# Recursive-NeRF: An Efficient and Dynamically Growing NeRF

Guo-Wei Yang, Wen-Yang Zhou, Hao-Yang Peng, Dun Liang, Tai-Jiang Mu, 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 three public datasets 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],

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  $800 \times 800$  pixels). Previous work [18] has improved NeRF's rendering speed by sampling more carefully using a sparse set of voxels, and

 G.-W. Yang, W.-Y. Zhou, H.-Y. Peng, D. Liang, T.-J. Mu and S.-M. Hu are with the BNRist, Tsinghua University, Beijing 100084, China. Shi-Min Hu is the corresponding author. E-mail: shimin@tsinghua.edu.cn avoiding evaluations on empty voxels.

Let us consider some drawbacks of NeRF's rendering process. Firstly, 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. Secondly, 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 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,

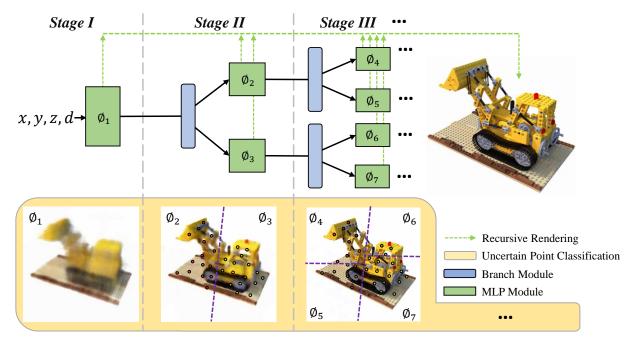


Fig. 1: Pipeline of Recursive-NeRF. Given a position (x, y, z) and viewing direction d, the initial network  $\Phi_1$  outputs color  $c_1$ , density  $\sigma_1$ , and uncertainty  $\delta_1$ . All uncertain points are divided into several categories, and then  $\Phi_1$  dynamically grows several branch 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.

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-theart 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 groundtruth 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.

However, these NeRF-based methods need a large number of samples in the rendering line of sight, so are slow. NSVF [18] focuses on the sampling strategy and introduces a sparse voxel octree on the basis of NeRF; speed is improved by avoiding calculating integrals in empty voxels. Lindell et al. [29] 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. It is complementary to NSVF [18], and could be combined for further speed.

Recently, a series of good speed improvements have been achieved by caching neural network results [30], [31], but these methods will bring additional memory consumption. In the future, we will consider researching ways to improve rendering speed without additional memory consumption.

## 2.2 Dynamic Neural networks

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

Multi-scale dense networks [32] train multiple classifiers according to different resource demands and adaptively apply them during testing. [33] proposes switchable batch normalization and slimmable neural networks, which can adjust width according to device resources. [34] extends this idea to execute at arbitrary width. [35] 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

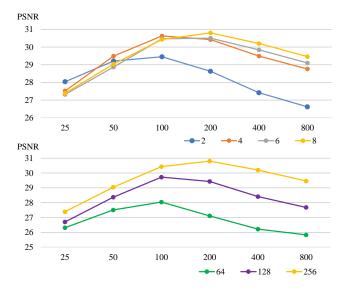


Fig. 2: Correlation between parameters and scene complexity. Different curves in the top and bottom plots represent various network depths and widths, respectively. The horizontal axis is the resolution of the square image, representing the complexity of the model. The vertical axis is the PSNR of the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

the color C(r) of a pixel in an image by integrating the ray from the camera to the pixel:

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

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

c and  $\sigma$  are directional emitted color and volume density which are calculated via an MLP network  $F_{\theta}$ :

$$F_{\theta}: (\mathbf{x}, \mathbf{d}) \to (c, \sigma)$$
 (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, image size is a proxy for complexity of the scene. It can be seen that as the complexity of the scene increases, the PSNR first increases and then decreases. It is because when the scene is too simple, there is too little information for learning, while the scene will be beyond the capability of the network as it gets too complicated. It can be seen from Fig. 2 that when the scene is relatively simple, the representational capabilities of different networks are similar. 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.

