A General Implicit Framework for Fast NeRF Composition and Rendering

Xinyu Gao[1], Ziyi Yang[1], Yunlu Zhao[1], Yuxiang Sun[2], Xiaogang Jin[1*], Changqing Zou[1,2]

[1]State Key Lab of CAD&CG, Zhejiang University, [2]Zhejiang Lab
{22121052, Ingram14, yunlu.zhao}@zju.edu.cn, sunyuxiangyx@gmail.com
jin@cad.zju.edu.cn, changqing.zou@zju.edu.cn

Abstract

A variety of Neural Radiance Fields (NeRF) methods have recently achieved remarkable success in high render speed. However, current accelerating methods are specialized and incompatible with various implicit methods, preventing real-time composition over various types of NeRF works. Because NeRF relies on sampling along rays, it is possible to provide general guidance for acceleration. To that end, we propose a general implicit pipeline for composing NeRF objects quickly. Our method enables the casting of dynamic shadows within or between objects using analytical light sources while allowing multiple NeRF objects to be seamlessly placed and rendered together with any arbitrary rigid transformations. Mainly, our work introduces a new surface representation known as Neural Depth Fields (NeDF) that quickly determines the spatial relationship between objects by allowing direct intersection computation between rays and implicit surfaces. It leverages an intersection neural network to query NeRF for acceleration instead of depending on an explicit spatial structure.Our proposed method is the first to enable both the progressive and interactive composition of NeRF objects. Additionally, it also serves as a previewing plugin for a range of existing NeRF works.

Introduction

The development of virtual worlds has revealed a strong preference for neural techniques. Neural Radiance Fields (NeRF), an image-based rendering method (**?**), has shown its proficiency in rendering with high fidelity while also offering the added advantage of compression capabilities, resulting in a low storage footprint. As a result, NeRF has been extensively studied in human face editing (**??**), autonomous driving (**??**), stylization (**?**), mesh reconstruction (**??**), inverse rendering (**???**), and so on.

Although a number of NeRF composition methods (**?????**) have achieved impressive results and improved the flexibility by modeling a scene as multiple semantic parts rather than as whole, compositing neural radiance fields in real-time still remains under-explored. Existing acceleration methods, such as implicit methods (**??**) that rely on a limited scope of camera activity, hybrid methods (**??**) that require a complex spatial structure for querying, or pre-computing methods (**??**) that incur significant storage costs, are far from straightforward in this task due to various challenges, let alone forming a general pipeline. First, it is difficult to satisfy low memory-storage cost and real-time speed while maintaining arbitrary affine transformations for implicit objects. Second, as the number of objects to be composed grows (e.g., a simple duplicate of 1k balls), rendering the entire scene (especially considering shadows among implicit objects) takes long, and the cost of hybrid or pre-computing methods becomes prohibitively expensive.

To address the aforementioned issues, we present a general implicit framework that can quickly perform compositing and shadow casting of NeRF objects without querying any complex structure, thereby avoiding the excessive memory-storage costs. Meanwhile, we propose Neural Depth Fields (NeDF), a novel implicit representation for implicit surface that aims for fast and precise depth estimation under a flexible camera activity scope and arbitrary affine transformations of objects, overcoming critical issues associated with previous implicit acceleration methods while retaining the benefits of implicit methods with a low storage cost. When working with a single object in a composed scene, manipulation speeds can even reach real-time in our CUDA implementation. Notably, we use the classic concept of shadow mapping based on NeDF for efficient rendering of dynamic shadows cast by either point light or parallel light sources, which represents a significant advancement over previous NeRF works aimed at acceleration.

Our work is able to provide users a fast NeRF scene composition tool allowing a quick preview of the composited results, as users may prefer interactive-level speed over perfect quality when manipulating each NeRF object. The key technical contributions are summarized below:

- A novel framework that enables interactive previewing of the NeRF composition process at a low storage cost, thereby providing a new interactive way to create novel realistic scenes from real-world scenes.
- A quick and efficient method for generating intra and inter-object dynamic shadows cast by either point light or parallel light sources.
- A new geometry representation for NeRF objects that aids intersecting/depth estimation during arbitrary ma-

*Corresponding author.

