Analyzing Gaussian Frosting

Mathis Wauquiez
Ecole des Ponts ParisTech
mathis.wauquiez@eleves.enpc.fr

Félix Fourreau
Ecole des Ponts ParisTech
felix.fourreau@eleves.enpc.fr

Abstract

Recently, 3D gaussian splatting demonstrated promising results in inverse image rendering by representing radiance fields as a set of gaussians, enabling fast and accurate 3D reconstruction. However, gaussian representations are incompatible with mesh-based 3D editing software. The SuGaR method addresses this by retrieving a mesh from the gaussians, facilitating integration with standard 3D modeling tools for editing, animating, sculpting and relightning. Although SuGaR is significantly faster than state-of-the-art Neural Signed Distance Fields (neural SDF), the conversion to a mesh incurs visual quality loss as the gaussians are constrained to the mesh, which limits volumetric rendering. To overcome this, Gaussian Frosting creates a "frosting" layer that encases the mesh, used for volumetric rendering using gaussian splatting. Additionally, the proposed approach supports deformable meshes, automatically adjusting the radiance field carried by the mesh, therefore enabling easy scene editing while maintaining competitive visual quality.

1. Introduction

Gaussian Frosting [8] is an article that describes a method to obtain a hybrid representation combining triangle mesh and radiance field modeled by a set of Gaussians, identical to the one proposed in Gaussian Splatting for Real-Time Radiance Field Rendering [10]. This joint representation is particularly useful as it combines the best of both worlds: a surface easily usable in 3D editing and animation software, while offering superior rendering quality compared to a simple triangle mesh. To thoroughly understand the challenges behind this hybrid modeling, we examine the various existing methods pursuing similar objectives, before describing the contributions made to the field by this article.

1.1. Related Work

Image Based Rendering (IBR) consists of creating a scene representation from multiple views of this scene. Several scene representations have been created, which are editable like triangle meshes or point clouds, or implicit like neural implicit radiance fields, voxel grids, or gaussian splatting.

Volumetric Approaches. Volumetric approaches are primarily based on radiance fields. NeRF [12] is a particular example of this approach: it was a major breakthrough in the field, using a MultiLayer Perceptron (MLP) to model the scene's radiance field, and using volumetric ray tracing through this radiance field to perform rendering. This approach has the advantage of modeling radiance fields of high complexity, allowing competitive rendering quality. This modeling capacity comes at the cost of higher memory usage, longer training time, and increased rendering time. Numerous subsequent works attempted to minimize its computational cost, like Instant-NGP [13] or Plenoctrees [21], reduce the number of poses necessary to estimate the radiance field, like PixelNeRF [22] and MVSNeRF [2]. Other approaches generalize NeRF to unbounded scenes and analyze its functioning, like NeRF++ [23] which proposes a new parameterization compatible with non-bounded scenes. This article [5] reviews the different applications of the NeRF framework until the end of 2023.

Gaussian Splatting. Gaussian splatting is an IBR method born in 2023, which models the radiance field as a large set of Gaussians, where each Gaussian g is equipped with a radiance parameterized by an opacity $\alpha_g \in [0, 1]$ and a set of coordinates defining the colors emitted in all directions, with a covariance matrix $\Sigma_g \in \mathbb{R}^{3x3}$ parameterized by a rotation $q_g \in \mathbb{R}^4$ and a size $s_g \in \mathbb{R}^3$, as well as a mean $\mu_g \in \mathbb{R}^3$.

Rendering is then performed using a rasterizer, which "splats" 3D Gaussians into ellipses and then aggregates them according to their opacity and depth to form the image. This process is extremely fast, enabling very rapid op-

timization and rendering, which has led to its widespread adoption despite sometimes less impressive results than those obtained by NeRF.

This adoption is highlighted by this paper [3], which pedagogically explains the method employed by 3DGS, the different applications created following its publication, the different research directions taken on the subject, and constitutes an excellent introduction to the topic.