#### 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  $\sigma_i$ , uncertainty  $\delta_i$  and a latent vector  $y_i$ :

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

where  $F_{\Phi_i}$  represents the *i*th subnetwork.  $F_{\Phi_1}$  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 Fig. 1 and 3, sub-network  $F_{\Phi_i}$  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  $\delta_i$  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  $F_{\Phi_i}$ . The Out module is responsible for decoding features into  $c_i$  and  $\sigma_i$ .

# 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

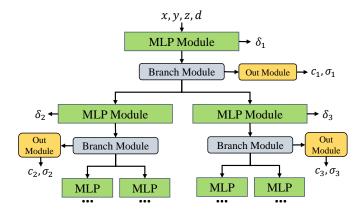


Fig. 3: Network architecture of Recursive-NeRF. For every query (x,y,z,d), the network predicts an uncertainty  $\delta$  used to decide if the query should be finalised early. If so, it will enter the OutNet to predict its color c and density  $\sigma$ . If not, the point split unit determines which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

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  $F_{\Phi_i}$  to calculate the uncertainty  $\delta_i$ . We use the squared error of the pixel to supervise  $\delta_i$ , the intent being that if a pixel has large error,  $\delta_i$  of the sample points that generate the uncertainty associated to the sample points should also be large. Therefore, we punish  $\delta_i$  less than the squared error:

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



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

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

where E(r) is the squared error of ray r,  $c_i$  is the sample point of ray r, and  $\delta_{c_i}$  is the predicted uncertainty for query coordinates  $c_i$ .

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

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

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  $\alpha_1$  and  $\alpha_2$  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  $\delta_i$  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  $\alpha_1$  and  $\alpha_2$ , 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  $\epsilon$  early, and forward points with uncertainty greater than or equal to  $\epsilon$  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  $\epsilon$  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 multiscale dense networks [32]: each time, all query coordinates are output through all outlets; images with early termination are also output, and their losses are weighted with

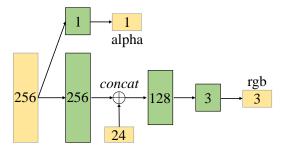


Fig. 5: Left: network structure of OutNet. Green block: fully connected layer. Yellow blocks: input and output variables of the network.

equal weight during training. Our overall loss function is thus:

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

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.  $\beta_1$  and  $\beta_2$  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  $\Phi_1$  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  $\epsilon$ ; 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 [36], with  $k \in [2,4]$ . The network grows k branches according to the cluster centers; these are e.g.  $\Phi_2$  and  $\Phi_3$ . 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. Firstly, splitting the points will reduce the complexity of the network, otherwise, a deeper network is required for all points. Secondly, 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 3 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 successively until the uncertainty of most points is less than  $\epsilon$ . 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.

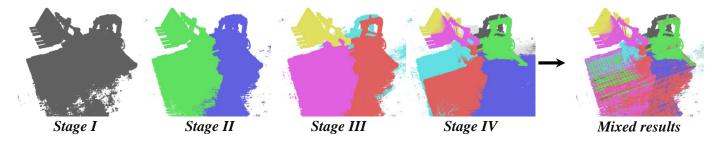


Fig. 6: Scene segmentation at different stages. Left: segmentation at different stages. Right: blocks owning each query point, indicating early termination.

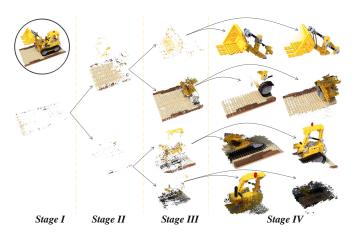


Fig. 7: Recursive rendering. Various query points are finalised early in different stages, and finally all points are aggregated to form the rendered image at the upper left.

During training, sample points can exit at multiple stages, while points will exit only once at a specific stage during inferencing. Points found earlier to be reliable will immediately exit and be rendered. Which branch is taken depends on the results of clustering. We cluster the uncertain points in the current stage and feed them to different child-branches with the same structure in the next stage.

