Gaussian4Triplane: Gaussian Feature Network for Triplane-based 3D Large Reconstruction Model with Very Few Parameters

Qi-Yuan Feng, Tai-Jiang Mu*, Shi-Min Hu
BNRist, Department of Computer Science and Technology, Tsinghua University
fqy22@mails.tsinghua.edu.cn

Abstract

This paper introduces Gaussian4Triplane, a novel 3D representation with very few parameters, based on our newly proposed Gaussian Feature Network, i.e. *GFeature*. During our research, we observed that the number of parameters in the 3D representation model of current large feed-forward generation models is severely constrained. This is because the number of parameter of the entire generation model increases along with the number of parameters of the 3D representation model. However, it is challenging to reduce the number of parameters in the 3D representation model without compromising the quality of the overall model. Therefore, we began by focusing on the basic feature representation module. To reduce the size of the neural network module, we propose GFeature, a novel neural network module designed to improve the efficiency of feature representation. GFeature operates by: 1) transforming discrete feature vectors into a latent space with a learnable 1D Gaussian mixture model, 2) enriching latent vectors with denser, multi-scale information, and 3) enabling the network to retain more relevant features while reducing computational overhead. Returning to 3D large reconstruction model, we propose Gaussian4Triplane, whose triplane features are from the GFeature. Gaussian4Triplane is a new 3D representation capable of reconstructing a complete 3D object with just 17.5K parameters and requiring less than 2GB of GPU memory. This representation can not only improve the reconstruction quality but also enhances the representational power of 3D large reconstruction and generation models.

***Keywords:** 3D reconstruction Triplane, 3D representation 3D Gaussian Splatting, 3D Neural Network Neural Rendering*

*Corresponding author

1. Instruction

3D large generation models can greatly improve the work efficiency of designers and modelers. These models are typically used to generate coarse models, which are then fine-tuned by designers. The most efficient type of 3D large generation models is the feed-forward model or diffusion model, such as LRM [11, 32]. However, these models often require a large number of parameters, and the number of parameter of the 3D generation model is directly related to the number of parameters of the final 3D representation it generates. Specifically, the resolution of the generated triplane is usually 64×64 [11, 34], which is too low for high-quality 3D representations. Therefore, we focus on addressing two main challenges: 1) How to represent a model that can achieve similar results with fewer parameters, thus reducing the computational load and improving generation speed; 2) How to represent a higher-quality 3D model with the same number of parameters, improving the overall generation quality. Given that this compression is challenging with the current neural network structures, our objective is to solve this problem by designing a new neural network architecture for feature representation.

The fundamental representation unit in neural networks is often treated as discrete data groups, such as vectors, as seen in architectures such as Multi-Perceptron (MLP) [21], Convolutional Neural Network (CNNs) [15], Recurrent Neural Network (RNN) [26], and Attention [1]. Is vector-based data representation the most effective? In the domain of 3D representation, a more explicit structure not only improves the explainability but also simplifies the process, leading to better performance. This suggests that incorporating explicit features and explainability into implicit neural networks could enhance their representational capabilities. Despite this, many current neural networks are limited to processing vectors. Since data in computers is inherently stored in discrete formats, representing data as vectors continues to be the most direct and convenient approach. As a result, it is challenging to move beyond the traditional discrete vector representation and explore new network architecture to process discrete data.

Drawing inspiration from the concepts of 3D Gaussian

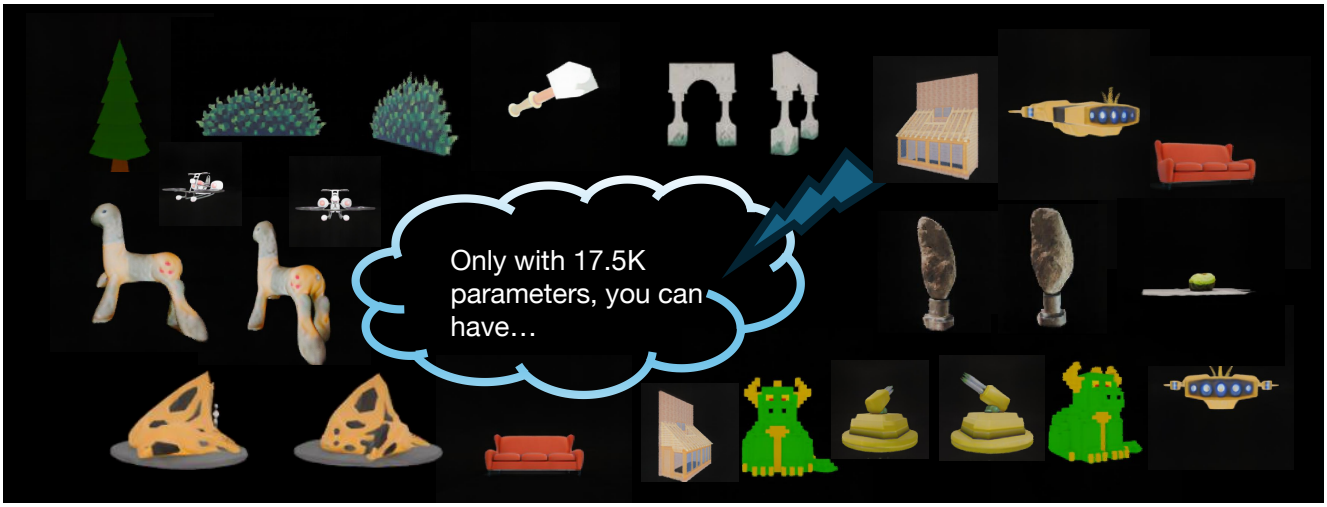

Figure 1. Our GFeature and Gaussian4Triplane can complete the training and inference of triplane based 3D large reconstruction model with very few parameters representation, i.e. only 17.5K. It is much smaller than the current commonly used methods and can be used as a generated hidden space for the generation model to get finer and better results.

