## A Atomized Proliferation Algorithm

The Atomized Proliferation algorithm is summarized in Algorithm 1. It starts with setting parameters for Clone Threshold ($\tau_c$), Split Threshold ($\tau_s$), Prune Threshold ($\varepsilon$), Atom Scale ($\mathcal{S}_a$), and defining duration limits for atomized proliferation ($t_a$) and warm-up phase ($t_w$). The algorithm iteratively processes each Gaussian property ($\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{c}, \alpha$) from the Gaussian set ($\boldsymbol{M}, \boldsymbol{S}, \boldsymbol{C}, \boldsymbol{A}$). A Gaussian is pruned if its $\alpha$ falls below the threshold $\varepsilon$ or its covariance ($\boldsymbol{\Sigma}$) is excessively large. If the gradient of the loss ($\nabla_p \mathcal{L}$) exceeds $\tau_c$, the Gaussian is cloned to potentially bridge geometry gaps. Additionally, the Gaussian is split when $\nabla_p \mathcal{L}$ meets a dynamically adjusted threshold based on the warm-up progress and if the norm of $\boldsymbol{\Sigma}$ exceeds the Atom Scale ($\mathcal{S}_a$). Atomization takes place when the minimum norm of $\boldsymbol{S}$ is less than or equal to Atom Scale $\mathcal{S}_a$ and within the proliferation timeframe ($t_a$), ensuring detail refinement before the proliferation endpoint.

**Algorithm 1** Atomized Proliferation

```
Require: Clone Threshold $\tau_c$, Split Threshold $\tau_s$, Prune Threshold $\varepsilon$,
         Atom Scale $\mathcal{S}_a$, Atomized Proliferation until $t_a$, Warm-Up until $t_w$
 1: for all Gaussian$(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{c}, \alpha)$ in $(\boldsymbol{M}, \boldsymbol{S}, \boldsymbol{C}, \boldsymbol{A})$ do
 2:     if $\alpha < \varepsilon$ or IsTooLarge$(\boldsymbol{\Sigma})$ then
 3:         Prune$(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{c}, \alpha)$
 4:     end if
 5:     if $\nabla_p \mathcal{L} \geq \tau_c$ then
 6:         Clone$(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{c}, \alpha)$
 7:     end if
 8:     if $\nabla_p \mathcal{L} \geq \min\left(\frac{i}{t_w}\tau_s, \tau_s\right)$ and $||\boldsymbol{S}||_{\max} > \mathcal{S}_a$ then
 9:         Split$(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{c}, \alpha)$
10:     end if
11:     if $||\boldsymbol{S}||_{\min} \leq \mathcal{S}_a$ and $i < t_a$ then
12:         Atomize$(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{c}, \alpha)$
13:     end if
14: end for
```

## B Gaussian Proliferation Trend

In Figure 1, we illustrate the Gaussian Proliferation Trend, which tracks the count of Gaussians across iterations for nine different scenes within the Mip-NeRF360 dataset. The depicted curve represents the average number of Gaussians across these scenes, with the curve's width indicates the standard deviation. The fluctuations observed highlight the effectiveness of the opacity resetting strategy in eliminating redundant Gaussians. Initially, the 3DGS method struggles to densify Gaussians, as indicated by an increasing standard deviation, and it appears unable to stabilize by the end of the proliferation stage. In contrast, our method employs a warm-up strategy that aggressively densifies Gaussians at the initial stage, followed by a phase where Gaussians begin to merge, leading to a declining and stabilizing trend in Gaussian proliferation.

On the Mip-NeRF360 dataset, our method demonstrates efficiency with an average training time of 0.28 hours and a final model size of 749MB, compared to 3DGS, which takes

0.40 hours for training and results in a model size of 869MB. This indicates that our approach achieves superior quality without compromising on training time or model size.

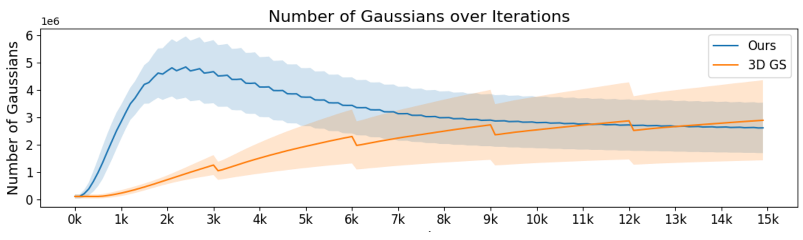

Figure 1: Gaussian Proliferation Trend.

