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

Kaiwen Song[1,2] Juyong Zhang[1]
[1]University of Science and Technology of China [2]Real Infinity Inc

Abstract

NeRF has significantly advanced 3D scene reconstruction, capturing intricate details across various environments. Existing methods have successfully leveraged radiance field baking to facilitate real-time rendering of small scenes. However, when applied to large-scale scenes, these techniques encounter significant challenges, struggling to provide a seamless real-time experience due to limited resources in computation, memory, and bandwidth. In this paper, we propose City-on-Web, which represents the whole scene by partitioning it into manageable blocks, each with its own Level-of-Detail, ensuring high fidelity, efficient memory management and fast rendering. Meanwhile, we carefully design the training and inference process such that the final rendering result on web is consistent with training. Thanks to our novel representation and carefully designed training/inference process, we are the first to achieve real-time rendering of large-scale scenes in resource-constrained environments. Extensive experimental results demonstrate that our method facilitates real-time rendering of large-scale scenes on a web platform, achieving 32FPS at 1080P resolution with an RTX 3060 GPU, while simultaneously achieving a quality that closely rivals that of state-of-the-art methods. Project page: https://ustc3dv.github.io/City-on-Web/.

1. Introduction

NeRF has significantly advanced the field of scene reconstruction, showing an unparalleled ability to capture complex details across diverse environments. Existing works have demonstrated its ability to render small scenes with exceptional quality and performance in real-time [3, 6, 11, 21, 24, 29, 36, 40, 43, 46–48]. NeRF has also been successfully applied to the rendering of large scenes in offline settings, achieving exceptional visual fidelity and generating intricately detailed results [14, 34, 37, 43, 45].

Despite these successes, real-time neural rendering of large scenes is profoundly challenging due to inherent computational power, memory, and bandwidth limitations across various devices. The challenges mainly include the following aspects. Firstly, traditional NeRF and its variants are resource-intensive, requiring substantial computational power that exceeds what is typically available in such constrained environments. Secondly, the video memory capacity on client devices is frequently limited, imposing significant restrictions on the capability to process and render substantial assets in real-time simultaneously. The substantial resource becomes a critical issue in the real-time rendering of large scenes, necessitating the quick loading and processing of extensive data sets. Lastly, the dependency on data retrieval from remote servers introduces latency, particularly under network bandwidth limitations, further complicating the real-time rendering process. These hurdles collectively form a significant barrier to delivering an uninterrupted and instantaneous visual experience for large-scale scenes.

To address these challenges in real-time rendering of large-scale scenes, we propose our proposed method, City-on-Web. Drawing inspiration from traditional graphics techniques for rendering large-scale scenes [7–10, 15, 23, 38], we partition the scene into manageable blocks and represent the scene with varying Levels-of-Detail (LOD). We utilize radiance field baking techniques [16, 30], which precompute and store rendering primitives into 3D atlas textures organized in a sparse grid within each block for real-time rendering. However, due to the unavoidable texture resource limitations of shaders, we cannot load all the atlas textures into a single shader. Hence, we represent the scene as a hierarchy of segmented blocks, each rendered by a dedicated shader during rendering.

Our block partition and LOD for scene representation bring major benefits for real-time rendering in environments with limited computing resources, memory, and bandwidth. **(1) High-Fidelity Reconstruction.** With divide and conquer strategy, we ensure that each block possesses sufficient representation ability to reconstruct fine details within the scene faithfully. Additionally, to ensure high fidelity in the rendered output during training, we simulate the blending of multiple shaders that are aligned with the rendering pipeline. **(2) Efficient Resource Management.** The block and LOD-based representation facilitates dynamic resource

management. It simplifies the loading and unloading process, adapting to the viewer's position and field of view in real-time. This dynamic load strategy greatly mitigates the bandwidth and memory demands typically associated with large-scale scene rendering, paving the way for smoother user experiences even on less capable devices. **(3) Fast Rendering.** Despite the abundance of resources required for rendering large scenes, we ensure real-time rendering efficiency by dividing the scene into non-overlapping blocks, with each shader granted access to resources within its designated block only. This block-rendering approach guarantees that performance does not degrade linearly with increased resources, even if we divide the scene into dozens of blocks. Our experiments demonstrate that City-on-Web can render photo-realistic large-scale scenes at 32FPS at 1080p resolution with an RTX 3060 GPU and uses only 18% of the VRAM and 16% of the payload size compared to current mesh-based methods [6, 46]. As our model maintains consistency between training and rendering, we have achieved similar reconstruction quality compared to state-of-the-art methods. To our knowledge, we are the first to achieve real-time neural rendering of large-scale scenes on the web.

2. Related Work

Large-scale Scene Reconstruction. For radiance field reconstruction of large-scale scenes, a key issue lies in enhancing the model's representational capacity to adequately capture and render extensive scenes. Block-NeRF [34] and Mega-NeRF [37] address this by adopting a divide-and-conquer strategy, segmenting expansive scenes into smaller blocks, and applying localized NeRF processing to each. This approach significantly improves both the reconstruction quality and the model's scalability to larger scenes. Switch-NeRF [50] employs a gating network to dispatch 3D points to different NeRF sub-networks. Grid-NeRF [45] utilizes a compact multiresolution feature plane and combines the strengths of smoothness from vanilla NeRF with the local detail capturing ability of feature grid-based methods [5, 26, 31], efficiently reconstructing large scenes with fine details. NeRF++ [48] enhances the reconstruction of unbounded scenes through its innovative multi-spherical representation. On the other hand, Mip-NeRF 360 [1] introduces a scene contraction function to effectively represent scenes that extend to infinity, addressing the challenge of vast spatial extents. F2-NeRF [41] takes this a step further by implementing a warping function for local spaces, ensuring a balance of computational resources and training data across different parts of the scene.

Real-time Rendering. Early works mainly focus on the real-time rendering of a simple single object. NSVF [21] improves NeRF by introducing a more efficient sparse voxel field, significantly accelerating rendering speed while maintaining high-quality output. KiloNeRF [29] utilizes thousands of small MLPs, each responsible for a tiny scene region, significantly reducing network evaluation time. In contrast, SNeRG [16] leverages pre-computed sparse grids, allowing for direct retrieval of radiance field information without needing network evaluation. Termi-NeRF [28] terminates ray marching in less impactful scene regions, slashing computation time. DONeRF [27] focuses on one sample using a depth oracle network, speeding up rendering while preserving scene quality. Recently, there have been developments that enable real-time rendering of neural radiance fields in small scenes. MERF [30] improves upon SNeRG by utilizing a voxel and triplane hybrid representation to reduce memory usage. MobileNeRF [6] introduces the polygon rasterization rendering pipeline, running NeRF-based novel view synthesis in real-time on mobile devices. BakedSDF [46] bakes volumetric representation into meshes and utilizes spherical harmonics for representing view-dependent color, while NeRF2Mesh [36] iteratively refine both the geometry and appearance of the mesh.

