3D Gaussian Texture for Real-time Mesoscale Appearance Synthesis and Rendering

Xiang Chen[1], Jia Li[2], Lu Wang[1†], and Beibei Wang[3†]

[1] Shandong University, China
[2] Monash University, Australia
[3] Nanjing University, China

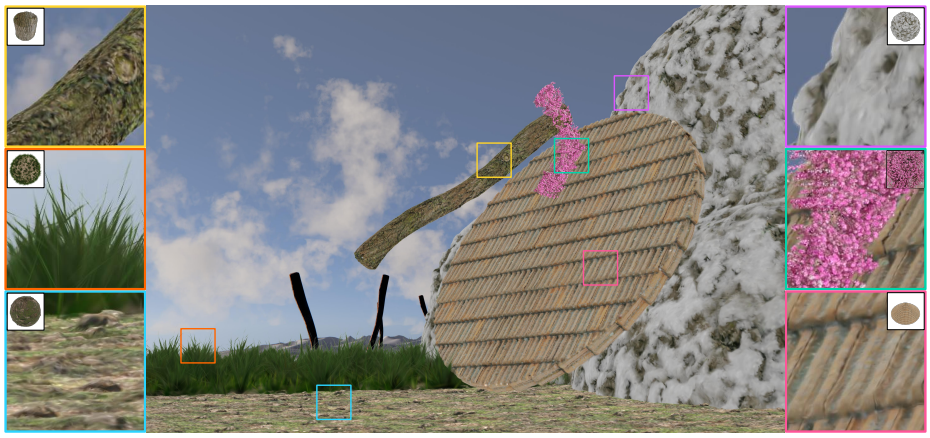

Fig. 1: We present a multi-layer Gaussian texture for real-time synthesis and rendering of mesoscale appearances modeled from multi-view images. In this scene, we synthesize and render six mesoscale appearances from both captured and synthesized images on new objects taking about 20 microseconds. Our method demonstrates a faithful representation of mesoscale appearances with preserved details and structures, which can be applied to arbitrary objects in real time.

Abstract. We propose a novel framework for modeling, synthesis, and rendering of mesoscale appearances based on 3D Gaussian Splatting (3DGS) for better quality and performance. The key problem is that synthesis of 3D Gaussians is difficult due to their explicit nature. Therefore, the core of our method is a multi-layer Gaussian texture representation to support high-quality and efficient synthesis of 3D Gaussians. Each layer of the texture contains Gaussians with similar heights to the object surface, which is compatible with 2D image texture synthesis approaches. Through by-example autocovariance-preserving synthesis, we achieve the run-time synthesis of mesoscale appearances with their structures preserved. Due to the deformation of mapping from texture to arbitrary objects and the baking of lighting, directly applying the Gaussian texture for rendering may cause visual artifacts. Thus, we

† Corresponding authors.

further introduce two components: a deformation-aware mapping and a material-lighting decomposition. The mapping is based on barycentric UV coordinates with a scaling constraint of Gaussians to reduce artifacts. The decomposed materials are represented by latent features to maintain the consistency of the material attributes during synthesis. Consequently, our approach achieves a faithful representation of mesoscale appearances modeled from multi-view images, which supports synthesis and rendering on arbitrary objects in real time, achieving about 500× acceleration compared to the latest NeRF-Texture. Our code is available at: https://github.com/ChenXiang0810/multilayer-gaussian-texture.

1 Introduction

Complex mesoscale appearances are very common and important in applications of real-time rendering, such as games and XR. Unfortunately, modeling the appearance by artists is time-consuming and labor-intensive. In addition, real-time synthesis and rendering of mesoscale appearances on arbitrary meshes are also challenging, due to their complex structures which are difficult to represent. Therefore, an efficient solution to model, synthesize and render the complex mesoscale appearance is significant.

Some existing methods model mesoscale structures through traditional representations such as maps [4, 33], curves [27] or volumes [43]. However, these methods construct mesoscale structures at the cost of heavy modeling or capturing. Moreover, most of them are designed for some particular types of structure and do not support synthesis for applying on arbitrary objects. Recently, thanks to the development of Neural Radiance Field (NeRF) [29], complex structures can be easily reconstructed from multi-view images. Taking NeRF as a texture primitive, NeRF-Tex [2] and NeRF-Texture [14] achieve a general implicit representation of mesoscale appearances, which can be applied to new objects once modeled. To render mesoscale structures on arbitrary meshes, NeRF-Tex repeats their primitives on the target mesh, while NeRF-Texture synthesizes a non-repetitive larger texture. Although the NeRF-based methods can support modeling, synthesis and rendering of the mesoscale appearances generally, they still suffer from blurry visual quality and slow speed.

In this paper, we propose a novel framework to model, synthesize and render mesoscale appearances based on 3D Gaussian Splatting (3DGS) [20]. 3DGS is well-known for its high quality and rapid speed, which can support real-time rendering with fine details, but using it on our task is not straightforward due to its explicit nature. The main problem is how to perform synthesis on explicit 3D Gaussians, which is more complex than blending in a latent space as NeRF. To overcome this, we introduce a multi-layer texture with an implicit material representation. Gaussians are organized as a texture, which is compatible with 2D image texture synthesis methods. To align 2D textures with 3D volumes of mesoscale structures, we construct the Gaussian texture with multiple layers for Gaussians with different heights to object surfaces. To achieve

run-time synthesis while preserving the mesoscale structure, we adopt a by-example autocovariance-preserving synthesis method with position conversion and opacity division. Each texel in the new texture is synthesized independently by blending multiple patches, which are selected through importance sampling based on the structure of the example texture.

Since rendering mesoscale structures on arbitrary objects under different lighting is necessary, we introduce a mapping of Gaussian textures to arbitrary objects and a decomposition of materials and light conditions for basic relighting. To reduce artifacts caused by mapping deformation from textures to objects, we map the 3D Gaussians based on their barycentric UV coordinates with a scaling constraint. We represent the decomposed material attributes by latent features, ensuring the consistency of material attributes during synthesis. To regularize the material latent space for better synthesis, we adopt an encoder-decoder structure while training. The encoder is dropped at last, and the decoder is tiny enough for real-time rendering. Eventually, our method achieves real-time synthesis and rendering of mesoscale appearances, with better quality and performance than previous works. In summary, our main contributions include:

- We propose a 3DGS-based framework—multi-layer Gaussian texture to model, synthesize and render the mesoscale appearance from multi-view images.
- We introduce a run-time synthesis method for the Gaussian texture which can preserve structures of mesoscale appearances based on a by-example autocovariance-preserving synthesis.
- We design a deformation-aware mapping approach to apply the Gaussian texture to arbitrary objects, and an implicit material representation for basic relighting that supports robust synthesis with an encoder for regularization.

