Block-NeRF: Scalable Large Scene Neural View Synthesis

Matthew Tancik[1*] Vincent Casser[2] Xinchen Yan[2] Sabeek Pradhan[2]
Ben Mildenhall[3] Pratul P. Srinivasan[3] Jonathan T. Barron[3] Henrik Kretzschmar[2]

[1]UC Berkeley [2]Waymo [3]Google Research

Figure 1. **Block-NeRF** is a method that enables large-scale scene reconstruction by representing the environment using multiple compact NeRFs that each fit into memory. At inference time, Block-NeRF seamlessly combines renderings of the relevant NeRFs for the given area. In this example, we reconstruct the Alamo Square neighborhood in San Francisco using data collected over 3 months. Block-NeRF can update individual blocks of the environment without retraining on the entire scene, as demonstrated by the construction on the right. Video results can be found on the project website waymo.com/research/block-nerf.

Abstract

We present Block-NeRF, a variant of Neural Radiance Fields that can represent large-scale environments. Specifically, we demonstrate that when scaling NeRF to render city-scale scenes spanning multiple blocks, it is vital to decompose the scene into individually trained NeRFs. This decomposition decouples rendering time from scene size, enables rendering to scale to arbitrarily large environments, and allows per-block updates of the environment. We adopt several architectural changes to make NeRF robust to data captured over months under different environmental conditions. We add appearance embeddings, learned pose refinement, and controllable exposure to each individual NeRF, and introduce a procedure for aligning appearance between adjacent NeRFs so that they can be seamlessly combined. We build a grid of Block-NeRFs from 2.8 million images to create the largest neural scene representation to date, capable of rendering an entire neighborhood of San Francisco.

1. Introduction

Recent advancements in neural rendering such as Neural Radiance Fields [42] have enabled photo-realistic reconstruction and novel view synthesis given a set of posed camera images [3, 40, 45]. Earlier works tended to focus on small-scale and object-centric reconstruction. Though some methods now address scenes the size of a single room or building, these are generally still limited and do not naïvely scale up to *city-scale* environments. Applying these methods to large environments typically leads to significant artifacts and low visual fidelity due to limited model capacity.

Reconstructing large-scale environments enables several important use-cases in domains such as autonomous driving [32, 44, 68] and aerial surveying [14, 35]. One example is mapping, where a high-fidelity map of the entire operating domain is created to act as a powerful prior for a variety of problems, including robot localization, navigation, and collision avoidance. Furthermore, large-scale scene reconstructions can be used for closed-loop robotic simulations [13]. Autonomous driving systems are commonly evaluated by re-simulating previously encountered scenarios; however, any deviation from the recorded encounter may change the vehicle's trajectory, requiring high-fidelity novel view renderings along the altered path. Beyond basic view synthesis, scene conditioned NeRFs are also capable of changing environmental lighting conditions such as camera exposure, weather, or time of day, which can be used to further augment simulation scenarios.

*Work done as an intern at Waymo.

Reconstructing such large-scale environments introduces additional challenges, including the presence of transient objects (cars and pedestrians), limitations in model capacity, along with memory and compute constraints. Furthermore, training data for such large environments is highly unlikely to be collected in a single capture under consistent conditions. Rather, data for different parts of the environment may need to be sourced from different data collection efforts, introducing variance in both scene geometry (*e.g.*, construction work and parked cars), as well as appearance (*e.g.*, weather conditions and time of day).

We extend NeRF with appearance embeddings and learned pose refinement to address the environmental changes and pose errors in the collected data. We additionally add exposure conditioning to provide the ability to modify the exposure during inference. We refer to this modified model as a Block-NeRF. Scaling up the network capacity of Block-NeRF enables the ability to represent increasingly large scenes. However this approach comes with a number of limitations; rendering time scales with the size of the network, networks can no longer fit on a single compute device, and updating or expanding the environment requires retraining the entire network.

To address these challenges, we propose dividing up large environments into individually trained Block-NeRFs, which are then rendered and combined dynamically at inference time. Modeling these Block-NeRFs independently allows for maximum flexibility, scales up to arbitrarily large environments and provides the ability to update or introduce new regions in a piecewise manner without retraining the entire environment as demonstrated in Figure 1. To compute a target view, only a subset of the Block-NeRFs are rendered and then composited based on their geographic location compared to the camera. To allow for more seamless compositing, we propose an appearance matching technique which brings different Block-NeRFs into visual alignment by optimizing their appearance embeddings.

2. Related Work

2.1. Large Scale 3D Reconstruction

Researchers have been developing and refining techniques for 3D reconstruction from large image collections for decades [1, 16, 33, 47, 57, 77], and much current work relies on mature and robust software implementations such as COLMAP to perform this task [55]. Nearly all of these reconstruction methods share a common pipeline: extract 2D image features (such as SIFT [39]), match these features across different images, and jointly optimize a set of 3D points and camera poses to be consistent with these matches (the well-explored problem of bundle adjustment [23, 65]). Extending this pipeline to city-scale data is largely a matter of implementing highly robust and parallelized versions of these algorithms, as explored in work such as Photo Tourism [57]

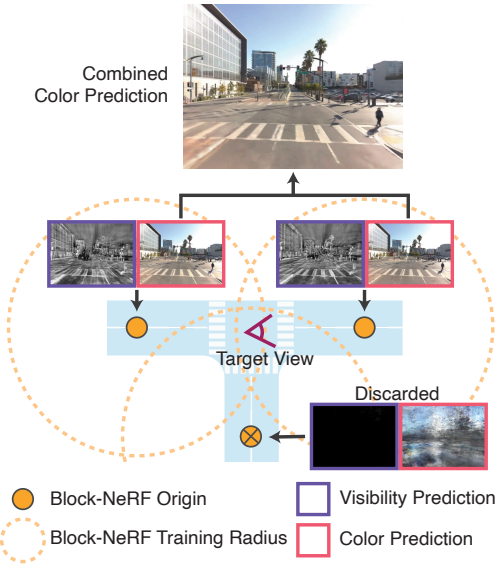

Figure 2. The scene is split into multiple Block-NeRFs that are each trained on data within some radius (dotted orange line) of a specific Block-NeRF origin coordinate (orange dot). To render a target view in the scene, the visibility maps are computed for all of the NeRFs within a given radius. Block-NeRFs with low visibility are discarded (bottom Block-NeRF) and the color output is rendered for the remaining blocks. The renderings are then merged based on each block origin's distance to the target view.

and Building Rome in a Day [1]. Core graphics research has also explored breaking up scenes for fast high quality rendering [38].

These approaches typically output a camera pose for each input image and a sparse 3D point cloud. To get a complete 3D scene model, these outputs must be further processed by a dense multi-view stereo algorithm (*e.g.*, PMVS [18]) to produce a dense point cloud or triangle mesh. This process presents its own scaling difficulties [17]. The resulting 3D models often contain artifacts or holes in areas with limited texture or specular reflections as they are challenging to triangulate across images. As such, they frequently require further postprocessing to create models that can be used to render convincing imagery [56]. However, this task is mainly the domain of novel view synthesis, and 3D reconstruction techniques primarily focus on geometric accuracy.

In contrast, our approach does not rely on large-scale SfM to produce camera poses, instead performing odometry using various sensors on the vehicle as the images are collected [64].

