EpiGRAF: Rethinking training of 3D GANs

Ivan Skorokhodov
KAUST

Sergey Tulyakov
Snap Inc.

Yiqun Wang
KAUST

Peter Wonka
KAUST

Abstract

A recent trend in generative modeling is building 3D-aware generators from 2D image collections. To induce the 3D bias, such models typically rely on volumetric rendering, which is expensive to employ at high resolutions. Over the past months, more than ten works have addressed this scaling issue by training a separate 2D decoder to upsample a low-resolution image (or a feature tensor) produced from a pure 3D generator. But this solution comes at a cost: not only does it break multi-view consistency (i.e., shape and texture change when the camera moves), but it also learns geometry in low fidelity. In this work, we show that obtaining a high-resolution 3D generator with SotA image quality is possible by following a completely different route of simply training the model patch-wise. We revisit and improve this optimization scheme in two ways. First, we design a location- and scale-aware discriminator to work on patches of different proportions and spatial positions. Second, we modify the patch sampling strategy based on an annealed beta distribution to stabilize training and accelerate the convergence. The resulting model, named EpiGRAF, is an efficient, high-resolution, pure 3D generator, and we test it on four datasets (two introduced in this work) at 256^2 and 512^2 resolutions. It obtains state-of-the-art image quality, high-fidelity geometry and trains $\approx 2.5\times$ *faster* than the upsampler-based counterparts.

Code/data/visualizations: https://universome.github.io/epigraf

1 Introduction

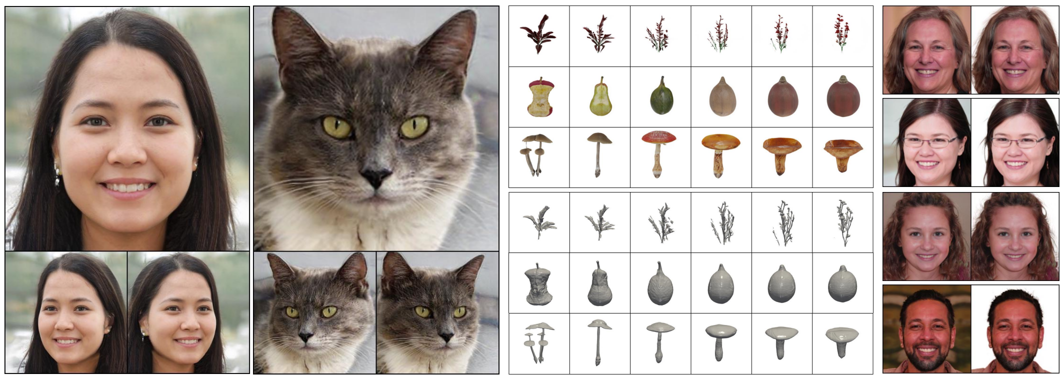


Figure 1: We build a pure NeRF-based generator trained in a patch-wise fashion. Left two grids: samples on FFHQ 512^2 [25] and Cats 256^2 [79]. Middle grids: interpolations between samples on M-Plants and M-Food (upper) and corresponding geometry interpolations (lower). Right grid: background separation examples. In contrast to the upsampler-based methods, one can naturally incorporate the techniques from the traditional NeRF literature into our generator: for background separation, we simply copy-pasted the corresponding code from NeRF++ [78].

Generative models for image synthesis achieved remarkable success in recent years and now enjoy a lot of practical applications [56, 24]. While initially they mainly focused on 2D images [21, 68, 25, 4, 28], recent research explored generative frameworks with partial 3D control over the underlying object in terms of texture/structure decomposition, novel view synthesis or lighting manipulation (e.g., [59, 57, 7, 70, 6, 12, 50]). These techniques are typically built on top of the recently emerged neural radiance fields (NeRF) [39] to explicitly represent the object (or its latent features) in 3D space.

NeRF is a powerful framework, which made it possible to build expressive 3D-aware generators from challenging RGB datasets [7, 12, 6]. Under the hood, it trains a multi-layer perceptron (MLP) $\mathsf{F}(\boldsymbol{x}; \boldsymbol{d}) = (\boldsymbol{c}, \sigma)$ to represent a scene by encoding a density $\sigma \in \mathbb{R}_+$ for each coordinate position $\boldsymbol{x} \in \mathbb{R}^3$ and a color value $\boldsymbol{c} \in \mathbb{R}^3$ from $\boldsymbol{x}$ and view direction $\boldsymbol{d} \in \mathbb{S}^2$ [39]. To synthesize an image, one renders each pixel independently by casting a ray $\boldsymbol{r}(q) = \boldsymbol{o} + q\boldsymbol{d}$ (for $q \in \mathbb{R}_+$) from origin $\boldsymbol{o} \in \mathbb{R}^3$ into the direction $\boldsymbol{d} \in \mathbb{S}^2$ and aggregating many color values along it with their corresponding densities. Such a representation is very expressive but comes at a cost: rendering a single pixel is computationally expensive and makes it intractable to produce a lot of pixels in one forward pass. It is *not* fatal for reconstruction tasks where the loss can be robustly computed on a subset of pixels, but it creates significant scaling problems for generative NeRFs: they are typically formulated in a GAN-based framework [14] with 2D convolutional discriminators requiring an entire image as input.

People address these scaling issues of NeRF-based GANs in different ways. The dominating approach is to train a separate 2D decoder to produce a high-resolution image from a low-resolution image or feature grid rendered from a NeRF backbone [44]. During the past six months, there appeared *more than a dozen* of methods that follow this paradigm (e.g., [6, 15, 73, 48, 81, 36, 77, 23, 74, 80, 66]). While using the upsampler allows scaling the model to high resolution, it comes with two severe limitations: 1) it breaks the multi-view consistency of a generated object, i.e., its texture and shape change when the camera moves; and 2) the geometry gets only represented in a low resolution ($\approx 64^3$). In our work, we show that by dropping the upsampler and using a simple patch-wise optimization scheme, one can build a 3D generator with better image quality, faster training speed, and without the above limitations.

Patch-wise training of NeRF-based GANs was initially proposed by GRAF [57] and got largely neglected by the community since then. The idea is simple: instead of training the generative model on full-size images, one does this on small random crops. Since the model is coordinate-based [60, 67], it does not face any issues to synthesize only a subset of pixels. This serves as an excellent way to save computation for both the generator and the discriminator since it makes them both operate on patches of small spatial resolution. To make the generator learn both the texture and the structure, crops are sampled to be of variable scales (but having the *same* number of pixels). In some sense, this can be seen as optimizing the model on low-resolution images + high-resolution patches.

In our work, we improve patch-wise training in two crucial ways. First, we redesign the discriminator by making it better suited to operating on image patches of variable scales and locations. Convolutional filters of a neural network learn to capture different patterns in their inputs depending on their semantic receptive fields [31, 47]. That's why it is detrimental to reuse the same discriminator to judge both high-resolution local and low-resolution global patches, inducing additional burden on it to mix filters' responses of different scales. To mitigate this, we propose to modulate the discriminator's filters with a hypernetwork [16], which predicts which filters to suppress or reinforce from a given patch scale and location.

Second, we change the random scale sampling strategy from an annealed uniform to an annealed beta distribution. Typically, patch scales are sampled from a uniform distribution $s \sim \mathcal{U}[s(t), 1]$ [57, 37, 5], where the minimum scale $s(t)$ is gradually decreased (i.e. annealed) till some iteration T from $s(0) = 0.9$ to a smaller value $s(T)$ (in the interval $[0.125 - 0.5]$) during training. This sampling strategy prevents learning high-frequency details early on in training and puts too little attention on the structure after $s(t)$ reaches its final value $s(T)$. This makes the overall convergence of the generator slower and less stable that's why we propose to sample patch scales using the beta distribution $\text{Beta}(1, \beta(t))$ instead, where $\beta(t)$ is gradually annealed from $\beta(0) \approx 0$ to some maximum value $\beta(T)$. In this way, the model starts learning high-frequency details immediately with the start of training and focuses more on the structure after the growth finishes. This simple change stabilizes the training and allows it to converge faster than the typically used uniform distribution [57, 5, 37].

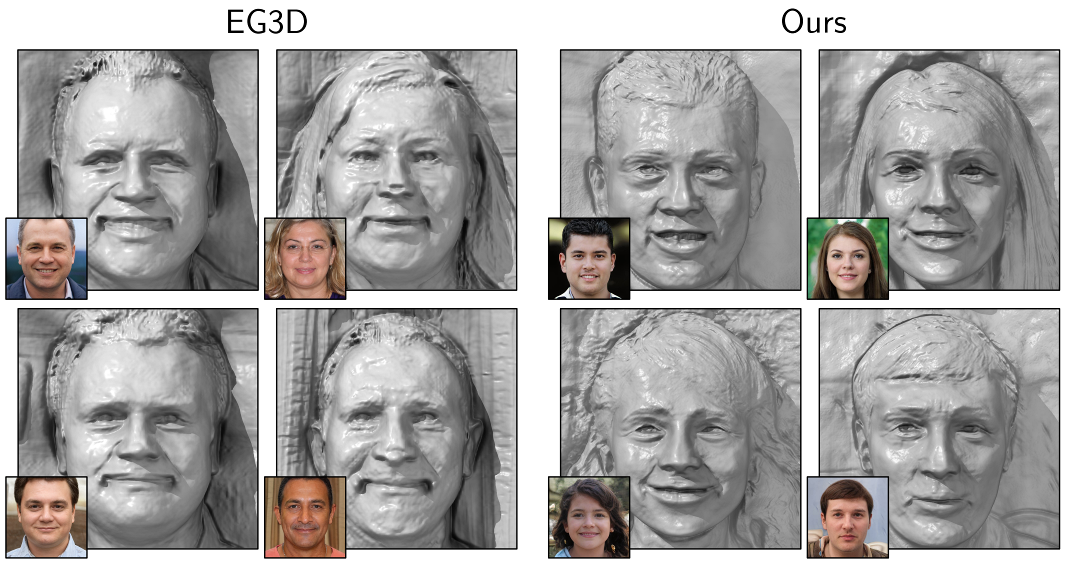


Figure 2: Comparing the geometry between EG3D [6] and our generator on FFHQ 512^2. For each method, we computed the density field in the 512^3 volume resolution and extracted the surfaces using marching cubes. The geometry of our generator contains more high-frequency details (e.g., hair strands are better separated) since it learns it in full resolution. EG3D uses the 64^2 rendering resolution (and 128^2 during the last 10% of the training) so its shapes appear over-smoothed.

