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

Antoine Guédon Vincent Lepetit
LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, France

https://anttwo.github.io/sugar/

Figure 1. We introduce a method that extracts accurate and editable meshes from 3D Gaussian Splatting representations within minutes on a single GPU. The meshes can be edited, animated, composited, etc. with very realistic Gaussian Splatting rendering, offering new possibilities for Computer Graphics. Note for example that we changed the posture of the robot between the captured scene on the bottom left and the composited scene on the right. The supplementary material provides more examples, including a video illustrating our results.

Abstract

We propose a method to allow precise and extremely fast mesh extraction from 3D Gaussian Splatting [15]. Gaussian Splatting has recently become very popular as it yields realistic rendering while being significantly faster to train than NeRFs. It is however challenging to extract a mesh from the millions of tiny 3D Gaussians as these Gaussians tend to be unorganized after optimization and no method has been proposed so far. Our first key contribution is a regularization term that encourages the Gaussians to align well with the surface of the scene. We then introduce a method that exploits this alignment to extract a mesh from the Gaussians using Poisson reconstruction, which is fast, scalable, and preserves details, in contrast to the Marching Cubes algorithm usually applied to extract meshes from Neural SDFs. Finally, we introduce an optional refinement strategy that binds Gaussians to the surface of the mesh, and jointly optimizes these Gaussians and the mesh through Gaussian splatting rendering. This enables easy editing, sculpting, animating, and relighting of the Gaussians by manipulating the mesh instead of the Gaussians themselves. Retrieving such an editable mesh for realistic rendering is done within minutes with our method, compared to hours with the state-of-the-art method on SDFs, while providing a better rendering quality.

Erratum

We identified a minor typographical error in Subsection 4.1 in the earlier version of the paper.
In the computation of our regularization term $\mathcal{R}$ in Equation 8, we use $p \rightarrow \pm s_{g*}\sqrt{-2\log\left(d(p)\right)}$ instead of $p \rightarrow \pm s_{g*}\sqrt{-2\log\left(\bar{d}(p)\right)}$ as an 'ideal' distance function associated with the density d (Equation 7). As detailed in the paper, this distance function aligns with the true surface of the scene in an ideal scenario where $d = \bar{d}$. We have updated Equation 7 to clarify this matter.

1. Introduction

After NeRFs [22], 3D Gaussian Splatting [15] has recently become very popular for capturing a 3D scene and rendering it from novel points of view. 3D Gaussian Splatting optimizes the positions, orientations, appearances (represented as spherical harmonics), and alpha blending of many tiny 3D Gaussians on the basis of a set of training images of the scene to capture the scene geometry and appearance. Because rendering the Gaussians is much faster than rendering a neural field, 3D Gaussian Splatting is much faster than NeRFs and can capture a scene in a few minutes.

While the Gaussians allow very realistic renderings of the scene, it is still however challenging to extract the sur-

Figure 2. Our algorithm can extract a highly detailed mesh from any 3D Gaussian Splatting scene [15] within minutes on a single GPU (**top:** Renderings of our meshes without texture, **bottom:** Renderings of the meshes with bound Gaussians).

face of the scene from them: As shown in Figure 3, after optimization by 3D Gaussian Splatting, the Gaussians do not take an ordered structure in general and do not correspond well to the actual surface of the scene. In addition to the surface itself, it is also often desirable to represent the scene as a mesh, which remains the representation of choice in many pipelines: A mesh-based representation allows for powerful tools for editing, sculpting, animating, and relighting the scene. Because the Gaussians after Gaussian Splatting are unstructured, it is very challenging to extract a mesh from them. Note that this is also challenging with NeRFs albeit for different reasons.

In this paper, we first propose a regularization term that encourages the Gaussians to be well distributed over the scene surface so that the Gaussians capture much better the scene geometry, as shown in Figure 3. Our approach is to derive a volume density from the Gaussians under the assumption that the Gaussians are flat and well distributed over the scene surface. By minimizing the difference between this density and the actual one computed from the Gaussians during optimization, we encourage the 3D Gaussians to represent well the surface geometry.

Thanks to this regularization term, it becomes easier to extract a mesh from the Gaussians. In fact, since we introduce a density function to evaluate our regularization term, a natural approach would be to extract level sets of this density function. However, Gaussian Splatting performs densification in order to capture details of the scene with high fidelity, which results in a drastic increase in the number of Gaussians. Real scenes typically end up with one or several millions of 3D Gaussians with different scales and rotations, the majority of them being extremely small in order to reproduce texture and details in the scene. This results in a density function that is close to zero almost everywhere, and the Marching Cubes algorithm [21] fails to extract proper level sets of such a sparse density function even with a fine voxel grid, as also shown in Figure 3.

Instead, we introduce a method that very efficiently samples points on the visible part of a level set of the density function, allowing us to run the Poisson reconstruction algorithm [14] on these points to obtain a triangle mesh. This approach is scalable, by contrast with the Marching Cubes algorithm for example, and reconstructs a surface mesh within minutes on a single GPU, compared to other state of the art methods relying on Neural SDFs for extracting meshes from radiance fields, that require at least 24 hours on one GPU [20, 36, 38, 39] and rely on multiple GPUs to speed up the process [26].

