Autodecoding Latent 3D Diffusion Models Supplementary Material

Anonymous Author(s)
Affiliation
Address
email

A Additional Experiments and Results

A.1 Geometry Generation Evaluation

Following the point cloud evaluation protocol of [1], we measure the Coverage Score (COV) and the Minimum Matching Distance (MMD) for points sampled from our generated density volumes. Given a distance metric for two point clouds X and Y, *e.g.* the Chamfer Distance (CD),

$$\text{CD}(X, Y) = \sum_{x \in X} \min_{y \in Y} \|x - y\|_2^2 + \sum_{y \in Y} \min_{x \in X} \|x - y\|_2^2, \tag{1}$$

COV measures the *diversity* of the generated point cloud set S_g, with respect to a reference point clout set S_r, by finding the closest neighbor in the reference set to each one in the sample set, and computing the fraction of the reference set covered by these samples:

$$\text{COV}(S_g, S_r) = \frac{|\{\arg\min_{Y \in S_r} CD(X, Y) | X \in S_g\}|}{|S_r|}. \tag{2}$$

MMD, in contrast, measures the the overall *quality* of these samples, by measuring the average distance between each sampled point cloud and its closest neighbor in the reference set:

$$\text{MMD}(S_g, S_r) = \frac{1}{|S_r|} \sum_{Y \in S_r} \min_{X \in S_g} CD(X, Y). \tag{3}$$

We compute these metrics for the PhotoShape Chairs and ABO Tables datasets, comparing our generated results to points sampled from the the same reference meshes used in the data splits from the evaluations in DiffRF [18]. For each generated object, we sample 2048 points from a mesh extracted from the decoded density volume V^{Density} (see Sec. 3.1) using the Marching Cubes [16] algorithm. We use a volume of resolution 64^3 and 128^3 for training the Chairs and Tables models, respectively. However, we note that downsampling these density volumes to 32^3, as is used in DiffRF, before applying this point-sampling operation did not noticeably impact the results of these evaluations.

The results can be seen in Tab. 4, alongside the perceptual metrics from the main paper. Interestingly, these results show that, despite the increased flexibility of our approach, and DiffRF's restrictive use of both 2D rendering and 3D supervision on synthetic data when training their diffusion model, we obtain comparable or superior geometry compared to their approach, while substantially increasing the overall perceptual quality for these datasets. We also substantially outperform prior state-of-the-art approaches using GAN-based [2, 3] methods across both perceptual and geometric comparisons with these metrics.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Method	PhotoShape Chairs [22] FID ↓	KID ↓	COV ↑	MMD ↓	ABO Tables [4] FID ↓	KID ↓	COV ↑	MMD ↓
π-GAN [2]	52.71	13.64	39.92	7.387	41.67	13.81	44.23	10.92
EG3D [3]	16.54	8.412	47.55	5.619	31.18	11.67	48.15	9.327
DiffRF [18]	15.95	7.935	58.93	**4.416**	27.06	10.03	**61.54**	7.610
Ours	**15.05**	**7.751**	**64.20**	4.4450	**18.44**	**6.854**	60.25	**6.684**

Table 4: **Quantitative comparison** of unconditional generation on the PhotoShape Chairs [22] and ABO Tables [4] datasets. Our method achieves a better perceptual quality, while maintaining similar geometric quality to the state-of-the-art diffusion-based approaches. MMD and KID scores are multiplied by 10^3.

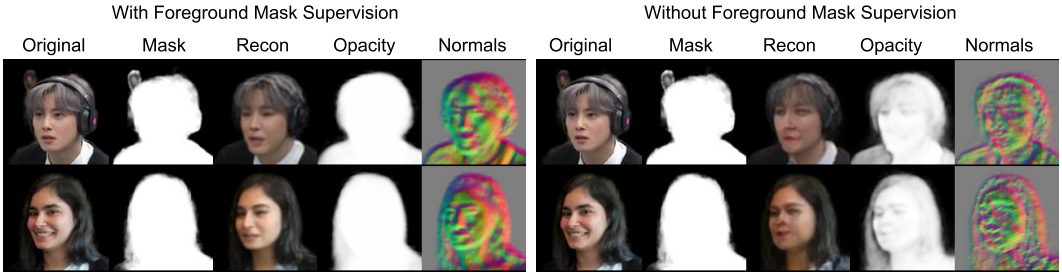

Figure 6: In real video datasets, *e.g*. CelebV-Text[36], we have a diverse set of foreground shapes and textures with a common background color. In these cases, we find that supervising the autodecoder with a foreground mask loss is important for the network to properly learn the shape of the object. Both examples shown after training for $\sim$9 million frames.

