Hierarchical Articulated NeRF for 3D Human Reconstruction and Rendering

Hao Chen[1,2] Shanxin Yuan[1] Helisa Dhamo[1] Ales Leonardis[1,2]

[1]Huawei Noah's Ark Lab [2]University of Birmingham

Abstract. Neural Radiance Fields (NeRF) have gained popularity for 3D representation and rendering of objects and scenes due to their high quality photo-realistic results. NeRFs are capable to synthesize novel views of rigid objects and have been extended to dynamic and deformable real scenes. The main initial limitation of NeRF is its speed. Several methods considerably improved the efficiency of training and inference, however, sacrifice either on quality or controllability. In this paper we propose *Hierarchical Articulated NeRF* (HANeRF), a method that overcomes this issue by simultaneously being compatible with fast data structures for sampling and high-resolution NeRF techniques. HANeRF leverages importance sampling to allow for efficient dynamic deformations with a low per sample computation. An hierarchical articulated deformation model is used where blendshape parameters are diffused into 3D in the observed domain for a fixed set of coarse grid points. To retrieve the deformation field, we use grid points to interpolate these blendshape parameters for the sampled points. Aside of extensive ablation studies on pipeline components, we present the results of HANeRF on the ZJU dataset where it is used to animate 3D virtual humans.

1 Introduction

Virtualization of content has become a vital element in many areas of modern life. Synthetic data presentation of products online, the 3D capture of scenes with drones and the advent of virtual reality setups are just some of the many examples where photorealistic rendering is the crucial technology to revive the scene content after the scene has been captured. Complex lighting conditions, varying textures and dynamic motion present challenges to extracting high quality scene representations and to synthesize novel views that with satisfying quality. The proposal of an implicit scene representation through a Neural Radiance Field (NeRF) [17] has inspired a number of works that try to tackle long-standing graphics problems in a novel way by using deep learning to represent a scene implicitly as a $5D$ function encapsulating the location (x, y, z) in $3D$, together with the emitted radiance θ in each direction ϕ. In only a short amount of time, NeRF-based methods have achieved state-of-the-art rendering quality in many areas [15] and creative extensions to the original idea have made it possible to implicitly cope with light changes [15], dynamic scene content [12, 23, 26] and even include human shape priors [24, 25].

Even though NeRFs store the scene content with a comparably low number of parameters, *i.e.*in the order of 1 M, training and evaluation of a NeRF is computationally complex. A VR-ready image rendering of the original NeRF, for instance, would require 37 petaFLOPS, which is far from what current GPUs can compute [18]. Several recent works have successfully proposed ways to achieve faster inference [11, 7, 27, 33, 6, 31, 4]. While training initially required orders of magnitude more time and resources [18, 13, 9], it has also been made significantly more efficient using direct voxel grid optimization [28].

To this end, we contribute:

1. We introduce iterative hierarchical interpolation of probability densities (PDF) at sample locations during NeRF progressive training. This allows us to *reduce the training time* by 13 times. The approach also carries over to the inference stage.
2. A coarse to fine deformation model HANeRF that diffuses blendshape parameters in a 3D grid and interpolates on sampled point locations with correct deformations.
3. HANeRF maintains the compatibility to other NeRF pipelines, such that dynamic, and deformable scenes can be learned and synthesized. We demonstrate this in the case of a controllable NeRF-driven human 3D model on the ZJU dataset.

2 Related Work

Analysis by synthesis [8] is a powerful paradigm, where the forward system produces appearance of a scene or an object and the training goal is to find the best reconstruction by maximising the agreement between the measurement and the prediction. Differentiable rendering techniques [14] are excellent tools for analysis by synthesis. They can be applied to various 3D reconstruction tasks, even from single images and they can be trained without explicit supervision. Neural rendering [30] applies a neural network (NN) in the rendering pipeline. In most cases a neural network represents the 3D volumes implicitly. They use the location as input to the network and the output is an attribute of the volume. Neural signed distance functions [22, **?**] encode the distance to the surface. Occupancy networks [16] and implicit fields [2] encode whether the location is inside or outside the object.

Neural radiance fields (NeRF) [15] is arguably one of the most successful neural rendering techniques. It represents the volume with a Multi Layer Perceptron (MLP) that computes the density and colour for each 3D location. Numerical integration is used to photometrically render pixel colours. NeRF has inspired many follow-up works and extensions, *e.g.*in the wild undergoing severe lighting changes [15], multi-view surface reconstruction [32], and learning surface light fields [20]. Mip-NeRF [1] improves upon NeRF through continuous scale representation of the scene while using anti-aliased frustums instead of rays. While these works are limited to static scenes, Nerfies [23] extend the idea to non-rigidly deforming objects within the scene through deformation field optimization with

rigidity priors. D-NeRF [26] also addresses dynamic scenes, where a temporal component is considered to map the scene from a canonical space with deformation through time. NeRF Volumes [12] deal with deformable 3D scenes with a voxel grid and a warp field that are regressed from a 3D CNN. To specifically address humans, Neural body [25] and [24] learn an implicit deformable neural body representation which can be controlled for 3D human reconstruction and synthesis. GIRAFFE [19] leverages NeRF for low resolution features where a 2D neural rendering network computes an output image from the feature grid.

