SuGaR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering

Supplementary Material

In this supplementary material, we provide the following elements:

- Details about our regularization methods
- Details about the parameterisation of the bound gaussians optimized during our joint refinement strategy.
- Additional implementation details.
- Quantitative results for 3D surface reconstruction
- Detailed quantitative results for real-time rendering of real scenes, and mesh rendering ablation.

We also provide a video that offers an overview of the approach and showcases additional qualitative results. Specifically, the video demonstrates how SuGaR meshes can be used to animate Gaussian Splatting representations.

7. About our Regularization

As we explain in the main paper, it is possible to regularize Gaussians with a simple term $\mathcal{R}_{density}$ that relies on the density function rather than the SDF:

$$\mathcal{R}_{density} = \frac{1}{|\mathcal{P}|} \sum_{p} |d(p) - \bar{d}(p)| \tag{11}$$

This regularization term works well in practice, especially for foreground objects, and allows for extracting good-looking meshes. However, this regularization is not strong enough for regularizing background accurately, and produces chaotic surfaces in the background, similarly to vanilla Gaussian Splatting.

The SDF regularization allows for introducing a strong depth regularization in the computation, as it uses distances defined in the 3D space rather than density values. Moreover, using Gaussian splats facing the camera for computing depth maps for the SDF estimator encourages visible Gaussians to face cameras on average, which provides a useful prior to regularize background compared to the simpler density loss.

8. Parameterisation of Gaussians bound to the surface

As we explained in Section 4, once we have extracted the mesh from the Gaussian Splatting representation, we refine this mesh by binding new Gaussians to the mesh triangles and optimize the Gaussians and the mesh jointly using the Gaussian Splatting rasterizer. To keep the Gaussians flat and aligned with the mesh triangles, we explicitly compute the means of the Gaussians from the mesh vertices using predefined barycentric coordinates in the corresponding triangles during optimization. Also, the Gaussians have only 2 learnable scaling factors instead of 3 and only 1 learnable 2D rotation. Indeed, we do not optimize a full quaternion that would encode a 3D rotation, as performed in [15]; Instead, we optimize a 2D rotation in the plane of the triangle. Therefore, the Gaussians stay aligned with the mesh triangles, but are allowed to rotate on the local surface. Like the original model, we also optimize an opacity value and a set of spherical harmonics for every Gaussian to encode the color emitted in all directions.

In practice, for each Gaussian, we optimize a learnable complex number $x + iy$ rather than a quaternion, encoding the 2D rotation inside the triangle's plane. During optimization, we still need to compute an explicit 3D quaternion encoding the 3D rotation of the Gaussians in the *world space* to apply the rasterizer. To recover the full 3D quaternion, we proceed as follows: For any 3D Gaussian g, we first compute the matrix $R = [R^{(0)}, R^{(1)}, R^{(2)}] \in \mathbb{R}^{3\times3}$ encoding the rotation of its corresponding triangle: We select as the first column $R^{(0)}$ of the matrix the normal of the triangle, and as the second column $R^{(1)}$ a fixed edge of the triangle. We compute the third column $R^{(2)}$ with a cross-product. Then, we compute the matrix R_g encoding the full 3D rotation of the Gaussian by applying the learned 2D complex number to the rotation of the triangle, as follows: $R_g^{(0)} = R^{(0)}, R_g^{(1)} = x'R^{(1)} + y'R^{(2)}$ and $R_g^{(2)} = -y'R^{(1)} + x'R^{(2)}$, where $x' = \frac{x}{|x^2+y^2|}$ and $y' = \frac{y}{|x^2+y^2|}$.

Adjusting parameters for edition. Because our learned complex numbers represent rotations in the space of the corresponding triangles, our representation is robust to mesh edition or animation: When editing the underlying mesh at inference, there is no need to update the learned 2D rotations as they remain the same when rotating or moving triangles.

Conversely, when scaling or deforming a mesh, the triangle sizes might change, necessitating adjustments to the learned scaling factors of the bound surface Gaussians. For example, if the mesh size doubles, all Gaussian scaling factors should similarly be multiplied by 2. In our implementation, when editing the mesh, we modify in real-time the learned scaling factors of a bound surface Gaussian by multiplying them by the ratio between (a) the average length of the triangle's sides after modification and (b) the average length of the original triangle's sides.

