MesoGAN: Generative Neural Reflectance Shells

S Diolatzis, J Novák, F Rousselle, J Granskog, M Aittala, R Ramamoorthi, George Drettakis

MesoGAN: Generative Neural Reflectance Shells

S. Diolatzis[1,2,3] , J. Novák[3] , F. Rousselle[3] , J. Granskog[4] , M. Aittala[3] , R. Ramamoorthi[3,5] , G. Drettakis[1,2]

stavros.diolatzis@inria.fr   jnovak@nvidia.com   frousselle@nvidia.com   jonathan@runwayml.com   maittala@nvidia.com   ravir@cs.ucsd.edu   george.drettakis@inria.fr

[1]INRIA [2] Université Côte d'Azur [3]NVIDIA [4]Runway [5]University of California San Diego
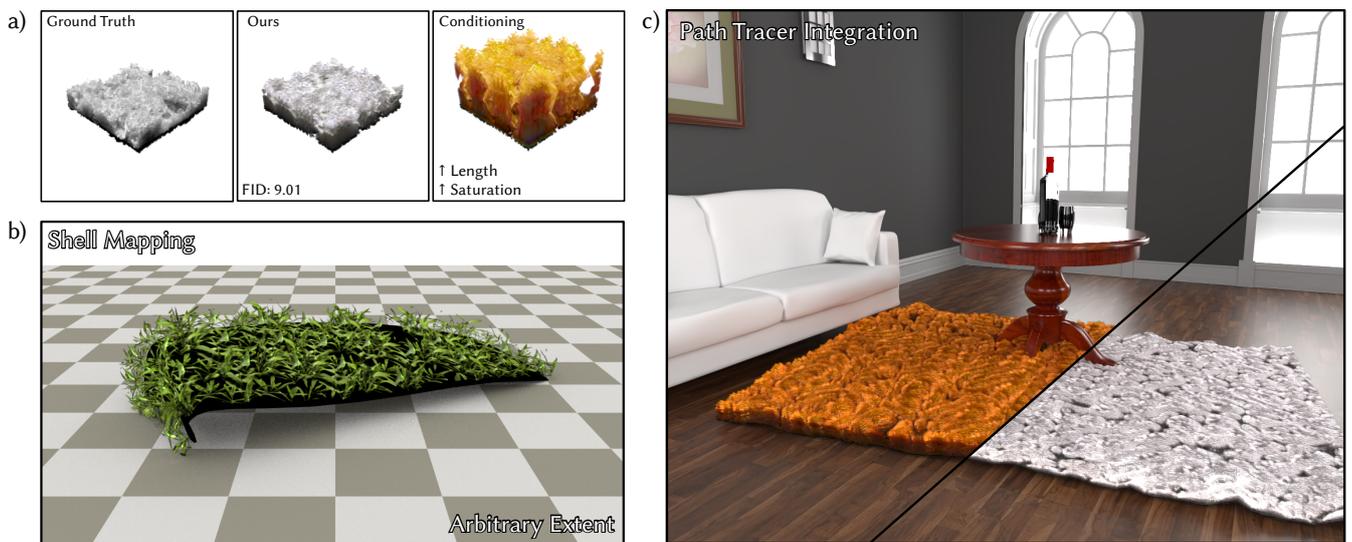
Figure 1: *Our method combines the strengths of StyleGAN and volumetric neural field rendering to generate a 3D mesoscale texture that can be mapped to objects and used in a path tracer (c). We train on datasets of synthetic patches (a); our method can generate textures that have artistic parameters (such as fur saturation and length) which can be used to create shell maps of arbitrary extent (b).*

Abstract
We introduce MesoGAN, a model for generative 3D neural textures. This new graphics primitive represents mesoscale appearance by combining the strengths of generative adversarial networks (StyleGAN) and volumetric neural field rendering. The primitive can be applied to surfaces as a neural reflectance shell; a thin volumetric layer above the surface with appearance parameters defined by a neural network. To construct the neural shell, we first generate a 2D feature texture using StyleGAN with carefully randomized Fourier features to support arbitrarily sized textures without repeating artifacts. We augment the 2D feature texture with a learned height feature, which aids the neural field renderer in producing volumetric parameters from the 2D texture. To facilitate filtering, and to enable end-to-end training within memory constraints of current hardware, we utilize a hierarchical texturing approach and train our model on multi-scale synthetic datasets of 3D mesoscale structures. We propose one possible approach for conditioning MesoGAN on artistic parameters (e.g., fiber length, density of strands, lighting direction) and demonstrate and discuss integration into physically based renderers.

Keywords: generative adversarial networks, ray tracing, shell mapping

CCS Concepts
• **Computing methodologies** → **Reflectance modeling; Neural networks; Ray tracing;**

1. Introduction

We tackle the problem of modeling and rendering complex mesoscale structures, such as grass, fur, or fabrics. We target a method that can be directly integrated into a path tracer, provides a compact representation that allows artistic control over a set of parameters, and that can be applied to objects as a non-repetitive 3D neural texture with arbitrary extent. Our solution leverages the combined power of volumetric neural fields and generative adversarial networks (GANs) to achieve these goals, as shown in Fig. 1.

The inherent volumetric nature of mesoscale structures [KK89, Ney98] makes neural fields [MST*20, XTS*22] a natural choice for their representation. Unfortunately, neural fields typically have restricted spatial extent, resulting in repetition artifacts and difficulty handling anisotropy when applied as a texture [BGP*21]. In addition, visual quality often degrades when they are conditioned with several variable parameters [MBRS*21, BGP*21]. GANs have shown great promise in rendering high-quality realistic images, e.g., StyleGAN and derivatives [KLA*20, KAL*21]. However they typically require convolutions in screen-space, which precludes their use as material primitives in light transport simulators such as path tracing, due to their effect on multi-view consistency and the inability to perform local operations when tracing independent rays for multiple bounces.

Our method uses StyleGAN3 to generate features that are then interpreted as a volumetric neural field. Such solutions have shown promising results for 3D synthesis, demonstrated on specific datasets such as faces [CMK*20, CLC*22]. However, end-to-end training of neural fields in conjunction with GANs is very memory intensive, severely limiting the resolution of the volumetrically rendered images.

None of the previous methods simultaneously overcomes the limitations mentioned above, i.e., avoiding screen-space convolutions to enable path tracer integration, generating texture of arbitrary extent and resolution and artistic control with several parameters. We address these shortcomings by introducing three contributions:

- We show that, for 3D mesoscale textures without large-scale structure, we can create 3D textures without repetition artifacts with arbitrary extent by introducing randomized Fourier features in the layers of StyleGAN3 [KAL*21] (Sec. 4.1).
- We introduce *neural shell maps* that take the volumetric texture generated by our generator and use the neural field to apply the learned mesoscale appearances to surfaces (Sec. 4.2). Our method allows artistic control by conditioning geometry and appearance parameters in the StyleGAN mapping network and the neural field multi-layer perceptron (MLP) decoder respectively (Sec. 4.3).
- To overcome memory limitations of the end-to-end pipeline and to reduce aliasing, we utilize filtering together with progressive multi-scale training (Sec. 5.1). This allows our combined StyleGAN3/neural field architecture to generate high-quality 3D textures of mesoscale materials at different scales.

Our results show that our approach successfully exploits the combined power of GANs and neural fields, allowing the generation of path-traced images with 3D mesoscale textures with artistic control such as length of fur, roughness, lighting direction, etc.

2. Related work

In this section, we relate our approach to previous work in the fields of material models and deep learning.

Conventional models of mesoscale appearance. Mesoscale materials are diverse and complex, with each class of mesostructures often being modeled using a tailored approach. For example, curves are typically used for fur and hair [MJC*03], alpha-masked polygonal meshes are commonly used for vegetation [WM19], whereas spherical proxies are used to approximate grains in large granular assemblies [MPG*16]. In some cases, volumetric primitives are employed to represent leaves and fur [KK89, DN09, Ney98]. Each representation has specific weaknesses, e.g., poor filtering (curves and meshes), high memory consumption, or limited artistic control (volumetric representations). The lack of a single versatile mesoscale primitive to handle all these cases robustly is apparent; MesoGAN addresses these shortcomings.

Neural material models. Recent years have seen the emergence of neural bidirectional texture function (BTF) models [DvGNK99] suitable for representing mesoscale structures [RJGW19, RGJW20]. Such neural models can be extended to perform filtered queries for robust level-of-detail rendering [KMX*21] and to take curvature into account to achieve accurate silhouettes [KWM*22]. However, BTFs still operate strictly in texture space and do not consider the spatial extent of these materials. Instead, we opt to explicitly model the mesoscale structure as a volume, which allows us to blend them with other traditional meshes in our renderings (Figure 1).

