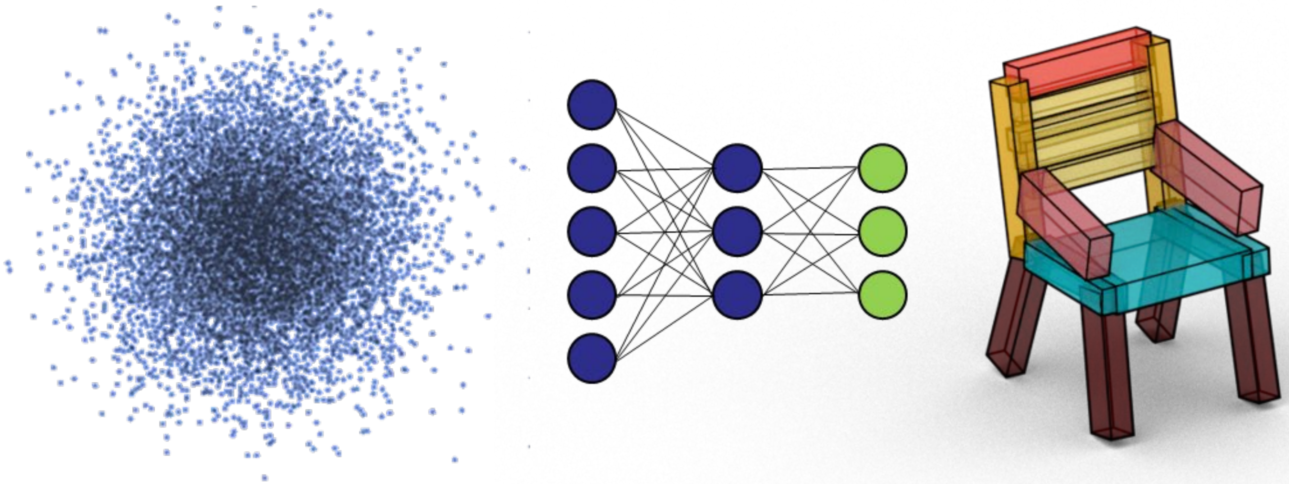
CS348n: Neural Representations and Generative Models for 3D Geometry



Leonidas Guibas
Computer Science Department
Stanford University

Leonidas Guibas Laboratory
Geometric Computing

Last Time: Autoregressive and Flow Models

Autoregressive Models

The term ***autoregressive*** originates from the literature on time-series models where observations from the previous time-steps are used to predict the value at the current time step.

Put simply, an autoregressive model is merely a feed-forward model which predicts future values from past values:

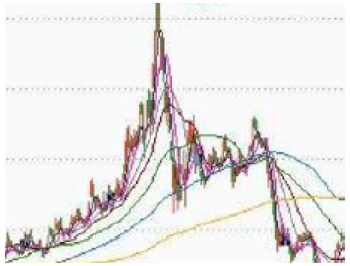
$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p\, y_{t-p} + \varepsilon_t \quad , \quad \varepsilon_t \sim \mathrm{N}(0, \sigma^2)$$

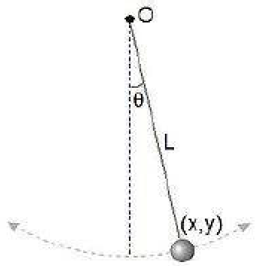
y_i could be:

The specific stock price of day i...

The amplitude of a simple pendulum at period i...

Or any variable that depends on its preceding values!





Autoregressive Models: Factorization

Main challenge: distributions over high dimensional objects is actually very sparse!!

Too many possibilities! ⟶ Main idea: **write as a product of simpler terms**

Definition of conditional probability:

