ATM-NeRF: Accelerating Training for NeRF Rendering on Mobile Devices via Geometric Regularization

Yang Chen, Lin Zhang, *Senior Member, IEEE,* Shengjie Zhao, *Senior Member, IEEE,* and Yicong Zhou, *Senior Member, IEEE*

***Abstract*—Recently, an increasing number of researchers have been dedicated to transferring the impressive novel view synthesis capability of Neural Radiance Fields (NeRF) to resource-constrained mobile devices. One common solution is to pre-train NeRF and bake it into textured meshes which are well supported by mobile graphics hardware. However, the training process of existing methods often requires several hours even with multiple high-end NVIDIA V100 GPUs. The underlying reason is that these schemes mainly rely on photometric rendering loss, neglecting the geometric relationship between the pre-trained NeRF and the baked results. Standing on this point, we present ATM-NeRF (Accelerating Training for Mobile rendering based on NeRF), which is the first to apply effective geometric regularization constraints during both the pre-training and the baking training stages for faster convergence. Specifically, in the initial NeRF pre-training stage, we enforce consistency of the multi-resolution density grids representing the scene geometry to mitigate the shape-radiance ambiguity problem to some extent, achieving a coarse mesh with smoothness. In the second stage, we utilize the positions and geometric features of 3D points projected from the pre-trained posed depths to provide geometric supervision for joint refinement of geometry and appearance of the coarse mesh. As a result, our ATM-NeRF achieves comparable rendering quality to MobileNeRF with a training speed that is about $30\times \sim 70\times$ faster while maintaining finer structure details of the exported mesh. Our source code and the demo video have been released at https://cslinzhang.github.io/ATM-NeRF/.**

***Index Terms*—Neural Radiance Fields, mobile rendering, novel view synthesis, image reconstruction.**

I. Introduction

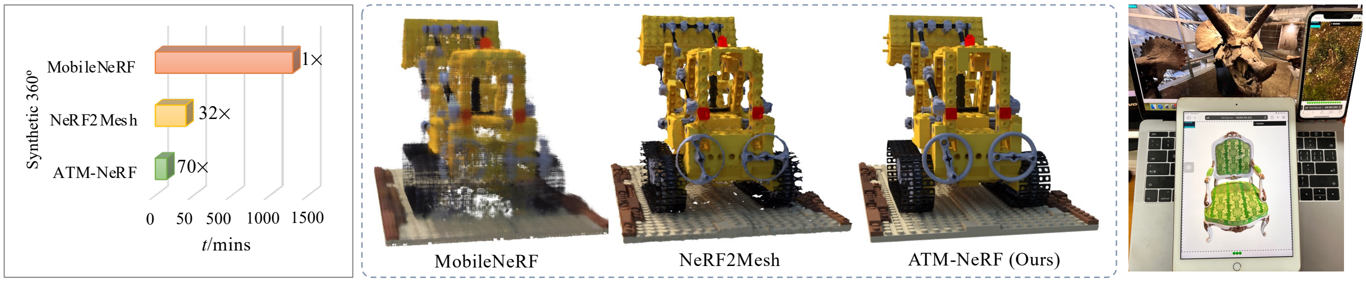
In recent years, Neural Radiance Fields (NeRF) [1] have gained significant popularity as an innovative solution to the task of novel view synthesis [2]–[4]. Its impressive ability to render scenes with photorealistic details has contributed to its growing acclaim. Subsequently, numerous follow-up works have sprung up to speed up its training [5]–[10], rendering [11]–[22], or expand its application scenarios [23]–[35]. Motivated by the above successful works in the NeRF field, more and more researchers are striving to introduce it into rendering on mobile devices [36] to serve Augmented Reality (AR) or Virtual Reality (VR) applications. However, the volume rendering algorithm of NeRF involves high computational complexity and mismatches the capacity of mobile graphic hardware. Existing solutions to this problem mainly fall into two categories, methods based on Neural Light Field (NeLF) [37]–[39] and methods based on NeRF baking [10], [11], [16], [18], [20], [49]–[51]. In the following, we will analyze their advantages and limitations in detail. Methods based on NeLF distill the NeRF representation into NeLF through a teacher-student model. Compared with NeRF where the colors of multiple sampling points along a ray need to be predicted for volume rendering, NeLF only requires one forward pass per ray through the neural field to obtain the final pixel color. Therefore, NeLF significantly reduces the computational cost of rendering and is mobile-friendly. Unfortunately, the distillation from NeRF to NeLF suffers from slow convergence for the following two reasons. On the one hand, NeLF directly takes a parameterized ray as input, which makes it hard to learn to predict the appearance of adjacent rays. Considering that adjacent pixels may have different appearances, especially in regions with high image gradients, how to distinguish adjacent rays with close positions in parameterization is an essential yet tricky issue to be explored. On the other hand, NeLF obtains fewer training samples than NeRF on the same amount of training images without ray sampling. Data augmentation is usually needed to prepare sufficient training data for the convergence of the NeLF model, which is undoubtedly cumbersome and time-consuming.

NeRF baking aims to train a NeRF model in the *NeRF pre-training* stage and store its geometric and photometric information in a new scene representation through the *baking training* stage. The most acceptable representation for mobile devices is polygonal meshes [10], [20], [49], whose rendering has been well supported by nearly all mobile devices. The baking output of these mesh-based schemes typically consists of a polygonal mesh, corresponding texture maps, and a lightweight neural shader. During inference, the textured mesh can be efficiently rendered under the mature rasterization pipeline and the neural shader decodes the features stored in the texture maps into view-dependent colors to achieve real-time rendering. While NeRF baking is efficient in terms of

This work was supported in part by the National Natural Science Foundation of China under Grant 62272343 and Grant 61936014; in part by the Shuguang Program of Shanghai Education Development Foundation and Shanghai Municipal Education Commission under Grant 21SG23; and in part by the Fundamental Research Funds for the Central Universities. (*Corresponding author: Lin Zhang.*)

Yang Chen, Lin Zhang, and Shengjie Zhao are with the School of Software Engineering, Tongji University, Shanghai 201804, China, and also with the Engineering Research Center of Key Software Technologies for Smart City Perception and Planning, Ministry of Education, Shanghai 201804, China. (email: {2011439, cslinzhang, shengjiezhao}@tongji.edu.cn).

Yicong Zhou is with the Department of Computer and Information Science, University of Macau, Macau 999078, China (e-mail: yicongzhou@um.edu.mo).



(a) (b) (c)

Fig. 1. **Training efficiency and compatibility of our ATM-NeRF.** All the training stages were run on our desktop computer with a single NVIDIA 3090 Ti. (a) Comparison of training time with typical competitors focusing on mobile NeRF rendering on the Synthetic 360° dataset along with the speedup factors compared with MobileNeRF. (b) Rendering results of compared methods after 10 minutes' training. Obviously, our ATM-NeRF achieves the most photorealistic rendering quality with the fastest training speed. (c) Compatibility of ATM-NeRF on a variety of common devices.

rendering speed, most existing schemes still cost *several hours* for training. The bottleneck is how to refine the mesh geometry and bake the proper appearance to each vertex at the same time. To address this problem, most existing methods tend to refine the baked mesh with a discrete optimization strategy by adding and removing mesh vertices or faces, which is slow and memory-intensive. Instead, our geometric regularizers enable mesh refinement through continuous gradient-based optimization to make gradient updates directly to vertex positions. This guarantees faster and more stable baking training.

On account of the limitations aforementioned, as far as we know, there is still no existing NeRF method specially designed for training acceleration for mobile rendering. To fill in this gap to some extent, we propose the geometry-aware ATM-NeRF under the mainstream NeRF baking framework with efficient geometric regularizations and our contributions are mainly threefold:

1) ATM-NeRF is proposed as the first work to incorporate geometric constraints to accelerate NeRF training for mobile rendering. These geometric constraints guide the optimization of the density field, mesh vertices and geometric features, respectively. As a result, our ATM-NeRF achieves comparable rendering quality to MobileNeRF with a rendering speed that is about $30\times \sim 70\times$ faster.
2) In NeRF pre-training, a simple yet effective density regularization strategy, that utilizes the density field of an underfitted grid-based NeRF to regularize that of the current model, is introduced. This regularized optimization of scene geometry alleviates photometric overfitting and the shape-ambiguity problem it causes in NeRF training.
3) Two novel depth-based regularization terms are constructed in the baking training stage. They provide continuous gradient updates to the mesh vertices, without mesh division or decimation, thereby promoting a fast and stable convergence in the bottleneck baking training stage.

II. Related Work

Neural Radiance Fields (NeRF) [1] have constituted a remarkable breakthrough in synthesizing novel views of complex scenes. Thanks to the implicit volumetric representation, NeRF achieves state-of-the-art rendering performance with photorealistic details. However, NeRF needs to query MLPs multiple times for densities and colors of all sampling points for alpha-composition, which can be time-consuming during both *training* and *inference*. To tackle this challenge, numerous advanced techniques have emerged, including notable works specifically designed for *NeRF rendering on mobile devices*.

Efficient NeRF Training. DS-NeRF [5] assumes that most of a scene's geometry consists of empty space and opaque surfaces. It takes advantage of the depths from structure-from-motion (SFM) as supervision for fast training. This geometric supervision makes DS-NeRF render better images given fewer training views while training 2-3× faster. DVGO [6] believes that explicit expression has a faster convergence speed compared to implicit expression of NeRF, so they model the geometry and the appearance of the scene as density voxel grids and feature voxel grids, respectively. In particular, the feature grids collaborate with a shallow MLP to obtain view-dependent colors from different views. Directly optimizing these density and feature grids reduces the rendering time of NeRF to 15 minutes. Fridovich-Keil *et al.* [8] proposed a view-dependent sparse voxel grid without any neural networks, named Plenoxels. Each voxel stores spherical harmonic coefficients, which are trilinearly interpolated to quickly compute the color and opacity of each sampled point. To achieve high efficiency on a single GPU, Plenoxels also prunes the empty voxels and follows a coarse-to-fine optimization strategy. TensoRF [9] represents the neural radiance as feature grids. Each feature grid is modeled as a 4-channel tensor, which stores the 3D position of a sampling point and the dimension of features. Then this 4D tensor can be factorized into multiple compact low-rank tensor components to compress high-dimensional data, significantly reducing the memory usage. The most outstanding work for speeding is the famous Instant-NGP [7], which replaces the frequency position encoding used in NeRF with a multi-resolution hash encoding structure with optimizable parameters, which is trivial to parallelize on modern GPUs. Making full use of the parallelism, Instant-NGP implemented the whole system using fully-fused CUDA kernels, producing reliable rendering results in seconds.

3D Gaussian Splatting. Recently, 3D Gaussian Splatting (3D-GS) [40] has emerged as a significant advancement in improving both training efficiency and rendering performance for novel-view synthesis. Compared with implicit NeRF, 3D-GS employs an explicit radiance field-based scene representation using a large number of 3D anisotropic balls, each modeled using a 3D Gaussian distribution.

Although the highly parallel optimization of these Gaussian balls significantly improves training efficiency, the parameters within millions of Gaussians used to represent the scene require a huge amount of memory space. Therefore reducing memory usage without sacrificing rendering quality is critical for the development of 3D-GS. Scaffold-GS [41] utilizes the initialization points in SfM to construct a sparse grid of anchor points, each of which has a set of learnable Gaussians attached to it. Aggregated gradients of these neural Gaussians guide the growth of anchor points while pruning strategies are developed to eliminate trivial anchors. Compact3D [42] tries to reduce the memory usage of 3D-GS by not only reducing the number of Gaussians but also compressing the Gaussian parameters based on a vector quantization algorithm. Besides, as a commonly used strategy for compression, several octree-based algorithms [43], [44] were developed for more efficient representation.

