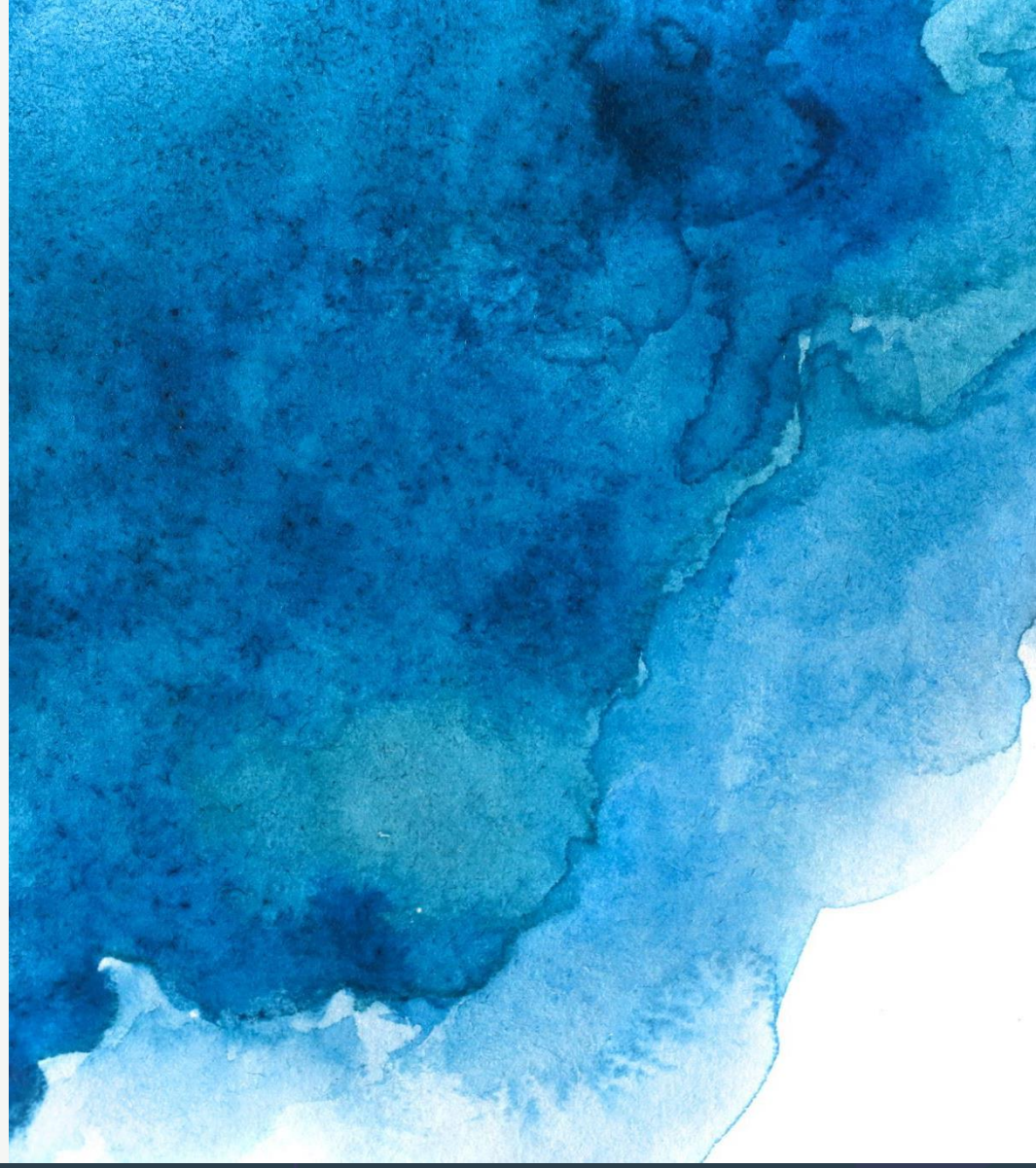
Manifold Diffusion Fields*

Ionescu Andrei, 507



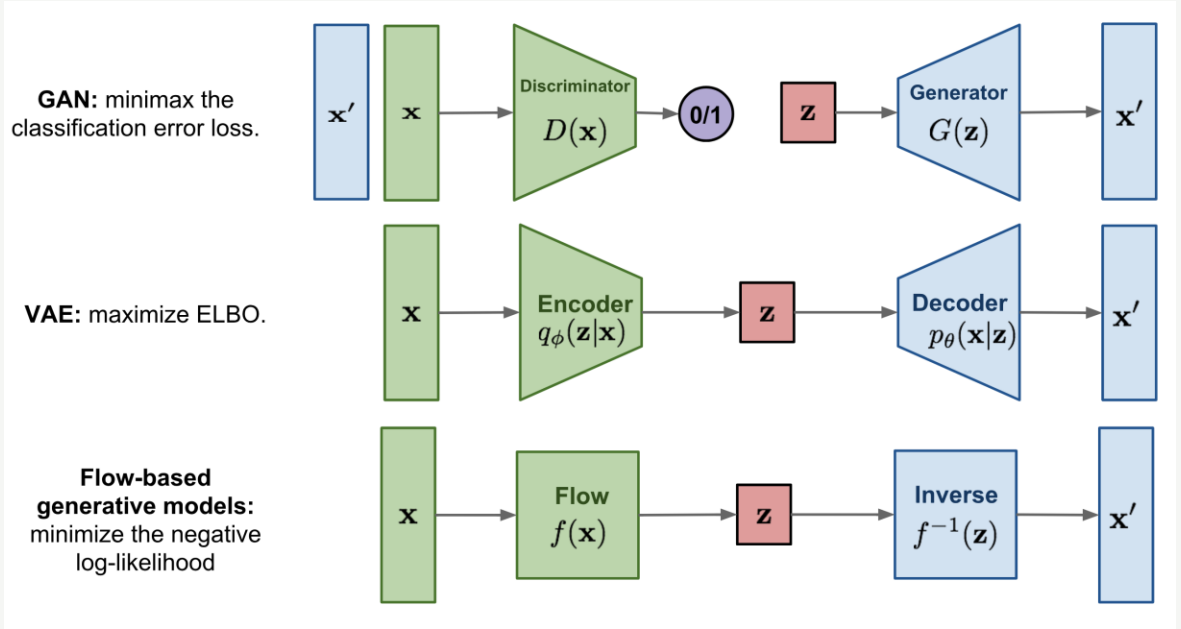
My Toy Implementation: https://github.com/fusedbloxxer/ub-g22-eddl-mdf/
***Paper Authors**: Ahmed A. Elhag and Yuyang Wang and Joshua M. Susskind and Miguel Angel Bautista
***Source**: https://arxiv.org/abs/2305.15586

Generative Models

They are a class of deep learning models that can be used to learn a **distribution** of a given dataset of **specific modality** and allow synthesizing new examples, either conditionally or **unconditionally**.

