Supplementary Material
Splat-SLAM: Globally Optimized RGB-only SLAM with 3D Gaussians

Erik Sandström[1,2†*] Ganlin Zhang[1*] Keisuke Tateno[2] Michael Oechsle[2] Michael Niemeyer[2]
Youmin Zhang[6] Manthan Patel[1] Luc Van Gool[5] Martin R. Oswald[4] Federico Tombari[2,3]
[1]ETH Zürich [2]Google [3]TU München [4]University of Amsterdam [5]INSAIT [6]Rock Universes
[*]Equal contribution [†]Work done while at internship at Google

This supplementary material accompanies the main paper and provides more details on the methodology and additional experimental results.

1. Method

We describe further details about our method that were left out from the main paper.

Comparison to Existing Works. To further clarify the differences between our method and existing 3DGS SLAM works, we classify each method in Tab. 9 based on important characteristics. It shows that our work is the first to include loop closure, proxy depth, RGB-only and online 3D Gaussian deformations.

	RGB-only	Loop Closure	Proxy Depth	Online 3DGS Deformations
GS-SLAM [13]	✗	✗	✓	✗
Gaussian-SLAM [15]	✗	✗	✓	✗
SplaTaM [4]	✗	✗	✓	✗
MonoGS [6]	✓	✗	✗	✗
Photo-SLAM [3]	✓	✓	✗	✗
Splat-SLAM (ours)	✓	✓	✓	✓

Table 9. **Method Classification.** We show that our method is the first to combine 3D Gaussian SLAM with loop closure, proxy depth and online 3D Gaussian map deformations in an RGB-only SLAM system.

Map Initialization. With map initialization, we refer to the process of anchoring new Gaussians during scene exploration. For every new keyframe to be mapped, we adopt the strategy that MonoGS [6] uses in pure RGBD mode. It works by unprojecting the depth reading per pixel to 3D and then downsampling this point cloud by a factor θ. New Gaussians are then assigned their means as the point cloud. The rotations are initialized to identity, the opacity to 0.5 and the scales are initialized related to their distance to the nearest neighbor point in the point cloud.

Keyframe Selection and Local Windowing. As mentioned in the main paper, we adopt the keyframe selection strategy from MonoGS [6]. We describe this strategy in the following.

Keyframes are selected based on the covisibility of the Gaussians. Between two keyframes i and j, the covisibility is defined using the Intersection over Union (IOU) and Overlap Coefficient (OC):

$$\text{IOU}_{\text{cov}}(i,j) = \frac{|\mathcal{G}_v^i \cap \mathcal{G}_v^j|}{|\mathcal{G}_v^i \cup \mathcal{G}_v^j|}, \tag{13}$$

$$\text{OC}_{\text{cov}}(i,j) = \frac{|\mathcal{G}_v^i \cap \mathcal{G}_v^j|}{\min(|\mathcal{G}_v^i|, |\mathcal{G}_v^j|)}, \tag{14}$$

where $\mathcal{G}_v^i$ are the Gaussians visible in keyframe i, based on the following definition of visibility. A Gaussian is seen as visible from a camera pose if it is used in the rasterization pipeline when rendering and if the accumulated transmittance $\prod_{j=1}^{i-1}(1-\alpha_j)$ has not yet reached 0.5.

A keyframe i is added to the keyframe window KFs if, given the last keyframe j, $\text{IOU}_{\text{cov}}(i,j) < k_{\text{fcov}}$ or if the relative translation $t_{ij} > k_{\text{fm}}\hat{D}_i$, where $\hat{D}_i$ is the median depth of frame i. For Replica, $k_{\text{fcov}} = 0.95$, $k_{\text{fm}} = 0.04$ and for TUM and ScanNet, $k_{\text{fcov}} = 0.90$, $k_{\text{fm}} = 0.08$. The registered keyframe j in KFs is removed if $\text{OC}_{\text{cov}}(i,j) < k_{\text{fc}}$, where keyframe i is the latest added keyframe. For all datasets, the cutoff is set to $k_{\text{fc}} = 0.3$. The size of the keyframe window is set to $|\text{KFs}| = 10$ for Replica and $|\text{KFs}| = 8$ for TUM and ScanNet.

Pruning and Densification We also follow [6] when it comes to Gaussian pruning and densification. Pruning is done based on the visibility: if new Gaussians inserted within the last 3 keyframes are not visible by at least 3 other frames in the keyframe window KFs, they are removed. Visibility-based pruning is only done when the keyframe window KFs is full. Additionally, every 150 mapping iterations, Gaussians with opacity lower than 0.7 are removed globally. Also Gaussians which project in 2D with a too large scale are removed. Densification is done as in [5], also at an interval of every 150 mapping iterations.

Final Refinement. We perform a few refinement iterations after the last final global BA. Also MonoGS [6] performs a

set of final iterations at the end of the SLAM trajectory to refine the colors.

Our refinement strategy is straight forward. We disable pruning and densification of the Gaussians and perform a set of optimization iterations β using the same loss function as in the main paper, but only sampling random single frames per iteration.

2. Mathematical Derivation of DSPO

In the main paper, we introduce DSPO (Disparity, Scale and Pose Optimization). Here, we provide the detailed derivation of how to optimize the proposed DSPO objective using the Gauss-Newton algorithm with the Schur Complement.

