ControlGS: Conditioning Neural Gaussians for Downstream-Processing–Aware XR Rendering

WEIKAI LIN, University of Rochester, USA
JUNJIE ZHAO, University of Rochester, USA
CARL MARSHALL, Reality Labs, Meta, USA
SUSHANT KONDGULI, Reality Labs, Meta, USA
YUHAO ZHU, University of Rochester, USA

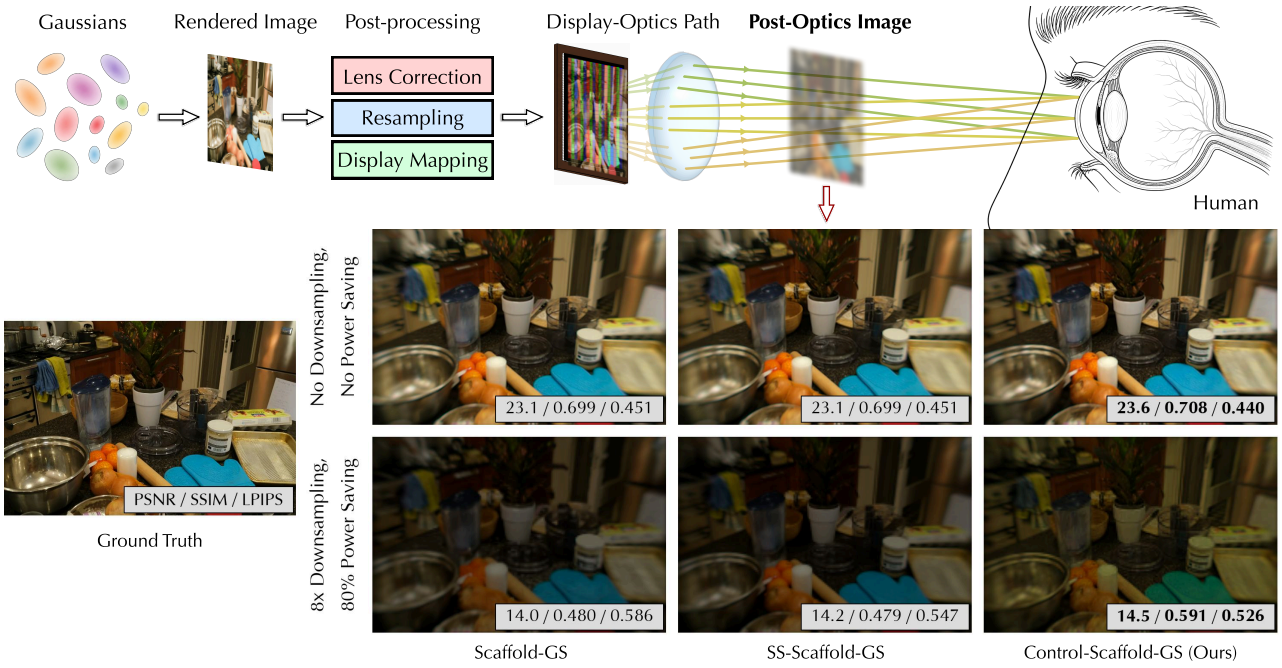

Fig. 1. **ControlGS enables downstream-processing–aware XR rendering.** Top: a Gaussian-rendered image passes through post-processing and the display-optics path before forming the post-optics virtual image. Bottom: compared with Scaffold-GS and its supersampling-enhanced variant (SS-Scaffold-GS), ControlGS produces higher-quality virtual images under both native resolution with no power saving and 8× downsampling with 80% power saving.

Extended Reality (XR) users do not directly perceive the output of a rendering engine. Instead, rendered images pass through a post-processing pipeline and the physical display-optics path before reaching the eye. Critically, the exact downstream processing can vary significantly at run time, influenced by, for instance, camera pose and display power budget. Traditional 3DGS methods either implicitly assume that this downstream pipeline preserves image quality or cannot adapt to downstream processing changes. To bridge this gap, we present ControlGS, an XR Gaussian rendering pipeline that optimizes end-to-end visual quality. ControlGS models and integrates the entire downstream processing, between the rendering output and the human eye, into the optimization objective. To adapt to downstream processing at run time, ControlGS dynamically generates Gaussian primitives conditioned upon the downstream processing parameters. Experiments show that ControlGS consistently improves end-to-end post-optics XR quality across different neural Gaussian backbones and datasets, with minimal overhead. Code is available at https://horizon-lab.org/controlgs/.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Image-based rendering**.

Additional Key Words and Phrases: 3D Gaussian splatting, AR/VR

ACM Reference Format:
Weikai Lin, Junjie Zhao, Carl Marshall, Sushant Kondguli, and Yuhao Zhu. 2026. ControlGS: Conditioning Neural Gaussians for Downstream-Processing–Aware XR Rendering. In *SIGGRAPH Asia 2026 Conference Papers (SA Conference Papers '26), December 01–04, 2026, Kuala Lumpur, Malaysia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3829340.3842281

Authors' Contact Information: Weikai Lin, University of Rochester, Rochester, USA, wlin33@ur.rochester.edu; Junjie Zhao, University of Rochester, Rochester, USA, jzhao58@u.rochester.edu; Carl Marshall, Reality Labs, Meta, Redmond, USA, csmarshall@meta.com; Sushant Kondguli, Reality Labs, Meta, Rochester, USA, sushant.kondguli@gmail.com; Yuhao Zhu, University of Rochester, Rochester, USA, yzhu@rochester.edu.

SA Conference Papers '26, Kuala Lumpur, Malaysia

ACM ISBN 979-8-4007-2842-6/2026/12
https://doi.org/10.1145/3829340.3842281

1 Introduction

Extended Reality (XR) is poised to become the next-generation human-computer interface. 3D Gaussian Splatting (3DGS) [Kerbl et al. 2023] is emerging as a promising rendering paradigm for XR due to its ability to produce photorealistic results at real-time frame rates while massively simplifying content authoring.

Directly using a Gaussian renderer in XR, however, creates a gap between the rendered image and the final image perceived by humans. This is because there is a sequence of signal processing, both computationally and in hardware, that takes place to transform the rendered image to photons entering the user's eye. Computationally, a rendered image must pass through three main processing stages, i.e., lens correction, anti-aliasing resampling, and display mapping under a power constraint [Chen et al. 2024; Duinkharjav et al. 2022; Lin et al. 2026]. These computational stages, followed by the physical display-optics path, transform the rendered image before it is perceived by humans.

Critically, the exact downstream processing can vary significantly at run time, influenced by, for instance, camera pose and display power budget. For instance, as the camera/user moves farther away from a scene, the effective display sampling rate of the scene is lower (equivalently, the spatial frequency of the scene on the display plane is higher). Traditional 3DGS-for-XR methods either implicitly assume that this downstream pipeline preserves image quality and, thus, optimize only for the renderer output [Franke et al. 2025; Lin et al. 2025a,b] or cannot adapt to downstream processing changes.

We introduce ControlGS, a framework for downstream-processing–aware Gaussian rendering. We model the entire downstream processing pipeline between the renderer output and the human eyes, and integrate such a model into the 3DGS training objective. Importantly, we observe that traditional 3DGS is fundamentally ill-suited to adapt to the variability in downstream processing, because its Gaussian primitives are fixed after training. Instead, our idea is to dynamically generate, from a set of learned latent vectors, the Gaussian primitives at run time. This generation process is, importantly, conditioned upon various downstream-specific parameters such as display sampling rate and the power budget to enable adaptation.

Specifically, ControlGS combines a condition-injection module with an MLP. For each computational stage, we learn a condition vector and a projection layer that embed the downstream parameters as a latent condition, which is injected into the MLP to control the attributes of the Gaussian primitives. The injected conditions can be individually and independently toggled depending on the exact downstream processing pipeline, broadening its applicability.

We evaluate ControlGS with diverse MLP-based Gaussian decoders under a typical post-processing pipeline covering lens correction, anti-aliasing resampling, and display mapping. Experiments show that ControlGS consistently improves the end-to-end image quality compared to baselines that do not consider downstream processing. Our contributions are summarized as follows:

- We formulate XR Gaussian rendering as end-to-end optimization across both Gaussian splatting and post-rendering processing.
- We propose ControlGS, a Gaussian splatting method that dynamically generates Gaussian primitives conditioned on downstream processing parameters.
- ControlGS supports various XR post-processing modules, achieving consistent end-to-end quality gains.