As illustrated in Figures 2 and 4, our method produces high quality meshes. The challenge is in efficiently identifying points lying on the level set. To do this, we rely on the Gaussians depth maps seen from the training viewpoints. These depth maps can be obtained by extending the Gaussian Splatting rasterizer, and we show how to ac-

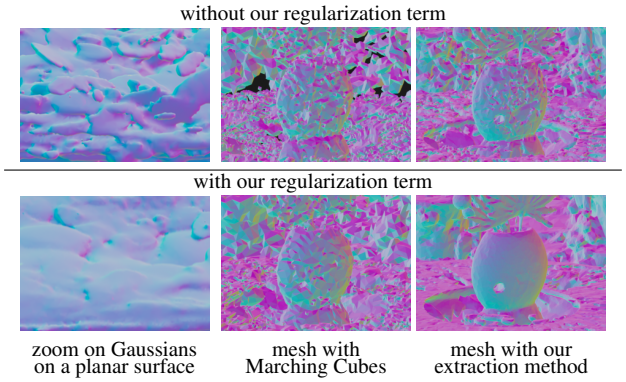


Figure 3. **Extracting a mesh from Gaussians.** Without regularization, the Gaussians have no special arrangement after optimization, which makes extracting a mesh very difficult. Without our regularization term, Marching Cubes fail to extract an acceptable mesh. With our regularization term, Marching Cubes recover an extremely noisy mesh even with a very fine 3D grid. Our scalable extraction method obtains a mesh even without our regularization term. Still, the mesh is noisy. By contrast, our full method succeeds in reconstructing an accurate mesh very efficiently.

curately sample points on the level set starting from these depth maps.

Finally, after extracting this mesh, we propose an optional refinement strategy that jointly optimizes the mesh and a set of 3D Gaussians through Gaussian splatting rendering only. This optimization enables high-quality rendering of the mesh using Gaussian splatting rendering rather than traditional textured mesh rendering. This results in higher performance in terms of rendering quality than other radiance field models relying on an underlying mesh at inference [6, 26, 39]. As shown in Figure 1, this makes possible the use of traditional mesh-editing tools for editing a Gaussian Splatting representation of a scene, offering endless possibilities for Computer Graphics.

To summarize, our contributions are:

- a regularization term that makes the Gaussians capture accurately the geometry of the scene;
- an efficient algorithm that extracts an accurate mesh from the Gaussians within minutes;
- a method to bind the Gaussians to the mesh, resulting in a more accurate mesh, higher rendering quality than state of the art methods using a mesh for Novel View Synthesis [6, 26, 39], and allowing editing the scene in many different ways.

We call our approach SuGaR. In the remainder of the paper, we discuss related work, give a brief overview of vanilla 3D Gaussian Splatting, describe SuGaR, and compare it to the state of the art.

2. Related Work

Image-based rendering (IBR) methods rely on a set of two-dimensional images of a scene to generate a representation of the scene and render novel views. The very first novel-view synthesis approaches were based on light fields [19], and developed the concept of volume rendering for novel views. Their work emphasized the importance of efficiently traversing volumetric data to produce realistic images.

Various scene representations have been proposed since, such as triangle meshes, point clouds, voxel grids, multi-plane images, or neural implicit functions.

Traditional mesh-based IBR methods. Structure-from-motion (SfM) [32] and subsequent multi-view stereo (MVS) [10] allow for 3D reconstruction of surfaces, leading to the development of several view synthesis algorithms relying on triangle meshes as the primary 3D representation of scenes. Such algorithms consider textured triangles or warp and blend captured images on the mesh surface to generate novel views [4, 12, 37]. [29, 30] consider deep learning-based mesh representations for better view synthesis, bridging the gap between traditional graphics and modern machine learning techniques. While these mesh-based methods take advantage of existing graphics hardware and software for efficient rendering, they struggle with the capture of accurate geometry and appearance in complex regions.

Volumetric IBR methods. Volumetric methods use voxel grids, multiplane images, or neural networks to represent scenes as continuous volumetric functions of density and color. Recently, Neural Radiance Fields (NeRF) [22] introduced a novel scene representation based on a continuous volumetric function parameterized by a multilayer perceptron (MLP). NeRF produces photorealistic renderings with fine details and view-dependent effects, achieved through volumetric ray tracing. However, the original NeRF is computationally expensive and memory intensive.

To address these challenges, several works have improved NeRF's performance and scalability. These methods leverage discretized or sparse volumetric representations like voxel grids and hash tables as ways to store learnable features acting as positional encodings for 3D points [5, 13, 23, 34, 41], hierarchical sampling strategies [2, 11, 28, 40], or low-rank approximations [5]. However, they still rely on volumetric ray marching, which is incompatible with standard graphics hardware and software designed for rendering polygonal surfaces. Recent works have proposed modifying the NeRF's representation of geometry and emitted radiance to allow for better reconstruction of specular materials [35] or relighting the scene through an explicit decomposition into material and lighting properties [3, 18, 33, 43].

Hybrid IBR methods. Some methods build on differentiable rendering to combine the advantages of mesh-based and volumetric methods, and allow for surface reconstruction as well as better editability. They use a hybrid volume-surface representation, which enables high-quality meshes suitable for downstream graphics applications while efficiently modeling view-dependent appearance. In particular, some works optimize neural signed distance functions (SDF) by training neural radiance fields in which the density is derived as a differentiable transformation of the SDF [7, 8, 20, 24, 36, 38]. A triangle mesh can finally be reconstructed from the SDF by applying the Marching Cubes algorithm [21]. However, most of these methods do not target real-time rendering.