## C Implementation Details

**Codebase:** We have developed AtomGS based on the 3D Gaussian Splatting (3DGS) framework [3]. To facilitate Edge-Aware Normal Loss computation and Poisson mesh extraction, we have implemented an additional feature renderer. This renderer generates various maps, including accumulation, median and mean depth, normal, and curvature maps. Additionally, we've developed an interactive real-time viewer that allows for the monitoring of these features, providing a detailed analysis of Gaussians in terms of both RGB and geometric information. For a detailed derivation of these implementations, please refer to Section D.
**Hyper Parameter Settings:** Following the 3DGS, we set the Clone Threshold at $\tau_c = 0.002$, Split Threshold at $\tau_s = 0.002$, and Prune Threshold at $\varepsilon = 0.005$. For the Atom-related settings, we set Atom Scale at the first percentile of distances from the input SfM points $\mathcal{S}_a = P_1(\boldsymbol{d})$, Atomized Proliferation until iteration at $t_a = 7000$, and Warm-Up until iteration at $t_w = 7000$. For optimization, the weights for MS-SSIM and normal loss calculations are both set at $\lambda_{ms-ssim} = \lambda_{normal} = 0.1$. When working with object-centered datasets that lack extensive backgrounds, we advise setting the scale learning rate $\eta_s = 0$ to maximize geometric accuracy. Specifically for the DTU dataset, we set Prune Threshold $\varepsilon = 0.5$, loss weight $\lambda_{ms-ssim} = 1$, Atom Scale $\mathcal{S}_a = P_{10}(\boldsymbol{d})$, ant total training iterations at 7k.
**Mesh Extraction:** Mesh extraction involves rendering depth maps from training views, which use median depth values from splats projected onto pixels. These maps are then converted back into 3D space to derive corresponding normal maps. The oriented colored point cloud generated from the RGB image, depth map, and normal map serves as the input for the Poisson extraction method [2], which is used to create the textured mesh. This process is illustrated in Figure 2.
**Hardware:** All experiments are conducted on a single NVIDIA GeForce RTX 4090 GPU.

## D Gaussian Splatting and Additional Feature Rendering

**Splatting:** During this stage, 3D Gaussians are projected into the 2D image space to facilitate rendering. Utilizing the viewing transformation $\boldsymbol{W}$ and the 3D covariance matrix $\boldsymbol{\Sigma}$, the projected 2D covariance matrix $\boldsymbol{\Sigma}'$ is computed through $\boldsymbol{\Sigma}' = \boldsymbol{J}\boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{W}^\top\boldsymbol{J}^\top$. Additionally,

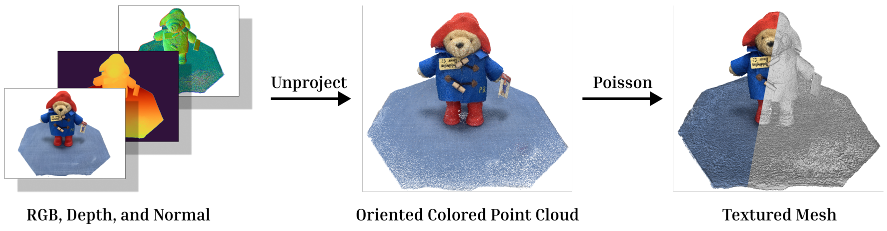

Figure 2: Poisson Mesh Extraction.

we can use the same transformation to compute $\boldsymbol{\mu}' \in \mathbb{R}^2$ in 2D projected space. Given the position of a pixel $\boldsymbol{x} \in \mathbb{R}^2$, 3D Gaussian splitting can be formed as follows:

$$\mathcal{G}_i(\boldsymbol{x}) := \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}'_i)^\top \boldsymbol{\Sigma}'^{-1}_i(\boldsymbol{x}-\boldsymbol{\mu}'_i)\right). \tag{1}$$

**Rendering:** Upon receiving the position of a pixel $\boldsymbol{x}$, the distances to all overlapping Gaussians are computed using the viewing transformation $\boldsymbol{W}$, thereby generating a sorted list of Gaussians $\mathcal{N} := \{G_1, ..., G_N\}$. Subsequently, alpha compositing is employed to render accumulated weight for each pixel:

$$w_i(\boldsymbol{x}) = \alpha_i \mathcal{G}_i(\boldsymbol{x}) \prod_{j=1}^{i-1}(1-\alpha_j \mathcal{G}_j(\boldsymbol{x})). \tag{2}$$

Using the above weight function, several key maps can be derived for each pixel:

1. RGB Color Map:

$$\boldsymbol{C}(\boldsymbol{x}) = \sum_{i=1}^{N} c_i w_i(\boldsymbol{x}) \tag{3}$$

accumulates the RGB colors $c_i$, each weighted by the respective $w_i(\boldsymbol{x})$, to produce the final color output for each pixel.

