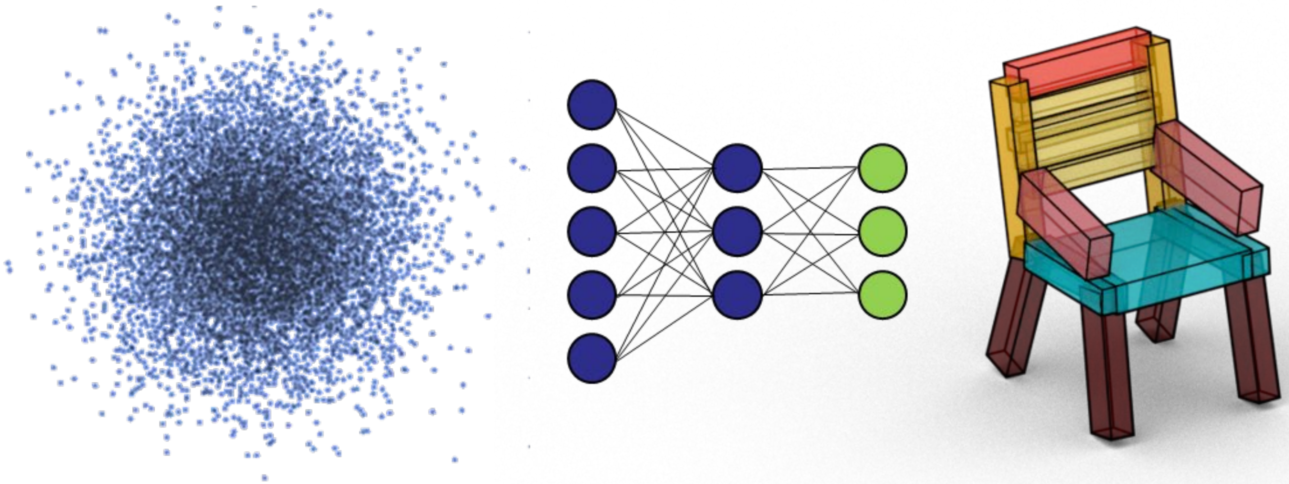
CS348n: Neural Representations and Generative Models for 3D Geometry



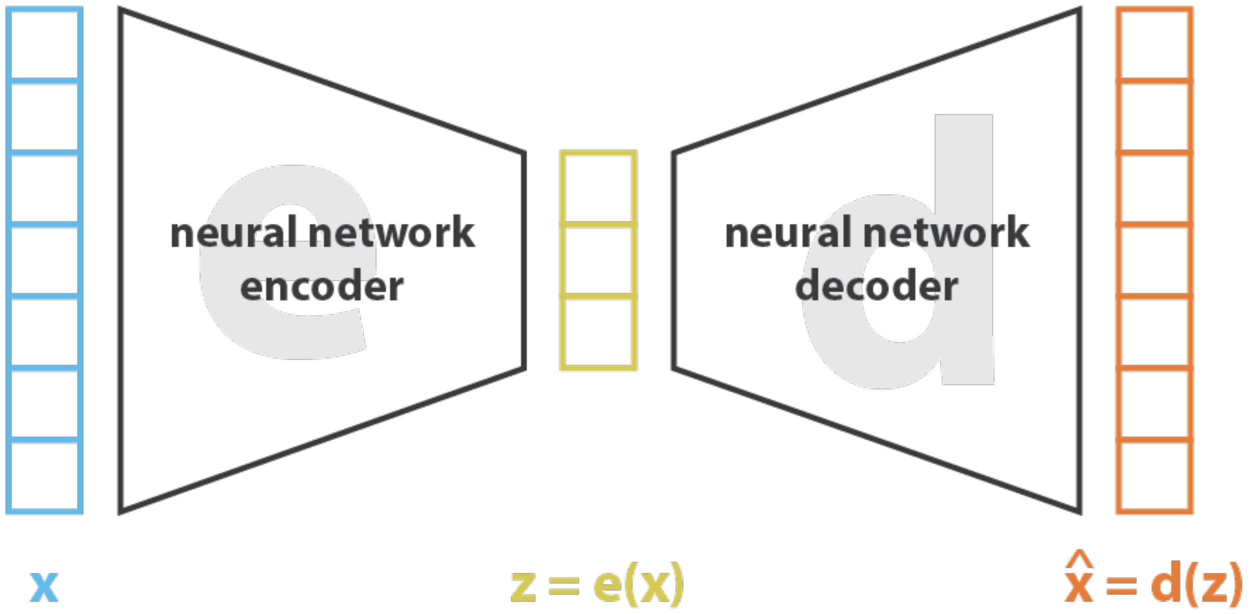
Leonidas Guibas
Computer Science Department
Stanford University

Leonidas Guibas Laboratory

Geometric Computing

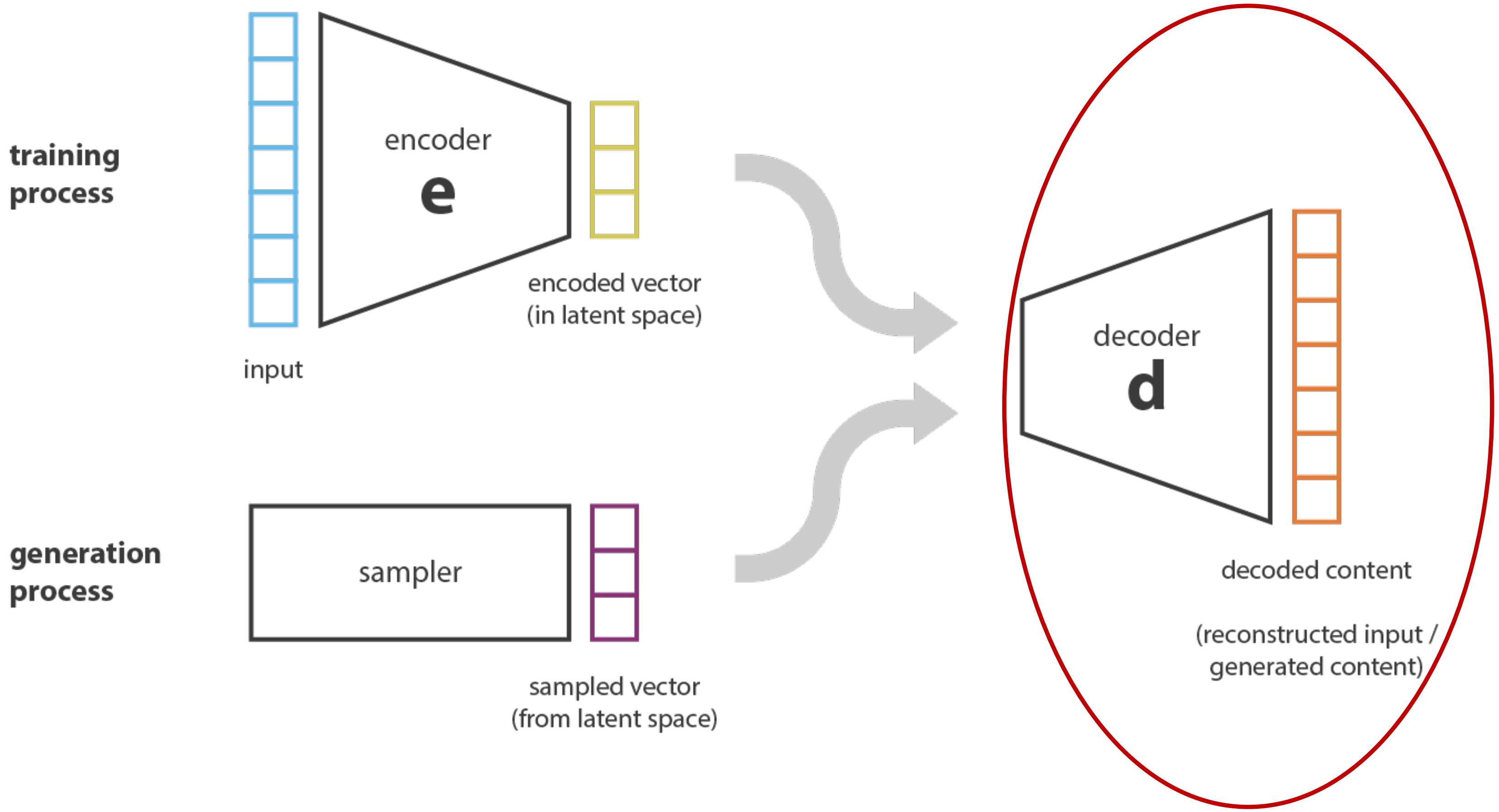
Last Time: Variational AutoEncoders, Neural Implicits

Autoencoders



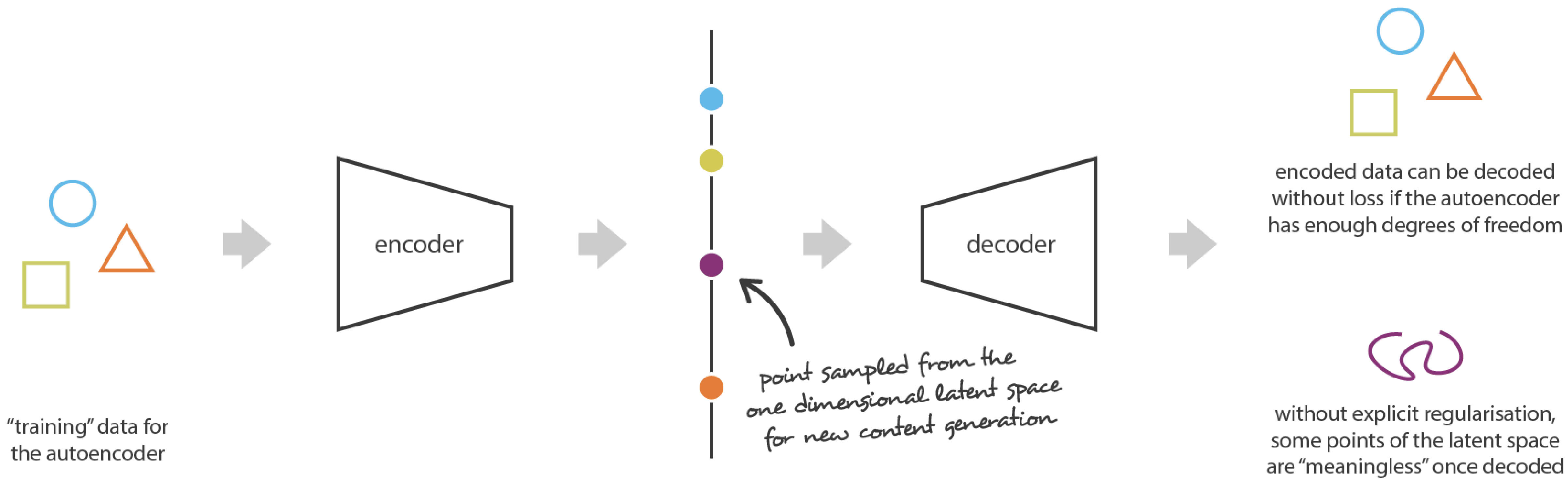
$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$

Autoencoders for Content Generation



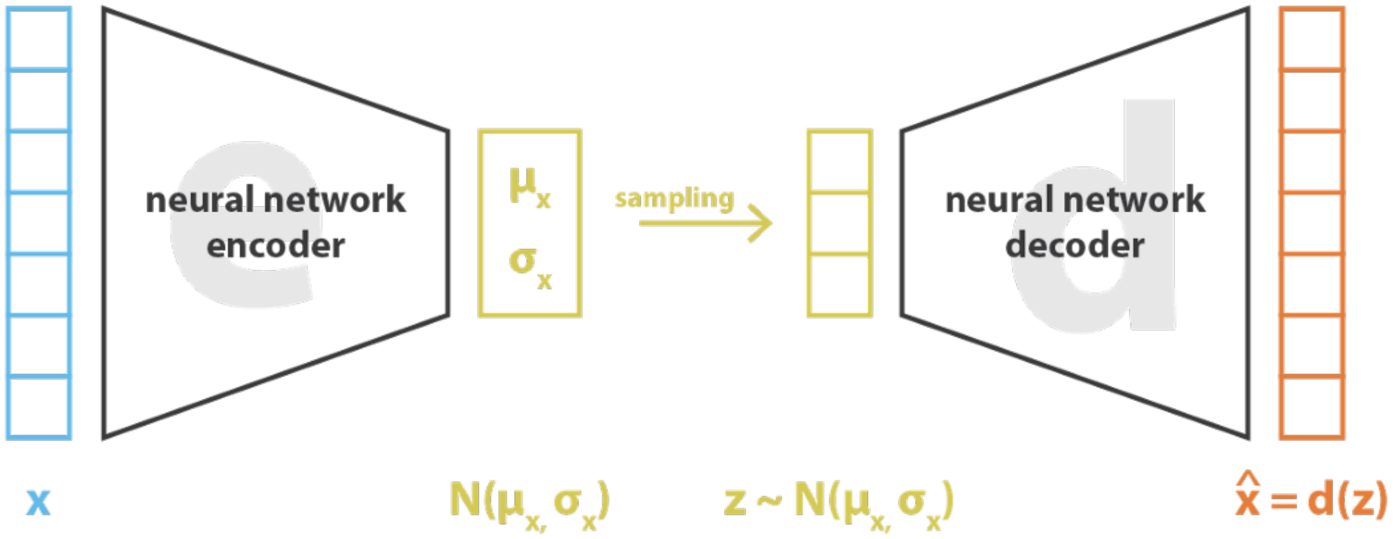
Severe overfitting

Autoencoders for Content Generation



An autoencoder is solely trained to encode and decode with as small loss as possible, no matter how the latent space is organized

Regularize the Distribution in Latent Space

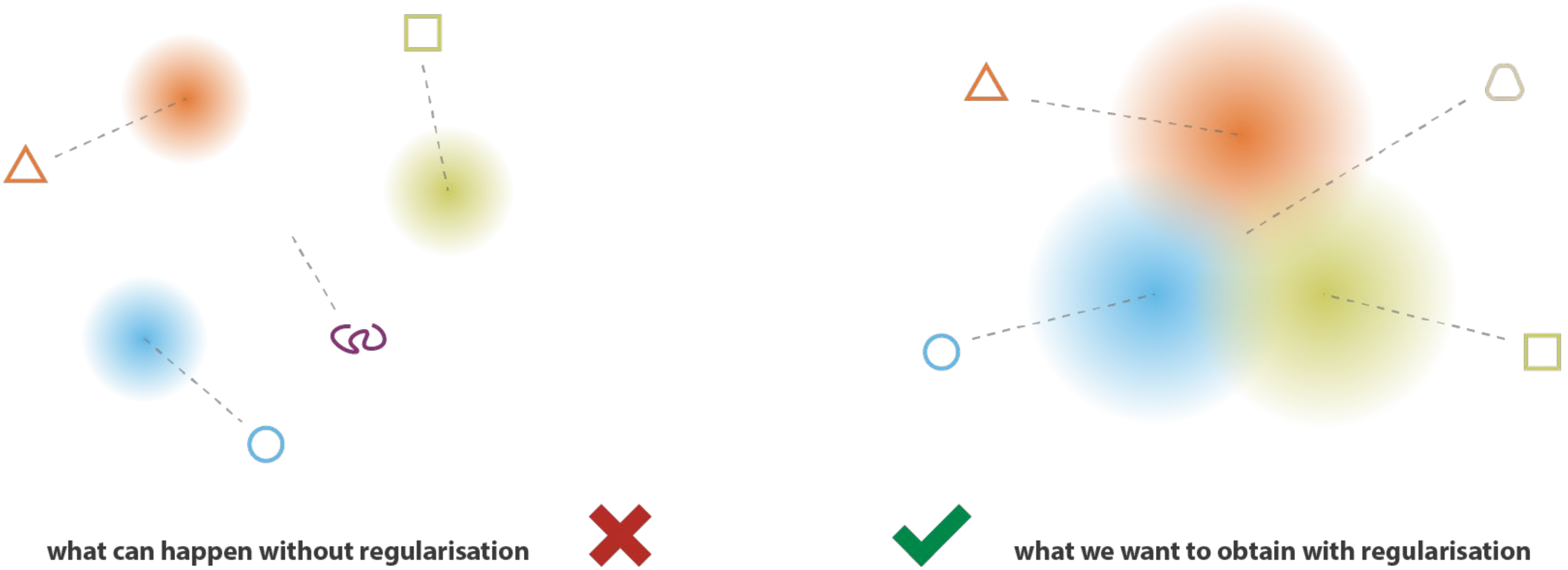


$$\text{loss} = \| x - \hat{x} \|^2 + \mathrm{KL}[\, N(\mu_x, \sigma_x), N(0, I) \,] = \| x - d(z) \|^2 + \mathrm{KL}[\, N(\mu_x, \sigma_x), N(0, I) \,]$$

Make the encoder probabilistic with a latent space distribution like a simple Gaussian

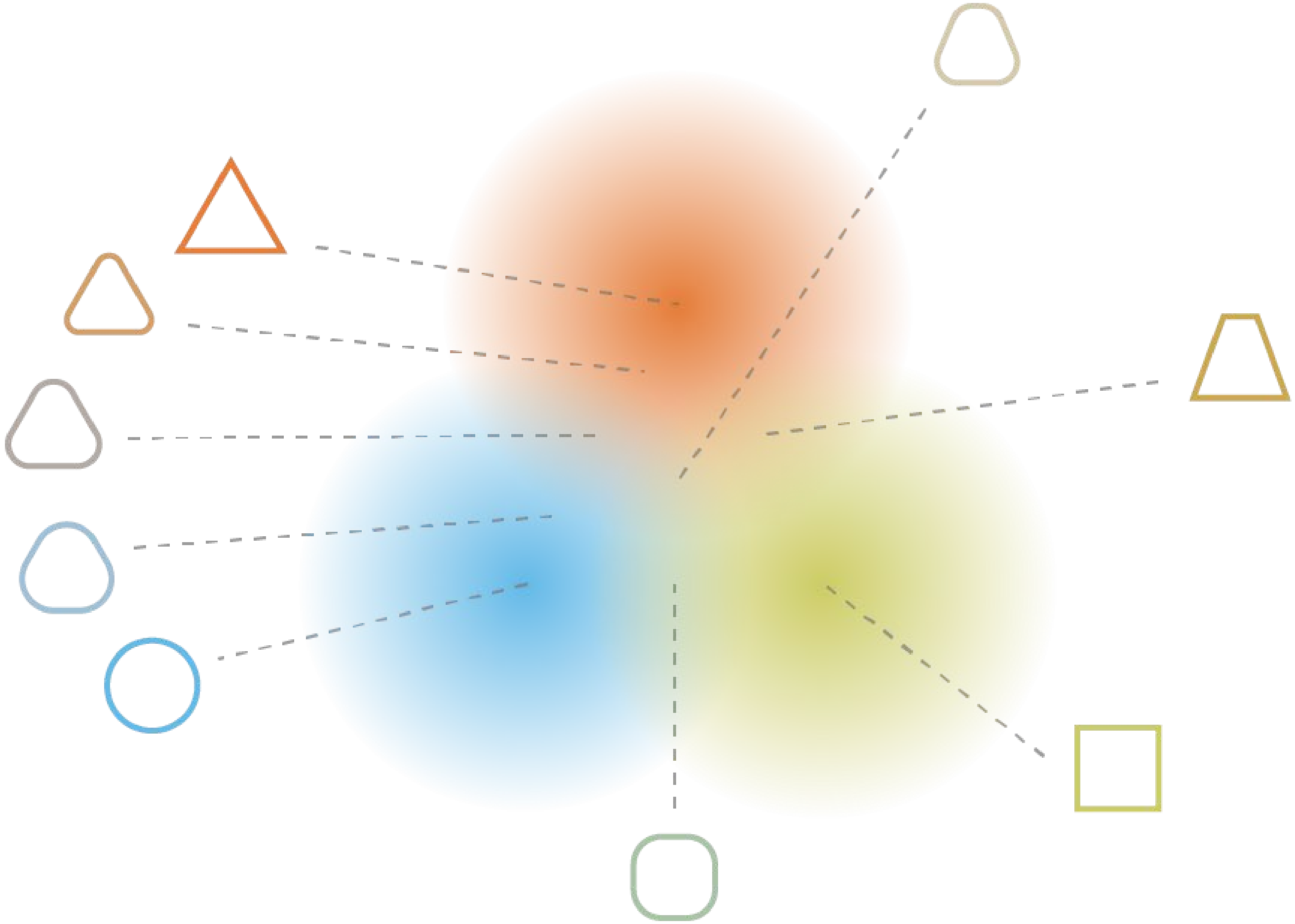
Add a second loss measuring distribution distance (via the Kulback-Leibler divergence)

The Effect of Regularization



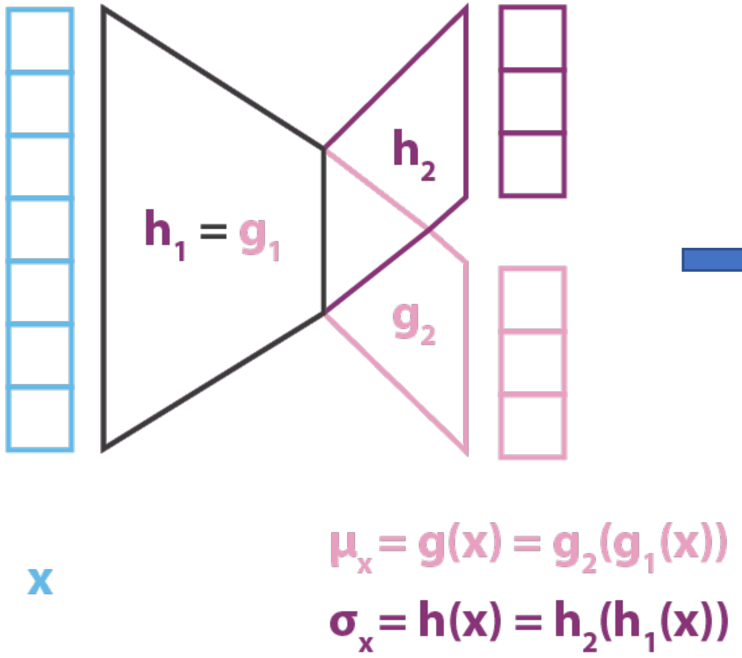
Overfitting with “punctual” distributions

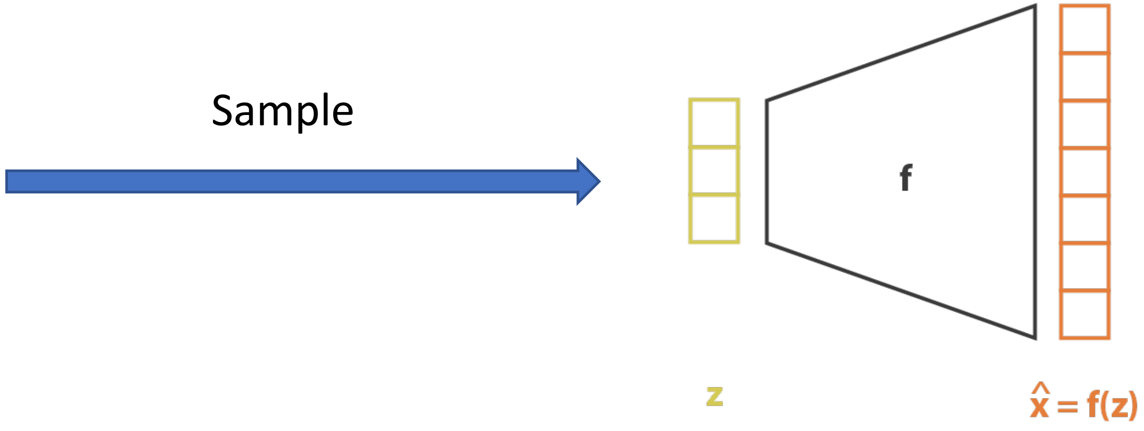
The Effect of Regularization



Create smooth gradients over the information encoded in the latent space

Variational AutoEncoder (VAE)





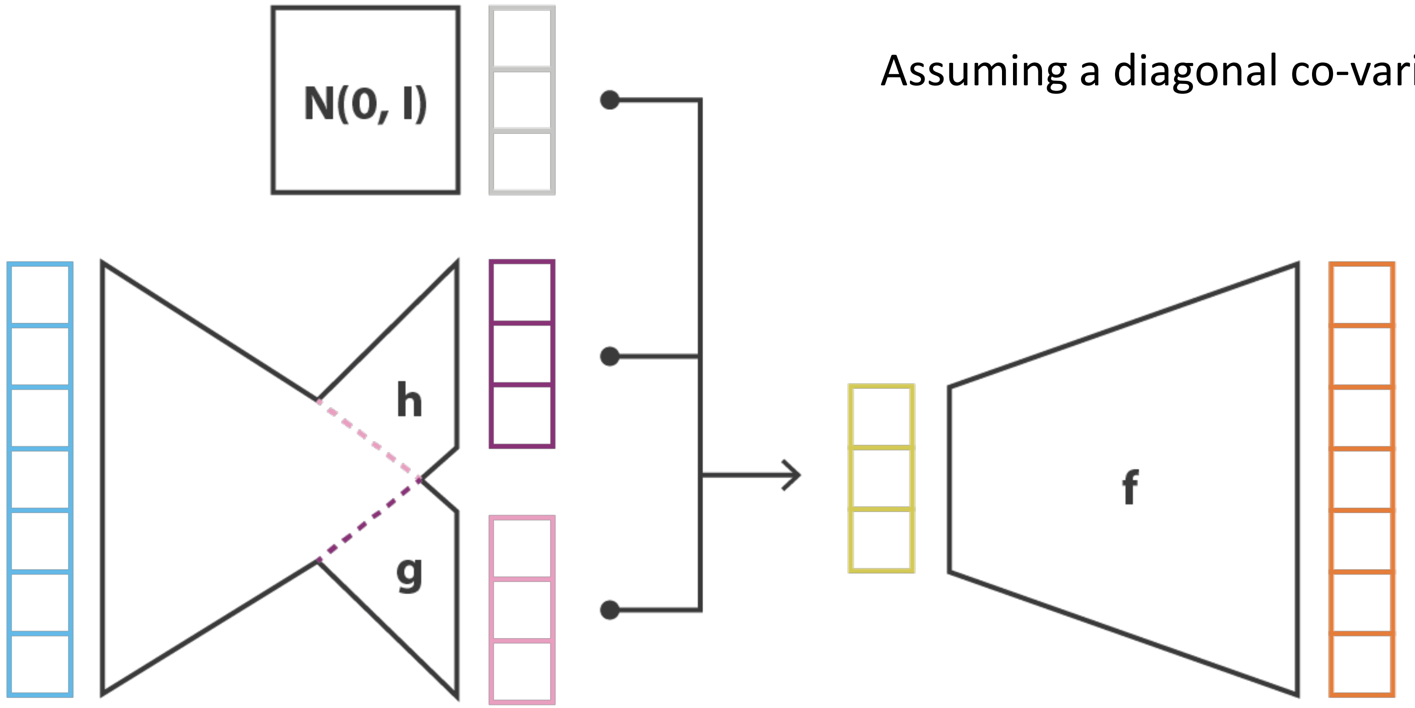
$g(x) = g_2\left(g_1(x)\right) \quad h(x) = h_2\left(h_1(x)\right) \quad g_1(x) = h_1(x)$