Alternatively, other approaches "bake" the rendering capacity of an optimized NeRF or neural SDF into a much efficient structure relying on an underlying triangle mesh [6] that could benefit from the traditional triangle rasterization pipeline. In particular, the recent BakedSDF [39] reconstructs high quality meshes by optimizing a full neural SDF model, baking it into a high-resolution triangle mesh that combines mesh rendering for interpolating features and deep learning to translate these features into images, and finally optimizes a view-dependent appearance model.

However, even though it achieves real-time rendering and produces impressive meshes of the surface of the scene, this model demands training a full neural SDF with an architecture identical to Mip-NeRF360 [1], which necessitates 48 hours of training.

Similarly, the recent method NeRFMeshing [26] proposes to also bake any NeRF model into a mesh structure, achieving real-time rendering. However, the meshing performed in this method lowers the quality of the rendering and results in a PSNR much lower than our method. Additionally, this method still requires training a full NeRF model beforehand, and needs approximately an hour of training on 8 V100 NVIDIA GPUs to allow for mesh training and extraction.

Our method is much faster at retrieveing a 3D mesh from 3D Gaussian Splatting, which is itself much faster than NeRFs. As our experiments show, our rendering done by bounding Gaussians to the mesh results in higher quality than previous solutions based on meshes.

Point-based IBR methods. Alternatively, point-based representations for radiance field excel at modeling thin geometry and leverage fast point rasterization pipelines to render images using α-blending rather than ray-marching [17, 31]. In particular, the very recent 3D Gaussian Splatting model [15] allows for optimizing and rendering scenes with speed and quality never seen before.

3. 3D Gaussian Splatting

For the sake of completeness, we briefly describe the original 3D Gaussian Splatting method here. The scene is represented as a (large) set of Gaussians, where each Gaussian g is represented by its mean μ_g and its covariance Σ_g is parameterized by a scaling vector $s_g \in \mathbb{R}^3$ and a quaternion $q_g \in \mathbb{R}^4$ encoding the rotation of the Gaussian. In addition, each Gaussian is associated with its opacity $\alpha_g \in [0, 1]$ and a set of spherical harmonics coordinates describing the colors emitted by the Gaussian for all directions.

An image of a set of Gaussians can be rendered from a given viewpoint thanks to a rasterizer. This rasterizer *splats* the 3D Gaussians into 2D Gaussians parallel to the image plane for rendering, which results in an extremely fast rendering process. This is the key component that makes 3D Gaussian Splatting much faster than NeRFs, as it is much faster than the ray-marching compositing required in the optimization of NeRFs.

Given a set of images, the set of Gaussians is initialized from the point cloud produced by SfM [32]. The Gaussians' parameters (means, quaternions, scaling vectors, but also opacities and spherical harmonics parameters) are optimized to make the renderings of the Gaussians match the input images. During optimization, more Gaussians are added to better fit the scene's geometry. As a consequence, Gaussian Splatting generally produces scenes with millions of Gaussians that can be extremely small.

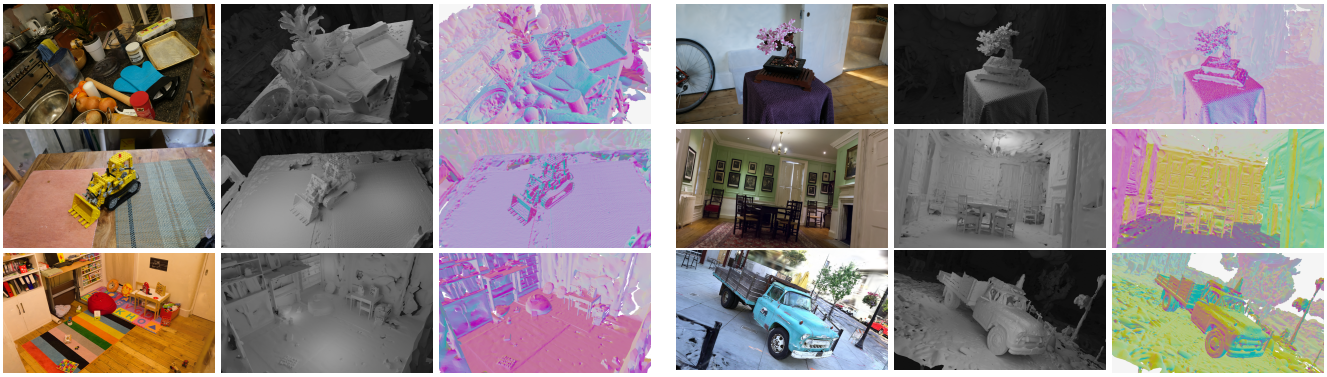
4. Method

We present our SuGaR in this section:

- First, we detail our loss term that enforces the alignment of the 3D Gaussians with the surface of the scene during the optimization of Gaussian Splatting.
- We then detail our method that exploits this alignment for extracting a highly detailed mesh from the Gaussians within minutes on a single GPU.
- Finally, we describe our optional refinement strategy that jointly optimizes the mesh and 3D Gaussians located on the surface of the mesh using Gaussian Splatting rendering. This strategy results in a new set of Gaussians bound to an editable mesh.

