Neural Adaptive SCEne Tracing (NAScenT)

Anonymous ECCV submission

Paper ID 7628

1 Pipeline and Algorithm

In this section, we give a brief introduction for our overall pipleline of algorithms for training and rendering our hierarchical neural networks (NAScenT).

Hierarchical Neural Network Training Our method takes multiple viewpoint images I_{gt} as input, and outputs the optimized octtree and contained neural networks $\mathcal{M}$. In Alg. (1), S is the set of sampled points by using sampling strategy in Sec. (3.3), and $I(r)$ is rendering RGB value for ray direction r. The loss function $\mathcal{L}$ is the photometric loss between rendered pixel values $I(\mathrm{r})$ and ground truth values $I_{gt}(\mathrm{r})$. The loss is backpropated to each sub-network to update the weights. Every T_B rounds of training, the octtree of scene will be updated to adaptively reallocate computational resources to regions with high density and high projected error, and the new sub-networks are directly pre-trained by using stratified samples from the previous sub-networks (see Sec. (3.5) in main).

Algorithm 1: Hierarchical Neural Network Training

Input: viewpoint images I_{gt}
Output: $\mathcal{M}$
Result: novel views I
Initialize Φ, $\mathcal{M}$;
while $t < T$ **do**
  $S = Sampler(\mathcal{M})$ Sec. (3.3) in main;
  $I(\mathrm{r}) = Render(S)$ Alg. (2);
  loss $= \mathcal{L}(I(\mathrm{r}), I_{gt}(\mathrm{r}))$;
  BackPropagate(loss);
  Step(Φ);
  if $t \mod T_B = 0$ **then**
    $\mathcal{M} = UpdateOctree()$ Eqn. (1);
    $\Phi = UpdateModel(\Phi, \mathcal{M})$ Sec. (3.5) in main;
  end
  t = t + 1;
end

Hierarchical Neural Network Rendering Rendering pipeline Alg. (2) takes batches of samples S, hierarchical neural networks Φ, and octtree models $\mathcal{M}$ as inputs. Sampling points S are scheduled to corresponding sub-networks Φ_i^l by their 3D sample location,. Samples are then evaluates by the respective sub-network Φ_i^l. To calculate the ordered integral along the ray direction, samples are sorted by Eqn. (4) in main and then composited by Eqn. (3) in main.

Algorithm 2: Hierarchical Neural Network Rendering

Input: 3D scene samples S, Φ, $\mathcal{M}$
Output: rendering value I
$\{S_i^l\} = \mathit{Scheduler}(S, \mathcal{M})$;
$C = \{\}$, $D = \{\}$;
for Φ_i^l *in* Φ **do**
    $\mathbf{c}^{B_i^l}$, $\sigma^{B_i^l} = \Phi_i^l(S_i^l)$;
    C.add($\mathbf{c}^{B_i^l}$); D.add($\sigma^{B_i^l}$);
end
$\{C, D\} = \mathit{SortByZ}(\{C, D\})$;   Eqn. (4) in main
$I(r) = \mathit{Composite}$(C, D);   Eqn. (3) in main

1.1 Importance Sampling

Fig. (1) gives a illustration of the importance sampling in Sec. (3.3).

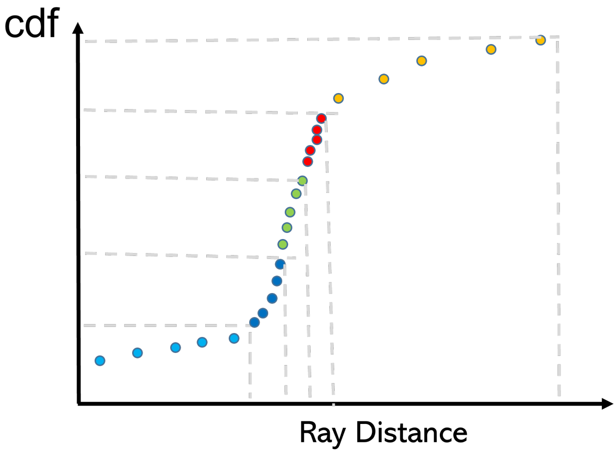


Fig. 1. Illustration of sampling scheme that based on cumulative density field.

2 OctTree Update Scheme

In this section, we discuss the details of the octtree update scheme. The intuition of updating structural octtree are (1) avoid time-costly sampling and computation inside empty node, (2) reallocating representation (sub-networks) and computational (number of samples) resource to complex or poorly represented part of the scene.