Neural Fields. Neural networks have proven to be remarkably efficient at encoding scene information, such as radiometric quantities [RWG*13, MRKN20, HCZ21], signed distance fields [SCT*20], and textures [OMN*19, HMR20a]. Neural radiance fields (NeRF), which are of particular relevance to our work, have found adoption in many applications [TTM*21, XTS*22]. In the original NeRF paper, Mildenhall et al. [MST*20] propose to represent volumetric radiance and volumetric density using an MLP. The MLP-inferred values are optimized by numerically estimating incident radiance on the image plane and minimizing the loss with respect to a ground truth image (typically a photo). A suitable encoding of the MLP input, such as Fourier features [TSM*20] or an integrated encoding [BMT*21], can significantly increase the reconstruction quality.

Baatz et al. [BGP*21] leverage the concept of neural fields for representing mesoscale structures. The NeRF texture is optimized to represent *reflectance* of a mesostructure slab, which permits instantiating (and relighting) small patches of the mesostructure on an arbitrary surface. Artistic control of the mesostructure appearance is paramount for future adoption in production. Hence the authors propose to condition the MLP on a set of parameters to facilitate different looks (e.g., straight vs. curly fur). Unfortunately, increasing the number of conditioning parameters leads to a drop in reconstruction quality. The limited degree of visual diversity and complete lack of stochastic detail in NeRF textures are severe drawbacks, which we solve using a generative approach that features rich, controllable variations.

Generative approaches. We learn generative textures, and our approach is related to a number of recent efforts on generative NeRF models [SLNG20, CMK*20, GLWT21, NG21] and 3D
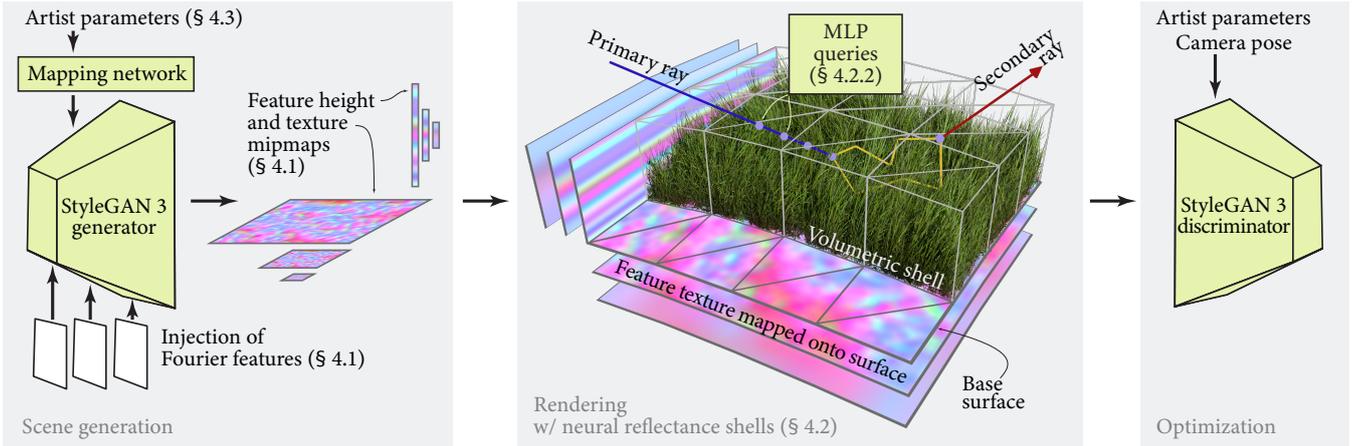
Figure 2: *Overview: We use the StyleGAN3 generator, which we extend by injection of phase-randomized Fourier features, to produce a hierarchical feature texture. The texture, when mapped on a surface, conditions an MLP that infers density and reflectance values of the mesostructure in a volumetric shell above the surface. Given a point (e.g., on a primary ray) inside the shell, the inferred reflectance values can be used to evaluate transport (illustrated as yellow paths) between this point and another point on the* top *boundary of the shell.*

texture synthesis [HMR20b, PBG20]. We draw inspiration from EG3D [CLC*22], which employs a two-stage process where a Style-GAN2 generator [KLA*20] drives a subsequent NeRF model using three orthogonal 2D textures. The generation of details and structures are done in 2D, then lifted into 3D using an MLP to provide multi-view consistency. We introduce randomized Fourier features to our StyleGAN generator to achieve *3D textures with arbitrary extent* that can be applied to objects in a path tracer.

3. Overview

We introduce a mesoscale primitive that provides 3D mesoscale detail mapped to a surface (see Figure 1). Our model needs to: i) generate a mesoscale structure of a given extent that is consistent when viewed from different directions, and free of seams, aliasing, and repetition artifacts, ii) integrate well into a path tracer avoiding neighborhood operations during rendering, such as convolutions and spatial upsampling, and iii) permit relighting and artistic control over a selected set of conditioning parameters.

To satisfy these requirements, we combine 2D generative modeling and volumetric neural fields as shown in Fig. 2. A 2D Style-GAN3 generator (left) is used to produce a feature texture and height in a preprocessing stage. The texture and height vector condition a MLP (middle) during rendering, which infers appearance parameters of the mesostructure. The powerful convolutional architecture of the generator enables the use of a smaller MLP [CLC*22], and scales better to high-dimensional conditioning parameters.

There are three key elements to our method: First, we inject randomized Fourier features in the StyleGAN3 generator to allow synthesizing feature textures of any size (Sec. 4.1). Second, we impose an interpretation on the MLP-inferred values that is compatible with path tracing queries (Section 4.2), and we reduce the evaluation cost by inferring only inside a geometric shell [PBFJ05] that bounds the mesostructure. Third, the model is trained end-to-end by

discriminating rendered and reference images, both conditioned on appearance parameters to enable artistic control.

4. Method

We next describe the key components of our method: the generator of arbitrarily-sized feature textures, the mesoscale neural reflectance shell, conditioning on artistic parameters, and multi-scale training.

4.1. Feature Texture & Height Generator

Our goal is to create a texture generator that avoids repetitions and outputs a feature texture and height that can be applied to objects of any size. We use a generator based on StyleGAN3 [KAL*21] that we augment by injecting noise in the form of random Fourier features.

| Translated by −1 in *x* | No translation | Translated by +1 in *x* |
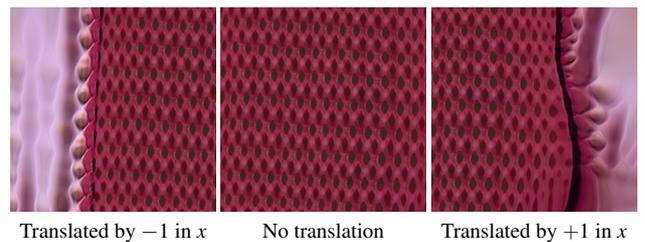
Figure 3: *StyleGAN3 can be employed to generate fixed-size textures (middle) but the static Fourier features utilized by the generator limit the amount of translation that can be applied until artifacts appear (left/right); the model was trained using the Describable Textures Dataset [CMK*14].*

The idea of injecting noise into the generator was first utilized in the original StyleGAN [KLA19], but dropped in later versions. In our case, we found that providing the generator with a well-formed

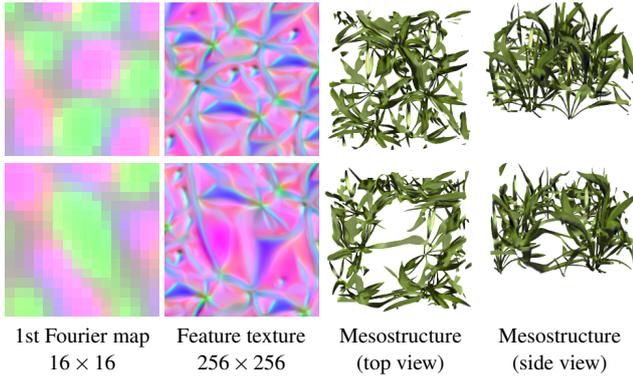| 1st Fourier map 16×16 | Feature texture 256×256 | Mesostructure (top view) | Mesostructure (side view) |

Figure 4: *Once trained, the StyleGAN generator transforms patterns in Fourier maps into patterns in the feature texture. We demonstrate this for two random instances by visualizing the first 3 channels in the Fourier map and the resulting feature texture. The rendered mesostructure is shown in the right two columns (top and side view). Notice how the mesostructure changes between the two rows.*

Figure 5: *Illustration of our randomized Fourier feature injections to allow the creation of feature textures with arbitrary size. Blue boxes depict new and changed components that we added to StyleGAN3.*

source of noise improves synthesis of stochastic details. Hence, we reintroduce randomness in the form of *randomized* Fourier features. Fourier features [TSM*20, XWC*21] have already been employed in the original StyleGAN3 architecture, but those were kept static to aid learning of specific visuals in specific locations (e.g., eyes, nose). The architecture is thus limited in the range of translation that can be applied, see Fig. 3, and can only target the specific texture resolution used during training.