How to train:

Sampling is a problem w. back propagation

The Final VAE

Assuming a diagonal co-variance matrix



x

$\mu_x = g(x)$

$\sigma_x = h(x)$

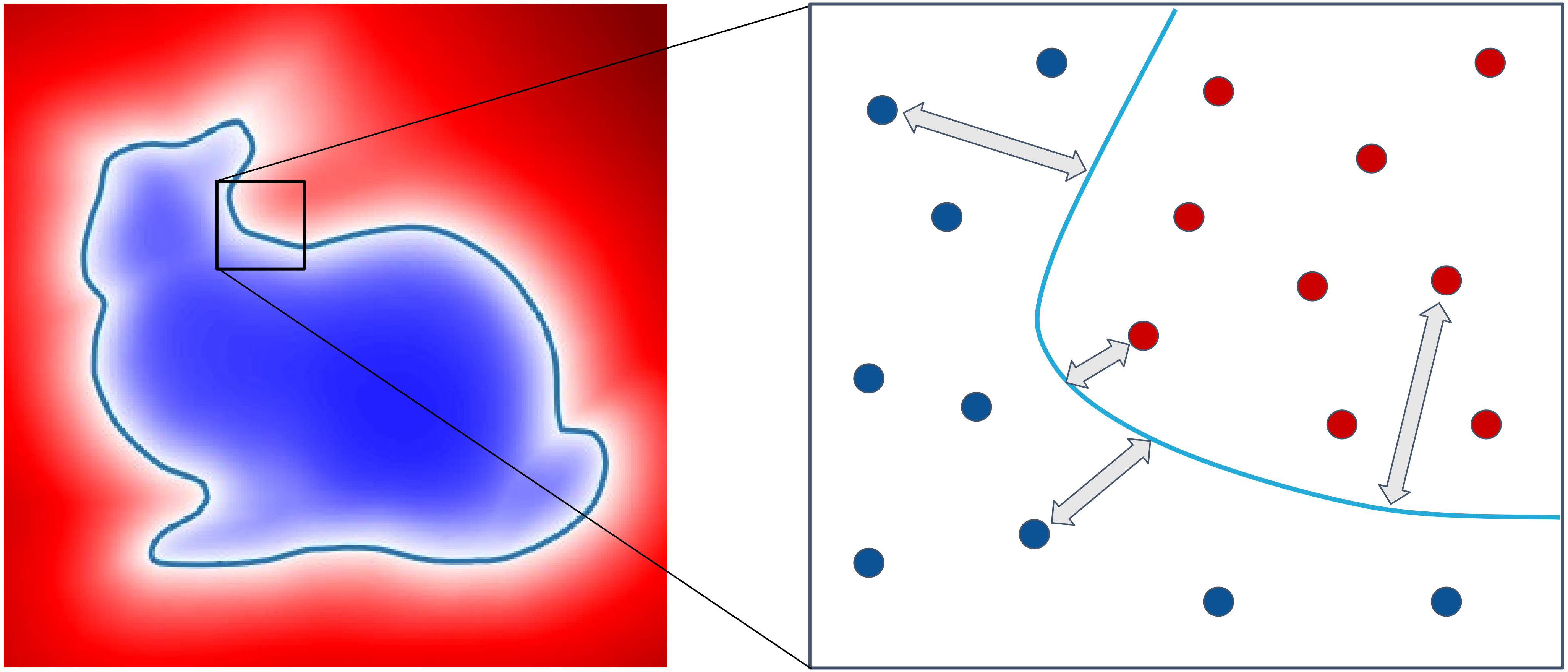
$\zeta \sim N(0, I)$

$z = \sigma_x \zeta + \mu_x$

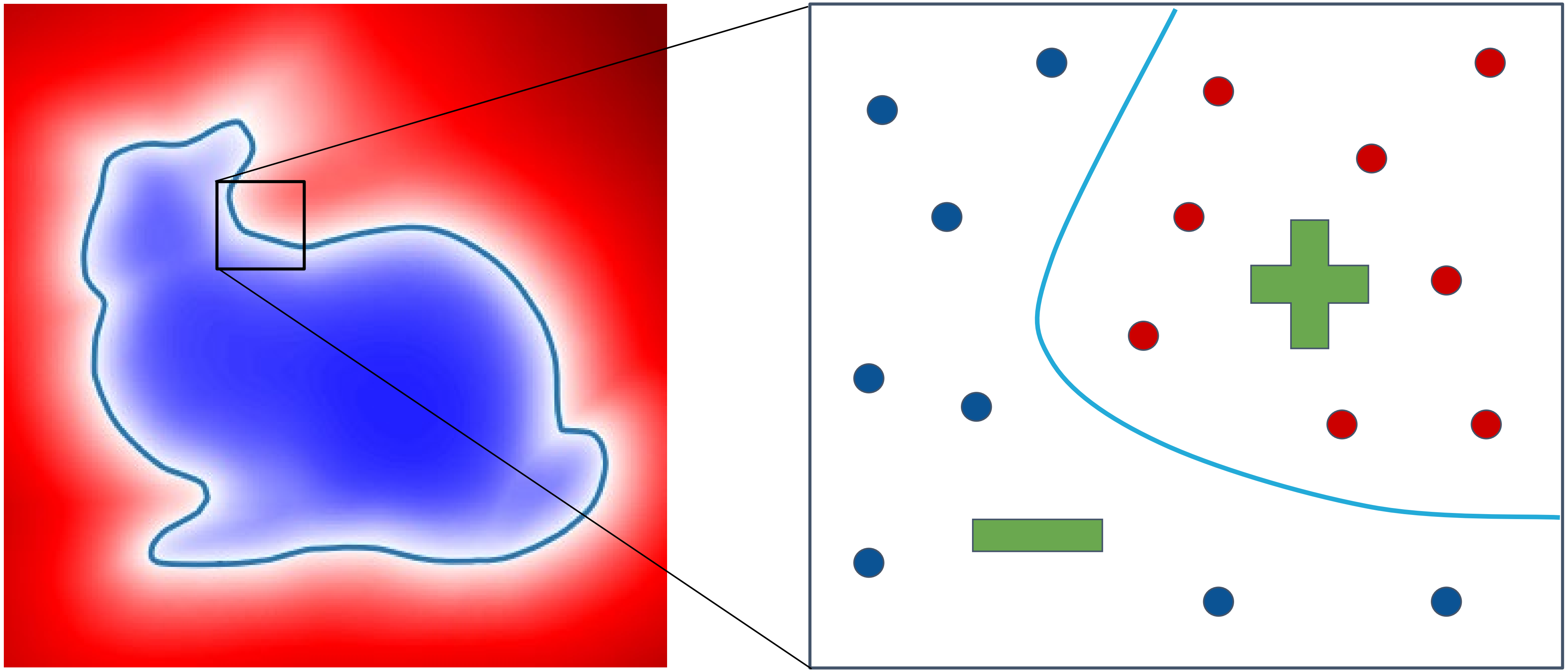
$\hat{x} = f(z)$

$$\text{loss} = C \,||\, x - \hat{x} \,||^2 + KL[\, N(\mu_x, \sigma_x), N(0, I) \,] = C \,||\, x - f(z) \,||^2 + KL[\, N(g(x), h(x)), N(0, I) \,]$$

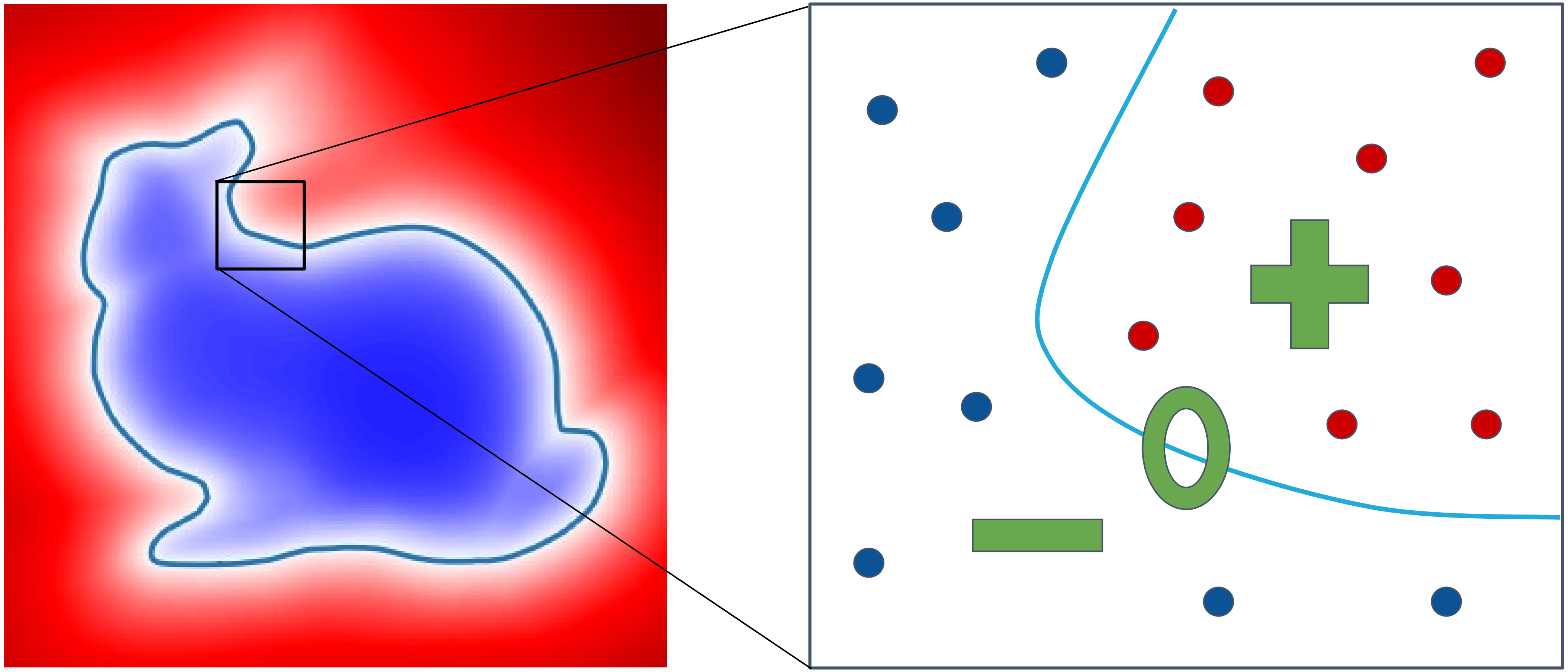
Shapes via Signed Distance Functions



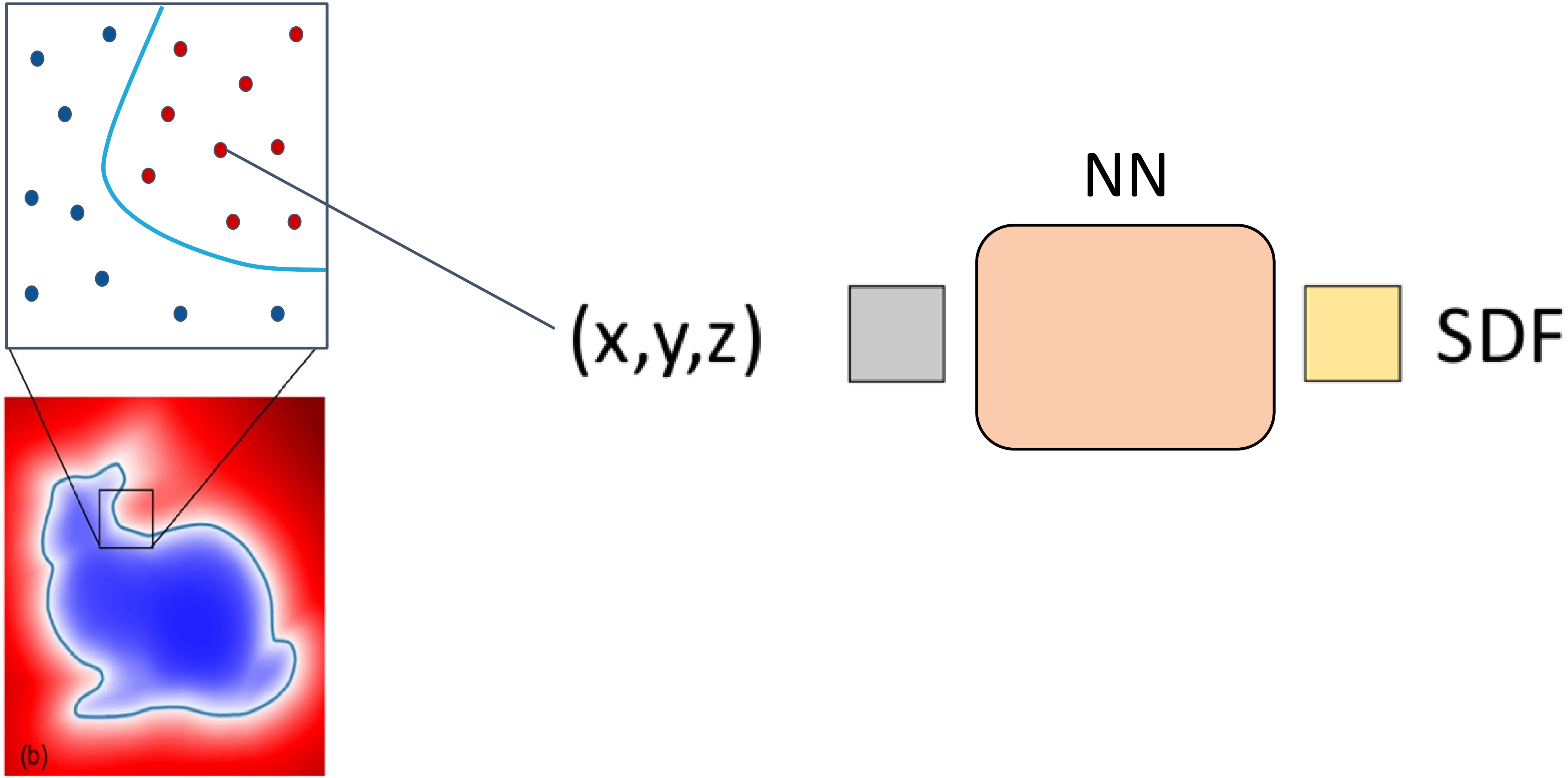
Signed Distance Function



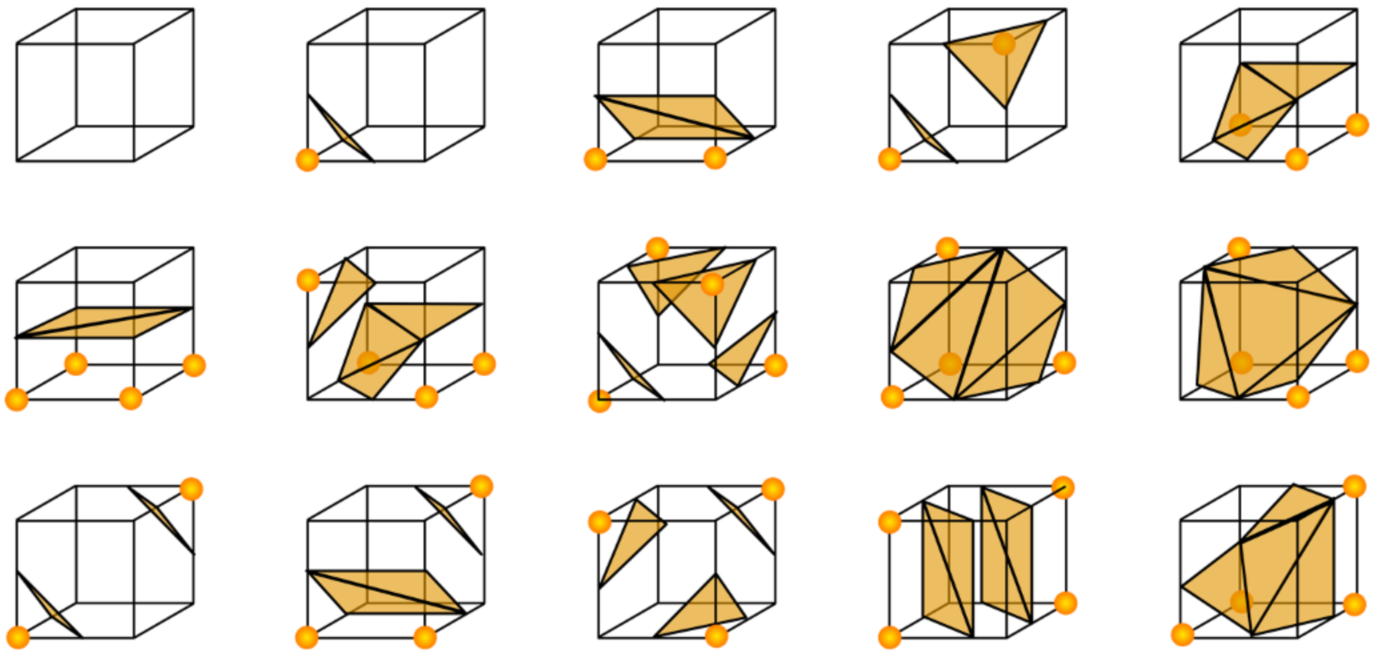
Shapes via Signed Distance Functions



Regression of Continuous SDF via a NN

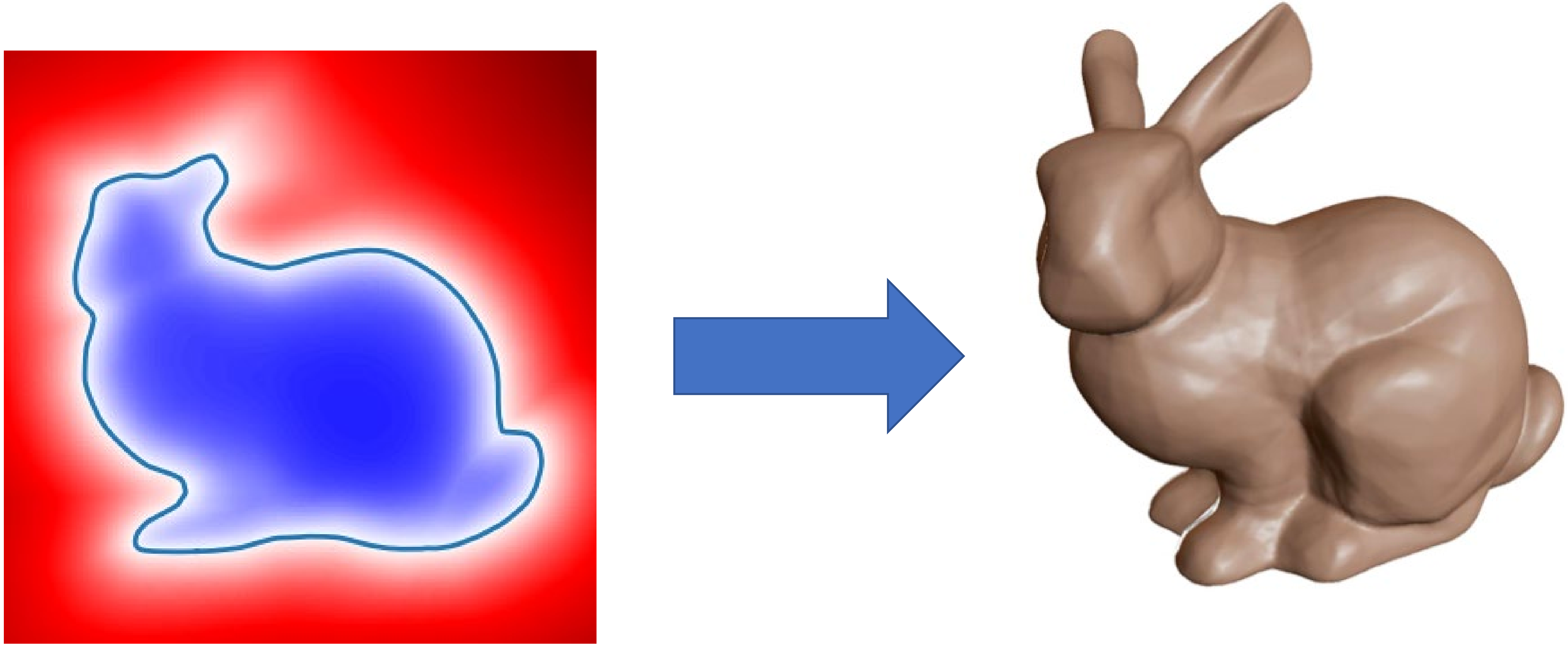


Can then Reconstruct via Marching Cubes

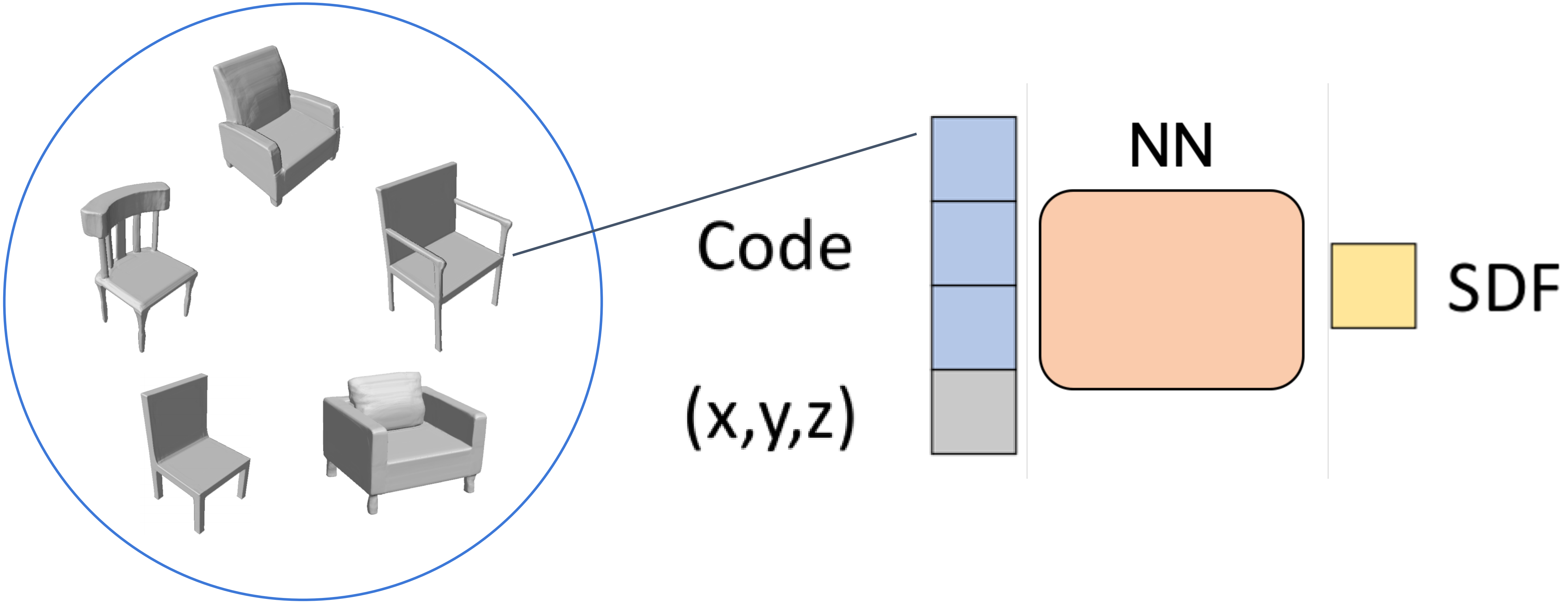


Lorensen et al., 1987

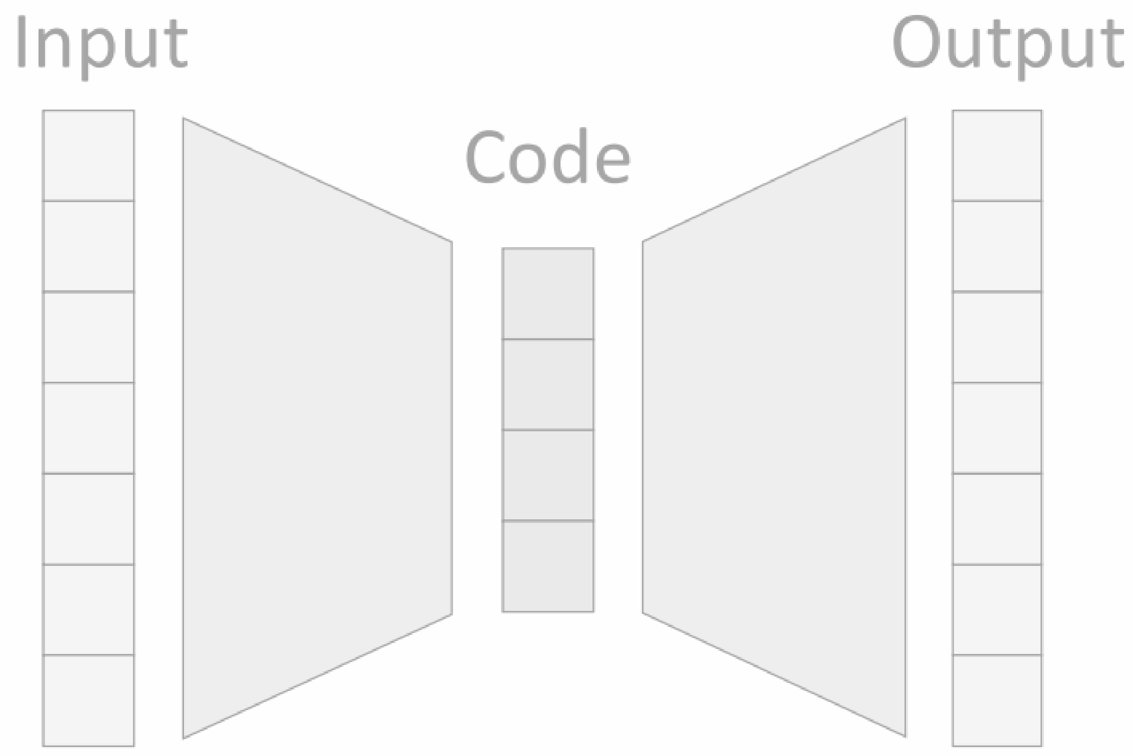
Implicit to Explicit Shape Representation