2 Related work

Meso-structure representation There are many approaches to represent diverse meso-structures, such as bumpy surface, fabric, fur and plant. Displacement mapping [4] is a common method for creating a bumpy surface, which produces parallax and silhouette effects on simple meshes with 2D displacement maps. Furthermore, the shell map [33] and its variants [24] utilize multi-layer textures to better represent the meso-structures. To represent fiber-based materials, such as hair and fur, geometric curves are widely used to model individual fibers [39]. Apart from curves, materials such as cloth can also be stored in memory-heavy volumetric representations [43], which are captured from high-resolution CT scans. Moreover, based on various volumetric primitives, it is possible to model meso-structures such as grass, leaves and fur [5, 18, 30]. More information can be found in the volumetric meso-structure texturing survey by Koniaris et al. [21].

Except for the above classical approaches (displacement maps, curves and voxels), some neural representations have also been proposed recently. Kuznetsov et al. [22] model appearances and meso-structures as latent textures from bidirectional texture functions (BTFs). However, this method is based on BTF, so it is not suitable to represent structures that are difficult to capture as BTF, such

as plants and fur. This is a similar problem preventing TextureDreamer [41] from transferring mesoscale appearances. To represent arbitrary structures, NeRF [29] is a good choice, which models a scene as a radiance field with particles emitting and blocking lights. Inspired by NeRF, NeRF-Tex [2] first investigated the possibility of modeling the texture for meso-structures. This neural texture is trained on synthetic datasets, which is made up of patch rendering results on a plane under known lighting conditions. The trained textures are applied to shapes by placing them repeatedly on surfaces. Later, targeting NeRF texture synthesis from real-world objects, NeRF-Texture [14] simultaneously learns the meso-structures, materials and lighting conditions with textures. Although these neural approaches can represent arbitrary meso-structures, they generally suffer from fuzzy visuals and slow speed. Recent progress in 3DGS [20] provides the possibility of better quality and performance for representation. Focusing on interactive scene editing, Pandey et al. [31] represent real-world objects with 3DGS as brushes and then paint them on new surfaces. While their method can transfer structures onto any surfaces with natural appearance, they require manual painting by artists and do not support relighting.

3D Gaussian Splatting Since our method is based on the 3DGS, we briefly review some related 3DGS-based works, which are mainly about deformation and relighting. The deformation of 3DGS has been explored in various fields [1, 9–12,15,17,40]. Some of these methods (e.g., SuGaR [12], Gaussian Frosting [11], Mesh Gaussian [9], Mani-GS [10]) extract explicit meshes from 3DGS and bind Gaussians to the mesh. Thus, the deformation of the mesh can guide that of the Gaussians. However, these approaches couple meshes with Gaussians, making the application of trained Gaussians to new objects difficult. Some other methods adopt the texture representation of 3DGS for deformation. For example, UV Gaussians [17] learns pose-dependent refined mesh and its Gaussian textures for the animation of human. GSM [1] builds shell maps of the Gaussians to achieve expressive representation of human with hair, clothing and so on. These texture-based methods are similar to our Gaussian texture, but they do not apply the obtained texture to a new object and do not support texture synthesis.

To support relighting with 3DGS, some methods incorporate BRDF into the attributes of Gaussians [8, 16, 25, 36, 37]. Similarly to these works, we represent color based on decomposed BRDF and lighting, and the BRDF attributes are decoded from latent features in a regularized latent space for better synthesis.

Texture synthesis It is a common demand in rendering applications to bind textures to a large surface. Unfortunately, a usual limited-scale texture cannot be applied directly since the texture may not be large enough even though it is 4K. A naive way is to utilize tiling to get a larger texture, but this always leads to undesirable repetitive patterns. Thus, to handle this problem, texture synthesis methods are necessary, which can be mainly divided into two categories.

One common way to perform texture synthesis is based on quilting and optimization [3, 6, 7, 19, 23, 38]. These methods gradually grow the synthesized region

by finding the best-matched candidate patches or texels from the original texture. This matching process is conducted through neighborhood searching or iterative optimization. Since the whole texture should be synthesized region by region before usage, these methods have difficulties in supporting run-time synthesis, so that they are always taken as an offline process. Thus, in this paper, targeting fast speed and on-the-fly synthesis, we do not choose these methods.

In the other aspect, by-example texture synthesis methods [13, 28, 42], which generate a new texture based on an example texture, are able to support synthesis on the fly. Unlike quilting-based synthesis, there is no need to pre-generate the whole texture, as any region of the synthesized texture is not related to other regions. The by-example texture synthesis mainly involves two steps: first select some patches from the example texture, and then blend them with weights together. Although these methods can support run-time synthesis, their quality may not be as high as that of the quilting-based methods.

Notably, despite the advantages and disadvantages of quilting-based and by-example synthesis, our Gaussian texture is actually orthogonal to these methods. Instead of improving current texture synthesis methods, our target is to introduce the synthesis of our Gaussian texture for better practicality. Thus, we mainly focus on by-example texture synthesis methods for run-time synthesis.

3 Method

The goal of our work is capturing mesoscale appearances from multi-view images based on 3DGS with real-time synthesis and rendering on arbitrary objects. To this end, the core of our framework is the multi-layer Gaussian texture representation (Sec. 3.2), where 3D Gaussians are organized as a multi-layer texture. Based on the representation, we further define the operations of our Gaussian texture, including structure-preserving synthesis (Sec. 3.3), deformation-aware mapping (Sec. 3.4), and real-time rendering with decomposed lighting and latent material (Sec. 3.5). Lastly, we present the pipeline (Sec. 3.6) to train our Gaussian texture. Fig. 2 illustrates the overview of our method.

3.1 Preliminary

Our work is mainly based on 3DGS [20] and texture synthesis methods. Since 3DGS is well-known these days, we put its background in the supplement and focus only on texture synthesis methods here.