2.1 Faster inference

Inference speed was a major downside of early NeRF approaches. However, many clever approaches and tricks have been presented recently to improve the inference speed of NeRF. Some of them are even capable of speeding up the forward path by a thousand times and thus enabling real-time inference.

Most techniques for fast inference resample NeRF into an data structure that is more render-friendly. Some use a grid [6], Plenoctrees [33] an octree, and NeX [31] uses multiplane images. Further speedup is achieved by reducing the number of samples to compute the pixel colour by omitting empty regions of the volume. Neural Sparse Voxel Fields [11] builds a grid in a self-supervised way to keep track of the empty volumes. KiloNeRF [27] trains separate networks for all voxels in a grid. As these networks are much smaller and faster to compute, they achieve 600x speedup. However they need a pre-trained NeRF in their training process. Autoint [10] learns to integrate ray segments to reduce the number of samples and Cole *et al.* [5] turn a pre-trained NeRF into a light-field, thus the method only has to access a neural network once per ray. They achieve 128x speedup.

The above mentioned methods however have shortcomings. Except for FastNeRF [6] they are unable to handle deformable, dynamic or controllable scenes and most lose the differentiability and trainability after resampling NeRF. Despite all the efforts of these pioneering works to speed up the inference, little progress has been made in reducing the lengthy training time. Therefore, our objective is to take a closer look at NeRF training and speed up the learning process.

2.2 Deformation & Controllability

Standard training strategies for NeRF require days to train for a single scene. To improve upon this, some works investigate methods for potential efficiency increase.

DONeRF [18] uses the ground truth depth to train a depth classifier to reduce the number of samples. The approach of Lomdardi et al. [13] uses volumetric primitives that help to ignore empty regions of space. While both ideas can improve training time, they require the knowledge of the scene depth a priori. DSNerf [9] speeds up training 2-6x using depth as a supervision signal (running SfM).

Methods like Learned initializations [29], Stereo Radiance Fields [3] and PixelNeRF [34] first perform a large scale pre-training process then use the encoded features to condition NeRF. Then only a few minutes of fine-tuning is needed for high quality results on novel view synthesis. We consider the ideas in these works orthogonal to ours and we focus on the original NeRF training task with multiview images. To the best of our knowledge only PlenOctrees [33] claims to speed up NeRF training in its original setting. First, after a small number of iterations they fix an octree, then use it to guide the sampling for the training and achieve 5x speedup.

To pave the way to a wider acceptance and use of NeRF, the training time needs to be reduced considerably. Moreover, we address common limitations of current methods. We do not require resampling at inference time, thus we preserve the differentiability. We are also able to handle dynamic and deformable scenes. Specifically we show animations of 3D humans.

contrast to neural body [25] which diffuses the latent vectors used to condition NeRF, we instead diffuse the blendshape parameters used to deform the point. Moreover, we use an interpolation on the blendshape parameter for the sampled point instead of an additional MLP [24] in order to calculate the deformation field. We provide the detail of these contributions in our method section 4. Before that, we introduce our NeRF notation and the necessary mathematical concepts.

3 Background: Neural Radiance Fields

Neural Radiance Field [17] were designed to solve the multiview stereo reconstruction problem, where the input of the training process is a set of images and the corresponding calibrated camera parameters, both intrinsic and extrinsic. The output of the training is an implicit representation of the scene, which can be used for novel view synthesis during inference. The depth information can also be extracted.

NeRF [17] represents the volume implicitly with a neural network F, an MLP in the original implementation. The network calculates the density $\sigma \in \mathbb{R}$ and RGB colour $\mathbf{c} \in \mathbb{R}^3$ values for each 3D location $\mathbf{x} \in \mathbb{R}^3$, as $[\sigma, \mathbf{c}] = F(\mathbf{x}, \mathbf{d})$. The scene is rendered by integrating the weighted colour values along rays cast from each pixel. Let us denote the ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, where $\mathbf{o} \in \mathbb{R}^3$ is the origin, $\mathbf{d} \in \mathbb{R}^3$ is the direction and $t_n < t < t_f$. Then, the expected colour C is calculated as

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})\, dt, \tag{1}$$

$$\text{where} \quad T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\, ds\right). \tag{2}$$

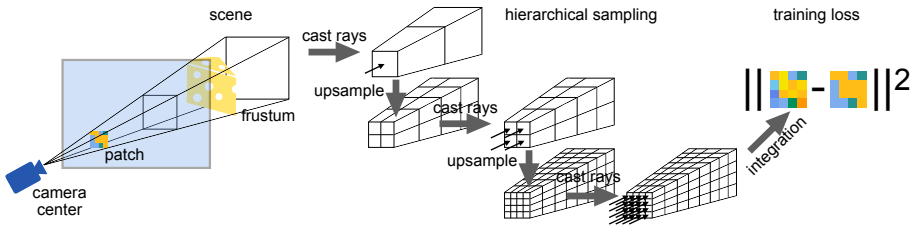

Fig. 1. Rays are cast for pixels in a patch and points are sampled along the rays and the transmission probabilities are obtained by the evaluation of a NN. Next the probabilities are upsampled, and sampling is repeated on a higher resolution but only at the most probable locations according to the previous estimate. The predicted patch is obtained by numerical integration. The training is done by minimizing the L^2 loss between the prediction and ground truth.