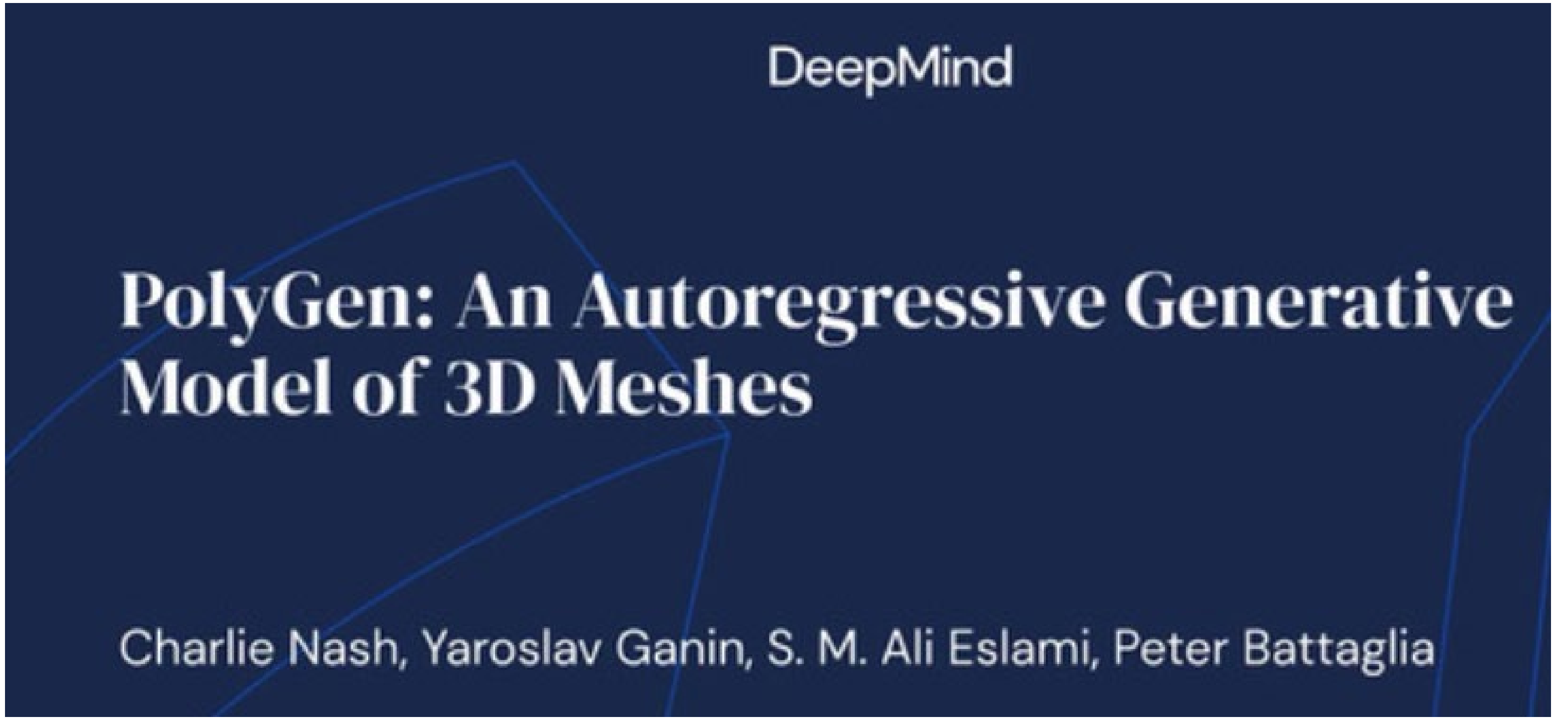
$$P(x_1, x_2) = P(x_1)\,P(x_2|x_1)$$

Product rule:

$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i|x_{<i})$$

Divide and conquer ! We can solve the joint distribution $P(\boldsymbol{x})$ by solving simpler conditional distributions $p_\theta(x_i|x_{<i})$ one by one

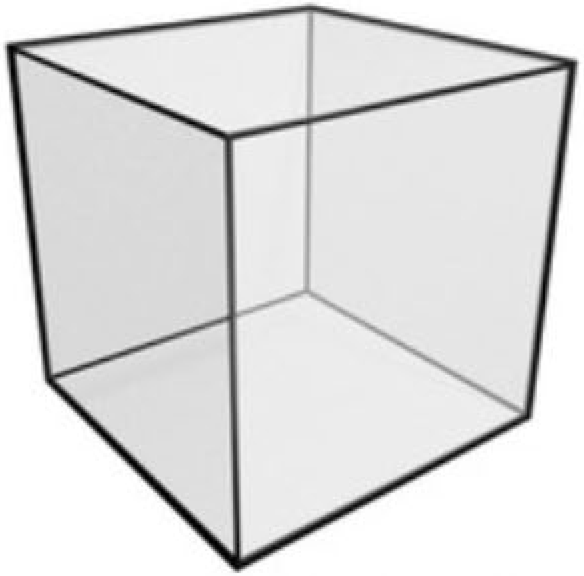
Can you tell the exact likelihood of the next pixel (noted as a red point) conditioned on the given pixels?