4.1. Aligning the Gaussians with the Surface

As discussed in the introduction, to facilitate the creation of a mesh from the Gaussians, we introduce a regularization term into the Gaussian Splatting optimization that encourages the Gaussians to be aligned with the surface of the scene and well distributed over this surface. Our approach is to derive an SDF from the Gaussians under the assumption that the Gaussians have the desired properties. By minimizing the difference between this SDF and the actual SDF computed for the Gaussians, we encourage the Gaussians to have these properties.



(a) Mesh & Gaussians (b) Mesh (No Texture) (c) Mesh normals (a) Mesh & Gaussians (b) Mesh (No Texture) (c) Mesh normals

Figure 4. **Examples of (a) renderings and (b) reconstructed meshes with SuGaR.** The **(c)** normal maps help visualize the geometry.

For a given Gaussian Splatting scene, we start by considering the corresponding density function $d : \mathbb{R}^3 \to \mathbb{R}_+$, computed as the sum of the Gaussian values weighted by their alpha-blending coefficients at any space location p:

$$d(p) = \sum_g \alpha_g \exp\left(-\frac{1}{2}(p-\mu_g)^T \Sigma_g^{-1}(p-\mu_g)\right) , \quad (1)$$

where the μ_g, Σ_g, and α_g are the centers, covariances, and alpha-blending coefficients of the Gaussians, respectively. Let us consider what this density function becomes if the Gaussians are well distributed and aligned with the surface.

First, in such scenario, the Gaussians would have limited overlap with their neighbors. As illustrated in Figure 3 (top-left), this is not the case in general. Then, for any point $p \in \mathbb{R}^3$ close to the surface of the scene, the Gaussian g^* closest to the point p is likely to contribute much more than others to the density value $d(p)$. We could then approximate the Gaussian density at p by:

$$\alpha_{g^*} \exp\left(-\frac{1}{2}(p-\mu_{g^*})^T \Sigma_{g^*}^{-1}(p-\mu_{g^*})\right) , \quad (2)$$

where the "closest Gaussian" g^* is taken as the Gaussian with the largest contribution at point p:

$$g^* = \arg\min_g \left\{(p-\mu_g)^T \Sigma_g^{-1}(p-\mu_g)\right\} . \quad (3)$$

Eq. (2) thus considers that the contribution of the closest Gaussian g^* to the density at p is much higher than the contribution of the other Gaussians. This will help us encourage the Gaussians to be well spread.

We also would like the 3D Gaussians to be flat, as they would then be aligned more closely with the surface of the mesh. Consequently, every Gaussian g would have one of its three scaling factors close to 0 and:

$$(p-\mu_g)^T \Sigma_g^{-1}(p-\mu_g) \approx \frac{1}{s_g^2}\langle p-\mu_g, n_g\rangle^2 , \quad (4)$$

where s_g the smallest scaling factor of the Gaussian and n_g the direction of the corresponding axis. Moreover, because we want Gaussians to describe the true surface of the scene, we need to avoid semi-transparent Gaussians. Therefore, we want Gaussians to be either opaque or fully transparent, in which case we can drop them for rendering. Consequently, we want to have $\alpha_g = 1$ for any Gaussian g.

In such scenario, the density of the Gaussians could finally be approximated by density $\bar{d}(p)$ with:

$$\bar{d}(p) = \exp\left(-\frac{1}{2s_{g^*}^2}\langle p-\mu_{g^*}, n_{g^*}\rangle^2\right) . \quad (5)$$

A first strategy to enforce our regularization is to add term $|d(p) - \bar{d}(p)|$ to the optimization loss. While this approach works well to align Gaussians with the surface, we noticed that computing a slightly different loss relying on an SDF rather than on density further increases the alignment of Gaussians with the surface of the scene. For a given flat Gaussian, i.e., $s_g = 0$, considering level sets is meaningless since all level sets would degenerate toward the plane passing through the center of the Gaussian μ_g with normal n_g. The distance between point p and the true surface of the scene would be approximately $|\langle p-\mu_{g'}, n_{g'}\rangle|$, the distance from p to this plane. Consequently, the zero-crossings of the Signed Distance Function

$$\bar{f}(p) = \pm s_{g*}\sqrt{-2\log\left(\bar{d}(p)\right)} \quad (6)$$

corresponds to the surface of the scene. More generally, we define

$$f(p) = \pm s_{g*}\sqrt{-2\log\left(d(p)\right)} \quad (7)$$

as the "ideal" distance function associated with the density

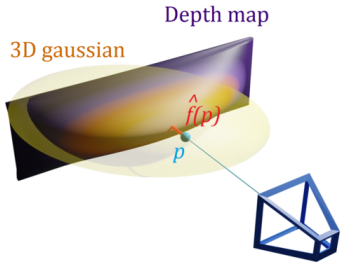


Figure 5. **Efficiently estimating $\hat{f}(p)$ of the SDF of the surface generated from Gaussians.** We render depth maps of the Gaussians, sample points p in the viewpoint according to the distribution of the Gaussians. Value $\hat{f}(p)$ is taken as the 3D distance between p and the intersection between the line of sight for p and the depth map.