In practice a numerical integration technique is used to approximate $C(\mathbf{r})$ on a finite number random samples along the ray.

$$C(\mathbf{r}) = \sum_{i=1}^{N} T_i (1 - \exp(-\sigma_i \delta_i)) \mathbf{c}_i, \tag{3}$$

$$\text{where} \quad T_i = \exp\left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right), \tag{4}$$

where $\delta_i = t_{i+1} - t_i$ are the distances between samples t_i along the ray. Here t_i are randomly sampled within the intervals $[t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)]$, where t_f and t_n are the closest and furthest points of the rendering interval respectively. The training is done by optimising the L^2 loss between the predicted pixel colours and the ground truth C_{gt}.

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{A}} \| C(\mathbf{r}) - C_{gt}(\mathbf{r}) \|_2^2 , \tag{5}$$

where $\mathcal{A}$ is the set of rays.

This process is very slow both for the training and for the inference due to the high number of points sampled for each ray.

4 Method

In vanilla NeRF, the samples are evenly distributed. This is inefficient, as most of the volume is empty. A lot of computation is used up on samples that do not contribute to the final pixel colour.

The rendering time is directly proportional to the number of samples, thus can be reduced by sampling a fewer times only from dense locations. The main challenge is to estimate the locations of those dense regions before the sampling. The key steps of our solution are

1. Render the image at a lower resolution.
2. Keep track of the probability densities (PDF) at the sample locations.
3. Upsample the parameters of the PDF to the high resolution.
4. Use the PDF estimate to sample only at dense locations for the high resolution image.
5. Repeat the process in an image pyramid.

Notice that we only have to sample densely at the first pyramid layer at a very low resolution. For the higher layers the number of samples are much fewer. The computation cost is dominated by the number of samples at the highest resolution layer. The method is illustrated on Figure 1 Next we explain the process in detail.

4.1 Rendering

We sample the volume in a regular 3D grid the frustum. Instead of sampling each point independently, we add the same random shift to each sample. We denote the ray with $\mathbf{r}(h, w, t)$ to highlight its dependence on the pixel locations $u \in [-1, +1]$ and $v \in [-1, +1]$ in clip space,

$$\mathbf{r}(u, v, t) = \mathbf{o} + t\mathbf{d}(u, v). \tag{6}$$

Note that the origin of the ray does not depend on the pixels, and the direction $\mathbf{d}$ is normalized. The grid locations are computed as

$$t_i = t_n + \frac{i}{N}(t_f - t_n) + \epsilon, \tag{7}$$

$$\text{where } \epsilon \sim \mathcal{U}[0, 1/N], \tag{8}$$

$$u_k = -1 + \frac{k + 0.5}{H}(1 - (-1)), \tag{9}$$

$$v_l = -1 + \frac{l + 0.5}{W}(1 - (-1)), \tag{10}$$

where $i < N$, $k < H$, $l < W$ are natural numbers and H, W image height and width respectively. We calculate $C(\mathbf{r})$ for all pixels according to 3.

During the computation we keep track of the probability density values with $w_{kli} = T_{kli}(1 - \exp(-\sigma_{kli}\delta)))$, where we omit the indices for δ as they are all the same.

During rendering we evaluate the network at multiple resolutions in a pyramid. Let us denote the index in the pyramid with a superscript. At the lowest resolution (H^0, W^0 and N^0) we evaluate all locations in the grid. At the higher levels we only evaluate it at the dense locations predicted by the previous level. The prediction is calculated by upscaling the values w_{kli}^{n-1} using bilinear interpolation.

$$\hat{\mathbf{w}}^n = \text{upscale3D}(\mathbf{w}^{n-1}), \tag{11}$$

where $\hat{\mathrm{w}}^n$ are the estimated probabilities. For each ray, only the most probable K^n locations are evaluated,

$$\{(k,l,i) : K^n > \sum_j 1[\hat{w}^n_{klj} > \hat{w}^n_{kli}]\}. \tag{12}$$

for the rest, w^n_{kli} are set to zero. The pixel colours are calculated the same way, except now with only K^n samples as the rest of the volume is considered empty.

We can repeat this process for each layer in the image pyramid, until we reach the highest resolution. The effective number of samples per pixel at the highest resolution can be computed as

$$K_{\text{eff}} = \sum_{n=0}^{L} K^n \frac{H^n W^n}{H^L W^L}. \tag{13}$$

We upscale the resolution by a factor of 2 between each level. Thus, the computation is dominated by K^L, as the contribution of the lower levels are small.