Mesh reconstruction is another challenging yet essential task for 3D-GS to support downstream visual applications. SuGaR [45] introduces a regularization term that encourages the alignment between 3D Gaussians and the surface of the scene. Then the mesh is extracted from these aligned Gaussians by Poisson reconstruction, which will be further refined with a refinement strategy. Chen *et al*. [46] proposed NeuSG, which combines NeuS [47] and 3D-GS by joint optimization to achieve highly detailed surface recovery. 3DGSR [48] demonstrates the advantage of incorporating an implicit signed distance field into 3D-GS in ensuring mesh reconstruction quality. Despite the above efforts to solve surface reconstruction from 3D-GS, the reconstructed mesh still suffers from aliasing and hole issues.

NeRF Baking for Fast Inference. NSVF [11] maintains a sparse voxel octree for storing voxel-bounded implicit fields, which ensures a more effective ray sampling strategy by skipping unnecessary points for fast rendering. It is usually more than 10 times faster than the original NeRF while achieving higher rendering quality. In SNeRG [50], the scene appearance is divided into the view-invariant diffuse component and the view-dependent specular component. Both components are stored in sparse voxel grids for efficient look-up. During inference, deferred rendering is used for decoding the grid features with the input view direction to achieve real-time rendering. Yu *et. al.* [18] proposed a novel data structure called PlenOctree to replace the traditional NeRF representation. Specifically, leaf nodes of this Octree structure store the density values and spherical harmonics required for scene representation. In addition, to convert NeRF representation to PlenOctree more directly, this study proposed an improved NeRF model (NeRF-SH) to generate spherical harmonic representations without inputting data from different views into the network. Compared to NeRF, PlenOctree renders more photorealistic images with a rendering speed that is over 3000 times faster. KiloNeRF [16] demonstrates that real-time rendering can be achieved by replacing NeRF's deep MLP networks with thousands of tiny MLPs. To train these tiny MLPs, the author proposed a three-stage training strategy. In the first stage, an original NeRF was trained and then a point-to-point distillation was conducted in the second stage to transfer the prediction results of the native NeRF MLPs to the tiny MLPs. Finally, these tiny MLPs are further fine-tuned with the empty space skipping strategy during training and the early ray termination strategy during rendering. MERF [51] works by projecting 3D samples onto three 2D projections that correspond to the cardinal axes, which are called tri-planes. To more efficiently deal with the large-scale unbounded scene, the authors of MERF modified the contraction function in [27] so that ray-AABB intersections can be computed trivially. During training, Instant-NGP [7] is used to predict the values of grid points and tri-plane pixels. Then the values of sampled points are obtained through trilinear or bilinear interpolation before volume rendering, ensuring a lossless baking result. Although the above methods can achieve real-time rendering, few of them can be applied to low-resource devices, especially mobile devices such as tablets and smartphones due to their voxel representation of the baking output, which is memory consuming.

NeRF for Mobile Devices. BakedSDF [10] is aimed at reconstructing high-quality meshes in large, unbounded real-world scenes through three stages. In the first stage, the scene was constructed by combining the benefits of mip-NeRF 360 [27] for representing unbounded scenes with the well-behaved surface properties of VolSDF's [52] hybrid volume-surface representation. Then in the second stage, this hybrid representation was baked into a high-resolution mesh and a visibility culling strategy was implemented to avoid generating unnecessary meshes in invisible areas. Finally, in the third stage, BakedSDF assigned a diffuse component and a set of spherical Gaussian lobes for each vertex of the mesh to construct their view-dependent appearance. Although BakedSDF can generate high-quality grids to support various downstream applications, it still needs a laptop for rendering and its webviewer cannot be viewed on a common smartphone.

MobileNeRF [20] is the first to achieve real-time rendering on all mobile devices based on NeRF, in which a new NeRF representation based on polygonal meshes to be rendered under the traditional polygon rasterization pipeline is proposed. It consists of two training stages and an export stage to output the textured mesh along with a neural shader. Specifically, in the first stage, the scene was initialized as a grid mesh with a fixed topology, whose vertex positions are trainable and optimized during the whole training process, and the intersection points of the camera ray with this initial mesh were regarded as the sampling 3D points to input into the NeRF model. Considering that the traditional rasterization pipeline shows a slow speed in coping with semi-transparent objects, the opacities of all intersection points predicted by the trained NeRF model are binarized to 0 or 1 for fast rendering. After training, the optimized mesh was exported as an OBJ file and the features of the intersection points were stored in the PNG file. Also, the weights of the shallow MLP taking the features and the view direction as inputs are stored in a JSON file for deferred rendering. Although MobileNeRF shows excellent rendering quality and speed even in smartphones, it takes over 20 hours for training for its time-consuming traverse of the mesh triangles in a z-buffer order and the ray intersection operation.

NeRF2Mesh [49] outputs a delicate mesh with specular and diffuse maps for more intuitive rendering. A coarse mesh is produced by a grid-based NeRF in the first stage and is refined via a novel adaptive refinement algorithm in the second stage. In terms of this refinement strategy, the mesh is rasterized under a differentiable framework and these rasterization points are utilized as the input of the network to modify the mesh vertices along with the network parameters. Meanwhile, the faces of the mesh are also decimated or divided adaptively according to the rendering error. Thanks to the decomposition of the scene appearance, the training time of NeRF2Mesh is nearly 1 hour. However, it sacrifices the rendering quality for the geometric structure and the rendering image may suffer from noise and needs anti-aliasing when rendering on the HTML pages.

In order to reduce latency when running neural rendering on mobile devices, the goal of MobileR2L [21] is to perform neural network forward propagation only once to obtain synthesized images. However, existing neural light field network designs require a significant amount of memory to render high-resolution images, which exceeds the memory limitations of mobile devices. To deal with this problem, they do not pass the same number of rays as pixels forward at once, but only pass a portion of the rays forward and upsample the output into a high-resolution image to learn all pixels by a newly introduced super-resolution module. Although it can achieve outperforming rendering speed on mobile devices due to the one-passing forward mechanism and the well-designed network, the convergence of the distillation model is extremely slow due to the characteristic of NeLF as analyzed in Sec. I, leading to a training time of over 1 day.

The latest work based on NeLF to speed up mobile training and rendering is LightSpeed [39]. It revisits the classic light slab (two-plane) representation, which is the preferred representation for interpolating between light field views. This low-dimensional ray representation enables the use of feature grids to learn 4D ray space, which significantly speeds up NeLF's training and rendering in LightSpeed. Although LightSpeed provides superior rendering quality and achieves a significantly improved training speed compared to the previous light field method [21], it still needs dozens of hours due to the difficult convergence of networks brought by the nature of NeLF and the same time-consuming training scheme, teacher-student model based distillation, as MobileR2L [21]. Also, since a single light slab is only suitable for modeling a frontal scene and cannot capture light rays that are parallel to the planes, LightSpeed costs extra efforts to composite multiple light slab representations to learn the full light field for non-frontal scenes. This divide-and-conquer strategy brings the inconvenience of manually setting hyperparameters (such as the number of partitions) when coping with non-frontal scenes.

III. Preliminaries of NeRF Baking

NeRF baking is currently the state-of-the-art framework for NeRF rendering on mobile devices. In general, NeRF baking is to extract and store necessary attributes (position, features, colors, *etc.*) of a precomputed NeRF into a new scene representation for fast rendering. During inference, the view-dependent color can be efficiently "recovered" by a neural decoder namely neural deferred shader from the baking outputs instead of large amounts of forward passing to MLPs in the original NeRF. The most commonly used baking representation currently is polygonal mesh with texture maps, as it can be well supported by modern graphic rendering pipelines. Next, we will introduce the typical framework of mesh-based NeRF baking containing two stages, which we named "NeRF pre-training" and "baking training", denoted by S1 and S2 respectively for brevity.

NeRF Pretraining. Motivated by the awesome rendering performance of NeRF, which demonstrates the potential of its implicit volumetric representation, a grid-based NeRF is usually pre-trained in the first stage to provide reliable rendering priors. As in NeRF, the typical rendering loss based on photometric differences is utilized as the dominant guidance in this stage. Concretely, the scene representation is optimized by penalizing the distance between the predicted colors through volume rendering with the ground truth colors of training images:

$$\mathcal{L}_{\mathbf{C}}^{S1} = \mathbb{E}_{\mathbf{r}} \|\mathbf{C}^{S1}(\mathbf{r}) - \mathbf{C}_{gt}(\mathbf{r})\|_2^2, \tag{1}$$

where $\mathbf{r}$ is the sampling ray formulated by $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, where $\mathbf{o}$ and $\mathbf{d}$ represent the origin and the direction of the ray respectively. The predicted color $\mathbf{C}^{S1}(\cdot)$ is obtained by alpha-compositing the radiance $\mathbf{c}_k$ at the depth t_k:

$$\mathbf{C}^{S1}(\mathbf{r}) = \sum_{k=1}^{K} T_k \alpha_k \mathbf{c}_k^{S1}, \quad T_k = \prod_{l=1}^{k-1}(1-\alpha_l), \tag{2}$$

where opacity α_k represents the point-wise weight and can be given by $\alpha_k = 1 - exp(-\sigma_k \delta_k)$. $\delta_k = t_{k+1} - t_k$ is the distance between adjacent samples on $\mathbf{r}$. The density σ_k and view-dependent appearance $\mathbf{c}_k$ can be queried by passing the 3D position of sampling points $\mathbf{p}_k = \mathbf{r}(t_k)$ to three important MLPs denoted by $\mathcal{D}_\theta, \mathcal{F}_\theta$ and $\mathcal{H}_\theta$ respectively:

$$\sigma_k = \mathcal{D}_\theta(\mathbf{p}_k; \boldsymbol{\theta}_{\mathcal{D}}), \tag{3}$$

$$\mathbf{f}_k^{S1} = \mathcal{F}_\theta(\mathbf{p}_k; \boldsymbol{\theta}_{\mathcal{F}}), \tag{4}$$

$$\mathbf{c}_k^{S1} = \mathcal{H}_\theta(\mathbf{f}_k^{S1}, \mathbf{d}; \boldsymbol{\theta}_{\mathcal{H}}), \tag{5}$$

where $\boldsymbol{\theta}$ is the set of MLP parameters and the subscript indicates the corresponding network it belongs to. The first part in $(\cdot)$ gives the input while the other one after the semicolon stands for the parameters to be optimized. $\mathcal{D}_\theta$ is the density network producing the density of the sampling points, determining the weight of each sampling point in accumulation. $\mathcal{F}_\theta$ only takes the 3D position as the input to produce the view-invariant geometric features. These features are stored in images as texture maps of the mesh. The small decoder MLP $\mathcal{H}_\theta$ acts as the deferred shader in the traditional rendering pipeline, which takes the direction and features output from $\mathcal{F}_\theta$. The deferred rendering strategy is the key to real-time (usually in milliseconds) NeRF rendering on mobile devices. After S1, a coarse mesh $\mathcal{M}$ can be generated as a geometric initialization for scene representation either by