2 Related Work

Gaussian Splatting for XR Rendering. 3DGS [Kerbl et al. 2023] represents scenes with explicit anisotropic Gaussians and supports real-time photorealistic rendering. Neural Gaussian methods [Chen et al. 2025; Liu et al. 2024; Lu et al. 2024; Ren et al. 2025; Wang et al. 2024] further replace explicit per-Gaussian attributes with features and neural decoders, which generate Gaussian attributes at render time. Several works improve Gaussian rendering for XR or multi-resolution settings, including anti-aliased Gaussian rendering [Yu et al. 2024a], foveated rendering [Fan et al. 2025; Franke et al. 2025; Lin et al. 2025a], and display-power-aware rendering [Lin et al. 2025b]. However, these methods optimize renderer-side quality and ignore quality changes introduced by post-rendering modules. In contrast, ControlGS optimizes for the end-to-end XR quality.

XR Post-Rendering Pipeline. After rendering, XR systems apply a post-rendering pipeline before the image reaches the user's eye. Lens correction pre-warps images to compensate for distortion and chromatic aberrations [Khronos OpenXR Working Group 2026; Martschinke et al. 2019; Pohl et al. 2013]. Display mapping converts rendered colors to display intensities and can reduce display power through color or luminance remapping [Chen et al. 2024; Duinkharjav et al. 2022; Lin et al. 2026]. Differentiable lens models have also been used for end-to-end camera imaging optimization [Tseng et al. 2021; Zhang et al. 2024]. ControlGS uses differentiable post-rendering modules, enabling an end-to-end loss path from the eye-space image to our condition-injection module.

Neural Gaussians. Our method shares similarities and key differences with neural gaussian models, which also use MLPs to dynamically decode Gaussian primitives during rendering [Chen et al. 2025; Liu et al. 2024; Lu et al. 2024; Ren et al. 2025; Wang et al. 2024]. However, the main objective of neural Gaussian models is to compress the model (through the MLP). ControlGS proposes *conditionally* dynamic decoding, where the MLP is conditioned upon downstream processing parameters. In addition, ControlGS's use of the MLP is less concerned with compression (although we do inherent the benefits of a small model size), but focuses on adapting the Gaussian primitives to fit the downstream processing.

3 Problem Formulation

Fig. 2 illustrates the XR pipeline for 3D Gaussian rendering. A 3D Gaussian renderer first produces a rendered image, which passes through a sequence of computational post-processing stages (Sec. 3.1), followed by the physical display-optics path (Sec. 3.2). Only after these stages does the image reach the user's eye. Our rendering objective integrates these two processes (Sec. 3.3).

3.1 Post-Processing Pipeline in XR

Given a rendered image I, an XR system applies a sequence of post-processing modules. In this paper, we focus on three common modules: lens correction $\mathcal{P}_{\text{lens}}$, anti-aliasing resampling $\mathcal{P}_{\text{res}}$, and display mapping $\mathcal{P}_{\text{disp}}$. Each stage is parameterized by a corresponding

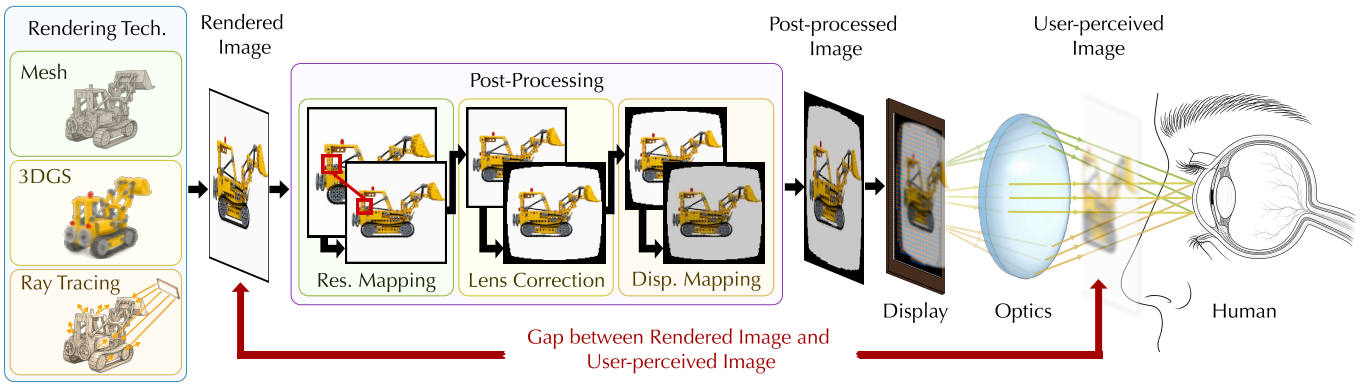

Fig. 2. XR pipeline for Gaussian rendering. The rendered image from a Gaussian renderer passes through post-processing modules and the physical display-optics path before reaching the user's eye. Thus, the user perceives the final eye-space image rather than the raw rendered image. For clarity, the figure places the perceived image on the right side of the optics, although an actual near-eye display forms a virtual image on the opposite side.

run-time parameter: rendering-time sampling rate s_{res}, lens model s_{lens}, and display power target s_{disp}. These stages collectively produce the post-processed image I_{post} given an image I generated by a 3DGS model:

$$I_{\text{post}} = \mathcal{P}_{\text{disp}}\big(\mathcal{P}_{\text{res}}(\mathcal{P}_{\text{lens}}(I; s_{\text{lens}}); s_{\text{res}}) ; s_{\text{disp}}\big) . \tag{1}$$

Lens Correction. The projection lens introduces spatially varying aberrations (Eqn. 8). Following prior work [Hiroi et al. 2022; Khronos OpenXR Working Group 2026; Martschinke et al. 2019; Pohl et al. 2013], we assume that the post-processing pipeline uses a color-dependent warping stage $\mathcal{P}_{\text{lens}}$ to computationally compensate for the lens aberrations (Eqn. 8). As is commonly done, we model $\mathcal{P}_{\text{lens}}$ as a per-channel distortion map between each input pixel $I_{x,y,c}$ and output pixel $I_{x',y',c}$, and the one-to-one mapping between the input pixel location (x, y, c) and output pixel location (x', y', c) is given by the lens model s_{lens}:

$$\mathcal{P}_{\text{lens}}(I_{x,y,c}; s_{\text{lens}}) : I_{x,y,c} \mapsto I_{x',y',c}, \quad (x', y', c) = s_{\text{lens}}(x, y, c). \tag{2}$$

Anti-Aliasing Resampling. From an abstract signal-processing perspective, and analogous to traditional surface and volume splatting [Zwicker et al. 2001a,b], a 3DGS model is best thought of as a pre-computed texture stored in the scene space. The scene sampling rate of this texture is determined by the image resolution and camera poses at the training time. Rendering the 3DGS model can be thought of as sampling the texture (or re-sampling the scene).

However, the rendering-time sampling rate and training-time sampling rate might differ — for two main reasons. First, the display resolution might be different from the resolution of the training images. Second, the rendering-time camera pose could be closer (upsampling) or farther (downsampling) than the training poses; similarly, the rendering lens might support zoom and/or focal-length adjustment. As a result, aliasing could occur.

A typical anti-aliasing strategy is to first super-sample the texture at a rate higher than the display sampling rate, followed by a reconstruction filter to reconstruct the continuous scene signal, followed by a pre-filter that band-limits the scene signal before re-sampling the band-limited signal at the display sampling rate. The reconstruction filter and the pre-filter are usually combined together and called the *resampling* filter.

In our formulation, $\mathcal{P}_{\text{res}}$ is expressed as:

$$\mathcal{P}_{\text{res}}(I_S; s_{\text{res}}) : I_S \mapsto \text{III}_{s_{\text{res}}}(I_S \otimes h) , \tag{3}$$

where I_S is the image rendered by a 3DGS model and lens-corrected at a sampling rate S higher than the display sampling rate s_{res}, h represents the resampling filter, $\otimes$ is convolution, and $\text{III}_{s_{\text{res}}}$ is a Dirac-comb function representing sampling at a rate of s_{res}.

Display Mapping. After anti-aliasing resampling, the image resolution matches the target display resolution, but the image values still need to be mapped to the pixel values for the physical display, a step we call display mapping $\mathcal{P}_{\text{disp}}$. Apart from color space conversion (CSC) and gamut mapping (GM), an important consideration of display mapping is to conserve display power, which has become increasingly important for XR devices [Chen et al. 2024; Duinkharjav et al. 2022]. Chen et al. [2024] shows that uniform dimming (UD) is a particularly effective method for saving display power while preserving perceptual quality. UD applies a single global scaling factor to all image pixels to meet a given display power target. Under UD, $\mathcal{P}_{\text{disp}}$ is expressed as (omitting CSC and GM):

$$\mathcal{P}_{\text{disp}}(I; s_{\text{disp}}) : I \mapsto \frac{s_{\text{disp}}}{P_{\text{disp}}(I)} I, \tag{4}$$

where $P_{\text{disp}}(\cdot)$ represents the display power of an image, and s_{disp} represents the target display power and can vary at run time.

