# FlexGS: Train Once, Deploy Everywhere with Many-in-One Flexible 3D Gaussian Splatting

Supplementary Material

## A. Theoretical Analysis

### A.1. Differentiable Gaussian Selection

To substantiate the claims made in Sec. 3.2 regarding the ability of the proposed GsNet and Gumbel-Softmax [6] mechanisms to enable Adaptive Gaussian Selection in a differentiable manner, the following derivation is provided. As shown in Eq. 1 and Eq. 2, with the logits $\boldsymbol{z} \in \mathbb{R}^{N\times C}$, where $N$ represent the number of Gaussians and $C$ denotes to the number of classes (set to 2), a Gumbel noise sampling is conducted, whereby noise is integrated and the temperature parameter is appropriately scaled by $\tau$.

$$g_{i,c} = -\log\left(-\log\left(U_{i,c}\right)\right), U_{i,c} \sim \text{Uniform}(0,1). \tag{1}$$

$$\tilde{\boldsymbol{z}}_{i,c} = \frac{\boldsymbol{z}_{i,c} + g_{i,c}}{\tau}. \tag{2}$$

Then a softmax function is used for calculating soft output.

$$\boldsymbol{z}_{\text{soft},i,c} = \frac{\exp\left(\tilde{\boldsymbol{z}}_{i,c}\right)}{\sum_{k=1}^{C}\exp\left(\tilde{\boldsymbol{z}}_{i,k}\right)} \tag{3}$$

Alternatively, discrete hard outputs may be derived from the soft outputs for utilization in forward propagation.

$$\boldsymbol{z}_{\text{hard},i,c} = \begin{cases} 1 & \text{if } c = \arg\max_k \boldsymbol{z}_{\text{soft},i,k} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The **Straight-Through Estimator** is employed to reconcile the discrete nature of hard outputs with the differentiable characteristics of soft outputs within the hard Gumbel-Softmax framework:

$$\begin{aligned} B_i =& \boldsymbol{z}_{\text{hard},i} - \boldsymbol{z}_{\text{soft},i} + \boldsymbol{z}_{\text{soft},i} \\ =& \boldsymbol{z}_{\text{hard},i} + \left(\boldsymbol{z}_{\text{soft},i} - \boldsymbol{z}_{\text{soft},i}\right) \\ =& \boldsymbol{z}_{\text{hard},i} + \text{stop\_gradient}(\boldsymbol{z}_{\text{soft},i}) \end{aligned} \tag{5}$$

where $\text{stop\_gradient}(\boldsymbol{z}_{\text{soft},i})$ signifies that during backpropagation, the gradient associated with $\boldsymbol{z}_{\text{soft},i}$ is disregarded, thereby exclusively preserving the value of $\boldsymbol{z}_{\text{hard},i}$.

For an entire batch of size $N$, let the input matrix $\boldsymbol{z} \in \mathbb{R}^{N\times 2}$ and the output matrix $B \in \mathbb{R}^{N\times 2}$ be defined. While the selected mask $\hat{\boldsymbol{M}} = \boldsymbol{z}_{output}[:, -1]$. The gradient matrix $\frac{\partial \hat{\boldsymbol{M}}}{\partial \boldsymbol{z}} \in \mathbb{R}^{N\times 2}$ is delineated as follows:

$$\frac{\partial \hat{\boldsymbol{M}}}{\partial \boldsymbol{z}} = \frac{1}{\tau}\begin{bmatrix} -B_{1,0}B_{1,1} & B_{1,1}\left(1-B_{1,1}\right) \\ -B_{2,0}B_{2,1} & B_{2,1}\left(1-B_{2,1}\right) \\ \vdots & \vdots \\ -B_{N,0}B_{N,1} & B_{N,1}\left(1-B_{N,1}\right) \end{bmatrix} \tag{6}$$

where, the temperature parameter is denoted by $\tau$. The probabilities of the $i$-th sample belonging to class 0 and class 1 are represented by $B_{i0}$ and $B_{i1}$, respectively. The ellipsis indicates that this pattern continues similarly for all $N$ samples. The aforementioned gradient matrix can also be expressed in a vectorized form as follows:

$$\frac{\partial \hat{\boldsymbol{M}}}{\partial \boldsymbol{z}} = \frac{1}{\tau}\begin{bmatrix} -B[:,0]\odot B[:,1] & B[:,1]\odot\left(1-B[:,1]\right) \end{bmatrix}, \tag{7}$$

where $\odot$ signifies element-wise multiplication. From Eq. 7, we can observe that the gradient of each parameter in GsNet can be calculated based on the "chain rule".

