3SGS : SaengSaeng Stylized Gaussian Splatting

Hail Song (20245183)[1], Young Bin Kim (20253151)[1], Hyeonjin Kim (20253240)[1]
[1]KAIST
{hail96, kimyb, hyeonjin}@kaist.ac.kr

https://github.com/hailsong/CS479-Project-StyleGaussian

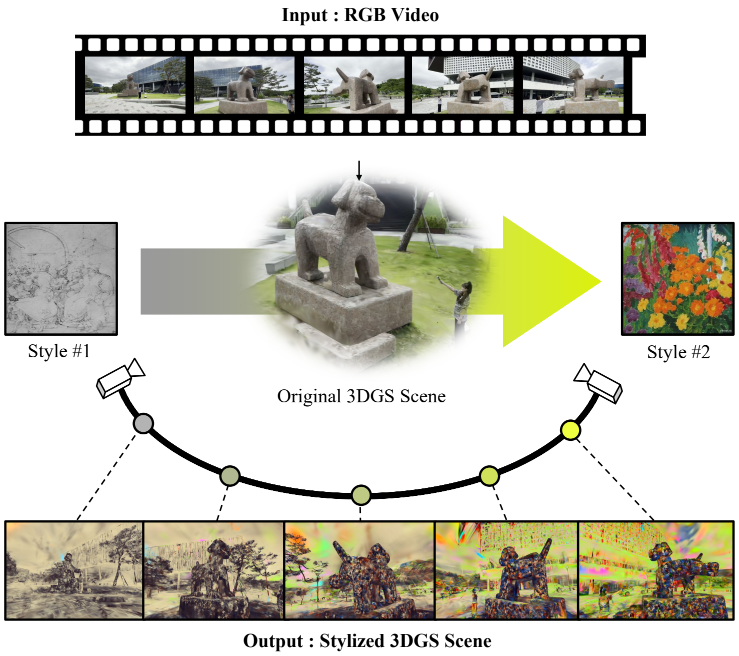

Figure 1. Teaser image of our project. The pipeline stylizes 3D scenes using camera-aware interpolation based on user-selected styles.

1. Introduction

As interest in customizing the visual appearance of 3D scenes continues to grow, so does the need for flexible and expressive style editing techniques. This is particularly important in domains such as digital art, animation, and immersive media, where visual stylization plays a key role in conveying emotional depth and narrative engagement. However, prior methods such as StyleRF [1] require lengthy test-time optimization and struggle to maintain consistency when rendering the scene from multiple viewpoints, limiting their practical applicability in interactive scenarios. 3D Gaussian Splatting [2] has recently emerged as a powerful representation for 3D scene reconstruction, offering efficient differentiable rendering and strong multi-view consistency. Its explicit structure makes it well-suited for integrating style transfer techniques.

In this project, we explore the creative potential of StyleGaussian [3] by applying position-aware style interpolation to a 3D scene. The framework extracts visual features from a style image, embeds them into 3D Gaussian points, and transforms them according to the desired style before decoding them into colors that define the scene's appearance. Instead of abrupt or scene-wide style changes, we implemented a smooth blending mechanism where styles transition gradually with camera movement, enabling immersive and emotionally resonant experiences across different viewpoints. The final output is a 10-second 3D rendering video centered on the theme of a painter reminiscing about "Saengsaeng," a companion from past walks on the KAIST campus. Produced for submission to a 3D Rendering Contest, this project investigates the expressive potential of neural rendering technologies as a creative tool that combines visual storytelling, spatial transformation, and stylistic experimentation.

2. Technical Aspects

In this work, we implement position-aware style interpolation based on the architecture of the StyleGaussian model. StyleGaussian is a real-time style transfer framework built upon 3D Gaussian Splatting (3DGS), a recently proposed representation for 3D scene reconstruction. It enables the direct mapping of arbitrary 2D style images onto 3D scenes with minimal latency. Unlike previous methods, it preserves stylistic consistency across diverse viewpoints, while supporting interactive feedback and exhibiting a high level of visual expressiveness.