Splatting [13] and Gaussian image [39], we propose Gaussian Feature Network, i.e. GFeature, a novel neural network module for discrete data processing. Our approach begins by decomposing a vector of length L into N tiles. We then utilize an optimizable tensor to represent the influenced location and range of each data unit. To achieve this, a Gaussian weighting strategy is applied to each tile separately, resulting in a target-weighted vector. This is the first attempt to integrate Gaussian feature fusion into a neural network inference module. Notably, this fusion process significantly reduces the parameter load during network inference. For instance, a vector of length 256 can be effectively represented using just 64 GFeatures, which can express multi-scale information and enhance the concentration of information density.

In 3D large generation models, such as [11], [33], and [34], the resolution of Triplanes often acts as a bottleneck for model accuracy, with most triplane resolutions typically set to 64. To address this, we integrate our GFeature module with triplane 3D representations, enabling higher triplane resolution and improving reconstruction results with minimal parameter overhead. Additionally, we have designed several acceleration strategies, including the batch-channel inversion, masked irregular rasterization, and rasterize-resolution warm-up, to enhance the inference speed of GFeature and Gaussian4Triplane. We validated our method on the MNIST [16], Objaverse [4], NeRF synthetic[22] and NeRF LLFF [22] datasets. The results demonstrate that our method achieves comparable performance with fewer parameters, or better performance with the same number of parameters.

In summary, our contributions are as follows:

- a novel neural network module, named GFeature, which can transform discrete input data to continuous efficiently for better feature representation.
- a new 3D representation, named Gaussian4Triplane, that significantly reduces GPU memory and network parameter usage for 3D large reconstruction models.
- several acceleration strategies for GFeature and Gaussian4Triplane, achieving comparable inference speeds with existing models.

2. Related work

2.1. Neural Network Modules

The common practice for processing vectors is the MLP [21], which involves multiplying an N-dimensional vector by an N×M weight matrix and adding an M-dimensional bias vector to obtain an output vector of length M, as shown in Fig. 2. It can perform real-time simultaneous localization [35], serve as 3D representations [22], enable object recognition [14], and accomplish simultaneous translation [19]. Similarly, CNN [15] performs block-based matrix multiplication and addition. Attention [1] calculates the inner product of vectors. The Attention architecture has been widely used in large model including large language model such as GPT[24, 25, 20] and diffusion model[10, 28]. While these methods process vectors as whole units, they often result in redundant computations and excessive parameters. Some approaches [29, 17] have sought to add explicit information to attention mechanisms, allowing the network to focus on relevant parts and improve both computational efficiency and performance. However, the fixed connectivity of neurons in these structures prevents optimization of the connections, meaning the network cannot adapt the structure to

better suit the specific task at hand.

2.2. 3D Reconstruction and Generation Model

3D representation refers to how a 3D model is represented in computers. It can be divided into two categories: implicit and explicit representations. Explicit representations include point clouds, meshes, voxels, and 3D Gaussian Splatting[13]. These representations are characterized by faster rendering speed and easier and convenient to process. In contrast, implicit representation usually represent the 3D surface with a functional field, such as singed distance field, occupancy field and NeRF [22, 23, 7, 6, 12].

The advantage of implicit networks lies in their natural adaptation to the function fitting ability of deep neural networks, enabling 3D reconstruction and generation from texts or images.

Influenced by large-scale image generation models [28, 10] and large language models [24, 25, 20], the development of large 3D generation models has also advanced rapidly. Many models now utilize diffusion's generative capabilities to create multi-view consistent images [18, 27, 11, 33, 32, 40] or latent vectors [31, 37, 36]. In addition to combining feed-forward models and diffusion for 3D generation, there are also numerous approaches using self-regressive networks to generate 3D models with more parameters[2, 3]. The feed-forward generation model offers better stability and faster generation speeds. However, the resolution of the triplane structure generated by many feed-forward models is often too low (e.g., $64 \times 64 \times 3$), leading to very coarse generated models. On the other hand, auto-regressive models generate more parameters, resulting in finer models. Therefore, our focus is how to represent a finer 3D model with fewer parameters, allowing feed-forward generation models to produce higher-quality 3D models within a small network parameter budget.

3. GFeature: Gaussian Feature Network

Let's first think about the weighting schemes used in current neural networks. For MLP, each dimension of the output feature vector $f_{out} \in R^M$ is determined by a free weight $W = [w_0^T, \cdots, w_{M-1}^T] \in R^{M \times N}$ for the input feature $f_{in} \in R^N$ and a bias $b \in R^M$:

$$f_{out} = W \times f_{in} + b \tag{1}$$

The Jacobi matrix of the fully trained MLP that impact of many dimensions of the neural network on the final results is almost 0. This shows that the network of MLP produces a large number of redundant calculation and intermediate results in the operation.

