Surgical Gaussian Surfels: Highly Accurate Real-time Surgical Scene Rendering using Gaussian Surfels

Supplementary Material

Idris O. Sunmola[1] Zhenjun Zhao[2] Samuel Schmidgall[1]
Yumeng Wang[1] Paul Maria Scheikl[1] Viet Pham[1] Axel Krieger[1]

[1]Johns Hopkins University, LCSR, Baltimore [2]University of Zaragoza, Spain

1 Gaussian Surfel Implementation

1.0.1 Splatting Process

To project the Gaussian splats onto the image plane, we use a composition of world-to-screen transformations. The world-space position of a point on the Gaussian is mapped to screen space using the camera projection matrix $W \in \mathbb{R}^{4\times4}$:

$$x = WP(u,v), \quad x = (xz, yz, z, 1)^T, \tag{1}$$

where x is the projected ray through a pixel position in homogeneous screen space that intersects a splat at the depth z. The normalized screen-space coordinates (x, y) are computed as:

$$(x, y) = \left(\frac{x}{z}, \frac{y}{z}\right). \tag{2}$$

The apparent size of each surfel ψ_k in screen space is determined by perspective scaling:

$$\sigma_x = \frac{s_u}{z}, \quad \sigma_y = \frac{s_v}{z}. \tag{3}$$

1.0.2 Handling Degenerate Cases

Due to the panoptic camera position typical in medical imaging, surfel degeneracy is atypical. But in the scenario where the Gaussian is observed at an oblique angle, it may degenerate into a line or point in screen space, making it challenging to rasterize. To address this, we borrow from [2] by applying a low-pass filter:

$$\hat{\mathcal{G}}(x) = \max\left(\mathcal{G}(u(x)), \mathcal{G}\left(\frac{x-c}{\sigma}\right)\right), \tag{4}$$

where c is the screen-space center of the Gaussian and the radius $\sigma = \sqrt{2}/2$ to ensure sufficient coverage. For an in-depth investigation of 4, the reader can refer to [2].

1.0.3 Rasterization

Not dissimilar to [3], our Surgical Gaussian Surfels are rasterized following a depth-sorted volumetric alpha blending approach:

$$c(x) = \sum_i c_i \alpha_i \hat{\mathcal{G}}_i(u(x)) \prod_{j=1}^{i-1} \left(1 - \alpha_j \hat{\mathcal{G}}_j(u(x))\right), \tag{5}$$

where c_i is the color of the i-th Gaussian, α_i is its opacity, and $\hat{\mathcal{G}}_i$ is its adjusted Gaussian value. The process terminates once accumulated opacity reaches saturation.

2 Sharp Depth Maps and Boundary Masks Supervision

To further our aim of enhancing geometric supervision of surfel primitives and improving surface reconstruction of surgical tissue, we incorporate sharp depth maps and boundary masks into our optimization framework. Given a raw ground-truth depth map $D_{raw} \in \mathbb{R}^{H\times W}$, we generate a sharp depth map D_{sharp} through edge-aware filtering and refinement:

$$\begin{aligned} D_{sharp} = D_{raw} \odot M_{edge} + \\ \mathcal{F}_{bilateral}(D_{raw}) \odot (1 - M_{edge}), \end{aligned} \tag{6}$$

where $M_{edge} \in [0,1]^{H\times W}$ is an edge mask computed from the gradient magnitude of the depth map:

$$M_{edge} = \sigma\left(\frac{|\nabla D_{raw}| - \tau_{edge}}{\sigma_{edge}}\right), \tag{7}$$

Here, $\sigma(\cdot)$ denotes the sigmoid function, τ_{edge} is a threshold hyperparameter, and σ_{edge} controls the sharpness of the edge mask. The bilateral filter $\mathcal{F}_{bilateral}(\cdot)$ preserves