As described in the main paper, we optimize the following two objective functions alternatingly. The first one (same as the DBA optimization in [11]) optimizes the poses and disparity maps of the involved keyframes,

$$\arg\min_{\omega,d} \sum_{(i,j)\in\mathcal{E}} \left\| \tilde{p}_{ij} - K\omega_j^{-1}(\omega_i(1/d_i)K^{-1}[p_i,1]^T) \right\|^2_{\Sigma_{ij}} \quad . \tag{15}$$

The second objective optimizes the scale and shift factors of the mono prior and also the disparity maps of the keyframes,

$$\begin{aligned}\arg\min_{d^h,\theta,\gamma} &\sum_{(i,j)\in\mathcal{E}} \left\| \tilde{p}_{ij} - K\omega_j^{-1}(\omega_i(1/d_i^h)K^{-1}[p_i,1]^T) \right\|^2_{\Sigma_{ij}} \\ &+\alpha_1 \sum_{i\in\mathcal{V}} \left\| d_i^h - (\theta_i(1/D_i^{\text{mono}}) + \gamma_i) \right\|^2 \\ &+\alpha_2 \sum_{i\in\mathcal{V}} \left\| d_i^l - (\theta_i(1/D_i^{\text{mono}}) + \gamma_i) \right\|^2 \quad . \end{aligned} \tag{16}$$

For better readability, we do not show the conversion of 3D points to homogeneous coordinates in all equations.

DBA Optimization. First, we introduce how to solve the DBA optimization. To keep the notation consistent with [12], we do a mapping of the symbols we use in Eq. (15) such that

$$\{(p_i, \tilde{p}_{i,j}) \,|\, (i,j) \in \mathcal{E}\} \rightarrow \mathcal{M} = \{(x_i^k, x_j^k)\}_{k=1}^M \quad , \tag{17}$$

where (x_i^k, x_j^k) is a corresponding pixel pair (a feature match found by the optical flow estimation), k is the index in the total set of matches $\mathcal{M}$. For each k there are specific indices i and j indicating the corresponding keyframes of x_i^k and x_j^k, *i.e.* i and j are functions of k, but we write them as a subscript directly for simplicity. Furthermore, we map the camera poses

$$\{\omega_i\}_{i=1}^N \rightarrow \mathbf{T} = \{T_i \in SE(3) \mid 1 \le i \le N\} \quad , \tag{18}$$

where $\mathbf{T}$ are the camera to world poses of the corresponding keyframes. Then we can rewrite the objective function in Eq. (15) as

$$f(\mathbf{T}, \mathbf{d}) = \frac{1}{2} \sum_{(x_i^k, x_j^k)\in\mathcal{M}} \left\| r_k(T_i, T_j, x_i^k, x_j^k, d_i^k) \right\|^2_{w_k} \quad , \tag{19}$$

where $\mathbf{d} = \{d_i^k\}_{k=1}^M$ are the disparities (*i.e.* inverse depth), d_i^k is the disparity of pixel x_i^k, where i is still a function of k, but we write it as a subscript for simplicity. We denote $\| \cdot \|_{w_k}$ as the Mahalanobis distance with weighting matrix w_k. $w_k = \text{diag}(w_k^1, w_k^2)$ is a 2×2 diagonal matrix of the per-pixel and per-direction uncertainties of the optical flow prediction. We add $1/2$ to the objective function for mathematical convenience as it does not change the optimal solution. $r_k(\cdot) \in \mathbb{R}^2$ is the reprojection error function and it is defined as

$$\begin{aligned} &r_k(T_i, T_j, x_i^k, x_j^k, d_i^k) \\ =&x_j^k - KT_j^{-1}T_i(1/d_i^k)K^{-1}[x_i^k, 1]^T \quad . \end{aligned} \tag{20}$$

The formulation in Eq. (20) is the per-pixel equivalent of the part inside the norm $\| \cdot \|$ in Eq. (15).

To minimize Eq. (19), we resort to the Gauss-Newton algorithm. We can define the model parameters by the column vector $\beta = [T_j, T_i, d_i^k]^T \in \mathbb{R}^{13}$. We parameterize rotations and translations with the lie algebra se(3), so each camera pose is parameterized by 6 values and the $d_i^k \in \mathbb{R}$. Despite being an RGB-only system, optimization is done on se(3) because we fix the poses of the two first keyframes after initialization. This prevents gauge freedom *i.e.* drift in the global scale of the scene during optimization and therefore, se(3) optimization is sufficient. We can define the model as

$$g_k(\beta) = \begin{bmatrix} g_k^1(\beta) \\ g_k^2(\beta) \end{bmatrix} = KT_j^{-1}T_i(1/d_i^k)K^{-1}[x_i^k, 1]^T \quad . \tag{21}$$

since $g_k \in \mathbb{R}^2$ is not linear with respect to the parameters β we approximate g_k around a current estimate β_0 (at time 0) of the parameters with a first-order Taylor expansion. For each dimension of g_k, we get (here for the first dimension)

$$g_k^1(\beta) \approx g_k^1(\beta_0) + J_k(\beta - \beta_0) \tag{22}$$

where $J_k \in \mathbb{R}^{13}$ is the gradient row vector of g_k with respect to β at β_0. Plugging back Eq. (22) into Eq. (20), we get (for the first dimension)

$$\begin{aligned} r_k^1(\beta) &\approx \{x_j^k\}^1 - g_k^1(\beta_0) - J_k(\beta - \beta_0) \\ &= r_k^1(\beta_0) - J_k(\beta - \beta_0) \quad , \end{aligned} \tag{23}$$

where $\{x_j^k\}^1$ denotes the first dimension of x_j^k. The goal of the optimization is to minimize the squared residuals, *i.e.* for pixel k and dimension 1, the term $(r_k^1)^2$. We differentiate $(r_k^1)^2$ with respect to β and set the derivative to zero. This yields the expression

