Gaussian Splatting Decoder for 3D-aware Generative Adversarial Networks

Florian Barthel[1, 2] Arian Beckmann[1] Wieland Morgenstern[1] Anna Hilsmann[1] Peter Eisert[1,2]

[1] Fraunhofer Heinrich Hertz Institute, HHI
[2] Humboldt University of Berlin

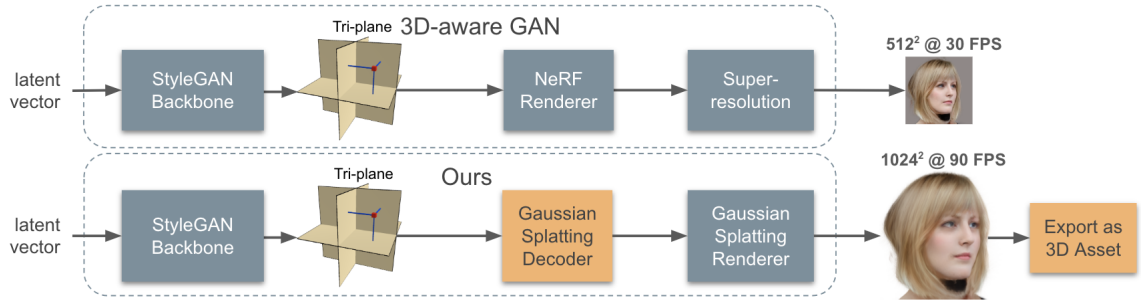

Figure 1. We propose a novel 3D Gaussian Splatting decoder that converts high quality results from pre-trained 3D-aware GANs into Gaussian Splatting scenes in real-time for efficient and high resolution rendering.

Abstract

NeRF-based 3D-aware Generative Adversarial Networks (GANs) like EG3D or GIRAFFE have shown very high rendering quality under large representational variety. However, rendering with Neural Radiance Fields poses challenges for 3D applications: First, the significant computational demands of NeRF rendering preclude its use on low-power devices, such as mobiles and VR/AR headsets. Second, implicit representations based on neural networks are difficult to incorporate into explicit 3D scenes, such as VR environments or video games. 3D Gaussian Splatting (3DGS) overcomes these limitations by providing an explicit 3D representation that can be rendered efficiently at high frame rates. In this work, we present a novel approach that combines the high rendering quality of NeRF-based 3D-aware GANs with the flexibility and computational advantages of 3DGS. By training a decoder that maps implicit NeRF representations to explicit 3D Gaussian Splatting attributes, we can integrate the representational diversity and quality of 3D GANs into the ecosystem of 3D Gaussian Splatting for the first time. Additionally, our approach allows for a high resolution GAN inversion and real-time GAN editing with 3D Gaussian Splatting scenes.
Project page: florian-barthel.github.io/gaussian_decoder

1. Introduction

Creating and editing realistic 3D assets is of vital importance for applications such as Virtual Reality (VR) or video games. Often, this process is very costly and requires a significant amount of manual editing. Over the last few years, there have been drastic improvements to 2D [14–16, 37] and 3D [2, 6–8, 33, 39, 42] image synthesis. These advancements increasingly narrow the gap between professionally created 3D assets and those that are automatically synthesized. One of the most notable recent methods is the *Efficient Geometry-aware 3D GAN (EG3D)* [8]. It successfully combines the strength of *StyleGAN* [16], originally built for 2D image generation, with a 3D NeRF renderer [3, 26], achieving state-of-the-art 3D renderings synthesized from a latent space. Despite EG3D's significant contributions to 3D rendering quality, its integration into 3D modeling environments like Unity or Blender remains difficult. This challenge stems from its NeRF dependency, which only generates 2D images from 3D scenes, without ever explicitly representing the 3D scene. As a result, EG3D cannot be imported or manipulated in these computer graphics tools.

Recently introduced, *3D Gaussian Splatting (3DGS)* [19] provides a novel explicit 3D scene representation, enabling high-quality renderings at high frame rates. Following its debut, numerous derivative techniques have already emerged, including the synthesis of controllable hu-

man heads [34, 43, 48], the rendering of full body humans [20] or the compression of the storage size of Gaussian objects [30]. On the one hand, 3DGS provides a substantial improvement in terms of rendering speed and flexibility compared to NeRF: The explicit modelling enables simple exporting of the scenes into 3D software environments. Furthermore, the novel and efficient renderer in 3DGS allows for high-resolutions renderings, and an increase in rendering speed with a factor of up to $1000\times$ over state-of-the-art NeRF frameworks [4, 18, 31, 36]. On the other hand, NeRF's implicit scene representation allows for straightforward decoding of scene information from latent spaces. Notably through the usage of tri-planes [8], which store visual and geometric information of the scene to be rendered. This enables the integration of NeRF rendering into GAN frameworks, lifting the representational variety and visual fidelity of GANs up into three-dimensional space. Combining NeRFs and GANs is highly advantageous, as rendering from a latent space offers multiple benefits: Firstly, it allows for rendering an unlimited amount of unique appearances. Secondly, a large variety of editing methods [1, 12, 29] can be applied. And thirdly, single 2D images can be inverted, using 3D GAN inversion [5, 15, 35], allowing for full 3D reconstructions from a single image.