	PSNR ↑	SSIM ↑	LPIPS ↓
Plenoxels [42]	21.07	0.719	0.379
INGP-Base [23]	21.72	0.723	0.330
INGP-Big [23]	21.92	0.744	0.304
Mip-NeRF360 [2]	22.22	0.758	0.257
3DGS [15]	**23.14**	**0.841**	**0.183**
R-SuGaR-2K (Ours)	19.70	0.743	0.284
R-SuGaR-7K (Ours)	21.09	0.786	0.233
R-SuGaR-15K (Ours)	21.58	0.795	0.219

Table 4. **Quantitative evaluation on Tanks&Temples [16].** SuGaR is not as good as as vanilla 3D Gaussian Splatting in terms of rendering quality as it relies on a mesh but higher than the other methods that do not recover a mesh.

	PSNR ↑	SSIM ↑	LPIPS ↓
Plenoxels [42]	23.06	0.794	0.510
INGP-Base [23]	23.62	0.796	0.423
INGP-Big [23]	24.96	0.817	0.390
Mip-NeRF360 [2]	29.40	0.901	0.244
3DGS [15]	**29.41**	**0.903**	**0.242**
R-SuGaR-2K (Ours)	27.31	0.873	0.303
R-SuGaR-7K (Ours)	29.30	0.893	0.273
R-SuGaR-15K (Ours)	**29.41**	0.893	0.267

Table 5. **Quantitative evaluation on DeepBlending [12].** SuGaR is not as good as as vanilla 3D Gaussian Splatting in terms of rendering quality as it relies on a mesh but higher than the other methods that do not recover a mesh.

9. Additional implementation details

Implementation We implemented our model with PyTorch [25] and use 3D data processing tools from PyTorch3D [27]. We also use the differentiable Gaussian Splatting rasterizer from the original 3D Gaussian Splatting paper [15]. We thank the authors for providing this amazing tool.

Mesh extraction. In practice, we apply two Poisson reconstructions for mesh extraction: one for foreground points, and one for background points. We define foreground points as points located inside the bounding box of all training camera poses, and background points as points located outside. We chose this simple distinction between foreground and background in order to design an approach as general as possible. However, depending on the content of the scene and the main objects to reconstruct, defining a custom bounding box for foreground points could improve the quality and precision of the extracted mesh.

Joint refinement. During joint refinement, we also compute a normal consistency term on the mesh's faces to further regularize the surface. This term doesn't affect performance in terms of PSNR, SSIM, or LPIPS. However, it does marginally enhance visual quality by promoting smoother surfaces.

10. Additional Results for Real-Time Rendering of Real Scenes

We compute the standard metrics PSNR, SSIM and LPIPS [44] to evaluate the quality of SuGaR's rendering using our extracted meshes and their bound surface Gaussians. Results on the Mip-NeRF360 dataset are given in Table 1 in the main paper. Results on Tanks&Temple and DeepBlending are given in Tables 4 and 5. Tables 7, 8 and 9 provide the detailed results for all scenes in the datasets.

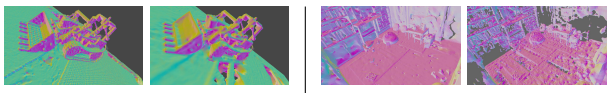
11. Quantitative comparison with NeuS [36]

Because the main motivation for SuGaR is to provide an editable and animatable 3DGS representation, we focused on rendering metrics in the main paper. However, evaluating the model with geometry metrics is also relevant to evaluate the accuracy of the surface reconstruction.

We compare SuGaR with an enhanced implementation of NeuS [36] leveraging hash encodings (Instant-NeuS) to greatly accelerate training. This method reaches an optimization time similar to SuGaR's. Table 6 provides a quantitative comparison in real scenes, and Figure 8 provides a qualitative comparison.