3.2 XR Physical Display-Optics Path

After post-processing, I_{post} is sent to the physical display $\mathcal{D}_{\text{phys}}$ and then transformed by the headset optics $\mathcal{O}_{\text{phys}}$ before it is perceived by the user. The post-optics virtual image I_{eye} is

$$I_{\text{eye}} = \mathcal{O}_{\text{phys}}\big(\mathcal{D}_{\text{phys}}(I_{\text{post}})\big) . \tag{5}$$

Physical Display. The display $\mathcal{D}_{\text{phys}}$ reconstructs a continuous optical signal from discrete image pixel values. This signal reconstruction is modeled by treating each display pixel as a box filter:

$$\mathcal{D}_{\text{phys}}(I_{\text{post}}) : I_{\text{post}} \mapsto I_{\text{disp}}, \qquad I_{\text{disp}} = I_{\text{post}} \otimes b, \tag{6}$$

where I_{disp} is the optical image emitted from the display, and b is a box filter. The resulting display power is modeled using a common linear model [Chen et al. 2024; Duinkharjav et al. 2022; Lin et al. 2025b], where the dynamic display power is modeled as a weighted sum of the linear RGB values:

$$P_{\text{disp}}(I) = \sum_{x,y} \big(\alpha I_R(x,y) + \beta I_G(x,y) + \gamma I_B(x,y)\big), \tag{7}$$

where (α, β, γ) are panel-dependent coefficients.

Projective Lens. The projective optics O_{phys} projects the displayed image to a larger virtual image at a comfortable viewing distance. At the same time, the optics introduce aberrations. The optical propagation can be modeled with a wavelength-dependent shift-variant point-spread function (PSF):

$$\begin{aligned} O_{\text{phys}}(I_{\text{disp}}) &: I_{\text{disp}} \mapsto I_{\text{eye}}, \\ I^{\lambda}_{\text{eye}}(x,y) &= \iint \text{PSF}_{\lambda}(x,y;u,v)\, I^{\lambda}_{\text{disp}}(x-u, y-v)\, du\, dv. \end{aligned} \tag{8}$$

where $\lambda \in \Lambda$ denotes the visible wavelength range, and the PSF depends on both field position and wavelength. This model captures spatially varying, wavelength-dependent (chromatic) aberrations.

3.3 Problem Formulation

To bridge the gap between a 3DGS-rendered image and the image that humans see, we optimize I_{eye} after the optics (Eqn. 5), rather than the intermediate rendered image. The end-to-end XR operator $\mathcal{E}_s$ includes post-processing, the physical display, and optics:

$$\mathcal{E}_s(I) = O_{\text{phys}}\big(\mathcal{D}_{\text{phys}}\big(\mathcal{P}_{\text{disp}}\big(\mathcal{P}_{\text{res}}\big(\mathcal{P}_{\text{lens}}(I; s_{\text{lens}}); s_{\text{res}}\big); s_{\text{disp}}\big)\big)\big). \tag{9}$$

Let q be the camera view and R_θ be a Gaussian renderer with parameters θ. The objective is:

$$\min_{\theta}\ \mathbb{E}_{q,s}\, \mathcal{L}_{\text{img}}\big(\mathcal{E}_s(R_\theta(q)), I_{\text{gt}}(q)\big). \tag{10}$$

Here $R_\theta(q)$ is the image rendered by the Gaussian renderer from view q, and $I_{\text{gt}}(q)$ is the target image for view q. $\mathcal{L}_{\text{img}}$ is an image loss that compares the difference between two images.

Importantly, the three downstream processing parameters (s_{lens}, s_{res}, and s_{disp}) might vary at the rendering time. Therefore, the renderer must adapt to the specific run-time states of these parameters in order to maximize the end-to-end visual quality.

- As the camera moves farther from the scene, or when an XR application simulates zooming out by reducing the lens focal length, the scene occupies a smaller region on the image plane, thereby reducing the effective sampling rate s_{res}.
- The display power budget and, accordingly, s_{disp}, might also change depending on the user preference or battery status.
- Finally, although the physical lens is fixed for a given XR device, its aberrations are spatially varying (i.e., $s_{\text{disp}}(\cdot)$ depends on both x and y, as seen in Eqn. 2). As the camera pose changes, the same Gaussian primitive may project to different image-plane locations and, thus, experience different aberrations.

We provide concrete examples of dynamic sampling-rate changes and view-dependent aberration-state changes in Supp. B.

4 ControlGS

ControlGS provides a solution to the problem in Eqn. 10. Fig. 3 gives an overview of ControlGS. The key idea is to dynamically generate the Gaussian primitives conditioned upon the downstream processing parameters (Sec. 4.1). The conditioning strength is controlled based on the exact parameter magnitude (Sec. 4.2). We discuss how to train ControlGS in a light-weight fashion (Sec. 4.3).

4.1 Conditional Dynamic Gaussians

Neural Gaussian Backbone. The formulation in Eqn. 10 requires the Gaussian renderer to adapt to run-time XR states. Explicit 3DGS methods [Fan et al. 2024; Kerbl et al. 2023; Yu et al. 2024a] use fixed Gaussian attributes at run time and cannot meet this requirement. Therefore, we must dynamically generate the Gaussian primitives based on the downstream processing parameters (s_{lens}, s_{res}, and s_{disp}). Inspired by the neural Gaussian-family methods [Lu et al. 2024], we use an MLP to generate the Gaussian primitives from learned latent features at rendering time.

As shown in Fig. 3 (a), we represent the scene with a set of sparse anchor features. Each feature v has a 3D position x_v, an associated feature $\hat{f}_v$, and a local scale factor l_v. At run time, anchor v emits k Gaussian primitives through learned offsets $\{\delta_m\}_{m=0}^{k-1}$:

$$\{\mu_0, \ldots, \mu_{k-1}\} = x_v + \{\delta_0, \ldots, \delta_{k-1}\}\, l_v, \tag{11}$$

where μ_m denotes the center of the m-th emitted Gaussian. The attributes of these Gaussians are decoded by an MLP at render time:

$$\{o, S, R, c\} = F_\theta\Big(\hat{f}_v, \Delta_{vc}, \tilde{d}_{vc}\Big), \tag{12}$$

where F_θ is the MLP attribute decoder, and $\{o, S, R, c\}$ are opacity, scale, rotation, and color. The inputs Δ_{vc} and $\tilde{d}_{vc}$ encode the relative position and viewing direction between anchor v and the camera c. Thus, the decoded Gaussian attributes depend on the camera pose.

Conditional Injection. While the MLP can dynamically generate the Gaussian primitives, it must do so based on the specific downstream processing parameters. The MLP decoder in neural Gaussian rendering provides a natural place to inject downstream conditions. For each post-processing stage m, ControlGS learns per-feature N-dimensional conditions $\{C_{v,m}\}$ and a shared projection $\mathcal{A}_m : \mathbb{R}^N \to \mathbb{R}^{D_1}$, where D_1 is the first hidden-layer width. The projected condition is added to this hidden layer to modulate decoded Gaussian attributes without changing the pretrained model.