$$J_k^T J_k(\beta - \beta_0) + J_k^T r_k^1(\beta_0) = 0. \tag{24}$$

Now that we have derived the equation to solve (for β) for a single pixel and dimension, we can combine all observations

from all pixels (further details can be found in section 2.4 in [12]). We achieve this by stacking all residuals as a column vector $\mathbf{r} = (r_1^1, r_1^2, \cdots, r_M^1, r_M^2)^T \in \mathbb{R}^{2M}$. This yields the equivalent, vector-valued equation

$$\begin{aligned} &\mathbf{J}^T\mathbf{W}\mathbf{J}(\boldsymbol{\beta} - \boldsymbol{\beta_0}) + \mathbf{J}^T\mathbf{W}\mathbf{r} \\ &= \mathbf{J}^T\mathbf{W}\mathbf{J}\Delta\boldsymbol{\beta} + \mathbf{J}^T\mathbf{W}\mathbf{r} \\ &= \mathbf{J}^T\mathbf{W}\mathbf{J}\begin{bmatrix}\Delta\mathbf{T} \\ \Delta\mathbf{d}\end{bmatrix} + \mathbf{J}^T\mathbf{W}\mathbf{r} = 0 \ . \end{aligned} \tag{25}$$

Here, we combine the uncertainties into a diagonal matrix as

$$\mathbf{W} = \text{diag}(w_1^1, w_1^2, \cdots, w_M^1, w_M^2) \ . \tag{26}$$

The full Jacobian $\mathbf{J}$ has the shape $[2M \times (6N + M)]$, $\Delta\mathbf{T}$ is $[6N \times 1]$ and $\Delta\mathbf{d}$ is $[M \times 1]$. To solve Eq. (25), we rewrite it as

$$\mathbf{J}^T\mathbf{W}\mathbf{J}\begin{bmatrix}\Delta\mathbf{T} \\ \Delta\mathbf{d}\end{bmatrix} = -\mathbf{J}^T\mathbf{W}\mathbf{r} \ . \tag{27}$$

We now arrange the full Jacobian matrix $\mathbf{J}$ into two blocks as

$$\mathbf{J} = \begin{bmatrix}\mathbf{J_T} & \mathbf{J_d}\end{bmatrix} \ , \tag{28}$$

where $\mathbf{J_T} \in \mathbb{R}^{2M\times 6N}$ is the Jacobian block of $\mathbf{r}$ with respect to poses $\mathbf{T}$, and $\mathbf{J_d} \in \mathbb{R}^{2M\times M}$ is the Jacobian block of $\mathbf{r}$ with respect to disparities $\mathbf{d}$. For the detailed derivation of $\mathbf{J_T}$ and $\mathbf{J_d}$, we refer to section 2.3.1 in [12]. This yields

$$\begin{bmatrix}\mathbf{J_T}^T\mathbf{W}\mathbf{J_T} & \mathbf{J_T}^T\mathbf{W}\mathbf{J_d} \\ \mathbf{J_d}^T\mathbf{W}\mathbf{J_T} & \mathbf{J_d}^T\mathbf{W}\mathbf{J_d}\end{bmatrix}\begin{bmatrix}\Delta\mathbf{T} \\ \Delta\mathbf{d}\end{bmatrix} = -\mathbf{J}^T\mathbf{W}\mathbf{r} \ . \tag{29}$$

We rewrite Eq. (29) to the following form,

$$\begin{bmatrix}\mathbf{B} & \mathbf{E} \\ \mathbf{E}^T & \mathbf{C}\end{bmatrix}\begin{bmatrix}\Delta\mathbf{T} \\ \Delta\mathbf{d}\end{bmatrix} = \begin{bmatrix}\mathbf{v} \\ \mathbf{w}\end{bmatrix} \ , \tag{30}$$

and use the Schur Complement to solve for $\Delta\mathbf{T}$ and $\Delta\mathbf{d}$,

$$\begin{aligned} \Delta\mathbf{T} &= \left[\mathbf{B} - \mathbf{E}\mathbf{C}^{-1}\mathbf{E}^T\right]^{-1}(\mathbf{v} - \mathbf{E}\mathbf{C}^{-1}\mathbf{w}) \\ \Delta\mathbf{d} &= \mathbf{C}^{-1}(\mathbf{w} - \mathbf{E}^T\Delta\mathbf{T}) \ . \end{aligned} \tag{31}$$

Note that $\mathbf{C}$ here is diagonal since each d_i^k is only involved in r_k. Though $\mathbf{J_d}$ is not perfectly diagonal (it has two non-zero values along each column), $\mathbf{J_d}^T\mathbf{W}\mathbf{J_d} = \mathbf{C} \in \mathbb{R}^{M\times M}$ is diagonal. Therefore, the inverse of $\mathbf{C}$ is easy to compute. Also, the size of $\mathbf{B} \in \mathbb{R}^{6N\times 6N}$ is relatively small and therefore it is tractable to compute $\left[\mathbf{B} - \mathbf{E}\mathbf{C}^{-1}\mathbf{E}^T\right]^{-1}$. This is achieved by Cholesky decomposition. Finally we can use $\Delta\mathbf{T}$ and $\Delta\mathbf{d}$ to update the poses and disparities accordingly.