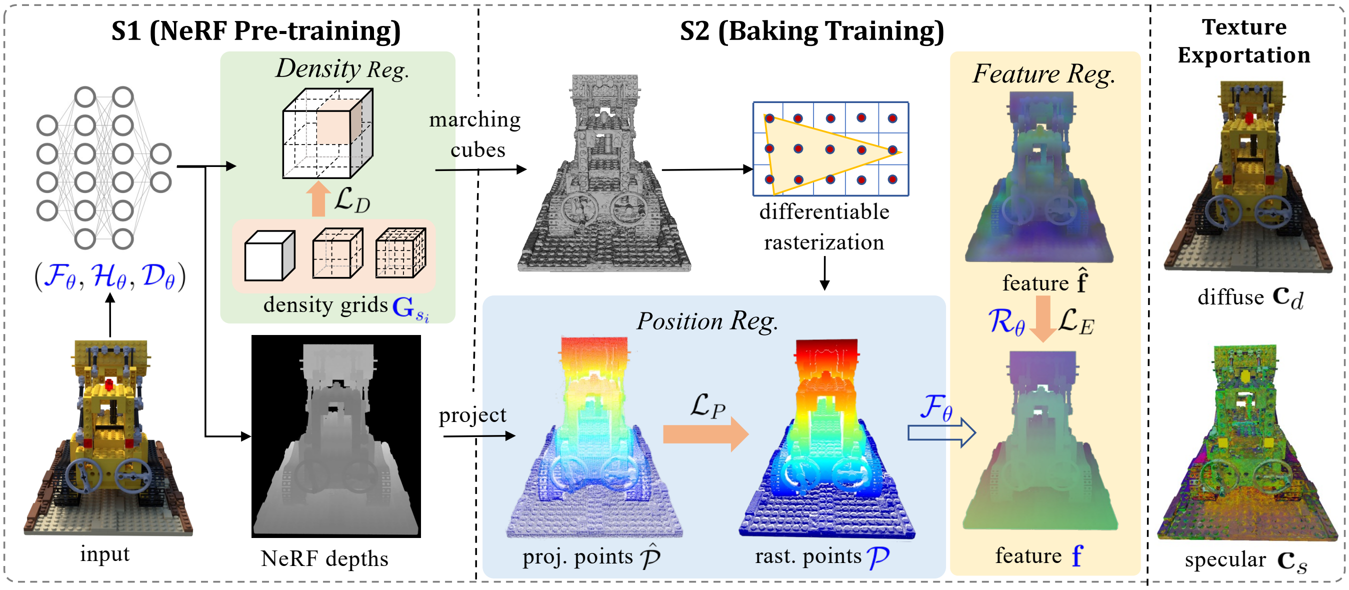


Fig. 2. **System overview of ATM-NeRF.** Posed RGB images are the only input required for our method, and after a two-stage training, we obtain a textured mesh that includes both specular and diffuse components. In addition to the classic rendering loss, important geometric regularization modules are integrated into the whole training, labeled with light-colored rectangles (green, blue and yellow). Optimizable parameters are colored in dark blue. The corresponding geometric supervision loss terms are marked with orange arrows. These regularization terms facilitate an accelerated convergence of the network without additional priors.

marching cubes [53] or based on a pre-defined mesh topology [20].

Baking Training. This stage is responsible for jointly refining the geometry and appearance of the coarse mesh to ensure the correct rendering results under the rasterization pipeline instead of volume rendering. It is worth mentioning that rendering pipelines implemented in typical hardware do not natively support semi-transparent meshes. Therefore, we have to rasterize the mesh and force the appearance of the intersection point to approach the ground truth rendering color. Mathematically, denoting the intersection point of an arbitrary ray $\mathbf{r}$ with the mesh $\mathcal{M}$ as $\mathbf{p}_{int}$, its predicted color can be given as:

$$\mathbf{f}_{int}^{S2} = \mathcal{F}_\theta(\mathbf{p}_{int}; \boldsymbol{\theta}_\mathcal{F}, \boldsymbol{\theta}_\mathcal{M}), \tag{6}$$

$$\mathbf{C}^{S2}(\mathbf{r}) = \mathcal{H}_\theta(\mathbf{f}_{int}^{S2}, \mathbf{d}; \boldsymbol{\theta}_\mathcal{H}, \boldsymbol{\theta}_\mathcal{M}), \tag{7}$$

where $\boldsymbol{\theta}_\mathcal{M}$ is the optimizable variables of $\mathcal{M}$ (such as vertex positions, faces, topology, *etc.*), In this stage, the optimization is still guided by the rendering loss:

$$\mathcal{L}_\mathbf{C}^{S2} = \mathbb{E}_\mathbf{r} \|\mathbf{C}^{S2}(\mathbf{r}) - \mathbf{C}_{gt}(\mathbf{r})\|_2^2. \tag{8}$$

After this stage, we can obtain the refined mesh and its feature/texture maps. Also, the weights of the neural deferred shader $\mathcal{H}$ can be recorded in advance to decode the view-dependent color during inference.

IV. Method

A. Overview

Following the classic framework of NeRF baking in Sec. III, our method comprises two stages: NeRF pre-training and baking training (denoted by S1 and S2 respectively for brevity), as depicted in Fig. 2. Existing schemes mainly relying on photometric supervision exhibit slow convergence during training. To overcome this problem, we introduce geometric regularization to both stages, not only accelerating training speed especially for the bottleneck baking stage, but also achieving comparable rendering quality with the state-of-the-art MobileNeRF. Technically, in S1, posed RGB images are required as input to train a grid-based NeRF to obtain a coarse mesh. During this stage of training, we utilize an underfitting grid-based NeRF storing scene densities with multiple resolutions for density regularization. After this stage, depth maps generated from the pre-trained NeRF corresponding to each training image are also stored for future use. In S2, the coarse mesh is rasterized under a differentiable rasterization framework, through which its geometry and appearance are optimized jointly. The posed depth maps from S1 provide geometric supervision in terms of the position and the geometric embeddings of the coarse mesh for refinement. Finally, we can obtain the refined mesh with diffuse and specular components, as well as a neural deferred shader for decoding view-dependent appearance, which is ready-to-use for real-time mobile rendering.

B. NeRF with Density Regularization (S1)

In S1, we train a grid-based NeRF model as in [49], taking advantage of its well-performing volume rendering algorithm to provide the *appearance* priors of the scene. Also, the density field of NeRF produces a topologically reliable *geometry* for further refinement.

Appearance. We decompose the appearance into view-independent diffuse reflection and view-dependent specular reflection as in [49]. Mathematically, the appearance $\mathbf{c}$ of an arbitrary 3D point $\mathbf{p}$ along the sampling ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, can be given by:

$$\mathbf{c}_d^{S1}, \mathbf{f}^{S1} = \mathcal{F}_\theta(\mathbf{p}; \boldsymbol{\theta}_\mathcal{F}), \tag{9}$$

$$\mathbf{c}_s^{S1} = \mathcal{H}_\theta(\mathbf{f}^{S1}, \mathbf{d}; \boldsymbol{\theta}_\mathcal{H}), \tag{10}$$

$$\mathbf{c}^{S1} = \mathbf{c}_d^{S1} + \mathbf{c}_s^{S1}, \tag{11}$$

where the subscripts "d" and "s" denote the diffuse and specular components, respectively. The index k is omitted for brevity. $\mathbf{c}_d$ is only dependent on the point position and $\mathbf{c}_f$ can be determined by both the diffuse feature $\mathbf{f}^{S1}$ and the direction $\mathbf{d}$, which is consistent with perception.

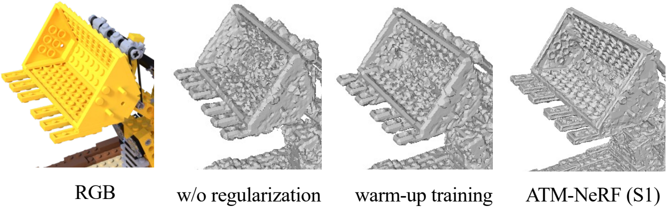


Fig. 3. **The effectiveness of the density regularization on the mesh geometry**. From left to right, we presented the GT RGB image, meshes generated by the variant without any regularization, the warm-up training and our ATM-NeRF respectively. For a clearer illustration, we do not employ any mesh post-processing techniques after marching cubes.

Geometry. As analyzed in NeRF++ [33], optimizing the 5D function in NeRF from a set of training images can encounter critical degenerate solutions that fail to generalize to novel test views, in the absence of any regularization. Such phenomena are encapsulated in the *shape-radiance ambiguity*, wherein the training field can fit training images perfectly for an incorrect geometry, leading to unsatisfactory rendering performance in novel views. To deal with this problem, we employ a simple yet effective density mesh regularization to utilize the density field of an underfitted grid-based NeRF model to regularize that of the current model. In detail, we conduct a warm-up training [54], [55] on a grid-based NeRF [7] for a few iterations only under photometric supervision to witness the abrupt decrease of rendering loss. Then we partition the scene into multi-resolution grids with three different grid sizes (128, 192 and 256) denoted by $\hat{\mathbf{G}}_{s_1}, \hat{\mathbf{G}}_{s_2}, \hat{\mathbf{G}}_{s_3}$ respectively, where each grid cell records the density σ of grid center point $\mathbf{p}_c$ predicted by this warm-up model:

$$\hat{\mathbf{G}}_{s_i}(\mathbf{p}_c) = \hat{\mathcal{D}}_\theta(\mathbf{p}_c; \hat{\boldsymbol{\theta}}_\mathcal{D}), \quad i = 1, 2, 3, \tag{12}$$

where $\hat{\mathcal{D}}_\theta$ is the density network in the warm-up stage parameterized by $\hat{\boldsymbol{\theta}}_\mathcal{D}$. Afterward, we guide the formal training by enforcing consistency between the predicted density field $\mathbf{G}_{s_i}(\mathbf{p}_c) = \mathcal{D}(\mathbf{p}_c; \boldsymbol{\theta}_{\mathcal{D}_\theta})$ and the warm-up one. Thus, the final density regularization loss term can be formulated by:

$$\mathcal{L}_D = \sum_{i=1}^{3} \sum_{\mathbf{p}_c} \|\hat{\mathbf{G}}_{s_i}(\mathbf{p}_c) - \mathbf{G}_{s_i}(\mathbf{p}_c)\|_2^2. \tag{13}$$

As shown in Fig. 3, although the mesh produced in the warm-up stage is coarse, it effectively regularizes the scene geometry via $\mathcal{L}_D$, improving the reconstruction quality of the mesh surface with fine details compared with the variant of ATM-NeRF without any regularization. Also, our experiments demonstrate that $\mathcal{L}_D$ can also bring benefit to the rendering quality based on the mechanism that better geometry may also encourages better appearance due to their correlation in joint optimization.

Loss Function. The total loss function of this stage can be formulated as:

$$\mathcal{L}^{S1} = \mathcal{L}_C^{S1} + \beta_1 \mathcal{L}_D, \tag{14}$$

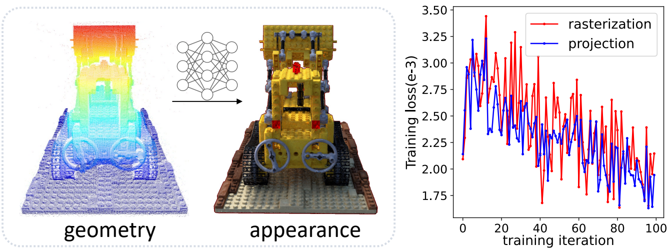
where β is set to 0.0001 and $\mathcal{L}_C^{S1}$ is given in Eq. 1.

After this stage, we generate a coarse mesh by applying the marching cubes algorithm [53] to the density grid $\mathbf{G}_{s_1}$ for further refinement. This coarse mesh provides a satisfactory geometric initialization of the scene with the assistance of our density regularization. Besides, the depth maps from all training views are also stored, based on which can recover a coarse point cloud of the scene to provide efficient point-to-point supervision when modifying the mesh vertex in the next stage.

