Recursive-NeRF: An Efficient and Dynamically Growing NeRF

Guo-Wei Yang, Wen-Yang Zhou, Hao-Yang Peng, Dun Liang, Tai-Jiang Mu, and Shi-Min Hu, *Senior Member, IEEE*

Abstract—View synthesis methods using implicit continuous shape representations learned from a set of images, such as the Neural Radiance Field (NeRF) method, have gained increasing attention due to their high quality imagery and scalability to high resolution. However, the heavy computation required by its volumetric approach prevents NeRF from being useful in practice; minutes are taken to render a single image of a few megapixels. Now, an image of a scene can be rendered in a level-of-detail manner, so we posit that a complicated region of the scene should be represented by a large neural network while a small neural network is capable of encoding a simple region, enabling a balance between efficiency and quality. *Recursive-NeRF* is our embodiment of this idea, providing an efficient and adaptive rendering and training approach for NeRF. The core of Recursive-NeRF learns uncertainties for query coordinates, representing the quality of the predicted color and volumetric intensity at each level. Only query coordinates with high uncertainties are forwarded to the next level to a bigger neural network with a more powerful representational capability. The final rendered image is a composition of results from neural networks of all levels. Our evaluation on public datasets and a large-scale scene dataset we collected shows that Recursive-NeRF is more efficient than NeRF while providing state-of-the-art quality. The code will be available at https://github.com/Gword/Recursive-NeRF

Index Terms—Scene representation, view synthesis, image-based rendering, volume rendering, 3D deep learning

1 Introduction

Image-based rendering (IBR) is a popular topic in computer graphics with demonstrated value in virtual reality and deep learning for novel view synthesis and data augmentation. The basic idea is to reconstruct the underlying geometry and appearance from a set of images, using representations which may be mesh-based [1], [2], [3], [4], volumetric [5], [6], [7], [8], [9], [10] or implicit [11], [12]. These are used to synthesize novel views by interpolation [13] or rendering techniques [11], [12]. A recent development, the Neural Radiance Field (NeRF) method [14] implicitly encodes a scene or object using a fully-connected neural network, optimized by a naturally differentiable method. It provides excellent novel high-resolution photorealistic views using a *continuous* volumetric representation. It thus has been extended to large-scale scenes [15], non-rigidly deforming scenes [16], dynamic lighting and appearance [17], etc.

Though NeRF achieves unprecedented synthesis quality, its rendering process is extremely slow and makes high memory demands, so is unattractive for practical use. The bottleneck is the calculation of each pixel value by integrating along a rendering ray, which is approximated by hierarchical volume sampling in a similar way to importance sampling. For each ray, NeRF samples 192 coordinates, each forward passing through the whole neural network, and in total millions of rays are required to render a single moderate-resolution image (say 800×800 pixels). Previous work [18] has improved NeRF's rendering speed by sampling more carefully using a sparse set of voxels, and avoiding evaluations on empty voxels.

Let us consider some drawbacks of NeRF's rendering process. First, NeRF uses the same network for all sample points, so NeRF encodes the whole scene using a single neural network model. For more complicated scenes, it is necessary to ensure the neural network has sufficient representational power. This is done by using an increased number of network parameters, additional hidden layers, or increased dimensions of the latent vectors. As a result, for every individual query sample, both the training and inference time of the network increase with greater scene complexity. Furthermore, NeRF cannot self-adapt to scenes of different complexity. Second, regardless of whether inidividual query samples are complicated or simple, NeRF treats them equally and passes them through the entire neural network, which is overkill for regions which are empty or have simple geometric structures and textures. These limitations seriously affect large-scale scene rendering.

Related problems in other rendering methods have been solved by using a level of detail (LOD) approach [19]. We suggest that it can be carried across to give an adaptive and efficient neural rendering approach based on NeRF: the rendered value of a sample needs to be further processed if and only its rendered quality is not high enough at the current level of the neural network. Moreover, a coarser level may be represented with a smaller neural network while more

• *The authors are with the BNRist, Tsinghua University, Beijing 100084, China. E-mail: {ygw19, zhouwy19, phy18}@mails.tsinghua.edu.cn, randonlang@gmail.com, {taijiang, shimin}@tsinghua.edu.cn.*

Manuscript received 17 May 2021; revised 10 August 2022; accepted 22 August 2022. Date of publication 4 October 2022; date of current version 10 November 2023.
This work was supported by the National Key R&D Program of China under Grant 2021ZD0112902.
(Corresponding author: Shi-Min Hu.)
Recommended for acceptance by T. Ju.
This article has supplementary downloadable material available at https://doi.org/10.1109/TVCG.2022.3204608, provided by the authors.
Digital Object Identifier no. 10.1109/TVCG.2022.3204608

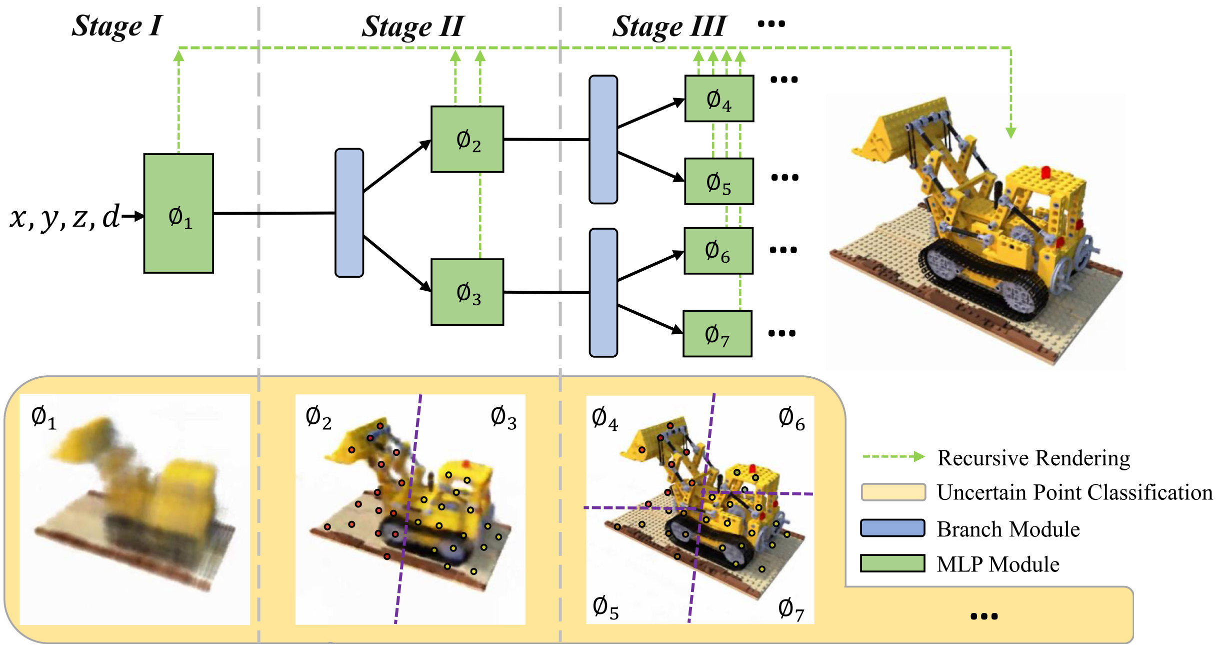

Fig. 1. Pipeline of Recursive-NeRF. Given a position (x, y, z) and viewing direction d, the initial network Φ_1 outputs color c_1, density σ_1, and uncertainty δ_1. All uncertain points are divided into several categories, and then Φ_1 dynamically grows several branch networks to continue training for each subset of uncertain points until the network is believes that all points are reliably predicted. When rendering, early termination allows different points to exit at different times, reducing the network load.

detailed levels are represented with larger neural networks. The rendering of a sample adaptively passes through the neural network according to that sample's complexity.

We embody this concept in a method we call Recursive-NeRF (see Fig. 1) , which recursively applies the NeRF structure with various number of linear layers in each stage when needed. Starting with a small neural network, at each level, in additional to the color and volumetric intensity, Recursive-NeRF also predicts an uncertainty, indicating the quality of the current results. Recursive-NeRF then directly outputs results for those query coordinates in the current level with low uncertainty, instead of passing them forward through the rest of the network. Query coordinates with high uncertainty are forwarded in clusters to the next level, represented as multiple neural networks with more powerful representational capability. The training process terminates when the uncertainties for all query coordinates are less than a user-specified threshold, or some maximal number of iterations is reached. In this way, Recursive-NeRF splits the work adaptively to decouple different parts of the underlying scene according to its complexity, helping to avoid unnecessary increase in network parameters. Experiments demonstrates that Recursive-NeRF achieves significant gains in speed while providing high quality view synthesis.

In summary, our work makes the following main contributions:

- a recursive scene rendering method, where early termination prevents further processing once output quality is good enough, achieving state-of-the-art novel view synthesis results with much reduced computation, and
- a novel multi-stage dynamic growth method, which divides uncertain queries in the shallow part of the network, and continues to refine them in differently grown deep networks, making the approach adaptive for scenes with areas of differing complexity.

2 Related Work

Our approach uses a neural 3D shape representation and dynamic neural networks for image based synthesis. A full review of these ideas is outside the scope of this paper, and we refer interested readers to [20] for classical IBR and [21], [22] for neural rendering. We consider the most closely related works below.