2.2. Novel View Synthesis

Given a set of input images of a given scene and their camera poses, novel view synthesis seeks to render observed scene content from previously unobserved viewpoints, allowing a user to navigate through a recreated environment with high visual fidelity.

Geometry-based Image Reprojection. Many approaches to view synthesis start by applying traditional 3D reconstruction techniques to build a point cloud or triangle mesh representing the scene. This geometric "proxy" is then used to reproject pixels from the input images into new camera views, where they are blended by heuristic [6] or learning-based methods [24, 52, 53]. This approach has been scaled to long trajectories of first-person video [31], panoramas collected along a city street [30], and single landmarks from the Photo Tourism dataset [41]. Methods reliant on geometry proxies are limited by the quality of the initial 3D reconstruction, which hurts their performance in scenes with complex geometry or reflectance effects.

Volumetric Scene Representations. Recent view synthesis work has focused on unifying reconstruction and rendering and learning this pipeline end-to-end, typically using a volumetric scene representation. Methods for rendering small baseline view interpolation often use feed-forward networks to learn a mapping directly from input images to an output volume [15, 76], while methods such as Neural Volumes [37] that target larger-baseline view synthesis run a global optimization over all input images to reconstruct every new scene, similar to traditional bundle adjustment.

Neural Radiance Fields (NeRF) [42] combines this single-scene optimization setting with a neural scene representation capable of representing complex scenes much more efficiently than a discrete 3D voxel grid; however, its rendering model scales very poorly to large-scale scenes in terms of compute. Followup work has proposed making NeRF more efficient by partitioning space into smaller regions, each containing its own lightweight NeRF network [48, 49]. Unlike our method, these network ensembles must be trained jointly, limiting their flexibility. Another approach is to provide extra capacity in the form of a coarse 3D grid of latent codes [36]. This approach has also been applied to compress detailed 3D shapes into neural signed distance functions [62] and to represent large scenes using occupancy networks [46].

We build our Block-NeRF implementation on top of mip-NeRF [3], which improves aliasing issues that hurt NeRF's performance in scenes where the input images observe the scene from many different distances. We incorporate techniques from NeRF in the Wild (NeRF-W) [40], which adds a latent code per training image to handle inconsistent scene appearance when applying NeRF to landmarks from the Photo Tourism dataset. NeRF-W creates a separate NeRF for each landmark from thousands of images, whereas our approach combines many NeRFs to reconstruct a coherent large environment from *millions* of images. Our model also incorporates a learned camera pose refinement which has been explored in previous works [34, 59, 66, 69, 70].

Some NeRF-based methods use segmentation data to isolate and reconstruct static [67] or moving objects (such as people or cars) [44, 73] across video sequences. As we focus primarily on reconstructing the environment itself, we choose to simply mask out dynamic objects during training.

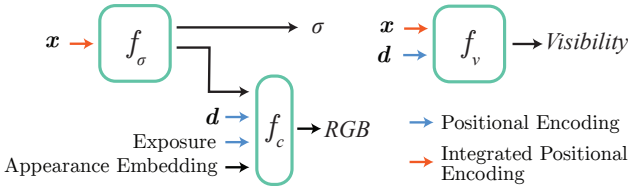

Figure 3. Our model is an extension of the model presented in mip-NeRF [3]. The first MLP f_σ predicts the density σ for a position $\mathbf{x}$ in space. The network also outputs a feature vector that is concatenated with viewing direction $\mathbf{d}$, the exposure level, and an appearance embedding. These are fed into a second MLP f_c that outputs the color for the point. We additionally train a visibility network f_v to predict whether a point in space was visible in the training views, which is used for culling Block-NeRFs during inference.

2.3. Urban Scene Camera Simulation

Camera simulation has become a popular data source for training and validating autonomous driving systems on interactive platforms [2, 28]. Early works [13, 19, 51, 54] synthesized data from scripted scenarios and manually created 3D assets. These methods suffered from domain mismatch and limited scene-level diversity. Several recent works tackle the simulation-to-reality gaps by minimizing the distribution shifts in the simulation and rendering pipeline. Kar *et al*. [26] and Devaranjan *et al*. [12] proposed to minimize the scene-level distribution shift from rendered outputs to real camera sensor data through a learned scenario generation framework. Richter *et al*. [50] leveraged intermediate rendering buffers in the graphics pipeline to improve photorealism of synthetically generated camera images.

Towards the goal of building photo-realistic and scalable camera simulation, prior methods [9, 32, 68] leverage rich multi-sensor driving data collected during a single drive to reconstruct 3D scenes for object injection [9] and novel view synthesis [68] using modern machine learning techniques, including image GANs for 2D neural rendering. Relying on a sophisticated surfel reconstruction pipeline, SurfelGAN [68] is still susceptible to errors in graphical reconstruction and can suffer from the limited range and vertical field-of-view of LiDAR scans. In contrast to existing efforts, our work tackles the 3D rendering problem and is capable of modeling the real camera data captured from multiple drives under varying environmental conditions, such as weather and time of day, which is a prerequisite for reconstructing large-scale areas.

3. Background

We build upon NeRF [42] and its extension mip-NeRF [3]. Here, we summarize relevant parts of these methods. For details, please refer to the original papers.

3.1. NeRF and mip-NeRF Preliminaries

Neural Radiance Fields (NeRF) [42] is a coordinate-based neural scene representation that is optimized through a differentiable rendering loss to reproduce the appearance of a set of input images from known camera poses. After optimization, the NeRF model can be used to render previously unseen viewpoints.

The NeRF scene representation is a pair of multilayer perceptrons (MLPs). The first MLP f_σ takes in a 3D position $\mathbf{x}$ and outputs volume density σ and a feature vector. This feature vector is concatenated with a 2D viewing direction $\mathbf{d}$ and fed into the second MLP f_c, which outputs an RGB color $\mathbf{c}$. This architecture ensures that the output color can vary when observed from different angles, allowing NeRF to represent reflections and glossy materials, but that the underlying geometry represented by σ is only a function of position.

Each pixel in an image corresponds to a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ through 3D space. To calculate the color of $\mathbf{r}$, NeRF randomly samples distances $\{t_i\}_{i=0}^N$ along the ray and passes the points $\mathbf{r}(t_i)$ and direction $\mathbf{d}$ through its MLPs to calculate σ_i and $\mathbf{c}_i$. The resulting output color is

$$\mathbf{c}_{\text{out}} = \sum_{i=1}^{N} w_i \mathbf{c}_i, \quad \text{where } w_i = T_i(1 - e^{-\Delta_i \sigma_i}), \tag{1}$$

$$T_i = \exp\left(-\sum_{j<i} \Delta_j \sigma_j\right), \quad \Delta_i = t_i - t_{i-1}\,. \tag{2}$$

The full implementation of NeRF iteratively resamples the points t_i (by treating the weights w_i as a probability distribution) in order to better concentrate samples in areas of high density.

To enable the NeRF MLPs to represent higher frequency detail [63], the inputs $\mathbf{x}$ and $\mathbf{d}$ are each preprocessed by a componentwise sinusoidal positional encoding γ_{PE}:

$$\gamma_{\text{PE}}(z) = [\sin(2^0 z), \cos(2^0 z), \ldots, \sin(2^{L-1} z), \cos(2^{L-1} z)] \tag{3}$$

where L is the number of levels of positional encoding.