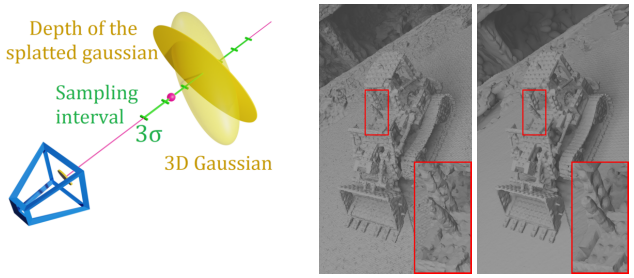


Figure 6. **Sampling points on a level set for Poisson reconstruction. Left:** We sample points on the depth maps of the Gaussians and refine the point locations to move the points on the level set. **Right:** Comparison between the extracted mesh without (left) and with (right) our refinement step. Since splatted depth maps are not exact, using directly the depth points for reconstruction usually results in a large amount of noise and missing details.

function d. This distance function corresponds to the true surface of the scene in an ideal scenario where $d = \bar{d}$. We therefore take our regularization term $\mathcal{R}$ as

$$\mathcal{R} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} |\hat{f}(p) - f(p)| \,, \tag{8}$$

by sampling 3D points p and summing the differences at these points between the ideal SDF $f(p)$ and an estimate $\hat{f}(p)$ of the SDF of the surface created by the current Gaussians. $\mathcal{P}$ refers to the set of sampled points.

Computing efficiently $\hat{f}(p)$ is *a priori* challenging. To do so, we propose to use the depth maps of the Gaussians from the viewpoints used for training—these depth maps can be rendered efficiently by extending the splatting rasterizer. Then, as shown in Figure 5, for a point p visible from a training viewpoint, $\hat{f}(p)$ is the difference between the depth of p and the depth in the corresponding depth map at the projection of p. Moreover, we sample points p following the distribution of the Gaussians:

$$p \sim \prod_g \mathcal{N}(.; \mu_g, \Sigma_g) \,, \tag{9}$$

with $\mathcal{N}(.; \mu_g, \Sigma_g)$ the Gaussian distribution of mean μ_g and covariance Σ_g as these points are likely to correspond to a high gradient for $\mathcal{R}$.

We also add a regularization term to encourage the normals of SDF f and the normals of SDF $\bar{f}$ to also be similar:

$$\mathcal{R}_{\text{Norm}} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \left\| \frac{\nabla f(p)}{\|\nabla f(p)\|_2} - n_{g^*} \right\|_2^2 . \tag{10}$$

4.2. Efficient Mesh Extraction

To create a mesh from the Gaussians obtained after optimization using our regularization terms in Eq. (8) and Eq. (10), we sample 3D points on a level set of the density computed from the Gaussians. The level set depends on a level parameter λ. Then, we obtain a mesh by simply running a Poisson reconstruction [14] on these points. Note that we can also easily assign the points with the normals of the SDF, which improves the mesh quality.

The challenge is in efficiently identifying points lying on the level set. For this, as shown in Figure 6, we again rely on the depth maps of the Gaussians as seen from the training viewpoints. We first randomly sample pixels from each depth map. For each pixel m, we sample its line of sight to find a 3D point on the level set. Formally, we sample n points $p + t_i v$, where p is the 3D point in the depth map that reprojects on pixel m, v is the direction of the line of sight, and $t_i \in [-3\sigma_g(v), 3\sigma_g(v)]$ where $\sigma_g(v)$ is the standard deviation of the 3D Gaussian g in the direction of the camera. The interval $[-3\sigma_g(v), 3\sigma_g(v)]$ is the confidence interval for the 99.7 confidence level of the 1D Gaussian function of t along the ray.

Then, we compute the density values $d_i = d(p + t_i v)$ from Eq. (1) of these sampled points. If there exist i, j such that $d_i < \lambda < d_j$, then there is a level set point located in this range. If so, we use linear interpolation to compute the coefficient t^* such that $p + t^* v$ is the level set point closest to the camera, verifying $d(p + t^* v) = \lambda$. We also compute the normals of the surface at points $\hat{p}$, which we naturally define as the normalized analytical gradient of the density $\frac{\nabla d(\hat{p})}{\|\nabla d(\hat{p})\|_2}$.

Finally, we apply Poisson reconstruction to reconstruct a surface mesh from the level set points and their normals.

4.3. Binding New 3D Gaussians to the Mesh

Once we have extracted a first mesh, we can refine this mesh by binding new Gaussians to the mesh triangles and optimize the Gaussians and the mesh jointly using the Gaussian Splatting rasterizer. This enables the edition of the Gaus-

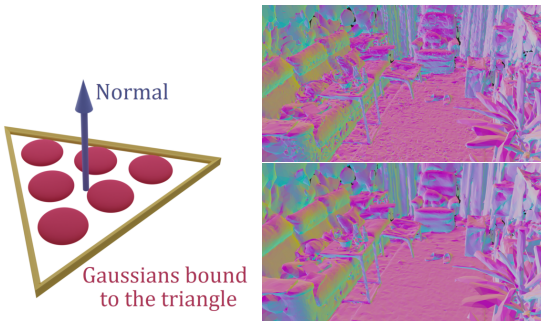


Figure 7. **Joint refinement of mesh and Gaussians. Left:** We bind Gaussians to the triangles of the mesh. Depending on the number of triangles in the scene, we bind a different number of Gaussians per triangle, with predefined barycentric coordinates. **Right:** Mesh before and after joint refinement.

sian splatting scene with popular mesh editing tools while keeping high-quality rendering thanks to the Gaussians.