Level of Detail. Substantial works are devoted to integrating LOD methods into the fabric of traditional computer graphics [7–9, 15, 17, 20, 22, 23], aiming to streamline rendering processes, reduce memory footprint, bolster interactive responsiveness. Recently, some works begin to apply LOD to the neural implicit reconstruction. NGLoD [32] represents LOD through a sparse voxel octree, where each level of the octree corresponds to a different LOD, allowing for a finer discretization of the surface and more detailed reconstruction as the tree depth increases. Takikawa *et al*. [33] efficiently encode 3D signals into a compact, hierarchical representation using vector-quantized auto decoder method. BungeeNeRF [43] employs a hierarchical network structure, where the base network focuses on learning a coarse representation of the scene, and subsequent residual blocks are tasked with progressively refining this representation. TrimipRF [18] and LoD-Neus [51] leverage multi-scale triplane and voxel representations to capture scene details at different scales, effectively implementing anti-aliasing to enhance the rendering and reconstruction quality.

3. Background and Motivation

Our exploration begins with an in-depth analysis of two influential works, SNeRG [16] and MERF [30], which have both set benchmarks for real-time rendering of radiance field. SNeRG precomputes and stores a Neural Radiance Fields model in a sparse 3D voxel grid. Each active voxel in SNeRG contains several attributes: density, diffuse color, and specular feature vector that captures view-dependent effects. Additionally, an indirection grid is used to enhance rendering by either indicating empty macroblocks or pointing to detailed texels in a 3D texture atlas. This representa-

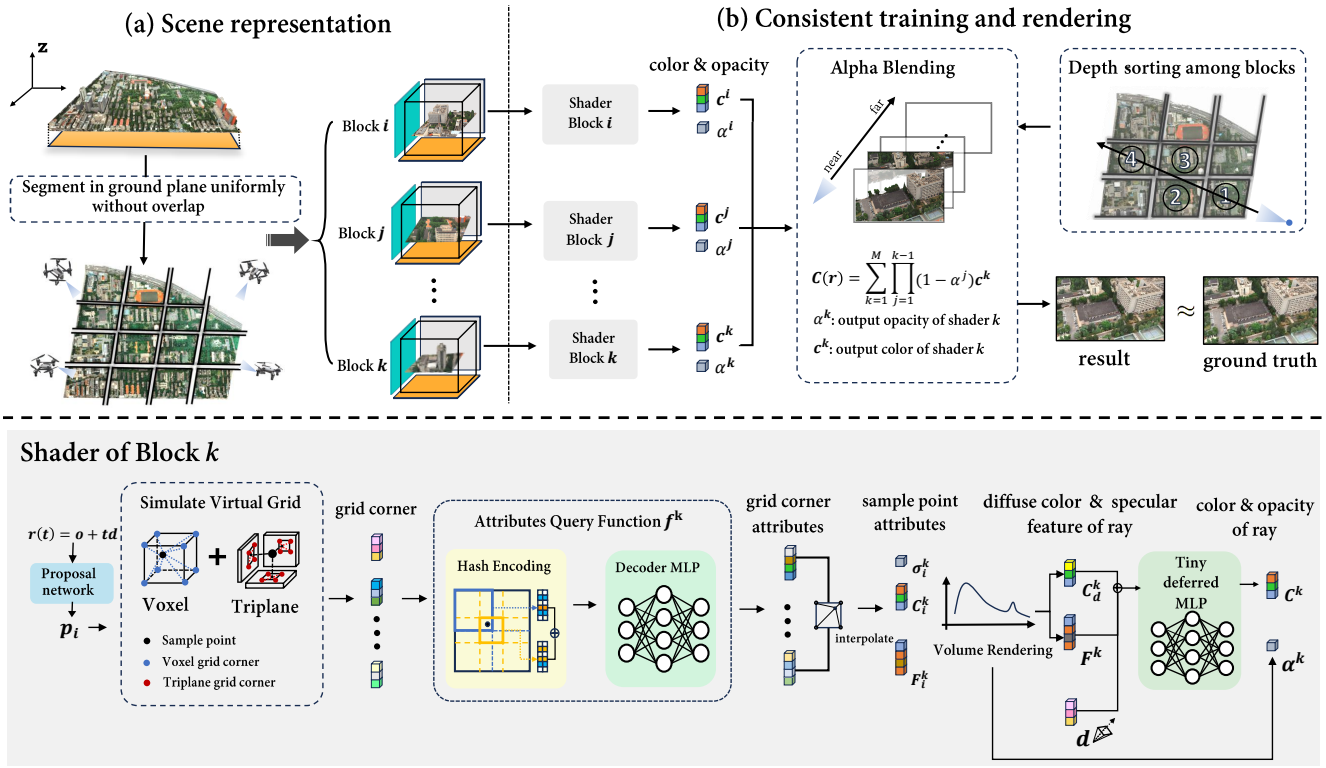

Figure 1. **Pipeline of training.** (a) We divide the entire scene into different blocks without overlap according to the ground plane. (b) For block k that the ray $\boldsymbol{r}(t)$ passes through, the corresponding shader renders the color $\boldsymbol{c}^k$ and opacity α^k of each block. We depth sort the blocks that ray traversed, and then render the final result through alpha blending that maintains 3D consistency.

tion allows real-time rendering on standard laptop GPUs.

The indirection grid assists in raymarching through the sparse 3D grid by passing empty regions and selectively accessing non-zero densities σ_i, diffuse colors $\boldsymbol{c}_i$, and feature vectors $\boldsymbol{f}_i$ during rendering. Integrating along each ray $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$, we compute the sum of the weights, which can be considered as the pixel's opacity:

$$\alpha(\boldsymbol{r}) = \sum_i w_i,\ w_i = \prod_{j=1}^{i-1}(1-\alpha_j)\alpha_i, \alpha_i = 1-e^{-\sigma_i \delta_i}. \quad (1)$$

The color $\boldsymbol{C}_d(\boldsymbol{r})$ and specular feature $\boldsymbol{F}_s(\boldsymbol{r})$ along the ray are accumulated using the same weights to compute the final diffuse color and specular feature of ray:

$$\boldsymbol{C}_d(\boldsymbol{r}) = \sum_i w_i \boldsymbol{c}_i, \quad \boldsymbol{F}_s(\boldsymbol{r}) = \sum_i w_i \boldsymbol{f}_i. \quad (2)$$

The step size during ray marching δ_i is equal to the voxel width for an occupied voxel. Subsequently, the accumulated diffuse color and specular feature vector, along with the positional encoding $PE(\cdot)$ of the ray's view direction, are concatenated to pass through a lightweight deferred MLP Φ to produce a view-dependent residual color:

$$\boldsymbol{C}(\boldsymbol{r}) = \boldsymbol{C}_d + \Phi(\boldsymbol{C}_d, \boldsymbol{F}_s, PE(\boldsymbol{d})). \quad (3)$$

While SNeRG achieves impressive real-time rendering results, its voxel representation demands substantial memory, which poses limitations for further applications. MERF presents a significant reduction in memory requirements in comparison to extant radiance field methods like SNeRG. By leveraging hybrid low-resolution sparse grid and 2D high-resolution triplanes, MERF optimizes the balance between performance and memory efficiency. Moreover, it incorporates two pivotal strategies to bridge the gap between training and rendering performance. First, MERF simulates finite grid approach during training, querying MLPs at virtual grid corners and applying interpolation to mimic the rendering process closely. Second, MERF simulates quantization during training and employs the straight-through estimator [2], allowing for the simulation of the quantization process while maintaining differentiability, enabling the model to learn and optimize with quantized values without introducing non-differentiable steps during the backward pass, ensuring a smooth training process.

These innovative methods for scene reconstruction offer promising results, but their direct applicability to large

scenes remains challenges. MERF's hybrid voxel-triplane representation, despite being memory-efficient, cannot capture large scenes with intricate details due to its fixed resolution constraint. In our efforts to reconstruct large scenes with high fidelity, dividing them into smaller blocks is a practical solution. This method helps in making sure the reconstruction is detailed and accurate.

