Shape-Shifting Splats: Realtime Context Translation for Gaussian Splatting in VR

Thomas Kernbauer*
Graz University of Technology

Simon Fussi†
Graz University of Technology

Philipp Fleck‡
VRVis GmbH

Clemens Arth§
Graz University of Technology

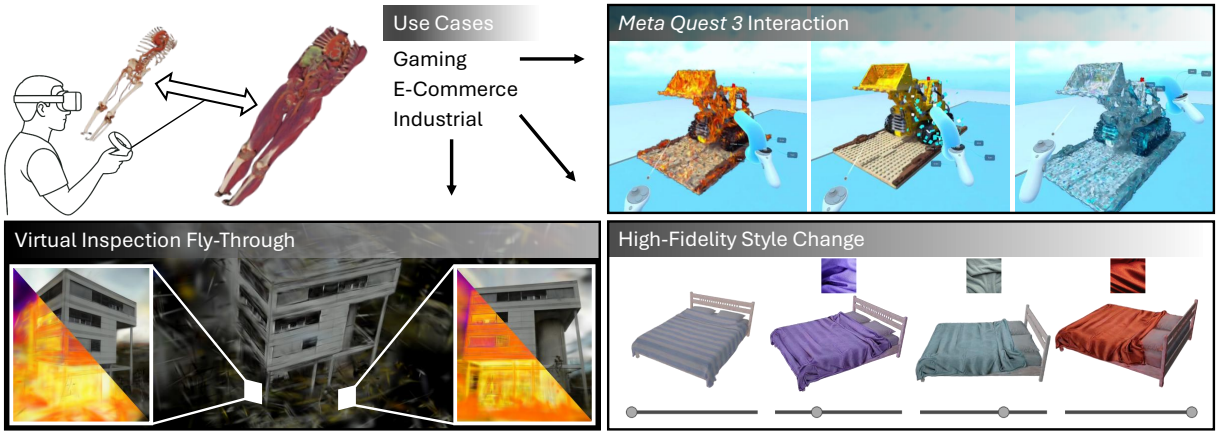

Figure 1: We introduce Shape-Shifting Splats, which enable us to change the appearance of 3D Gaussian Splats at render time. We demonstrate the power of our novel approach on three use cases, *i.e.,* Gaming (upper right), E-Commerce (bottom right), and Industrial (bottom left). In all of them, the appearance of our 3D Gaussian Splats can be dynamically changed after training at runtime on mobile VR devices.

Abstract

3D Gaussian Splatting (3DGS) has revolutionized novel-view synthesis by producing photorealistic renderings of virtually any object. However, 3DGS methods primarily support static scene representations, limiting flexibility and preventing direct application in scenarios that require visual variety. Our novel approach, *Shape-Shifting Splats*, addresses this limitation by enabling dynamic runtime transformations of Gaussian splats. Our versatile formulation allows changes in both color and geometry during render time, without increasing memory usage or introducing visual artifacts. Once trained, our Shape-Shifting Splats generate high-fidelity novel views while supporting modifications in geometry and appearance of the underlying model. In addition to applications in gaming and e-commerce, our method is applicable to mobile, industrial maintenance, and inspection. With multimodal data (*e.g.,* thermal and RGB), we create interactive 3D representations that allow inspectors wearing VR glasses to seamlessly switch between thermal and RGB views, helping to prevent oversights.

Index Terms: Gaussian Splatting, Multi-Model Inspection

1 Introduction

3D Gaussian Splatting (3DGS) has recently improved the quality of renderings in the graphics landscape. The biggest advancement is the ability to produce photorealistic renderings. But this comes at the cost of training. Based on a structure-from-motion (SfM) pipeline, the training stage computes the so-called *splats* in sufficient density to resemble the input data. The resulting 3DGS model, when trained with enough data, reaches photorealistic quality. However, what hides in plain sight is the static nature of the resulting data. Similarly to SfM, only one point in time is captured and later represented. Additionally, when considering different data domains, such as RGB, UV, IR, or artistic approaches, each domain has to be trained separately. For example, blending between realistic RGB and thermal views at runtime requires two sets of splats, which translates into two distinct training procedures.

Our work tries to fill this gap by allowing splats to shape-shift at runtime between target modalities. While our method can be applied in any 3DGS use case, we focus on three illustrative use cases. The **Art and Games** use case allows one to train a 3DGS model with a stylization, such as ”fire” and ”ice”. At runtime, when the 3DGS model is rendered, we define the power of effect additionally to the balance between fire and ice (Figure 1, *Lego*). Such mechanics (runtime transition between visual and geometric states of an asset) can be used in games with similar computational costs compared to traditional photorealistic assets, such as 3DGS models. We can think about *Mortal Combat: Sub-Zero*[1] turning everything into ice, without additional assets or efforts. Another important and trending area is **E-Commerce**. Platforms like *Ikea* or *Amazon* [2] utilize virtual content to showcase their products [16]. Others, such as *Splatter.App* [1] or *Arrival.space* [43] make use of 3DGS for a seamless online experience, but are not optimized for mobile hardware. Our approach reduces the amount of data necessary to present one product with different appearances (style and/or geometry) of similar quality, and enables 3D content creators to easily (*e.g.,* with text prompts) add different styles of products.

The third important area is the **industrial** field of maintenance and inspection. The so-called post-inspection or fly-through inspection is a tedious task that comes with man-built structures, such as

*e-mail: kernbauer@tugraz.at
†e-mail: simon.fussi@student.tugraz.at
‡e-mail: philipp.fleck@vrvis.at
§e-mail: clemens.arth@tugraz.at

[1] Mortal Combat: Sub-Zero

buildings, bridges, shop floors, ships, and other utility structures important to society. Since inspection plays a big role in predictive maintenance as part of traditional maintenance, it is worth exploring. Usually, training personnel, experts, scheduling sight visits, and inspecting the structure can include piping, electrical installations, concrete structure, rebar, etc. Sadly, there is much more structure to inspect as experts, leading to failures in inspection protocols likely missing crucial material degeneration, such as in the bridge collapse event in Dresden[2], the St. Laurent Tunnel[3], and others[4]. Thermal inspection, for example, is often part of the process, but is only applied ad hoc, since data re-mapping becomes inherently more difficult from plain video. Our work can use recordings such as LiDAR, thermal, and RGB to create interactive shape-shifting 3D representations. This is particularly interesting in the industrial setting where, for example, the inspector can use standalone Virtual Reality (VR) glasses to move through a photorealistic shopfloor and have a seamless transition between structure, thermal, and RGB representations, including all intermediate states. In more detail, as presented in Figure 1 (Industrial), the user can smoothly transition between the photorealistic RGB representation and the full change in shape to a pure thermal representation. Furthermore, due to the runtime-based nature of our approach, any state in between (*i.e.,* pure RGB to pure thermal) can also be computed at runtime.

In mythology and folklore, shape-shifting refers to the ability to change an object's appearance [49]. Similarly, our method allows us to change the color and geometry of a 3DGS at render time, which leads to the name of our work: *Shape-Shifting Splats*. Our contributions can be summarized as follows:

- a Gaussian Splatting framework, capable of adapting all Gaussian parameters, based on encodings learned in the training phase;
- an implementation, modified to work on a standalone, mobile VR headset as well as PC-VR systems; and
- a demonstration of the potential of our approach on three major domains: *Art*, *E-Commerce*, and *Industrial*.

2 Related Work

Our work is at the intersection of research on photorealistic rendering using Neural Radiance Fields (NeRF) and 3DGS, as well as on the application of these technologies in Mixed Reality and VR.

2.1 Style Transfer

The stylization — or in other words style transfer — of existing content has recently been revisited based on the advancements in NeRF [35] and further developments towards 3D Gaussian Splatting [19]. Chiang *et al*. [7] was one of the first methods to apply style transfer to NeRFs. They apply a network to a pre-trained NeRF model, which predicts the parameters for the appearance branch of the NeRF. Subsequently, Huang *et al*. [15] replace the color module of NeRFs with a style network and propagate spatial consistency back to the 2D decoder. ARF [56] introduces a feature matching approach based on nearest neighbors, replacing traditional Gram matrix-based loss functions for improved style transfer. Wang *et al*. [46] propose NeRF-Art, a text-guided framework that stylizes both the appearance and geometry of NeRF using contrastive learning and directional constraints. StyleRF [27] uses pre-trained CNNs to render full-resolution feature maps through volume rendering, unlike the RGB maps in standard NeRF. Their method involves sampling-invariant content transformation to improve multi-view consistency and efficiency. MM-NeRF [53] enhances stylization by incorporating multi-modal inputs and employs incremental learning to encode multiple styles within a single NeRF model, similar to our one-model strategy. However, their method relies on separate prediction heads to render each style, limiting flexibility. More recently, diffusion-based methods such as Style-NeRF2NeRF [8] have been adopted to achieve high-quality stylization with enhanced generative capabilities.

