (3) into Eq. (6), we demonstrate the equivalence between Eq. (1) and Eq. (4).

$$
\begin{aligned}
\boldsymbol{C}_d(\boldsymbol{r}) &= \sum_{k=1}^{K} \prod_{l=1}^{N_1+\cdots+N_{k-1}} (1-\alpha_l) \sum_{i=1}^{N_k} \prod_{j=1}^{i-1} (1-\alpha_j^k) \cdot \alpha_i^k \boldsymbol{c}_i^k \\
&= \underbrace{\alpha_1^1 \boldsymbol{c}_1^1 + \cdots + \prod_{i=1}^{N_1-1} (1-\alpha_i^1) \alpha_{N_1}^1 \boldsymbol{c}_{N_1}^1}_{block1} \\
&+ \underbrace{\prod_{j=1}^{N_1} (1-\alpha_j^1) \left(\alpha_1^2 \boldsymbol{c}_1^2 + \cdots + \prod_{i=1}^{N_2-1} (1-\alpha_i^2) \alpha_{N_2}^2 \boldsymbol{c}_{N_2}^2 \right)}_{block2} + \cdots \\
&+ \underbrace{\prod_{j=1}^{N_1} (1-\alpha_j^1) \cdots \prod_{j=1}^{N_{K-1}} (1-\alpha_j^{K-1}) \left(\alpha_1^K \boldsymbol{c}_1^K + \cdots + \prod_{i=1}^{N_K-1} (1-\alpha_i^K) \alpha_{N_K}^K \boldsymbol{c}_{N_K}^K \right)}_{blockK} \\
&= \underbrace{\alpha_1^1 \boldsymbol{c}_1^1 + \cdots + \prod_{i=1}^{N_1-1} (1-\alpha_i^1) \alpha_{N_1}^1 \boldsymbol{c}_{N_1}^1}_{block1} \\
&+ \underbrace{\prod_{j=1}^{N_1} (1-\alpha_j^1) \alpha_1^2 \boldsymbol{c}_1^2 + \cdots + \prod_{j=1}^{N_1} (1-\alpha_j^1) \prod_{i=1}^{N_2-1} (1-\alpha_i^2) \alpha_{N_2}^2 \boldsymbol{c}_{N_2}^2}_{block2} + \cdots \\
&+ \underbrace{\prod_{i=1}^{N_1} (1-\alpha_i^1) \cdots \prod_{i=1}^{N_{K-1}} (1-\alpha_i^{K-1}) \alpha_1^K \boldsymbol{c}_1^K + \cdots}_{blockK} \\
&+ \underbrace{\prod_{i=1}^{N_1} (1-\alpha_i^1) \cdots \prod_{i=1}^{N_{K-1}} (1-\alpha_i^{K-1}) \prod_{i=1}^{N_K-1} (1-\alpha_i^K) \alpha_{N_K}^K \boldsymbol{c}_{N_K}^K)}_{blockK} \\
&= \underbrace{\alpha_1 \boldsymbol{c}_1 + \cdots + \prod_{i=1}^{N_1-1} (1-\alpha_i) \alpha_{N_1} \boldsymbol{c}_{N_1}}_{block1} \\
&+ \underbrace{\prod_{j=1}^{N_1} (1-\alpha_j) \alpha_{N_1+1} \boldsymbol{c}_{N_1+1} + \cdots + \prod_{i=1}^{N_1+N_2-1} (1-\alpha_i) \alpha_{N_1+N_2} \boldsymbol{c}_{N_1+N_2}}_{block2} + \cdots
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
&+ \underbrace{\prod_{j=1}^{N_1+\cdots+N_{K-1}} (1-\alpha_j)\alpha_{N_1+\cdots+N_{K-1}+1}\boldsymbol{c}_{N_1+\cdots+N_{K-1}+1} + \cdots}_{blockK} \\
&+ \underbrace{\prod_{i=1}^{N_1+\cdots+N_K-1} (1-\alpha_i)\alpha_{N_1+\cdots+N_K}\boldsymbol{c}_{N_1+\cdots+N_K}}_{blockK} \\
&= \sum_{i=1}^{N_1+\cdots+N_K} \prod_{j=1}^{i-1}(1-\alpha_j)\alpha_i\boldsymbol{c}_i \\
&= \sum_{i=1}^{M}\prod_{j=1}^{i-1}(1-\alpha_j)\alpha_i\boldsymbol{c}_i = \hat{\boldsymbol{C}}_d(\boldsymbol{r})
\end{aligned}
\tag{8}
$$