However, this approach of dividing scenes means we will end up with more assets to deal during rendering. Web browsers have limits on how much memory they can use, which can make it hard to show these many pieces of large and detailed models. Another challenge is the data transmission during web rendering. It often requires pulling data from servers, and this can be slow due to network delays. As a result, users might experience long wait times when trying to load all the detailed pieces of a large scene at once for rendering. Drawing inspiration from traditional mesh dynamic resource loading and LOD [39], we generate LOD from our segmented reconstruction results which helps in minimizing the load of distant resources. Additionally, by employing a dynamic loading strategy for blocks, we significantly reduce VRAM usage and decrease the wait time for resource transmission.

4. Method

In this section, we present a method for representing and rendering large scenes on the web. Our approach uses hierarchical spatial partitioning and LOD to manage large-scale scenes dynamically (Sec. 4.1). We align the training and rendering stages to ensure consistency (Sec. 4.2), employing multiple shaders and alpha blending for seamless integration of scene blocks. Additionally, the framework includes optimization strategies (Sec. 4.3) and a process for generating LODs (Sec. 4.4) and baking the model (Sec. 4.5) for real-time rendering.

4.1. Large-scale Radiance Field

In the realm of scene reconstruction and rendering, NeRF has made significant strides, achieving compelling results. However, it faces inherent challenges when tasked with representing large scenes on the web. Using a single model to represent such vast scenes proves challenging due to its limited expressiveness, particularly in achieving a detailed and accurate reconstruction. Representing scenes with multiple models in a single resolution increases the overhead during rendering, leading to the loading of numerous resources, which is not conducive to efficient rendering.

To efficiently represent large scenes captured using the fly-through method, we employ hierarchical spatial partitioning combined with LOD to represent the scene. Specifically, we uniformly partition the area into varying blocks within our region of interest on the xy plane (i.e., the ground plane). Each set of partitioned blocks corresponds to a unique LOD level, allowing for dynamic and efficient representation. Within each block, we use a low resolution voxel with a high resolution triplane that stores density, diffuse color, and specular color, to represent the radiance field for web rendering. Additionally, for blocks along the periphery, we utilize the scene contraction function from MERF [30] to account for the data on the boundaries. For internal blocks, we simply adopt settings from the bounded scene.

During the training stage, our scene representation aligns with web rendering. We consistently utilize spatial partitioning to structure the scene into distinct blocks without overlap. However, we train the scene only with the finest level of LOD. Within block k , the following trainable components are introduced: (1) f^k: an attribute query function which adopts a hash encoding and a mlp decoder that outputs attributes of points such as densities, diffuse color and specular feature (2) Φ^k: a deferred MLP accounts for view-dependent effects. (3) ψ^k: a proposal MLP for sampling.

4.2. Consistent Training and Rendering

It is essential to ensure consistency between the training and rendering stages to achieve high-fidelity rendering results on the web as obtained during training. Due to the limited number of texture units within the web rendering environment, we are compelled to create multiple shaders to render distinct blocks. Specifically, one shader is allocated for storing the texture of an individual block. Each block subsequently renders an image respective to the current camera view. However, a simplistic averaging of these resultant rendering outputs can lead to discernible seams and does not ensure 3D consistency at the inter-block boundaries.

To address this problem, we simulate the process of multiple shaders rendering images and then linearly weighting them together using the volume rendering weights of blocks. For ray $\boldsymbol{r}(t)$, we uniformly sample between the near and far boundaries based on the scene's bounding box. Then, according to the sample coordinates, we query the corresponding block's proposal MLP to obtain the sample density, which is transformed into probability distributions along the rays. These probabilities guide a resampling strategy, ensuring a concentration on near-surface features with a few samples. Assuming that the proposal MLP yields samples passing through $\boldsymbol{M}$ blocks with a total of $\boldsymbol{N}$ samples, where each block k has $\boldsymbol{n}_k$ samples, we simulate web rendering by performing volume rendering within each block to obtain its individual rendering diffuse color $\boldsymbol{C}_d^k$, specular feature $\boldsymbol{F}^k$ and opacity α^k according to Eqs. (1) and (2). Then we get block k final rendering color $\boldsymbol{C}^k$ according to Eq. (3). Consequently, for the sake of 3D consistency in rendering, we depth-sort the blocks and apply volume rendering across multiple blocks in sequence, using

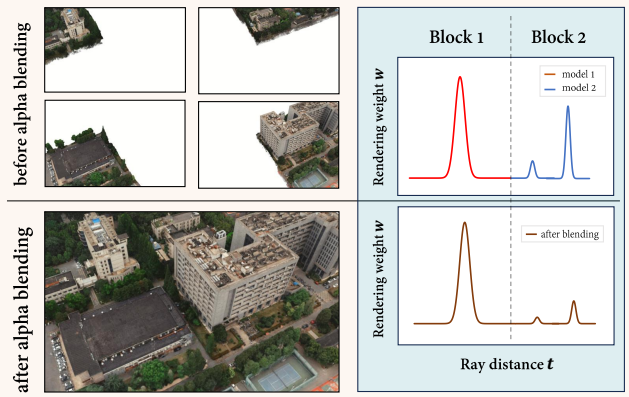

Figure 2. **Visualization of alpha blending during consistent training.** The left image shows the rendering results of four separate blocks by their shaders and the combined image after alpha blending, demonstrating how alpha blending correctly handles occlusion relationships. The right image visualizes the weights of rendering sample points before and after alpha blending. This illustration shows the correct occlusion management achieved through depth sorting of blocks followed by alpha blending.

opacity to generate the volume rendering weights:

$$\boldsymbol{C}(\boldsymbol{r}) = \sum_{k}^{M} \prod_{j=1}^{k-1} (1 - \alpha^j) \boldsymbol{C}^k. \tag{4}$$

Under the Lambertian surface setting where the specular color is zero, the diffuse color and feature vector obtained from volume rendering on the total of $\boldsymbol{N}$ ray samples from Eq. (2) are equal to the results produced by our approach of conducting volume rendering within each block followed by inter-block volume rendering Eq. (4). The proof is given in the supplementary. Thus, our rendering approach maintains 3D consistency and simulates multiple shader rendering on the web as shown in Fig. 2.

During rendering, for the sample point p_i, we need to access the voxel grid corner and triplane grid corner where the point is located and use interpolation to obtain the sample point's attributes. During training, we also simulate the voxel and triplane grid points to maintain consistency with the rendering process. By using grid corners' positions to query the attribute query function f_k, we obtain the attributes of the grid corners. Through interpolation, we acquire the attributes of the sample points in a manner similar to the rendering pipeline. This simulaition strategy ensures that the values used for volume rendering within each block during training are as closely matched as possible to the values queried from the baked textures.

4.3. Optimization

We use Charbonnier loss [4] for reconstruction and S3IM loss [44] to assist multiple blocks' model in capturing high-frequency details. Additionally, we use the interlevel loss to provide supervision signal for proposal MLP and distortion loss to reduce floaters like Mip-nerf 360 [1].