Our goal is to permit variations of a greater degree in order to create rich, non-repetitive textures that can be applied to objects of any size. We thus introduce two key novelties: 1) We randomize the phases of Fourier features during training, and 2) we inject a collection of these Fourier features into every level of the generator. We will refer to the collection of features as the Fourier map.

4.1.1. Phase Randomization.

A Fourier map Γ has N channels where each channel k contains a 2D sinusoid with different frequency $f_k \in \mathbb{R}^2$ and phase $\phi_k \in \mathbb{R}^2$; $\Gamma_k = \sin(2\pi(f_k \mathbf{x} + \phi_k))$. We randomize the sinusoids as follows. The frequencies of the sinusoid are initialized to random values and kept constant during training so that the generator can learn specific operations for each frequency. In contrast, the phase of each sinusoid ϕ_k is randomly sampled in each training iteration.

The phase randomization forces the network to translate local patterns in the Fourier map into patterns that exist in the target material, as shown in Fig. 4, allowing for a continuous synthesis of large mesostructures with unique details.

4.1.2. Per-Layer Noise Injection.

The StyleGAN3 generator consists of L levels. In our method, we inject a random unique Fourier map into each level of the generator, see Fig. 5 for an illustration. To ensure that the frequency content of each Fourier map is appropriate for the level, we inject frequencies only between the cutoff frequency of the previous layer and the
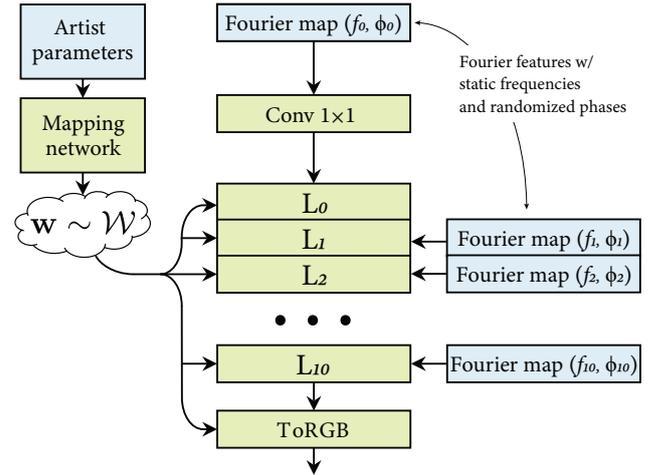
cutoff frequency of the current level; see Karras et al. [KAL*21] for a thorough explanation of cutoff frequencies.

More formally, for level l, which in the original generator consumes an input tensor with k channels, we create a Fourier map with k sinusoids. The frequencies of the sinusoids are chosen prior to training by sampling a 2D annulus with min and max radii set to the cutoff frequencies $f_{c,l-1}$ and $f_{c,l}$, respectively. The phases are randomized at each training iteration, $\phi_k \sim \mathcal{U}(0,1)$. The sinusoids are discretized using a grid with resolution equal to the width $\times$ height of the input tensor, and the Fourier map is injected by *addition* to the input tensor.

In the original StyleGAN3 generator, the width and height of input tensors at each level (and thus the resolution of our Fourier maps) are derived from the resolution of the output image using a carefully designed heuristic; we inherit this strategy from Karras et al. [KAL*21]. We only differ in using different output resolutions between training and inference. During training, the output feature textures have resolution 256×256, the feature height 1×256 and the rendered image depicts only a small extent of the mesostructure; this is to accommodate large memory requirements of the backward pass. At inference, the resolution of the feature texture is far less constrained as we perform only the forward pass. The resolution is set by the user according to the desired extent of the mesostructure; we used up to $3k \times 3k$ feature textures. The feature height has the same resolution (256) as in training, since the mesostructure has similar extent vertically between training and rendering.

4.2. Mesoscale 3D neural primitives

In this section, we describe the process of lifting the 2D feature texture into a 3D volume using an MLP. We start by postulating the requirements for integrating the resulting volumetric neural primitive into a light transport simulation. We are interested in

creating a volumetric primitive with a clearly defined boundary (e.g., the shell in Fig. 2). Such primitives can be easily integrated into physically based renderers, as long as we can relate the incident and outgoing radiance functions at the boundary.

4.2.1. Internal transport

The radiance $L_o(x, \omega)$ outgoing from a point on the boundary is defined as the amount of incident flux that hits the boundary, propagates through the volume, and exits at (x, ω). L_o is a sum of two terms: scattered radiance, i.e., light that interacts with the volume, and uncollided radiance, i.e., light that passes through unobstructed.

The *scattered* term L_s is illustrated in Fig. 6 and can be formalized by integrating contributions from the boundary along ray $(x, -\omega)$:

$$L_s = \int_0^b T(x, y)\sigma(y) \int_A \int_{H^2} f(y, \omega, z, \omega') L_i(z, \omega') d\omega' dz dt, \quad (1)$$

where b is the length of the ray, $T(x, y)$ is transmittance, and $\sigma(y)$ is the extinction coefficient at point $y = x - t\omega$. The inner double integral accounts for all incident radiance L_i that hits the boundary with total area A through the upper hemisphere H^2, and, after interacting with the medium, "enters" the ray by scattering at position y in direction ω'. The transport through the medium is quantified by the function $f(z, \omega', y, \omega)$, which includes also the albedo and phase function at y. Our goal is similar to other NeRF techniques, i.e., to estimate L_o by a small amount of numerical integration on top of values inferred by the MLP.

In order to facilitate pretraining of the model in a canonical setup, i.e., without having to account for macroscopic variations, we introduce the following assumption. We assume that the volume relevant for evaluating the triple integral is small in contrast to macroscale variations (NeRF-Tex [BGP*21] uses the same assumption). This allows us to treat all sources of illumination as distant, i.e., producing parallel, spatially invariant incident radiance $L_i(\omega')$. Second, we assume that the volume receives light through the top and bottom sides of the boundary only. Incorporating these assumptions and swapping the inner two integrals yields:

$$L_s = \int_0^b T(x, y)\sigma(y) \int_{H^2} L_i(\omega') \underbrace{\int_A f(y, \omega, z, \omega') dz}_{\rho(y, \omega, \omega')} d\omega' dt, \quad (2)$$

We train our neural model to infer i) the extinction coefficient $\sigma(y)$ and ii) the transport function f integrated over the boundary (the inner-most integral), which we will denote $\rho(y, \omega, \omega')$. The remaining two integrals are evaluated numerically. To estimate $L_i(\omega')$ we find an exit point z by shooting a ray from within the shell and intersecting it with the boundary.

The *uncollided* term equals to radiance incident at the other side of the boundary attenuated by transmittance through the shell.

The approximation error due to the aforementioned assumption manifests typically as incorrect brightness of the mesostructure in regions where the mesostructure ends. This is a price that we are currently willing to pay as this assumption allows us to create material primitives that can be applied to any geometry *after* the neural components have been trained, and allow easy integration into a path tracer.
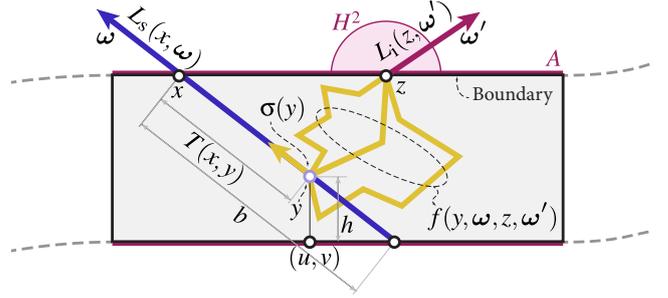

Figure 6: *In order to enable pre-training, we assume that i) internal light paths (yellow) that contribute radiance to a ray (blue) are confined to a locally flat region of the mesostructure, and ii) that incident radiance L_i is spatially invariant at the top and bottom sides of this region, i.e., the size of the region is negligible compared to the distance to sources of illumination.*

4.2.2. Conditioning on feature texture

Next, we describe how to condition the MLP using the generated feature texture and height vector to obtain the parameters of the mesostructure. Given a query point, we first map it to the base surface using orthogonal projection. This yields (u, v) coordinates, at which we bilinearly interpolate the feature texture F obtaining a feature vector $F(u, v)$. We then map the query point on the 1D feature vector using its height h and use linear interpolation to acquire the feature $F(h)$. We add the two vectors to get the final features $F_{total}(u, v, h) = F(u, v) + F(h)$. We also condition the MLP on the relative height h within the volumetric primitive, as well as the incoming and outgoing directions ω' and ω. The extinction coefficient and the transport function at the query point are obtained by evaluating the MLP as follows:

$$[\sigma(y), \rho(y, \omega, \omega')] = MLP(F_{total}(u, v, h), \omega, \omega'; \theta), \quad (3)$$

Our approach bears similarity with the EG3D method [CLC*22], which conditions the MLP using three orthogonal feature images generated by StyleGAN. While the EG3D setup is well suited for scenes that are well bounded by a cube, we opted for a single-texture plus height approach that is better suited for mesostructures that are typically shallow in contrast to their lateral extent.