Sampling visual information from latent spaces with large representational variety poses a challenge for rendering with 3DGS, as the framework requires the information for the appearance of the scene to be encoded as attributes of individual splats, rather than in the latent space itself. This severely complicates the task of fitting 3D Gaussian splats to variable latent spaces, given that the splats would need to be repositioned for every new latent code - a challenge that is not addressed in the original 3DGS framework. Several approaches tackling the problem of rendering with 3DGS from latent tri-planes have been proposed [20, 45, 49], but to the best of our knowledge, no method exists to create 3D heads rendered with Gaussian Splatting from a latent space.

In this work, we propose a framework for the synthesis of explicit 3D scenes representing human heads from a latent space. This is done by combining the representational variety and fidelity of 3D-aware GANs with the explicit scenes and fast rendering speed of 3D Gaussian Splatting. Our main contributions can be summarized as follows:

1. A novel method that allows for GAN-based synthesis of explicit 3D Gaussian Splatting scenes, additionally avoiding superresolution modules as used in the generation of implicit scene representations.
2. A novel sequential decoder architecture, a strategy for sampling Gaussian splat positions around human heads and a generator backbone fine-tuning technique to improve the decoders capacity.
3. An open source end-to-end pipeline for synthesizing state-of-the-art 3D assets to be used in 3D software.

2. Related Work

2.1. Neural Radiance Fields

In their foundational work on Neural Radiance Fields (NeRFs) Mildenhall *et al*. [27] propose to implicitly represent a 3D scene with an MLP that outputs color and density at any point in space from any viewing direction. This representation revolutionized novel-view synthesis, due to its ability to reconstruct scenes with high fidelity, high flexibility with respect to viewpoints, and compactness in representation through the usage of the MLP. To obtain a 2D rendering, a ray is cast for each pixel from the camera into the scene with multiple points sampled along each ray, which in turn are fed to the MLP in order to obtain their respective color and density values. NeRFs have proven to provide high-quality renderings, but are slow during both training and inference: the sampling and decoding process require querying a substantial number of points per ray, which has a high computational cost. Successors of the seminal NeRF approach are subject to improvements in quality [25] as well as training and inference speed [3, 31, 36]. Plenoxels [36] replaces the MLP representation of the scene by a sparse voxel grid representation, which leads to a speed-up in optimization time by two orders of magnitude compared to vanilla NeRF while maintaining high rendering quality. InstantNGP [31] proposes the usage of multi-resolution hash tables to store scene-related information. Through leveraging parallelism on GPUs and a very optimized implementation that fits the hash tables into the GPU cache, it achieves significant speedups in processing times, making real-time applications feasible.

Moreover, various approaches aiming to render in real-time propose to either store the view-dependent colors and opacities of NeRF in volumetric data representations or partition the scene into multiple voxels represented by small independent neural networks [9, 11, 13, 21, 23, 40, 44, 46].

2.2. 3D Gaussian Splatting

Recently, Kerbl *et al*. [18] proposed to represent scenes explicitly in the form of Gaussian splats. Each singular splat represents a three-dimensional Gaussian distribution with mean μ and covariance matrix Σ. For computational simplicity, the authors decide to represent the covariance matrix as the configuration of an ellipsoid, i.e. $\Sigma = RSS^TR^T$, with scaling and rotation matrices S and R. To characterize the appearance, each splat holds further attributes describing its opacity and view-dependent color through a set of spherical harmonics. Each splat's attributes are optimized in end-to-end training, utilizing a novel differentiable renderer. This renderer is essential for the success of 3D Gaussian Splatting. Its architectural design allows for high-resolution rendering in real-time through fast GPU execution utilizing anisotropic splatting while being visibility-aware. This ar-

chitectural design significantly accelerates the training process and novel-view rendering time. In general, 3DGS [18] is able to outperform several NeRF-based state-of-the-art approaches in rendering speed by factors of up to $1000\times$, while keeping competitive or better image quality.

Several approaches that utilize 3DGS for the representation and rendering of human heads have been proposed [34, 43, 48]. GaussianAvatar [34] allows editing of facial expressions within a scene of Gaussians already fitted to a specific identity. To do so, they use the FLAME [22] 3D morphable face model to create a triangle-mesh representing the head in 3D space and assign a splat to each triangle. Moreover, the geometric attributes of the splats are dynamically adjusted to align with the respective triangle's properties; for example, the global rotation of the splat is adjusted to match that of the triangle. Similarly, Xu *et al*. [43] anchor the Gaussian splats to a 3D triangle mesh fitted to a head that depicts a neutral expression. They utilize deformation MLPs conditioned on expression vectors to adjust the triangle-mesh and the resulting Gaussian positions to account for changes in expression. Ultimately, they render a feature map from their scene with 3DGS and translate those into high-fidelity head images in 2K resolution with a super-resolution network.

