LEARNING ANIMATION-AWARE 3D GAUSSIANS WITH NATIVE RIGGING

Anonymous authors
Paper under double-blind review

ABSTRACT

3D Gaussian Splatting (3DGS) has achieved remarkable performance in novel-view synthesis, yet directly converting static 3DGS into animatable assets remains challenging. Existing automatic rigging methods primarily target mesh-based representations, while large-scale rigged 3D Gaussian datasets are lacking. Moreover, even after rigging, Gaussians optimized for static rendering can produce artifacts during animation, particularly around joints and in poorly observed regions. To address these challenges, we present a framework that integrates automatic rigging with animation-aware Gaussian generation. Our method predicts skeletal structure and skinning weights directly from static 3DGS, then uses the predicted skeleton to guide Gaussian adaptation. This process refines Gaussian distributions and shapes to improve deformation robustness while preserving the original appearance. To support learning, we introduce a data construction pipeline that produces a large-scale paired dataset of vanilla and animation-aware Gaussian representations, together with skeletons and skinning weights. Extensive experiments demonstrate the effectiveness of our framework in automatic rigging and reducing deformation-induced rendering artifacts.

1 INTRODUCTION

Recently, 3DGS (Kerbl et al., 2023) has advanced rapidly, driven by its high efficiency and photorealistic rendering quality. These advantages have led to its widespread adoption in 3D reconstruction, yielding a wealth of digital assets represented directly as 3D Gaussians. The increasing availability of these assets motivates extending their use beyond static rendering to animation and deformation.

Animating these assets requires skeletal structures and skinning weights that define how they deform. However, existing automatic rigging methods are primarily developed for mesh-based representations, and differences in geometric structure and connectivity between meshes and 3D Gaussians can degrade their performance when directly applied to Gaussian inputs. The lack of large-scale rigged 3D Gaussian datasets further limits learning-based approaches. Consequently, predicting skeletons and skinning weights directly from unstructured 3D Gaussians with diverse skeletal topologies remains challenging.

Beyond automatic rigging, the Gaussian representation itself poses a further challenge to animation quality. Gaussians optimized for static rendering can exhibit severe artifacts under skeletal deformation, including blurring, tearing, holes, and spikes. These artifacts arise from two distinct limitations of static 3DGS. First, large or elongated Gaussians may span regions influenced by different bones. A single ellipsoidal primitive cannot faithfully capture the resulting nonuniform deformation, leading to stretching or spiky artifacts near bending joints, as illustrated in Fig. 1. Second, reconstruction from a static pose is limited by visibility: regions occluded or poorly observed in the training views, such as the underarms, may lack sufficient Gaussian cover-

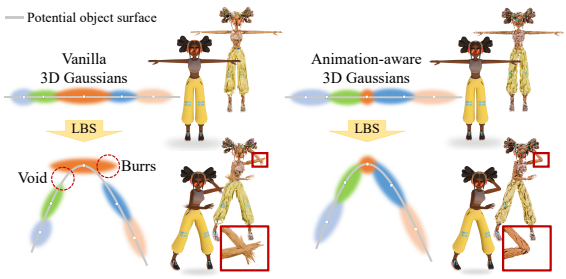

Figure 1: Rendering examples of Vanilla 3D Gaussians and our redistributed animation-aware 3D Gaussians under LBS deformation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

age. These regions can become exposed during articulation, revealing holes or incomplete surfaces that are not apparent in the original pose. Converting static 3DGS into animatable assets therefore requires both effective rigging and a Gaussian representation suited to deformation.

To address these challenges, we propose a framework that integrates automatic rigging with animation-aware Gaussian adaptation, transforming static 3DGS into animatable assets. We first fine-tune a mesh-based rigging model to better accommodate the input distribution of 3D Gaussians, improving skeleton and skinning weight prediction when applied directly to static 3DGS. Building on the predicted rig, we introduce a Gaussian adaptation module that transforms the input into an animation-aware representation by refining its distributions and shapes, improving deformation robustness while preserving the original appearance. To support both rigging and adaptation, we develop a data construction pipeline that produces a large-scale paired dataset of vanilla and animation-aware 3D Gaussian representations, together with their corresponding skeletons and skinning fields. Our framework thus converts static 3DGS into rigged, animation-aware assets from Gaussian inputs alone. Extensive experiments demonstrate its effectiveness in automatic rigging and reducing rendering artifacts during animation.

In summary, our contributions are as follows.

- We propose a framework that combines automatic rigging with Gaussian adaptation to convert static 3DGS into animatable assets. By adapting a mesh-based rigging model to Gaussian inputs, our framework predicts skeletons and skinning weights directly from static 3DGS.
- We introduce a rig-guided Gaussian adaptation module that refines Gaussian distributions and shapes, producing animation-aware representations with improved deformation robustness while preserving the original appearance.
- We develop a data construction pipeline and a *large-scale* paired dataset of vanilla and animation-aware 3D Gaussian representations, together with corresponding skeletons and skinning fields, to support learning both automatic rigging and Gaussian adaptation.

2 Related Work

Deformation-aware 3D Gaussian Splatting. To alleviate deformation-induced artifacts in 3D Gaussian Splatting, various strategies have been explored. VR-GS (Jiang et al., 2024) estimates Gaussian deformation through enclosing tetrahedral cages, but local cage transformations may be insufficient for Gaussians spanning regions with different deformation gradients. GASP (Borycki et al., 2026) deforms Gaussians using associated triangular structures, whose lack of globally consistent topology may introduce discontinuities.

Recent approaches leverage mesh representations and mature mesh deformation techniques to guide Gaussian deformation. SuGaR (Guédon & Lepetit, 2024) attaches flattened Gaussians to reconstructed mesh surfaces, while GaussianAvatars (Qian et al., 2024) binds Gaussians to a parametric FLAME mesh and transforms them with mesh triangles. GaussianMesh (Gao et al., 2024) jointly optimizes Gaussian densification and mesh subdivision for better geometric correspondence. Mani-GS (Gao et al., 2025) introduces adaptive relative rotations and scales to relax strict Gaussian-mesh coupling, while MaGS (Ma et al., 2025) learns mesh-pose-conditioned residuals to compensate for deformation discrepancies. Despite improved robustness, these methods depend on mesh quality and require additional mesh reconstruction and processing.

Overall, static-scene-based methods mainly rely on canonical geometric constraints and lack motion-dependent Gaussian adaptation priors. In contrast, methods that learn such adaptation from dynamic observations rely on video supervision (Shao et al., 2024; Ma et al., 2025; Yao et al., 2025; Wang et al., 2026). Learning to adapt static 3DGS for robust animation under unseen motions, without requiring dynamic observations at inference time, therefore remains an important challenge.

Automatic Rigging. Most existing automatic rigging methods operate on mesh-based representations. Early approaches rely primarily on geometric optimization. Pinocchio (Baran & Popovic, 2007) automatically fits a predefined skeleton to an input mesh and propagates bone influences over the surface, while BBW (Jacobson et al., 2011) computes smooth skinning weights by minimizing a bounded biharmonic energy. More recently, SkinCells (Larionov et al., 2026) models continuous

skinning fields using differentiable Voronoi cells. While effective, these methods mainly rely on geometric cues and hand-designed objectives without learned semantic priors.

Learning-based methods have made substantial progress in both skeleton generation and skinning weight prediction. For skeleton generation, RigNet (Xu et al., 2020) predicts skeletal structures from mesh geometry, while recent methods (Song et al., 2025; Zhang et al., 2025; Song et al., 2026; Liu et al., 2025) adopt autoregressive models to generate diverse skeletons. For skinning, NeuroSkinning (Liu et al., 2019) and RigNet (Xu et al., 2020) employ graph neural networks, while MagicArticulate (Song et al., 2025) adopts a diffusion-based formulation. Puppeteer (Song et al., 2026) incorporates topology-aware joint attention, and UniRig (Zhang et al., 2025) uses bone-point cross-attention for weight prediction. SkinTokens (Zhang et al., 2026) introduces a compact discrete representation of skinning weights, enabling skeleton generation and skinning prediction to be modeled within a unified autoregressive sequence. However, these methods are primarily designed for mesh-based assets and often rely on mesh-specific geometric information, leading to substantial performance degradation when applied directly to 3D Gaussians.

Automatic rigging of unstructured points or 3D Gaussians remains relatively underexplored. MakeIt-Animatable (Guo et al., 2025) supports multiple 3D representations through a particle-based formulation, but mainly targets humanoid and character-like shapes. Other methods (Wan et al., 2024; Yao et al., 2025; Wang et al., 2026) extend rigging to generic articulated objects but rely on motion cues from videos. G-Skin (Yao et al., 2026) directly binds static 3D Gaussians to arbitrary skeletons using a generative prior, but requires per-instance optimization. Therefore, feed-forward automatic rigging of discrete 3D Gaussians with arbitrary skeletal topologies, without dynamic observations or per-instance optimization, remains underexplored.

3 Proposed Method

Given an object represented by a set of 3D Gaussians $\mathcal{G} = \{G_i : \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i, \sigma_i, sh_i\}_{i=1}^{N_G}$, where $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, σ_i, and sh_i denote the center position, covariance matrix, opacity, and spherical harmonic coefficients of the i-th Gaussian, respectively, our goal illustrated in Fig. 2 is to directly predict its skeleton and skinning field and produce an animation-aware Gaussian representation. This representation adapts Gaussian distributions and shapes to support high-quality rendering with reduced artifacts under skeletal deformation. To this end, we first fine-tune an autoregressive model to predict skeletal structures and skinning weights directly from the input 3D Gaussians (Sec. 3.1). We then introduce a rig-guided Gaussian adaptation module that transforms the input into an animation-aware representation (Sec. 3.2). To support learning both automatic rigging and Gaussian adaptation, we construct a large-scale paired dataset of vanilla and animation-aware Gaussian representations with corresponding rigging annotations, as detailed in Sec. 4.

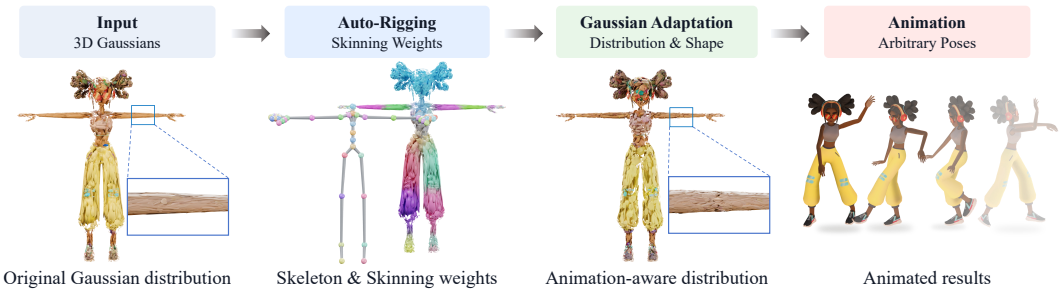

Figure 2: Overview of our pipeline.

3.1 3D Gaussian Rigging

Existing mesh-based rigging methods typically take surface points and normals sampled from meshes as input. Applying these methods to 3D Gaussians via mesh extraction and subsequent rigging transfer can introduce errors in both surface reconstruction and Gaussian–mesh correspondence. We instead fine-tune a rigging model on our paired Gaussian rigging dataset to predict skeletons and skinning weights directly from Gaussian primitives.

Rigging Model Fine-Tuning. We build upon SkinTokens (Zhang et al., 2026), which encodes bone-wise skinning weights into discrete tokens and uses TokenRig to autoregressively predict skeleton and skinning sequences. We retain its token representation and autoregressive formulation while adapting the geometric conditioning and skinning decoder to Gaussian inputs. Specifically, we freeze the token-producing encoder and quantizer of SkinVAE to preserve the discrete token space, and fine-tune the Gaussian-conditioned encoding and decoding pathway. The adapted decoder is then integrated into TokenRig, whose Gaussian geometry branch is fine-tuned with the original skeleton and skinning token prediction objectives.