2.1 Neural 3D Shape Representations for View Synthesis

Recently, there has been work training an MLP network to continuously represent a 3D scene, mapping 3D coordinates to an implicit representation, e.g., the signed distance function (SDF) [23], [24] or an occupancy field [25], [26]. Such approaches usually need to be supervised with ground-truth 3D geometry. An an alternative, learning from a set of images has benefits, since images are more readily available, and supervision can be implemented with neural rendering techniques [2], [27]. Scene Representation Networks (SRN) [11] uses an MLP network to learn scene geometry and appearance, proposing a differentiable ray-marcher to train the network end-to-end in an unsupervised manner. Neural Volumes (NV) [6] learns a dynamic irregular warp field during ray-marching. Local Light Field Fusion (LLFF) [8] expands each input view into a local light field through a multiplane image (MPI), then mixes adjacent local light fields to render novel views.

Recently proposed, NeRF [14] uses a sparse set of input views to optimize an MLP network which inputs a query point and outputs color and density. NeRF trains the network and renders the scene by sampling points in space by ray marching; it can generate high resolution images of high quality. This approach has been adapted to handle more complicated scenarios. For example, Zhang et al. [15] solve the parameterization problem arising when applying NeRF to object capture in 360° large-scale scenes. Srinivasan et al. [17] enhances NeRF for view synthesis under any lighting conditions. Schwarz et al. [28] propose generative radiance fields for 3D-aware image synthesis. Park et al. [16] optimize an additional continuous volumetric deformation field for non-rigidly deforming scenes. Ma et al. [29] composites virtual objects with a real photograph to emulate shadows and reflections. Transformer has made great progress in the field of vision recently [30], [31], [32], [33]. NeRFormer [34] combines NeRF with Transformer to reconstruct objects in a small number of views. HyperNeRF [35] maps the input image to the canonical template coordinate space to solve the problem of topological discontinuity in the deformation field. Barron et al. propose Mip-NeRF [36] which improves the quality of NeRF by using conical frustums to eliminate aliasing and blurring. Ren et al. [37] constructs sub-networks according to the structure of KD-Tree for predicting the illumination. Different from ours, they directly add subnetworks if the effect does not get better, and the re-division is canceled and then trained, which is very time-consuming.

However, these NeRF-based methods need a large number of samples in the rendering line of sight, so are slow. NSVF [18] introduces a sparse voxel octree to represent the space which can skip empty space and allocates more representational power to difficult regions areas by subdividing the corresponding voxels, resulting in improved speed and quality. DONeRF [38] reduces the number of sampling points by predicting the depth of the scene. Lindell et al. [39] propose automatic integration to estimate volume integrals along the viewing ray in closed-form to avoid sampling, but their method suffers from quality degradation due to the piecewise approximation. Although these methods have significantly accelerated NeRF, there is still much room for further improvement. Our dynamic network adapts to the complexity of the scene, significantly reducing the amount of calculation. Our method can further adaptively adjust the model complexity through the uncertainty ratio to adapt to scenarios of different complexity, as shown in Table 7. It is complementary to NSVF [18] and DONeRF [38], and could be combined with these methods for further speed-up and enhancing adaptability.

Recently, a series of good speed improvements have been achieved by caching neural network results [40], [41], [42], but these methods will bring additional memory consumption. In the future, we will consider researching ways to improve rendering speed without additional memory consumption.

DeRF [43] is the most similar approach to ours. DeRF uses the uniformity of sampling point division as a loss and optimize through neural network to decompose the scene. However, DeRF will lead to the problem of uneven division of complex scenes and a significant increase in the amount of computation as the number of heads increases. While our Recursive-NeRF can divide the scene more reasonably and adaptively according to the predicted distribution of uncertain points, thus achieving higher performance with more efficient computation.

2.2 Dynamic Neural Networks

Dynamic neural networks dynamically adjust the network architecture according to equipment resources.

Multi-scale dense networks [44] train multiple classifiers according to different resource demands and adaptively apply them during testing. [45] proposes switchable batch normalization and slimmable neural networks, which can adjust width according to device resources. [46] extends this idea to execute at arbitrary width. [47] extends slimmable neural networks to change numbers of channels for better accuracy with constrained resources.

These approaches adjust the network architecture according to equipment resources, whereas we adaptively adjust the network architecture according to the network training situation. Also, previous dynamic networks solve a classification task. Since the output of the classification network is the probability of predicting each category, the probability can naturally be used as the confidence. Ours is a regression task, which determines whether the network is exited by predicting a confidence value. This is harder to train than a classification task.

3 Analysis of Neural Radiance Fields

3.1 Neural Radiance Fields

NeRF inputs continuous 5D coordinates, composed of a 3D position and a 2D viewing direction, and estimates view-dependent radiance fields and volume density at the corresponding position. By producing radiance fields, NeRF can simulate highlights and reflections well. NeRF calculates the color C(r) of a pixel in an image by integrating the ray from the camera to the pixel:

$$C(\mathbf{r}) = \int_0^{+\infty} T(t)\sigma(r(t))c(\mathbf{r}(t), \mathbf{d})\, dt \tag{1}$$

where $r(t) = \mathbf{o} + t\mathbf{d}$ is the camera ray emitted from $\mathbf{o}$ in direction $\mathbf{d}$, and $T(t)$ represents the cumulative transparency from 0 to t:

$$T(t) = \exp\Big(-\int_0^t \sigma(\mathbf{r}(s))\, ds\Big). \tag{2}$$

c and σ are directional emitted color and volume density which are calculated via an MLP network F_θ:

$$F_\theta : (\mathbf{x}, \mathbf{d}) \rightarrow (c, \sigma) \tag{3}$$

3.2 Parameters and Scene Complexity

Scenes of greater complexity need to be represented using a larger number of parameters. At the same time, simple scenes can be represented by a small number of parameters.

We tested the PSNR of NeRF [14] on the Lego dataset for different numbers of network layers (2, 4, 6, 8), network widths (64, 128, 256) and image sizes (25, 50, 100, 200, 400, 800). The capacity of the network is positively corelated with the number of network layers and the network width. Here, images are downsampled from the original resolution of 800*800 pixels by anti-aliased resampling, and the image size is a proxy for the complexity of the scene. It can be seen from Fig. 2 that when the scene is relatively simple, the representational capabilities of different networks are relatively

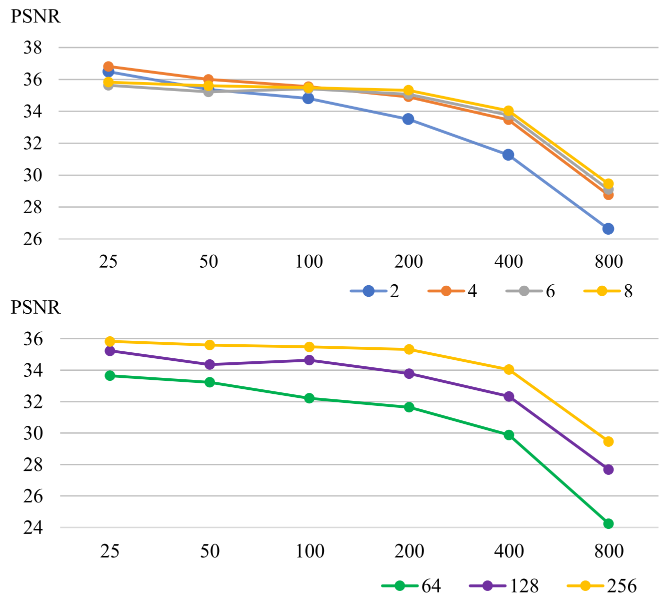

Fig. 2. Correlation between parameters and scene complexity. Different curves in the top and bottom plots represent various network depths and widths, respectively. The horizontal axis is the resolution of the square image, representing the complexity of the model. The vertical axis is the PSNR of the image, the higher the better.

close to each other. As the scene becomes complex, the gap between the representational capabilities of different networks widens.

There is an intuitive solution: simply split the scene into several parts, with each part being represented by an identical individual network. However, this solution has problems. Each part uses the same network architecture, while the complexity of different parts of the scene may differ, so ideally networks with different capabilities should be used to represent them. Furthermore, coarse-grained information will be learned repeatedly. Recursive-NeRF overcomes both of these issues using a more sophisticated approach.

4 Recursive Neural Radiance Fields

Recursive-NeRF use the NeRF approach in an LOD manner to adapt to the complexity of the underlying scene, which is trained in stages and changes dynamically, as shown in Fig. 1. At each stage, according to the predicted uncertainty, a query coordinate will be finalised or forwarded to the next stage which uses more powerful neural networks, controlled by an early termination mechanism. All finalised predictions of color and intensity from each stage are gathered to render the final image.

In this section, we first introduce neural recursive fields (Sec. 4.1) which represent the whole scene from coarse to fine. Early termination (Sec. 4.2) allows our network to finalise the prediction when the uncertainty is low enough, avoiding unnecessary calculations and speeding up rendering. We use the k-means algorithm to cluster the high uncertainty points in the current stage, thus dividing the scene into several parts for finer-grained prediction. Additionally, the network grows several child branch networks to achieve dynamic growth (Sec. 4.3). Overall, we recursively render (Sec. 4.4) the entire scene, with input coordinates entering different branches for network prediction based on previous clustering results.