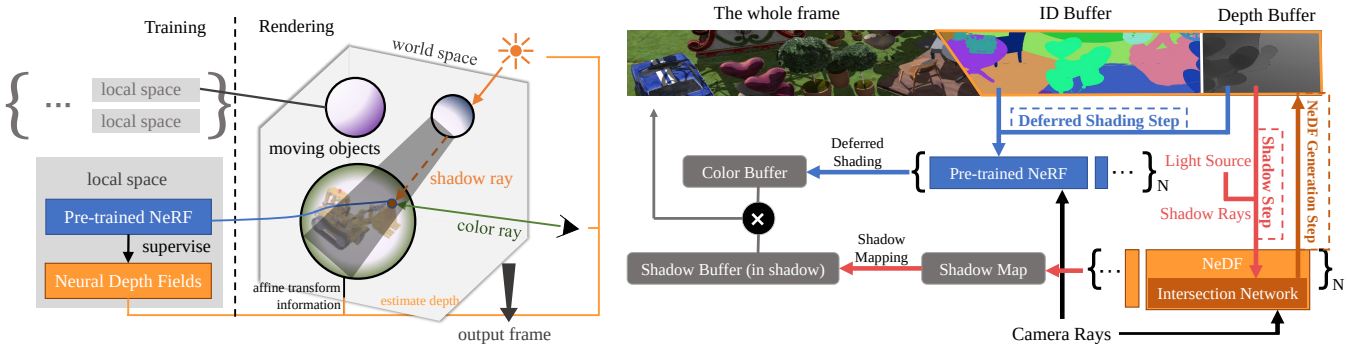


Figure 1: An overview of our framework (left) and data flow (right). In terms of framework, we use Neural Depth Fields (NeDF) as the representation of the implicit surface within each object's local space during training, under the supervision of pre-trained NeRF. There are three main steps in render time: 1) Step1 is "NeDFs generation step", depth buffer and ID buffer are created by a collection of NeDFs; 2) Step2 is "Deferred shading step", those 2 buffers are then used by a collection of NeRFs to fill a color buffer with no shadows; 3) Step3 is "Shadow step", shadow rays are generated based on the depth buffer and NeDFs are used again to provide a shadow map. Finally, the color buffer and shadow buffer are multiplied to produce the results.

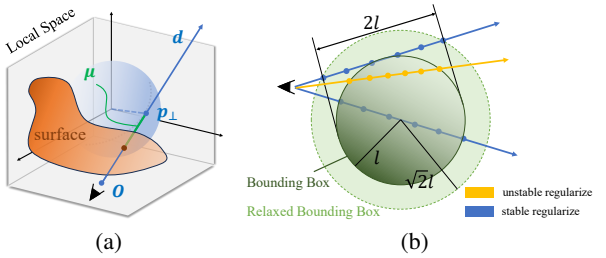
nipulation.

Related Work

Compositing Neural Radiance Fields. Modeling a scene as a single NeRF remains generally limited in functionality. In many scenarios, the ability to isolate individual objects and manipulate them under controlled constraints is necessary (e.g., driving scenes (**?**)). In order to enhance the controllability and editability of the NeRF-based virtual scenes, one straightforward idea is to model scenes as multiple semantic parts rather than a whole at the beginning. This can be achieved by separately compressing background and foreground objects into two branches represented by neural voxels (**?**) or a set of neural SDFs (**?**). In order to enable a better perception for foreground objects, clues such as 3D motion (**?**) or segmentation mask (**??**) in 2D image space may be required as a preprocessing step. Another idea is compositing pre-trained NeRF models together to form a more complex and controllable scene. To achieve this, explicit spatial structures like multi-granularity voxels (**?**) and 3D bounding box (**??**) are employed to indicate the sampling range for each nerf model in global space. By manipulating these voxels/boxes, NeRF objects are transformed from their local space coordinates to the global world space. Additionally, techniques that pre-compute radiance in explicit structures have shown promise for scene compositing. Despite the benefits of utilizing multiple neural fields, doing so may result in significant computational requirements, leading to notable storage overhead in the pre-compute methods or reduced rendering speeds in implicit methods, which can negatively affect the ease with which users can manipulate objects in an interactive preview and loading speed. This limitation contradicts the goal of enabling real-time scene editing and rendering. Unlike these methods, we attempt to composite NeRF objects interactively and progressively for a quick preview.

Accelerating Rendering of NeRF-modeled Scenes Real-time rendering has been a long-standing goal in the field of computer graphics, and this objective extends to the realm of NeRF. The explicit (pre-computed) methods achieved real-time by pre-computing radiance color as spherical Harmonics coefficients (**?**) or feature vectors (**??**), and then storing them in spatial structures like octree for quick inference, thereby reducing the computational overhead at render time. Among these explicit methods, the state-of-the-art method (**?**) converts NeRF scenes into a combination of meshes and spherical Gaussians, resulting in fast rendering speed on modern mobile graphics pipelines. However, the explicit method can result in high storage costs for even a single scene. In contrast, implicit (MLP-based) methods that mainly focus on importance sampling (**????**) significantly reduce storage overhead and computational burden. But these methods can limit the activity scope of the camera to a specific view cell. The hybrid method (**?**) incorporates a neural component within the explicit grid to achieve a cost-effective solution. All of these methods are highly specialized, designed to address their own accelerating path. Our goal, on the other hand, is to investigate a general pipeline that enables rapid composition across various NeRF works.