The StyleGaussian pipeline consists of three stages: feature embedding, style transfer, and RGB decoding. This structure plays a central role in ensuring efficiency, real-time performance, and multi-view consistency throughout the stylization process.

2.1. Feature Embedding

$$e(\mathbb{G}) = \mathbb{G}^e = \{g_p^e = (\mu_p, \Sigma_p, \sigma_p, f_p)\}_p^P$$

Before applying style transfer, StyleGaussian embeds VGG features [4] extracted from 2D images into each Gaussian. Multi-view 2D feature maps are obtained from intermediate layers of apretrained VGG network and embeddded onto the radiance field such that feature corresponds to a point in the 3D GAussian scene. Once the Gaussian structure has been reconstructed, the VGG features can be fixed on top of it, allowing the features to remain constant during the subsequent style transfer process and enabling efficient computation. As a result, each Gaussian g_p is associated with a learnable feature parameter $f_p \in \mathbb{R}^D$, which is used to render the per-pixel feature value $F \in \mathbb{R}^D$.

$$F = \sum_{i \in N} f_i w_i, w_i = \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j).$$

Here, f_i denotes the feature of the i-th Gaussian, and w_i is a visibility-based weight determined by the opacity and ordering of Gaussians. The set of learnable features $\{f_p\}$ is optimized using an L1 loss, encouraging the rendered features F to match the VGG features of the input image.

$$F = T(F') = AF' + \sum_{i \in N} b w_i.$$

However, computing Gaussian Splatting requires significant memory consumption [5, 6] . To address this efficiently, we first render a low-dimensional feature $F' \in \mathbb{R}^{32}$, and then apply an affine transformation using a learnable linear matrix A and bias b to expand it into a high-dimensional feature F. Each Gaussian feature is transformed as $f_p = T'(f_p') = Af_p' + b$, and the final feature value is computed by aggregating the features of neighboring Gaussians through blending weights w_i. In other words, the learned low-dimensional feature f_p' of each Gaussian can be efficiently mapped to a high-dimensional representation f_p. This approach preserves rich feature information while reducing computational cost, and ultimately, each Gaussian retains a VGG feature vector f_p that can be used in the style transfer process.

2.2. Style Transfer via AdaIN

$$t(\mathbb{G}^e, I^s) = \mathbb{G}^t = \{g_p^t = (\mu_p, \Sigma_p, \sigma_p, f_p^t)\}_p^P$$

After obtaining the 3D Gaussians $\mathbb{G}^e$ embedded with VGG features, the features are transformed according to a given style image I^s. In the style transfer stage, we apply the Adaptive Instance Normalization (AdaIN) technique [7] , which rescales each Gaussian feature f_p into a stylized feature f_p^t by normalizing it with the channel-wise mean and standard deviation of the VGG features extracted from the style image I^s. AdaIN is a parameter-free, zero-shot style transfer method that allows real-time stylization without the need for optimization on the target style image. The transformation is defined by the following equation, which adjusts the original feature using the statistics of the style image:

$$f_p^t = \sigma(F^s) \cdot \frac{f_p - \mu(\{f_p\}_p^P)}{\sigma(\{f_p\})} + \mu(F^s)$$

Here, F^s denotes the VGG feature vector of the style image, and μ and σ represent the mean and standard deviation, respectively. For each Gaussian, the stylized feature f_p^t is computed by first normalizing the original feature f_p using its own mean μ and standard deviation σ, and then rescaling and shifting it with the statistics of the style image F^s. As a result, each Gaussian obtains a stylized feature f_p^t that reflects the statistical characteristics of the style image I^s, yielding the transformed 3D Gaussian set $\mathbb{G}^t$.