Our objective mainly contains two parts, α_i is weighted average alpha vector of node i of sub-network, which indicates the opaque of node, β_i is projected rendering error vector of node i, since flat or smooth surfaces may converge quickly and be well-trained, while complex or poorly-represented scenes may still need more epochs to obtain better quality. Therefore, the β term will explore finer or coarser trees to encourage lower projected rendering error in octtree structure. Our objective is shown as,

$$\min \sum_i (1 - \alpha_i^\intercal) I_i + \beta_i^\intercal I_i, \quad \text{s.t.,} \begin{cases} I_i^\uparrow + I_i^= + I_i^\downarrow = 1, \\ \sum_i \frac{1}{N_c} I_i^\uparrow + I_i^= + N_c I_i^\downarrow \leq N_B, \end{cases} \tag{1}$$

where $I_i = [I_i^\uparrow, I_i^=, I_i^\downarrow]^\intercal$ are boolean flags of node operations, i.e., merge ($\uparrow$), split ($\downarrow$), and unchanged ($=$). $\alpha_i = [\alpha_i^\uparrow, \alpha_i^=, \alpha_i^\downarrow]^\intercal$ is the weighted average alpha in octree node i for three possible operations, if α_i . $\beta_i = [\beta_i^\uparrow, \beta_i^=, \beta_i^\downarrow]^\intercal$ is the weighted average projected rendering error respectively. N_B is user-defined maximal block in system.

To calculate value α_i, we first perform stratified sampling from top to bottom in the octree hierarchy and predict the density value for each sample by running the forward rendering network $\Phi(\mathbf{x}, \mathbf{d}) = (\mathbf{c}, \sigma)$, then the $\alpha_i^=$ for each block by,

$$\begin{cases} \alpha_i^= = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \delta(\mathbf{x}), \\ \alpha_i^\uparrow = \frac{1}{N_c} \alpha_{\mathcal{P}(i)}^=, \\ \alpha_i^\downarrow = \sum_{j \in \mathcal{C}(i)} \alpha_j^=, \end{cases} \tag{2}$$

where $\mathcal{P}$ and $\mathcal{C}$ are query functions for octree parent and child nodes. S_i denotes the samples inside an active block.

To calculate w_i for different cases, we first evaluate rendering error for each ray $E(\mathbf{r}) = \mathcal{L}(I(r), I_{gt}(r))$, and $\mathcal{L}$ is simple function for mean square error.

$$\begin{cases} \beta_i^= = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \frac{w(x)}{W} E(\mathbf{r}), \\ \beta_i^\uparrow = \frac{1}{N_c} \beta_{\mathcal{P}(i)}^=, \\ \beta_i^\downarrow = \sum_{j \in \mathcal{C}(i)} \beta_j^=, \end{cases} \tag{3}$$

where $w(x)$ is the weight in rendering function Eqn. (3) in the main paper (i.e., T_i) for samples $x \in \mathbf{r}$. $W = \sum_{x \in \mathbf{r}} w(x)$ is the total sum of weights along the ray direction. To optimize Eqn. (1), we use `or-tools` to solve MIP problems.

3 Additional Comparison and Results

(a) GT (b) NeRF[3] (c) KiloNeRF[4] (d) Our (e) Octree

Fig. 2. Visual Comparison on LLFF-NeRF dataset [2].(a) is ground truth view of flower scene with highlighting details. (b)-(d) are the novel view of NeRF [3], KiloNeRF [4] and our methods with highlight details. (e) the visualization of example view and octree optimization process from initial level 2 to level 3, and merge to simple structure to save computational and sampling resource.

In this section, we show extensive comparison in details and results. As discussed in the main text, for views close to the training views all methods produce visually very similar results; differences only become apparent at close inspection and when analyzing depth structure. However, as the results in the main paper show, the differences in the depth estimation amplify the visual quality differences for extrapolated views far from the training data.