Neural Light Fields (NeLF) The representation of Light Fields has been widely utilized in computer graphics to model a scene as a 4D representation, enabling fast image-based rendering. The emergence of deep learning in recent years has led to the investigation of Neural Light Fields (NeLF) as an extension of this concept (**?????**). Recently, R2L (**??**) successfully obtained NeLF with a comparable quality to NeRF, which provides a straightforward insight into the capability of light fields' representation to generate high-quality results under 360° cases. In comparison to NeRF, NeLF has the distinct advantage of requiring only a single evaluation per ray, thereby eliminating the need for time-consuming ray marching. However, the composition task is unsuitable for NeLF due to its lack of precise depth information, though several past approaches (**???**) have tried to leverage light fields to extract geometry information for depth estimation. Nevertheless, NeLF has inspired us to ef-



(a) (b)

Figure 2: (a) Given a ray traveling through an object's local space, the first intersection on the surface can be obtained by the distance between the intersection and the tangency point $\mathbf{p}_\perp$ (The sphere is in the center of local space). Since $\mathbf{p}_\perp$ can be quickly and uniquely determined, this representation leads to a direct intersection computation. (b) To improve the stability of NeDF, we regularize the main body of each ray, approximated by a tuple of points sampled along the ray, within the relaxed bounding box before fedding it into the intersection network.

ficiently composite NeRF in a geometry-aware manner.

Methodology

Given a set of objects represented by NeRF, our approach provides a quick preview of the resulting scene in which all of the NeRF objects are manipulated with arbitrary affine transformations and rendered together. The overview and data flow of our render framework is shown in Fig. 1, which is also described by the algorithm pseudo-codes in appendix.

Neural Depth Fields (NeDF)

Neural Depth Field is defined on a 5-dimensional continuous function $\mathcal{F}_\Phi$ that represents a scene. This function maps an arbitrary ray (represented by a starting point and a direction) to the distance/depth between the starting point of the ray $\mathbf{r}$ to the ray-object intersection point in a 3D virtual world space as shown in Fig. 2a. Assuming that each object used for scene composition is opaque, this output distance can enable the computation of the surface point color of composed NeRF, as well as the relative spatial occlusion relationship of each composed object, which are essential for scene composition and rendering.

The continuous function $\mathcal{F}_\Phi$ is implemented with a Multilayer Perceptron network, termed as *intersection network*. $\mathcal{F}_\Phi$ does not directly infer the depth between the starting point of the ray $\mathbf{r}$ and the ray-object intersection point. It instead infers the distance between the ray-object intersection point and the tangency point at which the ray $\mathbf{r}$ is tangent to a sphere centered at the origin in the local space of the object (i.e., $\mathcal{F}_\Phi$ is used to compute μ, and μ is the distance between the tangency point $\mathbf{p}_\perp$ of the ray $\mathbf{r}$ to the ray-object intersection point, as shown in Fig. 2a). We choose to learn μ rather than $D(\mathbf{r})$ is because it's impossible to learn an infinite space. μ is defined in the local space and has a limited range of values.

Specifically, for a given ray $\mathbf{r}$, we mathematically define $\mathcal{F}_\Phi$ as:

$$\mathcal{F}_\Phi : \mathbf{r} \mapsto (\mu, \hat{\alpha}), \tag{1}$$

where $\hat{\alpha}$ is a binary indicator that signifies whether the ray intersects the object's surface.

Once the μ is predicted by the intersection network, the depth $D(\mathbf{r})$ between the starting point of the ray $\mathbf{r}$ and the ray-object intersection point can be calculated as:

$$D(\mathbf{r}) = |\mathbf{o} - \mathbf{p}_\perp| - \mu, \tag{2}$$

where $\mathbf{o}$ is the starting point of the ray and $\mathbf{p}_\perp$ is the tangency point. We adopt a set of points sampled along rays to represent an oriented ray to improve the stable performance of the intersection network. Specifically, we regularize the sampling within a specific local range along rays and sample 16 points in total (see Fig. 2b).