Figs. 7 and 8 show qualitative comparisons between the unconditional generation results rendered using our method and DiffRF for each of these datasets. In each case, it is clear that for similar objects, our method produces more coherent and complete shapes without missing features, *e.g*. legs, and textures that are more realistic and detailed, leading to better and more consistent image synthesis results.

A.2 Foreground Supervision

For some datasets with foregrounds with complex and varying appearance which can easily be mixed with the background environment, we found it necessary to supplement our primary autodecoder reconstruction loss (Sec. 3.2) with an additional foreground supervision loss. This loss measures how well depicted objects are separated from the background during rendering. To evaluate the effect of this foreground supervision, we ran experiments on the CelebV-Text [36] dataset both with and without this loss. We conduct our training until the autodecoder has seen a total of 9 million frames from the training set, then reconstruct examples from the learned embeddings.

The result can be seen in Fig. 6. As depicted, the reconstructions without foreground supervision not only lack fidelity to the target appearance, but the estimated opacity and surfaces normals clearly show that the overall geometry is insufficiently recovered.

A.3 Animated Results

Please see the corresponding supplementary web page for additional video results, showing consistent novel-view synthesis for rigid objects from multi-category datasets and animated articulated objects sampled using our approach, and results demonstrating both conditional and unconditional generation.

Figure 7: **Qualitative comparison of unconditional generation** using DiffRF [18] (left) and our approach (right) on the ABO Tables dataset [4]. In contrast to DiffRF, we train diffusion in the latent features of an autodecoder. Decoupling the expensive and demanding training from the output voxel-grid size lets us increase the resolution of our 3D representation. For this dataset, our output voxel resolution is 128^3, compared to the 32^3 resolution of DiffRF. Our method improves the perceptual quality of the results, as it as shown in the reported FID and KID.

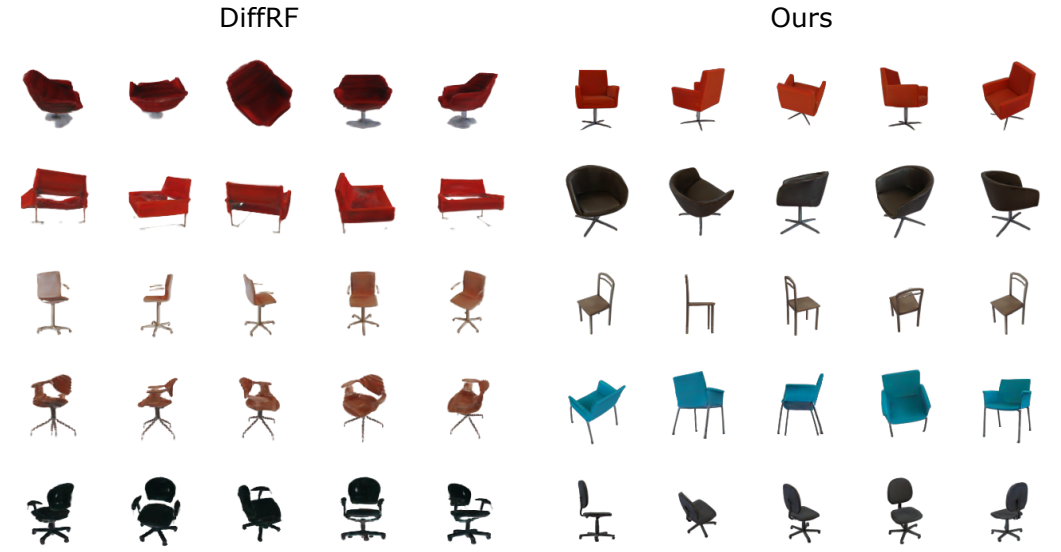

Figure 8: **Qualitative comparison of unconditional generation** using DiffRF [18] (left) and our approach (right) on the PhotoShapes Chairs dataset [22]. For this dataset, our output voxel resolution is 64^3. As above, our results are both qualitatively and quantitatively superior.