Scale, Shift and Disparity Optimization. We approach the problem of joint scale, shift and disparity optimization in a similar manner as above. First, we rewrite the objective function in Eq. (16) as,

$$\begin{aligned} f(\mathbf{s}, \mathbf{d}_h) = &\frac{1}{2}\sum_{\substack{(x_i^k, x_j^k)\in\mathcal{M}_h \\ d_i^k\in\mathbf{d}_h}}\Big(\left\|r_k(T_i, T_j, x_i^k, x_j^k, d_i^k)\right\|_{w_k}^2 \\ &+ \alpha_1\left\|t_k(d_i^k, s_i)\right\|^2\Big) + \frac{1}{2}\sum_{d_i^k\in\mathbf{d}_l}\alpha_2\left\|t_k(d_i^k, s_i)\right\|^2 \end{aligned} \tag{32}$$

where $s_i = (\theta_i, \gamma_i)$ is the scale and shift of frame i. We denote the set

$$\mathcal{M}_h = \{(x_i^k, x_j^k)\}_{k=1}^H \tag{33}$$

as the set of pixels where d_i^k is deemed to have a high error as defined by the multi-view filter in Eq. (6). The shape of $\mathbf{d}_h$ is $[1 \times H]$ $(H \leq M)$ and $\mathbf{d}_l$ denotes the set of disparities with a low error. Adding the cardinalities of $\mathbf{d}_h$ and $\mathbf{d}_l$ yields M *i.e.* $|\mathbf{d}_h| + |\mathbf{d}_l| = H + |\mathbf{d}_l| = M$. We denote the set of scales and shifts for all frames involves as $\mathbf{s} = (s_1, \ldots, s_N)$. $t_k(\cdot) \in \mathbb{R}$ is the residual term for the regularization by the monocular depth prior,

$$t_k(d_i^k, s_i) = d_i^k - (\theta_i \cdot d_{\text{mono,i}}^k + \gamma_i) \ . \tag{34}$$

Now, we collect all the residuals as a column vector

$$\hat{\mathbf{r}} = (\ldots, r_k^1, r_k^2, \ldots, r_H^1, r_H^2, t_1, \ldots, t_M)^T \ , \tag{35}$$

i.e. a collection of all r_k where k needs to ensure d_i^k is invalid (high error) and also all t. To define the corresponding linear system as in Eq. (29), we begin by defining the full Jacobian matrix $\widehat{\mathbf{J}}$ as a collection of two blocks, similar to Eq. (28) *i.e.* $\widehat{\mathbf{J}} = \begin{bmatrix}\widehat{\mathbf{J}}_\mathbf{s} & \widehat{\mathbf{J}}_\mathbf{d}\end{bmatrix}$, where $\widehat{\mathbf{J}}_\mathbf{s} \in \mathbb{R}^{(2H+M)\times 2N}$ is the Jacobian block of $\hat{\mathbf{r}}$ with respect to the scales and shifts $\mathbf{s}$, and $\hat{\mathbf{J}}_\mathbf{d} \in \mathbb{R}^{(2H+M)\times H}$ is the Jacobian block of $\hat{\mathbf{r}}$ with respect to the invalid disparities $\mathbf{d}_h$. Here both $\hat{\mathbf{J}}_\mathbf{s}$ and $\hat{\mathbf{J}}_\mathbf{d}$ consist of two parts. The first part comes from $r_k(\cdot)$ and second part from $t_k(\cdot)$ *i.e.*,

$$\hat{\mathbf{J}}_\mathbf{s} = \begin{bmatrix}\widehat{\mathbf{J}}_\mathbf{s}^r \\ \widehat{\mathbf{J}}_\mathbf{s}^t\end{bmatrix} \qquad \hat{\mathbf{J}}_\mathbf{d} = \begin{bmatrix}\widehat{\mathbf{J}}_\mathbf{d}^r \\ \widehat{\mathbf{J}}_\mathbf{d}^t\end{bmatrix} \qquad \hat{\mathbf{J}} = \begin{bmatrix}\widehat{\mathbf{J}}_\mathbf{s}^r & \widehat{\mathbf{J}}_\mathbf{d}^r \\ \widehat{\mathbf{J}}_\mathbf{s}^t & \widehat{\mathbf{J}}_\mathbf{d}^t\end{bmatrix} \ . \tag{36}$$

$\widehat{\mathbf{J}}_\mathbf{d}^r$ is the same as $\mathbf{J_d}$ in Eq. (28) except that now it only contains the invalid disparities $\mathbf{d}_h$ instead of $\mathbf{d}$, *i.e.* the shape of $\widehat{\mathbf{J}}_\mathbf{d}^r$ is $[2H \times H]$ instead of $[2M \times M]$. Note that we need to use a factor 2 here because each residual r_k has 2 dimensions. $\widehat{\mathbf{J}}_\mathbf{s}^r \in \mathbb{R}^{2H\times 2N}$ is 0 since none of the s_i are involved in any

of the $r_k(\cdot)$ residuals. The derivatives of $t_k(\cdot)$ with respect to θ_i, γ_i and d_i^k are,

$$\frac{\partial t_k}{\partial \theta_i} = -d^k_{\text{mono},i} \quad \frac{\partial t_k}{\partial \gamma_i} = -1 \quad \frac{\partial t_k}{\partial d_i^k} = 1 \ . \tag{37}$$

Thus the form of $\widehat{\mathbf{J}}_\mathbf{s}^t \in \mathbb{R}^{M \times 2N}$ is as follows,

$$\widehat{\mathbf{J}}_\mathbf{s}^t = \begin{bmatrix} J_{1,1} & \dots & J_{1,N} \\ \vdots & \ddots & \vdots \\ J_{M,1} & \dots & J_{M,N} \end{bmatrix} \quad J_{k,i} = \begin{bmatrix} -d^k_{\text{mono},i} & -1 \end{bmatrix} \ , \tag{38}$$