Secondly, let's take a look at the structure of another neural network, i.e. external-attention [8], which can be expressed:

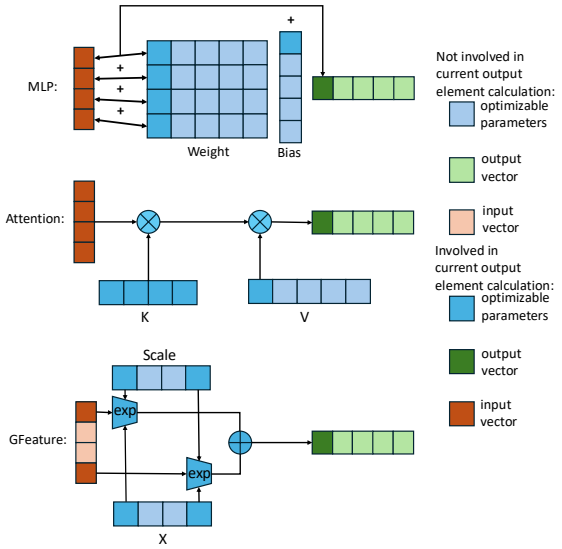

Figure 2. Comparison of three different neural network architectures. In this article, we compress the element dimension of attention to 1 for better illustration. Among these three architectures, MLP has the largest number of parameters. Our GFeature can be optimized to use the required parameter through calculation, and the other parameters in GFeature will be filtered out, so as to have less calculation and faster speed and more detailed calculations and operations can be carried out.

$$f_{out} = \frac{1}{Z} exp(\frac{< q(f_{in}), k >}{\sqrt{d}}) v \tag{2}$$

where Z is the normalization coefficient, q is the query vector, k and v is the key and value vector/matrix of the input vector f_{in}, d is the dimension of q or k. This way can improve adaptability more intuitively, and the weight can be directly calculated through the inner product. However, this operation still needs to calculate the weight of all positions once as shown in Fig. 2.

Can the model automatically learn which positions' weights to retain and which positions' weights to ignore? The answer is yes. Inspired by the rasterization function of 3D Gaussian Splatting [13], we realize that high-dimensional space can also be fit by Gaussian functions [9, 5] and the scope of each feature's impact in high-dimensional space can be characterized by its central position x and scale s. The input feature of a neural network layer is consist of many vectors or many scalars to represent the high-dimensional information. Therefore, we draw on the idea of 3D Gaussian and set a learnable 1D central position x and a learnable 1D scale s to learn each input vector's influence in the latent space. We thus design a new latent feature representation strategy $\mathcal{R}_{1d}$. To adapt to the downstream network, we regard each vector or scalar as the color field of the 3D Gaussian Splatting kernel. Correspond

to each input element position i, we preset a set of trainable parameters, including the offset coordinates, ox_i, and range of influence, s_i. So $\mathcal{R}_{1d}$ transforms the input feature f_{in} into f_{out} as:

$$f_{out} = \mathcal{R}_{1d}(f_{in}, S = \{s_i\}_0^{N-1}, X = \{i + ox_i\}_0^{N-1}) \quad (3)$$

If we raster each element in each output feature grid, the calculation will be very complex. In fact, the unrelated positions or position with little influence could be ignored. For this reason, we designed two schemes to reduce the unnecessary calculation. The first one is similar to the 3D Gaussian Splatting, where we divide a input feature into multiple tiles, ensuring that the positions of the input elements that intersect with the grid points inside the same tile are the same. We denote the set of indices of Gaussian kernels affecting each tile as $\mathbb{T}$, and then dynamically change this set so that we can optimize it to the required position as close as possible, just as GaussianImage[39] did. This process can be expressed as:

$$\begin{aligned} f_{out} = \mathcal{R}_{1d}^{\mathbb{T}}(&f_{in}, \\ &S = \{s_i\}_0^{N-1}, \\ &X = \{i + ox_i\}_0^{N-1}, \\ &\mathbb{T}) \end{aligned} \quad (4)$$

Furthermore, our rasterization process only affects the current output vector but not the multi-view images in 3D Gaussian Splatting[13], so the volume rendering process in rasterization is unnecessary. We thus can directly use a weighting scheme to replace this process. Therefore, the final computation of our Gaussian Feature Network, GFeature, can be written as:

$$f_{out}^i = \Sigma_{j=min(\mathbb{T})}^{max(\mathbb{T})} f_{in}^j \exp(-\frac{1}{2}\frac{(i-(j+ox_j))^2}{s_j}) \quad (5)$$

where f_{in}^j is the j-th element in the initial input feature, f_{out}^i is the i-th element in the output vectors. A more detailed comparison with other network architectures are shown in Fig. 2. Another advantage of GFeature is that it can compress features to the required length more efficiently compared with MLP.

Batch-channel inversion strategy At this point, there may be a question: if each output element is processed separately, wouldn't it be a waste of the parallel ability of the graphics card? In this way, the batch size can only be set to 1 during training. It seems that the above process can only calculate each output element separately in each batch. This is actually a brain teaser. Suppose we have an input tensor with the shape of $[B, N, C]$, where B is batch size, N is the number of element and C is the dimension of element. If we want to deal with this input tensor, in fact, a simple permute can achieve parallel: 1) Each of our process blocks needs to process a tile, which also corresponds to multiple Gaussian kernels. Therefore, we directly advance the element number dimension, N, that is to permute the shape as $[N, B, C]$. 2) To make the calculation faster, we further flatten it into $[N, B \times C]$. 3) After completing the calculation, we perform a reverse operation on the output in the shape of $[O, B \times C]$ to $[B, O, C]$, where O is the output element number dimension of our GFeature Module. In addition, if the shape of the object we handle is $[N, C]$, it is equivalent to putting the last dimension as 1.