NeRF's MLP f_σ takes a single 3D point as input. However, this ignores both the relative footprint of the corresponding image pixel and the length of the interval $[t_{i-1}, t_i]$ along the ray $\mathbf{r}$ containing the point, resulting in aliasing artifacts when rendering novel camera trajectories. Mip-NeRF [3] remedies this issue by using the projected pixel footprint to sample conical frustums along the ray rather than intervals. To feed these frustums into the MLP, mip-NeRF approximates each of them as Gaussian distributions with parameters $\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i$ and replaces the positional encoding γ_{PE} with its expectation over the input Gaussian

$$\gamma_{\text{IPE}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathbb{E}_{\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}[\gamma_{\text{PE}}(\boldsymbol{X})]\,, \tag{4}$$

referred to as an *integrated* positional encoding.

4. Method

Training a single NeRF does not scale when trying to represent scenes as large as cities. We instead propose splitting the environment into a set of Block-NeRFs that can be independently trained in parallel and composited during inference. This independence enables the ability to expand the environment with additional Block-NeRFs or update blocks without retraining the entire environment (see Figure 1). We dynamically select relevant Block-NeRFs for rendering, which are then composited in a smooth manner when traversing the scene. To aid with this compositing, we optimize the appearances codes to match lighting conditions and use interpolation weights computed based on each Block-NeRF's distance to the novel view.

4.1. Block Size and Placement

The individual Block-NeRFs should be arranged to collectively ensure full coverage of the target environment. We typically place one Block-NeRF at each intersection, covering the intersection itself and any connected street 75% of the way until it converges into the next intersection (see Figure 1). This results in a 50% overlap between any two adjacent blocks on the connecting street segment, making appearance alignment easier between them. Following this procedure means that the block size is variable; where necessary, additional blocks may be introduced as connectors between intersections. We ensure that the training data for each block stays exactly within its intended bounds by applying a geographical filter. This procedure can be automated and only relies on basic map data such as OpenStreetMap [22].

Note that other placement heuristics are also possible, as long as the entire environment is covered by at least one Block-NeRF. For example, for some of our experiments, we instead place blocks along a single street segment at uniform distances and define the block size as a sphere around the Block-NeRF Origin (see Figure 2).

4.2. Training Individual Block-NeRFs

4.2.1 Appearance Embeddings

Given that different parts of our data may be captured under different environmental conditions, we follow NeRF-W [40] and use Generative Latent Optimization [5] to optimize per-image appearance embedding vectors, as shown in Figure 3.

Figure 4. The appearance codes allow the model to represent different lighting and weather conditions.

This allows the NeRF to explain away several appearance-changing conditions, such as varying weather and lighting. We can additionally manipulate these appearance embeddings to interpolate between different conditions observed in the training data (such as cloudy versus clear skies, or day and night). Examples of rendering with different appearances can be seen in Figure 4. In § 4.3.3, we use test-time optimization over these embeddings to match the appearance of adjacent Block-NeRFs, which is important when combining multiple renderings.

4.2.2 Learned Pose Refinement

Although we assume that camera poses are provided, we find it advantageous to learn regularized pose offsets for further alignment. Pose refinement has been explored in previous NeRF based models [34,59,66,70]. These offsets are learned per driving segment and include both a translation and a rotation component. We optimize these offsets jointly with the NeRF itself, significantly regularizing the offsets in the early phase of training to allow the network to first learn a rough structure prior to modifying the poses.

4.2.3 Exposure Input

Training images may be captured across a wide range of exposure levels, which can impact NeRF training if left unaccounted for. We find that feeding the camera exposure information to the appearance prediction part of the model allows the NeRF to compensate for the visual differences (see Figure 3). Specifically, the exposure information is processed as $\gamma_{\text{PE}}(\text{shutter speed} \times \text{analog gain}/t)$ where γ_{PE} is a sinusoidal positional encoding with 4 levels, and t is a scaling factor (we use $1{,}000$ in practice). An example of different learned exposures can be found in Figure 5.

4.2.4 Transient Objects

While our method accounts for variation in appearance using the appearance embeddings, we assume that the scene geometry is consistent across the training data. Any movable objects (*e.g.* cars, pedestrians) typically violate this assumption. We therefore use a semantic segmentation model [10] to produce masks of common movable objects, and ignore masked areas during training. While this does not account for changes in otherwise static parts of the environment, *e.g.* construction, it accommodates most common types of geometric inconsistency.

4.2.5 Visibility Prediction

When merging multiple Block-NeRFs, it can be useful to know whether a specific region of space was visible to a given NeRF during training. We extend our model with an additional small MLP f_v that is trained to learn an approximation of the *visibility* of a sampled point (see Figure 3). For each sample along a training ray, f_v takes in the location and view direction and regresses the corresponding transmittance of the point (T_i in Equation 2). The model is trained alongside f_σ, which provides supervision. Transmittance represents how visible a point is from a particular input camera: points in free space or on the surface of the first intersected object will have transmittance near 1, and points inside or behind the first visible object will have transmittance near 0. If a point is seen from some viewpoints but not others, the regressed transmittance value will be the average over all training cameras and lie between zero and one, indicating that the point is partially observed. Our visibility prediction is similar to the visibility fields proposed by Srinivasan *et al.* [58]. However, they used an MLP to predict visibility to environment lighting for the purpose of recovering a relightable NeRF model, while we predict visibility to training rays.

The visibility network is small and can be run independently from the color and density networks. This proves useful when merging multiple NeRFs, since it can help to determine whether a specific NeRF is likely to produce meaningful outputs for a given location, as explained in § 4.3.1. The visibility predictions can also be used to determine locations to perform appearance matching between two NeRFs, as detailed in § 4.3.3.

4.3. Merging Multiple Block-NeRFs

4.3.1 Block-NeRF Selection

The environment can be composed of an arbitrary number of Block-NeRFs. For efficiency, we utilize two filtering mechanisms to only render relevant blocks for the given target viewpoint. We only consider Block-NeRFs that are

Figure 5. Our model is conditioned on exposure, which helps account for exposure changes present in the training data. This allows users to alter the appearance of the output images in a human-interpretable manner during inference.

within a set radius of the target viewpoint. Additionally, for each of these candidates, we compute the associated visibility. If the mean visibility is below a threshold, we discard the Block-NeRF. An example of visibility filtering is provided in Figure 2. Visibility can be computed quickly because its network is independent of the color network, and it does not need to be rendered at the target image resolution. After filtering, there are typically one to three Block-NeRFs left to merge.

4.3.2 Block-NeRF Compositing

We render color images from each of the filtered Block-NeRFs and interpolate between them using inverse distance weighting between the camera origin c and the centers x_i of each Block-NeRF. Specifically, we calculate the respective weights as $w_i \propto \text{distance}(c, x_i)^{-p}$, where p influences the rate of blending between Block-NeRF renders. The interpolation is done in 2D image space and produces smooth transitions between Block-NeRFs. We also explore other interpolation methods in § 5.4.

4.3.3 Appearance Matching

The appearance of our learned models can be controlled by an appearance latent code after the Block-NeRF has been trained. These codes are randomly initialized during training and therefore the same code typically leads to different appearances when fed into different Block-NeRFs. This is undesirable when compositing as it may lead to inconsistencies between views. Given a target appearance in one of the Block-NeRFs, we aim to match its appearance in the remaining blocks. To accomplish this, we first select a 3D *matching location* between pairs of adjacent Block-NeRFs. The visibility prediction at this location should be high for both Block-NeRFs.