Finally, we construct $\widehat{\mathbf{J}}_\mathbf{d}^t$ as follows. First define the diagonal square matrix $\mathbf{D} = \text{diag}(1, \dots, 1) \in \mathbb{R}^{M \times M}$. Then we define $\widehat{\mathbf{J}}_\mathbf{d}^t$ as,

$$\widehat{\mathbf{J}}_\mathbf{d}^t = [\cdots D_k \cdots] \in \mathbb{R}^{M \times H} \tag{39}$$

where D_k is the k_{th} column of $\mathbf{D}$, and k needs to ensure that $d_i^k \in \mathbf{d}_h$. Next, we define the weighting matrix $\widehat{\mathbf{W}}$ as

$$\widehat{\mathbf{W}} = \text{diag}(\cdots, w_k^1, w_k^2, \cdots, w_H^1, w_H^2, \sigma_1, \cdots, \sigma_M) \tag{40}$$

where k needs to ensure d_i^k is invalid (high error), and σ equals to α_1 or α_2, depending on the corresponding disparity is invalid or valid (*i.e.* same α_1 and α_2 as used in Eq. (32)). Then, similar to Eq. (29), we use the Gauss-Newton method to form a linear equation,

$$\begin{bmatrix} \widehat{\mathbf{J}}_\mathbf{s}^T \widehat{\mathbf{W}} \widehat{\mathbf{J}}_\mathbf{s} & \widehat{\mathbf{J}}_\mathbf{s}^T \widehat{\mathbf{W}} \widehat{\mathbf{J}}_\mathbf{d} \\ \widehat{\mathbf{J}}_\mathbf{d}^T \widehat{\mathbf{W}} \widehat{\mathbf{J}}_\mathbf{s} & \widehat{\mathbf{J}}_\mathbf{d}^T \widehat{\mathbf{W}} \widehat{\mathbf{J}}_\mathbf{d} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{s} \\ \Delta \mathbf{d} \end{bmatrix} = -\widehat{\mathbf{J}}^T \widehat{\mathbf{W}} \hat{\mathbf{r}} \ , \tag{41}$$

and again use the Schur Complement Eq. (30) to solve for $\Delta\mathbf{s}$ and $\Delta\mathbf{d}$. Note that the lower right corner block $\widehat{\mathbf{J}}_\mathbf{d}^T \widehat{\mathbf{W}} \widehat{\mathbf{J}}_\mathbf{d}$ is still diagonal. Therefore, it is easy to compute its inverse.

3. More Experiments

To accompany the evaluations provided in the main paper, we provide further experiments in this section.

Implementation Details. As the point cloud downsampling factor, we use $\theta = 32$ for all frames but the first frame where $\theta = 16$ is used. We use $\beta = 2000$, the number of iterations for the final refinement optimization, on the Replica dataset and $\beta = 26000$ on the TUM-RGBD [10] and ScanNet [1] datasets (same as MonoGS [6]). We benchmark the runtime on an AMD Ryzen Threadripper Pro 3945WX 12-Cores with an NVIDIA GeForce RTX 3090 Ti with 24 GB of memory. For the remaining hyperparameters, we refer to MonoGS [6] for the Gaussian mapping.

A Note on Rendering and Runtime with MonoGS. By default, MonoGS [6] does not evaluate the rendering error on the mapped keyframes nor implement the exposure compensation during rendering evaluation. To compare our results fairly to MonoGS, we implement these details and run the experiments with these settings enabled. Further, we report the runtime for MonoGS using a single process (same as us) compared to the reported number in the paper, which was using multiple processes at once.

A Note on Gaussian Deformation with Photo-SLAM. Though not fully clear from reading the paper, after discussing with the authors of Photo-SLAM [3], we find that they do, in fact, not deform the Gaussians as a result of global BA or loop closure. They found this to be unstable in their experiments. This suggests that our deformation strategy is non-trivial.

Justification of Monocular Depth Estimator. There are already numerous monocular depth estimators, but most of them are limited by speed, memory or quality. We use Omnidata [2] since empirically we found it still provides the best trade-off between output performance and runtime. We also tested our system with Depth Anything [14], but found that it was marginally worse in terms of the final reconstructed mesh accuracy.

3.1. Full Evaluations Data

In Tab. 10, Tab. 11 and Tab. 12, we provide the full per scene results on all commonly reported metrics on Replica [9], TUM-RGBD [10] and ScanNet [1].

The reconstruction results are only measured on Replica since the other two datasets are real world datasets which lack quality ground truth meshes.

We show the trajectory accuracy measurement of both keyframes and the full trajectory, which is obtained by first linear interpolation between keyframes and using optical flow to refine. The accuracy of these two trajectories are similar. In the main paper, the data we report is always measured on the full trajectory.

3.2. Influence of Monocular Depth

While we show that the monocular depth improves the geometric estimation capability of our framework, it may still be erroneous. To better understand the accuracy of the monocular depth, we replace it with the ground truth sensor depth instead. This experiment acts as the upper bound of our method if the monocular depth is perfect. The experiments are done on Replica [9] and are shown in Tab. 13. Compared with the standard setting with the monocular depth, the ground truth depth setting gives improvements on both reconstruction and rendering quality, which reveals that our method still has potential to achieve better mapping results once better monocular depth is available. Since our method does not require further training or fine-tuning for the monocular