By-example texture synthesis Among the numerous texture synthesis methods, the by-example texture synthesis is most suitable for our target, which supports dynamic and run-time synthesis. Unlike quilting-based methods, a new area to be synthesized by the by-example method does not depend on the previously synthesized areas, which means that the synthesis can be parallelized for higher speed. Given a target query position x^*, by-example texture synthesis selects I example patches $\mathcal{P}$ from an example texture T according to the position,

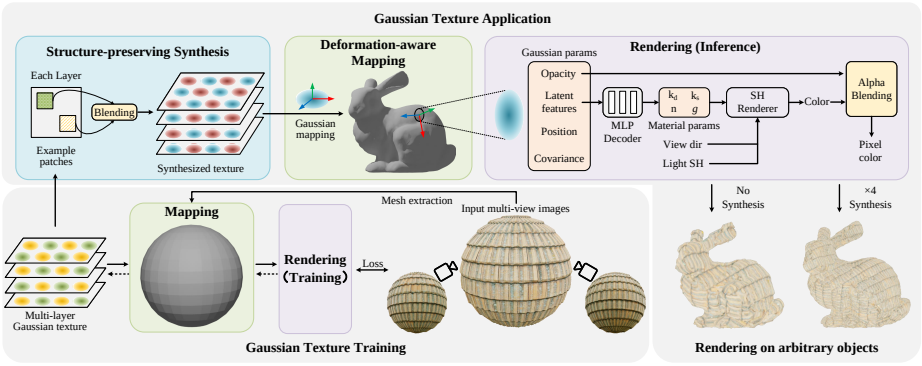

Fig. 2: We present a multi-layer texture of 3D Gaussians which supports synthesis, mapping, and rendering. Each texel of the texture stores parameters of a Gaussian. Through structure-preserving synthesis, Gaussian textures of arbitrary resolutions can be obtained. By a deformation-aware mapping method, the Gaussian texture can be applied to arbitrary objects. The rendering is based on a latent representation of material parameters for consistent synthesis and basic relighting. With the three defined operations, mesoscale appearances can be synthesized and rendered on arbitrary objects, and the Gaussian texture is trained on the mesh extracted from multi-view images.

and then blends them with weights w. The basic process can be formulated as $T^*(x^*) = Syn(T, x*, w)$, where T^* is the synthesized texture, and $Syn()$ denotes a specific synthesis method. In this process, the blending weights follow $\sum w_i = 1$, where w_i is the weight for the ith patch.

During synthesis, some properties can be preserved by modifying the blending weights or patch selection method. Taking a patch $\mathcal{P}^*$ to synthesize, linear blending $\mathcal{P}^* = \sum_{i=1}^{I} w_i \mathcal{P}_i$ keeps the mean unchanged $\mathbb{E}[\mathcal{P}^*] = \mathbb{E}[\mathcal{P}]$, but the variance becomes $\mathbb{V}[\mathcal{P}^*] = W^2\mathbb{V}[\mathcal{P}]$, where $W = \sqrt{\sum_{i=1}^{I} w_i^2}$. To preserve variance, yu et al. [42] propose the blending $\mathcal{P}^* = \frac{\sum_{i=1}^{I} w_i \mathcal{P}_i - \mathbb{E}[\mathcal{P}]}{W} + \mathbb{E}[\mathcal{P}]$. The above methods select patches randomly, which may destroy the structure of the example texture. Thus, lutz et al. [26] propose an importance patch selection method for structure-preserving synthesis, according to autocovariance for each texture translation τ. The autocovariance is calculated as $\int (T(x) - \mathbb{E}[T]) \odot (T(x+\tau) - \mathbb{E}[T]) dx$, where x is a position within the example texture.

3.2 Multi-layer Gaussian texture representation

The 3D Gaussians obtained by the original 3DGS method are aligned with the objects, so they cannot be applied to a new object without unwarping them into a texture. To build a texture of 3D Gaussians with a base mesh, a naive way is to project the Gaussians onto a plane and store them in a texture which supports querying by the UV. However, due to the limited size of a texture, many projected Gaussians may have close UVs and be stored in the same texel.

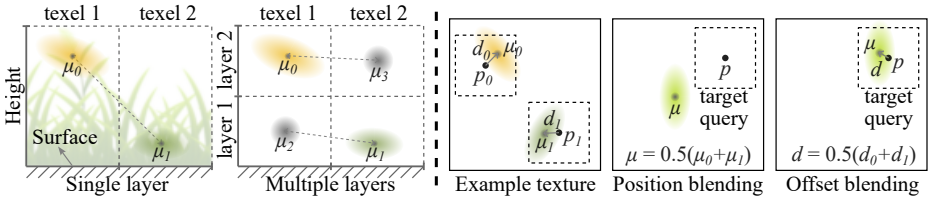

Fig. 3: Left: for a single-layer texture, Gaussians to be synthesized in two texels may have different heights, leading to artifacts or blur results after synthesis. In contrast, multi-layer Gaussian texture can perform synthesis on Gaussians with close heights. The transparent Gaussians are colored as grey. **Right**: directly blending Gaussian positions μ from two selected example patches may not match the target query, but blending offsets d to the centers p of each patch works well.

Moreover, since the Gaussians are not likely to be uniformly distributed after projection onto the mesh, the obtained texture may have different numbers of Gaussians in different texels. To avoid generating these irregular textures that are not suitable for synthesis, we construct a multi-layer texture including only one Gaussian in each texel for a regular representation, which can be mapped to arbitrary objects for training and rendering.

For a single-layer texture, Gaussians within different texels can vary greatly in their heights to the surface, as shown in Fig. 3. Directly blending these Gaussians destroys the original mesoscale structure and causes artifacts. Therefore, we design a multi-layer Gaussian texture, storing only one Gaussian in each texel, and restrict each Gaussian to be close to the texel center. This ensures that Gaussians in the same layer share similar heights. By performing synthesis for each layer of the texture, we avoid synthesis among too different heights to the surface, which preserves the original mesoscale structures.

Thus, we define a multi-layer Gaussian texture as $T^{N\times H\times W\times D}$, which means that the texture has N layers and the size of each layer is $H \times W$. Each texture texel stores D parameters of a 3D Gaussian. The stored parameters are similar to those in 3DGS, including 3D mean position μ, 1D opacity o, 3D scaling s, 4D rotation q and user-defined features to represent colors (48D SH in 3DGS). We decompose the color of 3DGS into materials and lighting SHs to support basic relighting, where the decomposed material is represented by 8D latent features for a meaningful synthesis. Moreover, we temporarily store shading normals only for training. To ensure that one texel corresponds to one Gaussian, the mean position should be in the vicinity of the corresponding texel, which is defined as a cube. The cube center is UV coordinate of the texel center, and its size is $\frac{1}{H} \times \frac{1}{W} \times \frac{L}{N}$, where L is an estimated thickness of the meso-structure.

Based on our multi-layer Gaussian texture, we then introduce operations of our Gaussian texture. To apply our Gaussian texture to arbitrary objects, *synthesis* of the Gaussian texture is essential to generate a large enough texture. Given a Gaussian texture of desired size, we define a *mapping* from the texture

to an object. Since objects may be placed in a scene with new lighting conditions, we should support *rendering* of the Gaussian texture with basic relighting.

3.3 Structure-preserving synthesis

The run-time synthesis of 3D Gaussians is not straightforward due to their explicit natures, and it becomes harder when we want to preserve mesoscale structures. Based on our core multi-layer Gaussian texture representation, we need a method to incorporate by-example texture synthesis discussed in preliminary. The challenge lies in that directly applying the synthesis may cause a mismatch in gaussian positions as shown in Fig. 3, and blending Gaussians with too different opacity may destroy the structure (e.g., blending an opaque Gaussian with a transparent Gaussian generates a translucent Gaussian).