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 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  $\epsilon$ . We use the color  $c_i$  and density  $\sigma_i$  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.

$$\sigma(r(t)) = \sigma_i, i = \min\{i | \delta_i < \epsilon \land r \in R_i\}$$
 (12)

$$c(r(t)) = c_i, i = \min\{i | \delta_i < \epsilon \land r \in R_i\}$$
(13)

where  $R_i$  is the set of points contained in the i-th 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 [37]. 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], and Cornell Box dataset [38]. **Synthetic-NeRF** contains eight man-made objects with complicated geometry and materials. Each object is realistically rendered in 300 views at a resolution of  $800 \times 800$  pixels. We used the same split into

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

ataset [14]					
Method	$PSNR^{\uparrow}$	SSIM↑	$LPIPS^{\downarrow}$	$FLOPs^{\downarrow}$	$Time^{\downarrow}$
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	591488	26.14
Ours	31.34	0.953	0.052	377159	17.68

training and testing data as NeRF [14]. 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  $800 \times 800$  pixels from views uniformly sampled with the camera moving along a spiral curve, and randomly selected 200 pictures as the training set, and the remainder for testing.

#### 5.1.3 Baseline

Using the above datasets, we compared our approach to several current state-of-the-art methods: SRN [11], NV [6], LLFF [8], and NeRF [14]. As explained in Sec. 2.1, NSVF focuses on avoiding calculating integrals for empty voxels by using a more reasonable sampling strategy, while each sampled point still goes through a whole NeRF. Recursive-NeRF improves upon NeRF in a complementary way by avoiding unnecessary calculation for each sample point by using an adaptive early termination strategy based on the learned uncertainty, which can be combined with NSVF to further improve the original NeRF. We thus do not make a comparison between ours and NSVF in what follows.

## 5.1.4 Implementation details

The batchsize was set to 4096 in our training stage, 64 and 128 sampling points were used for the coarse and 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 grown networks had 2, 2, 4, and 4 linear layers respectively. We used Adam as our optimizer and set a learning rate with initial value of  $5\times 10^{-4}$  and 10 times exponential learning rate decay after 250k iterations.

# 5.2 Results

# 5.2.1 Qualitative comparison

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 [39]

TABLE 2: Quantitative Comparison on Cornell Box

TIBEL 2. Quantum Companion on Content Sex							
Method	PSNR↑	SSIM↑	LPIPS↓	FLOPs↓	Time↓		
NeRF Ours	49.237 48.010	0.996 0.996	0.015 0.010	591488 198361	26.14 10.82		

(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 floating point operations FLOPs and rendering time in both Table 1 and 2. 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 on 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 to render a image at the same 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.

#### 5.3 Ablation Study

We conducted ablation experiments as described below; the results are shown in Table. 3. A qualitative comparison of results on the Lego dataset is shown in Fig. 11.

# 5.3.1 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.

# 5.3.2 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. 12 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.

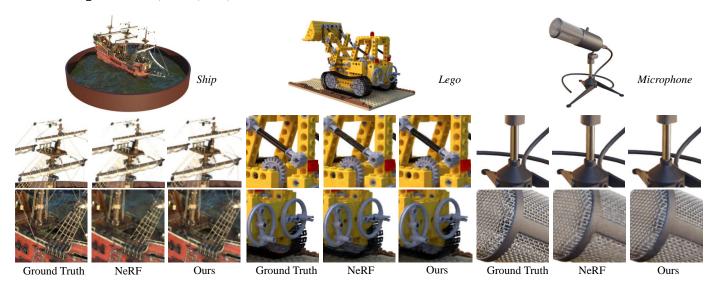


Fig. 8: Qualitative results. Top: scene. Middle, below: Two close-ups of the scene. We show the ground truth, the results of NeRF rendering, and of our method in turn.

Full model



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.