Generative Models

Some underlying **architectures** and approaches attempted in solving these kinds of tasks:



Denoising Diffusion Probabilistic Models

More recently a paper called **DDPM** introduced an approach that is based on training a Markov chain using variational inference to learn a **reverse** diffusion process:

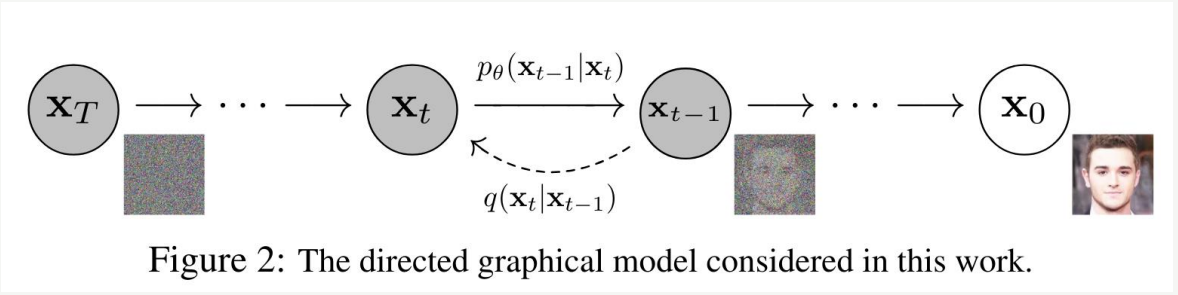


Figure 2: The directed graphical model considered in this work.

Denoising Diffusion Probabilistic Models

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Functa: Data points as INRs

One issue with discrete array representations is **high dimensionality.**

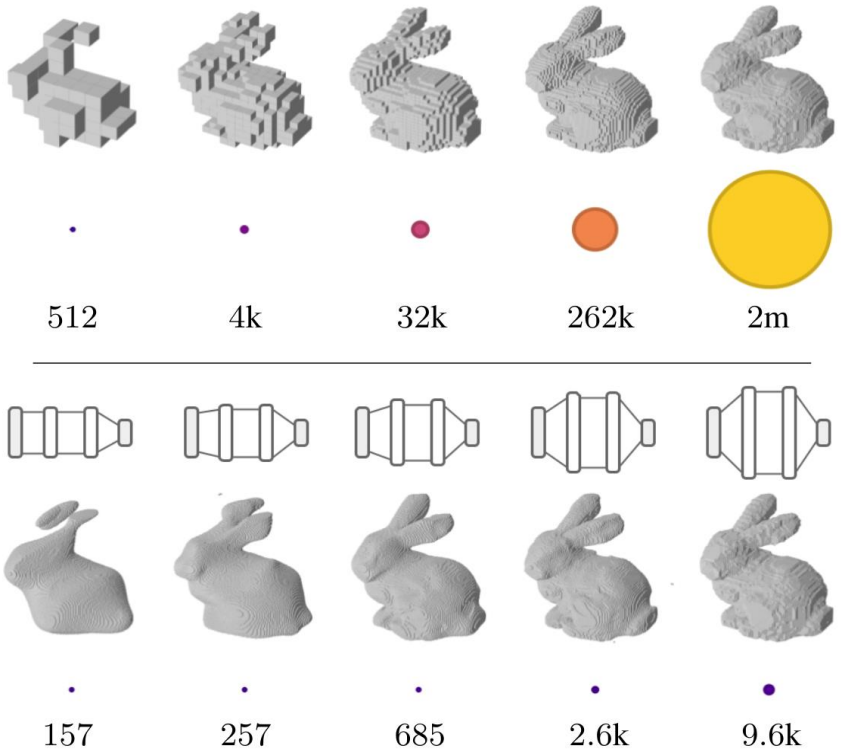


Figure 2. Functa scale much more gracefully with resolution than array representations. Circle area reflects the numerical size of the array (top) / function (bottom). See Appendix A.9 for details.

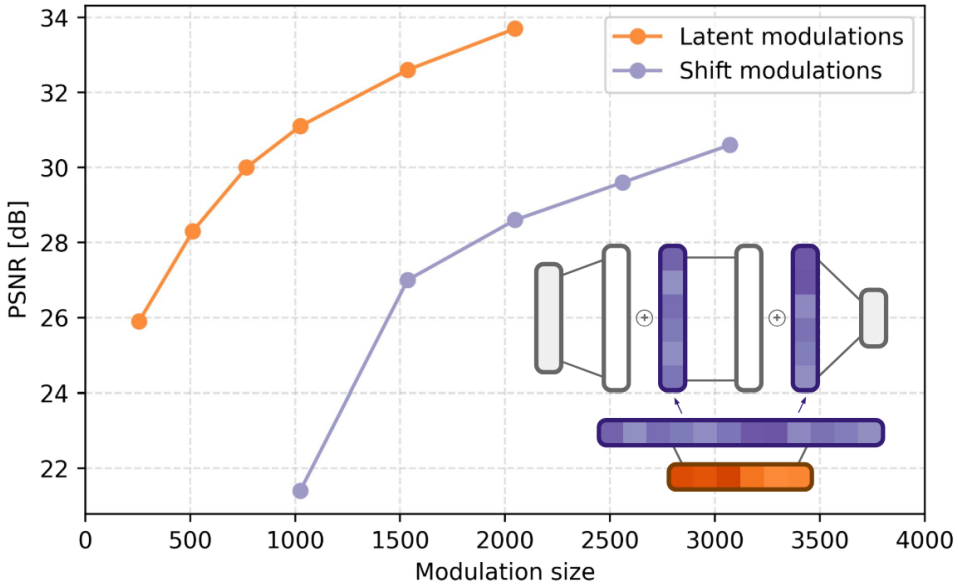


Figure 3. Reconstruction accuracy (in PSNR) vs modulation dimensionality on CelebA-HQ 64×64. Reconstruction accuracy is computed from the MSE between the image array and its INR evaluated at each pixel location. The model architecture is shown on the bottom right, with purple vectors corresponding to shift modulations and orange vectors to latent modulations.

Functa: Data points as INRs

The key idea is to learn a **neural function** as an **implicit representation** that can be parameterized by a **latent vector.** In this manner we could also learn **non-euclidean** representations!