C. Depth Guided Baking Training (S2)

Baking training in S2 refines the geometry and the appearance of the coarse mesh under a differentiable rasterization framework to approach the ground truth rendering results. However, in practice, this training process can be really time-consuming as the bottleneck of training efficiency since it forces the distribution of sampling points on the ray to be single-hot, which is a more strict constraint for the network. Although existing schemes introduce extra loss terms such as the binary loss [20] for the opacity or the smoothness loss [49] for mesh refinement to assist convergence, their training speed still needs improvement. Instead, we introduce depth-based regularization terms in S2 to make full use of the geometric priors obtained in S1 and dig into the geometric relationship between the pre-trained mesh and the refined one. Specifically, we have discovered that: (1) The 3D points projected by training poses and corresponding depths to the world coordinate system, namely *projected points*, can effectively capture both the appearance and the geometry of the scene even without fine-tuning as shown in Fig. 4 (a); (2) The MLP networks achieve a faster convergence when queried with these *projected points* than *rasterization points*, which are obtained by vertex interpolation after mesh rasterization, as proved in Fig. 4 (b). Based on the above observations, we make full use of the geometric cues provided by the projected points to form two regularization loss terms, achieving state-of-the-art training speed. In the following, we will provide a detailed explanation of these two regularizations respectively: *position regularization* and *geometric feature regularization*.

Position Regularization. The vertex positions of the coarse mesh are optimized along with the network parameters to ensure accurate rasterization rendering results. The optimization of vertex positions instead of mesh division and decimation enables continuous gradient updates to a consistent set of mesh points. This continuous strategy is faster compared with discrete optimization which involves adding and removing vertices. In addition, the position regularizer further improves the



(a) (b)

Fig. 4. **Inspiration of depth-guided regularization.** (a) visualizes the geometry and appearance of *projected points* synthesized by a training pose and corresponding depths. (b) gives the comparison of the training loss during the first 100 iterations by inputting the *projected points* and the *rasterization points* to MLPs, respectively.

convergence speed by utilizing the projected points' positions as supervision to the vertex positions to help faster and more stable mesh refinement.

Formally, denoting the set of vertices in the initial mesh by $\mathcal{V}$, for each vertex $\mathbf{v}_i \in \mathcal{V}$, we add a trainable offset $\Delta\mathbf{v}_i$ to form $\hat{\mathcal{V}}$ for rasterization. For each training image, we rasterize the mesh under each sampling ray $\mathbf{r}_j$ ($j \leq N_r$) to generate the rasterization points and record their 3D positions $\mathbf{p}_j (j \leq N_r)$ in $\mathcal{P}$. N_r is the number of sampling rays. This rasterization step enables a more efficient sampling strategy, targeting points near the surfaces based on depth maps from S1. Thanks to nvdiffrast [56], the rasterization and interpolation operations are fully differentiable. During vertex optimization, we add a direct supervision on $\mathcal{P}$ provided by the projected points whose positions can be given by $\hat{\mathcal{P}} = \{\hat{\mathbf{p}}_j | \hat{\mathbf{p}}_j = \mathbf{o}_j + t_j \mathbf{d}_j\}$. t_j is the depth value of the j-th pixel in the depth image. Thus, the position regularization loss $\mathcal{L}_P$ can be formed as the weighted distance between the $\mathcal{P}$ and $\hat{\mathcal{P}}$:

$$\mathcal{L}_P = \sum_{j=1}^{N_r} w_j \|\mathbf{p}_j - \hat{\mathbf{p}}_j\|_1, \tag{15}$$

$$w_j = \psi(\|\mathbf{p}_j - \hat{\mathbf{p}}_j\|_2 - \delta), \tag{16}$$

where w_j is the adaptive weight that avoids depth outliers to dominate the regularization. $\psi(\cdot)$ is the exponential function. δ is set to 0.1 empirically. L1 loss is applied to be more resistant to outliers.

Geometric Feature Regularization. Compared with the view-independent diffuse reflection, the specular one is more difficult to learn. The geometric features $\mathbf{f}$ in Eq. 9 encode the view-independent geometry, helping to recover the specular color with different view directions. Similar to position regularization, we employ the geometric features of $\hat{\mathbf{p}}_j \in \hat{\mathcal{P}}$ as supervision for fast training. To further reduce the difficulty of convergence, we employ a residual network $\mathcal{R}_\theta$ to learn a residual feature [57], [58] $\Delta\mathbf{f}$ to help penalize the distance between the geometric embeddings of $\mathcal{P}$ and those of $\hat{\mathcal{P}}$. Mathematically,

$$\Delta\mathbf{f}_j^{S2} = \mathcal{R}_\theta(\mathbf{p}_j; \boldsymbol{\theta}_\mathcal{R}, \mathcal{V}), \tag{17}$$

$$\mathbf{f}_j^{S2} = \Delta\mathbf{f}_j + \mathcal{F}_\theta(\mathbf{p}_j; \boldsymbol{\theta}_\mathcal{F}, \mathcal{V}), \tag{18}$$

$$\hat{\mathbf{f}}_j^{S2} = \mathcal{F}_\theta(\hat{\mathbf{p}}_j; \boldsymbol{\theta}_\mathcal{F}, \mathcal{V}). \tag{19}$$

The final geometric feature loss can be formulated by:

$$\mathcal{L}_E = \sum_{j}^{N_r} w_j \|\mathbf{f}_j^{S2} - \hat{\mathbf{f}}_j^{S2}\|_2^2, \tag{20}$$

where w_j is given in Eq. 16.

Loss Function. Similar to S1, the rendering loss $\mathcal{L}_C^{S2}$ can be calculated by Eq. 8 where $\mathbf{C}^{S2}(\mathbf{r})$ can be obtained by substituting $\mathbf{p}$ with $\mathbf{p}_j$ in Eq. 9-11. The total loss function for S2 consists of the rendering loss, the position regularization loss and the feature regularization loss:

$$\mathcal{L}^{S2} = \mathcal{L}_C^{S2} + \beta_2(\mathcal{L}_P + \mathcal{L}_E), \tag{21}$$

where β_2 is set to 0.1.

After finishing this stage, the refined mesh with both diffuse textures and specular feature maps is generated for mobile rendering. Besides, the weights of the deferred shader $\mathcal{H}$ are also recorded in a JSON file for decoding view-dependent color with arbitrary input direction during inference. It is worth mentioning that the export of the final textured mesh can be highly paralleled on GPU without traversing all mesh fragments with certain z-buffering orders as in MobileNeRF [20].

V. Experiment

A. Experimental Setup

Implementation Details. All training and testing experiments in this paper were conducted on the same desktop computer equipped with a single GPU of NVIDIA GeForce RTX 3090 Ti except for the mobile rendering speed testing in Table IV. As for network architectures, following [49], we utilized the multi-resolution hash-grid encoder [7] and shallow MLPs to construct the density network $\mathcal{D}_\theta$ and the feature network $\mathcal{F}_\theta$. Specifically, $\mathcal{D}_\theta$ consists of a hash-grid encoder with 16 resolution levels and a 2-layer MLP with 32 hidden channels as the decoder. The density of an arbitrary 3D point in the scene can be output from $\mathcal{D}_\theta$ using its 3D position as the input. In a similar way, $\mathcal{F}_\theta$ is composed of a hash-grid encoder with 16 resolution levels, followed by an encoder with a 3-layer MLP with 64 hidden channels to achieve the 3-channel geometric features $\mathbf{f}$ and the 3-channel view-independent diffuse component $\mathbf{c}_d$. As for the neural deferred shader $\mathcal{H}_\theta$, it is formed by a 2-layer MLP with 32 hidden channels to convert the geometric features $\mathbf{f}$ to the view-dependent specular color $\mathbf{c}_s$. Additionally, the residual network $\mathcal{R}_\theta$ to assist the optimization of geometric features possesses the same MLP structure as $\mathcal{F}_\theta$.

Datasets. We evaluated our ATM-NeRF on three typical datasets: the synthetic 360° scenes from [1], the forward-facing scenes from [59], and unbounded 360° outdoor scenes from [27]. In detail, we conducted our experiments (training and testing) on the resolution of 800×800 for synthetic 360° and 1008×756 (4× down-scaled from the original resolution) for forward-facing, and 1237×825 for Mip-NeRF 360. These three datasets encompass a wide range of synthetic and real-world scenes, as well as bounded and unbounded environments, on which we all achieved excellent performance on training speed and rendering quality.

TABLE I
QUANTITATIVE COMPARISON ON TRAINING TIME ↓ (MINS) WITH RELATED METHODS ON TYPICAL DATASETS

Method	Synthetic 360° [1]				Forward-facing [59]				Mip-NeRF 360 [27]			
	S1	S2	total	speedup	S1	S2	total	speedup	S1	S2	total	speedup
LightSpeed [39]	-	-	240	5.16×	-	-	900	0.92×	-	-	-	-
3D-GS [40]	-	-	9	137.67×	-	-	35	24.49×	-	-	37	51.97×
SuGaR [45]	-	-	-	-	10	97	107	8.01×	10	171	181	10.62×
MobileNeRF [20]	326	913	1239	1×	327	530	857	1×	627	1296	1923	1×
NeRF2Mesh [49]	7	32	39	31.77×	10	49	59	14.53×	11	63	74	25.99×
ATM-NeRF	7	11	**18**	**68.83**×	10	19	**29**	**30.00**×	12	27	**39**	**49.31**×

B. Training Efficiency

To demonstrate our superior performance in training acceleration, we compared our ATM-NeRF with two representative NeRF baking schemes and two GS-based competitors. The former includes MobileNeRF [20] and NeRF2Mesh [49], which enable real-time rendering on nearly all mobile devices (especially smartphones). The latter consists of the original 3D-GS [40] and mesh-targeting SuGaR [45]. Also, the results of the SOTA NeLF-based method, LightSpeed [39], were also provided for comparison. We trained all the compared methods on the three datasets introduced in Sec. V-A and recorded the training time in Table I. In detail, we provided the training time of both training stages (if has) as well as the total time. For a fair comparison, we uniformly included the time for mesh exportation in S2. It can be seen from Table I that the S2 (NeRF baking) stage is the bottleneck of training speed acceleration, and our ATM-NeRF tackles it with our introduced depth-based regularization terms, achieving high training efficiency, accelerating MobileNeRF by about 30 $\sim$ 70 times. The reason accounting for our slightly slower training speed in S1 compared with NeRF2Mesh is the density regularization, which is of benefit in the convergence speed in S2 and the improvement of rendering quality.

Additionally, SuGaR forces to encourage the alignment between 3D Gaussians and the surface of the scene, which is time-consuming. On the contrary, our training time is comparable to the fast 3D-GS on the Mip-NeRF 360 dataset and even better than 3D-GS on the Forward-facing dataset. 3D-GS loses its superiority in training efficiency on the Forward-facing dataset as this dataset provides images from limited camera viewpoints other than 360° covered views, which is still an immature case in the area of 3D-GS.

For a more intuitive demonstration of our capability of rapid convergence, we offered two visualization results that showcase the updates in rendering quality during the training process. First, we plotted the curves of validation PSNR over training time for all evaluated methods on typical scenes in Fig. 5. From Fig. 5, it can be observed that our ATM-NeRF converges to excellent mobile rendering quality with the fastest training speed even for difficult real-world scenes. Second, to present our acceleration to the bottleneck NeRF baking stage (S2), we presented the rendered images of testing samples under rasterization from competitors after a certain training time ($1min$ and $5mins$) during S2 in Fig. 6. NeRF2Mesh [49] witnesses a decrease in rendering quality at 5 minutes, implying its instability of training, while MobileNeRF [20] suffers from an extremely slow convergence. Only our method achieves a satisfactory rendering quality within just 1 minute's baking training while maintaining a stable training process.