2.3. RGB Decoding via KNN-based 3D CNN

$$d(\mathbb{G}^t) = \mathbb{G}^s = \{g_p^s = (\mu_p, \Sigma_p, \sigma_p, c_p^s)\}_p^P$$

The stylized feature f_p^t is ultimately decoded into an RGB color c_p^s. Previous methods typically rendered a feature map using 2D convolutional neural networks (CNNs) and then converted it into a stylized RGB image [1, 8, 9], however, this approach often leads to multi-view inconsistency. To address this issue, StyleGaussian adopts a 3D

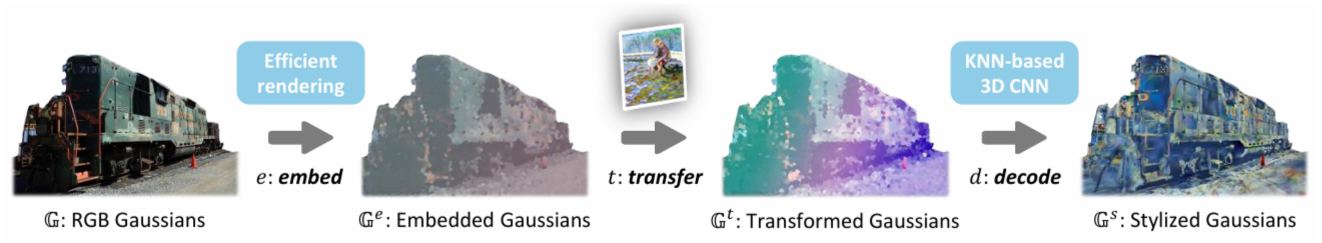
Efficient rendering
e: embed
t: transfer
KNN-based 3D CNN
d: decode
$\mathbb{G}$: RGB Gaussians
$\mathbb{G}^e$: Embedded Gaussians
$\mathbb{G}^t$: Transformed Gaussians
$\mathbb{G}^s$: Stylized Gaussians

Figure 2. Overview of the StyleGaussian pipeline [3]. Starting from reconstructed 3D Gaussians $\mathbb{G}$, the system first embeds VGG features (e), transforms them based on a given style image via AdaIN (t), and decodes the stylized features into RGB using a KNN-based 3D CNN (d). This enables real-time rendering with strong multi-view consistency.

CNN architecture that leverages the K-nearest neighbors (KNN) of each Gaussian. For each Gaussian, the features of neighboring points are aggregated using a sliding window–based convolution operation, and the final color of the central Gaussian is computed as follows:

$$f_p^{out} = \phi(\sum_{k=1}^{K} W^k f_k^{in} + B).$$

Here, f_k^{in} denotes the features of the k nearest neighbors, W^k are learnable weight matrices, and ϕ represents a sigmoid activation function. The final output f_p^{out} is then mapped to the RGB color c_p^s. This operation is implemented efficiently in parallel via matrix multiplication and can be executed rapidly on modern GPUs. Once the stylized Gaussian set $\mathbb{G}^s$ is obtained, the system enables real-time rendering from novel viewpoints without requiring repeated style transfer. The training process for style transfer utilizes the following loss function: $\mathcal{L} = \mathcal{L}_c + \lambda\mathcal{L}_s$, where $\mathcal{L}_c$ denotes the mean squared error (MSE) between content features, and $\mathcal{L}_s$ is the MSE that measures the deviation between the statistical characteristics (i.e., channel-wise mean and variance) of the rendered output and those of the style image. The total loss $\mathcal{L}$ balances content preservation and stylization through the weighting factor λ.

As a result, the initial color c_p is first replaced by a feature vector f_p during the embedding step, transformed into a stylized feature f_p^t via style transfer, and finally decoded into the output color c_p^s: $c_p \xrightarrow{e} f_p \xrightarrow{t} f_p^t \xrightarrow{d} c_p^s$. Throughout the style transfer process, the geometric attributes of each Gaussians—its position μ_p, covariance Σ_p, and opacity σ_p—remain fixed, and only the color component is modified. The overall style transfer function is thus defined as: $\{c_p \xrightarrow{f} c_p^s; I^s\}_p^P, f = e \circ t \circ d.$

3. Reproduction Steps

In order to reproduce StyleGaussian [3] and create a dramatic video that interpolates between two different styles based on camera movement, we followed the process below. To reflect the theme of a painter reminiscing about "Saengsaeng," we selected the 22nd and 43rd style images used in StyleGaussian to demonstrate a style transition from monochromatic to a colorful 3D scene. The following three steps were conducted to reconstruct the 3D scene, extract features and train for stylization, and render the final video.