DeepMind
PolyGen: An Autoregressive Generative Model of 3D Meshes
Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, Peter Battaglia

ICML 2020

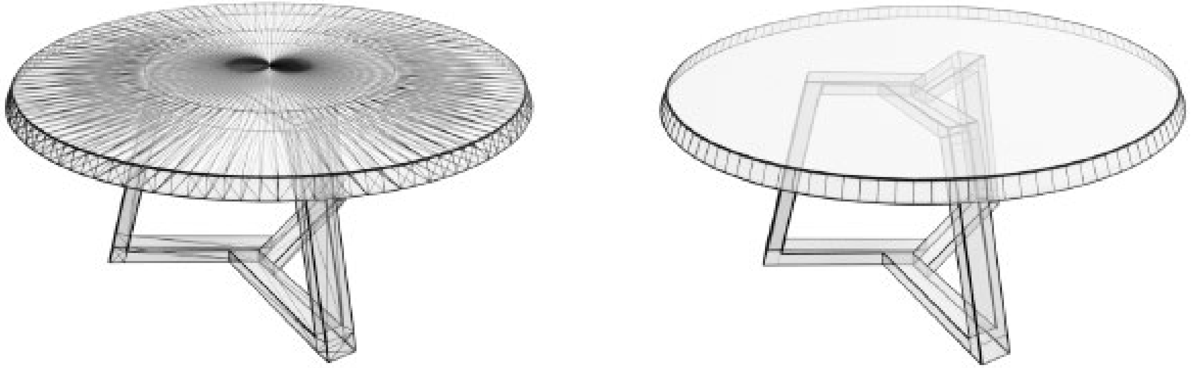
Mesh Representations: OBJ Format

```
# cube.obj
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
f 1 5 7 3
f 4 3 7 8
f 8 7 5 6
f 6 2 4 8
f 2 1 3 4
f 6 5 1 2
```



$$\mathcal{M} = (\mathcal{V}, \mathcal{F})$$

N-Gons Allow More Efficient Representations



(a) Triangle mesh (b) n-gon mesh

Allows

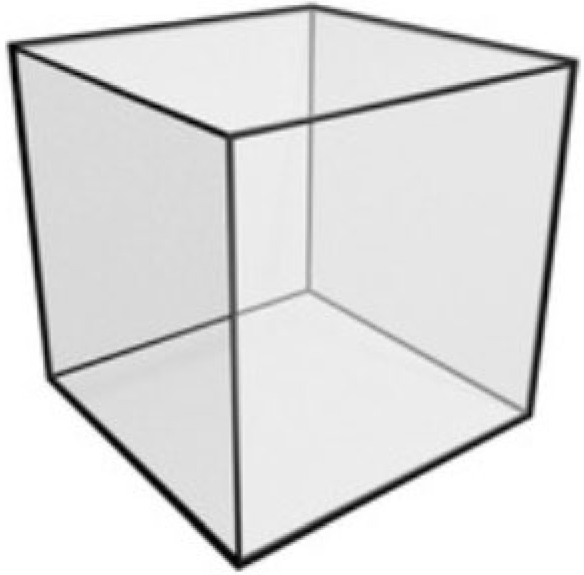
- fewer elements
- more canonical meshes (easier to learn)
- but, polygons need to be planar

Modeling Strategy

$$\boxed{p(\mathcal{V}, \mathcal{F}) = p(\mathcal{V})p(\mathcal{F}|\mathcal{V})}$$

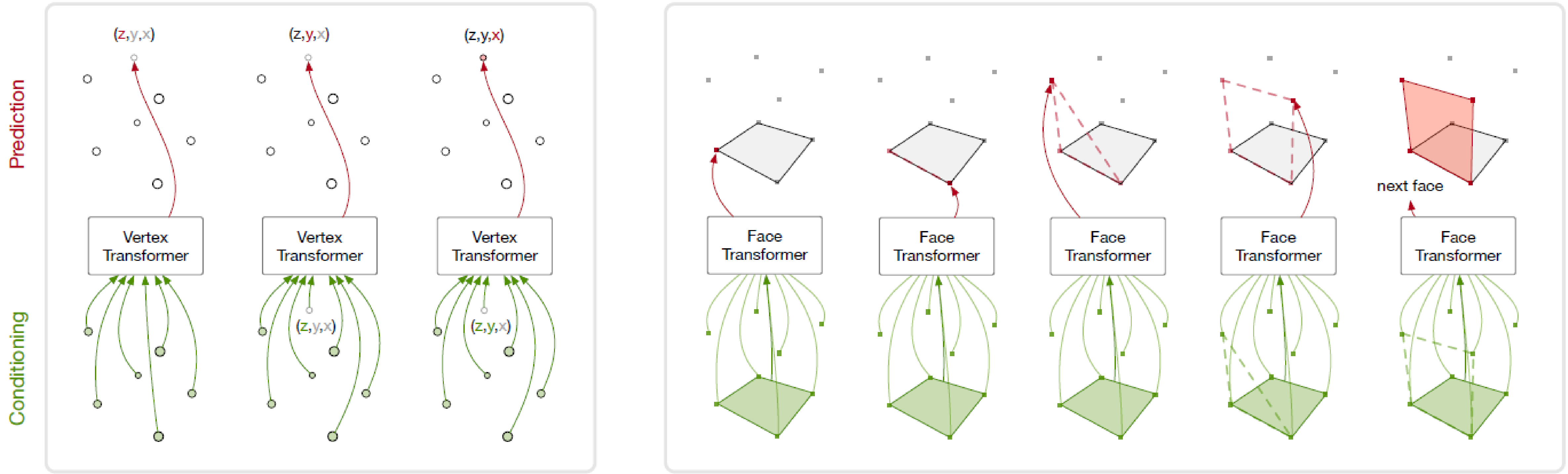
1. Model vertices
2. Model faces given vertices