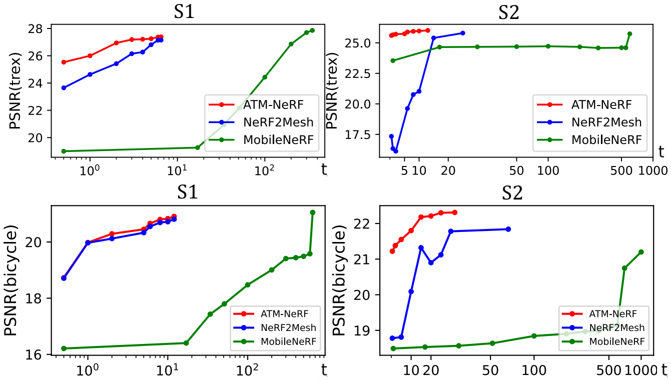


Fig. 5. **PSNR over training time (mins) of two stages.** The $trex, bicycle$ scenes are selected from the Forward-facing dataset and the Mip-NeRF 360 dataset, respectively.

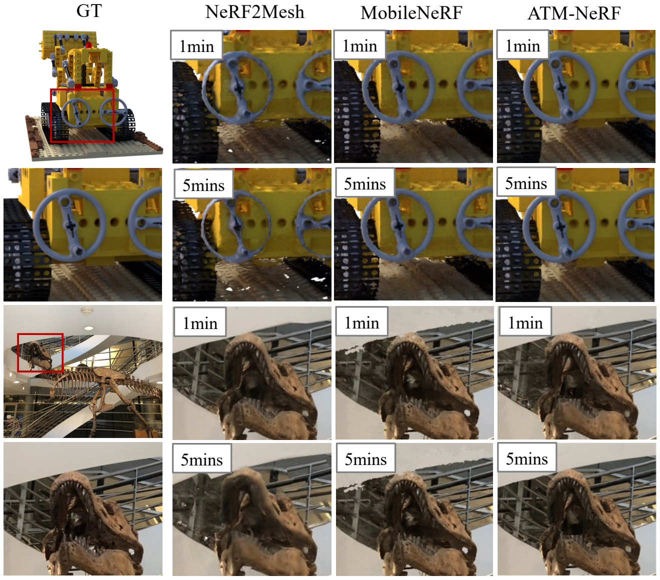


Fig. 6. **Rendered samples during baking training.** The rasterization rendering results on the *lego* scene (from Synthetic 360°) and the *trex* scene (from Forward-facing) rendered by NeRF2Mesh, MobileNeRF and our ATM-NeRF after training for 1 minute and 5 minutes in the S2 stage.

C. Rendering Performance

To evaluate the rendering ability and compatibility of our method, we tested the rendering quality and inference speeds on various devices as listed in Table II. The power is the GPU power for the NVIDIA card, and the combined max CPU and GPU power for integrated GPUs.

TABLE II
HARDWARE CONFIGURATIONS OF DEVICES USED IN OUR RENDERING EXPERIMENTS

Device	Type	OS	GPU	Power
iPhone XR	Phone	iOS 15.4	Integrated GPU	6W
iPad	Tablet	iOS 15.5	Integrated GPU	10W
Macbook Pro	Laptop	macOS Catalina 10.15.6	Integrated GPU	61W
Desktop	PC	Ubuntu 18.04	NVIDIA RTX 3090 Ti	350W

Rendering Quality. We reported the rendering quality on tested datasets quantitatively and qualitatively in Table III and Fig. 7 respectively. In the quantitative experiment, the common metrics PSNR, SSIM [60], and LPIPS [61] were utilized. “-” stands for not applicable to the corresponding dataset. The best and the second-best results are highlighted in bold black and blue, respectively. We also provided the results of the pioneering work in NeRF baking, namely SNeRG [50] for reference.

It can be seen from Table III that SNeRG [50] is unable to represent the complex unbounded 360° scenes. In terms of scene geometry, only our method and NeRF2Mesh [49] are capable of generating meshes with regular geometric surfaces, which can be applied to various downstream reconstruction tasks, and our ATM-NeRF outperforms NeRF2Mesh in rendering quality. In terms of the NeLF-based LightSpeed [39], it focuses on photorealistic rendering while its training speed still falls far behind methods based on 3D-GS or NeRF as proved in Tabel I. Also, it can only provide the light field representation that needs further interpretation for visualization rather than an intuitive mesh with geometric surfaces. Although 3D-GS [40] shows outstanding performance due to flexible 3D Gaussian representation, its Gaussians do not correspond well to the actual surface of the scene, making mesh exportation a difficult task. SuGaR [45] forces these Gaussians to align well with the surface of the scene for better geometry yet witnesses an obvious decrease in rendering quality.

On the other side, in the qualitative evaluation, we tested all compared methods directly on the HTML pages with a zoom-in operation to examine more rendering details. From the zoom-in comparison recorded in Fig. 7, it can be seen that ATM-NeRF exhibit comparable (even better in the *ficus* scene) rendering quality as MobileNeRF in the zoom-in view. NeRF2Mesh [49] exhibits white noises around the cables in the *room* scene and blur in the grass in the *stump* scene. In contrast, our method achieves the most photorealistic results even under magnification.

Rendering Speed. We revealed the rendering speed measured by Frame Per Second (FPS) on various devices in Table IV, and the results of competitors are obtained from their official interactive demos on their project pages. The notation $\frac{M}{N}$ indicates that M out of N testing scenes failed to run due to out-of-memory errors. The best and the second-best results are highlighted in pink and green, respectively. Table IV clearly indicates that ATM-NeRF evidently outperforms MobileNeRF [20] with non-surfaced mesh thanks to our geometry-regularized mesh. What's more, we can operate at interactive frame rates across all test scenarios on all test devices and achieve state-of-the-art rendering speed on Synthetic 360°. For example, MobileNeRF [20] fails to render the “$stump$” scene from Mip-NeRF 360 on iPhone XR while our ATM-NeRF can give satisfactory results as shown in Fig. 1 (c), proving our compatibility on mobile devices.

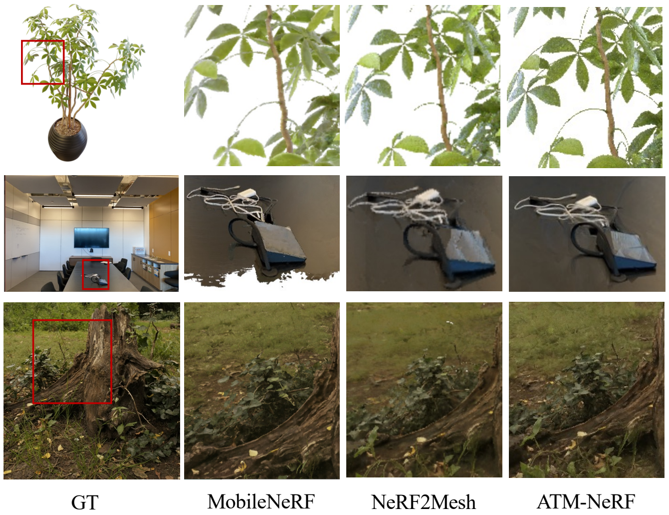


Fig. 7. **Zoom-in comparison.** The rendering results of other rivals were obtained from the demos released on their project pages. Our ATM-NeRF used the same web viewer provided by NeRF2Mesh for a fair comparison. From top to bottom are the *ficus* scene from Synthetic 360°, the *room* scene from Forward-facing dataset and the *stump* scene from Mip NeRF 360 dataset. Our approach renders high-quality images even for zoom-in views.

D. Mesh Quality

Mesh Geometry. Apart from the exceptional rendering performance, our ATM-NeRF is also capable of establishing meshes with fine geometry. We provided qualitative assessments of the extracted meshes produced by different methods, as shown in Fig. 8 (NeRF-based) and Fig. 9 (GS-based). It can be observed in Fig. 8 that MobileNeRF [20] performs poorly in scene geometry recovery and our ATM-NeRF reconstructed reasonable geometric structures of scenes with fine details, which is comparable to NeRF2Mesh [49] specially designed for mesh recovery. Actually, MobileNeRF achieves excellent rendering results via photometric overfitting to cause the shape-ambiguous problem explained in Sec. IV-B, leading to an incorrect geometry. On the contrary, thanks to our geometric regularization terms, we can produce a smooth mesh surface consistent with realistic scene geometry. Compared with the adaptive mesh refinement strategy in NeRF2Mesh [49] with face decimation and division, ATM-NeRF only modifies the vertex positions for training efficiency and does not significantly reduce mesh quality.

For GS-based approaches, it is challenging to extract mesh from the millions of tiny 3D Gaussians as these Gaussians tend to be unorganized after optimization. However, the geometric

TABLE III
RENDERING QUALITY COMPARISON ON THREE DATASETS COMPARED AGAINST METHODS WITH DIFFERENT BAKING REPRESENTATIONS

Method		Representation	Synthetic 360° [1]			Forward-facing [59]			Mip-NeRF 360 [27]		
			PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
GS Based	3D-GS [40]	Gaussians	33.20	0.969	0.033	27.58	0.886	0.104	26.40	0.805	0.173
	SuGaR [45]	Surface mesh	-	-	-	25.98	0.834	0.178	24.40	0.699	0.301
NeLF Based	LightSpeed [39]	NeLF	32.23	0.994	0.038	26.50	0.968	0.173	23.09	-	-
NeRF Based	SNeRG [50]	Volume	30.38	**0.950**	**0.050**	**25.63**	**0.818**	**0.183**	-	-	-
	MobileNeRF [20]	Non-surface mesh	**30.90**	**0.947**	**0.062**	**25.91**	**0.825**	**0.183**	**23.06**	**0.527**	**0.434**
	NeRF2Mesh [49]	Surface mesh	29.76	0.940	0.072	24.75	0.780	0.267	22.36	0.493	0.478
	ATM-NeRF	Surface mesh	**30.89**	0.946	0.069	24.97	0.799	**0.204**	**22.94**	**0.543**	**0.439**

TABLE IV
RENDERING SPEED (FPS) ON VARIOUS DEVICES ON THE THREE TESTING DATASETS

Dataset	Method	iPhone XR	iPad	Macbook Pro	Desktop
Synthetic 360° [1]	SNeRG [50]	0.0 $\frac{8}{8}$	0.0 $\frac{8}{8}$	16.85 $\frac{1}{8}$	51.57 $\frac{2}{8}$
	MobileNeRF [20]	54.16	32.81	57.67	60.05
	NeRF2Mesh [49]	**58.15**	**35.70**	**58.36**	**60.19**
	ATM-NeRF	**58.75**	**36.02**	**59.41**	**60.25**
Forward-facing [59]	SNeRG [50]	0.0 $\frac{8}{8}$	0.0 $\frac{8}{8}$	0.0 $\frac{8}{8}$	0.0 $\frac{8}{8}$
	MobileNeRF [20]	36.55 $\frac{2}{8}$	28.55 $\frac{2}{8}$	27.08	**60.1**
	NeRF2Mesh [49]	**57.25**	**35.60**	**50.87**	**60.3**
	ATM-NeRF	**49.43**	**31.23**	**40.03**	60.09
Mip-NeRF 360 [27]	SNeRG [50]	-	-	-	-
	MobileNeRF [20]	40.9 $\frac{2}{3}$	28.92 $\frac{2}{3}$	27.27	58.67
	NeRF2Mesh [49]	**58.33**	**37.79**	**51.67**	**60.32**
	ATM-NeRF	**54.33**	**36.77**	**44.33**	**59.88**

mesh is an essential representation for editing, sculpting, animating, and relighting the scene to support various visual applications. Few solutions have been proposed so far and SuGaR [45] is the SOTA one. Fig. 9 shows the qualitative comparison results of 3D-GS [40], SuGaR [45] and ATM-NeRF in mesh reconstruction. For comparison, the meshes of 3D-GS were exported based on its output point clouds by Poisson reconstruction. It can be seen from Fig. 9 that 3D-GS produces sparse and fragmented mesh faces and comparably SuGaR can recover denser mesh surfaces. However, SuGaR's meshes still suffer from floaters (*lego*), rough edges (*ship* and *bicycle*) or lost details (*garden*). Also, it can be observed from the zoom-in blocks in the last two rows that SuGaR extracts more delicate mesh faces, explained for its large memory storage in Table V.