Surface Approaches In 2006, the Structure-from-Motion (SfM) [18] method was published. This IBR method allows retrieving a dense point cloud describing a scene from which a series of photos were captured. This method, along with Multi-View Stereo (MVS) [6], were used to create representations in the form of triangle meshes for new view generation. Deep learning methods also exist for describing observed scenes with meshes, with notable applications in the medical field and for determining meshes associated with humans.

This representation is particularly convenient as it is compatible with most 3D editing and animation programs. However, in terms of rendering quality, a mesh does not allow capturing volumetric effects and provides qualitatively inferior results, especially for fuzzy materials.

Hybrid Approaches There are also hybrid methods that use differentiable rendering developed for volumetric approaches to combine the advantages of mesh-based methods and volumetric methods, enabling a greater diversity of applications. This is notably the case for methods based on Signed Distance Function (SDF), which represents the distance between a space point and the nearest surface, positive outside and negative inside. From this function, a surface mesh can be extracted, using for example the Marching Cubes algorithm [11].

Compared to a NeRF approach, learning the SDF allows for stronger regularization of the underlying geometry. We can notably cite [14], which proposes a method and in-depth analysis for applying neural networks to SDF modeling. These results were subsequently improved, particularly by realizing that using a ReLU activation function implies that modeled functions are piecewise affine and not always easy to train. Thus, [17] proposes replacing this activation function with a periodic function, namely a sine.

Once this SDF is found, directly providing the distance to the nearest surface, it allows for efficient ray casting using the SDF sphere tracing method, or the more efficient frustum ray casting method proposed in [16], which also explores other geometric possibilities offered by these implicit representations.

Another example of a hybrid approach is Adaptive Shells [20]. Its main objective is to improve NeRF rendering speed, based on the observation that not all scene areas require the same rendering style, with homogeneous surfaces perfectly modelable by flat surfaces, while fuzzy surfaces need more intense volume rendering.

Therefore, they develop an approach inspired by recent advances in SDFs to extract a shell around the surface, with a width that adapts to the local complexity of the scene, enabling volumetric rendering to be restricted and thereby accelerated. They use a NeRF to model the radiance field, and their approach allows them to completely ignore empty spaces during ray casting. A key element of their paper is that constraining volumetric rendering to a specific region of space improved rendering quality compared to a more naive NeRF approach and allowed for a much better distribution of the rendering's computational cost, echoing the findings of Instant-NGP [13]. This could be explained by partially resolving the radiance-shape ambiguity, highlighted by [23], through stronger surface regularization.

Thus, Gaussian Frosting positions itself in the lineage of Adaptive Shells: like them, their method is based on extracting an envelope to constrain volumetric rendering. But where Adaptive Shells used a radiance field modeled by NeRF, Gaussian Frosting prefers a field modeled by 3DGS, which is computationally much more efficient and has a natural way of being deformed simultaneously with the mesh for editing and animation in software.

Mesh Extraction Another difference with Adaptive Shells lies in the extraction of the shell. Adaptive Shells uses a method derived from NeuS,[19] which proposes reconstructing the volumetric density $\sigma(\mathbf{x})$ in space using an SDF f defined by a network, as $\sigma = \max\left(-\frac{\frac{d\Phi_s}{d\tau}(f)}{\Phi_s(f)}, 0\right)$, where f is the SDF and $\Phi_s(f)$ is a sigmoid: $\Phi_s(f) = (1+\exp(-f/s))^{-1}$. This SDF-based formulation allows the use of specific loss functions, notably the Eikonal regularizer, which encourages f to behave as a true distance function. Adaptive Shells, as cited by the Gaussian Frosting paper, enables shell extraction by revisiting the SDF formulation of NeuS to introduce a spatially-varying kernel size, where the kernel size used to determine σ is a space-dependent parameter predicted by the same network that defines the SDF.