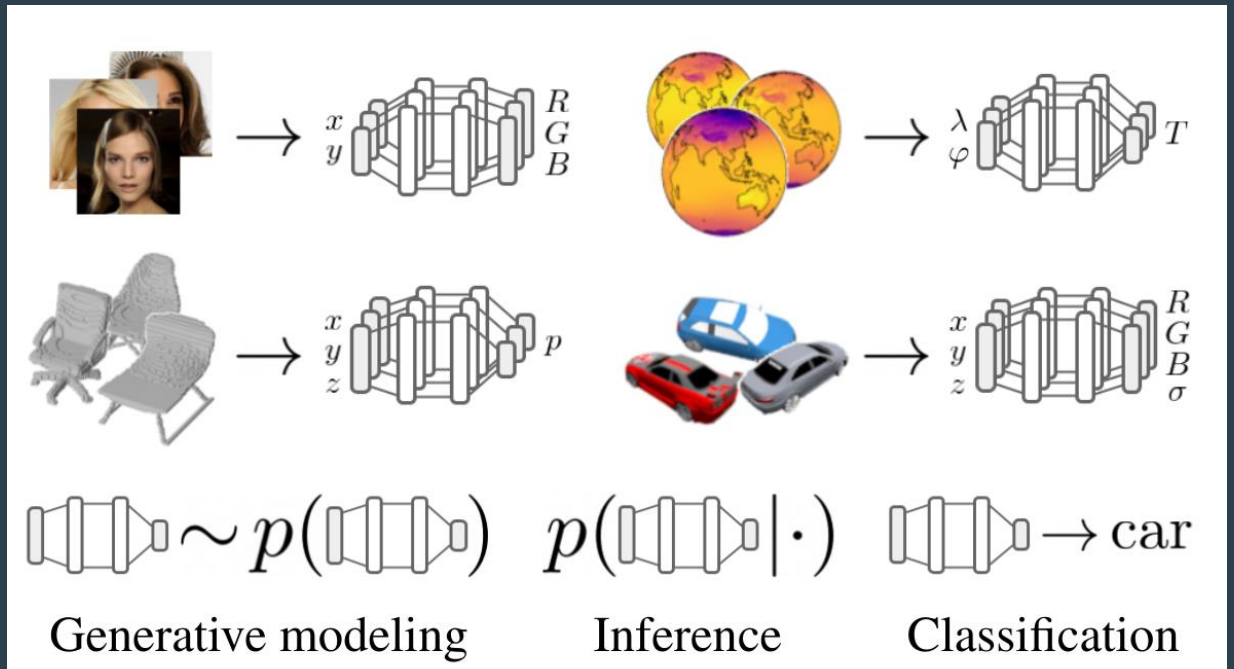


Figure 1. We convert array data into functional data parameterized by neural networks, termed *functa*, and treat these as data points for various downstream machine learning tasks.

MANIFOLD DIFFUSION FIELDS

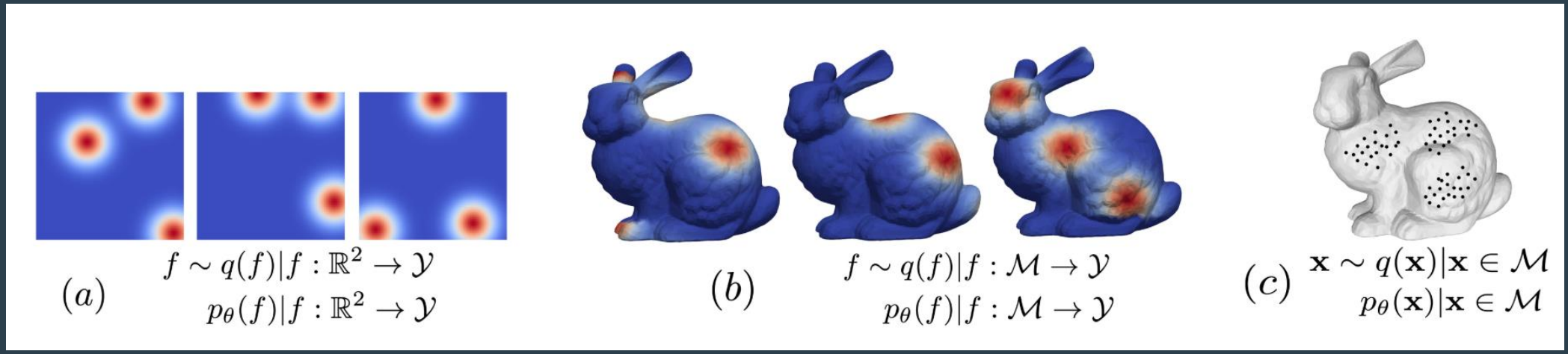
Ahmed A. Elhag*, **Yuyang Wang, Joshua M. Susskind, Miguel Angel Bautista**
Apple
{aa_elhag, yuyang_wang4, jsusskind, mbautistamartin}@apple.com

ABSTRACT

We present Manifold Diffusion Fields (MDF), an approach that unlocks learning of diffusion models of data in general non-Euclidean geometries. Leveraging insights from spectral geometry analysis, we define an intrinsic coordinate system on the manifold via the eigen-functions of the Laplace-Beltrami Operator. MDF represents functions using an explicit parametrization formed by a set of multiple input-output pairs. Our approach allows to sample continuous functions on manifolds and is invariant with respect to rigid and isometric transformations of the manifold.

Manifold Diffusion Fields

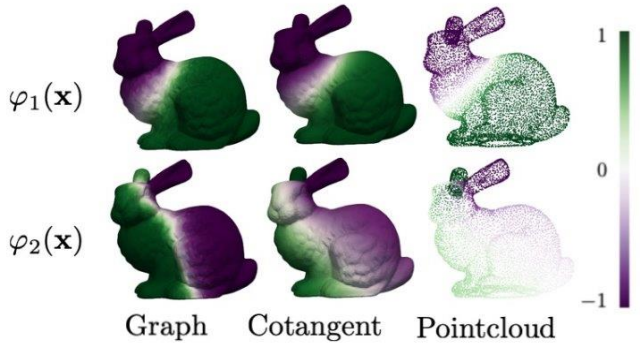
Their idea is to leverage a **diffusion model** that can learn **distributions** of functions, also called **fields**, that take points from a **manifold** (pointcloud, 3d mesh, graph, etc.) and outputs a **signal** value for each **f : M -> Y**



Manifold Positional Embedding

In order to be able to learn mappings from a **manifold** to a **signal** domain, the points on the manifold are encoded using **k eigenvectors** given by the smallest first k eigenvalues obtained by decomposing the **Laplace-Beltrami Operator** on the given manifold.

In practice, for general geom ices) we compute eigenvectors of the symmetric normalize ws:



(3)

Figure 8: Visualizing top-2 eigenvectors on the bunny manifold for Graph, Cotangent and Point-cloud (Sharp & Crane, 2020) Laplacians.

Manifold Positional Embedding

in spectral geometry analysis (Lévy, 2006). The eigen-decomposition of $\Delta_{\mathcal{M}}$ are the non-trivial solutions to the equation $\Delta_{\mathcal{M}}\varphi_i = \lambda_i\varphi_i$. The eigen-functions $\varphi_i : \mathcal{M} \to \mathbb{R}$ represent an orthonormal functional basis for the space of square integrable functions (Lévy, 2006; Minakshisundaram & Pleijel, 1949). Thus, one can express a square integrable function $f : \mathcal{M} \to \mathcal{Y}$, with $f \in L^2$ as a linear combination of the functional basis, as follows: $f = \sum_{i=1}^{\infty} \langle f, \varphi_i \rangle \varphi_i$.