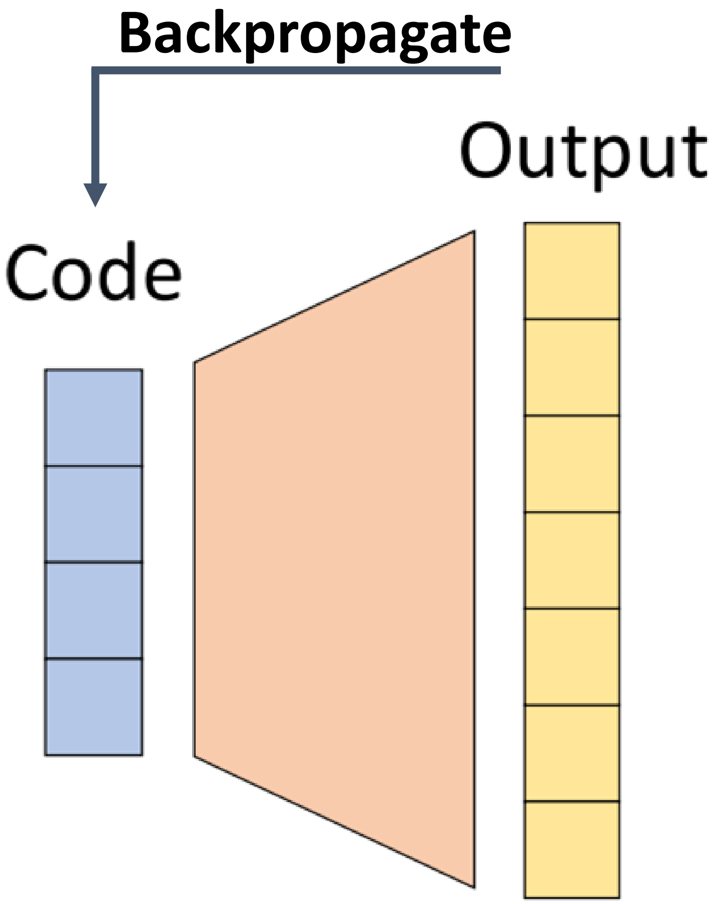
Coding Multiple Shapes



Auto-Decoder

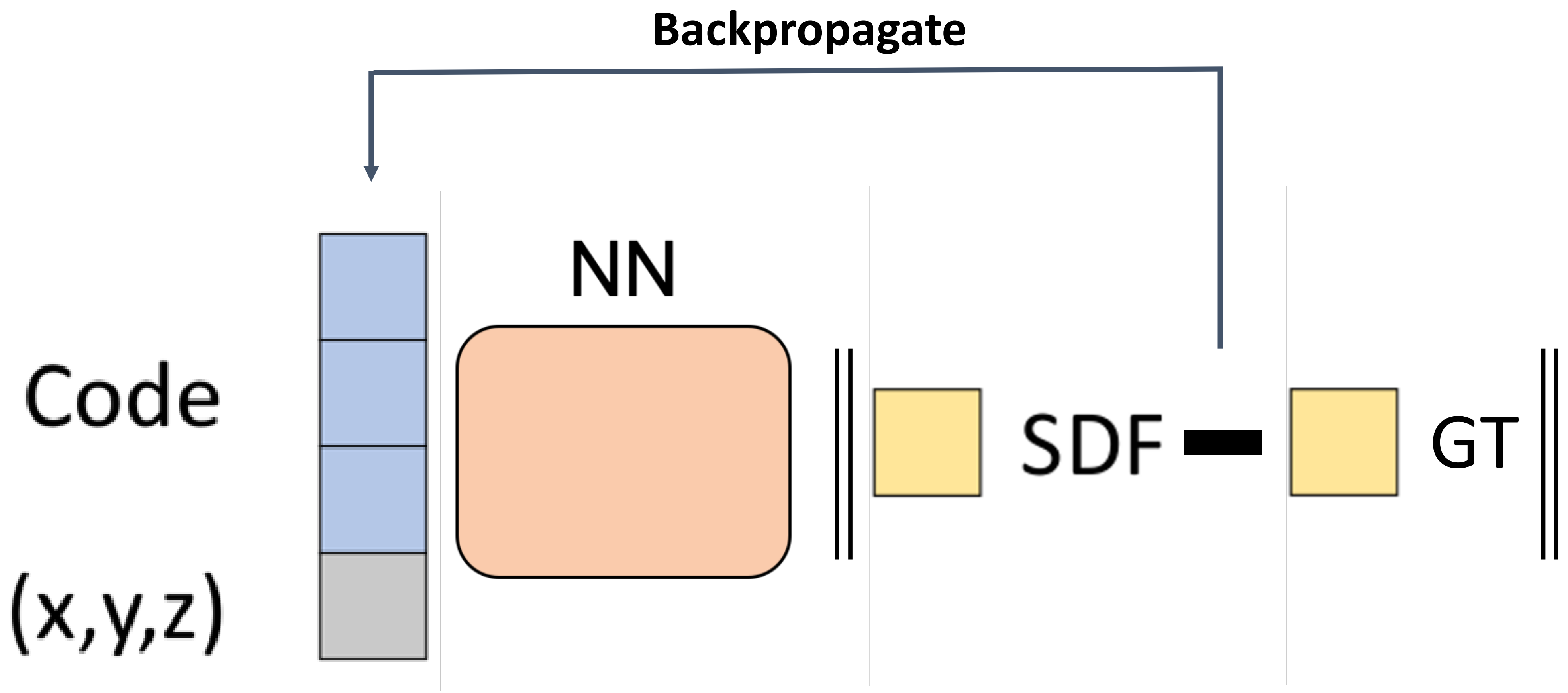


Auto-Encoder

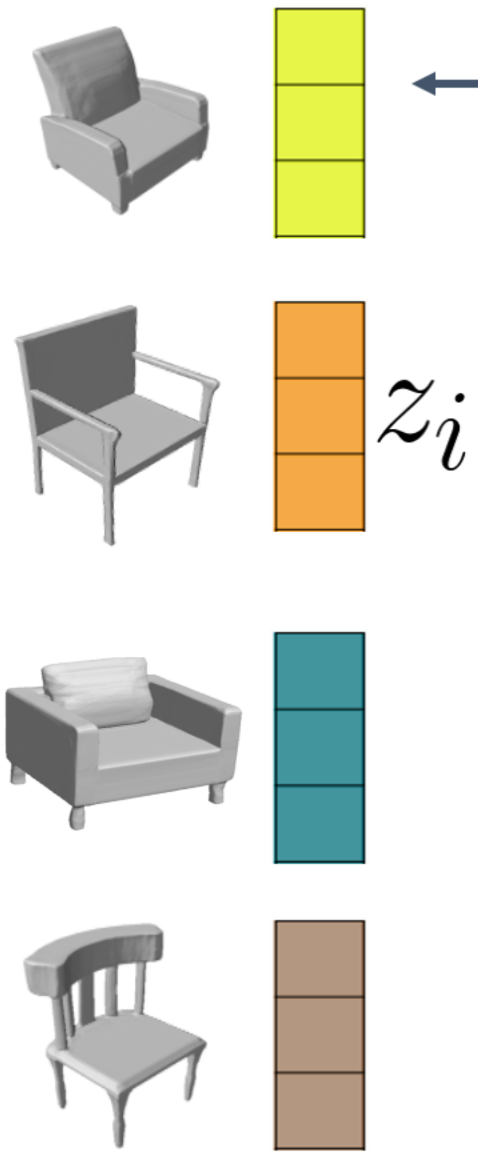


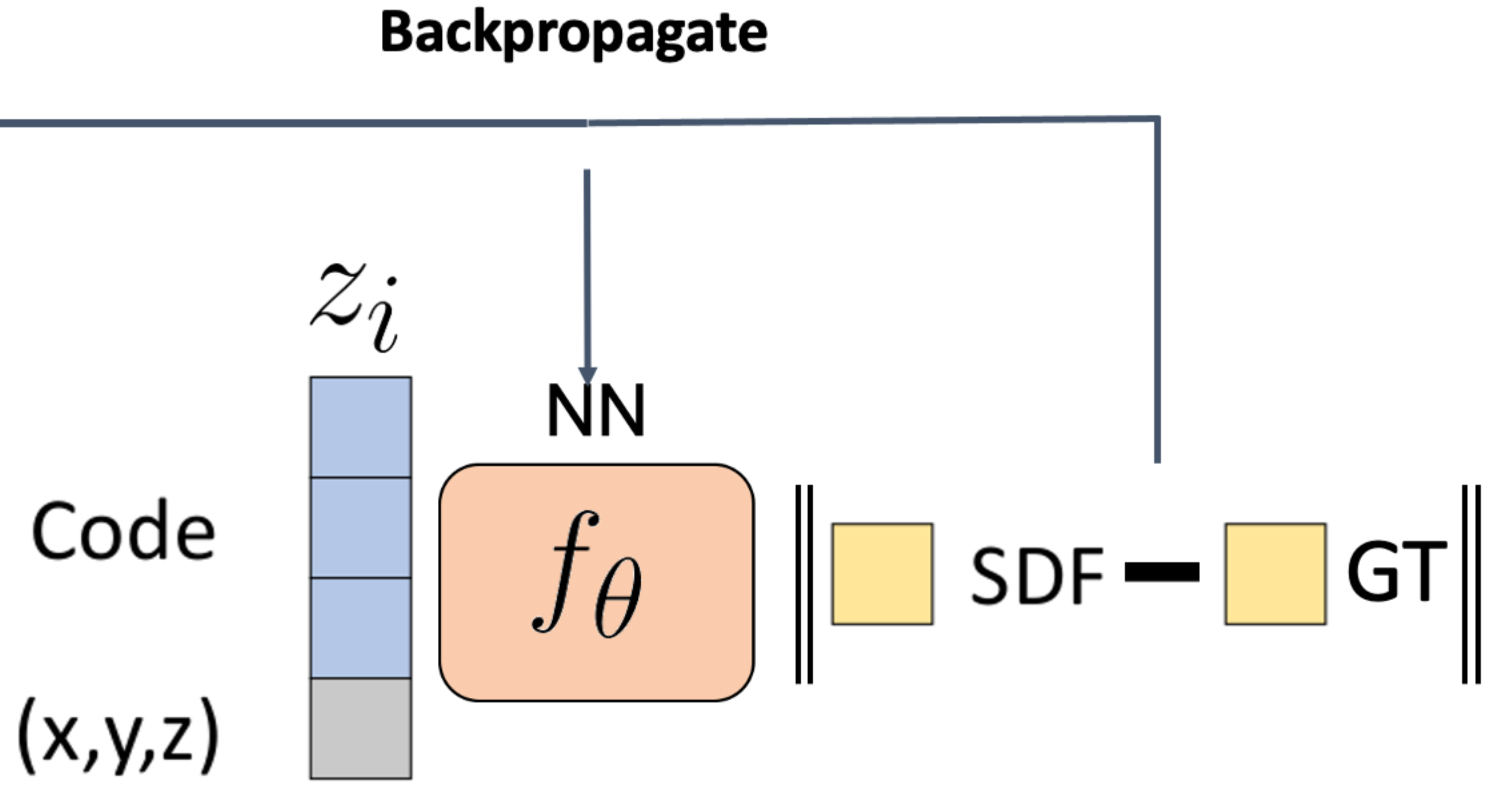
Auto-Decoder

Auto-Decoder

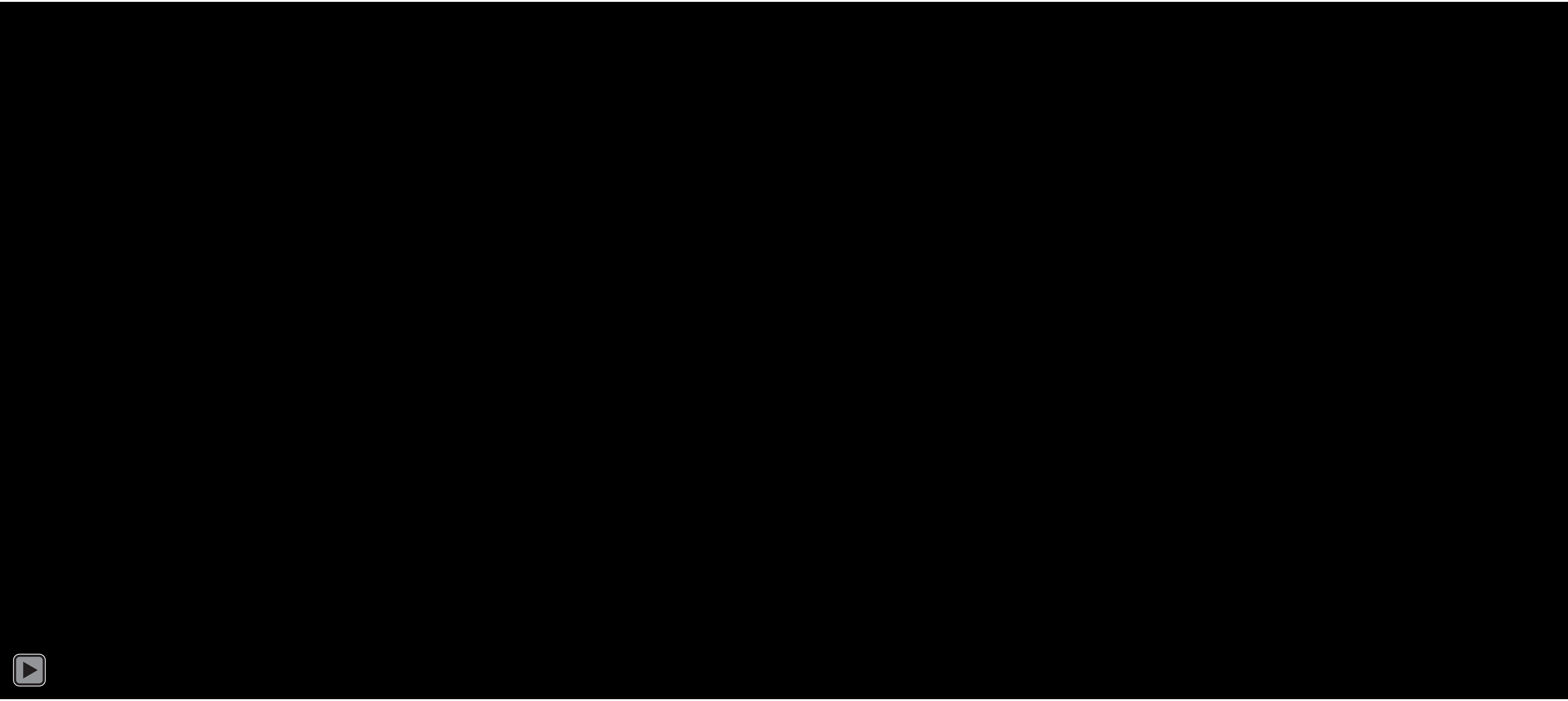


Auto-Decoder Training

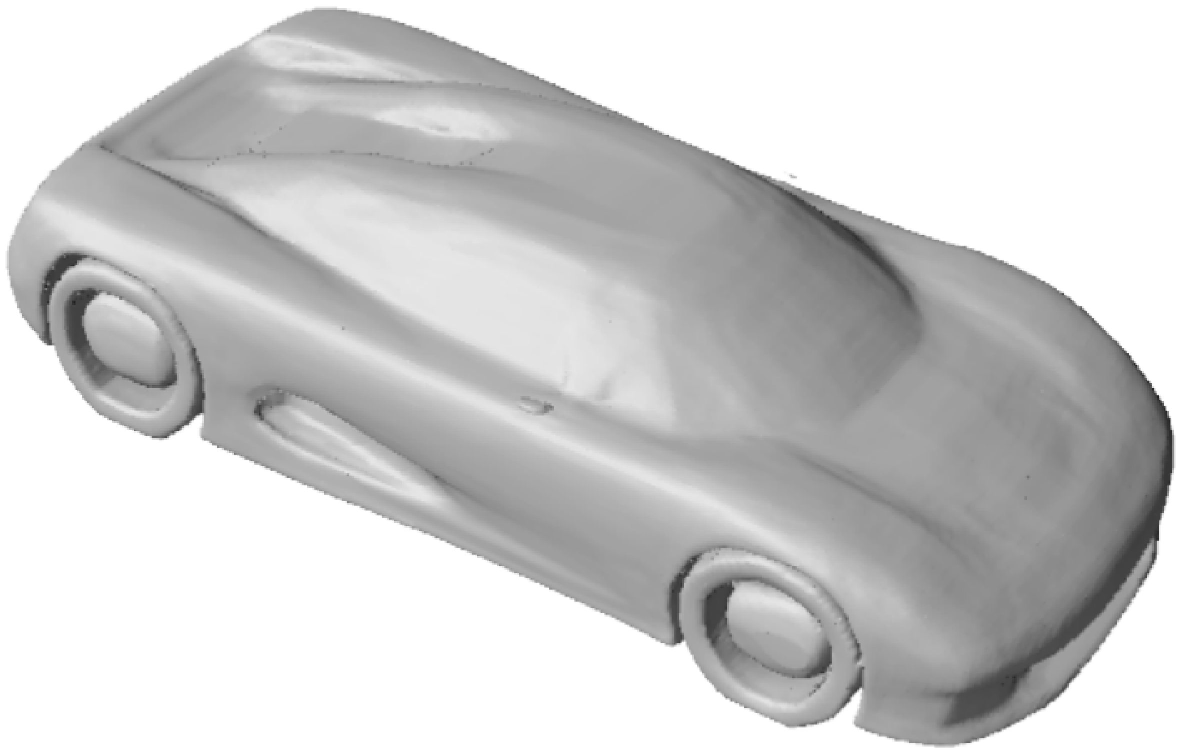




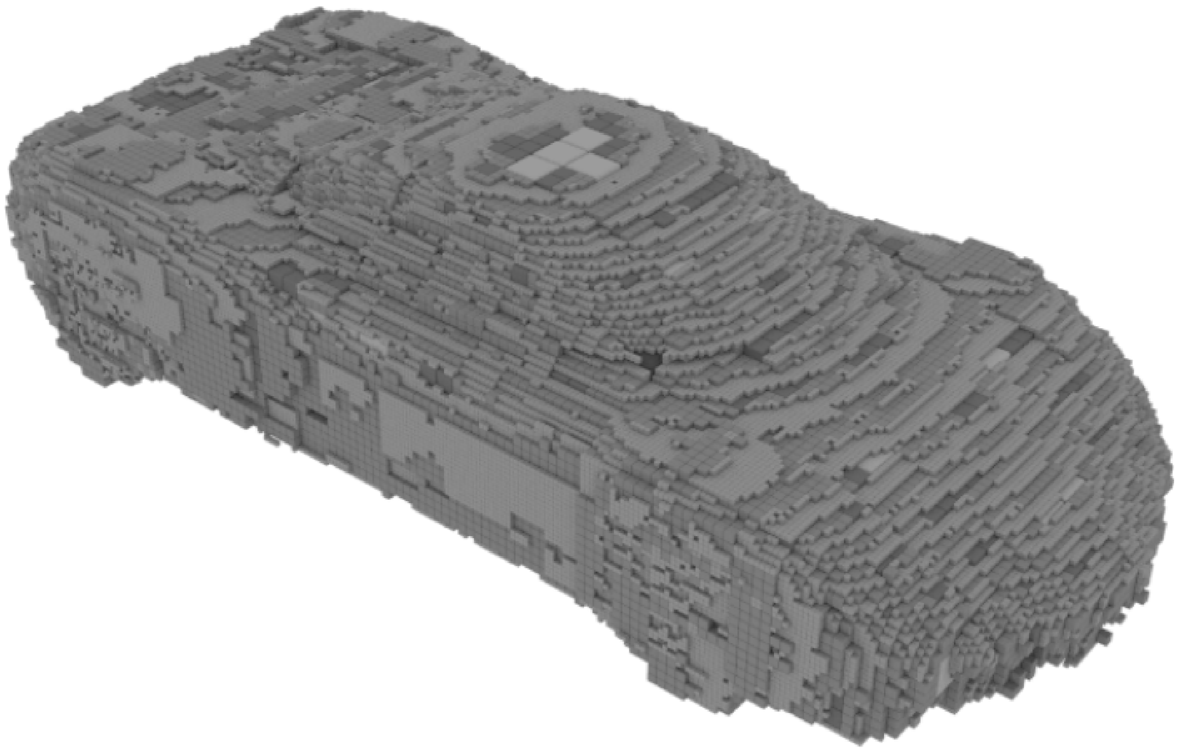
$$\underset{\theta, \{\boldsymbol{z}_i\}_{i=1}^{N}}{\arg\min} \sum_{i=1}^{N} \left(\sum_{j=1}^{K} \mathcal{L}(f_\theta(\boldsymbol{z}_i, \boldsymbol{x}_j), s_j) \right)$$



Results: Comparison with Octree-Based



Our
Reconstruction



Octree Based

Results: Comparisons with Direct Mesh-Based

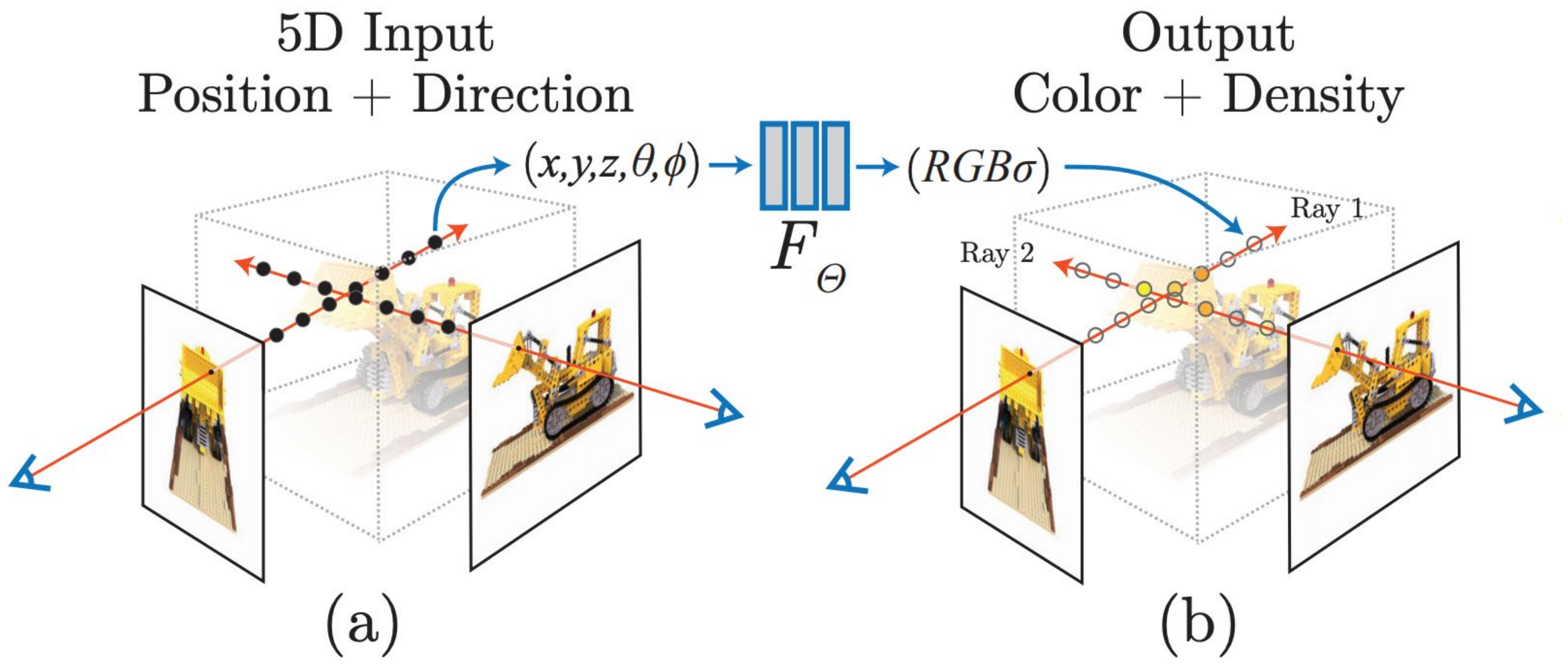
Ground Truth

Our Reconstruction

Atlasnet (25 Patches)

Atlasnet (1 Patch)

DeepSDF Extensions: NeRF



Mildenhall et al. 2020

Neural Parametrics:
AtlasNet

Implicit Curves and Surfaces via Functions

- Kernel of a scalar function $f : \mathbb{R}^m \to \mathbb{R}$

  - Curve in 2D: $S = \{x \in \mathbb{R}^2 | f(x) = 0\}$
  - Surface in 3D: $S = \{x \in \mathbb{R}^3 | f(x) = 0\}$