4. Gaussian4Triplane

We observed that in many existing 3D generation models, the parameter quantity is positively related to the representation ability of the generated 3D model. However, the resolution of the triplane generated in current large reconstruction model [11] is only 64×64, severely limiting the finer details of the results. Since our GFeature can efficiently compress an extremely long feature vector set into a shorter feature vector set. Therefore, if we can compress the structure of the triplane and improve its resolution with negligible computational overhead. We extend the Eq. 5 from 1D to 2D and divide the optimizable parameters into three parts, 2d coordinates xy, 2d covariance s and features f_{in} for each plane, as illustrated in Fig. 3. We complete the weighting operation of Gaussian kernels in the plane:

$$\begin{aligned} f_{out}^{ij} = &\Sigma_{p=min(\mathbb{T})}^{max(\mathbb{T})} \Sigma_{k=min(\mathbb{T})}^{max(\mathbb{T})} \\ &f_{in}^{pk} \exp(-\frac{1}{2}[\delta_x, \delta_y]^T s_{pk}^{-1} [\delta_x, \delta_y]), \\ &\delta_x = i - p - xy_p, \\ &\delta_y = j - k - xy_k \end{aligned} \quad (6)$$

Here, f_{out}^{ij} is the output feature at the grid in the i-th row and the j-th column, and f_{in}^{pk} is the input feature at the grid in the p-th row and the k-th column.

We also observed that when rendering the triplane, many positions in 3D space has a zero volume density, and many grids in the triplane correspond to such locations with zero volume density. Therefore, we can keep and update a triplane mask to filter out the coordinates that need not to be weighted.

We can render all the 3D grids to form a mask according to their densities. But the triplane grid corresponding to this mask is not that regular as before. Specifically, the shape of the masked grid is $[B]$ instead of $[H, W]$, where B is the number of valid grid masked in the $[H, W]$

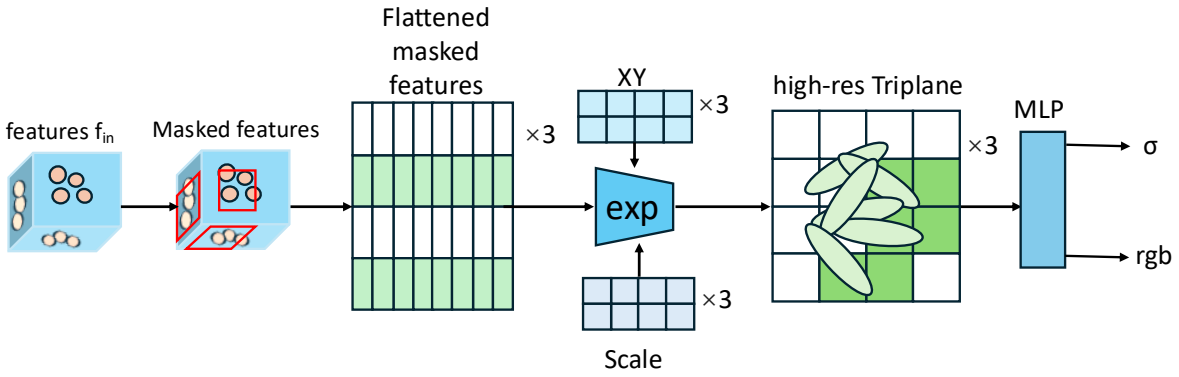

Figure 3. Gaussian4Triplane. Given the input feature f_{in}, Gaussian4Triplane first obtains the masked features via the masked tiles and $\mathbb{T}$. Then the masked features are flattened and weighted according to optimizable position XY and scale $Scale$ using the aforementioned GFeature. Finally, the output high resolution triplane can be used to reconstruct the spatial density σ and color rgb.

plane. However, the irregular output and requirements actually lead to the correspondence between each tile and B masked grid points. For example, if the size of the first unmasked tile is $[16, 16]$, the index of the corresponding grid is $\{(x, y)|x \in [0, 16], y \in [0, 16]\}$, which be paralleled easily. But it is challenging for the masked irregular tile.

Masked irregular rasterization strategy To solve the aobve problem, we designed an irregular rasterization strategy. Specifically, we flattened the 2D structure of the grid points, and set all of the grid points that need to be calculated as a column with the shape of $[B]$. Then we pre-store a list of grid indices, each of which indicates the grid point that needs to be involved in later calculation. Thirdly, we calculated the indices in the tile corresponding to grid point according to the position of each valid grid:

$$\begin{aligned} t_x &= \min(g_x // \sqrt{|\mathbb{T}|}, |I_x| // \sqrt{|\mathbb{T}|}) \\ t_y &= \min(g_y // \sqrt{|\mathbb{T}|}, |I_y| // \sqrt{|\mathbb{T}|}) \end{aligned} \tag{7}$$

where (g_x, g_y) is the index of the grid, $|\mathbb{T}|$ is the size of the tile, (I_x, I_y) is the shape of the image. We put all the grid points in the same tile together with their 2D grid index $g_p = (g_x, g_y)$. With these, we can get the scope of the grid point corresponding to each tile, $\mathbb{G}_{pk}$. Finally, we can weight the feature according to the size of the corresponding tile:

$$\begin{aligned} &\{f^i_{out}|i \in \mathbb{G}_{pk}\} = \\ &\quad \{\Sigma_{\mathbb{T}} f^{jk}_{in} \exp(-\frac{1}{2}[\delta_x, \delta_y]^T s^{-1}_{pk}[\delta_x, \delta_y])| \\ &\quad\quad \delta_x = g_p[i][0] - p - xy_p, \\ &\quad\quad \delta_y = g_p[i][1] - k - xy_k, \\ &\quad\quad i \in \mathbb{G}_{pk}]\} \end{aligned} \tag{8}$$

Rasterize-resolution warm-up strategy Moreover, since the 3D position initialization is a key step in 3D Gaussian Splatting [13], the quality of initialization have a great impact on the final reconstruction result. We think the reasons maybe: 1) most of the Gaussian kernels are actually located inside a specific tile. 2) when the gradient is back-propagated, every Gaussian kernel optimizes their position inside a tile of each picture. So, in the limited number of iterations, it's difficult for Gaussian kernels to move from one tile of a picture to another. To solve this problem, we designed a progressive tile strategy. Specifically, suppose we need to build a triplane with the resolution of $512 \times 512 \times 3$. To better fit the Gaussian kernels to the required position, we let each Gaussian kernels affect more grid points in the first step. At the beginning, we set the resolution of the calculated triplanes to $64 \times 64 \times 3$. Then we double the output resolution of the target triplanes every 200 steps. In this way, the Gaussian kernels can be optimized to the required position faster with a larger initial range of influence.

