UE4-NeRF:Neural Radiant Field for Real-Time Rendering of Large-Scale Scene

Anonymous Author(s)

Abstract

Neural Radiance Field (NeRF) is an implicit 3D reconstruction method that has shown immense potential and has gained significant attention for its ability to reconstruct 3D scenes solely from a set of photographs. However, its real-time rendering capability, especially for interactive real-time rendering of large-scale scenes, has significant limitations. To address this challenge, we propose a novel neural rendering system called UE4-NeRF that is designed for real-time rendering of large-scale scenes. Our proposed approach partitions large scenes into sub-NeRFs, and uses polygonal meshes to represent them. In order to represent the partitioned independent scene, we initialize polygonal meshes by constructing multiple regular octahedra within the scene and the vertices of the polygonal faces are continuously optimized during the training process. Drawing inspiration from the Level of Detail (LOD) techniques, we train meshes with varying levels of detail for different observation levels. Our approach combines with the rasterization pipeline in Unreal Engine 4 (UE4), achieving real-time rendering of large-scale scenes at 4K resolution with a frame rate of up to 43 FPS. Our experimental results demonstrate that our method attains rendering quality on par with state-of-the-art approaches, while additionally offering the advantage of real-time performance. Furthermore, rendering within UE4 facilitates scene editing in subsequent stages. Project page: https://jamchaos.github.io/UE4-NeRF/.

1 Introduction

Real-time interactive rendering capability for 3D large-scale scenes is a crucial tool in creating digital worlds for applications such as virtual reality (VR), computer games, and the Metaverse. Neural Radiance Fields (NeRF) [23] is a pioneering method for 3D reconstruction and synthesis of novel views, which is different from traditional 3D reconstruction techniques that represent scenes using explicit expressions such as point clouds, grids, and voxels [5, 4, 14, 1]. NeRF samples each ray and retrieves the 3D location of each sampling point and the 2D viewing direction of the ray. Subsequently, these 5D vector values are fed into a neural network to determine the color and volume density of each sampling point. NeRF constructs a field parameterized by an MLP neural network [16, 34] to continuously optimize parameters and reconstruct the scene. NeRF has demonstrated remarkable performance for 3D reconstruction when provided with a set of posed camera images [29, 2, 23], but early NeRF works [2, 23] had only focused on small-scale scenes and object-centric reconstruction due to the model computational complexity. Although Block-NeRF [37] and BungeeNeRF [43] are capable of reconstructing large-scale environments, they are still unable to achieve real-time rendering. Traditional NeRF obtains the color of each pixel through volume rendering, but this rendering process is far too slow for interactive visualization and compromises high fidelity. Recently, Mobile-NeRF [6] combined with classic polygon rasterization pipeline has shown to achieve real-time rendering on various devices. However, training just a small-scale scene incurs significant overheads, such as training time and GPU resources. It requires 8 v100 GPUs to train the model successfully over several days. Additionally, due to opacity binarization, Mobile-NeRF cannot handle outdoor scenes with a variety of materials, including transparent or semi-transparent objects such as water,

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

(a) captured by UAV (b) real-time rendering of large-scale scene in UE4 (c) edition

Figure 1: (a) Large-scale scenes characterized by complex textures and significant height differences are captured with monocular or multi-camera drone footage. We divide scene into multiple individual sub-NeRFs.(b) Real-time interactive rendering in UE4 with distinct levels of detail, along with frame rate disparities among them. (c) Defferent editing capabilities on the scenes.

glass and plastic shed. Performing subsequent editing on the scenes generated by NeRF is another meaningful but challenging task, and previous works [13, 44, 19] required training complex neural networks to achieve this capability. These methods are time-consuming and are limited in their editing capabilities.

This paper introduces UE4-NeRF to address the aforementioned issues regarding large-scale scene rendering and editing. UE4-NeRF combines NeRF with the Unreal Engine 4 (UE4) [17, 30, 17] to achieve real-time interactive rendering of large-scale scenes with the scene editing ability. As shown in Figure 1(a) , to achieve this, large-scale scene such as industrial parks with large height differences, roads and farmlands are partitioned into separately trained blocks. The size of individual scenes captured by drones is in the range of several square kilometers. The multi-resolution hash encoding proposed in Instant-NGP [25] is employed for training acceleration. In order to efficiently utilize the parallel computing capabilities of GPUs, the computationally intensive portions are implemented using CUDA. Therefore, the training speed of UE4-NeRF is 1000$\times$ faster than Mobile-NeRF. For each block, we represent the scene using textured polygonal meshes obtained from regular tetrahedrons. Similar to Mobile-NeRF, the texture atlas stores feature vectors and opacity information. We utilize an MLP as the High-Level Shading Language(HLSL) fragment shader in UE4, which takes in the feature vectors and outputs colors. Furthermore, to ensure real-time rendering performance for different observation heights viewpoints, we adopted a method akin to the Level of Detail (LOD) [8, 21] approach commonly used in computer graphics, where the complexity of 3D model representation is quantified in terms of metrics such as geometric detail and texture resolution. Figure 1(b) demonstrates that UE4-NeRF is capable of achieving real-time rendering at different relative observation heights by training different levels of polygon meshes, where coarser meshes are used for relatively higher observation heights. One of the other advantages of combining NeRF with UE4 is the powerful development capabilities of the rendering engine, allowing further processing of the final scene rendered in UE4. Users can freely combine different sub-NeRFs and add objects to the scene as needed. Moreover, object manipulation, as illustrated in Figure 1(c) , is also supported.