### A.2. Gradients of Gaussian Attributes

To elucidate the computational procedure, we hereby redefine the notations previously employed in Sec. 3.1. The current opacity of the specific Gaussian $i$ within the rendering process for pixel $p$ is illustrated in Eq. 8.

$$\alpha_i = \hat{o}_i \cdot \boldsymbol{G}_2(i,p), \quad \hat{o}_i = o_i \cdot \hat{\boldsymbol{M}}_i, \tag{8}$$

where $\hat{o}_i$ is the masked opacity for the selected Gaussians and $\boldsymbol{G}_2(i,p)$ denotes the effect coefficient of the 2D projection of the Gaussian $i$ to the pixel $p$.

With the given ratio $e$, the rendering loss of the selected Gaussian can be calculated as below:

$$\mathcal{L}_s = |\boldsymbol{I}_s^e - \boldsymbol{I}_{GT}|. \tag{9}$$

Specially, for the $i-$th Gaussian interacted with pixel $p$ on rendered Image $\boldsymbol{I}_s^e$, the gradient of the Gaussian attribute $\mu$ can be calculated as shown in Eq. 10.

$$\begin{aligned} \frac{\partial \mathcal{L}_s}{\partial \mu_i} =& \frac{\partial \mathcal{L}_s}{\partial \hat{\boldsymbol{o}}_i}\cdot\frac{\partial \hat{\boldsymbol{o}}_i}{\partial \mu_i}\cdot\boldsymbol{G}_2 + \frac{\partial \mathcal{L}_s}{\partial \boldsymbol{G}_2}\cdot\frac{\partial \boldsymbol{G}_2}{\mu_i}\cdot\hat{\boldsymbol{o}}_i \\ =& \boldsymbol{G}_2\cdot\frac{\partial \mathcal{L}_s}{\partial \hat{\boldsymbol{o}}_i}\frac{\partial o_i\cdot\hat{\boldsymbol{M}}_i}{\partial \mu_i} + o_i\hat{\boldsymbol{M}}_i\cdot\frac{\partial \mathcal{L}_s}{\partial \boldsymbol{G}_2}\frac{\partial \boldsymbol{G}_2}{\mu_i} \\ =& o_i\boldsymbol{G}_2\cdot\frac{\partial \mathcal{L}_s}{\partial \hat{\boldsymbol{o}}_i}\frac{\partial \hat{\boldsymbol{M}}_i}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \mu_i} + o_i\hat{\boldsymbol{M}}_i\cdot\frac{\partial \mathcal{L}_s}{\partial \boldsymbol{G}_2}\frac{\partial \boldsymbol{G}_2}{\mu_i} \end{aligned} \tag{10}$$

Where $\frac{\partial \boldsymbol{z}}{\partial \mu_i}$ is the gradient of GsNet to $\mu_i$. Other attributes of Gaussians can also be calculated in the same process, with the gradient $\frac{\partial \hat{\boldsymbol{M}}_i}{\partial \boldsymbol{z}}$ is calculated in 7.

## B. Implementation Details

### B.1. Training Details

In Sec. 3.2, the dimensionality $D$ of the hidden features in GsNet, employed for adaptive selection, is set to 64. To en-