In order to place the neural mesostructure into a 3D scene, we use a base mesh extruded into a volumetric shell [PBFJ05]. We intersect each ray against the shell and perform ray marching to sample a collision with the mesostructure, performing the aforementioned queries of the MLP at each step. Depending on whether a collision is simulated, we evaluate either the scattered or the uncollided radiance. In both cases the ray is continued further into the scene, either from the exit point at the boundary (uncollided radiance), or from a sampled location at the boundary to estimate the incident radiance $L_i(\omega')$. Details of ray marching through the shell are discussed in Appendix A.
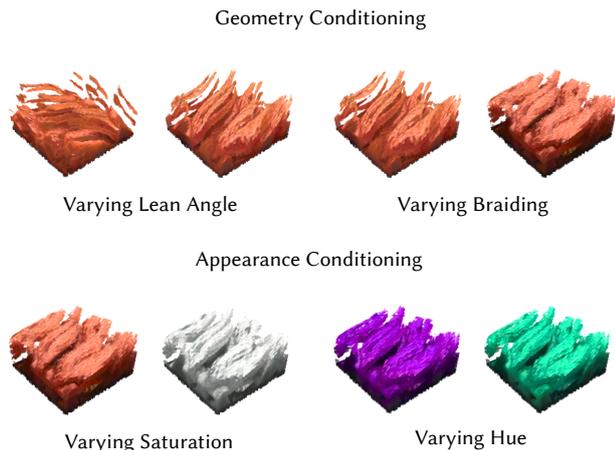
Geometry Conditioning

Varying Lean Angle   Varying Braiding

Appearance Conditioning

Varying Saturation   Varying Hue

Figure 7: *Our neural 3D textures can be controlled by artist-friendly parameters modifying the geometry and appearance of the material. Here we show an instance of fur with varying length and color.*

4.3. Artistic Control

As artistic control is crucial in our generative neural primitives, we condition the model on additional parameters to enable appearance modifications at inference time. Since we train on synthetic data, we can easily construct large labeled datasets with multiple modes of variation. We split the labels into two categories: geometry and appearance parameters represented by β and α respectively.

The generator of StyleGAN is conditioned on different styles **w**, which are generated by a mapping network (an MLP). In our case, it maps the geometric conditioning parameters β, such as fur length, to the style latent space $\mathcal{W}$ (see Fig. 7 and 9). We input only the geometric parameters, which change the structure of the texture, to the mapping network. The appearance parameters α are concatenated to the feature vector $F(u,v)$ to condition only the MLP as they affect the final color and not the geometry of the mesostructure.

5. Data Generation and Training

Our goal is to apply the once-trained mesostructures to many surfaces of different sizes. We therefore train the model in a canonical scene with a relatively small volume of the mesostructure that is free of any macro-scale variations. The mesostructure is placed inside a unit box and illuminated by a single distant light source. In this setup, mapping of points inside the cube to UV and height coordinates becomes the trivial identity map.

5.1. Multi-scale training

The main challenge with training our model is overcoming the high memory requirements during optimization. When combining StyleGAN-based generators with volumetric neural fields, the maximum training resolution of ray-marched images is 128x128 before GPU memory runs out (we used V100 GPUs). This heavily restricts the amount of visual detail that can be learned by the model.

Multi-Scale Dataset   No Mipmapping   Ours

Figure 8: *When we train on a multi-scale dataset (left), the network is not aware of the scale when queried, resulting in aliasing (middle). Our mipmapping approach on the feature textures removes the aliasing issue (right).*

Previous methods try to overcome this using convolutional super-resolution leading to view inconsistencies [CLC*22] or by limiting gradients [ZXNT21] resulting in unreasonably long training times. Neither is acceptable in our context.

Instead, we propose to train the model using a progressive multi-scale dataset, where individual images capture a patch of mesostructure from different distances. Naively training on such dataset with point queries leads to aliasing when the material is rendered from far away and blurriness when rendering close ups (see Figure 8). It is thus vital that we employ the filtered lookups.

To overcome the issues of aliasing, we use mipmapping for the learned feature texture and height. We follow MipNeRF [BMT*21] using cones instead of rays, with cone footprints being approximated by 3D Gaussians. We project each 3D Gaussian to the tangent plane at the corresponding (u,v,h) coordinate. For the learned height, we use the extent of the 3D Gaussian for filtering by bilinearly interpolating based on the height and scale. For the learned 2D texture, the 3D Gaussian becomes an ellipse $(x,y) = (a\cos t, b\sin t)$, which we approximate with a circle of filtering radius r, where $r = \min(a,b)$. The size of the circle is used to trilinearly interpolate the levels of the feature mipmap.

Our multi-scale training procedure optimizes fine mip levels of the feature texture when the camera is close to the mesostructure, and coarse mip levels when the camera is far (Figure 8). With this approach, the visual quality of mesostructures produced by our model is less restricted by the low resolution of training images (128×128) but rather by the resolution of the StyleGAN-generated feature texture; we were able to generate feature textures up to $3k \times 3k$ resolution in our experiments.

To ensure that the generator learns the higher level structure first, we place the camera far away from the mesostructure during early parts of the training and progressively allow it to get closer as the training converges.

Discriminator. We use the same architecture of the discriminating network and the non-saturating logistic loss as in Style-GAN3 [KAL*21]. The main difference is that we also provide the discriminator with the geometry and appearance labels, and the camera pose. We train the model end-to-end using datasets discussed next.
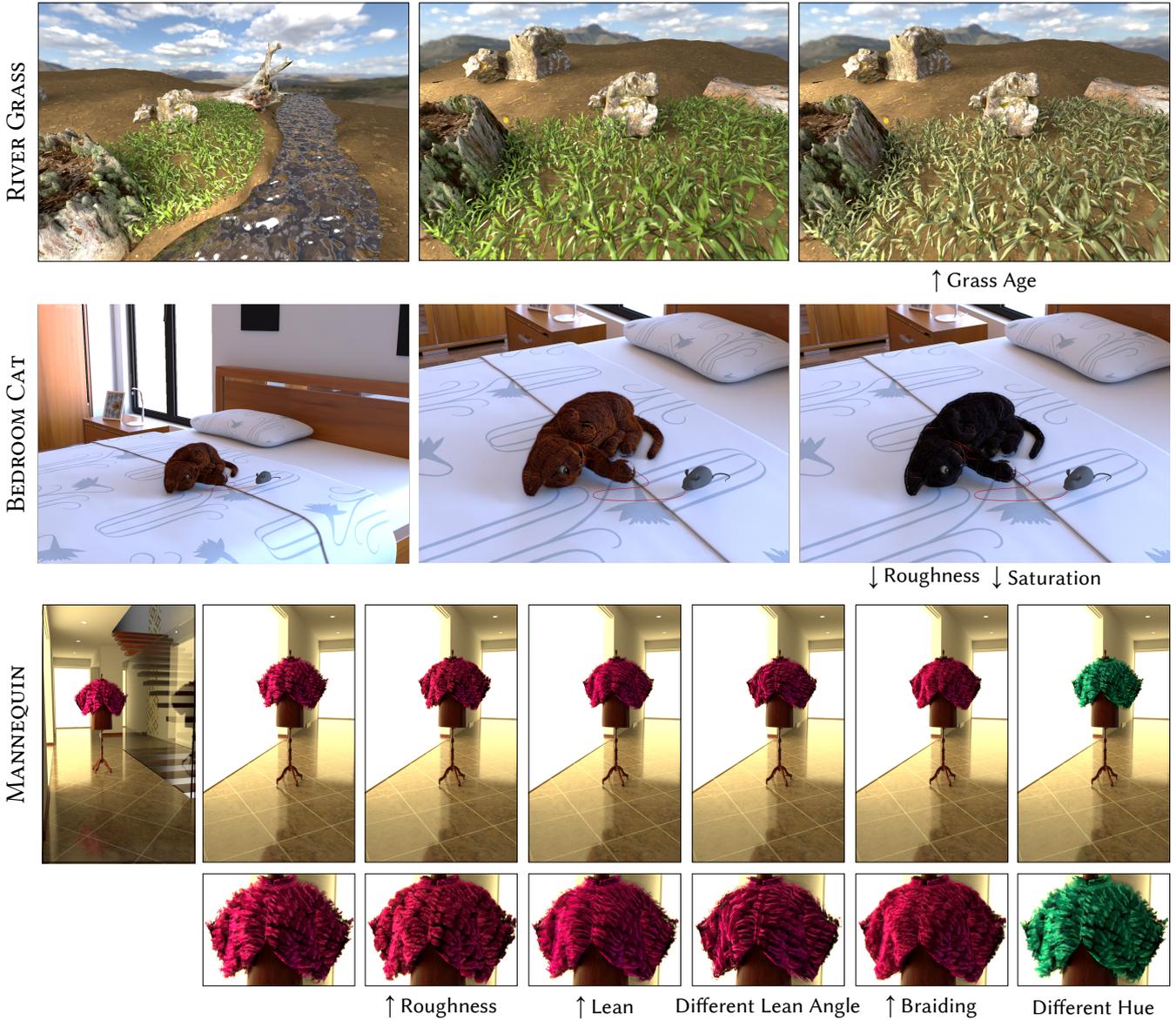
↑ Grass Age

↓ Roughness ↓ Saturation

↑ Roughness ↑ Lean Different Lean Angle ↑ Braiding Different Hue

Figure 9: *Path-traced images of scenes with global illumination featuring three different mesostructures using our generative models. With Meso-GAN these materials can be rendered with full support of global illumination effects such as the reflections in* RIVER GRASS *and in* MANNEQUIN. *We also show the many different appearances that our models can represent in* MANNEQUIN.