In summary, our paper presents the following contributions:(1) We propose a novel UE4-NeRF technique which has 1000 times faster training times than MobileNeRF, while maintaining higher quality and smaller storage overhead;(2) The demonstration of the capability of UE4-NeRF to reconstruct large-scale scenes in real-time and showcase the practical viability of NeRF;(3) The integration of NeRF and UE4 facilitates editing and manipulation of 3D scenes.

2 Related work

Large Scale 3D Reconstruction. The original NeRF algorithm was limited to object-centric scenes or scenes with boundaries. NeRF++ [46] expanded upon NeRF by introducing an inverted sphere parameterization to handle unbounded scenarios. Mip-NeRF 360 [3] addressed the challenges associated with unbounded scenes through the use of non-linear scene parameterization, online

distillation and a novel distortion-based regularizer. However, it was observed that simply expanding the scene without adjusting the model capacity, i.e., using a fixed-size MLP, can lead to unstable training and loss of high-fidelity. Meanwhile, naively increasing the model capacity can significantly increase the training cost. BungeeNeRF [43] proposed a progressive approach to build and train large-scale scene models, which promoted layer specialization and unleashed the power of positional encoding over the full frequency range. Block-NeRF [37] and Mega-NeRF [40] spatially partitioned a large scene into several individually trained sub-NeRFs. They also incorporated appearance embeddings like NeRF-W [22] to handle appearance variations in the scene. Similar to Block-NeRF, our approach partitions the large-scale scene into sub-NeRFs, and trains a coarse scene model to assist in accurately delineating each sub-NeRF. *However, our approach is not only applicable to the grid structure like Block-NeRF, but is also generally suitable for various types of scenes, e.g. mountainous terrain. Additionally, in contrast to the aforementioned NeRFs designed for large-scale scenes, our method accomplishes real-time rendering.*

Rendering of Neural Fields. NeRF has tremendous potential for rendering and has gained attention for 3D reconstruction from a set of posed camera images [29, 2, 23]. However, the computational overhead of NeRF limits its practical applications. Some prior works [25, 11] have addressed training costs, while others have focused on rendering [9, 36, 27, 31]. SNeRG [20] and PlenOctrees [45] improved performance by baking the components of NeRF. However, these methods do not take sufficient advantage of the texture-based rasterization pipeline and require significant GPU memory to store volumetric textures. MobileNeRF [6] is a variation of NeRF that can run in real-time on a variety of common mobile devices. To render an image, Mobile-NeRF utilized the classic polygon rasterization pipeline with Z-buffering to produce a feature vector for each pixel and then passed it to a lightweight MLP running in a GLSL fragment shader to produce the output color. *In this work, we accelerate the rendering of NeRF by converting the learned knowledge into a mesh representation for efficient mesh rasterization pipelines. Moreover, we achieve a significant milestone by successfully implementing NeRF for real-time interactive rendering of large-scale scenes, which was previously unexplored.*

Manipulation and Composition of NeRF. Practical applications of 3D representations require effective manipulation and composition capabilities. While, explicit 3D representations, such as meshes and voxels, inherently support editing and composition, it becomes challenging for neural network-based implicit representations such as vanilla NeRF [23]. NSVF [20] uses a sparse voxel octree for modeling, employing voxel embeddings for each scene while sharing the same MLP for predicting density and color. However, the flexibility of this approach is somewhat constrained. GIRAFFE [28], combined with GAN [49, 18], enabled the addition of multiple objects in a scene, expanding from single-object to multi-object generation, even in the absence of such objects in the training data. Plenoxels [11] utlizes explicit sparse voxel representation for direct object composition of different objects, but suffers from large storage requirements of the dense index matrix. None of these methods support object manipulation in a 3D scene. *In contrast, our approach leverages UE4 to achieve large-scale scene rendering, enabling users to freely add and manipulate objects within the UE4-NeRF environment. For instance, users can use a broomstick to fly or drive a car within the rendered environment powered by UE4-NeRF.*

3 Proposed Method

Our novel method involves representing large scenes as polygon meshes, where the positions of mesh vertices are iteratively optimized during the training phase. The encoding network generates texture maps that store opacity and feature vectors, essential for rendering. To accomplish real-time rendering of large scenes, we combine the pre-rendered polygon meshes with the rasterization pipeline available in Unreal Engine 4 (UE4). In the subsequent subsections, we present a detailed and comprehensive description of entire procedure.

