EAGLES: Efficient Accelerated 3D Gaussians with Lightweight EncodingS

Sharath Girish, Kamal Gupta, and Abhinav Shrivastava

University of Maryland, College Park
sgirish@umd.edu, kamalgupta308@gmail.com, abhinav@cs.umd.edu

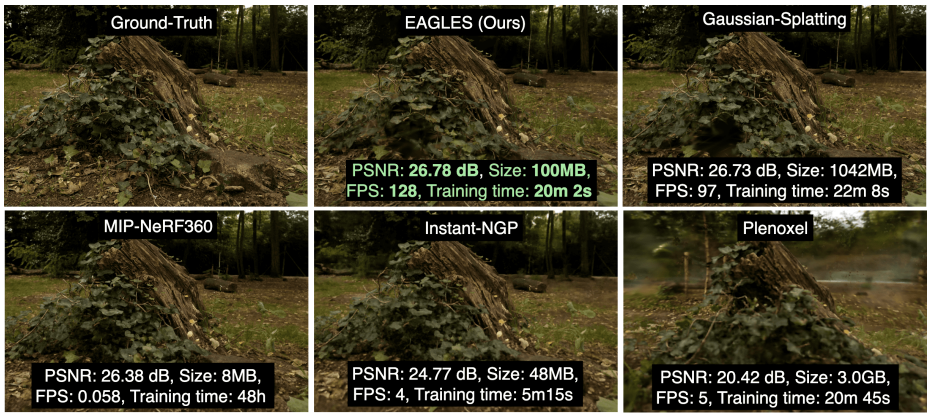

Fig. 1: We propose EAGLES, a technique to significantly reduce the memory footprint of 3D-Gaussian splatting by an order of magnitude while offering further speed-ups in training time and rendering FPS all while maintaining the reconstruction quality.

Abstract. Recently, 3D Gaussian splatting (3D-GS) has gained popularity in novel-view scene synthesis. It addresses the challenges of lengthy training times and slow rendering speeds associated with Neural Radiance Fields (NeRFs). Through rapid, differentiable rasterization of 3D Gaussians, 3D-GS achieves real-time rendering and accelerated training. They, however, demand substantial memory resources for both training and storage, as they require millions of Gaussians in their point cloud representation for each scene. We present a technique utilizing quantized embeddings to significantly reduce per-point memory storage requirements and a coarse-to-fine training strategy for a faster and more stable optimization of the Gaussian point clouds. Our approach develops a pruning stage which results in scene representations with fewer Gaussians, leading to faster training times and rendering speeds for real-time rendering of high resolution scenes. We reduce storage memory by more than an order of magnitude all while preserving the reconstruction quality. We validate the effectiveness of our approach on a variety of datasets and scenes preserving the visual quality while consuming 10-20$\times$ less memory and faster training/inference speed. Code is available here.

1 Introduction

Neural Radiance Fields [29] (NeRF) have become widespread in their use as 3D scene representations achieving high visual quality by training implicit neural networks via differentiable volume rendering. They however come at the cost of high training and rendering costs. While more recent works such as Plenoxels [14] or Multiresolution Hashgrids [30] have significantly reduced the training times, they are still slow to render for high resolution scenes and do not reach the same visual quality as NeRF methods such as [3, 4]. To overcome these issues, 3D Gaussian splatting [22] (3D-GS) proposed an approach to learn 3D gaussian point clouds as scene representations. Unlike the slow volume rendering of NeRFs, they utilize a fast differentiable rasterizer, to project the points on the 2D plane for rendering views. They achieve state-of-the-art (SOTA) reconstruction quality while still obtaining similar training times as the efficient NeRF variants. Through their fast tile-based rasterizer, they also achieve real-time rendering speeds at 1080p scene resolutions, significantly faster than NeRF approaches.

While 3D-GS has several advantages over NeRFs for novel view synthesis, they come at the cost of high memory usage. A high resolution scene requires several millions of Gaussians to achieve high quality view reconstructions. Each point consists of several attributes such as position, color, rotation, opacity and scaling. This leads to scene representations requiring high amounts of memory for storage ($>$1GB). The GPU runtime memory requirements during training and rendering is also much higher compared to standard NeRF methods, requiring almost 20GB of GPU RAM for several high-resolution scenes. They are thus not very practical for graphic systems with strong memory-constraints of storage or runtime memory or in low-bandwidth applications.

Our approach aims to decrease both storage and runtime memory costs while enhancing both training and rendering speeds, and maintaining view synthesis quality on par with the SOTA, 3D-GS. The color attribute, represented by spherical harmonic (SH) coefficients, and the rotation attribute, represented by covariance matrices, utilize more than 80% of the memory cost of all attributes. Our approach significantly reduces the memory usage of each Gaussian by compressing the color and rotation attributes via a latent quantization framework. We also quantize the opacity coefficients of the Gaussians improving the optimization and leading to fewer floaters or visual artifacts in novel view reconstructions. Additionally, we propose a coarse-to-fine training strategy which improves the training stability and convergence speed while also obtaining better reconstructions. Finally, to reduce the number of redundant Gaussians resulting from frequent densification (via cloning and splitting), we utilize a pruning stage identifying Gaussians with the least influence in the full reconstruction. This further reduces the memory cost of the scene representation while improving the rendering and training speed due to faster rasterization. To summarize, our contributions are as follows:

- We propose a simple yet powerful approach for compressing 3D Gaussian point clouds by quantizing per-point attributes leading to lower storage memory.
- We further improve the optimization of the Gaussians by quantizing the opacity coefficients and utilizing a progressive training strategy while controlling the number of Gaussians with a pruning stage.

– We ablate the different components of our approach showing their effectiveness in producing efficient 3D Gaussian representations. We achieve comparable quality as 3D-GS on a variety of datasets while being faster and more efficient.

2 Related Work

Neural fields or Implicit Neural Representations (INRs) have recently become a dominant representation for not just 3D objects [29, 30], but also audio [27, 39], images [11, 39, 40], and videos [6, 28]. Consequently, there is a big focus on improving the speed and efficiency of this line of methods. Since neural fields essentially use a neural network to represent a physical field, a number of works have been inspired by and have borrowed from the neural network compression techniques that we discuss first.