$p(\mathcal{V})$	Vertex model
$p(\mathcal{F}\|\mathcal{V})$	Face model



$$\mathcal{M} = (\mathcal{V}, \mathcal{F})$$

Modeling Strategy, in More Detail

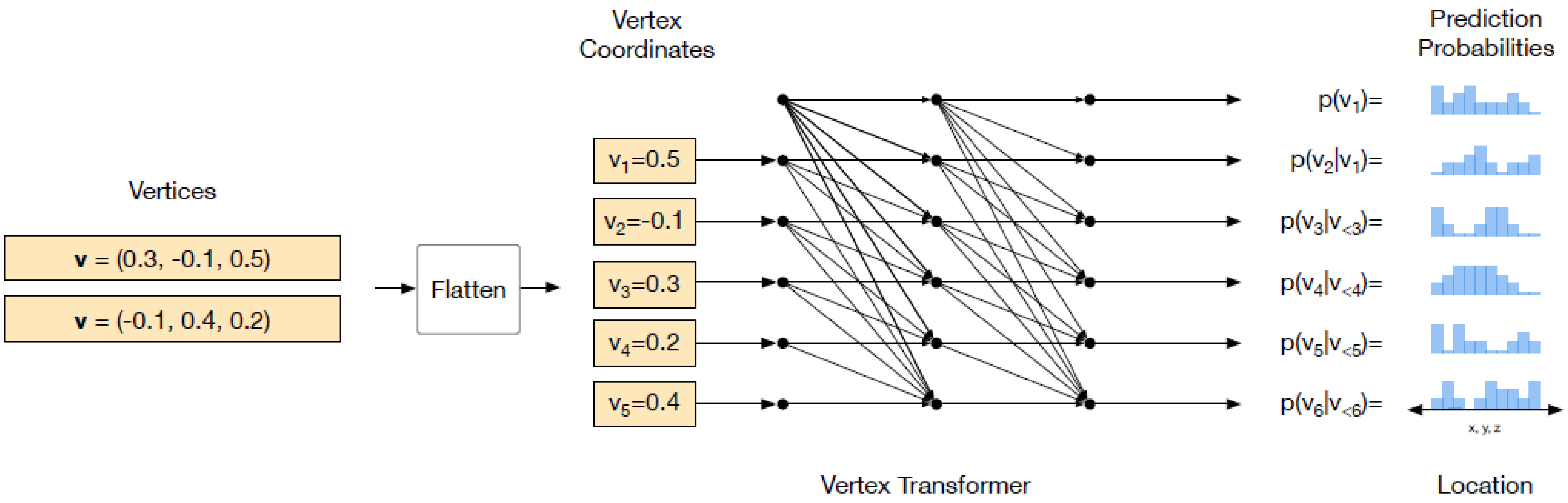


Vertex Model

Face Model

Must mask invalid predictions

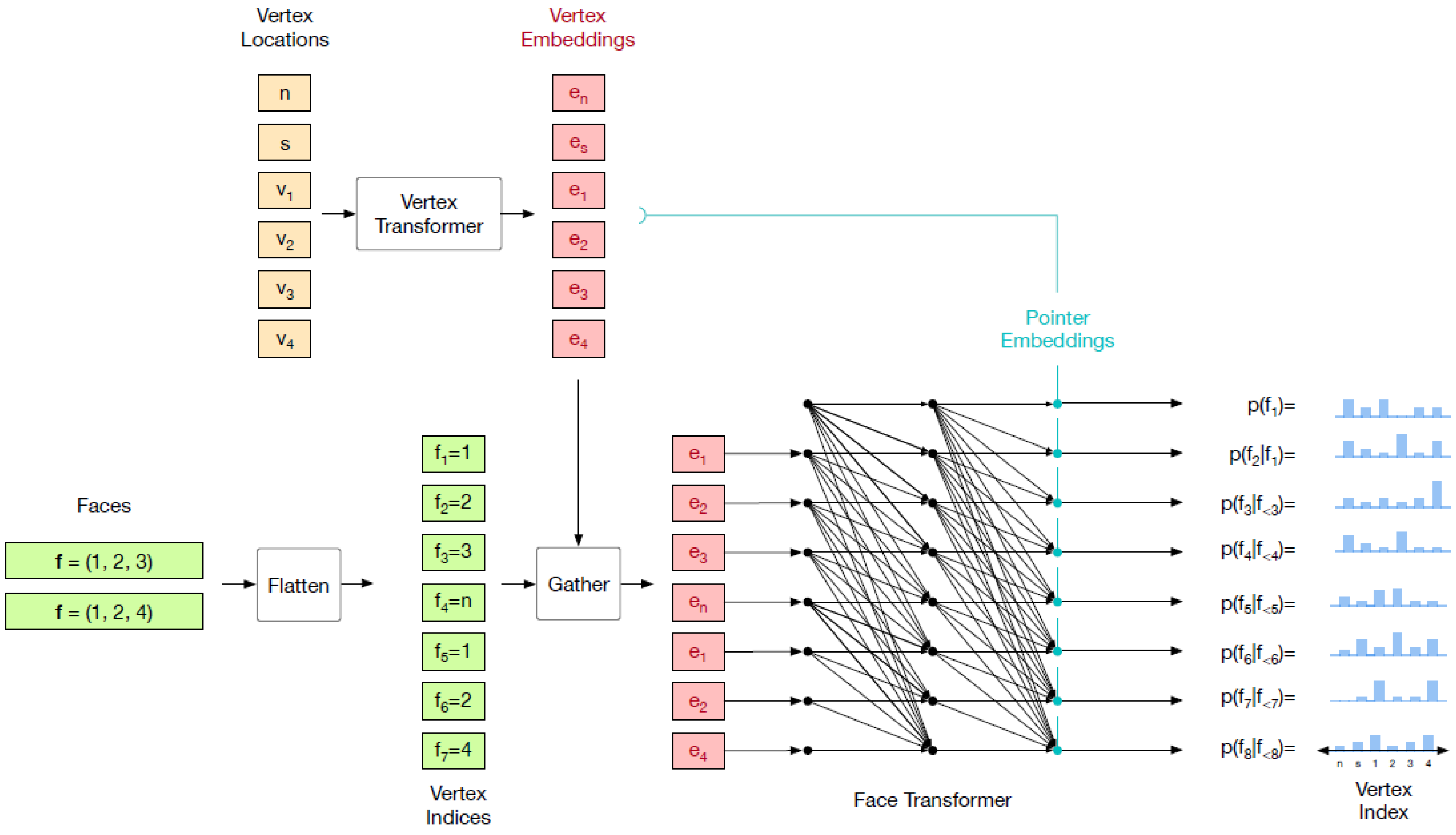
Auto-Regressive Vertex Model



- Outputs **predictive distribution** for sequence of vertex coordinates.
- Train to maximize summed log-probability of sequence (aka cross-entropy loss)