StyleGaussian [28] was the first method to apply stylization methods from radiance fields on a 3DGS model. They apply nearest neighbor feature matching (NNFM) between the rendered and stylized images to achieve style transfer. Other stylization methods for 3DGS distinguish themselves by utilizing text-encoding [22, 14] — which is one possible method to create *Shape-Shifting Splats*— or novel loss-functions [29]. Diffusion-based 3DGS stylization is proposed with Morpheus [51]. StyleSplat [17] utilizes a segmentation and tracking model to facilitate 2D object masks from multi-view input. 3D Gaussians encode geometry, color, and a feature vector, which a linear classifier decodes into labels, where then styles are applied by users to the labels. Multi-StyleGS [26] adds an additional trainable feature by using multi-layer perceptrons (MLPs) to decode the semantic category and a more advanced training data generation method. Local style transfer is supported by using the NNFM loss [17], which is augmented by bipartite matching. Furthermore, the VGG features utilized in [27, 28] can introduce multiview inconsistencies and memory issues, which are solved by a novel semantic style loss. The most similar work to ours is proposed by Saroha *et al*. [41]. They apply a multi-level hash grid encoding and a network to convert a pre-trained 3DGS model into a stylized one and achieve multi-style results by interpolating the latent space when encoding. However, their method changes only the color, not the geometry of the stylized scene.

Despite many efforts in 3DGS stylization, as of now, no method allows for changing and interpolating all aspects of 3D Gaussians across different styles solely at render time within a single model.

2.2 3DGS Application Cases

ClotheDreamer [31] uses Gaussian splatting for the generation of 3D garments. The novel "Disentangled Clothe Gaussian Splatting" method separates the clothed body into SMPL and garment parts. DCGS freezes body Gaussians for supervision, while Bidirectional Score Distillation Sampling (SDS) [37] is used to guide clothing and body RGBD renderings separately. In MeshGaussian [10], a mesh is constructed from calibrated images before initializing Gaussians, allowing real-time mesh deformation where normals drive the Gaussian splats during deformation.

VRSplat [45] utilizes recent developments in 3DGS, such as StopthePop [40], and minimizes the reprojection error for a better VR experience. StopthePop implements hierarchical per-pixel sorting along viewing rays of Gaussian splats, reducing popping artifacts in low primitive Mini-Splatting scenes. Many methods extend Kerbl *et al.* [19] work toward avatar generation [59, 30, 24], support for dynamic scenes [34, 25, 50], or encoding of thermal images [6, 32, 52], which is also of great interest for our industrial use case. Scaffold-GS [33] uses MLPs to decode implicit features to 3DGS parameters. This approach was further adapted by multiple methods, such as virtual avatar handling [3], and SLAM [48].

None of the currently proposed methods utilizes the effect to dynamically change 3D Gaussians for VR applications.

2.3 Virtual Reality in Industries

The immersive nature of VR, with its benefits for training and similar applications, has reached various industries with serious applications. Beh *et al*. [4] compared a VR training platform with paper-based training for utility inspection through user-experience and performance evaluations. The subjective results indicated that the proposed method was faster, but the new platform had better usability and user-friendliness. Zhang *et al*. [55] created a virtual inspection prototype for Android as a case study for annual fire inspection through building information modeling, VR, and indoor

[2] Stadtportal Dresden
[3] St. Laurent Tunnel
[4] Building Failures

real-time localization. Henstrom *et al*. [12] explored the effectiveness of a VR system for building inspection and compared it to traditional desktop-based building inspection, evaluating the effects on cognitive load, user experience, and usability. Similarly to the findings of Beh *et al*. [4], they also reported a more positive user experience, while users did not report an improvement in usability, they reported a lower perceived effort.

Omer *et al*. [36] compared traditional visual bridge inspection with a VR inspection of a digital twin, created using LiDAR and RGB images, similar to the earlier work by Jauregui *et al*. [18]. They highlight several advantages of VR inspection, including ease of use, repeatability, and less effort beyond data collection. However, inspection quality depends on LiDAR quality, limiting defect detection to sizes larger than 1 mm. Checa *et al*. [5] explore the natural interaction in VR for condition-based maintenance of induction motors, emphasizing VR-based teaching and training that approximates traditional methods while cutting costs and preventing damage to the equipment. The application was tested by students to provide basic knowledge and practical skills, showing a preference for VR. Yuhou *et al*. [13] utilized RGB images to generate point clouds and to extract camera poses, compensating for features absent in thermal images. Gaussians are initially randomly colored and adjusted to fit thermal images during training. Their findings highlight GS's superiority over photogrammetric point clouds, producing smoother images compared to point clouds that typically show holes in featureless areas, such as windows. Lu *et al*. [32] simultaneously train Gaussians for RGB and thermal modalities, and combine their respective losses for training, to optimize both modalities. They can render high-quality RGB and thermal images from both GS modalities, while relying only on cameras, such as the FLIR E6 Pro[5] or PortalCam[6]. **None** of the previously proposed methods have been used **on standalone VR headsets** before, nor have they investigated an **interactive shifting mechanism** to **change appearance and geometry during runtime**. Standalone VR allows for an untethered, independent, immersive experience without the need for a desktop computer. The cost of freedom is limited computation performance: Common devices such as the *Meta Quest 2* (Snapdragon XR2 Gen1), *Meta Quest 3* (Snapdragon XR2 Gen2), or the *Apple Vision Pro* (Apple M2 or M5+R1), where the latter being the most powerful, reach their computational limits relatively early. In other words, fewer Gaussians can be rendered, and (re)loading is costly. A common workaround is to use fewer resources, such as rendering fewer Gaussians, resulting in worse visual quality compared to a desktop computer. We demonstrate this effects in Section 5.

3 Method

Our approach builds on 3DGS and Scaffold-GS [33], with Scaffold-GS being our base implementation. We change the base implementation, in order to predict every parameter of the 3D Gaussians via shallow, fully-connected MLPs. Doing so, we are able to condition the MLPs in the training phase and are therefore left with the ability to change all attributes of the 3D Gaussians at render time (Figure 2). In other words, we train the MLPs (which predict the 3D Gaussians) to adhere to a control parameter (which encodes the wanted style or modality).

In this section, we explain the technical details of our approach. Starting with the preliminaries of 3DGS in Section 3.1, we explain the architecture of *Shape-Shifting Splats* and introduce our control encoding in Section 3.2. In Section 3.3, we elaborate on how to train our method for different use-cases.

[5] https://www.flir.eu/products/e6-pro/
[6] https://store.xgrids.com/?dt_id=2619070

3.1 Preliminaries

Given posed images, a sparse input point cloud is derived via a Structure-from-Motion (SfM) pipeline such as COLMAP [42] and used for initialization. The scene is represented with 3D anisotropic Gaussians G_i. For the sake of simplicity, we omit the index of each Gaussian i in the following section. Therefore, each 3D Gaussian G consists of a mean $\mu \in \mathbb{R}^3$, a covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{3\times3}$, an opacity value $\alpha \in \mathbb{R}$, and a color $\mathbf{c} \in \mathbb{R}^3$. $\mathbf{\Sigma}$ is parameterized by a scaling matrix $\boldsymbol{S} \in \mathbb{R}^{3\times3}$ and a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$, *i.e.,*

$$\mathbf{\Sigma} = \mathbf{R}\mathbf{S}\mathbf{S}^T\mathbf{R}^T, \tag{1}$$

to ensure that the matrix maintains semi-positive properties. To train a 3DGS model, a differentiable, tile-based rasterization is used. More precisely, the final color $\mathbf{c}_p$ at pixel p is calculated by alpha-blending N sorted Gaussians in the view frustum as

$$\mathbf{c}_p = \sum_{i\in N} \mathbf{c}_i \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j). \tag{2}$$

Scaffold-GS [33] is a modification of the original 3DGS pipeline. By implementing a slightly more implicit representation while achieving higher fidelity and lower memory requirements. They separate the scene into v voxels, where each voxel has one anchor point $\mathbf{a}_v \in \mathbb{R}^3$, one feature vector $\mathbf{f}_v \in \mathbb{R}^{32}$, a voxel scaling parameter $\mathbf{l}_v \in \mathbb{R}^3$, and k offsets $\mathbf{o}_v^k \in \mathbb{R}^3$ as parameters. The position μ of each Gaussian is derived with the voxel parameters as follows

$$\mu = \mathbf{a}_v + \mathbf{o}_v^k \mathbf{l}_v. \tag{3}$$

The other necessary parameters to splat the Gaussians, *i.e.,* $\mathbf{\Sigma}, \alpha$, and $\mathbf{c}$, are generated via fully connected Multi-Layer-Perceptrons (MLPs). As input to these MLPs, the voxel features $\mathbf{f}_v$ and the viewing direction $\delta_{\mathbf{x_c},\mathbf{a_v}} \in \mathbb{R}^3$ are used which is derived from an anchor point $\mathbf{a}_v$ and a camera position $\mathbf{x}_c \in \mathbb{R}^3$ as

$$\delta_{\mathbf{x_c},\mathbf{a_v}} = \frac{\mathbf{a}_v - \mathbf{x}_c}{\|\mathbf{a}_v - \mathbf{x}_c\|_2}. \tag{4}$$

Each voxel feature $\mathbf{f}_v$ is used to predict k Gaussians, with $k = 10$ in the original Scaffold-GS implementation. To summarize, Scaffold-GS predicts color, opacity, rotation, and scale using respective shallow and fully-connected MLPs. The subsequent rasterization and splatting process remains unchanged compared to the original 3DGS method.