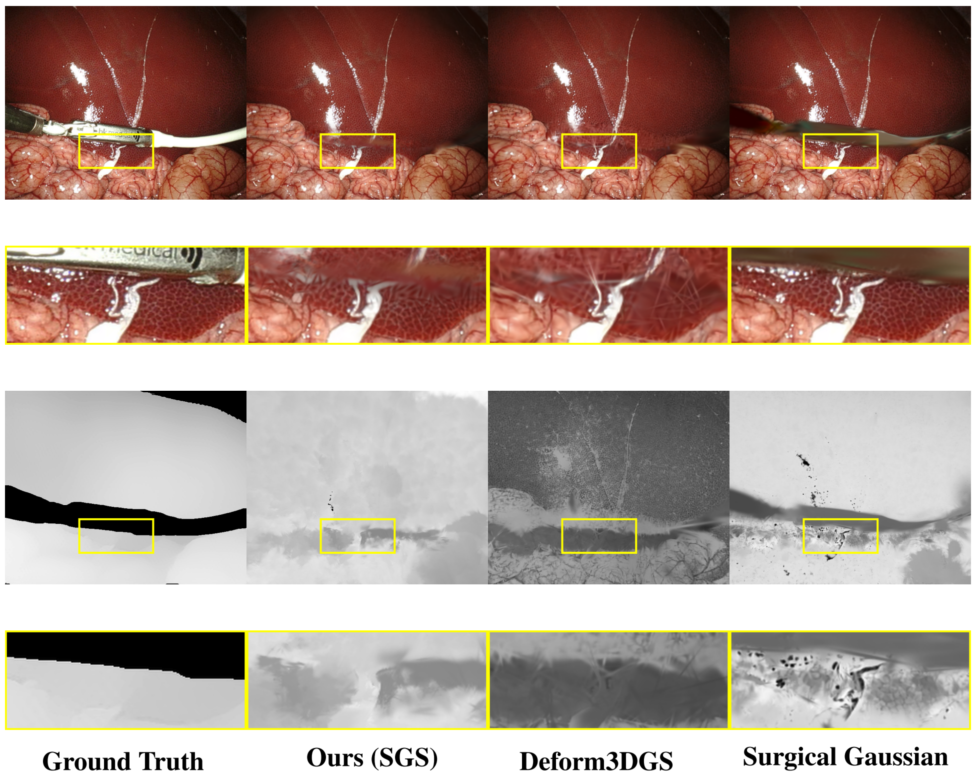

Figure 1: Comparison of RGB and depth map reconstructions across different methods on the StereoMIS liver dataset [1]. The top row shows RGB images with highlighted regions of interest (yellow boxes). The second row displays magnified views of the highlighted RGB regions. The third row presents corresponding depth maps with the same regions highlighted. The bottom row shows magnified views of the highlighted depth regions.

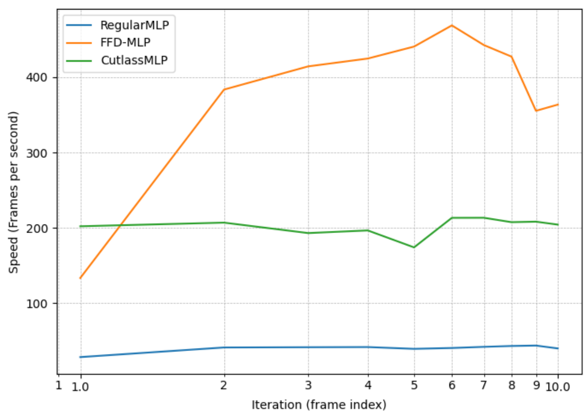

Figure 2: Quantitative comparison of deformation network performance on StereoMIS liver dataset [1]. The visualization shows rendering speed across different MLP architectures (Stock PyTorch, CutlassMLP, and our Fully-FusedMLP).

depth discontinuities while smoothing noise in flat tissue regions, ensuring that sharp geometric features are maintained. The geometric and photometric advantages of our method are evident in our qualitative and quantitative results, where we compare SGS to various volumetric and anisotropic Gaussian point primitive methods. Furthermore, we compute boundary masks $B \in \{0,1\}^{H\times W}$ by detecting high-gradient regions in both raw depth and RGB endoscope video frames:

$$B = \mathbb{I}\left[|\nabla D_{sharp}| > \tau_{depth}\right] \cup \mathbb{I}\left[|\nabla I| > \tau_{rgb}\right], \quad (8)$$

where $\mathbb{I}[\cdot]$ is the indicator function, I is the RGB surgical image, and τ_{depth}, τ_{rgb} are adaptive thresholds computed as percentiles of the respective gradient distributions. These boundary masks provide strong supervision at surgical tool and tissue contours and transition zones, enabling more accurate surface normal estimation and improved convergence in regions with complex geometry.