3.1. Video recording and Training 3DGS

We captured a video of the statue "SaengSaeng" located on the main KAIST campus, along with a person standing in front of it, using a iPhone 15 Pro. The video was recorded in an orbital motion, with the camera circling around the subject while keeping both the person and the statue centered in the frame. Each frame was then processed using the structure-from-motion algorithm COLMAP [10] to estimate the camera matrices and obtain initial 3D points for training 3D Gaussian Splatting (3DGS) [2]. The code for training 3D Gaussian Splatting with the results from COLMAP can be found in the project repository under `train_reconstruction.py`.

3.2. Feature extraction and 3DGS stylization

After reconstructing the 3D scene using COLMAP and 3D Gaussian Splatting, we replaced the RGB color attributes of each Gaussian with learnable feature vectors. These features are extracted using a pretrained VGG encoder and optimized to match the corresponding VGG features of the input images.

This process is implemented in two stages. In the first stage, we used the script `train_feature.py` to embed VGG features into the Gaussian points. The script loads a reconstructed Gaussian model from a `.ply` file and performs feature-space rendering using randomly sampled camera views. An $\mathcal{L}_1$ loss is applied between the rendered features and VGG features of the corresponding views to supervise the training.

In the second stage, stylization is performed using the script `train_artistic.py`. This script loads the feature-trained model and applies style transfer via Adaptive Instance Normalization (AdaIN), using randomly sam-

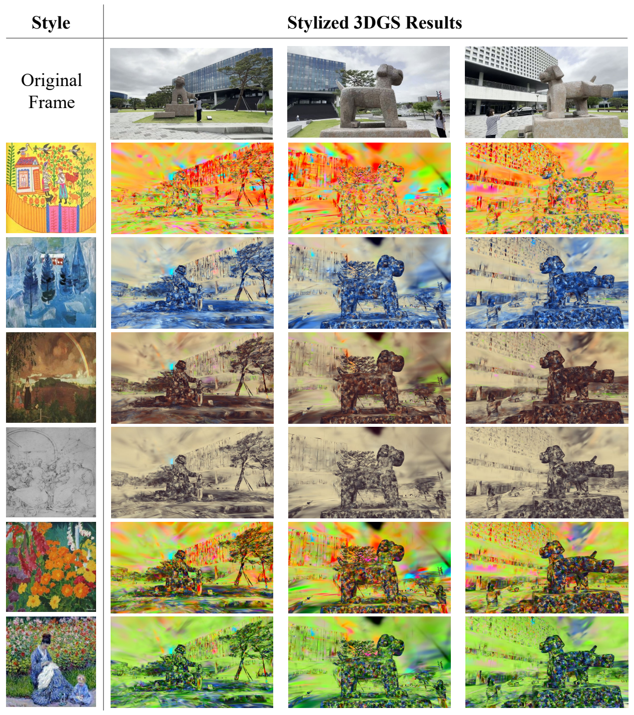

Figure 3. The same 3D Gaussian scene is stylized using different reference images from the WikiArt dataset (style IDs: 1, 6, 12, 22, 43, 49). The results illustrate the model's ability to produce diverse artistic appearances while preserving the geometric structure of the scene.

pled style images from the WikiArt dataset. A learnable decoder then maps the stylized features back to RGB. The training minimizes both content and style losses in VGG space, with optional content-preserving regularization.

Throughout this stage, the geometry of each Gaussian, the position, scale, and opacity, remains fixed. Only the appearance attributes are updated to reflect the target style.

3.3. Rendering Phase

To produce a smooth and stylistically interpolated video, we rendered a continuous camera trajectory alongside a gradual transition between two distinct style embeddings. This was accomplished using the script `render_main.py`, which performs joint interpolation in both camera space and feature space.