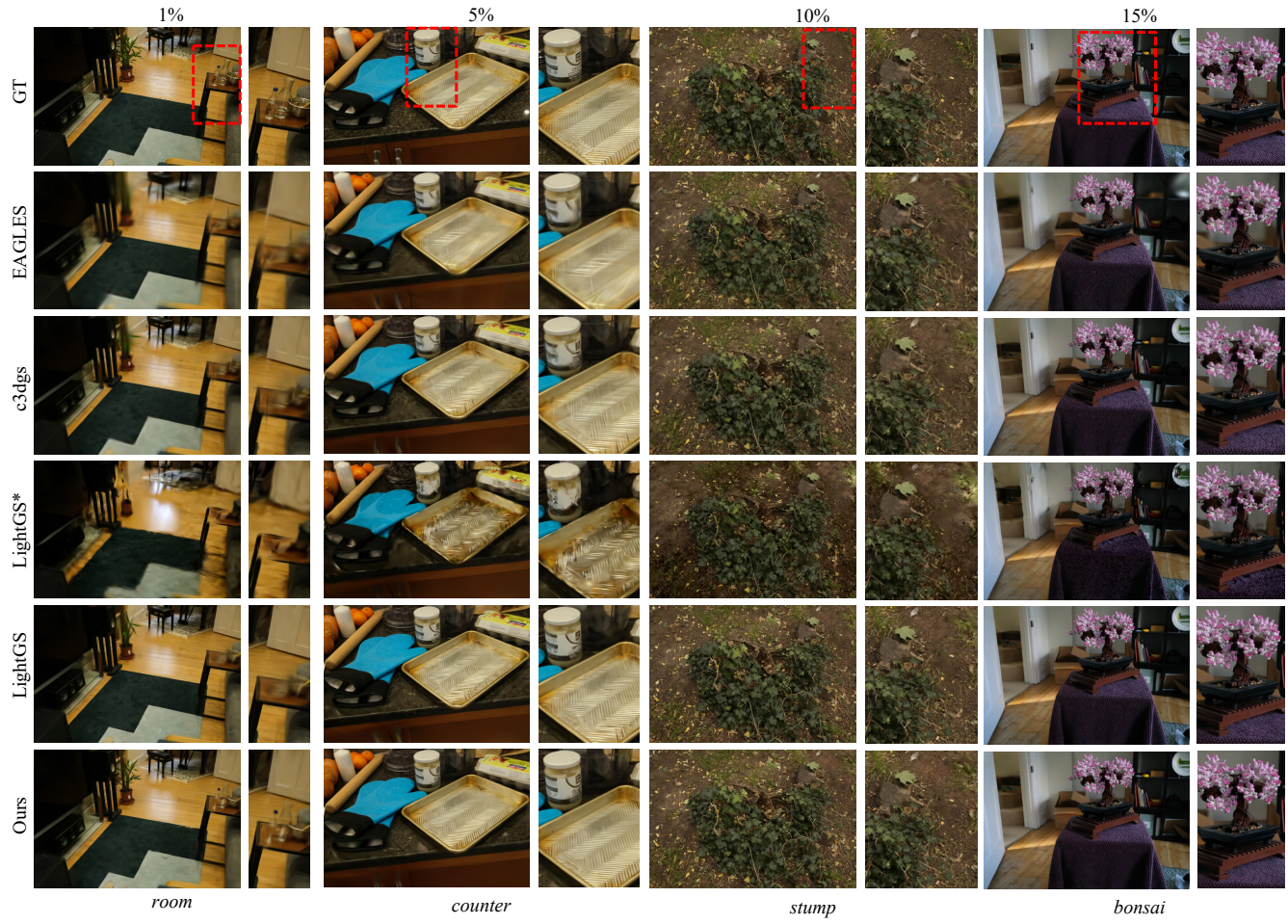


Figure 1. Visual results compared with other methods on various elastic ratios:{0.01,0.05,0.10,0.15} on Mip-NeRF360 [1]: {*bicycle*, *room, counter, stump, bonsai*}.

hance computational efficiency, the implementation of the Spatial-Ratio Neural Field proposed in Sec. 3.3 adheres to the configurations outlined in [4, 9], utilizing six planes $\{(x, y), (x, z), (y, z), (x, e), (y, e), (z, e)\}$ to model the 4D voxel space. The resolutions across the four dimensions ($x$, $y$, $z$, and $e$) are configured as $\{64, 64, 64, 100\}$. Additionally, the hidden feature dimension of the Multi-Head Predictor for predicting the transformation under the given ratio is set to $64$.

### B.2. Inference Details

During elastic inference, in contrast to the training phase where the opacity of Gaussians is multiplied by the binary mask values, we directly discard the unselected Gaussians. Furthermore, we observe that despite enforcing sparsity supervision on the masks predicted by GsNet, the number of activated entries within the predicted mask does not exactly achieve the desired ratio. For instance, a target ratio of $0.20$ results in approximately selecting $19.5\%$ of all Gaussians. Therefore, to attain an accurate elastic ratio, during inference, we employ Pytorch's `F.gumbel_softmax` function with its parameter `hard=False` to output continuous logits and select the top $\lfloor eN \rfloor$ logits out of $N$.