We use those two ideas to develop a novel state-of-the-art 3D GAN: Efficient patch-informed Generative Radiance Fields (EpiGRAF). We employ it for high-resolution 3D-aware image synthesis on four datasets: FFHQ [25], Cats [79], Megascans Plants, and Megascans Food. The last two benchmarks are introduced in our work and contain $360°$ renderings of photo-realistic scans of different plants and food objects (described in §4). They are much more complex in terms of geometry and are well-suited for assessing the structural limitations of modern 3D-aware generators.

Our model uses a pure NeRF-based backbone, that's why it represents geometry in high resolution and does not suffer from multi-view synthesis artifacts, as opposed to upsampler-based generators. Moreover, it has higher or comparable image quality (as measured by FID [20]) and $2.5\times$ lower training cost. Also, in contrast to upsampler-based 3D GANs, our generator can naturally incorporate the techniques from the traditional NeRF literature. To demonstrate this, we incorporate background separation into our framework by simply copy-pasting the corresponding code from NeRF++ [78].

2 Related work

Neural Radiance Fields. Neural Radiance Fields (NeRF) is an emerging area [39], which combines neural networks with volumetric rendering techniques to perform novel-view synthesis [39, 78, 2], image-to-scene generation [76, 61], surface reconstruction [46, 71, 45] and other tasks [9, 17, 51, 29]. In our work, we employ them in the context of 3D-aware generation from a dataset of RGB images [57, 7].

3D generative models. A popular way to learn a 3D generative model is to train it on 3D data or in an autoencoder's latent space (e.g., [10, 72, 1, 35, 32, 40, 30]). This requires explicit 3D supervision and there appeared methods which train from RGB datasets with segmentation masks, keypoints or multiple object views [13, 33, 55]. Recently, there appeared works which train from single-view RGB only, including mesh-generation methods [19, 75, 54] and methods that extract 3D structure from pretrained 2D GANs [59, 49]. And recent neural rendering advancements allowed to train NeRF-based generators [57, 7, 43] from purely RGB data from scratch, which became the dominating direction since then and which are typically formulated in the GAN-based framework [14].

NeRF-based GANs. HoloGAN [42] generates a 3D feature voxel grid which is projected on a plane and then upsampled. GRAF [57] trains a noise-conditioned NeRF in an adversarial manner.

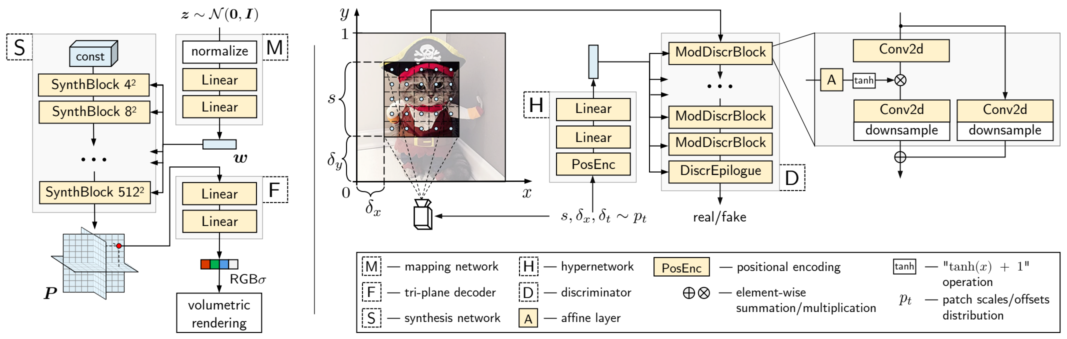


Figure 3: Our generator (left) is purely NeRF-based and uses the tri-plane backbone [6] with the StyleGAN2 [26] decoder (but without the 2D upsampler). Our discriminator (right) is also based on StyleGAN2, but is modulated by the patch location and scale parameters. We use the patch-wise optimization for training [57] with our proposed Beta scale sampling, which allows our model to converge ×2-3 faster than the upsampler-based architectures despite the generator modeling geometry in full resolution (see Tab 1).

π-GAN [7] builds upon it and uses progressive growing and hypernetwork-based [16] conditioning in the generator. GRAM [12] builds on top of π-GAN and samples ray points on a set of learnable iso-surfaces. GNeRF [37] adapts GRAF for learning a scene representation from RGB images without known camera parameters. GIRAFFE [44] uses a composite scene representation for better controllability. CAMPARI [43] learns a camera distribution and a background separation network with inverse sphere parametrization [78]. To mitigate the scaling issue of volumetric rendering, many recent works train a 2D decoder under different multi-view consistency regularizations to upsample a low-resolution volumetrically rendered feature grid [6, 15, 73, 48, 81, 74, 80]. However, none of such regularizations can currently provide the multi-view consistency of pure-NeRF-based generators.

Patch-wise generative models. Patch-wise training had been routinely utilized to learn the textural component of image distribution when the global structure is provided from segmentation masks, sketches, latents or other sources (e.g., [22, 58, 11, 69, 53, 52, 34, 63]). Recently, there appeared works which sample patches at variable scales, in which way a patch can carry global information about the whole image. Recent works use it to train a generative NeRF [57], fit a neural representation in an adversarial manner [37] or to train a 2D GAN on a dataset of variable resolution [5].

3 Model

We build upon StyleGAN2 [26], replacing its generator with the tri-plane-based NeRF model [6] and using its discriminator as the backbone. We train the model on $r \times r$ patches (we use $r = 64$ everywhere) of random scales instead of the full images of resolution $R \times R$. Scales $s \in [\frac{r}{R}, 1]$ are randomly sampled from a time-varying distribution $s \sim p_t(s)$.

3.1 3D generator

Compared to upsampler-based 3D GANs [15, 44, 74, 81, 6, 80], we use a pure NeRF [39] as our generator G and utilize the tri-plane representation [6, 8] as the backbone. It consists of three components: 1) mapping network $\mathsf{M} : \boldsymbol{z} \mapsto \boldsymbol{w}$ which transforms a noise vector $\boldsymbol{z} \sim \mathbb{R}^{512}$ into the latent vector $\boldsymbol{w} \sim \mathbb{R}^{512}$; 2) synthesis network $\mathsf{S} : \boldsymbol{w} \mapsto \boldsymbol{P}$ which takes the latent vector $\boldsymbol{w}$ and synthesizes three 32-dimensional feature planes $\boldsymbol{P} = (\boldsymbol{P}_{xy}, \boldsymbol{P}_{yz}, \boldsymbol{P}_{xz})$ of resolution $R_p \times R_p$ (i.e. $\boldsymbol{P}_{(*)} \in \mathbb{R}^{R_p \times R_p \times 32}$); 3) tri-plane decoder network $\mathsf{F} : (\boldsymbol{x}, \boldsymbol{P}) \mapsto (\boldsymbol{c}, \sigma) \in \mathbb{R}^4$, which takes the space coordinate $\boldsymbol{x} \in \mathbb{R}^3$ and tri-planes $\boldsymbol{P}$ as input and produces the RGB color $\boldsymbol{c} \in \mathbb{R}^3$ and density value $\sigma \in \mathbb{R}_+$ at that point by interpolating the tri-plane features in the given coordinate and processing them with a tiny MLP. In contrast to classical NeRF [39], we do not utilize view direction conditioning since it worsens multi-view consistency [7] in GANs, which are trained on RGB datasets with a single view per instance. To render a single pixel, we follow the classical volumetric rendering pipeline with hierarchical sampling [39, 7], using 48 ray steps in coarse and 48 in fine sampling stages. See the accompanying source code for more details.

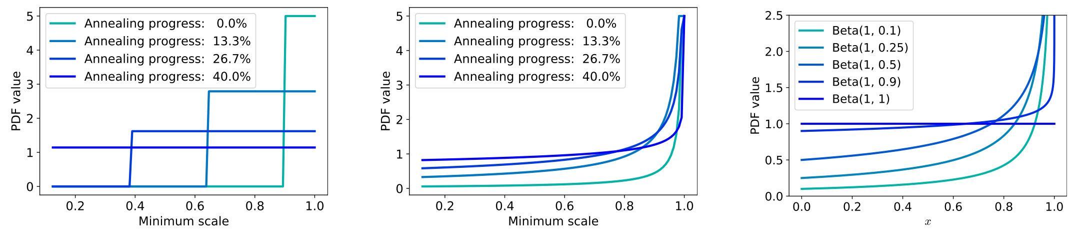


Figure 4: Comparing uniform (left) and beta (middle) annealed patch scale sampling in terms of their probability density function (PDF) (for visualization purposes, we clamp the maximum density value to 5); (right) PDF of Beta$(1, \beta)$, provided for completeness. Uniform distribution with annealed $s_{\min}(0) = 0.9$ from 0.9 to $s_{\min}(T) = 0.125$ does not put any attention to high-frequency details in the beginning and treats small-scale and large-scale patches equally at the end of the annealing. Beta distribution with annealed $\beta(0) \approx 0$ to $\beta(T) \approx 1$, in contrast, lets the model learn high-resolution texture immediately after the training starts, and puts more focus on the structure at the end.

3.2 2D scale/location-aware discriminator

Our discriminator D is built on top of StyleGAN2 [26]. Since we train the model in a patch-wise fashion, the original backbone is not well suited for this: convolutional filters are forced to adapt to signals of very different scales and extracted from different locations. A natural way to resolve this problem is to use separate discriminators depending on the scale, but that strategy has three limitations: 1) each particular discriminator receives less overall training signal (since the batch size is limited); 2) from an engineering perspective, it is more expensive to evaluate a convolutional kernel with different parameters on different inputs; 3) one can use only a small fixed amount of possible patch scales. This is why we develop a novel hypernetwork-modulated [16, 64] discriminator architecture to operate on patches with continuously varying scales.