B Method Details

B.1 Volumetric Autodecoder

Volumetric Rendering. We use learnable volumetric rendering [17] to generate the final images from the final decoded volume. Given a camera intrinsic and extrinsic parameters for a target image, and the radiance field volumes generated by the decoder, for each pixel in the image, we cast a ray through the volume, sampling the color and density values to compute the color $C(\mathbf{r})$ by integrating the radiance along the ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, with near and far bounds t_n and t_f:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\delta(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt, \tag{4}$$

where δ, $\mathbf{c}$ are the density and RGB values from the radiance field volumes sampled along these rays, and $T(t) = \exp\Big\{-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\Big\}$ is the accumulated transmittance between t_n and t.

To supervise the silhouette of objects, we also render the 2D occupancy map O using the volumetric equation:

$$O(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\delta(\mathbf{r}(t))dt. \tag{5}$$

We sample 128 points across these rays for radiance field rendering during training and inference.

Articulated Animation. As our approach is flexibly designed to support both rigid and articulated subjects, we employ different approaches to pose supervision to better handle each of these cases.

For articulated subjects, poses are estimated during training, using a set of learnable 3D keypoints K^{3D} and their predicted 2D projections K^{2D} in each image in an extended version of the Perspective-n-Point (PnP) algorithm [12]. To handle articulated animation, however, rather than learn a single pose per image using these points, we assume that the target subjects can be decomposed into N_p regions, each containing N_k points K_p^{3D} points and their corresponding K_p^{2D} projections per image. These points are shared across all subjects, and are aligned in the learned canonical space, allowing for realistic generation and motion transfer between these subjects. This allows for learning N_p poses per-frame defining the pose of each region p relative to its pose in the learned canonical pose.

To successfully reconstruct the training images for each subject thus requires learning the appropriate canonical locations for each region's 3D keypoints, to predict the 2D projections of these keypoints in each frame, and the pose best matching the 3D points and 2D projections for these regions. We can then use this information in our volumetric rendering framework to sample appropriately from the canonical space such that the subject's appearance and pose are consistent and appropriate throughout their video sequence. Using this approach, this information can be learned along with our autodecoder parameters for articulated objects using the reconstruction and foreground supervision losses used for our rigid object datasets.

As noted in Sec. 3.2, to better handle non-rigid shape deformations corresponding to this articulated motion, we employ volumetric linear blend skinning (LBS) [13]. This allows us to learn the weight each component p in the canonical space contributes to a sampled point point in the deformed space based on the spatial correspondence between these two spaces:

$$x_d = \sum_{p=1}^{N_p} w_p^c(x_c)\left(R_p x_c + \mathrm{tr}_p\right), \tag{6}$$

where $T_p = [R_p, t_p] = [R^{-1}, -R^{-1}\,\mathrm{tr}]$ is the estimated pose of part p relative to the camera (where $T = [R, \mathrm{tr}] \in \mathbb{R}^{3\times4}$ is the estimated camera pose with respect to our canonical volume) ; x_d is the 3D point deformed to correspond to the current pose; x_c is its corresponding point when aligned in the canonical volume; and $w_p^c(x_c)$ is the learned LBS weight for component p, sampled at position x_c in the volume, used to define this correspondence. [1]

[1] In practice, as in [30], we compute an approximate solution using the inverse LBS weights following HumanNeRF [33] to avoid the excessive computation required by the direct solution.

Thus, for our non-rigid subjects, in addition to the density and color volumes needed to integrate Eqns. 4 and 5 above, our autodecoder learns to produce a volume $V^{LBS} \in \mathbb{R}^{S^3 \times N_p}$ containing the LBS weights for each of the N_p locally rigid regions constituting the subject.