5. Experiments

The experiments are divided into two parts. The first part evaluates the effectiveness and efficiency of our GFeature on the MNIST [16] dataset. The second part tests the performance of our Gaussian4Triplane on the Objaverse [4] dataset, along with several ablation studies to demonstrate the effectiveness of the core components of Gaussian4Triplane. We conducted all the experiments on a single Nvidia A100PRO GPU device.

5.1. The Properties of GFeature

To illustrate the validity of GFeature, we first test some basic attributes of this network module compared with MLP and Attention. Suppose the shape of the input vector is $[B, N]$, and the shape of the output vector is $[B, M]$. So,

Table 1. We fix M to 512 and change N to test the storage occupancy.

storage occ	128	256	512	2048
MLP	258KB	514KB	1026KB	4098KB
Attention	260KB	518KB	1030KB	4114KB
GFeature(ours)	1KB	2KB	4KB	16KB

Table 2. We fix N to 512 and change M to test the storage occupancy.

storage occ	128	256	512	2048
MLP	257KB	513KB	1026KB	4104KB
Attention	261KB	517KB	1030KB	4120KB
GFeature(ours)	4KB	4KB	4KB	4KB

Table 3. We fix M to 512 and change N to test the GPU memory occupancy.

GPU mem	128	256	512	2048
MLP	8.73MB	9.01MB	9.57MB	12.94MB
Attention	8.87MB	9.15MB	9.71MB	13.09MB
GFeature(ours)	0.36MB	0.40MB	0.47MB	0.86MB

Table 4. We fix N to 512 and change M to test the GPU memory occupancy.

GPU mem	128	256	512	2048
MLP	8.72MB	9.01MB	9.57MB	12.95MB
Attention	8.77MB	9.09MB	9.71MB	13.48MB
GFeature(ours)	0.37MB	0.40MB	0.47MB	0.86MB

the number of parameters of a one-layer MLP is $O(N \times M + M)$, because the optimizable variable is the weight matrix of shape $[M, N]$ and a bias vector of shape $[M]$. The number of parameters of the external attention[8] is $O(N \times M \times 2 + M \times 4)$, since there are two linear layers and two vector sets as Key and Value. Our GFeature only needs two vectors to record the position and scale corresponding to each input feature element, so the number of parameters if $O(N \times 2)$.

Therefore, we tested the effects of different M and N on the storage memory of the model, and the results are shown in Tables 1 and 2 (In this part of experiment, we fixed B to 64 for testing). It can be seen that our model occupies the smallest amount of storage memory, which is much smaller than the other two network architectures. It can be also observed that our module is a very convenient module to transform features to any latent size without increasing the network parameters.

We also tested the graphics memory occupancy of different network architectures, and the results are shown in Tables 3 and 4. As we can see, our GFeature also occupied much less GPU memory than the other two architectures.

As for the time complexity, due to the strong parallel ability of GPU, the size of latent vector does not affect processing time very much. Specifically, the time consumption of MLP, Attention and Our GFeature are 0.0001s, 0.0005s and 0.0006s, respectively. The calculation time of our parallel module is still within the acceptable range considering the small network paremeters and GPU memory used.

Table 5. The comparison of fitting ability of different models on a general fitting task.

	MSE ↓	storage occ.↓	GPU mem.↓
MLP	0.2863	4MB	16.32MB
Attention	0.3628	8.1MB	25.33MB
GFeature(ours)	0.4263	8KB	1.75MB
GFeature(ours)+MLP	0.2562	4MB	18.13MB

We also tested the generalization effect of our model on MNIST[16]. We replace the first layer of two-layer MLP($[9152, 128]$) in MNIST with our GFeature and retained the MLP of the second layer, achieving an classification accuracy of 96%, compared with the classification accuracy of 99% of the original two-layer MLP. However, the parameter amount of our model is less than 1/50 of the original one. And we also tried to add our GFeature as an activate layer before the original two-layer MLP and the classification accuracy also comes to 99%. Specifically, in fact, our GFeature is actually a sampling of effective parameters in MLP. That is, the effective parameter position in the original network is calculated by our Gaussian weighting. Therefore, after training, if the GPU memory is sufficient, the stored GFeature can be converted to the weight matrix of MLP by setting the weight of positions beyond the influence of the Gaussian function to zero, to achieve faster inference speed.

Despite the specific MNIST task, we further designed a general task to test the fitting ability of different models, including the one-layer MLP, external attention, our GFeature, and our GFeature followed by one-layer MLP. Specifically, we set up the input dataset (512 random input vectors with a length of 1024) and the output dataset (512 random output vectors with a length of 1024) to test the fitting error (i.e., mean square error, MSE) of MLP, Attention and GFeature after training. The training results are shown in Table 5. As we can see, our GFeature can utilize very few parameters to fit the target. Besides, our module can be combined with MLP to achieve better fitting results.