Mesh Storage. Considering the resource limitation of mobile devices, we also evaluated the practical applicability by comparing the disk storage requirement and the number of vertices and faces in the exported meshes, as shown in Table V. "#V" denotes the number of vertices of the exported mesh. The disk storage contains all products of S2, including the exported meshes, their texture maps and the JSON file recording the weights of neural shader $\mathcal{H}_\theta$. Instead of the subdivision and decimation algorithm in the mesh refinement of NeRF2Mesh [49], our refinement only focuses on vertex positions under geometric regularizations, leading to the least vertices and disk storage requirements. As for the Gaussian methods, they require a large number of 3D Gaussian functions to represent scenes with complex details, which inevitably take up a lot of memory storage, and current solutions still focus on memory saving for Gaussians rather than exported mesh.

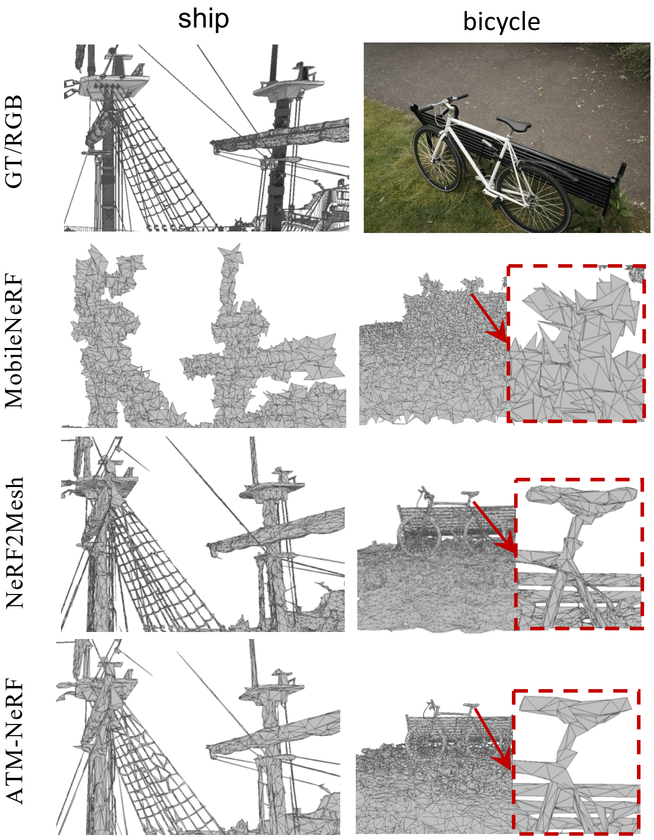


Fig. 8. **Surface reconstruction quality compared with NeRF-based methods** on Synthetic 360° dataset and Mip-NeRF 360 dataset. Our method achieves comparable mesh reconstruction quality compared to NeRF2Mesh by only refining the vertex positions.

We also provided our memory usage for training and volume rendering during S1 in Table V. From this table, we can observe that to train the full ATM-NeRF, a GPU with at least 8G memory is needed for training and volume rendering before exporting meshes. Taking both memory and processing capability into consideration, we recommend using GPU above NVIDIA GeForce RTX 3060, otherwise the training may fail or take longer time than we reported.

E. Ablation Study

We demonstrated how the introduced geometric regularization terms in our framework affect the results by comparing ATM-NeRF against three variants that exclude either the

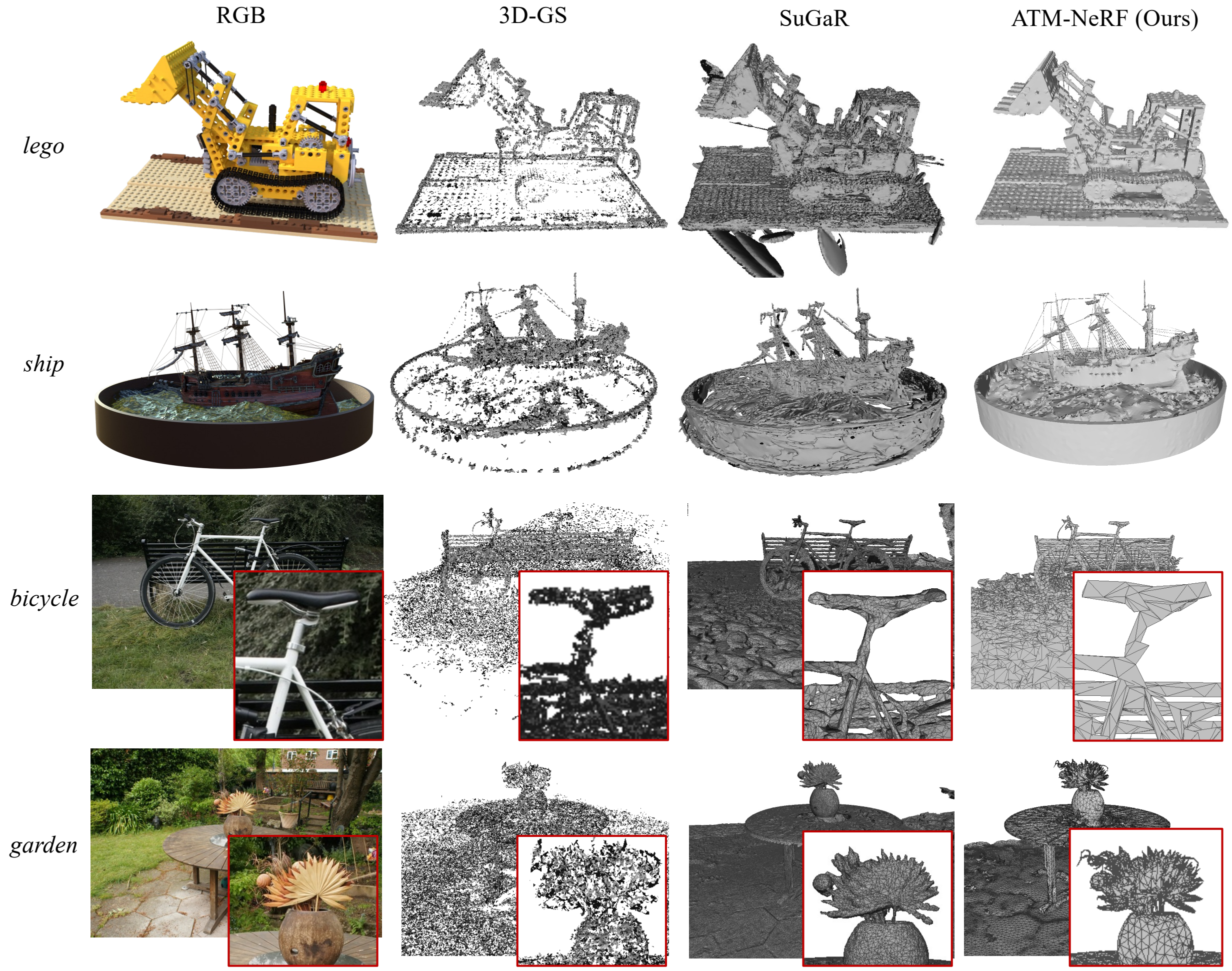


Fig. 9. **Surface reconstruction quality compared with GS-based methods** on Synthetic 360° dataset and Mip-NeRF 360 dataset. The mesh reconstruction results of 3D-GS were generated from Poisson reconstruction based on its output point clouds. SuGaR focuses more on real-world scenes and cannot effectively deal with scenes without background.

TABLE V
AVERAGE NUMBER OF VERTICES ↓ (10^3) AND MEMORY USAGE ↓ (MB) OF EXPORTED MESHES

	Synthetic 360° [1]		Forward-facing [59]		Mip-NeRF 360 [27]	
	#V	Mem.	#V	Mem.	#V	Mem.
GT	614	30.49	-	-	-	-
MobileNeRF [20]	494	126	830	202	1436	345
NeRF2Mesh [49]	200	74	397	125	718	187
3D-GS [40]	289	**53**	1017	238	4808	1198
SuGaR [45]	-	-	489	173	1045	384
ATM-NeRF-S1 (volume)	-	4185	-	3175	-	3959
ATM-NeRF-S2 (mesh)	**131**	54	**209**	**120**	**439**	**158**

[*] ATM-NeRF-S1 is for presenting the memory usage of training and volume rendering during S1.

density regularization, the position regularization or the feature regularization, denoted by "w/o $\mathcal{L}_D/\mathcal{L}_P/\mathcal{L}_E$", respectively. Table VI, Table VII and Table VIII provide the quantitative results of these evaluated variants in terms of the rendering quality and the training speed on the Synthetic 360° dataset [1], the Forward-facing dataset [59] and the Mip-NeRF 360 dataset [27], respectively. For more intuitive observation, we offer the validation PSNR during training on typical scenes from three tested datasets in Fig. 10. It can be observed from the above quantitative results that $\mathcal{L}_D, \mathcal{L}_P, \mathcal{L}_E$ all play an important role in improving rendering performance in their stage, especially for S2. The underlying reason is that such geometric constraints are of great benefit in recovering reasonable scene geometry, based on which the rasterization points for rendering can be more accurate and consistent with perception. What's more, $\mathcal{L}_P$ and $\mathcal{L}_E$ encourage fast convergence of the training model in the bottleneck stage S2 via their direct point-to-point supervision on the vertex position and the geometric feature respectively. Besides, Fig. 11 gives the rendering results of variants "w/o $\mathcal{L}_P$", "w/o $\mathcal{L}_E$" and our ATM-NeRF in S2. By observing Fig. 11, we can conclude that both loss terms are of great assistance in synthesizing a remarkable rendering image with photorealistic details. Furthermore, $\mathcal{L}_P$ brings more improvement than $\mathcal{L}_E$ since it directly guides the vertex position optimization to determine the mesh geometry.

All the above quantitative results corroborate our claim that our full ATM-NeRF achieves the best rendering quality while using the least training iterations, demonstrating the effectiveness of our designed regularization constraints $\mathcal{L}_D, \mathcal{L}_P, \mathcal{L}_E$ in improving both rendering quality and training efficiency.