In practice, the infinite sum is truncated to the k eigen-functions with lowest eigen-values, where the ordering of the eigen-values $\lambda_1 < \lambda_2 \cdots < \lambda_k$ enables a low-pass filter of the basis. Moreover, (Lévy, 2006) shows that the eigen-functions of Δ_M can be interpreted as a Fourier-like function basis (Vallet & Lévy, 2008) on the manifold, *e.g.* an intrinsic coordinate system for the manifold. In

Manifold Positional Embedding

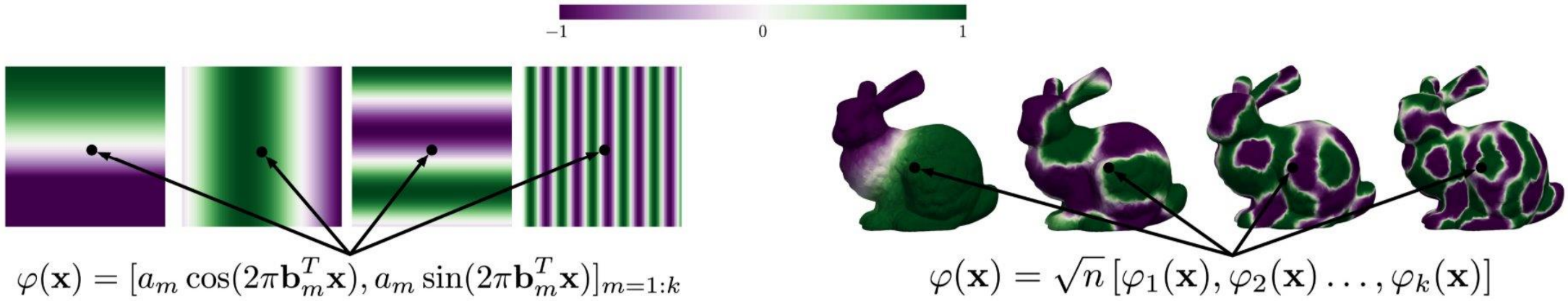


Figure 3: **Left:** Fourier PE of a point $\boldsymbol{x}$ in 2D Euclidean space. Generative models of functions in ambient space (Zhuang et al., 2023; Dupont et al., 2022b;a; Du et al., 2021) use this representation to encode a function's input. **Right:** MDF uses the eigen-functions φ_i of the Laplace-Beltrami Operator (LBO) $\Delta_{\mathcal{M}}$ evaluated at a point $\boldsymbol{x} \in \mathcal{M}$.

Field Representation & Score

$$\mathbf{C}_t = [\varphi(\mathbf{X}_c), \mathbf{Y}_{(c,t)} = \sqrt{\bar{\alpha}_t}\mathbf{Y}_{(c,0)} + \sqrt{1-\bar{\alpha}_t}\epsilon_c],$$

$$\mathbf{Q}_t = [\varphi(\mathbf{X}_q), \mathbf{Y}_{(q,t)} = \sqrt{\bar{\alpha}_t}\mathbf{Y}_{(q,0)} + \sqrt{1-\bar{\alpha}_t}\epsilon_q],$$

Training Algorithm

Algorithm 1 Training

1: $\Delta_{\mathcal{M}}\varphi_i = \varphi_i\lambda_i$ // LBO eigen-decomposition
2: **repeat**
3: $(\mathbf{C}_0, \mathbf{Q}_0) \sim \text{Uniform}(q(f_0))$
4: $t \sim \text{Uniform}(\{1, \ldots, T\})$
5: $\epsilon_c \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \epsilon_q \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6: $\mathbf{C}_t = [\varphi(\mathbf{X}_c), \sqrt{\bar{\alpha}_t}\mathbf{Y}_{(c,0)} + \sqrt{1-\bar{\alpha}_t}\epsilon_c]$
7: $\mathbf{Q}_t = [\varphi(\mathbf{X}_q), \sqrt{\bar{\alpha}_t}\mathbf{Y}_{(q,0)} + \sqrt{1-\bar{\alpha}_t}\epsilon_q]$
8: Take gradient descent step on
$\nabla_\theta \left\|\epsilon_q - \epsilon_\theta(\mathbf{C}_t, t, \mathbf{Q}_t)\right\|^2$
9: **until** converged

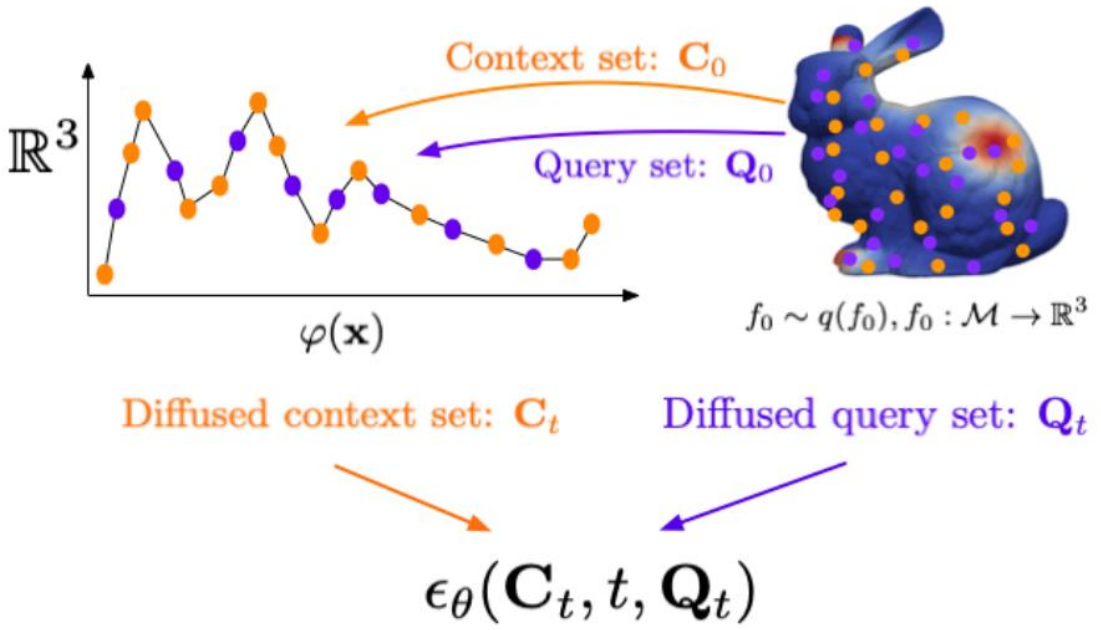


Figure 4: **Left:** MDF training algorithm. **Right**: Visual depiction of a training iteration for a field on the bunny manifold $\mathcal{M}$. See Sect. 4 for definitions.