To handle the position blending problem, we convert the mean position μ of each Gaussian to an offset d to its corresponding texel center p: $d = \mu - p$. Then, we blend the offsets and add them by the target query positions, as shown in offset blending of Fig. 3. To reduce opacity difference during synthesis, we modify the blending weights w_i of example patches based on an opacity threshold o_m:

$$w_i' = \begin{cases} \dfrac{w_i(o_i >= o_m)}{sum_w}, sum_w >= 0.5 \\ \dfrac{w_i(o_i < o_m)}{1 - sum_w}, sum_w < 0.5 \end{cases}, \tag{1}$$

where w_i' is the modified blending weight and $sum_w = \sum_{i=1}^{I} w_i(o_i >= o_m)$. By splitting Gaussians into two batches based on the opacity threshold o_m, Gaussians with close opacity are in the same batch, and we only perform blending within the batch occupying a larger proportion of original weights. For example, if $sum_w >= 0.5$, we blend Gaussians with opacity greater than o_m by modifying other weights to zero, and the remaining weights are normalized to ensure that they sum up to 1. In practice, we find that $o_m = 0.1$ is effective in most cases. Moreover, to preserve the structure across layers, we perform syntheses on each Gaussian texture layer with example patches of the same position and size.

Based on above methods, our Gaussian texture can fit by-example texture synthesis well. In practice, we adopt variance-preserving method [42] to improve quality, with autocovariance-preserving synthesis [26] to further preserve structure. More specifically, we select patches that better preserve the exemplar structure on the fly based on the autocovariance, which is computed from Gaussian texture opacity and latent features, representing the texture structural correlations and indicating which local patterns should be preserved during synthesis.

3.4 Deformation-aware mapping

Since we need to map our Gaussian texture to arbitrary meshes, this may cause a large scale deformation. To avoid artifacts arising from the deformation, our mapping is performed according to barycentric UV coordinates. Additionally, we

apply a scale constraint on Gaussian scales based on the texel size. This can reduce the number of Gaussians with irrational shapes and thus avoid deformation artifacts, which will be detailed in Sec. 3.6.

Given a mesh with a UV map, each triangle of the mesh corresponds to a triangle in the UV space. For one triangle in the mesh, we have a face normal n^w, three vertices v_1, v_2, v_3, and three corresponding UV coordinates t_1, t_2, t_3. Taking $n^t = (0, 0, 1)$, we define the rotation R^t and scaling S^t mapping from UV triangle to mesh triangle as:

$$\begin{aligned} R^t =& \left[\frac{v_2 - v_1}{\|v_2 - v_1\|}, \frac{v_2 - v_1}{\|v_2 - v_1\|} \times n^w, n^w \right] \left[\frac{t_2 - t_1}{\|t_2 - t_1\|}, \frac{t_2 - t_1}{\|t_2 - t_1\|} \times n^t, n^t \right]^\top, \\ S^t =& \frac{\|v_2 - v_1\| + \|v_3 - v_2\| + \|v_1 - v_3\|}{\|t_2 - t_1\| + \|t_3 - t_2\| + \|t_1 - t_3\|}. \end{aligned} \tag{2}$$

To find Gaussians belonging to a given UV area for mapping, we check whether the projection of its mean position to the UV plane μ_{xy} is within the area. For the Gaussians in this area, we compute their barycentric coordinates b_1, b_2, b_3 by $\mu_{xy} = b_1 t_1 + b_2 t_2 + b_3 t_3$, since μ_{xy}, t_1, t_2, t_3 are known. Thus, we can build the mapping of a Gaussian from its texture to a mesh as:

$$\mu' = b_1 v_1 + b_2 v_2 + b_3 v_3 + S^t \mu_z n^w, \quad R' = R^t R, \quad s' = S^t s, \tag{3}$$

where μ_z is the z component of μ, R is the rotation matrix computed by the quaternion q, and μ', R', s' are the mean position, rotation matrix and scaling factor of the mapped Gaussian. Besides these parameters mentioned above, the direction-related parameters such as normal and SH should be changed by applying the rotation matrix. Other properties remain the same after mapping.

3.5 Rendering with relighting

To support different lightings in different scenes, we decompose materials and lighting conditions for basic relighting. We use SHs to represent illumination and replace original color SHs of 3DGS with some material parameters. In this paper, we adopt the Phong shading model [32] for rendering, but other shading models are also supported. Under this condition, we model the reflection through four attributes: 3D diffuse color k_d, 1D specular intensity k_s, 1D glossiness g and 3D shading normal n. These material parameters are decoded through an MLP from latent features, which better suit the synthesis as they naturally preserve correlations among material parameters. Following the approach proposed by Ramamoorthi and Hanrahan [34], the view-dependent color c is computed through SH convolution given a view direction. Taking this color into the computation of pixel color in 3DGS, we can get the final rendering results. In this way, we achieve rendering with basic relighting.

3.6 Multi-stage pipeline

We introduce the pipeline of our Gaussian texture, targeting 3D Gaussian training with a compact latent feature space to support synthesis. Our pipeline in-

cludes mapping texture to the extracted mesh from multi-view images and rendering it to calculate loss for back propagation. The mapping is the same as the method in Sec. 3.4, and the mesh extraction is detailed in the supplement.

For more robust training and a compact latent space, our training process has three stages in total. The first stage mainly consists of training the non-material parameters, of which colors are represented by SH to exclude the impact of latent representation. Concurrently, we train shading normals supervised by the pseudo normals computed from depth. The second and third stages introduce training of light SHs and material latent features. In the second stage, we make use of an encoder-decoder structure to regularize the latent space. The input is color SHs and shading normals, and its output is material latent features. In the third stage, we drop the encoder and only train latent features and decoder. A detailed illustration of our pipeline can be found in the supplementary material.