2.3. 3D-aware GANs

Following the success of 2D GANs in recent years, several methods have been proposed to synthesize 3D content with GANs as well. To achieve this, the generator component of a GAN is modified to create an internal 3D representation suitable for output through differentiable renderers. Given that these renderers return a 2D representation of the 3D model, the framework can be trained with 2D data. This is crucial, as high-quality 2D datasets are more readily available compared to their 3D counterparts. One of the first high-resolution 3D-aware GANs is *LiftingStyleGAN* [38]. Its architecture extends the 2D *StyleGAN* [17] with a custom built renderer based on texture and depth maps. Shortly after, π*-GAN* [7] and *GIRAFFE* [33] have been introduced that both use a Neural Radiance Field (NeRF) renderer. Those methods show very promising visual results. Nevertheless, while π-GAN is very slow in rendering, only achieving 1 frame per second, GIRAFFE fails to estimate a good 3D geometry. Both challenges are solved with the introduction of the *Efficient Geometry-aware 3D Generative Adversarial Network (EG3D)* by Chan *et al*. [8]. EG3D combines the strength of the StyleGAN architecture for 2D image synthesis with the rendering quality of a NeRF renderer. This is done by reshaping the output features of a StyleGAN generator into a three-dimensional triplane structure to span a 3D space. From this tri-plane, 3D points are projected onto 2D feature maps and forwarded to a NeRF renderer. The renderer creates a 2D image at a small resolution, which is then forwarded to a super-resolution module. This approach returns state-of-the-art renderings at a resolution of 512x512 pixels, while maintaining reasonable rendering speeds of about 30 FPS. The use of the super-resolution module effectively locks in the output size and aspect ratio, making it impossible to adjust or enlarge them without training a completely new network. This limitation of EG3D contrasts with explicit methods, which feature a renderer capable of adjusting the rendering resolution on demand.

Since the inception of EG3D, several approaches have adopted its architecture, extending the rendering capabilities [2, 28, 32, 42]. *PanoHead* [2], stands out in particular, as it addresses synthesizing full 360° heads. This is done by adding further training data that shows heads from the back and by disentangling the head from the background. The latter is done by separately generating the background and blending it with the foreground using a foreground mask obtained during the rendering process.

2.4. Decoding Gaussian Attributes from Tri-planes

Decoding NeRF attributes, i.e. color and density, from a triplane has proven to produce state-of-the-art frameworks. Decoding tri-planes into Gaussian Splatting attributes, on the other hand, induces further complexity. This is because a Gaussian splat, located at a specific position on the triplane need not represent the color and density of this specific location, but instead a 3D shape with a scale that extends into other regions of the scene. Naively, this could be solved by treating Gaussian splats as a point cloud with a very high number of tiny colored points. This approach would however neglect the advantages of Gaussian Splatting and reintroduce high computational costs for rendering the point cloud. Instead, when decoding Gaussian attributes [18], we seek to find suitable representations, such that Gaussian splats adapt their geometry to represent the structure of the target shape. Thus, smooth surfaces should be represented by wide flat Gaussian splats, while fine structures are best represented by thin long Gaussians.

Recent work already investigated the ability to decode Gaussian splats from tri-planes. *HUGS* [20] uses a small fully connected network to predict the Gaussian attributes to render full body humans in 3D. Contrary to our approach, HUGS overfits a single identity iteratively instead of converting any IDs from a latent space in a single shot. Similarly, [49] uses a transformer network to decode Gaussian attributes from a tri-plane in order to synthesize 3D objects. A different approach that also combines 3DGS with triplanes is *Gaussian3Diff* [45]. Instead of decoding Gaussian attributes from a tri-plane, they equip each Gaussian with a local tri-plane that is attached to its position. This hybrid approach shows promising quality, although the rendering speed is lower compared to 3DGS.

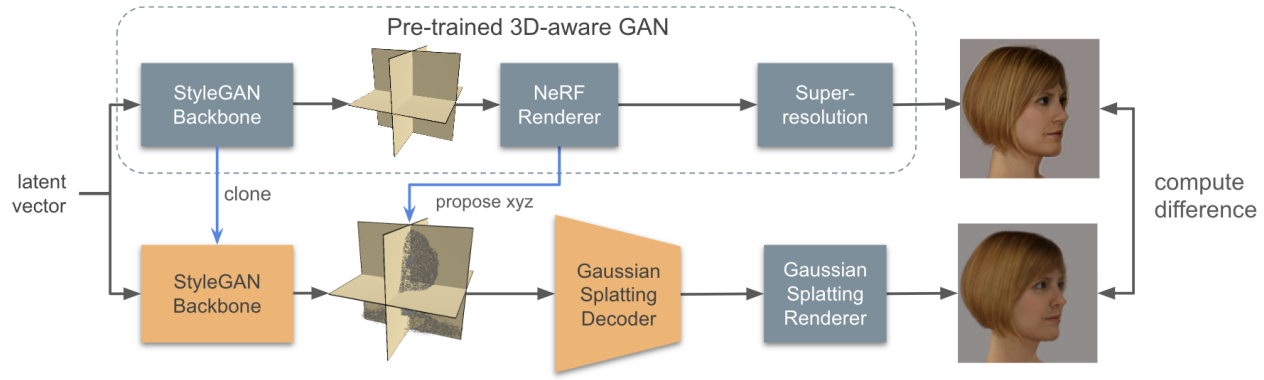

Figure 2. Visualization of our method (orange parts are optimized). We initially clone the backbone of the 3D-aware GAN. Afterwards, we iteratively optimize the Gaussian Splatting decoder by comparing the output of the pre-trained GAN, after super-resolution, with the output of the decoder. The xyz coordinates at which the tri-plane is sampled originates from the density information of the NeRF renderer.