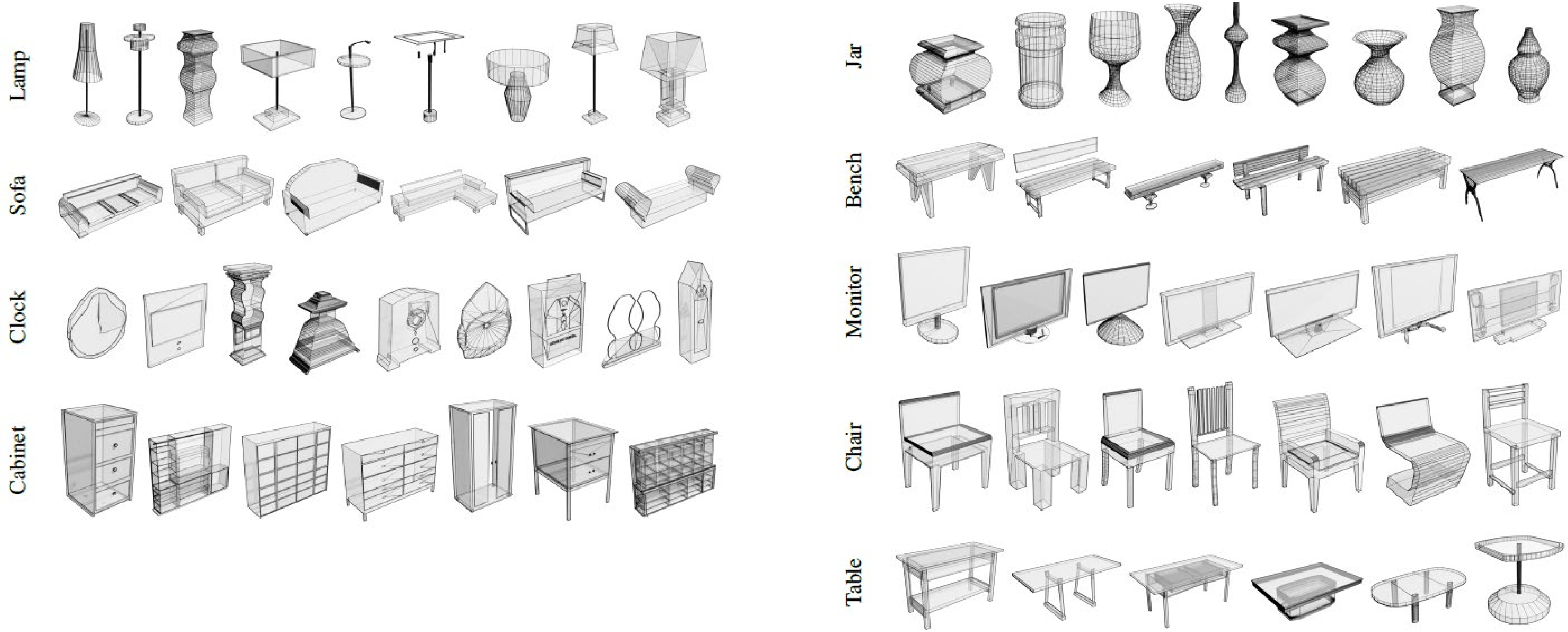
Transformer can learn long-range dependencies: e.g., symmetries

Auto-Regressive Face Model