We define different losses for the three training stages. Because the Gaussian texture only stores one Gaussian in each texel, we define a position restriction loss to ensure this relation:

$$\mathcal{L}_{\mu} = max\{(\mu - p) - 0.5 \times (\frac{1}{H}, \frac{1}{W}, \frac{L}{N}), 0\}. \tag{4}$$

Considering the deformation from mapping, a scaling constraint loss is necessary to reduce artifacts by regularizing Gaussian shapes as in Sec. 3.4, which is defined as:

$$\mathcal{L}_{s} = max\{s - \frac{3}{max\{H, W, N\}}, 0\}. \tag{5}$$

In addition to these texture losses, the loss function of the first stage includes a color loss $\mathcal{L}_c$, a depth distribution loss $\mathcal{L}_d$, and a normal loss $\mathcal{L}_{n'}$. The color loss is a L_1 loss between the renderings and the ground truths (GTs), the depth distribution loss drives Gaussian points to the object surface, and the normal loss encourages consistency between the trained normals n' and the normals computed from depth. The definitions of depth distribution loss and normal loss are the same as in Relightable 3D Gaussians [8], which are defined as:

$$\mathcal{L}_{d} = \mathcal{D}_{sq} - \mathcal{D}^2, \quad \mathcal{L}_{n'} = \|\mathcal{N}' - \tilde{\mathcal{N}}\|_2, \tag{6}$$

where $\mathcal{D}_{sq}$, $\mathcal{D}$ are square depth and original depth maps respectively, and $\mathcal{N}'$, $\tilde{\mathcal{N}}$ are trained normal and pseudo normal maps respectively. The complete loss of the first stage is

$$\mathcal{L} = \mathcal{L}_c + \mathcal{L}_{\mu} + \mathcal{L}_s + \lambda_d \mathcal{L}_d + \lambda_n \mathcal{L}_{n'}, \tag{7}$$

where λ_d and λ_n are weights set to 0.01 and 0.1, respectively. For the second and third stages, the loss function is:

$$\mathcal{L} = \mathcal{L}_c + \mathcal{L}_{\mu} + \mathcal{L}_s + \lambda_d \mathcal{L}_d + \lambda_n \mathcal{L}_n + \lambda_{k_d} \mathcal{L}_{k_d} + \lambda_{k_s} \mathcal{L}_{k_s}, \tag{8}$$

where $\mathcal{L}_c$ and $\mathcal{L}_d$ are inherited from the first stage. $\mathcal{L}_n$ is a loss similar to $\mathcal{L}_{n'}$ between normals predicted by decoder and computed from depth. $\mathcal{L}_{k_d}, \mathcal{L}_{k_s}$ are

also constraints from Relightable 3D Gaussian on diffuse color and specular intensity to make them smooth, following the form:

$$\mathcal{L}_k = \|\nabla K\| e^{-\|\nabla K^{GT}\|}, \tag{9}$$

where K indicates the rendered k_d or k_s map. In practice, we set $\lambda_d = 0.01, \lambda_n = 0.1, \lambda_{k_d} = \lambda_{k_s} = 0.01$.

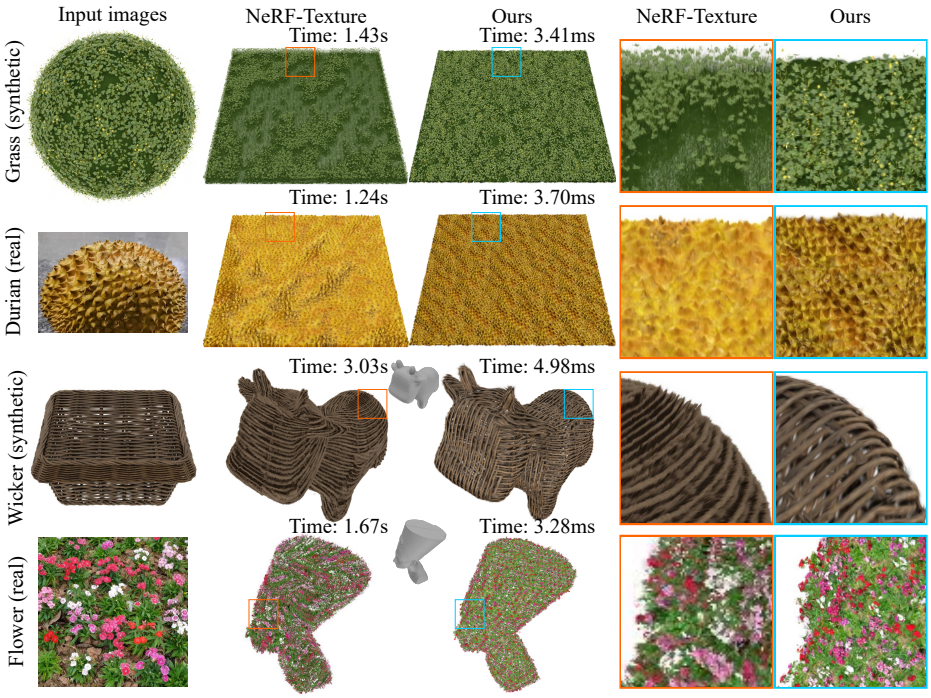

Fig. 4: The comparison of NeRF-Texture and our method on mesoscale appearance synthesis and rendering. Both far and close views show that our method preserves details and structures lost in NeRF-Texture, and achieves about 500× acceleration.

4 Results

In this section, we evaluate our method using some experiments on an NVIDIA RTX 4090. We first compare our results on the synthesis and rendering of mesoscale appearances with NeRF-Texture [14] (Sec. 4.1). Then we conduct some ablation studies to prove the effectiveness of our design (Sec. 4.2). In the end, we discuss the limitations of our work (Sec. 4.3).

Fig. 5: Our relighting results. Our method supports basic relighting thanks to the decomposition of materials and lighting. The input images and used lighting are shown.

Table 1: Quantitative comparison of our method and NeRF-Texture. Our method outperforms NeRF-Texture in terms of SIFID and memory for four examples in Fig. 4.

SIFID ↓ / Memory ↓	Grass	Durian	Wicker	Flower
Ours	**10.86** **280 MB**	**4.82** **140 MB**	**4.38** **140 MB**	**5.26** **420 MB**
NeRF-Texture	15.94 1.1 GB	10.71 1.1 GB	6.89 1.1 GB	6.96 855.7 MB

4.1 Mesoscale appearance synthesis and rendering

In Fig. 4, we compare our method with NeRF-Texture on both synthetic and real data, and our training settings are shown in the supplement. Our method only takes about several microseconds, while NeRF-Texture takes up to seconds. The first two rows show synthesis on a plane, which are synthetic images of grass with flowers and real images of a durian. It can be found that our method preserves the structure better during synthesis, such as the gap density inside the grass and spikes of durian. Moreover, NeRF-Texture has trouble in representing the details of the scene, like the little yellow flowers among the grass. In contrast, our approach can capture these flowers, achieving a higher similarity to the original mesoscale appearance and better visual quality. For the last two input multi-view images, we show some results on complex meshes. our method synthesizes the gaps among the wickers, and produces more obvious flowers on the new objects. Our method has slight differences in color with NeRF-Texture, which may come from different lighting learned from the scene during training. Following NeRF-Texture, we also provide a quantitative comparison based on SIFID [35] in Table 1. We show more synthesis and rendering results in Fig. 6 and relighting results under a novel lighting in Fig. 5, presenting the capability to preserve highly-detailed structures and support basic relighting. We also provide more relighting and numerical results in the supplement.