To modulate the convolutional kernels of D, we define a hypernetwork $\mathsf{H} : (s, \delta_x, \delta_y) :\mapsto (\boldsymbol{\sigma}_1, ..., \boldsymbol{\sigma}_L)$ as a 2-layer MLP with tanh non-linearity at the end which takes patch scale s and its cropping offsets δ_x, δ_y as input and produces modulations $\boldsymbol{\sigma}_\ell \in (0, 2)^{c^\ell_\text{out}}$ (we shift the tanh output by 1 to map into the 1-centered interval), where c^ℓ_out is the number of output channels in the ℓ-th convolutional layer. Given a convolutional kernel $\boldsymbol{W}^\ell \in \mathbb{R}^{c^\ell_\text{out} \times c^\ell_\text{in} \times k \times k}$ and input $\boldsymbol{x} \in \mathbb{R}^{c_\text{in}}$, a straightforward strategy to apply the modulation is to multiply $\boldsymbol{\sigma}$ on the weights (depicting the convolution operation by $\texttt{conv2d}(.)$ and omitting its other parameters for simplicity):

$$\boldsymbol{y} = \texttt{conv2d}(\boldsymbol{W}^\ell \odot \boldsymbol{\sigma}, \boldsymbol{x}), \tag{1}$$

where we broadcast the remaining axes and $\boldsymbol{y} \in \mathbb{R}^{c_\text{out}}$ is the layer output (before the non-linearity). However, using different kernel weights on top of different inputs is inefficient in modern deep learning frameworks (even with the group-wise convolution trick [26]). That's why we use an equivalent strategy of multiplying the weights on $\boldsymbol{x}$ instead:

$$\boldsymbol{y} = \boldsymbol{\sigma} \odot \texttt{conv2d}(\boldsymbol{W}^\ell, \boldsymbol{x}). \tag{2}$$

This suppresses and reinforces different convolutional filters of the layer depending on the patch scale and location. And to incorporate even stronger conditioning, we also use the projection strategy [41] in the final discriminator block. We depict our discriminator architecture in Fig 3. As we show in Tab 2, it allows us to obtain ≈15% lower FID compared to the standard discriminator.

3.3 Patch-wise optimization with Beta-distributed scales

Training NeRF-based GANs is computationally expensive because rendering each pixel via volumetric rendering requires many evaluations (e.g., in our case, 96) of the underlying MLP. For scene reconstruction tasks, it does not create issues since the typically used $\mathcal{L}_2$ loss [39, 78, 71] can be robustly computed on a sparse subset of the pixels. But for NeRF-based GANs, it becomes prohibitively expensive for high resolutions since convolutional discriminators operate on dense full-size images. The currently dominating approach to mitigate this is to train a separate 2D decoder to upsample a low-resolution image representation rendered from a NeRF-based MLP. But this breaks multi-view consistency (i.e., object's shape and texture change when the camera is moving) and learns the 3D geometry in a low resolution (from $\approx 16^2$ [74] to $\approx 128^2$ [6]). This is why we build upon the multi-scale patch-wise training scheme [57] and demonstrate that it can give state-of-the-art image quality and training speed without the above limitations.

Patch-wise optimization works the following way. On each iteration, instead of passing the full-size $R \times R$ image to D, we instead input only a small patch with resolution $r \times r$ of random scale $s \in [r/R, 1]$ and extracted with a random offset $(\delta_x, \delta_y) \in [0, 1-s]^2$. We illustrate this procedure in Fig 3. Patch parameters are sampled from distribution:

$$s, \delta_x, \delta_y \sim p_t(s, \delta_x, \delta_y) \triangleq p_t(s)p(\delta_x|s)p(\delta_y|s) \tag{3}$$

where t is the current training iteration. In this way, patch scales depend on the current training iteration t, and offsets are sampled independently after we know s. As we show next, the choice of distribution $p_t(s)$ has a crucial influence on the learning speed and stability.

Typically, patch scales are sampled from the annealed uniform distribution [57, 37, 5] s:

$$p_t(s) = U[s_{\min}(t), 1], \qquad s_{\min}(t) = \texttt{lerp}\left[1, r/R, \min(t/T, 1)\right], \tag{4}$$

where `lerp` is the linear interpolation function[1], and the left interval bound $s_{\min}(t)$ is gradually annealed during the first T iterations until it reaches the minimum possible value of r/R.[2] But this strategy does not let the model learn high-frequency details early on in training and puts little focus on the structure when $s_{\min}(t)$ is fully annealed to r/R (which is usually very small, e.g., $r/R = 0.125$ for a typical 64^2 patch-wise training on 512^2 resolution). As we show, the first issue makes the generator converge slower, and the second one makes the overall optimization less stable.

To mitigate this, we propose a small change in the pipeline by simply replacing the uniform scale sampling distribution with:

$$s \sim \text{Beta}(1, \beta(t)) \cdot (1 - r/R) + r/R, \tag{5}$$

where $\beta(t)$ is gradually annealed from $\beta(0)$ to some final value $\beta(T)$. Using beta distribution instead of the uniform one gives a very convenient knob to shift the training focus between large patch scales $s \to 1$ (carrying the global information about the whole image) and small patch scales $r \to r/R$ (representing high-resolution local crops).

A natural way to do the annealing is to anneal from 0 to 1: at the start, the model focuses entirely on the structure, while at the end, it transforms into the uniform distribution (See Fig 4). We follow this strategy, but from the design perspective, set $\beta(T)$ to a value that is slightly smaller than 1 (we use $\beta(T) = 0.8$ everywhere) to keep more focus on the structure at the end of the annealing as well. In our initial experiments, $\beta(T) \in [0.7, 1]$ performs similarly. The scales distributions comparison between beta and uniform sampling is provided in Fig 4 and the convergence comparison in Fig 7.

3.4 Training details

We inherit the training procedure from StyleGAN2-ADA [24] with minimal changes. The optimization is performed by Adam [27] with a learning rate of 0.002 and betas of 0 and 0.99 for both G and D. We use $\beta(T) = 0.8$ for $T = 10000$, $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and set $R_p = 512$. D is trained with R1 regularization [38] with $\gamma = 0.05$. We train with the overall batch size of 64 for $\approx$15M images seen by D for 256^2 resolution and $\approx$20M for 512^2. Similar to previous works [6, 12], we use pose supervision for D for the FFHQ and Cats dataset to avoid geometry ambiguity. For this, we take the rotation and elevation angles, encode them with positional embeddings [60, 67] and feed them into a 2-layer MLP. After that, we multiply the obtained vector with the last hidden representation in the discriminator, following the Projection GAN [41] strategy from StyleGAN2-ADA [24]. We train G in full precision and use mixed precision for D. Since FFHQ has too noticeable 3D biases, we use generator pose conditioning for it [6]. Further details can be found in the source code.

4 Experiments

4.1 Experimental setup

Benchmarks. In our study, we consider four benchmarks: 1) FFHQ [25] in 256^2 and 512^2 resolutions, consisting of 70,000 (mostly front-view) human face images; 2) Cats 256^2 [79], consisting of 9,998

[1] $\texttt{lerp}(x, y, \alpha) = (1-\alpha) \cdot x + \alpha \cdot y$ for $x, y \in \mathbb{R}$ and $\alpha \in [0, 1]$.

[2] In practice, those methods use a *very* slightly different distribution (see Appx B)

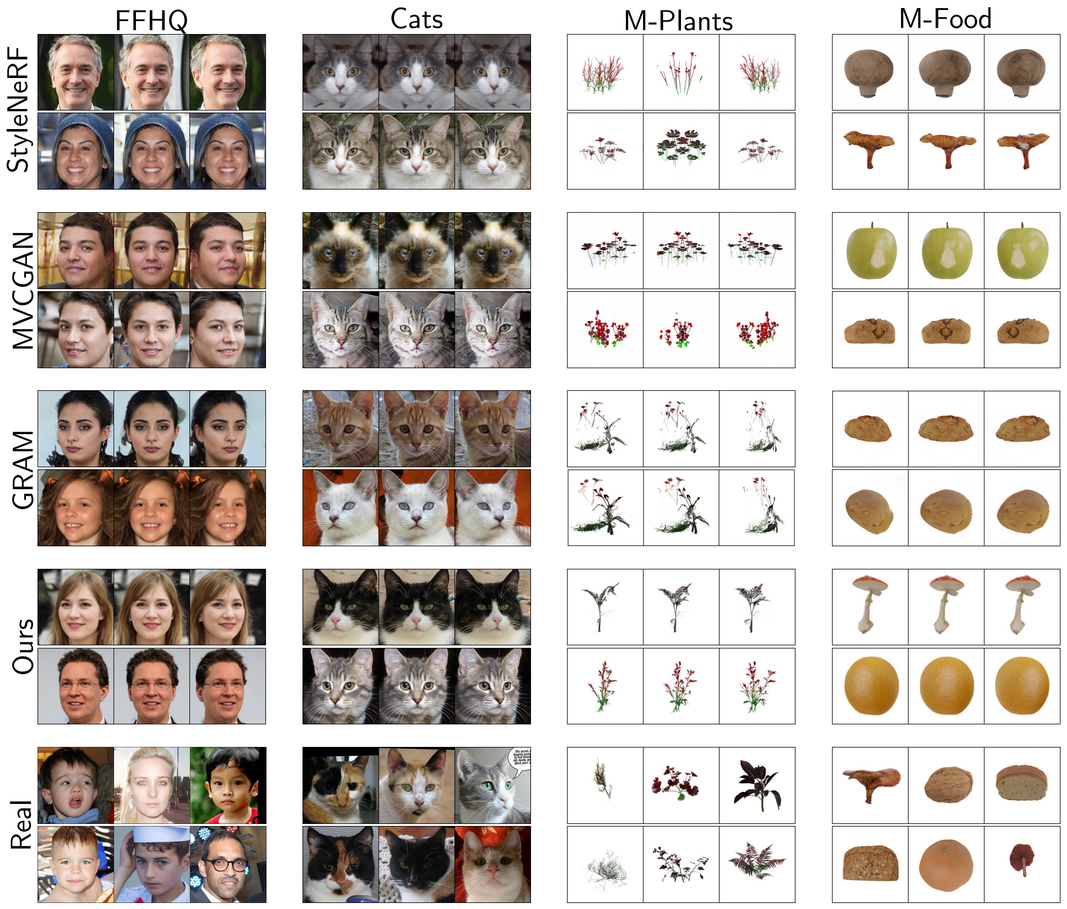


Figure 5: Comparing samples of EpiGRAF and modern 3D-aware generators. Our method attains state-of-the-art image quality, recovers high-fidelity geometry and preserves multi-view consistency for both simple-shape (FFHQ and Cats) and variable-shape (M-Plants and M-Food) datasets. We refer the reader to the supplementary for the video comparisons to evaluate multi-view consistency.