$x := (a, b)$

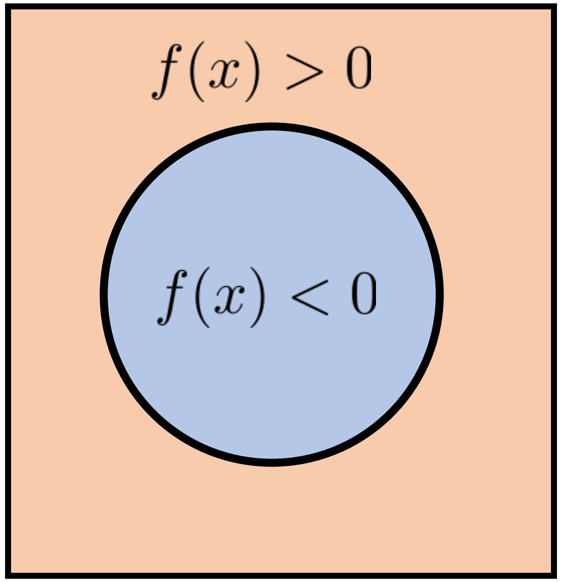
$a^2 + b^2 - 1$

- Space partitioning

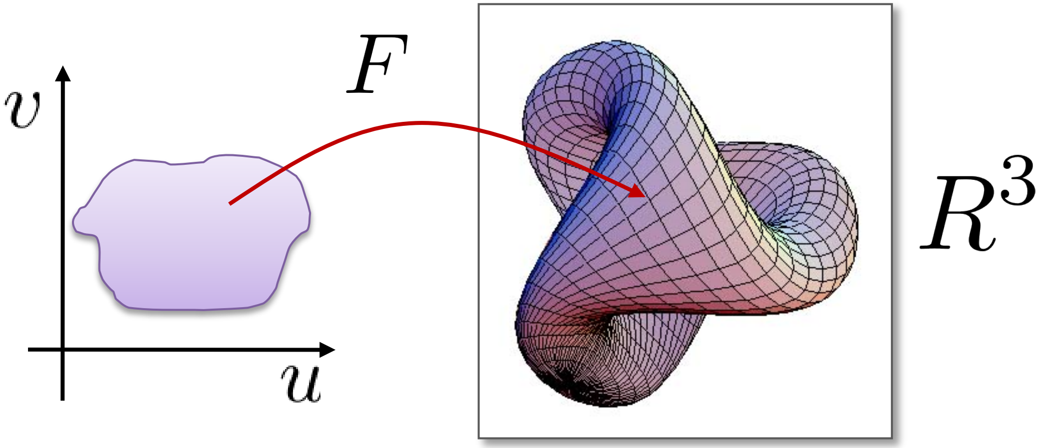
$\{x \in \mathbb{R}^m | f(x) > 0\}$ Outside

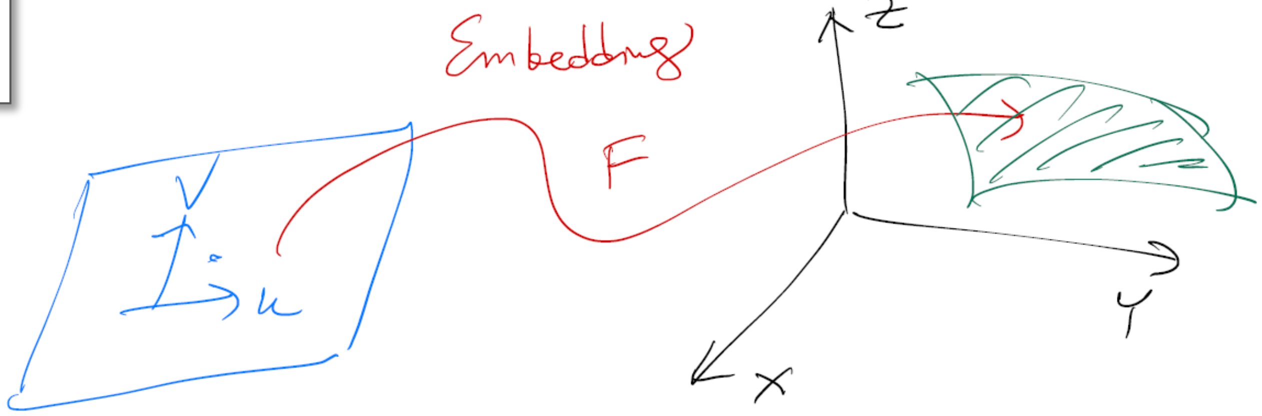
$\{x \in \mathbb{R}^m | f(x) = 0\}$ Curve/Surface

$\{x \in \mathbb{R}^m | f(x) < 0\}$ Inside



Parametric Curves and Surfaces via Functions





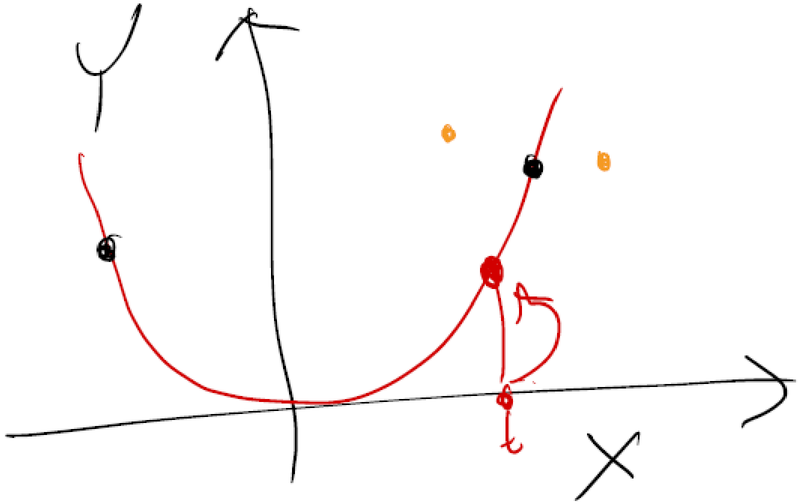
$$F(u,v) \equiv (X(u,v), Y(u,v), Z(u,v))$$

$$F : R^2 \longrightarrow R^3 \leftarrow \text{ambient space}$$

$\uparrow$ parameter space

These parametric mappings can be explicit functions, but can also be neural networks

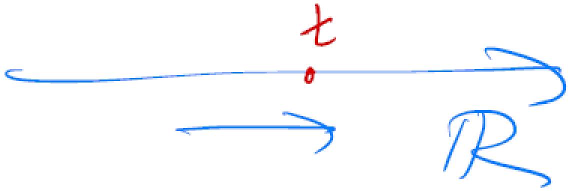
Implicits and Parametrics are Complementary



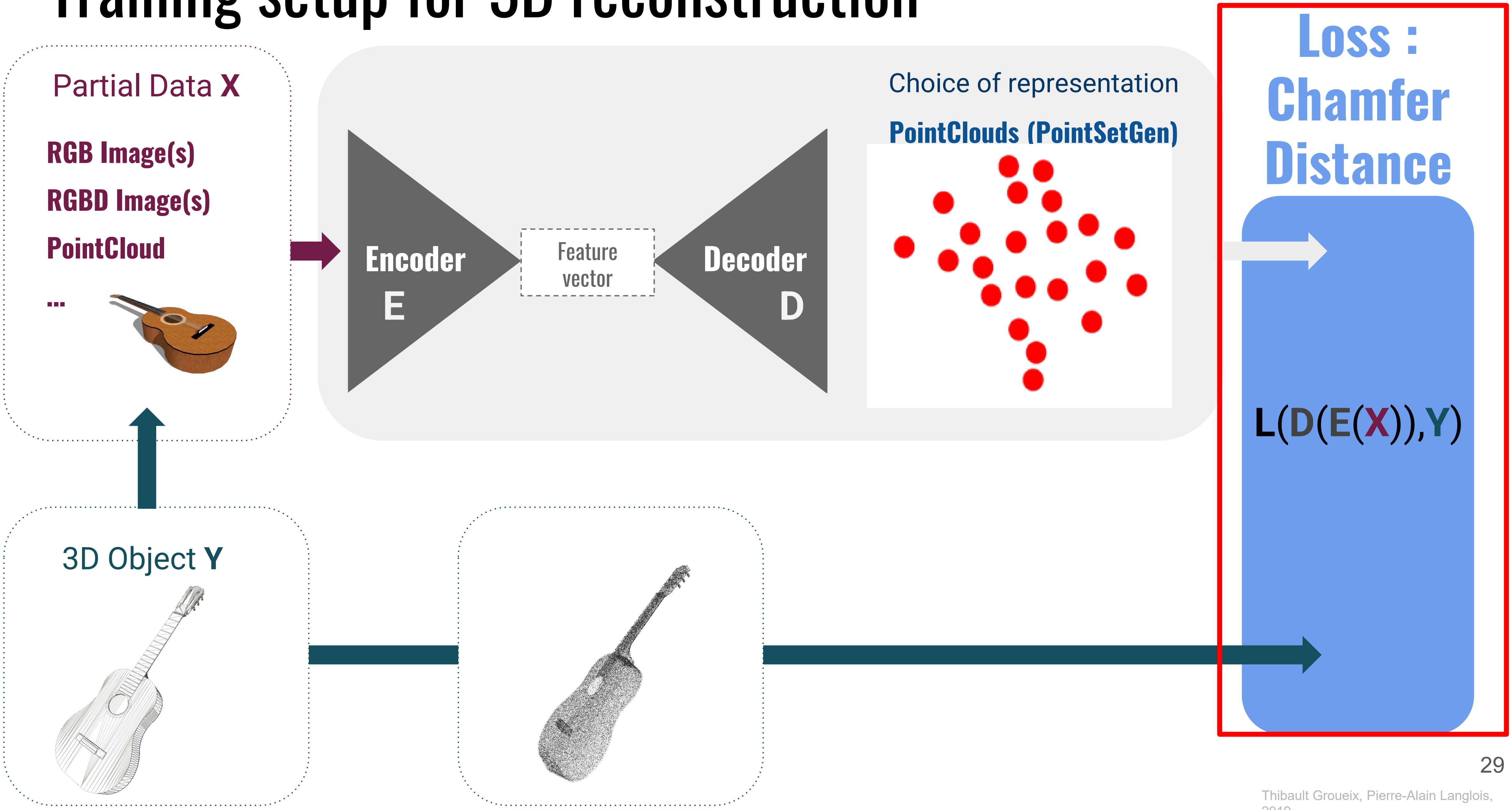
$$\left.\begin{array}{l} Y = x^2 \\ Y - x^2 = 0 \end{array}\right\} \text{implicit}$$

$$\boxed{F(t) = (t, t^2)} \quad \text{Parametric}$$

$$X(t) \quad Y(t)$$



Training setup for 3D reconstruction



Thibault Groueix, Pierre-Alain Langlois, 2019

Generating points

Encoder
E

Decoder
D

Test Shape

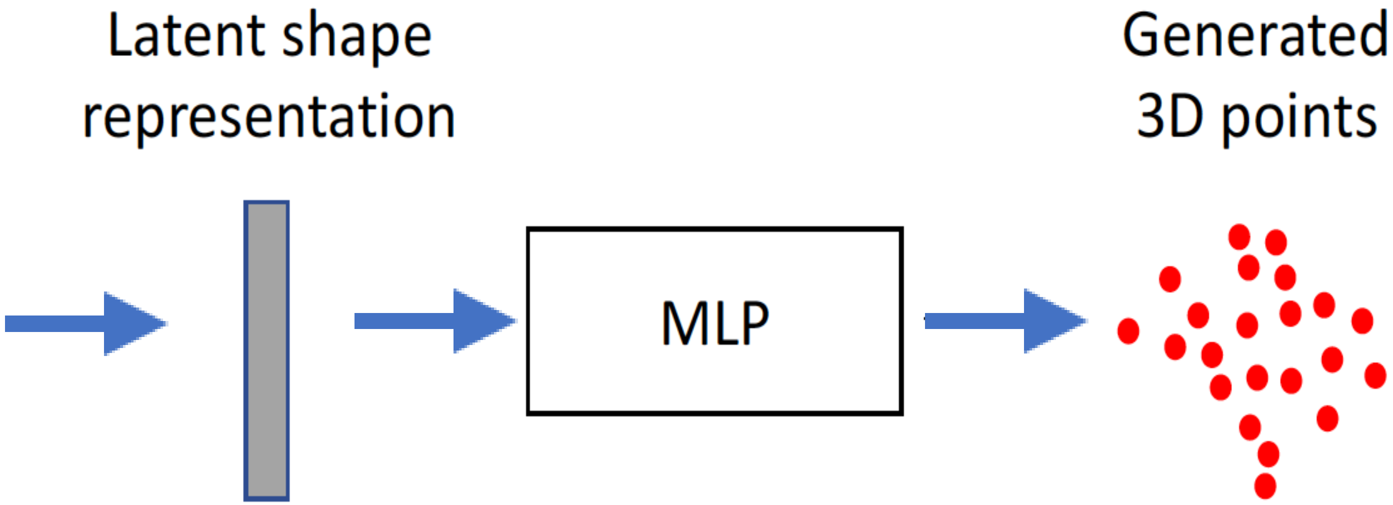
AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation
Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, Mathieu Aubry
https://arxiv.org/abs/1802.05384

Generating points

Encoder
E

Decoder
D

Test Shape



Generating points

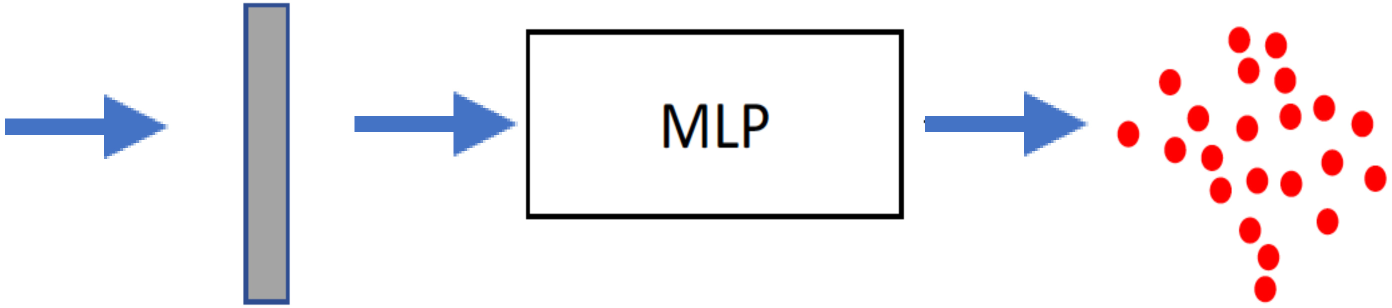
Encoder
E

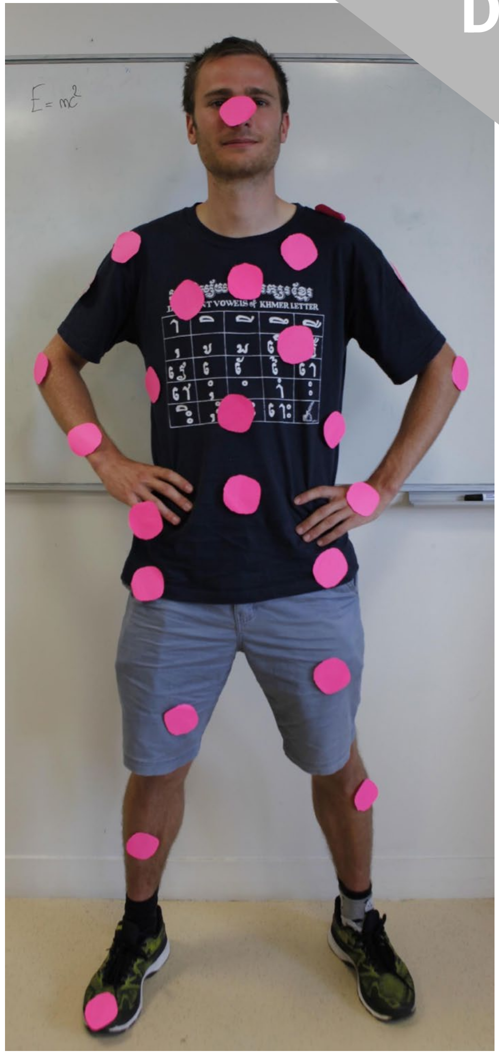
Decoder
D

Test Shape

Latent shape representation

Generated 3D points



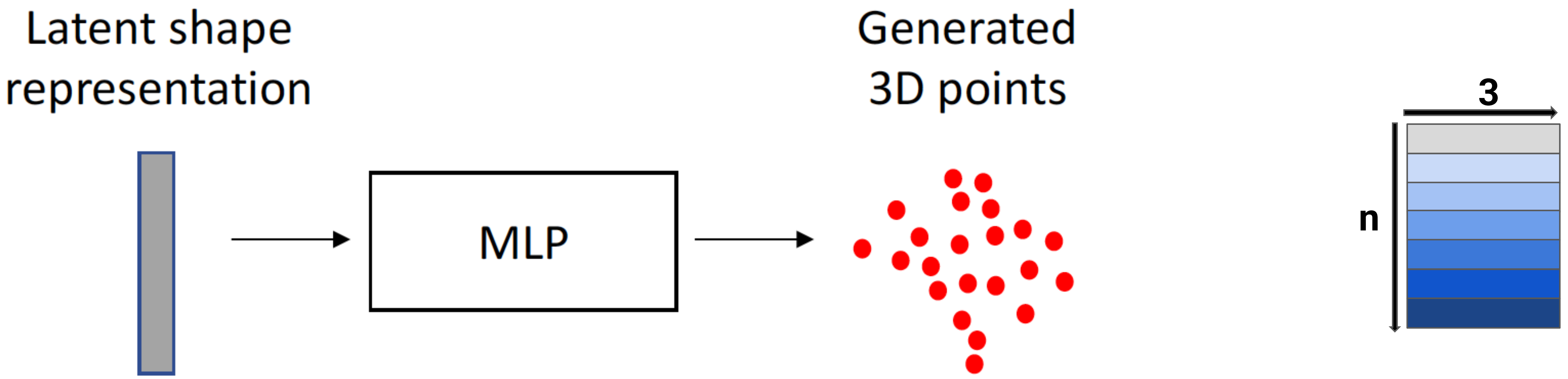


Issue: no idea of surfaces

Thibault Groueix, Pierre-Alain Langlois, 2019

Limitation of PointSetGen [Fan2017]

- ➔ **Generate a fixed number of points**
- ➔ Points connectivity is missing
- ➔ Generated points are not correlated enough to belong to an implicit surface



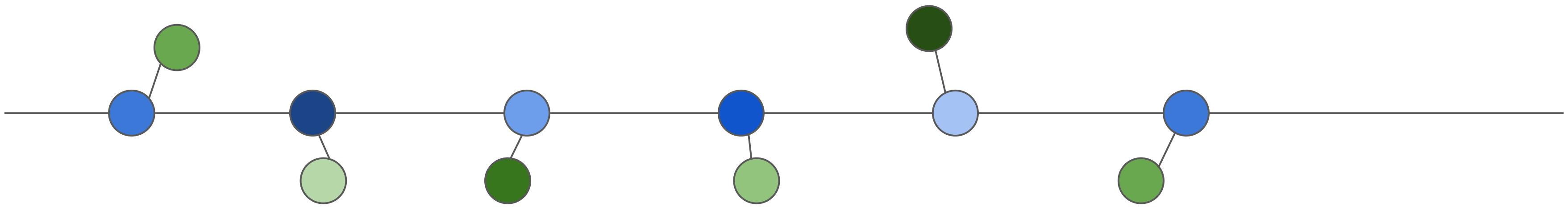
Thibault Groueix, Pierre-Alain Langlois, 2019

Limitation of PointSetGen [Fan2017]

- ➔ Generate a fixed number of points
- ➔ **Points connectivity is missing**
- ➔ Generated points are not correlated enough to belong to an implicit surface

Limitation of PointSetGen [Fan2017]

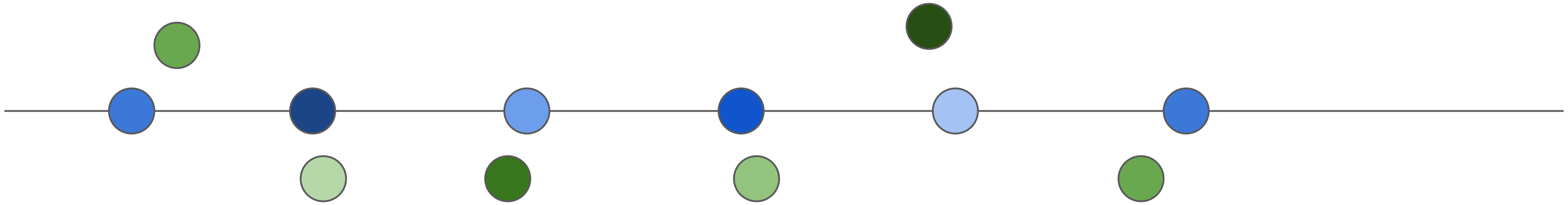
- ➔ Generate a fixed number of points
- ➔ **Points connectivity is missing**
- ➔ Generated points are not correlated enough to belong to an implicit surface



Thibault Groueix, Pierre-Alain Langlois, 2019

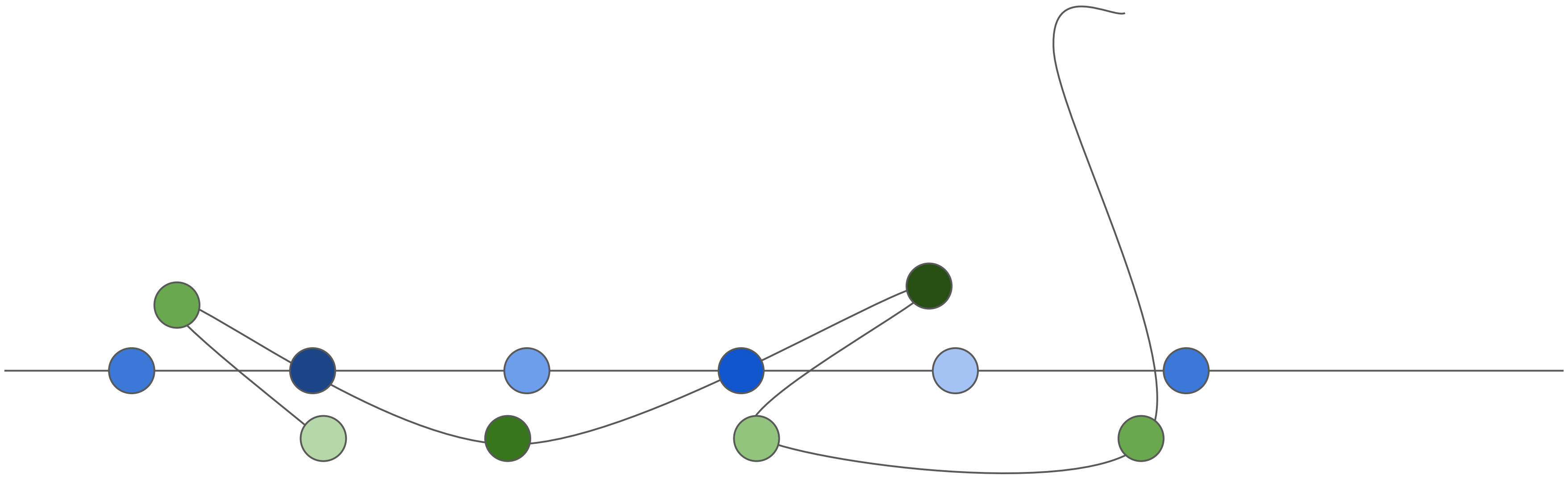
Limitation of PointSetGen [Fan2017]

- ➔ Generate a fixed number of points
- ➔ **Points connectivity is missing**
- ➔ Generated points are not correlated enough to belong to an implicit surface



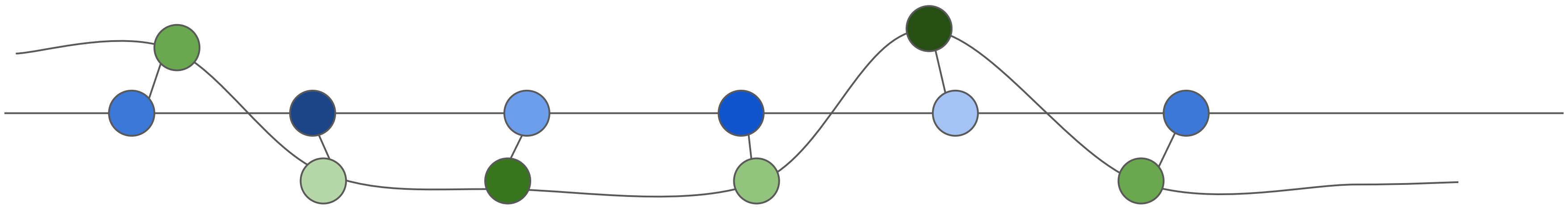
Limitation of PointSetGen [Fan2017]

- ➔ Generate a fixed number of points
- ➔ **Points connectivity is missing**
- ➔ Generated points are not correlated enough to belong to an implicit surface