We aim to exploit this pyramid structure during training as well.

4.2 Training

The training is done on patches instead of rays. We set the patch size so it covers one pixel at the lowest resolution. We cast rays for each pixel inside the patch for each resolution. We compute the pixel colours for each layer similarly as described for the full image rendering. At the margins we pad the patch with repeated values of the lower resolution probability densities in order to use bilinear interpolation for upscaling. For the lowest resolution this effectively extends the value for that one pixel to estimate the probabilities for all rays at the next pyramid level.

The patches are randomly selected at the highest resolution and the ground truth colours for the lower resolution samples are computed by downscaling the patches using area interpolation. The loss function is the L^2 loss between the predicted and ground truth patches. For level n in the pyramid the loss is

$$\mathcal{L}^n = \sum_{p \in \mathcal{P}^n} \sum_{\mathbf{r} \in p} \left\| C^n(\mathbf{r}) - C^n_{gt}(\mathbf{r}) \right\|_2^2, \tag{14}$$

where p is a patch in the training set of patches $\mathcal{P}^n$. Note that the set of training patches are different at each level, as the density of rays inside the frustum depends on the resolution. Similarly, the ground truth colours C^n_{gt} have to be calculated by downscaling the high resolution training images by the appropriate amount.

In our method we propose to learn a separate MLP F^i for each level in the pyramid, as it is easier to train one network for one resolution. The training or inference time is not affected by this change, only the model size, which grows 5x for the 5 levels in the pyramid. In order to help the training at higher levels, we first train the lower levels in a progressive way. We iteratively optimize $\mathcal{L}^0$, then $\mathcal{L}^1$, and so forth up to $\mathcal{L}^L$.

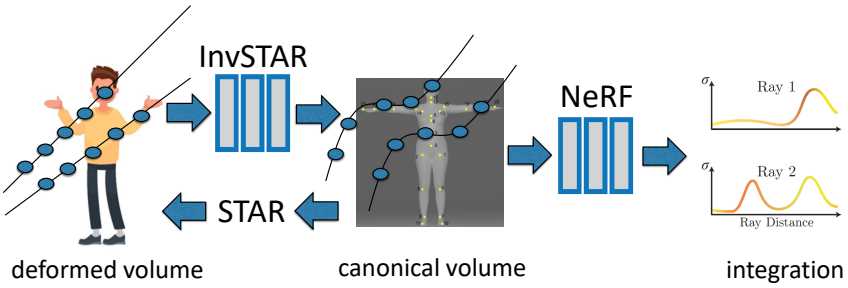

Fig. 2. Our setup for animating virtual 3D humans. We diffuse the STAR human 3D model into the full deformed volume, then invert it to get the points in the canonical volume. The output pixel values are computed by numerical integration.

4.3 Virtual human

Our method is compatible with controllable models such as the STAR [21] human 3D model, thus we can speed up the training and rendering of deformable objects. Similar to prior works [23, **?**], our model consist of a canonical volume that is modelled by NeRF and a deformed volume to which the deformation model can transform each point in the canonical volume. The (forward) deformation model in our case is the STAR human 3D model,

$$\mathbf{y} = \mathrm{STAR}(\mathbf{x}, \mathbf{s}), \tag{15}$$

where $\mathbf{x} \in \mathbb{R}^3$ and $\mathbf{y} \in \mathbb{R}^3$ are points in the canonical and the deformed volumes respectively. The STAR model can be controlled by $\mathbf{s}$, which contains the pose and shape parameters for a human model.

In order to render deformed scenes one needs to invert the STAR model, as the rays are cast inside the deformed volume and we have to get the corresponding points in the canonical volume for rendering. We call this function InvSTAR,

$$\mathbf{x} = \mathrm{InvSTAR}(\mathbf{y}, \mathbf{s}). \tag{16}$$

Note that we the STAR model is only interpreted on the surface and does not have a meaning in the whole 3D scene outside of the surface. Therefore its extension has to satisfy

$$\mathbf{x} = \mathrm{InvSTAR}(\mathrm{STAR}(\mathbf{x}, \mathbf{s}), \mathbf{s}), \tag{17}$$

where $\mathbf{x}$ is inside the domain of STAR. Outside we prefer smooth solutions to prevent the deformation from causing artefacts. We can analytically invert STAR, if we can diffuse some of its model parameters (not $\mathbf{s}$) into the 3D volume. The rendering process is illustrated on Figure 2.

Next we explain the STAR blendshape model and the inversion in detail. The deformations can be defined as:

$$\mathbf{y} = \sum_j a_j(\mathbf{x}) P_j(\mathbf{s})(\mathbf{x} + B(\mathbf{x})\mathbf{s}), \tag{18}$$

where $\mathbf{x}$ and $\mathbf{y}$ are represented in homogeneous coordinates. The shape is computed via a the linear combination $B(\mathbf{x})\mathbf{s}$, then the points are reposed with $P_j(\mathbf{s})$, which are 4×4 rigid transformation matrices. $P_j(\mathbf{s})$ and $B(\mathbf{x})$ are the position and shape blendshape fuctions respectively and the subscripts j denote the indices of the joints. The final output is computed by using a linear combination of the $P_j(\mathbf{s})$ transformations, where the weights $a_j(\mathbf{x})$ are position dependent. In the STAR model $B(\mathbf{x})$ and $a_j(\mathbf{x})$ are explicitly given for a set of set of locations (vertices in a surface mesh). $P_j(\mathbf{s})$ are factorized into a chain of rigid transformations, where each transformation describes a rotation around a joint.