Moreover, we random uniform sample points set $\mathcal{P}$ within the bounding box of the scene and apply L_1 regularization on the alpha to encourage the blocks' model to predict sparse occupied space:

$$\mathcal{L}_{\text{sparse}} = \frac{1}{|\mathcal{P}|} \sum_{p_i \in \mathcal{P}} |\alpha_i| = \frac{1}{|\mathcal{P}|} \sum_{p_i \in \mathcal{P}} |1 - \sigma_i v|, \tag{5}$$

where v is the step size used in real-time rendering. Additionally, we introduce a regularization term for the opacity of the block. This regularization encourages the opacity of the block to be as close to 0 or 1 as possible, implying either full transparency or full opaqueness:

$$\mathcal{L}_{\text{opacity}} = -\sum_{k} (\alpha^k log(\alpha^k) + (1 - \alpha^k) log(1 - \alpha^k)). \tag{6}$$

In summary, the overall loss function is:

$$\begin{aligned} \mathcal{L}_{\text{train}} = \mathcal{L}_{\text{charbonnier}} + \lambda_1 \mathcal{L}_{\text{S3IM}} + \lambda_2 \mathcal{L}_{\text{interlevel}} + \\ \lambda_3 \mathcal{L}_{\text{distortion}} + \lambda_4 \mathcal{L}_{\text{sparse}} + \lambda_5 \mathcal{L}_{\text{opacity}}. \end{aligned} \tag{7}$$

4.4. LOD Generation

To guarantee superior rendering quality from elevated perspectives and simultaneously diminish the resource demand for distant scene elements, our method involves generating multiple LOD for the scene. We successfully attained the scene's lowest LOD in the training stage, ensuring maximal visual fidelity. For the downsampling and integration baking of multiple block models into a unified model, we initially freeze the training of hash encoding and decoder MLP components within these models. Subsequently, we proceed to retrain a new tiny deferred MLP. Following the deferred MLP's successful retraining, we simulate lower resolution virtual voxels and triplane grid corners within the scenes of these multiple blocks. Lastly, this retrained deferred MLP is refined in collaboration with the network responsible for generating grid corner attributes, thereby optimizing the entire rendering process.

4.5. Baking

After the training stage, we conduct block-based evaluation and store the MLP's outputs onto discrete grids, which generate segmented rendering resources. This approach facilitates efficient resource management for real-time rendering, as each block's resources are handled independently. Initially, we render all training rays to collect ray samples. Samples with alpha and weight values above a certain threshold are retained, and samples below the threshold are discarded. The preserved samples are used to mark the

Method	*BlockA*			*BlockE*			*Campus*			*Rubble*			*Building*		
Metric	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑
NeRFacto	25.28	0.469	0.691	24.61	0.441	0.687	23.47	0.255	0.689	19.02	0.538	0.512	17.70	0.442	0.502
Instant-NGP	24.16	0.595	0.643	22.93	0.599	0.619	**24.93**	0.380	0.703	20.37	0.629	0.478	17.92	0.625	0.424
Mega-NeRF	25.24	0.566	0.668	**25.61**	0.468	0.680	22.28	0.472	0.565	**23.68**	0.558	0.525	**20.30**	0.526	0.506
Grid-NeRF	25.37	0.478	0.705	24.43	0.483	0.693	-	-	-	-	-	-	-	-	-
Ours	**26.07**	**0.357**	**0.739**	25.07	**0.357**	**0.756**	24.73	**0.192**	**0.736**	21.32	**0.482**	**0.539**	19.71	**0.439**	**0.520**

Table 1. **Quantitative comparison on five large scene datasets.** We report PSNR, LPIPS, and SSIM on the test views. The **best** and second best results are highlighted. The '-' symbol indicates that Grid-NeRF [45] was not evaluated on these datasets due to difficulties in adjusting its training configs beyond the provided configurations for *Matrix City* [19], resulting in poor performance on other datasets.

Area(m^2)	Images	Overlap rate	Resolution	Altitude
1200×800	6515	95%	8192×5460	180m

Table 2. Overview of *Campus* dataset.

adjacent eight grid points as occupied in the binary grids. After generating binary grids to identify occupied voxels, we follow the MERF by baking high-resolution 2D planes and a low-resolution 3D voxel grid in each block. Only the non-empty 3D voxels are stored using a block-sparse format. We downsample the occupancy grid with max-pooling for efficient rendering and skip empty space. To further save storage, we compress textures into the PNG format.

5. Experiments

5.1. Experiments setup

Dataset and Metric. Our experiments span across various scales and environments. We have incorporated a real-world urban scene dataset (*Campus*) and public datasets consisting of real-world rural rubble scenes (*Rublle, Building*) [37] and synthetic city-scale data (*BlockA* and *BlockE* in *MatrixCity*) [19]. Our datasets were recorded under uniform, cloudy lighting conditions to minimize variation. To obtain precise pose information, we employed an annular capturing approach, which has a higher overlap rate compared to grid-based capturing methods. Tab. 2 presents an overview of our dataset. To assess the quality and fidelity of our reconstructions, we employ various evaluation metrics, including **PSNR**, **SSIM** and **LPIPS** [49].

Implementations and Baselines. Our method takes posed multi-view images captured using a fly-through camera as input. The training code is built on the nerfstudio framework [35] with tiny-cuda-nn [25] extension. And our real-time viewer is a JavaScript web application whose rendering is implemented through GLS. 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}. Our model is trained with $50k$ iterations on one NVIDIA A100 GPU. We split the scene into 24 non-overlapping blocks for *Campus* scene and split other scenes into four blocks. Moreover, we benchmark current real-time rendering methods using three critical parameters: Payload (**PL**), GPU Memory (**VRAM**), and Frames Per Scond (**FPS**). Payload refers to the essential data transmitted during the rendering process. We perform qualitative comparisons between our method and existing SOTA methods for large-scale reconstruction. The *Campus* dataset is partitioned into six sections based on the reconstruction content. NeRFacto, Instant-NGP, and Grid-NeRF were applied to one of these sections, while in other datasets , they are applied to the entire scene. NeRFacto and Instant-NGP are utilized with the highest hash encoding resolution of 8192^3. Similarly, Mega-NeRF divides the Campus dataset into 24 blocks and other datasets into four blocks. Our experiments focus on a single campus section for comparative analysis with existing real-time rendering methods.

5.2. Results Analysis

We systematically evaluate the performance of both baseline models and our method through qualitative and quantitative comparisons in Tab. 1 and Fig. 3. Notably, our method demonstrates a remarkable enhancement in visual fidelity as reflected by the SSIM and LPIPS metrics, which indicate the extent of detail restoration. Despite a reduction in PSNR compared to the SOTA methods, this is attributable to the fact that LPIPS and SSIM are more sensitive to the recovery of fine details, whereas PSNR mainly measures pixel-wise color accuracy. Our approach achieves higher fi-