Compression for neural networks. Since the explosion of neural networks and their proliferation in the industry and applications, neural network compression and efficiency has gained a lot of attention. A typical compression scheme used for neural networks is quantization or discretization of the parameters to a smaller, finite precision and using entropy coding or other lossless compression methods to further store the parameters. While some approaches directly train binary or finite precision networks [9, 10, 25, 34], others attempt to quantize the network using non-uniform scalar quantization [2, 15, 32, 46], or vector quantization [7, 8, 18]. Advantage of former techniques is typically cheaper setup cost and training time, however they can often result in sub-optimal network performance at the inference time. Another line of work attempt to prune the networks either during the training [19,24,35] or in a post-hoc optimization step [12, 13, 16, 36] which may require retraining the entire network. While pruning can be often a good compression strategy, these method may require a lot more training to reach a competitive performance as an unpruned network.

Compression for neural fields. Several neural field compression approaches [38, 40, 43] propose a meta learning approach that learns a network on auxiliary datasets which can provide a good initialization for the downstream network. While our method can benefit from meta-learning as well, we restrict our current approach to compressing a single scene for brevity. VQAD [41] propose a vector quantization for a hierarchical feature grids used in NGLOD [42]. Their method is able to achieve higher compression as compared to other feature-grid methods such as Instant NGP [30] however its training can be memory intensive and it struggles to achieve the same quality of reconstructions as compared to some other NeRF variants such as MipNeRF. [26] propose a similar compression approach using voxel pruning and codebook quantization. Scalar quantization approaches [5, 17] reparameterize the network weights with integers and apply further entropy regularization to compress the scene even further. While these approaches require lower training memory as compared to [42], they are sensitive to hyperparameters and the reconstruction efficacy of these approaches remain lower as compared to MipNeRF360 or Gaussian Splatting.

In this work, we show that it is possible to compress 3D Gaussian point cloud representations which can retain high reconstruction quality with much smaller storage memory, lower training time and higher FPS for inference.

3 Background

3D-GS consists of a Gaussian point cloud representation in 3D space. Each Gaussian consists of various attributes such as the position (for mean), scaling and rotation coefficients (for covariance), opacity and color attributes. These Gaussians are used for rendering images from certain viewpoints by anisotropic volumetric "splatting" [47,48] of 3D Gaussians onto a 2D plane. This is done by projecting the 3D points to 2D and then using a differentiable tile-based rasterizer for blending together different Gaussians.
3D Gaussians with a mean 3D position vector $\boldsymbol{x}$ and covariance matrix Σ are defined as

$$G(\boldsymbol{x}) = e^{-\frac{1}{2}\boldsymbol{x}^T\Sigma^{-1}\boldsymbol{x}} \tag{1}$$

The 3D covariance matrix is in turn defined using a scale matrix S (represented using a 3D scale vector s) and rotation matrix R (represented using a 4D rotation vector r) as

$$\Sigma = RSS^TR^T \tag{2}$$

For a camera viewpoint with a projective transform P (world-to-camera matrix) and J as the Jacobian of the affine approximation of the projective transform, the corresponding covariance matrix projection [21] to 2D is written as:

$$\Sigma' = JP\Sigma P^TJ^T \tag{3}$$

The color of a pixel C is then computed using $\mathcal{N}$ Gaussian points overlapping the pixel. The points are sorted based on their depth values and blended as:

$$C = \sum_{i\in\mathcal{N}} \boldsymbol{c}_i\alpha_i \prod_{j=1}^{i-1}(1-\alpha_j) \tag{4}$$

where α_i is obtained by computing the 2D Gaussian at the pixel location multiplied with a scalar opacity value. The color c_i of each Gaussian is then computed using spherical harmonic coefficients [37].

The Gaussians are initialized using the sparse point clouds created by Structure from Motion (SfM) [44]. For a view sampled from the training dataset, the corresponding image is projected and rasterized with the forward process explained above. The reconstruction loss is then computed by combining $\mathcal{L}_1$ with SSIM loss as

$$\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{SSIM}} \tag{5}$$

with λ set to 0.2. The attributes are optimized by Stochastic Gradient Descent as the rendering process is fully differentiable.

Another key step in the optimization of the Gaussians is controlling the number of Gaussians. After a warmup-phase, Gaussians with a low opacity value α below a threshold are removed every 100 iterations. Additionally, large Gaussians (bigger than the corresponding geometry) are split while small Gaussians are cloned in order to better fit the underlying geometric shape. Only Gaussians with positional gradients above a threshold τ_{thresh} after every 100 iterations are split or cloned.

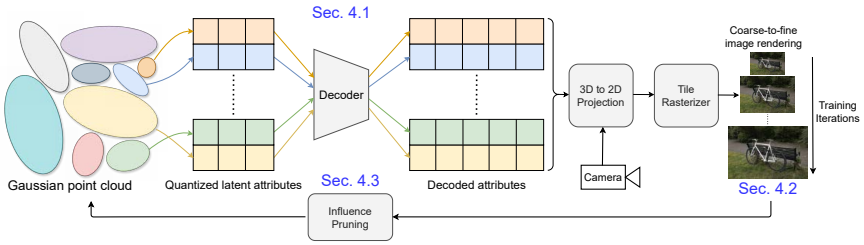

Fig. 2: Approach: 1) We quantize the attributes of the latents to reduce the storage memory of the Gaussians (Sec. 4.1), 2) progressively train using a coarse-to-fine rendering resolution schedule to obtain higher quality reconstructions (Sec. 4.1 and 3) utilize a pruning stage to obtain fewer Gaussians and faster training/rendering speeds (Sec. 4.3).