It is possible to invert STAR analytically, if we have access to the values $a_j(\mathbf{x})$ and $B(\mathbf{x})$,

$$\mathbf{x} = \left(\sum_j a_j(\mathbf{x}) P_j(\mathbf{s}) \right)^{-1} \mathbf{y} - B(\mathbf{x})\mathbf{s}. \tag{19}$$

We estimate these values with $\bar{a}_j(\mathbf{y}) \approx a_j(\mathbf{x})$ and $\bar{B}(\mathbf{y}) \approx B(\mathbf{x})$ by diffusing the blendshape parameters in the 3D deformed volume. Let $\mathbf{y}_i = \mathrm{STAR}(\mathbf{x}_i, \mathbf{s})$ for the locations where a_j and B are defined. Then,

$$\bar{B}(\mathbf{y}) = \frac{\sum_i \exp\left(-\frac{|\mathbf{y} - \mathbf{y}_i|^2}{2\Delta^2} \right) B(\mathbf{x}_i)}{\sum_i \exp\left(-\frac{|\mathbf{y} - \mathbf{y}_i|^2}{2\Delta^2} \right)}, \tag{20}$$

where $\Delta > 0$ is a width parameter. The $\bar{a}_j(\mathbf{y})$ values are computed similarly.

In general, there could be multiple $\mathbf{x}$ locations that produce the same $\mathbf{y}$. Therefore we check consistency by calculating the difference $\mathbf{y} - \mathrm{STAR}(\mathrm{InvSTAR}(\mathbf{y}, \mathbf{s}), \mathbf{s})$, where STAR is extended to the full 3D volume by diffusing its parameters. We only consider the points in the rendering where the difference is under a threshold, otherwise we set its contribution to zero.

Our speedup technique is fully compatible with using any kind of deformation model, as we only need to apply the deformation (in our case InvSTAR) on the location inputs for the NN F. As the number of InvSTAR evaluations are the same as the number of samples, the speedup for the deformable model is the same as for the rigid.

We train our model with multiview images of a synthetic avatar rendered in standard T-pose from multiple views. This way we do not estimate $\mathbf{s}$ during the training. We consider estimating and refining STAR parameters during training orthogonal to our approach.

Table 1. Comparison with state-of-the-art on 'ship' scene.

Method	NeRF [17]	PlenOctrees [33]	Ours (HIPNeRF)
PSNR	**29.1**	**29.1**	27.5
Time (hours)	60	16	**4.5**

4.4 Implementation details

In our implementation we use an eight layer MLP with width of 256 for each layer and encoding in the same way as in [17]. We use positional encoding to convert the input coordinates $\mathbf{x}$. For the implementation, we use an adaptive batch size that contains 64k samples at a time, which is 512 for 128 samples per ray with patch size one. We use different batch sizes for different settings since a too large batch size slows down convergence and a too small batch size slows down computation speed, as it cannot take advantage of the GPU.

5 Experiments

We first conduct several ablation studies on the number of samples needed for reconstruction at different levels of upsampling. Then we investigate the effect of different samples selected for each level of the pyramid in a multi-stage setup. Then we compare our method with state-of-the-art for static scenes. We also demonstrate that our model is suitable for deformable object and can be used for animating 3D humans.

5.1 Dataset and evaluation metrics

Following NeRF [17], we choose synthetic renderings of 8 scenes ('chair', 'drums', 'ficus', 'hotdog', 'lego', 'materials', 'mic' and 'ship') to compare with state-of-the-art. For each scene, 100 views are rendered for training and 400 views are rendered for testing, all at 800×800 resolution. We report PSNR (higher is better) for quantitative comparison. For the ablation studies, we choose the 'lego' scene. To demonstrate our model's ability to deal with deformable objects, we create a new synthetic sequence called 'Claudia' using traditional rendering techniques. We render 100 images from random viewpoints in standard T-pose.