Hybrid Gaussian Normal Estimation. A key domain gap lies in the geometric input: Gaussian centers are not surface samples and do not provide reliable oriented normals. For each Gaussian, we obtain a covariance normal $\mathbf{n}_i^G$ from its shortest principal axis and a local normal $\mathbf{n}_i^P$ from the smallest-eigenvalue eigenvector of the covariance of neighboring Gaussian centers. After aligning their signs, we blend the two estimates according to the Gaussian anisotropy:

$$\widetilde{\mathbf{n}}_i = \mathrm{Norm}\left(\alpha_i \mathbf{n}_i^G + (1-\alpha_i)\mathbf{n}_i^P\right), \qquad \alpha_i = \mathrm{Sigmoid}\left(\frac{\log(s_{i,\mathrm{mid}}/s_{i,\mathrm{min}}) - \log\tau}{T}\right). \tag{1}$$

Thus, anisotropic Gaussians rely more on their covariance orientation, whereas near-isotropic Gaussians rely more on the local point distribution. We resolve the remaining sign ambiguity using an outward orientation prior and local neighborhood consistency.

Detailed training procedures and loss formulations are provided in Appendix A.4.

3.2 Animation-aware 3D Gaussian Adaptation

Given an input Gaussian representation $\mathcal{G}$ and its predicted rig, our goal is to construct an animation-aware representation $\mathcal{G}^{\mathrm{ani}}$ in canonical space that preserves the input's static appearance while reducing artifacts under skeletal deformation. Regions predominantly influenced by a single bone tend to undergo approximately rigid motion, whereas regions with substantial contributions from multiple bones can exhibit more complex deformation. This motivates a rig-guided two-branch decoder that refines Gaussians in rigid regions while regenerating those in non-rigid regions (Fig. 3).

Using the predicted skinning weights at each input Gaussian center, we classify a Gaussian as approximately rigid if $\max_b(w_{ib}) > \tau_{\mathrm{rigid}}$, where w_{ib} denotes the influence of bone b on Gaussian i. The remaining Gaussians are classified as non-rigid. This partition serves as a proxy for deformation complexity and determines the update freedom permitted by each branch.

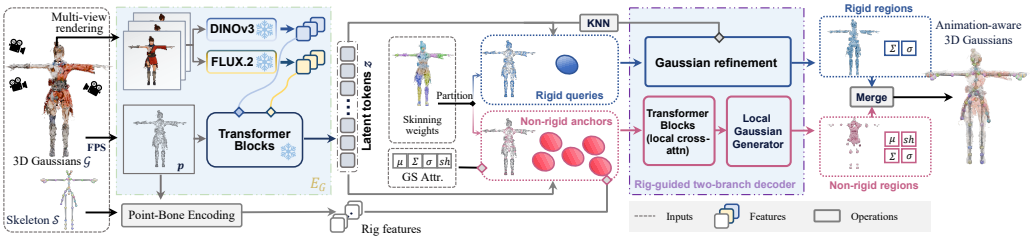

Figure 3: Overview of animation-aware 3D Gaussian adaptation.

Global Context Encoder. Local geometry alone can be ambiguous about part identity and appearance. We therefore use the frozen pretrained encoder from the TripoSplat VAE (Yan et al., 2026) to provide object-level context for both decoder branches. We render the input Gaussians into multi-view images, extract semantic features using DINOv3 (Siméoni et al., 2025) and appearance features using the FLUX.2 VAE (Labs, 2025), and lift these features to 3D points. The encoder aggregates the resulting point features into spatially indexed latent tokens $\mathcal{Z} = \{(\boldsymbol{p}_\ell, \boldsymbol{z}_\ell)\}_{\ell=1}^{N_z} = E_{\mathrm{G}}(\mathcal{G})$, where $\boldsymbol{p}_\ell$ is a farthest-point-sampled encoder query location and $\boldsymbol{z}_\ell$ is its latent feature. Each token can aggregate information from the full conditioning point set, so its spatial index does not restrict it to a purely local observation. These latent locations are distinct from the decoder anchors selected below.

Rigid-Region Refinement Decoder. The refinement branch preserves the input Gaussian structure in rigid regions while allowing restricted attribute updates. It selects existing rigid-region Gaussian

centers as anchors and constructs anchor tokens from their positions, input attributes, and skeleton-relative features.

For an anchor $\boldsymbol{a}_j$, the skeleton-relative feature is $\boldsymbol{r}_j = \Psi(\boldsymbol{a}_j, \mathcal{S})$, where $\mathcal{S}$ denotes the predicted skeleton. We identify the nearest bone segment using the point-to-segment distance with the projection clamped to the segment endpoints. The encoding includes this distance, the normalized projection coordinate, the bone direction, relative position information between the projected point and the anchor, and a learned bone-identity embedding. The feature is projected and added to the anchor's positional and input-attribute embeddings. Both decoder branches use this skeleton-relative conditioning: the skinning-based partition determines where each branch operates, while the geometric features guide its predictions.

The refinement backbone uses global self-attention over the selected anchors, with additional local self-attention and local latent cross-attention every two blocks. The latter retrieves spatially relevant context from $\mathcal{Z}$ and is also used by the generation branch. Let $\mathcal{N}_Z(j)$ denote the nearest latent tokens to anchor j. For one attention head, the matching score is

$$s_{j\ell} = \frac{(R_{\alpha_h}(\boldsymbol{a}_j)\boldsymbol{q}_j)^T (R_{\alpha_h}(\boldsymbol{p}_\ell)\boldsymbol{k}_\ell)}{\sqrt{d_h}} - \gamma \|\boldsymbol{a}_j - \boldsymbol{p}_\ell\|_2^2, \qquad \ell \in \mathcal{N}_Z(j), \tag{2}$$

where $\boldsymbol{q}_j$ and $\boldsymbol{k}_\ell$ are projected anchor queries and latent keys, and d_h is the head dimension. Anchors, latent locations, and bones share a normalized canonical coordinates. $R_{\alpha_h}(\cdot)$ denotes 3D rotary positional encoding with rotation angles scaled by a learned per-head parameter α_h, initialized to zero. The distance penalty favors nearby tokens, while the rotary encoding allows matching to depend on relative displacement beyond distance alone. The scores are softmax-normalized over $\mathcal{N}_Z(j)$, and the weighted latent values are projected and added residually to the anchor token.

Given the contextualized feature $\boldsymbol{h}_j^{\mathrm{ref}}$, the refinement output head predicts bounded residuals:

$$\Delta\boldsymbol{\theta}_j^{\mathrm{ref}} = D_{\mathrm{ref}}\left(\boldsymbol{h}_j^{\mathrm{ref}}\right). \tag{3}$$

These residuals update log-scale, quaternion rotation, and opacity logit, with quaternion renormalization. Gaussian centers and all input spherical-harmonic coefficients remain unchanged. Rigid Gaussians outside the refinement query budget receive zero residual updates and are included unchanged in $\mathcal{G}^{\mathrm{ref}}$.

Non-Rigid-Region Generation Decoder. The generation branch redistributes Gaussians in non-rigid regions. We first use farthest-point sampling on the input Gaussian centers in these regions to select anchors. Each anchor is encoded using its position, input Gaussian attributes, and the skeleton-relative conditioning defined above.

To recover local shape and appearance details that may be lost through latent compression and anchor subsampling, we additionally aggregate neighboring Gaussians from the full input representation. The neighborhood includes both rigid and non-rigid Gaussians, including those not selected as anchors. For each generation anchor $\boldsymbol{a}_j$, we compute

$$\begin{aligned} \boldsymbol{f}_{ji} &= \Phi_G\left(\left[\frac{\boldsymbol{\mu}_i - \boldsymbol{a}_j}{\rho_j}, \frac{\|\boldsymbol{\mu}_i - \boldsymbol{a}_j\|_2}{\rho_j}, \widetilde{\boldsymbol{\theta}}_i, \boldsymbol{c}_i\right]\right), \\ \boldsymbol{g}_j &= \mathrm{Linear}_G\left(\left[\max_{i\in\mathcal{N}_G(j)} \boldsymbol{f}_{ji}, \frac{1}{|\mathcal{N}_G(j)|}\sum_{i\in\mathcal{N}_G(j)} \boldsymbol{f}_{ji}\right]\right), \end{aligned} \tag{4}$$

where $\mathcal{N}_G(j)$ contains spatially neighboring input Gaussians, $\boldsymbol{\mu}_i$ is an input Gaussian center, and ρ_j is the median anchor-to-neighbor distance clamped below by a small positive constant. $\widetilde{\boldsymbol{\theta}}_i$ contains standardized input attributes, and $\boldsymbol{c}_i$ is a two-dimensional one-hot encoding of the rigid/non-rigid assignment. The shared MLP Φ_G encodes each neighbor, while max and mean pooling summarize the neighborhood. The projected context $\boldsymbol{g}_j$ is added to the anchor's positional, input-attribute, and skeleton-relative embeddings.

The resulting anchor tokens are processed by a Transformer generation decoder. Because neighboring Gaussians jointly represent a surface through overlapping contributions, their positions, scales, orientations, and opacities should be predicted in relation to one another. We therefore replace the

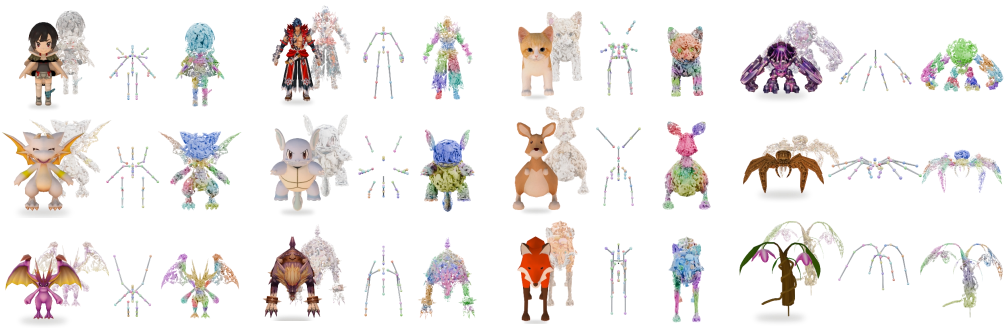

Figure 4: Our constructed dataset. Each case shows the rendering images of animation-aware 3D Gaussians, Gaussian primitives, skeletons, and the corresponding skinning fields, respectively.

global self-attention of the pretrained Gaussian decoder with spatial KNN self-attention in the generation backbone. This lets each anchor incorporate neighboring anchors' evolving features and coordinate local redistribution. Each block combines KNN self-attention, global cross-attention to $\mathcal{Z}$, and a feed-forward layer. The local latent cross-attention defined above is additionally inserted every two blocks, combining spatially relevant latent features with whole-object guidance.

Finally, the generation output head maps each updated anchor feature to the parameters of M child Gaussians:

$$\{\boldsymbol{\theta}_{j,m}^{\text{gen}}\}_{m=1}^{M} = D_{\text{gen}}\left(\boldsymbol{h}_j^{\text{gen}}\right). \tag{5}$$

Each $\boldsymbol{\theta}_{j,m}^{\text{gen}}$ contains an anchor-relative position offset, scale, rotation, opacity, and SH DC coefficients. The generated center is $\boldsymbol{\mu}_{j,m}^{\text{gen}} = \boldsymbol{a}_j + \Delta\boldsymbol{\mu}_{j,m}$. While M is fixed, the anchor count is determined by a sampling rule based on the number of non-rigid input Gaussians. This rule sets the output primitive count, while the network predicts their spatial distribution and attributes without restricting them to small residual updates.

The two branch outputs are assembled into

$$\mathcal{G}^{\text{ani}} = \mathcal{G}^{\text{ref}} \cup \mathcal{G}^{\text{gen}}, \tag{6}$$

where $\mathcal{G}^{\text{ref}}$ contains the rigid-region Gaussians after refinement, including those with zero residual updates, and $\mathcal{G}^{\text{gen}}$ replaces the non-rigid input subset with newly generated primitives. We train both branches using the paired animation-aware targets described in Sec. 4, while keeping the encoder frozen. Detailed feature definitions, residual parameterizations, and attention configurations are provided in Appendix A.5.

4 Dataset Construction

In this section, we describe the data sources and construction pipeline for our dataset of paired vanilla and animation-aware 3D Gaussian representations, together with their corresponding skeletons and skinning fields.