	Barn	Caterpillar	Ignatius	Meetingroom	Truck
Instant-NeuS	0.8894	0.2034	0.0930	2.7102	0.2119
SuGaR (Ours)	**0.2279**	**0.1611**	**0.0380**	**0.2394**	**0.0888**

Table 6. **Chamfer distance (↓) on scenes from Tanks&Temples.**



SuGaR (Ours) Instant-NeuS SuGaR (Ours) Instant-NeuS

Figure 8. **Qualitative comparison between SuGaR (< 2M triangles) and Instant-NeuS (> 4M triangles).** Even with less triangles, SuGaR produces much finer results as well as better background geometry.

Instant-NeuS retrieves much lower quality meshes than SuGaR on real scenes, as shown in Figure 8. Instant-NeuS produces good-looking meshes on synthetic datasets or the DTU dataset, which provides the foreground masks and almost perfect camera calibration. Instant-NeuS is however not able to achieve a high-quality reconstruction of real scenes with an unmasked background, even after tuning the hyperparameters. On the contrary, as the teaser image and the video show, SuGaR produces high-quality meshes with background geometry even for casual captures with a

(a) 2,000 iterations (b) 7,000 iterations (c) 15,000 iterations

Figure 9. **Refined SuGaR renderings with different numbers of refinement iterations.** 2,000 iterations are usually enough to obtain high quality rendering **(a)**, since the extracted mesh "textured" with surface Gaussians is already an excellent initialization for optimizing the model. However, further refinement helps the Gaussians to capture texturing details and reconstruct extremely thin geometry that is finer that the resolution of the mesh, such as the spokes of the bicycle, as seen in **(b)**, **(c)**.

(a) SuGaR render (b) Mesh (No texture) (c) Mesh normals

Figure 10. **SuGaR renderings** with (**top:**) 200,000 and (**bottom:**) 1,000,000 vertices. Even with low-poly meshes, the 3D Gaussians bound to the mesh produce high quality renderings. Moreover, low-poly meshes help to better regularize the surface.

low-cost smartphone and without needing any precomputed mask. To the extent of our knowledge, no other hybrid representation method is able to achieve the same performance and versatility as SuGaR, in such a short training time.

12. Additional Results for Mesh Renderig Ablation

We provide additional qualitative results to illustrate how various parameters impact rendering performance.

First, we provide in Figure 9 a simple example showing how the Gaussians constrained to remain on the surface during refinement greatly increase the rendering quality as they play the role of an efficient texturing tool and help reconstructing very fine details missing in the extracted mesh.

Then, in Figure 10 we illustrate how the resolution of the mesh extraction, i.e., the number of triangles, modifies the rendering quality. For fair comparison, we increase the number of surface-aligned Gaussians per triangle when we decrease the number of triangles. Results show that increasing the number of vertices increases the quality of rendering with surface Gaussians, but meshes with lower triangles are already able to reach state of the art results.

Finally, Figure 11 illustrates the benefits of using Gaussians aligned on the surface as a texturing tool for rendering meshes. To this end, we also optimize traditional UV textures on our meshes using differentiable mesh rendering with traditional triangle rasterization. Even though rendering with surface-aligned Gaussians provides better performance, rendering our meshes with traditional UV textures still produces satisfying results, which further illustrates the quality of our extracted meshes.

Figure 11. **Qualitative comparison between (bottom:) a traditional UV texture optimized from training images, and (top:) the bound surface Gaussians.** Even though high resolution UV textures have good quality and can be rendered with our meshes using any traditional software, using 3D Gaussians bound to the surface of the mesh greatly improves the rendering quality. Meshes in these images have 200,000 vertices only.

	Mip-NeRF360 [2]							DeepBlending [12]		Tanks&Temples [16]	
	Garden	Kitchen	Room	Bicycle	Counter	Bonsai	Stump	Playroom	Dr. Johnson	Train	Truck
200K vertices											
R-SuGaR-2K	23.30	25.74	27.58	21.53	24.41	26.50	23.45	27.83	26.51	18.15	21.03
R-SuGaR-7K	24.99	28.78	29.47	22.69	26.86	29.33	24.45	30.02	28.41	19.82	22.31
R-SuGaR-15K	25.29	29.38	29.95	22.91	27.47	30.42	24.55	30.08	28.59	20.40	22.65
1M vertices											
R-SuGaR-2K	23.56	26.15	27.68	21.80	24.62	26.70	23.56	27.93	26.70	18.32	21.09
R-SuGaR-7K	25.06	28.96	29.57	22.86	26.92	29.47	24.55	**30.13**	28.47	19.85	22.34
R-SuGaR-15K	**25.36**	**29.56**	**30.03**	**23.14**	**27.62**	**30.51**	**24.70**	30.12	**28.71**	**20.50**	**22.67**