Given the initial mesh, we instantiate new 3D Gaussians on the mesh. More exactly, we associate a set of n thin 3D Gaussians to each triangle of the mesh, sampled on the surface of the triangle, as illustrated in Figure 7. To do so, we slightly modify the structure of the original 3D Gaussian Splatting model.

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 encoded with a complex number rather than a quaternion, to keep the Gaussians flat and aligned with the mesh triangles. More details about this parameterisation are given in the supplementary material. 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.

Figure 7 shows an example of a mesh before and after refinement. Figure 1 and the supplementary material give examples of what can be done by editing the mesh.

5. Experiments

5.1. Implementation details

All our models are optimized on a single GPU Nvidia Tesla V100 SXM2 32 Go.

Regularization. For all scenes, we start by optimizing a Gaussian Splatting with no regularization for 7,000 iterations in order to let the 3D Gaussians position themselves without any additional constraint. Then, we perform 2,000 iterations with an additional entropy loss on the opacities α_g of the Gaussians, as a way to enforce them to become binary.

Finally, we remove Gaussians with opacity values under 0.5 and perform 6,000 iterations with the regularization term introduced in Subsection 4.1, which makes a total of 15,000 iterations. To compute the density values of points from a Gaussian g, we sum only the Gaussian functions from the 16 nearest Gaussians of g and update the list of nearest neighbors every 500 iterations. Optimization typically takes between 15 and 45 minutes depending on the scene.

Mesh extraction. For all experiments except the ablation presented in Table 2, we extract the λ-level set of the density function for $\lambda = 0.3$. We perform Poisson reconstruction with depth 10 and apply mesh simplification using quadric error metrics [9] to decrease the resolution of the meshes. Mesh extraction generally takes between 5 and 10 minutes depending on the scene.

Joint refinement. We jointly refine the mesh and the bound 3D Gaussians for either 2,000, 7,000 or 15,000 iterations. Depending on the number of iterations, the duration of refinement goes from a few minutes to an hour.

5.2. Real-Time Rendering of Real Scenes

For evaluating our model, we follow the approach from the original 3D Gaussian Splatting paper [15] and compare the performance of several variations of our method SuGaR after refinement on real 3D scenes from 3 different datasets: Mip-NeRF360 [1], DeepBlending [12] and Tanks&Temples [16]. We call R-SuGaR-*N*K a refined SuGaR model optimized for *N* iterations during refinement.

Following [15], we select the same sets of 2 scenes from Tanks&Temples (*Truck* and *Train*) and 2 scenes from DeepBlending (*Playroom* and *Dr. Johnson*). However, due to licensing issues and the unavailability of the scenes *Flowers* and *Treehill*, we perform the evaluation of all methods only on 7 scenes from Mip-NeRF360 instead of the full set of 9 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. Note that [6, 26, 39] also do not use plain textured mesh rendering. We compare to several baselines, some of them focusing only on Novel View Synthesis [2, 15, 23, 41] and others relying on a reconstructed mesh [6, 26, 39], just like our method SuGaR. Results on the Mip-NeRF360 dataset are given in Table 1. Results on Tanks&Temple and DeepBlending are similar and can be found in the supplementary material.

Even though SuGaR focuses on aligning 3D Gaussians for reconstructing a high quality mesh during the first stage of its optimization, it significantly outperforms the state of the art methods for Novel View Synthesis using a mesh and reaches better performance than several famous models

	Indoor scenes			Outdoor scenes			Average on all scenes		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
No mesh (except SuGaR)									
Plenoxels [42]	24.83	0.766	0.426	22.02	0.542	0.465	23.62	0.670	0.443
INGP-Base [23]	28.65	0.840	0.281	23.47	0.571	0.416	26.43	0.725	0.339
INGP-Big [23]	29.14	0.863	0.242	23.57	0.602	0.375	26.75	0.751	0.299
Mip-NeRF360 [2]	**31.58**	0.914	**0.182**	25.79	0.746	0.247	**29.09**	0.842	0.210
3DGS [15]	30.41	**0.920**	0.189	**26.40**	**0.805**	**0.173**	28.69	**0.870**	**0.182**
R-SuGaR-15K (Ours)	29.43	0.910	0.216	24.40	0.699	0.301	27.27	0.820	0.253
With mesh									
Mobile-NeRF [6]	–	–	–	21.95	0.470	0.470	–	–	–
NeRFMeshing [26]	23.83	–	–	22.23	–	–	23.15	–	–
BakedSDF [39]	27.06	0.836	0.258	–	–	–	–	–	–
R-SuGaR-2K (Ours)	26.29	0.872	0.262	22.97	0.648	0.360	24.87	0.776	0.304
R-SuGaR-7K (Ours)	28.73	0.904	0.226	24.16	0.691	0.313	26.77	0.813	0.263
R-SuGaR-15K (Ours)	**29.43**	**0.910**	**0.216**	**24.40**	**0.699**	**0.301**	**27.27**	**0.820**	**0.253**

Table 1. **Quantitative evaluation of rendering quality on the Mip-NeRF360 dataset [2].** SuGaR is best among the methods that recover a mesh, and still performs well compared to NeRF methods and vanilla 3D Gaussian Splatting.