Data Source. We obtain the source data from Articulation-XL 2.0 (Song et al., 2025), which is constructed from the original objects in Objaverse (Deitke et al., 2023b) and Objaverse-XL (Deitke et al., 2023a). Song et al. (2025) filters objects with high-quality skeleton structures and skinning information, processes the corresponding rigging data, and performs geometry normalization. However, Articulation-XL 2.0 does not preserve the object textures. To recover the missing appearance information, we retrieve the original textures from the source datasets and render both static and posed images as ground-truth supervision for our 3D Gaussian construction pipeline. Through texture recovery and quality filtering, we obtain a final set of 20,056 textured rigged meshes for constructing our 3D Gaussian dataset. For each mesh, we render 100 images at a resolution of 1024×1024 from uniformly distributed camera viewpoints and optimize a 3D Gaussian representation per object. The mesh skeleton is directly reused as the skeleton for the Gaussian representation. We next describe how we optimize the skinning field and the corresponding animation-aware 3D Gaussians.

Neural Skinning Field. Skinning weights should be defined as a function of spatial coordinates, independently of how an object is discretized into mesh vertices or Gaussian primitives. A unified spatial field therefore allows the source mesh, vanilla Gaussians, and animation-aware Gaussians to share the same skinning function despite their different geometric discretizations. To this end, we represent and optimize skinning weights using an object-specific neural field. Compared with independently optimizing per-point skinning weights, this shared parameterization encourages spatial coherence and facilitates more stable optimization.

The design of this field is motivated by the spatial locality of skinning. For example, adjusting skinning weights around the head should not alter those around the feet. We therefore adopt a voxel-based CNN with a local spatial inductive bias. To further exploit the sparse spatial distribution of object geometry, we implement the network using efficient sparse 3D convolutional operators from TRELLIS.2 (Xiang et al., 2026). Computation is concentrated on active voxels, enabling a high-resolution representation without allocating a dense feature volume.

Specifically, a lightweight multiscale sparse CNN $F_{\mathbf{\Phi}}$ produces voxel features at a nominal resolution of 512^3. For a query position $\mathbf{x}_i$, we interpolate neighboring voxel features and decode the skinning weights as

$$\mathbf{w}_i = \mathrm{softmax}\left(g_{\boldsymbol{\theta}} \left[\sum_{\mathbf{v}\in\mathcal{N}(\mathbf{x}_i)} a_{\mathbf{v}}(\mathbf{x}_i) F_{\mathbf{\Phi}}(\mathbf{v}),\ \mathbf{q}(\mathbf{x}_i) \right] \right), \tag{7}$$

where $\mathcal{N}(\mathbf{x}_i)$ contains the eight neighboring lattice nodes, $a_{\mathbf{v}}(\mathbf{x}_i)$ denotes the trilinear interpolation coefficient associated with node $\mathbf{v}$, and $\mathbf{q}(\mathbf{x}_i)$ denotes positional features. The MLP $g_{\boldsymbol{\theta}}$ maps the concatenated interpolated voxel features and positional features to per-bone logits. Softmax produces nonnegative skinning weights that sum to one over the bones. This formulation supports queries at both mesh vertices and Gaussian centers, providing a shared skinning field across the three representations.

Gaussian and Skinning Field Optimization. We first initialize the skinning field using mesh-derived skinning weights. For each Gaussian center $\boldsymbol{\mu}_i$, we find its closest mesh triangle and interpolate the vertex skinning weights using the barycentric coordinates. We additionally sample N_G points uniformly over the mesh surface and obtain their target weights $\widetilde{\mathbf{w}}_j^S$ using the same barycentric interpolation. The skinning field is initialized by soft-label cross-entropy loss.

To construct animation-aware targets for training the Gaussian adaptation network, we jointly optimize the vanilla Gaussian representation and the object-specific skinning field. Following the skinning-based rigid/non-rigid partition introduced above, we assign the same branch-specific choices of optimizable attributes for better training. For rigid-region Gaussians, we preserve their centers and appearance coefficients while optimizing scale, rotation, and opacity. This allows primitives extending into articulated regions to adjust their spatial extent without unnecessarily altering well-reconstructed regions. For other Gaussians, we optimize all attributes and enable densification, allowing their positions, shapes, and density to adapt to more complex deformations.

We use images rendered from the LBS-deformed mesh as supervision. Specifically, given a skeletal pose, we animate the corresponding mesh using its ground-truth rig and deform the Gaussians using weights queried from the skinning field. We render both representations under the same camera views and minimize their rendering discrepancy to jointly optimize the Gaussian attributes and skinning field. To keep the optimized representation close to the input and avoid excessive attribute changes that could complicate training of the adaptation network, we additionally apply a shape-confidence-weighted ℓ_2 penalty to the learnable attribute offsets of rigid-region Gaussians. Further details about the optimization strategies and losses are provided in Appendix A.6.

5 Experiments

In this section, we introduce the experimental setup and evaluation metrics, and evaluate the effectiveness of our fine-tuned rigging model, Gaussian adaptation framework, and data construction pipeline. Comprehensive Training and optimization details are provided in Appendix.

Comparisons of Rigging. Our dataset contains 20,056 objects, with 19,456 drawn from the training split of Articulation-XL 2.0 (Song et al., 2025) and 600 from its test split. To ensure a fair

comparison of skeleton and skinning prediction quality, we conduct our evaluations on these 600 test objects.

We compare our method with four existing approaches for skeleton extraction and skinning prediction on vanilla 3D Gaussian representations. Since all methods require sampled points and surface normals as input, we use the same normal estimation strategy described in Eq. 1 across all methods for a fair comparison. We assess skeleton and skinning prediction quality using the geometric error metrics adopted in (Song et al., 2026). The results are reported in Table 1.

Table 1: Geometric errors in skeleton ($\times 0.01$) and skinning prediction. P and R represent Precision and Recall, respectively.

Method	J2J ↓	J2B ↓	B2B ↓	L1 ↓	P ↑	R ↑
RigAnything	8.92	7.66	6.90	0.91	0.53	0.80
UniRig	4.68	3.69	3.20	0.68	0.74	0.78
Puppeteer	11.28	9.93	8.88	0.46	**0.84**	0.74
SkinTokens	5.06	3.93	3.44	0.37	0.82	0.85
Ours	**3.20**	**2.34**	**1.98**	**0.27**	**0.84**	**0.91**

The test objects include motion sequences (52,246 frames), allowing us to render the source meshes under the corresponding poses to obtain ground-truth images for evaluating animation quality. The results are reported in Table 2. These results show that our method consistently outperforms the compared approaches, highlighting the limitations of directly applying mesh-based rigging methods to 3D Gaussian representations and demonstrating the effectiveness of adapting the rigging model to Gaussian inputs. Visual results can be found in the appendix.

Table 2: Comparison of rendering quality.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
RigAnything	24.16	0.9561	0.0599
UniRig	27.07	0.9726	0.0334
Puppeteer	29.22	0.9798	0.0230
SkinTokens	29.69	0.9803	0.0238
Ours	**32.78**	**0.9859**	**0.0172**

Effectiveness of Gaussian Animation. To evaluate the rendering quality during animation of our method, we deform the 3D Gaussians using the predicted skinning weights together with the ground-truth skeletons and poses, and measure the rendering errors against these reference images. We show the numerical comparisons with existing methods in Table 2, and show the visual results in Fig. 5. The show the effectiveness of our methods. More additional qualitative results can be found in Figures 9 and 10 of the appendix, including examples from the test set and assets synthesized by a 3D generative model.

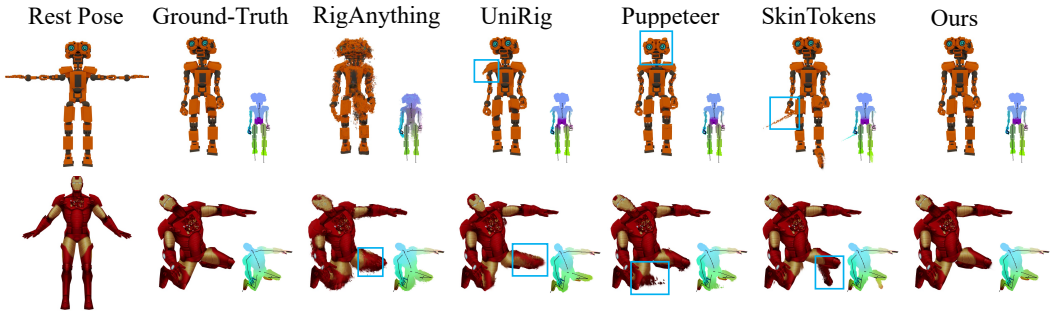

Figure 5: Visual comparisons of Gaussian animation.

Effectiveness of Data Construction. We validate our method on a small dataset from Mixamo (Adobe Systems Inc., 2026) that features diverse appearances and motion sequences with large pose variations. We select 107 objects and two motion sequences, *jumpingjacks* and *MmaKick2*, comprising 82 frames in total, and evaluate their rendering quality from a frontal viewpoint.

We compared our construction method with vanilla 3DGS (Kerbl et al., 2023), Improved-GS (Deng et al., 2026), and Mesh-GS (Gao et al., 2024). Improved-GS employs a Long-Axis Split strategy to improve rendering quality and efficiency, while Mesh-GS binds Gaussians to a mesh surface and constrains their geometry accordingly.

We further consider a simple baseline that improves animation robustness by constraining Gaussian scales. In standard 3DGS, limiting Gaussian sizes naturally encourages densification to maintain rendering quality. Specifically, we augment the original 3DGS objective with a scale regulariza-

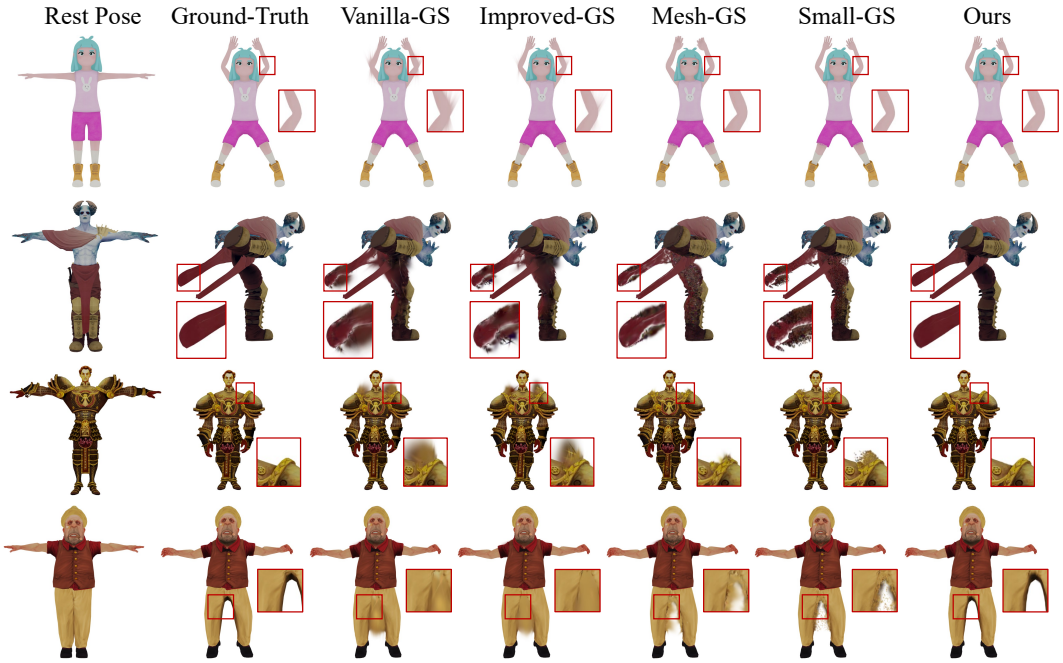

Figure 6: Visual comparisons of our animation-aware 3D Gaussian with vanilla 3D Gaussian and its variants.

Table 3: Comparisons (PSNR, SSIM, LPIPS) of 3D Gaussian rendering quality on the Mixamo dataset (Adobe Systems Inc., 2026). "↑" (resp. ↓) indicates the larger (resp. smaller), the better. The best results are highlighted in bold.