Intersection network

Directly implementing the intersection network as a regression problem of the distance μ is not a good choice and tends to produce an inaccurate estimation because of the depth discontinuity issue (**?**) during composition, as illustrated in Fig. 3(a). This inaccurate estimation around edges can lead to floating artifacts during rendering as shown in Fig. 3(c).

To address this problem, we instead build the intersection network as a multi-level classifier. As shown in Fig. 3(b), we first regularize μ in a limited range $[-l, l]$ with symmetric center as $\mathbf{p}_\perp$. We then divide the limited range into two levels using a segment operator $\mathcal{S}$, namely the coarse level with N_c bins and the fine level with N_f bins. The intersection network then is formulated as:

$$\mathcal{F}_\Phi : \mathbf{r} \mapsto (\hat{v}_c, \hat{v}_f, \hat{\alpha}), \tag{3}$$

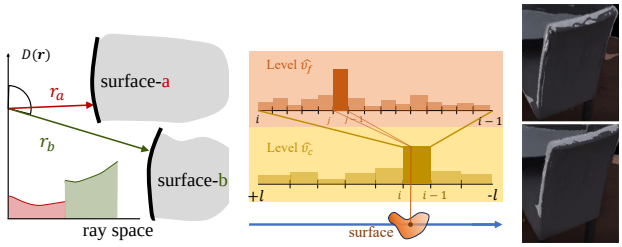
where $\hat{v}_c, \hat{v}_f$ are 1D vectors representing the coarse- and fine-level target bins to which the surface belongs, respectively. The resolution of the two-level classifier is $N_c \times N_f$ (where N_c and N_f are set to 64 and 128, respectively). Then, μ is calculated as follows:

$$\mu = \mathcal{S}^{-1} \circ \mathcal{F}_\Phi = \lambda_1 p_c(\hat{v}_c) + \lambda_2 p_f(\hat{v}_f) - l, \tag{4}$$

where $\mathcal{S}^{-1}$ is the inverse operator of $\mathcal{S}$, $p_c(\hat{v}_c) = argmax(\hat{v}_c)/N_c$, $p_f(\hat{v}_f) = argmax(\hat{v}_f)/N_f$, λ_1 is $2l$ and λ_2 is $2l/N_c$. Because the target bin's indicator operation is a non-differentiable operation, we choose to use binary cross-entropy (BCE) loss (see Eq. 6) during training. We use the *argmax* function to pick out the target bin during testing.

In our implementation, the intersection network is an MLP. The MLP's head is a single linear layer; the MLP's body is made up of 16 residual blocks, each of which is made up of two fully connected layers followed by ReLU activation; the MLP's tail has two branches to produces $\hat{\alpha}$, $\hat{v}_c$ and $\hat{v}_f$ (see Appendix for more details). We supervise this intersection network with the pre-trained NeRF models. Specifically, the distance from the camera to the first ray-object intersection can be measured by:

$$D(\mathbf{r}) = \sum_{i=1}^{N} w_i \left(t_i |\mathbf{d}|\right), t_i \in [t_n, t_f], \tag{5}$$



(a) (b) (c)

Figure 3: Directly regressing μ may result in significant errors in the discontinuity area (a) during composition (see the top row of c). We address this problem by replacing the regression of μ with a multi-level classifier (b), which yields a better result (see the bottom row of c).

where w_i is the weight of i-th sample along rays in NeRF's render equation, $\mathbf{d}$ is the direction of ray $\mathbf{r}$, t_n and t_f are the near and far clip ranges of the camera, respectively. This depth is then converted to μ using the inverse of Eq. 2 for supervision.

The total training loss for the intersection network can be formulated as:

$$\mathcal{L}_{NeDF} = \text{BCE}(\hat{v}_c, v_c) + \text{BCE}(\hat{v}_f, v_f) + 0.1 \times \text{BCE}(\hat{\alpha}, \alpha), \tag{6}$$

where v_c, v_f are one-hot vectors of the same dimension as $\hat{v}_c, \hat{v}_f$. They can be obtained by applying segment operator $\mathcal{S}$ on μ generated by NeRF. α is the ground truth mask generated by NeRF.

Fast NeRF Composition with NeDF

Built on NeDF, we are able to achieve a fast composition and rendering framework. We illustrate the structure and data processing flow of this framework on the figure in the right of Fig. 1. It generally consists of three steps. First, the NeDF generation step takes all of the rays cast from the camera (eye position) through each pixel of the frame as input to output a depth, $D(\mathbf{r})$, and object ID buffers. It saves the corresponding depth and object identification information for each element of $D(\mathbf{r})$. Deferred Shading step, as the second step, takes the ID and depth buffers outputted by the previous NeDF Generation step and the rays $\mathbf{r}$ as an input. Then, it outputs a color buffer that holds the raw color (without shadow) information about the pixels. Lasty, the Shadow step uses the shadow rays, $D(\mathbf{r})$, and $\mathbf{r}$ to generate the shadow information in a shadow buffer output. The final rendering result is produced by multiplying the color buffer and shadow buffer. Next, we first describe how the method handles the affine transformation of each NeRF object, as well as the necessary details for rendering acceleration.