2. Accumulation Map:

$$\boldsymbol{A}(\boldsymbol{x}) = \sum_{i=1}^{N} w_i(\boldsymbol{x}), \tag{4}$$

which aggregates the computed weights across all Gaussians.

3. Mean (Expected) Depth Map:

$$\boldsymbol{D}_{mean}(\boldsymbol{x}) = \sum_{i=1}^{N} z_i w_i(\boldsymbol{x}), \tag{5}$$

where $z_i$ represents the depth associated with each Gaussian, weighted by $w_i(\boldsymbol{x})$.

4. Median Depth Map:

$$\boldsymbol{D}_{median}(\boldsymbol{x}) = z_i \quad \text{where} \quad i = \min_i \left\{ \prod_{j=1}^{i-1}(1-\alpha_j \mathcal{G}_j(\boldsymbol{x})) > 0.5 \right\} \tag{6}$$

calculates the median depth by identifying the first Gaussian for which the Transmittance value exceeds 0.5.

**Depth Map Unprojection:** Given a depth map $\boldsymbol{D}'$ of size $H \times W$, where $(i, j)$ are pixel coordinates and $d_{i,j}$ is the depth at pixel $(i, j)$, the unprojecting steps are as follows:

1. **Normalization:** The pixel coordinates are normalized to the range $[-1, 1]$ using $x_{\text{norm}} = \frac{2j}{W-1} - 1$ and $y_{\text{norm}} = \frac{2i}{H-1} - 1$.
2. **Homogeneous Coordinates in Camera Space:** The normalized coordinates are then transformed into homogeneous camera space coordinates $\mathbf{p}_{\text{camera}} = [x_{\text{norm}}, y_{\text{norm}}, d_{i,j}]^T$.
3. **Depth Scaling:** Using elements $f_1$ and $f_2$ from the camera projection matrix $\mathbf{K}$, the depth values are scaled as $s_{d_{i,j}} = \frac{f_1 \cdot d_{i,j} + f_2}{d_{i,j}}$. The adjusted camera space coordinates are set to $\mathbf{p}'_{\text{camera}} = [x_{\text{norm}}, y_{\text{norm}}, s_{d_{i,j}}]^T$.
4. **World Space Transformation:** The transformed camera space coordinates are then multiplied by the inverse of the full projection transform matrix $\mathbf{T}$, resulting in $\mathbf{p}_{\text{world}} = \mathbf{T} \times [x_{\text{norm}}, y_{\text{norm}}, s_{d_{i,j}}, 1]^T$.
5. **Discarding Homogeneous Coordinate:** Finally, to obtain Cartesian coordinates, the homogeneous coordinate is discarded: $\mathbf{p}'_{\text{world}} = \frac{\mathbf{p}_{\text{world}}[0:3]}{\mathbf{p}_{\text{world}}[3]}$. This results in $\boldsymbol{D} = \mathbf{p}'_{\text{world}}$ being the 3D coordinates in world space for each pixel.

**Normal Map Calculation:** Given an unprojected depth map, $\boldsymbol{D} \in \mathbb{R}^{H \times W \times 3}$, we can output the corresponding normal map using the cross product of the depth map's gradients:

$$\boldsymbol{N} = \frac{\nabla_x \boldsymbol{D} \times \nabla_y \boldsymbol{D}}{||\nabla_x \boldsymbol{D} \times \nabla_y \boldsymbol{D}||} \tag{7}$$

## E Additional Results

We provide detailed per-scene metrics for the Mip-NeRF360 and Tank & Temple datasets in Table 1. Additionally, we offer further insights through 2D rendering comparisons in Figure 3 and 3D mesh comparisons in Figure 4.