Extraction method	PSNR ↑	SSIM ↑	LPIPS ↓
Marching Cubes [21]	23.91	0.703	0.392
Poisson (centers) [14]	23.76	0.756	0.340
Ours (Surface level 0.1)	24.62	0.765	0.313
Ours (Surface level 0.3)	24.87	**0.776**	**0.304**
Ours (Surface level 0.5)	**24.91**	**0.777**	**0.304**

Table 2. **Ablation for different mesh extraction methods on the Mip-NeRF360 dataset [2] after applying our regularization term.** For 'Poisson (centers)', we apply Poisson reconstruction [14] using as surface points the centers of the 3D Gaussians. For fair comparison, we calibrate the methods to enforce all extracted meshes to have approximately 1,000,000 vertices.

	PSNR ↑	SSIM ↑	LPIPS ↓
1M vertices (3DGS)	**24.51**	**0.768**	**0.295**
1M vertices (UV)	21.24	0.609	0.478
200K vertices (3DGS)	24.24	0.757	0.300
200K vertices (UV)	21.44	0.656	0.419

Table 3. **Comparison between surface-aligned 3D Gaussians and an optimized traditional UV texture on the Mip-NeRF360 dataset [2].** For fair comparison, we only use the diffuse spherical harmonics component when rendering images with SuGaR. Using 3D Gaussians bound to the mesh greatly improves rendering quality, even though it contains less parameters than the UV texture.

that focus only on rendering, such as Instant-NGP [23] and Plenoxels [41]. This performance is remarkable as SuGaR is able to extract a mesh significantly faster than other methods.

Moreover, SuGaR even reaches performance similar to state-of-the-art models for rendering quality [2, 15] on some of the scenes used for evaluation. Two main reasons explain this performance. First, the mesh extracted after the first stage of optimization serves as an excellent initialization for positioning Gaussians when starting the refinement phase. Then, 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. Additional qualitative results are available in Figure 4.

5.3. Mesh Extraction

To demonstrate the ability of our mesh extraction method for reconstructing high-quality meshes that are well-suited for view synthesis, we compare different mesh extraction algorithms. In particular, we optimize several variations of SuGaR by following the exact same pipeline as our standard model, except for the mesh extraction process: We either extract the mesh using a very fine marching cubes algorithm [21], by applying Poisson reconstruction [14] using the centers of the 3D Gaussians as the surface point cloud, or by applying our mesh extraction method on different level sets. Quantitative results are available in Table 2 and show the clear superiority of our approach for meshing 3D Gaussians. Figure 3 also illustrates how the marching cubes algorithm fails in this context.

5.4. Mesh Rendering Ablation

Table 3 provides additional results to quantify how various parameters impact rendering performance. In particular, we evaluate 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 less triangles are already able to reach state of the art results.

Then, we illustrate 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. Qualitative comparisons are provided in the supplementary material.

6. Conclusion

We proposed a very fast algorithm to obtain an accurate 3D triangle mesh for a scene via Gaussian Splatting. Moreover, by combining meshing and Gaussian Splatting, we make possible intuitive manipulation of the captured scenes and realistic rendering, offering new possibilities for creators.

Acknowledgements.

This work was granted access to the HPC resources of IDRIS under the allocation 2023-AD011013387R1 made by GENCI. We thank George Drettakis and Elliot Vincent for inspiring discussions and valuable feedback.

References

[1] Jonathan T. Barron. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. In *International Conference on Computer Vision*, 2021. 4, 7

[2] Jonathan T. Barron. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *Conference on Computer Vision and Pattern Recognition*, 2022. 3, 7, 8, 2, 4

[3] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik P. A. Lensch. NeRD: Neural Reflectance Decomposition from Image Collections. In *International Conference on Computer Vision*, 2021. 3

[4] Chris Buehler, Michael Bosse, Leonard Mcmillan, Steven Gortler, and Michael Cohen. Unstructured Lumigraph Rendering. In *ACM SIGGRAPH*, 2001. 3

[5] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial Radiance Fields. In *European Conference on Computer Vision*, 2022. 3

[6] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. MobileNeRF: Exploiting the Polygon Rasterization Pipeline for Efficient Neural Field Rendering on Mobile Architectures. In *Conference on Computer Vision and Pattern Recognition*, 2023. 3, 4, 7, 8

[7] Chong Bao and Bangbang Yang, Zeng Junyi, Bao Hujun, Zhang Yinda, Cui Zhaopeng, and Zhang Guofeng. NeuMesh: Learning Disentangled Neural Mesh-Based Implicit Field for Geometry and Texture Editing. In *European Conference on Computer Vision*, 2022. 4

[8] François Darmon, Bénédicte Bascle, Jean-Clément Devaux, Pascal Monasse, and Mathieu Aubry. Improving Neural Implicit Surfaces Geometry with Patch Warping. In *Conference on Computer Vision and Pattern Recognition*, 2022. 4

[9] Michael Garland and Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. In *ACM SIGGRAPH*, 1997. 7

[10] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven Seitz. Multi-View Stereo for Community Photo Collections. In *International Conference on Computer Vision*, 2007. 3