Manipulating Objects. Given a 3D vector $\mathbf{v}$ in world space and a corresponding 3D vector $\mathbf{q}$ in canonical space, we define the rigid transformation of a NeRF object as a mapping function $\mathcal{G} : \mathbf{q}(\mathbf{R}, \mathbf{T}, s) \mapsto \mathbf{v}$ where $\mathbf{R}, \mathbf{T}, s$ specify the rotation, translation, and scale, respectively. Previous NeRF works focusing on deformation such as (**????**) transform the sampling points to canonical space to achieve the transformation. We instead directly transform the starting point and direction of rays to local canonical space and use this transformed rays to query the implicit field. As the intersection network takes rays $\mathbf{r}$ as input, the depth $\hat{D}(\mathbf{r})$ in global space can be computed using the inverse of the affine transformation function $\mathcal{G}$:

$$\hat{D}(\mathbf{r}) = |\mathbf{o} - \mathbf{p}_\perp| - s\left(\mathcal{S}^{-1} \circ \mathcal{F}_\Phi\right)\left(\mathcal{G}^{-1}(\mathbf{r})\right). \tag{7}$$

Deferred Shading. We term the step generating the raw color information as Deferred Shading step because each pixel only requires one NeRF network $\mathcal{F}_\Theta$ to shade in our framework. As the intersection network can accurately estimate the depth information, we are able to replace the ray marching step in previous NeRF works with a degenerated ray casting form, i.e., a pixel color can be acquired by a single evaluation of $\mathcal{F}_\Theta$. To further shade all objects in a lighting-aware manner, it is necessary to handle different lighting conditions in the original space from which the objects are collected, which is a non-trivial task. We simplify this processing by only shading with the primary color. Specifically, we obtain each ray's color by:

$$\mathbf{c}, \sigma = \mathcal{F}_\Theta\left(\mathcal{G}^{-1}(\mathbf{o} + \hat{D}(\mathbf{r})\mathbf{d}), \mathcal{G}^{-1}(\mathbf{d})\right), \tag{8}$$

where color c at the ray-object intersection point is directly used to approximate the ray's color $\hat{C}(\mathbf{r})$. Density σ is employed to check whether the estimated $\hat{D}$ is convincible. An extremely small σ indicates that $\hat{D}$ does not reach the object surface. In this situation, we set a threshold that optionally assists in resampling these outliers using primary volume render equation in NeRF for better quality but a slight speed penalty (see our experimental results in Table 2). Although replacing the raymarching step with a single sample limits its usefulness, it is adequate for previewing. Users may prefer interactive-level speed over perfect quality when manipulating each object. In addition, once the adjustment is confirmed, the pipeline can re-render the scene and generate a set of high-quality results using only the NeRF models.

Fast Dynamic shadows Casting

Shadows for NeRF models can be generated using the traditional volume rendering technique developed by (**?**). It, however, requires an additional volumetric evaluation for each sample point along a ray to determine whether the point is in shadows, which significantly increases the time consumption. Given that target objects have opaque in our cases, we generate the dynamic shadows based on the fundamental concept of shadow mapping (**?**) with NeDF for quick preview. We detail this dynamic shadow casting as an additional step (Shadow Step, also named step 3) in appendix.

Experiments

In this section, we first demonstrate that our pipeline indeed takes effect for various NeRF works. Following that, we show how our framework can assemble a large number of neural objects to form a virtual 3D world. Furthermore,

(a) NeDF+Mip-NeRF (b) NeDF+SNeRF (c) NeDF+Neus

Figure 4: Visual quality illustration of the general plugin. For each method, the left column shows the previewing result, the middle column shows the naive result, and the right column visualizes the flip errors (**?**).