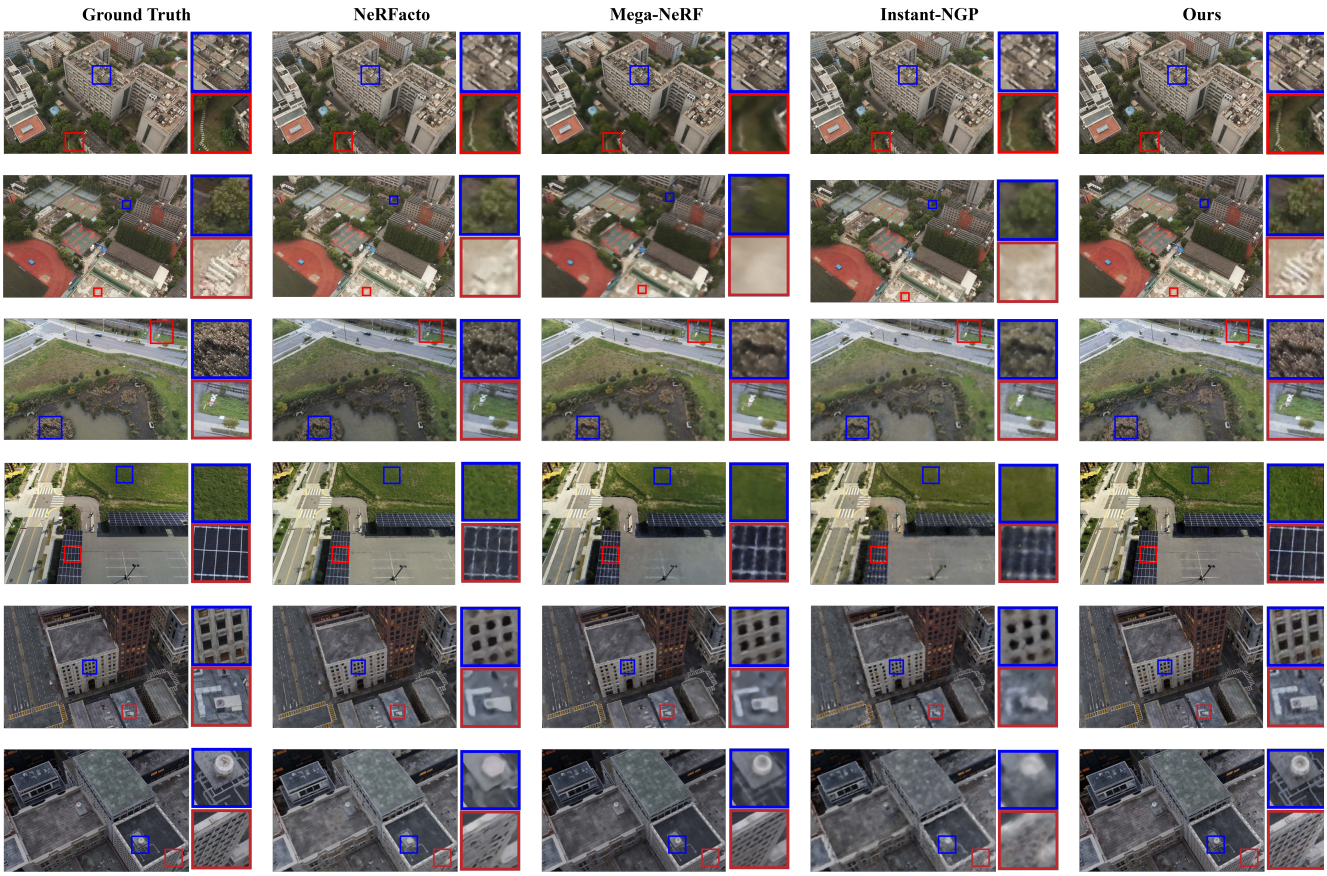

Figure 3. **Qualitative comparisons with existing SOTA methods.** By testing different methods across diverse scales and environments, it clearly reveals that our approach excels in recovering finer details and achieves a higher quality of reconstruction.

delity reconstructions, revealing finer details due to our partitioned reconstruction strategy. In our method, each ray is evaluated by the deferred MLP only once, as opposed to other methods that evaluate the MLP at every sample point. Consequently, while our method recovers more intricate geometrical detail, it frequently results in color discrepancies with the ground truth image due to unstable lighting conditions and variable exposure, as shown in Fig. 4.

In our evaluation, detailed in Tab. 3, we compare our method with current real-time rendering methods, using one segment of the *Campus* dataset for testing. These tests, are performed on an NVIDIA RTX 3060 Laptop GPU at a 1920×1080 resolution. The results demonstrate that our method excels in reconstruction quality. We represent each scene block using voxels and triplanes, and store the baked grid attributes as images. This strategy significantly reduces the payload. This reduction notably accelerates resource transmission for web-based rendering applications. However, it is observed that our frame rate during rendering is lower compared to other methods. This is attributed to their rendering pipeline based on mesh rasterization, in contrast to our method, which utilizes volume rendering.

	PSNR↑	SSIM↑	LPIPS↓	VRAM↓	PL↓	FPS↑
MobileNeRF	19.99	0.516	0.712	544.8	515.3	223
BakedSDF	22.24	0.627	0.413	712.3	242.1	68
Ours(Block)	**24.82**	**0.741**	**0.190**	**128.1**	**40.6**	51
Ours(Total)				526.6	114.4	46

Table 3. **Comparison with existing real-time methods.** We segment the scene into four blocks, while other methods reconstruct the whole scene. 'Ours(Block)' denotes the resource usage for one of these blocks, providing a fair comparison with other methods that process the scene as a single block. 'Ours (Total)' represents the cumulative resource usage for the entire scene.

5.3. LOD Result

Tab. 4 presents the quantitative rendering results at various LOD, along with the corresponding payload and VRAM usage. With increasing LOD, the resources required for rendering significantly decrease. Notably, our method's lowest LOD level still maintains high fidelity rendering results, as demonstrated in Fig. 5. Our LOD strategy significantly

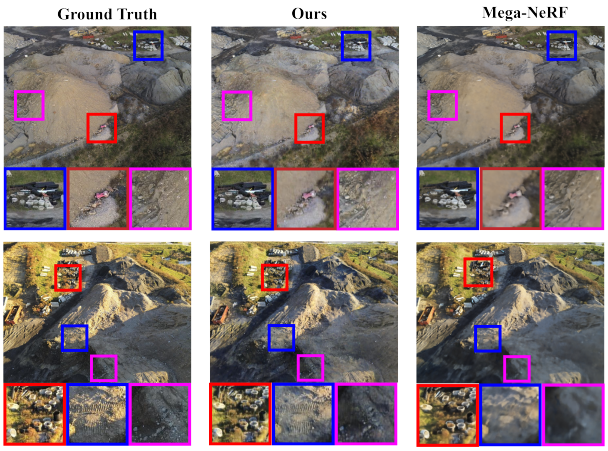

Figure 4. **Comparison to Mega-NeRF in *Rubble* dataset.** The *Rubble* dataset presents significant variations in lighting. Although our method recovers more detail than Mega-NeRF, our deferred shading model has limited ability to represent view-dependent colors and cannot accurately represent lighting and exposure changes in the data as view-dependent effects. This limitation results in slightly lower PSNR values.

	PSNR↑	SSIM↑	LPIPS↓	VRAM↓	PL↓
LOD3	23.72	0.660	0.306	132.1	40.2
LOD2	24.23	0.682	0.297	841.6	201.7
LOD1	24.73	0.736	0.192	3970.2	1259.6

Table 4. The LOD results on the whole *Campus* dataset.