Fig. (2) shows visual comparisons of novel view synthesis on real scenes from the LLFF-NeRF dataset [2]. As we can see in the figure, NeRF [3] can miss surfaces with its sampling process so that back surfaces can "shine through". This is due NeRF's sampling scheme that applies stratified search for a coarse-level density distribution estimation and then sampling according to coarse density distribution along the ray to give more samples near the object surface. Thus, for a thin structure, the coarse level search may missing the important part of scene, leads to leaking light effect in rendering novel view. KiloNeRF only applies dense stratified sampling inside each block, and collects output samples along the ray. Thus, blocking-effect are again visible just like in the synthetic data. Our method achieves sharper novel depth map, and the light-weight sub-network enables a dense coarse level surface search, alleviate blocking effect as well as light leakage.

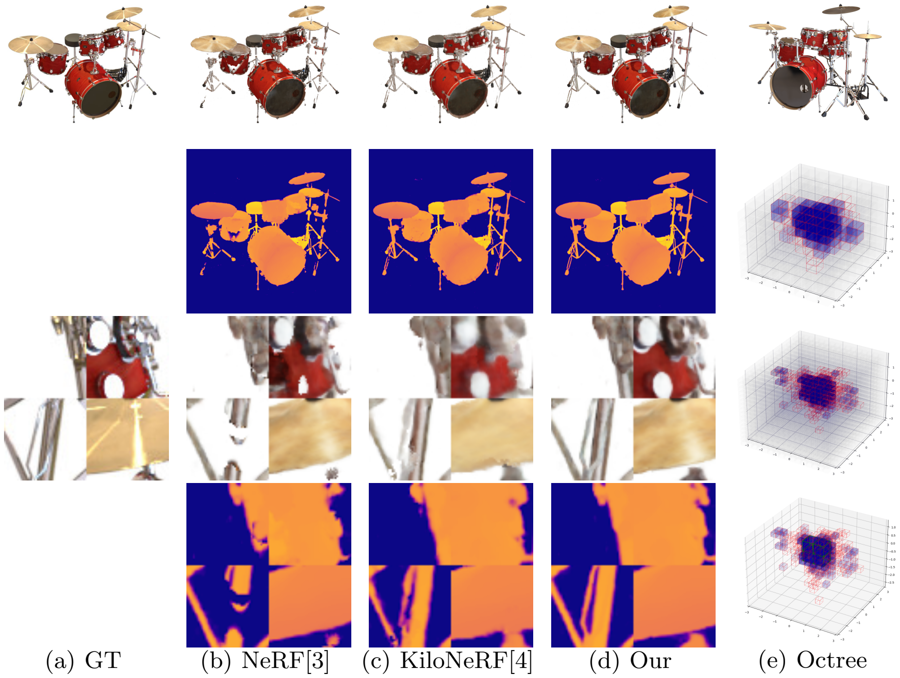


Fig. 3. Visual Comparison on synthetic NeRF dataset [3]. (a) is ground truth view with zoom in details. (b), (c), (d) are state-of-the-arts methods of NeRF [3], KiloNeRF [4], and our proposed method with highlighted detail regions. (e) illustrate example view and octree optimization process from coarse to fine (Blocks with green line indicates block merging, and red line indicates blocks splitting).

Fig. (3) shows visual comparisons on the synthetic NeRF dataset. All the state-of-the-art methods achieve reasonable performance in rendering novel views from camera positions close to the training views. However, in visualizations of the depth map scene, NeRF [3] shows blurring and topological artifacts in the depth, indicating that the actual 3D structure is less accurate. As we show below, this has an impact on the quality of extrapolated views far from the training data. KiloNeRF [4] shows high quality results in the RGB view, but exhibits a slight blocking effect when visualizing the depth view, since KiloNeRF's sub-network are pre-trained by a global network, i.e., NeRF, each sub-network is independent in the pre-training stage and mix the query results in the fine-tuning stage, which introduces a discontinuity. Our method takes advantage of the tree structure for flexible and scalable representation with an adaptive training scheme for computational resource allocation, all the sub-networks are trained from coarse to fine. Therefore, no pre-training is required, and back propagation will update all sub-network along the integral ray direction, which shows consistent and smooth rendering results in both RGB view and depth. To better illustrate our octree-based representation, we also show a progression of the octree update last column (ground truth rendering on top).