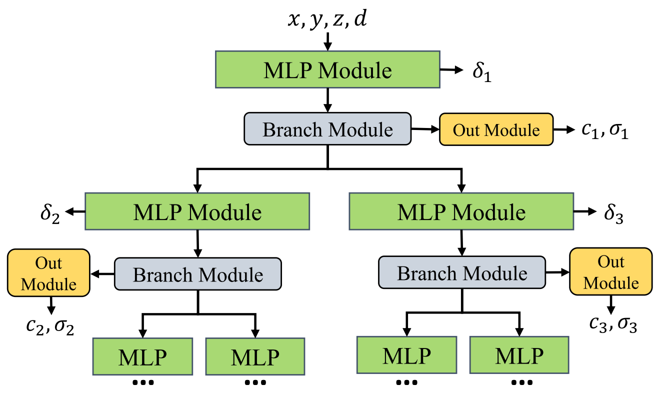

Fig. 3. Network architecture of Recursive-NeRF. For every query (x, y, z, d), the network predicts an uncertainty δ used to decide if the query should be finalised early. If so, it will enter the OutNet to predict its color c and density σ. If not, the point split unit determines which branch it should subsequently enter.

4.1 Recursive Neural Fields

A recursive neural field takes its parent branch's output y_{p_i} and the viewing direction d as inputs, and predicts color c_i, density σ_i, uncertainty δ_i and a latent vector y_i:

$$F_{\Phi_i} : (y_{p_i}, d) \rightarrow (c_i, \sigma_i, y_i, \delta_i) \tag{4}$$

where F_{Φ_i} represents the ith subnetwork. F_{Φ_1} is the root of our recursive network; in this case, y_{p_i} is set to the query coordinate (x, y, z).

As shown in Figs. 1 and 3, sub-network F_{Φ_i} consists of three main components: an MLP module, a Branch module and an Out module. The MLP module includes two or more Linear layers to ensure that the MLP module performs sufficiently complex processing of features. The Branch module predicts the uncertainty δ_i of each query point, forwards the points with low uncertainty to the Out module for output, and distributes points with high uncertainty to different sub-networks according to their distances to the k_i cluster centers of F_{Φ_i}. The Out module is responsible for decoding features into c_i and σ_i.

4.2 Early Termination

Our early termination mechanism allows the query coordinate to be finalised early (so not processed further) when its predicted uncertainty is less than a certain threshold. We next present our novel uncertainty prediction method for ray marching, then explain the special training method for Recursive-NeRF.

4.2.1 Uncertainty Prediction

Each branch network predicts an uncertainty for the query coordinate, which we use to determine where the branch network exits. We use the original NeRF loss to help us predict uncertainty. NeRF adopts mean square error (MSE) between rendered images and real images as the loss for training coarse and fine networks:

$$\mathcal{L}_{MSEc} = \sum_{r \in \mathcal{R}} \left\| \hat{C}_c(r) - C(r) \right\|_2^2 \tag{5}$$

$$\mathcal{L}_{MSEf} = \sum_{r \in \mathcal{R}} \left\| \hat{C}_f(r) - C(r) \right\|_2^2 \tag{6}$$

where $\mathcal{R}$ contains rays in a mini-batch, $\hat{C}_c(r)$ and $\hat{C}_f(r)$ are rendered colors from coarse and fine networks, respectively, and $C(r)$ is the ground truth. Our coarse and fine networks have the same network structure and training. The coarse and fine architecture is dedicated to sampling points since our network is designed to avoid unnecessary calculation for each sample point. Indeed, the previous uncertainty response can be used as sampling guidance for NeRF's fine network. For simplicity, we no longer distinguish coarse and fine networks, and use $\hat{C}(r)$ to represent the color rendered by any network.

We introduce two regularization losses to train the uncertainty effectively. We use a Linear layer following the output feature y_{p_i} of F_{Φ_i} to calculate the uncertainty δ_i. We use the squared error of the pixel to supervise δ_i, the intent being that if a pixel has large error, δ_i of the sample points that generate the uncertainty associated to the sample points should also be large. Therefore, we punish δ_i less than the squared error:

$$\mathcal{L}_{SE} = \sum_{r\in\mathcal{R}}\sum_{i=1}^{N} max(E(r) - \delta_{r,i}, 0) \tag{7}$$

$$E(r) = \left\|\hat{C}(r) - C(r)\right\|_2^2 \tag{8}$$

where $E(r)$ is the squared error of ray r, N is the number of sampling points for ray r, $\delta_{r,i}$ is the predicted uncertainty for the i-th sample point of ray r.

To prevent δ_i from blowing up, we introduce another regularization loss: for every query point, we encourage δ_i to be as close to zero as possible:

$$\mathcal{L}_0 = \sum_{r\in\mathcal{R}}\sum_{i=1}^{N} max(\delta_{r,i}, 0) \tag{9}$$

A weighted sum of $\mathcal{L}_{SE}$ and $\mathcal{L}_0$ gives the overall uncertainty loss:

$$\mathcal{L}_{unct} = \alpha_1\mathcal{L}_{SE} + \alpha_2\mathcal{L}_0 \tag{10}$$

where α_1 and α_2 are weights, here set to $\alpha_1 = 1.0$ and $\alpha_2 = 0.01$.

We use regularization loss instead of directly using L_1 loss to train δ_i because the difficulty of accurately predicting $E(r)$ is about the same as directly predicting the color of the query coordinates for the neural network. In our network structure, it is difficult for a shallow network to have accurate $E(r)$. Therefore, we use regularization loss with unbalanced values for α_1 and α_2, so that the network can use larger penalties for points with uncertainty lower than loss, while uncertainty higher than loss will be less punished. In this way, the network learns the uncertainty into the upper bound of the complex loss function, so that only a truly certain point can terminate early.

4.2.2 *Multi-Scale Joint Training*

We thus finalise queries with uncertainty lower than ϵ early, and forward points with uncertainty greater than or equal to ϵ to the deeper network. This early termination mechanism can reduce unnecessary calculations, but unfortunately, also brings training difficulties. For a query coordinate (x, y, z, d), the uncertainty may exceed ϵ at some stage, and the coordinate will be sent to a deeper network. However, if the deeper network has not been trained on this coordinate before, it will output an almost random value. This will cause great instability in the loss, affecting the training and may even cause gradient explosion.

To solve this problem, we follow the practice in multi-scale dense networks [44]: each time, all query coordinates are output through all outlets; images with early termination are also output, and their losses are weighted with equal weight during training. Our overall loss function is thus:

$$\mathcal{L} = \sum_{i=1}^{D} \beta_1\mathcal{L}^i_{MSE} + \beta_2 L^i_{unct} \tag{11}$$

where D is current number of stages (also $D-1$ times of network growth), and L^i_{MSE} and L^i_{unct} are the MSE loss and uncertainty loss of the output image of layer i, respectively. β_1 and β_2 are weights, set to $\beta_1 = 1.0$ and $\beta_2 = 0.1$.

4.3 Dynamic Growth

We now explain our adaptive dynamic growth strategy which clusters the uncertain queries at the current stage, and grows deeper networks according to the clustering result.

As shown in Fig. 1, in the initial stage, the network contains only one sub-network Φ_1 which consists of two linear layers. After I_1 iterations of training for the initial network, we sample a number of points in space and calculate their uncertainties. We then cluster those points for which the uncertainty is higher than ϵ; the clustering result determines the growth of the next stage network. To ensure that clustering is simple and controllable, we use the k-means algorithm, which can be replaced by a more efficient clustering algorithm such as methods in [48], with $k \in [2, 4]$. The network grows k branches according to the cluster centers; these are e.g., Φ_2 and Φ_3. Downstream, query points are assigned to the branch with the closest cluster center.

When scenes become complicated, NeRF has to deepen its network, while we can simply add further branches to get the same result. There are two reasons why we split the grown network into two. First, splitting the points will reduce the complexity of the network, otherwise, a deeper network is required for all points. Second, each child-network is only responsible for part of the scene independently, making it more effective and adaptive. Ablation experiment named "no branching" in Table 5 shows that our full model are better than using a single branch.

The growth-based network is trained for several iterations, and then clustered and grown. This process can be repeated until the uncertainty of most points is less than ϵ_{grow}. In order to finish training in a reasonable amount of time, we specify that Recursive NeRF grows 3 times in total. The value of k used for each growth step can be different, but by default, we set k for each to be 2. Recursive-NeRF has the ability to adaptively grow according to the complexity of the scene. We define the uncertainty ratio:

$$\mathcal{R} = \frac{1}{M}\sum_{i=1}^{M} a_i,\ a_i = \begin{cases} 1, & \delta_i > \epsilon_{grow} \\ 0, & \delta_i \le \epsilon_{grow} \end{cases} \tag{12}$$

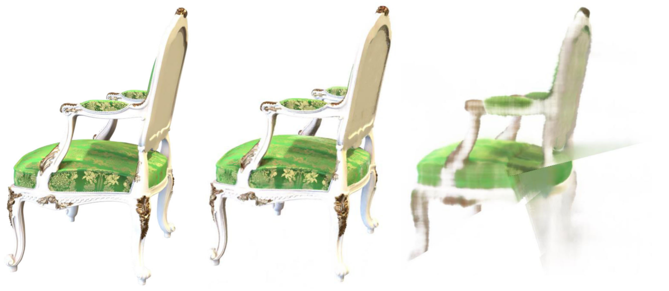
Fig. 4. Alpha linear initialization comparison. Left: ground truth. Middle: our full model's result. Right: result of model without alpha linear initialization.

where M is the number of points sampled in space while growing, δ_i is the predicted uncertainty, $a_i = 1$ if i-th sample point is uncertain, otherwise, $a_i = 0$. When R is greater than the growth threshold T, we think that the result is not good enough and the network will continue to grow, otherwise we will stop growing. The 2D image memorization experiment in Section 5.6 shows Recursive-NeRF can automatically grow for different times until the quality is high enough for tasks of different resolutions.