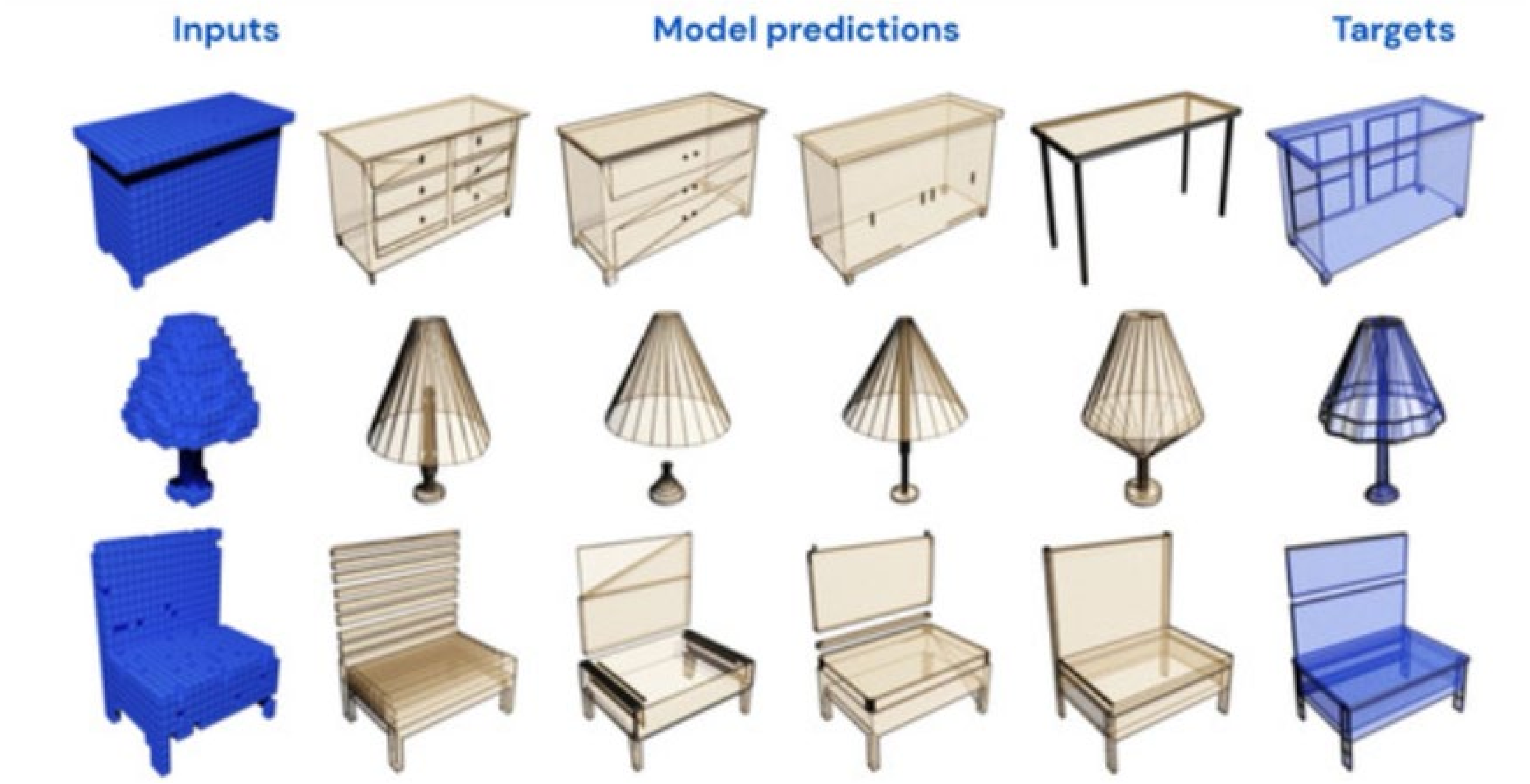
- Vertex encoder produces **contextual vertex embeddings**
- Face model uses vertex embeddings as input -> **outputs pointers**