Method	PSNR↑	SSIM ↑	LPIPS ↓ ($\times 10^{-2}$)	# Gaussians
Vanilla-GS (Kerbl et al., 2023)	36.77	0.9938	0.7084	193,735
Improved-GS (Deng et al., 2026)	39.79	0.9955	0.4751	244,703
Mesh-GS (Gao et al., 2024)	34.44	0.9878	0.9886	201,544
Small-GS	39.03	0.9951	0.4466	1,393,646
Ours	**41.86**	**0.9964**	**0.3467**	208,292

tion term $L_{\text{scale}} = \frac{1}{N_G}\sum_{i=1}^{N_G}\sum_{j=1}^{3}\left\|\max(s_i^j - \epsilon, 0)\right\|^2$, where N_G is the number of Gaussians, s_i^j denotes the scale of the i-th Gaussian along the j-th axis, and ϵ is a predefined threshold. To compensate for the restricted Gaussian sizes, we increase the densification frequency by reducing the densification interval from 100 to 10 iterations. We denote this variant as **Small-GS**.

As shown in Table 3 and Fig. 6, our construction method consistently achieves better performance and effectively reduces animation artifacts, including burrs, blurring, and holes, particularly around joints and poorly observed regions.

6 Conclusion

We presented a framework for converting static 3DGS into rigged, animation-aware assets by combining automatic rigging with rig-guided Gaussian adaptation. Our method predicts skeletons and skinning weights directly from Gaussian inputs and refines Gaussian distributions and shapes to reduce deformation artifacts while preserving the original appearance. To support learning, we constructed a large-scale paired dataset of vanilla and animation-aware Gaussian representations with corresponding rigging annotations. Experiments demonstrate the effectiveness of our framework in both automatic rigging and animation quality, providing a practical path toward animating static Gaussian assets.

AI USE STATEMENT

We used generative AI tools to assist with language polishing and to improve the readability of the manuscript. We did not use generative AI tools to generate research results, conduct experiments, or formulate scientific claims. All AI-assisted revisions have been reviewed and verified by the authors. We take responsibility for the final content of this study, including any text generated with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide comprehensive details of the model design and training procedures in the appendix. We will also publicly release our code.

REFERENCES

Adobe Systems Inc. Mixamo. https://www.mixamo.com, 2026. Accessed: 2026-09-26.

Ilya Baran and Jovan Popovic. Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, 26(3):72, 2007.

Piotr Borycki, Weronika Smolak-Dyżewska, Joanna Waczyńska, Marcin Mazur, Sławomir Tadeja, and Przemysław Spurek. Gasp: Gaussian splatting for physics-based simulations. *Computer Vision and Image Understanding*, 265:104699, 2026.

Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. *Advances in Neural Information Processing Systems*, 36:35799–35813, 2023a.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13142–13153, 2023b.

Xiaobin Deng, Changyu Diao, Min Li, Ruohan Yu, and Duanqing Xu. Improving densification in 3d gaussian splatting for high-fidelity rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Findings*, pp. 223–232, June 2026.

Lin Gao, Jie Yang, Bo-Tao Zhang, Jia-Mu Sun, Yu-Jie Yuan, Hongbo Fu, and Yu-Kun Lai. Real-time large-scale deformation of gaussian splatting. *ACM Transactions on Graphics (TOG)*, 43(6): 1–17, 2024.

Xiangjun Gao, Xiaoyu Li, Yiyu Zhuang, Qi Zhang, Wenbo Hu, Chaopeng Zhang, Yao Yao, Ying Shan, and Long Quan. Mani-gs: Gaussian splatting manipulation with triangular mesh. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 21392–21402, 2025.

Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5354–5363, 2024.

Zhiyang Guo, Jinxu Xiang, Kai Ma, Wengang Zhou, Houqiang Li, and Ran Zhang. Make-it-animatable: An efficient framework for authoring animation-ready 3d characters. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 10783–10792, 2025.

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78, 2011.

Ying Jiang, Chang Yu, Tianyi Xie, Xuan Li, Yutao Feng, Huamin Wang, Minchen Li, Henry Lau, Feng Gao, Yin Yang, et al. Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–1, 2024.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.

Black Forest Labs. FLUX.2: Frontier Visual Intelligence. https://bfl.ai/blog/flux-2, 2025.

Egor Larionov, Igor Santesteban, Hsiao-yu Chen, Gene Wei-Chin Lin, Philipp Herholz, Ryan Goldade, Ladislav Kavan, Doug Roble, and Tuur Stuyck. Skincells: Sparse skinning using voronoi cells. In *Computer Graphics Forum*, pp. e70381. Wiley Online Library, 2026.

J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pp. 165–172, USA, 2000. ACM Press/Addison-Wesley Publishing Co. ISBN 1581132085.

Isabella Liu, Zhan Xu, Wang Yifan, Hao Tan, Zexiang Xu, Xiaolong Wang, Hao Su, and Zifan Shi. Riganything: Template-free autoregressive rigging for diverse 3d assets. *ACM Trans. Graph.*, 44 (4):122:1–122:12, 2025. doi: 10.1145/3731149.

Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. Neuroskinning: automatic skin binding for production characters with deep graph networks. *ACM Trans. Graph.*, 38 (4):114:1–114:12, 2019.

Shaojie Ma, Yawei Luo, Wei Yang, and Yi Yang. Mags: Reconstructing and simulating dynamic 3d objects with mesh-adsorbed gaussian splatting. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8745–8755. IEEE, 2025.

Shenhan Qian, Tobias Kirschstein, Liam Schoneveld, Davide Davoli, Simon Giebenhain, and Matthias Nießner. Gaussianavatars: Photorealistic head avatars with rigged 3d gaussians. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20299–20309, 2024.

Zhijing Shao, Zhaolong Wang, Zhuang Li, Duotun Wang, Xiangru Lin, Yu Zhang, Mingming Fan, and Zeyu Wang. Splattingavatar: Realistic real-time human avatars with mesh-embedded gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1606–1616, 2024.

Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, Jamie Tolan, John Brandt, Camille Couprie, Julien Mairal, Hervé Jégou, Patrick Labatut, and Piotr Bojanowski. DINOv3, 2025. URL https://arxiv.org/abs/2508.10104.

Chaoyue Song, Jianfeng Zhang, Xiu Li, Fan Yang, Yiwen Chen, Zhongcong Xu, Jun Hao Liew, Xiaoyang Guo, Fayao Liu, Jiashi Feng, and Guosheng Lin. Magicarticulate: Make your 3d models articulation-ready. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pp. 15998–16007, June 2025.

Chaoyue Song, Xiu Li, Fan Yang, Zhongcong Xu, Jiacheng Wei, Fayao Liu, Jiashi Feng, Guosheng Lin, and Jianfeng Zhang. Puppeteer: Rig and animate your 3d models. *Advances in Neural Information Processing Systems*, 38:72152–72184, 2026.

Diwen Wan, Yuxiang Wang, Ruijie Lu, and Gang Zeng. Template-free articulated gaussian splatting for real-time reposable dynamic view synthesis. *Advances in Neural Information Processing Systems*, 37:62000–62023, 2024.

Jiaxin Wang, Dongxin Lyu, Zeyu Cai, Zhiyang Dou, Cheng Lin, Anpei Chen, and Yuliang Xiu. Gaussianimate: Reconstruct and rig animatable categories with level of dynamics. In *SIGGRAPH Asia 2026 conference papers*, 2026.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 21469–21480, 2025.

Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, and Jiaolong Yang. Native and compact structured latents for 3d generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. Rignet: Neural rigging for articulated characters. *ACM Trans. Graph.*, 39(4), 2020. ISSN 0730-0301.

Runjie Yan, Yan-Pei Cao, Peng Wang, Ding Liang, and Yuan-Chen Guo. Generative 3d gaussians with learned density control, 2026. URL https://arxiv.org/abs/2605.16355.

Yuxin Yao, Zhi Deng, and Junhui Hou. Riggs: Rigging of 3d gaussians for modeling articulated objects in videos. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Yuxin Yao, Kendong Liu, Shiqi Zhou, Jiazhi Xia, and Junhui Hou. G-skin: Learning to bind 3d gaussians with generative visual priors. *arXiv preprint arXiv:2608.01726*, 2026.

Jia-Peng Zhang, Cheng-Feng Pu, Meng-Hao Guo, Yan-Pei Cao, and Shi-Min Hu. One model to rig them all: Diverse skeleton rigging with unirig. *TOG*, 2025.

Jia-Peng Zhang, Cheng-Feng Pu, Meng-Hao Guo, Yan-Pei Cao, and Shi-Min Hu. Skintokens: A learned compact representation for unified autoregressive rigging. *arXiv preprint arXiv:2602.04805*, 2026.

A Appendix

A.1 More Ablation Studies

Fine-tuned Rigging Model. To better preserve the geometric prior learned from meshes, we retain the original SkinTokens (Zhang et al., 2026) shape encoder and autoregressive Transformer whenever possible, and adapt only the geometric input pathway. The main discrepancy between mesh and Gaussian inputs comes from two aspects: Gaussian centers do not necessarily lie on the underlying surface, and Gaussian primitives do not provide consistently oriented surface normals. So we study the influence of these two factors in Tab. 4.

Table 4: The geometric error of skeleton and skinning prediction. J2J, J2B, and B2B are multiplied by 10^2. "↑" (resp. ↓) indicates the larger (resp. smaller), the better.

Input and adaptation	Skeleton			Skinning		
	J2J ↓	J2B ↓	B2B ↓	L1 ↓	Precision ↑	Recall ↑
SkinTokens (mesh)	2.624	1.908	1.651	0.2277	0.8593	0.9361
SkinTokens	5.057	3.929	3.437	0.3720	0.8207	0.8471
PCA normals	3.260	2.391	2.041	0.2746	0.8350	0.9093
Learned normal res.	3.209	2.382	2.074	0.2746	0.8356	0.9084
Learned normal/position res.	3.248	2.422	2.075	**0.2706**	0.8370	**0.9101**
Hybrid normal (ours)	**3.197**	**2.336**	**1.981**	0.2732	**0.8376**	0.9069

We first directly apply the pretrained SkinTokens checkpoint to Gaussian inputs without fine-tuning. Its performance drops considerably compared with the mesh-input result evaluated on the same object IDs, demonstrating a clear input-domain gap. We then fine-tune the Gaussian shape encoder using normals estimated solely by local PCA. Although fine-tuning substantially improves over the zero-shot Gaussian baseline, PCA normals remain unreliable around thin structures, sparse regions, and nearby disconnected parts. Replacing them with our anisotropy-aware hybrid normals further improves the rigging accuracy, showing that Gaussian covariance and local point geometry provide complementary normal cues.

We additionally evaluate two learned geometry adapters. The first predicts a normal residual from the Gaussian position, the estimated normal, and pooled shape features. The second predicts both the normal residual and a bounded position offset, allowing the Gaussian centers to move toward locations that are more compatible with the mesh-pretrained shape encoder. Both residual heads are initialized to zero and trained jointly with the Gaussian shape encoder, while the autoregressive Transformer and SkinVAE remain fixed. All fine-tuned variants otherwise use the same training data, optimization schedule, and evaluation protocol.

Among the Gaussian-input variants, the proposed hybrid-normal model achieves the lowest skeleton errors and the best overall balance between skeleton and skinning accuracy. Learning normal residuals does not provide a consistent improvement over the analytical hybrid estimator. Jointly adjusting normals and positions slightly improves the skinning metrics, but degrades skeleton prediction, suggesting that unrestricted adaptation of Gaussian geometry can weaken the geometric prior inherited from the mesh-based model. We therefore use the hybrid-normal formulation as our default setting.

Optimization of Skinning Field. We also compare renderings of the input Gaussians under test poses using skinning weights interpolated from the mesh at Gaussian centers and those queried from our optimized skinning field. The results in Table 5 show that our optimized skinning weights are better suited to the 3D Gaussian representation than weights directly transferred from the mesh.

Ablations on GS-Generator Decoder . We conduct three ablations of the rig-guided two-branch decoder (Sec. 3.2), using the component names defined in Sec. A.5: *Without spatial KNN self-attention* replaces spatial KNN self-attention with global self-attention in the non-rigid-region generation decoder; *Without local latent cross-attention* removes local latent cross-attention from the generation decoder while retaining its global latent cross-attention and the rigid-region refinement decoder's local latent cross-attention; and *Without skeleton-relative conditioning* removes the skeleton-relative features from both decoder branches while retaining the rigid/non-rigid partition and the

Table 5: Comparisons (PSNR, SSIM, LPIPS) of 3D Gaussian rendering quality on the Mixamo dataset. "↑" (resp. ↓) indicates the larger (resp. smaller), the better. The best results are highlighted in bold.