3.2 *Shape-Shifting Splats* Architecture

We omit the feature-bank implementation of Scaffold-GS, as it introduced instability during training. In contrast to Scaffold-GS, we introduce two additional shallow, fully-connected MLPs to predict the voxel scales $\mathbf{l}_v$ and the offsets $\mathbf{o}_v^k$. Consequently, the stored Gaussian parameters are reduced to anchor points $\mathbf{a_v}$ and anchor features $\mathbf{f_c}$, while all other attributes are inferred via separate MLPs. With this formulation, no attributes of the predicted Gaussians are stored as external parameters.

This allows us to introduce a parameter $x \in \mathbb{R}$ to the MLP input to encode the current modality or style. Conceptually, this parameter conditions the MLPs to reconstruct the specific target domain. For instance, when rendering two separate modalities, we append $x = 0$ or $x = 1$ to represent RGB and thermal data, respectively. Rather than passing x directly, we map this low-dimensional input to a higher-dimensional embedding space via d-dimensional encoding $\Psi : \mathbb{R} \to \mathbb{R}^d$. Such encodings aid MLPs in representing complex content, as demonstrated by Mildenhall *et al*. [35].

During the training phase, we pass the encoding $\Psi(x)$ to the model with a loss function corresponding to the given input. At

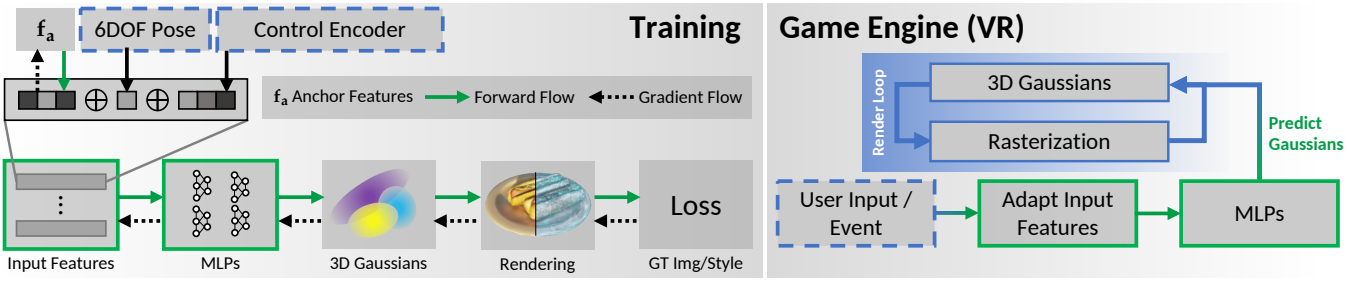

Figure 2: While training (left), the feature vectors $\mathbf{f_a}$, the 6DOF viewing direction $\delta_{\mathbf{x_c},\mathbf{a_v}}$, and the control encoding $\Psi(x)$ are concatenated ($\oplus$) and forwarded to the fully-connected MLPs, which predict the 3D Gaussians. They are then rendered using the standard 3DGS rendering pipeline. The gradient is calculated via the standard 3DGS losses and stylization losses (if required). The style source can be either a text prompt or an example image. In the game loop (right), a user input or event can trigger the shape-shifting of a trained model by changing the control encoding (blue dashed outline) and querying the MLPs. Note that only the color MLP requires the current pose in its input features. Once the MLPs are queried, the standard 3DGS render loop applies.

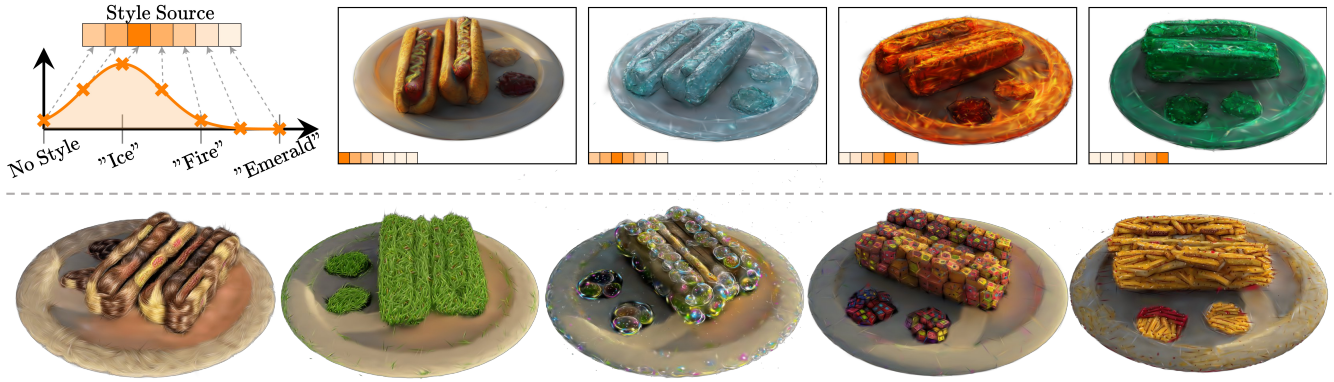

Figure 3: With *Shape-Shifting Splats*, we achieve render time control over 3D Gaussians. Therefore, all the renderings in the first row of this figure are done within a single 3D Gaussian Splatting model, which is controlled by its encodings (left). The orange vector controls the style direction, *i.e.,* different styles from text prompts. In the second row, we showcase the ability to change not only color but also geometrical aspects of the underlying model. The five shown renderings are stylized with the text prompts "Hair", "Grass", "Bubbles", "Cubes", and "Fries". In all renderings, not only the color but also the geometric properties differ from those of the original object.

inference, the provided encodings are reflected in the predicted splats, yielding the properties associated with the specific training modality. Because these encodings dictate the characteristics of the splats predicted by the MLPs, we refer to them as *control* encodings throughout this work.

A schematic illustration of our workflow is shown in Figure 2. The features $\mathbf{f_a}$ are concatenated with the control encoding $\Psi(x)$ and, in case of the color MLP, with the 6DOF viewing direction $\delta_{\mathbf{x_c},\mathbf{a_v}}$. Each Gaussian parameter (*i.e.,* scale, rotation, offset, color, and opacity) is predicted by its respective MLP. For example, the color $\mathbf{c}$ of a Gaussian is computed as

$$\mathbf{c} = \mathcal{F}_{\text{color}}\left(\mathbf{f_a} \oplus \delta_{\mathbf{x_c},\mathbf{a_v}} \oplus \Psi(x)\right), \tag{5}$$

where $\mathcal{F}_{\text{color}}$ denotes the color MLP and $\oplus$ denotes concatenation. The MLPs predict for each anchor feature k parameters. Once all parameters required for a 3D Gaussian are inferred, we perform the standard 3DGS workflow and render the images via rasterization. Hence, our approach can be implemented in any framework using the tile-based rasterization approach of the original 3DGS method.

We adopt a 1D Gaussian, similar to the Gaussian encoding described by Zheng *et al*. [58], as control encoding $\Psi(x)$. Formally, we define the Gaussian embedder $\psi : \mathbb{R} \to \mathbb{R}$ and the resulting encoding $\Psi(x)$ as

$$\psi(t,x) = \exp\left(-\frac{\|t-x\|^2}{2\sigma^2}\right) \quad \text{and} \tag{6}$$

$$\Psi(x) = [\psi(0,x), \psi(s,x), \ldots, \psi((d-1)s,x)]^\top, \tag{7}$$

where σ is the standard deviation and s the sampling interval. The plots on the left in Figure 3, illustrate an example of our sampling for the control encoding. An advantage to using a 1D Gaussian as an embedder is the ability to interpolate between encodings. Other encodings, such as the frequency encoding in the original NeRF [35] have different properties regarding interpolation, as encodings which are far afield in the input space (*e.g.,* $\Psi(x=0)$ and $\Psi(x=1)$), might be closer in the embedded space than other encodings which are closer in the input space, and vice versa, which is due the similarity in different frequency bands. When trained with multiple styles, this could lead to reappearing styles at arbitrary x.