(mostly front-view) cat face images; 3) Megascans Food (M-Food) 256^2 consisting of 231 models of different food items with 128 views per model (25472 images in total); and 4) Megascans Plants (M-Plants) 256^2 consisting of 1166 different plant models with 128 views per model (141824 images in total). The last two datasets are introduced in our work to fix two issues with the modern 3D generation benchmarks. First, existing benchmarks have low variability of global object geometry, focusing entirely on a single class of objects, like human/cat faces or cars, that do not vary much from instance to instance. Second, they all have limited camera pose distribution: for example, FFHQ [25] and Cats [79] are completely dominated by the frontal and near-frontal views (see Appx E). That's why we obtain and render 1307 Megascans models from Quixel, which are photo-realistic (barely distinguishable from real) scans of real-life objects with complex geometry. Those benchmarks and the rendering code will be made publicly available.

Metrics. We use FID [20] to measure image quality and estimate the training cost for each method in terms of NVidia V100 GPU days needed to complete the training process.

Baselines. For upsampler-based baselines, we compare to the following generators: StyleNeRF [15], StyleSDF [48], EG3D [6], VolumeGAN [73], MVCGAN [80] and GIRAFFE-HD [74]. Apart from that, we also compare to pi-GAN [7] and GRAM [12], which are non-upsampler-based GANs. To compare on Megascans, we train StyleNeRF, MVCGAN, pi-GAN, and GRAM from scratch using their official code repositories (obtained online or requested from the authors), using their FFHQ or CARLA hyperparameters, except for the camera distribution and rendering settings. We also train StyleNeRF, MVCGAN, and π-GAN on Cats 256^2. GRAM [12] restricts the sampling space to a set of learnable iso-surfaces, which makes it not well-suited for datasets with varying geometry.

Table 1: FID scores of modern 3D GANs. "†" — evaluated on a re-aligned version of FFHQ (different from original FFHQ [25]). Training cost is measured in terms of NVidia V100 GPU days. "OOM" denotes out-of-memory error.

Method	FFHQ 256^2	FFHQ 512^2	Cats 256^2	M-Plants 256^2	M-Food 256^2	Training cost 256^2	Training cost 512^2	Geometry constraints
StyleNeRF [15]	8.00	7.8	5.91	19.32	16.75	40	56	32^2-res + 2D upsampler
StyleSDF [48]	11.5	11.19	–	–	–	42	56	64^2-res + 2D upsampler
EG3D [6]	4.8†	4.7†	–	–	–	N/A	76	128^2-res + 2D upsampler
VolumeGAN [73]	9.1	–	–	–	–	N/A	N/A	64^2-res + 2D upsampler
MVCGAN [80]	13.7	13.4	39.16	31.70	29.29	42	64	64^2-res + 2D upsampler
GIRAFFE-HD [74]	11.93	–	12.36	–	–	N/A	N/A	16^2-res + 2D upsampler
pi-GAN [7]	53.2	OOM	68.28	75.64	51.99	56	∞	none
GRAM [12]	13.78	OOM	13.40	188.6	178.9	56	∞	iso-surfaces
EpiGRAF (ours)	9.71	9.92	6.93	19.42	18.15	16	24	none

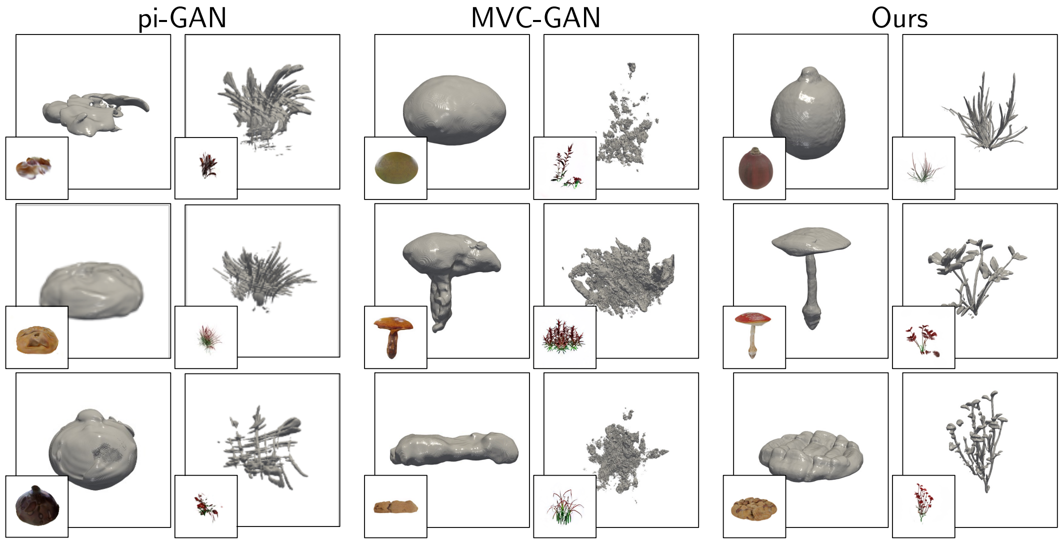


Figure 6: Visualizing the learned geometry for different methods. π-GAN [7] recovers high-fidelity shapes, but has worse image quality (see Table 1) and is much more expensive to train than our model. MVC-GAN [80] fails to capture good geometry because of the 2D upsampler. Our method learns proper geometry and achieves state-of-the-art image quality. We extracted the surfaces using marching cubes from the density fields sampled on 256^3 grid and visualized them in PyVista [65]. We manually optimized the marching cubes contouring threshold for each checkpoint of each method. We noticed that π-GAN [7] produces a lot of "spurious" density which makes.

4.2 Results

EpiGRAF achieves state-of-the-art image quality. For Cats 256^2, M-Plants 256^2 and M-Food 256^2, EpiGRAF outperforms all the baselines in terms of FID except for StyleNeRF, performing very similar to it on all the datasets even though it does not have a 2D upsampler. For FFHQ, our model attains very similar FID scores as the other methods, ranking 4/9 (including older π-GAN [7]), noticeably losing only to EG3D [6], which trains and evaluates on a different version of FFHQ and uses pose conditioning in the generator (which potentially improves FID at the cost of multi-view consistency). We provide a visual comparison for different methods in Fig 5.

EpiGRAF is much faster to train. As reported in Tab 1, existing methods typically train for ≈1 week on 8 V100s, EpiGRAF finishes training in just 2 days for 256^2 and 3 days for 512^2 resolutions, which is $2-3\times$ faster. Note that this high training efficiency is achieved without using an upsampler, which initially enabled the high-resolution synthesis of 3D-aware GANs. As to the non-upsampler methods, we couldn't train GRAM or π-GAN on 512^2 resolution due to the memory limitations of the setup with 8 NVidia V100 32GB GPUs (i.e., 256GB of GPU memory in total).

EpiGRAF learns high-fidelity geometry. Using a pure NeRF-based backbone has two crucial benefits: it provides multi-view consistency and allows learning the geometry in the full dataset resolution. In Fig 6, we visualize the learned shapes on M-Food and M-Plants for 1) π-GAN: a pure NeRF-based generator without the geometry constraints; 2) MVC-GAN [80]: an upsampler-based generator with strong multi-view consistency regularization; 3) our model. We provide the details and analysis in the caption of Fig 6. We also provide the geometry comparison with EG3D on FFHQ 512^2 in Fig 2.

EpiGRAF easily capitalizes on techniques from the NeRF literature. Since our generator is purely NeRF based and renders images without a 2D upsampler, it is well coupled with the existing techniques from the NeRF scene reconstruction field. To demonstrate this, we adopted background separation from NeRF++ [78] using the inverse sphere parametrization by simply copy-pasting the corresponding code from their repo. We depict the results in Fig 1 and provide the details in Appx B.

4.3 Ablations

We report the ablations for different discriminator architectures and patch sizes on FFHQ 512^2 and M-Plants 256^2 in Tab 2. Using a traditional discriminator architecture results in $\approx 15\%$ worse performance. Using several ones (via the group-wise convolution trick [26]) results in a noticeably slower training time and dramatically degrades the image quality. We hypothesize that the reason for it was the reduced overall training signal each discriminator receives, which we tried to alleviate by increasing their learning rate, but that did not improve the results. A too-small patch size hampers the learning process and produces a $\approx 80\%$ worse FID. A too-large one provides decent image quality but greatly reduces the training speed. Using a single scale/position-aware discriminator achieves the best performance, outperforming the standard one by $\approx 15\%$ on average.

To assess the convergence of our proposed patch sampling scheme, we compared against uniform sampling on Cats 256^2 for $T \in \{1000, 5000, 10000\}$, representing different annealing speeds. We show the results for it in Fig 7: our proposed beta scale sampling strategy with $T = 10\text{k}$ schedule robustly converges to lower values than the uniform one with $T = 5\text{k}$ or $T = 10\text{k}$ and does not fluctuate much compared to the $T = 1k$ uniform one (where the model reached its final annealing stage in just 1k kilo-images seen by D).

To analyze how hyper-modulation manipulates the convolutional filters of the discriminator, we visualize the modulation weights $\boldsymbol{\sigma}$, predicted by H, in Fig 8 (see the caption for the details). These visualizations show that some of the filters are always switched on, regardless of the patch scale; while others are always switched off, providing potential room for pruning [18]. And $\approx 40\%$ of the filters are getting switched on and off depending on the patch scale, which shows that H indeed learns to perform meaningful modulation.

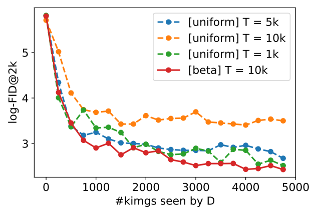


Figure 7: Convergence comparison on Cats 256^2 for different sampling strategies.

Experiment	FFHQ 512^2	M-Plants 256^2	Training cost
GRAF (with tri-planes)	13.41	24.99	24
+ beta scale sampling ($T = 5k$)	11.57	21.77	24
+ 2 scale-specific D-s	10.87	21.02	28
+ 4 scale-specific D-s	21.56	43.11	28
+ 1 scale/position-aware D	9.92	19.42	24
– 32^2 patch resolution	17.44	34.32	19
– 64^2 patch resolution (default)	9.92	19.42	24
– 128^2 patch resolution	11.36	18.90	34

Table 2: Ablating the discriminator architecture and patch sizes in terms of FID scores and training cost (V100 GPU days).