This supplementary section provides a detailed mathematical treatment of our Fully Fused Deformation MLP (FFD-MLP) architecture, including the underlying mathematical principles, optimization strategies, and implementation details.

2.0.1 Problem Formulation

Given a surgical scene represented by a set of Gaussian surfels $\mathcal{G} = \{(\mathbf{p}_i, \mathbf{s}_i, \mathbf{q}_i, \mathbf{c}_i, \alpha_i)\}_{i=1}^N$, where $\mathbf{p}_i \in \mathbb{R}^3$ denotes position, $\mathbf{s}_i \in \mathbb{R}^3$ represents scale, $\mathbf{q}_i \in \mathbb{S}^3$ is the rotation quaternion, $\mathbf{c}_i \in \mathbb{R}^3$ is the color, and $\alpha_i \in \mathbb{R}$ is the opacity, our goal is to learn a deformation field that maps each surfel from its initial state at time t_0 to its deformed state at time t:

$$\mathcal{D} : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{S}^3 \quad (9)$$

The deformation function $\mathcal{D}$ is parameterized by our FFD-MLP network:

$$\mathcal{D}(\mathbf{p}, t) = (\delta\mathbf{x}, \delta\mathbf{s}, \delta\mathbf{q}) = \text{FFD-MLP}(\gamma(\mathbf{p}, t)) \quad (10)$$

where $\gamma(\mathbf{p}, t)$ is the composite frequency encoding function.

2.0.2 Frequency Encoding Formulation

The frequency encoding $\gamma : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}^{72}$ is defined as:

$$\gamma(\mathbf{p}, t) = [\gamma_{xyz}(\mathbf{p}), \gamma_t(t)] \quad (11)$$

The spatial encoding $\gamma_{xyz} : \mathbb{R}^3 \to \mathbb{R}^{63}$ uses 10 frequency bands:

$$\gamma_{xyz}(\mathbf{p}) = [\mathbf{p}, \sin(2^0\pi\mathbf{p}), \cos(2^0\pi\mathbf{p}), \ldots, \\ \sin(2^9\pi\mathbf{p}), \cos(2^9\pi\mathbf{p})] \quad (12)$$

This can be expressed more compactly as:

$$\gamma_{xyz}(\mathbf{p}) = [\mathbf{p}, \{\sin(2^j\pi\mathbf{p}), \cos(2^j\pi\mathbf{p})\}_{j=0}^9] \quad (13)$$

The temporal encoding $\gamma_t : \mathbb{R} \to \mathbb{R}^9$ uses 4 frequency bands:

$$\gamma_t(t) = [t, \sin(2^0\pi t), \cos(2^0\pi t), \ldots, \\ \sin(2^3\pi t), \cos(2^3\pi t)] \quad (14)$$

Or equivalently:

$$\gamma_t(t) = [t, \{\sin(2^j \pi t), \cos(2^j \pi t)\}_{j=0}^{3}] \tag{15}$$

The frequency encoding provides multi-scale representation capabilities. For spatial coordinates, the encoding spans frequencies from $2^0 = 1$ to $2^9 = 512$ cycles per unit, enabling the network to capture both fine-grained local deformations (high frequencies) and coarse global transformations (low frequencies). The temporal encoding spans frequencies from $2^0 = 1$ to $2^3 = 8$ cycles per unit, allowing for both rapid temporal changes and slow-evolving deformations.