SuGaR. The method used by the article for envelope extraction is based on SuGaR [7], which allows rapid mesh extraction using Gaussian splatting Gaussians. SuGaR encourages Gaussians to align with the surface using a regularization term, based on the observation that the volumetric rendering equation for Gaussians $d(p) = \sum_g \alpha_g \exp\left(-\frac{1}{2}(p-\mu_g)^T\Sigma^{-1}(p-\mu_g)\right)$ would ideally have the expression $\bar{d}(p) = \exp\left(-\frac{1}{2\sigma_{g^*}^2} < p-\mu_{g^*}, n_{g^*} >^2\right)$ if Gaussians were

perfectly surface-aligned and flat, with g^* being the closest Gaussian to p, $\sigma^2_{g^*}$ the smallest covariance factor, and n_{g^*} the corresponding axis.

More specifically, the regularization term is the Mean Absolute Error (MAE) between the ideal distance function $\bar{f}(p) = \pm\sqrt{-2log(\bar{d}(p))}$ and the actual term $f(p) = \pm\sqrt{-2log(d(p))}$. They also use an L2 regularization term encouraging similarity between the ideal SDF normals n_{g^*} and the real normals $\frac{\nabla f(p)}{||\nabla f(p)||_2}$. After optimization, they use surface-aligned Gaussians to reconstruct the mesh by sampling points at the λ density level determined by the Gaussians and applying Poisson reconstruction [9]. This allows very rapid mesh reconstruction, and they combine this mesh with Gaussian field optimization constrained to the mesh to benefit from high rendering quality. It is worth noting that Poisson reconstruction introduces some hyperparameters, particularly the octree size used for reconstruction.

1.2. Contributions

Gaussian Frosting proposes a new hybrid representation, combining a mesh and a 3DGS radiance field constrained by this mesh, along with an automatic method to determine the optimal octree size D for Poisson reconstruction, which adapts to the scene's complexity.

2. Methodology

This section describes the approach used in the paper. It begins by retrieving the surface mesh with a small enhancement to SuGaR, then defines the inner and outer bounds of the frosting layer, and finally optimizes its representation.

2.1. Retrieving the surface mesh

In order to retrieve the mesh, the authors use SuGaR [7], a method we described above. A downside of SuGaR is the need for the depth of the octree used for the Poisson reconstruction. They propose a method that automatically determines the optimal depth of the octree, which removes the need for user trial and error. By default, SuGaR [7] uses a large depth for the octree depth, but a resolution that is too high has the downside of letting gaussians become visible, as ellipsoidal bumps on the surface of the mesh, or can lead to holes in the mesh. One possible approach would be to design the depth of the octree such that its leaves matches the average size of the gaussians. Yet, this approach fails in practice because of the high variance in the gaussians' shapes. The authors noticed that a better indicator for the geometrical complexity of the scene is the distance between gaussians. Indeed, very detailed shapes can be recomposed using large but close gaussians, whereas gaussians far from each other produce a rough appearance. Using this insight, they define the formulation for the geometrical complexity score CS:

$$\text{CS} = Q_{0.1}\left(\left\{\min_{g' \neq g} \frac{\|\mu_g - \mu_{g'}\|_2}{L}\right\}_{g \in \mathcal{G}}\right)$$

With $\mathcal{G}$ the set of gaussians, L the length of the longest edge of the bounding bo of the point cloud to use in Poisson reconstruction, and $Q_{0.1}$ the 0.1-quantile function. They motivate the use of the quantile because it is the gaussians that have a close neighbors that encode most details (and that thus the average would not describe the adequate property), and that it it more robust to extreme values. This makes CS an estimator of scene complexity, based on the distances between the closest Gaussians. And since the normalized length of an octree leaf is $2^{-\bar{D}}$, it makes sense that:

$$\bar{D} = \lfloor \log_2(\gamma.\text{CS} \rfloor$$

, with $\gamma > 0$ a hyperparameter **independent** of the scene. They choose $\gamma = 100$ and their results validates that this method results in greater rendering performance.