Method	PSNR ↑	SSIM ↑	LPIPS ↓ ($\times 10^{-2}$)
Mesh-transferred	36.77	0.9930	0.7084
Ours	37.03	0.9939	0.6797

Table 6: Ablations on gs-generator decoder. Comparisons (PSNR, SSIM, LPIPS) of 3D Gaussian rendering quality on the Mixamo dataset. "↑" (resp. ↓) indicates the larger (resp. smaller), the better.

Variant	PSNR ↑	SSIM ↑	LPIPS ↓
Without spatial KNN self-attention	28.3429	0.966834	0.034559
Without local latent cross-attention	30.3917	0.979179	0.02423
Without skeleton-relative conditioning	29.1745	0.977199	0.031276
Ours	32.9484	0.986567	0.016297

skeleton for animation. All variants use matched training settings and are evaluated on the same dataset poses and predicted-gaussian-skinning.

A.2 More Visual Results

We present additional qualitative results. Fig. 7 compares the quality of our extracted skeletons with those produced by existing methods. Fig. 8 presents additional visual comparisons between our constructed animation-aware Gaussians and vanilla Gaussians under unseen test poses. Fig. 9 presents additional visual comparisons between our generated animation-aware Gaussians and vanilla Gaussians under unseen test poses. In addition to evaluating on the existing dataset, we use the image-to-3D model TRELLIS (Xiang et al., 2025) to generate two characters from two input images and apply our method to these examples. The results are shown in Fig. 10. For these examples, we use skeletons extracted by Make-it-Animatable (Guo et al., 2025) because they are compatible with a rich collection of human motion sequences, allowing us to select suitable animations. This choice is motivated by motion compatibility rather than limitations in our skeleton prediction quality.

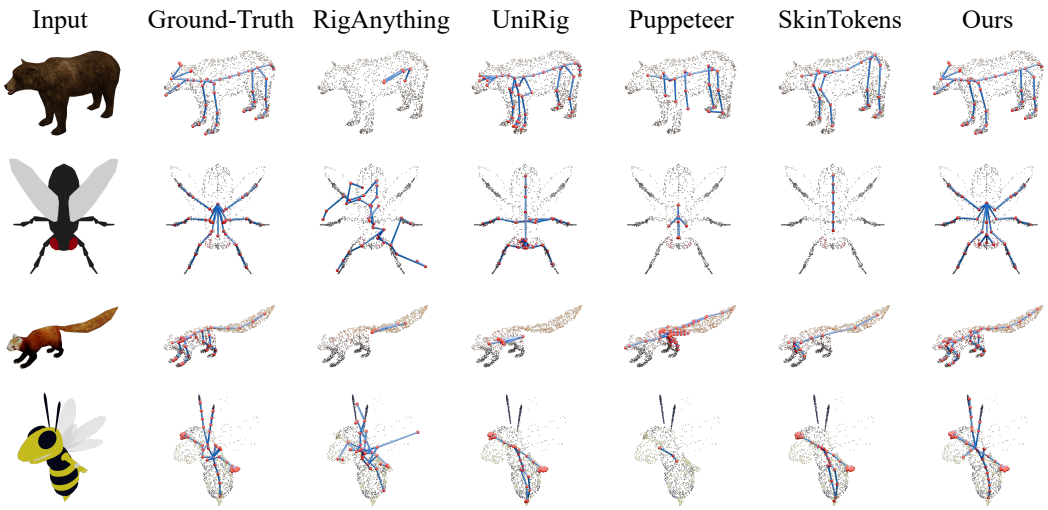

Figure 7: Visual comparisons of skeleton prediction.

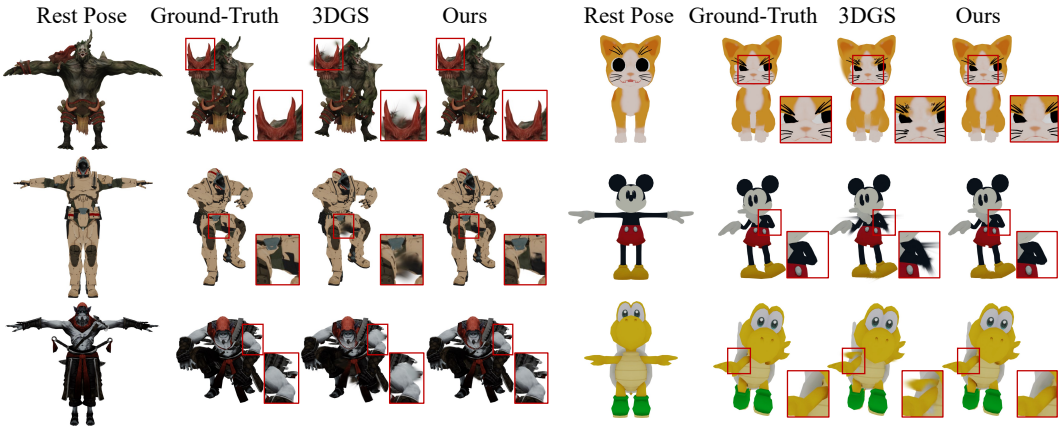

Figure 8: Visual comparisons of vanilla 3D Gaussians and our constructed animation-aware 3D Gaussians.

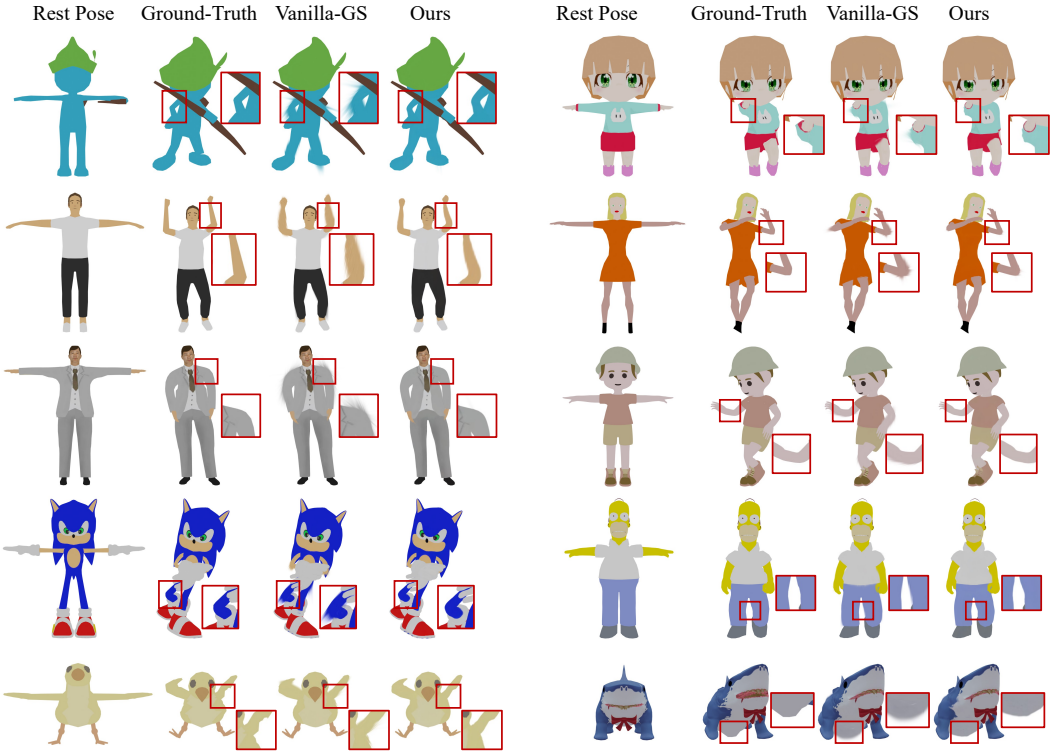

Figure 9: Visual comparisons of vanilla 3D Gaussians and our generated animation-aware 3D Gaussians.

A.3 Details of Skeletal Gaussian Deformation

Skeletal Deformation for 3D Gaussians. We represent the skeleton as $\mathcal{S} = (\mathcal{B}, \mathcal{H})$, where

$$\mathcal{B} = \left\{ \left(\mathbf{J}_b^{\text{head}}, \mathbf{J}_b^{\text{tail}} \right) \right\}_{b=1}^{|\mathcal{B}|} \tag{8}$$

contains the head and tail positions of the bones, and

$$\mathcal{H} = \{(b, h_b)\}_{b=1}^{|\mathcal{B}|} \tag{9}$$

records the parent h_b of each bone b. We set $h_b = \emptyset$ when b is the root. For skeleton representations that store only one joint position per bone, the tail of the parent bone can equivalently be treated as the head of its child bone.

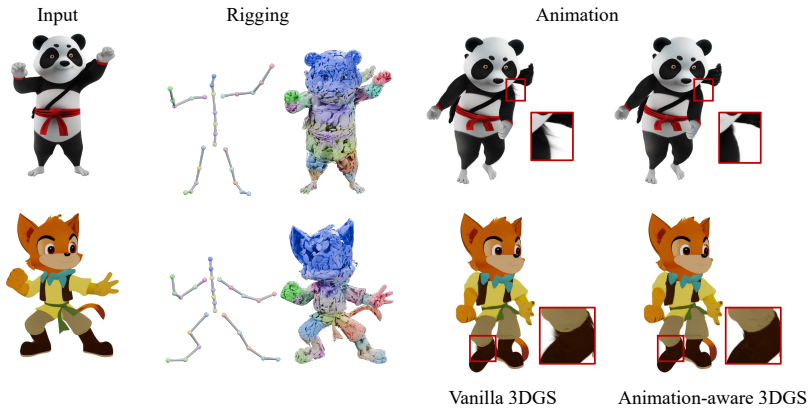

Figure 10: Visualization of our skinning and animation applied to 3D assets generated from a single image.

For each Gaussian center $\boldsymbol{\mu}_i$, we query its normalized skinning weights $\mathbf{w}_i = [w_{i,1}, \ldots, w_{i,|\mathcal{B}|}]$. Given the global bone transformations $\{\mathbf{P}_b\}_{b=1}^{|\mathcal{B}|}$, the Gaussian center is deformed using Linear Blend Skinning (LBS) (Lewis et al., 2000):

$$\widehat{\boldsymbol{\mu}}_i = \mathbf{t}_{\text{root}} + \sum_{b=1}^{|\mathcal{B}|} w_{i,b} \left[\mathbf{P}_b \overline{\boldsymbol{\mu}}_i\right]_{1:3}, \tag{10}$$

where $\overline{\boldsymbol{\mu}}_i = [\boldsymbol{\mu}_i^T, 1]^T$ is the homogeneous Gaussian center and $\mathbf{t}_{\text{root}}$ is the global root translation.

To deform the spatial extent of a Gaussian consistently with its LBS motion, we construct its blended local linear transformation as

$$\mathbf{T}_i = \sum_{b=1}^{|\mathcal{B}|} w_{i,b} \mathbf{Q}_b, \qquad \mathbf{Q}_b = [\mathbf{P}_b]_{1:3,1:3}. \tag{11}$$

Locally treating the skinning weights as constant over the support of each Gaussian, its covariance is transformed according to $\widehat{\boldsymbol{\Sigma}}_i = \mathbf{T}_i \boldsymbol{\Sigma}_i \mathbf{T}_i^T$. Finally, we extract the closest proper rotation $\widetilde{\mathbf{R}}_i \in \mathrm{SO}(3)$ from $\mathbf{T}_i$ and use it to rotate the viewing direction when evaluating the spherical harmonics.