4 Method

4.1 Attribute quantization

Each Gaussian point consists of a position vector $\boldsymbol{p} \in \mathbb{R}^3$, scaling coefficient $\boldsymbol{s} \in \mathbb{R}^3$, rotation quarternion vector $\boldsymbol{r} \in \mathbb{R}^4$, opacity scalar $o \in \mathbb{R}$ and spherical harmonics coefficients $\boldsymbol{c} \in \mathbb{R}^d$, with $d = 3 * f^2$, where f corresponds to the harmonics degree. Thus, for a degree of 4 (as is used in [22]), the color coefficients make up more than 80% of the dimensions of the full attribute vector. 3D-GS typically requires millions of Gaussians for representing the scene with high quality. However, a set of 1 million Gaussians consume around 236 MB of disk space when storing the full attribute vector with a 32-bit floating point. Thus, to reduce the memory required for storing each attribute vector, we propose to use a set of quantized representations. A visualization of the various components of our approach is provided in Fig. 2.

For any given attribute, we maintain a quantized latent vector $\boldsymbol{q} \in \mathbb{Z}^l$ with dimension l, consisting of integer values. We then use an MLP decoder $D : \mathbb{Z}^l \rightarrow \mathbb{R}^k$ to decode the latents and obtain the attributes. As quantized vectors are not differentiable, we maintain continuous approximations $\widehat{\boldsymbol{q}}$ during training and use the Straight-Through Estimator (STE) which rounds off $\widehat{\boldsymbol{q}}$ to the nearest integer and directly passes the gradient during backpropagation. We get

$$\boldsymbol{q} = STE(\widehat{\boldsymbol{q}}), \quad \boldsymbol{a} = D(\boldsymbol{q}) \tag{6}$$

The latents are thus trained end-to-end similar to the standard procedure of 3D-GS. Post training, we round $\widehat{\boldsymbol{q}}$ to the nearest integer and use entropy coding for efficiently storing the latents along with the decoder D. While each vector in the attribute set $\mathbb{A} = \{\boldsymbol{p}, \boldsymbol{s}, \boldsymbol{r}, \boldsymbol{c}, o\}$ can be quantized, we do not encode the base band color SH coefficient, the scaling coefficients and the position vector as they are sensitive to initialization and result in large performance drops when quantized. While it is possible to improve feature compression with additional tools such as complex decoders, learnable probability models [1] or Gumbel annealing [45] and so on, they introduce a large overhead in various metrics such as runtime GPU memory and training speed. We aim to utilize an approach which quantizes per-point attributes with little to no cost to these efficiency metrics while still maintaining the reconstruction quality.

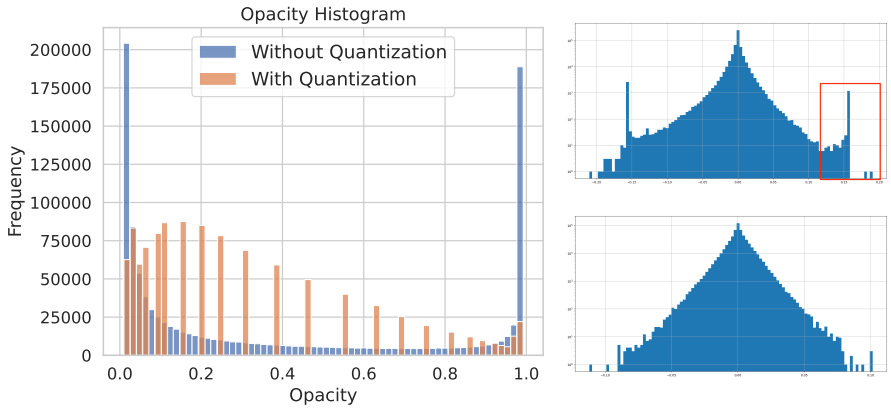

Fig. 3: Left: Histogram of opacity coefficients with and without quantization. Most coefficients result in values of 0 or 1 without quantization while quantization spreads the opacity values and allows for better blending. Right: Opacity gradient visualization for 3D-GS (top) and EAGLES (bottom). We reduce outlier gradients while high positive gradients in 3D-GS saturates erroneous Gaussians which are not pruned.

Opacity quantization for improved optimization. While the color and rotation attributes are quantized to reduce the memory footprint, quantizing the opacity coefficients not only reduces memory but improves the optimization process resulting in lesser artifacts in the rendered views. In Fig. 3 (left), we visualize the histogram of opacity coefficients of all Gaussian points, with and without quantization. We see that most points converge to 0 or 1 without quantization which is primarily due to large magnitude gradients (top right). While a large negative gradient reduces opacity which can be pruned, a large positive gradient saturates the opacity to 1 leading to artifacts in the rasterization process which is not removed. In contrast, the gradient distribution with quantized opacity coefficients (bottom right) shows fewer outlier gradients and produces a relatively more uniform set of opacities (left). Quantization acts as a soft regularizer as it requires more gradient updates to increase the opacity value from one quantization bin to the next higher bin, thus preventing opacity saturation. In Sec. 5.3, we show how opacity quantization has the added benefit of removing artifacts normally present in 3D-GS.

4.2 Progressive training

Standard training of the Gaussians proceeds by computing the loss over the full image resolution. This results in a more complex loss landscape as the Gaussians are forced to fit to fine features of the scene early in the training. As the SfM initialization is only sparse and several attributes are initialized with rough estimates, the optimization can be suboptimal and result in floating artifacts from Gaussians which are not removed during the optimization. We thus propose a coarse-to-fine training strategy by initially rendering at a small scene resolution and gradually increasing the size of the rendered image views over a period of the training iterations until reaching the full resolution. By starting with small images, the Gaussian points easily converge to a good loss min-

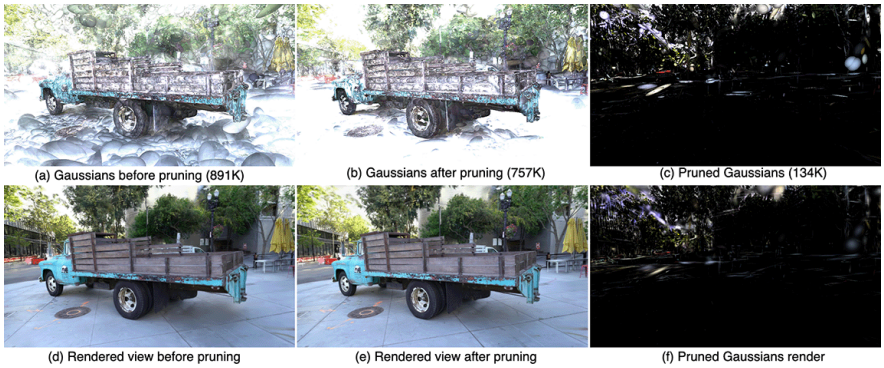

Fig. 4: Effect of influence pruning: Our approach identifies Gaussians which do not contribute significantly to the rasterization process and prune them without drop in reconstruction quality.