5.2. Experiments of Gaussian4Triplane

5.2.1 Comparison to other methods

To ensure that our Gaussian4Triplane is applicable to the large 3D generation model, we use the same structure as InstantMesh [34] with four layers of MLP and the dimension of each layer is 64. But in instantMesh, the resolution of the triplane structure is similar to that of other networks, saying $64 \times 64 \times 3$. We randomly selected 10 objects from Objaverse[4] for experiment. We use a learning

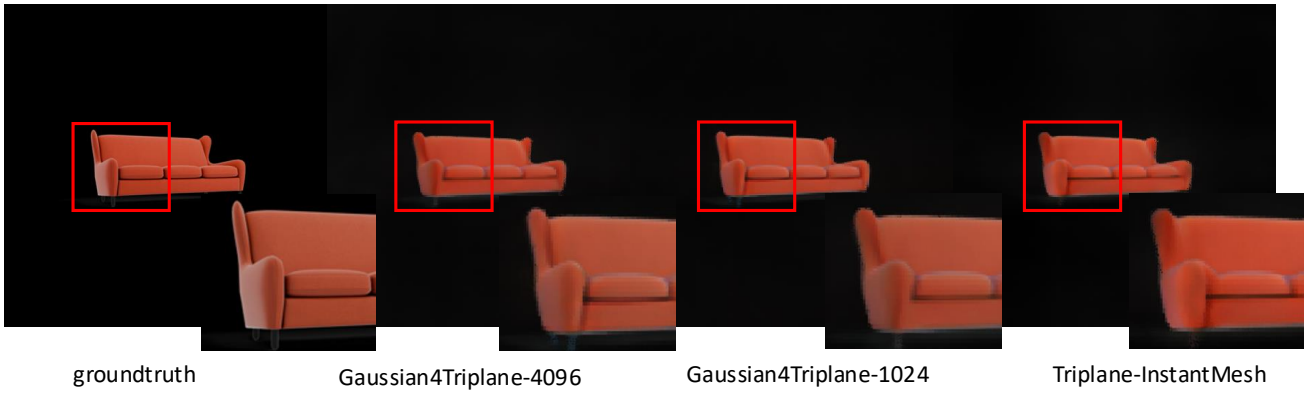

Figure 4. Visual comparison of different methods. In this case, our method can fit the color of the object more accurately and also have a better 3D geometry. While in the reconstruction results of the triplane method [34], the armrest and the cushion are glued together.

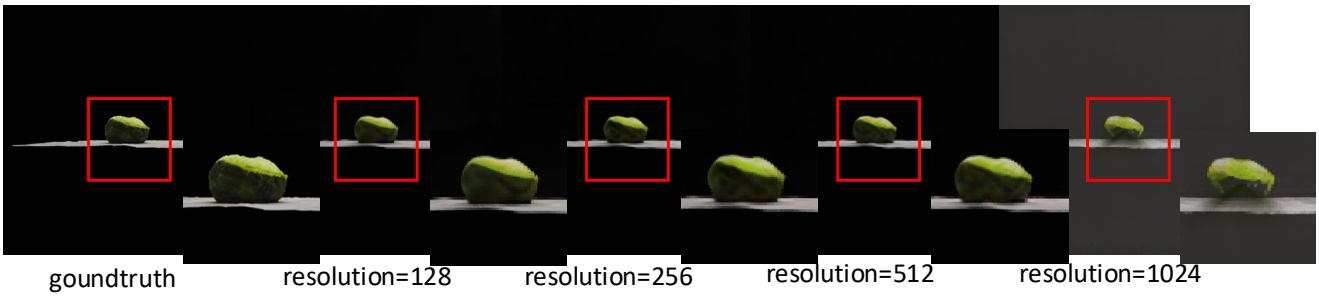

Figure 5. We fix the number of feature elements in f_{in} to 4096, and change the resolution of the target feature plane for comparison. We found that the higher the resolution of the target plane, the more details and the finer the texture. But when the resolution is too high, there will be fewer elements in each tile, leading to worse results.

rate of 0.001 to train our model for 10000 iterations with the Adam optimizer. We reported several metrics, including PSNR (Peak Signal to Noise Ratio), SSIM (Structural Similarity Index[30]) and LPIPS (Learned Perceptual Image Patch Similarity[38]). The quantitative comparison results are shown in Table 6 and the visual comparison results are show in Fig 4 and 6. The model ours(X) means that the number of feature elements in f_{in} is X in each plane during reconstruction. We set the resolution of the generated high-res plane to 256×256 and 512×512 for ours(1024) and ours(4096), respectively, to achieve the best effect. It can be seen that our Gaussian4Triplane also benefits from the excellent properties of GFeature. Ours(1024) can reconstruct more accurate 3D models with fewer parameters. And ours(4096) can achieve even better reconstruction effect than ours(1024). Besides, the inference speed is 0.36s and 0.33s for ours(4096), ours(1024), respectively, very comparable to Triplane-InstantMesh (0.32s).

We also compared our method with NeRF[22] and 3D Gaussian Splatting[13] on NeRF Synthetic dataset. The results are shown in Table 7. It can be seen that although the 3D Gaussian Splatting[13] representation has higher rendering quality, it takes up a huge amount of parameters, which is very inconvenient for the generation and the storage of the model. Our model can take up very few parameters while achieving comparable reconstruction quality, which is still better than NeRF[22]. More results are shown in the supplementary materials. In addition, if we do not use the resolution-warmup strategy, our model will not be able to fit to the corresponding structure, causing the entire reconstruction to fail.