Starting from the discrete set of training cameras $\{C_0, C_1, \ldots, C_N\}$ estimated via COLMAP, we inserted k virtual cameras between each pair of adjacent views to simulate continuous camera motion. For a virtual camera between camera i and $i+1$, the rotation $\mathbf{R}$ and translation $\mathbf{T}$ are interpolated linearly as:

$$\mathbf{R}_{\text{interp}} = (1-\alpha)\mathbf{R}_i + \alpha\mathbf{R}_{i+1} \tag{1}$$

$$\mathbf{T}_{\text{interp}} = (1-\alpha)\mathbf{T}_i + \alpha\mathbf{T}_{i+1} \tag{2}$$

$$\alpha = \frac{j}{k}, \quad j \in [0, k) \tag{3}$$

In parallel, we interpolated between two stylized versions of the scene. Let $\mathbf{f}_{\text{start}}$ and $\mathbf{f}_{\text{end}}$ be the stylized feature embeddings obtained from applying AdaIN with the source and target style images, respectively. For frame index t out of T total frames, the interpolated feature $\mathbf{f}_t$ is computed as:

$$\mathbf{f}_t = (1-\beta_t)\mathbf{f}_{\text{start}} + \beta_t\mathbf{f}_{\text{end}}, \quad \beta_t = \frac{t}{T}$$

These features are decoded into RGB colors using a learned 3D CNN decoder and rendered with the interpolated virtual camera using the 3D Gaussian Splatting pipeline. The resulting images were stored as individual frames, forming a smooth visual transition in both viewpoint and style space.

This rendering approach effectively captures the evolving aesthetics of the scene as the virtual camera orbits around the subject, creating a dynamic impression of style morphing in 3D.

3.4. Interactive Viewer

We implemented an interactive viewer to visualize style interpolation in real-time. A slider was added to interpolate between two styles (Style ID 22 $\rightarrow$ 43), enabling smooth blending of style embeddings. Additionally, we added an Auto Style Slide checkbox to animate the transition automatically over time, mimicking camera-based style changes. These features help users intuitively explore the effect of different styles on the same 3D scene. The implementation of viewer can be found at `viewer.py`.

4. Conclusion

This project demonstrates that 3D Gaussian Splatting, based on neural rendering, can serve not only as a means of realistic reconstruction but also as a tool for artistic expression. In particular, the efficient embedding–transfer–decoding structure of StyleGaussian enables smooth interpolation between different styles, providing an immersive visual experience. This allows for fluid transitions between different artistic expressions, which can be modulated not by discrete events, but by spatial cues such as camera movement and viewpoint. In our implementation, the applied styles gradually change with the camera angle, allowing for a form of visual storytelling that evokes the flow of memory and emotion. This illustrates how style transfer in 3D can evolve beyond purely technical applications to support emotionally rich, narrative-driven content. Ultimately, our project showcases the potential of 3D style transfer technologies to harmonize artistic intent with real-time performance, offering concrete insights into their viability as future creative tools.

References

[1] Kunhao Liu, Fangneng Zhan, Yiwen Chen, Jiahui Zhang, Yingchen Yu, Abdulmotaleb El Saddik, Shijian Lu, and Eric P Xing. Stylerf: Zero-shot 3d style transfer of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8338–8348, 2023. 1, 2

[2] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. 2023. 1, 3

[3] Kunhao Liu, Fangneng Zhan, Muyu Xu, Christian Theobalt, Ling Shao, and Shijian Lu. Stylegaussian: Instant 3d style transfer with gaussian splatting. In *SIGGRAPH Asia 2024 Technical Communications*, pages 1–4. 2024. 1, 3

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2

[5] Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. Langsplat: 3d language gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20051–20060, 2024. 2

[6] Shijie Zhou, Haoran Chang, Sicheng Jiang, Zhiwen Fan, Zehao Zhu, Dejia Xu, Pradyumna Chari, Suya You, Zhangyang Wang, and Achuta Kadambi. Feature 3dgs: Supercharging 3d gaussian splatting to enable distilled feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21676–21685, 2024. 2

[7] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510, 2017. 2

[8] Hsin-Ping Huang, Hung-Yu Tseng, Saurabh Saini, Maneesh Singh, and Ming-Hsuan Yang. Learning to stylize novel views. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13869–13878, 2021. 2

[9] Fangzhou Mu, Jian Wang, Yicheng Wu, and Yin Li. 3d photo stylization: Learning to generate stylized novel views from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16273–16282, 2022. 2

[10] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016. 3