5.2. Datasets

We tested our method on three different mesostructures; *grass*, *carpet* and *fur*. The grass mesostructure has artistically controllable appearance using the age parameter, which modulates the albedo and transparency. The appearance of the carpet can be controlled via hue and saturation parameters, and its geometry via length, lean, lean angle, roughness and braiding parameters. The fur model allows control of its geometry through length and roughness and its appearance through color.

The datasets for these mesostructures were created in Blender

using the Cycles renderer. Each dataset contains 25k images, and is rendered as follows: We instantiate a slab of the mesostructure randomizing the procedural modeling through geometric and appearance parameters. Only about one ninth of the slab (a cubical box in the center) is visible to primary rays, the rest of the mesostructure is visible to secondary rays only to correctly simulate global illumination near the box boundaries. We place the camera randomly on a hemisphere above the scene and point it towards the center of the box. The hemisphere has randomized radius to facilitate multi-scale training described in Sec. 5.1. Our datasets bear similarity with those created for NeRF-Tex [BGP*21]. The main difference is the random

instancing of geometry to facilitate generative modeling in our case. The remaining differences are described in Appendix B.
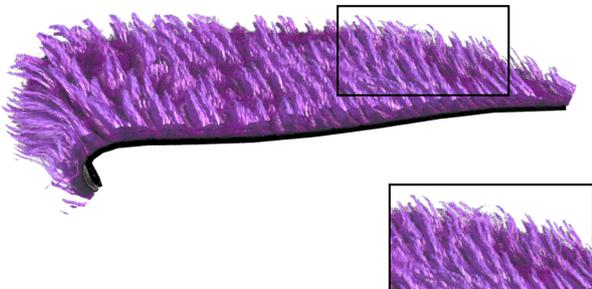
6. Implementation and Results

Figure 10: *Our model naturally supports anisotropic materials.*

We integrated our method in the Mitsuba 2 path tracer [ND-VZJ19], and used the system to render images in Figures 1, 9 and 10. We will release all code for training and rendering) and datasets on publication. The architecture of the feature-texture generator matches StyleGAN3 [KAL*21]. The MLP that infers the parameters of the mesostructure has 4 hidden layers with 64 neurons and ReLU activations.

We trained our models on 8 NVIDIA V100 32GB GPUs for 4 days. Our GPU memory usage for each GPU is 21 GB for a resolution of 128x128 and a batch size of 4 per GPU. Higher resolutions require GPUs with larger memory, or smaller batch sizes and consequently smaller optimization steps, in both cases yielding training times on the order of weeks.

In Figures 1 and 9 we demonstrate how our mesoscale 3D neural textures can be used in scenes with global illumination effects. In the MANNEQUIN scene we demonstrate the range of appearances our models can represent while being used in a path tracer with accurate reflections and global illumination effects. Please also see the supplemental videos, showing paths in these scenes with varying parameters during camera motion. These videos illustrate the power of Meso-GAN that allows direct generation of an infinite variety of mesoscale appearance.

6.1. Comparison to π-GAN and StyleNeRF

A number of 3D-aware image synthesis techniques combining concepts from NeRF [MST*20] and StyleGAN [KLA19] have been proposed. In Fig. 11, we compare to two of these, π-GAN [CMK*20] and StyleNeRF [GLWT21].

π-GAN is arguably the most direct realization of this idea, proposing a generative NeRF modeled using an MLP that is conditioned on a noise vector produced by a StyleGAN-inspired mapping network. As the MLP output is further processed only by a simple integration scheme (ray marching), π-GAN naturally inherits the multi-view consistency of NeRF, and is amenable for integration into path tracers.

StyleNeRF offers a marked visual improvement over π-GAN achieved through the use of a post-process spatial upsampling. While the use of spatial upsampling offers compelling benefits, it also comes at the cost of (mild) multi-view inconsistency. Most importantly though, spatial upsampling in post-process is fundamentally at odds with integrating the method as a scene primitive into a path tracer, where the primitive will be hit by incoherent secondary rays.

We report the Fréchet inception distance (FID) [HRU*17] for our π-GAN, StyleNeRF and our method in Table 1. We compute these metrics at the same resolution (128x128) to be fair to all methods. We explain why StyleNeRF has higher FID scores in Appendix C.

Our approach shares similarities with the EG3D [CLC*22] architecture, in the sense that we first employ a convolutional component and afterwards interpret its output using point-wise MLP queries. However, unlike StyleNeRF and EG3D, we do not further upsample the MLP output using spatial convolutions. This allows us to reap the benefits of the powerful StyleGAN generator in ways that are compatible with integration into path tracers. The added benefit is that the expensive feature-texture generation happens in a pre-process; during rendering we evaluate only the MLP.

	CARPET 2D	GRASS
π-GAN	200.51	214.98
StyleNeRF	60.24	32.54
Ours	9.01	4.40

Table 1: *FID results for our comparisons to StyleNeRF and π-GAN.*

6.2. Comparison to EG3D

Our method is inspired by EG3D and its triplane volumetric representation. In this comparison we demonstrate the limitations of EG3D in terms of view consistency and show that there is no elegant way to directly apply the triplane representation in our scenario of neural reflectance shells, motivating the use of our proposed neural feature plane and height approach.

In Figure 12 we show qualitative and FID results when we train EG3D on our synthetic GRASS patch dataset. In the supplementary video we also demonstrate that like StyleNeRF, EG3D is not view consistent. To improve the view consistency we would have to increase the ray marching resolution and rely less on image space upsampling. As we show in Table 2 this takes the memory usage over the available 32GB in each GPU when we train on 8 V100 GPUs. This motivates our mipmapping approach which improves the quality of the learned volume without an increase in memory usage.

While EG3D achieves an FID of 7.44 (see appendix C) the vanilla approach cannot be integrated directly with our shell maps. Their super-resolution module operates on image space and needs access to neighboring pixels to apply convolutions and bilinear upsampling. As a result we cannot compare with EG3D on full path traced scenes but we choose to provide an alternative variant of our method named *Triplanes* using the orthogonal planes suggested by EG3D.