## C. More Experimental Results

In this section, we provide more pre-scene results. Visual comparisons on four scenes of Mip-NeRF360 [1] $\{$*bonsai*, *counter*, *room*, *stump*$\}$ under the given ratios $\{0.01, 0.05, 0.10, 0.15\}$ are shown in Fig. 1. The breakdown results of quantitative comparisons on each scene of the tested datasets are from Tab. 1 to Tab. 8.

**How useful is the elastic inference in real application scenarios?** In practical applications, the loading and deployment of pre-trained Gaussian models inherently de-

Ratio: 1% Load time: **2s** Ratio: 10% Load time: **10s** Ratio: 10% Load time: **2min**

Figure 2. Potential use of elastic inference in the application scenario of incremental scene loading.

mand a considerable amount of time. Moreover, as shown in Fig 2, the aggregate loading time escalates proportionally with the number of Gaussian models being deployed. Elastic inference enables the rapid deployment of lower-precision, coarse-grained models, while simultaneously maximizing rendering quality within a given resource budget. It can further allow for the incremental loading of higher-precision, detail-rich models. This enhances the user experience of 3D scene deployment scenarios over time, like mobile gaming and online VR shopping.

Table 1. Quantitative results of FlexGS across various elastic ratios compared with other methods on Mip-NeRF360:{*bicycle*} [1] (LightGS* denotes the LightGaussian without finetuning after pruning).

| Method | 1% | | | 5% | | | 10% | | | 15% | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| LightGS* [3] | 14.561 | 0.3327 | 0.5782 | 16.379 | 0.4445 | 0.4498 | 18.235 | 0.5388 | 0.3716 | 20.042 | 0.6207 | 0.3163 |
| LightGS [3] | 21.896 | 0.4814 | 0.5335 | 23.222 | 0.5952 | 0.3989 | 24.131 | 0.6769 | 0.3209 | **24.714** | 0.7230 | 0.2714 |
| C3DGS [8] | 21.740 | 0.4770 | 0.5350 | 23.110 | 0.5780 | 0.4110 | 23.910 | 0.6560 | 0.3390 | 24.460 | 0.7060 | 0.2930 |
| EAGLES [5] | 21.326 | 0.4549 | 0.5618 | 22.970 | 0.5600 | 0.4400 | 23.690 | 0.6300 | 0.3900 | 23.530 | 0.6300 | 0.3600 |
| Ours | **22.385** | **0.5350** | **0.4806** | **23.988** | **0.6865** | **0.3330** | **24.476** | **0.7302** | **0.2769** | 24.596 | **0.7408** | **0.2576** |

Table 2. Quantitative results of FlexGS across various elastic ratios compared with other methods on Mip-NeRF360:{*bonsai*} [1] (LightGS* denotes the LightGaussian without finetuning after pruning).

| Method | 1% | | | 5% | | | 10% | | | 15% | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| LightGS* [3]* | 15.231 | 0.4839 | 0.5181 | 19.071 | 0.6481 | 0.3567 | 21.774 | 0.7607 | 0.2632 | 24.025 | 0.8326 | 0.2020 |
| LightGS [3] | 22.397 | 0.6844 | 0.4185 | 27.359 | 0.8399 | 0.2327 | 29.374 | 0.9025 | 0.1580 | 30.590 | 0.9289 | 0.1220 |
| C3DGS [8] | 21.810 | 0.6640 | 0.4420 | 25.700 | 0.8020 | 0.2660 | 28.240 | 0.8810 | 0.1780 | 29.560 | 0.9150 | 0.1380 |
| EAGLES [5] | 20.601 | 0.6263 | 0.4921 | 24.660 | 0.7700 | 0.3300 | 26.420 | 0.8300 | 0.2500 | 27.450 | 0.8600 | 0.2200 |
| Ours | **24.420** | **0.7415** | **0.3511** | **28.449** | **0.8775** | **0.2024** | **30.605** | **0.9296** | **0.1324** | **31.606** | **0.9452** | **0.1063** |