Sampling Algorithm

Algorithm 2 Sampling

1: $\Delta_{\mathcal{M}}\varphi_i = \varphi_i\lambda_i$ // LBO eigen-decomposition
2: $\mathbf{Q}_T = [\varphi(\mathbf{X}_q), \mathbf{Y}_{(q,t)} \sim \mathcal{N}(\mathbf{0}_q, \mathbf{I}_q)]$
3: $\mathbf{C}_T \subseteq \mathbf{Q}_T$ ▷ Random subset
4: **for** $t = T, \ldots, 1$ **do**
5: $\quad \boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\boldsymbol{z} = \mathbf{0}$
6: $\quad \mathbf{Y}_{(q,t-1)} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{Y}_{(q,t)} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{C}_t, t, \mathbf{Q}_t)\right) + \sigma_t \boldsymbol{z}$
7: $\quad \mathbf{Q}_{t-1} = [\mathbf{M}_q, \mathbf{Y}_{(q,t-1)}]$
8: $\quad \mathbf{C}_{t-1} \subseteq \mathbf{Q}_{t-1}$ ▷ Same subset as in step 2
9: **end for**
10: **return** f_0 evaluated at coordinates $\varphi(\mathbf{X}_q)$

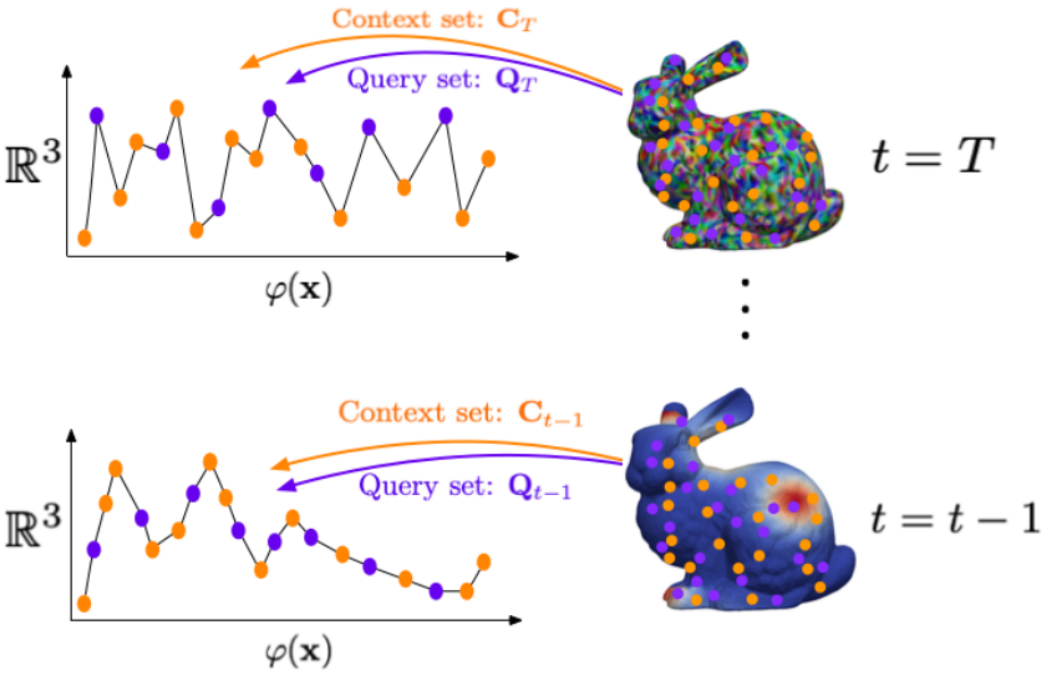
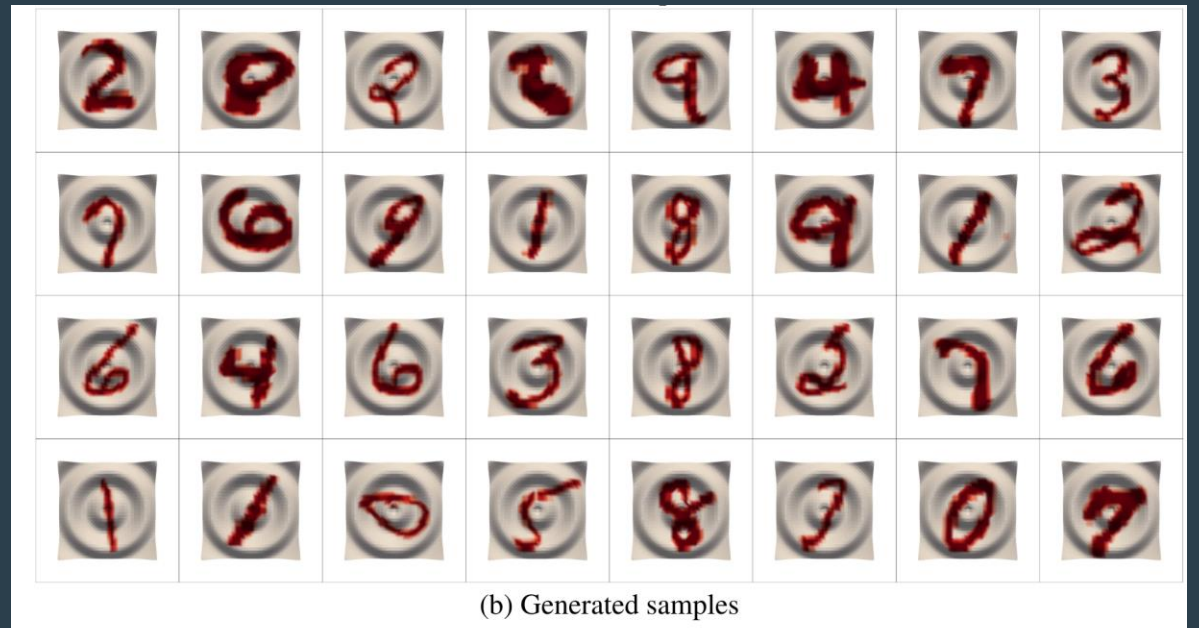


Figure 5: **Left:** MDF sampling algorithm. **Right**: Visual depiction of the sampling process for a field on the bunny manifold.

Applications

Figure 6: CelebA-HQ samples generated by MDF and GASP (Dupont et al., 2022b) on the bunny.



(b) Generated samples

Figure 18: Real and generated samples for MNIST (LeCun et al., 1998) digits on the wave manifold.

Project Idea

Motivation:

- Learn diffusion models better
- Learn key ideas about Laplacians and eigen-decomposition
- Learn more about mesh representation and manifolds

Idea:

- Try to implement MDF paper with different manifold & signal!

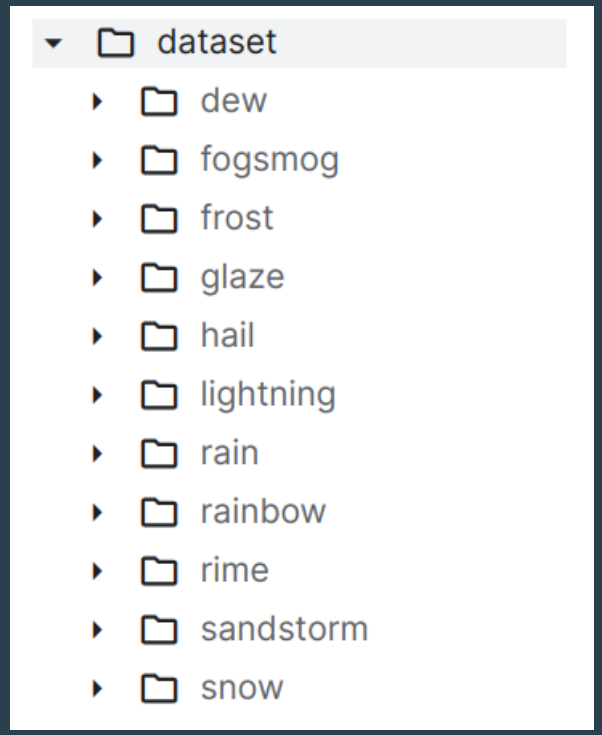
JEHAN BHATHENA · UPDATED 2 YEARS AGO

159 | New Notebook | Download (615 MB)