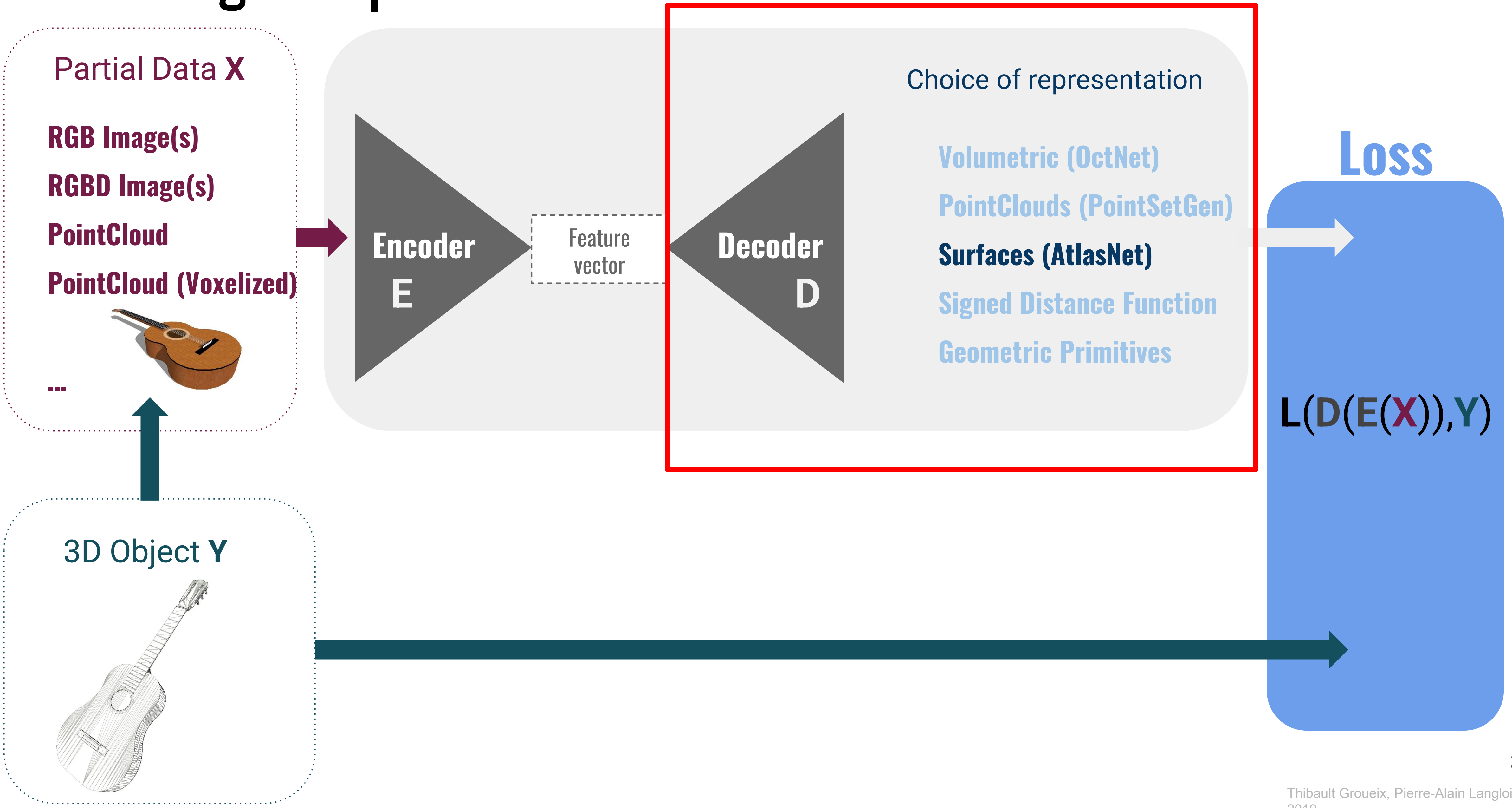
Limitation of PointSetGen [Fan2017]

- ➔ Generate a fixed number of points
- ➔ Points connectivity is missing
- ➔ **Generated points are not correlated enough to belong to an implicit surface**

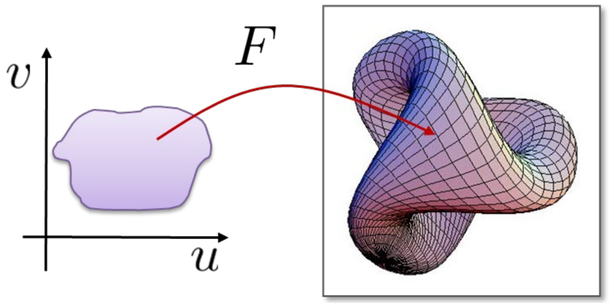


Reconstructing the mesh from a pointcloud :
Poisson Surface Reconstruction [**Kazhdan2013**]

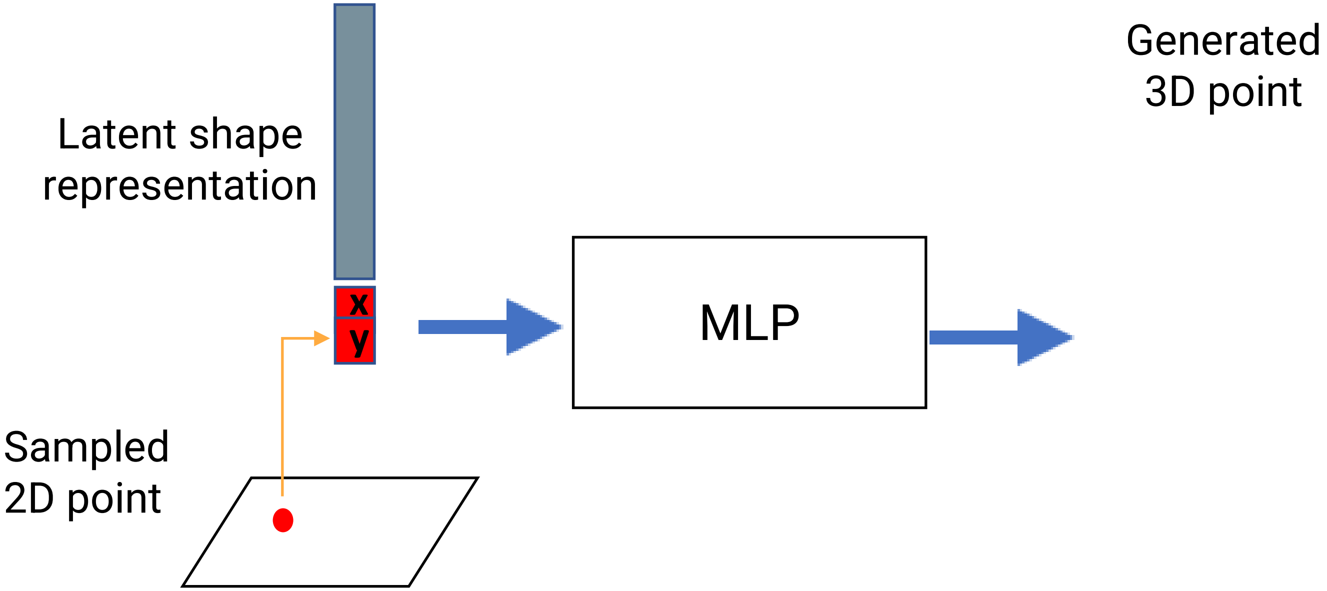
Training setup for 3D reconstruction



Deform a surface [Groueix2018]

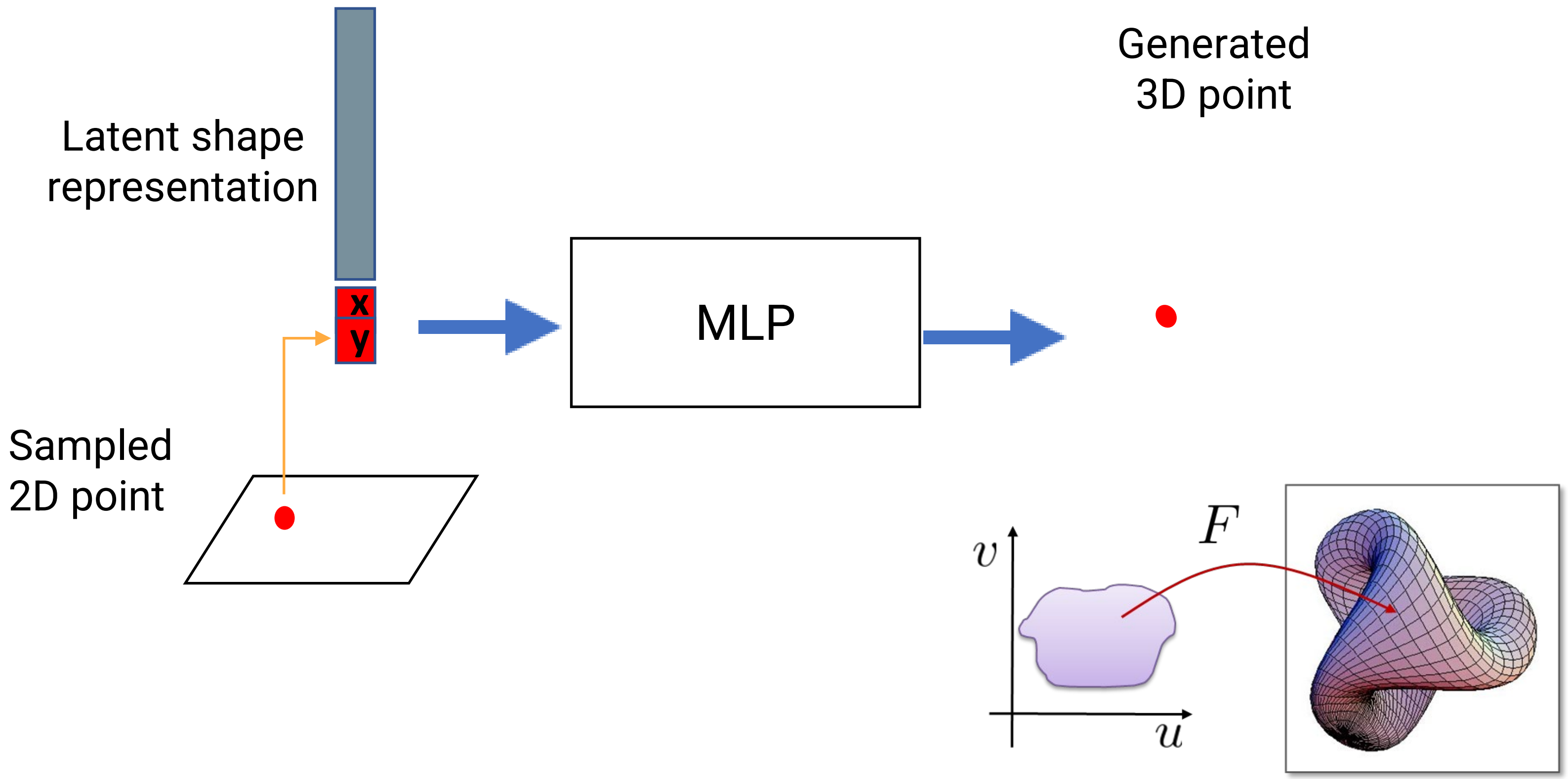


Decoder
D



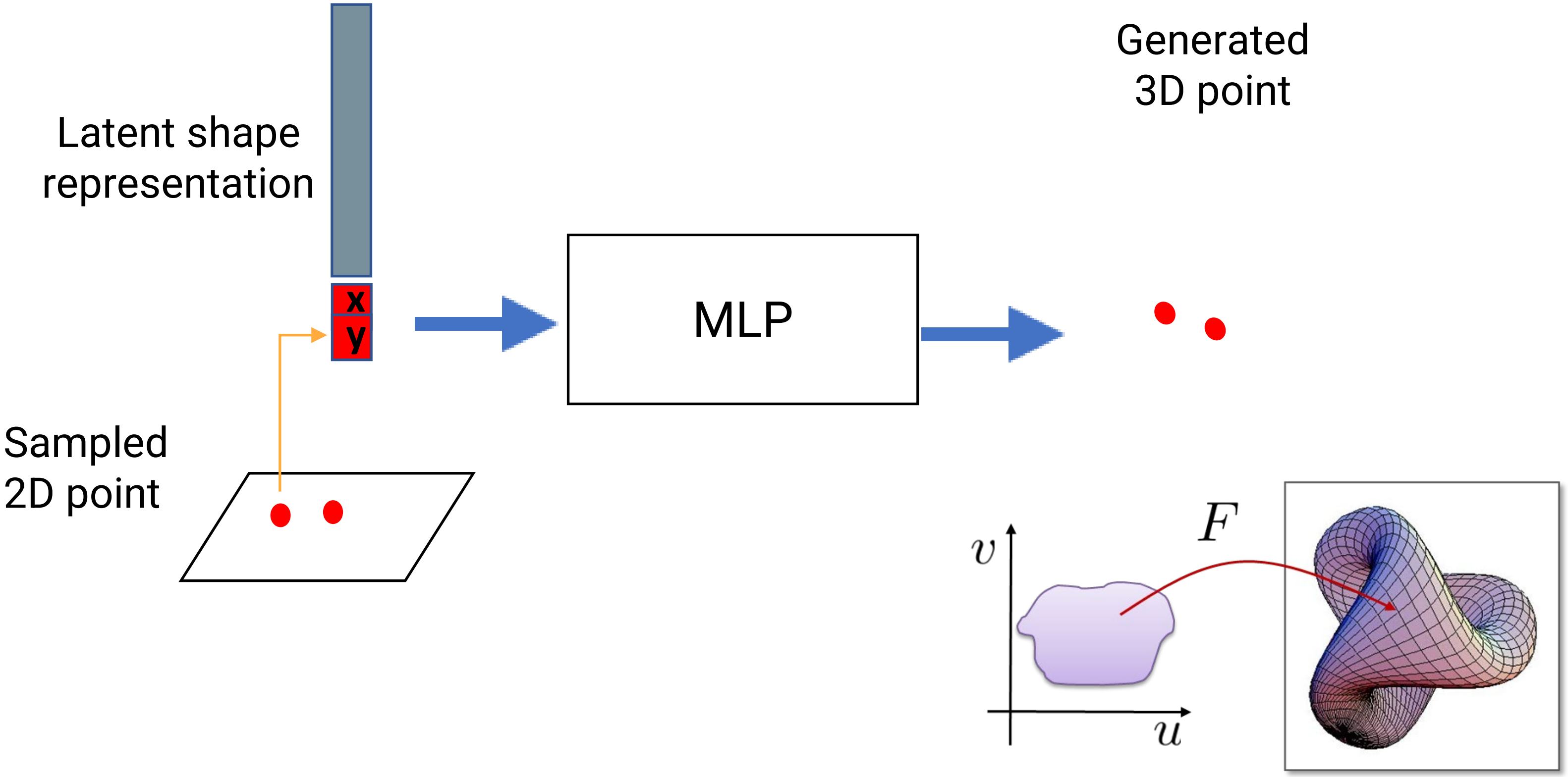
Deform a surface : space mapping trick [Groueix2018]

Decoder
D



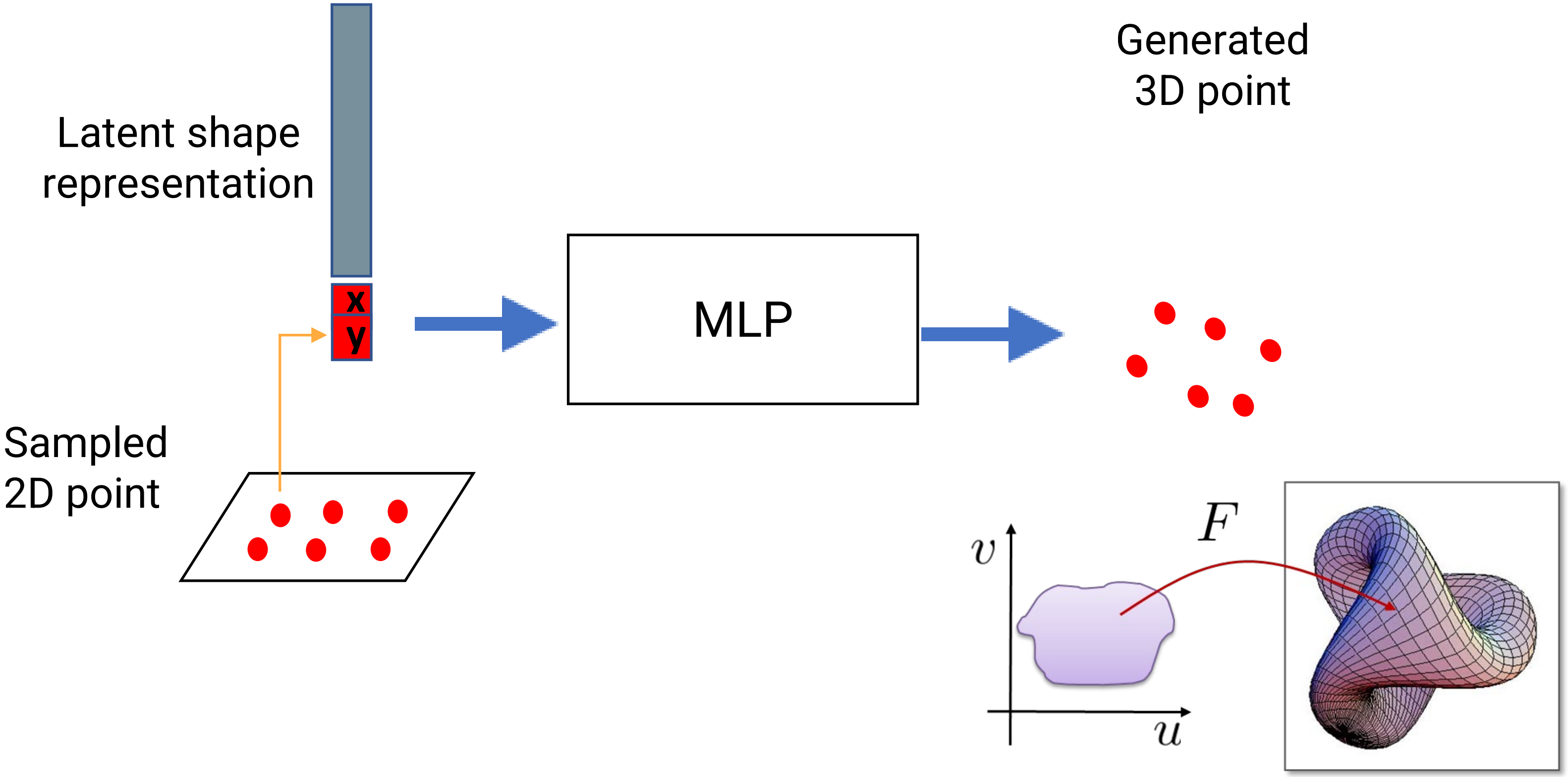
Thibault Groueix, Pierre-Alain Langlois, 2019

Deform a surface [Groueix2018]

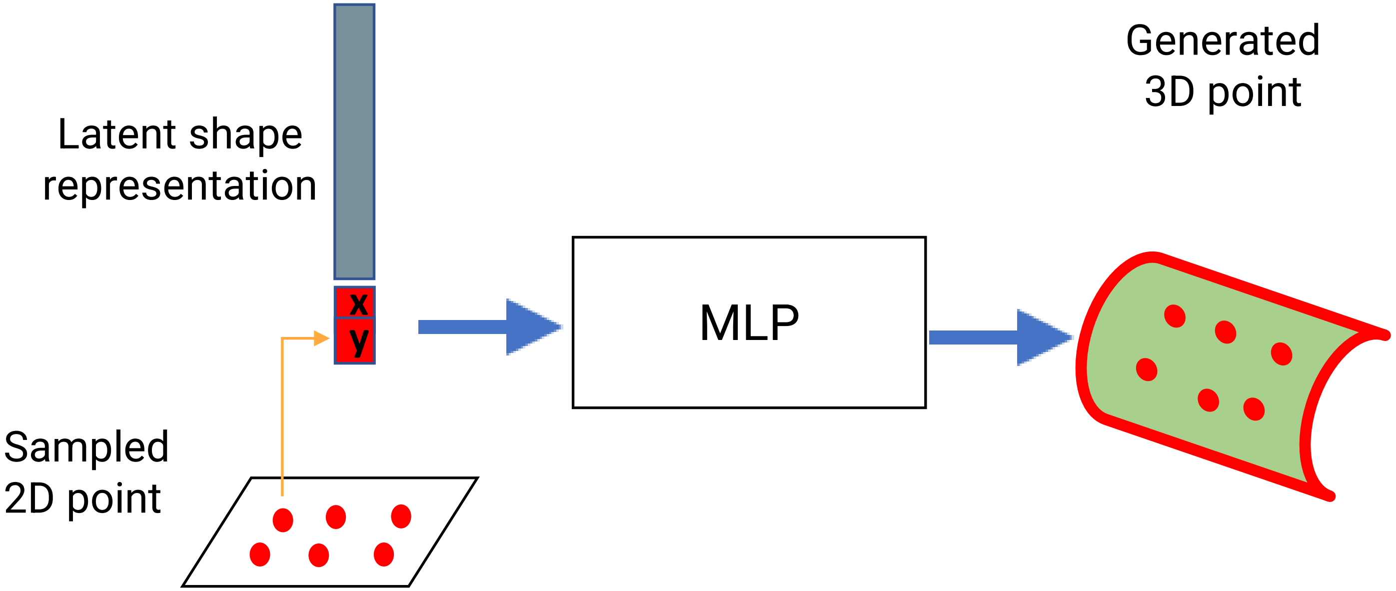


Thibault Groueix, Pierre-Alain Langlois, 2019

Deform a surface [Groueix2018]

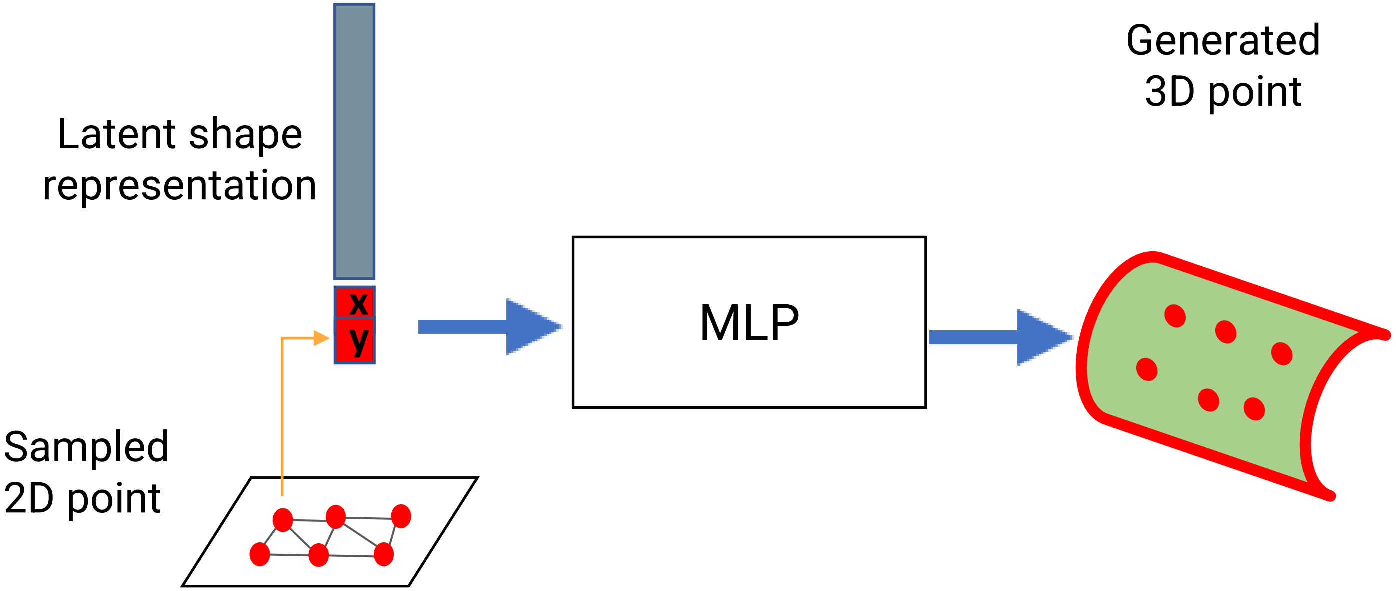


Deform a surface [Groueix2018]



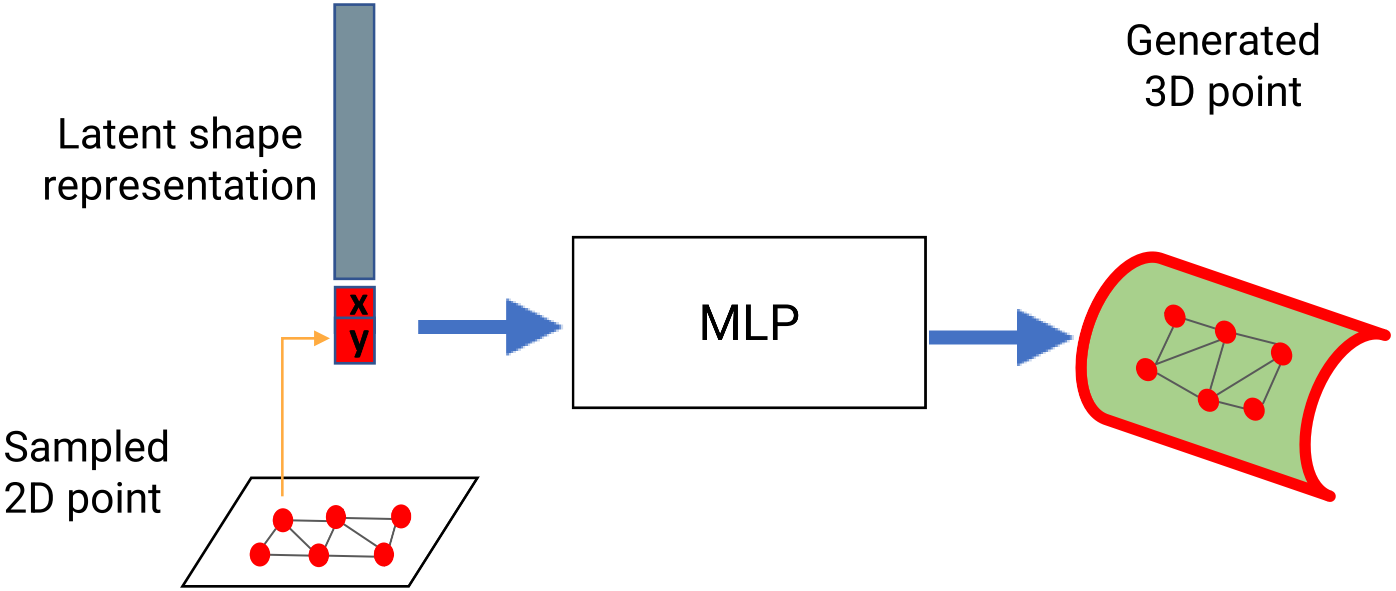
Deform a surface [Groueix2018]