		Metric	R-0	R-1	R-2	O-0	O-1	O-2	O-3	O-4	Avg.
Reconstruction		Render Depth L1 ↓	2.90	2.16	2.18	2.44	1.97	2.46	2.62	2.53	**2.41**
		Accuracy ↓	1.99	1.91	2.06	3.96	2.03	3.45	2.15	1.89	**2.43**
		Completion ↓	3.78	3.38	3.34	2.75	3.33	4.36	3.96	4.25	**3.64**
		Comp. Rat. ↑	85.47	86.88	86.12	87.32	85.17	81.37	82.25	82.95	**84.69**
Rendering	Keyframes	PSNR ↑	32.25	34.31	35.95	40.81	40.64	35.19	35.03	37.40	**36.45**
		SSIM ↑	0.91	0.93	0.95	0.98	0.97	0.96	0.95	0.98	**0.95**
		LPIPS ↓	0.10	0.09	0.06	0.05	0.05	0.07	0.06	0.04	**0.06**
Tracking	Keyframes Trajectory	ATE RMSE ↓	0.29	0.38	0.24	0.27	0.35	0.34	0.42	0.43	**0.34**
	Full Trajectory	ATE RMSE ↓	0.29	0.33	0.25	0.29	0.35	0.34	0.42	0.43	**0.34**
Number of Gaussians		1000x	116	116	91	76	66	134	114	106	**102**

Table 10. **Full Evaluation on Replica [9].** We show the ATE RMSE [cm] evaluation on the keyframes as well as on the full trajectory.

		Metric	f1/desk	f1/desk2	f1/room	f2/xyz	f3/office	Avg.
Rendering	Keyframes	PSNR ↑	25.61	23.98	24.06	29.53	26.05	**25.85**
		SSIM ↑	0.84	0.81	0.80	0.90	0.84	**0.84**
		LPIPS ↓	0.18	0.23	0.24	0.08	0.20	**0.19**
Depth Rendering	Keyframes	Depth L1↓ [cm]	8.05	15.70	15.05	14.53	25.59	**15.78**
Tracking	Key Frames Trajectory	ATE RMSE ↓	1.92	3.05	4.43	0.23	1.41	**2.21**
	Full Trajectory	ATE RMSE ↓	1.65	2.79	4.16	0.22	1.44	**2.05**
Number of Gaussians		1000x	88	78	211	173	114	**133**

Table 11. **Full Evaluation on TUM-RGBD [10].**

		Metric	0000	0054	0059	0106	0169	0181	0207	0233	Avg.
Rendering	Keyframes	PSNR↑	28.68	30.21	27.69	27.70	31.14	31.15	30.49	27.48	**29.32**
		SSIM ↑	0.83	0.85	0.87	0.86	0.87	0.84	0.84	0.78	**0.84**
		LPIPS ↓	0.19	0.22	0.15	0.18	0.15	0.23	0.19	0.22	**0.19**
Depth Rendering	Keyframes	Depth L1↓ [cm]	8.24	18.24	13.39	23.5	11.49	18.35	13.78	10.19	**11.37**
Tracking	Key Frames Trajectory	ATE RMSE ↓	5.66	9.17	9.48	7.03	8.72	8.42	7.47	4.97	**7.61**
	Full Trajectory	ATE RMSE ↓	5.57	9.50	9.11	7.09	8.26	8.39	7.53	5.17	**7.58**
Number of Gaussians		1000x	144	157	84	108	52	127	121	191	**123**

Table 12. **Full Evaluation on ScanNet [1].**

depth, it is quite easy to just replace the current off-the-shelf monocular depth estimator with a better one.

3.3. Impact of Deformation

During runtime, we deform the 3D Gaussian map to account for adjustments to poses and depth that have already been integrated into the existing map. An alternative to performing the deformation is to solely rely on optimization to resolve the new map. We conduct two experiments to show the benefit of performing the deformation, especially when it comes to rendering accuracy. In Tab. 15, we vary the number of final refinement iterations and evaluate the rendering depth L1 and PSNR on the Replica `office 0` scene. We find that utilizing online 3D Gaussian deformations yields better rendering and depth L1 accuracy regardless of the number of iterations. In Tab. 14 we conduct the same experiment, but over a set of scenes on ScanNet. We find that on average, by enabling the deformation, we achieve higher rendering

	Metric	R-0	R-1	R-2	O-0	O-1	O-2	O-3	O-4	Avg.
Reconstruction	Render Depth L1 ↓	2.38	1.31	1.73	1.15	1.60	1.29	5.71	1.93	**2.14**
	Accuracy ↓	1.29	0.91	1.05	1.22	0.83	0.96	1.24	1.07	**1.07**
	Completion ↓	3.43	2.83	2.66	1.50	2.46	3.57	3.46	3.61	**2.94**
	Comp. Rat. ↑	86.61	88.69	88.70	93.44	89.09	85.20	84.60	85.32	**87.71**
Rendering	PSNR ↑	35.66	37.65	38.87	43.95	43.28	37.93	37.41	39.88	**39.33**
	SSIM ↑	0.96	0.96	0.97	0.99	0.98	0.96	0.96	0.98	**0.97**
	LPIPS ↓	0.04	0.05	0.03	0.02	0.02	0.06	0.04	0.03	**0.04**
Tracking	ATE RMSE ↓	0.29	0.38	0.24	0.28	0.39	0.35	0.45	0.40	**0.35**

Table 13. **Full Evaluations on Replica [9] with ground truth depth.** Both reconstruction and rendering results improve significantly with the ground truth depth, suggesting that our method is bounded by the quality of current day monocular depth estimation. Since we do not require any extra training or fine-tuning of the monocular depth estimator, it is easy to plug in a better estimator once available. Tracking performance does not change much.

accuracy and lower depth L1 error. The improvement is, however, more significant when it comes to the rendering accuracy.