2.2. Defining the bounds of the frosting layer

In order to define the frosting layer, for each vertex v_i they define δ_i^{in} and δ_i^{out}, two values that parameterizes the inner and outer bounds for the frosting layers. This gives two surfaces, to which we can later constraint the volumetric rendering to the inside of two surfaces with vertices $(\mathbf{v}_i + \delta_i^{\text{in}}\mathbf{n}_i)_i$ and $(\mathbf{v}_i + \delta_i^{\text{out}}\mathbf{n}_i)_i$, where $\mathbf{n}_i$ is the mesh normal at vertex $\mathbf{v}_i$. To find these two values, they use both the unconstrained and regularized Gaussians: they estimate the Frosting thickness from the unconstrained Gaussians, using isosurfaces close o the regularized Gaussians. The search for δ_i^{in} and δ_i^{out} involves two steps: identifying a reliable interval J_i and estimating the values from Gaussian isosurfaces.

More precisely, they define: $I_i = [-3\sigma_i, 3\sigma i]$, with σ_i the standard deviation in the direction of $\mathbf{n}_i$ of the regularized Gaussian the closest to v_i. This is the 99.7 confidence interval of the 1D Gaussian function of t along the normal. Usually, the gaussians misalign with the surface in fuzzy surfaces, and as such the variance along the normal is higher compared to perfectly flat surfaces. It is highly likely that δ_i^{in} and δ_i^{out} are inside this interval. But instead of using I_i to find both, the authors state that there exists a much more reliable interval J_i. To find it, they solve:

$$\epsilon_i^{\text{in}} = \inf(T), \quad \epsilon_i^{\text{out}} = \sup(T)$$
$$\text{with} \quad T = \{t \in I_i \mid d_r(\mathbf{v}_i + t\mathbf{n}_i) \geq \lambda\},$$

where d_r is the density function for the regularized Gaussians.

Let's interpet this. First of all, let's recall that δ_i^{in} and δ_i^{out} aim to be the intersection of the isosurfaces of level λ

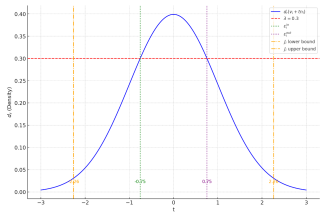


Figure 1. ϵ_i^{in} and ϵ_i^{out} are found by finding the intersection of the isosurface of level λ of the density function and a ray $(v_i + tn_i)_{t \in I_i}$

of the gaussian density and the ray cast from the vertex v_i with direction n_i. Formally, this means that $\delta_{\text{i}}^{\text{in}}, \delta_{\text{i}}^{\text{out}} \in \bar{T} = \{t \in \mathbb{R} | d_r(v_i + t\mathbf{n_i}) = \lambda\}$. However, searching for $t \in \mathbb{R}$ is infeasible. Restricting the search to I_i allows to make the computation. Here, if $\delta_{\text{i}}^{\text{in}}$*(respectively* δ_{i}^{out}*)* $\in T$, then we have that $\delta_{\text{i}}^{\text{in}} = \epsilon_i^{\text{in}}$ and that $\delta_{\text{i}}^{\text{out}} = \epsilon_i^{\text{out}}$. If not, $\epsilon_i^{\text{in}} = -3\sigma_i$ (respectively, $\epsilon_i^{\text{out}} = 3\sigma_i$). 1 Illustrates the search for ϵ_i^{in} and ϵ_i^{out}.

But as there are multiple gaussians contributing to the radiance field, the algorihm might miss on a gaussian outside of I_i. This is the reason why they calculate $J_i = \left[\epsilon_i^{\text{mid}} - k\epsilon_i^{\text{half}}, \epsilon_i^{\text{mid}} + k\epsilon_i^{\text{half}}\right]$, with $\epsilon_i^{\text{mid}} = \frac{\epsilon_i^{\text{in}} + \epsilon_i^{\text{out}}}{2}$, $\epsilon_i^{\text{half}} = \frac{\epsilon_i^{\text{out}} - \epsilon_i^{\text{in}}}{2}$ and $k = 3$, which is a much larger interval.