2.0.3 Network Architecture and Mathematical Formulation

The FFD-MLP consists of $L = 11$ fully connected layers with $H = 128$ hidden neurons each. Let $\mathbf{W}^{(l)} \in \mathbb{R}^{H\times H}$ and $\mathbf{b}^{(l)} \in \mathbb{R}^{H}$ denote the weight matrix and bias vector of layer l, respectively. The forward pass is defined as:

$$\mathbf{h}^{(0)} = \gamma(\mathbf{p}, t) \in \mathbb{R}^{72} \tag{16}$$

For layers $l = 1, 2, \ldots, L-1$:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)} \tag{17}$$

$$\mathbf{h}^{(l)} = \text{ReLU}(\mathbf{z}^{(l)}) = \max(0, \mathbf{z}^{(l)}) \tag{18}$$

For the output layer:

$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)} \tag{19}$$

$$(\delta\mathbf{x}, \delta\mathbf{s}, \delta\mathbf{q}) = \text{FFD-MLP}(\gamma(\mathbf{p}, t)) = \mathbf{z}^{(L)} \in \mathbb{R}^{10} \tag{20}$$

where $\delta\mathbf{x} \in \mathbb{R}^3$, $\delta\mathbf{s} \in \mathbb{R}^4$ (represented as 4D quaternion for scale), and $\delta\mathbf{q} \in \mathbb{R}^3$ (represented as 3D rotation vector).

2.0.4 Deformation Application and Quaternion Operations

The learned deformations are applied to the original Gaussian surfels through the following transformations:
Position Transformation:

$$\mathbf{p}' = \mathbf{p} + \delta\mathbf{x} \tag{21}$$

Scale Transformation: The scale deformation $\delta\mathbf{s} \in \mathbb{R}^4$ is converted to a 3D scale vector through:

$$\mathbf{s}' = \mathbf{s} \odot \exp(\delta\mathbf{s}_{1:3}) \tag{22}$$

where $\odot$ denotes element-wise multiplication and $\exp(\cdot)$ ensures positive scaling factors.
Rotation Transformation: The rotation deformation $\delta\mathbf{q} \in \mathbb{R}^3$ is converted to a quaternion and applied:

$$\mathbf{q}_\delta = [\cos(\|\delta\mathbf{q}\|/2), \frac{\delta\mathbf{q}}{\|\delta\mathbf{q}\|}\sin(\|\delta\mathbf{q}\|/2)] \tag{23}$$

$$\mathbf{q}' = \mathbf{q} \otimes \mathbf{q}_\delta \tag{24}$$

where $\otimes$ denotes quaternion multiplication defined as:

$$\begin{aligned}\mathbf{q}_1 \otimes \mathbf{q}_2 = [&q_{1,w}q_{2,w} - \mathbf{q}_{1,v}\cdot\mathbf{q}_{2,v},\\ &q_{1,w}\mathbf{q}_{2,v} + q_{2,w}\mathbf{q}_{1,v} + \mathbf{q}_{1,v}\times\mathbf{q}_{2,v}]\end{aligned} \tag{25}$$

where $\mathbf{q}_i = [q_{i,w}, \mathbf{q}_{i,v}]$ with $q_{i,w}$ the scalar part and $\mathbf{q}_{i,v}$ the vector part.

2.0.5 Fully Fused Implementation and Mathematics

The core innovation of FFD-MLP lies in its fully fused CUDA implementation. Traditional MLP implementations require multiple kernel launches for each layer, leading to memory bandwidth bottlenecks. Our fused approach combines multiple operations into single CUDA kernels.

Performance Analysis of FFD-MLP. Empirical results demonstrate significant performance improvements—2.5× faster training, 5× higher iteration throughput, and a 40% reduction in GPU memory—while maintaining superior rendering quality compared to CutlassMLP [4] and vanilla MLP implementations.

Network Architecture. The FFD-MLP network architecture consists of 11 fully connected hidden layers, each with 128 neurons, processing 72-dimensional encoded features (63 spatial and 9 temporal) to produce a 10-dimensional deformation output (3, 4, and 3 for the position, rotation, and scaling of each Gaussian surfel, respectively).