3.3 Training and Inference

To encode RGB images into our method, we follow the standard 3DGS training by Kerbl *et al*. [19]. There, the L1 distance and the similarity term SSIM [47], between the ground truth image and the rendered image, are calculated as

$$\mathrm{L}_{\text{3DGS}} = \lambda_{\text{3DGS}}\mathrm{L}_1 + (1-\lambda_{\text{3DGS}})\mathrm{L}_{\text{D-SSIM}} + \mathrm{L}_{\text{reg}}. \tag{8}$$

To encourage fewer Gaussians and avoid large-scale and low-opacity Gaussians, we employ opacity α and scale $\mathbf{S}$ regularization similar to the regularization proposed by Kheradmand *et al*. [20]. Therefore, L_{reg} results in

$$\mathrm{L}_{\text{reg}} = \lambda_\alpha \frac{1}{N}\sum_i^N |\alpha_i| + \lambda_{\mathbf{S}} \frac{1}{N}\sum_i^N \text{mean}(|\mathbf{S}_i|) \tag{9}$$

with λ_α and $\lambda_\mathbf{S}$ being hyperparameters. To effectively prune unnecessary Gaussians, we use an anchor-wise opacity and view thresholds. Therefore, if the Gaussians spawned by an anchor do not contribute to the scene, the anchor gets pruned. With an anchor-based approach, the memory requirements when saving a scene are reduced significantly, as only the anchor positions, corresponding features, and the weights for each MLP have to be stored. We encode other modalities by adding images with a different encoding to our training set. As we know at training time, which modality the current image is from, the vector passed to the MLPs is the anchor feature concatenated with the control encoding of the respective modality. The input images do not need to be aligned; however, their relative pose to each other has to be known.

When a change in style is required, we apply a stylization loss. Here, we rely on the CLIP [39] features, similar to previous stylization methods [14, 9, 23] as a loss function. We choose the text prompt "A Photo" as the starting location in the CLIP space for the directional loss. Furthermore, we utilize the *ImageNet* prompt templates to enhance the CLIP embedding. For a style source $\mathcal{S}$, *i.e.,* a text prompt or an image, the CLIP-direction loss is calculated as

$$\Delta\mathbf{T} = \Phi_{\text{CLIP}}(\mathcal{S}) - \Phi_{\text{CLIP}}(\text{"A Photo"}), \tag{10}$$

$$\Delta\mathbf{I} = \Phi_{\text{CLIP}}(\mathcal{R}(\hat{I}^n_{\text{R}})) - \Phi_{\text{CLIP}}(I_{\text{GT}}), \tag{11}$$

$$p_n = D(\Delta\mathbf{T}, \Delta\mathbf{I}), \tag{12}$$

$$\mathcal{L}_{\text{CLIP}} = \frac{1}{N}\sum_n p_n, \tag{13}$$

with Φ_{CLIP} being the CLIP encoder, $\hat{I}^n_{\text{R}}$ being the n-th image patch of the rendered (stylized) image and I_{GT} the ground truth image from the input dataset. $D(\cdot)$ represents the cosine distance between two vectors. With $\mathcal{R}(\cdot)$, we denote random cropping and perspective distortion, which are applied to the rendered image. To focus on more global style attributes, we can also omit these augmentations, resulting in a single patch p for the loss calculation. An example of our stylization result is showcased in Figure 3. The first row shows $n = 4$ different stylization examples. Therefore, the respective control encoding is $\Psi(\frac{l}{n})$ with $l \in [1, \dots, n]$. For a certain stylization, we employ the stylization loss of the wanted style at a fixed location x in the control encoding $\Psi(x)$. At $\Psi(0)$, we encode the original scene, *i.e.,* "No Style". In this example, we enforce three different stylization losses (encoded with the orange control encoding in the first row).

To exclude certain parts of the object from the stylization, *e.g.,* the plate in the second row of Figure 3 or the armrests in Figure 6, we pass a segmentation mask with the pixels excluded in the stylization process to our training. Hence, the loss is only calculated on the pixels outside the mask, avoiding a change in appearance in the masked pixels. As an additional regularization loss within the stylization process, we employ the background loss $\mathcal{L}_{\text{BG}}$ as proposed by Howil *et al*. [14]. It is defined as the L1 distance between the ground truth pixels of the background in the rendered image and the pixels in the background of the GT image, and prevents the model from introducing unwanted style artifacts in the background. Note that our method is not limited to CLIP-based stylization losses.

Inference. Once training is complete, only the forward flow in Figure 2 is necessary to render images. The control encoding passed to the MLPs determines the object to be rendered at runtime (render-time). Therefore, the MLPs are invoked whenever a change in the encoding is necessary. An exception to this is the color MLP. To account for scene colors that change with the viewing direction, the MLP must be re-evaluated whenever the viewing angle shifts. All other MLPs do not need to be inferred for every viewing direction; instead, each MLP predicts k parameters for each anchor point.

4 Implementation and Examples

We chose *Unity* as our demonstration platform; however, our approach is not limited to Unity, as other game engines, such as *Unreal Engine*[7], are capable of rendering Gaussian splats as well.

We focus on three representative use-cases around Gaming, E-Commerce, and Industrial to demonstrate the strength and versatility of our approach. In addition to these areas, there are many more, such as accessibility modes in VR or architecture worth exploring, but they remain outside the scope of this work. However, we present feasibility examples for solving dedicated problems in each category. To recap, what distinguishes our approach from others is the ability to encode multiple versions of objects in a single Gaussian splatting model, while maintaining low memory requirements and high reconstruction quality. One single model is capable of interpolating between different appearances at runtime, utilizing shallow MLPs, without the need for additional models or retraining.

4.1 Real-Time Ready Implementation in Unity

Our implementation builds upon the work of the existing 3DGS rendering in Unity[8], and the respective improvements by Kleinbeck *et al*. [21]. To predict the Gaussian Splats from the anchors, feature vectors, and control encoding, we leverage the `Unity Sentis`[9] framework. With this, we are able to build the necessary model graphs and load the weights of each MLP at runtime. Moreover, each MLP is translated via Sentis to run as a custom compute shader. To avoid unnecessary CPU-GPU readbacks, we upload the anchor features and their respective positions at initialization and pin them in the GPU memory. Through the Sentis framework, we can access the computed 3D Gaussians in compute buffers and pass them to the standard 3DGS renderer. To change the appearance at runtime, we simply invoke the MLPs with a different encoding $\Psi(x)$. Since the number of predicted Gaussians stays the same (only their attributes change), no reallocation of GPU memory is necessary. The MLPs are loaded at initialization and serialized from `.onnx` to `.sentis` models[10].

To avoid unnecessary rendering of all predicted Gaussians, we cull them if their opacity is negative. This is necessary because Gaussians that are generated and trained for a specific modality may or may not contribute to the current, visualized modality. In Gaussian rasterization, each 3D Gaussian is represented as a quad. To avoid overdraw and achieve smooth rendering, the quad should fit the respective Gaussian tightly. The standard 3DGS rasterization decomposes the eigenvalues of the anisotropic 3D Gaussians into their respective rotation and scale components, *i.e.,* their 2D Eigenvalues in screenspace. Although the original EWA Splatting [60] uses a larger rectangular box that encompasses all affected tiles on screen, modern 3DGS rendering implementations, such as GSplat [54], achieve tighter fits around the splats by computing a non-axis-aligned quad and further incorporating the splat alpha value into the calculation of the bound rectangle. We use the calculated size of the screenspace quad to discard a Gaussian with a maximum axis above a clipping threshold τ_λ, when rendering on mobile hardware. This helps to speed up rendering, as Gaussians with a mean outside, but with large splats inside the view frustum, are not processed. Figure 4 shows an example with a rather large τ_λ, together with the non-processed Gaussians in blue.

For mobile rendering, we add a second threshold λ_{dist} to discard distant Gaussians. This is helpful, as the Gaussian formulation encourages splats that are far away from the camera to represent far away scenery (*e.g.,* the sky). When blended and projected onto the

[7] Introduction to Gaussian Splatting in Unreal Engine

[8] `https://github.com/aras-p/UnityGaussianSplatting`

[9] `https://unity.com/products/sentis`

[10] The source code will be made publicly available on Github upon acceptance of this paper.

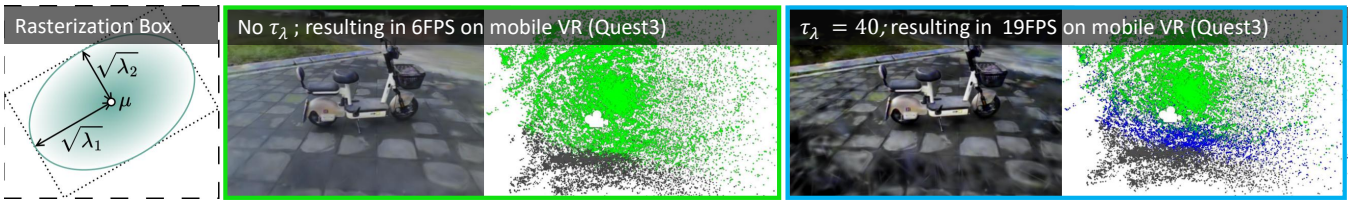

Figure 4: The rendering performance on mobile VR largely depends on τ_λ. The `Rasterization Box` shows our quads of our rendering approach (small dots) and the axis-aligned box from Zwicker *et al.* [60]. Without τ_λ threshold (left), the scene renders at 6 FPS on mobile VR. With τ_λ threshold (right), we reach 19 FPS on mobile VR. The birds-eye-view point clouds depict the mean position of rendered splats (green points), clipped points (blue points), and splats outside the view frustum (gray points), respectively. The scene is the `Bike` scene with 790k splats from ThermalGaussian [32].