They then hope that $\delta_{\text{i}}^{\text{in}}$ and $\delta_{\text{i}}^{\text{out}}$ belong to J_i, and sets them to the solution of the exact problem, replacing ϵ_i^{in} by $\delta_{\text{i}}^{\text{in}}$, ϵ_i^{out} by $\delta_{\text{i}}^{\text{out}}$, and I_i by J_i.

In practice, this does improve the rendering quality, as demonstrated in their ablation study for the impact of thickness of the frosting layer: the results using I_i instead of J_i are slightly worse.

It is to be noted that an interval too big might also be an issue, as the Gaussians far from the surface, for example in another object, could participate to the definition of the frosting layer.

2.3. Radiance field optimization

In this section we describe how:

- The authors propose a new parametrization for the Gaussians
- This parameterization can be adapted for editing, animating, and deforming
- The initialization and optimization are done

2.3.1. Mesh-bound Parameterization

The main objective of the parameterization is to introduce a way to optimize the radiance field using standard differentiable rendering mechanisms developped for 3DGS. To this end, for each triangle of the base mesh, which contains at most one gaussian g, they parameterize this gaussian's mean μ_g by its barycentric coordinates in the irregular triangular prism formed by $(v_i + \delta_{\text{i}}^{\text{in}} n_i)_{i=0,1,2}$ and $(v_i + \delta_{\text{i}}^{\text{out}} n_i)_{i=0,1,2}$, with v_0, v_1, v_2 being the three vertices of the triangle and n_0, n_1, n_2 their corresponding normals. Practically, this means that:

$$\mu_g = \sum_{i=0}^{2} \left(b_g^{(i)} \left(\mathbf{v}_i + \delta_i^{\text{out}} \mathbf{n}_i \right) + \beta_g^{(i)} \left(\mathbf{v}_i + \delta_i^{\text{in}} \mathbf{n}_i \right) \right),$$

with $(b_i^{(i)})_{i=0,1,2}$ and $(\beta_i^{(i)})_{i=0,1,2}$ the barycentric coordinates, that sums up to one. A practical tip is that instead of using constrained optimization to optimize the radiance field with respect to the means, they apply a softmax activation to the parameters to optimize such that they sum up to 1, meaning there is no particular adaptation needed.

2.3.2. Deforming the Gaussians with the mesh

The author proposes a strategy to automatically adjust the parameters of the Gaussians. First of all, the mean of the Gaussians do not need adjustment as it is determined using its barycentric coordinates, that remains the same when the mesh is deformed. Secondly, to determine the scaling and rotation of the gaussian, for each triangle of the surface mesh they first estimate the local transformation applied on each of the vertices $(v_i)_{i=0,\dots,6}$ forming the two triangles belonging to the shell. The rotations for each vertex v_i are computed using an axis-angle representation, where the axis angle is the previous and current values of $(c - v_i)$, with c the mean of the vertices. The local rescaling is determined as the scaling along the axis $(c - v_i)$. Then, to update the scaling factors, rotations, and spherical harmonics of a Gaussian g, they apply each of these six transformations on the three main axes of the Gaussian, before averaging the resulting axes using the barycentric coordinates of the center μ_g of the Gaussian, and finally orthonormalize the three resulting axes.

This section lacks rigor, as it is unclear how the rescaling transformations are computed, and the article does not say how to update the spherical harmonics of the Gaussians, which are not even mentioned in this section. Moreover, an approach based on explicit homography estimation could have been more rigorous.