3.1 Training

3.1.1 Block Partition

As depicted in Figure 2, a large scene is partitioned into multiple blocks. To ensure smooth transitions at the boundaries of the model, each training area, indicated by dashed lines, is slightly larger than

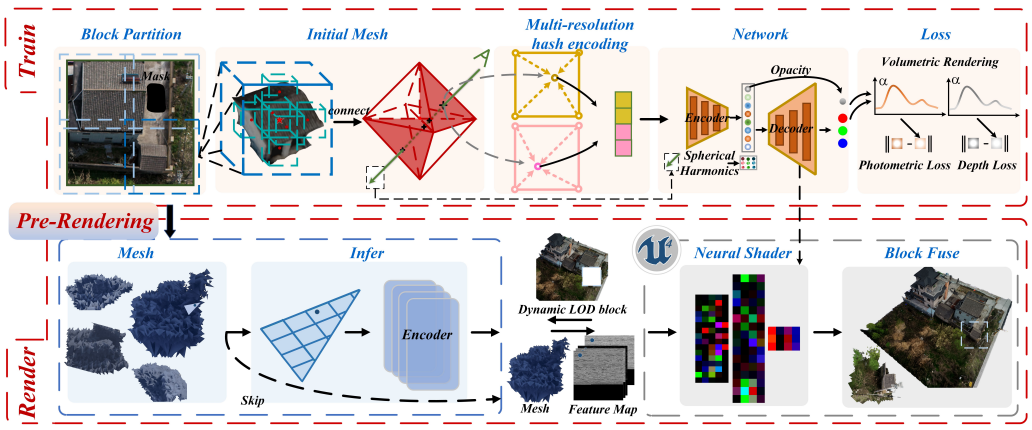

Figure 2: **Overview of UE4-NeRF.** UE4-NeRF consists of three modules: (1) In the training module, we partition the sub-NeRFs and initialize the grids in each partitioned scene. During the training process, we continuously optimize the parameters of the Encoder-Decoder network and the vertex positions of the meshes; (2) Through the pre-rendering module, we extract the polygonal meshes at different levels of detail that will be used for the final rendering; (3) The rendering module consists of an inference submodule and an UE4 submodule, which communicate with each other to achieve the final real-time rendering.

its corresponding scene area, with overlapping borders with neighboring blocks. We train a coarse model for sub-region segmentation, focusing on capturing the overall terrain and building outlines. We then filter the images associated with the sub-regions and use a mask to exclude pixels outside the scene area.

3.1.2 Optimization

We begin by initializing a 128×128×128 grid, and selecting the center point of each grid along with its six neighboring grids (front, back, left, right, top, and bottom) to create polygonal meshes. ***To address the slow and unstable convergence issue encountered by Mobile-NeRF when dealing with tilted surfaces, we utilize a regular octahedron with 20 faces.*** This is illustrated in the "Initial Mesh" section of Figure 2. The ray emitted from the camera origin to the pixel is defined as $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ and $\mathbf{d} = (\theta, \phi)$ is the viewing direction of the ray. For each ray emitted from the camera origin to the pixel, we calculate its intersection with the polygonal meshes. The point of intersection serves as the sampling point for that ray. For each block, we define an Encoder-Decoder network:

$$\mathcal{E}(hash(\mathbf{p_i}); \theta_{\mathcal{E}}) \rightarrow \mathbf{M_i}, \alpha_i \tag{1}$$

$$\mathcal{D}(\mathbf{M_i}, \mathcal{SH}(\mathbf{d_i}); \theta_{\mathcal{D}}) \rightarrow \mathbf{c_i} \tag{2}$$

The encoder network, denoted as MLP $\mathcal{E}$, takes positional information encoded by multi-resolution hash functions as input. It generates an 8D feature vector $\mathbf{M_i}$, which incorporates opacity information. On the other hand, the decoder network $\mathcal{D}$ utilizes $\mathbf{M_i}$ and the ray direction as inputs and predicts the color of the sampling point. The direction of the ray is encoded using Spherical Harmonics.

We use the traditional NeRF volume rendering method (Eq. 3) to obtain the predicted color $\hat{\mathbf{C}}(\mathbf{r})$. Due to the complexity of calculating the distance between sampling points, we simplify the process by fixing the distance value. The opacity α is then directly predicted instead of the volume density.

$$\hat{\mathbf{C}}(\mathbf{r}) = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c_i}, \quad T_i = \prod_{j=0}^{i-1} (1 - \alpha_j) \tag{3}$$

Unlike the traditional photometric loss, our proposed photometric loss consists of two components. The first component referred to as $level$-1 photometric loss is calculated using Eq. 4. It quantifies the mean squared error between the predicted colors and the ground truth colors of the corresponding pixel.