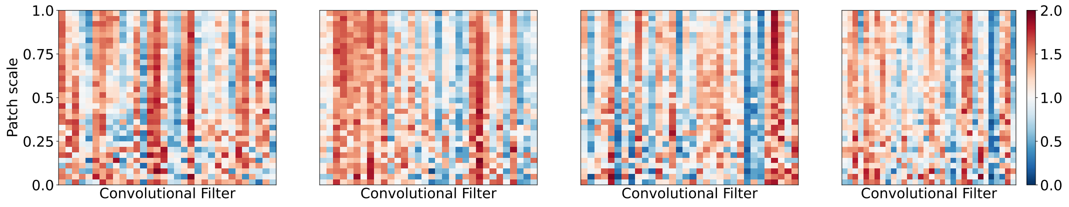


Figure 8: Visualizing modulation weights $\boldsymbol{\sigma}$, predicted by H for 2-nd, 6-th, 10-th and 14-th convolutional layers. Each subplot denotes a separate layer and we visualize random 32 filters for it.

5 Limitations

Performance drop for 2D generation. Before switching to training 3D-aware generators, we spent a considerable amount of time, exploring our ideas on top of StyleGAN2 [24] for traditional 2D generation since it is faster, less error-prone and more robust to a hyperparameters choice. What we observed is that despite our best efforts (see C) and even with longer training, we couldn't obtain the same image quality as the full-resolution StyleGAN2 generator.

Table 3: Trying to train a traditional StyleGAN2 [26] generator in the patch-wise fashion. We tried to train longer to compensate for a smaller learning signal overall (a 64^2 patch is $1/64$ of information compared to a 512^2 image), but this didn't allow to catch up. Note, however, that AnyResGAN [5] reaches SotA when training on 256^2 patches compared to 1024^2 images.

Method	FFHQ 512^2		LSUN Bedroom 256^2	
	FID	Training cost	FID	Training cost
StyleGAN2-ADA [24]	3.83	8	4.12	5
+ multi-scale 64^2 patch-wise training	7.11	6	6.73	4
+ $\times 2$ longer training	5.71	12	5.42	8
+ $\times 4$ longer training	4.76	24	4.31	16

A range of possible patch sizes is restricted. Tab 2 shows the performance drop when using the 32^2 patch size instead of the default 64^2 one without any dramatic improvement in speed. Trying to decrease it further would produce even worse performance (imagine training in the extreme case of 2^2 patches). Increasing the patch size is also not desirable since it decreases the training speed a lot: going from 64^2 to 128^2 resulted in 30% cost increase without clear performance benefits. In this way, we are very constrained in what patch size one can use.

Discriminator does not see the global context. When the discriminator classifies patches of small scale, it is forced to do so without relying on the global image information, which could be useful for this. Our attempts to incorporate it (see Appx C) did not improve the performance (though we believe we under-explored this).

Low-resolution artifacts. While our generator achieves good FID on FFHQ 512^2, we noticed that it has some blurriness when one zooms-in into the samples. It is not well captured by FID since it always resizes images to the 299×299 resolution. We attribute this problem to our patch-wise training scheme, which puts too much focus on the structure and believe that it could be resolved.

6 Conclusion

In this work, we showed that it is possible to build a state-of-the-art 3D GAN framework without a 2D upsampler, but using a pure NeRF-based generator trained in a multi-scale patch-wise fashion. For this, we improved the traditional patch-wise training scheme in two important ways. First, we proposed to use a scale/location-aware discriminator with convolutional filters modulated by a hypernetwork depending on the patch parameters. Second, we developed a schedule for patch scale sampling based on the beta distribution, that leads to faster and more robust convergence. We believe that the future of 3D GANs is a combination of efficient volumetric representations, regularized 2D upsamplers, and patch-wise training. We propose this avenue of research for future work.

Our method also has several limitations. Before switching to training 3D-aware generators, we spent a considerable amount of time exploring our ideas on top of StyleGAN2 for traditional 2D generation, which always resulted in higher FID scores. Further, the discriminator loses information about global context. We tried multiple ideas to incorporate global context, but it did not lead to an improvement. Next, our current patch-wise training scheme might cause some low-res artifacts. Finally, 3D GANs generating faces and humans may have negative societal impact as discussed in Appx H.

7 Acknowledgements

We would like to acknowledge support from the SDAIA-KAUST Center of Excellence in Data Science and Artificial Intelligence.

References

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018.

[2] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

[3] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2022.

[4] A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[5] L. Chai, M. Gharbi, E. Shechtman, P. Isola, and R. Zhang. Any-resolution training for high-resolution image synthesis. *arXiv preprint arXiv:2204.07156*, 2022.

[6] E. R. Chan, C. Z. Lin, M. A. Chan, K. Nagano, B. Pan, S. D. Mello, O. Gallo, L. Guibas, J. Tremblay, S. Khamis, T. Karras, and G. Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *arXiv*, 2021.

[7] E. R. Chan, M. Monteiro, P. Kellnhofer, J. Wu, and G. Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5799–5809, 2021.

[8] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su. Tensorf: Tensorial radiance fields. *arXiv preprint arXiv:2203.09517*, 2022.

[9] H. Chen, B. He, H. Wang, Y. Ren, S.-N. Lim, and A. Shrivastava. Nerv: Neural representations for videos. *arXiv preprint arXiv:2110.13903*, 2021.

[10] Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019.

[11] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797, 2018.

[12] Y. Deng, J. Yang, J. Xiang, and X. Tong. Gram: Generative radiance manifolds for 3d-aware image generation. In *IEEE Computer Vision and Pattern Recognition*, 2022.

[13] M. Gadelha, S. Maji, and R. Wang. 3d shape induction from 2d views of multiple objects. In *2017 International Conference on 3D Vision (3DV)*, pages 402–411. IEEE, 2017.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[15] J. Gu, L. Liu, P. Wang, and C. Theobalt. Stylenerf: A style-based 3d aware generator for high-resolution image synthesis. In *International Conference on Learning Representations*, 2022.

[16] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[17] Z. Hao, A. Mallya, S. Belongie, and M.-Y. Liu. GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds. In *ICCV*, 2021.

[18] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[19] P. Henderson, V. Tsiminaki, and C. H. Lampert. Leveraging 2d data to learn textured 3d mesh generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7498–7507, 2020.

[20] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[21] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[22] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[23] K. Jo, G. Shim, S. Jung, S. Yang, and J. Choo. Cg-nerf: Conditional generative neural radiance fields. *arXiv preprint arXiv:2112.03517*, 2021.

[24] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila. Training generative adversarial networks with limited data. *arXiv preprint arXiv:2006.06676*, 2020.

[25] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[26] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

[29] S. Kobayashi, E. Matsumoto, and V. Sitzmann. Decomposing nerf for editing via feature field distillation. *arXiv preprint arXiv:2205.15585*, 2022.

[30] A. R. Kosiorek, H. Strathmann, D. Zoran, P. Moreno, R. Schneider, S. Mokrá, and D. J. Rezende. Nerf-vae: A geometry aware 3d scene generative model. *arXiv preprint arXiv:2104.00587*, 2021.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[32] R. Li, X. Li, K.-H. Hui, and C.-W. Fu. Sp-gan: Sphere-guided 3d shape generation and manipulation. *ACM Transactions on Graphics (TOG)*, 40(4):1–12, 2021.

[33] X. Li, Y. Dong, P. Peers, and X. Tong. Synthesizing 3d shapes from silhouette image collections using multi-projection generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5535–5544, 2019.

[34] C. H. Lin, H.-Y. Lee, Y.-C. Cheng, S. Tulyakov, and M.-H. Yang. Infinitygan: Towards infinite-resolution image synthesis. *arXiv preprint arXiv:2104.03963*, 2021.

[35] S. Luo and W. Hu. Diffusion probabilistic models for 3d point cloud generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021.

[36] Y. A. Mejjati, I. Milefchik, A. Gokaslan, O. Wang, K. I. Kim, and J. Tompkin. Gaussigan: Controllable image synthesis with 3d gaussians from unposed silhouettes. *arXiv preprint arXiv:2106.13215*, 2021.

[37] Q. Meng, A. Chen, H. Luo, M. Wu, H. Su, L. Xu, X. He, and J. Yu. Gnerf: Gan-based neural radiance field without posed camera. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6351–6361, 2021.

[38] L. Mescheder, A. Geiger, and S. Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018.

[39] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[40] P. Mittal, Y.-C. Cheng, M. Singh, and S. Tulsiani. AutoSDF: Shape priors for 3d completion, reconstruction and generation. In *CVPR*, 2022.

[41] T. Miyato and M. Koyama. cgans with projection discriminator. *arXiv preprint arXiv:1802.05637*, 2018.

[42] T. Nguyen-Phuoc, C. Li, L. Theis, C. Richardt, and Y.-L. Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *The IEEE International Conference on Computer Vision (ICCV)*, Nov 2019.

[43] M. Niemeyer and A. Geiger. Campari: Camera-aware decomposed generative neural radiance fields. In *2021 International Conference on 3D Vision (3DV)*, pages 951–961. IEEE, 2021.

[44] M. Niemeyer and A. Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11453–11464, 2021.

[45] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020.

[46] M. Oechsle, S. Peng, and A. Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5589–5599, 2021.

[47] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[48] R. Or-El, X. Luo, M. Shan, E. Shechtman, J. J. Park, and I. Kemelmacher-Shlizerman. StyleSDF: High-Resolution 3D-Consistent Image and Geometry Generation. *arXiv preprint arXiv:2112.11427*, 2021.

[49] X. Pan, B. Dai, Z. Liu, C. C. Loy, and P. Luo. Do 2d gans know 3d shape? unsupervised 3d shape reconstruction from 2d image gans. *arXiv preprint arXiv:2011.00844*, 2020.

[50] X. Pan, X. Xu, C. C. Loy, C. Theobalt, and B. Dai. A shading-guided generative implicit model for shape-accurate 3d-aware image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[51] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla. Deformable neural radiance fields. *arXiv preprint arXiv:2011.12948*, 2020.

[52] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2337–2346, 2019.

[53] T. Park, J.-Y. Zhu, O. Wang, J. Lu, E. Shechtman, A. Efros, and R. Zhang. Swapping autoencoder for deep image manipulation. *Advances in Neural Information Processing Systems*, 33:7198–7211, 2020.

[54] D. Pavllo, J. Kohler, T. Hofmann, and A. Lucchi. Learning generative models of textured 3d meshes from real-world images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 13879–13889, October 2021.

[55] D. Pavllo, G. Spinks, T. Hofmann, M.-F. Moens, and A. Lucchi. Convolutional generation of textured 3d meshes. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[56] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[57] K. Schwarz, Y. Liao, M. Niemeyer, and A. Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[58] T. R. Shaham, T. Dekel, and T. Michaeli. Singan: Learning a generative model from a single natural image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4570–4580, 2019.