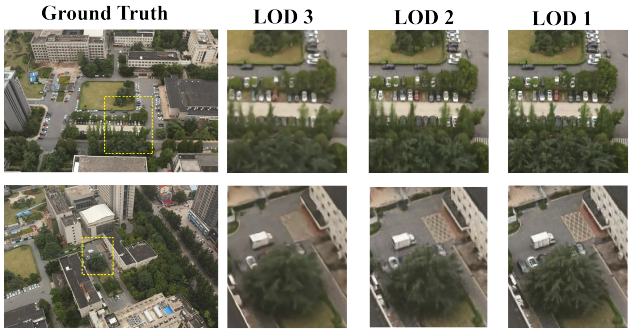

Figure 5. **Visualization of our LOD result.**

streamlines the management of resource loading on web platforms, which is particularly advantageous in rendering distant blocks, as it requires less VRAM. It is worth noting that the VRAM usage presented in Tab. 4 represents the cumulative memory consumption of all blocks. Our dynamic loading strategy adaptively selects resources to load based on the camera's field of view and the distance to each block, effectively keeping the peak VRAM usage around 1100MB.

	PSNR↑	SSIM↑	LPIPS↓	VRAM↓	PL↓	FPS↑
high res. model	24.05	0.710	0.201	540.9	147	32
w/o consistent	24.21	0.702	0.281	514.4	110	49
ours	24.82	0.741	0.190	526.6	114	46

Table 5. **Ablation Study.** The result is test on the one section of *Campus* dataset.

Figure 6. **Limitations of Web-on-City.**

5.4. Ablation Study

In Tab. 5, we conduct an ablation study of our method on one section of *Campus* dataset. Our model is trained for four blocks with low resolution (voxel resolution of 512^3 and triplane resolution of 2048^2). We also train a single model for the entire scene with high resolution (voxel resolution of 1024^3 and triplane resolution of 4096^2). Due to our non-overlapping scene partitioning strategy, these two representations have the same resolution across the entire scene. However, our reconstruction quality is higher, achieving better FPS and lower GPU memory usage, as well as reduced payload. We also removed our consistent training, including virtual grids and alpha blending during training. As shown in the table, our consistent training significantly improves reconstruction quality.

6. Conclusion and Discussion

In this work, we introduced City-on-Web, which to our knowledge is the first system that enables real-time neural rendering of large-scale scenes over web using laptop GPUs. Our integration of block partitioning with LOD has significantly reduced the payload on the web platform and improved resource management efficiency. We ensured high-fidelity rendering quality by maintaining consistency between training and rendering. Extensive experiments have also fully proved the effectiveness of City-on-Web.

Limitation & Future Work. As shown in Fig. 6, our method still has some limitations. Since we derive alpha blending across shaders based on the Lambertian surface assumption, visible seams may occur at the boundaries between blocks on non-Lambertian surfaces, such as water surfaces. Combining physically-based rendering with multiple shaders blending may alleviate this problem. The deferred MLP in City-on-Web has limited representation abil-

ity for view-dependent color, which might cause numerous near-camera floaters. Taking the similar strategy used in [12, 42] by utilizing priors to pre-trim the scene or preprocessing the data are possible solutions.

References

[1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 2, 5

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 3

[3] Junli Cao, Huan Wang, Pavlo Chemerys, Vladislav Shakhrai, Ju Hu, Yun Fu, Denys Makoviichuk, Sergey Tulyakov, and Jian Ren. Real-time neural light field on mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8328–8337, 2023. 1, 3

[4] Pierre Charbonnier, Laure Blanc-Féraud, Gilles Aubert, and Michel Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on image processing*, 6(2):298–311, 1997. 5

[5] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision*, pages 333–350. Springer, 2022. 2

[6] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. 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*, pages 16569–16578, 2023. 1, 2

[7] James H Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10): 547–554, 1976. 1, 2

[8] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 15–22, 2009.

[9] Mark Duchaineau, Murray Wolinsky, David E Sigeti, Mark C Miller, Charles Aldrich, and Mark B Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. In *Proceedings. Visualization'97 (Cat. No. 97CB36155)*, pages 81–88. IEEE, 1997. 2

[10] Epic Games. Unreal engine 4. https://www.unrealengine.com, 2023. Version 4.27. 1

[11] 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

[12] Lily Goli, Cody Reading, Silvia Selllán, Alec Jacobson, and Andrea Tagliasacchi. Bayes' rays: Uncertainty quantification for neural radiance fields. *arXiv preprint arXiv:2309.03185*, 2023. 9

[13] Jiaming Gu, Minchao Jiang, Hongsheng Li, Xiaoyuan Lu, Guangming Zhu, Syed Afaq Ali Shah, Liang Zhang, and Mohammed Bennamoun. Ue4-nerf: Neural radiance field for real-time rendering of large-scale scene. *arXiv preprint arXiv:2310.13263*, 2023. 2

[14] Jianfei Guo, Nianchen Deng, Xinyang Li, Yeqi Bai, Botian Shi, Chiyu Wang, Chenjing Ding, Dongliang Wang, and Yikang Li. Streetsurf: Extending multi-view implicit surface reconstruction to street views. *arXiv preprint arXiv:2306.04988*, 2023. 1

[15] Stefan Guthe, Michael Wand, Julius Gonser, and Wolfgang Straßer. Interactive rendering of large volume data sets. In *IEEE Visualization, 2002. VIS 2002.*, pages 53–60. IEEE, 2002. 1, 2

[16] 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, 2

[17] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery. 2

[18] Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19774–19783, 2023. 2

[19] Yixuan Li, Lihan Jiang, Linning Xu, Yuanbo Xiangli, Zhenzhi Wang, Dahua Lin, and Bo Dai. Matrixcity: A large-scale city dataset for city-scale neural rendering and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3205–3215, 2023. 6

[20] Peter Lindstrom and Valerio Pascucci. Visualization of large terrains made easy. In *Proceedings Visualization, 2001. VIS'01.*, pages 363–574. IEEE, 2001. 2

[21] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020. 1, 2

[22] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. In *ACM Siggraph 2004 Papers*, pages 769–776. 2004. 2

[23] David Luebke. *Level of detail for 3D graphics*. Morgan Kaufmann, 2003. 1, 2

[24] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021. 1

[25] Thomas Müller. tiny-cuda-nn, 2021. 6

[26] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 2

[27] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H Mueller, Chakravarty R Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. Donerf: Towards real-time rendering of compact neural radiance fields using depth oracle networks. In *Computer Graphics Forum*, pages 45–59. Wiley Online Library, 2021. 2

[28] Martin Piala and Ronald Clark. Terminerf: Ray termination prediction for efficient neural rendering. In *2021 International Conference on 3D Vision (3DV)*, pages 1106–1114. IEEE, 2021. 2

[29] 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, 2

[30] Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *ACM Transactions on Graphics (TOG)*, 42(4):1–12, 2023. 1, 2, 4

[31] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022. 2

[32] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11358–11367, 2021. 2

[33] Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings*. Association for Computing Machinery, 2022. 2

[34] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8248–8258, 2022. 1, 2

[35] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, et al. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–12, 2023. 6

[36] Jiaxiang Tang, Hang Zhou, Xiaokang Chen, Tianshu Hu, Errui Ding, Jingdong Wang, and Gang Zeng. Delicate textured mesh recovery from nerf via adaptive surface refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. 1, 2

[37] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. 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*, pages 12922–12931, 2022. 1, 2, 6

[38] Unity Technologies. Unity. https://unity.com, 2023. Version 2020.3. 1

[39] Gokul Varadhan and Dinesh Manocha. Out-of-core rendering of massive geometric environments. In *IEEE Visualization, 2002. VIS 2002.*, pages 69–76. IEEE, 2002. 4

[40] Ziyu Wan, Christian Richardt, Aljaž Božič, Chao Li, Vijay Rengarajan, Seonghyeon Nam, Xiaoyu Xiang, Tuotuo Li, Bo Zhu, Rakesh Ranjan, et al. Learning neural duplex radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8307–8316, 2023. 1