4.2 Ablation study

We validate some key components of our method here. To reduce artifacts caused by deformation from mapping, we have a scaling constraint on Gaussians. We

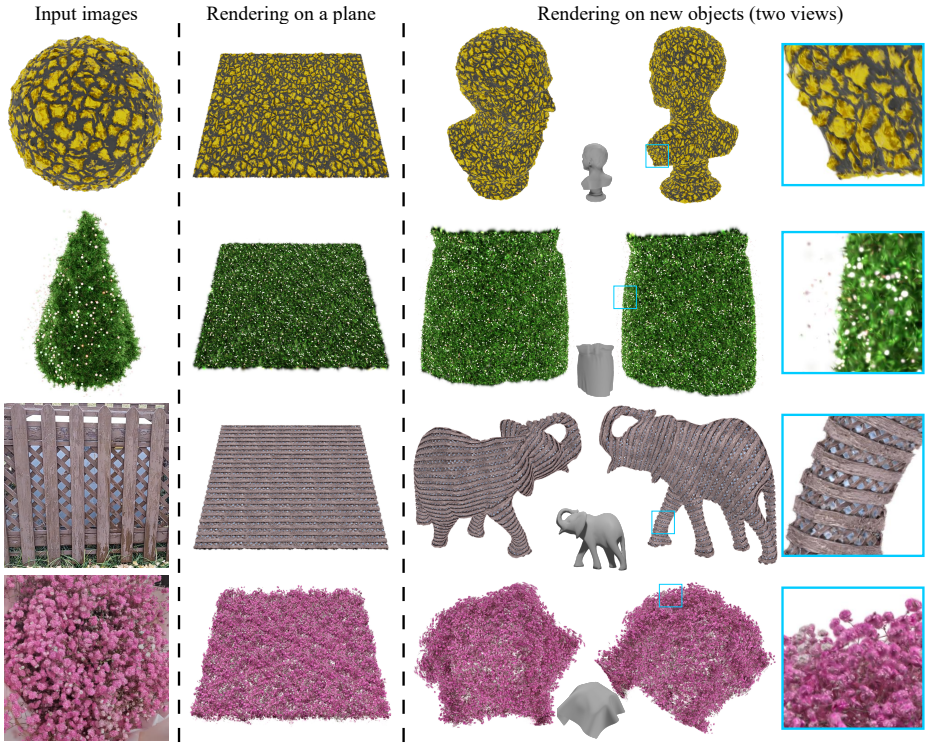

Fig. 6: More synthesis and rendering results of our method. Our method can represent the details of the mesoscale appearance and preserve the original structure.

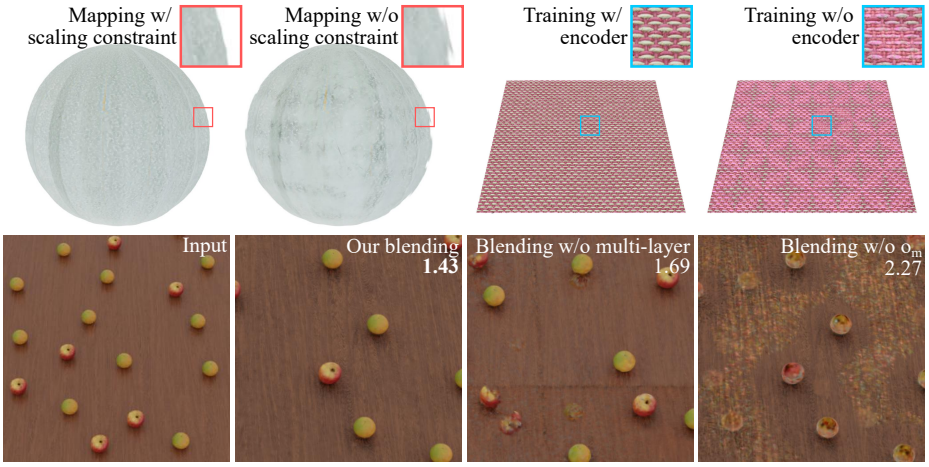

Fig. 7: Ablation of the scaling constraint loss to reduce artifacts during mapping, and encoder in training, multi-layer texture, opacity-aware blending for better synthesis. With these components, our method produces higher-quality results. SIFID values are also provided in the figure for quantitative comparison.

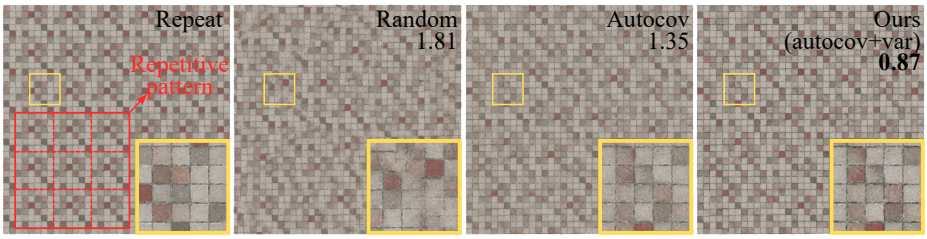

Fig. 8: Comparison of different synthesis methods. By combining variance-preserving and autovariance-preserving synthesis together, our method can obtain a result with similar structure to the input but no repetitive patterns or artifacts. SIFID values are also provided in the figure for quantitative comparison.

show its impact in the top left of Fig. 7. In comparison, the result from the mapping without a scaling constraint loss produces many needle-like artifacts, which are not found with the loss. This proves the necessity of scaling limitation. During training, we additionally use an encoder to regularize the latent space for better synthesis. The impact of the encoder is validated in the top right row of Fig. 7, by comparing the rendering results of synthesized textures on a plane after training with and without encoder. We found that the result from Gaussian texture trained without encoder includes many crossing artifacts, which means the encoder can improve the synthesis quality. In the bottom row of Fig. 7, we validate the impact of multi-layer texture and opacity-aware blending, and the results have been cropped for better clarity. Our multi-layer texture avoids blending Gaussians with different heights (fruits and table), which would otherwise bake the fruit color into the synthesized table, and our opacity-aware blending (by o_m) prevents colorful artifacts from synthesized semi-transparent Gaussians. The SIFID values are provided in the figure for quantitative comparison.

We also show the influence of synthesis strategies in Fig. 8 with SIFID values. Repeating without synthesis leads to repetitive patterns, while random synthesis disrupts brick's structure. By taking autocovariance-preserving synthesis, the result shows a better structure. By incorporating variance-preserving synthesis, we obtain results with more similar color to the non-synthesized result (repeat).