ima. This produces better initializations for the creation of further Gaussians through the densification process of cloning and splitting. As the render resolution increases, more Gaussians can be fit to better reconstruct the finer features of the scene. Such a progressive training procedure also helps remove artifacts typically obtained from the rasterization of ill-optimized Gaussians as we show in Sec. 5.3. This serves as a soft regularization scheme for the creation and deletion of Gaussians. Another added benefit of progressive training is that fewer Gaussians are required to represent coarser scenes while also rendering fewer pixel locations, thereby leading to faster rendering and backpropagation during training. This directly leads to lower training times while still improving the reconstruction quality of the scene upon convergence.

4.3 Influence Pruning

The densification process of cloning and splitting occurs every 100 iterations. This, however, leads to an explosion in the number of Gaussians as a large number exceed the gradient threshold and are either cloned or split. While this can allow for representing finer scene details, a significant fraction of the Gaussians are redundant and lead to large training, rendering times as well as memory usage. 3D-GS removes Gaussians below an opacity threshold. However, the Gaussians can have high opacity values while not influencing the rasterization process due to occlusions (when transmittance T reaches 0 after rendering previous Gaussians in the depth order). The points can also have small scale values influencing very few pixels. To identify the Gaussians most important for reconstructing the full scene, we utilize an influence metric during the rasterization process. More specifically, for the i^{th} Gaussian to be rendered at pixel location p, we define its influence on the pixel and its influence on the full scene as

$$W_{\text{i,p}} = \alpha_i T_i = \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j), \quad W_{\text{i}} = \sum_{p} W_{\text{i,p}} \tag{7}$$

where T_i measures the transmittance upto Gaussian i. We thus obtain a weight vector $\mathbf{W}$, with each element representing the importance of the corresponding Gaussian

for rendering in the full scene. A similar score was concurrently proposed in [31] but utilizing the max operator instead of sum as in our work. Gaussians with small scale values or low opacity values influence fewer pixels and have lower weight values when summed across all pixel locations. Additionally, Gaussians which do not influence the rasterization process ($T_i = 0$) have a weight value of zero. This is further visualized in Fig. 4, where for a given view render and a set of Gaussians (left), we obtain nearly identical reconstruction quality (center) with fewer Gaussians. On the right, we visualize the pruned Gaussians which correspond to either highly saturated regions with low transmittance or very small Gaussians with low scale. This metric has no computational overhead as the weight values are calculated directly during the rasterization process in Eq. (4). We thus obtain weight vectors for each iteration and accumulate the weight values over N iterations (set as a hyperparameter) to account for all training views of the full scene. After computing the weight vector, we identify a percentage of the Gaussians with lowest weights and prune them and continue the training process. We show further ablations in Sec. 5.3 to show the effect of the pruning stage in reducing the number of Gaussians while maintaining reconstruction quality. The proposed pruning stage thus removes the Gaussians with the least footprint for scene rendering while the densification process allows for increasing Gaussians to fit to finer scene details.

5 Experiments

5.1 Implementation and evaluation

We implemented our method by building on [22] which uses a PyTorch framework [33] with a CUDA backend for the rasterization operation. All numbers are reported on the RTX A6000 GPU, for consistency. Additional hyperparameters are provided in the supplementary. For progressive scaling, we start with a scale factor of 0.3, increasing to 1.0 in a cosine schedule. We provide sensitivity analysis of this scale factor in Sec. 5.3. We perform the scaling schedule for 70% of the total iterations after which training continues at the full resolution. We fix the opacity reset interval to every 2500 iterations, the densification frequency to 175 iterations and the pruning stage for every 5000 iterations until 25000 iterations. At each stage we remove 15% of the Gaussians although a higher value can lead to larger reductions at the cost of reconstruction quality. We optimize for 30000 iterations but can be controlled based on the time and memory budget for training. We fix the color SH degree to 3 as higher values result in little performance gain for a large increase in memory cost, even with quantization. We use this configuration of hyperparameters for all of our experiments unless mentioned otherwise.

We provide results on 9 scenes from the Mip-Nerf360 dataset [4], and 2 scenes each from Tanks&Temples [23], Deep Blending [20] for a total of 13 scenes. These datasets correspond to real-world high resolution scenes which can be unbounded and provide a challenging scenario with parts of the scene scarcely seen during training. We follow the methodology of [4, 22] with every 8th view used for evaluation and the rest for training. We evaluate the quality of reconstructions primarily with PSNR, and also with the SSIM and LPIPS metrics. We calculate memory of all quantized and non-quantized parameters of the Gaussians for the storage size. The rendering and training memory measures the peak GPU RAM for the full training/rendering phase.

Table 1: Comparison of our approach with prior work in view synthesis on three datasets. * corresponds to our runs on the existing codebase for fair evaluation. We perform competitively in terms of reconstruction metrics while outperforming in terms of efficiency metrics.