3. Method

Our goal is to design a decoder network that converts the output of a 3D-aware GAN, specifically tailored for human head generation, into a 3D Gaussian Splatting scene without requiring an iterative scene optimization process. An overview of our method is shown in Figure 2. We extract the tri-plane of the 3D GAN, which is originally used to render a NeRF scene, and train a decoder network to obtain 3D Gaussian Splatting attributes (i.e. position, color, rotation, scale, and opacity). For simplicity, we omit the estimation of view-dependent spherical harmonics coefficients. For training, we compare the synthesized images from the 3D GAN to the renderings of the decoded 3D Gaussian Splatting scenes. Importantly, our decoder does not use any super-resolution module. Instead, we render the decoded Gaussian Splatting scene already at the same resolution as the final output of the 3D GAN. The absence of a super-resolution module allows the export of the decoded scene directly into 3D modeling environments, and for rendering at different resolutions and aspect ratios at high frame rates.

3.1. Position Initialization

Given that the 3D Gaussian splats, being described with multiple attributes (position, color, rotation, scale, and opacity), have multiple degrees of freedom, it is difficult to receive a meaningful gradient for the position during optimization. To overcome this issue, 3DGS uses several strategies to prune, clone, and split Gaussians during the training in order to spawn new Gaussians at fitting locations in the scene or remove redundant ones. For example, if a Gaussian splat is located at an incorrect position, 3DGS prefers to make the Gaussian splat vanish by reducing the opacity or to change its color to fit the current position, rather than moving its position. For our purpose of training a decoder that automatically creates new Gaussian scenes in a single forward pass, this iterative approach is not available. Instead, we take advantage of the geometric information already contained in the pre-trained 3D GAN's tri-plane. This is done by decoding the tri-plane features into opacity values using the pre-trained MLP of the NeRF renderer followed by a surface estimation based on the opacity values. Specifically, we uniformly sample a cube of points ($128 \times 128 \times 128$), decode the opacity and estimate the surface with marching cubes [24]. On this surface, we sample 500k points at random positions and slightly interpolate the points randomly towards the center, thus creating a thick surface. This provides us with a good position initialization for the Gaussians representing any head created by the 3D GAN. Even so, sampling the opacity from the NeRF renderer is computationally expensive. Nevertheless, this only has to be done once per ID / latent vector. After the 3D Gaussian scene is created, it can be rendered very efficiently.

3.2. Decoder Architecture

Recent work that use a decoder network to estimate Gaussian Splatting attributes from tri-plane features use fully connected networks [20] or transformer-based models [49]. For our approach, we also use a fully connected network, however, instead of computing all Gaussian attributes at once, we calculate them sequentially. Specifically, we first forward the tri-plane features to the first decoder that estimates the color. After that, we use the information of the color together with the tri-plane features and feed them to the next decoder that estimates the opacity. This is done iteratively until all attributes are decoded (color $\rightarrow$ opacity $\rightarrow$ rotation $\rightarrow$ scale $\rightarrow$ position offset). Thus, the last decoder receives all preceding attributes along with the tri-plane features. The intuition behind this approach is to create a dependency between the attributes. We hypothesize that, for instance, the scale decoder benefits from information about

the color or rotation, in order to decide how large the respective Gaussian splat will be. Additionally, the high degrees of freedom of the combined Gaussian splat attributes get reduced heavily for each decoder, allowing for easier specialization.

Inside each decoder, we use three hidden layers, each equipped with 128 neurons and a *GELU* activation. The output layer has no activation function, except for the scaling decoder. There, we apply an inverted *Softplus* activation to keep the splats from getting too large, avoiding excessive GPU memory usage during rasterization.

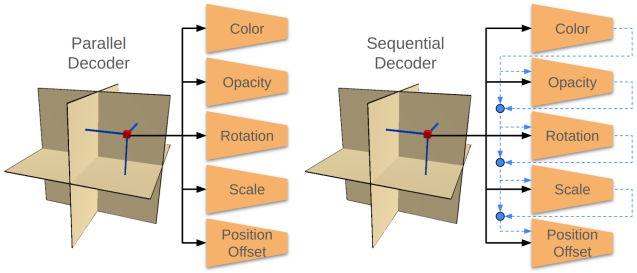

Figure 3. A comparison between a parallel decoder that maps all Gaussian attributes at once to our sequential decoder, where each attribute is decoded after another using the prior information.