Hierarchical bone transformations. The global bone transformation $\{\mathbf{P}_b\}_{b=1}^{|\mathcal{B}|}$ can be computed by the local rotations of child joints around their parent joints. Let $\mathbf{R}_b$ denote the local rotation of bone b. Its rotation around the bone head is represented by the homogeneous transformation

$$\widehat{\mathbf{P}}_b = \begin{bmatrix} \mathbf{R}_b & \mathbf{J}_b^{\text{head}} - \mathbf{R}_b \mathbf{J}_b^{\text{head}} \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{12}$$

The corresponding global bone transformation is recursively computed as

$$\mathbf{P}_b = \begin{cases} \widehat{\mathbf{P}}_b, & h_b = \emptyset, \\ \mathbf{P}_{h_b} \widehat{\mathbf{P}}_b, & \text{otherwise.} \end{cases} \tag{13}$$

The global root translation is applied separately through $\mathbf{t}_{\text{root}}$.

Given the homogeneous Gaussian center

$$\overline{\boldsymbol{\mu}}_i = \begin{bmatrix} \boldsymbol{\mu}_i \\ 1 \end{bmatrix}, \tag{14}$$

its deformed position is

$$\widehat{\boldsymbol{\mu}}_i = \mathbf{t}_{\text{root}} + \sum_{b=1}^{|\mathcal{B}|} w_{i,b} \left[\mathbf{P}_b \overline{\boldsymbol{\mu}}_i\right]_{1:3}. \tag{15}$$

Local covariance transformation. The skinning field generally varies over space. To obtain a compact Gaussian deformation model, we approximate it as locally constant within the support of each Gaussian. Under this approximation, the local linear part of the LBS deformation is

$$\mathbf{T}_i = \sum_{b=1}^{|\mathcal{B}|} w_{i,b}\mathbf{Q}_b, \qquad \mathbf{Q}_b = [\mathbf{P}_b]_{1:3,1:3}. \tag{16}$$

The Gaussian covariance is therefore transformed using the affine covariance push-forward:

$$\widehat{\mathbf{\Sigma}}_i = \mathbf{T}_i \mathbf{\Sigma}_i \mathbf{T}_i^T. \tag{17}$$

Although each $\mathbf{Q}_b$ is a rotation matrix, their weighted sum $\mathbf{T}_i$ is generally not a rotation. It may contain anisotropic contraction and shear caused by blending transformations from multiple bones.

Rotation extraction for view-dependent appearance. The complete blended transformation is used for covariance deformation, whereas only its rotational component is used to update the view-dependent appearance. Given the singular value decomposition

$$\mathbf{T}_i = \mathbf{U}_i \mathbf{D}_i \mathbf{V}_i^T, \tag{18}$$

we compute the closest proper rotation as

$$\widetilde{\mathbf{R}}_i = \mathbf{U}_i \,\mathrm{diag}\left(1,\, 1,\, \det(\mathbf{U}_i\mathbf{V}_i^T)\right) \mathbf{V}_i^T. \tag{19}$$

The determinant correction guarantees that $\widetilde{\mathbf{R}}_i \in \mathrm{SO}(3)$ and avoids introducing a reflection. For a viewing direction $\mathbf{d}_i$ in the deformed coordinate frame, we evaluate the spherical harmonics using

$$\widehat{\mathbf{d}}_i = \widetilde{\mathbf{R}}_i^T \mathbf{d}_i. \tag{20}$$

This inverse rotation maps the viewing direction to the local orientation in which the Gaussian's spherical-harmonic coefficients are defined.

A.4 Implementation Details of 3D Gaussian Rigging

Gaussian normal estimation. A Gaussian primitive is parameterized by its center $\boldsymbol{\mu}_i$, rotation $\mathbf{R}_i$, and axis scales $(s_{i,1}, s_{i,2}, s_{i,3})$. We take the rotated axis associated with the smallest scale as its covariance-based normal:

$$\mathbf{n}_i^G = \mathbf{R}_i \mathbf{e}_{\arg\min_d s_{i,d}}. \tag{21}$$

Because the shortest axis becomes unreliable for nearly isotropic Gaussians, we additionally estimate a local normal from neighboring Gaussian centers. For the k nearest neighbors $\mathcal{N}_i$, we compute

$$\mathbf{C}_i = \frac{1}{|\mathcal{N}_i| - 1} \sum_{j \in \mathcal{N}_i} (\boldsymbol{\mu}_j - \mathbf{y}_i)(\boldsymbol{\mu}_j - \mathbf{y}_i)^\top, \tag{22}$$

where $\mathbf{v}_i$ is the neighborhood mean. The eigenvector associated with the smallest eigenvalue of $\mathbf{C}_i$ is used as the PCA normal $\mathbf{n}_i^P$. We use $k = 32$ in all experiments.

We first flip $\mathbf{n}_i^P$ when necessary such that $\langle \mathbf{n}_i^P, \mathbf{n}_i^G \rangle \geq 0$. The blending coefficient is computed from the ratio between the middle and smallest Gaussian scales:

$$\alpha_i = \mathrm{Sigmoid}\left(\frac{\log(s_{i,\mathrm{mid}}/s_{i,\min}) - \log 1.5}{0.2}\right). \tag{23}$$

The initial normal is then

$$\widetilde{\mathbf{n}}_i = \mathrm{Norm}\left(\alpha_i \mathbf{n}_i^G + (1 - \alpha_i)\mathbf{n}_i^P\right). \tag{24}$$

Thus, the covariance orientation dominates for clearly anisotropic Gaussians, while local PCA contributes more to nearly isotropic primitives.

Both covariance and PCA normals have an inherent sign ambiguity. We first orient each normal toward the direction from the object center to the Gaussian center. We then perform a conservative sign-consistency update using the eight nearest Gaussian centers. Finally, if the normals are globally oriented inward on average, we flip all normals simultaneously. This procedure provides consistent input normals without requiring ground-truth mesh normals at inference time.

Fixed-token SkinVAE adaptation. The released SkinVAE assigns four discrete tokens to each bone-wise skinning field. Directly re-encoding the training data after modifying the SkinVAE would change the token targets used by TokenRig. We therefore keep the token-producing branch fixed, including the skinning encoder, its latent projection, and the finite scalar quantizer. The original discrete token targets are computed once and remain unchanged throughout training.

We adapt only the Gaussian-conditioned branch and the decoding branch, including the condition encoder, condition projection, post-quantization projection, and decoder. The condition encoder receives $54{,}000$ Gaussian centers and their estimated normals. During training, decoder queries contain both uniformly sampled Gaussians and Gaussians near the influence region of the supervised bone. This query sampling is used only to construct the reconstruction objective and does not require skinning weights at inference time.

For a predicted bone-wise skinning field $\widehat{\mathbf{w}}$ and its target $\mathbf{w}$, the reconstruction objective is

$$\mathcal{L}_{\mathrm{VAE}} = \mathcal{L}_{\mathrm{BCE}} + \mathcal{L}_{\mathrm{Dice}} + 0.1\mathcal{L}_{\mathrm{MSE}}, \tag{25}$$

where

$$\mathcal{L}_{\mathrm{Dice}} = 1 - \frac{2\sum_i \widehat{w}_i w_i + \epsilon}{\sum_i \widehat{w}_i^2 + \sum_i w_i^2 + \epsilon}. \tag{26}$$

The losses are first averaged within each supervised bone and then averaged across objects, preventing objects with more bones from receiving a disproportionately large weight.

We train the SkinVAE adaptation for 10 epochs and use AdamW with a learning rate of 10^{-5} for both the conditional and decoding branches, a weight decay of 0.01, a 5% linear warm-up, and cosine learning-rate decay.

TokenRig adaptation. After SkinVAE adaptation, we replace the original SkinVAE decoder in TokenRig with the adapted decoder while retaining the same token vocabulary and token targets. For each object, the geometric condition is extracted from $54{,}000$ Gaussian centers and their estimated normals. The resulting shape features are projected to the conditioning tokens used by the autoregressive Transformer.

The training sequence follows the original SkinTokens formulation and contains the skeleton tokens followed by four skinning tokens for each bone. We use teacher forcing and compute the skeleton and skinning cross-entropies separately:

$$\mathcal{L}_{\mathrm{AR}} = \frac{1}{2}\left(\mathcal{L}_{\mathrm{skel}}^{\mathrm{CE}} + \mathcal{L}_{\mathrm{skin}}^{\mathrm{CE}}\right). \tag{27}$$

Each term is independently averaged over its valid tokens before the two terms are combined. Consequently, the larger number of skinning tokens does not implicitly dominate the skeleton objective.

For the direct-continuation model used in our main evaluation, we optimize the Gaussian shape encoder and its output projection while keeping the adapted SkinVAE and autoregressive Transformer fixed. We train for 12 epochs with a batch size of four per device on four devices, resulting in an effective batch size of 16 and $2{,}358$ updates per epoch. The shape encoder and output projection use learning rates of 10^{-5} and 2×10^{-5}, respectively. We use AdamW with a weight decay of 0.01, a 5% warm-up followed by cosine decay, and a maximum gradient norm of 1. We apply Gaussian noise with a standard deviation of 0.01 to the input normals and randomly drop 10% of the point normals during training.

Inference. At inference time, TokenRig first autoregressively predicts the skeleton sequence and subsequently predicts four skinning tokens for each generated bone. The adapted SkinVAE decodes these tokens into bone-wise weights on the queried Gaussian centers. We clamp negative numerical values and normalize the weights across all predicted bones so that

$$w_{ij} \geq 0, \qquad \sum_{j=1}^{K} w_{ij} = 1. \tag{28}$$

When inference is performed on a sampled Gaussian subset, weights for the remaining Gaussian primitives are obtained using inverse-distance interpolation from the eight nearest decoded Gaussian centers.

We first adapt the SkinVAE while preserving its discrete latent space. Specifically, we freeze the skinning encoder, latent projection, and finite scalar quantizer that produce the skinning tokens, and optimize only the Gaussian-conditioned encoder, the corresponding latent projections, and the decoder. Both the conditioning points and the queried skinning weights are defined on Gaussian primitives. The adapted SkinVAE is trained using $\mathcal{L}_{\mathrm{VAE}}$ in Eq. 25. Because the token-producing branch remains frozen, the resulting decoder can be inserted into TokenRig without changing the discrete skinning targets. We then adapt TokenRig to Gaussian geometry using the original autoregressive objective $\mathcal{L}_{\mathrm{AR}}$.

A.5 Implementation Details of Gaussian Adaptation

This section supplements Sec. 3.2 with the feature definitions, decoder parameterizations, and training configuration of our Gaussian adaptation model. The model consists of a frozen context encoder, a rigid-region refinement decoder, and a non-rigid-region generation decoder. The rigid/non-rigid partition is computed during preprocessing using the skinning-based criterion described in the main text and supplied to the adaptation model.

A.5.1 Input Representation and Context Encoder

Coordinates and Gaussian attributes. Input Gaussian centers and skeleton endpoints are expressed in the normalized object coordinate system $[-0.5, 0.5]^3$. Before network conditioning, their coordinates are shifted by 0.5 and clipped to $[0, 1]$. Encoder query locations, decoder anchors, and skeleton features therefore use the same coordinate system. Rendering and geometric supervision are evaluated in the original normalized object coordinates.

For each input Gaussian, let $\boldsymbol{\ell}_i = \log \boldsymbol{s}_i$ denote its three log-scales, $\boldsymbol{q}_i$ its unit quaternion, o_i its opacity logit, and $\boldsymbol{f}_i^{\mathrm{DC}}$ its three SH DC coefficients. We construct the 11-dimensional attribute condition

$$\widetilde{\boldsymbol{\theta}}_i = \left[\frac{\boldsymbol{\ell}_i + 7}{2}, \widehat{\boldsymbol{q}}_i, \frac{o_i + 1.4}{2}, \boldsymbol{f}_i^{\mathrm{DC}} + 0.95 \right], \tag{29}$$

where the quaternion is normalized and its sign is chosen such that its scalar component is nonnegative. These fixed affine transformations standardize the attribute channels before their learned projection. Higher-order SH coefficients are not included in this condition; they are preserved separately for the rigid-region output.

Frozen encoder. The encoder receives two sets of spatially aligned point features: 1280-dimensional DINOv3 features and 32-dimensional FLUX.2 VAE features. We sample up to 16,384 conditioning points from each feature set. The encoder uses 16 Transformer blocks with hidden width 1024, 16 attention heads, and an MLP expansion ratio of 4. The two feature streams are incorporated through separate cross-attention operations.