TABLE VI
ABLATION STUDY– RENDERING QUALITY AND THE NUMBER OF TRAINING EPOCHS FOR DIFFERENT VARIANTS TO CONVERGE ON THE SYNTHETIC 360° DATASET [1]

Stage	Variants	PSNR	SSIM	LPIPS	epoch
Pre-training	w/o $\mathcal{L}_D$	30.88	0.951	0.079	300
	ATM-NeRF (S1)	**31.87**	**0.956**	**0.061**	300
Baking	w/o $\mathcal{L}_P$	30.65	0.931	0.070	300
	w/o $\mathcal{L}_E$	30.76	0.938	0.079	240
	ATM-NeRF (S2)	**30.89**	**0.946**	**0.069**	**180**

TABLE VII
ABLATION STUDY– RENDERING QUALITY AND THE NUMBER OF TRAINING EPOCHS FOR DIFFERENT VARIANTS TO CONVERGE ON THE FORWARD-FACING DATASET [59]

Stage	Variants	PSNR	SSIM	LPIPS	epoch
Pre-training	w/o $\mathcal{L}_D$	26.42	0.824	0.218	1200
	ATM-NeRF (S1)	**26.97**	**0.839**	**0.204**	1200
Baking	w/o $\mathcal{L}_P$	22.91	0.680	0.201	500
	w/o $\mathcal{L}_E$	23.56	0.681	0.289	500
	ATM-NeRF (S2)	**24.97**	**0.799**	**0.204**	**350**

F. Discussion

Rendering Quality on the Foward-facing Dataset. It can be observed from Table III that our ATM-NeRF underperforms MobileNeRF especially on the Forward-facing dataset. We found that the main reason lies in the grid resolution used in S1 to generate the rasterized mesh, which will be further refined in S2. To demonstrate this, we provided experimental results configured with different resolution values on this dataset in Table IX. “-r” denotes different resolution values. “†” denotes the configuration we used in this paper. As mentioned in Sec. IV-B, we empirically used 128 ($\mathbf{G}_{s_1}$) as the mesh resolution for the sake of training efficiency.

As demonstrated in Table IX, the resolution we chose is not delicate enough to represent the challenging Forward-facing dataset without 360° view coverage, leading to a decrease in the rendering quality. However, a finer resolution can bring an increase to rendering quality but also to the disk storage requirement and training time. Considering that our goal is to accelerate training, we expected the exported mesh to be as lightweight as possible to reduce the optimization time of mesh vertices. Also, a low-resolution mesh takes up less disk storage, which is more friendly to resource-constrained mobile devices. From the perspective of mobile rendering, we prefer 128 as the resolution to lay more emphasis on time efficiency and memory efficiency, and we believe the loss in rendering quality is acceptable. In practice, users can modify the resolution based on their needs.

Summary. On the one hand, although MobileNeRF [20] and GS-based schemes [40], [45] demonstrate excellent performance in rendering quality, they take mesh geometry as a sacrifice, which not only hinders their support for various downstream tasks but also contributes to larger disk storage requirements. As a result, MobileNeRF owns the lowest rendering frame rate and even fails to render some large-scale outdoor scenes from Mip-NeRF 360 [27] on our testing iPhone XR due to the out-of-memory errors. Also, MobileNeRF and SuGaR suffer from lengthy training time for mesh exportation.

TABLE VIII
ABLATION STUDY– RENDERING QUALITY AND THE NUMBER OF TRAINING EPOCHS FOR DIFFERENT VARIANTS TO CONVERGE ON THE MIP-NERF 360 DATASET [27]

Stage	Variants	PSNR	SSIM	LPIPS	epoch
Pre-training	w/o $\mathcal{L}_D$	22.33	0.538	**0.481**	180
	ATM-NeRF (S1)	**22.54**	**0.545**	0.497	180
Baking	w/o $\mathcal{L}_P$	22.03	0.526	0.468	120
	w/o $\mathcal{L}_E$	21.54	0.512	0.474	100
	ATM-NeRF (S2)	**22.94**	**0.543**	**0.439**	**70**

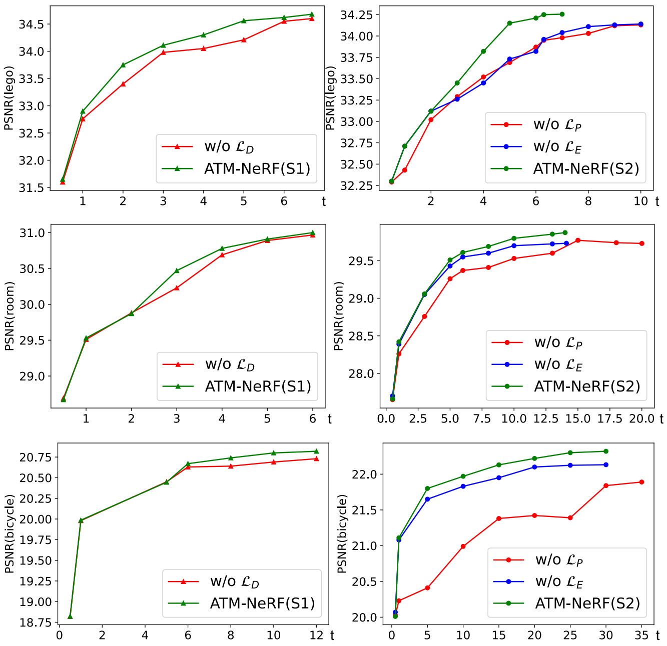


Fig. 10. **Ablation study– validation PSNR during two training stages** compared with all variants on typical scenes from the Synthetic 360° dataset, the Forward-facing dataset and the Mip-NeRF 360 dataset, respectively.

On the other hand, NeRF2Mesh [49] performs well in restoring geometric details of the scene and also achieves brilliant rendering speed thanks to its adaptive mesh refinement algorithm. However, this heuristic refinement algorithm brings instability to training, leaving room for improvement in training efficiency. In addition, it sacrifices rendering quality for the sake of geometric structure, resulting in noticeable white noise and blurring in the rendering results.

Compared with the above methods, ATM-NeRF is comparable to the SOTA NeRF methods in terms of the rendering quality and the mesh quality. Besides, ATM-NeRF costs the least disk storage due to our effective mesh refinement strategy on vertices, reducing the computational power requirement for the rendering hardware and ensuring a rendering speed at about 30 $\sim$ 60 FPS on iPhone XR. The most important thing to emphasize is that owing to our regularization constraints to guide the continuous gradient updates for mesh refinement, ATM-NeRF hold an overwhelming advantage in training speed which is 30 $\sim$ 70× faster than MobileNeRF [20], about 2× faster than NeRF2Mesh [49] and 4 $\sim$ 5× faster than SuGaR [45]. *Taking all factors into account, our ATM-NeRF is the best candidate for mobile NeRF rendering.*

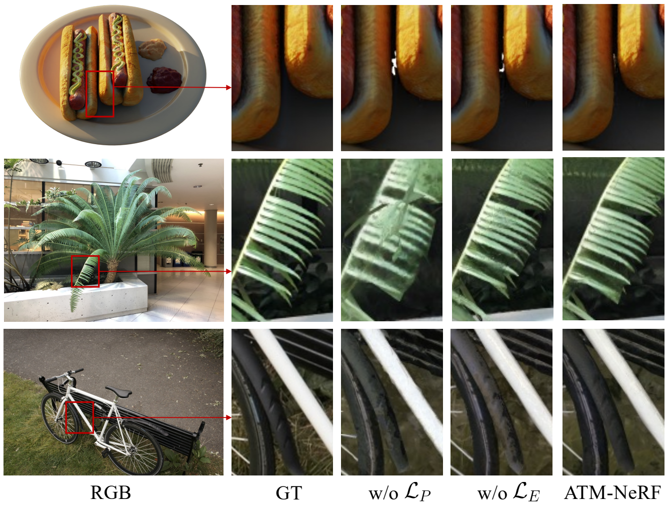


Fig. 11. **Ablation – rendering performance in S2** of different variants on the typical scenes from the Synthetic 360° dataset, the Forward-facing dataset and the Mip-NeRF 360 dataset, respectively.

TABLE IX
COMPARISON OF ATM-NERF WITH DIFFERENT RESOLUTION CONFIGURATIONS ON PSNR↑, DISK STORAGE REQUIREMENT↓ (MB) AND TRAINING TIME↓ (MINS) ON THE FORWARD-FACING DATASET

Methods	PSNR	Disk	Training Time
MobileNeRF [20]	25.91	201.50	857.5
ATM-NeRF-r128†	24.97	**120.30**	**29.60**
ATM-NeRF-r192	25.38	136.34	58.54
ATM-NeRF-r256	**25.77**	158.91	91.37

VI. CONCLUSION

In summary, we presented ATM-NeRF, the first work to employ geometric constraints as regularizations to accelerate training for NeRF rendering on mobile devices. In the first pre-training stage, we enhance the rendering quality and achieve smoother meshes by enforcing stable density grids with multiple resolutions. In the second stage, the projected points synthesized from posed depths generated from pre-training are fully utilized, whose positions and geometric features are of great use in training convergence. As a result, our work achieves comparable rendering quality with MobileNeRF [20], approximately 70× faster on the Synthetic 360° dataset, and around 50× faster on real-world Mip-NeRF 360 scenes. In addition, we also maintain a satisfactory mesh with a delicate geometric structure to support zoom-in/out operation for mobile applications. What's more, we keep a faster rendering speed than MobileNeRF [20] due to surfaced meshes and low disk storage. Although ATM-NeRF has demonstrated excellent performance in training speed and rendering quality, there are still two limitations of the current work that can be improved. First, we may have a problem with mesh relighting since our specular model records the illumination of the training set and fails to generalize to unknown illumination environments without re-training. Second, the training speed and rendering quality of ATM-NeRF are still expected to be improved when dealing with the input from limited views such as the Forward-facing dataset. In our future work, we will continue to devote our efforts to improving the generation ability of ATM-NeRF and to mobile rendering with sparse input even from a single view.

REFERENCES

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 405-421.

[2] P. Ndjiki-Nya, M. Koppel, D. Doshkov, H. Lakshman, P. Merkle, K. Muller, and T. Wiegand, "Depth image-based rendering with advanced texture synthesis for 3-D video," *IEEE Trans. Multimedia*, vol. 13, no. 3, pp. 453-465, 2011.

[3] B. Cao, H. Cao, J. Liu, P. Zhu, C. Zhang, and Q. Hu, "Autoencoder-based collaborative attention GAN for multi-modal image synthesis," *IEEE Trans. Multimedia*, vol. 26, pp. 995-1010, 2024.

[4] Z. Liu, W. Jia, M. Yang, P. Luo, Y. Guo, and M. Tan, "Deep view synthesis via self-consistent generative network," *IEEE Trans. Multimedia*, vol. 24, pp. 451-465, 2022.

[5] K. Deng, A. Liu, J. Y. Zhu, and D. Ramanan, "Depth-supervised NeRF: Fewer views and faster training for free," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 12872-12881.

[6] C. Sun, M. Sun, and H. T. Chen, "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 5449-5459.

[7] T. Müller, A. Evans, and C. S. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp.1-15, 2022.

[8] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 5491-5500.

[9] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "TensoRF: Tensorial radiance fields," in *Proc. Eur. Conf. Comput. Vis.*, 2022, pp. 333-350.

[10] L. Yariv, P. Hedman, C. Reiser, D. Verbin, P. P. Srinivasan, R. Szeliski, J. T. Barron, and B. Mildenhall, "BakedSDF: Meshing neural SDFs for real-time view synthesis," *ACM Trans. Graph.*, vol. 46, pp. 1-9, 2023.

[11] L. Liu, J. Gu, K. Z. Lin, T. Chua, and C. Theobalt, "Neural sparse voxel fields," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 15651-15663, 2020.

[12] D. B. Lindell, J. N. P. Martel, and G. Wetzstein, "AutoInt: Automatic integration for fast neural volume rendering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 14551-14560.

[13] D. Rebain, W. Jiang, S. Yazdani, K. Li, K. M. Yi, and A. Tagliasacchi, "DeRF: Decomposed radiance fields," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 14148-14156.