[41] Peng Wang, Yuan Liu, Zhaoxi Chen, Lingjie Liu, Ziwei Liu, Taku Komura, Christian Theobalt, and Wenping Wang. F2-nerf: Fast neural radiance field training with free camera trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4150–4159, 2023. 2

[42] Frederik Warburg, Ethan Weber, Matthew Tancik, Aleksander Holynski, and Angjoo Kanazawa. Nerfbusters: Removing ghostly artifacts from casually captured nerfs. *arXiv preprint arXiv:2304.10532*, 2023. 9

[43] Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In *European conference on computer vision*, pages 106–122. Springer, 2022. 1, 2

[44] Zeke Xie, Xindi Yang, Yujie Yang, Qi Sun, Yixiang Jiang, Haoran Wang, Yunfeng Cai, and Mingming Sun. S3im: Stochastic structural similarity and its unreasonable effectiveness for neural fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18024–18034, 2023. 5

[45] Linning Xu, Yuanbo Xiangli, Sida Peng, Xingang Pan, Nanxuan Zhao, Christian Theobalt, Bo Dai, and Dahua Lin. Grid-guided neural radiance fields for large urban scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8296–8306, 2023. 1, 2, 6

[46] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. 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*, pages 46:1–46:9. ACM, 2023. 1, 2

[47] 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.

[48] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 1, 2

[49] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 6

[50] MI Zhenxing and Dan Xu. Switch-nerf: Learning scene decomposition with mixture of experts for large-scale neural radiance fields. In *The Eleventh International Conference on Learning Representations*, 2022. 2

[51] Yiyu Zhuang, Qi Zhang, Ying Feng, Hao Zhu, Yao Yao, Xiaoyu Li, Yan-Pei Cao, Ying Shan, and Xun Cao. Anti-aliased neural implicit surfaces with encoding level of detail. *arXiv preprint arXiv:2309.10336*, 2023. 2

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

Supplementary Material

7. Proof of 3D consistency

For a given sampling point i, suppose it is located within the region $\mathcal{K}$ of block k, which contains a total of N_k sampling points. The diffuse color, feature, and opacity output by the shader of block k are denoted as $\boldsymbol{c}_i^k$, $\boldsymbol{F}_i^k$, and α_i^k respectively. Then, for this ray, the output diffuse color $\boldsymbol{c}_d^k$ and specular feature $\boldsymbol{F}^k$of block k are caculated as follows. $\boldsymbol{h}^k$ represents either the diffuse color or specular feature.

$$\begin{aligned} \boldsymbol{h}^k(\boldsymbol{r}) &= \sum_{i=1}^{N_k} \prod_{j=1}^{i-1} (1-\alpha_j^k) \cdot \alpha_i^k \boldsymbol{h}_i^k \\ \alpha^k &= \sum_{i=1}^{N_k} \prod_{j=1}^{i-1} (1-\alpha_j^k) \cdot \alpha_i^k \end{aligned} \tag{8}$$

By integrating the rendering results of each block's shader through the volume rendering between blocks, the final diffuse color $\boldsymbol{c}_d$ and specular feature $\boldsymbol{F}$ of the ray can be obtained.

$$\boldsymbol{h}(\boldsymbol{r}) = \sum_{k} \prod_{j=1}^{k-1} (1-\alpha^j) \cdot \boldsymbol{h}^k \tag{9}$$

Note that

$$\begin{aligned} & 1-\alpha^k \\ &= 1 - \sum_{i=1}^{N_k} \prod_{j=1}^{i-1} (1-\alpha_j^k) \cdot \alpha_i^k \\ &= 1 - \alpha_1^k - (1-\alpha_1^k)\alpha_2^k - (1-\alpha_1^k)(1-\alpha_2^k)\alpha_3^k - \cdots \\ &= (1-\alpha_1^k)(1-\alpha_2^k - (1-\alpha_2^k)\alpha_3^k - \cdots) \\ &= (1-\alpha_1^k)(1-\alpha_2^k)(1-\alpha_3^k - \cdots) \\ &\quad \vdots \\ &= \prod_{i=1}^{N_k} (1-\alpha_i^k) \end{aligned} \tag{10}$$

Assuming the block positions are already depth-sorted, meaning there are M sampling points along the ray, with each block k containing N_k sampling points, let α_i denote the i-th sampling point on the ray, and α_i^k denote the i-th sampling point in block k along the ray, then

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{r}) &= \sum_{k} \prod_{j=1}^{k-1} \prod_{i=1}^{N_j} (1-\alpha_i^j) \cdot \boldsymbol{h}^k \\ &= \sum_{k} \prod_{i=1}^{N_1+\cdots+N_{k-1}} (1-\alpha_i^j) \cdot \left(\sum_{i=1}^{N_k} \prod_{j=1}^{i-1} (1-\alpha_j^k) \cdot \alpha_i^k \boldsymbol{h}_i^k \right) \\ &= \sum_{k} \sum_{i=1}^{N_k} \prod_{i=1}^{N_1+\cdots+N_{k-1}} (1-\alpha_i^j) \prod_{j=1}^{i-1} (1-\alpha_j^k) \cdot \alpha_i^k \boldsymbol{h}_i^k \\ &= \sum_{i=1}^{M} \prod_{i=1}^{i-1} (1-\alpha_j) \alpha_i \boldsymbol{h}_i \end{aligned} \tag{11}$$

Eq. (11) shows that the diffuse color and specular feature we finally obtain by depth-sorting blocks and alpha blending along blocks are consistent with the results of volume rendering integration along the entire ray. Therefore, our method ensures the three-dimensional consistency of rendering.

8. Dataset

To demonstrate the effectiveness of our method, experiments are conducted on a variety of large scenes. The main experiments reported in this paper involve three types of environmental scene datasets of various scales. The *Campus* dataset is captured at a altitude of about 180 meters, covering an area of approximately 960,000 m^2. The *Matrix City* Dataset, captured at a altitude of 200 meters, is sparser than the *Campus* dataset, thus covering a larger area. The *Mill 19* dataset covers a total of about 200,000 m^2. We adopt a circular data capture method for photographing, as shown in Fig. 7. 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 times. We used Colmap 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.

9. Efficiency of Non-overlapping Partition Strategy

In the stage of segmenting the scene into distinct blocks, we initially rotate the scene to align it parallel with the xy-plane, then proceed to segment the entire space based on

(a) Camera Poses

(b) Examples from *Campus* Dataset

Figure 7. **Visulization of *Campus* Dataset.**

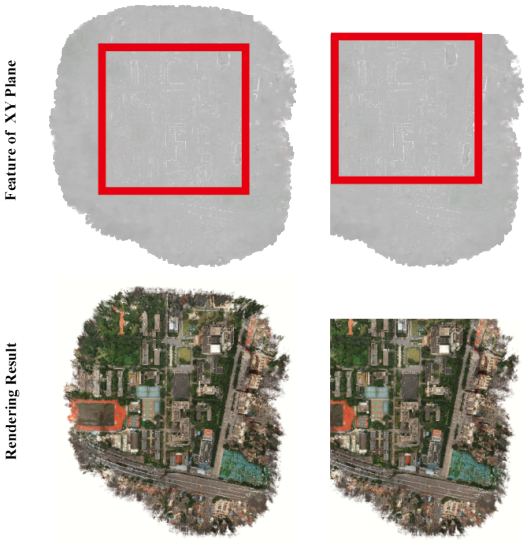

Figure 8. **Visulization of ou efficient non-overlapping partition strategy. The red square represents the** $[-1, 1]^2$**.**

the xy coordinates of spatial points. This strategy's merit lies in ensuring each segmented block is a bounded area. This is in contrast to methods like segmentation strategy of block-nerf [34] and [37], which cannot assure boundedness in the reconstructed area, potentially leading to substantial memory resource wastage as illustrated in the Fig. 8. Our approach allows us to represent the same size area with a bounded region of $[-1, 1]^3$, maintaining the same representation resolution, instead of using an unbounded region that contract to $[-2, 2]^3$ like in the MERF [30] and Mip-NeRF 360 [1]. Consequently, this enables the reduction of the resolution of the xy-plane from 4096^2 to 2048^2 without loss in performance. Thus we can effectively reduce the usage of VRAM, especially across the three planes.