Decoder D



Deform a surface [Groueix2018]

Decoder D

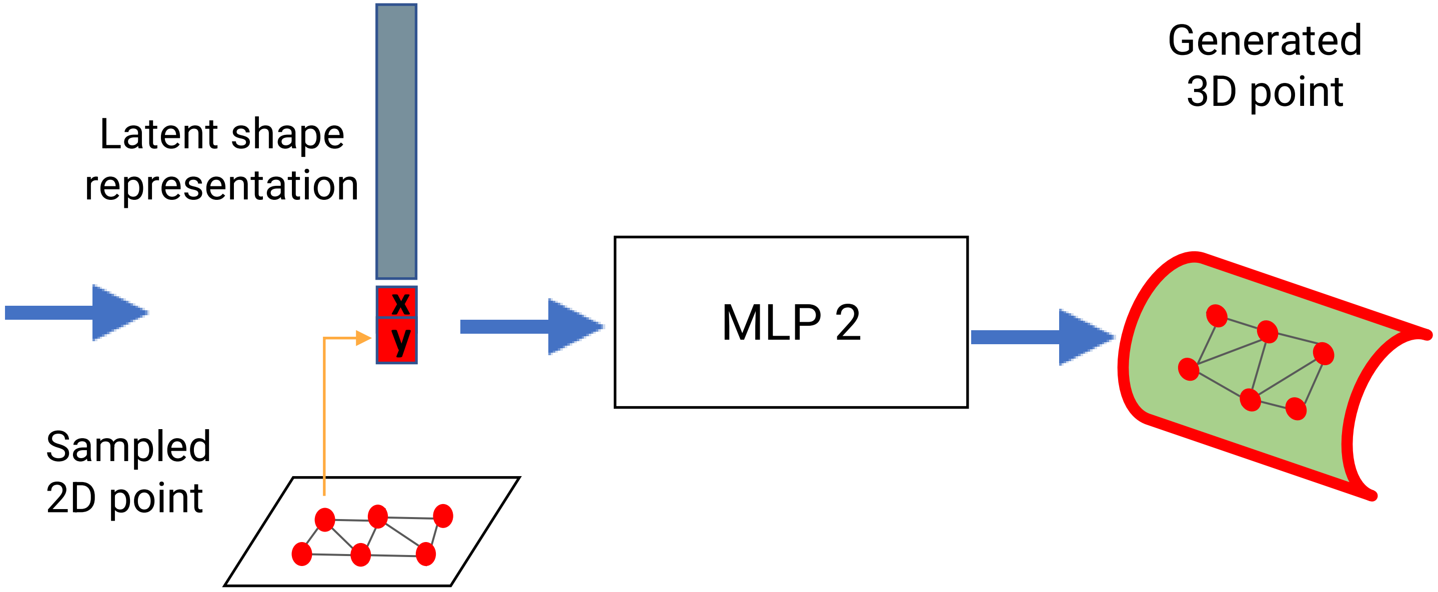


Deform a surface [Groueix2018]

Encoder
E

Decoder
D

Test Shape

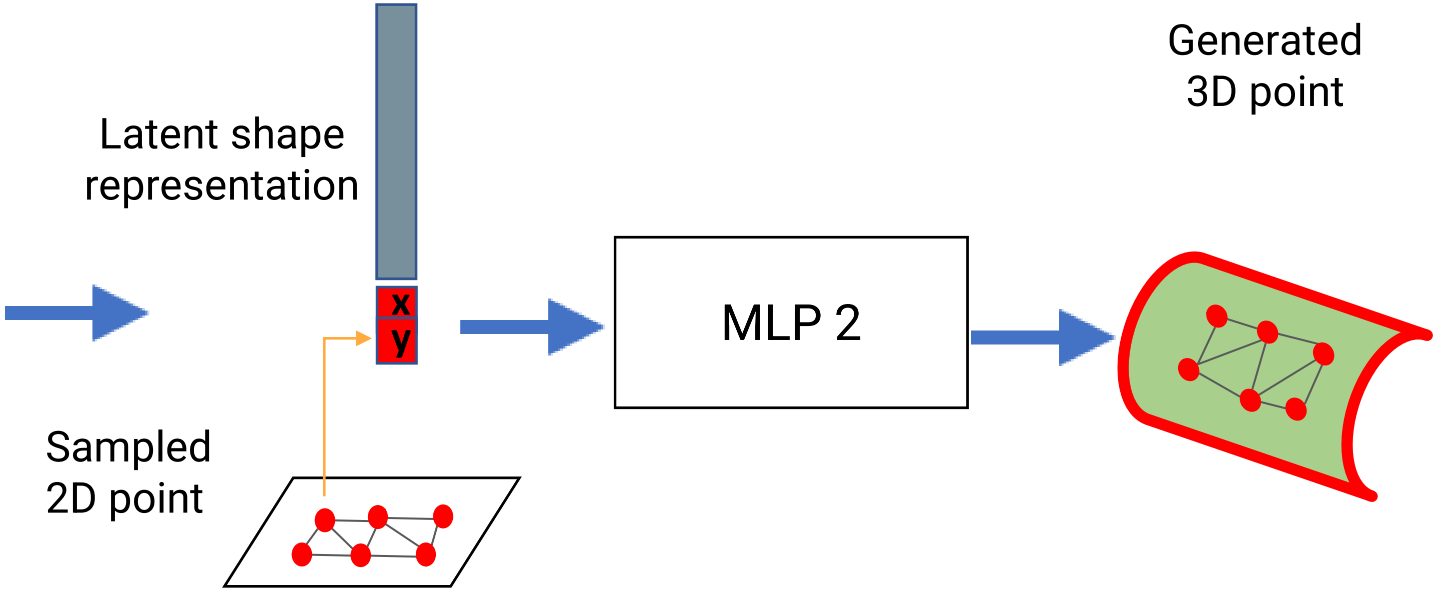


Thibault Groueix, Pierre-Alain Langlois, 2019

Deform a surface [Groueix2018]

Encoder
E

Decoder
D



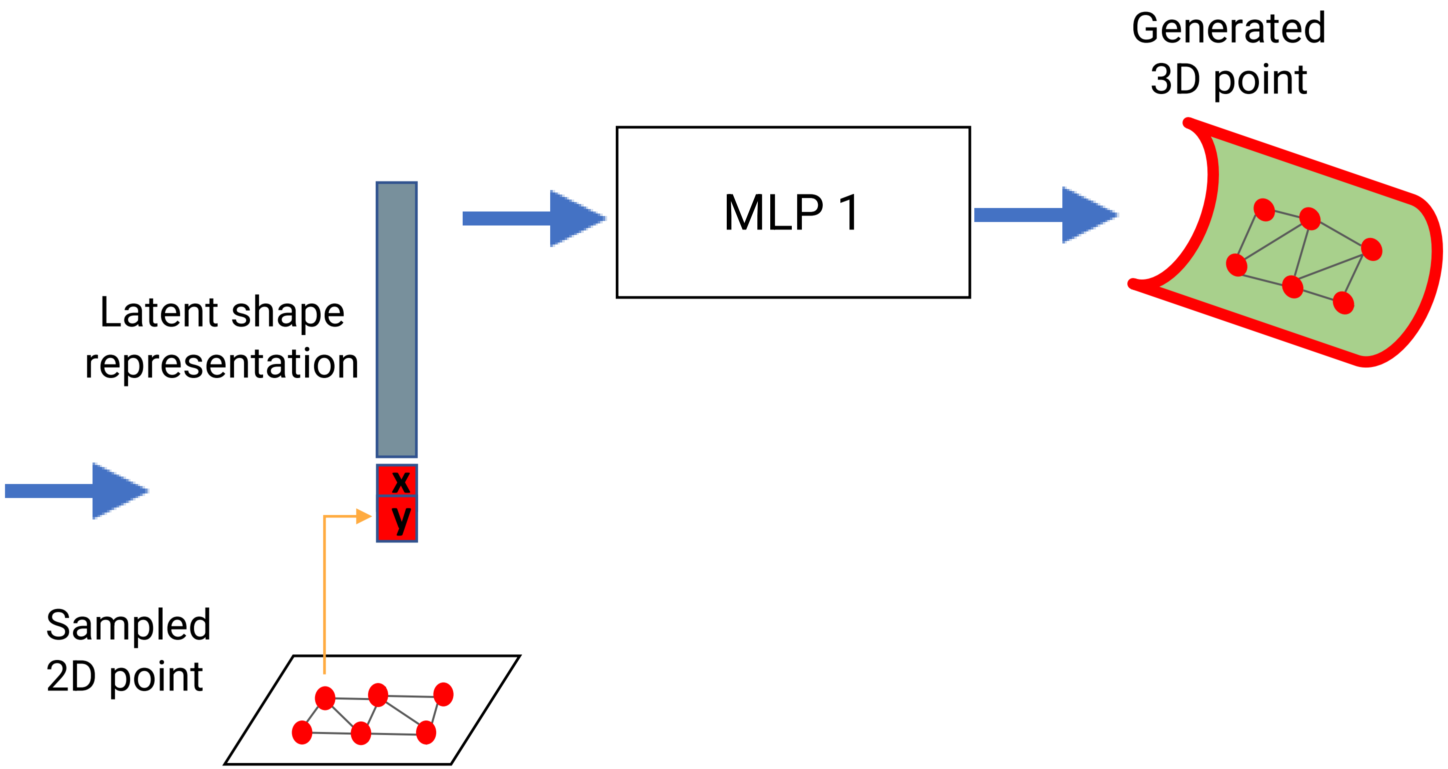
Test Shape

Thibault Groueix, Pierre-Alain Langlois, 2019

Deform a surface [Groueix2018]

Encoder
E

Decoder
D



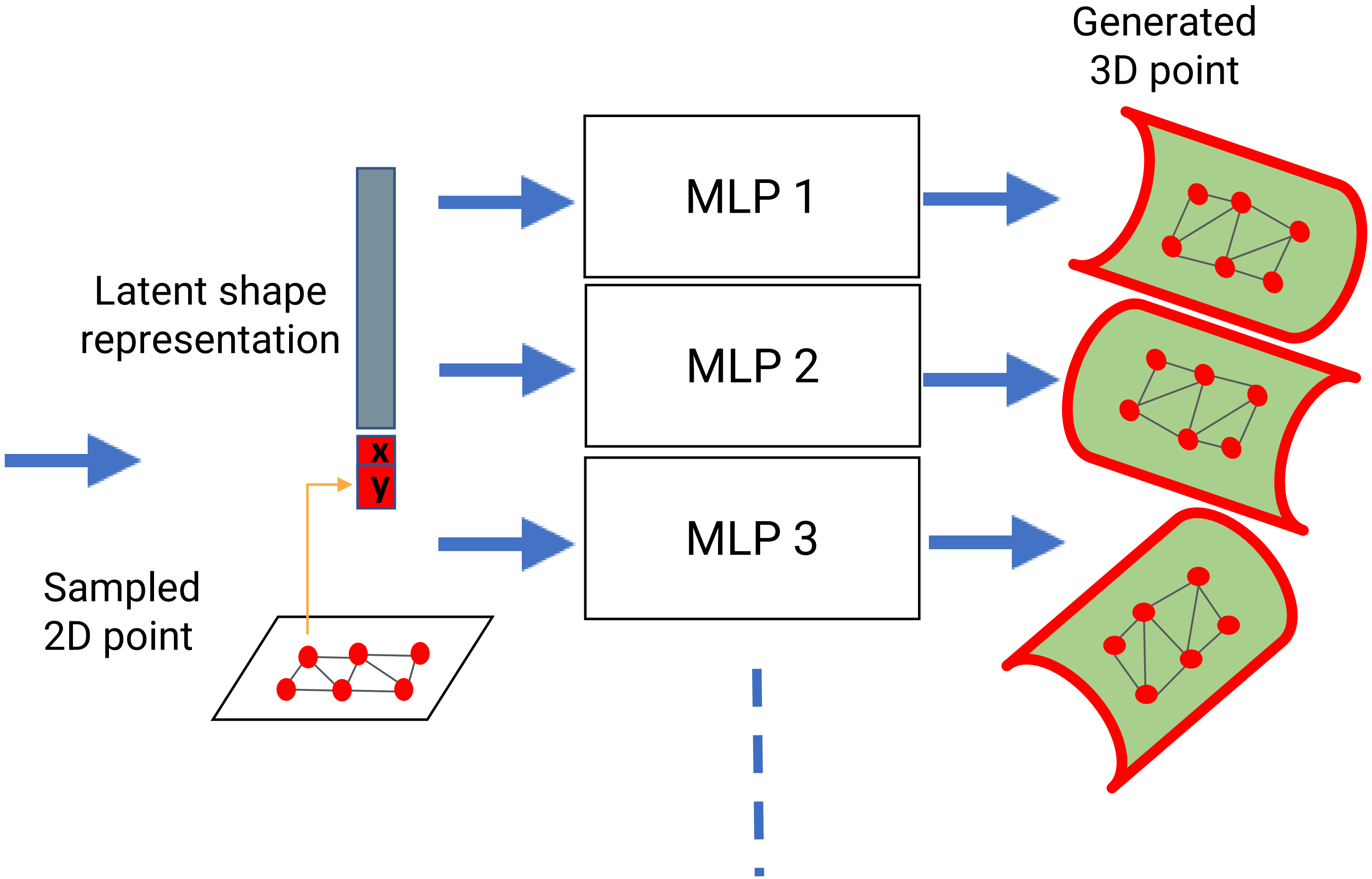
Test Shape

Encoder
E

Deform a surface [Groueix2018]

Decoder
D

Test Shape



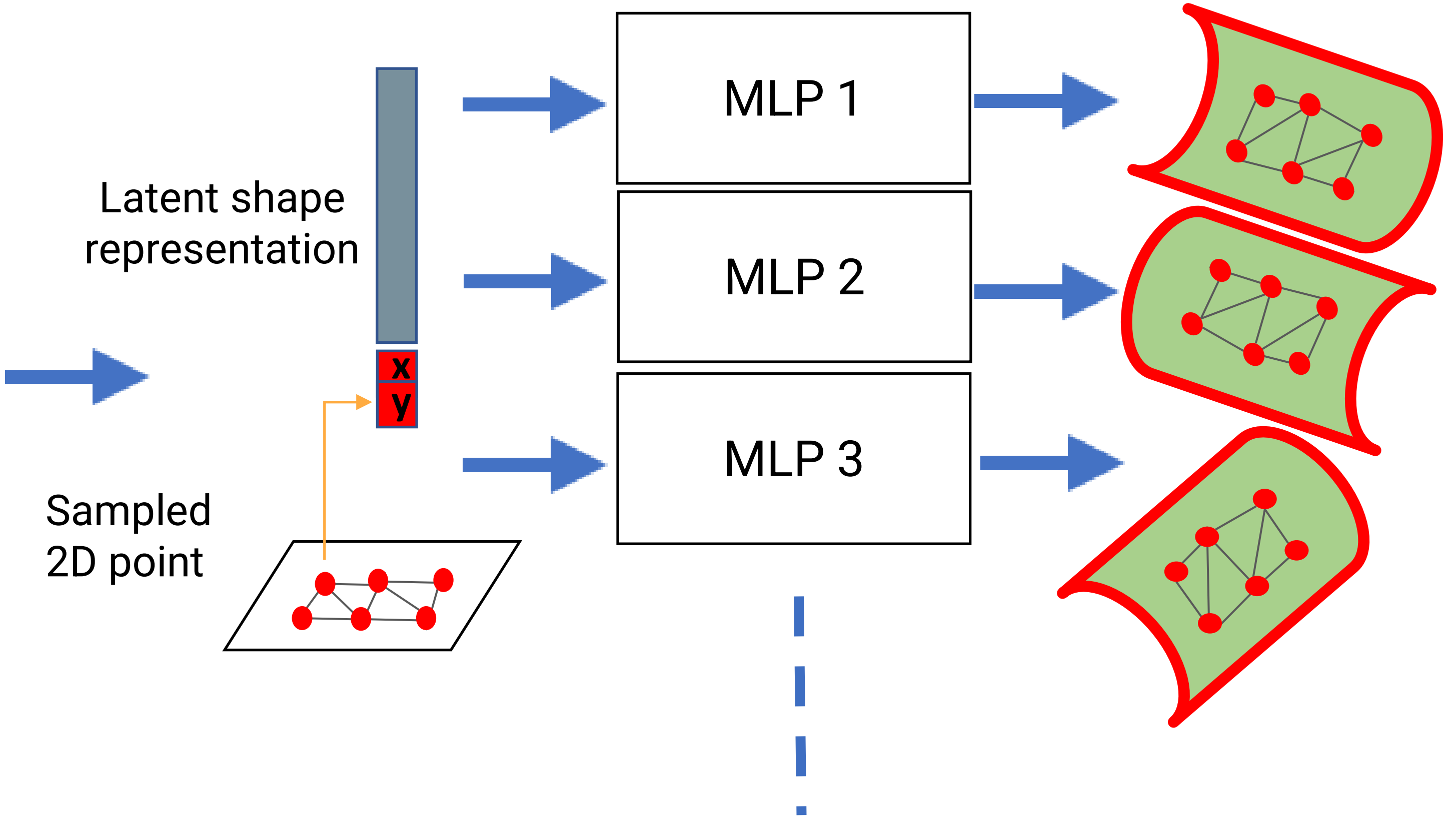
Piecewise Parametric Surfaces

Thibault Groueix, Pierre-Alain Langlois, 2019

Deform a surface [Groueix2018]

Encoder
E

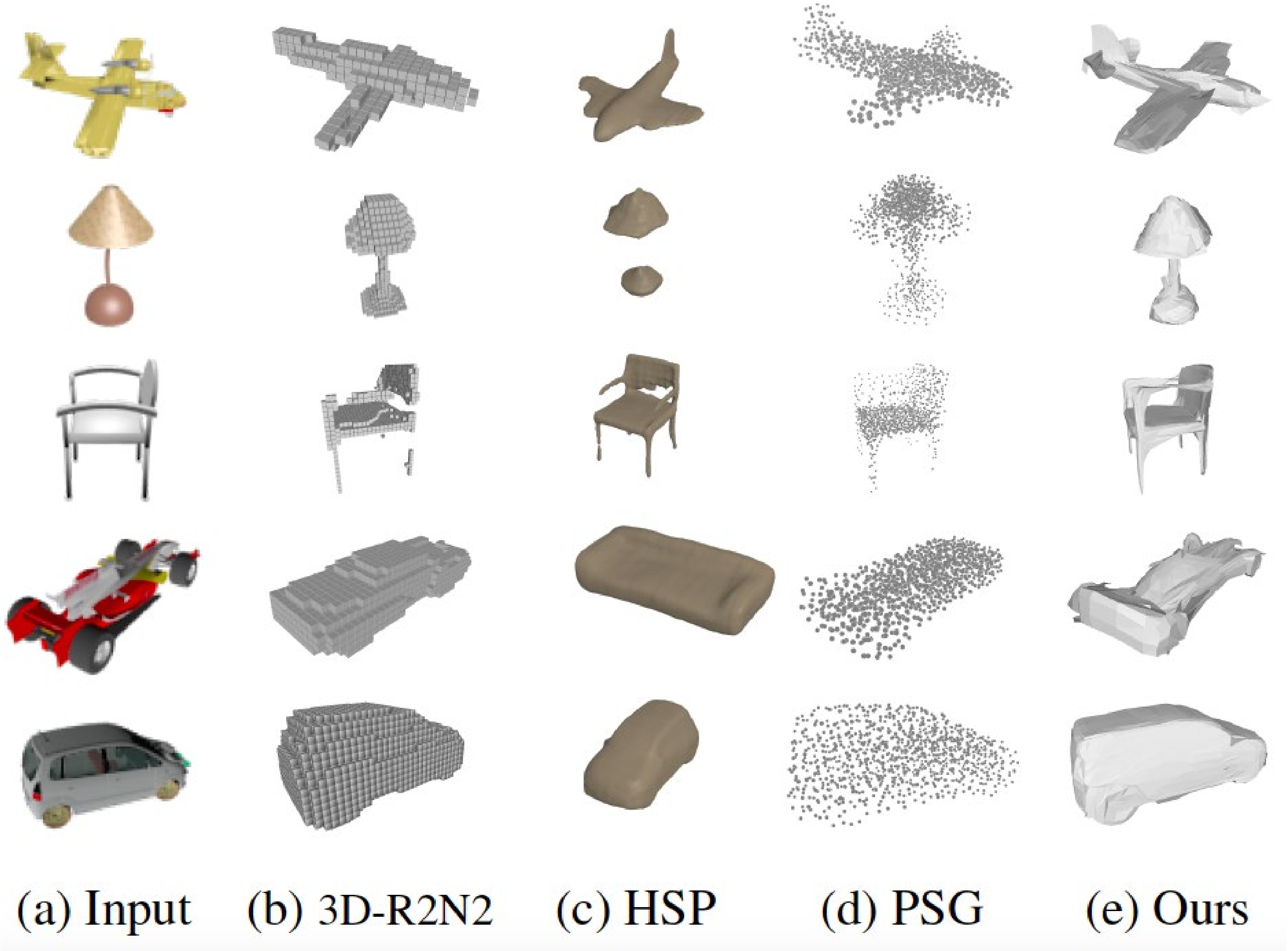
Test Shape



Decoder
D

Thibault Groueix, Pierre-Alain Langlois, 2019

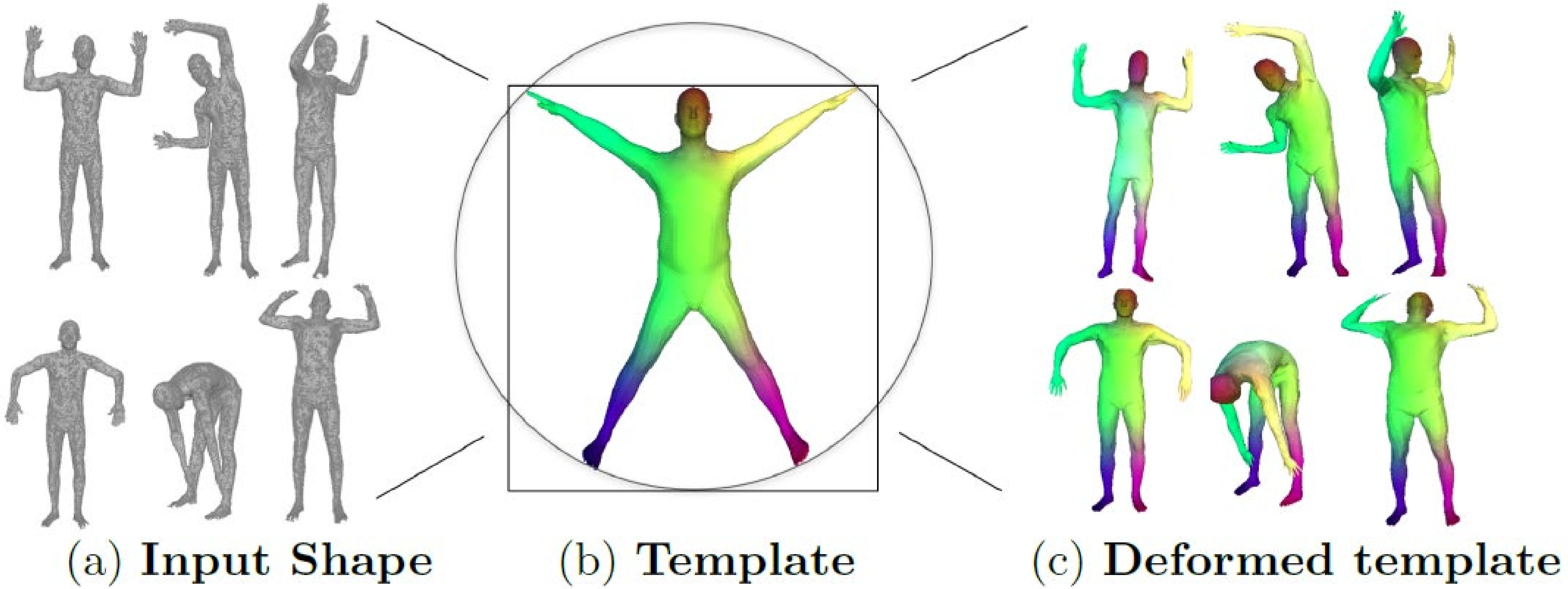
Results : Single View Reconstruction



Thibault Groueix, Pierre-Alain Langlois, 2019

Direct application : mesh parametrization

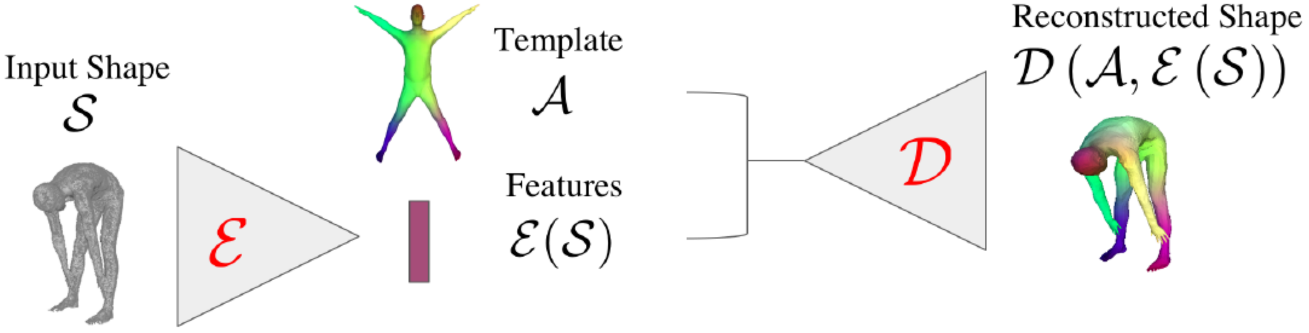
Semantic Parametr Spaces / Templates: 3D-CODED