$$\mathcal{L}_{rgb}^{part1}(\theta, V_p) = \sum_{r \in \mathcal{R}} \left\| \hat{\mathbf{C}}(\mathbf{r}) - \mathbf{C}(\mathbf{r}) \right\|_2^2 \tag{4}$$

where V_p is the position of the vertices of the polygonal mesh under the corresponding level and $\mathcal{R}$ is the set of rays in each batch. $\mathbf{C}(\mathbf{r})$ represents the ground truth RGB colors for ray r. In addition, to improve the proximity of the triangle meshes to the object surface, we design a second component of the photometric loss, denoted as $\mathcal{L}_{rgb}^{part2}$. As depicted in Figure 3, in contrast to the previous sampling rule, we only select sampling points with opacity greater than a threshold f. Furthermore, the volumetric rendering process is halted when the accumulated opacity exceeds 0.8, indicating that the remaining opacity T' is less than 0.2. The threshold value f is computed using the formula provided in Eq. 5. Initiatlly, during the first 10,000 epochs, f is fixed at 0.3. Subsequently, f increases gradually with the progression of epochs.

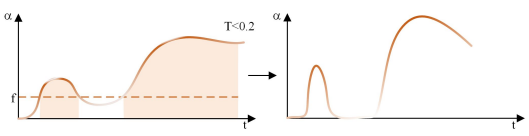

Figure 3: **Illustration of sampling details.** Our approach involves selectively sampling intersection points that are located above a threshold value f and terminating the volume rendering process when the remaining opacity drops below 0.2. This strategy effectively concentrates opacity on a limited number of specific meshes and compresses it towards the object, leading to improved rendering results.

$$f = \min\{0.3, \lfloor \frac{epoch - 10000}{10000} \rfloor^2 \times 0.012\} \tag{5}$$

The photometric loss for the second part of the $level$-1 can be obtained using the following formula:

$$\mathcal{L}_{rgb}^{part2}(\theta, V_p) = \sum_{r \in \mathcal{R}} \left\| \hat{\mathbf{C}}'(\mathbf{r}) - \mathbf{C}(\mathbf{r}) \right\|_2^2, \quad \hat{\mathbf{C}}'(\mathbf{r}) = \frac{\sum_{\phi=1}^{N} T_\phi \alpha_\phi \mathbf{c}_\phi}{1 - T'_\phi} \tag{6}$$

Where $\hat{\mathbf{C}}'(\mathbf{r})$ is obtained through volume rendering under the aforementioned sampling scheme. To cover the entire color range, we divide the expression by $1 - T'_\phi$ as shown in Eq. 6, since our volume rendering stops when the remaining opacity is less than 0.2. We obtain the final photometric loss of the $level$-1 as follows:

$$\beta \mathcal{L}_{rgb}^{part1}(\theta, V_p) + (1 - \beta)\mathcal{L}_{rgb}^{part2}(\theta, V_p), \quad \beta = 0.8^{\lfloor \frac{epoch}{10000} \rfloor} \tag{7}$$

During the initial 10,000 epochs, the second component of the photometric loss does not have any impact. This is done to align with the traditional volume rendering approach and adhere to the modeling principles of NeRF. ***However, as the training progresses, the weight for the second component gradually increases. This enables the preservation of opacity only on specific meshes and ensures consistency with the later stages, where only meshes with opacity greater than 0.3 are exported for rasterization.***

After 10000 epochs, a process is initiated where we randomly select sampling points within each grid, calculate the opacity of each grid, and select the vertex of the grid with the highest opacity among the eight grids as the vertex of the merged grid. This allows us to generate grids of 64×64×64, 32×32×32, 16×16×16, and 8×8×8 sequentially. Each grid level is tailored for specific observation heights, enabling us to minimize the rendering cost of the scene. It is important to note that for the same block, different levels always use the same network. The final loss function of the photometric part is calculated as follows:

$$\mathcal{L}_{rgb} = \sum_{l=1}^{5} \beta \mathcal{L}_{rgb}^{part1}(\theta, V_p) + (1 - \beta)\mathcal{L}_{rgb}^{part2}(\theta, V_p) \tag{8}$$

Pseudo-depth. Utilizing depth information [10, 33] as a supervisory signal has demonstrated the ability to enhance the convergence rate and improve the quality of reconstructions. However, when operating under the influence of ambient light in outdoor environments, RGB-D cameras are susceptible to produce inaccurate depth measurements and have limited measurement ranges. In our approach, we propose to use the pseudo-depth as the supervision information for NeRF. In conventional NeRF, structure-from-motion (SFM) [35] solvers such as COLMAP [15, 39, 12] are typically utilized to estimate the camera poses. As part of this process, a sparse point cloud can be obtained. By aligning the camera and point cloud in the same coordinate system, we calculate the pseudo-depth by measuring the distance between the sparse point cloud and the camera position. For

a detailed explanation of the pseudo-depth loss ($\mathcal{L}_D$) calculation, please refer to the supplementary material. The final loss $\mathcal{L}$ of the model is calculated as follows:

$$\mathcal{L} = \mathcal{L}_{rgb} + \mathcal{L}_D \tag{9}$$

Transient Objects. Moving/dynamic objects, such as pedestrians and cars, have the potential to cause inconsistency in the observed scene across different viewpoints and can also create gaps or "holes" in the depth map. To address this challenge, we integrate a semantic segmentation model [7, 48] into our training process. By leveraging this model, we generate masks that identify the regions occupied by the moving objects, effectively excluding them from the training of the scene. This strategy mitigates the impact of moving objects on the scene's depth map, and thus improves the overall fidelity and generalization performance of the model.