ZJU dataset [**?**] is captured using a light stage, where 24 industrial cameras are

5.2 Comparison with state-of-the-art

5.3 Ablation studies

Vanilla sampling We first train several vanilla NeRF [17] models for different resolutions at resolution 50×50, 100×100, 200×200 and 400×400. We use a fixed number of samples for each resolution, 64 samples for 50×50, 128 for 100×100,

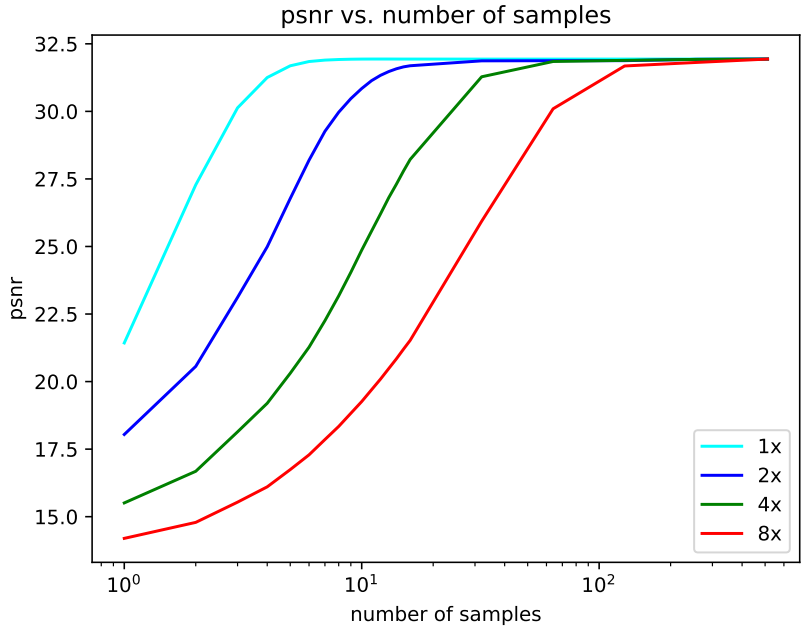

Fig. 3. Effect of number of samples: we show four scenarios on a two-stage pyramid with upsampling factors 1x, 2x,3x and 4x. The horizontal axis shows the number of samples selected based on the lower resolution probabilities.

and so on, doubling the depth resolution along with the image resolution. We tested the trained model in a of two-stage pyramid. For example, we could use the trained vanilla NeRF model of 200×200 as the first stage of the pyramid and the model of 400×400 as the second level. In this case there is a resolution change (see Figure 3 '2x'), where we use bilinear interpolation to upsample the first stage to the second stage. As shown in Figure 3, by varying the number of samples (the x-axis) of the second stage from 1 to the maximum number of samples, the PSNR (the y-axis) of results from the second stage changes accordingly. We found that the model can produce descent results without using the maximum number of samples. Figure 3 shows that 20, 50, 128 samples are needed for '2x', '4x', '8x' settings, respectively, to reach the peak performances.

Number of effective samples In this experiment, we design a four-stage pyramid and find out the best number of samples for each stage. We set the number of samples to $K^0 = 64$ in the lowest level in the pyramid. Then we test 1840 combinations for the second (K^1), third (K_2), and fourth (K_3) stage. For each combination $[K^0, K^1, K^2, K^3]$, we calculate the effective number of samples

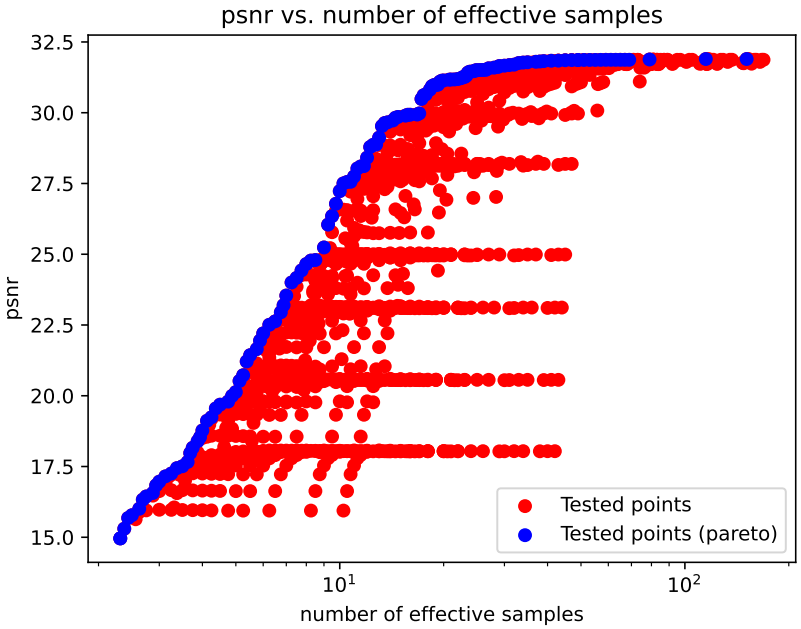

Fig. 4. Effect of pyramid sampling. Each point represents a four-stage pyramid with different number of samples for their corresponding stages.

through

$$K_{\text{eff}} = \sum_{n=0}^{3} (K^n/\mu_n^2), \tag{21}$$

with the scale μ and $K^n \leq 2 \cdot K^{n-1}, n = 1, 2, 3$. Figure 4 shows the scatter plot for all the combinations. The obvious trend is that more effective samples lead to better PSNR. Among the Pareto optimal points, there are a few PSNR jumps, the curve is not smooth. The jumps correspond to increasing the number of samples at earlier levels. The PSNR saturates at 32.5 effective samples, where the Pareto optimal sampling combination is $[K^0 = 64, K^1 = 24, K^2 = 20, K^3 = 16]$. We use this sampling combination for the first four stages of the pyramid training for all following experiments. We selected $K^4 = 6$ and $N^4 = 1024$ at 800×800 resolution. The number of effective samples are 11.875, which that the inference speedup is 16x, as NeRF that uses 192 samples. the training the speedup is , as progressive training adds overhead.