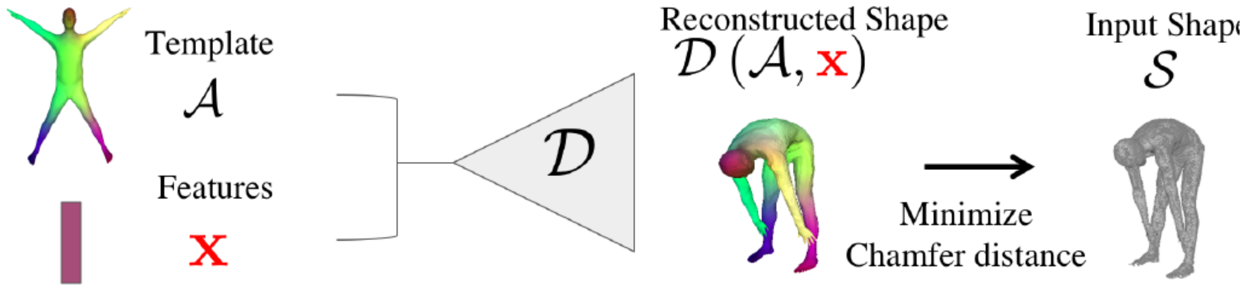
3D-CODED : 3D Correspondences by Deep Deformation

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, Mathieu Aubry https://arxiv.org/abs/1806.05228

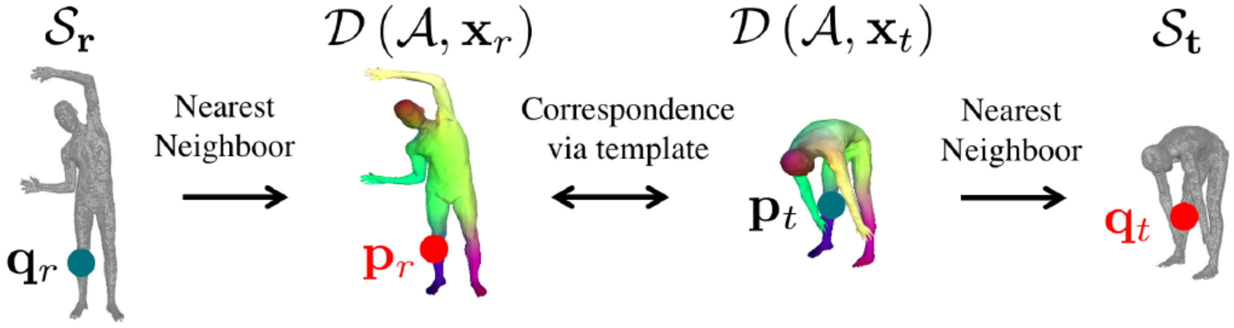
Semantic Parametr Spaces / Templates: 3D-CODED



(a) **Network training**



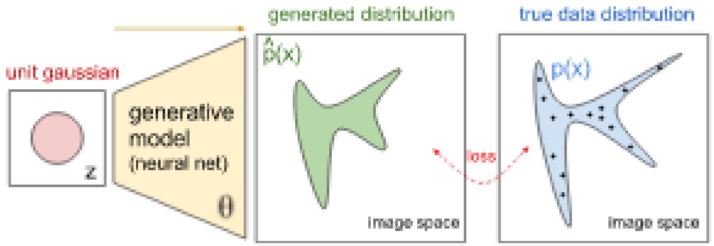
(b) **Local optimization of feature** $\mathbf{x}$



(c) **Correspondences**

State-of-the-art correspondences of FAUST [Groueix2018b]

Generative Adversarial Networks

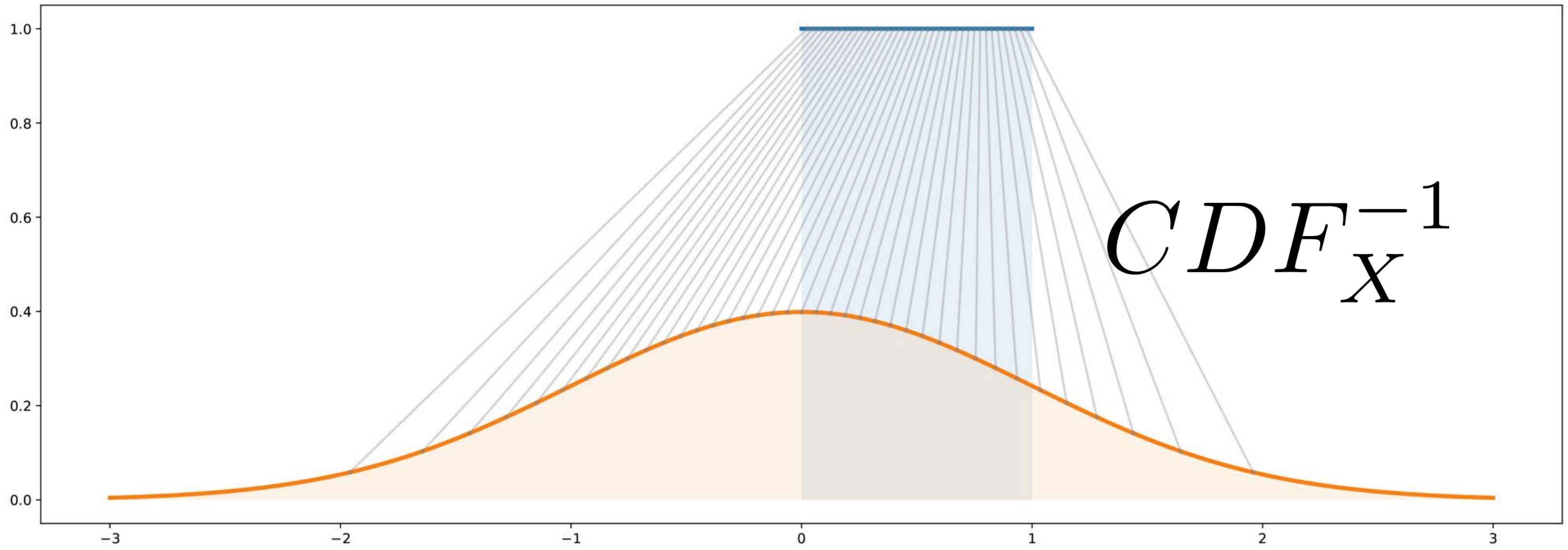


Transform Functions for Random Variables

$$CDF_U(u) = \mathbb{P}(U \leq u) = u \quad \forall u \in [0,1]$$

Cumulative Distribution Function (CDF)

$$CDF_X(x) = \mathbb{P}(X \leq x) \quad \in [0,1]$$



U

CDF_X^{-1}

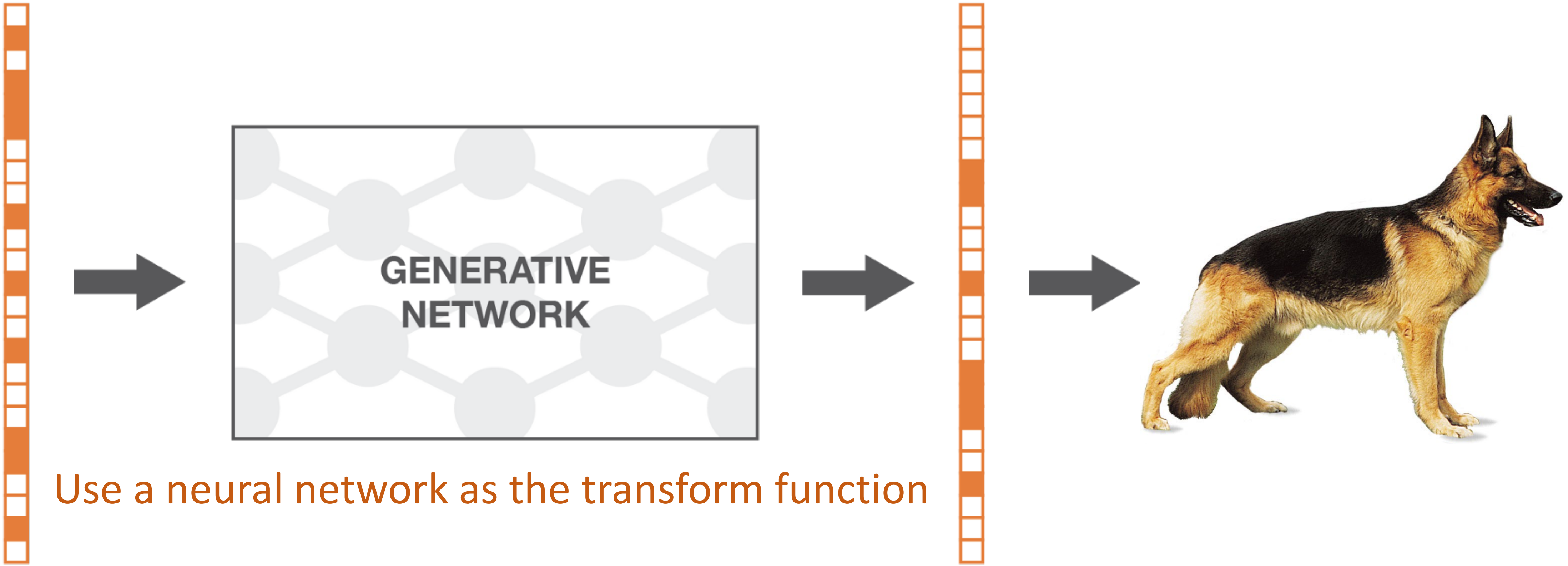
$$Y = CDF_X^{-1}(U)$$

X

$$CDF_Y(y) = \mathbb{P}(Y \leq y) = \mathbb{P}\left(CDF_X^{-1}(U) \leq y\right) = \mathbb{P}\left(U \leq CDF_X(y)\right) = CDF_X(y)$$

Conceptually, the purpose of the “transform function” is to deform/reshape the initial probability distribution: the transform function takes from where the initial distribution is too high compared to the targeted distribution and puts it where it is too low.

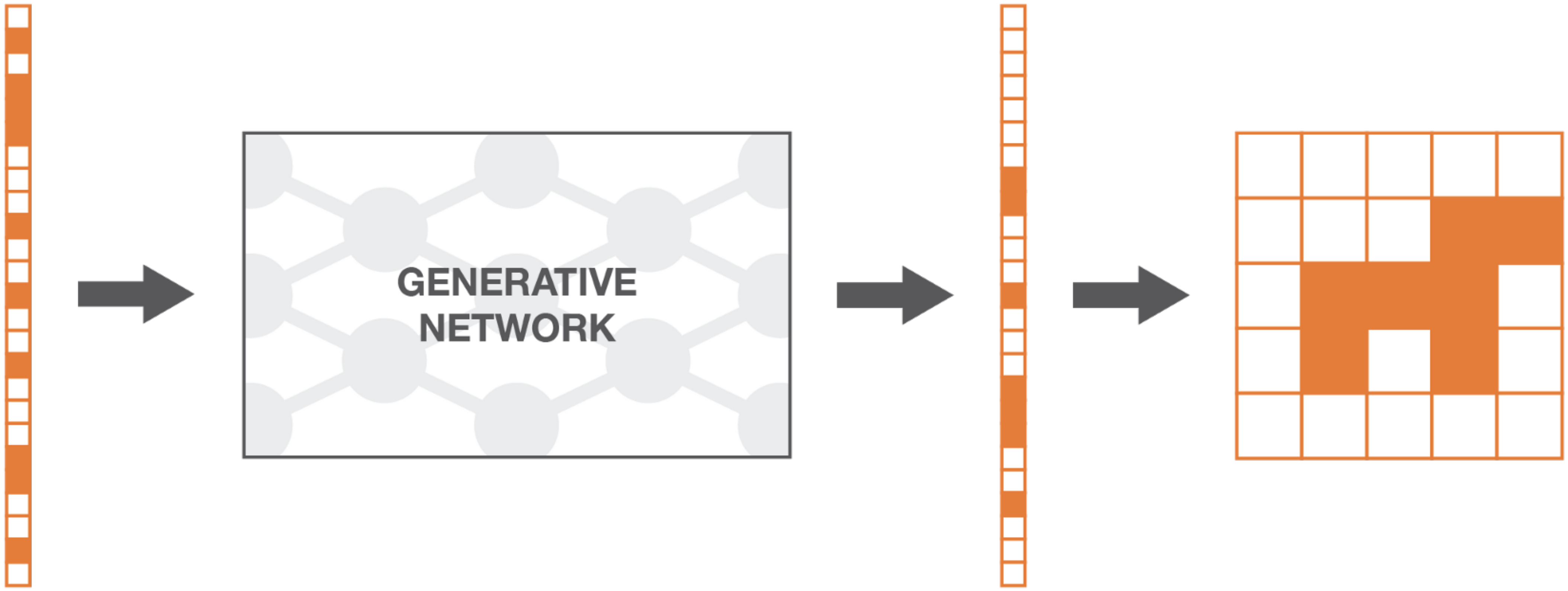
Complex Output Distributions



Use a neural network as the transform function

The problem of generating a new image of dog is equivalent to the problem of generating a new vector following the “dog probability distribution” over the N dimensional vector space.
So we are, in fact, facing a problem of generating a random variable with respect to a specific probability distribution.

Generative Model Structure



Input random variable (drawn from a simple distribution, for example uniform).

The generative network transforms the simple random variable into a more complex one.

Output random variable (should follow the targeted distribution, after training the generative network).

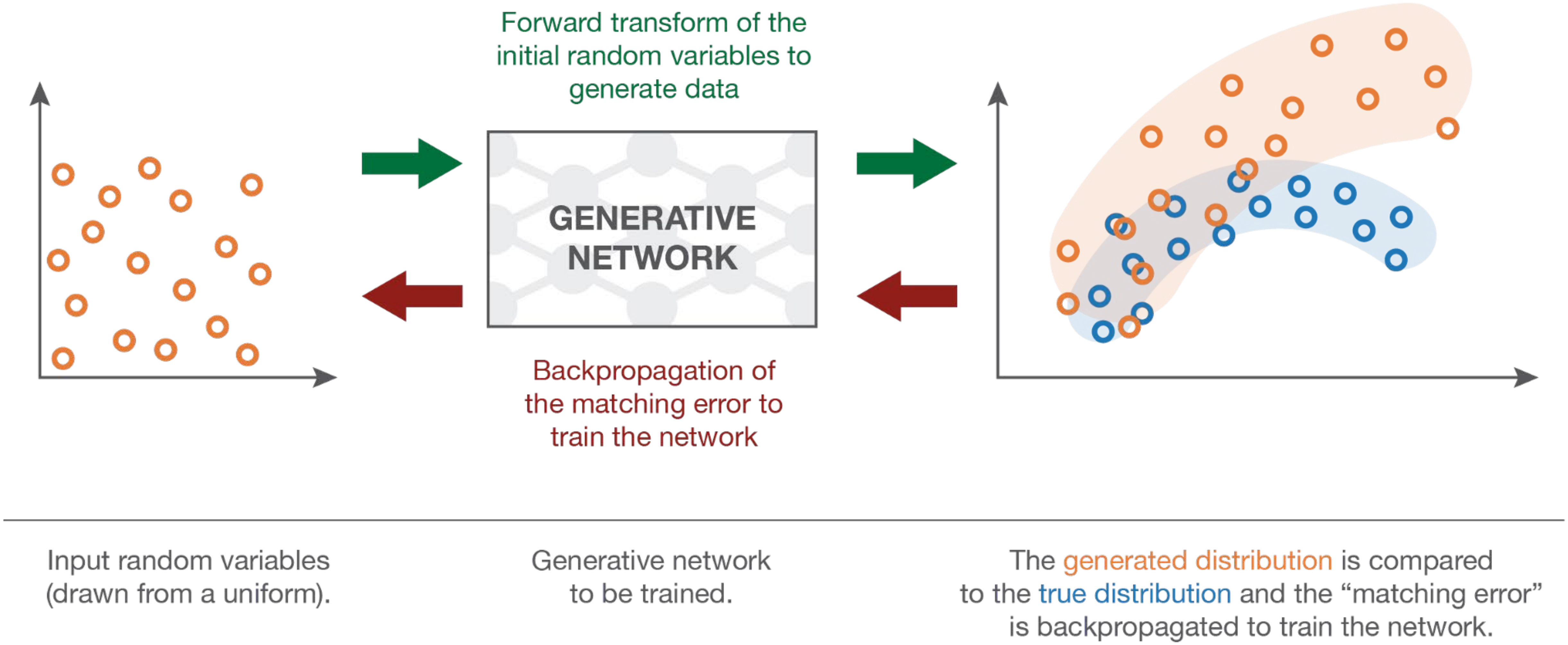
The output of the generative network once reshaped.

Loss: Comparing Distributions Based on Samples

- generate some uniform inputs
- make these inputs go through the network and collect the generated outputs
- compare the true “dog probability distribution” and the generated one based on the available samples
- use backpropagation to make one step of gradient descent to lower the distance between true and generated distributions

- This is a very hard problem
  - Maximum Mean Discrepancy (MMD)
  - compute the MMD distance between the sample of true dog images and the sample of generated ones

Gradient Descent Based on this Loss for Training



Extremely expensive!

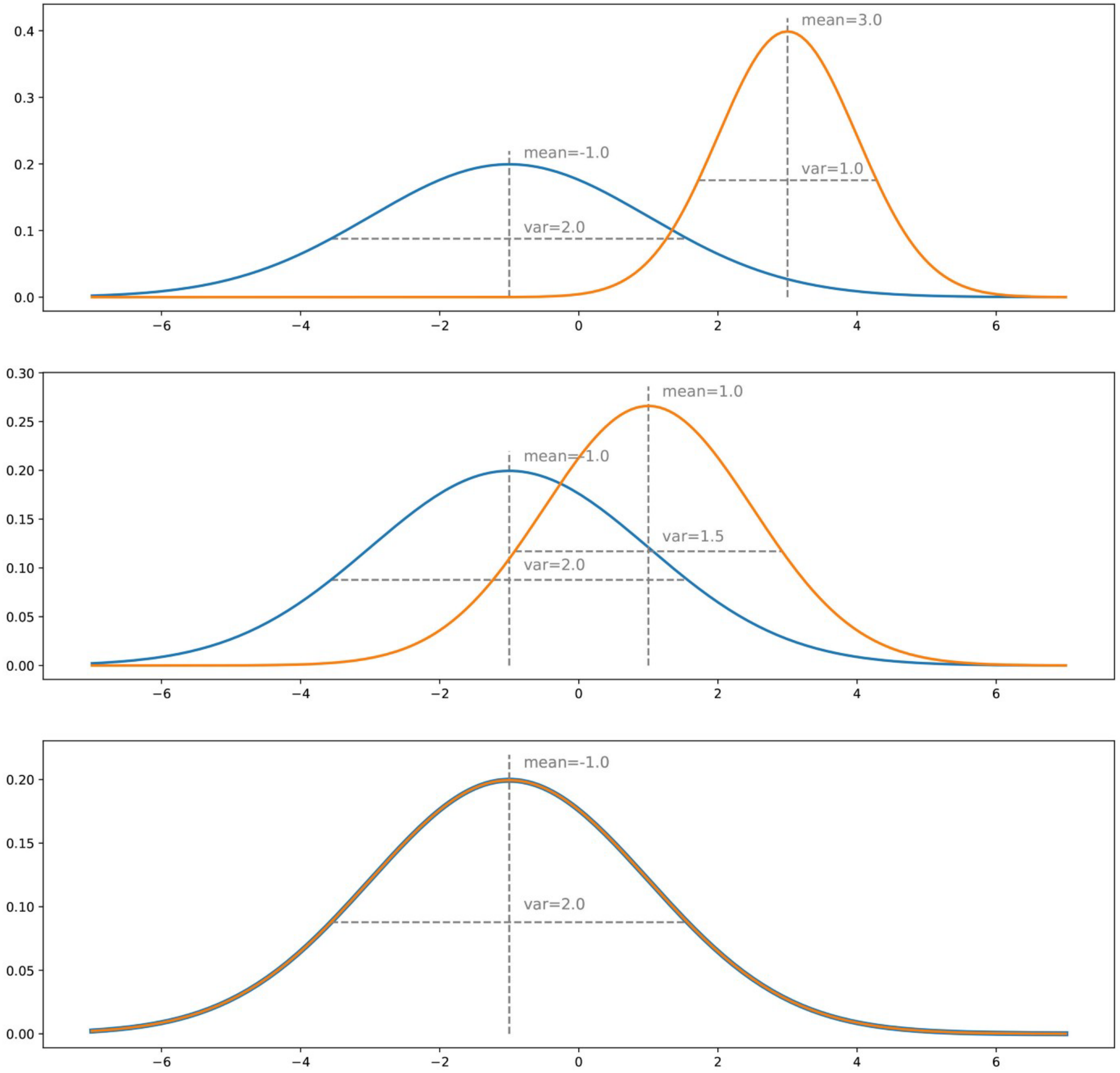
An Alternative: Compare on a Downstream Task

- An indirect loss
- **Generative Adversarial Networks (GANs)**: compare distributions through a downstream task
- Use the loss of that task to improve the generator
- Make that task itself be a trainable neural network

A Distribution Discriminator as that Task

Baseline: the direct approach

The distribution in blue is the true one while the generated distribution is depicted in orange. Iteration by iteration, we compare the two distributions and adjust the networks weights through gradient descent steps. Here the comparison is done over the mean and the variance (similar to a truncated moments matching method). Notice that (obviously) this example is so simple that it doesn’t require an iterative approach: the purpose is only to illustrate the intuition given above.

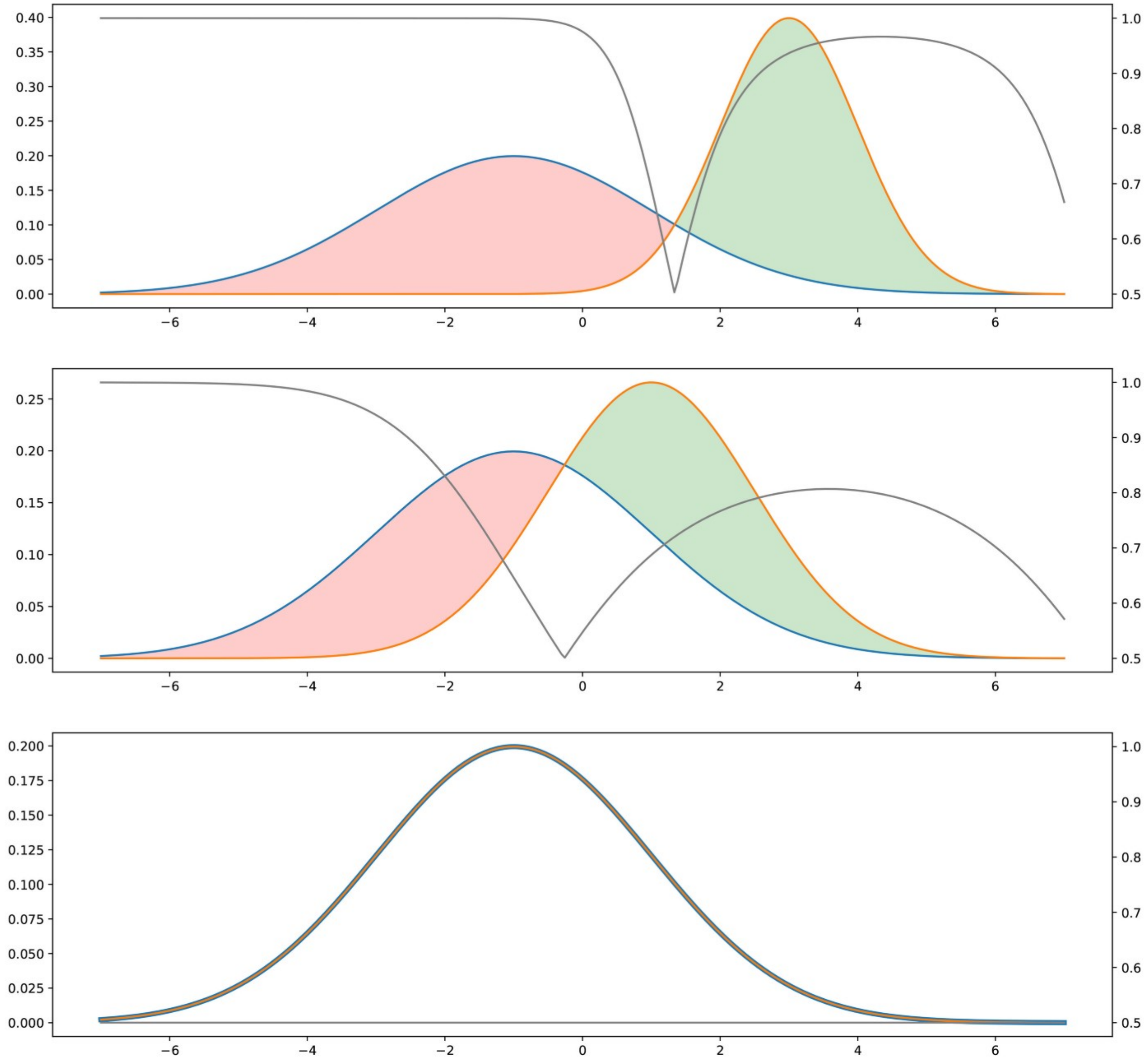


A Distribution Discriminator as that Task

Indirect: use a discriminator

differentiate samples between the two distributions

The blue distribution is the true one, the orange is the generated one. In grey, with corresponding y-axis on the right, we displayed the probability to be true for the discriminator if it chooses the class with the higher density in each point (assuming “true” and “generated” data are in equal proportions). The closer the two distributions are, the more often the discriminator is wrong. When training, the goal is to “move the green area” (generated distribution is too high) towards the red area (generated distribution is too low).