Given the matching location, we freeze the Block-NeRF network weights and only optimize the appearance code of the target in order to reduce the ℓ_2 loss between the respective area renders. This optimization is quick, converging within 100 iterations. While not necessarily yielding perfect alignment, this procedure aligns most global and low-frequency attributes of the scene, such as time of day, color balance, and weather, which is a prerequisite for successful compositing. Figure 6 shows an example optimization, where appearance matching turns a daytime scene into nighttime to match the adjacent Block-NeRF.

The optimized appearance is iteratively propagated through the scene. Starting from one root Block-NeRF, we optimize the appearance of the neighboring ones and continue the process from there. If multiple blocks surrounding a target Block-NeRF have already been optimized, we consider each of them when computing the loss.

5. Results and Experiments

In this section we will discuss our datasets and experiments. The architectural and optimization specifics are provided in the supplement. The supplement also provides comparisons to reconstructions from COLMAP [55], a traditional Structure from Motion approach. This reconstruction is sparse and fails to represent reflective surfaces and the sky.

5.1. Datasets

We perform experiments on datasets that we collect specifically for the task of novel view synthesis of large-scale scenes. Our dataset is collected on public roads using data collection vehicles. While several large-scale driving datasets already exist, they are not designed for the task of view synthesis. For example, some datasets lack sufficient camera coverage (*e.g.*, KITTI [21], Cityscapes [11]) or prioritize visual diversity over repeated observations of a target area (*e.g.*, NuScenes [7], Waymo Open Dataset [61], Argoverse [8]). Instead, they are typically designed for tasks such as object detection or tracking, where similar observations across drives can lead to generalization issues.

We capture both long-term sequence data (100 s or more), as well as distinct sequences captured repeatedly in a particular target area over a period of several months. We use image data captured from 12 cameras that collectively provide a 360° view. 8 of the cameras provide a complete surround view from the roof of the car, with 4 additional cameras located at the vehicle front pointing forward and sideways. Each camera captures images at 10 Hz and stores a scalar exposure value. The vehicle pose is known and all cameras are calibrated. Using this information, we calculate the corresponding camera ray origins and directions in a common coordinate system, also accounting for the rolling shutter of the cameras. As described in § 4.2.4, we use a semantic segmentation model [10] to detect movable objects.

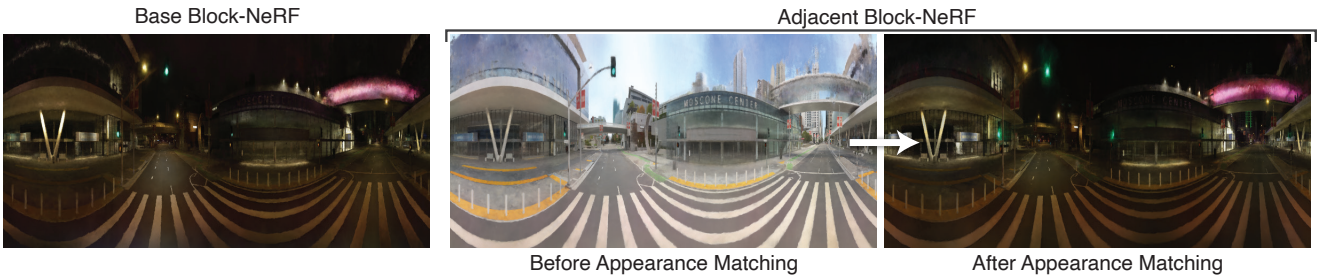

Figure 6. When rendering scenes based on multiple Block-NeRFs, we use appearance matching to obtain a consistent appearance across the scene. Given a fixed target appearance for one of the Block-NeRFs (left image), we optimize the appearances of the adjacent Block-NeRFs to match. In this example, appearance matching produces a consistent night appearance across Block-NeRFs.

San Francisco Alamo Square Dataset. We select San Francisco's Alamo Square neighborhood as the target area for our scalability experiments. The dataset spans an area of approximately $960\,\mathrm{m} \times 570\,\mathrm{m}$, and was recorded in June, July, and August of 2021. We divide this dataset into 35 Block-NeRFs. Example renderings and Block-NeRF placements can be seen in Figure 1. To best appreciate the scale of the reconstruction, please refer to supplementary videos. Each Block-NeRF was trained on data from 38 to 48 different data collection runs, adding up to a total driving time of 18 to 28 minutes each. After filtering out some redundant image captures (*e.g.* stationary captures), each Block-NeRF is trained on between 64,575 to 108,216 images. The overall dataset is composed of $13.4\,\mathrm{h}$ of driving time sourced from 1,330 different data collection runs, with a total of 2,818,745 training images.

San Francisco Mission Bay Dataset. We choose San Francisco's Mission Bay District as the target area for our baseline, block size, and placement experiments. Mission Bay is an urban environment with challenging geometry and reflective facades. We identified a long stretch on Third Street with far-range visibility, making it an interesting test case. Notably, this dataset was recorded in a single capture in November 2020, with consistent environmental conditions allowing for simple evaluation. This dataset was recorded over $100\,\mathrm{s}$, in which the data collection vehicle traveled $1.08\,\mathrm{km}$ and captured 12,000 total images from 12 cameras. We will release this single-capture dataset to aid reproducibility.

5.2. Model Ablations

We ablate our model modifications on a single intersection from the Alamo Square dataset. We report PSNR, SSIM, and LPIPS [75] metrics for the test image reconstructions in Table 1. The test images are split in half vertically, with the appearance embeddings being optimized on one half and tested on the other. We also provide qualitative examples in Figure 7. Mip-NeRF alone fails to properly reconstruct the scene and is prone to adding non-existent geometry and cloudy artifacts to explain the differences in appearance.

	NeRFs	PSNR↑	SSIM↑	LPIPS↓
	mip-NeRF	17.86	0.563	0.509
Ours	-Appearance	20.13	0.611	0.458
Ours	-Exposure	23.55	**0.649**	0.418
Ours	-Pose Opt.	23.05	0.625	0.442
Ours	Full	**23.60**	**0.649**	**0.417**

Table 1. Ablations of different Block-NeRF components on a single intersection in the Alamo Square dataset. We show the performance of mip-NeRF as a baseline, as well as the effect of removing individual components from our method.

When our method is not trained with appearance embeddings, these artifacts are still present. If our method is not trained with pose optimization, the resulting scene is blurrier and can contain duplicated objects due to pose misalignment. Finally, the exposure input marginally improves the reconstruction, but more importantly provides us with the ability to change the exposure during inference.

5.3. Block-NeRF Size and Placement

# Blocks	Weights / Total	Size	Compute	PSNR↑	SSIM↑	LPIPS↓
1	0.25M / 0.25M	$544\,\mathrm{m}$	$1\times$	23.83	0.825	0.381
4	0.25M / 1.00M	$271\,\mathrm{m}$	$2\times$	25.55	0.868	0.318
8	0.25M / 2.00M	$116\,\mathrm{m}$	$2\times$	26.59	0.890	0.278
16	0.25M / 4.00M	$54\,\mathrm{m}$	$2\times$	**27.40**	**0.907**	**0.242**
1	1.00M / 1.00M	$544\,\mathrm{m}$	$1\times$	24.90	0.852	0.340
4	0.25M / 1.00M	$271\,\mathrm{m}$	$0.5\times$	25.55	0.868	0.318
8	0.13M / 1.00M	$116\,\mathrm{m}$	$0.25\times$	25.92	0.875	0.306
16	0.07M / 1.00M	$54\,\mathrm{m}$	$0.125\times$	**25.98**	**0.877**	**0.305**

Table 2. Comparison of different numbers of Block-NeRFs for reconstructing the Mission Bay dataset. Splitting the scene into multiple Block-NeRFs improves the reconstruction accuracy, even when holding the total number of weights constant (bottom section). The number of blocks determines the size of the area each block is trained on and the relative compute expense at inference time.

