Supplementary for GS-LRM: Large Reconstruction Model for 3D Gaussian Splatting

Kai Zhang*[1] Sai Bi*[1] Hao Tan*[1] Yuanbo Xiangli[2]
Nanxuan Zhao[1] Kalyan Sunkavalli[1] Zexiang Xu[1]

[1]Adobe Research [2]Cornell University

1 More Results

Please refer to our anonymous website https://sai-bi.github.io/project/gs-lrm/ for more visual results. We have provided **interactive viewers** on websites for better visualization experience.

More results of each datasets are in the sub-pages of the website as well, e.g., GSO in https://sai-bi.github.io/project/gs-lrm/page_gso.html, ABO in https://sai-bi.github.io/project/gs-lrm/page_abo.html, and RealEstate10k in https://sai-bi.github.io/project/gs-lrm/page_rel10k_1.html and https://sai-bi.github.io/project/gs-lrm/page_rel10k_2.html. We also provide a website clone in the supplementary materials, which the reviewers can open locally.

2 Implementation Details

2.1 Pseudo Code

We list the pseudo code of our *GS-LRM* in Algorithm 1 [1]. The code implements the method that we discussed in the main method section, and also the Gaussian parametrization detailed later in Sec. 2.4.

2.2 Additional Model Details

We do not use bias term in the whole model, which includes both Linear and LayerNorm layers. We initialize the model weights with a normal distribution of zero-mean and 0.02-stddev.

[0] * Equal contribution.

[1] To avoid the the shaping constraints of the CUDNN kernel, we use Einops [7] and Linear layer to implement the patchify and unpatchify operator, but they are conceptually the same to the conv/deconv operator. For clarity, we use conv/deconv in pseudo code.

Algorithm 1 GS-LRM pseudo code.

```
# Input list:
# image: [b, n, h, w, 3], n is number of views; h and w are the height and width
# extrinsic: [b, n, 4, 4]
# intrinsic: [b, n, 4]
#
# Output list (3D GS parameters):
# xyz: [b, *, 3], rgb [b, *, 3], scaling [b, *, 3], rotation [b, *, 4], opacity [b, *, 1]

# GS-LRM transformer
o, d = rays_from_camera(extrinsic, intrinsic) # rays_origin, rays_direction: [b, n, h, w,
      3]
x = concat([image, o, cross(o, d)], dim=-1) # [b, n, h, w, 9]
x = conv(x, out=d, kernel=8, stride=8) # patchify to [b, n, h/8, w/8, d]
x = x.reshape(b, -1, d) # Sequentialize as transformer input [b, n * h/8 * w/8, d]
x = transformer(LN(x))
x = x.reshape(b, n, h//8, w//8, d)
x = deconv(LN(x), out=12, kernel=8, stride=8) # unpatchify to GS output: [b, n, h, w, 12]
x = x.reshape(b, -1, 12) # Simply merge all Gaussians together [b, n * h * w, 12]

# GS parameterization
distance, rgb, scaling, rotation, opacity = x.split([1, 3, 3, 4, 1], dim=-1)
w = sigmoid(distance)
xyz = o + d * (near * (1 - w) + far * w)
scaling = min(exp(scaling - 2.3), 0.3)
rotation = rotation / rotation.norm(dim=-1, keepdim=True)
opacity = sigmoid(opacity - 2.0)

return xyz, rgb, scaling, rotation, opacity
```

2.3 Additional Training Details

We train our model with AdamW [5] optimizer. The β_1, β_2 are set to 0.9 and 0.95 respectively. We use a weight decay of 0.05 on all parameters except the those of the LayNorm layers. We use a cosine learning rate decay with linear warmup. We take 2000 step of warm up and the peak learning rate is set to $4e-4$. The model is trained for 80K iterations on the 256-res training, and then fine-tuned with 512-res for another 20K iterations. Finetuing at 512-res uses 500-step warmup with the peak learning rate $1e-4$ in the cosine learning rate decay schedule. We use a per-GPU batch size of 8 objects/scenes during 256-res training, and a per-GPU batch size of 2 during 512-res finetuning. For each object, we use 4 input views and 4 novel supervision views at each iteration of 256-res and 512-res training; for each scene, we use 2 input views and 6 novel supervison views, following the protocol of pixelSplat. We use $\lambda = 0.5$ to balance the MSE loss and Perceptual loss. To enable efficient training and inference, we adopt Flash-Attention-v2 [2] in the xFormers [4] library, gradient checkpointing [1], and mixed-precision training [6] with BF16 data type. We also apply deferred backpropagation [8] for rendering the GS to save GPU memory. 256-res training takes about 2 days on 64 A100 (40G VRAM) GPUs, while 512-res finetuning costs 1 additional day.

2.4 3D Gaussian Parameterization

As 3D Gaussians are unstructured explicit representation (i.e., different from Triplane-NeRF's structural implicit representation), the parameterization of the

output parameters can largely affect the model's convergence. For here, the 'structural' and 'unstructual' mainly refer to whether each token have a determinitic spatial meaning. We discuss in detail how we implement the parameterization for reproducibility.