3.2 Pre-rendering

To optimize computational efficiency, we leverage the predicted values from the acceleration grid to identify and remove meshes that do not possess significant geometric surfaces. Once the training phase is complete, we obtain the network weights as well as the meshes with optimized vertex positions. Subsequently, we perform the pre-rendering process. The overall workflow is illustrated in Figure 4. ***First***, for each block, in addition to the camera view used for training, we incorporate parallel lights from various angles are added above. We compute the intersection of each ray with the polygon mesh, and the traversal of light is halted when the cumulative opacity surpasses 0.8. We filter out meshes that have an intersection opacity less 0.3, resulting in the retention of only those meshes that meet this criterion. From this, we obtain the final polygon mesh representations of the scene. Based on our statistical analysis, the final extracted meshes account for only 5% of the original mesh count. This significant reduction effectively improves mesh utilization, inference speed during rendering, and reduces storage costs. ***Secend***, we perform coordinate transformation to ensure the obtained mesh aligns with real-world scale. ***Third***, we map the vertex coordinates of the triangles to their corresponding positions in UV coordinates.

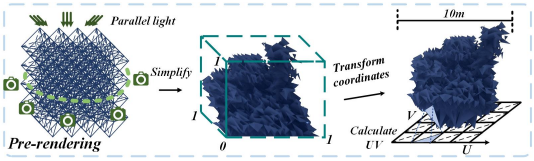

Figure 4: **Structure of the pre-rendering.**We employ a multi-viewpoint approach to extract polygonal faces, generating multiple rays from different perspectives. Subsequently, we apply scaling transformations and calculate UV coordinates to further process the extracted data.

3.3 Rendering

After training, we obtain the weights of the Encoder-Decoder network, and refined polygonal meshes at different levels through vertex optimization for each indivicual block. Subsequently, we perform pre-rendering and extract the final meshes. For each polygonal mesh, we perform rasterization using a texture map of size 32×32. However, storing texture maps for numerous polygonal meshes in a large-scale scene can lead to significant storage overhead. To address this, we utilize dynamic inference instead of directly storing all texture maps, as done in Mobile-NeRF [6]. Real-time rendering is achieved through continuous and efficient communication between the inference submodule and the UE4 submodule. During the block composition process, since the large-scale scene is accurately divided into multiple blocks and scene fusion is also considered during training, transitions between different blocks in a large-scale scene are smooth and interpolation is not required at the junction.

Texture map. Prior to rasterizing the faces of the polygonal mesh, we perform sampling on each mesh and use the encoder network to obtain 8 channels feature vectors. These 8 channels along with the opacity α obtained are processed to generate nine single-channel texture maps using BC4 compression [26]. Compared to the original texture map, this reduces 2/3 of the GPU memory consumption during rendering without significantly compromising the image quality. *Please note that texture maps do not store colors, but rather spatial features, which also contain transparency information.* Taking into account the principle of locality, we employ a strategy where we collectively infer and store information from nearby regions into texture maps. This enables us to directly access the texture maps without the need for re-inference when traversing these regions again.

Table 1: **Details of the real-world scene captured by UAV.** The memory requirement increases with the size of the scene, while the variation in FPS is primarily influenced by the materials present in the scene itself.

	Memory(4k) Host	Memory(4k) GPU	Area(m^2)	FPS 2k	FPS 4k	CA
FL	$\sim$ 12GB	$\sim$ 5GB	180×240	55	36	120m
VL	$\sim$ 16GB	$\sim$ 7GB	180×240	46	30	70m
CS	$\sim$ 18GB	$\sim$ 8GB	240×240	66	43	70m
IP	$\sim$ 40GB	$\sim$ 14GB	600×600	58	35	120m

Table 2: **Quantitative Comparisons with Existing Methods.** Our method achieves remarkable results in terms of both rendering quality and speed. Among the compared methods, UE4-NeRF requires the least GPU memory. The training costs are also provided.