Table 3. Quantitative results of FlexGS across various elastic ratios compared with other methods on Mip-NeRF360:{*counter*} [1] (LightGS* denotes the LightGaussian without finetuning after pruning).

| Method | 1% | | | 5% | | | 10% | | | 15% | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| LightGS* [3] | 14.866 | 0.4669 | 0.5304 | 18.713 | 0.6149 | 0.3806 | 21.300 | 0.7064 | 0.3025 | 23.176 | 0.7671 | 0.2496 |
| LightGS [3] | 22.474 | 0.6886 | 0.4167 | 25.882 | 0.8085 | 0.2644 | 27.481 | 0.8590 | 0.2002 | 28.348 | 0.8857 | 0.1652 |
| C3DGS [8] | 21.920 | 0.6640 | 0.4420 | 25.820 | 0.8040 | 0.2660 | 27.390 | 0.8640 | 0.1920 | 28.120 | 0.8890 | 0.1580 |
| EAGLES [5] | 21.750 | 0.6700 | 0.4400 | 23.330 | 0.7400 | 0.3600 | 24.640 | 0.7900 | 0.2900 | 25.420 | 0.8200 | 0.2500 |
| Ours | **23.367** | **0.7282** | **0.3647** | **26.151** | **0.8298** | **0.2402** | **27.577** | **0.8807** | **0.1733** | **28.264** | **0.8997** | **0.1453** |

Table 4. Quantitative results of FlexGS across various elastic ratios compared with other methods on Mip-NeRF360:{*flowers*} [1] (LightGS* denotes the LightGaussian without finetuning after pruning).

| Method | 1% | | | 5% | | | 10% | | | 15% | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| LightGS* [3] | 12.883 | 0.2349 | 0.6578 | 15.128 | 0.3436 | 0.5264 | 16.853 | 0.4242 | 0.4596 | 18.211 | 0.4808 | 0.4204 |
| LightGS [3] | 18.274 | 0.3487 | 0.6104 | 19.837 | 0.4813 | 0.4680 | 20.763 | 0.5511 | 0.4090 | 21.306 | 0.5857 | 0.3776 |
| C3DGS [8] | 18.100 | 0.3340 | 0.6160 | 19.570 | 0.4530 | 0.4790 | 20.400 | 0.5200 | 0.4250 | 20.920 | 0.5560 | 0.3970 |
| EAGLES [5] | 17.862 | 0.3158 | 0.6461 | 19.390 | 0.4300 | 0.5300 | 20.110 | 0.4800 | 0.4700 | 20.530 | 0.5200 | 0.4400 |
| Ours | **18.363** | **0.3771** | **0.5656** | **21.489** | **0.5828** | **0.3836** | **21.691** | **0.5972** | **0.3646** | **21.699** | **0.5989** | **0.3581** |

Table 5. Quantitative results of FlexGS across various elastic ratios compared with other methods on T&T:{*train*} [7] (LightGS* denotes the LightGaussian without finetuning after pruning).

| Method | 1% | | | 5% | | | 10% | | | 15% | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| LightGS* | 11.907 | 0.3357 | 0.5643 | 15.106 | 0.5287 | 0.3958 | 17.144 | 0.6497 | 0.2905 | 18.711 | 0.7287 | 0.2264 |
| LightGS | 18.304 | 0.5845 | 0.4506 | 21.201 | 0.7721 | 0.2476 | 22.241 | 0.8363 | 0.1682 | **22.835** | **0.8655** | **0.1298** |
| C3DGS | 17.657 | 0.5397 | 0.4790 | 20.526 | 0.7540 | 0.2159 | 21.595 | 0.8217 | 0.1852 | 22.274 | 0.8513 | 0.1427 |
| EAGLES | 16.715 | 0.4825 | 0.5370 | 18.794 | 0.6452 | 0.3798 | 19.633 | 0.7145 | 0.3021 | 20.175 | 0.7545 | 0.2583 |
| Ours | **19.189** | **0.6607** | **0.3726** | **21.629** | **0.8042** | **0.2135** | **22.475** | **0.8519** | **0.155** | 22.830 | 0.8645 | 0.1354 |

Table 6. Quantitative results of FlexGS across various elastic ratios compared with other methods on T&T:{*truck*} [7] (LightGS* denotes the LightGaussian without finetuning after pruning).