Farthest-point sampling selects 8192 spatial queries from the first conditioning point set. The encoder produces one 16-dimensional latent vector per query, yielding $\mathcal{Z} = \{(\boldsymbol{p}_\ell, \boldsymbol{z}_\ell)\}_{\ell=1}^{8192}$. We use the posterior mean rather than sampling from the latent posterior. The encoder remains frozen throughout decoder training, and no KL-divergence objective is applied in this adaptation stage.

A.5.2 Rigid-Region Refinement Decoder

Skeleton-relative conditioning. Both decoder branches condition their anchor tokens on the nearest bone segment. Let $\boldsymbol{J}_b^{\mathrm{head}}$ and $\boldsymbol{J}_b^{\mathrm{tail}}$ denote the endpoints of bone b, and define $\boldsymbol{v}_b = \boldsymbol{J}_b^{\mathrm{tail}} - \boldsymbol{J}_b^{\mathrm{head}}$. For anchor $\boldsymbol{a}_j$, we compute

$$t_{jb} = \mathrm{clip}_{[0,1]} \left(\frac{(\boldsymbol{a}_j - \boldsymbol{J}_b^{\mathrm{head}})^T \boldsymbol{v}_b}{\max(\|\boldsymbol{v}_b\|_2^2, \epsilon)} \right), \tag{30}$$

$$\boldsymbol{\pi}_{jb} = \boldsymbol{J}_b^{\mathrm{head}} + t_{jb} \boldsymbol{v}_b, \tag{31}$$

$$b_j^\star = \arg\min_b \|\boldsymbol{a}_j - \boldsymbol{\pi}_{jb}\|_2. \tag{32}$$

Writing $\boldsymbol{\delta}_j = \boldsymbol{a}_j - \boldsymbol{\pi}_{jb_j^\star}$ and $d_j = \|\boldsymbol{\delta}_j\|_2$, the refinement branch uses

$$\boldsymbol{r}_j^{\mathrm{ref}} = \left[d_j,\, t_{jb_j^\star},\, \frac{\boldsymbol{v}_{b_j^\star}}{\max(\|\boldsymbol{v}_{b_j^\star}\|_2, \epsilon)},\, \frac{\boldsymbol{\delta}_j}{\max(d_j, \epsilon)},\, \boldsymbol{e}_{b_j^\star}^{\mathrm{ref}} \right]. \tag{33}$$

The generation branch uses the same construction but retains the unnormalized displacement $\boldsymbol{\delta}_j$ in place of its unit direction. Each branch has its own learned 16-dimensional bone embedding, giving a 24-dimensional skeleton-relative feature. The features are projected to the corresponding decoder width and added to the positional and input-attribute embeddings.

Architecture and query budget. The refinement decoder contains six Transformer blocks with hidden width 512 and eight attention heads. Each backbone block applies global self-attention over the selected rigid-region anchors followed by a feed-forward layer. After blocks 2, 4, and 6, we additionally apply local latent cross-attention and local anchor self-attention.

For each object and forward pass, we randomly select at most 40,000 rigid-region Gaussians for refinement. All remaining rigid-region Gaussians receive zero residual updates and remain part of $\mathcal{G}^{\mathrm{ref}}$. Thus, the sampling budget limits decoder queries without removing rigid-region primitives from the assembled output.

Local latent cross-attention. Both decoder branches use local latent cross-attention with bottleneck width 128 and four attention heads. The refinement branch retrieves the 64 nearest latent locations, whereas the generation branch retrieves the 128 nearest. Neighborhood sizes are capped by the number of available tokens. The squared-distance bias uses $\gamma = 50$ in the normalized network coordinates.

For each 32-dimensional attention head, the rotary channels are divided among the three coordinate axes with dimensions 10, 10, and 12. For an axis with d channels, the $d/2$ rotation frequencies are

$$\omega_k = 2^{10k/(d/2-1)}, \qquad k = 0, \ldots, d/2 - 1. \tag{34}$$

A coordinate x produces rotation angles $\alpha_h \pi \omega_k x$, where α_h is a learned per-head parameter. These parameters are initialized to zero when the rotary modules are introduced, so the initial rotations are identities. Each local attention module has independent projections and rotary parameters. Neighbor indices are determined from anchor and latent coordinates and reused across the corresponding attention layers within a forward pass.

Local anchor self-attention. The additional refinement self-attention modules operate over 48 neighboring refinement anchors, excluding the query itself. They use bottleneck width 128 and four heads. In addition to the query–key dot product, they apply a learned per-head positional bias obtained from the relative displacement and distance, normalized by the median neighbor distance. The bias MLP has hidden width 32 with GELU activation. The attended features are projected back to width 512 and added residually to the anchor token. These local modules supplement the global backbone attention.

Residual parameterization. The refinement output head maps each final anchor feature to eight raw values: $\boldsymbol{u}_j^s \in \mathbb{R}^3$, $\boldsymbol{u}_j^q \in \mathbb{R}^4$, and $u_j^o \in \mathbb{R}$. The updated attributes are

$$\boldsymbol{\ell}_j' = \boldsymbol{\ell}_j + 1.5 \tanh(\boldsymbol{u}_j^s), \tag{35}$$

$$\boldsymbol{q}_j' = \mathrm{normalize}\left(\boldsymbol{q}_j + 0.5 \tanh(\boldsymbol{u}_j^q)\right), \tag{36}$$

$$o_j' = o_j + 4.0 \tanh(u_j^o). \tag{37}$$

The physical scales and opacities are $\boldsymbol{s}_j' = \exp(\boldsymbol{\ell}_j')$ and $\alpha_j' = \sigma(o_j')$. Gaussian centers and all input SH coefficients are copied unchanged. In particular, the scale update acts directly in log-scale space without an additional range remapping.

A.5.3 Non-Rigid-Region Generation Decoder

Anchor selection. Let N_{nr} be the number of non-rigid input Gaussians. For $N_{\mathrm{nr}} > 0$, deterministic farthest-point sampling selects

$$N_a = \min\left(N_{\mathrm{nr}}, \max\left(256, \mathrm{round}(N_{\mathrm{nr}}/7)\right), 16384\right) \tag{38}$$

anchors from their centers. The generation branch is skipped when $N_{\text{nr}} = 0$. Each anchor generates $M = 8$ child Gaussians, so the branch produces exactly $8N_a$ primitives. The anchor budget is determined by this sampling rule rather than predicted by the network.

Input Gaussian context. Each generation anchor gathers 16 neighboring Gaussians from the full input representation, excluding itself. The neighborhood may contain both rigid and non-rigid Gaussians. For each neighbor, we concatenate the normalized relative displacement (3 dimensions), normalized distance (1 dimension), standardized Gaussian attributes (11 dimensions), and rigid/non-rigid indicator (2 dimensions). The resulting 17-dimensional edge feature is processed by an MLP with two 128-dimensional layers and GELU activations. Max and mean pooling over neighbors produce a 256-dimensional descriptor, which is linearly projected to width 1024 and added to the anchor token.

Relative displacements and distances are divided by the median anchor-to-neighbor distance, clamped below at 10^{-6}. The neighborhood is constructed from the full input Gaussian set rather than only the sampled anchors, preserving access to details omitted by anchor subsampling.

Transformer architecture. The generation backbone has 16 Transformer blocks with hidden width 1024, 16 attention heads, and an MLP expansion ratio of 4. Each block applies, in order, spatial KNN self-attention, global cross-attention to all 8192 latent tokens, and a feed-forward layer, with residual connections. The self-attention window contains the query anchor and up to 32 neighboring anchors. Attention weights are normalized over this window. Local latent cross-attention is additionally applied after every two backbone blocks, giving eight local latent modules.

The anchor neighborhoods remain fixed within a forward pass, while their token features evolve through the Transformer. Unlike the refinement branch's additional local self-attention, the generation backbone's KNN self-attention does not introduce a separate learned relative-position bias. We use activation checkpointing for the window attention and local latent attention during training.

Generated Gaussian parameterization. The output head maps each 1024-dimensional anchor feature to 120 values, corresponding to 15 values per child: a three-dimensional position parameter, three SH DC coefficients, three scale parameters, four rotation parameters, one opacity parameter, and one offset-scale parameter.

For child m of anchor j, let $\boldsymbol{u}^x_{jm}$ be the raw position parameter and u^r_{jm} the raw offset-scale parameter. Its displacement is

$$r_{jm} = \text{softplus}\left(u^r_{jm} + \text{softplus}^{-1}(0.05)\right), \tag{39}$$

$$\Delta\boldsymbol{\mu}_{jm} = 0.75\, r_{jm} \tanh\left(\boldsymbol{u}^x_{jm} + \boldsymbol{\xi}_m\right), \tag{40}$$

where $\boldsymbol{\xi}_m$ is a fixed child-slot perturbation derived from a Hammersley sequence. These perturbations distinguish the child slots and are retained when transferring compatible decoder weights. The output center is obtained by adding the displacement to the anchor and converting back to object coordinates. The $\tanh$ bounds the displacement for a given r_{jm}, while the positive offset scale remains learnable.

For raw scale, rotation, and opacity outputs $\boldsymbol{u}^s_{jm}$, $\boldsymbol{u}^q_{jm}$, and u^o_{jm}, we use

$$\boldsymbol{s}_{jm} = \sqrt{\text{softplus}\left(\boldsymbol{u}^s_{jm} + \text{softplus}^{-1}(0.004)\right)^2 + (9 \times 10^{-4})^2}, \tag{41}$$

$$\boldsymbol{q}_{jm} = \text{normalize}\left(\boldsymbol{u}^q_{jm} + (1,0,0,0)^T\right), \tag{42}$$

$$\alpha_{jm} = \sigma\left(u^o_{jm} + \text{logit}(0.1)\right). \tag{43}$$

Scale operations are elementwise. The generated appearance uses directly predicted SH DC coefficients. When assembling the two branch outputs, generated Gaussians receive zero higher-order SH coefficients, while refined Gaussians retain their input coefficients.

A.5.4 Supervision and Optimization

Target correspondences. Training uses the paired input and animation-aware target Gaussian representations described in Sec. 4. For the refinement branch, preprocessing records the identity of each surviving original Gaussian, providing direct input-to-target attribute correspondences. Paired

attribute supervision is applied only to queried rigid Gaussians with valid surviving targets. Input Gaussians removed during target construction are excluded from this paired attribute loss.

For the generation branch, the complete target Gaussian set is first sampled by FPS to at most 32,768 primitives. We then select the target primitives associated with non-rigid input regions using the recorded preprocessing lineage. Generated primitives are matched to this target subset by spatial nearest neighbors. Image supervision always uses the complete target representation, rather than this sampled subset.

Refinement attribute loss. Let $\mathcal{P}_{\mathrm{ref}}$ denote the valid rigid-region prediction–target pairs. Define the sign-invariant quaternion error

$$d_q(\boldsymbol{q}, \boldsymbol{q}^\star) = \min\left(\|\boldsymbol{q} - \boldsymbol{q}^\star\|_1, \|\boldsymbol{q} + \boldsymbol{q}^\star\|_1\right). \tag{44}$$

The refinement loss is

$$\mathcal{L}_{\mathrm{ref}} = \frac{1}{|\mathcal{P}_{\mathrm{ref}}|} \sum_{(i,t)\in\mathcal{P}_{\mathrm{ref}}} \Big(|o_i' - o_t^\star| + \frac{1}{3}\|\boldsymbol{\ell}_i' - \boldsymbol{\ell}_t^\star\|_1 + d_q(\boldsymbol{q}_i', \boldsymbol{q}_t^\star)\Big). \tag{45}$$

No separate position or color attribute loss is applied to this branch because those attributes are inherited from the input.

Generation geometry loss. Let $\mathcal{X}$ and $\mathcal{Y}$ denote the generated and sampled target non-rigid center sets, respectively. We use the bidirectional Chamfer loss. We use Euclidean rather than squared Euclidean distances. Nearest-neighbor assignments are computed without gradients, while the selected distances remain differentiable with respect to the predicted centers.