	NeDF+Mip-NeRF vs. Mip-NeRF		NeDF+SNeRF vs. SNeRF		NeDF+Neus vs. Neus	
	time (s)	s-time(%)↑	time (s)	s-time(%)	time (s)	s-time(%)
Outlier Processing (rs)	0.417/15.677	97.34	0.519/15.528	96.66	2.11/21.72	90.29
w/o Outlier Processing (wo rs)	0.234/15.677	98.51	0.225/15.528	98.55	0.202/21.72	99.07

Table 1: Our proposed framework serves as a plugin for three representative NeRF models: Mip-NeRF, SNeRF, and Neus. Term "s-time" refers to the percentage of time that our method saves. The optional outlier processing in Eq. 8 provides better quality but a slight speed penalty. The resolution is set to 800×800.

we show our approach can be combined with traditional rendering techniques to create a mixed rendering pipeline. We implement our framework in two versions: Pytorch version for a convenient reproduction and CUDA version for testing performance limitation. We conduct experiments on our new N-object dataset which has been described in section 'N -object dataset testing'.

Training details. A single Nvidia A100 GPU is used to train NeDF for each scene from our N-object dataset. We employ Adam as the optimizer and set the learning rate to 5e-4. We use the pre-trained NeRF models to generate 500 random views (more random views, less artifacts) for supervision for each NeDF model of a single object, and train the intersection network over 60W iterations until convergence. The batchsize of training rays is set to 4,096 during each iteration.

General Acceleration Plugin

To evaluate the efficacy of the proposed framework acting as a general pipepline, we conduct experiments on the pre-trained models of certain classic implicit NeRFs. Three representative implicit neural representation based NeRF methods, i.e., Mip-NeRF (**?**), SNeRF (**?**), and Neus (**?**), are employed for this set of experiments. Mip-NeRF represents the kind of NeRF methods that focus on novel view synthesis and rendering. SNeRF is a typical NeRF method that supports appearance style transfer, while Neus is a representative for NeRF methods designed for task of geometry reconstruction. We integrate these three representative NeRF techniques on our dataset and show their corresponding quantitative performance results in Table 1 and visual results in Fig. 4. Evidently, we can see that these representative NeRF methods, when used in conjunction with our proposed framework, significantly reduce rendering time while maintaining an acceptable level of quality degradation.

Storage and Speed

As shown in SNeRF (the same architecture with vanilla NeRF) column in Table 1, our framework can speed up rendering more than $30 \sim 40$ times over the vanilla NeRF.

Comparision with other solutions. Although there's no general pipeline for NeRF composition, we force a comparsion with NSVF (**?**), the closest competitor for NeRF fast composition, and some other works focus on storage in compositing. In terms of speed, it has demonstrated that our framework can save more than twice as much time as NSVF ($\sim 20\times$, see Figure 2 in the supplementary appendix). In terms of storage, our models (NeDF+NeRF) requires 13MB (3MB for NeRF), NSVF (grid+NeRF) requires 3.2$\sim$16MB, DONeRF requires 3.6MB, and CC-NeRF requires 1.5MB. It should be noted that CC-NeRF is not intended for fast composition, and DONeRF is incapable of handling arbitrary composition with affine transformation. By comparing our method to these, we demonstrate that our method does not require excessive storage and is sufficiently fast. What's more, our framework has several significant advantages that

	air_baloons	idr_chair	nerf_chair	car	lego4pi	carousel	carousel_hors	cherryBlosso	ferrise-base	ferrise-wheel	ferrise-bin	plant	basketball	football	wood_chair	wood_table	wood_floor	ground1	ground2
b3															1	1	1		
b10			1		1			1				1			1	1		4	
b20			1	1	1	1	5	1				3			2	1		4	
b50	3	2	1	1	1	1	8	2	2	2	10	3			2	1	1	4	6

Figure 5: Number of objects contained in *b3*,*b10*,*b20*,*b50* scene. The number in two ball cases is empty, which is because we use them for the capability pressure test and repeated basketballs and footballs (sharing fifty-fifty) in the 100-ball and 1k-ball scenes.

the other approaches (NSVF and CC-NeRF) lack, including: 1) unlike to our framework, neither NSVF nor CC-NeRF can be used as a general acceleration plugin and 2) NSVF and CC-NeRF cannot achieve interactive-level manipulation speeds or generate shadows like our method and 3) both of CC-NeRF and NSVF incur additional memory cost when rendering repeated objects while our framework avoids this issue.