We compare performance on our Mission Bay dataset versus the number of Block-NeRFs used. We show details in Table 2, where depending on granularity, the Block-NeRF sizes range from as small as $54\,\mathrm{m}$ to as large as $544\,\mathrm{m}$. We ensure that each pair of adjacent blocks overlaps by 50% and compare other overlap percentages in the supplement.

Figure 7. Model ablation results on multi segment data. Appearance embeddings help the network avoid adding cloudy geometry to explain away changes in the environment like weather and lighting. Removing exposure slightly decreases the accuracy. The pose optimization helps sharpen the results and removes ghosting from repeated objects, as observed with the telephone pole in the first row.

All were evaluated on the same set of held-out test images spanning the entire trajectory. We consider two regimes, one where each Block-NeRF contains the same number of weights (top section) and one where the total number of weights across all Block-NeRFs is fixed (bottom section). In both cases, we observe that increasing the number of models improves the reconstruction metrics. In terms of computational expense, parallelization during training is trivial as each model can be optimized independently across devices. At inference, our method only requires rendering Block-NeRFs near the target view. Depending on the scene and NeRF layout, we typically render between one to three NeRFs. We report the relative compute expense in each setting without assuming any parallelization, which however would be possible and lead to an additional speed-up. Our results imply that splitting the scene into multiple lower capacity models can reduce the overall computational cost as not all of the models need to be evaluated (see bottom section of Table 2).

5.4. Interpolation Methods

Interpolation	Consistent?	PSNR↑	SSIM↑	LPIPS↓
Nearest	–	26.40	0.887	0.280
IDW 2D	✓	26.59	0.890	0.278
IDW 3D	–	26.57	0.890	0.278
Pixelwise Visibility	–	27.39	0.906	0.242
Imagewise Visibility	–	27.41	0.907	0.242

Table 3. Comparison of interpolation methods. For our flythrough video results, we opt for 2D inverse distance weighting (IDW) as it produces temporally consistent results.

We explore different interpolation methods in Table 3. The simple method of only rendering the nearest Block-NeRF to the camera requires the least amount of compute but results in harsh jumps when transitioning between blocks. These transitions can be smoothed by using inverse distance weighting (IDW) between the camera and Block-NeRF centers, as described in § 4.3.2. We also explored a variant of IDW where the interpolation was performed over projected 3D points predicted by the expected Block-NeRF depth. This method suffers when the depth prediction is incorrect, leading to artifacts and temporal incoherence.

Finally, we experiment with weighing the Block-NeRFs based on per-pixel and per-image predicted visibility. This produces sharper reconstructions of further-away areas but is prone to temporal inconsistency. Therefore, these methods are best used only when rendering still images. Further details are provided in the supplement.

6. Limitations and Future Work

The proposed method handles transient objects by filtering them out during training via masking using a segmentation algorithm. If objects are not properly masked, they can cause artifacts in the resulting renderings. For example, the shadows of cars often remain, even when the car itself is correctly removed. Vegetation also breaks this assumption as foliage changes seasonally and moves in the wind; this results in blurred representations of trees and plants. Similarly, temporal inconsistencies in the training data, such as construction work, are not automatically handled and require the manual retraining of the affected blocks. Further, the inability to render scenes containing dynamic objects currently limits the applicability of Block-NeRF towards closed-loop simulation tasks in robotics. In the future, these issues could be addressed by learning transient objects during the optimization [40], or directly modeling dynamic objects [44, 67]. In particular, the scene could be composed of multiple Block-NeRFs of the environment and individual controllable object NeRFs. Separation can be facilitated by the use of segmentation masks or bounding boxes.

In our model, distant objects in the scene are not sampled with the same density as nearby objects which leads to blurrier reconstructions. This is an issue with sampling unbounded volumetric representations. Techniques proposed in NeRF++ [74] and concurrent Mip-NeRF 360 [4] could potentially be used to produce sharper renderings of distant objects.

In many applications, real-time rendering is key, but NeRFs are computationally expensive to render (up to multiple seconds per image). Several NeRF caching techniques [20, 25, 72] or a sparse voxel grid [36] could be used to enable real-time Block-NeRF rendering. Similarly, multiple concurrent works have demonstrated techniques to speed up training of NeRF style representations by multiple orders of magnitude [43, 60, 71].

7. Conclusion

In this paper we propose Block-NeRF, a method that reconstructs arbitrarily large environments using NeRFs. We demonstrate the method's efficacy by building an entire neighborhood in San Francisco from 2.8M images, forming the largest neural scene representation to date. We accomplish this scale by splitting our representation into multiple blocks that can be optimized independently. At such a scale, the data collected will necessarily have transient objects and variations in appearance, which we account for by modifying the underlying NeRF architecture. We hope that this can inspire future work in large-scale scene reconstruction using modern neural rendering methods.

References

[1] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 2011. 2

[2] Alexander Amini, Igor Gilitschenski, Jacob Phillips, Julia Moseyko, Rohan Banerjee, Sertac Karaman, and Daniela Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters*, 2020. 3

[3] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021. 1, 3, 4

[4] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *arXiv preprint arXiv:2111.12077*, 2021. 9

[5] Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the latent space of generative networks. *arXiv:1707.05776*, 2017. 4

[6] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. *Computer graphics and interactive techniques*, 2001. 3

[7] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *CVPR*, 2020. 6

[8] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. *CVPR*, 2019. 6

[9] Yun Chen, Frieda Rong, Shivam Duggal, Shenlong Wang, Xinchen Yan, Sivabalan Manivasagam, Shangjie Xue, Ersin Yumer, and Raquel Urtasun. Geosim: Realistic video simulation via geometry-aware composition for self-driving. *CVPR*, 2021. 3

[10] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. *CVPR*, 2020. 5, 6

[11] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CVPR*, 2016. 6

[12] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. Meta-sim2: Unsupervised learning of scene structure for synthetic data generation. *ECCV*, 2020. 3

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *Conference on robot learning*, 2017. 1, 3

[14] Dawei Du, Yuankai Qi, Hongyang Yu, Yifan Yang, Kaiwen Duan, Guorong Li, Weigang Zhang, Qingming Huang, and Qi Tian. The unmanned aerial vehicle benchmark: Object detection and tracking. *ECCV*, 2018. 1

[15] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. *CVPR*, 2016. 3

[16] Christian Früh and Avideh Zakhor. An automated method for large-scale, ground-based city model acquisition. *IJCV*, 2004. 2

[17] Yasutaka Furukawa, Brian Curless, Steven M Seitz, and Richard Szeliski. Towards internet-scale multi-view stereo. *CVPR*, 2010. 2

[18] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis. *IEEE TPAMI*, 2010. 2

[19] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. *CVPR*, 2016. 3

[20] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *arXiv:2103.10380*, 2021. 9