Formally, let FC_1 be the first linear layer of the decoder and let $F_{\theta,\text{rest}}$ be the remaining decoder layers. For anchor feature $\hat{f}_v$, view inputs $(\Delta_{vc}, \tilde{d}_{vc})$, and active modules $\mathcal{M}$, we decode Gaussian attributes as

$$\{o, S, R, c\} = F_{\theta,\text{rest}}\left(\text{FC}_1(\hat{f}_v, \Delta_{vc}, \tilde{d}_{vc}) + \sum_{m \in \mathcal{M}} \lambda_{v,m}(s_{v,m})\, \mathcal{A}_m(C_{v,m})\right). \tag{13}$$

The decoded attributes are then combined with the pretrained Gaussian positions (Eqn. 11) for rasterization. Here $C_{v,m}$ is the condition vector for feature v and module m, while $\mathcal{A}_m$ is the projection layer shared across all features for module m. The function $\lambda_{v,m}(s_{v,m})$ maps the run-time state of feature v under module m to an injection

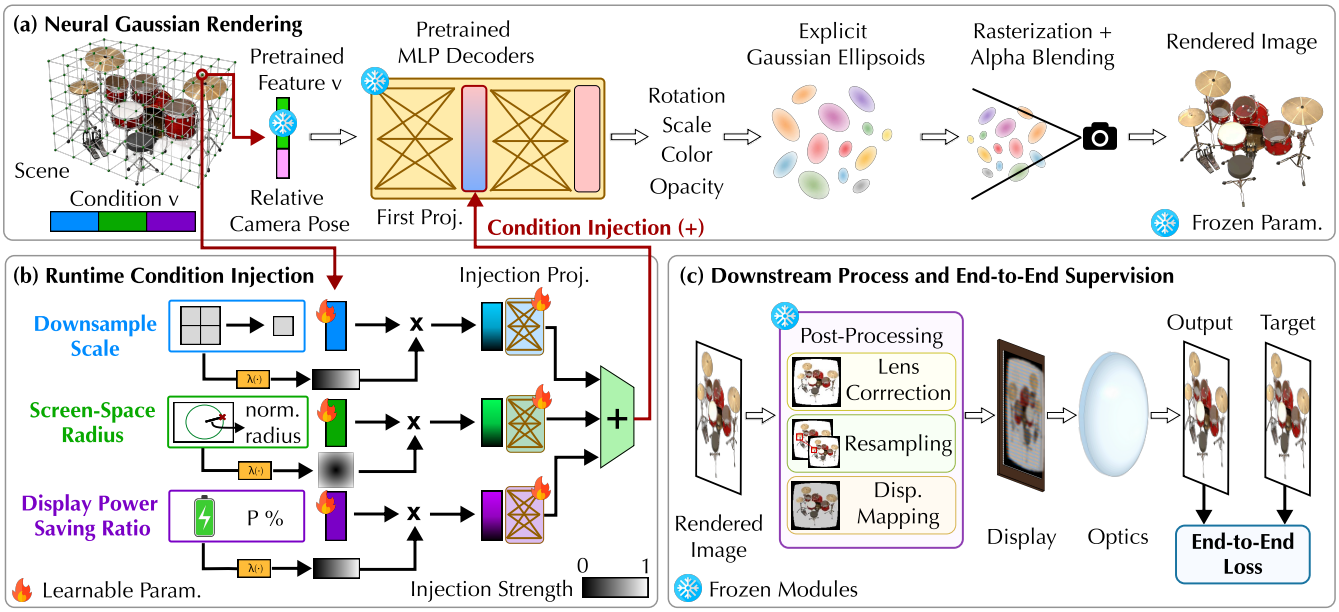

Fig. 3. Overview of ControlGS. (a) A neural Gaussian renderer decodes Gaussian attributes from pretrained feature anchors and rasterizes them into an image. (b) For each feature and downstream module, ControlGS learns a condition vector, projects it into the decoder hidden space, and scales it based on injection strength before injection (Eqn. 13). (c) The conditioned rendering result is passed through fixed XR downstream modules and the plug-in is trained with end-to-end supervision on the final output image (Eqn. 16).

strength, as defined in Sec. 4.2. As shown in Fig. 3(b), different module conditions are summed before injection, making module-specific conditions composable through the same decoder interface.

4.2 Controlling Run-time Injection Strength

The learned condition defines an adaptation direction for a post-processing stage. At the run time, the downstream parameter can change as discussed in Sec. 3.3. The decoding therefore also needs to know how strongly this condition should be injected for each Gaussian feature. We control this with a feature-specific injection strength $\lambda_{v,m}(s_{v,m}) \in [0,1]$. When $\lambda_{v,m}(s_{v,m}) = 1$, the condition for module m is fully injected for feature v. When $\lambda_{v,m}(s_{v,m}) = 0$, the condition is removed from Eqn. 13 for feature v. In this case, the method degenerates to the frozen neural Gaussian backbone.

To compute the strength, each module first converts its run-time state to a scalar control value $\alpha_{v,m}$ using a module-specific function $f_m(\cdot)$. We then define two endpoints, $\alpha_m^{\min}$ and $\alpha_m^{\max}$, over the supported run-time range. The minimum endpoint maps to $\lambda_{v,m} = 0$, and the maximum endpoint maps to $\lambda_{v,m} = 1$. States between them are handled by linear interpolation:

$$\lambda_{v,m}(s_{v,m}) = \frac{\alpha_{v,m} - \alpha_m^{\min}}{\alpha_m^{\max} - \alpha_m^{\min}} = \frac{f_m(s_{v,m}) - f_m(s_m^{\min})}{f_m(s_m^{\max}) - f_m(s_m^{\min})}. \tag{14}$$

Thus the minimum supported state produces no injection for feature v, and the maximum supported state produces full injection. We choose these endpoints to cover the XR run-time range specified by the application. If a run-time state falls outside this range, we clamp the interpolated strength to $[0,1]$ before injection.

We now discuss, for each of the three post-processing stages, how to design the run-time state $s_{v,m}$, its endpoints $s_m^{\min}$ and $s_m^{\max}$, and $f_m(\cdot)$ used to compute injection strength (Eqn. 14).

Lens Correction. Most projective lenses in XR are radially symmetric about the optical center. Therefore, the distance to the optical center determines the amount of aberrations (although not the exact PSF) introduced by the lens. For each Gaussian feature v, the run-time state is its projected screen-space location, $s_{v,\text{lens}} = (x_v, y_v)$, as defined in Eqn. 2. We normalize this location to $p_v = \text{Norm}(x_v, y_v) \in [-1,1]^2$ and use its radius as the control value: $\alpha_{v,\text{lens}} = f_{\text{lens}}(s_{v,\text{lens}}) = \|p_v\|_2$. Thus, $\alpha_{\text{lens}}^{\min} = 0$ at the optical center and $\alpha_{\text{lens}}^{\max} = \sqrt{2}$ at the screen corner. The injection strength $\lambda_{v,\text{lens}}(s_{v,\text{lens}})$ is computed by Eqn. 15.

Anti-Aliasing Resampling. The run-time state s_{res} is the display sampling rate and is shared by all Gaussian features v, i.e., $s_{v,\text{res}} = s_{\text{res}}$ for all v. We convert it to a downsampling scale $d_{\text{res}} = S/s_{\text{res}}$, where S is the Gaussian training resolution. We support $d_{\text{res}} \in [1,8]$: $d_{\text{res}} = 1$ is native resolution with no injection, and $d_{\text{res}} = 8$ is the strongest downsampling with full injection. We use $\alpha_{v,\text{res}} = f_{\text{res}}(s_{v,\text{res}}) = \log_2 d_{\text{res}}$, so $\alpha_{\text{res}}^{\min} = 0$ and $\alpha_{\text{res}}^{\max} = \log_2 8$. $\lambda_{v,\text{res}}$ is computed by Eqn. 15.

Display Mapping. The run-time state s_{disp} is the target display power P_{target} and is shared by all Gaussian features v, i.e., $s_{v,\text{disp}} = s_{\text{disp}}$ for all v. We convert it to a power-saving ratio using the reference display power P_{ref}, which is the display power consumed by the ground truth image: $\rho = f_{\text{disp}}(s_{\text{disp}}) = 1 - s_{\text{disp}}/P_{\text{ref}} = 1 - P_{\text{target}}/P_{\text{ref}}$. We use $\alpha_{v,\text{disp}} = \rho$, with $\alpha_{\text{disp}}^{\min} = 0$ and $\alpha_{\text{disp}}^{\max} = 0.8$. Thus, a larger

power-saving ratio produces stronger condition injection. The injection strength $\lambda_{v,\text{map}}$ is computed by Eqn. 15.

With these choices, Eqn. 14 for module m and feature v becomes

$$\lambda_{v,\text{res}} = \frac{\log_2 d_{\text{res}}}{\log_2 8}, \quad \lambda_{v,\text{lens}} = \frac{\|p_v\|_2}{\sqrt{2}}, \quad \lambda_{v,\text{disp}} = \frac{\rho}{0.8}. \tag{15}$$

4.3 Training and Inference

To minimize training overhead, we use pre-trained neural Gaussian MLPs and train only the run-time injection module. Let θ be the fixed pretrained neural Gaussian parameters, and let $\phi = \{C_{v,m}, \mathcal{A}_m\}_{v,m}$ be the trainable condition-injection parameters. We denote the conditioned renderer by $R_{\theta,\phi}(q; s)$. It renders view q while modulating Gaussian attribute decoding with run-time states s through Eqn. 13. Because the pretrained renderer parameters θ are fixed, we optimize only the condition-injection parameters ϕ. Thus, the formulation in Eqn. 10 becomes the training objective:

$$\min_{\phi} \ \mathbb{E}_{q,s}\, \mathcal{L}_{\text{img}}\big(\mathcal{E}_s\big(R_{\theta,\phi}(q; s)\big), I_{\text{gt}}(q)\big)\,. \tag{16}$$

$\mathcal{E}_s$ applies the downstream modules (Eqn. 9). $I_{\text{gt}}(q)$ is the ground-truth image for view q. We use the image loss

$$\mathcal{L}_{\text{img}} = \alpha\,\mathcal{L}_1 + \beta\,\mathcal{L}_{\text{SSIM}} + \gamma\,\mathcal{L}_{\text{MSE}} + \delta\,\mathcal{L}_{\text{LPIPS}}, \tag{17}$$

where α, β, γ, and δ are scalar loss weights. We follow 3DGS [Kerbl et al. 2023] for $\mathcal{L}_1$ and $\mathcal{L}_{\text{SSIM}}$, and add MSE and LPIPS [Zhang et al. 2018] to improve optimization through the display-optics pipeline.