3.4. Final Refinement

After the final global BA step, we perform a final refinement, similar to MonoGS [6], but include the geometric depth loss as well and do not only refine with a color loss. We ablate the influence on the results by varying the number of iterations of the final refinement in Tab. 16. We find that the rendering accuracy increases monotonically with the number of iterations while the geometric accuracy decreases with more than 2K iterations. We believe this to be a result of fitting to the noisy monocular depth. We choose to use 2K iterations since this provides the best trade-off between rendering and geometric accuracy. 2K iterations takes around 15 seconds on our benchmark hardware which consists of an AMD Ryzen Threadripper Pro 3945WX 12-Cores with an NVIDIA GeForce RTX 3090 Ti with 24 GB of memory.

Next, we investigate the benefit of the final refinement (and final global BA) in Tab. 17. The table shows the geometric accuracy as the depth L1 rendering error and the common RGB rendering metrics on the keyframes. We show both before and after the final refinement + global BA. The final refinement constitutes to an improvement in performance. The only existing method that does not use final refinement is Photo-SLAM [3], to which our results are comparable.

3.5. Impact of Downsampling Factor

During mapping, the point cloud formed from unprojecting the depth input is downsampled to avoid adding redundant Gaussians to the scene representation. We investigate the impact of using stronger versus weaker downsampling in Tab. 18 where we also compare to the sensitivity of MonoGS[6] with respect to the same parameter. Table 18 shows that both systems are not very sensitive to the model compression as a result of a larger downsampling factor θ. When both systems use the same number of Gaussians on average ($\theta = 32$ for MonoGS and $\theta = 64$ for our method), we find that our method performs significantly better in terms of depth rerendering and photometric accuracy. For all results in the main paper, we use $\theta = 32$.

3.6. Impact of Scene Representation

In our pipeline, we use 3D Gaussian Splatting as map representation. We also investigate the impact of using another map representation, namely the neural point cloud as in Point-SLAM [7]. To compare the effect of the scene representation, we keep the tracking part fixed, and replace 3DGS with neural points. The map deformation is also modified to fit the neural point cloud setting: before each mapping step, we re-anchor the neural points on the surface of scene by the updated camera poses and updated proxy depth map. The experiment results are shown in Tab. 20, the results of neural pointcloud representation is denoted as "*ours w/ npc*". The 3DGS representation demonstrates superior results in both reconstruction and rendering metrics.

3.7. Runtime Evaluation

To be consistent with the keyframe selection hyperparameters of MonoGS [6], we report on the same parameters as MonoGS uses by default. In practice, this means that few keyframes from the tracking system (determined via mean optical flow thresholding) are actually filtered out and not mapped. In Tab. 19, we show that by altering the hyperparamters, we can speed up the system during runtime, while still rendering and reconstructing the scene well. Note that we evaluate the rendering performance on the same set of views for all runs. We benchmark the runtime on an AMD Ryzen Threadripper Pro 3945WX 12-Cores with an NVIDIA GeForce RTX 3090 Ti with 24 GB of memory. We note that we currently do not leverage multiprocessing to the amount possible in practice *i.e.* currently we first do tracking and then mapping *i.e.* there is no simultaneous tracking and mapping. This is, however, straightforward to include, which should further speed up the runtime.

In the main paper, only the total runtime (including tracking and mapping) is reported. We also divide the runtime for

		Metric	0000	0054	0059	0106	0169	0181	0207	Avg.
Rendering	W/O Deform	PSNR↑	25.15	28.39	**27.77**	25.25	29.41	30.38	29.30	27.95
	W Deform	PSNR↑	**28.68**	**30.21**	27.69	**27.70**	**31.14**	**31.15**	**30.49**	**29.58**
Depth Rendering	W/O Deform	L1↓ [cm]	**7.86**	22.81	**10.51**	24.19	11.54	18.48	**13.66**	15.58
	W Deform		8.24	**18.24**	13.39	**23.5**	**11.49**	**18.35**	13.78	**15.28**

Table 14. **Gaussian Deformation Ablation on ScanNet [1].**

Nbr of Final Iterations β			0K	0.5K	1K	2K
Reconstruction	W/O Deform	Render Depth L1 ↓	8.84	3.49	2.64	2.6
	W Deform	Render Depth L1 ↓	**6.55**	**2.37**	**2.34**	**2.40**
Rendering	W/O Deform	PSNR ↑	22.86	34.30	37.66	37.86
	W Deform	PSNR ↑	**30.50**	**39.87**	**40.59**	**41.20**

Table 15. **Gaussian Deformation Ablation on Replica [9]** `office 0`.

Nbr of Final Iterations β		2K	5K	10K	26K
Reconstruction	Render Depth L1 ↓	**2.36**	2.45	2.51	2.59
	Accuracy ↓	**2.46**	2.66	2.84	3.02
	Completion ↓	3.60	**3.61**	3.59	3.60
	Comp. Rat. ↑	**84.87**	84.71	84.80	84.77
Rendering	Keyframes PSNR ↑	36.77	37.80	38.41	**38.95**

Table 16. **Final Refinement Iterations Ablation on Replica [9].** The results are averaged over the 8 scenes.

Metric	R-0	R-1	R-2	O-0	O-1	O-2	O-3	O-4	Avg.
Before Refinement									
Render Depth L1 ↓	5.90	5.01	3.82	4.53	2.89	4.43	3.91	3.46	**4.24**
PSNR ↑	27.55	29.47	30.77	35.75	37.58	31.67	31.90	33.25	**32.24**
SSIM ↑	0.83	0.86	0.90	0.94	0.96	0.91	0.93	0.94	**0.91**
LPIPS ↓	0.18	0.18	0.13	0.10	0.08	0.14	0.09	0.10	**0.13**
After Refinement									
Render Depth L1 ↓	2.90	2.16	2.18	2.44	1.97	2.46	2.62	2.53	**2.41**
PSNR ↑	32.25	34.31	35.95	40.81	40.64	35.19	35.03	37.40	**36.45**
SSIM ↑	0.91	0.93	0.95	0.98	0.97	0.96	0.95	0.98	**0.95**
LPIPS ↓	0.10	0.09	0.06	0.05	0.05	0.07	0.06	0.04	**0.06**

Table 17. **Evaluation on Replica [8].** Comparison before and after final global BA + refinement. We show the depth L1 metric in [cm] on the rendered depth maps along the recorded trajectory and the re-rendering metrics for the same frames.