4.3 Discussion and limitations

Although our method can model mesoscale appearances from multi-view images and synthesize and render it on arbitrary objects in real time, there are still some limitations. One problem is handling objects with complex geometry. Our method is to represent mesoscale appearance based on mesh extraction of multi-view images, so an object with complex geometry but no apparent mesoscale appearance is difficult and not our target, which may cause a failure as shown in Fig. 9. To handle this problem, a method to extract a detailed mesh with simple UV coordinates may be necessary.

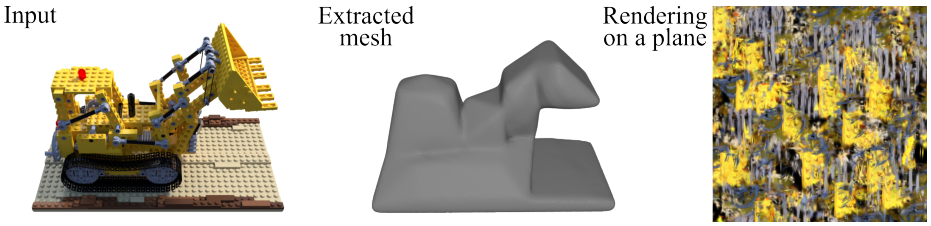

Fig. 9: A failure case of applying our method to an object with complex geometry but no apparent mesoscale structure, which is actually not our target.

Another issue we face is the training time of our method. Because the Gaussian parameters are constantly changing during training, we must recompute the barycentric UV coordinates in each iteration. This results in additional training time, which can range from 0.5 to 3 hours, depending on the texture resolution and the complexity of the scene.

5 Conclusion and future work

In this paper, we present a 3DGS-based framework to model, synthesize and render the mesoscale appearance. The key is to build a multi-layer Gaussian texture representation of 3DGS, which makes explicit 3D Gaussians compatible with 2D image texture synthesis methods. Real-time synthesis with preserved structures is achieved through a specialized by-example autocovariance-preserving synthesis. We further introduce the deformation-aware mapping and material-lighting decomposition to render Gaussian textures on arbitrary objects under different lightings. The mapping is based on barycentric UV coordinates with a scaling limitation to reduce artifacts caused by deformation. The decomposed material, which is used for basic relighting, is represented by latent features with a multi-stage training process to better support synthesis. We have demonstrated that our method can represent mesoscale appearances with details, and synthesize and render them on arbitrary meshes in real time.

In the future, there are still many potential research directions. Our work currently only supports basic relighting with lighting represented by SH. Supporting a complete relighting that allows for more complicated lighting conditions is important to represent the appearance more realistically. Moreover, it is also interesting to integrate our method into the traditional rendering pipeline for real-time rendering of complex mesoscale appearances.

Acknowledgements

We thank the reviewers for the valuable comments. This work has been partially supported by the National Natural Science Foundation of China under grant No. 62272275, No. 62502285 and No. 62572230.

References

1. Abdal, R., Yifan, W., Shi, Z., Xu, Y., Po, R., Kuang, Z., Chen, Q., Yeung, D.Y., Wetzstein, G.: Gaussian shell maps for efficient 3d human generation. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 9441–9451 (2024). https://doi.org/10.1109/CVPR52733.2024.00902
2. Baatz, H., Granskog, J., Papas, M., Rousselle, F., Novák, J.: Nerf-tex: Neural reflectance field textures. Computer Graphics Forum **41**(6), 287–301 (March 2022). https://doi.org/https://doi.org/10.1111/cgf.14449
3. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: Patchmatch: a randomized correspondence algorithm for structural image editing. ACM Trans. Graph. **28**(3) (Jul 2009). https://doi.org/10.1145/1531326.1531330, https://doi.org/10.1145/1531326.1531330
4. Cook, R.L.: Shade trees. SIGGRAPH Comput. Graph. **18**(3), 223–231 (Jan 1984). https://doi.org/10.1145/964965.808602, https://doi.org/10.1145/964965.808602
5. Decaudin, P., Neyret, F.: Volumetric billboards. Computer Graphics Forum **28**(8), 2079–2089 (2009). https://doi.org/https://doi.org/10.1111/j.1467-8659.2009.01354.x, https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01354.x
6. Efros, A., Leung, T.: Texture synthesis by non-parametric sampling. In: Proceedings of the Seventh IEEE International Conference on Computer Vision. vol. 2, pp. 1033–1038 vol.2 (1999). https://doi.org/10.1109/ICCV.1999.790383
7. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. p. 341–346. SIGGRAPH '01, Association for Computing Machinery, New York, NY, USA (2001). https://doi.org/10.1145/383259.383296, https://doi.org/10.1145/383259.383296
8. Gao, J., Gu, C., Lin, Y., Li, Z., Zhu, H., Cao, X., Zhang, L., Yao, Y.: Relightable 3d gaussians: Realistic point cloud relighting with brdf decomposition and ray tracing. In: Leonardis, A., Ricci, E., Roth, S., Russakovsky, O., Sattler, T., Varol, G. (eds.) Computer Vision – ECCV 2024. pp. 73–89. Springer Nature Switzerland, Cham (2025)
9. Gao, L., Yang, J., Zhang, B., Sun, J., Yuan, Y., Fu, H., Lai, Y.K.: Real-time large-scale deformation of gaussian splatting. ACM Transactions on Graphics (SIGGRAPH Asia 2024) (2024)
10. Gao, X., Li, X., Zhuang, Y., Zhang, Q., Hu, W., Zhang, C., Yao, Y., Shan, Y., Quan, L.: Mani-gs: Gaussian splatting manipulation with triangular mesh. arXiv preprint arXiv:2405.17811 (2024)
11. Guédon, A., Lepetit, V.: Gaussian frosting: Editable complex radiance fields with real-time rendering. ECCV (2024)
12. Guédon, A., Lepetit, V.: Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5354–5363 (2024). https://doi.org/10.1109/CVPR52733.2024.00512
13. Heitz, E., Neyret, F.: High-performance by-example noise using a histogram-preserving blending operator. Proc. ACM Comput. Graph. Interact. Tech. **1**(2) (Aug 2018). https://doi.org/10.1145/3233304, https://doi.org/10.1145/3233304