Examples of Generations

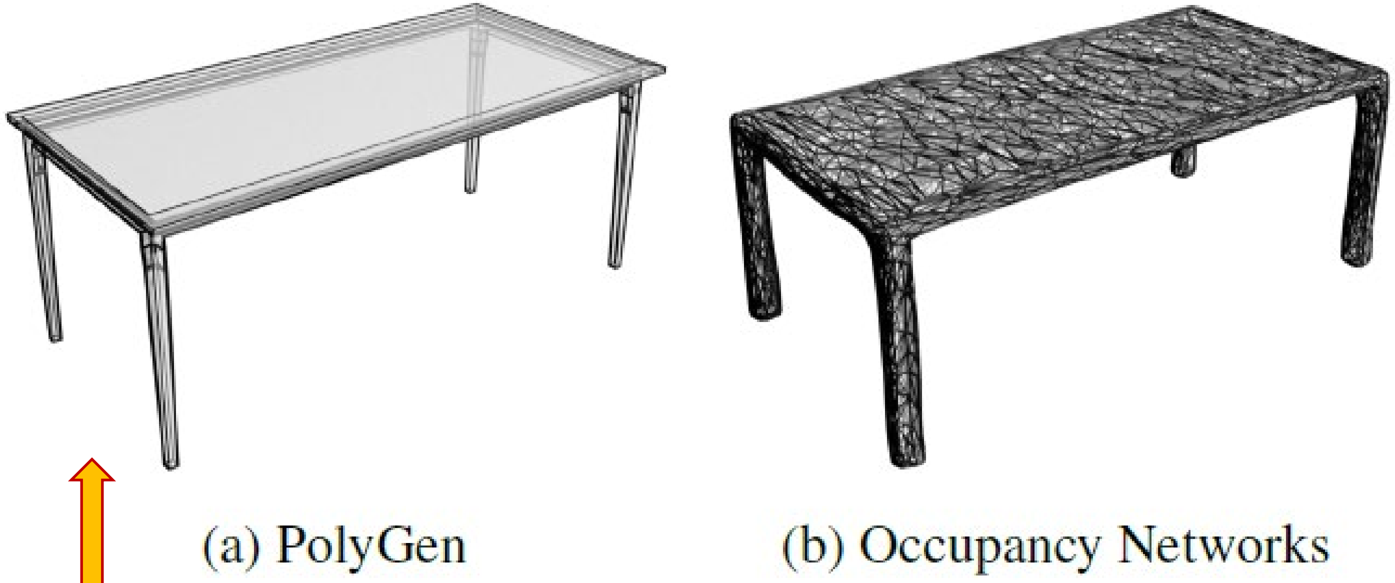


Conditioning on Images

Conditioning on Voxels



Optimizing Chamfer Can Lead to Noisy Meshes

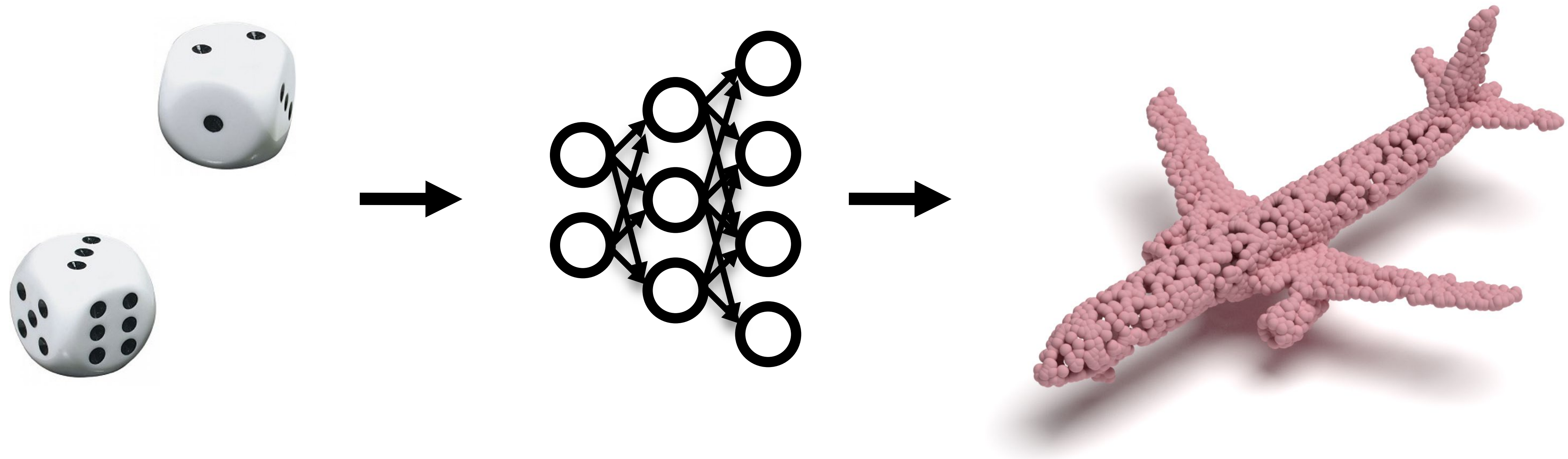


"Vector" representations of geometry – more on Wed