5.4 Comparison with SOTA on static scenes

In Table 1, we compare our method with the original NeRF [17] and PlenOctrees [33] on the 'ship' scene. NeRF takes 60 hours to reach the PSNR of 29.1,

Fig. 5. Qualitative results on Claudia data. The top row shows frontal views and the bottom shows side views.

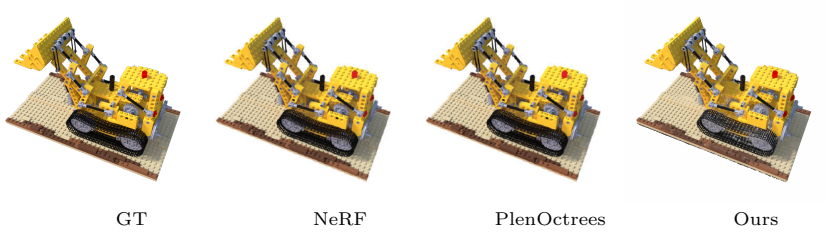

Fig. 6. Qualitative comparison with the methods from the original NeRF [17] and PlenOctrees [33] for the 'lego' scene.

PlenOctrees achieves that (29.1) with less time (16 hours), while our method can achieve a competitive performance with only 4.5 hours of training. Figure 6 shows qualitative comparison with NeRF [17] and PlenOctree [33] using the Lego scene in the blender dataset. Figure 7 shows our results on all eight scenes.

5.5 Animating virtual humans

Our model is trained with multi-view images on a synthetically rendered virtual asset 'Claudia' in T-pose with uniform background. Figure 5 shows qualitative results on synthesizing new poses. The experiment demonstrates that our model can speed up NeRF for animating with deformable objects, like 3D humans.

6 Conclusion

In this paper, we propose a novel method HIPNeRF, which speeds up NeRF training using importance sampling. During the volumetric rendering, we sample the NeRF network at different resolutions in a hierarchical pyramid. At the

Fig. 7. Our results on eight scenes. The first and third rows are the ground truth, the second and fourth rows are our results.

lowest resolution volume sampling follows the vanilla NeRF. For higher resolutions, we interpolate the probability densities (PDF) of samples of the lower resolution rendering, which allows us to sample points only at the most probable locations. By repeating the process we can achieve up to 13× speedup at full resolution. The approach carries over to the inference stage and also allows for dealing with scenarios containing highly deformable and articulated objects. We present the results of HIPNeRF on standard datasets, as well as on dynamic scenes where HIPNeRF is used to animate 3D virtual humans.

7 Societal and ethical impact

The proposed work addresses photo-realistic rendering which has great potential for bettering various aspects of our society including content creation for education, social interaction and entertainment to name a few. However, as with every new technological advancements we need to be aware of the ethical concerns which in the case of virtual and augmented reality range from manipulation (including consumer manipulation), privacy and data issue, to psychological cases of

desensitization and social isolation. As responsible developers of this technology we should make every effort to initiate and participate in open and transparent procedures that would prevent and mitigate these risks.