The three orthogonal planes of EG3D are generated by the same generator and they are entangled. New variations in the shape of the
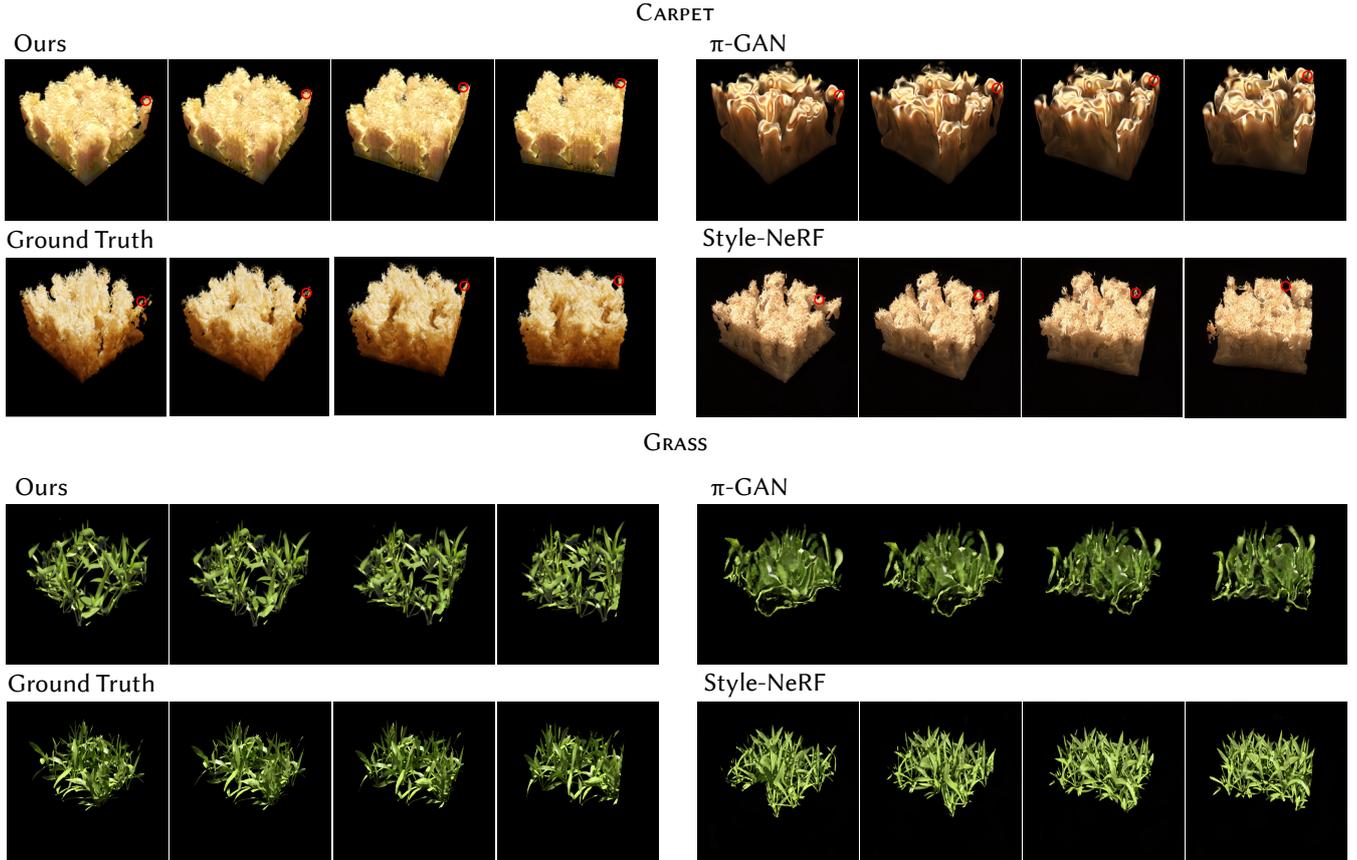
CARPET

Ours

Ground Truth

π-GAN

Style-NeRF

GRASS

Ours

Ground Truth

π-GAN

Style-NeRF

Figure 11: *Four views of a single instance of the carpet and the grass mesostructures synthesized by other generative approaches and our method. In contrast to StyleNeRF, which yields comparable quality to our method, our method is multi-view consistent and free of appearing and disappearing visual features (see the highlighted region and supplementary video).*

mesoscale material are controlled by the random noise z. To use this approach without changing the triplane approach we break the single shell into smaller randomly rotated ones. Each patch now has its own triplanes to make use of the generative aspect of the method. The resulting rendering is shown in Figure 12. The instantiated patches result in $9.78\times$ more rendering time since different rays need to decode several feature planes to get reflectance. In comparison our method has more stochastic structure and runs the decoder only once on the scaled feature planes and height, demonstrating that in the scenario of integrating neural reflectance shells into path tracing our approach is the best fit.

6.3. Comparison to NeRF-Tex

NeRF-Tex [BGP*21] proposes to model mesostructures using an MLP conditioned on various artistic parameters authored as 2D textures. Our work can be viewed as an extension to NeRF-Tex [BGP*21] where we add an extra, StyleGAN-generated feature texture for conditioning the MLP. This greatly expands the capabilities of the model: instead of optimizing the MLP to "memorize" a single instance of the mesostructure, we train it to map feature

	EG3D	
Ray Marching Resolution	128	256
Memory Usage	25 GB	Over 32 GB
	Ours	
Ray Marching Resolution	128	128 + MipMapping
Memory Usage	20 GB	20 GB

Table 2: *The only option to improve view consistency in EG3D is to increase the ray marching resolution and reduce the importance of the image space upsampling. This leads to an explosion in memory usage going over the 32GB of a modern GPU. In comparison our proposed mipmapping can improve the quality of the learned 3D volume, as is shown in Table 3, without an increase in memory usage.*

textures to 3D assemblies, thereby allowing to create an arbitrary number of novel mesostructures.

We compare our method to NeRF-Tex in Fig. 13 where we increase the number of conditioning artistic parameters from 2 to 8. The modeling capacity of the NeRF-Tex MLP is quickly exhausted,
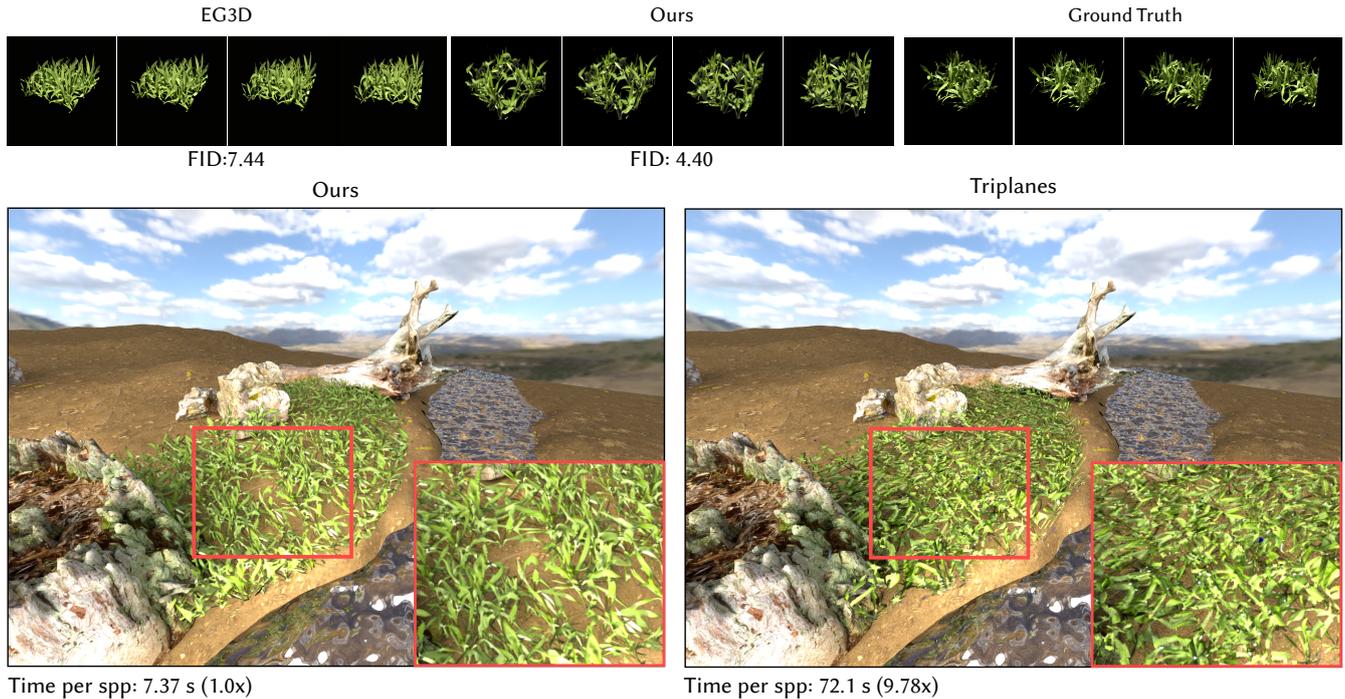
EG3D   Ours   Ground Truth

FID:7.44   FID: 4.40

Ours   Triplanes

Time per spp: 7.37 s (1.0x)   Time per spp: 72.1 s (9.78x)

Figure 12: *Comparison of our method to EG3D on the Grass dataset with FID (top). We also create the Triplanes variant of our method to demonstrate the limitations of the orthogonal planes representation when applied to shell mapping. This variant uses the same triplanes approach as EG3D but has to resort to multiple instances of mesoscale material resulting in 9.78× longer rendering and overall less natural variation in appearance.*
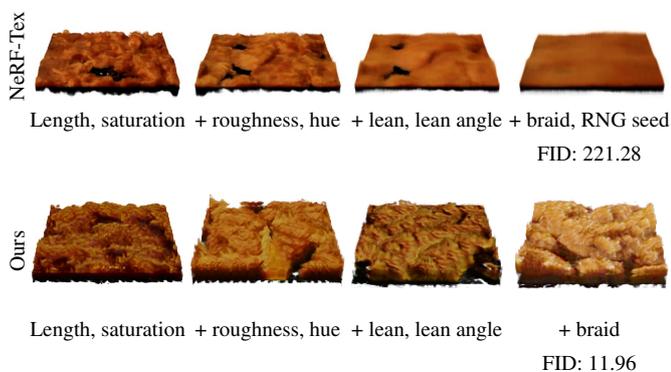

NeRF-Tex

Length, saturation + roughness, hue + lean, lean angle + braid, RNG seed

FID: 221.28

Ours

Length, saturation + roughness, hue + lean, lean angle   + braid

FID: 11.96

Figure 13: *From left to right, we show results of training the parametric NeRF-Tex [BGP*21] and our generative method on datasets with 2, 4, 6, and 8 parameters conditioning the look. The visual quality of NeRF-Tex reduces quickly as more parameters are added. In the extreme of conditioning on the RNG seed (last column), the model collapses producing average appearance. We can treat this iteration of NeRF-Tex as a generative model and compare it against our method quantitatively using FID. Our generative approach is agnostic to the number of parameters leading to much more detailed appearance and improved FID.*

leading to increasingly blurry results. We also tested training NeRF-Tex on a dataset with random instances of the mesostructure, conditioning the MLP on the unique seeds used by the procedural modeling tool (right-most column). As expected, the method fails to learn such a complex mapping collapsing to an "average" mesostructure. What we observe qualitatively is confirmed quantitatively if we treat this iteration of NeRF-Tex as a generative model and compute the FID. NeRF-Tex achieves a score of 221.28 while our method achieves 11.96.