3.3. Backbone Fine-tuning

In addition to optimizing the weights of the decoder network, we create a copy of the pre-trained 3D generator and optimize its weights as well. This fine-tuning allows the optimization process to adapt the tri-plane features to provide a better basis for creating Gaussian Splatting attributes, as they are inherently different. While NeRFs only require the color and density of a specific location, Gaussian splats additionally have a scale and rotation, thus influencing adjacent regions too.

3.4. Loss Functions

The vanilla 3D Gaussian Splatting algorithm uses a combination of L1 loss and structural similarity. This combination has proven to be very successful for learning static scenes. For our purpose, however, of learning a decoder network that is able to synthesize an tremendous diversity of images, it requires a loss function that provides better perceptual feedback. This is because we aim to produce a 3D Gaussian Splatting face that looks perceptually very close to the GAN rendered face, without penalizing the model too much if small structures like hair do not align perfectly. For that reason, we supplement the existing L1 and structural similarity loss with an LPIPS norm [47] and an ID similarity [10] loss. This ID loss is based on a pre-trained face detector (ArcFace) and estimates how similar two faces are. Since PanoHead renders the head from all 360° views, we only apply the ID loss, when the face is viewed from a frontal viewpoint. Additionally, to guide the decoder towards areas needing finer structural details, we calculate the difference between the synthesized image and target image after applying a Sobel filter. Formally, our loss function can be expressed as follows:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{\text{L1}} + \lambda_2 \mathcal{L}_{\text{SSIM}} + \lambda_3 \mathcal{L}_{\text{LPIPS}} + \lambda_4 \mathcal{L}_{\text{ID}} + \lambda_5 \mathcal{L}_{\text{Sobel}}. \quad (1)$$

4. Experiments

4.1. Implementation Details

In the following experiments, we train our Gaussian splatting decoder for multiple pre-trained target GANs. These are: EG3D trained on the FFHQ [15] dataset, EG3D trained on the LPFF dataset [41], and PanoHead trained on the FFHQ-H [2] dataset. We train for 500k iterations with an Adam optimizer using a learning rate of 0.0009. Loss weights are set to $(\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5) = (0.2, 0.5, 1.0, 1.0, 0.2)$ for all experiments unless stated otherwise. For PanoHead, we sample random cameras all around the head, and for EG3D, we sample mainly frontal views with small vertical and horizontal rotations.

We have optimized all training parameters for PanoHead, since it synthesizes full 360° views, making it ideally suited for being rendered in an explicit 3D space.

4.2. Metrics

To evaluate the performance of our decoder, we measure the image similarity for 10k images using MSE, LPIPS, ID similarity and structural similarity. In order to measure the frame rate, we use a custom visualization tool that is based on the EG3D visualizer. This way, we ensure that the performance differences are purely due to the renderer and not dependent on the programming language or compiler. With very efficient Gaussian splatting renderer like the SIBR viewer [19] that is purely written in C++, we could achieve even higher FPS.

4.3. Quantitative Results

After training our decoders, we observe a high image similarity between the decoded Gaussian Splatting scene and the respective target GAN as stated in Table 1. The low MSE and SSIM indicate that the renderings have similar colors and structures, respectively. In addition, the LPIPS and ID similarity metrics underline that the images are perceptually very close. The highest ID similarity is found when decoding the $\text{EG3D}_{\text{FFHQ}}$ model. Here, we reach a similarity score of 0.968. A possible explanation for this is that the FFHQ training dataset contains the fewest images across all three comparisons, making it the easiest to decode, given that there is less variation. The lowest ID similarity is found for

Model	Training Data	MSE ↓	LPIPS ↓	SSIM ↑	ID Sim. ↑	FPS@512 ↑	FPS@1024 ↑
PanoHead	FFHQ-H					37	N/A
Ours	PanoHead	0.002	0.161	0.820	0.902	170	90
$EG3D_{LPFF}$	LPFF					32	N/A
Ours	$EG3D_{LPFF}$	0.004	0.248	0.852	0.946	135	96
$EG3D_{FFHQ}$	FFHQ					31	N/A
Ours	$EG3D_{FFHQ}$	0.002	0.195	0.842	0.968	164	132

Table 1. Results for training our decoder for different pre-trained 3D-aware GANs. Columns with *Ours* refer to a decoder that was trained with the GAN specified in the column above. For all three decoders we observe high similarity scores along with high rendering speeds.

the decoder trained with the PanoHead GAN. In this case, the decoder has to learn a full 360° view of the head. This is not regarded by the ID similarity as it is only computed for renderings showing a frontal view.

Considering render speed for each model, rendering the Gaussian Splatting scene achieves about four times the FPS compared to rendering the 3D-aware GANs. Furthermore, as the rendering resolution for Gaussian Splatting is not limited by any super-resolution module, it can be rendered at arbitrary resolution. Here, we observe that when increasing resolution four-fold, we still achieve more than three times the framerate of the GAN models at the lower resolution.