Dataset (→)	Mip-NeRF360						Tanks&Temples					
Method	PSNR ↑	SSIM ↑	LPIPS ↓	Storage Mem ↓	FPS ↑	Train Time ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Storage Mem ↓	FPS ↑	Train Time ↓
Plenoxels	23.08	0.63	0.46	2.1GB	7	25m49s	21.08	0.72	0.38	2.3GB	13	25m5s
INGP	25.59	0.70	0.33	48MB	9	7m30s	21.92	0.75	0.31	48MB	14	6m59s
M-NeRF360	27.69	0.79	0.24	9MB	0.06	48h	22.22	0.76	0.26	9MB	0.14	48h
3D-GS	27.21	0.82	0.21	734MB	134	41m33s	23.61	0.84	0.18	411MB	154	26m54s
3D-GS*	27.45	0.81	0.22	745MB	110	23m20s	23.63	0.85	0.18	430MB	157	12m5s
EAGLES (Ours)	27.23	0.81	0.24	54MB	131	21m34s	23.37	0.84	0.20	29MB	227	11m39s
EAGLES-Small	26.94	0.80	0.25	47MB	166	17m3s	23.10	0.82	0.22	19MB	272	10m7s
EAGLES-Fast	26.99	0.81	0.23	71MB	111	16m24s	23.02	0.83	0.20	38MB	190	8m43s

We measure the frame rate or Frames Per Second (FPS) based on the time taken to render from all cameras in the scene dataset. Before measuring FPS, we decode all latent attributes using our decoder which is a one-time amortized cost of loading the parameters. For a fair benchmark, the quantitative results comparison of other works in Tab. 1 are provided by using the numbers reported in [22], unless mentioned otherwise. The qualitative results are from our own runs of the respective methods.

5.2 Benchmark comparison

For NeRFs, we compare against MipNeRF360 [4] and two fast approaches of INGP [30], and Plenoxels [14]. For our primary baseline, 3D-GS, we provide numbers as reported in [22] and also from our own runs. We show results of our approach for 3 variants: a) training for 30K iterations until convergence b) a smaller configuration corresponding to more pruning c) training for 21K iterations which is the end of the progressive training schedule. We summarize the results on all 3 datasets in Tables 1 and 2.

We outperform the voxel-grid based method of Plenoxels on all datasets and metrics. Compared to INGP [30], another fast Nerf-based method, our approach at 21K iterations, obtains better quality reconstructions at comparable training times on Deep Blending, Tanks&Temples but higher training times for Mip-NeRF360 dataset. We bridge the gap between NeRF-based methods and Gaussian Splatting in terms of storage memory, by obtaining smaller sizes than INGP (Ours-Small configuration) while still obtaining better reconstruction metrics. We also obtain much higher rendering speeds ($>15\times$) compared to INGP on all datasets paving the way for compact 3D representations with high quality reconstructions and real-time rendering. Against the Mip-NeRF360 approach, we perform competitively in terms of PSNR with a 0.45 dB drop on their dataset and 1.15dB,0.56dB gain on Tanks&Temples, Deep Blending respectively. While their model is compact in terms of number of parameters they are extremely slow to train ($\sim$48h) and render ($<$1FPS). Finally, our reconstructions are on par with 3D-GS achieving minimal performance drops of 0.22dB, 0.26dB PSNR

Table 2: Comparison of our approach on the Deep Blending dataset. We improve reconstruction quality in terms of PSNR compared to 3D-GS while also improving on all efficiency metrics of storage memory, FPS and training time.

Dataset (→)	Deep Blending					
Method	PSNR ↑	SSIM ↑	LPIPS ↓	Storage Mem ↓	FPS ↑	Train Time ↓
Plenoxels	23.06	0.80	0.51	2.7GB	11	27m49s
INGP	24.96	0.82	0.39	48MB	3	8m
M-NeRF360	29.40	0.90	0.25	8.6MB	0.09	48h
3D-GS	29.41	0.90	0.24	676MB	137	36m2s
3D-GS*	29.55	0.90	0.25	656MB	123	23m5s
EAGLES (Ours)	29.86	0.91	0.25	52MB	130	21m50s
EAGLES-Small	29.92	0.90	0.25	33MB	160	17m40s
EAGLES-Fast	29.85	0.91	0.25	63MB	108	16m30s

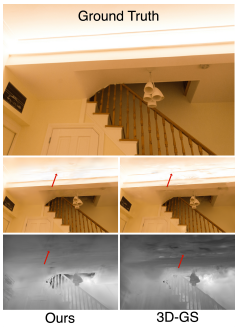

Fig. 5: 3D-GS produces artifacts (right) at various scene locations (arrow) while our rendering produces more scene consistent depth (left).

on the Mip-NeRF360, Tanks&Temples datasets respectively while gaining 0.31dB on Deep Blending. We reduce storage size by $\sim 15\times$ making the representation suitable for devices with limited memory budgets. We also accelerate training and rendering compared to 3D-GS obtaining higher FPS and lower train times on all scenes. Our approach reaches close to convergence with good visual quality at 21K iterations (marking the end of the progressive scaling period). Note that a fair amount of time is spent on training after 21K iterations due to the full scale render resolution.

We show qualitative results of our approach and other baselines on unseen test views from indoor and outdoor scenes in Fig. 6. Mip-NeRF360 exhibits blurry artifacts such as the grass in the Stump scene (2nd from left) or even incorrect artifacts as seen in the edges of the leaf in Kitchen (right). We obtain reconstructions with quality on-par with 3D-GS or even better reconstructions close to scene boundaries such as the branches in Bicycle (left), grass in Stump (2nd from left). Notably, 3D-GS tends to exhibit numerous floaters at the edges, especially in areas not frequently observed during training. We provide additional visualizations of this in Fig. 5, showcasing notably smoother reconstructions at scene boundaries, such as the Room's ceiling (3rd from left). This points to a more refined optimization of the point cloud using our approach. We ablate the different components of our approach and analyze their effect in Sec. 5.3 below.