We assign $N_k = 125$ 3D keypoints to each of the $N_p = 10$ regions. For these tests, we assume fixed camera intrinsics with a field-of-view of 0.175 radians, as in [19]. We use the differentiable Perspective-n-Point (PnP) algorithm [12] implementation from PyTorch3D [26] to accelerate this training process.

As this approach suffices for objects with standard canonical shapes (*e.g.*, human faces) performing non-rigid motion in continuous video sequences, we employ this approach for our tests on the CelebV-Text dataset. While in theory, such an approach could be used for pose estimation for rigid objects (with only 1 component) in each view, for we find that this approach is less reliable for our rigid object datasets, which contain sparse, multi-view images from randomly sampled, non-continuous camera poses, depicting content with drastically varying shapes and appearances (*e.g.*, the multi-category object datasets described below). Thus, for these objects, we use as input either known ground-truth or estimated camera poses (using [28]), for synthetic renderings or real images, respectively. While some works [32, 14, 35] perform category-agnostic object or camera pose estimation without predefined keypoints from sparse images of arbitrary objects or scenes, employing such techniques for such data is beyond the scope of this work.

Architecture. Our volumetric autodecoder architecture follows that of [30], with the key extensions described in this work. Given an embedding vector $\mathbf{e}$ of size 1024, we use a fully-connected layer followed by a reshape operation to transform it into a 4^3 volume with 512 features per cell. This is followed by a series of four 3D residual blocks, each of which upsamples the volume resolution in each dimension and halves the features per cell, to a final resolution of 64^3 and 32 features. [2] These blocks consist of two $3 \times 3 \times 3$ convolution blocks each followed by batch normalization in the main path, while the residual path consists of four $1 \times 1 \times 1$ convolutions, with ReLU applied after these operations. After the first of these blocks we have the 8^3 volume with 256 features per cell used for training our diffusion network, as in our final experiments. In this and the subsequent block, we apply self-attention layers [31] as described in Sec. 3.1. After the final upsampling block, we apply a final batch normalization followed by a $1 \times 1 \times 1$ convolution to produce the final $1 + 3$ density V^{Density} and RGB color features V^{RGB} used in our volumetric renderer.

Non-Rigid Architecture. For non-rigid subjects, our architecture produces $1 + 3 + 10$ output channels, with the latter group with the LBS weights for the $n_p = 10$ locally rigid components each region corresponds to in our canonical space. Our unsupervised 2D keypoint predictor uses the U-Net architecture of [29], which operates on a downsampled 64×64 input image to predict the locations of the keypoints corresponding to each of the 3D keypoints used to determine the pose of the camera relative to each region of the subject when it is aligned in the canonical volumetric space.

B.2 Latent 3D Diffusion

Diffusion Architecture and Sampling. For our base diffusion model architecture, we use the Ablated Diffusion Model (ADM) of Dhariwal *et al*. (2021) [7], a U-Net architecture originally designed for 2D image synthesis. We incorporate the preconditioning enhancements to this model described in Karras *et al*. (2022) [9]. As this architecture was originally designed for 2D, we adapt all convolutions and normalizations operations, as well as the attention mechanisms, to 3D.

For the cross-attention mechanism used for our conditioning experiments, we likewise extend the latent-space cross-attention mechanism from Rombach *et al*. (2022) [27] to our 3D latent space.

Robust Normalization. Autoencoder-based latent diffusion models impose a prior to the learned latent vector [27]. We find the latent features learned by our 3D autodecoder already form a bell-like curve. However, we also observe extreme values that can severely affect the calculation of the mean and standard deviation. As discussed in the main manuscript, we deploy the use of *robust normalization* to adjust the latent features. In particular, we take the *median* m as the center of the distribution and approximate its scale using the Normalized InterQuartile Range (IQR) [34] for a normal distribution: $0.7413 \times IQR$. We visualize its effect in Fig. 9. This is a crucial aspect of our approach, as in our experiments we find that without it, our diffusion training is unable to converge.

[2] We add one block to upsample to 128^3 for our aforementioned experiments with the ABO Tables dataset.