The other disadvantage of NeRF-Tex is the limited extent to which it can produce anisotropic assemblies (e.g., combed hair). The method models a slab of the mesostructure by repeatedly instantiating the same patch over the surface while randomly rotating the instances to prevent repetitive appearance. The randomization of rotation, however, precludes creating anisotropic assemblies; these would require orienting all instances in a similar way. In contrast, our approach is readily capable of handling anisotropic structures as illustrated in Figure 10.

6.4. Ablations

We perform three ablations on the main components of our approach: we first disable Fourier Features and also disable mipmapping. The numerical results are shown in Tab. 3 for the FID metric [HRU*17]. We see that in both cases mipmapping has a significant effect, while Fourier features are also beneficial to a lesser degree.

	CARPET 8D
Full	11.96
No-Fourier Features	26.8
No-MipMapping	58.21

Table 3: *FID results on ablations of our method.*

We also disable the phase randomization during training and demonstrate how the generator generalizes when it has only seen the frequencies at specific phases. Similar to what we observe in Figure 3, in Figure 14 we see that the generator operates well within the ranges of the patch but outside of that it produces only low frequency patterns.
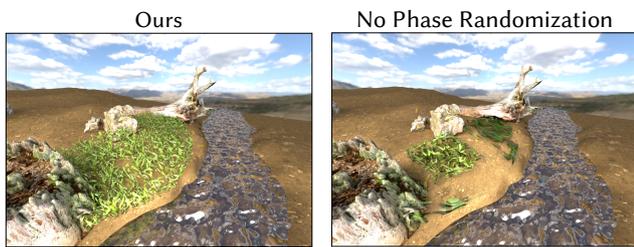

Ours   No Phase Randomization

Figure 14: *We disable the randomization of phases (right) and compare with our full pipeline (left). When we apply this variant on a shell map the generator is able to reproduce the mesoscale material only in the known patch region.*

7. Conclusion and Future Directions

We have introduced MesoGAN, a generative mesoscale material primitive, that enables synthesizing neural mesoscale materials of variable extent without loss of detail, seams, or aliasing artifacts. This is due to the injection of randomized Fourier features, filtering, and multi-scale training. Relighting and artistic control are retained by conditioning the model on parameters during training. We propose to use the shell-mapping approach for placing the mesostructure on triangle meshes, which sidesteps the limitations of prior works. The visual quality is encouraging, yet still far from production requirements.

Future research should focus on increasing the resolution of the feature texture during training without increasing memory requirements. Promising alternatives in this direction are patch based methods such as EpiGRAF [STWW22]. We also believe that adapting the training procedure of our method to photographic datasets captured in the wild is crucial for the success of neural materials in the future. In this scenario, new challenges arise as we need to estimate camera poses, lighting conditions and material parameters, all of which are given in our synthetic datasets. Our work serves as the first step towards utilizing generative neural models for creating faithful reproductions of real mesostructures and integrating these into physically based renderers.

Appendix A: Marching through shells

We bound the mesostructure using a geometric shell [PBFJ05], which consists of triangular prisms extruded from triangles of the base mesh along vertex normals. For each ray hitting the shell, we perform the following steps. We find the first intersection with the boundary then we step through the prisms by intersecting the ray against bilinear patches [Res19] that form the sides of the prisms. Given a budget of N steps we first compute all the intersections for each ray within the shell. Then we sort and batch based on the number of intersections I. We add N/I new steps by computing the UV coordinates $(u,v)_x$ and height h_x for each point in the steps x, by linearly interpolating these coordinates.

In our implementation, we use ray marching that generates K sampled locations along the ray segment overlapping the shell. For each location x_k, we repeat the procedure of finding the bounding prism, computing the $(u,v,h,r)_{x_k}$, fetching the generated features $F(u,v)$ and $F(h)$ and appearance parameters α_{x_k}. We collect these queries from all rays inputting them into the MLP in a single batch.

Appendix B: Other differences to NeRF-Tex
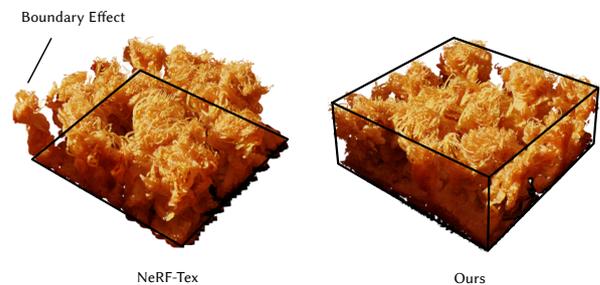

Boundary Effect

NeRF-Tex   Ours

Figure 15: *Comparison of training images used for NeRF-Tex [BGP*21] and our method (with added illustrations). In NeRF-Tex the images depict unclipped geometric assemblies that are surrounded by a ring of primitives visible to secondary rays only. In our case, we use a slab of the mesostructure rendering directly any part of the mesostructure inside a box (the rest of the slab is visible to secondary rays only).*

Here we describe technical differences to NeRF-Tex [BGP*21]. NeRF-Tex constructs the mesostructure by instantiating a single patch of the mesostructure at random locations with randomized orientation. We utilize shell mapping. Each approach requires slightly different training data. The training images for NeRF-Tex depict

a patch of complete geometric primitives. The primitives typically start on a tile and "grow" arbitrarily far from it. For shell mapping, the training images need to depict primitives that are clipped against a cubical boundary. This clipping is present only in the training images. At inference time, there is no notion of instancing of a patch; we generate a single feature texture for the entire base mesh. The difference in training images is illustrated in Fig. 15.

NeRF-Tex uses a parametric model that is optimized by penalizing deviations in radiance (SMAPE loss) and transmittance (MSE loss) from values in reference images in linear space. Our model is generative, relying on a convolutional loss that discriminates at the level of full images. We follow the standard practice when dealing with high dynamic range images and tonemap all images using $\frac{x}{x+1}$ [RSSF02].

Appendix C: Comparison to StyleNeRF and EG3D

Even though both StyleNeRF and EG3D achieve higher resolution results as shown in Figure 11 and Figure 12 (top), they score worse than our method in FID. When we train StyleNeRF and EG3D we observe that they struggle to recreate the pure black background due to their screen space operations. For StyleNeRF this also leads to artifacts, such as the one shown in Figure 16, that overall increases the FID score when its outputs are compared to the real pure black images.

Figure 16: *Artifacts observed when training StyleNeRF on the carpet mesostructure.*

References

[BGP*21] BAATZ H., GRANSKOG J., PAPAS M., ROUSELLE F., NOVÁK J.: Nerf-tex: Neural reflectance field textures. In *Eurographics Symposium on Rendering* (June 2021), The Eurographics Association. 2, 5, 7, 9, 10, 11

[BMT*21] BARRON J. T., MILDENHALL B., TANCIK M., HEDMAN P., MARTIN-BRUALLA R., SRINIVASAN P. P.: Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICCV* (2021). 2, 6

[CLC*22] CHAN E. R., LIN C. Z., CHAN M. A., NAGANO K., PAN B., MELLO S. D., GALLO O., GUIBAS L. J., TREMBLAY J., KHAMIS S., KARRAS T., WETZSTEIN G.: Efficient geometry-aware 3d generative adversarial networks. vol. abs/2112.07945. URL: https://arxiv.org/abs/2112.07945, arXiv:2112.07945. 2, 3, 5, 6, 8

[CMK*14] CIMPOI M., MAJI S., KOKKINOS I., MOHAMED S., , VEDALDI A.: Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2014). 3

[CMK*20] CHAN E., MONTEIRO M., KELLNHOFER P., WU J., WETZSTEIN G.: pi-GAN: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *arXiv* (2020). 2, 8