2.3.3. Gaussians Initialization and Optimization

To initialize N Gaussians, with N provided by the user, they first initialize their centers. The first half of the Gaussians are iniatialized in random cells, with a probability proportional to the volume of the frosting for this cell. The center of the Gaussian is then determined using random barycentric coordinates. For the second half, the probability is uniform. This split allows to distribute the rendering complexity in a manner that allows to correctly model fuzzy areas with complex geometries, as well as flat parts that have a lot of texture details. The colors are simply initialized using the color of the closest Gaussian in the unconstrained representation and the opacity, scaling factors and opacity are randomly determined.

The Gaussians' parameters are then optimized using a classical 3DGS differentiable renderer, adding backpropagation through the parameterization of the means μ_g to optimize the barycentric coordinates.

3. Results

In this section, we detail the practical implementation of the Gaussian Frosting method :

Input from COLMAP The process begins with data acquisition using COLMAP[15], a Structure-from-Motion (SfM) and Multi-View Stereo (MVS) pipeline that reconstructs 3D scenes from image sequences It estimates camera poses (positions and orientations) and generates point clouds representing the geometry of the scene. The original gaussian splatting paper recommended [10] to use between 100 and 300, using ffmpeg we always made sure to stay around 200 when extracting from video.

Vanilla gaussian splatting From theses outputs the process follows by the application of gaussian splatting. The first gaussians are initialized using the sfm points given by COLMAP[**?**], and are being divided and optimized though learning to math the real scene. Once the training is complete the vanilla gaussians are stored.

Initial Mesh with SuGaR The newly obtained gaussian are then used by SUGAR to get a primary editable mesh by aligning the vanilla Gaussians with the scene's surface.

Frosting layer The mesh obtained from SUGAR are then used by the the Frosting process to add variable thickness to the mesh corresponding to the material "fuzinness" present in the scene.

Output We then obtain two primary outputs:

- **Refined Gaussians**: An optimized set of Gaussians confined within the Frosting layer, representing the scene's radiance field with enhanced volumetric effects.
- **Associated Mesh**: A triangular mesh extracted from the aligned Gaussians. This mesh serves as the editable base for further modifications using 3D editing tools like Blender.

Configurable Parameters

Upon calling the `train_full_pipeline.py` we have some choice to make between the different parameters :

- **Regularization Method** `-r`:
  - `-r "dn_consistency"`: Directional normal consistency regularization.
  - `-r "density"`: Density-based regularization
  - `-r "sdf"`: Signed distance

  We chose the `"dn_consistency"` normalisation as it was the one recommended in the paper. We weren't able to test the other regularization much as each training was taking a long time and most of them failed (due to vram limits)
- **Number of Gaussians** `-n`:
  - `-n N`: Sets the total number of Gaussians to N. The paper recommends 2,000,000 for synthetic scenes and 5,000,000 for realistic scenes. However due to our limited computational resources we used between 1,000,000 and 2,000,000.
- **Occlusion Culling** `--use_occlusion_culling`:
  - Enables occlusion culling, used to optimizes rendering by excluding Gaussians occluded by others. This enhances performance without compromising visual quality.

Practical Example The example scene was captured using a drone flying over a field, focusing on a person (the writer). For computational efficiency, the video was downscaled to 720p. Due to hardware limitations (using an RTX 3060 with lower VRAM compared to the Nvidia Tesla V100 SXM2 32GB used in the paper), we used 2,000,000 Gaussians instead of the recommended 5,000,000 for realistic scenes

We were able to view and manipulate the extracted mesh with Blender [4] using the provided Blender add-on [1]. This allows to manipulate, animate, and edit the mesh while maintaining the initial rendering provided by the Frosting layer.