| | | Bicycle | Flowers | Garden | Stump | Treehill | Room | Counter | Kitchen | Bonsai | Truck | Train | Mean |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PSNR | 3DGS | 25.10 | 21.52 | 27.18 | 26.49 | 22.37 | 31.22 | 28.96 | 30.98 | 32.18 | 25.39 | 22.02 | 26.67 |
| | SuGaR | 23.13 | 19.67 | 25.30 | 24.23 | 21.44 | 29.85 | 27.53 | 29.33 | 30.47 | 22.69 | 20.47 | 24.92 |
| | Ours | 25.33 | 21.71 | 27.44 | 26.58 | 22.11 | 31.30 | 28.97 | 30.88 | 32.15 | 25.47 | 21.93 | 26.72 |
| SSIM | 3DGS | 0.763 | 0.603 | 0.860 | 0.763 | 0.626 | 0.916 | 0.905 | 0.923 | 0.939 | 0.878 | 0.812 | 0.817 |
| | SuGaR | 0.663 | 0.514 | 0.793 | 0.669 | 0.558 | 0.901 | 0.882 | 0.892 | 0.928 | 0.827 | 0.762 | 0.763 |
| | Ours | 0.772 | 0.611 | 0.865 | 0.774 | 0.633 | 0.918 | 0.906 | 0.925 | 0.938 | 0.880 | 0.817 | 0.822 |
| LPIPS | 3DGS | 0.205 | 0.332 | 0.107 | 0.213 | 0.326 | 0.219 | 0.200 | 0.127 | 0.204 | 0.148 | 0.208 | 0.208 |
| | SuGaR | 0.307 | 0.378 | 0.182 | 0.307 | 0.408 | 0.2395 | 0.222 | 0.167 | 0.211 | 0.175 | 0.259 | 0.260 |
| | Ours | 0.203 | 0.325 | 0.104 | 0.202 | 0.317 | 0.222 | 0.202 | 0.127 | 0.202 | 0.133 | 0.200 | 0.203 |

Table 1: PSNR$^\uparrow$, SSIM$^\uparrow$, LPIPS$^\downarrow$ metrics for Mip-NeRF360 and Tank&Temple datasets

In Figure 3, the RGB rendering results of our AtomGS show enhanced detail compared to those of Sugar. This is evident in the regions observed on the tree trunk in the "stump" scene and the slender, curly hay near the dried grass ornament in the "garden" scene, as highlighted in the magnified areas. While AtomGS's RGB renderings appear visually similar to those of 3DGS, the normal maps reveal that AtomGS better preserves geometry accuracy, such as the tree trunk in the "stump" scene and both the ground beneath the table and the surface of the soccer ball in the "garden" scene.