[14] T. Neff, P. Stadlbauer, M. Parger, A. Kurz, J. H. Mueller, C. R. A. Chaitanya, A. Kaplanyan, and M. Steinberger, "DoNeRF: Towards real-time rendering of compact neural radiance fields using depth oracle networks," *Comput. Graph. Forum*, vol. 40, no. 4, pp. 45-59, 2021.

[15] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "FastNeRF: High-fidelity neural rendering at 200FPS," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 14326-14335.

[16] C. Reiser, S. Peng, Y. Liao, and A. Geiger, "KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs, " in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 14315–14325.

[17] V. Sitzmann, S. Rezchikov, B. Freeman, J. Tenenbaum, and F. Durand, "Light field networks: Neural scene representations with single-evaluation rendering, " *Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 19313-19325, 2021.

[18] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "PlenOctrees for real-time rendering of neural radiance fields," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 5732-5741.

[19] H. Wang, J. Ren, Z. Huang, K. Olszewski, M. Chai, Y. Fu, and S. Tulyakov, "R2L: Distilling neural radiance field to neural light field for efficient novel view synthesis, " in *Proc. Eur. Conf. Comput. Vis.*, 2022, pp. 612-629.

[20] Z. Chen, T. Funkhouser, P. Hedman, and A. Tagliasacchi, "MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 16569-16578.

[21] J. Cao, H. Wang, P. Chemerysa, V. Shakhrai, J. Hu, Y. Fu, D. Makoviichuk, S. Tulyakov, and J. Ren, "Real-time neural light field on mobile devices, " in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 8328-8337.

[22] H. Yu, J. Julin, Z. Á. Milacski, K. Niinuma, and L. A. Jeni, "DyLiN: Making light field networks dynamic," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 12397-12406.

[23] R. Martin-Brualla, N. Radwan, M. S. M. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, "NeRF in the wild: Neural radiance fields for unconstrained photo collections," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 7206-7215.

[24] X. Chen, Q. Zhang, X. Li, Y. Chen, Y. Feng, X. Wang, and J. Wang, "Ha-NeRF: Hallucinated neural radiance fields in the wild," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 12943-12952.

[25] K. Jun-Seong, K. Yu-Ji, M. Ye-Bin, and T. Oh, "HDR-Plenoxels: Self-calibrating high dynamic range radiance fields," in *Proc. Eur. Conf. Comput. Vis.*, 2022, pp. 384-401.

[26] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 5835-5844.

[27] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-NeRF 360: Unbounded anti-aliased neural radiance fields, " in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 5460-5469.

[28] A. Yu, V. Ye, M. Tancik, and A. Kanazawa, "PixelNeRF: Neural radiance fields from one or few images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 4576-4585.

[29] T. DeVries, M. A. Bautista, N. Srivastava, G. W. Taylor, and J. M. Susskind, "Unconstrained scene generation with locally conditioned radiance fields," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 14284-14293.

[30] Y. Liu, S. Peng, L. Liu, Q. Wang, P. Wang, C. Theobalt, X. Zhou, and W. Wang, "Neural rays for occlusion-aware image-based rendering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 7814-7823.

[31] B. Attal, E. Laidlaw, A. Gokaslan, C. Kim, C. Richardt, J. Tompkin, and M. O'Toole, "TöRF: Time-of-flight radiance fields for dynamic scene view synthesis," *Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 26289-26301, 2021.

[32] B. Mildenhall, P. Hedman, R. Martin-Brualla, P. P. Srinivasan, and J. T. Barron, "NeRF in the dark: High dynamic range view synthesis from noisy raw images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 16169-16178.

[33] K. Zhang, G. Riegler, N. Snavely, and V. Koltun, "NeRF++: Analyzing and improving neural radiance fields," 2010, *arXiv:2010.07492*. [Online]. Available: http://arxiv.org/abs/2010.07492.

[34] K. Guo, T. Chen, S. Ren, B. Hu, Z. Wu, S. Guo, and H. Fang, "ReE3D: Boosting novel view synthesis for monocular images using residual encoders," *IEEE Trans. Multimedia*, early access, 2023.

[35] S. Shen, W. Li, X. Huang, Z. Zhu, J. Zhou, and J. Lu, "SD-NeRF: Towards lifelike talking head animation via spatially-adaptive dual-driven NeRFs," *IEEE Trans. Multimedia*, early access, 2023.

[36] P. Bao and D. Gourlay, "A framework for remote rendering of 3-D scenes on limited mobile devices," *IEEE Trans. Multimedia*, vol. 8, no. 2, pp. 382-389, 2006.

[37] M. Bemana, K. Myszkowski, H. Seidel, and T. Ritschel, "X-fields: Implicit neural view-, light-and time-image interpolation," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1-15, 2020.

[38] N. K. Kalantari, T. Wang, and R. Ramamoorthi, "Learning-based view synthesis for light field cameras," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 1-10, 2016.

[39] A. Gupta, J. Cao, C. Wang, J. Hu, S. Tulyakov, J. Ren, and L. Jeni, "LightSpeed: Light and fast neural light fields on mobile devices," *Adv. Neural Inf. Process. Syst.*, vol. 36, pp. 31021-31037, 2023.

[40] B. Kerbl, G. Kopanas, T. Leimkuhler, and G. Drettakis, "3D Gaussian Splatting for real-time radiance field rendering," *ACM Trans. Graph.*, vol. 42, no. 4, pp. 139:1-14, 2023.

[41] T. Lu, M. Yu, L. Xu, Y. Xiangli, L. Wang, D. Lin, and B. Dai, "Scaffold-GS: Structured 3D Gaussians for view-adaptive rendering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2024, pp. 20654-20664.

[42] K. Navaneet, K. P. Meibodi, S. A. Koohpayegani, and H. Pirsiavash, "Compact3D: Smaller and faster Gaussian Splatting with vector quantization," 2023, *arXiv:2311.18159*. [Online]. Available: https://arxiv.org/abs/2311.18159.

[43] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "Light-Gaussian: Unbounded 3D Gaussian compression with 15x reduction and 200+ fps," 2023, *arXiv:2311.17245*. [Online]. Available: https://arxiv.org/abs/2311.17245.

[44] K. Ren, L. Jiang, T. Lu, M. Yu, L. Xu, Z. Ni, and B. Dai, "Octree-GS: Towards consistent real-time rendering with LOD-structured 3D Gaussians," 2024, *arXiv:2403.17898*. [Online]. Available: https://arxiv.org/abs/2403.17898.

[45] A. Guédon, V. Lepetit, "SuGaR: Surface-aligned Gaussian Splatting for efficient 3D mesh reconstruction and high-quality mesh rendering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2024, pp. 5354-5363.

[46] H. Chen, C. Li, and G. H. Lee, "NeuSG: Neural implicit surface reconstruction with 3D Gaussian Splatting guidance," 2023, *arXiv:2312.00846*. [Online]. Available: https://arxiv.org/abs/2312.00846.

[47] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," *Adv. Neural Inf. Process. Syst.*, pp. 171-183, 2021.

[48] X. Lyu, Y. Sun, Y. Huang, X. Wu, Z. Yang, Y. Chen, J. Pang, and X. Qi, "3DGSR: Implicit surface reconstruction with 3D Gaussian Splatting," 2024, *arXiv:2404.00409*. [Online]. Available: https://arxiv.org/abs/2404.00409.

[49] J. Tang, H. Zhou, X. Chen, T. Hu, E. Ding, J. Wang, and G. Zeng, "Delicate textured mesh recovery from NeRF via adaptive surface refinement," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2023, pp. 9187-9198.

[50] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, "Baking neural radiance fields for real-time view synthesis," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 5855-5864.

[51] C. Reiser, R. Szeliski, D. Verbin, P. P. Srinivasan, B. Mildenhall, A. Geiger, J. T. Barron, and P. Hedman, "Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes," *ACM Trans. Graph.*, vol. 42, no. 4, pp. 1-12, 2023.

[52] L. Yariv, J. Gu, Y. Kasten, and Y. Lipman, "Volume rendering of neural implicit surfaces," *Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 4805-4815, 2021.

[53] W. E. Lorensen, and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Special Interest Group for Comput. Graph.*, 1987, pp. 347-353.

[54] A. Gotmare, N. S. Keskar, C. Xiong, and R. Socher, "A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation," 2018, *arXiv:1810.13243*. [Online]. Available: http://arxiv.org/abs/1810.13243.

[55] J. Ma and D. Yarats, "On the adequacy of untuned warmup for adaptive optimization," in *AAAI Conf. Artif. Intell.*, 2021, pp. 8828-8836.

[56] J. Munkberg, J. Hasselgren, T. Shen, J. Gao, W. Chen, A. Evans, T. Muller, and S. Fidler, "Extracting triangular 3D models, materials, and lighting from images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 8280-8290.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770-778.

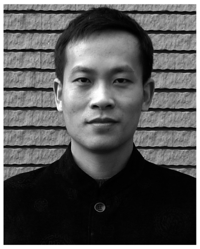
[58] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016, *arXiv:1605.07146*. [Online]. Available: http://arxiv.org/abs/1605.07146.

[59] B. Mildenhall, P. P. Srinivasan, R. Ortiz-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. 1-14, 2019.

[60] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600-612, 2004.

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

Yang Chen received her B.S. degree from the School of Software Engineering, Tongji University, Shanghai, China, in 2020. She is now pursuing her Ph.D. degree at the same department of Tongji University. Her research interests include SLAM systems, computer vision, and machine learning.



Lin Zhang (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2003 and 2006, respectively. He received the Ph.D. degree from the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, in 2011. From March 2011 to August 2011, he was a Research Associate with the Department of Computing, The Hong Kong Polytechnic University. In Aug. 2011, he joined the School of Software Engineering, Tongji University, Shanghai, China, where he is currently a Full Professor. His current research interests include environment perception of intelligent vehicle, pattern recognition, computer vision, and perceptual image/video quality assessment. He serves as an Associate Editor for IEEE Robotics and Automation Letters, and Journal of Visual Communication and Image Representation. He was awarded as a Young Scholar of Changjiang Scholars Program, Ministry of Education, China.

Shengjie Zhao (Senior Member, IEEE) received the B.S. degree in electrical engineering from the University of Science and Technology of China, Hefei, China, in 1988, the M.S. degree in electrical and computer engineering from the China Aerospace Institute, Beijing, China, in 1991, and the Ph.D. degree in electrical and computer engineering from Texas AM University, College Station, TX, USA, in 2004. He is currently the Dean of the College of Software Engineering and a Professor with the College of Software Engineering and the College of Electronics and Information Engineering, Tongji University, Shanghai, China. In previous postings, he conducted research at Lucent Technologies, Whippany, NJ, USA, and the China Aerospace Science and Industry Corporation, Beijing. He is a fellow of the Thousand Talents Program of China and an Academician of the International Eurasian Academy of Sciences. His research interests include artificial intelligence, big data, wireless communications, image processing, and signal processing.

Yicong Zhou (Senior Member, IEEE) received the B.S. degree in electrical engineering from Hunan University, Changsha, China, and the M.S. and Ph.D. degrees in electrical engineering from Tufts University, Medford, MA, USA. He is currently a Full Professor and the Director of the Vision and Image Processing Laboratory, Department of Computer and Information Science, University of Macau, Macau, China. His research interests include chaotic systems, multimedia security, computer vision, and machine learning. Dr. Zhou was a recipient of the Third Price of Macau Natural Science Award in 2014. He serves as an Associate Editor for Neurocomputing, Journal of Visual Communication and Image Representation, and Signal Processing: Image Communication. He is a Co-Chair of the Technical Committee on Cognitive Computing in the IEEE Systems, Man, and Cybernetics Society. He is a Senior Member of the International Society for Optical Engineering (SPIE).