[21] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. *CVPR*, 2012. 6

[22] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive computing*, 2008. 4

[23] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004. 2

[24] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (TOG)*, 2018. 3

[25] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *arXiv:2103.14645*, 2021. 9

[26] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. *ICCV*, 2019. 3

[27] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 2013. 13

[28] Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. Drivegan: Towards a controllable high-quality neural simulation. *CVPR*, 2021. 3

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 12

[30] Johannes Kopf, Billy Chen, Richard Szeliski, and Michael Cohen. Street slide: browsing street level imagery. *ACM Transactions on Graphics (TOG)*, 2010. 3

[31] Johannes Kopf, Michael Cohen, and Rick Szeliski. First-person hyperlapse videos. *SIGGRAPH*, 2014. 3

[32] Wei Li, CW Pan, Rong Zhang, JP Ren, YX Ma, Jin Fang, FL Yan, QC Geng, XY Huang, HJ Gong, et al. Aads: Augmented autonomous driving simulation using data-driven algorithms. *Science robotics*, 2019. 1, 3

[33] Xiaowei Li, Changchang Wu, Christopher Zach, Svetlana Lazebnik, and Jan-Michael Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. *ECCV*, 2008. 2

[34] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. *arXiv preprint arXiv:2104.06405*, 2021. 3, 5

[35] Andrew Liu, Richard Tucker, Varun Jampani, Ameesh Makadia, Noah Snavely, and Angjoo Kanazawa. Infinite nature: Perpetual view generation of natural scenes from a single image. *ICCV*, 2021. 1

[36] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 3, 9

[37] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *SIGGRAPH*, 2019. 3

[38] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. *Siggraph*, 2004. 2

[39] David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004. 2

[40] 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. *CVPR*, 2021. 1, 3, 4, 8

[41] Moustafa Meshry, Dan B. Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural rerendering in the wild. *CVPR*, 2019. 3

[42] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *ECCV*, 2020. 1, 3, 4

[43] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv:2201.05989*, Jan. 2022. 9

[44] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. *CVPR*, 2021. 1, 3, 8

[45] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021. 1

[46] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020. 3

[47] Marc Pollefeys, David Nistér, J-M Frahm, Amir Akbarzadeh, Philippos Mordohai, Brian Clipp, Chris Engels, David Gallup, S-J Kim, Paul Merrell, et al. Detailed real-time urban 3d reconstruction from video. *IJCV*, 2008. 2

[48] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. Derf: Decomposed radiance fields. *CVPR*, 2021. 3

[49] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. *ICCV*, 2021. 3

[50] Stephan R Richter, Hassan Abu AlHaija, and Vladlen Koltun. Enhancing photorealism enhancement. *arXiv:2105.04619*, 2021. 3

[51] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. *ECCV*, 2016. 3

[52] Gernot Riegler and Vladlen Koltun. Free view synthesis. *ECCV*, 2020. 3

[53] Gernot Riegler and Vladlen Koltun. Stable view synthesis. *CVPR*, 2021. 3

[54] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. *CVPR*, 2016. 3

[55] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. *CVPR*, 2016. 2, 6, 12

[56] Qi Shan, Riley Adams, Brian Curless, Yasutaka Furukawa, and Steven M. Seitz. The visual turing test for scene reconstruction. *3DV*, 2013. 2

[57] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. *SIGGRAPH*, 2006. 2

[58] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. NeRV:

Neural reflectance and visibility fields for relighting and view synthesis. *CVPR*, 2021. 5

[59] Shih-Yang Su, Frank Yu, Michael Zollhöfer, and Helge Rhodin. A-nerf: Articulated neural radiance fields for learning human shape, appearance, and pose. *Advances in Neural Information Processing Systems*, 34, 2021. 3, 5

[60] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. *arXiv preprint arXiv:2111.11215*, 2021. 9

[61] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. *CVPR*, 2020. 6

[62] 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. *CVPR*, 2021. 3

[63] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020. 4

[64] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 2002. 2

[65] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. *International workshop on vision algorithms*, 1999. 2

[66] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf–: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021. 3, 5

[67] Bangbang Yang, Yinda Zhang, Yinghao Xu, Yijin Li, Han Zhou, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. Learning object-compositional neural radiance field for editable scene rendering. *ICCV*, 2021. 3, 8

[68] Zhenpei Yang, Yuning Chai, Dragomir Anguelov, Yin Zhou, Pei Sun, Dumitru Erhan, Sean Rafferty, and Henrik Kretzschmar. Surfelgan: Synthesizing realistic sensor data for autonomous driving. *CVPR*, 2020. 1, 3

[69] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33, 2020. 3

[70] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. iNeRF: Inverting neural radiance fields for pose estimation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021. 3, 5

[71] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021. 9

[72] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. *arXiv:2103.14024*, 2021. 9

[73] Jiakai Zhang, Xinhang Liu, Xinyi Ye, Fuqiang Zhao, Yanshun Zhang, Minye Wu, Yingliang Zhang, Lan Xu, and Jingyi Yu. Editable free-viewpoint video using a layered neural representation. *ACM Transactions on Graphics (TOG)*, 2021. 3

[74] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 9

[75] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CVPR*, 2018. 7

[76] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *arXiv:1805.09817*, 2018. 3

[77] Siyu Zhu, Runze Zhang, Lei Zhou, Tianwei Shen, Tian Fang, Ping Tan, and Long Quan. Very large-scale global SFM by distributed motion averaging. *CVPR*, 2018. 2

A. Model Parameters / Optimization Details

Our network follows the mip-NeRF structure. The network f_σ is composed of 8 layers with width 512 (Mission Bay experiments) or 1024 (all other experiments). f_c has 3 layers with width 128 and f_v has 4 layers with width 128. The appearance embeddings are 32 dimensional. We train each Block-NeRF using the Adam [29] optimizer for $300\,\mathrm{K}$ iterations with a batch size of 16384. Similar to mip-NeRF, the learning rate is an annealed logarithmically from $2 \cdot 10^{-3}$ to $2 \cdot 10^{-5}$, with a warm up phase during the first 1024 iterations. The coarse and fine networks are sampled 256 times during training and 512 times when rendering the videos. The visibility is supervised with MSE loss and is scaled by 10^{-6}. The learned pose correction consists of a position offset and a 3×3 residual rotation matrix, which is added to the identity matrix and normalized before being applied to ensure it is orthogonal. The pose corrections are initialized to 0 and their element-wise $\ell 2$ norm is regularized during training. This regularization is scaled by 10^5 at the start of training and linearly decays to 10^{-1} after 5000 iterations. This allows the network to learn initial geometry prior to applying pose offsets.

Each Block-NeRF takes between 9 and 24 hours to train (depending on hyperparameters). We train each Block-NeRF on 32 TPU v3 cores available through Google Cloud Compute, which combined offer a total of 1680 TFLOPS and 512 GB memory. Rendering an 1200×900px image for a single Block-NeRF takes approximately 5.9 seconds. Multiple Block-NeRF can be processed in parallel during inference (typically fewer than 3 Block-NeRFs need to be rendered for a single frame).

B. Block-NeRF Size and Placement

We include qualitative comparisons in Figure 9 on the Mission Bay dataset to complement the quantitative comparisons in (§5.3, Table 2). In this figure, we provide comparisons on two regimes, one where each Block-NeRF contains the same number of weights (left section) and one where the total number of weights across all Block-NeRFs is fixed (right section).