# 5.3.3 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 3). The chain network had a #2FC - #2FC - #4FC - #4FC structure where #iFC represents a fully connected layer with i-

TABLE 5: Ablation experiment							
Method	PSNR↑	SSIM↑	LPIPS↓	FLOPs↓			
No classification No branch No early termination	32.697 32.374 33.118	0.966 0.962 0.970	0.035 0.040 0.032	326463 349504 854656			

0.967

0.033

347765

32.900

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.

#### 5.4 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.

In principle Recursive-NeRF can add as many stages as desired to represent a large and complicated scene. Currently, however, the maximal number of stages is fixed to 4. Also, the injection of sub-networks for a query coordinate is also fixed during training once the cluster centers of uncertain points at each stage has been determined, making it unadjustable in later training processes. This could be improved in two ways to make our approach more suitable for large and complicated scenes. Firstly, we could select an appropriate number of stages to best represent the whole scene by using techniques from neural architecture search [40]. Secondly, learnable clustering [41] of uncertain

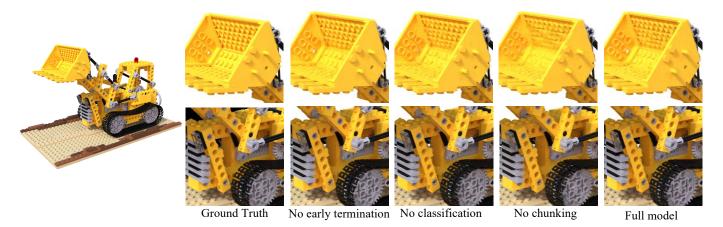


Fig. 11: Qualitative comparison of ablation experiment.



Fig. 12: Scene segmentation results. Left: original Lego scene. Middle: result using random division (no clustering). Right: result of clustering uncertain points using *K*-means.

points could be exploited to make the branched networks more adaptive to complexity of parts of the underlying scene.

# 6 CONCLUSIONS

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

## **ACKNOWLEDGMENTS**

This work was supported by the Natural Science Foundation of China (Project Number 61521002). 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 *ICCV*. IEEE, 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 SIGGRAPH. ACM, 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 CVPR. Computer Vision Foundation / IEEE, 2019, pp. 2437–2446.
- [8] B. Mildenhall, P. P. Srinivasan, R. O. Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar, "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
- [11] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, "Scene representation networks: Continuous 3d-structure-aware neural scene representations," in *NeurIPS*, 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 CVPR. IEEE, 2020, pp. 3501–3512.
- [13] W. Chen, H. Ling, J. Gao, E. J. Smith, J. Lehtinen, A. Jacobson, and S. Fidler, "Learning to predict 3d objects with an interpolationbased differentiable renderer," in *NeurIPS*, 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 ECCV (1), ser. Lecture Notes in Computer Science, vol. 12346. Springer, 2020, pp. 405–421.
  [15] K. Zhang, G. Riegler, N. Snavely, and V. Koltun, "Nerf++: Ana-
- [15] K. Zhang, G. Riegler, N. Snavely, and V. Koltun, "Nerf++: Analyzing and improving neural radiance fields," arXiv preprint, vol. arXiv:2010.07492, 2020.
- [16] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla, "Deformable neural radiance fields," arXiv preprint, vol. arXiv:2011.12948, 2020.