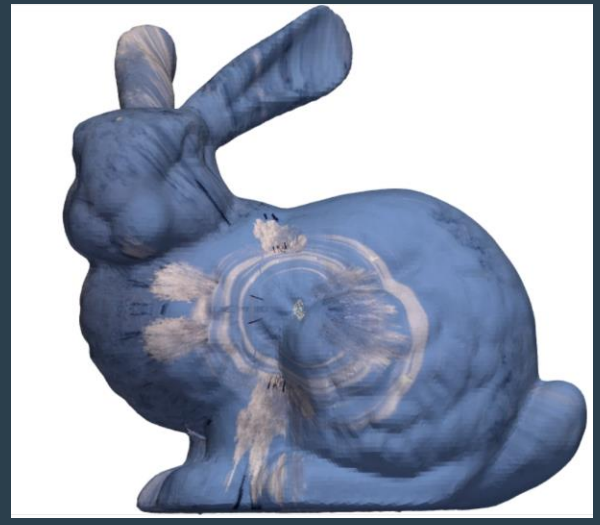
Weather Image Recognition

This dataset contains labeled 6862 images of different types of weather

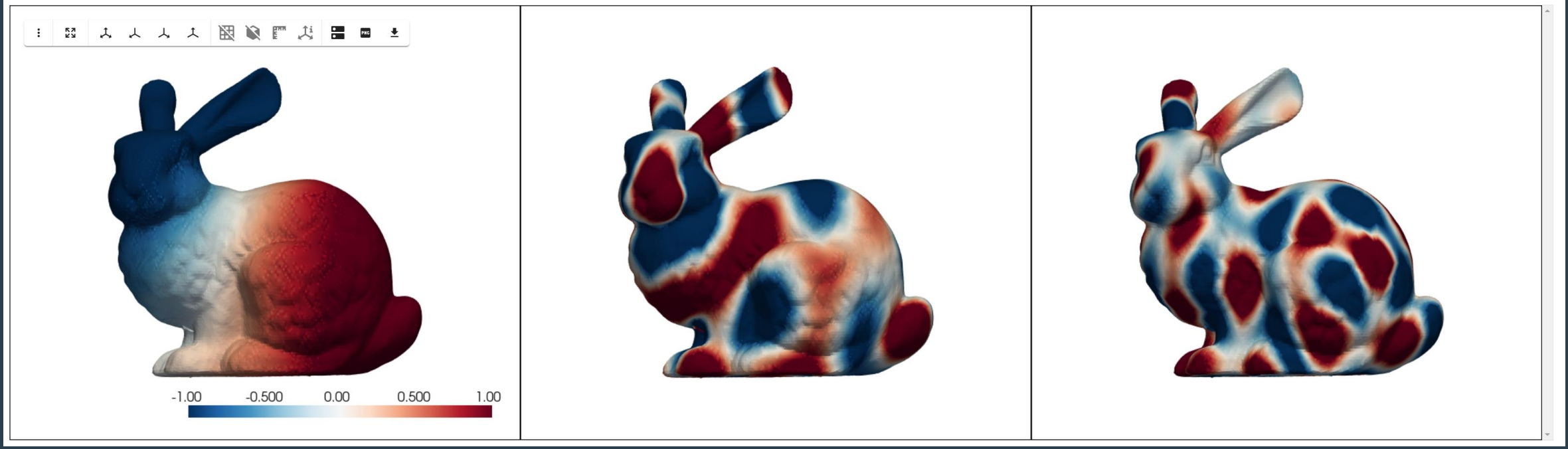
Data Card | Code (56) | Discussion (3) | Suggestions (0)



Texture Mapping



Positional Embedding using LBO



Sampling on the 3D Mesh

In order to sample points on the 3D Mesh I implemented a custom interpolation using Dirichlet distribution as **barycentric** coefficients:

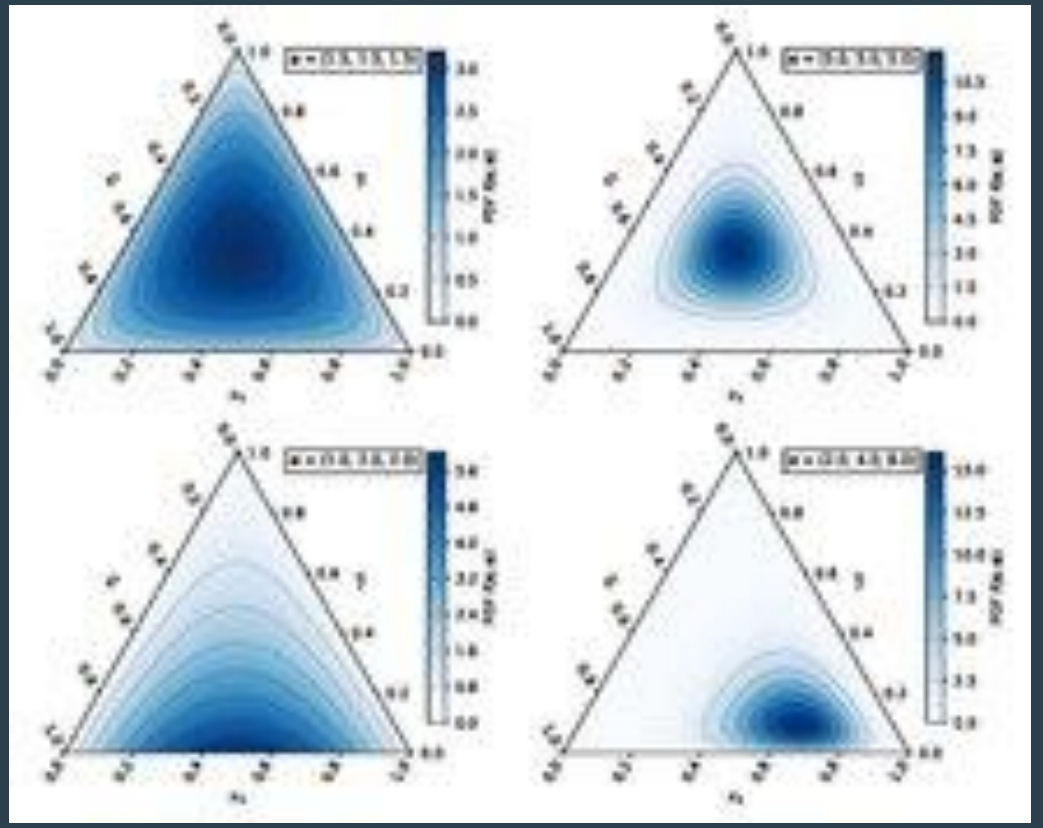
Probability density function [edit]

The Dirichlet distribution of order $K \geq 2$ with parameters $\alpha_1, \ldots, \alpha_K > 0$ has a probability density function with respect to Lebesgue measure on the Euclidean space $\mathbf{R}^{K-1}$ given by