5.3 Ablations

For a deeper understanding of our approach, we provide qualitative visualizations for the Train scene and quantitative results for three scenes from each of the 3 datasets, incorporating each component step by step. Results are summarized in Table 3 and Fig. 7. "Vanilla" corresponds to the baseline 3D-GS. First, quantizing the color, rotation and opacity attributes for each Gaussian significantly reduces storage memory with a

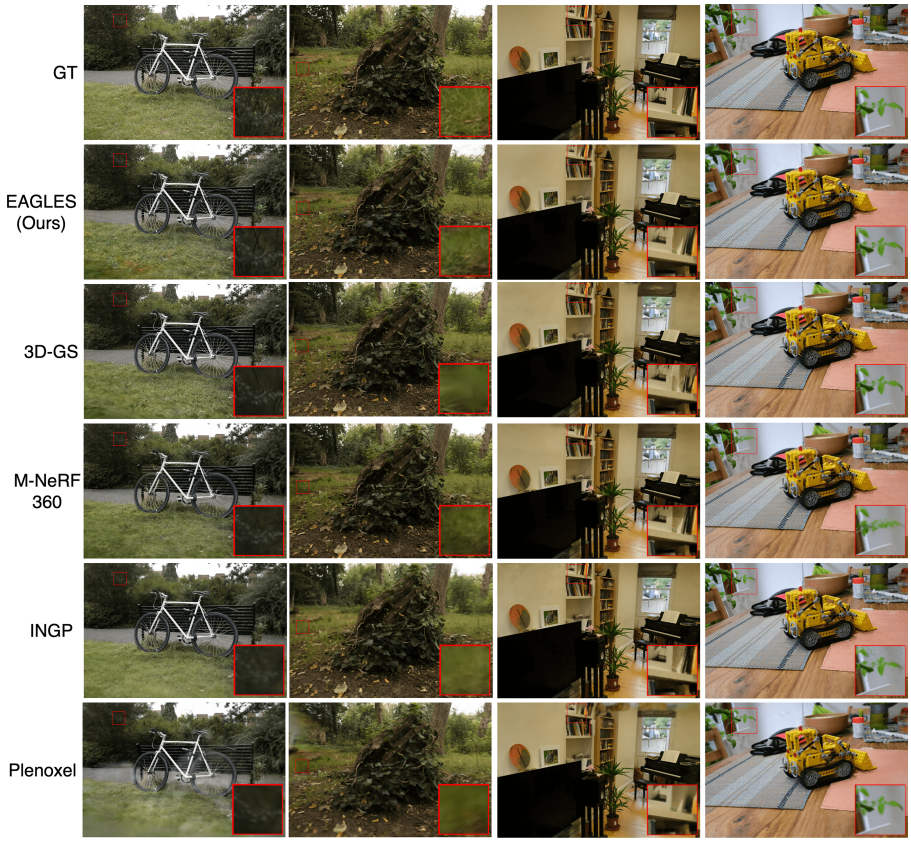

Fig. 6: Qualitative comparison of our method with prior work on various scenes. We obtain on par or better reconstructions with the SOTA 3D-GS and MiP-NeRF360 while efficient NeRF methods such as INGP, Plenoxels struggle to reconstruct fine detail. We avoid blurry artifacts at scene boundaries exhibited by 3D-GS and MiP-NeRF360.

small drop in PSNR or reconstruction quality. Note that the bulk of the memory post quantization is from the non quantized attributes of scale, position, base color. The quantized attributes are compressed from 220 MB, 452 MB, 1046 MB to 6 MB, 12 MB, 28 MB for the 3 scenes respectively achieving $\sim 20-30\times$ memory reduction. We visualize the effect of color and rotation quantization for a single unseen view from the "Train" scene in Fig. 7. Notice the floaters/rendering artifacts for 3D-GS at the top left of the scene as it has little overlap with training views. Quantizing color and rotation does not directly remove these artifacts but opacity quantization significantly improves the visual quality of the rendering as erroneous Gaussians do not saturate quickly.

We then include progressive scaling increasing the rendering resolution in a cosine schedule. We achieve gains in PSNR with fewer floating artifacts due to a more stable optimization while significantly reducing training time as we show in Tab. 4. Progressive scaling also provides a better optimization of the loss landscape removing any re-

Table 3: Ablation of various components of our approach. Attribute quantization significantly reduces the storage memory for a marginal PSNR cost. Progressive training mostly improves the PSNR due to better optimization while the controlled densification and pruning stage signficantly speeds up training and rendering while also reducing storage size at the same or higher PSNR.

Method	Train (Tanks & Temples)				Playroom (Deep Blending)				Bicycle (Mip-NeRF360)			
	PSNR	Storage Mem	Num. Gaussians	FPS	PSNR	Storage Mem	Num. Gaussians	FPS	PSNR	Storage Mem	Num. Gaussians	FPS
Vanilla	**21.94**	262MB	1.11M	177	30.07	542MB	2.29M	144	**25.13**	1254MB	5.31M	61
+ Quantization	21.60	46MB	1.03M	179	**30.48**	82MB	1.81M	142	24.86	192MB	4.19M	65
+ Progressive	21.63	38MB	0.85M	194	30.39	75MB	1.67M	140	25.07	190MB	4.19M	71
+ Densification	21.62	29MB	0.64M	202	30.40	54MB	1.20M	146	25.02	142MB	3.11M	82
+ Pruning	21.65	**21MB**	**0.46M**	**234**	30.38	**36MB**	**0.80M**	**169**	25.04	**104MB**	**2.26M**	**87**

Fig. 7: Opacity quantization and progressive retraining removes floaters. Adding the pruning stage does not significantly affect quality while improving efficiency.

maining foggy artifacts as seen in Fig. 7. Finally, increasing the densification interval to 175 leads to fewer Gaussians without loss in reconstruction quality (penultimate row). Beyond this value, we observe a sharp drop off in reconstruction quality. However, the pruning stage continues to decrease the number of Gaussians resulting in lower storage memory, training time, and higher FPS without sacrificing on reconstruction quality in terms of PSNR. This is depicted in the views at Figs. 4 and 7 where the reconstruction quality is similar to without pruning although it reintroduces minor artifacts.