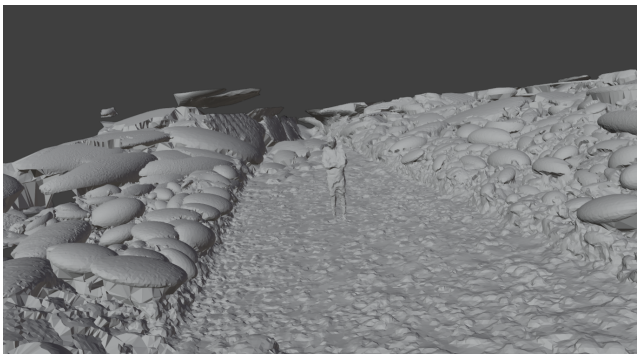
Despite using fewer Gaussians and a less powerful GPU, we achieved a recognizable reconstruction of the scene (see Figure 2). The person and the drone pilot are clearly identifiable, although the rendering quality is not as refined as that of the paper's implementation. The bumpy ground surface seems realistic; however, the side grass isn't as clear. This is due to the low quality of the video, which doesn't allow for accurately assigning the Gaussians.

We can compare these results (see Figure 3) with the original Gaussian Splatting with a similar number of iterations (the Gaussian Splatting alone is, of course, much faster).

The colors appear more vibrant in the vanilla gaussian and are closer the photo, but the edge of the grass is less pronounced and less realistic than for gaussian frosting, the ground itself appears flat in the vanilla version yet it wasn't the case.

Therefore, the Gaussian Frosting implementation demonstrated reasonable performance given the hardware constraints. Training the model with 2,000,000 Gaussians on an RTX 3060 GPU took approximately 3 hours, achieving a recognizable but lower-quality reconstruction compared to the paper's results, which utilized a more

(a) With shaders



(b) Without shaders

Figure 2. 3D reconstruction of the scene using Gaussian Frosting.

(a) Original Input Image for comparison.

(b) Vanilla Gaussian Splatting: Offers vibrant and finer detail but lacks volumetric detail.

(c) Gaussian Frosting: Provides better volumetric detail in complex regions.

Figure 3. Comparison between input image, vanilla Gaussian Splatting output, and Gaussian Frosting output.

powerful Nvidia Tesla V100 SXM2 32GB GPU and recommended 5,000,000 Gaussians.

4. Discussion

While Gaussian Frosting presents a significant advancement in the realm of editable radiance fields, several limitations and challenges warrant attention. This discussion critically examines both the inherent constraints outlined in the original work and those encountered during our implementation. Furthermore, potential avenues for improvement are explored to enhance the method's practicality and performance.

4.1. Limitations of the Original Method

High Resource Requirements One of the primary limitations of the Gaussian Frosting method, as presented in the original article, is its substantial demand for computational resources. The method recommends utilizing up to 5,000,000 Gaussians for realistic scenes, which necessitates powerful hardware such as the Nvidia Tesla V100 SXM2 32GB GPU. This high computational and memory footprint poses a significant barrier to entry for practitioners without access to such specialized hardware, effectively limiting the method's accessibility and broader adoption.

Experimental Nature and Hyperparameter Sensitivity The Gaussian Frosting approach is highly experimental, relying on a multitude of hyperparameters that appear to be set empirically without particular reason. Parameters such as the octree depth in Poisson surface reconstruction and the thickness of the Frosting layer are crucial for the method's success yet are determined through heuristic methods. Although the octree depth is determined automatically, this process introduces another hyperparameter, γ, which is chosen somewhat arbitrarily (e.g., $\gamma = 100$). Additionally, the method employs a quantile of 0.1 for certain computations without providing a justification for selecting this specific value over others (such as 0.2 or 0.01) and lacks validation to support the suitability of this choice. This sensitivity to hyperparameter settings not only complicates the implementation process but also raises concerns about the method's robustness and generalizability across diverse scenes. The reliance on arbitrary hyperparameter values without thorough validation undermines confidence in the method's ability to consistently perform well in varied real-world applications.

Reproductibility issues Another significant limitation of the Gaussian Frosting method pertains to reproducibility. The original article lacks comprehensive details on updating the spherical harmonics coefficients. Furthermore, the strategy employed for initializing Gaussians within the Frosting layer appears unconventional and almost arbitrary. The method combines uniform and volume-proportional sampling without a clear justification for the chosen proportions or sampling techniques. This approach introduces an element of randomness that can lead to inconsistent Gaussian distributions across different implementations.