4.4. Qualitative Results

In addition to the quantitative measures, we also demonstrate our method qualitatively. Figure 4 shows one example rendering for each of the three 3D-GAN methods, along with our decoded Gaussian Splatting scenes. We observe a high visual similarity between target and rendering as indicated by the image similarity metrics. While EG3D uses one single 3D scene that combines head and background, PanoHead uses two separate renderings. This allows our decoder to exclusively learn the head and rotate it independently to the background. An example of a full 360° rotation of two decoded Gaussian Splatting heads is shown in Figure 6.

Figure 4. Example renderings of the target images produced by respective 3D-aware GAN (top row) and the renderings of the decoded 3D Gaussian Splatting scene (bottom row, *Ours*). Additional renderings can be found in the supplementary material.

Figure 5. Renderings for example interpolating paths, demonstrating the possibility for applying GAN editing methods.

Additionally, we observe a reduction in aliasing and texture sticking artifacts with our 3D representation when rotating the camera around the head. This was often observed when rendering with EG3D or PanoHead. Specifically, some structures like hair or skin pores shifted when chaining the camera viewpoint, instead of moving along with the 3D head. This is no longer the case with our Gaussian Splatting representation, as we produce one fixed explicit 3D scene for each ID.

We also demonstrate in Figure 5 that our decoder allows latent interpolation. This opens up various GAN editing or GAN inversion methods to be applicable to our method.

In some renderings, we observe that the eyes appear uncanny or blurry. We believe this occurs because the underlying target data produced from the 3D-aware GAN almost exclusively shows renderings where the person is looking towards the camera. The GAN's NeRF-renderer, being view-dependent, likely learns to place the pupils to align with the camera angle. However, as we disabled the spherical harmonics to reduce complexity, our decoder is not able to learn any view dependencies. Instead, it learns an averaged eye, which is slightly blurry and always looks forward.

Figure 6. Example 360° rendering of our Gaussian Splatting decoder trained with PanoHead.

To overcome this limitation, it might be beneficial to incorporate the spherical harmonics into the decoder training for future work.

4.5. Ablation Study

In the following, we will justify our design decisions by performing an ablation study to the main components.

Position Initialization: The position initialization is a crucial component of our decoder as it decides where to place the Gaussian splats. For this, multiple approaches are possible: Sampling the points on a 3D grid, sampling the points randomly in a 3D cube or sampling the points on the surface of a 3D shape, created by marching cubes. Interestingly, when looking at the quantitative results in for all three approaches in Table 2, we clearly favor sampling on a 3D grid, as it achieves the overall best scores. Nevertheless, when inspecting the resulting renderings in Figure 7, we observe that grid sampling creates some artifacts. We see some horizontal and vertical lines on the surface of the head, where the splats are placed. Although this is not penalized by the chosen metrics, it significantly decreases the level of realism. Therefore, given that the marching cube sampling scores second best, while producing good visual results, we chose it for our decoder.

Sampling Method	LPIPS ↓	SSIM ↑	ID Sim ↑
Random Pos	0.179	0.839	0.856
3D Grid	**0.167**	**0.851**	**0.898**
March. Cubes	0.176	0.842	0.883

Table 2. Comparing different position sampling methods.

Decoder Architecture: The core component of our method is the decoder. Its architecture can have a big influence on the capacity to learn a mapping between tri-plane features and Gaussian Splatting attributes. In the following, we will look into three different architecture types. First, the sequential decoder, decoding each attribute with the information of the previous one (color → opacity → rotation → scale → position offset), a parallel decoder that maps all attributes at once and again a sequential decoder where we invert the order of the decoders. The results for all three decoder are listed in Table 3. We notice that the sequential decoder is the overall best, although with a very slim margin to the parallel decoder. Interestingly, despite having the same amount of connections in the network as the parallel decoder, the reversed sequential decoder performs worse, suggesting that the order of decoding significantly impacts performance. A possible explanation for this disparity is that the outputs from earlier stages in the sequential version might introduce noise, thereby impeding the optimization process.

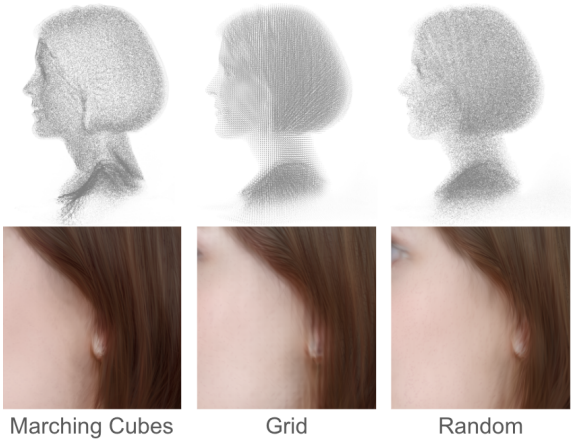

Figure 7. A visual comparison between different position sampling approaches. When using grid sampling, we observe some grid artifacts on the surface.

Architecture	LPIPS ↓	SSIM ↑	ID Sim ↑
Sequential	**0.176**	**0.842**	**0.883**
Parallel	0.177	0.841	0.879
Sequential Reversed	0.228	0.803	0.765

Table 3. Image difference metrics for decoder architectures.