It can be observed that block-based volume rendering approach in the resource-independent environment, such as in our case where multiple shaders are used to render the entire scene, is equivalent to the MERF's volume rendering method [7] without deferred rendering.

2 More Details on Experiments

In this section, we provide more details on our custom *Campus* dataset, implemention details and settings of comparison methods.

2.1 Dataset

The *Campus* dataset is captured at an altitude of about 180 meters, covering an area of approximately 960,000 m^2. We adopt a circular data capture method for areial photography, as shown in Fig. 1. We find that this method often results in a higher overlap rate, allowing for a more accurate estimation of camera poses. Our dataset was captured over 8 hours on a cloudy day, with a fixed exposure setting to ensure almost identical appearance of photos taken at different time. We used Colmap [8] to estimate camera poses. Feature matching was done using a vocabulary tree, followed by a hierarchical mapper followed by a few iterations of triangulation and bundle adjustment to estimate camera poses.

2.2 Implementation Details

For blocks at the boundaries of the entire scene, an unbounded scene representation is required to represent areas outside the block boundaries. We follow the same approach as MERF [7] to compute ray-AABB intersections trivially. To be specific, we employ the scene contraction function to project the scene external

to the unit sphere into a cube, which has a radius of 2. The definition of the $j-th$ coordinate for a contracted point is as follows:

$$\text{contract}(\mathbf{x})_j = \begin{cases} x_j & \text{if } \|\mathbf{x}\|_\infty \leq 1 \\ \frac{x_j}{\|\mathbf{x}\|_\infty} & \text{if } x_j \neq \|\mathbf{x}\|_\infty > 1 \\ \left(2 - \frac{1}{|x_j|}\right) \frac{x_j}{|x_j|} & \text{if } x_j = \|\mathbf{x}\|_\infty > 1 \end{cases}, \tag{9}$$

Our method takes posed multi-view images captured using a fly-through camera as input. The training code is built on the nerfstudio framework [10] with tiny-cuda-nn extension. Our real-time viewer is a JavaScript web application whose rendering is implemented through GLSL. We set the 512^3 resolution for the voxel and 2048^2 resolution for the triplane within each block. We use a 4-layer MLP with 64 hidden dimensions as an encoder after multi-resolution hash encoding to output density, color, and specular feature. Moreover, a 3-layer MLP with 16 hidden dimensions tiny deferred MLP is developed to predict residual view-dependent color. We sample 16384 rays per batch and use Adam optimizer with an initial learning rate of 1×10^{-2} decaying exponentially to 1×10^{-3}. The global deferred MLP is a 6-layer MLP with 128 hidden dimensions. Our model is trained with $50k$ iterations on one NVIDIA A100 GPU. We split the scene into 24 blocks for *Campus* scene, and split other scenes into four blocks to reconstruct these scenes. We perform qualitative comparisons between our method and existing SOTA methods for large-scale reconstruction. The *Campus* dataset is partitioned into six parts. NeRFacto, Instant-NGP, and Grid-NeRF were applied to reconstruct one of these parts. NeRFacto and Instant-NGP are utilized with the highest hash encoding resolution of 8192^3. Mega-NeRF divides the *Campus* scene into 24 blocks and splits another dataset into four blocks to evaluate its performance.