Inspired by ControlNet [Zhang et al. 2023], we zero-initialize the condition vectors so the conditioned renderer starts from the original model. During training, we sample display sampling rates and target powers from their supported ranges, while aberration states are determined by each feature's projected screen-space position and is naturally sampled by the camera view.

5 Experiments

5.1 Experimental Setup

Datasets. We evaluate on Mip-NeRF 360 [Barron et al. 2022] and NeRF Synthetic [Mildenhall et al. 2020]. Mip-NeRF 360 contains real unbounded scenes with background, representing immersive VR rendering. NeRF Synthetic contains object-centric scenes with only foreground objects, representing AR scenes.

Backbones. ControlGS uses Scaffold-GS [Lu et al. 2024], a canonical neural Gaussian splatting, as the main MLP backbone. To show broad applicability, we also experiment with other common neural Gaussian methods, including Octree-GS [Ren et al. 2025], HAC++ [Chen et al. 2025], ContextGS [Wang et al. 2024], and CompGS [Liu et al. 2024]. All backbones are pretrained and frozen; only the condition-injection parameters are learned (Sec. 4).

Implementation Details. For each downstream module, we set the dimension of per-feature condition $C_{v,m}$ to 8. The full plug-in therefore uses a 24-dimensional condition for three modules. We train the plug-in for 10,000 iterations with a learning rate of 3×10^{-3}. For the loss in Eqn. 17, we set $(\alpha, \beta, \gamma, \delta)$ to $(0.8, 0.2, 10, 0.1)$ for Mip-NeRF 360 and $(0.8, 0.2, 100, 1)$ for NeRF Synthetic. We use larger MSE and LPIPS weights on NeRF Synthetic to compensate for its smaller residual errors. Additional details are provided in Supp. A.

Table 1. End-to-end quality after the XR post-processing pipeline, averaged over all scenes and 20 resolution-power run-time settings. ControlGS here uses the MLP in SS-Scaffold-GS; results on other MLPs are in Supp. D.1.

Method / Dataset	Mip-NeRF 360			NeRF Synthetic		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Run-Time–Fixed Gaussian Splatting						
3DGS	18.15	0.478	0.579	21.96	0.840	0.166
SS-3DGS	**18.93**	**0.496**	**0.558**	**23.36**	**0.869**	**0.146**
Mip-Splatting	18.87	0.493	0.564	23.17	0.862	0.152
SS-Mip-Splatting	**18.93**	**0.497**	**0.557**	**23.37**	**0.870**	**0.145**
Neural Gaussian Splatting						
Scaffold-GS	18.19	0.479	0.578	21.94	0.839	0.167
SS-Scaffold-GS	18.96	0.496	0.558	23.35	0.869	0.147
ControlGS (Ours)	**19.33**	**0.524**	**0.528**	**23.59**	**0.871**	**0.137**

Downstream Modules. For post-processing, we use the box filter for anti-aliasing resampling $\mathcal{P}_{\text{res}}$, pre-computed pixel-wise warping for lens correction $\mathcal{P}_{\text{lens}}$, and uniform dimming (UD) [Chen et al. 2024] for display mapping $\mathcal{P}_{\text{map}}$. For the physical display processes, we model the OLED display pixels as box filters and use the an OLED display power model from prior work [Duinkharjav et al. 2022]. For optics, we use a convex lens design based on Google Cardboard [Google 2014], and generate the spatially-variant PSFs using Zemax. The implementation details and supporting validation are provided in Supp. C.

Metrics. We compare the final virtual image from the XR pipeline (Eqn. 9) with the ground-truth image using PSNR [Huynh-Thu and Ghanbari 2008], SSIM [Wang et al. 2004], and LPIPS [Zhang et al. 2018]. We evaluate four effective display sampling rates or, equivalently, downsampling factors $d_{\text{res}} \in \{1, 2, 4, 8\}$ ($d_{\text{res}} = 1$ is equivalent to training-time resolution), and five display-power-saving ratios $\{0\%, 20\%, 40\%, 60\%, 80\%\}$, yielding 20 run-time settings. Note that the downsampling factor here is not intended to model different physical display resolutions. Rather, it emulates changes in effective sampling rate under camera pose and lens zoom changes, as discussed in Sec. 3.3.

Baselines. We use pretrained neural Gaussian backbones as baselines. We also include two explicit 3DGS-based baselines with fixed primitives: 3DGS [Kerbl et al. 2023] and Mip-Splatting [Yu et al. 2024a]. The baselines directly render at the target display sampling rate. Mip-Splatting has its own set of anti-aliasing methods. Each baseline also has a variant that uses supersampling (SS) for anti-aliasing, where we always render at the training-time resolution before resampling to the display sampling rate. We discuss the details in Supp. C.2. For lens correction and display mapping, all baselines use the same modules as ControlGS.

5.2 Main Results

End-to-End Quality. Tbl. 1 compares end-to-end quality averaged over all scenes and all 20 run-time settings. Using Scaffold-GS as backbones, ControlGS achieves the best quality among baselines. We also enhance each baseline with the same supersampling

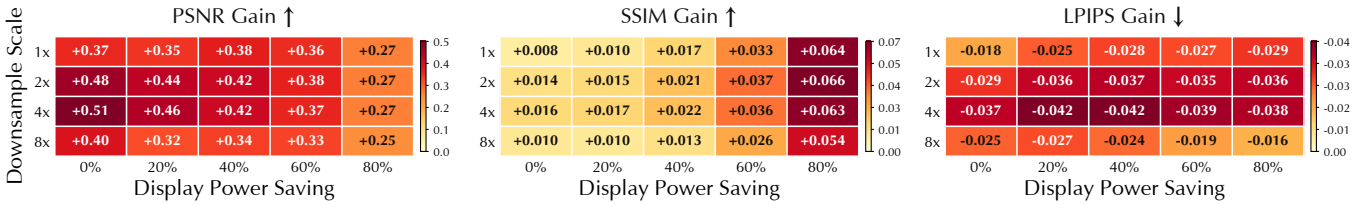

Fig. 4. Average quality improvement of ControlGS over SS-Scaffold-GS on Mip-NeRF 360. Heatmaps show PSNR, SSIM, and LPIPS gains across downsample scale and display-power-saving settings, averaged over all scenes. ControlGS consistently improves end-to-end quality .

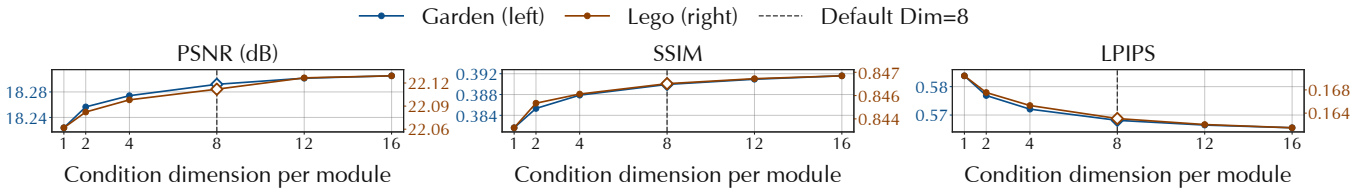

Fig. 5. Effect of the condition dimension on end-to-end quality. The x-axis is the total condition dimension across anti-aliasing resampling, lens correction, and display mapping. The y-axis reports PSNR, SSIM, and LPIPS on the Garden and Lego scenes.

Table 2. Perceptual quality on Mip-NeRF 360.

Method	ColorVideoVDP ↑	HDR-VDP-3 ↑	HDR-FLIP ↓	FovVideoVDP ↑
SS-Scaffold-GS	7.179	6.971	0.455	7.001
ControlGS	**7.232**	**7.111**	**0.418**	**7.079**

Table 3. FPS and model storage.