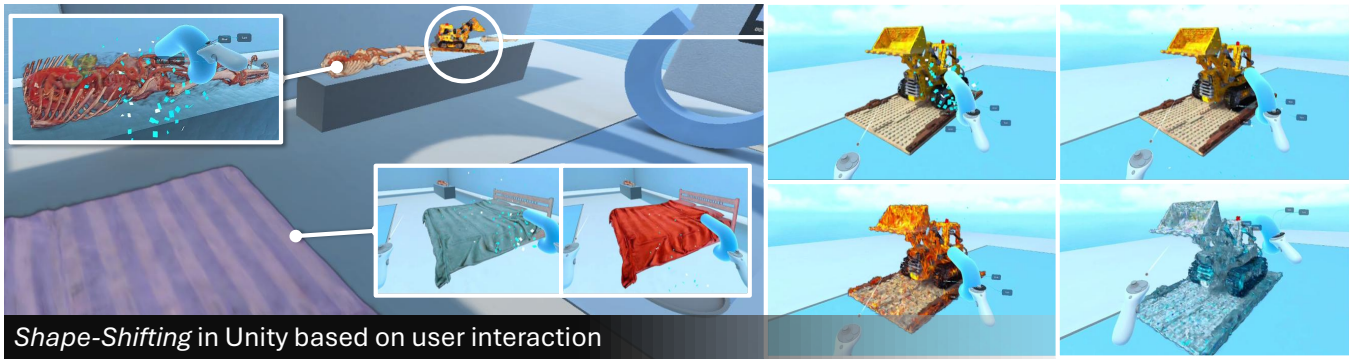

Figure 5: Example of changing the style and shape of three different objects in a gaming scenario at runtime on the *Meta Quest 3* with *Shape-Shifting Splats*. One object stems from the anatomy dataset [21] and one is a bed switching between individual shapes. The last object is a Lego toy excavator. The user can use a confetti gun to interact with each model and seamlessly transition along the encoded shapes.

screen from the image position in the training set, these far away appear correct. However, in out-of-distribution views, they produce unwanted artifacts such as floaters and contribute negatively to the scene content and the rendering speed (especially on mobile devices). With λ_{dist}, we create a threshold to stop the rendering of the Gaussian at a certain distance from the camera. This increases the framerate on unbounded, non-object-centric scenes, *e.g.,* the industrial inspection use case.

4.2 Art and Games

Ever since, games have been constrained by limited hardware resources. Similarly to fitting pixel art into small single-core systems, the challenge nowadays lies in fitting many Gaussian splats onto modern hardware. Games on mobiles already demand high-quality graphics, yet mobile hardware delivers computational performance an order of magnitude less than that of desktop computers.

In the gaming use case, we deliberately focus on individual object transformation, such as burning or freezing. The traditional way usually uses a less detailed model to perform manipulations. If we think of games where the player can manipulate objects, *e.g.,* burning something with a flame thrower, the effect would have been modeled additionally with a "no-effect" version for the default case. With our technique, we can encode the state of the object, such as "burning", "burned", "frozen", "charcoal", etc. For example, shooting a hot dog with ice will freeze and become ice; shooting it with fire will remove the ice and "normalize" it again, as shown in Figure 3. We based our sample game on Unity's VR template and added 3DGS objects created with our method. The player can interact with the added objects through collision or by shooting them with a confetti blaster, which is part of the template. At collision time, the MLP-based encoders are activated with a new value, instantly producing a new 3DGS state resulting in a different style. Figure 5 shows how a number of objects in the scene can be changed. In standalone VR, the photorealistic quality might be reduced to fit mobile devices such as the *Meta Quest 3*. In PC-VR, the capabilities of modern CPUs and graphics cards can be further exploited to establish high-fidelity content.

Recently, a continued trend towards photorealistic product placement in commercials[11] has been observed. Usually, the amount of data and the graphics power of the client devices, as well as the data connection, have been the limiting factors. Companies, such as Ikea, run an online warehouse with thousands of products for virtual preview in the browser[12] or in AR and VR using mobile devices. Photorealistic content has not yet reached online platforms in production. Furthermore, such platforms use man-made models, which are produced under high effort and, in most cases, do not reach a photorealistic appearance. Objects that are not virtually created are basically non-existent in the e-commerce world besides photographs.

With our presented method, we can drive multiple styles or variations in one Gaussian Splatting model, ready for mobile platforms such as the *Meta Quest 3* (for example, the bed in Figure 5) or similar devices. Furthermore, we demonstrate our benefits in e-commerce on a number of examples and compare our multivariate version against what it takes to cover the same variations in a more traditional way, such as having one model per style or one texture per style. Figure 6 demonstrates not only different styles, but also different geometry for the garment, where, in contrast to the `Default` garment, the leather version naturally has more crinkles, *e.g.,* `Black Leather`. Moreover, generating training data for *Shape-Shifting Splats* is straightforward, using tools such as BlenderNeRF [38] to create 3DGS datasets from synthetic 3D models. To apply our method, the content generator either takes different appearances as 3D models or applies stylization losses. For this, simple style prompts (*e.g.,* `Black Leather` in Figure 6) or images depicting the style are needed.

[11] Luma Labs and e-Commerce (blog.playcanvas.com).
[12] `https://www.ikea.com/at/en/home-design/`

Figure 6: Prompt based style alterations at runtime based on the model `Chair` [44]. "Default" is the unchanged model, and the other labels (*e.g.,* `Red Leather`) were used as prompts to create the Shape-Shifting Splats.

4.3 Industrial

In industries, the demand for inspection and maintenance has been present for years and has been driving technological advances for decades. The constant battle against growing amounts of data, including recordings, sensor data, measurements, documents, etc., has led to a rethink of approaches and methods on a yearly basis. With the rise of 3DGS, photorealism is now reachable at scale for visual inspection tasks, in a freely navigable manner, in contrast to traditional video.

As often, image data are not enough as a decision-making basis. The combination of different data domains allows experts to gather deeper insight into the nature of the targeted domain. For example, examine hot steam pipes through a factory to check stressed material, which is immediately visible in the thermal domain through higher temperatures. Not all areas in factories and similar facilities are easily and securely accessible by humans and therefore require equipment such as rope cameras, drones, or other UAVs. Our proposed approach takes multimodal data, such as RGB-video together with thermal data, and provides an interactive 3DGS model allowing seamless transition through said domains. One beneficial aspect is that the expert can be decoupled from the facility, allowing much higher flexibility. A schooled novice could record the data using panoramic video capturing devices, while the expert could perform a photorealistic virtual fly-through. By doing so, the expert can interact with the data domain at no additional cost within one model. In Figure 7, we demonstrate that a user interactively travels through a scene of an office building while visually inspecting its thermal signature for any irregularities. Figure 8 depicts the same building but uses a selective visualization to only partially overlap with the RGB depiction.

The presented examples are representative for a wide range of use cases usable with our presented method. Our exemplary implementation is a demonstrator and cannot be considered a full application, solely focusing on interactively showcasing a range of varying examples interesting for different industries, such as games, e-commerce, maintenance and others.

5 Evaluation

In the following, we briefly discuss the details of the training in our method. We evaluated our approach by comparing it to previous methods in two distinct domains - anatomy visualization in VR [21] and thermal scene representation [32]. To highlight the rendering quality and memory efficiency of *Shape-Shifting Splats* we report the Peak Signal-to-Noise Ratio (PSNR), the Structural Similarity Index (SSIM) [47], and Perceptual Similarity Index (LPIPS) [57] as well as the size of the resulting splats (in MB) for different scenes and use cases. Furthermore, we report the rendering speed of our Unity implementation on three platforms, such as standalone PC, PC-VR, and mobile VR.

5.1 Training Details

Our training phase has the same requirements as the standard NeRF/3DGS training: posed images. With an off-the-shelf SfM method, we derive a sparse point cloud and initialize the splats. After training the scene with the above described $\mathcal{L}_{\text{3DGS}}$, we alternate between $\mathcal{L}_{\text{3DGS}}$ and the respective control losses when necessary, *i.e.,* $\mathcal{L}_{\text{Style}}$. We use an *NVIDIA RTX 4090* GPU with an *Intel Core i7-12700K* CPU with 64GB of RAM for all our experiments. Both initial training and stylization take less than 10 minutes on this hardware.