Backbone Fine-tuning: During the training, we fine-tune the weights of the pre-trained StyleGAN backbone that produces the tri-plane features. This distributes some of the work load from the decoder to the backbone, as we penalize the tri-plane creation if the decoder cannot easily create Gaussian splats from it. Disabling this component leads to a decline in all performance metrics, especially the ID similarity, which drops from 0.883 to 0.858 as shown in Table 4. This demonstrates that fine-tuning the StyleGAN backbone enhances the tri-plane features for decoding them into high-quality Gaussian Splatting scenes.

Loss Functions: Training the decoder network requires appropriate loss functions that yield meaningful gradients. For our proposed decoder training, we employ a combination of several different loss functions. To better understand their individual impact, we conduct an ablation study by training multiple models, each with one loss function deactivated. The resulting renderings compared using the same ID can be seen in Figure 8. Here, the biggest difference is visible when deactivating the LPIPS loss. In this case, the rendering starts to become very blurry. This is surprising, given that L1 or SSIM are expected to penalize such blurry renderings. Instead, when disabling them, some artifacts are created at the edges. This hints that those loss functions help building the coarse geometry for the face, while LPIPS provides a gradient that creates fine structures.

Method	LPIPS ↓	SSIM ↑	ID Sim ↑
Baseline	0.176	0.842	0.883
w/o fine-tuning	0.188	0.837	0.858
w/o L1 Loss	0.175	0.841	0.881
w/o LPIPS Loss	0.260	**0.859**	**0.885**
w/o SSIM Loss	**0.174**	0.832	0.880
w/o Sobel Loss	0.175	0.839	0.880
w/o ID Loss	0.176	0.842	0.827

Table 4. Comparison of our baseline model with variants, each with a single component deactivated. While the baseline does not achieve the highest score for each metric, it offers a balanced trade-off along all three metrics combined.

5. Limitations & Future Work

Since all target images we use to train our framework are generated by either PanoHead or EG3D, the output fidelity of our method is bounded by the fidelity of these 3D GANs. A possible approach to push the visual quality of our renderings closer to photorealism would be to train the entire pipeline, i.e. training the generator backbone alongside the decoder, from scratch in a GAN based end-to-end manner. This approach, while being straightforward in theory, is subject to some challenges including the localization of good initial positions for the Gaussians in 3D space and, especially, handling the unstable nature of adversarial training. We aim to tackle these challenges in succeeding works, using the general structure of this framework and the insights obtained while developing it as a foundation.

Moreover, we observe that the eyes of the faces in our scenes appear uncanny or blurry. As described in 4.4, we expect to solve this issue by including view-dependent spherical harmonics in the future.

Figure 8. A visual comparison between decoders that have been trained while deactivating the respective loss function.

6. Conclusion

We have presented a framework that decodes tri-plane features of pre-trained 3D aware GANs for facial image generation like PanoHead or EG3D into scenes suitable for rendering with 3D Gaussian Splatting. This not only allows for rendering at up to 5 times higher frame rates with flexible image resolutions but also enables to export the resulting scenes into 3D software environments, allowing for realistic 3D asset creation in real-time. As our decoders show very high visual similarity to the 3D-aware target GANs, we are able to maintain high visual quality along interpolation paths, paving the way for applying GAN editing or GAN inversion methods to explicit 3D Gaussian Splatting scenes for the first time. In an in-depth ablation study, we discuss all components of our method, providing a basis for future works dealing with decoding Gaussian Splatting attributes. For succeeding work, we plan to broaden our training scheme to be able to train a GAN in an adversarial training for the generation of scenes compatible with 3DGS, as mentioned in the previous section.

7. Acknowledgements

This research has partly been funded by the German Research Foundation (3DIL, grant no. 502864329), the European Union's Horizon Europe research and innovation programme (Luminous, grant no. 101135724), and the German Ministry of Education and Research (MoDL, grant no. 01IS20044).

References

[1] Rameen Abdal, Peihao Zhu, Niloy J. Mitra, and Peter Wonka. Styleflow: Attribute-conditioned exploration of stylegan-generated images using conditional continuous normalizing flows. *ACM Trans. Graph.*, 40(3), 2021.

[2] Sizhe An, Hongyi Xu, Yichun Shi, Guoxian Song, Umit Y. Ogras, and Linjie Luo. Panohead: Geometry-aware 3d full-head synthesis in 360deg. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20950–20959, 2023.

[3] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields, 2021.

[4] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5460–5469, 2021.

[5] Florian Barthel, Anna Hilsmann, and Peter Eisert. Multi-view inversion for 3d-aware generative adversarial networks. In *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. SCITEPRESS - Science and Technology Publications, 2024.

[6] Stephan Brehm, Florian Barthel, and Rainer Lienhart. Controlling 3d objects in 2d image synthesis. *SN Computer Science*, 4(1), 2022.

[7] Eric Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *arXiv*, 2020.

[8] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *arXiv*, 2021.

[9] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*, page 333–350, Berlin, Heidelberg, 2022. Springer-Verlag.

[10] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4685–4694, 2019.

[11] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14326–14335, Los Alamitos, CA, USA, 2021. IEEE Computer Society.