During training, sample points can exit at multiple stages, while points will exit only once at a specific stage during inferencing. Sepcifically, points found earlier to be reliable (i.e., its uncertainty is lower than a given threshold) will immediately exit and the remaining deeper network computations will not be executed. We cluster the uncertain points in the current stage and feed them to different child branches with the same structure in the next stage. Which branch is taken depends on the results of clustering. We add residual links between every two Linear layers. In this way, our method enables earlier stages to learn simple and rough structures for reliable points; meanwhile, the later stages are capable of learning more details based on results of earlier stages for the uncertainty points.

Trials show that direct growth of a randomly initialized network results in instability in the staged training, causing the density of some of the grown networks to reduce to 0. As a result the rendered scene can lack pieces, as shown in Fig. 4. The specific structure of the network's Outnet module is shown in Fig. 5, where alpha linear is responsible for decoding features into the density of query points. Our approach to overcoming this problem is to initialize the alpha linear weights of grown sub-network to be same as those of the parent. This enables the density generation network of the subnet to inherit part of the density information of the parent, avoiding this instability.

We show staged clustering results for the Lego model in Fig. 6. It can be seen that the image includes finer and finer detail from the initial to final stage.

4.4 Recursive Rendering

Unlike NeRF which outputs the color and density for all points in the last layer of the network, Recursive-NeRF renders the final image recursively. In the current view, all points whose uncertainty is lower than the threshold at the current stage can render a relatively fuzzy image. All points with uncertainty higher than the threshold enter the next stage network to be further refined and other points of low

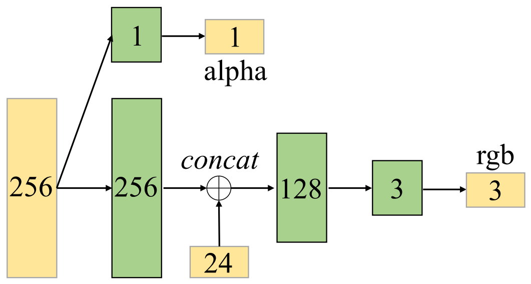

Fig. 5. Left: network structure of OutNet. Green block: fully connected layer. Yellow blocks: input and output variables of the network.

uncertainty can exit from this stage. These points together with ones from all previous stages can render a clear image. Fig. 7 renders images using the points finalised at different stages, these being merged to form the final image at top-left. The uncertainty is implicitly visualized in Fig. 7, where areas of low uncertainty at earlier stages are mainly empty spaces and surfaces with simple structure, such as the floor of the Lego model.

Each input query point $r(t)$ exits from the first branch network in which its uncertainty probability is less than ϵ. We use the color c_i and density σ_i predicted by the branch network to represent $c(r(t))$ and $\sigma(r(t))$ needed by Eq. (1). Then we use Eq. (1) to calculate the color of the query point.

$$\sigma(r(t)) = \sigma_i, i = min\{i|\delta_i < \epsilon \wedge r \in R_i\} \tag{13}$$
$$c(r(t)) = c_i, i = min\{i|\delta_i < \epsilon \wedge r \in R_i\} \tag{14}$$

where R_i is the set of points contained in the ith branch.

5 Experiments and Discussion

In this section, we first evaluate our Recursive-NeRF on different datasets and compare it with state-of-art alternatives. Then we conduct ablation studies to validate the design choices of our approach, including early termination, uncertain point clustering and the branching mechanism.

5.1 Experimental Settings

5.1.1 Deep Learning Framework

All of our experiments were implemented using the Jittor deep learning framework [49]. Jittor supports dynamic graph execution, allowing the neural network to be dynamically changed during each training stage, so is well suited to training our Recursive-NeRF.

5.1.2 Datasets

We evaluated our method on Synthetic-NeRF [14], LLFF [8], Cornell Box dataset [50], Google Earth Studio [51] and Mars image [52]. *Synthetic-NeRF* contains eight man-made objects with complicated geometry and materials. Each object is realistically rendered in 300 views at a resolution of 800×800 pixels. We used the same split into training and testing data as NeRF [14]. *LLFF* is a real-world scenes dataset, and we tested our method on the "fern" scene of this dataset. *Cornell Box* is a relatively simple synthetic scene, mainly composed of boxes. We adopted this dataset to demonstrate the effectiveness of our method. We rendered 400 pictures at a resolution of 800×800 pixels from views uniformly

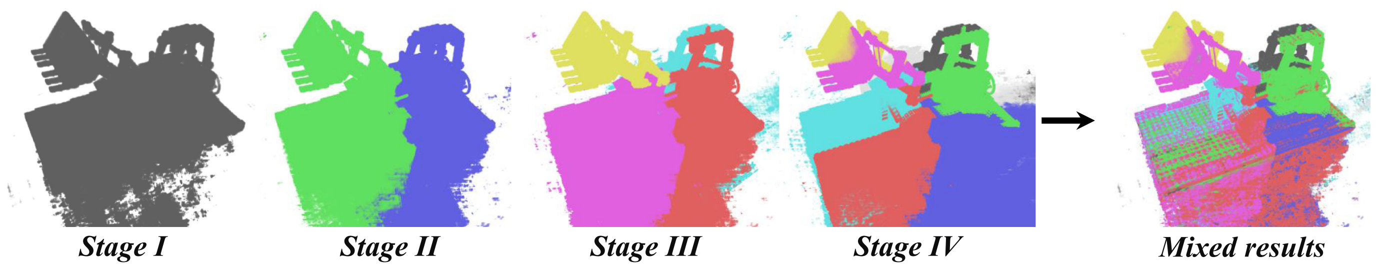

Fig. 6. Scene segmentation at different stages. Left: segmentation at different stages. Right: blocks owning each query point, indicating early termination.

sampled with the camera moving along a spiral curve, and randomly selected 200 pictures as the training set, and the remainder for testing. We take a city-level data from *Google Earth Studio* to demonstrate the capability of Recursive-NeRF in large-scale complex scenes. We followed the method of Xiangli et al. [53] by collecting images taken around a circle at the same altitude over New York City. We collected a total of 413 images with a resolution of 960*540, and uniformly selected 13 images as the test set and 400 images as the training set. We use 8000*8000 resolution image of *Mars* as a dataset for 2D image memorization experiments.

5.2 Results on Synthetic data

5.2.1 Qualitative Comparison

For the Synthetic dataset, the batch size was set to 4096 in our training stage, 64 and 192 sampling points were used for the coarse and fine networks respectively, and each model was trained for 300k iterations, all as in the NeRF paper. Our network underwent four stages of training, as it was grown three times. The initial and three grown networks had 2, 2, 4, and 4 linear layers respectively. We used Adam as the optimizer and set a learning rate with an initial value of 5×10^{-4} and 10 times exponential learning rate decay after 250k iterations. During rendering, we perform a preprocessing that reorders the sample points to make the memory access more continuous. We compared our approach to several current state-of-the-art methods: SRN [11], NV [6], LLFF [8], and NeRF [14].

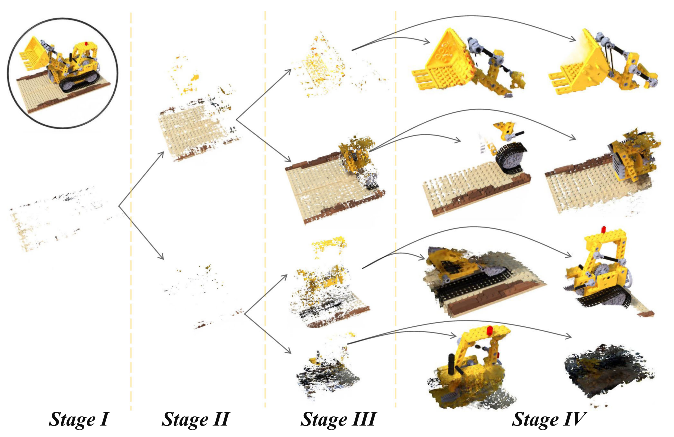

Fig. 7. Recursive rendering. Various query points are finalised early in different stages, and finally all points are aggregated to form the rendered image at the upper left.

We show different rendering results on Synthetic-NeRF dataset in Fig. 8. Comparative results on the Cornell Box dataset are shown in Fig. 9. Our method generates much clearer local details than the baseline NeRF on the Synthetic-NeRF dataset and achieves comparable results on Cornell Box dataset. Fig. 10 galleries more results Recursive-NeRF renders at other viewpoints on both Synthetic-NeRF and Cornell Box dataset.

5.2.2 Quantitative Comparison

We also used PSNR and SSIM to enable a quantitative comparison of the results (higher is better), as well as LPIPS [54] (lower is better). Results for the Synthetic-NeRF dataset are shown in Table 1, and demonstrate that our method can perform well on the general dataset. Results for the Cornell Box dataset are shown in Table 2. We have reduced the amount of calculation by about 2/3, with only a slight loss of accuracy.