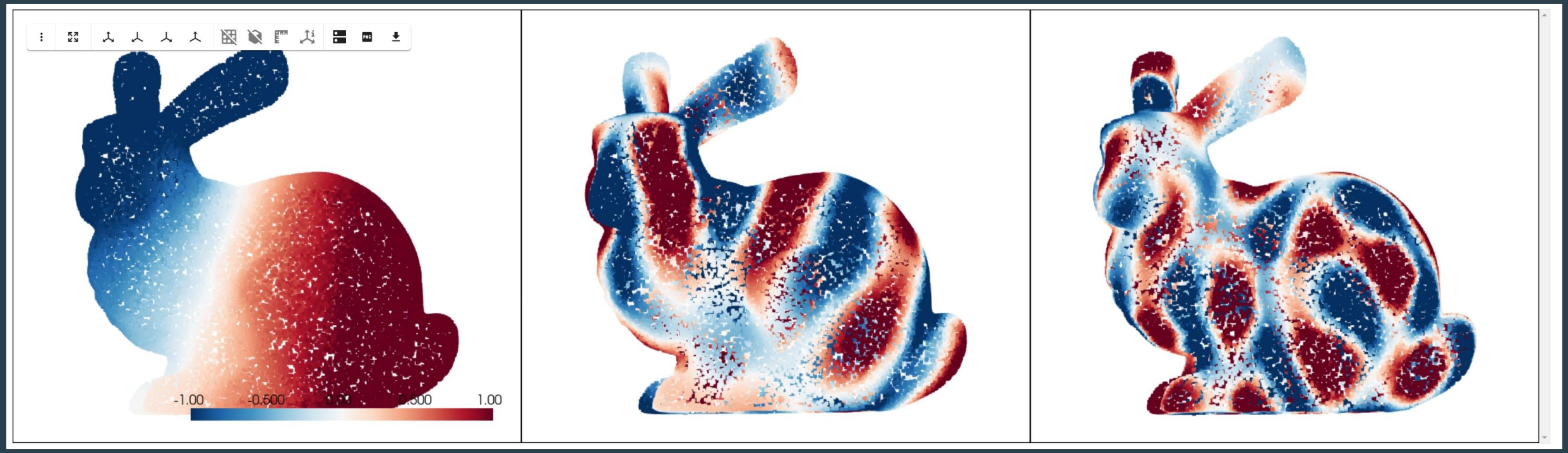
$$f(x_1, \ldots, x_K; \alpha_1, \ldots, \alpha_K) = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$$

where $\{x_k\}_{k=1}^{k=K}$ belong to the standard $K-1$ simplex, or in other words:

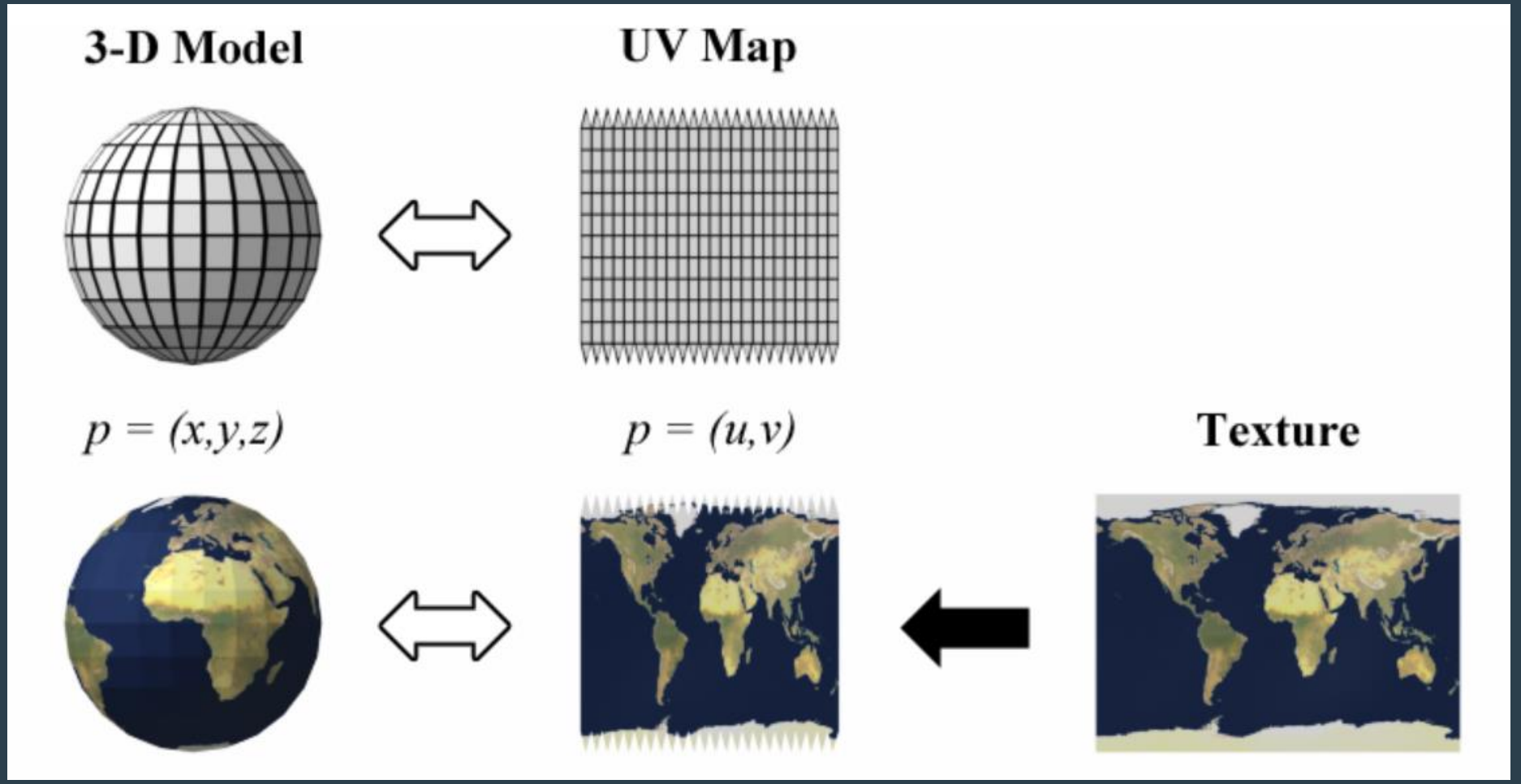
$$\sum_{i=1}^{K} x_i = 1 \text{ and } x_i \in [0,1] \text{ for all } i \in \{1, \ldots, K\}$$