- [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," arXiv preprint, vol. arXiv:2012.03927, 2020.
- [18] L. Liu, J. Gu, K. Z. Lin, T. Chua, and C. Theobalt, "Neural sparse voxel fields," in NeurIPS, 2020.
- [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 Francisco: 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, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. M. Saragih, M. Nießner, R. Pandey, S. R. Fanello, G. Wetzstein, J. Zhu, C. Theobalt, M. Agrawala, E. Shechtman, D. B. Goldman, and M. Zollhöfer, "State of the art on neural rendering," Comput. Graph. Forum, vol. 39, no. 2, pp. 701–727, 2020.
- [22] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon, "Differentiable rendering: A survey," *arXiv preprint*, vol. arXiv:2006.12057, 2020.
- [23] C. M. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, and T. A. Funkhouser, "Local implicit grid representations for 3d scenes," in CVPR. IEEE, 2020, pp. 6000–6009.
- [24] J. J. Park, P. Florence, J. Straub, R. A. Newcombe, and S. Love-grove, "Deepsdf: Learning continuous signed distance functions for shape representation," in CVPR. Computer Vision Foundation / IEEE, 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 CVPR. Computer Vision Foundation / IEEE, 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 CVPR. IEEE, 2020, pp. 4856–4865.
- [27] H. Kato, Y. Ushiku, and T. Harada, "Neural 3d mesh renderer," in *CVPR*. IEEE Computer Society, 2018, pp. 3907–3916.
- [28] K. Schwarz, Y. Liao, M. Niemeyer, and A. Geiger, "GRAF: generative radiance fields for 3d-aware image synthesis," in *NeurIPS*, 2020
- [29] D. B. Lindell, J. N. P. Martel, and G. Wetzstein, "Autoint: Automatic integration for fast neural volume rendering," *arXiv preprint*, vol. arXiv:2012.01714, 2020.
- [30] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "Fastnerf: High-fidelity neural rendering at 200fps," arXiv preprint arXiv:2103.10380, 2021.
- [31] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "Plenoctrees for real-time rendering of neural radiance fields," *arXiv* preprint arXiv:2103.14024, 2021.
- [32] 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 *ICLR*. OpenReview.net, 2018.
- [33] J. Yu, L. Yang, N. Xu, J. Yang, and T. S. Huang, "Slimmable neural networks," in *ICLR (Poster)*. OpenReview.net, 2019.
- [34] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *ICCV*. IEEE, 2019, pp. 1803– 1811.
- [35] J. Yu and T. Huang, "Autoslim: Towards one-shot architecture search for channel numbers," *arXiv preprint*, vol. arXiv:1903.11728, 2019
- [36] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," Annals of Data Science, vol. 2, no. 2, pp. 165–193, 2015.
- [37] 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," *Science China Information Sciences*, vol. 63, no. 12, pp. 222 103:1–222 103:21, 2020.
- [38] Cornell.edu. (2021) The cornell box. [Online]. Available: http://www.graphics.cornell.edu/online/box/
- [39] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*. IEEE Computer Society, 2018, pp. 586–595.
- [40] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," J. Mach. Learn. Res., vol. 20, pp. 55:1–55:21, 2019.

[41] F. Williams, J. Parent-Lévesque, D. Nowrouzezahrai, D. Panozzo, K. M. Yi, and A. Tagliasacchi, "Voronoinet: General functional approximators with local support," in CVPR Workshops. IEEE, 2020, pp. 1069–1073.



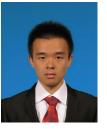
**Guo-Wei Yang** is currently a Ph.D. student in the Department of Computer Science and Technology, Tsinghua University. His research interests include computer graphics, image analysis, and computer vision.



**Wen-Yang Zhou** is currently a Ph.D. student 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 an undergraduate student at Tsinghua University. His research interests include computer graphics and computer vision.



**Dun Liang** is a Ph.D. candidate in the Department of Computer Science and Technology at Tsinghua University, where he received his B.S. degree in computer science and technology, in 2016. His research interests include computer graphics, visual media learning and high-performance computing.



**Tai-Jiang Mu** is currently an assistant researcher at Tsinghua University, where he received his B.S. and Ph.D. degrees in computer science and technology in 2011 and 2016, respectively. His research interests include computer vision, robotics and computer graphics.



Shi-Min Hu received his Ph.D. degree from Zhejiang University, in 1996. He is currently a professor with the Department of Computer Science and Technology, Tsinghua University. He has authored over 100 papers. His research interests include digital geometry processing, video processing, rendering, computer animation, and computer-aided geometric design. He is the Editor-in-Chief of Computational Visual Media, and on the Editorial Board of several other journals, including Computer Aided Design

and Computer & Graphics (both Elsevier).