5.2.3 Speed Comparison

We show the number of million floating point operations (MFLOPs) and inference time (in seconds) in both Tables 1 and 2. Note that the parameters of NeRF are $2.4M$ and the parameters of Recursive-NeRF are $14.6M$. Although we have more parameters than NeRF, our network requires fewer operations for both simple and complex scenes, with greater speed improvement for the simple scenes as to be expected. The maximum depth of our network can reach 12 layers, which is deeper than NeRF's 8 layers. Using our early termination strategy, we are able to finalise a large number of simple query points in the shallow part of the network, leaving the deep network to focus on the complex information, thereby providing better results. Although the number of deepest layers in our network is greater, our early termination strategy results in Recursive-NeRF reducing NeRF's computational effort by 37% and 32.36% less time.

5.2.4 Distribution of Sample Termination

The distribution of sample point termination shows that the ratios of points terminated in the 2nd, 4th, 8th, and 12th layers are 45.3%, 27.9%, 7.2%, and 19.6%, respectively. The sample points will go through 4.95 layers on average in our network, while 8 layers are required in NeRF, and only 19.6% of points go through a deeper network than in NeRF. Thus, our adaptive approach can effectively reduce the computation according to the learned uncertainty: it is more than a simply deeper NeRF.

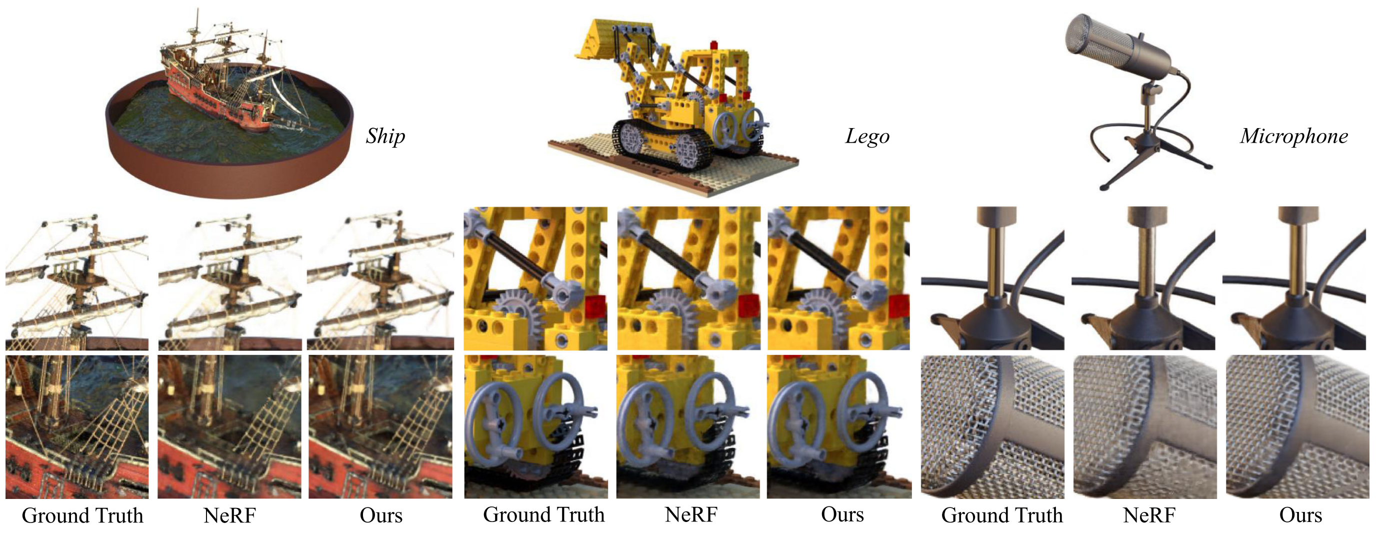

Fig. 8. Qualitative results. Top: scene. Middle, below: Two close-ups of the scene. We show the ground truth, the results of NeRF rendering, and of our method in turn.

5.3 Comparison With DeRF on Real-World Data

We compare our method with DeRF [43] on the “fern” scene of the real-world scene dataset LLFF. We follow the setting of DeRF by using a batch size of 512, 128 samples per ray, 8 layers Linear and training for 300k iterations. We conducted experiments in a similar manner to the DeRF paper, setting the number of heads to 1, 4, 8, and the number of hidden units to 64, 96, 128, 192, 256, and we used the results reported in the DeRF paper for comparison.

Table 3 shows the comparison of our results with DeRF and Fig. 11 shows the curve comparison of PSNR with MFLOPs under different heads. It can be seen that we mostly achieve better performance with lower FLOPs under the same head and unit settings, and the larger the number of heads, the more significant the advantage we have over DeRF. Although we performed slightly worse than DeRF in some experiments, the FLOPs of these results were significantly lower than DeRF. A representative example is, in the 8-head experiment our method at 128-unit gets better performance than DeRF at 256-unit with 83.6% fewer FLOPs. Fig. 12 shows a visual comparison with DeRF at 8 heads, and 256 hidden units. It can be seen that our method produces sharper details.

Similar to ours, DeRF also decomposes the scene, but it uses the uniformity of the sampling point division as a loss and optimizes it through a neural network. This method has two disadvantages. First, the distribution of sampling points in the scene is not consistent with the distribution of complex parts. Simply dividing the scene evenly according to the sampling points may lead to the fact that different head would have to represent parts of differnt complexity. This can lead to insufficient learning of complex parts and

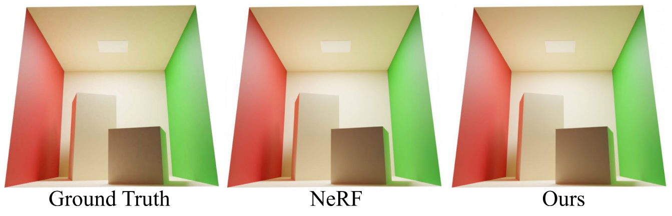

Fig. 9. Qualitative comparison for Cornell Box. Left: original image. Middle: NeRF’s result. Right: our result.

Fig. 10. Further rendering results from Recursive-NeRF.

TABLE 1
Quantitative Comparison on the Synthetic-NeRF Dataset [14]

Method	PSNR↑	SSIM↑	LPIPS↓	MFLOPs↓	Time↓
SRN [11]	22.26	0.846	0.170	-	-
NV [6]	26.05	0.893	0.160	-	-
LLFF [8]	24.88	0.911	0.114	-	-
NeRF [14]	31.01	0.947	0.081	1.18	26.14
Ours	**31.34**	**0.953**	**0.052**	**0.75**	**17.68**

TABLE 2
Quantitative Comparison on Cornell Box

Method	PSNR↑	SSIM↑	LPIPS↓	MFLOPs↓	Time↓
NeRF	49.237	0.996	0.015	1.18	26.14
Ours	48.010	0.996	0.010	0.40	10.82

TABLE 3
Quantitative Comparison With DeRF on the "FERN" Scene of LLFF Dataset [14]

		1 Head				4 Heads				8 Heads			
	Unit	PSNR↑	SSIM↑	LPIPS↓	MFLOPs↓	PSNR↑	SSIM↑	LPIPS↓	MFLOPs↓	PSNR↑	SSIM↑	LPIPS↓	MFLOPs↓
DeRF	64	22.15	0.61	0.52	0.149	23.08	0.66	0.46	0.201	23.34	0.67	0.44	0.216
	96	22.87	0.64	0.48	0.267	23.56	0.68	0.43	0.299	23.77	0.70	0.41	0.387
	128	23.20	0.66	0.46	0.353	23.74	0.69	0.42	0.472	23.82	0.70	0.40	0.573
	192	23.83	0.70	0.42	0.854	24.01	0.71	0.39	0.894	24.17	0.72	0.37	1.138
	256	23.88	0.70	0.41	1.186	24.22	0.72	0.38	1.650	24.26	0.73	0.36	1.913
Ours	64	21.97	0.6	0.56	**0.088**	22.69	0.65	0.46	**0.088**	23.1	0.67	0.41	**0.088**
	96	22.72	0.64	0.47	0.183	23.48	0.69	0.37	0.183	23.93	0.72	0.32	0.183
	128	23.36	0.68	0.4	0.314	23.95	0.73	0.31	0.314	24.39	0.75	0.27	0.313
	192	24.1	0.73	0.32	0.679	24.7	0.77	0.24	0.677	24.96	0.79	0.21	0.674
	256	**24.61**	**0.76**	**0.26**	1.185	**25.05**	**0.79**	**0.2**	1.182	**25.18**	**0.8**	**0.18**	1.172

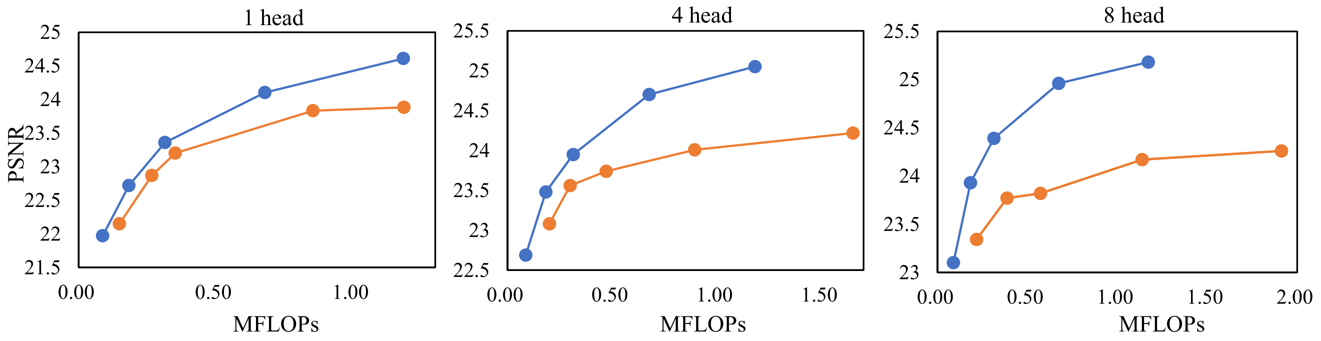

Fig. 11. Visualization of our performance (colored in blue) compared with DeRF's (colored in orange) at 1, 4, and 8 heads.

affect the final performance. While Recursive-NeRF can divide the scene more reasonably and adaptively through the distribution of uncertain points, and thus can represent parts of similar complexity with different netowrks. Second, by optimizing the scene division by loss, it is necessary to ensure that the division is differentiable. So there must be overlaps between the heads, which will lead to a more number of the heads and bring more additional computation. This can be observed from Table 3 that the MFLOPs of DeRF increase significantly as the number of heads increases. Head increase in Recursive-NeRF will not cause increase in computation. On the contrary, since there are more heads to share the amount of computation, combined with our early termination strategy, FLOPs will decrease slightly.