We use an A100 GPU for training. In Sections 5.1 and 5.2, we perform training for 50,000 iterations with a batch size of 16384 pixels for the *Campus* dataset, taking approximately 48 hours, while the training for other datasets takes around 24 hours. The experiments described in Sections 5.2 and 5.4 are subjected to training for 30,000 iterations with the same batch size, which is completed in approximately 12 hours. Training losses are initially balanced with $\lambda_1 = 1.0$, $\lambda_2 = 1.0$, $\lambda_3 = 0.01$, $\lambda_4 = 0.05$, $\lambda_5 = 0.001$ and sample 2^{14} samples for computing sparsity loss. During LOD genreation phase, we utilize the same loss function and hyperparameters as those employed during the training stage. We freeze the training of submodels for 5,000 iterations and then refine them and shared global deferred mlp jointly for an additional 10,000 iterations.

2.3 Comparative Method Settings

In Sec. 5.1, we adopt the official implementations of Mega-NeRF [11], NeRFacto [10], and Grid-NeRF [12]. Additionally, we use an unofficial implementation of Instant-NGP[3]. Specifically, NeRFacto is trained with a batch size of

[3] https://github.com/ashawkey/torch-ngp

(a) Camera Poses (b) Examples from *Campus* Dataset

Fig. 1: Visulization of *Campus* Dataset.

65,536 for 30,000 iterations. Instant-NGP [6] is trained for 500,000 iterations with a batch size of 4096. Grid-NeRF [12] is trained for 50,000 iterations with a batch size of 16384. Mega-NeRF [11] is trained for 500,000 iterations for each block, using a batch size of 1024. In Sec. 5.2, For MobileNeRF [1], We initialize a 192^3 grid to generate polygonal meshes while adhering to default parameters for other setups. We use the open-source version[4] for BakedSDF [13], conducting training in two phases: 20,000 and 50,000 iterations, respectively with a batch size of 16,384 and use 1024^3 to extract meshes by marching cubes [5]. We employ the official implementation of MERF [7] and 3D Gaussian Splatting [3]. During the MERF [7] experiments, the data is divided into four parts for a fair comparison with our method, and each block is trained with 32,768 batch size for 30,000 iterations, using default parameter settings. For 3D Gaussian splatting [3], we demonstrate the results after training for 30,000 and 100,000 iterations using the default settings.

Table 1: Comparison on *Residence*, *Building* and *Sci-art* scenes.

	Residence			*Sci-art*			*Building*		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRFacto	21.92	0.642	0.475	24.65	0.781	0.312	17.70	0.502	0.442
Mega-NeRF	22.08	0.628	0.489	25.60	0.770	0.390	**20.93**	0.547	0.504
Ours	**22.54**	**0.680**	**0.386**	**25.71**	**0.802**	**0.241**	20.13	**0.578**	**0.397**

3 More Results

We present more qualitative comparisons among our method, MERF [7] (4 blocks), and 3D Gaussian Splatting [3] (100k iters) on the *Campus* Dataset