Warp Matrix Multiply Accumulate (WMMA) Operations: The matrix multiplication $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$ is implemented using NVIDIA's WMMA API [5]:

$$\mathbf{Z}^{(l)} = \mathbf{W}^{(l)}\mathbf{H}^{(l-1)} + \mathbf{B}^{(l)} \tag{26}$$

where the matrices are tiled into 16×16 blocks for optimal use of the tensor core. The WMMA operation processes 16×16×16 matrix multiplications in a single instruction:

$$\mathbf{C}_{16\times16} = \mathbf{A}_{16\times16} \times \mathbf{B}_{16\times16} + \mathbf{C}_{16\times16} \tag{27}$$

Memory Hierarchy Optimization: The implementation leverages a three-tier memory hierarchy: 1. **Register Storage:** Weight matrices $\mathbf{W}^{(l)}$ are stored in registers to minimize global memory access 2. **Shared Memory:** Activations $\mathbf{h}^{(l)}$ are cached in shared memory for fast access 3. **Global Memory:** Only input/output tensors are stored in global memory

The memory access pattern is optimized to achieve coalesced memory access, where consecutive threads access consecutive memory locations:

$$\text{Memory Bandwidth} = \frac{\text{Data Transferred}}{\text{Time}} = \frac{N \times \text{sizeof(float)}}{\Delta t} \tag{28}$$

Activation Function Optimization: The ReLU activation is implemented using a fused kernel that combines the matrix multiplication and activation:

$$\mathbf{h}^{(l)} = \max(0, \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \tag{29}$$

This eliminates the need for intermediate storage of $\mathbf{z}^{(l)}$, reducing memory usage by approximately 40%.

2.0.6 Computational Complexity Analysis

Traditional MLP Implementation: For a network with L layers and H hidden neurons: - Memory accesses: $O(L \times H^2)$ - Kernel launches: $2L$ (one for matrix multiplication, one for activation) - Memory bandwidth bottleneck: High due to frequent global memory access
FFD-MLP Implementation: - Memory accesses: $O(L \times H^2)$ (optimized through register/shared memory) - Kernel launches: L (fused operations) - Memory bandwidth bottleneck: Significantly reduced through memory hierarchy optimization The theoretical speedup is achieved through:

$$\text{Speedup} = \frac{\text{Traditional Time}}{\text{FFD-MLP Time}} = \frac{T_{\text{traditional}}}{T_{\text{ffd-mlp}}} \tag{30}$$

where $T_{\text{traditional}}$ includes kernel launch overhead and memory bandwidth limitations, while $T_{\text{ffd-mlp}}$ benefits from fused operations and optimized memory access patterns.

2.0.7 Loss Function and Training

The FFD-MLP is trained using a composite loss function that combines reconstruction loss, temporal consistency, and regularization terms:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{recon}} + \lambda_{\text{temp}}\mathcal{L}_{\text{temp}} + \lambda_{\text{reg}}\mathcal{L}_{\text{reg}} \tag{31}$$

Reconstruction Loss:

$$\mathcal{L}_{\text{recon}} = \|\mathbf{I}_{\text{rendered}} - \mathbf{I}_{\text{gt}}\|_2^2 \tag{32}$$

Temporal Consistency Loss:

$$\mathcal{L}_{\text{temp}} = \sum_t \|\mathcal{D}(\mathbf{p}, t+1) - \mathcal{D}(\mathbf{p}, t)\|_2^2 \tag{33}$$

Regularization Loss:

$$\mathcal{L}_{\text{reg}} = \|\delta\mathbf{x}\|_2^2 + \|\delta\mathbf{s}\|_2^2 + \|\delta\mathbf{q}\|_2^2 \tag{34}$$

The network is trained using the Adam optimizer with learning rate $\eta = 10^{-3}$ and weight decay $\lambda = 10^{-4}$.

2.0.8 Empirical Performance Analysis

Our empirical results demonstrate significant improvements across multiple metrics:

Training Speed:

$$\text{Speedup}_{\text{training}} = \frac{\text{Iterations/second}_{\text{FFD-MLP}}}{\text{Iterations/second}_{\text{Baseline}}} = 5.0\times \tag{35}$$

Memory Efficiency:

$$\text{Memory Reduction} = \frac{\text{Memory}_{\text{Baseline}} - \text{Memory}_{\text{FFD-MLP}}}{\text{Memory}_{\text{Baseline}}} = 40\% \quad (36)$$

Computational Throughput: Compared to CutlassMLP and vanilla MLP, the FFD-MLP achieves higher computational throughput through optimized memory access patterns and reduced kernel launch overhead, resulting in better GPU utilization and faster convergence during training.