After the initial training phase, we do not grow or prune Gaussians. The 1D Gaussian embedder is of dimension $d = 16$ with a standard deviation of $\sigma = 0.04$. For each anchor point $\mathbf{a}_v$, we predict $k = 10$ offsets. We choose the feature vectors $\mathbf{f}_a \in \mathbb{R}^{16}$. Our fully connected MLPs utilize a rectified linear (ReLU) activation function. The learning rates $\{\eta_f, \eta_a\}$ for the features $\mathbf{f}_v$ and anchor points $\mathbf{a}_v$ are 0.0075 and 0.001, respectively. For the MLPs $\{\mathcal{F}_{\text{opa.}}, \mathcal{F}_{\text{color}}, \mathcal{F}_{\text{quaternion}}, \mathcal{F}_{\text{scale}}, \mathcal{F}_{\text{offset}}\}$, we employ learning rates of $\{0.0005, 0.008, 0.0004, 0.001, 0.001\}$. To scale the loss functions, we use $\{5, 45, 0.2, 0.01, 1\text{e}{-9}, 1\text{e-}9\}$ for respective factors $\{\lambda_{\text{CLIP}}, \lambda_{\text{BG}}, \lambda_{\text{3DGS}}, \lambda_{\alpha}, \lambda_{\mathbf{S}}\}$. To train the 3DGS model on transparent scenes, we set the background to random colors in the training phase, which enforces the prediction of alpha values.

5.2 Quantitative Analysis

First, we evaluate our method on the anatomy dataset against the Multi-Layer Gaussians by Kleinbeck *et al.* [21]. In their method, they train standard 3DGS with an additional parameter, indicating to which layer the Gaussians belong. Their anatomy dataset consists of three body parts, each having three distinctive layers (*bone*, *muscle*, and *organs*). To train our method, we pass the control encoding $\Psi(0)$ for images from the bone layer, $\Psi(0.5)$ for images from the organ layer, and $\Psi(1)$ for images from the muscle layer. To enable a fair comparison, we evaluate our approach on every 8th image, identically to the Multi-Layer Gaussians method.

Table 1 underlines the reconstruction quality of our method. Our method achieves higher performance metrics in terms of PSNR, SSIM, and LPIPS, while consuming less on-disk memory. Moreover, we do not cluster or compress our resulting Gaussians while using less memory than the multi-layered Gaussian approach. Note that in their method, the file size is scaled down via compression and cross-layer clustering, whereas our files are not reduced in size after training. As a baseline, we also evaluate the original 3DGS [19] method. Note that this evaluation requires training nine 3DGS models (*i.e.,* three per scene).

Table 2 provides the average results on 10 objects for thermal and RGB modality from the ThermalGaussian [32] dataset. Again, our method proves its efficacy in both representation quality and memory consumption. Overall, our method performs on par with other thermal 3GDS approaches while using substantially less on-disk memory. Notably, training thermal and RGB images in a single model improves performance compared to training them separately.

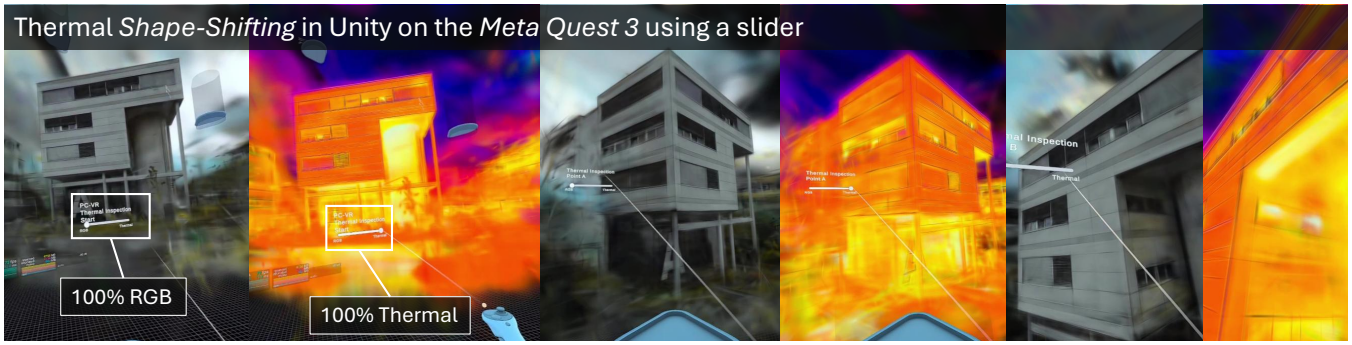

Figure 7: Thermal Shape-Shifting Splats: Three pairs of images with different viewpoints, resembling a user inspecting a building thermally while wearing the *Meta Quest 3*. Each pair shows the extremes between purely RGB and purely thermal encodings. Using the slider, the user can choose between the modes (and any mixed mode in between), which are baked into the 3DGS model.

Figure 8: We apply a selector for the different modalities, as described in Figure 4 to achieve a selective visualization. Scene from the ThermoNeRF [11] dataset.

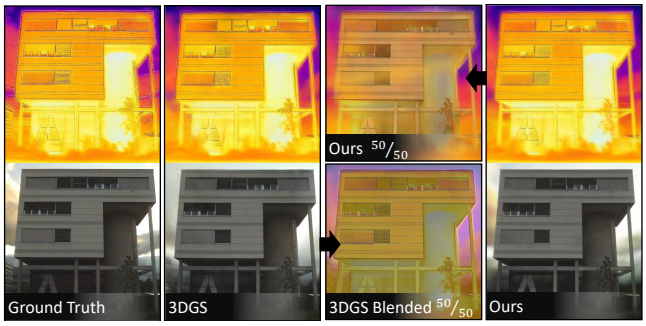

Figure 9: Qualitative comparison between the ground truth images, 3DGS [19] renderings, and our approach. Our method only requires one model for all shown renderings, while not introducing any artifacts compared to the other models.

This is due to the fact that the joint training benefits the reconstruction of the underlying geometry. In Figure 9, we provide a quantitative comparison between two blended 3DGS methods and our method. As depicted, our approach can produce blended views as well as RGB and thermal renderings within a single model, whereas 3DGS requires two separate models to blend both renderings.

5.3 Runtime Evaluation

We evaluate the runtime of our method on two representative scenes: the *Leg* scene with three layers from the multilayer Gaussian splatting dataset and the *Bike* scene with RGB and thermal information from the ThermalGaussian [32] dataset. All experiments are carried out on an *NVIDIA 4090 RTX* for the PC and the PC-VR setups, and on *Meta Quest 3* VR glasses. We measure the render time in Unity for these two scenes on different hardware, reporting

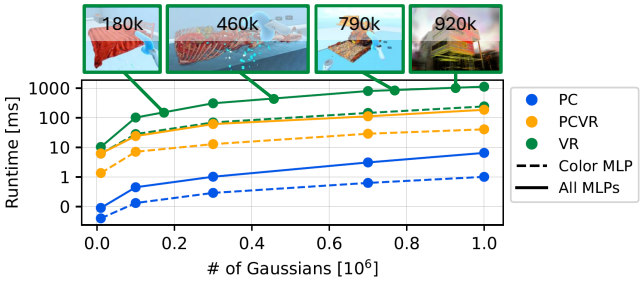

Figure 10: Runtime evaluation of the MLP inferences on the PC, PCVR, and on the mobile *Meta Quest 3*, with exemplary scenes showcasing the size of the scene. The runtime depends on the number of input features. Note that in this experiment, we used all Gaussians from the scene (not exclusively the input features in the view frustum).

the mean runtime over 1000 frames.

As shown in Table 3, both scenes can be rendered at sufficient speed on a standalone PC and with the PC-VR solution. On mobile hardware, we achieve adequately high frame rates in object-centric standalone scenes. For larger unbounded scenes (*e.g.,* industrial scenes in Figure 7), using τ_λ and τ_{dist} are necessary engineering trade-offs to maintain a smooth visualization experience. Similarly, the runtime of the MLPs depends on the number of input features and the size of those features, as MLPs primarily perform matrix multiplications. In Figure 10, we report the results of our evaluation for a shape-shifting operation (where all MLPs are queried) as well as for querying only the color MLP on different hardware. A complete shape-shifting operation, *i.e.,* querying all MLPs, completes within one second, even for large scenes (1 million Gaussians), in the standalone VR setting. By contrast, on a PC, it requires less than 10 ms.

5.4 Limitations

The inherent limitations of 3DGS regarding runtime are amplified when deployed on mobile devices. For example, the view-dependent sorting order leads to a bottleneck that requires significant computing resources. This results in an upper bound on the maximum number of splats for mobile VR hardware. In our experience, scenes with more than 2 million splats could not run smoothly on the headset without additional tweaks.

In addition to these inherited limitations, our method induces a maximum computational load when MLPs are invoked. Even though our MLPs are shallow and employed in a compute shader, the amount of required operations (when changing the encoding, we deploy all five MLPs) leads to a short drop in the frame rate on mobile VR. This could be mitigated by deploying only specific MLPs; however, doing so would restrict the flexibility of our method.