Method	PSNR↑	SSIM↑	LPIPS↓	FPS(2K)	Memory(GB) Host	Memory(GB) GPU	time(mins)
NeRFacto [38]	20.99	0.663	0.389	0.5	$\sim$ 14	$\sim$ 3	$\sim$ 45
Mip-NeRF [2]	16.99	0.516	0.458	✘	$\sim$ 19	$\sim$ 6	$\sim$ 120
Urban-NeRF [32]	20.42	0.563	0.409	✘	-	-	-
NeRF-w [22]	17.16	0.455	0.620	✘	-	-	-
Instant-NGP [25]	22.00	0.631	0.426	11	$\sim$ 10	$\sim$ 32	$\sim$ 15
UE4-NeRF	**25.03**	**0.704**	**0.287**	**55**	$\sim$ 19	$\sim$ **3**	$\sim$ 40

Rasterization. We utilize the traditional rasterization pipeline in UE4 to rasterize each face of the polygonal mesh. Subsequently, the decoder network is applied to convert the 17D features of each pixel into RGB colors (see UE4 submodule in Figure 2). The 17D features comprise 8 channels of learned features and a 9D view direction, which is encoded using spherical harmonic functions [11, 24]. The decoder network is implemented as a HLSL fragment shader of the UE4, which is called Neural Shader. To achieve translucent effects without the need for volume rendering, we employ alpha-dithered [42], which offers a relatively low-overhead approach.

Dynamic inference. The interaction between the UE4 submodule and the inference submodule is depicated in Figure 2. When a block requiring rendering is encountered, the UE4 submodule supplies the block and level index information, while the inference submodule provides the corresponding regional meshes and feature maps. In cases where feature maps are not readibly for a specific region, the inference operation is carried out to generate them. This communication mechanism ensures efficient and prompt processing between these two submodules.

4 Experiments

4.1 Implementation Details

Similar to the Instant-NGP approach, we employ multi-resolution hash encoding and construct an Encoder network comprising a 4-layer MLP (32×64,64×64,64×64,64×8) to estimate the opacity and an 8-dimensional feature vector. Additionally, we develop a Decoder network with 3-layer MLP (17×16,16×16,16×3) to predict the final color of the sampled points. For each block, we train the model using one Nvidia RTX 3090 GPU for 80,000 epochs to achieve convergence, with an approximate duration of 40 minutes.

Datasets and Metric. Our objective is to enable real-time rendering of large-scale scenes with complex textures and materials. Table 1 presents four distinct scenarios, namely Farmland(FL), Village(VL), Construction site(CS) and Industrial parks(IP) which were captured using drones to address the challenges that UE4-NeRF was designed to tackle. Each captured image has a resolution of 5472×3648 pixels. At $level$-1, UE4-NeRF can achieve an maximum frame rate of around 43 FPS at 4K resolution. Due to the limited battery life of the drone, the captured areas were constrained in size. Nevertheless, subsequent experiments have revealed that limitation does not represent the upper limit of UE4-NeRF. In fact, UE4-NeRF can render even larger-scale scenes in real-time. To evaluate the reconstruction quality and fidelity of our model, we use evaluation metrics such as PSNR/SSIM [41] and LPIPS [47]. We also compare the real-time rendering speed of UE4-NeRF with other models when rendering vast scenes.

4.2 Comparisons with Existing Methods

We performed qualitative comparisons between UE4-NeRF and other NeRF implementations on four scenes captured by UAV, and the results are depicted in Figure 5. Existing tools like NeRFacto [38] and Instant-NGP [25] aim to achieve real-time rendering, but tend to procude subpar results when applied to large-scale scenes, as evident in Figure 5. Mip-NeRF, despite increasing the number of training iterations, still exhibits noticeable blurriness when rendering natural scenes. In contrast, UE4-NeRF showcases remarkable accuracy in rendering small objects, such as power lines, in the