[11] Peter Hedman and Pratul P. Srinivasan. Baking Neural Radiance Fields for Real-Time View Synthesis. In *International Conference on Computer Vision*, 2021. 3

[12] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep Blending for Free-Viewpoint Image-Based Rendering. In *ACM SIGGRAPH*, 2018. 3, 7, 2, 4

[13] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. ReLU Fields: The Little Non-Linearity That Could. In *ACM SIGGRAPH*, 2022. 3

[14] Michael M. Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. In *Eurographics*, 2006. 2, 6, 8

[15] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. In *ACM SIGGRAPH*, 2023. 1, 2, 4, 7, 8

[16] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. In *ACM SIGGRAPH*, 2017. 7, 2, 4

[17] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-Based Neural Rendering with Per-View Optimization. In *Computer Graphics Forum*, 2021. 4

[18] Zhengfei Kuang, Kyle Olszewski, Menglei Chai, Zeng Huang, Panos Achlioptas, and Sergey Tulyakov. NeROIC: Neural Rendering of Objects from Online Image Collections. In *ACM SIGGRAPH*, 2022. 3

[19] Marc Levoy and Pat Hanrahan. Light Field Rendering. In *ACM SIGGRAPH*, 1996. 3

[20] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H. Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-Fidelity Neural Surface Reconstruction. In *Conference on Computer Vision and Pattern Recognition*, 2023. 2, 4

[21] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *ACM SIGGRAPH*, 1987. 2, 4, 8

[22] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *European Conference on Computer Vision*, 2020. 1, 3

[23] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. In *ACM SIGGRAPH*, 2022. 3, 7, 8, 2

[24] Michael Oechsle, Songyou Peng, and Andreas Geiger. UNISURF: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction. In *International Conference on Computer Vision*, 2021. 4

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary Devito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*. Curran Associates Inc., 2019. 1

[26] Marie-Julie Rakotosaona, Fabian Manhardt, Diego Martin Arroyo, Michael Niemeyer, Abhijit Kundu, and Federico Tombari. NeRFMeshing: Distilling Neural Radiance Fields into Geometrically-Accurate 3D Meshes. In *DV*, 2023. 2, 3, 4, 7, 8

[27] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D Deep Learning with PyTorch3D. In *arXiv Preprint*, 2020. 1

[28] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding Up Neural Radiance Fields with Thousands of Tiny MLPs. In *International Conference on Computer Vision*, 2021. 3

[29] Gernot Riegler and Vladlen Koltun. Free View Synthesis. In *European Conference on Computer Vision*, 2020. 3

[30] Gernot Riegler and Vladlen Koltun. Stable View Synthesis. In *Conference on Computer Vision and Pattern Recognition*, 2021. 3

[31] Darius Rückert, Linus Franke, and Marc Stamminger. ADOP: Approximate Differentiable One-Pixel Point Rendering. In *ACM SIGGRAPH*, 2022. 4

[32] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo Tourism: Exploring Photo Collections in 3D. In *ACM SIGGRAPH*, 2006. 3, 4

[33] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis. In *Conference on Computer Vision and Pattern Recognition*, 2021. 3

[34] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct Voxel Grid Optimization: Super-Fast Convergence for Radiance Fields Reconstruction. In *Conference on Computer Vision and Pattern Recognition*, 2022. 3

[35] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields. In *Conference on Computer Vision and Pattern Recognition*, 2022. 3

[36] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-View Reconstruction. In *Advances in Neural Information Processing Systems*, 2021. 2, 4

[37] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface Light Fields for 3D Photography. In *ACM SIGGRAPH*, 2000. 3

[38] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume Rendering of Neural Implicit Surfaces. In *Advances in Neural Information Processing Systems*, 2021. 2, 4

[39] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, and Jonathan T. Barron. BakedSDF: Meshing Neural SDFs for Real-Time View Synthesis. In *ACM SIGGRAPH*, 2023. 2, 3, 4, 7, 8

[40] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees For Real-Time Rendering of Neural Radiance Fields. In *International Conference on Computer Vision*, 2021. 3

[41] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance Fields Without Neural Networks. In *Conference on Computer Vision and Pattern Recognition*, 2022. 3, 7, 8

[42] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance Fields Without Neural Networks. In *Conference on Computer Vision and Pattern Recognition*, 2022. 8, 2

[43] Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. PhySG: Inverse Rendering with Spherical Gaussians for Physics-Based Material Editing and Relighting. In *Conference on Computer Vision and Pattern Recognition*, 2021. 3

[44] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Conference on Computer Vision and Pattern Recognition*, 2018. 7, 2, 4

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 the parameterisation of the bound gaussians optimized during our joint refinement strategy.
- Additional implementation details.
- 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. 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.

8. 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

	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.

marginally enhance visual quality by promoting smoother surfaces.

9. 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 6, 7 and 8 provide the detailed results for all scenes in the datasets.

10. Additional Results for Mesh Renderig Ablation

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

First, we provide in Figure 8 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.

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

Figure 8. **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 9. **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.

Figure 10. **Qualitative comparison between (top:) a traditional UV texture optimized from training images, and (bottom:) 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.

Then, in Figure 9 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 10 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.

	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 6. **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 7. **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 8. **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.