Table 6. We compared PSNR, SSIM and LPIPS in the reconstruction results. Our GFeatures(1024) can use fewer parameters, about 0.07M strorage space, to complete the reconstruction of the 3D model.

	PSNR↑	SSIM↑	LPIPS↓	model size↓
Triplane[34]	29.63	0.516	0.040	0.2M
ours(1024)	29.99	0.581	0.038	0.07M
ours(4096)	31.11	0.656	0.035	0.2M

Table 7. We compared PSNR, SSIM and LPIPS in the NeRF synthetic dataset[22] with NeRF and 3D Gaussian Splatting

	PSNR↑	SSIM↑	LPIPS↓	model size↓
NeRF[22]	31.01	0.947	0.081	2M
3DGS[13]	31.54	0.961	0.043	¿100M
ours	31.12	0.950	0.079	0.3M

5.2.2 Ablation Experiments

We conducted ablation studies on the resolution of each high-res plane and the number of feature element in f_{in}. The experiment setting is the same as Sec.5.2.1. We also tested PSNR, SSIM[30] and LPIPS[38] on the reconstructed 3D objects.

The influence of the resolution of each high-res plane In this experiment, we fixed the number of feature element in f_{in} to 1024 or 4096, and change the resolution of each high-res plane from 64 to 1024. The results are shown in Table 8 and Fig. 5. As we can see, the quality of the reconstruction results goes up first and degrades suddenly as the resolution increases. That is because our strategy constrains that each element in f_{in} is distributed in one tile. If the number of tiles increases, the range of influence of each element in f_{in} decreases, reducing the number of weighting Gaussian's and their scopes of influence. Therefore, for each grid point, the effect of weighting would get worse. However, when the resolution is relatively high, the details conveyed by the plane increase. According to the Nyquist–Shannon sampling theorem,if the sampling frequency is high enough, the reconstruction effect is enhanced. Besides, we empirically found that the relationship between the resolution R of the best reconstruction effect and the number of feature elements in f_{in}, N, satisfies $R^2 = 64 \times N$.

The influence of the number of the feature element In this experiment, we fixed the resolution of each high-res plane to 256, and change the the number of the feature element in f_{in} from 512 to 8192. The results are shown in Table.9. We can see that the reconstruction quality basically improves with the increase in the number of feature elements. However, the improvement effect would not significant when it surpass a threshold, e.g. 1024. This is because the representation ability of the GFeature is strong enough, and the reconstruction quality is actually limited by the resolution of the high-res plane. We have also tested on NeRF synthetic[22] and LLFF dataset[22], and we can get similar reconstruction results as prior works while occupying fewer parameters. Please refer to the supplementary materials for more results.

Table 8. We compared PSNR, SSIM and LPIPS in the reconstruction results of different resolution and the number of feature elements in f_{in} is 4096/1024.

resolution/elements number	PSNR↑	SSIM↑	LPIPS↓
1024/4096	22.12	0.197	0.0683
512/4096	31.11	0.656	0.0354
256/4096	30.21	0.632	0.0358
128/4096	30.09	0.628	0.0391
64/4096	30.27	0.587	0.0434
1024/1024	19.48	0.068	0.0823
512/1024	14.46	0.069	0.118
256/1024	30.14	0.635	0.0383
128/1024	29.61	0.468	0.0401
64/1024	29.73	0.536	0.0433

Table 9. We compared PSNR, SSIM and LPIPS in the reconstruction results by fixing the resolution of the target plane to 256 and changing the number of feature element in f_{in}.

Elements Number	PSNR↑	SSIM↑	LPIPS↓
512	25.37	0.279	0.0622
1024	30.14	0.635	0.0383
2048	30.45	0.630	0.0364
4096	30.21	0.632	0.0358
8192	30.61	0.669	0.0349

6. Conclusion

This paper tackles a challenging objective for 3D large reconstruction model: how to reconstruct a higher-quality 3D model with a limited number of parameters. To achieve this, we propose a novel Gaussian feature network named GFeature which occupies much fewer parameters to transform input features into any latent size. Furthermore, we combine this module with the triplane strategy to achieve better reconstruction accuracy with a very few number of parameters.

In the future, our GFeature can be applied to various large generation models to obtain stronger representation while maintaining a certain amount of parameters. And our Gaussian4Triplane can also be used as a 3D representation with fewer parameters to further reduce the number of parameters and enhance the reconstruction results of 3D large generation model such as LRM[11], VecSet[36], Trellis[32], etc.

References

[1] D. Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 1, 2

[2] Y. Chen, T. He, D. Huang, W. Ye, S. Chen, J. Tang, X. Chen, Z. Cai, L. Yang, G. Yu, G. Lin, and C. Zhang. Meshanything: Artist-created mesh generation with autoregressive transformers. *CoRR*, abs/2406.10163, 2024. 3

[3] Y. Chen, Y. Wang, Y. Luo, Z. Wang, Z. Chen, J. Zhu, C. Zhang, and G. Lin. Meshanything V2: artist-created mesh generation with adjacent mesh tokenization. *CoRR*, abs/2408.02555, 2024. 3

[4] M. Deitke, R. Liu, M. Wallingford, H. Ngo, O. Michel, A. Kusupati, A. Fan, C. Laforte, V. Voleti, S. Y. Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. *Advances in Neural Information Processing Systems*, 36, 2024. 2, 5, 6, 11

[5] Q.-Y. Feng, G.-C. Cao, H.-X. Chen, Q.-C. Xu, T.-J. Mu, R. Martin, and S.-M. Hu. Evsplitting: an efficient and visually consistent splitting algorithm for 3d gaussian splatting. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11, 2024. 3