Scale and opacity. For the scale and opacity of the Gaussian Splatting, we apply the default activations used by Gaussian Splatting [3] to map the range to $\mathbb{R}^+$ and $(0, 1)$. For scale, we use exponential activation (which is $\mathbb{R} \to \mathbb{R}^+$). For opacity, we use the Sigmoid activation defined as $\sigma(x) = 1/(1+\exp(-x))$ (which is $\mathbb{R} \to (0, 1)$). Besides the activations, we also want the initial output to be close to 0.1. We accompolished this by adding constant bias to the transformer's output to shift the initialization. We also clip the scales at a maximum size of 0.3. This max-scale clipping is applied because we empirically found that the scale of the 3D Gaussian can be very large (and the Gaussian will be degenerated to a long line) without such clipping. These line-shaped Gaussians can spread the gradients to multiple pixels after splatting and we found that it hurt the training stability. In summary, according to the main method section, suppose $\mathbf{G}_{\text{ij}}$ is the output parameters of Gaussians, and we split it into the components of $(\mathbf{G}_{\text{rgb}}, \mathbf{G}_{\text{scale}}, \mathbf{G}_{\text{rotation}}, \mathbf{G}_{\text{opacity}}, \mathbf{G}_{\text{distance}})$; he equation for scale and opacity would be

$$\text{scale} = \min\{\exp\left(\mathbf{G}_{\text{scale}} - 2.3\right), 0.3\}, \tag{1}$$

$$\text{opacity} = \sigma(\mathbf{G}_{\text{opacity}} - 2.0), \tag{2}$$

where $\exp(-2.3)$ and $\sigma(-2.0)$ are both 0.1 approximately. As the output $\mathbf{G}_{\text{ij}}$ is designed to be zero-mean after the model-weight initialization (we also remove all bias terms as mentioned earlier). We can approximately get an output GS initialization that we desire. Note that this initialization does not need to be accurate because it is mostly used to help training stability.

Rotation. We predict unnormalized quaternions and use L2-normalize as activations to get unit quaternions.

RGB. We directly interpret the model output as the zero-order Spherical Harmonics coefficients used in Gaussian Splatting implementation [3]. We do not use higher-order Spherical Harmonics in this work for simplicity. We leave it to future work to improve the view-dependent modelling of our *GS-LRM*.

XYZ and Distance. In the main paper, we briefly mention that the Gaussian center is parameterized as $\text{xyz} = \text{ray}_o + t \cdot \text{ray}_d$. To convert the transformers' output $\mathbf{G}_{\text{distance}}$ to t, we employ an empirical near distance d_{near} and far distance d_{far}. We then map our model output to the range of $(d_{\text{near}}, d_{\text{far}})$. The conversion is defined as

$$\omega = \sigma(\mathbf{G}_{\text{distance}}), \tag{3}$$

$$t = (1 - \omega)\ d_{\text{near}} + \omega\ d_{\text{far}}. \tag{4}$$

The d_{near} and d_{far} are set differently for the object-level *GS-LRM* and scene-level *GS-LRM*. For object-level *GS-LRM*, we use $d_{\text{near}} = 0.1$ and $d_{\text{far}} = 4.5$;

this is aligned with our rendered training data. For scene-level *GS-LRM*, we use $d_{\text{near}} = 0.0$ and $d_{\text{far}} = 500$ to account for the large depth variations of indoor and outdoor scenes.

2.5 Camera Pose Normalization

For object-level *GS-LRM*, we did not use any camera normalization, as the object sizes are pre-normalized before data generation. For scene-level *GS-LRM*, we first compute the mean camera pose by averaging all input camera poses and consider the coordinate frame of the mean camera as the world space, i.e. making input poses relative to the mean pose for subsequent processes. We then scale all input camera locations, so that they are located within the unit box ($[-1, 1]^3$) in the world space.

References

1. Chen, T., Xu, B., Zhang, C., Guestrin, C.: Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174 (2016) 2
2. Dao, T.: Flashattention-2: Faster attention with better parallelism and work partitioning. arXiv preprint arXiv:2307.08691 (2023) 2
3. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics **42**(4) (2023) 3
4. Lefaudeux, B., Massa, F., Liskovich, D., Xiong, W., Caggiano, V., Naren, S., Xu, M., Hu, J., Tintore, M., Zhang, S., Labatut, P., Haziza, D., Wehrstedt, L., Reizenstein, J., Sizov, G.: xformers: A modular and hackable transformer modelling library. https://github.com/facebookresearch/xformers (2022) 2
5. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017) 2
6. Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al.: Mixed precision training. In: International Conference on Learning Representations (2018) 2
7. Rogozhnikov, A.: Einops: Clear and reliable tensor manipulations with einstein-like notation. In: International Conference on Learning Representations (2021) 1
8. Zhang, K., Kolkin, N., Bi, S., Luan, F., Xu, Z., Shechtman, E., Snavely, N.: Arf: Artistic radiance fields (2022) 2