To further analyze the strength of progressive training, we vary the resize scale and visualize the PSNR-model size tradeoff and the convergence speed as well in Fig. 8. We run experiments on the Truck scene and average over 3 random seeds with error intervals reported. On the left, we see that decreasing the scale upto 0.3 has no effect on PSNR but reduces the number of Gaussians needed to represent that scene. Beyond this value, dropoffs in PSNR is observed for lower storage memory. On the right, we analyze the convergence speed in terms of the iteration time over the course of training

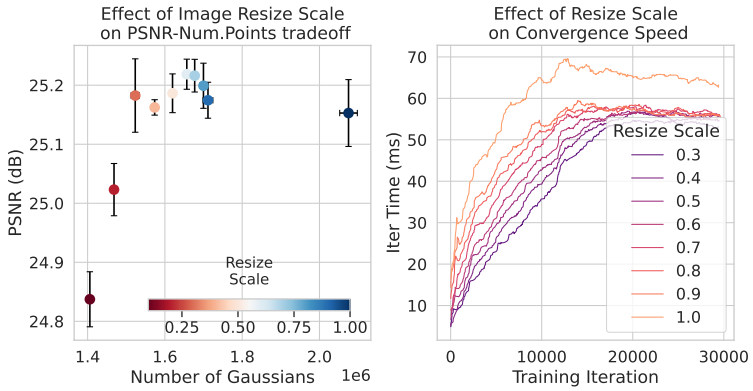

Fig. 8: Effect of progressive resize scale. (Left) Decreasing initial scale reduces number of Gaussians with no PSNR cost. (Right) This improves training speed with lesser time per iteration.

for various scaling values. As expected, we consistently obtain lower iteration time for lower scale values even with no loss in PSNR (left).

5.4 Progressive scaling variants

We now analyze the effect of applying different types of filters to the image as part of the coarse-to-fine training procedure. Results are summarized in Table 4. We try different types of strategies such as a) the mean filter which corresponds to downsampling and re-upsampling the image with bilinear interpolation b) a gaussian filter c) standard downsampling as used for our experiments and d) with no type of progressive training. For downsampling and mean filtering, we start with a scale of 0.3 and end at 1.0 which corresponds to resizing the image to 30% of its dimensions and scaling upto its original size gradually for a period of 70% of the iterations. For Gaussian filtering, we progressively decrease the filter size from the initial value specified in the table down to 1×1, equating to no filtering. Compared to the no filter case, all other types of filters result in fewer Gaussians leading to lower memory, training time and higher FPS. Both Gaussian and Mean filters provide large gains in efficiency metrics almost no PSNR drops. The Gaussian filter naturally provides a coarse-to-fine schedule for training Gaussian points. Nonetheless, the training still proceeds at full resolution and the largest gains in terms of training time is produced with downsampling. The 5×5 Gaussian filter produces similar results as downsampling albeit with higher training times while we observe a larger Gaussian filter 15×15 leads to much higher efficiency at the cost of PSNR.

5.5 Training and Rendering Memory

In this section, we show the memory consumption of our approach and 3D-GS on the 3 datasets in Table 5. We measure peak GPU memory used during the training or rendering phase by our approach and 3D-GS. We see that we require much lesser memory

Table 4: Various types of progressive scaling. Downsampling reduces training time the most while high Gaussian filter sizes improves size and FPS at the cost of reconstruction quality.

Filter Type	PSNR	Storage Mem	Num. Gaussians	FPS	Training Time
None	23.34dB	43MB	0.95M	211	13m27s
Mean	23.31dB	27MB	0.61M	280	11m41s
Gaussian (5×5)	23.41dB	34MB	0.74M	248	12m8s
Gaussian (7×7)	23.36dB	28MB	0.61M	276	11m32s
Gaussian (15×15)	23.17dB	**21MB**	**0.46M**	**321**	10m51s
Downsample	**23.41dB**	34MB	0.75M	244	**9m49s**

Table 5: We consume lesser training, rendering memory than 3D-GS across all scenes.

Method	Bicycle		Truck		Playroom	
	Train	Render	Train	Render	Train	Render
3D-GS	17.4G	9.5G	8.5G	4.8G	9.6G	6.0G
EAGLES	**10G**	**7.4G**	**5.3G**	**3.6G**	**7.1G**	**5.3G**

during training even with latents and decoders. Since our quantization decodes the latents to floating point values before a forward or backward pass, no gains are obtained in terms of runtime memory consumption for each Gaussian. However, with progressive training and the pruning stage, we obtain significantly lower number of Gaussians leading to lower runtime memory during training/rendering. For the Bicycle scene especially, compared to the 17G required by 3D-G, we consume only 10G GPU RAM during training making it practical for many consumer GPUs with 12G RAM.

5.6 Limitations

The proposed quantization is sensitive to initialization of position, scaling and baseband color attributes and quantization of these attributes for further compression is an interesting avenue for future research. Additionally, we perform rendering at full-precision after decoding. Further run-time speedups can be obtained by half-precision floats or integer representations via accelerated arithmetic operations while rendering.

6 Conclusion

In this work, we proposed a simple yet powerful approach for efficient novel view synthesis. We build upon the seminal work of 3D Gaussian Splatting [22], and propose major improvements that reduces the storage requirements by 10-20$\times$, along with lower training cost, faster inference time, and on par reconstruction quality. We achieve this by 3 major improvements over the prior work - attribute quantization for per-point compression, progressive training for faster training and better reconstruction, and a pruning stage for reducing number of points for the scene representation. Our extensive quantitative and qualitative analyses shows the efficacy of our approach in 3D representation.

Acknowledgements: This work was partially supported by IARPA via Department of Interior/Interior Business Center (DOI/IBC) contract number 140D0423C0076. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The authors acknowledge UMD's supercomputing resources made available for conducting this research. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DOI/IBC, or the U.S. Government.