Table 1: Comparison of the reconstruction quality of anatomy data between 3DGS [19], Multi-Layer Gaussians [21] (ML-3DGS), and our approach. Best in **bold**. Each scene (*i.e., Leg*, *Lowerbody*, and *Fullbody*) consists of three layers. We report the average performance metrics over all layers with the required disk file size (in MB). For ML-3DGS, we report the file size of their high-quality model after compression (roughly a third of the .ply file size). Our method achieves higher reconstruction values while requiring less on-disk memory.

Method	Leg				Lowerbody				Fullbody			
	PSNR ↑	SSIM ↑	LPIPS ↓	Mem ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Mem ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Mem ↓
3DGS [19]	37.11	0.969	0.044	45.9	37.7	0.97	0.041	35.6	36.41	0.981	0.031	101.1
ML-3DGS [21]	36.93	0.973	0.043	22.9	35.49	0.966	0.052	16.7	35.71	0.976	0.042	33.9
Ours	**38.52**	**0.978**	**0.037**	**3.5**	**38.25**	**0.978**	**0.035**	**4.2**	**36.56**	**0.984**	**0.025**	**3.6**

Table 2: Comparison of the reconstruction quality of the multimodal data, *i.e.,* **thermal** (Ther) and **RGB** images, from the ThermalGaussian (TGS) dataset of Lu *et al.* [32]. Best in **bold**. We report the average performance metrics over all 10 scenes for each modality. Our method performs on par with or better than the other thermal reconstruction methods. The asterisk (*) indicates the results of two models (trained separately for each modality).

Methods	Mode	PSNR ↑	SSIM ↑	LPIPS ↓	Mem.
ThermoNeRF [11]	Ther	21.13	0.765	0.28	412
	RGB	18.46	0.586	0.303	
3DGS* [19]	Ther	24.31	0.859	0.206	165
	RGB	23.2	0.82	0.219	59
TGS_{OMMG} [32]	Ther	**25.64**	0.883	0.169	136
	RGB	24.34	0.8	0.203	
$TGS_{MSMG,\ MR}$ [32]	Ther	25.09	0.88	0.189	18
	RGB	24.21	0.84	0.235	
Ours*	Ther	24.48	0.878	0.128	9
	RGB	24.29	0.84	0.169	6
Ours	Ther	25.17	**0.886**	**0.118**	**10**
	RGB	**26.13**	**0.881**	**0.168**	

Table 3: Rendering times in ms of our Unity implementation. We evaluate the impact of τ_λ and τ_{dist} for two scenes. Here, we use the `RadixSort` implementation from Multi-Layer Gaussians [21]. For the bike scene, we sort every 20th frame for one eye.

τ_λ	τ_{dist}	Leg (456k)			Bike (790k)		
		PC	PCVR	VR	PC	PCVR	VR
×	×	1.2	14.1	14.1	1.2	14.2	161.5
×	20	1.2	14.0	14.1	1.2	14.2	52.2
×	40	1.3	14.1	14.1	1.2	14.2	64.1
10	×	1.2	14.1	14.1	1.2	14.2	26.5
20	×	1.3	14.1	14.1	1.2	14.2	29.4
10	20	1.3	14.1	14.1	1.2	14.2	24.4
20	40	1.2	14.0	14.1	1.2	14.2	25.4

Further limitations come with the sparsity of available datasets for our thermal inspection use case. While there exist RGB-Thermal datasets for 3DGS, they often lack diversity in viewpoints, causing the center of the scene to be well reconstructed only under the given (non-diverse) viewpoints.

6 Conclusion

In this work, we introduce *Shape-Shifting Splats*, a 3D Gaussian Splatting method which can change its attributes at runtime. Our approach allows us to encode control information into feature vectors used for rendering. With this, we are able to encode multiple styles or modalities within a single 3DGS model in the training phase, and consequently change the appearance at render time by changing the control encoding. More precisely, we leverage shallow, fully-connected MLPs to predict the 3D anisotropic Gaussians.

Although we achieve sufficient frame rates on commercially available mobile VR headsets (*e.g.,* the *Meta Quest 3*), the computational limits are reached with increasingly large scenes with many splats. In addition, changing the encoding on the VR headset comes with a peak in compute operations. This is not noticeable in desktop setups or PC-VR, but is noticeable in mobile VR headsets.

Still, we are able to encode large scenes on mobile VR headsets, which allows us to present three use cases where *Shape-Shifting Splats* has an interesting application, *i.e.,* gaming, e-commerce, and industrial inspection. Overall, our method maintains real-time rendering speed while enabling control over geometry and appearance at inference time. This makes our approach particularly well-suited for augmented and virtual reality, where dynamic appearance changes are essential, at the same time surpassing previous methods in terms of quality.

Acknowledgments

The VRVis GmbH is funded by BMK, BMAW, Tyrol, and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (911654) which is managed by FFG.

Funded by the European Union under Grant Agreement No. 101092861. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Splatter.App. https://splatter.app/. Accessed: 2025-09-01. 1

[2] Amazon. Amazon AR View. https://www.amazon.com/products. Accessed: 2025-09-01. 1

[3] S. Aneja, S. Weiss, I. Baeza, P. Chandran, G. Zoss, M. Nießner, and D. Bradley. ScaffoldAvatar: High-Fidelity Gaussian Avatars with Patch Expressions. *arXiv*, abs/2507.10542, 2025. 2

[4] H. J. Beh, A. Rashidi, A. Talei, and Y. S. Lee. Developing engineering students' capabilities through game-based virtual reality technology for building utility inspection. *Engineering, Construction and Architectural Management*, 29(7):2854–2877, 2022. 2, 3

[5] D. Checa, J. J. Saucedo-Dorantes, R. A. Osornio-Rios, J. A. Antonino-Daviu, and A. Bustillo. Virtual Reality Training Application for the Condition-Based Maintenance of Induction Motors. *Applied Sciences*, 12(1):414, 2022. 3

[6] Q. Chen, S. Shu, and X. Bai. Thermal3D-GS: Physics-induced 3D Gaussians for thermal infrared novel-view synthesis. In *European Conf. on Computer Vision (ECCV)*, pp. 253–269, 2024. 2

[7] P.-Z. Chiang, M.-S. Tsai, H.-Y. Tseng, W.-S. Lai, and W.-C. Chiu. Stylizing 3D Scene via Implicit Representation and Hypernetwork. In *IEEE Winter Conf. on Applications of Computer Vision*, 2022. 2

[8] H. Fujiwara, Y. Mukuta, and T. Harada. Style-NeRF2NeRF: 3D Style Transfer From Style-Aligned Multi-View Images. In *ACM SIGGRAPH Asia*, pp. 1–10, 2024. 2

[9] R. Gal, O. Patashnik, H. Maron, A. H. Bermano, G. Chechik, and D. Cohen-Or. StyleGAN-NADA: CLIP-Guided Domain Adaptation of Image Generators. *ACM Trans. on Graphics*, 41(4):1–13, 2022. 5

[10] L. Gao, J. Yang, B. Zhang, J. Sun, Y. Yuan, H. Fu, and Y.-K. Lai. Real-time Large-scale Deformation of Gaussian Splatting. *ACM Trans. on Graphics*, 2024. 2

[11] M. Hassan, F. Forest, O. Fink, and M. Mielle. ThermoNeRF: A multimodal Neural Radiance Field for joint RGB-thermal novel view synthesis of building facades. *Advanced Engineering Informatics*, 65:103345, 2025. 8, 9

[12] J. Henstrom, R. De Amicis, C. A. Sanchez, and Y. Turkan. Immersive learning in engineering: A comparative study of VR and traditional building inspection methods. In *ACM Conf. on 3D web technology*, pp. 1–11, 2023. 3

[13] Y. Hou, M. Chen, A. Feng, and S. Li. Gaussian Splatting-Based Thermographic Modeling for Building Envelope Energy Audits. In *IEEE Int. Conference on Intelligent Reality (ICIR)*, pp. 1–2, 2024. doi: 10.1109/ICIR64558.2024.10976794 3

[14] K. Howil, J. Waczyńska, P. Borycki, T. Dziarmaga, M. Mazur, and P. Spurek. CLIPGaussian: Universal and Multimodal Style Transfer Based on Gaussian Splatting. *arXiv*, abs/2505.22854, 2025. 2, 5

[15] Y.-H. Huang, Y. He, Y.-J. Yuan, Y.-K. Lai, and L. Gao. StylizedNeRF: Consistent 3D Scene Stylization as Stylized Nerf via 2D-3D Mutual Learning. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[16] Ikea. Ikea Online Planar. https://www.ikea.com/at/de/planners/. Accessed: 2025-09-01. 1

[17] S. Jain, A. Kuthiala, P. S. Sethi, and P. Saxena. StyleSplat: 3D Object Style Transfer with Gaussian Splatting. *arXiv*, 2407.09473, 2024. 2