[59] Y. Shi, D. Aggarwal, and A. K. Jain. Lifting 2d stylegan for 3d-aware face generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6258–6266, 2021.

[60] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[61] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019.

[62] I. Skorokhodov, S. Ignatyev, and M. Elhoseiny. Adversarial generation of continuous images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10753–10764, 2021.

[63] I. Skorokhodov, G. Sotnikov, and M. Elhoseiny. Aligning latent and image spaces to connect the unconnectable. *arXiv preprint arXiv:2104.06954*, 2021.

[64] I. Skorokhodov, S. Tulyakov, and M. Elhoseiny. Stylegan-v: A continuous video generator with the price, image quality and perks of stylegan2. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3626–3636, 2022.

[65] C. B. Sullivan and A. Kaszynski. PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, may 2019.

[66] F. Tan, S. Fanello, A. Meka, S. Orts-Escolano, D. Tang, R. Pandey, J. Taylor, P. Tan, and Y. Zhang. Volux-gan: A generative model for 3d face synthesis with hdri relighting. *arXiv preprint arXiv:2201.04873*, 2022.

[67] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.

[68] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

[69] Y. Vinker, E. Horwitz, N. Zabari, and Y. Hoshen. Image shape manipulation from a single augmented training sample. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 13769–13778, October 2021.

[70] C. Wang, M. Chai, M. He, D. Chen, and J. Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. *arXiv preprint arXiv:2112.05139*, 2021.

[71] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021.

[72] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[73] Y. Xu, S. Peng, C. Yang, Y. Shen, and B. Zhou. 3d-aware image synthesis via learning structural and textural representations. *arXiv preprint arXiv:2112.10759*, 2021.

[74] Y. Xue, Y. Li, K. K. Singh, and Y. J. Lee. Giraffe hd: A high-resolution 3d-aware generative model. *arXiv preprint arXiv:2203.14954*, 2022.

[75] Y. Ye, S. Tulsiani, and A. Gupta. Shelf-supervised mesh prediction in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8843–8852, June 2021.

[76] A. Yu, V. Ye, M. Tancik, and A. Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4578–4587, June 2021.

[77] J. Zhang, E. Sangineto, H. Tang, A. Siarohin, Z. Zhong, N. Sebe, and W. Wang. 3d-aware semantic-guided generative model for human synthesis. *arXiv preprint arXiv:2112.01422*, 2021.

[78] K. Zhang, G. Riegler, N. Snavely, and V. Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020.

[79] W. Zhang, J. Sun, and X. Tang. Cat head detection-how to effectively exploit shape and texture features. In *European conference on computer vision*, pages 802–816. Springer, 2008.

[80] X. Zhang, Z. Zheng, D. Gao, B. Zhang, P. Pan, and Y. Yang. Multi-view consistent generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

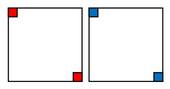
[81] P. Zhou, L. Xie, B. Ni, and Q. Tian. Cips-3d: A 3d-aware generator of gans based on conditionally-independent pixel synthesis. *arXiv preprint arXiv:2110.09788*, 2021.

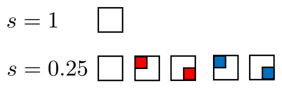
Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See §5 and Appx A.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We do this in Appendix G.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We discuss the potential ethical concerns of using our model in Appendix G.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide the code/data and additional visualizations on https://universome.github.io/epigraf(as specified in the introduction).
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide the most important training details in §3.4. The rest of the details are provided in Appx B and the provided source code.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] . That's too computationally expensive and single-run results are typically reliable in the GAN field.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We report this numbers in Appx B.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite all the sources of the datasets which were used or mentioned in our submission.
   (b) Did you mention the license of the assets? [Yes] In this work, we release two new datasets: Megascans Plants and Megascans Food. We discuss their licensing in Appx E.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide our datasets on the project website: https://universome.github.io/epigraf.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] We specify the information on dataset collection in Appx E.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] As discussed in Appx E, the released data does not contain personally identifiable information or offensive content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Additional limitations due to aliasing

Current patch-wise training strategies (both ours and in prior works [57, 37]) do not take aliasing into account when extracting patches. This leads to additional problems, which we illustrate in Figure 9. Basically, sampling patches from images (when performed naively) is prone to aliasing, which can potentially result in learning an incorrect distribution.


(a) Training dataset



(b) Patches seen during training

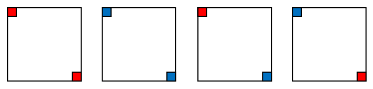

(c) Learnt distribution

Figure 9: Illustrating fundamental limitations of a non-anti-aliased patch-wise training scheme on a toy example. Consider a simple dataset consisting of just two images, illustrated on Figure 9a, which are mostly white except for the colored corners. Consider a generator trained with the patch-wise scheme without taking anti-aliasing into account, where we sample patches of just two scales: $s = 1$ and $s = 0.25$. Then, during training it would only see patches as depicted in Figure 9b, where color correlation in the corners would not be noticeable since large-scale patches will lose such find-grained details during aliased down-sampling (as in both our and previous patch-wise training schemes [57, 37]). As a result, the generator will learn to generate images without taking correlations between high-frequency details into account, as illustrated in Figure 9c.

We can put this more rigorously. ‘ Imagine that we have a distribution $p(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathbb{R}^{R \times R}$, where we consider images to be single-channel (for simplicity) and R is the image size. If we train with patch size of r, then each optimization step uses random $r \times r$ pixels out of $R \times R$, i.e. we optimize over the distribution of all possible marginals $p(\boldsymbol{x}_p)$ where $\boldsymbol{x}_p = e(\boldsymbol{x}; \xi) \in \mathbb{R}^{r \times r}$ is an image patch and $e(\boldsymbol{x}, \xi)$ is an aliased patch extraction function with nearest neighbor interpolation and random seed ξ.[3] This means that our minimax objective becomes:

$$\min_{\mathsf{G}} \max_{\mathsf{D}} \mathbb{E}_{p(\boldsymbol{x}_p)}[\log \mathsf{D}(\boldsymbol{x}_p)] + \mathbb{E}_{p(\boldsymbol{z}),p(\xi)}[\log(1 - \mathsf{D}(e(\mathsf{G}(\boldsymbol{z}), \xi)))] \tag{6}$$

If we rely on the GAN convergence theorem [14], stating that we recover the training distribution as the solution of the minimax problem, then G will learn to approximate all the possible marginal distributions $p(\boldsymbol{x}_p)$ instead of the full joint distribution $p(\boldsymbol{x})$, that we seek.

Sampling patches in an alias-free manner is tricky for our generator, since we cannot obtain intermediate high-resolution representations (which are needed to address aliasing) from tri-planes due to the computational overhead. We leave the development of patch sampling schemes for NeRF-based generators for future work.

B Training details

B.1 Hyper-parameters and optimization details

We inherit most of the hyperparameters from the StyleGAN2-ADA repo [24] repo which we build on top[4]. In this way, we use the dimensionalities of 512 for both $\boldsymbol{z}$ and $\boldsymbol{w}$. The mapping network has 2 layers of dimensionality 512 with LeakyReLU non-linearities with the negative slope of -0.2. Synthesis network S produced three 512^2 planes of 32 channels each. We use the SoftPlus non-linearity instead of typically used ReLU [39] as a way to clamp the volumetric density. Similar to π-GAN, we also randomize

For FFHQ and Cats, we also use camera conditioning in D. For this, we encode yaw and pitch angles (roll is always set to 0) with Fourier positional encoding [60, 67], apply dropout with 0.5 probability (otherwise, D can start judging generations from 3D biases in the dataset, hurting the image quality), pass through a 2-layer MLP with LeakyReLU activations to obtain a 512-dimensional vector, which is finally as a projection conditioning [41]. Cameras positions were extracted in the same way as in GRAM [12].

[3]For brevity, we “hide” all the randomness of the patch sampling process into ξ.
[4]https://github.com/NVlabs/stylegan2-ada-pytorch

We optimize both G and D with the batch size of 64 until D sees 25,000,000 real images, which is the default setting from StyleGAN2-ADA. We the default setup of adaptive augmentations, except for random horizontal flipping, since it would require the corresponding change in the yaw angle at augmentation time, which was not convenient to incorporate from the engineering perspective. Instead, random horizontal flipping is used non-adaptively as a dataset mirroring where flipping the yaw angles is more accessible. We train G in full precision, while D uses mixed precision.

Hypernetwork H is structured very similar to the generator's mapping network. It consists of 2 layers with LeakyReLU non-linearities with the negative slope of -0.2. Its input is the positional embedding of the patch scales and offsets s, δ_x, δ_y, encoded with Fourier features [60, 67] and concatenated into a single vector of dimensionality 828. It produces a patch representation vector $\boldsymbol{p} \in \mathbb{R}^{512}$, which is then adapted for each convolutional layer via:

$$\boldsymbol{\sigma} = \tanh(\boldsymbol{W}_\ell \boldsymbol{p} + \boldsymbol{b}_\ell) + 1, \tag{7}$$

where $\boldsymbol{\sigma} \in [0, 2]^{c^\ell_\text{out}}$ is the modulation vector, $(\boldsymbol{W}_\ell, \boldsymbol{b}_\ell)$ is the layer-specific affine transformation, c^ℓ_out is the amount of output filters in the ℓ-th layer. In this way, H has layer-specific adapters.

For the background separation experiment, we adapt the neural representation MLP from INR-GAN [62], but passing 4 coordinates (for the inverse sphere parametrization [78]) instead of 2 as an input. It consists of 2 blocks with 2 linear layers each. We use 16 steps per ray for the background without hierarchical sampling.

Further details could be found in the accompanying source code.

B.2 Utilized computational resources

While developing our model, we had been launching experiments on $4\times$ NVidia A100 81GB or Nvidia V100 32GB GPUs with the AMD EPYC 7713P 64-Core processor. We found that in practice, running the model on A100s gives a $2\times$ speed-up compared to V100s due to the possibility of increasing the batch size from 32 to 64. In this way, training EpiGRAF on $4\times$ A100s gives the same training speed as training it $8\times$ V100s.

For the baselines, we were running them on 4-8$\times$ V100s GPUs as was specified by the original papers unless the model could fit into 4 V100s without decreasing the batch size (it was only possible for StyleNeRF [15]).