Capbility evalution

N-object dataset testing. First, we collected an N-object dataset consisting of 22 distinct NeRF objects: 7 objects were chosen from previous work (3 from Neus' dataset of real-world objects, 4 from synthesis dataset (**???**)), and the remaining 15 objects were newly created by Blender 3D. To evaluate the framework's ability to composite complex virtual scenes (see Fig. 6), we conduct a set of virtual scene compositions with four-level increasing complexity: *b3, b10, b20, b50*. The *b3* and *b10* are two static scenes with lower complexity, and the *b20* and *b50* are dynamic scenes where some objects move according to user-defined transformation settings. We allow to insert the same NeRF object instances repeatedly into the scene to increase the complexity of the virtual scene. For each scene, we set different camera views (such as the views in Fig. 6) and render 120 frames to evaluate the average performance. Table 2 shows the consumed time and quality of each composition. Figure 8 depicts the time consumption ratio of each rendering step (see appendix) in rendering a single frame. As complexity increases the resampling ratio increases, which results in significant differences in computational time across different scenes in the *Deferred Shading step*. To test the capability upper bound of our framework, we also conduct a pressure test on virtual scenes with 100 and 1k balls to see how stable the proposed approach is for scene composition and shadow casting (see Fig. 6). Despite taking a longer time (see Tab. 2), our pipeline delivers stable performance and does not incur additional memory cost for those repeated balls.

CUDA implementation. CUDA version of the proposed framework is built with Vulkan, TensorRT, and a customized CUDA kernel to enable real-time manipulation on Nvidia RTX 4090 GPU. With this, we enable the user to construct a

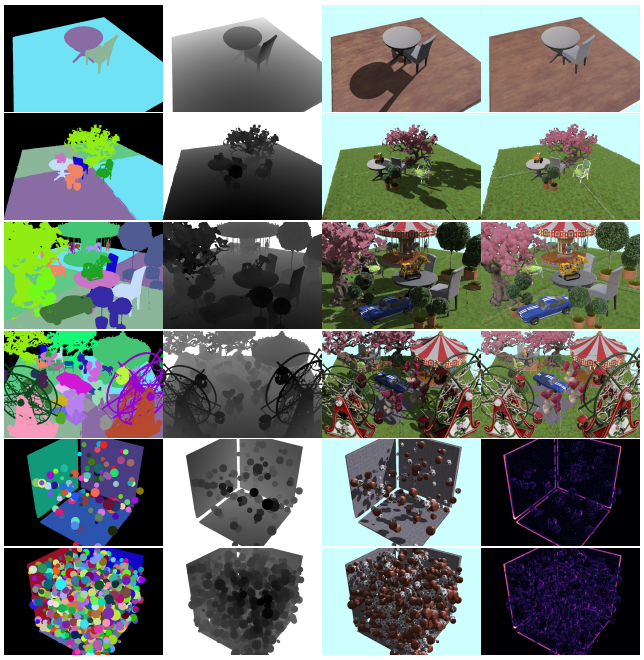


Figure 6: Results on our N-object dataset. Each 4-image pair shows: ID map, depth map, results with shadows using our previewing framework, and results with only naive NeRFs (shadowless). The Rightside of last two rows is the flip error (**?**) between previewing and NeRF's naive composition results). From top to bottom: *b3 and b10, b20 and b50, 100-ball and 1k-ball* virtual scenes. Scenes *b20* and *b50* are both dynamic, with carousels and ferris wheels moving through user control settings; scenes *100-ball* and *1k-ball* have 100 and 1,000 balls (half basketballs and half footballs), respectively.

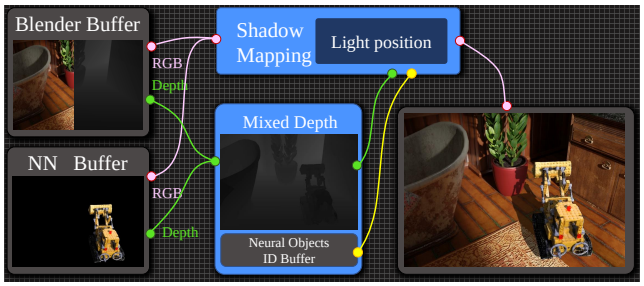


Figure 7: Given the information of camera view and light position, the buffer produced by the proposed framework (NN Buffer) and the *Cycles* engine of Blender 3D can be combined to form a mixed render result (see our video for a dynamic result). The *Bathroom* scene is from (**?**).

		b3-scene		b10-scene		b20-scene		b50-scene		100-ball		1k-ball	
		time(s)	PSNR	time	PSNR	time	PSNR	time	PSNR	time	PSNR	time	PSNR
wo s	rs	0.240	25.78	0.658	23.32	1.678	20.02	1.881	18.67	1.887	26.54	16.63	25.26
	wo rs	0.209	23.97	0.443	19.92	0.873	15.26	1.209	15.43	1.741	24.19	16.34	22.58
shadow	rs	0.398	\	1.040	\	2.441	\	2.943	\	3.073	\	28.97	\
	wo rs	0.367		0.813		1.628		2.286		3.002		27.66	

Table 2: Performance of the implemented Pytorch framework on an N-object dataset ('wo s' means no shadow is calculated; 'rs' stands for resampling, i.e., outlier processing in Table 1). The resolution is set to 900×600. We use the naive composition (right side in Fig. 6) by NeRF as ground truth for composition without shadows. We only test the average time (in seconds) in each frame for shadow composition. It should be noted that the quality of the 1K-ball pressure testing was superior to the former due to the simpler geometry of the objects. Nonetheless, due to excessive overlapping between balls, the time increases rapidly.