[12] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls. In *Proc. NeurIPS*, 2020.

[13] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. Relu fields: The little non-linearity that could. In *ACM SIGGRAPH 2022 Conference Proceedings*, New York, NY, USA, 2022. Association for Computing Machinery.

[14] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

[15] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks, 2019. arXiv:1812.04948 [cs, stat].

[16] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8107–8116, Seattle, WA, USA, 2020. IEEE.

[17] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. In *Proc. NeurIPS*, 2021.

[18] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4), 2023.

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

[20] Muhammed Kocabas, Rick Chang, James Gabriel, Oncel Tuzel, and Anurag Ranjan. Hugs: Human gaussian splats, 2023.

[21] Yongjae Lee, Li Yang, and Deliang Fan. Mf-nerf: Memory efficient nerf with mixed-feature hash table, 2023.

[22] Tianye Li, Timo Bolkart, Michael. J. Black, Hao Li, and Javier Romero. Learning a model of facial shape and expression from 4D scans. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 36(6):194:1–194:17, 2017.

[23] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *Advances in Neural Information Processing Systems*, pages 15651–15663. Curran Associates, Inc., 2020.

[24] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*, pages 163–169. ACM, 1987.

[25] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR*, 2021.

[26] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[27] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: representing scenes as neural radiance fields for view synthesis. *Commun. ACM*, 65(1):99–106, 2021.

[28] Li Ming, Zhou Pan, Liu Jia-Wei, Keppo Jussi, Lin Min, Yan Shuicheng, and Xu Xiangyu. Instant3d: Instant text-to-3d generation. *arxiv: 2311.08403*, 2023.

[29] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets, 2014. arXiv:1411.1784 [cs, stat].

[30] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids, 2023.

[31] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4), 2022.

[32] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting triangular 3d models, materials, and lighting from images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8280–8290, 2022.

[33] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11448–11459, Nashville, TN, USA, 2021. IEEE.

[34] Shenhan Qian, Tobias Kirschstein, Liam Schoneveld, Davide Davoli, Simon Giebenhain, and Matthias Nießner. Gaussianavatars: Photorealistic head avatars with rigged 3d gaussians. *arXiv preprint arXiv:2312.02069*, 2023.

[35] Daniel Roich, Ron Mokady, Amit H Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. *ACM Trans. Graph.*, 2021.

[36] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022.

[37] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. In *ACM SIGGRAPH 2022 Conference Proceedings*, New York, NY, USA, 2022. Association for Computing Machinery.

[38] Yichun Shi, Divyansh Aggarwal, and Anil K. Jain. Lifting 2d stylegan for 3d-aware face generation. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6254–6262, 2020.

[39] Yichun Shi, Divyansh Aggarwal, and Anil K. Jain. Lifting 2D StyleGAN for 3D-Aware Face Generation. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6254–6262, Nashville, TN, USA, 2021. IEEE.

[40] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5449–5459, 2021.

[41] Yiqian Wu, Jing Zhang, Hongbo Fu, and Xiaogang Jin. Lpff: A portrait dataset for face generators across large poses. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 20327–20337, 2023.

[42] Hongyi Xu, Guoxian Song, Zihang Jiang, Jianfeng Zhang, Yichun Shi, Jing Liu, Wanchun Ma, Jiashi Feng, and Linjie Luo. Omniavatar: Geometry-guided controllable 3d head synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12814–12824, 2023.

[43] Yuelang Xu, Benwang Chen, Zhe Li, Hongwen Zhang, Lizhen Wang, Zerong Zheng, and Yebin Liu. Gaussian head avatar: Ultra high-fidelity head avatar via dynamic gaussians. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[44] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021.

[45] Lan Yushi, Tan Feitong, Qiu Di, Xu Qiangeng, Genova Kyle, Huang Zeng, Fanello Sean, Pandey Rohit, Funkhouser Thomas, Loy Chen Change, and Zhang Yinda. Gaussian3diff: 3d gaussian diffusion for 3d full head synthesis and editing. *arXiv*, 2023.

[46] Jian Zhang, Jinchi Huang, Bowen Cai, Huan Fu, Mingming Gong, Chaohui Wang, Jiaming Wang, Hongchen Luo, Rongfei Jia, Binqiang Zhao, and Xing Tang. Digging intonbsp;radiance grid fornbsp;real-time view synthesis withnbsp;detail preservation. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XV*, page 724–740, Berlin, Heidelberg, 2022. Springer-Verlag.

[47] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

[48] Zhongyuan Zhao, Zhenyu Bao, and Qing Li. Psavatar: A point-based morphable shape model for real-time head avatar animation with 3d gaussian splatting. https://synthical.com/article/c23e360e-3bbb-4410-a7d7-dc4202a76501, 2024.

[49] Zi-Xin Zou, Zhipeng Yu, Yuan-Chen Guo, Yangguang Li, Ding Liang, Yan-Pei Cao, and Song-Hai Zhang. Triplane meets gaussian splatting: Fast and generalizable single-view 3d reconstruction with transformers. *arXiv preprint arXiv:2312.09147*, 2023.