3.1 Extreme Novel View Comparison

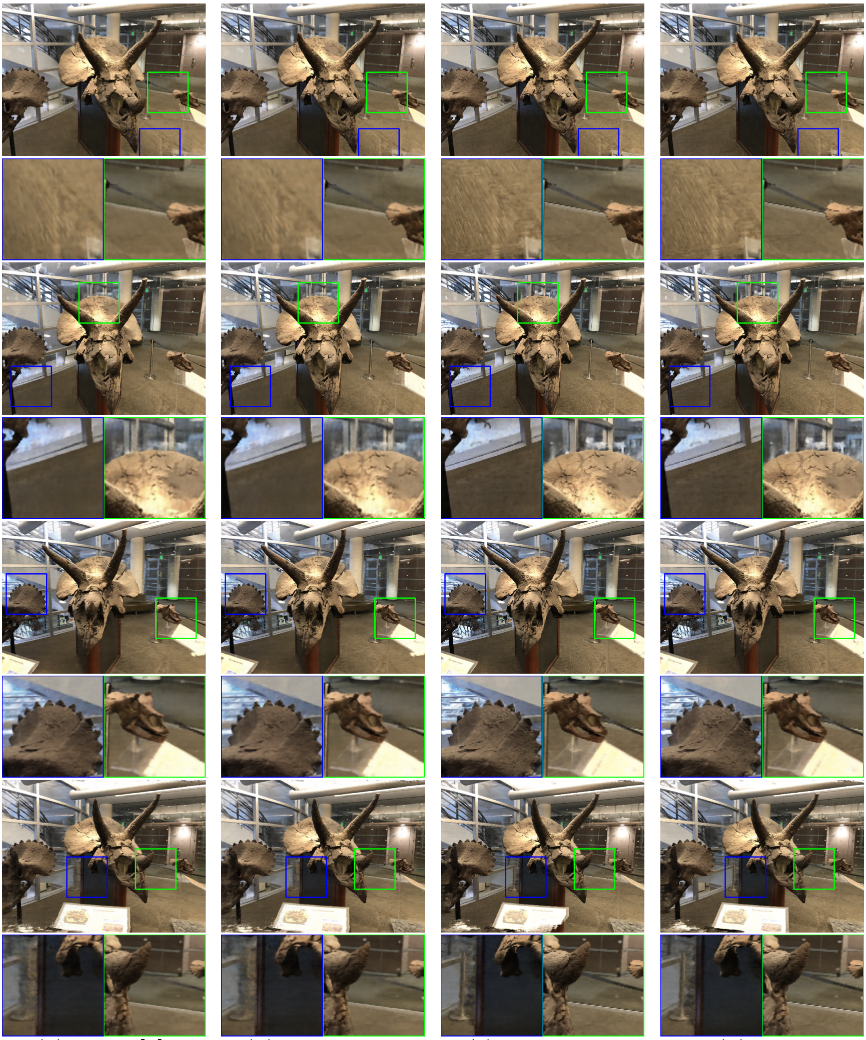
In this section, we show the extensive comparison in details for extreme view of real scene dataset. Fig. (4) shows the novel view synthesis with view rotation radius $R = 1.5$, our method shows similar performance with MipNeRF [1], but outperforms NeRF and KiloNeRF in details texture recovering. Fig. (5) increase rotation radius for extreme view rendering, NeRF, KiloNeRF and MipNeRF show significant false trails in extrapolated view due to neural network tends to output unknown density value in extrapolated part of scene. Octtree could explicitly define rendering space, and significantly allevate rendering trails. Alought KiloNeRF can also use pre-trained octtree to accelerate rendering process, it still require train a extra single network for model distilling, thus reserve same trails effect in the fine-tune stage. Fig. (6) also shows more comparison for extrapolated novel view synthesis, our method outperform other alternatives, see details for better visual comparison.

3.2 UAV Scene Reconstruction

In this section, we show the results of ground scene reconstruction from UAV video in Fig. (7). UAV views contain comparatively large viewpoint and camera pose change, which is a challengine task for neural rendering. NeRF [3] shows blur rendering results due to incorrect density estimation of scene. MipNeRF [1] fails to estimate correct density in second row of results, partially because a large viewpoint changes leads to sample in the space that has insufficient training samples and outputs random density values. Our method takes advantage of (1) octtree that bounds whole scene and skip empty space, and (2) distributed subnetwork architecture that trains and renders locally to avoid inbalanced sampling inside each octtree nodes and adaptively reallocate computational resource for each node.