Generation attribute loss. For each generated Gaussian i, let $\nu(i)$ denote its nearest target Gaussian and d_i the matching distance. Attribute supervision is restricted to

$$\mathcal{A} = \{i \mid d_i \le \mathrm{median}_k\, d_k\}. \tag{46}$$

To emphasize shape and orientation errors for anisotropic targets, we define

$$a_i = \exp\left(\max_d \ell^\star_{\nu(i),d} - \min_d \ell^\star_{\nu(i),d}\right) - 1, \tag{47}$$

$$\omega_i = \frac{\max(a_i, 0)}{|\mathcal{A}|^{-1}\sum_{k\in\mathcal{A}} \max(a_k, 0) + 10^{-8}}. \tag{48}$$

The attribute loss is

$$\mathcal{L}_{\mathrm{gen}} = \frac{1}{|\mathcal{A}|}\sum_{i\in\mathcal{A}} \Big(|\alpha_i - \alpha^\star_{\nu(i)}| + \frac{\omega_i}{3}\|\log \boldsymbol{s}_i - \boldsymbol{\ell}^\star_{\nu(i)}\|_1 + \omega_i d_q(\boldsymbol{q}_i, \boldsymbol{q}^\star_{\nu(i)}) + \frac{1}{3}\|\boldsymbol{f}_i^{\mathrm{DC}} - \boldsymbol{f}^{\mathrm{DC},\star}_{\nu(i)}\|_1\Big). \tag{49}$$

A small numerical constant is added before taking the predicted scale logarithm. The anisotropy weights apply only to scale and rotation. Opacity supervision uses activated opacity for the generation branch and opacity logits for the refinement branch. A branch loss is set to zero when its valid supervision set is empty.

Rendering loss and total objective. We render the assembled output and complete target representation from four fixed, uniformly spaced views per object at 512×512 resolution. Images are composited against a black background and clamped to $[0, 1]$. The rendering objective is

$$\mathcal{L}_{\mathrm{img}} = \mathrm{MAE}(I, I^\star) + 0.2\,(1 - \mathrm{SSIM}(I, I^\star)), \tag{50}$$

averaged over the selected views. The full training objective is

$$\mathcal{L} = 10\,\mathcal{L}_{\mathrm{geo}} + 0.05\,\mathcal{L}_{\mathrm{gen}} + 0.2\,\mathcal{L}_{\mathrm{ref}} + 50\,\mathcal{L}_{\mathrm{img}}. \tag{51}$$

The configuration described here does not apply an additional posed-rendering loss. Animation-related supervision is provided through the paired animation-aware targets, together with the rig-conditioned inputs and region-dependent adaptation strategies.

Initialization and optimization. The encoder is initialized from pretrained weights and frozen. The two decoders are initialized from a compatible adaptation checkpoint. When reducing the generation head to eight children, we retain the corresponding child output weights and fixed slot perturbations

from the source checkpoint. Thus, the adaptation stage continues from transferred decoder weights rather than assuming that all residual predictions are initially zero.

The training budget is 50,000 optimization steps with one object per device per step. Eligible two-dimensional weight matrices are optimized with Muon, while embeddings, biases, normalization parameters, and other remaining parameters use AdamW. The base learning rates are 10^{-4} for the refinement decoder and 10^{-5} for the generation decoder. Muon parameter groups use a learning-rate multiplier of 5, momentum 0.95, Nesterov updates, and five Newton–Schulz iterations. AdamW uses $\beta_1 = 0.9$, $\beta_2 = 0.95$, and zero weight decay.

All parameter groups follow cosine decay to one tenth of their initial learning rates. We clip the combined gradient norm to 1.0. The encoder and generation backbone use half-precision computation where supported, while optimization maintains full-precision master parameters.

A.6 Optimization Details for Animation-Aware Gaussians and Skinning Field

Directly optimizing the Gaussian representation and the neural skinning field from posed-image supervision is unstable. Geometric artifacts in the input Gaussians may be incorrectly explained by the skinning field, while inaccurate skinning weights may in turn produce misleading gradients for Gaussian optimization. We therefore adopt a progressive optimization strategy that first estimates the skinning field while restricting Gaussian changes, and finally adapts the Gaussians using the fixed skinning field. The following implementation details describe how we optimize the skinning field and animation-aware Gaussians using rendering supervision under deformation after skinning initialization.

Offline Pose Augmentation. The original animation sequences cover only a limited portion of the articulation space. We therefore generate additional poses offline to provide broad motion coverage for every joint. For each non-root joint, we construct a candidate set of 100 rotations by combining 20 approximately uniform axes on the unit sphere with 5 uniformly spaced rotation angles.

To obtain simultaneous multi-joint motion while maintaining joint-wise coverage, we order the bones by a depth-first traversal of the skeleton and divide them into

$$K_{tg} = \max(1, \lfloor 0.25(|\mathcal{B}| - 1) \rfloor) \tag{52}$$

topologically local groups. For each synthesized pose, we select one joint from every group and assign one of its candidate rotations, while keeping the remaining joints in their canonical states. Both joints within each group and candidate rotations for each joint are visited using shuffled cyclic lists. Consequently, all joints and their candidate rotations are covered before sampling begins to repeat.

We generate 400 articulated poses per object, together with one canonical pose, resulting in 401 pose configurations. For every configuration, we deform the source mesh using its ground-truth rig and render an RGB image and foreground mask from the associated camera view. All supervision is generated offline at a resolution of 1024×1024 with a white background. During optimization, we randomly sample one of these pose–view pairs and use the corresponding mesh rendering as image-space supervision.

Scale-Aware Skinning Field Optimization. After initialization, we refine the skinning field using the augmented posed-image observations. Directly fixing all Gaussian attributes would cause artifacts in the vanilla Gaussians to be absorbed by the learned skinning weights. We observe that overly elongated Gaussian scales are a major source of spike-like artifacts around articulated regions. Therefore, we fix Gaussian centers, rotations, opacities, and SH appearance coefficients, while jointly optimizing the skinning field and Gaussian log-scales.

Given a sampled pose–view pair, we deform the Gaussians using weights queried from the field and minimize

$$\mathcal{L}_{\text{render}} = 0.8\mathcal{L}_1 + 0.2\,(1 - \text{SSIM}) + \lambda_{\text{mask}}\mathcal{L}_{\text{mask}}, \tag{53}$$

where $\mathcal{L}_{\text{mask}}$ penalizes rendered content outside the ground-truth foreground mask and $\lambda_{\text{mask}} = 1$. Allowing the Gaussian scales to adapt reduces geometric interference in the image-space supervision and provides cleaner gradients for estimating the skinning field. We perform this optimization for $10{,}000$ iterations without densification or pruning.

Skinning Confidence. After optimizing the field, we estimate its reliability over the spatial extent of each Gaussian. The center confidence is defined as the largest predicted bone weight:

$$c_i^{\text{center}} = \max_b w_b(\boldsymbol{\mu}_i). \tag{54}$$

To measure whether the predicted binding remains consistent over the Gaussian support, we query the field at six points along its three principal axes:

$$\mathbf{p}_{i,a}^{\pm} = \boldsymbol{\mu}_i \pm 2\mathbf{R}_i \,\mathrm{diag}(\mathbf{s}_i)\mathbf{e}_a, \qquad a \in \{1,2,3\}. \tag{55}$$

The shape confidence is

$$c_i^{\text{shape}} = 1 - \max_{a,\pm} \frac{1}{2} \left\| \mathbf{w}(\mathbf{p}_{i,a}^{\pm}) - \mathbf{w}(\boldsymbol{\mu}_i) \right\|_1. \tag{56}$$

A high value indicates that the field predicts a consistent binding throughout the Gaussian support. We use a center-confidence threshold of 0.9 to separate rigid and non-rigid regions. Queries without valid sparse-field support are excluded from adaptation and remain fixed.

Confidence-Guided Gaussian Optimization. We subsequently freeze the optimized skinning field and restart Gaussian adaptation from the original canonical representation $\mathcal{G}_0$. Restarting from $\mathcal{G}_0$, rather than continuing from the scale-adjusted representation used for field estimation, prevents temporary scale changes from being carried into the final animation-aware representation.

For rigid-region Gaussians, we fix their centers and SH appearance coefficients and optimize only their scales, rotations, and opacities. This preserves reliable geometry and texture while allowing their spatial support to adapt to articulated deformation. For non-rigid Gaussians, we optimize all Gaussian attributes, including centers and SH coefficients, and enable densification. This gives uncertain articulated regions sufficient flexibility to recover geometry and appearance that cannot be represented by the input primitives.

Confidence-Weighted Attribute Regularization. We constrain Gaussian adaptation relative to the input representation. For a rigid-region Gaussian, we define its reliability and regularizer weight as

$$\rho_i = \min\left(c_i^{\text{center}}, c_i^{\text{shape}}\right), \qquad \alpha_i = 0.05 + 0.95\rho_i^4. \tag{57}$$

The attribute regularization is

$$\mathcal{L}_{\text{attr}} = \frac{\sum_{i\in\mathcal{R}} \alpha_i \left(d_i^s + d_i^r + d_i^o\right)}{\sum_{i\in\mathcal{R}} \alpha_i}, \tag{58}$$

where $\mathcal{R}$ denotes rigid-region Gaussians. The scale term d_i^s is the mean per-axis relative change of the activated scale, the rotation term is

$$d_i^r = 1 - \left| \left\langle \frac{\mathbf{q}_i}{\|\mathbf{q}_i\|}, \frac{\mathbf{q}_i^0}{\|\mathbf{q}_i^0\|} \right\rangle \right|, \tag{59}$$

and d_i^o is the absolute difference between the activated opacities. We set the weight of $\mathcal{L}_{\text{attr}}$ to 10^{-3}. We also apply a weaker attribute-offset regularizer with weight 10^{-5} to the adaptable non-rigid Gaussians.

For a movable Gaussian with original ancestor center $\boldsymbol{\mu}_{a(i)}^0$, we additionally use

$$\mathcal{L}_{\text{move}} = \frac{1}{|\mathcal{D}|} \sum_{i\in\mathcal{D}} \frac{\left\| \boldsymbol{\mu}_i - \boldsymbol{\mu}_{a(i)}^0 \right\|_2^2}{r_{a(i)}^2}, \tag{60}$$

where $\mathcal{D}$ denotes non-rigid Gaussians and $r_{a(i)}$ is the larger of the original maximum Gaussian scale and the sparse-voxel width. Its weight is set to 10^{-5}. This term limits excessive position drift without preventing local geometric adaptation.

The complete Gaussian objective is

$$\mathcal{L}_{\text{GS}} = \mathcal{L}_{\text{render}} + \lambda_{\text{rest}}\mathcal{L}_{\text{rest}} + 10^{-3}\mathcal{L}_{\text{attr}} + 10^{-5}\mathcal{L}_{\text{delta}} + 10^{-5}\mathcal{L}_{\text{move}}. \tag{61}$$

We sample a canonical-pose view every four iterations and set $\lambda_{\text{rest}} = 1$ to preserve the original appearance.

Densification, Pruning, and Skinning Binding. Only non-rigid Gaussians participate in densification. Densification starts at iteration 500, is performed every 100 iterations, and stops at iteration $12{,}000$. The screen-space gradient threshold is 2×10^{-4}. Both rigid and non-rigid Gaussians are subject to opacity-based pruning with threshold 0.005. We optimize for $15{,}000$ iterations in total, so the final $3{,}000$ iterations retain attribute optimization and pruning while disabling the creation of new Gaussians.

A Gaussian created by cloning initially inherits the current skinning weights of its parent. For a Gaussian generated by splitting, we query the frozen field at its new center whenever the complete trilinear neighborhood is supported by the sparse field. If the new position lies outside the valid support, it retains the parent binding. Similarly, when an existing Gaussian moves, its weights are updated at the current position only while the query remains valid; otherwise, its most recent valid weights are retained. This preserves continuous bindings under topology changes without extrapolating the sparse field into unobserved regions.

Implementation Settings. The sparse skinning field uses a nominal resolution of 512^3. During scale-aware field optimization, we use Adam with an initial field learning rate of 10^{-4}, decayed to 10^{-5} over the first 500 iterations, and a Gaussian-scale learning rate of 5×10^{-3}. During Gaussian adaptation, the position learning rate is decayed from 8×10^{-5} to 8×10^{-7}. The learning rates for SH features, opacity, scale, and rotation are 1.25×10^{-3}, 2.5×10^{-2}, 5×10^{-4}, and 5×10^{-4}, respectively. We use online hard-example mining after a 100-iteration warm-up.