10. Impletation 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 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:

$$\text{contract}(\mathbf{x})_j = \begin{cases} x_j & \text{if } \|\mathbf{x}\|_\infty \leq 1 \\ \frac{x_j}{\|\mathbf{x}\|_\infty} & \text{if } x_j \neq \|\mathbf{x}\|_\infty > 1 \\ \left(2 - \frac{1}{|x_j|}\right) \frac{x_j}{|x_j|} & \text{if } x_j = \|\mathbf{x}\|_\infty > 1 \end{cases}, \tag{12}$$

11. More Results

We provide additional qualitative results and on *Matrix City*, *Mill 19*, *Campus* dataset as shown in Fig. 9.

12. Discussion

Recently, some research has also enabled real-time rendering of large scenes. UE4-NERF [13], building on the MobileNerf framework, divides large scenes into smaller segments for reconstruction and then renders the large-scale scene using the mesh rasterization pipeline. Like MobileNerf, UE4-NERF begins with a 128^3 grid, which assumes an

even distribution of scene details in all directions. However, data captured through oblique photography often appears 'flat', meaning there's dense information when projected onto the xy plane, but sparser in the vertical direction due to mostly empty areas. Therefore, UE4-NERF needs more segments for large-scale reconstruction to ensure an even distribution of details in all directions in a block. For example, their Construction Site scene required about 40 blocks to reconstruct a $420m^2 \times 240m^2$ area, leading to significant memory and VRAM usage, approximately 25GB. Even with UE4's dynamic mesh-based loading, VRAM usage is around 11GB. Such high payload and VRAM demands make it challenging to extend this technology to web platforms and consumer-grade GPUs.

Additionally, NeuRas [3] has also made progress in real-time rendering of large scenes. It uses texture-less geometry from already reconstructed large scenes. NeuRas applys feature map textures and combines mesh rasterization results with view direction to query a small MLP for view-dependent colors. They optimize the feature texture and view-dependent MLP to enhance rendering results. However, this method also suffers from high memory usage. Our experiments show that obtaining texture-less geometry with ContextCapture[1] for the Campus scene requires about 6GB, and surface reconstruction via neural rendering methods needs approximately 10GB without mesh simplification.

Both these methods have achieved excellent results in real-time rendering of large scenes, but their significant memory and VRAM requirements limit their expansion to web platforms. Our method can implement real-time rendering of large scenes with a payload and VRAM consumption acceptable for web platforms and consumer-grade graphics cards.

[1] https://www.bentley.com/software/itwin-capture-modeler/

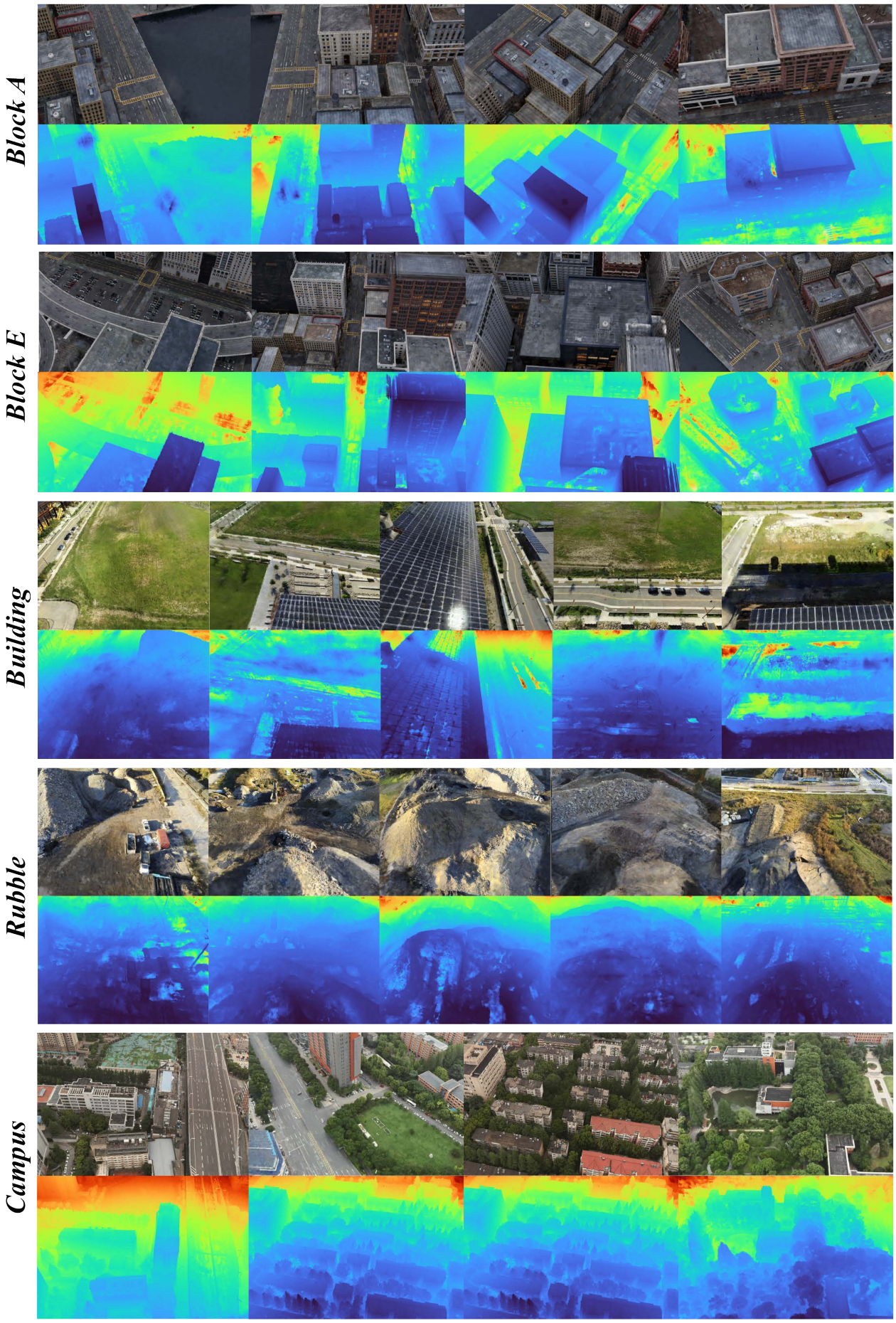

Figure 9. **Qualitative visualization of geometry and rendering result on five datasets.**