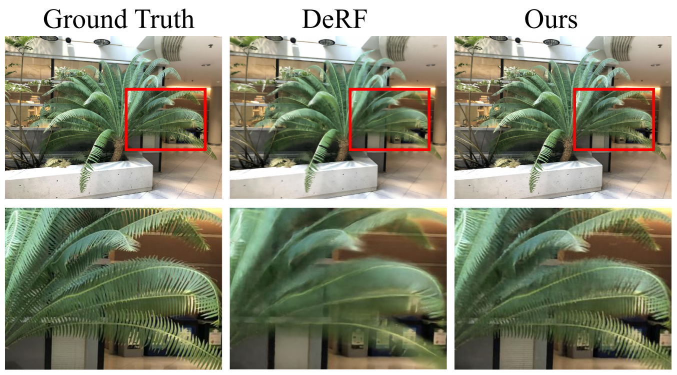

Fig. 12. Visual comparison with DeRF on the "Fern" scene. Left: original image; middle: DeRF's result; right: our result.

5.4 Performance on Large Scale Scene

We compare with NeRF on the large-scene city-level dataset to demonstrate the advantages of our method in complex and large-scale scenes. We trained all methods for 900k iterations. The batch size is set to 512. To compare the performance of NeRF with our method under similar FLOPs and parameters, we introduce three variants of NeRF: NeRF(SF), having similar FLOPs with our method with the number of layers extended to 12, NeRF(SPW) and NeRF(SPD) having similar parameters to our method with wider and deeper networks respectively. Specifically, NeRF(SPW) has 12 layers and 768 hidden units; NeRF (SPD) has 110 layers and 256 hidden units. To make NeRF (SPD) easier to converge, we add residual connections between every two consecutive layers. Quantitative results are shown in Table 4.

Compared to NeRF, we significantly improve PSNR by 3.14. Comparison to NeRF(SF) shows that we improve PSNR by 2.79 with basically similar FLOPs. For NeRF(SPW) and NeRF(SPD) with a similar amount of parameters, our method still achieves better results while reducing FLOPs by about 88%. We also show some qualitative comparisons in Fig. 13. It can be seen that our method generates finer details than NeRF. This is because we predict the uncertainty of each point in the scene, and for complex areas, more sub-networks will be subdivided to predict details. The shallow layers of the network can learn the rough outline, and each piece of local information has a separate subnetwork to learn it, so more local details can be obtained with our Recursive-NeRF.

TABLE 4
Quantitative Comparison on Large Scale Scene

Method	PSNR↑	SSIM↑	LPIPS↓	MFLOPs↓	Params↓
NeRF	18.17	0.43	0.52	**1.19**	**2.1**
NeRF(SF)	18.52	0.45	0.49	1.71	3.1
NeRF(SPW)	20.10	0.62	0.33	14.51	27.2
NeRF(SPD)	20.21	0.63	0.32	14.56	27.6
Ours	**21.31**	**0.69**	**0.28**	1.71	27.9

TABLE 5
Ablation Experiment on our Proposed Method

Method	PSNR↑	SSIM↑	LPIPS↓	MFLOPs↓
No classification	32.697	0.966	0.035	0.65
No branch	32.374	0.962	0.040	0.70
No early termination	33.118	0.970	0.032	1.71
Full method	32.900	0.967	0.033	0.70

5.5 Ablation Study

We conducted ablation experiments as described below; the results are shown in Table 5. A qualitative comparison of results on the Lego dataset is shown in Fig. 14.

5.5.1 *Effect of Early Termination*

Early termination is a key part of our method. We trained our model without early termination, which means all sampling points leave at the last exit. It can be seen that the amount of computation increased significantly. The early termination mechanism greatly improves performance and only causes a very minor degradation of the results.

5.5.2 *Effect of Uncertain Point Classification*

We use K-means to divide the scene into blocks. As an alternative, we randomly divided the scene into blocks and compared the outcomes. Fig. 15 shows the results. Without clustering, many blocks contain many discontinuous parts, and the block size is also uneven, reducing the quality of the final image.

5.5.3 *Effect of Branching*

To demonstrate the effectiveness of our network block structure, we compare the results with a chain structure of the same depth network (No branch in Table 5). The chain network had a $\#2FC-\#2FC-\#4FC-\#4FC$ structure where $\#iFC$ represents a fully connected layer with i-layers. The query points could terminate early from the 2nd, 4th, 8th, and 12th levels. The branching strategy divides complex parts into different branch networks for learning, allowing the network to decouple complex scenarios and conduct targeted learning. Thus branch strategy brings a significant increase in quality.

5.5.4 *Effect of Uncertainty Threshold*

To demonstrate the effect of uncertainty threshold, we test the results under different uncertainty thresholds on the "fern" dataset. The results are shown in Table 6. From Fig. 16 it can be seen that as the uncertainty threshold decreases, more points will enter the deeper network for prediction; meanwhile, the PSNR gets better, also demanding more compuation effort.

5.6 Adaptive Growth

We tested the adaptive growth ability of Recursive-NeRF through the 2D image memorization experiment. We take the original image with a resolution of 8000*8000 and images downscaled to resolutions of 4000*4000, 2000*2000, 1000*1000 by anti-aliased resampling as different complex tasks. The larger the image resolution, the higher the complexity. Since the original resolution image is too large, we do not count LPIPS. As stated in Section 4.3, when the uncertainty ratio R is larger than the growth threshold T, we indicate that the quality is not high enough, and

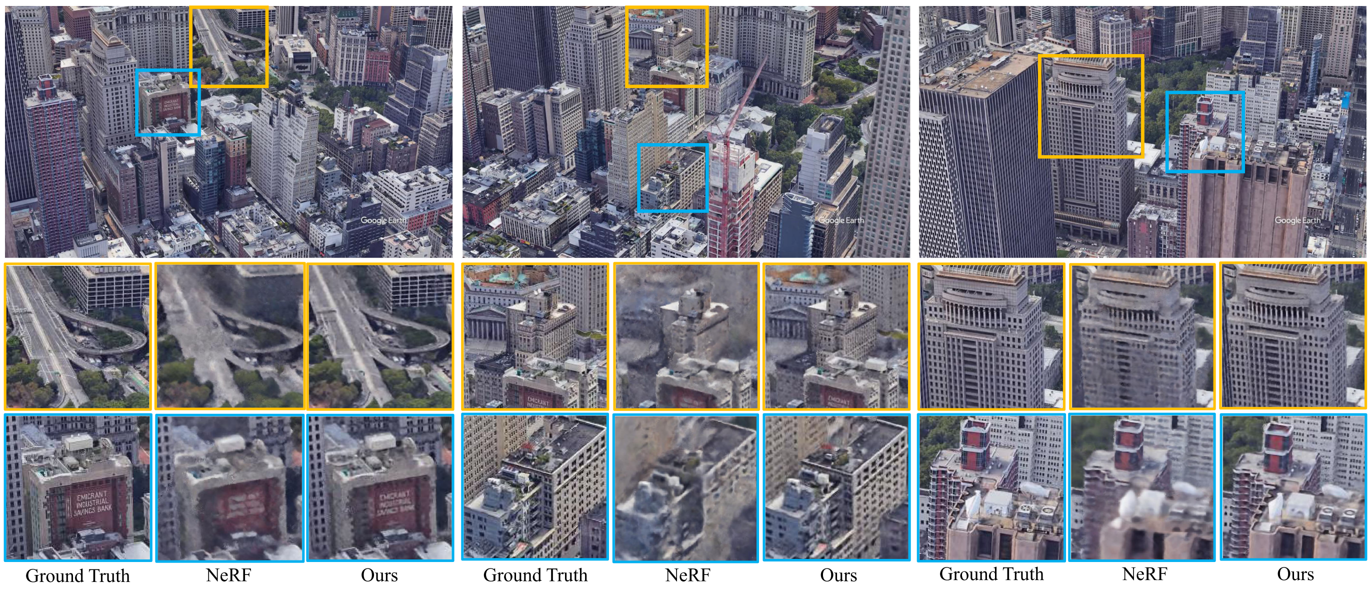

Fig. 13. Qualitative comparison for a large-scale scene. For three selected views, we also show close-up results of groundtruth, NeRF, and our method.