For rendering Megascans, we used $4\times$ NVIDIA TITAN RTX with 24GB memory each. But resource utilization for rendering is negligible compared to training the generators.

In total, the project consumed $\approx$4 A100s GPU-years, $\approx$4 V100s GPU-years, and $\approx$20 TITAN RTX GPU-days. Note, that out of this time, training the baselines consumed $\approx$1.5 V100s GPU-years.

B.3 Annealing schedule details

As being said in §3.3, the existing multi-scale patch-wise generators [57, 37] use uniform distribution $U[s_{\min(t)}, 1]$ to sample patch scales, where $s_{\min(t)}$ is gradually annealed during training from 0.9 (or 0.8 [37]) to r/R with different speeds. We visualize the annealing schedule for both GRAF and GNeRF on Fig 10, which demonstrates that their schedules are very close to `lerp`-based one, described in §3.3.

C Failed experiments

Modern GANs are a lot of engineering and it often takes a lot of futile experiments to get to a point where the obtained performance is acceptable. We want to enumerate some experiments which did not work out (despite looking like they should work) — either because the idea was fundamentally flawed on its own or because we've under-explored it (or both).

Conditioning D on global context worsened the performance. In Appx 5, we argued that when D processes a small-scale patch, it does not have access to the global image information, which might be a source of decreased image quality. We tried several strategies to compensate for this. Our first attempt was to generate a low-resolution image, bilinearly upsample it to the target size, and then

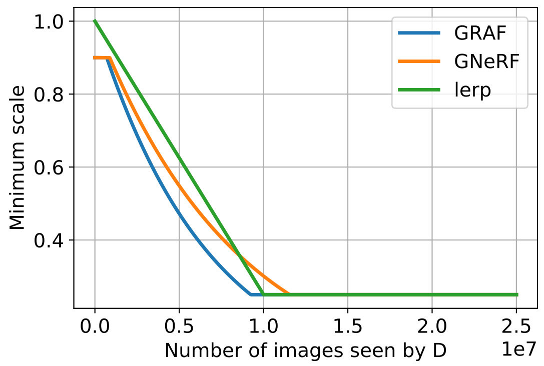


Figure 10: Comparing annealing schedules for GRAF [57], GNeRF [37] and the `lerp`-based schedule from §3.3. We simplified the exposition by stating that GRAF and GNeRF use the `lerp`-based schedule, which is *very* close to reality.

"grid paste" a high-resolution patch into it. The second attempt was to simply always concatenate a low-resolution version of an image as 3 additional channels. However, in both cases, generator learned to produce low-resolution version of images well, but the texture was poor. We hypothesize that it was due to D starting to produce its prediction almost entirely based on the low-resolution image, ignoring the high-resolution patches since they are harder to discriminate.

Patch importance sampling did not work. Almost all the datasets used for 3D-aware image synthesis have regions of difficult content and regions with simpler content — it is especially noticeable for CARLA [57] and our Megascans datasets, which contain a lot of white background. That's why, patch-wise sampling could be improved if we sample patches from the more difficult regions more frequently. We tried this strategy in the GNeRF [37] problem setup on the NeRF-Synthetic dataset [39] of fitting a scene without known camera parameters. We sampled patches from regions with high average gradient norm more frequently. For some scenes, it helped, for other ones, it worsened the performance.

View direction conditioning breaks multi-view consistency. Similar to the prior works [57, 7], our attempt to condition the radiance (but not density) MLP on ray direction (similar to NeRF [39]) led to poor multi-view consistency with radiance changing with camera moving. We tested this on FFHQ [25], which has only a single view per object instance and suspect that it wouldn't be happening on Megascans, where view coverage is very rich.

Tri-planes produced from convolutional layers are much harder to optimize for reconstruction. While debugging our tri-plane representation, we found that tri-planes produced with convolutional layers are extremely difficult to optimize for reconstruction. I.e., if one fits a 3D scene while optimizing tri-planes directly, then everything goes smoothly, but when those tri-planes are being produced by the synthesis network of StyleGAN2 [26], then PNSR scores (and the loss values) are plateauing very soon.

D Additional patch size ablation

Table 2 shows that the generator achieves the best performance for the 64^2 patch resolution. The comparison between patch sizes was performed while keeping all other hyperparameters fixed. This creates an issue since StyleGAN-based generators should use different values for the R1 regularization weight γ depending on the training resolutions.[5] This is why in Table 4, we provide the results for a 3×3 grid search over patch sizes and R1 regularization gamma.

And this better aligns with intuition: increasing the patch size should improve the performance (at the loss of the training speed) since the model uses more information during training. The main reason why we fixed the patch resolution to 64^2 is because we considered the computational overhead not to be worth the quality improvements it brings: while it is not expensive to run several individual

[5] See https://github.com/NVlabs/stylegan2-ada-pytorch/blob/main/train.py#L173.

Table 4: FID@2k scores for the grid search over the patch size and R1 regularization weight γ on Cats 256^2. Each model was trained for 5M seen images and FID was measured using 2048 fake and all the real images.

Patch size	$\gamma = 0.01$	$\gamma = 0.1$	$\gamma = 1$	Training speed
32^2	20.31	30.72	335.32	9.84 seconds / 1K images
64^2	24.93	18.13	20.07	11.36 seconds / 1K images
128^2	21.21	18.72	16.96	17.45 seconds / 1K images

experiments with the 128^2 patch resolution, it is expensive to develop the whole project around the 128^2 patch resolution generator.

E Datasets details

E.1 Megascans dataset

Modern 3D-aware image synthesis benchmarks have two issues: 1) they contain objects of very similar global geometry (like, human or cat faces, cars and chairs), and 2) they have poor camera coverage. Moreover, some of them (e.g., FFHQ), contain 3D-biases, when an object features (e.g., smiling probability, gaze direction, posture or haircut) correlate with the camera position [6]. As a result, this does not allow to evaluate a model's ability to represent the underlying geometry and makes it harder to understand whether performance comes from methodological changes or better data preprocessing.

To mitigate these issues, we introduce two new datasets: Megascans Plants (M-Plants) and Megascans Food (M-Food). To build them, we obtain $\approx 1,500$ models from Quixel Megascans[6] from Plants, Mushrooms and Food categories. Megascans are very high-quality scans of real objects which are almost indistinguishable from real. For Mushrooms and Plants, we merge them into the same Food category since they have too few models on their own.

We render all the models in Blender [3] with cameras, distributed uniformly at random over the sphere of radius 3.5 and field-of-view of $\pi/4$. While rendering, we scale each model into $[-1, 1]^3$ cube and discard those models, which has the dimension produce of less than 2. We render 128 views per object from a fixed distance to the object center from uniformly sampled points on the entire sphere (even from below). For M-Plants, we additionally remove those models which have less than 0.03 pixel intensity on average (computed as the mean alpha value over the pixels and views). This is needed to remove small grass or leaves which will be occupying a too small amount of pixels. As a result, this procedure produces 1,108 models for the Plants category and 199 models for the Food category.

We include the rendering script as a part of the released source code. We cannot release the source models or textures due to the copyright restrictions. We release all the images under the CC BY-NC-SA 4.0 license[7]. Apart from the images, we also release the class categories for both M-Plants and M-Food.

The released datasets do not contain any personally identifiable information or offensive content since it does not have any human subjects, animals or other creatures with scientifically proven cognitive abilities. One concern that might arise is the inclusion of Amanita muscaria[8] into the Megascans Food dataset, which is poisonous (when consumed by ingestion without any specific preparation). This is why we urge the reader not to treat the included objects as edible items, even though they are a part of the "food" category. We provide random samples from both of them in Fig 11 and Fig 12. Note that they are almost indistinguishable from real objects.

[6] https://quixel.com/megascans
[7] https://creativecommons.org/licenses/by-nc-sa/4.0
[8] https://en.wikipedia.org/wiki/Amanita_muscaria

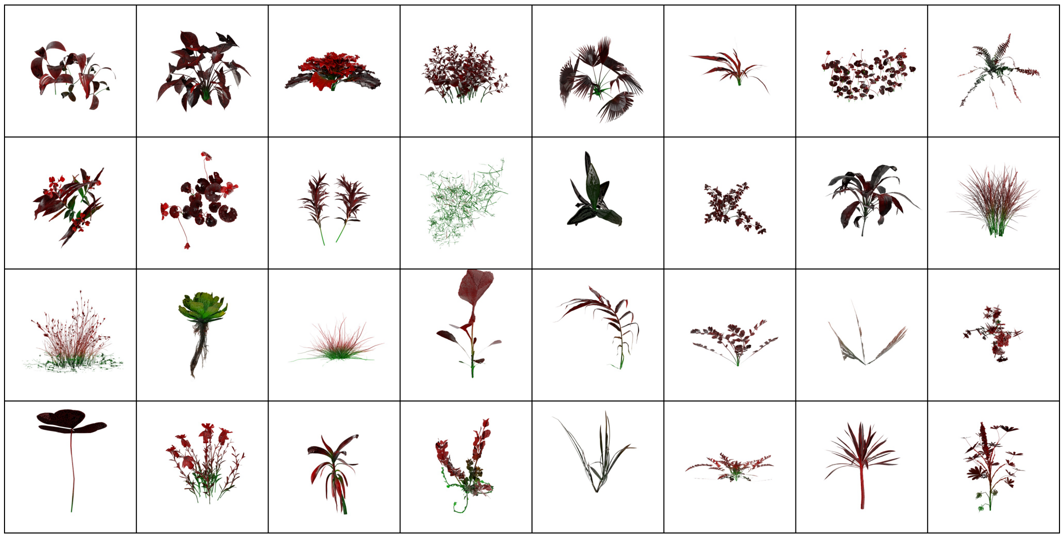


Figure 11: Real images from the Megascans Plants dataset. This dataset contains very complex geometry and texture, while having good camera coverage.