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. arXiv preprint arXiv:2103.13415 (2021)
2. Chen, Z., Zhang, H.: Learning implicit fields for generative shape modeling. In: CVPR. pp. 5939–5948 (2019)
3. Chibane, J., Bansal, A., Lazova, V., Pons-Moll, G.: Stereo radiance fields (srf): Learning view synthesis for sparse views of novel scenes. In: CVPR. pp. 7911–7920 (2021)
4. Cole, F., Genova, K., Sud, A., Vlasic, D., Zhang, Z.: Differentiable surface rendering via non-differentiable sampling. In: ICCV. pp. 6088–6097 (2021)
5. Cole, F., Genova, K., Sud, A., Vlasic, D., Zhang, Z.: Differentiable surface rendering via non-differentiable sampling. In: ICCV. pp. 6088–6097 (October 2021)
6. Garbin, S.J., Kowalski, M., Johnson, M., Shotton, J., Valentin, J.: Fastnerf: High-fidelity neural rendering at 200fps. arXiv preprint arXiv:2103.10380 (2021)
7. Hedman, P., Srinivasan, P.P., Mildenhall, B., Barron, J.T., Debevec, P.: Baking neural radiance fields for real-time view synthesis. In: ICCV (2021)
8. Hejrati, M., Ramanan, D.: Analysis by synthesis: 3d object recognition by object reconstruction. In: CVPR. pp. 2449–2456 (2014)
9. Kangle Deng, Andrew Liu, J.Y.Z., Ramanan, D.: Depth-supervised nerf: Fewer views and faster training for free. arXiv preprint arXiv:2107.02791 (2021)
10. Lindell, D.B., Martel, J.N., Wetzstein, G.: Autoint: Automatic integration for fast neural volume rendering. In: CVPR. pp. 14556–14565 (2021)
11. Liu, L., Gu, J., Lin, K.Z., Chua, T.S., Theobalt, C.: Neural sparse voxel fields. arXiv:2007.11571 (2020)
12. Lombardi, S., Simon, T., Saragih, J., Schwartz, G., Lehrmann, A., Sheikh, Y.: Neural volumes: Learning dynamic renderable volumes from images. arXiv preprint arXiv:1906.07751 (2019)
13. Lombardi, S., Simon, T., Schwartz, G., Zollhoefer, M., Sheikh, Y., Saragih, J.: Mixture of volumetric primitives for efficient neural rendering. ACM TOG (2021)
14. Loper, M.M., Black, M.J.: Opendr: An approximate differentiable renderer. In: ECCV. pp. 154–169. Springer (2014)
15. Martin-Brualla, R., Radwan, N., Sajjadi, M.S., Barron, J.T., Dosovitskiy, A., Duckworth, D.: Nerf in the wild: Neural radiance fields for unconstrained photo collections. arXiv preprint arXiv:2008.02268 (2020)
16. Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy networks: Learning 3d reconstruction in function space. In: CVPR. pp. 4460–4470 (2019)
17. 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. pp. 405–421. Springer (2020)
18. Neff, T., Stadlbauer, P., Parger, M., Kurz, A., Alla Chaitanya, C.R., Kaplanyan, A., Steinberger, M.: Donerf: Towards real-time rendering of neural radiance fields using depth oracle networks. arXiv e-prints pp. arXiv–2103 (2021)
19. Niemeyer, M., Geiger, A.: Giraffe: Representing scenes as compositional generative neural feature fields. In: CVPR. pp. 11453–11464 (2021)
20. Oechsle, M., Niemeyer, M., Reiser, C., Mescheder, L., Strauss, T., Geiger, A.: Learning implicit surface light fields. In: 2020 International Conference on 3D Vision (3DV). pp. 452–462. IEEE (2020)

21. Osman, A.A.A., Bolkart, T., Black, M.J.: STAR: A sparse trained articulated human body regressor. In: European Conference on Computer Vision (ECCV). pp. 598–613 (2020), https://star.is.tue.mpg.de
22. Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation. In: CVPR. pp. 165–174 (2019)
23. Park, K., Sinha, U., Barron, J.T., Bouaziz, S., Goldman, D.B., Seitz, S.M., Martin-Brualla, R.: Nerfies: Deformable neural radiance fields. In: ICCV. pp. 5865–5874 (2021)
24. Peng, S., Dong, J., Wang, Q., Zhang, S., Shuai, Q., Zhou, X., Bao, H.: Animatable neural radiance fields for modeling dynamic human bodies. In: ICCV (2021)
25. Peng, S., Zhang, Y., Xu, Y., Wang, Q., Shuai, Q., Bao, H., Zhou, X.: Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans. In: CVPR. pp. 9054–9063 (2021)
26. Pumarola, A., Corona, E., Pons-Moll, G., Moreno-Noguer, F.: D-NeRF: Neural Radiance Fields for Dynamic Scenes. In: CVPR (2021)
27. Reiser, C., Peng, S., Liao, Y., Geiger, A.: Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. arXiv preprint arXiv:2103.13744 (2021)
28. Sun, C., Sun, M., Chen, H.T.: Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. arXiv preprint arXiv:2111.11215 (2021)
29. Tancik, M., Mildenhall, B., Wang, T., Schmidt, D., Srinivasan, P.P., Barron, J.T., Ng, R.: Learned initializations for optimizing coordinate-based neural representations. In: CVPR. pp. 2846–2855 (2021)
30. Tewari, A., Fried, O., Thies, J., Sitzmann, V., Lombardi, S., Sunkavalli, K., Martin-Brualla, R., Simon, T., Saragih, J., Nießner, M., et al.: State of the art on neural rendering. In: Computer Graphics Forum. vol. 39, pp. 701–727. Wiley Online Library (2020)
31. Wizadwongsa, S., Phongthawee, P., Yenphraphai, J., Suwajanakorn, S.: Nex: Real-time view synthesis with neural basis expansion. In: CVPR. pp. 8534–8543 (2021)
32. Yariv, L., Kasten, Y., Moran, D., Galun, M., Atzmon, M., Ronen, B., Lipman, Y.: Multiview neural surface reconstruction by disentangling geometry and appearance. NeurIPS **33** (2020)
33. Yu, A., Li, R., Tancik, M., Li, H., Ng, R., Kanazawa, A.: Plenoctrees for real-time rendering of neural radiance fields. arXiv preprint arXiv:2103.14024 (2021)
34. Yu, A., Ye, V., Tancik, M., Kanazawa, A.: Pixelnerf: Neural radiance fields from one or few images. In: CVPR. pp. 4578–4587 (2021)