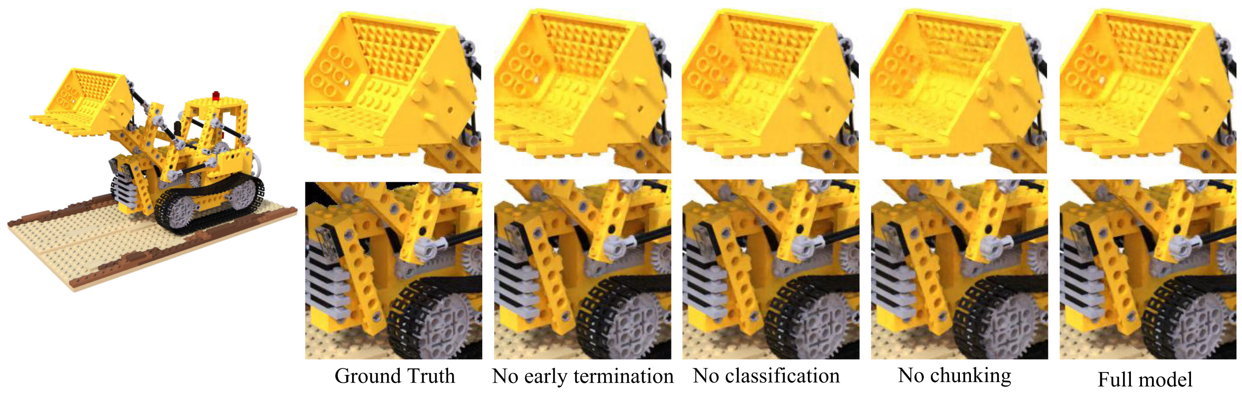

Fig. 14. Qualitative comparison of ablation experiment.

Recursive-NeRF will grow. Here we set T to 0.03, and set the maximum times of growths to 3.

Table 7 shows the results at different resolutions and the corresponding growth times. Fig. 17 shows the resulting images at different resolutions. It can be seen that as the difficulty of the task increases, Recursive-NeRF can adaptively grow until the quality is high enough. The adaptive growth mechanism enables Recursive-NeRF to achieve similar performance for tasks of different difficulty, while simpler tasks can save more computation.

5.7 Limitations and Future Works

Recursive-NeRF rendering for large-scale scenes is still challenging. Inaccurate camera position and motion blur will degrade the performance of both Recursive-NeRF and NeRF, which restricts their application to real scenes. Although the speed of Recursive-NeRF is better than that of the original NeRF, it is still not adequate for real-time use. We hope to further improve the performance of Recursive-NeRF, to adapt it to more complex scenes.

Although we have demonstrated the performance of Recursive-NeRF in large-scale complex scenes, there is still room for improvement. Introducing a more effective sampling strategy [18], [38] and better position encoding [36] will further improve the performance of Recursive-NeRF. Second, learnable clustering [55] of uncertain points could be exploited to make the branched networks more adaptive to complexity of parts of the underlying scene. Lastly, our method can leverage geometric priors learned from geometric learning [56], [57] to improve training performance with a small number of views.

6 Conclusion

In this paper, we have proposed the idea of adaptively modeling parts of a scene with different complexity using neural networks of different representation capability, analogous to level of detail. We have used it to construct a dynamically growing neural network for novel view synthesis, called Recursive-NeRF. It extends basic NeRF by additionally predicting uncertainty for the results, and uses it to dynamically branch new, more powerful neural networks to represent more uncertain regions, allowing it to efficiently learn implicit geometric and appearance representations for complicated scenes. Our experiments have demonstrated the effectiveness of our method and show that, compared to

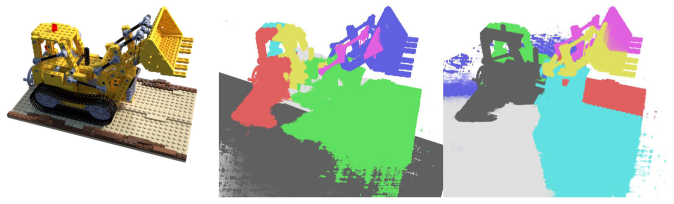

Fig. 15. Scene segmentation results. Left: original Lego scene. Middle: result using random division (no clustering). Right: result of clustering uncertain points using K-means.

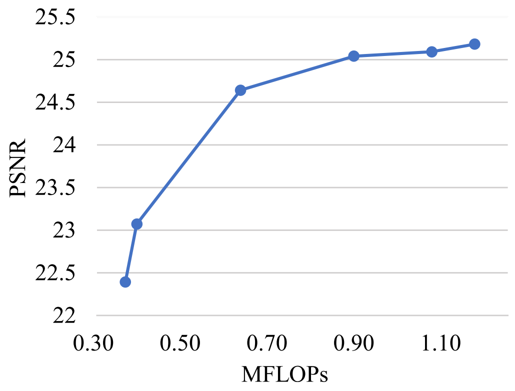

Fig. 16. PSNR follows the trend of MFLOPs under different thresholds.

TABLE 6
Effect of Uncertainty Threshold

Threshold	PSNR↑	SSIM↑	LPIPS↓	MFLOPs↓
0.001	**25.18**	**0.800**	0.184	1.17
0.005	25.09	0.799	**0.182**	1.07
0.01	25.04	0.793	0.193	0.89
0.05	24.64	0.751	0.285	0.63
0.1	23.07	0.655	0.452	0.39
0.5	22.39	0.623	0.526	**0.37**

TABLE 7
Adaptive Growth of our Network in Terms of the Complexity of Image Memorization

Resolution	PSNR↑	SSIM↑	MFLOPs↓	Growth times
1000	34.29	0.8898	0.37	0
2000	34.81	0.8806	0.47	1
4000	35.03	0.9113	0.66	2
8000	32.21	0.8650	0.88	3

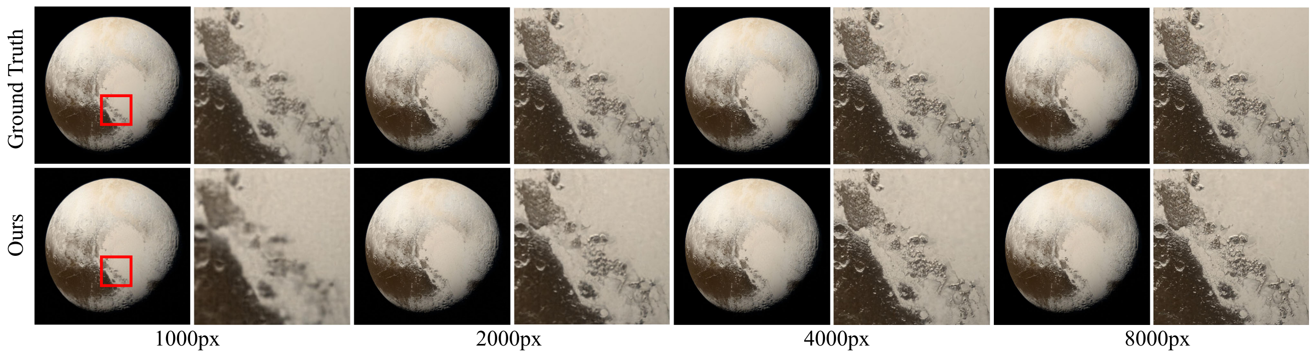

Fig. 17. Image memorization results and a zoom-in at different resolution.

NeRF, Recursive-NeRF can generate more photorealistic views in a more efficient computation.

Acknowledgments

We would like to thank Guo-Ye Yang for his kindly help in experimentation and Prof. Ralph R. Martin for his help in writing.