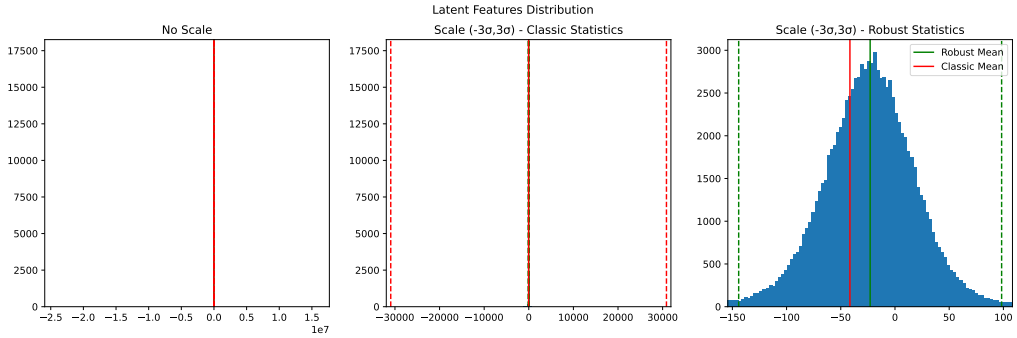

Figure 9: We present the latent feature distribution of a 3D *AutoDecoder* trained on MVImgNet[37]. The features are extracted at the 8^3 resolution, where we apply diffusion. The three subplots show different levels of "zooming in." We see that the distribution spans a great range due to extreme outliers. Using classic mean and standard deviation computation, as we see in the middle subplot, still provides quite a large range of values. Normalizing the features using classic statistics leads to convergence failure for the diffusion model. We propose using robust statistics to normalize the distribution to $[-1, 1]$, before training the diffusion model. During inference, we de-normalize the diffusion output before feeding them to the upsampling layers of the autodecoder.

Figure 10: **Qualitative comparison of models trained at different latent resolutions.** All visualizations produced with 64 diffusion steps. We find that the model train on 8^3 latent features gives the best trade-off between quality and training speed, rendering it the best option for training on large-scale 3D datasets.

Ablating the latent volume resolution used for diffusion. We trained three diffusion models models for the same time, resources, and number of parameters, for diffusion at 3 resolutions in our autodecoder: 4^3, 8^3, and 16^3. We find that the 4^3 models, even when they train faster, often fail to converge to something meaning full and produce partial results. Most samples produced by the 16^3 models are of reasonable quality. However, many samples also exhibit spurious density values. We hypothesize that this is due to the model being under-trained. The 8^3 model produces the best results, and its fast training speed makes it suitable for large-scale training. We visualize the results in Fig. 10

C Implementation Details

C.1 Dataset Filtering

CelebV-Text [36]. Some heuristic filtering was necessary to obtain sufficient video quality and continuity for our purposes. We omit the first and last 10% of each video to remove fade-in/out effects, and any frames with less than 25% estimated foreground pixels. We also remove videos with less than 4 frames remaining after this, and any videos less than 200 kilobytes due to their relatively low quality. We also omit a small number of videos that were unavailable for download at the time of our experiments (the dataset is provided as a set of URLs for the video sources).

MVImgNet [37]. For these annotated video frames depicting real objects in unconstrained settings and environments, we applied Grounded Segment Anything [11] for background removal. However, as this process sometimes failed to produce acceptable segmentation results, we apply filtering to detect these case. We first remove objects for which Grounding DINO [15] fails to detect bounding boxes. We then fit our volumetric autodecoder (Secs. 3.1-2) to only the *masks* produced by this segmentation (as monochrome images with a white foreground and a black background). For objects that are properly segmented in each frame, this produces a reasonable approximation of the object's shape that is consistent in each of the input frames, while objects with incorrect or inconsistent segmentation will not be fit properly to the input images. Thus, objects for which the fitting loss is unsually high are removed.