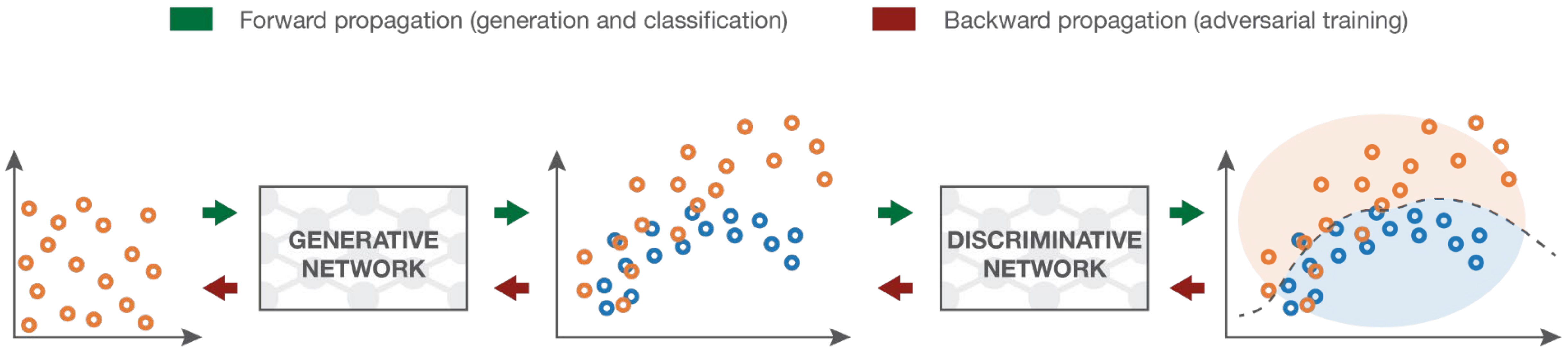
How to Get a Discriminator?

- Learn a discriminator through another neural network

- the goal of the generator is to fool the discriminator, so the generative neural network is trained to maximize the final classification error (between true and generated data)
- the goal of the discriminator is to detect fake generated data, so the discriminative neural network is trained to minimize the final classification error

At each iteration of the training process, the weights of the generative network are updated in order to increase the classification error (error gradient ascent over the generator’s parameters) whereas the weights of the discriminative network are updated so that to decrease this error (error gradient descent over the discriminator’s parameters).

An Adversarial Discriminator



Input random variables.	The generative network is trained to **maximise** the final classification error.	The generated distribution and the true distribution are not compared directly.	The discriminative network is trained to **minimise** the final classification error.	The classification error is the basis metric for the training of both networks.

A Mathematical Formulation

- a generative network $G(.)$ that takes a random input z with density p_z (the “noise vector”) and returns an output x_g = G(z) that should follow (after training) the targeted probability distribution
- a discriminative network $D(.)$ that takes an input x that can be a “true” one (x_t, whose density is denoted p_t) or a “generated” one (x_g, whose density p_g is the density induced by the density p_z going through G) and that returns the probability $D(x)$ of to be a “true” data

Error
$$
\begin{aligned}
E(G,D) &= \frac{1}{2}\mathbb{E}_{x\sim p_t}[1-D(x)] + \frac{1}{2}\mathbb{E}_{z\sim p_z}[D(G(z))] \\
&= \frac{1}{2}\left(\mathbb{E}_{x\sim p_t}[1-D(x)] + \mathbb{E}_{x\sim p_g}[D(x)]\right)
\end{aligned}
$$

$$
\max_G \left(\min_D E(G,D) \right)
$$

A minimax Nash equilibrium

Direct vs. Indirect Losses

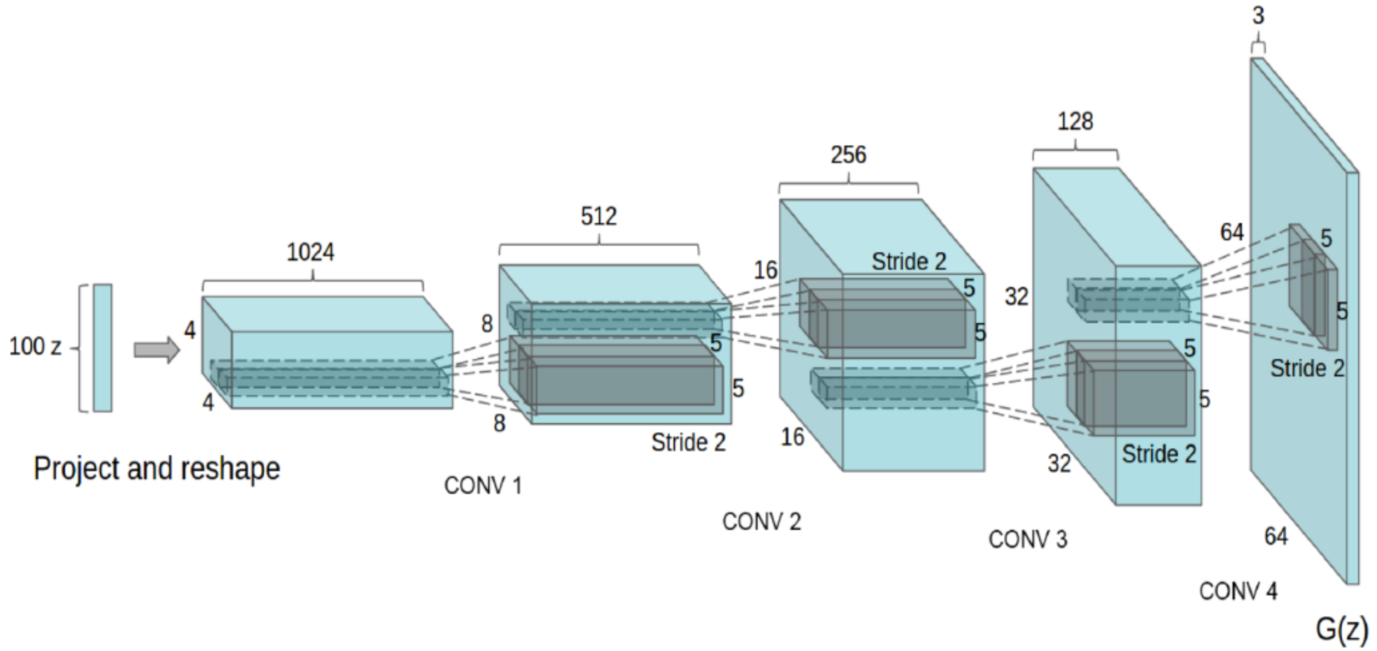
In an idealized setting of unlimited capacity generator and discriminator and smoothness of the underlying distributions:

it can be shown that the learned generator produces the same density as the true density and the learned discriminator can’t do better than being true in one case out of two.

An Example: DCGAN

Deep Convolutional GANs (DCGANs)

Generator Architecture



Key ideas:

- Replace FC hidden layers with Convolutions
  - **Generator:** Fractional-Strided convolutions
- Use Batch Normalization after each layer
- **Inside Generator**
  - Use ReLU for hidden layers
  - Use Tanh for the output layer

Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).

An Example: DCGAN

DCGAN: Bedroom images

Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).

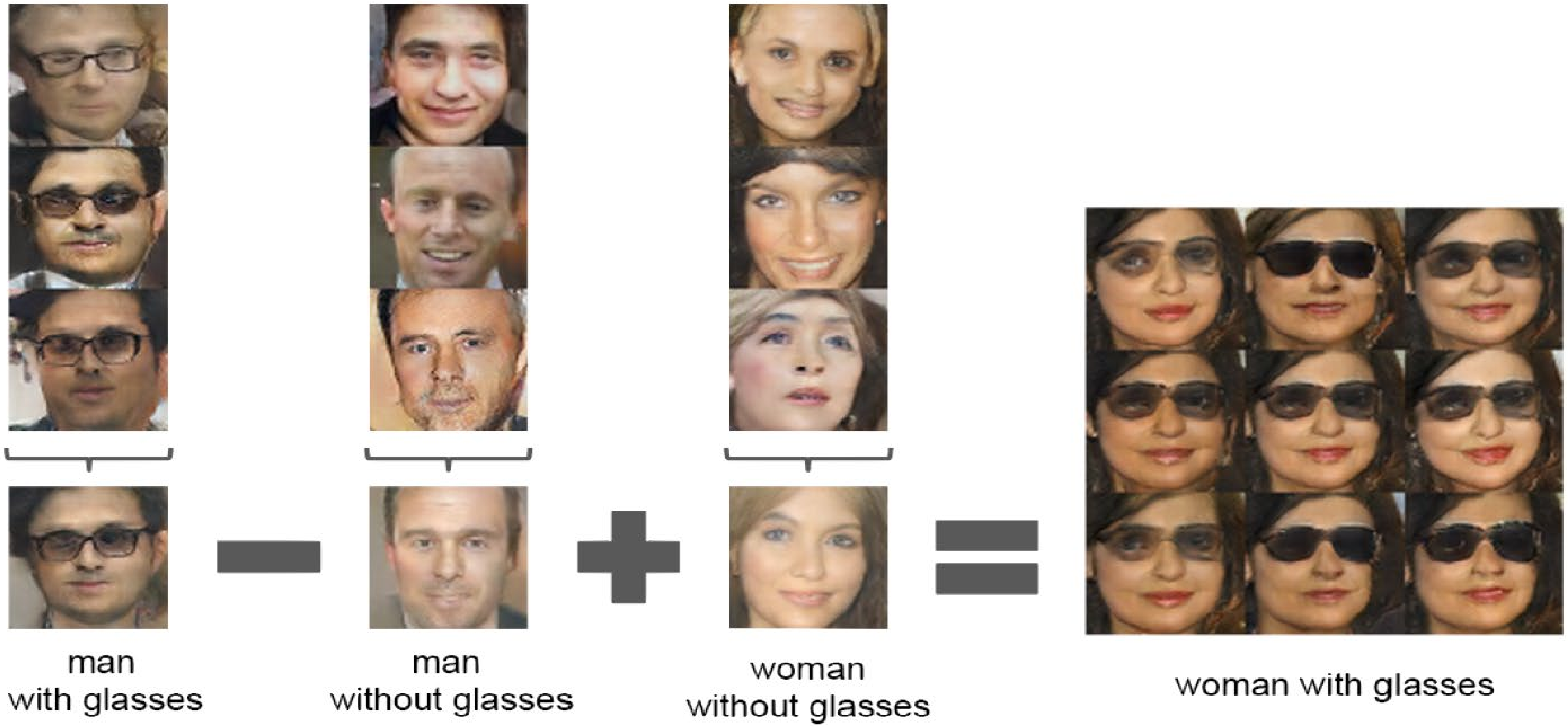
Also for 3D

3D Chairs

(a) Rotation (b) Width

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. InfoGAN: Interpretable Representation Learning by Information Maximization Generative Adversarial Nets, NIPS (2016).

Latent Space Arithmetic



Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).

GAN Advantages

- Generation is straightforward
- Mode detail is captured
- Training does not require MLE estimation
- Robust to overfitting (generator never sees the training data)
- Impressive empirical results

GAN Issues

- Learned probability distribution is implicit
  - Vanilla GANS only good for sampling/generation

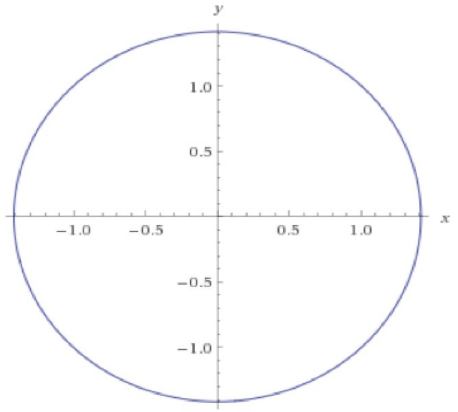
- Training is difficult and often unstable
  - Non-convergence
  - Vanishing gradients
  - Mode collapse

Explanation and Some Remedies

- Non-convergence
  - Stochastic gradient descent was not designed to find the Nash equilibria in multi-player games
  - Competition between generator and discriminator can cause instabilities
  - A black art, addressed through adding noise to discriminator inputs, toying with learning rates, and various regularizations
- Vanishing gradients
  - Generator training can fail if the discriminator is too good -- an optimal discriminator doesn’t provide enough information for the generator to make progress
  - As gradients flow backwards, they can become so small that the early generator layers stop changing
  - A key contribution to address this was the Wasserstein loss, using transportation metrics

$$\min_x \max_y \, xy$$

- $\frac{\partial}{\partial x} = -\mathrm{y} \quad \ldots \quad \frac{\partial}{\partial y} = \mathrm{x}$
- $\frac{\partial^2}{\partial y^2} = \frac{\partial}{\partial x} = -y$

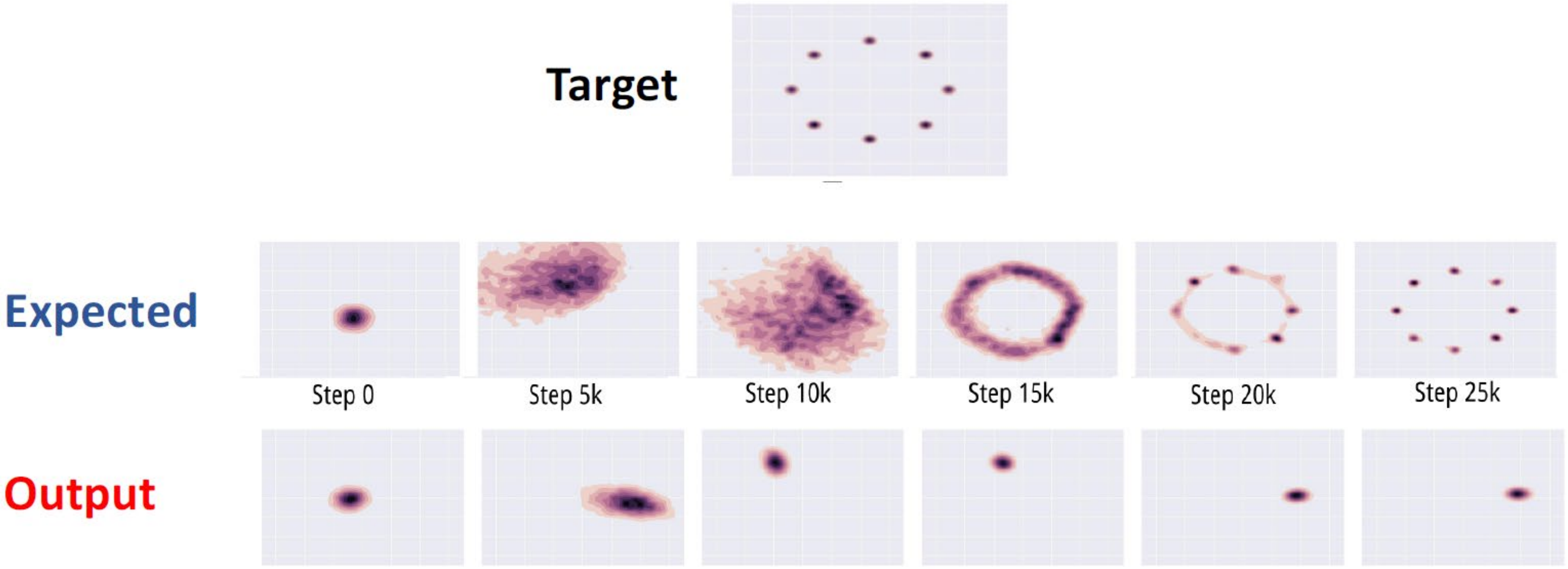


- diff eq has sinusoidal terms
- will not converge, even with small leaning rate

Wasserstein GAN, Martin Arjovsky, Soumith Chintala, Léon Bottou, 2017 https://arxiv.org/abs/1701.07875

Explanation and Some Remedies

- Mode collapse
  - Generator fails to produce diverse-enough samples



Metz, Luke, et al. **"Unrolled Generative Adversarial Networks."** arXiv preprint arXiv:1611.02163 (2016).

  - Remedy: let the discriminator looks at the entire batch, not just a single sample – mark as "fake" if there is lack of diversity

Disentanglement: InfoGAN

- Disentanglement means that individual latent dimensions capture independent key attribute of the output
- How to achieve disentanglement without explicit supervision?

- InfoGAN approach:
  - partition the noise vector into 2 parts
    - a *z* vector that will capture slight/local variations in the output
    - a small *c* vector will capture the main attributes of the output
  - maximize mutual information between *c* and the generated data

$$I(X;Y) = \sum_{x,y} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right)$$

$$I(X;Y) = H(X) - H(X \mid Y) = H(Y) - \mathrm{H}(Y \mid X)$$

$$\min_G \max_D E_I(D,G) = E(D,G) - \lambda I(c; G(z,c))$$

  - For MNIST, *c* dimensions correlate with digit class, thickness, slope, etc.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. InfoGAN:
Interpretable Representation Learning by Information Maximization Generative Adversarial Nets https://arxiv.org/abs/1606.03657

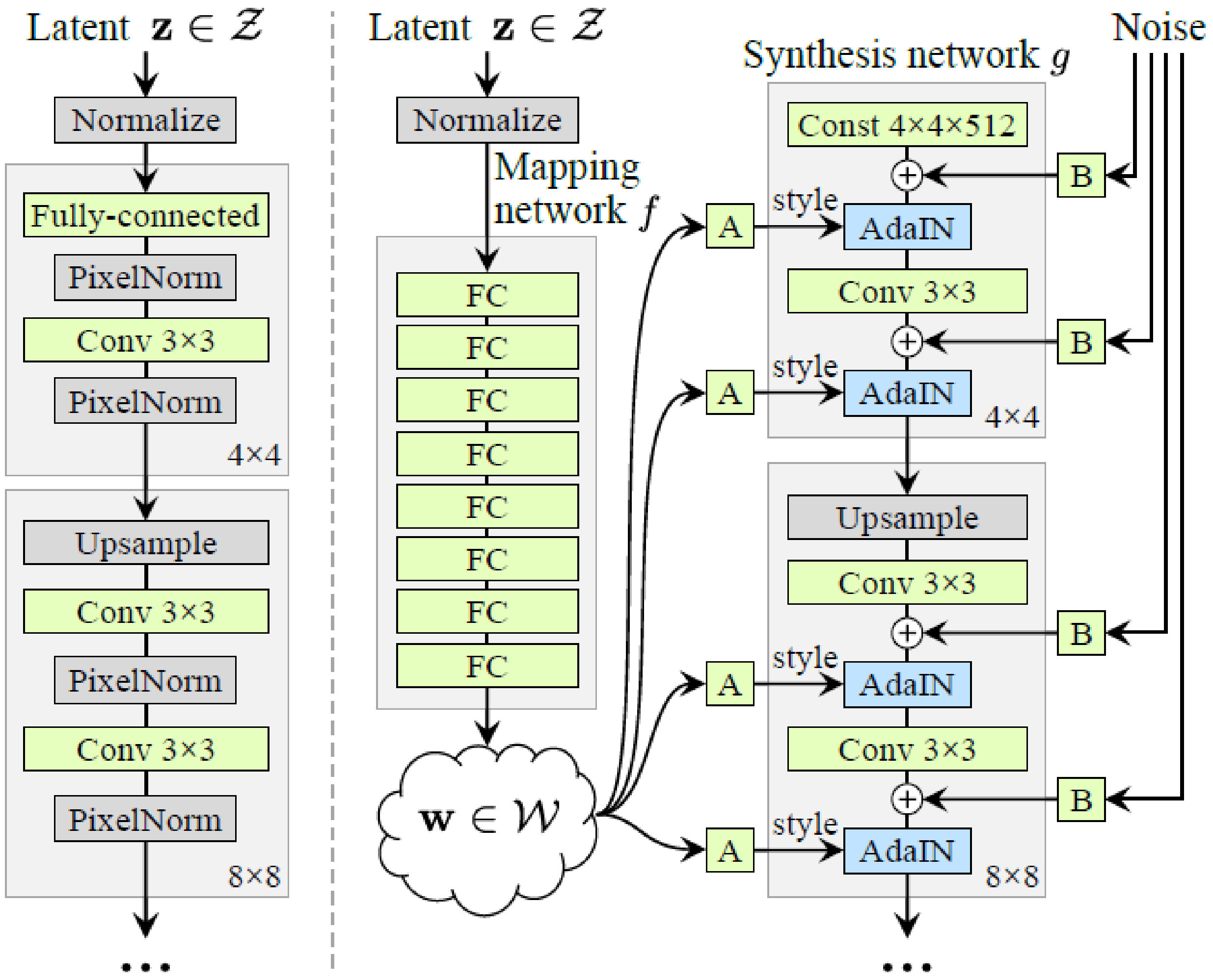
Disentanglement: StyleGAN

- An alternative architecture based on the style transfer literature

- Allows unsupervised separation of high-level attributes (e.g., pose vs identity for human faces) as well as stochastic variation (freckles, facial hair)

- Allows control of the synthesis

StyleGAN Architecture (Image Generation)



(a) Traditional (b) Style-based generator

A = learned affine transform

B = per-channel scaling factors

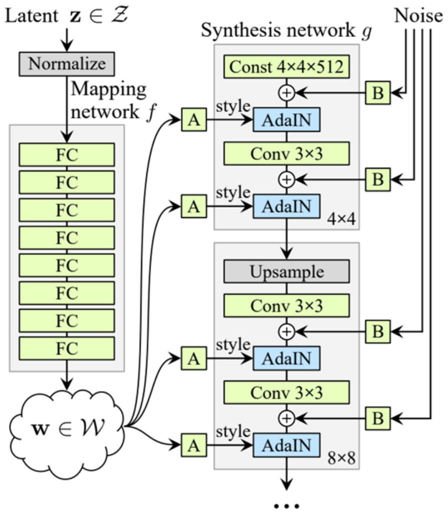
AdaIN = adaptive instance normalization

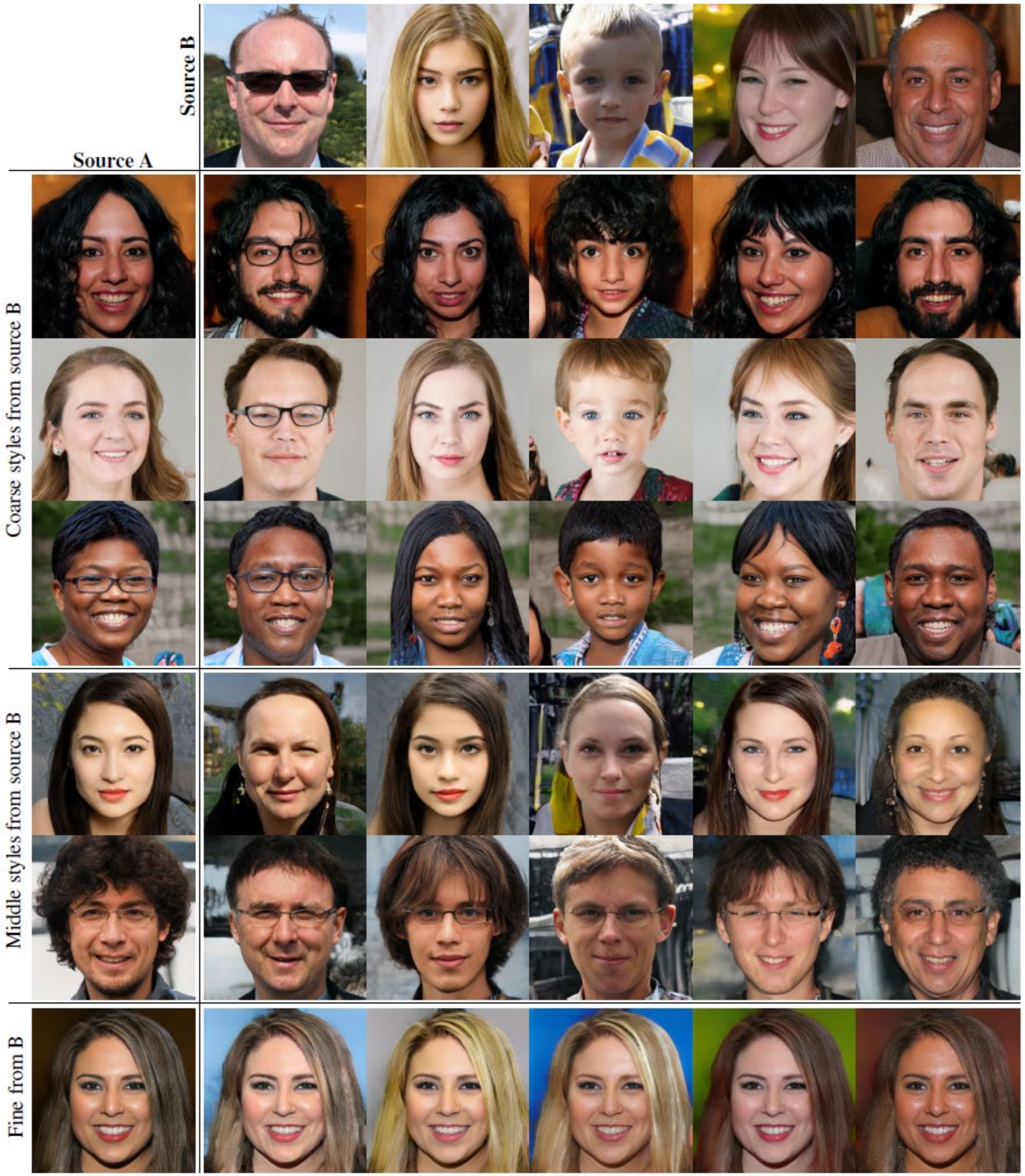
In *Instance Normalization*, mean and variance are calculated for each individual channel for each individual sample across both spatial dimensions

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$

Example Generations

Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions (4^2 – 8^2) brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A. If we instead copy the styles of middle resolutions (16^2 – 322) from B, we inherit smaller scale facial features, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved. Finally, copying the fine styles (64^2 – 1024^2) from B brings mainly the color scheme and microstructure.





StyleGAN Encoder

Learn a **latent representation** of an image from a (pre-trained) StyleGAN generator

+

Latent directions trained from a predictive classifier using latent space as features

=

Image transformations

How does the image look like in latent space?

To manipulate the images to e.g. smile

Learn a **latent representation** of an image from a (pre-trained) StyleGAN generator

Image + Pre-trained StyleGAN generator

Latent directions trained from a predictive classifier using latent space as features

Image + A handful of labelled attributes e.g. is the image smiling? Gender? Age?

That’s All