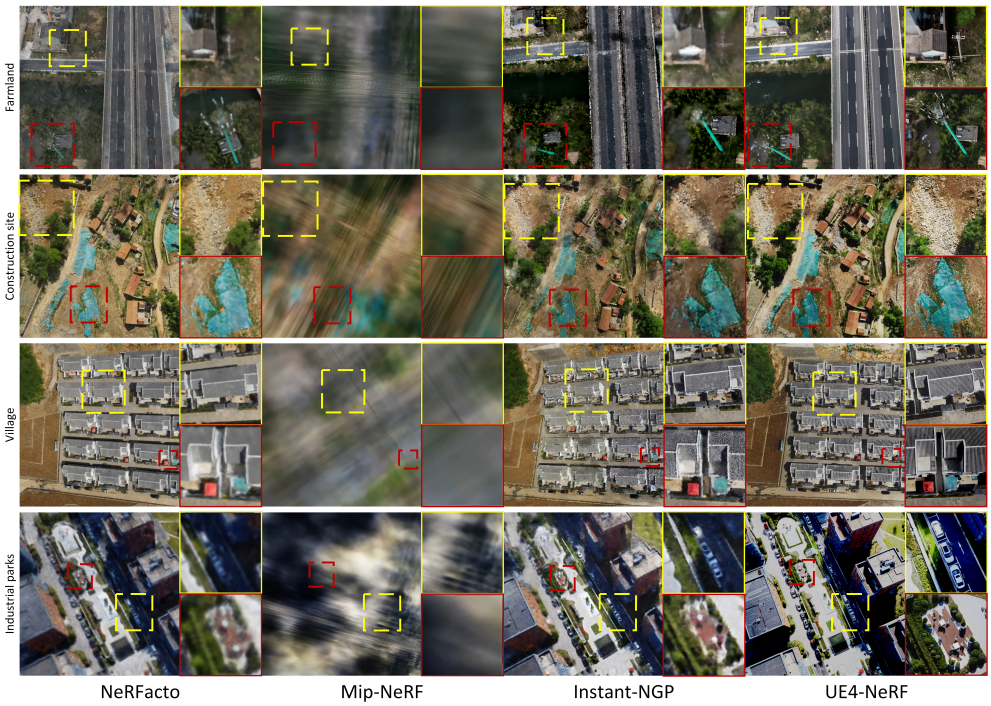

Figure 5: **Qualitative Comparisons with Existing Methods.** We visually compare our proposed technique with recently implemented real-time rendering NeRF models [25, 38] and Mip-NeRF [3]. All these methods are implemented within nerfstudio [38]. It is evident that UE4-NeRF yields the best reconstruction for all the four scenes.

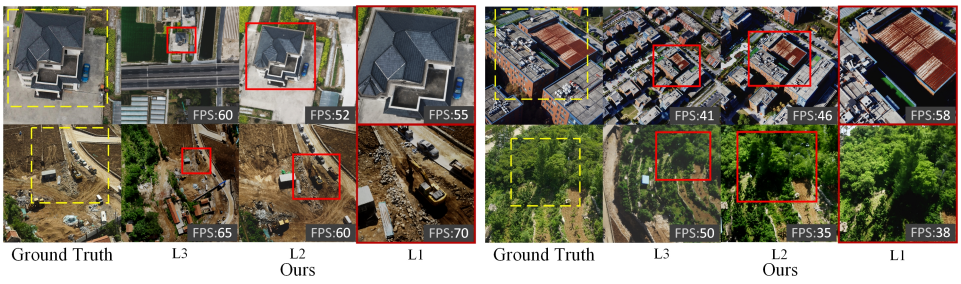

Figure 6: **Rendering result at different LOD.** The red boxes indicate the presentation of the same content at different levels. In addition to achieving high FPS rendering at different LOD in 2K resolution, UE4-NeRF also demonstrates a remarkable photo-realistic.

"farmland" scene. It also excels in handling dense textures, as demonstrated in the "village" scene. The "construction site" scene showcases the rendering effects of translucent objects. In Table 2, we provide a quantitative comparison of our proposed technique with other existing methods using different metrics. It is important to note that the actual rendering quality of UE4-NeRF is expected to surpass the metric's performance, as achieving consistent exposure matching with the original image is challenging within the Unreal Engine environment. Despite this limitation, UE4-NeRF still outperforms other methods by a significant margin and its real-time rendering capability surpasses that of the current state-of-the-art real-time rendering models.

4.3 LOD render results

Figure 6 showcases the qualitative rendering results at different levels of detail, along with the corresponding real-time frame rates, at a resolution of 2K. As the level of hierarchy decreases, the

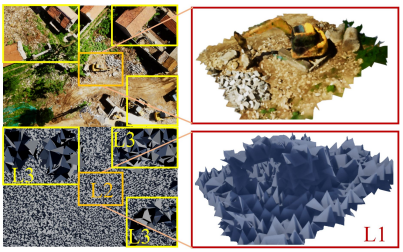

Figure 7: **Shading mesh**– not textured. Polygonal meshes at different levels are displayed, revealing the varying complexity of the scene.

Figure 8: **Scene edition.**The results of multi-view scene editing are presented. One can carefully observe the occlusion relationship between objects and the scene.

rendering quality and clarity progressively improve. UE4-NeRF demonstrates a high level of fidelity when compared to the ground truth. In addition, it can be observed that the LOD approach effectively addresses the challenge of real-time rendering at higher levels although it may sacrifice some rendering quality at higher levels for the sake of real-time performance. When comparing materials and frame rates across different scenes, it is worth noting that the presence of semi-transparent objects, such as vegetation(as seen in the last scene of Figure 6), can potentially lead to a decrease in frame rate due to increased computational overhead.

Shading mesh. Figure 7 showcases the shading meshes extracted without textures. Despite the coarse appearance of the meshes and the limited level of detail even in the closest $level$-1, such as excavators, the final rendering quality remains impressive. This can be attributed to the vertex optimization carried out during the training process, the texture maps encoded in the network outputs, and the deployment of the final neural shader during rendering.

4.4 Scene edition

Editing scenes in UE4-NeRF, especially the composition of different scenes, is a seamless process. UE4 supports multiple 3D model file formats, allowing us to import and edit the rendered scenes. The Unreal Engine automatically handles occlusion between the original scene and added objects, making the editing process effortless. We have complete freedom to manipulate the newly added objects according to our needs. In Figure 8, when an object obstructs the NeRF-rendered scene, the occluded parts do not require shading by the neural shader. Interestingly, we observed that after adding and manipulating objects, the frame rate of the scene actually slightly improves. This is attributed to the reduction in NeRF computation, resulting in improved frame rates.