[DN09] DECAUDIN P., NEYRET F.: Volumetric billboards. *Computer Graphics Forum 28*, 8 (2009), 2079–2089. doi:10.1111/j.1467-8659.2009.01354.x. 2

[DvGNK99] DANA K. J., VAN GINNEKEN B., NAYAR S. K., KOENDERINK J. J.: Reflectance and texture of real-world surfaces. *ACM Trans. Graph. 18*, 1 (jan 1999), 1–34. URL: https://doi.org/10.1145/300776.300778, doi:10.1145/300776.300778. 2

[GLWT21] GU J., LIU L., WANG P., THEOBALT C.: Stylenerf: A style-based 3d-aware generator for high-resolution image synthesis. *CoRR abs/2110.08985* (2021). URL: https://arxiv.org/abs/2110.08985, arXiv:2110.08985. 2, 8

[HCZ21] HADADAN S., CHEN S., ZWICKER M.: Neural radiosity. *ACM Transactions on Graphics 40*, 6 (dec 2021). URL: https://doi.org/10.1145/3478513.3480569, doi:10.1145/3478513.3480569. 2

[HMR20a] HENZLER P., MITRA N. J., RITSCHEL T.: Learning a neural 3d texture space from 2d exemplars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020). 2

[HMR20b] HENZLER P., MITRA N. J., RITSCHEL T.: Learning a neural 3d texture space from 2d exemplars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 8356–8364. 3

[HRU*17] HEUSEL M., RAMSAUER H., UNTERTHINER T., NESSLER B., HOCHREITER S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems 30* (2017). 8, 10

[KAL*21] KARRAS T., AITTALA M., LAINE S., HÄRKÖNEN E., HELLSTEN J., LEHTINEN J., AILA T.: Alias-free generative adversarial networks. In *Proc. NeurIPS* (2021). 2, 3, 4, 6, 8

[KK89] KAJIYA J. T., KAY T. L.: Rendering fur with three dimensional textures. *SIGGRAPH Comput. Graph. 23*, 3 (July 1989), 271–280. doi:10.1145/74334.74361. 2

[KLA19] KARRAS T., LAINE S., AILA T.: A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2019), pp. 4401–4410. 3, 8

[KLA*20] KARRAS T., LAINE S., AITTALA M., HELLSTEN J., LEHTINEN J., AILA T.: Analyzing and improving the image quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020). 2, 3

[KMX*21] KUZNETSOV A., MULLIA K., XU Z., HAŠAN M., RAMAMOORTHI R.: NeuMIP: Multi-resolution neural materials. *ACM Transactions on Graphics (Proc. SIGGRAPH 2021) 40*, 4 (2021). 2

[KWM*22] KUZNETSOV A., WANG X., MULLIA K., LUAN F., XU Z., HAŠAN M., RAMAMOORTHI R.: Rendering neural materials on curved surfaces. In *SIGGRAPH 22* (2022). 2

[MBRS*21] MARTIN-BRUALLA R., RADWAN N., SAJJADI M. S. M., BARRON J. T., DOSOVITSKIY A., DUCKWORTH D.: NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR* (2021). 2

[MJC*03] MARSCHNER S. R., JENSEN H. W., CAMMARANO M., WORLEY S., HANRAHAN P.: Light scattering from human hair fibers. *ACM Trans. Graph. 22*, 3 (July 2003), 780–791. doi:10.1145/882262.882345. 2

[MPG*16] MÜLLER T., PAPAS M., GROSS M., JAROSZ W., NOVÁK J.: Efficient rendering of heterogeneous polydisperse granular media. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 168:1–168:14. doi:10.1145/2980179.2982429. 2

[MRKN20] MÜLLER T., ROUSSELLE F., KELLER A., NOVÁK J.: Neural control variates. *ACM Trans. Graph. 39*, 6 (Nov. 2020), 243:1–243:19. doi:10.1145/3414685.3417804. 2

[MST*20] 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). 2, 8

[NDVZJ19] NIMIER-DAVID M., VICINI D., ZELTNER T., JAKOB W.: Mitsuba 2: A retargetable forward and inverse renderer. *ACM Trans. Graph. 38*, 6 (nov 2019). URL: https://doi.org/10.1145/3355089.3356498, doi:10.1145/3355089.3356498. 8

[Ney98] NEYRET F.: Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (1998), 55–70. doi:10.1109/2945.675652. 2

[NG21] NIEMEYER M., GEIGER A.: GIRAFFE: Representing scenes as compositional generative neural feature fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2021). 2

[OMN*19] OECHSLE M., MESCHEDER L., NIEMEYER M., STRAUSS T., GEIGER A.: Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2019). 2

[PBFJ05] PORUMBESCU S. D., BUDGE B., FENG L., JOY K. I.: Shell maps. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, Association for Computing Machinery, p. 626–633. doi:10.1145/1186822.1073239. 3, 5, 11

[PBG20] PORTENIER T., BIGDELI S. A., GOKSEL O.: Gramgan: Deep 3d texture synthesis from 2d exemplars. *CoRR abs/2006.16112* (2020). URL: https://arxiv.org/abs/2006.16112, arXiv:2006.16112. 3

[Res19] RESHETOV A.: Cool patches: A geometric approach to ray/bilinear patch intersections. In *Ray Tracing Gems*. Springer, 2019, pp. 95–109. 11

[RGJW20] RAINER G., GHOSH A., JAKOB W., WEYRICH T.: Unified neural encoding of BTFs. *Computer Graphics Forum (Proc. Eurographics) 39*, 2 (July 2020), 167–178. doi:10.1111/cgf.13921. 2

[RJGW19] RAINER G., JAKOB W., GHOSH A., WEYRICH T.: Neural BTF compression and interpolation. *Computer Graphics Forum (Proc. Eurographics) 38*, 2 (Mar. 2019), 235–244. doi:10.1111/cgf.13633. 2

[RSSF02] REINHARD E., STARK M., SHIRLEY P., FERWERDA J.: Photographic tone reproduction for digital images. *ACM Trans. Graph. 21*, 3 (jul 2002), 267–276. URL: https://doi.org/10.1145/566654.566575, doi:10.1145/566654.566575. 12

[RWG*13] REN P., WANG J., GONG M., LIN S., TONG X., GUO B.: Global illumination with radiance regression functions. *ACM Trans. Graph. 32*, 4 (July 2013). doi:10.1145/2461912.2462009. 2

[SCT*20] SITZMANN V., CHAN E., TUCKER R., SNAVELY N., WETZSTEIN G.: MetaSDF: Meta-learning signed distance functions. In *Advances in Neural Information Processing Systems* (2020), Larochelle H., Ranzato M., Hadsell R., Balcan M. F., Lin H., (Eds.), vol. 33, Curran Associates, Inc., pp. 10136–10147. URL: https://proceedings.neurips.cc/paper/2020/file/731c83db8d2ff01bdc000083fd3c3740-Paper.pdf. 2

[SLNG20] SCHWARZ K., LIAO Y., NIEMEYER M., GEIGER A.: GRAF: Generative radiance fields for 3d-aware image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)* (2020). 2

[STWW22] SKOROKHODOV I., TULYAKOV S., WANG Y., WONKA P.: Epigraf: Rethinking training of 3d gans. *arXiv preprint arXiv:2206.10535* (2022). 11

[TSM*20] TANCIK M., SRINIVASAN P. P., MILDENHALL B., FRIDOVICH-KEIL S., RAGHAVAN N., SINGHAL U., RAMAMOORTHI R., BARRON J. T., NG R.: Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS* (2020). 2, 4

[TTM*21] TEWARI A., THIES J., MILDENHALL B., SRINIVASAN P., TRETSCHK E., WANG Y., LASSNER C., SITZMANN V., MARTIN-BRUALLA R., LOMBARDI S., ET AL.: Advances in neural rendering. *arXiv preprint arXiv:2111.05849* (2021). 2

[WM19] WYMAN C., MCGUIRE M.: Improved alpha testing using hashed sampling. *IEEE Transactions on Visualization and Computer Graphics 25*, 2 (2019), 1309–1320. doi:10.1109/TVCG.2017.2739149. 2

[XTS*22] XIE Y., TAKIKAWA T., SAITO S., LITANY O., YAN S., KHAN N., TOMBARI F., TOMPKIN J., SITZMANN V., SRIDHAR S.: Neural fields in visual computing and beyond. *Computer Graphics Forum* (2022). doi:10.1111/cgf.14505. 2

[XWC*21] XU R., WANG X., CHEN K., ZHOU B., LOY C. C.: Positional encoding as spatial inductive bias in gans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 13569–13578. 4

[ZXNT21] ZHOU P., XIE L., NI B., TIAN Q.: CIPS-3D: A 3D-Aware Generator of GANs Based on Conditionally-Independent Pixel Synthesis, 2021. arXiv:2110.09788. 6