Sampling on the 3D Mesh



Texture Mapping

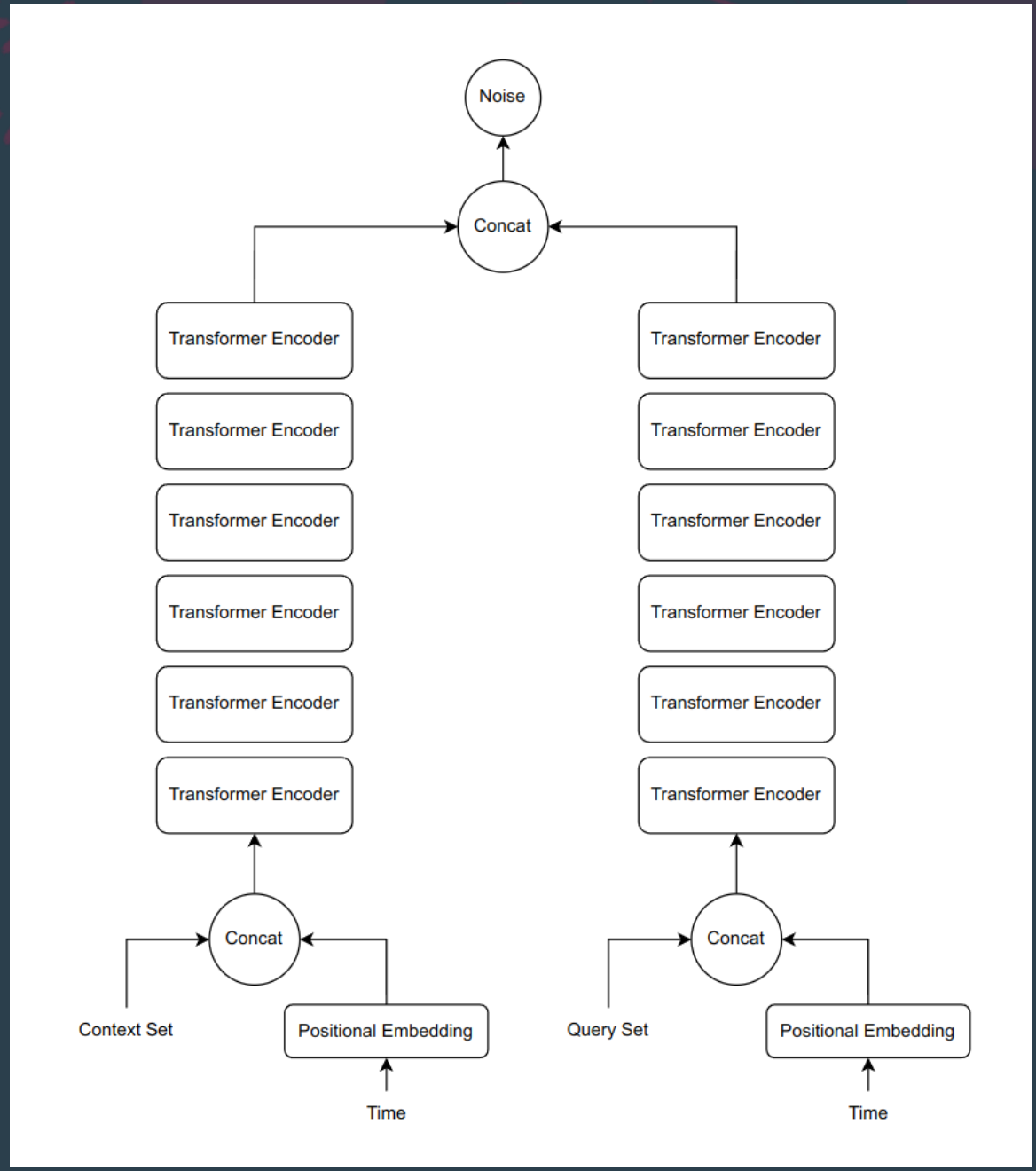


Texture Mapping

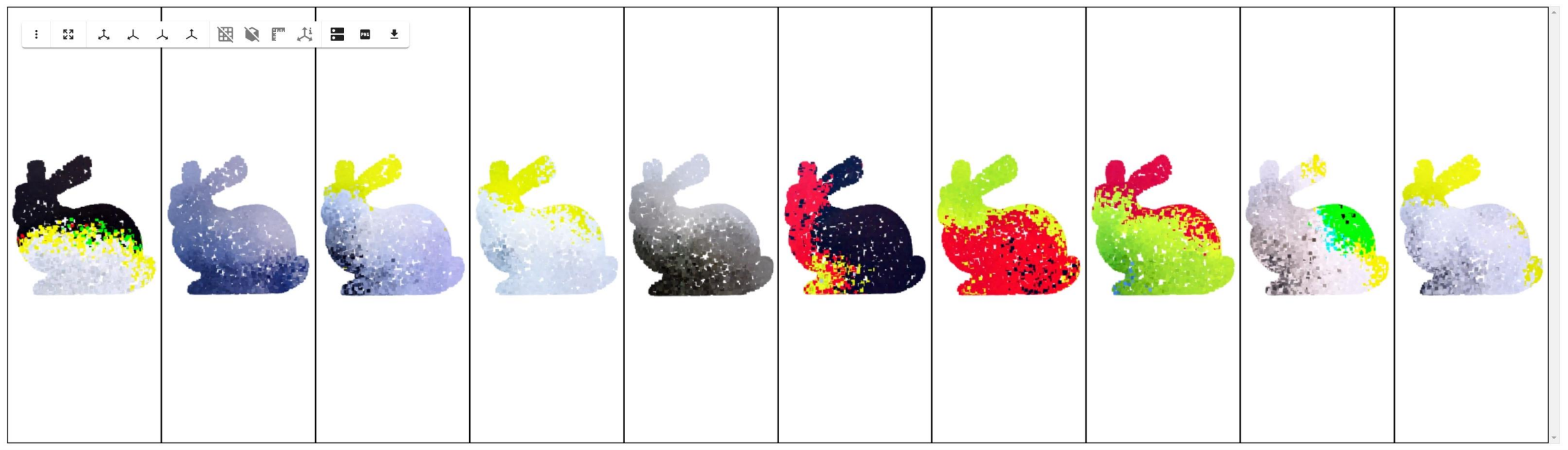
Texture Mapping

Architecture

- DDPM Sampler
- Transformer Encoder Layers for Query and Context
- MLP on the concatenation of Query and Context
- Time Embedding is implemented using 64dim of cosine embedding
- 1e-4 Adam, 40 epochs, 1000 time steps


Noise
Concat
Transformer Encoder
Transformer Encoder
Transformer Encoder
Transformer Encoder
Transformer Encoder
Transformer Encoder
Transformer Encoder
Transformer Encoder
Transformer Encoder
Transformer Encoder
Transformer Encoder
Transformer Encoder
Concat
Concat
Context Set
Positional Embedding
Time
Query Set
Positional Embedding
Time

Sampling Results



Resources

- https://arxiv.org/pdf/2006.11239
- https://arxiv.org/pdf/2305.15586
- https://arxiv.org/pdf/2201.12204
- https://www.kaggle.com/datasets/jehanbhathena/weather-dataset
- https://github.com/alecjacobson/common-3d-test-models
- https://en.wikipedia.org/wiki/Dirichlet_distribution
- https://github.com/mikonvergence/DiffusionFastForward/blob/master/notes/01-Diffusion-Theory.md