Table 7. **Quantitative evaluation of rendering quality in terms of PSNR on all scenes.** A higher PSNR indicates better rendering quality. We adjust the number of bound surface-aligned Gaussians per triangle when we reduce the number of vertices, aiming for a similar count across all models. Results show that increasing the number of vertices (*i.e.* increasing the resolution of the geometry) increases the quality of rendering with surface Gaussians, but meshes with less triangles are already able to reach state of the art results.

	Mip-NeRF360 [2]							DeepBlending [12]		Tanks&Temples [16]	
	Garden	Kitchen	Room	Bicycle	Counter	Bonsai	Stump	Playroom	Dr. Johnson	Train	Truck
200K vertices											
R-SuGaR-2K	0.713	0.859	0.881	0.572	0.844	0.895	0.641	0.883	0.864	0.694	0.787
R-SuGaR-7K	0.762	0.901	0.904	0.621	0.883	0.926	0.679	0.898	0.888	0.749	0.822
R-SuGaR-15K	0.771	0.907	**0.909**	0.631	0.890	**0.933**	0.681	0.897	0.888	0.763	**0.827**
1M vertices											
R-SuGaR-2K	0.719	0.866	0.882	0.583	0.846	0.894	0.642	0.883	0.863	0.698	0.788
R-SuGaR-7K	0.764	0.903	0.905	0.628	0.884	0.925	0.680	**0.899**	0.887	0.750	0.821
R-SuGaR-15K	**0.775**	**0.908**	**0.909**	**0.640**	**0.891**	0.932	**0.683**	0.898	**0.889**	**0.764**	**0.827**

Table 8. **Quantitative evaluation of rendering quality in terms of SSIM on all scenes.** A higher SSIM indicates better rendering quality. We adjust the number of bound surface-aligned Gaussians per triangle when we reduce the number of vertices, aiming for a similar count across all models. Results show that increasing the number of vertices (*i.e.* increasing the resolution of the geometry) increases the quality of rendering with surface Gaussians, but meshes with less triangles are already able to reach state of the art results.

	Mip-NeRF360 [2]							DeepBlending [12]		Tanks&Temples [16]	
	Garden	Kitchen	Room	Bicycle	Counter	Bonsai	Stump	Playroom	Dr. Johnson	Train	Truck
200K vertices											
R-SuGaR-2K	0.280	0.221	0.280	0.413	0.288	0.259	0.390	0.284	0.314	0.335	0.235
R-SuGaR-7K	0.232	0.175	0.252	0.363	0.245	0.228	0.345	0.260	0.277	0.274	0.187
R-SuGaR-15K	**0.218**	0.166	**0.243**	0.349	**0.234**	**0.219**	**0.336**	**0.257**	**0.268**	**0.258**	**0.174**
1M vertices											
R-SuGaR-2K	0.281	0.215	0.282	0.408	0.287	0.262	0.391	0.286	0.319	0.333	0.236
R-SuGaR-7K	0.233	0.173	0.253	0.360	0.245	0.231	0.347	0.265	0.282	0.275	0.190
R-SuGaR-15K	0.220	**0.165**	0.246	**0.345**	**0.234**	0.221	0.338	0.261	0.273	0.260	0.178

Table 9. **Quantitative evaluation of rendering quality in terms of LPIPS [44] on all scenes.** A lower LPIPS indicates better rendering quality. We adjust the number of bound surface-aligned Gaussians per triangle when we reduce the number of vertices, aiming for a similar count across all models. The results indicate that the stronger regularity due to a smaller number of vertices leads to smoother surfaces and higher LPIPS metrics when using the bound Gaussians.