References

[1] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob, “Mitsuba 2: A retargetable forward and inverse renderer,” *ACM Trans. Graph.*, vol. 38, no. 6, pp. 203:1–203:17, 2019.
[2] S. Liu, W. Chen, T. Li, and H. Li, “Soft rasterizer: A differentiable renderer for image-based 3d reasoning,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 7707–7716.
[3] T. Li, M. Aittala, F. Durand, and J. Lehtinen, “Differentiable monte carlo ray tracing through edge sampling,” *ACM Trans. Graph.*, vol. 37, no. 6, pp. 222:1–222:11, 2018.
[4] C. Buehler, M. Bosse, L. McMillan, S. J. Gortler, and M. F. Cohen, “Unstructured lumigraph rendering,” in *Proc. 28th Annu. Conf. Comput. Graph. Interactive Techn.*, 2001, pp. 425–432.
[5] S. M. Seitz and C. R. Dyer, “Photorealistic scene reconstruction by voxel coloring,” *Int. J. Comput. Vis.*, vol. 35, no. 2, pp. 151–173, 1999.
[6] S. Lombardi, T. Simon, J. M. Saragih, G. Schwartz, A. M. Lehrmann, and Y. Sheikh, “Neural volumes: Learning dynamic renderable volumes from images,” *ACM Trans. Graph.*, vol. 38, no. 4, pp. 65:1–65:14, 2019.
[7] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhöfer, “Deepvoxels: Learning persistent 3D feature embeddings,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2437–2446.
[8] B. Mildenhall et al., “Local light field fusion: Practical view synthesis with prescriptive sampling guidelines,” *ACM Trans. Graph.*, vol. 38, no. 4, pp. 29:1–29:14, 2019.
[9] E. Penner and L. Zhang, “Soft 3d reconstruction for view synthesis,” *ACM Trans. Graph.*, vol. 36, no. 6, pp. 235:1–235:11, 2017.
[10] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely, “Stereo magnification: Learning view synthesis using multiplane images,” *ACM Trans. Graph.*, vol. 37, no. 4, pp. 65:1–65:12, 2018.
[11] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, “Scene representation networks: Continuous 3D-structure-aware neural scene representations,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 1119–1130.
[12] M. Niemeyer, L. M. Mescheder, M. Oechsle, and A. Geiger, “Differentiable volumetric rendering: Learning implicit 3D representations without 3d supervision,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 3501–3512.
[13] W. Chen et al., “Learning to predict 3D objects with an interpolation-based differentiable renderer,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 9605–9616.
[14] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “NeRF: Representing scenes as neural radiance fields for view synthesis,” in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 405–421.
[15] K. Zhang, G. Riegler, N. Snavely, and V. Koltun, “NeRF++: Analyzing and improving neural radiance fields,” 2020, *arXiv:2010.07492*.
[16] K. Park et al., “Nerfies: Deformable neural radiance fields,” in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 5865–5874.
[17] P. P. Srinivasan, B. Deng, X. Zhang, M. Tancik, B. Mildenhall, and J. T. Barron, “NeRV: Neural reflectance and visibility fields for relighting and view synthesis,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 7495–7504.
[18] L. Liu, J. Gu, K. Z. Lin, T. Chua, and C. Theobalt, “Neural sparse voxel fields,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 15651–15663.
[19] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner, “Chapter 1 - Introduction,” in *Level of Detail for 3D Graphics*, ser. The Morgan Kaufmann Series in Computer Graphics, D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner, Eds. San Mateo, CA, USA: Morgan Kaufmann, 2003, pp. 3–ii.
[20] C. Zhang and T. Chen, “A survey on image-based rendering - representation, sampling and compression,” *Signal Process. Image Commun.*, vol. 19, no. 1, pp. 1–28, 2004.
[21] A. Tewari et al., “State of the art on neural rendering,” *Comput. Graph. Forum*, vol. 39, no. 2, pp. 701–727, 2020.
[22] H. Kato et al., “Differentiable rendering: A survey,” 2020, *arXiv:2006.12057*.
[23] C. M. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, and T. A. Funkhouser, “Local implicit grid representations for 3D scenes,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 6000–6009.
[24] J. J. Park, P. Florence, J. Straub, R. A. Newcombe, and S. Lovegrove, “DeepSDF: Learning continuous signed distance functions for shape representation,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 165–174.
[25] L. M. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, “Occupancy networks: Learning 3D reconstruction in function space,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4460–4470.
[26] K. Genova, F. Cole, A. Sud, A. Sarna, and T. A. Funkhouser, “Local deep implicit functions for 3D shape,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 4856–4865.
[27] H. Kato, Y. Ushiku, and T. Harada, “Neural 3D mesh renderer,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 3907–3916.
[28] K. Schwarz, Y. Liao, M. Niemeyer, and A. Geiger, “GRAF: Generative radiance fields for 3D-aware image synthesis,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 20154–20166.
[29] S. Ma, Q. Shen, Q. Hou, Z. Ren, and K. Zhou, “Neural compositing for real-time augmented reality rendering in low-frequency lighting environments,” *Sci. China Inf. Sci.*, vol. 64, no. 2, pp. 1–15, 2021.
[30] A. Dosovitskiy et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” in *Proc. Int. Conf. Learn. Representations*, 2021, pp. 1–22.
[31] M.-H. Guo, C.-Z. Lu, Z.-N. Liu, M.-M. Cheng, and S.-M. Hu, “Visual attention network,” 2022, *arXiv:2202.09741*.
[32] M.-H. Guo et al., “Attention mechanisms in computer vision: A survey,” *Comput. Vis. Media*, vol. 8, no. 3, pp. 331–368, 2022.
[33] Y. Xu et al., “Transformers in computational visual media: A survey,” *Comput. Vis. Media*, vol. 8, no. 1, pp. 33–62, 2022.
[34] J. Reizenstein, R. Shapovalov, P. Henzler, L. Sbordone, P. Labatut, and D. Novotny, “Common objects in 3D: Large-scale learning and evaluation of real-life 3d category reconstruction,” in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 10901–10911.
[35] K. Park et al., “HyperNeRF: A higher-dimensional representation for topologically varying neural radiance fields,” *ACM Trans. Graph.*, vol. 40, no. 6, pp. 1–12, 2021.

[36] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields," in *Proc. IEEE/CVF Int. Conf. Comput. Vis*, 2021, pp. 5855–5864.

[37] P. Ren, J. Wang, M. Gong, S. Lin, X. Tong, and B. Guo, "Global illumination with radiance regression functions," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 1–12, 2013.

[38] T. Neff et al., "DoNeRF: Towards real-time rendering of compact neural radiance fields using depth oracle networks," in *Computers Graphics Forum*, vol. 40, no. 4. Hoboken, NJ, USA: Wiley, 2021, pp. 45–59.

[39] D. B. Lindell, J. N. Martel, and G. Wetzstein, "Autoint: Automatic integration for fast neural volume rendering," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 14 556–14 565.

[40] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "FastNeRF: High-fidelity neural rendering at 200FPS," in *Proc. IEEE/CVF Int. Conf. Comput. Vis*, 2021, pp. 14 346–14 355.

[41] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "Plenoctrees for real-time rendering of neural radiance fields," in *Proc. IEEE/CVF Int. Conf. Comput. Vis*, 2021, pp. 5752–5761.

[42] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, "Baking neural radiance fields for real-time view synthesis," in *Proc. IEEE/CVF Int. Conf. Comput. Vis*, 2021, pp. 5875–5884.

[43] D. Rebain, W. Jiang, S. Yazdani, K. Li, K. M. Yi, and A. Tagliasacchi, "DeRF: Decomposed radiance fields," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 14 153–14 161.

[44] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–14.

[45] J. Yu, L. Yang, N. Xu, J. Yang, and T. S. Huang, "Slimmable neural networks," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–12.

[46] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 1803–1811.

[47] J. Yu and T. Huang, "AUTOSLIM: Towards one-shot architecture search for channel numbers," 2019, *arXiv:1903.11728*.

[48] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Ann. Data Sci.*, vol. 2, no. 2, pp. 165–193, 2015.

[49] S.-M. Hu, D. Liang, G.-Y. Yang, G.-W. Yang, and W.-Y. Zhou, "Jittor: A novel deep learning framework with meta-operators and unified graph execution," *Sci. China Inf. Sci.*, vol. 63, no. 12, pp. 222103:1–222103:21, 2020.

[50] Cornell.edu. The cornell box. 2021. [Online]. Available: http://www.graphics.cornell.edu/online/box/

[51] Google earth studio, 2018. [Online]. Available: https://earth.google.com/studio/

[52] J. N. P. Martel, D. B. Lindell, C. Z. Lin, E. R. Chan, M. Monteiro, and G. Wetzstein, "Acorn: Adaptive coordinate networks for neural scene representation," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–13, 2021.

[53] Y. Xiangli et al., "BungeeNeRF: Progressive neural radiance field for extreme multi-scale scene rendering," in *Eur. Conf. Comput. Vis.*, 2022, pp. 1–17.

[54] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 586–595.

[55] F. Williams, J. Parent-Lévesque, D. Nowrouzezahrai, D. Panozzo, K. M. Yi, and A. Tagliasacchi, "VoronoiNet: General functional approximators with local support," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2020, pp. 1069–1073.

[56] Y.-P. Xiao, Y.-K. Lai, F.-L. Zhang, C. Li, and L. Gao, "A survey on deep geometry learning: From a representation perspective," *Comput. Vis. Media*, vol. 6, no. 2, pp. 113–133, 2020.

[57] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "PCT: Point cloud transformer," *Comput. Vis. Media*, vol. 7, no. 2, pp. 187–199, 2021.

Guo-Wei Yang received the BS degree in computer science and technology, in 2019. He is currently working toward the PhD degree in the Department of Computer Science and Technology, Tsinghua University. His research interests include computer graphics, neural rendering, and computer vision.

Wen-Yang Zhou is currently working toward the PhD degree in the Department of Computer Science and Technology, Tsinghua University, Beijing. His research interests include computer graphics, image analysis, and computer vision.

Hao-Yang Peng is currently working toward the graduate degree with Tsinghua University. His research interests include computer graphics and computer vision.

Dun Liang received the BS and PhD degrees in computer science and technology, in 2016 and 2021, respectively. He is a research assistant in the Department of Computer Science and Technology with Tsinghua University. His research interests include computer graphics, visual media learning and high-performance computing.

Tai-Jiang Mu received the BS and PhD degrees in 2011 and 2016, respectively. He is currently an assistant researcher in the Department of Computer Science and Technology, Tsinghua University. His research interests include computer graphics, visual media learning and image processing. He has published more than 20 paper in journals and refereed conference, such as *ACM Transactions on Graphics*, *IEEE Transactions on Visualization and Computer Graphics*, *the Visual Computer*, *Graphical Models*, *Computational Visual Media*, CVPR, ICRA, etc.

Shi-Min Hu (Senior Member, IEEE) received the PhD degree from Zhejiang University, in 1996. He is current a professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include digital geometry processing, video processing, rendering, computer animation, and computer-aided geometric design. He has published more than 100 papers in journals and refereed conferences. He is the Editor-in-Chief of *Computational Visual Media*, and on editorial boards of several journals, including *Computer Aided Design and Computer & Graphics*.