Objaverse [5]. While Objaverse contains $\sim$800K 3D models, we found that the overall quality of these varied greatly, making many of them unsuitable for multi-view rendering. We thus filtered models without texture, material maps, or other color and appearance properties suitable, as well as models with an insufficient polygon count for realistic rendering. Interestingly, given the simplicity of the objects when rendered against a monochrome background, we found that the foreground segmentation supervision used for the other experiments described in Sec. 3.2 of the main paper was unnecessary. Given the scale of this dataset ($\sim$300K unique objects, with 6 frames per object), we thus omit this loss from our training process for this dataset for our final experiments for the sake of improved training efficiency. For datasets with more complex motion and real backgrounds, such as the real image datasets mentioned above, we found this supervision to be essential, as shown in Sec. A.2 and Fig. 6.

Training Details. Our experiments are implemented in the PyTorch [23, 24], using the PyTorch Lightning [8] framework for fast automatic differentiation and scalable GPU-accelerated parallelization. For calculating the perceptual metrics (FID and KID), we used the Torch Fidelity [21] library.

We run our experiments on 8 NVIDIA A100 40GB GPUs per node. For some experiments, we use a single node, while for larger-scale experiments, we use up to 8 nodes in parallel.

We use the Adam optimizer [10] to train both the autodecoder and the diffusion Model. For the first network, we use a learning rate $lr = 5e-4$ and beta parameters $\beta = (0.5, 0.999)$. For diffusion, we set the learning rate to $lr = 4.5e-4$. We apply linear decay to the learning rate.

Preparing the Text Embeddings for Text-Driven Generation. We train our model for text-conditioned image generation on three datasets: CelebV-Text [36], MVImgNet [37] and Objaverse [5]. The two latter datasets provide the object category of each sample, but they do not provide text descriptions. Using MiniGPT4 [38], we extract a description by providing a *hint* and the first view of each object along with the question: "*<Img><ImageHere></Img> Describe this <hint> in one sentence. Describe its shape and color. Be concise, use only a single sentence.*" For MVImgNet, this hint is the "class name", while it is the "asset name" for Objaverse.

With the text-image pairs for these three datasets, we use the 11-billion parameter T5 [25] model to extract a sequence of text-embedding vectors. The dimensionality of these vectors is 1024. During training, we fix the length of the embedding sequence to 32 elements. We trim longer sentences and pad smaller sentences with zeroes.

D Additional Observations from Our Experiments

D.1 Hash Embedding

Each object in the training set is encoded by an embedding vector. However, as we employ multi-view datasets of various scales, up to $\sim$300K unique targets from multiple categories, storing a separate embedding vector for each object depicted in the training images is burdensome [3]. As such, we experimented with a technique enabling the effective use of a significantly reduced number of embeddings (no more than $\sim$32K are required for any of our evaluations), while allowing effective content generation from large-scale datasets.

[3] *E.g.*, the codebook *alone* would require *six* times the parameters of the largest model in our experiments.

	ABO-Tables	*Chairs*	*CelebV-Text*	*MVImgNet*	*Objaverse*
			3D AutoDecoder		
z-length	1024	1024	1024	1024	1024
MaxChannels	512	512	512	512	512
Depth	2	4	2	4	4
SA-Resolutions	8,16	8,16	8,16	8,16	8,16
ForegroundLoss λ	10	10	10	10	0
#Renders/batch	4	4	4	4	4
VoxelGridSize	$128^3 \times 4$	$64^3 \times 4$	$64^3 \times 14$	$64^3 \times 4$	$64^3 \times 4$
Learning Rate	5e-4	5e-4	5e-4	5e-4	5e-4
			Latent 3D Diffusion Model		
z-shape	$8^3 \times 256$	$8^3 \times 256$	$8^3 \times 256$	$8^3 \times 256$	$8^3 \times 256$
Sampler	edm	edm	edm	edm	edm
Channels	128	128	192	192	192
Depth	2	2	3	3	3
Channel Multiplier	3,4	3,4	3,4	3,4	3,4
SA-resolutions	8,4	8,4	8,4	8,4	8,4
Learning Rate	4.5e-5	4.5e-5	4.5e-5	4.5e-5	4.5e-5
Conditioning	None	None	None/CA	None/CA	None/CA
CA-resolutions	-	-	8,4	8,4	8,4
Embedding Dimension	-	-	1024	1024	1024
Transformers Depth	-	-	1	1	2

Table 5: Architecture details for our models for each dataset. *SA* and *CA* stand for *Self-Attention* and *Cross-Attention* respectively. z refers to our 1D embedding vector and our latent 3D volume for the autodecoder and diffusion models, respectively. Note that for CelebV-Text, the output volume has 14 channels per cell: 3 for color values, 1 for density and 10 for part assignment.