C. Block-NeRF Overlap Comparison

In the main paper, we include experiments on Block-NeRF size and placement (§5.3). For these experiments, we assumed a relative overlap of 50% between each pair of Block-NeRFs, which aids with appearance alignment.

Table 4 is a direct extension of Table 2 in the main paper and shows the effect of varying block overlap in the 8 block scenario. Note that varying the overlap changes the spatial block size. The original setting in the main paper is marked with an asterisk.

The metrics imply that reducing overlap is beneficial for image quality metrics. However, this can likely be attributed to the resulting reduction in block size. In practice, having an overlap between blocks is important to avoid temporal artifacts when interpolating between Block-NeRFs.

Overlap	Size	PSNR↑	SSIM↑	LPIPS↓
0%	$77\,\mathrm{m}$	26.77	0.895	0.262
25%	$97\,\mathrm{m}$	26.75	0.894	0.269
50%*	$116\,\mathrm{m}$	26.59	0.890	0.278
75%	$136\,\mathrm{m}$	26.51	0.887	0.283

Table 4. Effect of different NeRF overlaps in the 8 block scenario with 0.25M weights per block (2M weights in total). The original setting used in the main paper is marked*.

D. Block-NeRF Interpolation Details

We experiment with multiple methods to interpolate between Block-NeRFs and find that simple inverse distance weighting (IDW) in image space produces the most appealing videos due to temporal smoothness. We use an IDW power p of 4 for the Alamo Square renderings and a power of 1 for the Mission Bay renderings. We experiment with 3D inverse distance weighting for each individual pixel by projecting the rendered pixels into 3D space using the expected ray termination depth from the Block-NeRF closest to the target view. The color value of the projected pixel is then determined using inverse distance weighting with the nearest Block-NeRFs. Artifacts occur in the resulting composited renders due to noise in the depth predictions. We also experiment with using the Block-NeRF predicted visibility for interpolation. We consider imagewise visibility where we take the mean visibility of the entire image and pixelwise visibility where were directly utilize the per-pixel visibility predictions. Both of these methods lead to sharper results but come at the cost of temporal inconsistencies. Finally we compare to nearest neighbor interpolation where we only render the Block-NeRF closest to the target view. This results in harsh jumps when transiting between Block-NeRFs.

E. Structure from Motion (COLMAP)

We use COLMAP [55] to reconstruct the Mission Bay dataset. We first split the dataset into 8 overlapping blocks with $97\,\mathrm{m}$ radius each based on camera positions (each block has roughly 25% overlap with the adjacent block). The bundle adjustment step takes most of the time in reconstruction and we do not see significant improvements if we increase the radius per block. We mask out movable objects when extracting feature points for matching, using the same segmentation model as Block-NeRF. We assume a pinhole camera model and provide camera intrinsics and camera pose as priors for running structure-from-motion. We then run

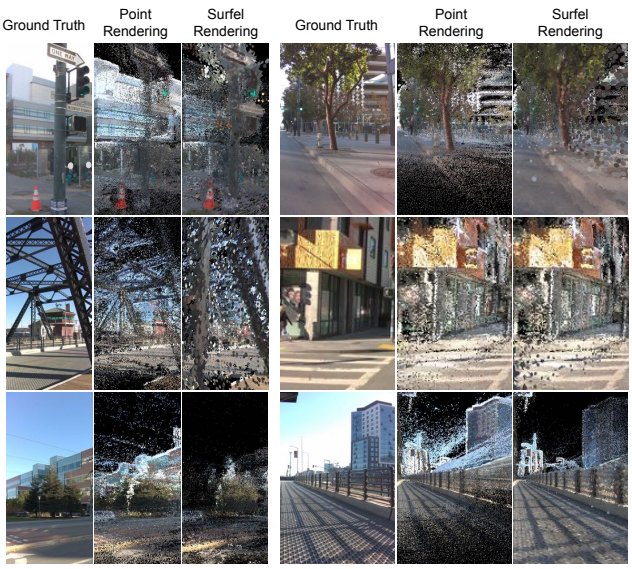

Figure 8. Qualitative results for COLMAP. We demonstrate the two rendering options using the fused pointcloud computed by COLMAP.

multi-view stereo within each block to produce dense depth and normal maps in 3D and produce a dense point cloud of the scene. In our preliminary experiments, we ran Poisson meshing [27] on the fused dense pointcloud to reconstruct textured meshes but found that the method fails to produce reasonably-looking results due to the challenging geometry and depth errors introduced by reflective surfaces and the sky. Instead, we leverage the fused pointcloud and explore two alternatives, namely, point rendering and surfel rendering, respectively. To render the test view, we selected the nearest scene and use `OSMesa` off-screen rendering assuming the Lambertian model and a single light source.

In Table 5, we compare two different rendering options for the densely reconstructed pointcloud. We discard the invisible pixels when computing the PSNR for both methods, making the quantitative results comparable to our Block-NeRF setting.

In Figure 8, we show the qualitative comparisons between two rendering options with PSNR on the corresponding images. This reconstruction is sparse and fails to represent reflective surfaces and the sky.

Method	PSNR* (`train`) ↑	PSNR* (`test`) ↑
COLMAP (point)	13.019	11.933
COLMAP (surfel)	13.291	12.343

Table 5. Quantitative results for COLMAP. We discard invisible pixels (e.g., sky pixels that COLMAP fails to reconstruct) when computing the PSNR.

https://docs.mesa3d.org/osmesa.html

F. Examples from our Datasets

In Figure 10, we show the camera images from our Mission Bay dataset. In Figure 11, we show both camera images and corresponding segmentation masks from our Alamo Square dataset.

G. Societal Impact

G.1. Methodological

Our method inherits the heavy compute footprint of NeRF models and we propose to apply them at an unprecedented scale. Our method also unlocks new use-cases for neural rendering, such as building detailed maps of the environment (mapping), which could cause more wide-spread use in favor of less computationally involved alternatives. Depending on the scale this work is being applied at, its compute demands can lead to or worsen environmental damage if the energy used for compute leads to increased carbon emissions. As mentioned in the paper, we foresee further work, such as caching methods, that could reduce the compute demands and thus mitigate the environmental damage.

G.2. Application

We apply our method to real city environments. During our own data collection efforts for this paper, we were careful to blur faces and sensitive information, such as license plates, and limited our driving to public roads. Future applications of this work might entail even larger data collection efforts, which raises further privacy concerns. While detailed imagery of public roads can already be found on services like Google Street View, our methodology could promote repeated and more regular scans of the environment. Several companies in the autonomous vehicle space are also known to perform regular area scans using their fleet of vehicles; however some might only utilize LiDAR scans which can be less sensitive than collecting camera imagery.