14. Huang, Y.H., Cao, Y.P., Lai, Y.K., Shan, Y., Gao, L.: Nerf-texture: Texture synthesis with neural radiance fields. In: ACM SIGGRAPH 2023 Conference Proceedings. pp. 1–10 (2023)
15. Huang, Y.H., Sun, Y.T., Yang, Z., Lyu, X., Cao, Y.P., Qi, X.: Sc-gs: Sparse-controlled gaussian splatting for editable dynamic scenes. arXiv preprint arXiv:2312.14937 (2023)
16. Jiang, Y., Tu, J., Liu, Y., Gao, X., Long, X., Wang, W., Ma, Y.: Gaussianshader: 3d gaussian splatting with shading functions for reflective surfaces. arXiv preprint arXiv:2311.17977 (2023)
17. Jiang, Y., Liao, Q., Li, X., Ma, L., Zhang, Q., Zhang, C., Lu, Z., Shan, Y.: Uv gaussians: Joint learning of mesh deformation and gaussian textures for human avatar modeling. arXiv preprint arXiv:2403.11589 (2024)
18. Kajiya, J.T., Kay, T.L.: Rendering fur with three dimensional textures. SIGGRAPH Comput. Graph. **23**(3), 271–280 (Jul 1989). https://doi.org/10.1145/74334.74361, https://doi.org/10.1145/74334.74361
19. Kaspar, A., Neubert, B., Lischinski, D., Pauly, M., Kopf, J.: Self tuning texture optimization. Computer Graphics Forum **34**(2), 349–359 (2015). https://doi.org/https://doi.org/10.1111/cgf.12565, https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12565
20. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. ACM Transactions on Graphics **42**(4) (July 2023), https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/
21. Koniaris, C., Cosker, D., Yang, X., Mitchell, K.: Texture mapping techniques for volumetric mesostructure. Journal of Computer Graphics Techniques (JCGT) **3**(1), 18–59 (February 2014), http://jcgt.org/published/0003/01/02/
22. Kuznetsov, A., Wang, X., Mullia, K., Luan, F., Xu, Z., Hašan, M., Ramamoorthi, R.: Rendering neural materials on curved surfaces. SIGGRAPH '22 Conference Proceedings (2022). https://doi.org/10.1145/3528233.3530721
23. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. ACM Trans. Graph. **24**(3), 795–802 (Jul 2005). https://doi.org/10.1145/1073204.1073263, https://doi.org/10.1145/1073204.1073263
24. Lengyel, J., Praun, E., Finkelstein, A., Hoppe, H.: Real-time fur over arbitrary surfaces. In: Proceedings of the 2001 Symposium on Interactive 3D Graphics. p. 227–232. I3D '01, Association for Computing Machinery, New York, NY, USA (2001). https://doi.org/10.1145/364338.364407, https://doi.org/10.1145/364338.364407
25. Liang, Z., Zhang, Q., Feng, Y., Shan, Y., Jia, K.: Gs-ir: 3d gaussian splatting for inverse rendering. In: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 21644–21653 (2024). https://doi.org/10.1109/CVPR52733.2024.02045
26. Lutz, N., Sauvage, B., Dischler, J.M.: Preserving the autocovariance of texture tilings using importance sampling. Computer Graphics Forum **42**(2), 347–358 (2023). https://doi.org/https://doi.org/10.1111/cgf.14766, https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14766
27. Marschner, S.R., Jensen, H.W., Cammarano, M., Worley, S., Hanrahan, P.: Light scattering from human hair fibers. ACM Trans. Graph. **22**(3), 780–791 (Jul 2003). https://doi.org/10.1145/882262.882345, https://doi.org/10.1145/882262.882345
28. Mikkelsen, M.S.: Practical real-time hex-tiling. Journal of Computer Graphics Techniques (JCGT) **11**(3), 77–94 (August 2022), http://jcgt.org/published/0011/03/05/

29. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: ECCV (2020)
30. Neyret, F.: Modeling, animating, and rendering complex scenes using volumetric textures. IEEE Transactions on Visualization and Computer Graphics **4**(1), 55–70 (1998). https://doi.org/10.1109/2945.675652
31. Pandey, K., Hu, A., Tsang, C.F., Perel, O., Singh, K., Shugrina, M.: Painting with 3d gaussian splat brushes. In: Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers. pp. 1–10 (2025)
32. Phong, B.T.: Illumination for computer generated pictures. Commun. ACM **18**(6), 311–317 (Jun 1975). https://doi.org/10.1145/360825.360839, https://doi.org/10.1145/360825.360839
33. Porumbescu, S.D., Budge, B., Feng, L., Joy, K.I.: Shell maps. In: ACM SIGGRAPH 2005 Papers. p. 626–633. SIGGRAPH '05, Association for Computing Machinery, New York, NY, USA (2005). https://doi.org/10.1145/1186822.1073239, https://doi.org/10.1145/1186822.1073239
34. Ramamoorthi, R., Hanrahan, P.: A signal-processing framework for inverse rendering. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. p. 117–128. SIGGRAPH '01, Association for Computing Machinery, New York, NY, USA (2001). https://doi.org/10.1145/383259.383271, https://doi.org/10.1145/383259.383271
35. Rott Shaham, T., Dekel, T., Michaeli, T.: Singan: Learning a generative model from a single natural image. In: Computer Vision (ICCV), IEEE International Conference on (2019)
36. Shi, Y., Wu, Y., Wu, C., Liu, X., Zhao, C., Feng, H., Zhang, J., Zhou, B., Ding, E., Wang, J.: Gir: 3d gaussian inverse rendering for relightable scene factorization. IEEE Transactions on Transactions on Pattern Analysis and Machine Intelligence (2025)
37. Sun, H., Gao, Y., Xie, J., Yang, J., Wang, B.: Svg-ir: Spatially-varying gaussian splatting for inverse rendering. Proceedings of CVPR 2025 (2025)
38. Wei, L.Y., Levoy, M.: Texture synthesis over arbitrary manifold surfaces. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. pp. 355–360 (2001)
39. Wu, K., Yuksel, C.: Real-time fiber-level cloth rendering. In: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D 2017). ACM, New York, NY, USA (2017). https://doi.org/10.1145/3023368.3023372, http://doi.acm.org/10.1145/3023368.3023372
40. Xie, T., Zong, Z., Qiu, Y., Li, X., Feng, Y., Yang, Y., Jiang, C.: Physgaussian: Physics-integrated 3d gaussians for generative dynamics. arXiv preprint arXiv:2311.12198 (2023)
41. Yeh, Y.Y., Huang, J.B., Kim, C., Xiao, L., Nguyen-Phuoc, T., Khan, N., Zhang, C., Chandraker, M., Marshall, C.S., Dong, Z., et al.: Texturedreamer: Image-guided texture synthesis through geometry-aware diffusion. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4304–4314 (2024)
42. Yu, Q., Neyret, F., Bruneton, E., Holzschuch, N.: Lagrangian texture advection: Preserving both spectrum and velocity field. IEEE Transactions on Visualization and Computer Graphics **17**(11), 1612–1623 (2011). https://doi.org/10.1109/TVCG.2010.263

43. Zhao, S., Jakob, W., Marschner, S., Bala, K.: Building volumetric appearance models of fabric using micro ct imaging. ACM Trans. Graph. **30**(4) (Jul 2011). https://doi.org/10.1145/2010324.1964939, https://doi.org/10.1145/2010324.1964939