[4] https://github.com/hugoycj/torch-bakedsdf

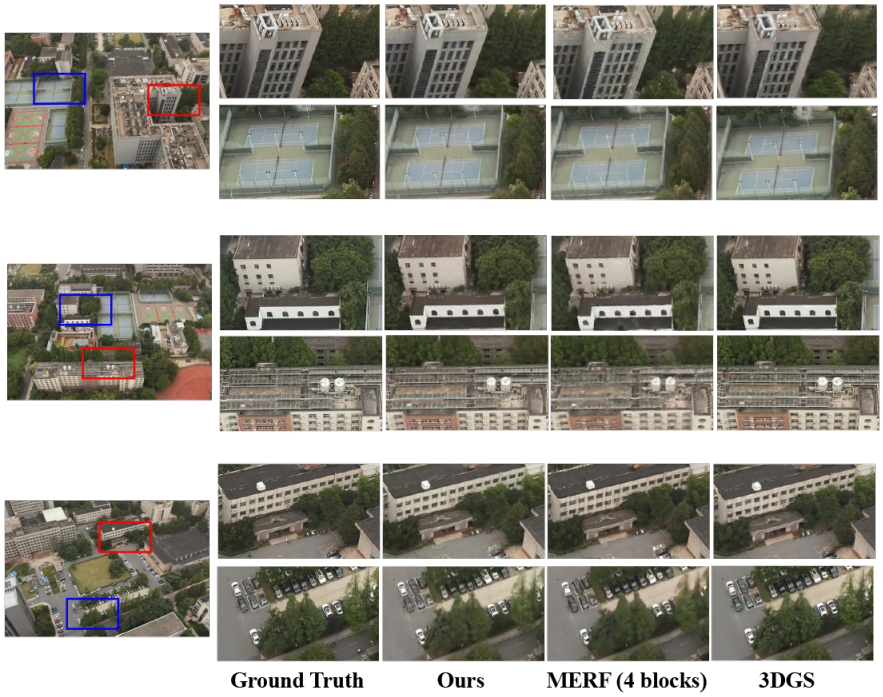

Fig. 2: Qualitative comparison of real-time rendering methods.

in Fig. 2. Additionally, we showcase rendering results from different LODs to demonstrate the appearance consistency across various LODs in Fig. 3. We conduct more experiments on the *Residence*, *Building*, and *Sci-art* scenes datasets. The hyperparameters used in these additional experiments are the same as described in the main paper. The results of these extended experiments are presented in Tab. 1. Our method outperforms Mega-NeRF and NeRFacto in all the scenes.

Ablations on shared heads. In contrast to Grid-NeRF [12], we don't share heads. This strategy reduces the model's representational capacity, consequently decreasing reconstruction quality as shown in the first two rows of Tab. 2. Additionally, we tested the speed of generating LOD under the shared heads condition in Tab. 2. It can be observed that sharing heads does not exert an influence on the quality of lod generation.

4 Real-Time Viewer

Our real-time viewer platform is based on the MERF volume renderer [7], leveraging an OpenGL fragment shader to execute ray marching and deferred rendering. It accesses feature and density information through texture look-ups.

Table 2: Ablation study on sharing heads.

	PSNR↑	SSIM↑	LPIPS↓	Training Time ↓
training with shared heads (4 blocks)	24.36	0.734	0.201	-
training with non-shared heads (4 blocks)	24.82	0.741	0.190	-
generate lod with shared heads (2 hours)	24.19	0.718	0.213	2 hours
generate lod with shared heads (4 hours)	24.22	0.729	0.204	4 hours
generate lod with non-shared heads	24.20	0.724	0.204	4 hours

However, we have implemented several improvements tailored for large scenes to enhance performance.

Dynamic Loading. We employ dynamic loading strategy to determine the level-of-detail and blocks to be loaded, thereby reducing VRAM usage. We use the center points obtained in Section 4.1 for each block on the xy plane as the xy-components, and assign the height of the highest part within the block as the z-component. This defines the 3D center point for each block during the rendering stage. Additionally, we use the rectangle formed by the block's four xy corner points and the previously determined z value to define the block's region. By projecting this region onto the camera plane, we can determine whether the block is visible to the camera.

During the rendering phase, we first eliminate blocks that are not visible for cameras. Starting with the coarsest LOD, we check if the distance from the camera to all visible blocks within the finer LOD exceeds a certain threshold. For blocks beyond this threshold, rendering is performed using the resources loaded at the current LOD. Otherwise, the evaluation progresses to the subsequent, finer LOD. This stepwise process continues until we reach the finest LOD or complete the rendering of the entire image.

Depth sorting. We compute the distance from each block's center point to the camera using only their xy-components, leveraging this calculated distance for depth sorting. We adopt this approach as our scene segmentation occurs solely in the xy plane, without division along the z-axis.