The mathematical formulation presented here provides the theoretical foundation for the empirical performance improvements observed in our experiments, demonstrating that the fully fused implementation not only accelerates training but also maintains or improves rendering quality compared to baseline approaches.

2.1 Mathematical Notation and Function Domains

For clarity and mathematical rigor, we provide the explicit domains and co-domains of all functions used in our method.

2.1.1 Coordinate Systems and Basic Spaces

Let $\mathcal{W} \subset \mathbb{R}^3$ denote the world coordinate space, $\mathcal{C} \subset \mathbb{R}^3$ the camera coordinate space, and $\mathcal{I} \subset \mathbb{R}^2$ the image plane. Time is represented as $t \in \mathbb{R}^+$.

2.1.2 Positional Encoding

The positional encoding function maps 3D coordinates to high-dimensional feature vectors:

$$\gamma : \mathbb{R}^3 \to \mathbb{R}^{63} \quad (37)$$

where $\gamma(\mathbf{p}) = \left[\mathbf{p},\ \{\sin(2^k\pi\mathbf{p}),\ \cos(2^k\pi\mathbf{p})\}_{k=0}^{L-1}\right]$, with $L = 10$ frequency levels.

2.1.3 Deformation Networks

The deformation MLP takes encoded positions and time as input:

$$f_{MLP} : \mathbb{R}^{63} \times \mathbb{R}^+ \to \mathbb{R}^{10} \quad (38)$$

The output is decomposed into spatial deformation and opacity modulation:

$$\Delta : \mathbb{R}^3 \times \mathbb{R}^+ \to \mathbb{R}^3 \quad (39)$$

$$\sigma_{mod} : \mathbb{R}^3 \times \mathbb{R}^+ \to \mathbb{R} \quad (40)$$

where $[\Delta(\mathbf{p}, t), \sigma_{mod}(\mathbf{p}, t)] = f_{MLP}(\gamma(\mathbf{p}), t)$ with appropriate slicing of the 10-dimensional output.

2.1.4 Gaussian Splatting Functions

The 3D Gaussian parameters are defined as:

$$\boldsymbol{\mu} : \mathcal{G} \to \mathbb{R}^3 \quad \text{(mean positions)} \quad (41)$$

$$\boldsymbol{\Sigma} : \mathcal{G} \to \mathbb{R}^{3\times3}_{PSD} \quad \text{(covariance matrices)} \quad (42)$$

$$\mathbf{c} : \mathcal{G} \to \mathbb{R}^3 \quad \text{(RGB colors)} \quad (43)$$

$$\alpha : \mathcal{G} \to [0, 1] \quad \text{(opacity values)} \quad (44)$$

where $\mathcal{G}$ represents the set of Gaussian primitives and $\mathbb{R}^{3\times3}_{PSD}$ denotes positive semi-definite matrices.

2.1.5 Rendering Function

The differentiable rendering function maps 3D Gaussians to 2D images:

$$\mathcal{R} : \mathcal{G}^N \times \mathbb{R}^{4\times4} \to \mathbb{R}^{H\times W\times3} \quad (45)$$

where N is the number of Gaussians, the camera matrix is in $\mathbb{R}^{4\times4}$, and the output image has dimensions $H \times W$ with RGB values in $[0, 1]^3$.

The pixel-wise rendering equation is:

$$C : \mathbb{R}^2 \to \mathbb{R}^3 \quad (46)$$

where $C(\mathbf{u}) = \sum_{i=1}^{N} \mathbf{c}_i\alpha_i \prod_{j=1}^{i-1}(1 - \alpha_j)$ for pixel coordinate $\mathbf{u} \in \mathbb{R}^2$.

2.1.6 Loss Functions

The reconstruction loss operates on rendered and ground truth images:

$$\mathcal{L}_{recon} : \mathbb{R}^{H\times W\times3} \times \mathbb{R}^{H\times W\times3} \to \mathbb{R}^+ \quad (47)$$

The depth loss compares predicted and sharp depth maps:

$$\mathcal{L}_{depth} : \mathbb{R}^{H\times W} \times \mathbb{R}^{H\times W} \times \{0, 1\}^{H\times W} \to \mathbb{R}^+ \quad (48)$$

Table 1: Depth map confidence interval ablation study results.