4.2. Limitations Observed in the Implementation

Lack of Checkpointing and Error Recovery Our implementation of Gaussian Frosting faced practical challenges, notably the absence of checkpointing mechanisms. In scenarios where computational processes are interrupted due to errors or system failures, the entire optimization process must be restarted from the beginning. This inefficiency exacerbates the already high computational costs, making iterative experimentation and debugging cumbersome and time-consuming.

Worse Visual Quality Compared to Vanilla Gaussian Splatting Despite adhering to the method's prescribed parameters, our results exhibited visually inferior quality compared to vanilla Gaussian splatting. Specifically, the Frosting layer, while effective in capturing volumetric details, did not achieve the same level of 'visual' detail.

4.3. Potential Improvements

Computational Efficiency To mitigate the high resource requirements, future work could explore techniques for reducing the number of Gaussians without compromising visual quality. Methods such as Gaussian pruning could substantially decrease both memory usage and computational time.

Checkpointing Mechanisms Integrating checkpointing functionality would allow the optimization process to resume from intermediate states in the event of interruptions. This enhancement would save significant time as for instance some step like vanilla gaussian computation are redundant when changing parameters such as regularization model.

5. Conclusion

Gaussian Frosting offers a promising approach for combining high-quality rendering with editability. It holds considerable potential for applications requiring real-time rendering and editable 3D representations, such as video games and virtual reality. This approach seems particularly applicable, as it is relatively simple and yet very effective, demonstrating rendering results comparable to vanilla 3DGS at similar frame rates.

This approach could however benefit from a few improvements, notably practical improvements for the code distributed by the authors, as well as improvements to enhance the experimental parts and the hyperparameter sensitivity.

References

[1] Anttwo. Sugar frosting blender add-on. https://github.com/Anttwo/sugar_frosting_blender_addon/, 2024. Accessed: 2024-04-27. 5

[2] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo, 2021. 1

[3] Guikun Chen and Wenguan Wang. A survey on 3d gaussian splatting, 2024. 2

[4] Blender Foundation. Blender: Open source 3d creation suite. https://www.blender.org/, 2023. Accessed: 2024-04-27. 5

[5] Kyle Gao, Yina Gao, Hongjie He, Dening Lu, Linlin Xu, and Jonathan Li. Nerf: Neural radiance field in 3d vision, a comprehensive review, 2023. 1

[6] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M. Seitz. Multi-view stereo for community photo collections. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, 2007. 2

[7] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering, 2023. 2, 3

[8] Antoine Guédon and Vincent Lepetit. Gaussian frosting: Editable complex radiance fields with real-time rendering, 2024. 1

[9] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, page 61–70, Goslar, DEU, 2006. Eurographics Association. 3

[10] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering, 2023. 1, 5

[11] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery. 2

[12] 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, 2020. 1

[13] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41 (4):1–15, 2022. 1, 2

[14] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019. 2

[15] Johannes L Schönberger and Jens Frahm. Colmap: A general-purpose structure-from-motion and multi-view stereo pipeline. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2133–2142, 2016. 5

[16] Nicholas Sharp and Alec Jacobson. Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis, 2022. 2

[17] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems*, pages 7462–7473. Curran Associates, Inc., 2020. 2

[18] Noah Snavely, Steven Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Transactions on Graphics (2006)*, 25:835–846, 2006. 2

[19] 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, 2023. 2

[20] Zian Wang, Tianchang Shen, Merlin Nimier-David, Nicholas Sharp, Jun Gao, Alexander Keller, Sanja Fidler, Thomas Müller, and Zan Gojcic. Adaptive shells for efficient neural radiance field rendering, 2023. 2

[21] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields, 2021. 1

[22] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images, 2021. 1

[23] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields, 2020. 1, 2