[6] Q.-Y. Feng, H.-X. Chen, Q.-C. Xu, and T.-J. Mu. Sls4d: Sparse latent space for 4d novel view synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 2024. 3

[7] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5501–5510, 2022. 3

[8] M.-H. Guo, Z.-N. Liu, T.-J. Mu, and S.-M. Hu. Beyond self-attention: External attention using two linear layers for visual tasks, 2021. 3, 6

[9] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979. 3

[10] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 2, 3

[11] Y. Hong, K. Zhang, J. Gu, S. Bi, Y. Zhou, D. Liu, F. Liu, K. Sunkavalli, T. Bui, and H. Tan. Lrm: Large reconstruction model for single image to 3d. *arXiv preprint arXiv:2311.04400*, 2023. 1, 2, 3, 4, 8

[12] L. Jiang, J. Lin, K. Chen, W. Ge, X. Yang, Y. Jiang, Y. Lyu, X. Zheng, Y. Li, and Y. Chen. Dimer: Disentangled mesh reconstruction model, 2025. 3

[13] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 2, 3, 4, 5, 7

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012 alexnet. *Adv. Neural Inf. Process. Syst*, pages 1–9, 2012. 2

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 1, 2

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2, 5, 6

[17] P. Li, Y. Liu, X. Long, F. Zhang, C. Lin, M. Li, X. Qi, S. Zhang, W. Luo, P. Tan, et al. Era3d: High-resolution multiview diffusion using efficient row-wise attention. *arXiv preprint arXiv:2405.11616*, 2024. 2

[18] R. Liu, R. Wu, B. Van Hoorick, P. Tokmakov, S. Zakharov, and C. Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9298–9309, 2023. 3

[19] Z. Ma, Q. Fang, S. Zhang, S. Guo, Y. Feng, and M. Zhang. A non-autoregressive generation framework for end-to-end simultaneous speech-to-any translation. *arXiv preprint arXiv:2406.06937*, 2024. 2

[20] B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 1, 2020. 2, 3

[21] J. L. McClelland, D. E. Rumelhart, P. R. Group, et al. *Parallel distributed processing, volume 2: Explorations in the microstructure of cognition: Psychological and biological models*, volume 2. MIT press, 1987. 1, 2

[22] 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. *Communications of the ACM*, 65(1):99–106, 2021. 2, 3, 7, 8

[23] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022. 3

[24] A. Radford. Improving language understanding by generative pre-training. 2018. 2, 3

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 2, 3

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 1

[27] R. Shi, H. Chen, Z. Zhang, M. Liu, C. Xu, X. Wei, L. Chen, C. Zeng, and H. Su. Zero123++: a single image to consistent multi-view diffusion base model. *arXiv preprint arXiv:2310.15110*, 2023. 3

[28] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 2, 3

[29] X. Wang, Z. Zhu, G. Huang, F. Qin, Y. Ye, Y. He, X. Chi, and X. Wang. Mvster: Epipolar transformer for efficient multi-view stereo. In *European Conference on Computer Vision*, pages 573–591. Springer, 2022. 2

[30] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 7, 8

[31] S. Wu, Y. Lin, F. Zhang, Y. Zeng, J. Xu, P. Torr, X. Cao, and Y. Yao. Direct3d: Scalable image-to-3d generation via 3d latent diffusion transformer. *arXiv preprint arXiv:2405.14832*, 2024. 3

[32] J. Xiang, Z. Lv, S. Xu, Y. Deng, R. Wang, B. Zhang, D. Chen, X. Tong, and J. Yang. Structured 3d latents for scalable and versatile 3d generation. *arXiv preprint arXiv:2412.01506*, 2024. 1, 3, 8

[33] D. Xie, S. Bi, Z. Shu, K. Zhang, Z. Xu, Y. Zhou, S. Pirk, A. Kaufman, X. Sun, and H. Tan. Lrm-zero: Training large reconstruction models with synthesized data. *arXiv preprint arXiv:2406.09371*, 2024. 2, 3

[34] J. Xu, W. Cheng, Y. Gao, X. Wang, S. Gao, and Y. Shan. Instantmesh: Efficient 3d mesh generation from a single image with sparse-view large reconstruction models. *arXiv preprint arXiv:2404.07191*, 2024. 1, 2, 6, 7

[35] Z. Yin and J. Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1983–1992, 2018. 2

[36] B. Zhang, J. Tang, M. Niessner, and P. Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Transactions on Graphics (TOG)*, 42(4):1–16, 2023. 3, 8

[37] L. Zhang, Z. Wang, Q. Zhang, Q. Qiu, A. Pang, H. Jiang, W. Yang, L. Xu, and J. Yu. Clay: A controllable large-scale generative model for creating high-quality 3d assets. *ACM Transactions on Graphics (TOG)*, 43(4):1–20, 2024. 3

[38] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 7, 8

[39] X. Zhang, X. Ge, T. Xu, D. He, Y. Wang, H. Qin, G. Lu, J. Geng, and J. Zhang. Gaussianimage: 1000 fps image representation and compression by 2d gaussian splatting. In *European Conference on Computer Vision*, pages 327–345. Springer, 2025. 2, 4

[40] Z.-X. Zou, Z. Yu, Y.-C. Guo, Y. Li, D. Liang, Y.-P. Cao, and S.-H. Zhang. Triplane meets gaussian splatting: Fast and generalizable single-view 3d reconstruction with transformers. *arXiv preprint arXiv:2312.09147*, 2023. 3

Figure 6. The reconstruction results of some objects in Objaverse[4]. It can be seen that our model achieves the best quality.