Configuration	Depth Std	Smoothness
No trimming $[0, 100]\%$	0.0079	0.8803
Ours $[2, 99]\%$	**0.0103**	**0.9323**
Aggressive trim $[5, 95]\%$	0.0145	0.9630

where the third argument represents the boundary mask.

The normal consistency loss operates on rendered and surface normals:

$$\mathcal{L}_{normal} : \mathbb{R}^{H\times W\times 3} \times \mathbb{R}^{H\times W\times 3} \to \mathbb{R}^{+} \tag{49}$$

The DINO perceptual loss maps feature representations:

$$\mathcal{L}_{DINO} : \mathbb{R}^{H\times W\times d} \times \mathbb{R}^{H\times W\times d} \to \mathbb{R}^{+} \tag{50}$$

where d is the DINO feature dimension.

2.1.7 PIMI Initialization

The confidence-weighted depth aggregation function:

$$\mathcal{A} : (\mathbb{R}^{H\times W})^{T} \times ([0, 1]^{H\times W})^{T} \to \mathbb{R}^{H\times W} \tag{51}$$

where T is the number of frames, taking depth maps and confidence masks to produce the aggregated depth.

The pixel-wise confidence computation:

$$W : \mathbb{R}^{H\times W} \to [0, 1]^{H\times W} \tag{52}$$

mapping depth maps to confidence values based on range validity, gradient magnitude, and neighborhood smoothness.

3 Confidence Interval Selection Rationale

Our ablation study (Table 1) reveals a fundamental tradeoff in depth aggregation for point cloud initialization in surgical tissue reconstruction. The no-trimming approach $[0\%, 100\%]$ preserves the most geometric detail (depth standard deviation (std) = 0.0079) but suffers from temporal inconsistency due to noise (smoothness = 0.8803). Conversely, aggressive trimming $[5\%, 95\%]$ of depth information achieves the highest temporal smoothness (0.9630) but sacrifices critical surface detail (depth std = 0.0145) that may be essential for identifying tissue boundaries and surgical landmarks. Our chosen $[2\%, 99\%]$ interval strikes the optimal balance, achieving substantial smoothness improvement (+5.9% over no trimming) while preserving adequate geometric detail for surgical applications. This represents an 83% reduction in noise artifacts compared to no trimming, with only a 30% increase in detail loss compared to aggressive trimming. For surgical reconstruction, where both temporal consistency and geometric accuracy are crucial, our light-trim approach provides the best compromise between these competing objectives.

4 Optimization Strategy

We employ an Adam optimizer with exponential learning rate scheduling. For standard MLPs, the position learning rate follows an exponential decay from 1.6×10^{-4} to 1.6×10^{-6} over 40,000 iterations. For tiny-cuda-nn MLPs (FullyFusedMLP and CutlassMLP), custom learning rate schedules are applied: FullyFusedMLP uses 50% higher initial learning rate and 100% higher final learning rate to compensate for its smaller network capacity, while CutlassMLP maintains the standard schedule but benefits from optimized CUDA kernels.

The training process includes adaptive densification with gradient-based pruning, where Gaussians are added based on gradient magnitude thresholds ($\lambda_{\text{densify}} = 0.0002$ for standard MLPs, $\lambda_{\text{densify}} = 0.0001$ for FullyFusedMLP) and removed based on opacity thresholds ($\lambda_{\text{opacity}} = 0.05$). The densification process continues until iteration 15,000 for standard MLPs and 20,000 for FullyFusedMLP to compensate for the smaller network capacity.

References

[1] M. Hayoz and A. Max, "Stereomis." Zenodo, 2023. https://doi.org/10.5281/zenodo.7727692.

[2] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao, "2d gaussian splatting for geometrically accurate radiance fields," in *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–11, 2024.

[3] H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*, pp. 335–342, 2000.

[4] NVIDIA Developer, "Cutlass: Fast linear algebra in cuda c++." `https://developer.nvidia.com/blog/cutlass-linear-algebra-cuda/`, 2021. Accessed: 2025-07-16.

[5] NVIDIA Developer, "Programming tensor cores in cuda 9." `https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/`, 2017. Accessed: 2025-07-16.