References

1. Barron, J.T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., Srinivasan, P.P.: Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In: ICCV (2021)
2. Mildenhall, B., Srinivasan, P.P., Ortiz-Cayon, R., Kalantari, N.K., Ramamoorthi, R., Ng, R., Kar, A.: Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Transactions on Graphics (TOG) (2019)

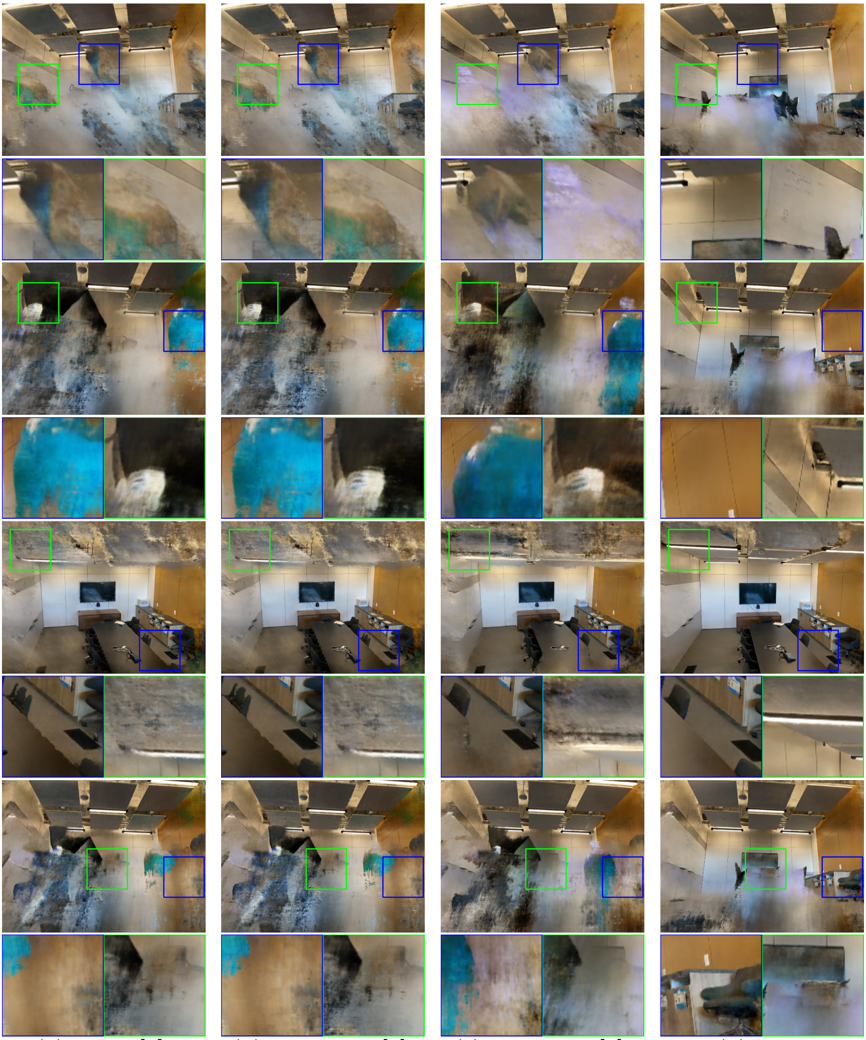


(a) NeRF[3] (b) KiloNeRF (c) MipNeRF (d) Our

Fig. 4. Extreme novel view synthesis for HORNS dataset with view rotation $R = 1.5$. We compare our method against NeRF [3], KiloNeRF [4], MipNeRF [1].

(a) NeRF[3] (b) KiloNeRF[4] (c) MipNeRF[1] (d) Our

Fig. 5. Extreme novel view synthesis for HORNS dataset with view rotation $R = 3.0$. We compare our method against NeRF [3], KiloNeRF [4], MipNeRF [1].



(a) NeRF[3] (b) KiloNeRF[4] (c) MipNeRF[1] (d) Our

Fig. 6. Extreme novel view synthesis for ROOMS dataset with view rotation $R = 3.0$. We compare our method against NeRF [3], KiloNeRF [4], MipNeRF [1].

(a) NeRF[3] (b) MipNeRF[1] (c) Our

Fig. 7. UAV scene reconstruction. We compare our method against NeRF [3], KiloNeRF [4], MipNeRF [1].

3. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: ECCV (2020)
4. Reiser, C., Peng, S., Liao, Y., Geiger, A.: KiloNeRF: Speeding up neural radiance fields with thousands of tiny mlps. In: International Conference on Computer Vision (ICCV) (2021)