Method / Dataset	Mip-NeRF 360		NeRF Synthetic	
	FPS↑	Storage (MB)↓	FPS↑	Storage (MB)↓
SS-3DGS	214.9	647.51	764.0	58.51
SS-Scaffold-GS	337.9	180.53	920.5	15.13
ControlGS	268.9	235.36	702.7	19.72

(SS) strategy used by ControlGS. SS improves all baselines, confirming the importance of supersampling. Compared with these SS-enhanced baselines, ControlGS still consistently achieves the best end-to-end quality, showing the benefit of adapting to run-time downstream states through condition injection. ControlGS generalizes to other neural Gaussian backbones, with results provided in Supp. D.1. Per-scene results are in Supp. D.2.

We further evaluate post-optics image quality using four perceptual metrics: ColorVideoVDP [Mantiuk et al. 2024], HDR-VDP-3 [Mantiuk et al. 2023], HDR-FLIP [Andersson et al. 2021], and FovVideoVDP [Mantiuk et al. 2021]. These metrics serve as proxies for the human visual system and complement the standard image-quality metrics above. Tbl. 2 reports results on Mip-NeRF 360. For all metrics, we use the `sRGB_display` setting in HDR-VDP-3 and compute pixels per degree from our optical design. ControlGS consistently improves over SS-Scaffold-GS across all four metrics.

Per-Setting Analysis. To show that ControlGS improves quality across run-time states rather than only on average, we visualize its gains over the SS-enhanced baseline on the full 4×5 run-time grid. Using Scaffold-GS as the backbone, Fig. 4 reports PSNR, SSIM, and LPIPS gains under all resolution and power-saving settings. ControlGS consistently outperforms the SS-enhanced baseline. Even at 1× downsampling and 0% power saving, ControlGS still improves quality through the lens condition, which adapts Gaussian decoding based on each feature's projected screen-space location. Additional heatmaps for NeRF Synthetic are provided in Supp. D.3.

Qualitative Comparison. Beyond objective metrics, Fig. 6 compares the end-to-end visual results of the original baseline, the SS-enhanced baseline, and ControlGS after optics simulation. At the highest resolution without display-power saving, ControlGS produces sharper peripheral details by compensating for lens-induced blur. Under aggressive downsampling and display-power saving, ControlGS adapts to the downsampling ratio to preserve sharper structures while shifting colors toward lower-power yellowish tones for display mapping. This preserves more details that would otherwise be suppressed by dimming. Additional qualitative results across more scenes are provided in Supp. D.5.

Overhead Analysis. Efficiency remains important for XR rendering. We evaluate ControlGS in a split-rendering setting, where a tethered server or cloud GPU renders Gaussian frames before streaming them to the XR device. Using Scaffold-GS, Tbl. 3 reports averaged render-side FPS and storage on Mip-NeRF 360 and NeRF Synthetic. ControlGS achieves 268.9 FPS on Mip-NeRF 360 and 702.7 FPS on NeRF Synthetic, remaining well above typical XR frame-rate requirements. Compared with SS-Scaffold-GS, ControlGS increases storage by about 30% due to the conditional injection module, while remaining substantially smaller than the original 3DGS. We also discussed impact of supersampling in Supp. D.4.

Effect of Injection Strength. Fig. 8 visualizes how injection strength changes the raw rendered output before the downstream pipeline.

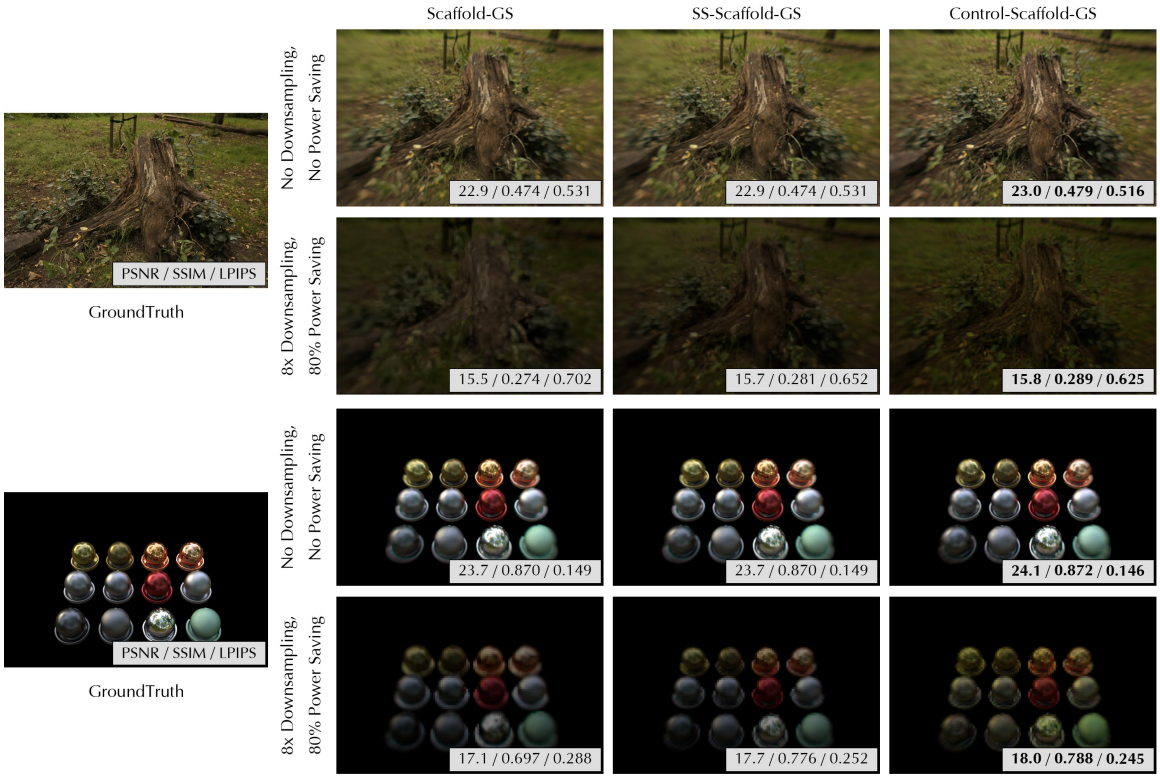

Fig. 6. Qualitative end-to-end comparison on two representative scenes. We compare Scaffold-GS, its supersampled variant, and ControlGS under no downsampling/no power saving and under 8× downsampling with 80% display-power saving. Numbers report PSNR / SSIM / LPIPS for the shown view; bold indicates the best value among the three methods.

Using Scaffold-GS as the backbone, we vary one condition at a time while disabling the others. More examples are provided in Supp. D.6.

From top to bottom, we show the results for anti-aliasing resampling, display mapping, and lens correction. Increasing the display-resolution condition produces sharper outputs to compensate for stronger downsampling. Increasing the display-mapping condition shifts colors toward lower-power yellowish tones. For lens correction, we highlight peripheral crops where optical blur is stronger. The lens-correction condition, whose strength grows from the optical center to the periphery, enhances peripheral details to compensate for the optical distortion. Fig. 7 further visualizes the relationship across the spatially varying PSFs, injection strength, and rendered/post-optics images, providing an intuitive view of how the lens-correction condition depends on image position.

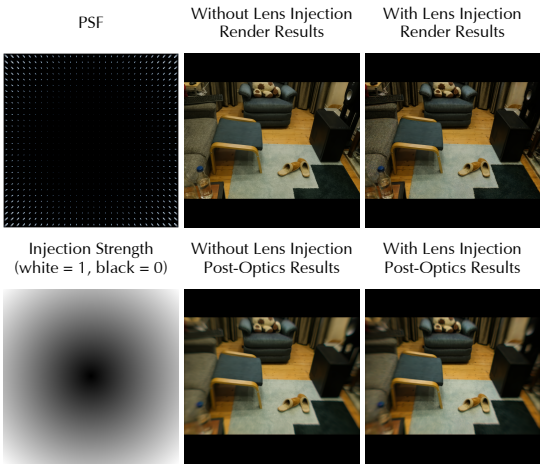

Fig. 7. Lens-aware condition injection under spatially varying aberrations. Left: the spatially varying PSFs (top) and injection-strength map (bottom). Middle and right: results without and with lens injection, respectively; the top and bottom rows show the rendered and post-optics images.