References

1. Ballé, J., Minnen, D., Singh, S., Hwang, S.J., Johnston, N.: Variational image compression with a scale hyperprior. arXiv preprint arXiv:1802.01436 (2018)
2. Banner, R., Nahshan, Y., Hoffer, E., Soudry, D.: Post-training 4-bit quantization of convolution networks for rapid-deployment. arXiv preprint arXiv:1810.05723 (2018)
3. Barron, J.T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., Srinivasan, P.P.: Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5855–5864 (2021)
4. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5470–5479 (2022)
5. Bird, T., Ballé, J., Singh, S., Chou, P.A.: 3d scene compression through entropy penalized neural representation functions. In: 2021 Picture Coding Symposium (PCS). pp. 1–5. IEEE (2021)
6. Chen, H., He, B., Wang, H., Ren, Y., Lim, S.N., Shrivastava, A.: Nerv: Neural representations for videos. Advances in Neural Information Processing Systems **34**, 21557–21568 (2021)
7. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing neural networks with the hashing trick. In: International conference on machine learning. pp. 2285–2294. PMLR (2015)
8. Chen, W., Wilson, J., Tyree, S., Weinberger, K.Q., Chen, Y.: Compressing convolutional neural networks in the frequency domain. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1475–1484 (2016)
9. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in neural information processing systems. pp. 3123–3131 (2015)
10. Dettmers, T., Lewis, M., Belkada, Y., Zettlemoyer, L.: Llm. int8 (): 8-bit matrix multiplication for transformers at scale. arXiv preprint arXiv:2208.07339 (2022)
11. Dupont, E., Goliński, A., Alizadeh, M., Teh, Y.W., Doucet, A.: Coin: Compression with implicit neural representations. arXiv preprint arXiv:2103.03123 (2021)
12. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2018)
13. Frankle, J., Dziugaite, G.K., Roy, D.M., Carbin, M.: Pruning neural networks at initialization: Why are we missing the mark? arXiv preprint arXiv:2009.08576 (2020)
14. Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance fields without neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5501–5510 (2022)
15. Girish, S., Gupta, K., Singh, S., Shrivastava, A.: Lilnetx: Lightweight networks with extreme model compression and structured sparsification. arXiv preprint arXiv:2204.02965 (2022)
16. Girish, S., Maiya, S.R., Gupta, K., Chen, H., Davis, L.S., Shrivastava, A.: The lottery ticket hypothesis for object recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 762–771 (2021)

17. Girish, S., Shrivastava, A., Gupta, K.: Shacira: Scalable hash-grid compression for implicit neural representations. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 17513–17524 (2023)
18. Gong, Y., Liu, L., Yang, M., Bourdev, L.: Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115 (2014)
19. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. arXiv preprint arXiv:1506.02626 (2015)
20. Hedman, P., Philip, J., Price, T., Frahm, J.M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering. ACM Transactions on Graphics (ToG) **37**(6), 1–15 (2018)
21. Hoaglin, D.C., Welsch, R.E.: The hat matrix in regression and anova. The American Statistician **32**(1), 17–22 (1978)
22. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics (ToG) **42**(4), 1–14 (2023)
23. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and temples: Benchmarking large-scale scene reconstruction. ACM Transactions on Graphics (ToG) **36**(4), 1–13 (2017)
24. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in neural information processing systems. pp. 598–605 (1990)
25. Li, F., Zhang, B., Liu, B.: Ternary weight networks. arXiv preprint arXiv:1605.04711 (2016)
26. Li, L., Shen, Z., Wang, Z., Shen, L., Bo, L.: Compressing volumetric radiance fields to 1 mb. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4222–4231 (2023)
27. Luo, A., Du, Y., Tarr, M., Tenenbaum, J., Torralba, A., Gan, C.: Learning neural acoustic fields. Advances in Neural Information Processing Systems **35**, 3165–3177 (2022)
28. Maiya, S.R., Girish, S., Ehrlich, M., Wang, H., Lee, K.S., Poirson, P., Wu, P., Wang, C., Shrivastava, A.: Nirvana: Neural implicit representations of videos with adaptive networks and autoregressive patch-wise modeling. arXiv preprint arXiv:2212.14593 (2022)
29. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: ECCV (2020)
30. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics (ToG) **41**(4), 1–15 (2022)
31. Niemeyer, M., Manhardt, F., Rakotosaona, M.J., Oechsle, M., Duckworth, D., Gosula, R., Tateno, K., Bates, J., Kaeser, D., Tombari, F.: Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. arXiv preprint arXiv:2403.13806 (2024)
32. Oktay, D., Ballé, J., Singh, S., Shrivastava, A.: Scalable model compression by entropy penalized reparameterization. arXiv preprint arXiv:1906.06624 (2019)
33. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
34. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European conference on computer vision. pp. 525–542. Springer (2016)
35. Reed, R.: Pruning algorithms-a survey. IEEE transactions on Neural Networks **4**(5), 740–747 (1993)
36. Savarese, P., Silva, H., Maire, M.: Winning the lottery with continuous sparsification. Advances in Neural Information Processing Systems **33**, 11380–11390 (2020)
37. Seeley, R.T.: Spherical harmonics. The American Mathematical Monthly **73**(4P2), 115–121 (1966)

38. Sitzmann, V., Chan, E., Tucker, R., Snavely, N., Wetzstein, G.: Metasdf: Meta-learning signed distance functions. Advances in Neural Information Processing Systems **33**, 10136–10147 (2020)
39. Sitzmann, V., Martel, J., Bergman, A., Lindell, D., Wetzstein, G.: Implicit neural representations with periodic activation functions. Advances in neural information processing systems **33**, 7462–7473 (2020)
40. Strümpler, Y., Postels, J., Yang, R., Gool, L.V., Tombari, F.: Implicit neural representations for image compression. In: Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVI. pp. 74–91. Springer (2022)
41. Takikawa, T., Evans, A., Tremblay, J., Müller, T., McGuire, M., Jacobson, A., Fidler, S.: Variable bitrate neural fields. In: ACM SIGGRAPH 2022 Conference Proceedings. pp. 1–9 (2022)
42. Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M., Fidler, S.: Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11358–11367 (2021)
43. 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: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2846–2855 (2021)
44. Ullman, S.: The interpretation of structure from motion. Proceedings of the Royal Society of London. Series B. Biological Sciences **203**(1153), 405–426 (1979)
45. Yang, Y., Bamler, R., Mandt, S.: Improving inference for neural image compression. Advances in Neural Information Processing Systems **33**, 573–584 (2020)
46. Zhang, D., Yang, J., Ye, D., Hua, G.: Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In: Proceedings of the European conference on computer vision (ECCV). pp. 365–382 (2018)
47. Zwicker, M., Pfister, H., Van Baar, J., Gross, M.: Ewa volume splatting. In: Proceedings Visualization, 2001. VIS'01. pp. 29–538. IEEE (2001)
48. Zwicker, M., Pfister, H., Van Baar, J., Gross, M.: Surface splatting. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. pp. 371–378 (2001)