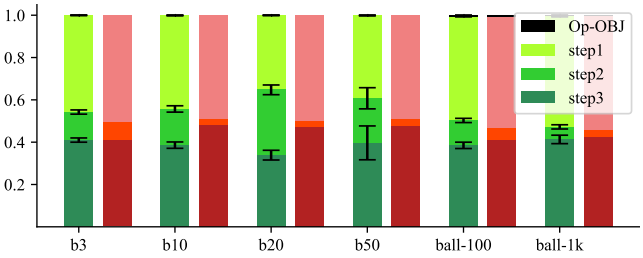


Figure 8: The time consumption ratio of our framework for the main steps. *Green* and *Red* bars represent the results with and without outliers resampling, respectively. Regarding the green bar, refer to this figure in conjunction with Fig. 6, when the camera view varies, certain objects which occupy a larger portion of the frame, e.g., the trees, have a higher resampling ratio and thus increase the processing time. Focusing attention on the red bar, the Step 2 (deferred shading step) consistently assumes a relatively minor temporal allocation across all scenarios. This is because, regardless of the number of objects in the scene, a pixel will be shaded only once with a single NeRF object.

virtual scene in a progressive and interactive manner. The real-time manipulation benefits from a buffer reuse technique to accelerate the performance during manipulation. Specifically, when the user manipulates an object, the other objects' rendering processes are frozen, and their buffers are reused to decrease the computational load.

Compatibility with traditional graphics pipelines

The proposed framework is also suitable for integration with traditional rendering pipelines to build a mixed pipeline that supports dynamic shadow casting across meshes and neural objects in a single scene (see Fig. 7). Specifically, we first use a shared camera & light settings across mesh and neural scene to generate rendering buffers. Subsequently, these two scenes are seamlessly integrated using depth-based composition. Finally, shadow mapping is applied to produced a mixed result.

Conclusions and Limitations

We present a general framework for a fast composition of NeRF objects with dynamic shadows. The core technical contribution is the use of Neural Depth Fields to allow for the direct intersection of rays and implicit surfaces to quickly determine the spatial relationship between objects as well as surface color computation of NeRF objects. To the best of our knowledge, this is the first framework that allows for the rapid creation of a virtual scene composed of a large number of NeRF objects with dynamic shadows. Our method can also be combined with traditional renderers to form a mixed pipeline, increasing its versatility and applicability.

The proposed framework is currently limited to solid surface objects, and predicting weight distribution in local space rather than a single depth may be a better option. In the future, this compositing framework can be further enhanced with more powerful editable NeRF works (**???????**) to achieve global illumination and improved visual quality.

Acknowledgments

This work was supported by Key R&D Program of Zhejiang (No. 2023C01047) and the National Natural Science Foundation of China (Grant No. 62036010). The authors acknowledge the support of Bytedance's MM Lab's GPU cluster, as well as Yanzhen Chen's early assistance in searching for various works.

Aperiam aliquid quam placeat laboriosam tempora velit quaerat aliquam, rem fugiat minus amet odio cum, sunt numquam praesentium corrupti culpa eveniet aspernatur molestiae dicta dolor ad.Nostrum aliquam blanditiis in deserunt assumenda harum alias id, fugiat libero eligendi.Minus facere placeat et explicabo sit vel inventore optio consequatur aliquid, dignissimos consequatur fuga dolorem maiores recusandae minima amet provident, eligendi quasi mollitia ducimus dolores vel, at nulla nihil cum ducimus labore dignissimos aspernatur?Soluta placeat quam sunt odio, necessitatibus natus libero distinctio illum.Minus ipsam nisi vero, necessitatibus ex rerum magnam illum numquam quis cumque aut, nesciunt error quisquam odit distinctio quod porro nostrum ea cum veritatis esse, officiis doloremque nesciunt possimus sit repudiandae libero, molestias quidem nesciunt non magnam ducimus porro aliquam

esse beatae?Voluptates iusto animi ducimus eum et quasi doloremque quaerat voluptatem, eius eveniet suscipit consectetur nam omnis ullam assumenda, in