Robustness to Downstream-Model Mismatch. We evaluate robustness by training ControlGS with one canonical set of downstream-pipeline parameters and testing it under perturbed parameters that emulate simulation-to-hardware mismatch. Tbl. 4 reports the improvement of ControlGS over SS-Scaffold-GS under downstream-model perturbations, where each setting perturbs only one parameter. Across variations in PSF spatial extent, RGB wavelengths, display gamma, and the display-power model, ControlGS consistently improves quality. These results demonstrate its robustness to moderate inaccuracies in the simulation.

Table 4. Robustness to downstream-model mismatch on Mip-NeRF 360. Values report the quality improvement of ControlGS over SS-Scaffold-GS.

Setting	ΔPSNR↑	ΔSSIM↑	ΔLPIPS↓
Canonical	+0.37	+0.028	-0.031
PSF 0.9×	+0.38	+0.028	-0.032
PSF 1.1×	+0.37	+0.026	-0.028
RGB wavelengths −10 nm	+0.37	+0.028	-0.030
RGB wavelengths +10 nm	+0.35	+0.027	-0.029
Gamma 2.3	+0.36	+0.027	-0.031
Gamma 2.5	+0.38	+0.029	-0.030
Blue power 0.9×	+0.33	+0.027	-0.030
Blue power 0.8×	+0.28	+0.026	-0.030

Table 5. Ablation studies on Mip-NeRF 360 and NeRF Synthetic. Results are averaged over all scenes and downstream-pipeline settings.

Method	Mip-NeRF 360			NeRF Synthetic		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
ControlGS	19.33	0.524	0.528	23.59	0.871	0.137
w/o Lens	19.25	0.521	0.534	23.52	0.870	0.139
w/o Display Mapping	19.07	0.511	0.531	23.42	0.870	0.138
w/o Resampling	19.32	0.523	0.530	23.58	0.871	0.137
w/o State-Aware	19.05	0.513	0.526	23.40	0.870	0.136
SS-Scaffold-GS	18.96	0.496	0.558	23.35	0.869	0.147

5.3 Design Choice Analysis and Ablation Studies

Condition Dimension. A larger condition dimension gives the plug-in more capacity, but also increases overhead. Fig. 5 studies this trade-off by varying the per-module condition dimension on the Garden and Lego scenes. We therefore choose $N = 8$ for each downstream module, giving a total condition dimension of 24, as the default balance between quality and overhead.

Per-module Condition Ablation. Tbl. 5 ablates the condition modules on Scaffold-GS over all scenes in Mip-NeRF 360 and NeRF Synthetic. We remove the lens-correction, resampling, and display-mapping conditions one at a time while keeping the training and evaluation protocol unchanged. The results show that the module contributions are unequal: display mapping and lens correction provide larger gains, whereas removing the resampling condition causes only a small degradation, which is expected as supersampling already compensates for much of the quality loss caused by sampling-rate changes. This suggests that future work could allocate more condition budget to more important downstream modules.

State-aware Injection Ablation. To verify that the gains do not merely result from additional conditioning capacity, we compare ControlGS against a state-blind baseline with the same 24-dimensional per-feature conditions. This baseline fixes the injection strength to 1 during both training and inference, thereby removing run-time state awareness. Tbl. 5 shows that state-aware injection achieves higher overall quality than this parameter-matched baseline.

6 Discussion and Limitations

Sim-to-Real Gap. ControlGS evaluates the virtual image after a simulated display-optics path, as justified in Supp. C.5. However, this path does not model much of the human visual system (HVS)[1]. Integrating human vision into end-to-end XR evaluation remain an open challenge [Chapiro et al. 2024; Mantiuk et al. 2024; Padmanaban et al. 2020; Zhao et al. 2022]. This sim-to-real gap is orthogonal to our main contribution of adapting rendering to downstream processing parameters. We leave HVS-in-the-loop optimization and user study to future work.

Additional Downstream Modules. We do not model run-time reprojection (e.g., space/time warping) in XR [Evangelakos and Mara 2016; Mark 1999; Van Waveren 2016]. Such warping requires accurate depth and visibility, while reliable GS geometry remains an active research area [Huang et al. 2024; Wolf et al. 2024; Yu et al. 2024b]. We leave incorporating warping as an additional downstream module to future work.

Nevertheless, ControlGS is not limited to the downstream modules considered in this work. Its conditional-injection framework can be extended to other modules through module-specific designs. For example, in foveated rendering, the injection strength could be conditioned on the projected eccentricity of each Gaussian feature. Changes to the downstream modules or physical headset may therefore require retraining or fine-tuning. Such system-specific calibration is common in XR; for example, lens correction requires knowledge of optical aberrations, while display mapping depends on the display power model.

Generalization to Asymmetric Optics. Our current injection-strength design targets radially symmetric lenses, with blur extent varying by distance from the optical center. The framework can support asymmetric lenses by conditioning injection strength on each Gaussian's projected image-plane coordinates (x, y) rather than radius alone.

Gains with Compact Optics. The gains of ControlGS may appear modest because Google Cardboard, used for its open optical parameters, uses a bulky optical design with relatively low aberration, much of which is already corrected by post-processing. As the optics in future XR devices become more compact, they will introduce stronger aberrations that might be be fully corrected by post-processing alone, increasing the value and benefits of ControlGS.

7 Conclusion

We introduced ControlGS, a downstream-aware condition-injection framework for XR neural Gaussian rendering. ControlGS integrate the entire downstream processing pipeline, both the computational and the hardware components, into the optimization objective. ControlGS dynamically generates Gaussian primitives conditioned upon the downstream processing parameters. ControlGS consistently improves end-to-end quality with small overhead.

Acknowledgments

The work is partially supported by NSF Award #2225860, #2126642, and #2044963 and a Meta research grant.

[1]With the exception of the eye optics (e.g., cornea and pupil), which we must model in order to generate post-optics virtual images.

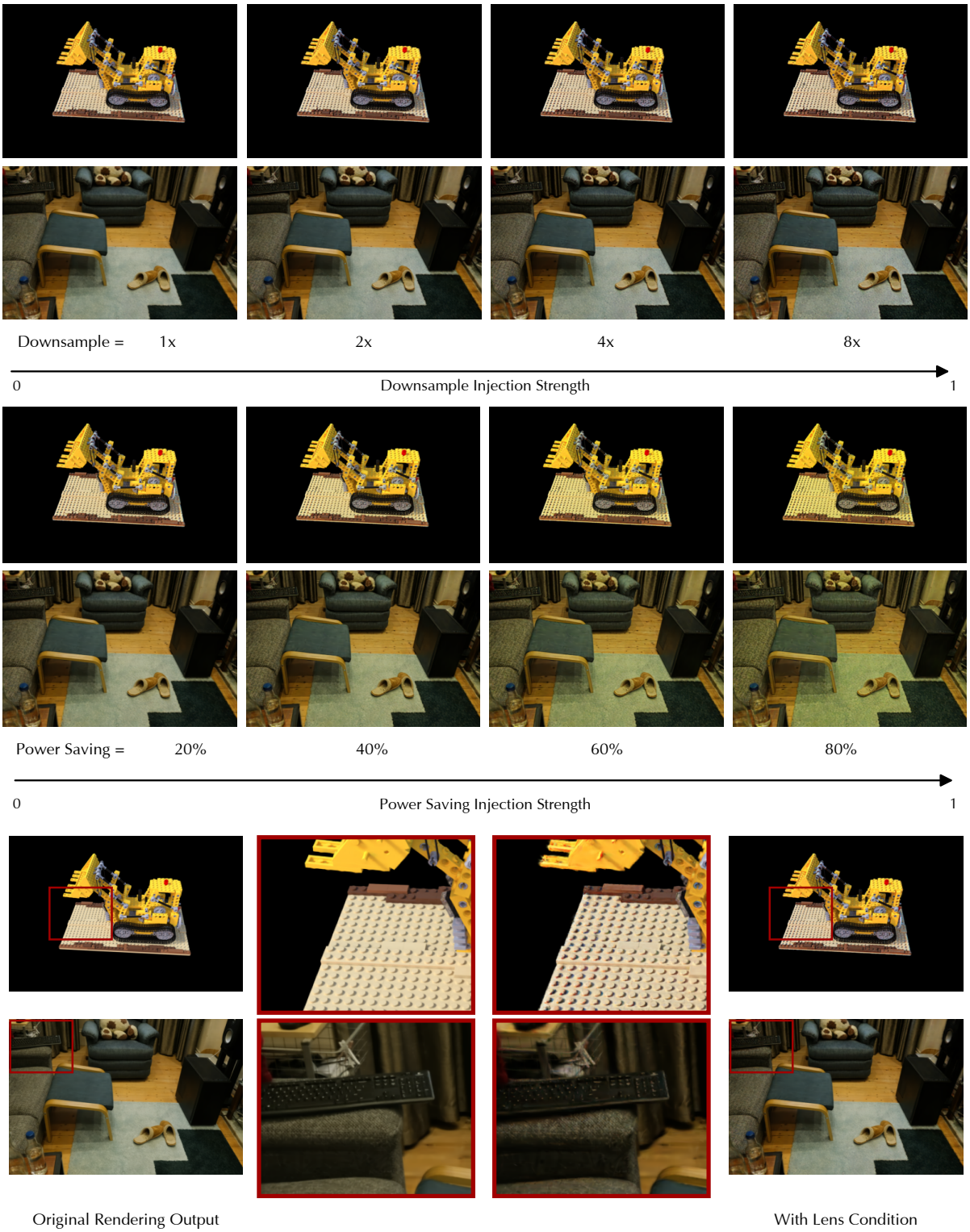

Fig. 8. Condition-response visualization on Lego and Room. We vary one condition at a time and show the raw rendered outputs before the downstream pipeline. From top to bottom, the groups show the effects of anti-aliasing resampling, display mapping, and lens correction. For lens correction, we compare the original Scaffold-GS rendering with the ControlGS-conditioned rendering at the peripheral crops.