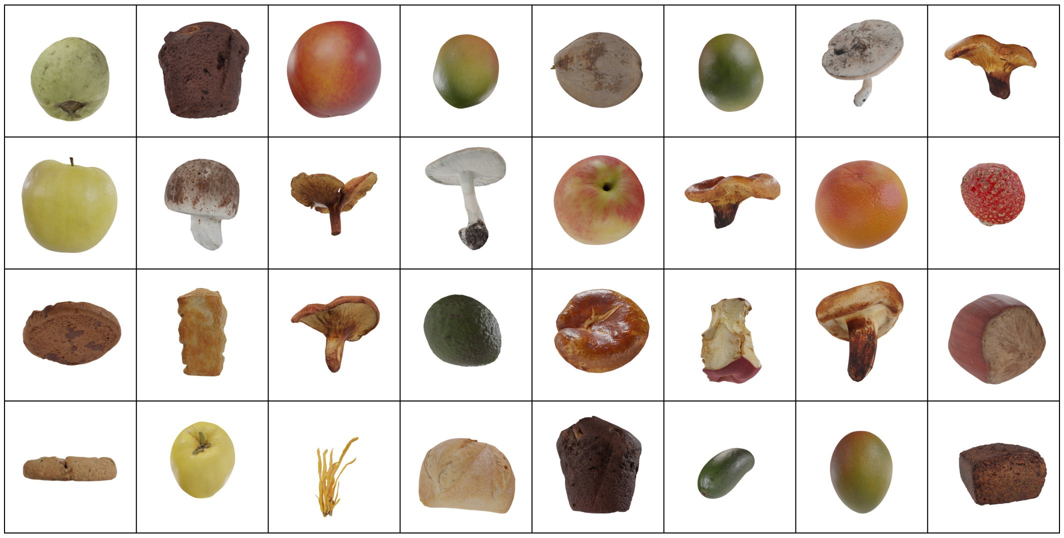
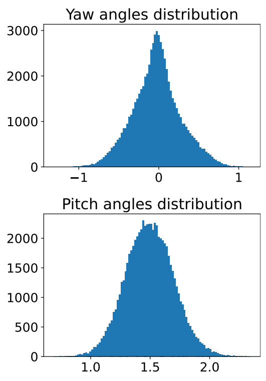


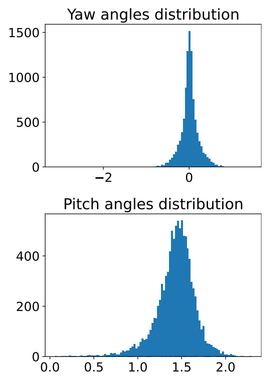
Figure 12: Real images from the Megascans Food dataset. Caution: some objects in this dataset could be poisonous.

Table 5: Comparing 3D datasets. Megascans Plants and Megascans Food are much more complex in terms of geometry and has much better camera coverage than FFHQ [25] or Cats [79]. The abbreviation "USphere(μ, ζ)" denotes uniform distribution on a sphere (see π-GAN [7]) with mean μ and pitch interval of $[\mu - \zeta, \mu + \zeta]$. For Cats, the final resolution depends on the cropping and we report the original dataset resolution.

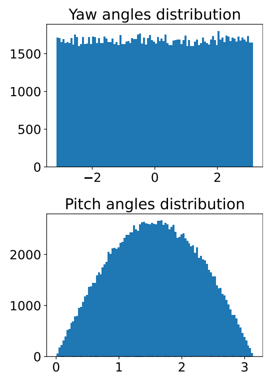
Dataset	Number of images	Yaw distribution	Pitch distribution	Resolution
FFHQ [25]	70,000	Normal(0, 0.3)	Normal($\pi/2$, 0.2)	1024^2
Cats	10,000	Normal(0, 0.2)	Normal($\pi/2$, 0.2)	$\approx 604 \times 520$
CARLA	10,000	USphere(0, π)	USphere($\pi/4$, $\pi/4$)	512^2
M-Plants	141,824	USphere(0, π)	USphere($\pi/2$, $\pi/2$)	1024^2
M-Food	25,472	USphere(0, π)	USphere($\pi/2$, $\pi/2$)	1024^2



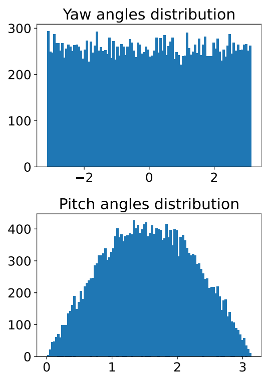
(a) FFHQ [25].



(b) Cats [79].



(c) Megascans Plants.



(d) Megascans Food.

Figure 13: Comparing yaw/pitch angles distribution for different datasets.

E.2 Datasets statistics

We provide the datasets statistics in Tab 5. For CARLA [57], we provide them for comparison and do not use this dataset as a benchmark since it is small, has simple geometry and texture.

F Additional samples

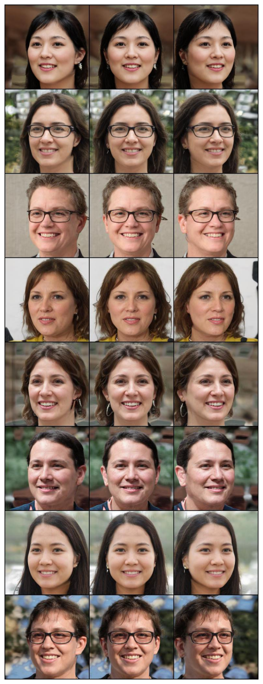
We provide random non-cherry-picked samples from our model in Fig 14, but we recommend visiting the website for video illustrations: https://universome.github.io/epigraf.

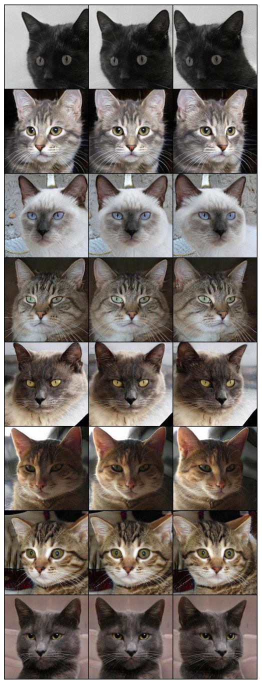
To demonstrate GRAM's [12] mode collapse, we provide more its samples in Fig 15.

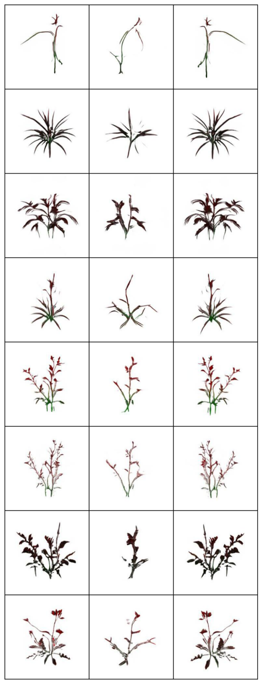
G Potential negative societal impacts

Our developed method is in the general family of media synthesis algorithms, that could be used for automatized creation and manipulation of different types of media content, like images, videos or 3D scenes. Of particular concern is creation of deepfakes[9] — photo-realistic replacing of one person's identity with another one in images and videos. While our model does not yet rich good enough quality to have perceptually indistinguishable generations from real media, such concerns should be kept in mind when developing this technology further.

[9]https://en.wikipedia.org/wiki/Deepfake


(a) FFHQ 512^2 [25].


(b) Cats 256^2 [79]


(c) Megascans Plants 256^2

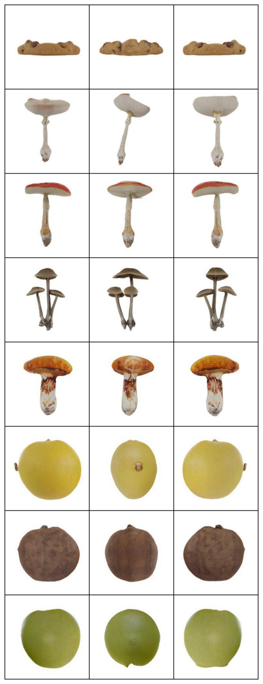

(d) Megascans Food 256^2

Figure 14: Random samples (without any cherry-picking) for our model. Zoom-in is recommended.

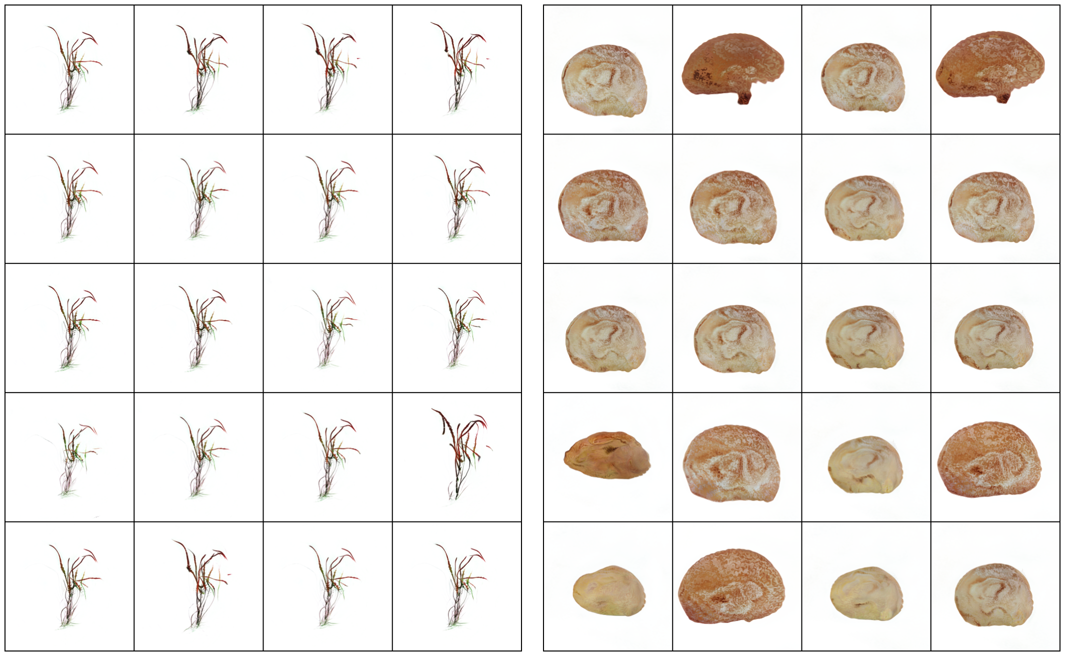


Figure 15: Random samples from GRAM [12] on M-Plants 256^2 (left) and M-Food 256^2 (right). Since it uses the same set of iso-surfaces for each sample to represent the geometry, it struggles to fit the datasets with variable structure, suffering from a very severe mode collapse.

H Ethical concerns

We have reviewed the ethics guidelines[10] and confirm that our work complies with them. As discussed in Appx E.1, our released datasets are not human-derived and hence do not contain any personally identifiable information and are not biased against any groups of people.

[10] https://nips.cc/public/EthicsGuidelines