Similar to the approach in [20], we instead employ concatenations of smaller embedding vectors to create more combinations of unique embedding vectors used during training. For an embedding vector length l_v, the input embedding vector $H_k \in \mathbb{R}^l$ used for an object to be decoded is a concatenation of smaller embedding vectors h_i^j, where each vector is selected from an ordered codebook with n_c entries, with each entry containing collection of n_h embedding vectors of length l_v/n_c:

$$H_k = \left[h_1^{k_1}, h_2^{k_2}, ..., h_{n_c}^{k_{n_c}} \right], \tag{7}$$

where $k_i \in \{1, 2, ..., n_h\}$ is the set of indices used to select from the n_h possible codebook entries for position i in the final vector. This method allows for exponentially more combinations of embedding vectors to be provided during training than must be stored in learned embedding vector library.

However, while in [20], the index j for the vector h_i^j at position i is randomly selected for each position to access its corresponding codebook entry, we instead use a deterministic mapping from each training object index to its corresponding concatenated embedding vector. This function is implemented using a hashing function employing the multiplication method [6] for fast indexing using efficient bitwise operations. For object index k, the corresponding embedding index is:

$$m(k) = [(a \cdot k) \bmod 2^w] \gg (w - r), \tag{8}$$

where the table has 2^r entries. w and a are heuristic hashing parameters used to reduce the number of collisions while maintaining an appropriate table size. We use 32 for w. a must be an odd integer between 2^{w-1} and 2^w [6]. We give each smaller codebook its own a value:

$$a_i = 2^{w-1} + 2 * i^2 + 1, \tag{9}$$

where i is the index of the codebook.

Discussion. In our experiments, we found that employing this approach had negligible impact on the overall speed and quality of our training and synthesis process. During training the memory of the GPU is predominantly occupied by the gradients, which are not affected by this hashing scheme. For Objaverse, our largest dataset using $\sim$300K images, using this technique saves approximately 800MB of storage space.

Interestingly, this also suggests that scaling this approach to larger datasets, should they become available, will require special handling. Learning this per-object embedding would soon become intractable. However, simply using this *hash embedding* approach reduces the model storage requirements by $\sim 75\%$ for this dataset.

In our experiments, we use hashing for ABO Tables, CelebV-Text and Objaverse, with codebook sizes $n_c =$ of 256, 8192 and 32768, respectively. We set the number of smaller codebooks (n_h) to 256 for each dataset.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning Representations and Generative Models for 3D Point Clouds. In *Proceedings of the International Conference on Machine Learning*, 2018.

[2] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-GAN: Periodic Implicit Generative Adversarial Networks for 3D-Aware Image Synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[3] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient Geometry-aware 3D Generative Adversarial Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[4] Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F Yago Vicente, Thomas Dideriksen, Himanshu Arora, Matthieu Guillaumin, and Jitendra Malik. ABO: Dataset and Benchmarks for Real-World 3D Object Understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[5] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A Universe of Annotated 3D Objects. In *arXiv*, 2022.

[6] Srini Devadas and Konstantinos Daskalakis. MIT 6.006, Lecture 5: Hashing I: Chaining, Hash Functions, 2009.

[7] Prafulla Dhariwal and Alexander Nichol. Diffusion Models Beat Gans on Image Synthesis. In *Proceedings of the Neural Information Processing Systems Conference*, 2021.

[8] William Falcon et al. PyTorch Lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[9] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the Design Space of Diffusion-Based Generative Models. In *Proceedings of the Neural Information Processing Systems Conference*, 2022.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[11] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment Anything. In *arXiv*, 2023.

[12] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. EPnP: An Accurate O(n) Solution to the PnP Problem. In *International Journal of Computer Vision*, 2009.