5 Conclusion and Limitations

In this research paper, we introduce UE4-NeRF, a real-time interactive rendering system designed for large-scale scenes. UE4-NeRF partitions the scene into blocks and trains a separate NeRF model for each block. Inspired by LOD techniques, different levels of mesh precision are used for different layers, and the NeRF model is integrated with the rasterization pipeline in UE4. Currently, UE4-NeRF stands as the only NeRF model capable of achieving real-time interactive rendering of large-scale realistic scenes while maintaining high fidelity. Moreover, its integration with UE4 opens up possibilities of scene editing and manipulation, allowing for the flexible addition and manipulation of traditional 3D models such as $.obj$ and $.fbx$ within the scene. Overall, UE4-NeRF presents a comprehensive neural rendering system that combines large-scale, real-time interactive rendering with photo-realism and editability. ***However***, there are still some issues that we aim to address in the future: 1) UE4-NeRF requires the direct use of CUDA and therefore relies on NVIDIA GPUs. Rendering large scenes incurs significant memory overhead, and increasing rendering accuracy further amplifies memory consumption. 2) During the pre-rendering process, it is difficult to capture rays from any viewing angles and extract meshes for each viewpoint, resulting in gaps or holes in the final rendered scene. The strategy of pre-rendering can be optimized.

References

[1] John Ashburner and Karl J Friston. Voxel-based morphometry—the methods. *Neuroimage*, 11(6):805–821, 2000.

[2] 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. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

[3] 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.

[4] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer graphics forum*, volume 36, pages 301–329. Wiley Online Library, 2017.

[5] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. CRC press, 2010.

[6] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv preprint arXiv:2208.00277*, 2022.

[7] 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. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12475–12485, 2020.

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

[9] Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. High-quality streamable free-viewpoint video. *ACM Transactions on Graphics (ToG)*, 34(4):1–13, 2015.

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

[11] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022.

[12] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis (pmvs). In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, 2007.

[13] Michelle Guo, Alireza Fathi, Jiajun Wu, and Thomas Funkhouser. Object-centric neural scene rendering. *arXiv preprint arXiv:2012.08503*, 2020.

[14] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(12): 4338–4364, 2020.

[15] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[16] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[17] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4(3):1, 2013.

[18] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[19] Verica Lazova, Vladimir Guzov, Kyle Olszewski, Sergey Tulyakov, and Gerard Pons-Moll. Control-nerf: Editable feature volumes for scene rendering and manipulation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4340–4350, 2023.

[20] 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.

[21] David Luebke, Martin Reddy, Jonathan D Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of detail for 3D graphics*. Morgan Kaufmann, 2003.

[22] 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.

[23] 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. *Communications of the ACM*, 65 (1):99–106, 2021.

[24] Claus Müller. *Spherical harmonics*, volume 17. Springer, 2006.

[25] 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.

[26] Jae-Ho Nah. Quicketc2: Fast etc2 texture compression using luma differences. *ACM Transactions on Graphics (TOG)*, 39(6):1–10, 2020.

[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*, volume 40, pages 45–59. Wiley Online Library, 2021.

[28] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11453–11464, 2021.

[29] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021.

[30] Weichao Qiu and Alan Yuille. Unrealcv: Connecting computer vision to unreal engine. In *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*, pages 909–916. Springer, 2016.

[31] 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.

[32] Konstantinos Rematas, Andrew Liu, Pratul P Srinivasan, Jonathan T Barron, Andrea Tagliasacchi, Thomas Funkhouser, and Vittorio Ferrari. Urban radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12932–12942, 2022.

[33] Antoni Rosinol, John J Leonard, and Luca Carlone. Nerf-slam: Real-time dense monocular slam with neural radiance fields. *arXiv preprint arXiv:2210.13641*, 2022.

[34] Franco Scarselli and Ah Chung Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks*, 11(1):15–37, 1998.

[35] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016.

[36] Vincent Sitzmann, Semon Rezchikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *Advances in Neural Information Processing Systems*, 34:19313–19325, 2021.

[37] 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.

[38] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, et al. Nerfstudio: A modular framework for neural radiance field development. *arXiv preprint arXiv:2302.04264*, 2023.

[39] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*, pages 298–372. Springer, 2000.

[40] 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.

[41] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[42] Peter L Williams, Randall J Frank, and Eric C LaMar. Alpha dithering to correct low-opacity 8 bit compositing errors. *Lawrence Livermore National Laboratory Technical Report UCRL-ID-153185*, 2003.

[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 *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*, pages 106–122. Springer, 2022.

[44] 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. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13779–13788, 2021.

[45] 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.

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

[47] 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.

[48] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[49] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5 Limitation.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We discuss it in supplementary materials.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.1.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes] See supplementary materials.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]