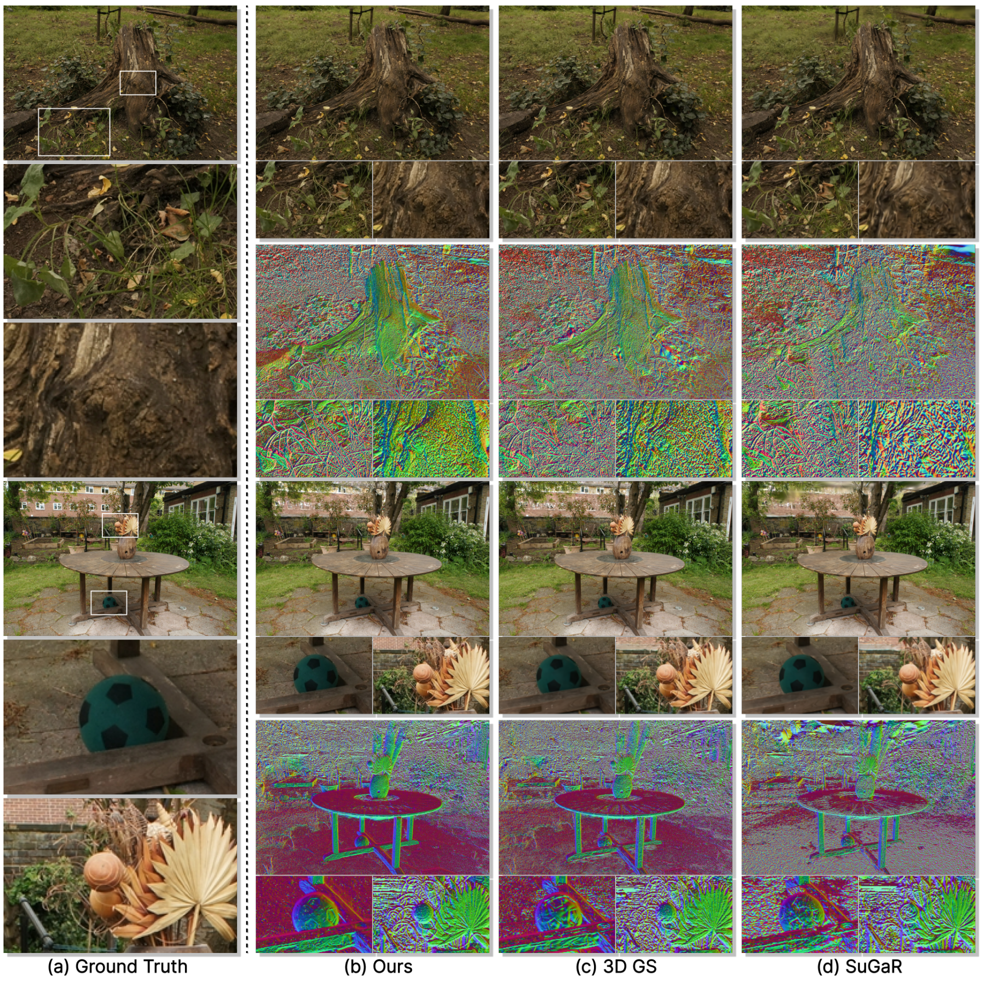

Figure 3: Radiance Field Comparison on the Mip-NeRF360 Dataset.

In Figure 4, Neus, which uses Signed Distance Functions (SDF), produces the smoothest surfaces. However it sometimes sacrifices sharp features, leading to overly smoothed surfaces. SuGaR attempts to convert every Gaussian into 2D ellipsoidal disks, resulting in relatively smooth surfaces. However, the disks do not always align perfectly with the surfaces, creating noticeable disk-shaped artifacts and sometimes overfitting the background. In contrast, AtomGS achieves smooth surfaces while retaining detailed geometries.

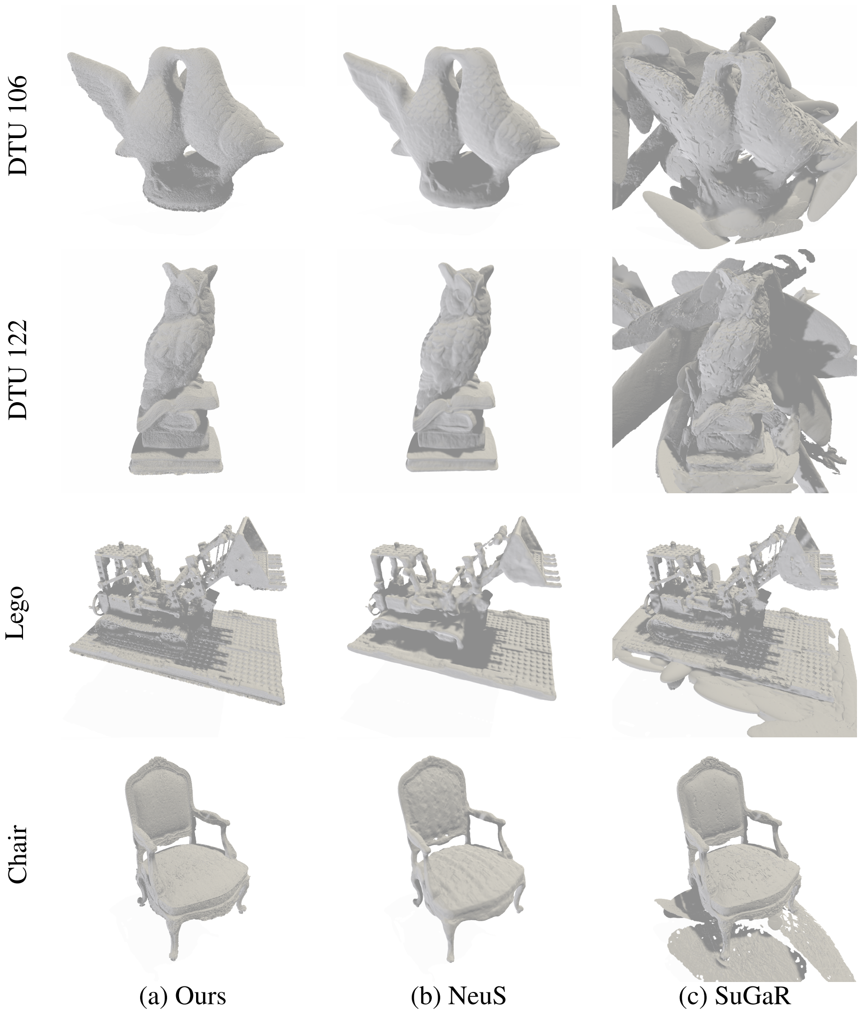

Figure 4: Mesh Comparison on the DTU and NeRF Synthetic Datasets [1, 4]

## References

[1] Henrik Aanæs, Rasmus Ramsbøl Jensen, George Vogiatzis, Engin Tola, and Anders Bjorholm Dahl. Large-scale data for multiple-view stereopsis. *International Journal of Computer Vision*, pages 1–16, 2016.

[2] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[3] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42(4):1–14, 2023.

[4] 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. *Communications of the ACM*, 65(1):99–106, 2021.