[18] D. V. Jauregui and K. R. White. Bridge Inspection using Virtual Reality and Photogrammetry. *Inspection and Monitoring Techniques for Bridges and Civil Structures*, pp. 216–246, 2005. 3

[19] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. on Graphics*, 42(4):139–1, 2023. 2, 4, 7, 8, 9

[20] S. Kheradmand, D. Rebain, G. Sharma, W. Sun, Y.-C. Tseng, H. Isack, A. Kar, A. Tagliasacchi, and K. M. Yi. 3D Gaussian Splatting as Markov Chain Monte Carlo. In *Advances in Neural Information Processing Systems (NIPS)*, 2024. 4

[21] C. Kleinbeck, H. Schieber, K. Engel, R. Gutjahr, and D. Roth. Multilayer Gaussian Splatting for immersive anatomy visualization. *IEEE Trans. on Vis. and Computer Graphics (TVCG)*, 2025. 5, 6, 7, 9

[22] Á. S. Kovács, P. Hermosilla, and R. G. Raidou. G-Style: Stylized Gaussian Splatting. In *Computer Graphics Forum*, vol. 43 (7), 2024. 2

[23] G. Kwon and J. C. Ye. CLIPstyler: Image Style Transfer With a Single Text Condition. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 5

[24] J. Lei, Y. Wang, G. Pavlakos, L. Liu, and K. Daniilidis. GART: Gaussian Articulated Template Models. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2

[25] Z. Li, Z. Chen, Z. Li, and Y. Xu. Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2

[26] Y. Lin, J. Lei, and K. Jia. Multi-StyleGS: Stylized Gaussian Splatting with Multiple Styles. *AAAI Conference on Artificial Intelligence*, 39(5):5289–5297, Apr. 2025. doi: 10.1609/aaai.v39i5.32562 2

[27] K. Liu, F. Zhan, Y. Chen, J. Zhang, Y. Yu, A. E. Saddik, S. Lu, and E. Xing. StyleRF: Zero-shot 3D Style Transfer of Neural Radiance Fields. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2

[28] K. Liu, F. Zhan, M. Xu, C. Theobalt, L. Shao, and S. Lu. StyleGaussian: Instant 3D Style Transfer with Gaussian Splatting. *arXiv*, 2403.07807, 2024. 2

[29] W. Liu, Z. Liu, X. Yang, M. Sha, and Y. Li. ABC-GS: Alignment-Based Controllable Style Transfer for 3D Gaussian Splatting. In *Int. Conf. on Multimedia and Expo*, 2025. 2

[30] X. Liu, X. Zhan, J. Tang, Y. Shan, G. Zeng, D. Lin, X. Liu, and Z. Liu. HumanGaussian: Text-Driven 3D Human Generation With Gaussian Splatting. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2

[31] Y. Liu, J. Tang, C. Zheng, S. Zhang, J. Hao, J. Zhu, and D. Huang. ClotheDreamer: Text-Guided Garment Generation with 3D Gaussians. *arXiv*, 2406.16815, 2024. 2

[32] R. Lu, H. Chen, Z. Zhu, Y. Qin, M. Lu, L. Zhang, C. Yan, and A. Xue. ThermalGaussian: Thermal 3d Gaussian splatting. In *Int. Conf. for Learning Representations*, 2025. 2, 3, 6, 7, 8, 9

[33] T. Lu, M. Yu, L. Xu, Y. Xiangli, L. Wang, D. Lin, and B. Dai. Scaffold-GS: Structured 3D Gaussians for View-Adaptive Rendering. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2, 3

[34] J. Luiten, G. Kopanas, B. Leibe, and D. Ramanan. Dynamic 3D Gaussians: Tracking by Persistent Dynamic View Synthesis. In *Int. Conf. on 3D Vision (3DV)*, 2024. 2

[35] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *European Conf. on Computer Vision (ECCV)*, 2020. 2, 3, 4

[36] M. Omer, L. Margetts, M. H. Mosleh, and L. S. Cunningham. Inspection of Concrete Bridge Structures: Case Study Comparing Conventional Techniques with a Virtual Reality Approach. *J. of Bridge Engineering*, 26(10), 2021. doi: 10.1061/(ASCE)BE.1943-5592.0001759 3

[37] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall. DreamFusion: Text-to-3D using 2D Diffusion. *arXiv*, 2209.14988, 2022. 2

[38] M. Raafat. BlenderNeRF, 2024. doi: 10.5281/zenodo.13250725 6

[39] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, et al. Learning Transferable Visual Models From Natural Language Supervision. In *Int. Conf. on Machine Learning*, 2021. 5

[40] L. Radl, M. Steiner, M. Parger, A. Weinrauch, B. Kerbl, and M. Steinberger. Stopthepop: Sorted Gaussian Splatting for View-Consistent Real-Time Rendering. *ACM Trans. on Graphics*, 43(4):1–17, 2024. 2

[41] A. Saroha, M. Gladkova, C. Curreli, D. Muhle, T. Yenamandra, and D. Cremers. Gaussian Splatting in Style. In *GCPR*, pp. 234–251, 2024. 2

[42] J. L. Schönberger and J.-M. Frahm. Structure-from-Motion Revisited. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[43] STRATUM 1 GmbH. Arrival.Space. https://arrival.space/welcome. Accessed: 2025-09-01. 1

[44] STRATUM 1 GmbH. Synthetic NeRF Dataset. https://www.kaggle.com/datasets/nguyenhung1903/nerf-synthetic-dataset. Accessed: 2025-09-01. 7

[45] X. Tu, L. Radl, M. Steiner, M. Steinberger, B. Kerbl, and F. de la Torre. VRSplat: Fast and Robust Gaussian Splatting for Virtual Reality. *Proc. ACM Comput. Graph. Interact. Tech.*, 8(1), May 2025. doi: 10.1145/3728311 2

[46] C. Wang, R. Jiang, M. Chai, M. He, D. Chen, and J. Liao. NeRF-Art: Text-Driven Neural Radiance Fields Stylization. *IEEE Trans. on Vis. and Computer Graphics (TVCG)*, 30(8):4983–4996, 2023. 2

[47] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *TIP*, 13(4):600–612, 2004. 4, 7

[48] T. Wen, Z. Liu, B. Lu, and Y. Fang. Scaffold-SLAM: Structured 3D Gaussians for Simultaneous Localization and Photorealistic Mapping. *arXiv*, abs/2501.05242, 2025. 2

[49] Wikipedia. Shapeshifting. https://en.wikipedia.org/wiki/Shapeshifting. Accessed: 2025-12-22. 2

[50] G. Wu, T. Yi, J. Fang, L. Xie, X. Zhang, W. Wei, W. Liu, Q. Tian, and X. Wang. 4D Gaussian Splatting for Real-Time Dynamic Scene Rendering. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2

[51] J. Wynn, Z. Qureshi, J. Powierza, J. Watson, and M. Sayed. Morpheus: Text-Driven 3D Gaussian Splat Shape and Color Stylization. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2025. 2

[52] K. Yang, Y. Liu, Z. Cui, Y. Liu, M. Zhang, S. Yan, and Q. Wang. NTR-Gaussian: Nighttime Dynamic Thermal Reconstruction with 4D Gaussian Splatting Based on Thermodynamics. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2025. 2

[53] Z. Yang, Z. Qiu, C. Xu, and D. Fu. MM-NeRF: Multimodal-Guided 3D Multi-Style Transfer of Neural Radiance Field. *IEEE Trans. on Vis. and Computer Graphics (TVCG)*, 31(9):5842–5853, 2024. 2

[54] V. Ye, R. Li, J. Kerr, M. Turkulainen, B. Yi, Z. Pan, O. Seiskari, J. Ye, J. Hu, M. Tancik, et al. gsplat: An open-source Library for Gaussian

splatting. *J. of Machine Learning Research*, 26(34):1–17, 2025. 5

[55] D. Zhang, J. Zhang, H. Xiong, Z. Cui, and D. Lu. Taking advantage of collective intelligence and BIM-based virtual reality in fire safety inspection for commercial and public buildings. *Applied Sciences*, 9(23):5068, 2019. 2

[56] K. Zhang, N. Kolkin, S. Bi, F. Luan, Z. Xu, E. Shechtman, and N. Snavely. Arf: Artistic Radiance Fields. In *European Conf. on Computer Vision (ECCV)*, 2022. 2

[57] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 586–595, 2018. 7

[58] J. Zheng, S. Ramasinghe, X. Li, and S. Lucey. Trading Positional Complexity vs Deepness in Coordinate Networks. In *European Conf. on Computer Vision (ECCV)*, 2022. 4

[59] W. Zielonka, T. Bagautdinov, S. Saito, M. Zollhöfer, J. Thies, and J. Romero. Drivable 3D Gaussian Avatars. In *Int. Conf. on 3D Vision (3DV)*, 2025. 2

[60] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross. Ewa Volume Splatting. In *Proceedings Visualization*, pp. 29–538. IEEE, 2001. 5, 6