Downsampling Factor θ			16	32	64
Reconstruction	Ours	Render Depth L1 ↓	2.38	2.40	2.46
	MonoGS [6]		33.43	28.47	28.09
Rendering	Ours	PSNR ↑	36.63	36.45	36.31
	MonoGS [6]		31.17	30.87	29.64
Number of Gaussians	Ours	1000x↓	141	102	83
	MonoGS [6]		97	83	73

Table 18. **Downsampling Factor θ Ablation on Replica [9].** The results are averaged over the 8 scenes.

tracking and mapping separately in Tab. 21.

k_{fcov}, k_{fm}		0.95, 0.04	0.90, 0.08	0.85, 0.08	0.80, 0.12	0.70, 0.16	0.60, 0.20	0.50, 0.30
Reconstruction	Render Depth L1 ↓	**2.90**	2.94	2.97	3.08	3.37	3.53	4.78
	Accuracy ↓	1.99	**1.94**	2.06	2.04	2.54	3.20	6.20
	Completion ↓	3.78	**3.76**	3.79	3.77	3.86	3.93	5.23
	Comp. Rat. ↑	85.47	**85.58**	85.39	85.53	85.03	84.33	80.38
Rendering	PSNR ↑	**32.25**	31.65	31.31	30.59	30.12	29.25	27.59
Runtime	FPS ↑	1.24	1.45	1.62	2.02	2.50	3.03	**3.67**

Table 19. **Keyframe Hyperparameter Search on Replica [9] `room 0`.** By changing the keyframe selection hyperparameters, we can speed up our runtime without impacting reconstruction and rendering too much. We evaluate the rendering performance on the same set of frames for all runs. In comparison, with the default $k_{\text{fcov}} = 0.95$, $k_{\text{fm}} = 0.04$, MonoGS [6] yields PSNR: 26.12 and render depth L1: 17.38 cm.

Dataset	Metric	Ours w/ npc	Ours
Replica	Accuracy ↓	2.96	**2.43**
	Completion ↓	3.95	**3.64**
	Comp. Rat. ↑	83.72	**84.69**
	PSNR↑	31.04	**36.45**
	SSIM ↑	0.91	**0.95**
	LPIPS ↓	0.12	**0.06**
ScanNet	PSNR↑	22.45	**29.48**
	SSIM ↑	**0.85**	**0.85**
	LPIPS ↓	0.30	**0.18**
TUM-RGBD	PSNR↑	20.99	**25.85**
	SSIM ↑	0.77	**0.84**
	LPIPS ↓	0.30	**0.19**

Table 20. **Scene Representation Ablation on Replica [9], ScanNet [1] and TUM-RGBD [10].** Both rendering and reconstruction metrics improve with the 3DGS (Ours) representation over the neural point cloud representation found in [7] (Ours w/ npc). The results are averaged for each dataset across the test scenes.

	Total Tracking FPS	Total Mapping FPS	Total FPS
Avg. FPS	4.66	1.65	1.24

Table 21. **Runtime Evaluation on Replica [9]** `room0`. Separated into tracking and mapping as well.

References

[1] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE/CVF, 2017. 4, 5, 7, 8

[2] Ainaz Eftekhar, Alexander Sax, Jitendra Malik, and Amir Zamir. Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10786–10796, 2021. 4

[3] Huajian Huang, Longwei Li, Hui Cheng, and Sai-Kit Yeung. Photo-slam: Real-time simultaneous localization and photo-realistic mapping for monocular, stereo, and rgb-d cameras. *arXiv preprint arXiv:2311.16728*, 2023. 1, 4, 6

[4] Nikhil Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon Luiten. Splatam: Splat, track and map 3d gaussians for dense rgb-d slam. *arXiv preprint*, 2023. 1

[5] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023. 1

[6] Hidenobu Matsuki, Riku Murai, Paul HJ Kelly, and Andrew J Davison. Gaussian splatting slam. *arXiv preprint arXiv:2312.06741*, 2023. 1, 4, 6, 7, 8

[7] Erik Sandström, Yue Li, Luc Van Gool, and Martin R Oswald. Point-slam: Dense neural point cloud-based slam. In *International Conference on Computer Vision (ICCV)*. IEEE/CVF, 2023. 6, 8

[8] Julian Straub and etal. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 7

[9] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 4, 5, 6, 7, 8

[10] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ, 2012. 4, 5, 8

[11] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in neural information processing systems*, 34:16558–16569, 2021. 2

[12] Zachary Teed et al. *Optimization Inspired Neural Networks for Multiview 3D Reconstruction*. PhD thesis, Princeton University, 2022. 2, 3

[13] Chi Yan, Delin Qu, Dong Wang, Dan Xu, Zhigang Wang, Bin Zhao, and Xuelong Li. Gs-slam: Dense visual slam with 3d gaussian splatting. *arXiv preprint arXiv:2311.11700*, 2023. 1

[14] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. *arXiv preprint arXiv:2401.10891*, 2024. 4

[15] Vladimir Yugay, Yue Li, Theo Gevers, and Martin R. Oswald. Gaussian-slam: Photo-realistic dense slam with gaussian splatting, 2023. 1