5 Discussion

UE4-NeRF [2] and NeuRas [4], both mesh-based rendering approaches, offer fast rendering speed but face challenges when representing large, detail-rich scenes like those including dense foliage. These mesh representations can consume extensive memory, often amounting to several gigabytes. Additionally, 3D Gaussian Splatting [3], typically requires millions to tens of millions of points. The inherent properties of Gaussian splatting, which involve numerous Gauss-related attributes, further increase the VRAM needed for rendering. The substantial VRAM consumption characteristic of these methods poses significant challenges for implementing rendering on web platforms and resource-constrained devices, as they struggle to accommodate the high memory demands.

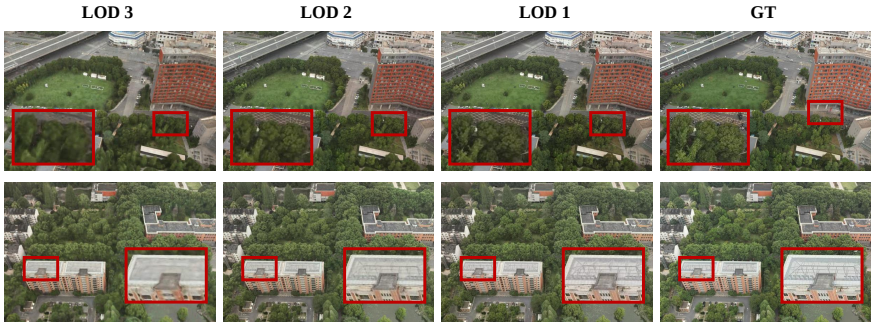

Fig. 3: Qualitative comparison of different levels-of-detail.

References

1. Chen, Z., Funkhouser, T., Hedman, P., Tagliasacchi, A.: Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16569–16578 (2023)
2. Gu, J., Jiang, M., Li, H., Lu, X., Zhu, G., Shah, S.A.A., Zhang, L., Bennamoun, M.: Ue4-nerf: Neural radiance field for real-time rendering of large-scale scene. Advances in Neural Information Processing Systems **36** (2024)
3. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics **42**(4) (July 2023), `https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/`
4. Liu, J.Y., Chen, Y., Yang, Z., Wang, J., Manivasagam, S., Urtasun, R.: Neural scene rasterization for large scene rendering in real time. In: The IEEE International Conference on Computer Vision (ICCV) (2023)
5. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: Seminal graphics: pioneering efforts that shaped the field, pp. 347–353 (1998)
6. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics (ToG) **41**(4), 1–15 (2022)
7. Reiser, C., Szeliski, R., Verbin, D., Srinivasan, P., Mildenhall, B., Geiger, A., Barron, J., Hedman, P.: Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. ACM Transactions on Graphics (TOG) **42**(4), 1–12 (2023)
8. Schönberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
9. Tancik, M., Casser, V., Yan, X., Pradhan, S., Mildenhall, B., Srinivasan, P.P., Barron, J.T., Kretzschmar, H.: Block-nerf: Scalable large scene neural view synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8248–8258 (2022)
10. Tancik, M., Weber, E., Ng, E., Li, R., Yi, B., Wang, T., Kristoffersen, A., Austin, J., Salahi, K., Ahuja, A., et al.: Nerfstudio: A modular framework for neural radiance field development. In: ACM SIGGRAPH 2023 Conference Proceedings. pp. 1–12 (2023)

11. Turki, H., Ramanan, D., Satyanarayanan, M.: Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12922–12931 (2022)
12. Xu, L., Xiangli, Y., Peng, S., Pan, X., Zhao, N., Theobalt, C., Dai, B., Lin, D.: Grid-guided neural radiance fields for large urban scenes. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8296–8306 (2023)
13. Yariv, L., Hedman, P., Reiser, C., Verbin, D., Srinivasan, P.P., Szeliski, R., Barron, J.T., Mildenhall, B.: Bakedsdf: Meshing neural sdfs for real-time view synthesis. In: ACM SIGGRAPH 2023 Conference Proceedings, SIGGRAPH 2023, Los Angeles, CA, USA, August 6-10, 2023. pp. 46:1–46:9. ACM (2023)