References

Pontus Andersson, Jim Nilsson, Peter Shirley, and Tomas Akenine-Möller. 2021. Visualizing errors in rendered high dynamic range images. In *Eurographics-Short Papers*. Eurographics-European Association for Computer Graphics, 25–28.

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. 2022. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5470–5479.

Alexandre Chapiro, Dongyeon Kim, Yuta Asano, and Rafał K Mantiuk. 2024. Ar-david: Augmented reality display artifact video dataset. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–11.

Kenneth Chen, Thomas Wan, Nathan Matsuda, Ajit Ninan, Alexandre Chapiro, and Qi Sun. 2024. Pea-pods: Perceptual evaluation of algorithms for power optimization in xr displays. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–17.

Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. 2025. Hac++: Towards 100x compression of 3d gaussian splatting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2025).

Budmonde Duinkharjav, Kenneth Chen, Abhishek Tyagi, Jiayi He, Yuhao Zhu, and Qi Sun. 2022. Color-perception-guided display power reduction for virtual reality. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–16.

Daniel Evangelakos and Michael Mara. 2016. Extended timewarp latency compensation for virtual reality. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 193–194.

Runze Fan, Jian Wu, Xuehuai Shi, Lizhi Zhao, Qixiang Ma, and Lili Wang. 2025. Fov-gs: Foveated 3d gaussian splatting for dynamic scenes. *IEEE Transactions on Visualization and Computer Graphics* (2025).

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. 2024. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *Advances in neural information processing systems* 37 (2024), 140138–140158.

Linus Franke, Laura Fink, and Marc Stamminger. 2025. Vr-splatting: Foveated radiance field rendering via 3d gaussian splatting and neural points. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 8, 1 (2025), 1–21.

Google. 2014. Google Cardboard. https://arvr.google.com/cardboard/.

Yuichi Hiroi, Kiyosato Someya, and Yuta Itoh. 2022. Neural distortion fields for spatial calibration of wide field-of-view near-eye displays. *Optics Express* 30, 22 (2022), 40628–40644.

Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2024. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 conference papers*. 1–11.

Quan Huynh-Thu and Mohammed Ghanbari. 2008. Scope of validity of PSNR in image/video quality assessment. *Electronics letters* 44, 13 (2008), 800–801.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, George Drettakis, et al. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.* 42, 4 (2023), 139–1.

Khronos OpenXR Working Group. 2026. OpenXR Specification. https://registry.khronos.org/OpenXR/specs/1.1/html/xrspec.html Version 1.1.59.

Weikai Lin, Yu Feng, and Yuhao Zhu. 2025a. Metasapiens: Real-time neural rendering with efficiency-aware pruning and accelerated foveated rendering. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 669–682.

Weikai Lin, Sushant Kondguli, Carl Marshall, and Yuhao Zhu. 2025b. PowerGS: Display-Rendering Power Co-Optimization for Neural Rendering in Power-Constrained XR Systems. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*. 1–12.

Weikai Lin, Sheng Zhao, Ian Ross, Carl Marshall, Sushant Kondguli, and Yuhao Zhu. 2026. LowPowAR: Power-Constrained Tone Mapping for Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics* (2026). To appear.

Xiangrui Liu, Xinju Wu, Pingping Zhang, Shiqi Wang, Zhu Li, and Sam Kwong. 2024. Compgs: Efficient 3d scene representation via compressed gaussian splatting. In *Proceedings of the 32nd ACM International Conference on Multimedia*. 2936–2944.

Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. 2024. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 20654–20664.

Rafał K. Mantiuk, Alexandre Chapiro, Gizem Rufo, Trisha Lian, Rafał K. Mantiuk, Gyorgy Denes, Alexandre Chapiro, and Anton Kaplanyan. 2021. FovVideoVDP. *ACM Transactions on Graphics (TOG)* 40 (2021), 1 – 19.

Rafal K Mantiuk, Dounia Hammou, and Param Hanji. 2023. HDR-VDP-3: A multi-metric for predicting image differences, quality and contrast distortions in high dynamic range and regular content. *arXiv preprint arXiv:2304.13625* (2023).

Rafal K Mantiuk, Param Hanji, Maliha Ashraf, Yuta Asano, and Alexandre Chapiro. 2024. ColorVideoVDP: A visual difference predictor for image, video and display distortions. *arXiv preprint arXiv:2401.11485* (2024).

William R Mark. 1999. *Postrendering 3D image warping: Visibility, reconstruction, and performance for depth-image warping*. The University of North Carolina at Chapel Hill.

Jonathan Martschinke, Jana Martschinke, Marc Stamminger, and Frank Bauer. 2019. Gaze-dependent distortion correction for thick lenses in hmds. In *2019 IEEE Conference on virtual reality and 3D user interfaces (VR)*. IEEE, 1848–1851.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Nitish Padmanaban, Robert Konrad, and Gordon Wetzstein. 2020. A Perceptual Eyebox for Near-Eye Displays. *Optics Express* 28, 25 (2020), 38008–38028. doi:10.1364/OE.408764

Daniel Pohl, Gregory S Johnson, and Timo Bolkart. 2013. Improved pre-warping for wide angle, head mounted displays. In *Proceedings of the 19th ACM symposium on Virtual Reality Software and Technology*. 259–262.

Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. 2025. Octree-GS: Towards Consistent Real-time Rendering with LOD-Structured 3D Gaussians. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2025).

Ethan Tseng, Ali Mosleh, Fahim Mannan, Karl St-Arnaud, Avinash Sharma, Yifan Peng, Alexander Braun, Derek Nowrouzezahrai, Jean-Francois Lalonde, and Felix Heide. 2021. Differentiable compound optics and processing pipeline optimization for end-to-end camera design. *ACM Transactions on Graphics (TOG)* 40, 2 (2021), 1–19.

Johannes Marinus Paulus Van Waveren. 2016. The asynchronous time warp for virtual reality on consumer hardware. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*. 37–46.

Yufei Wang, Zhihao Li, Lanqing Guo, Wenhan Yang, Alex C Kot, and Bihan Wen. 2024. Contextgs: Compact 3d gaussian splatting with anchor level context model. *Advances in neural information processing systems* 37 (2024), 51532–51551.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

Yaniv Wolf, Amit Bracha, and Ron Kimmel. 2024. Gs2mesh: Surface reconstruction from gaussian splatting via novel stereo views. In *European Conference on Computer Vision*. Springer, 207–224.

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. 2024a. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 19447–19456.

Zehao Yu, Torsten Sattler, and Andreas Geiger. 2024b. Gaussian opacity fields: Efficient adaptive surface reconstruction in unbounded scenes. *ACM Transactions on Graphics (ToG)* 43, 6 (2024), 1–13.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*. 3836–3847.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 586–595.

Wenguan Zhang, Zheng Ren, Jingwen Zhou, Shiqi Chen, Huajun Feng, Qi Li, Zhihai Xu, and Yueting Chen. 2024. End-to-end automatic lens design with a differentiable diffraction model. *Optics Express* 32, 25 (2024), 44328–44345.

Chumin Zhao, Andrea S Kim, Ryan Beams, and Aldo Badano. 2022. Spatiotemporal image quality of virtual reality head mounted displays. *Scientific Reports* 12, 1 (2022), 20235.

Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. 2001a. Ewa volume splatting. In *Proceedings Visualization, 2001. VIS'01.* IEEE.

Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. 2001b. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 371–378.