| Method | 0.0100 | | | 0.0500 | | | 0.1000 | | | 0.1500 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| LightGS* | 11.449 | 0.4011 | 0.4998 | 15.287 | 0.6407 | 0.2709 | 18.501 | 0.7652 | 0.1750 | 21.179 | 0.8381 | 0.1261 |
| LightGS | 20.641 | 0.7227 | 0.3079 | 24.993 | 0.8952 | 0.1057 | 26.478 | 0.9284 | 0.0631 | **27.130** | **0.9395** | 0.0505 |
| C3DGS | 20.551 | 0.7201 | 0.3087 | 24.457 | 0.8877 | 0.1135 | 26.028 | 0.9241 | 0.0681 | 26.736 | 0.9360 | 0.0532 |
| EAGLES | 18.346 | 0.6084 | 0.4331 | 21.753 | 0.7874 | 0.2317 | 23.164 | 0.8466 | 0.1624 | 24.034 | 0.8773 | 0.1282 |
| Ours | **22.433** | **0.8061** | **0.2248** | **25.374** | **0.9062** | **0.0934** | **26.587** | **0.9324** | **0.0579** | 27.000 | 0.9393 | **0.0501** |

Table 7. Quantitative results of FlexGS across various elastic ratios compared with other methods on Zip-NeRF:{*Berlin*} [2] (LightGS* denotes the LightGaussian without finetuning after pruning).

| Method | 1% | | | 5% | | | 10% | | | 15% | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| LightGS* | 15.628 | 0.6578 | 0.5101 | 20.121 | 0.7457 | 0.4177 | 22.374 | 0.7951 | 0.3629 | 23.897 | 0.8278 | 0.3287 |
| LightGS | 20.693 | 0.7432 | 0.4554 | 24.272 | 0.8194 | 0.3672 | 25.782 | 0.8549 | 0.3204 | 26.563 | 0.8722 | 0.2958 |
| C3DGS | 20.243 | 0.7351 | 0.4679 | 23.055 | 0.7887 | 0.4047 | 24.694 | 0.8290 | 0.3491 | 25.383 | 0.8479 | 0.3238 |
| EAGLES | 18.984 | 0.7198 | 0.4728 | 21.543 | 0.7598 | 0.4422 | 23.045 | 0.7927 | 0.4001 | 23.987 | 0.8134 | 0.3724 |
| Ours | **21.560** | **0.7664** | **0.4257** | **24.936** | **0.8377** | **0.3407** | **26.242** | **0.8669** | **0.3000** | **26.770** | **0.8793** | **0.2794** |

Table 8. Quantitative results of FlexGS across various elastic ratios compared with other methods on Zip-NeRF:{*London*} [2] (LightGS* denotes the LightGaussian without finetuning after pruning).

| Method | 1% | | | 5% | | | 10% | | | 15% | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| LightGS* | 15.829 | 0.5365 | 0.5922 | 19.372 | 0.6468 | 0.4942 | 21.456 | 0.7029 | 0.4324 | 22.798 | 0.7400 | 0.3916 |
| LightGS | 20.481 | 0.6535 | 0.5475 | 23.474 | 0.7274 | 0.4498 | 24.681 | 0.7677 | 0.3935 | 25.358 | 0.7914 | 0.3592 |
| C3DGS | 20.162 | 0.6426 | 0.5582 | 22.843 | 0.7028 | 0.4833 | 24.032 | 0.7428 | 0.4291 | 24.644 | 0.7674 | 0.3857 |
| EAGLES | 18.241 | 0.6136 | 0.5655 | 20.826 | 0.6638 | 0.5370 | 22.248 | 0.6943 | 0.4956 | 23.027 | 0.7164 | 0.4643 |
| Ours | **21.29** | **0.6697** | **0.5234** | **24.015** | **0.7458** | **0.4200** | **25.086** | **0.7834** | **0.3652** | **25.540** | **0.8012** | **0.3383** |

## References


[1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 2, 4

[2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19697–19705, 2023. 5

[3] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245*, 2023. 4

[4] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12479–12488, 2023. 2

[5] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. *arXiv preprint arXiv:2312.04564*, 2023. 4

[6] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 1

[7] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4): 1–13, 2017. 5

[8] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10349–10358, 2024. 4

[9] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20310–20320, 2024. 2