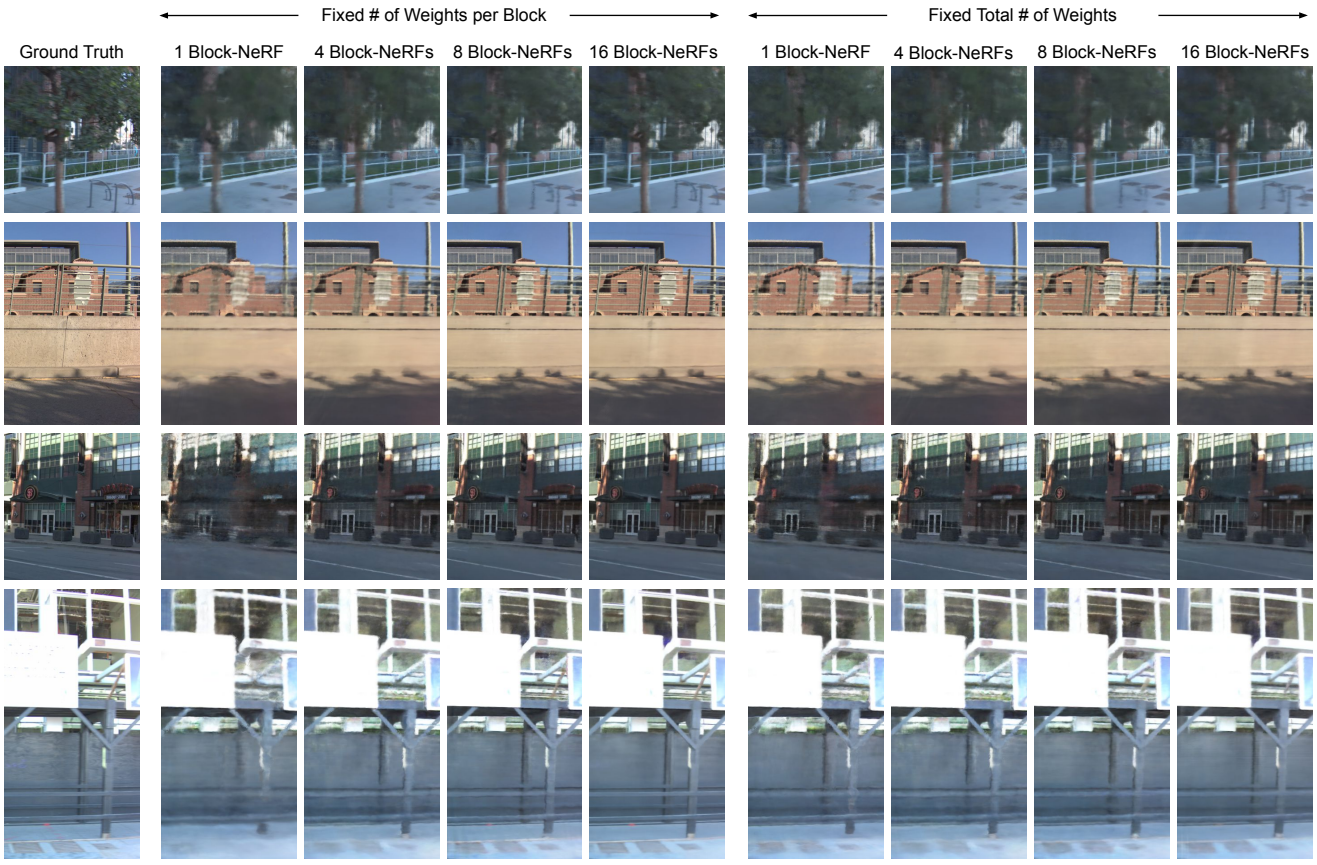

Figure 9. Qualitative results on Block-NeRF size and placement. We show results on the Mission Bay dataset using different options discussed in § 5.3 of the main paper.

Figure 10. Selection of images from our Mission Bay Dataset.

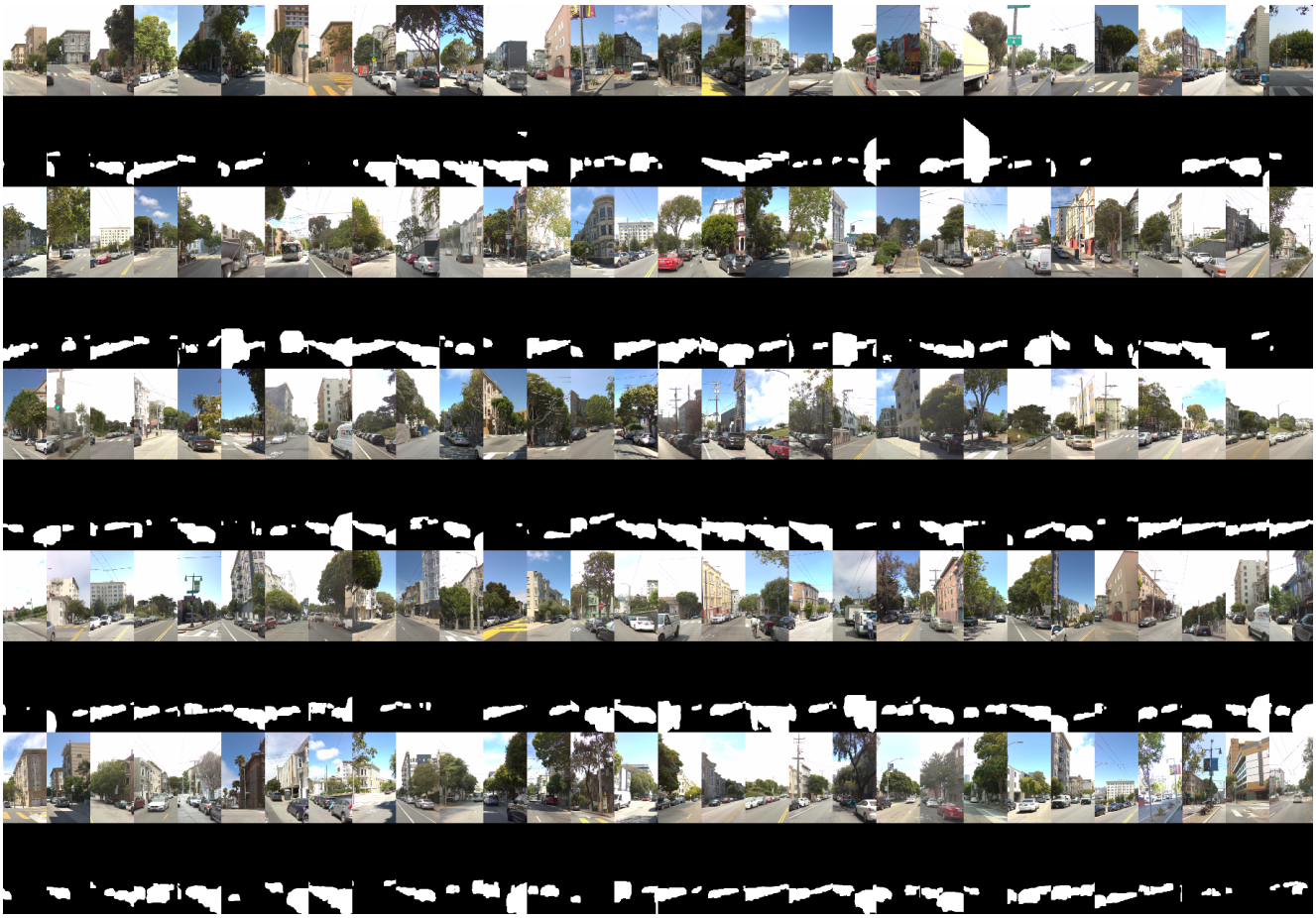

Figure 11. Selection of front-facing images from our Alamo Square Dataset, alongside their transient object mask predicted by a pretrained semantic segmentation model.