[13] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation. In *ACM Transactions on Graphics*, 2000.

[14] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. BARF: Bundle-Adjusting Neural Radiance Fields. In *Proceedings of the IEEE International Conference on Computer Vision*, 2021.

[15] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection. In *arXiv*, 2023.

[16] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *ACM Transactions on Graphics*, 1987.

[17] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as Neural Radiance Fields for View Synthesis. In *Proceedings of the European Conference on Computer Vision*, 2020.

[18] Norman Müller, Yawar Siddiqui, Lorenzo Porzi, Samuel Rota Bulò, Peter Kontschieder, and Matthias NieSSner. DiffRF: Rendering-Guided 3D Radiance Field Diffusion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

[19] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[20] Evangelos Ntavelis, Mohamad Shahbazi, Iason Kastanis, Radu Timofte, Martin Danelljan, and Luc Van Gool. StyleGenes: Discrete and Efficient Latent Distributions for GANs. In *arXiv*, 2023.

[21] Anton Obukhov, Maximilian Seitzer, Po-Wei Wu, Semen Zhydenko, Jonathan Kyl, and Elvis Yu-Jing Lin. High-fidelity performance metrics for generative models in PyTorch, 2020. URL `https://github.com/toshas/torch-fidelity`. Version: 0.3.0, DOI: 10.5281/zenodo.4957738.

[22] Keunhong Park, Konstantinos Rematas, Ali Farhadi, and Steven M. Seitz. PhotoShape: Photorealistic Materials for Large-Scale Shape Collections. In *ACM Transactions on Graphics*, 2018.

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic Differentiation in PyTorch, 2017.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of the Neural Information Processing Systems Conference*, 2019.

[25] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. In *The Journal of Machine Learning Research*, 2020.

[26] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D Deep Learning with PyTorch3D. In *arXiv*, 2020.

[27] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis With Latent Diffusion Models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[28] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[29] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. First Order Motion Model for Image Animation. In *Proceedings of the Neural Information Processing Systems Conference*, 2019.

[30] Aliaksandr Siarohin, Willi Menapace, Ivan Skorokhodov, Kyle Olszewski, Hsin-Ying Lee, Jian Ren, Menglei Chai, and Sergey Tulyakov. Unsupervised Volumetric Animation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

[31] Vaswani, Ashish and Shazeer, Noam and Parmar, Niki and Uszkoreit, Jakob and Jones, Llion and Gomez, Aidan N and Kaiser, Łukasz and Polosukhin, Illia. Attention is all you need. In *Proceedings of the Neural Information Processing Systems Conference*, 2017.

[32] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF−−: Neural Radiance Fields Without Known Camera Parameters. In *arXiv*, 2021.

[33] Chung-Yi Weng, Brian Curless, Pratul P Srinivasan, Jonathan T Barron, and Ira Kemelmacher-Shlizerman. HumanNeRF: Free-Viewpoint Rendering of Moving People from Monocular Video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[34] Dewey Lonzo Whaley III. *The Interquartile Range: Theory and Estimation*. PhD thesis, East Tennessee State University, 2005.

[35] Lumin Xu, Sheng Jin, Wang Zeng, Wentao Liu, Chen Qian, Wanli Ouyang, Ping Luo, and Xiaogang Wang. Pose for Everything: Towards Category-Agnostic Pose Estimation. In *Proceedings of the European Conference on Computer Vision*, 2022.

[36] Jianhui Yu, Hao Zhu, Liming Jiang, Chen Change Loy, Weidong Cai, and Wayne Wu. CelebV-Text: A Large-Scale Facial Text-Video Dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

[37] Xianggang Yu, Mutian Xu, Yidan Zhang, Haolin Liu, Chongjie Ye, Yushuang Wu, Zizheng Yan, Chenming Zhu, Zhangyang Xiong, Tianyou Liang, Guanying Chen, Shuguang Cui, and Xiaoguang Han. MVImgNet: A Large-scale Dataset of Multi-view Images. In *arXiv*, 2023.

[38] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models. In *arXiv*, 2023.