Flow Models, PointFlow

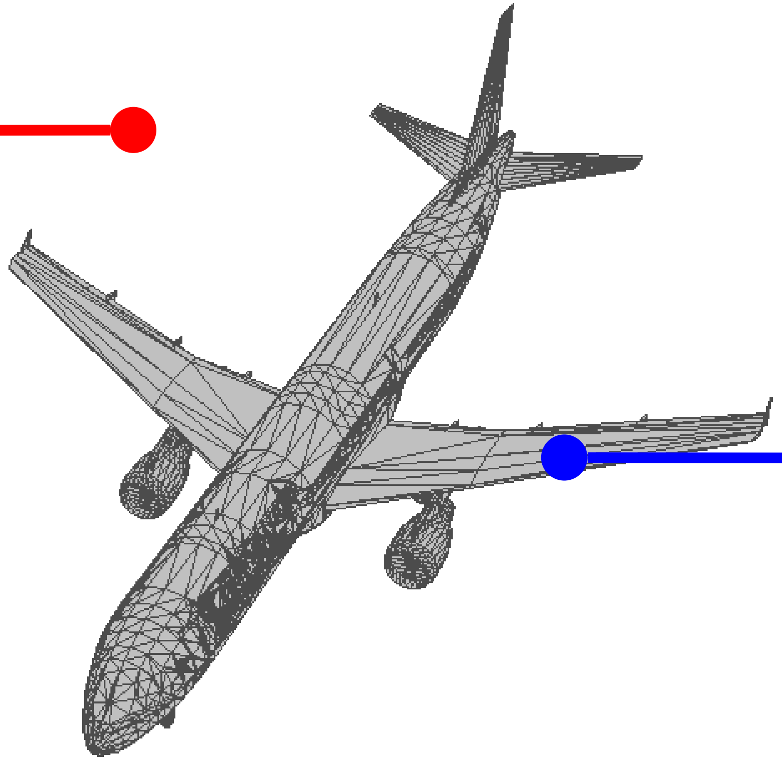
PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows. Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, Bharath Hariharan. ICCV'19

Point Cloud Generation



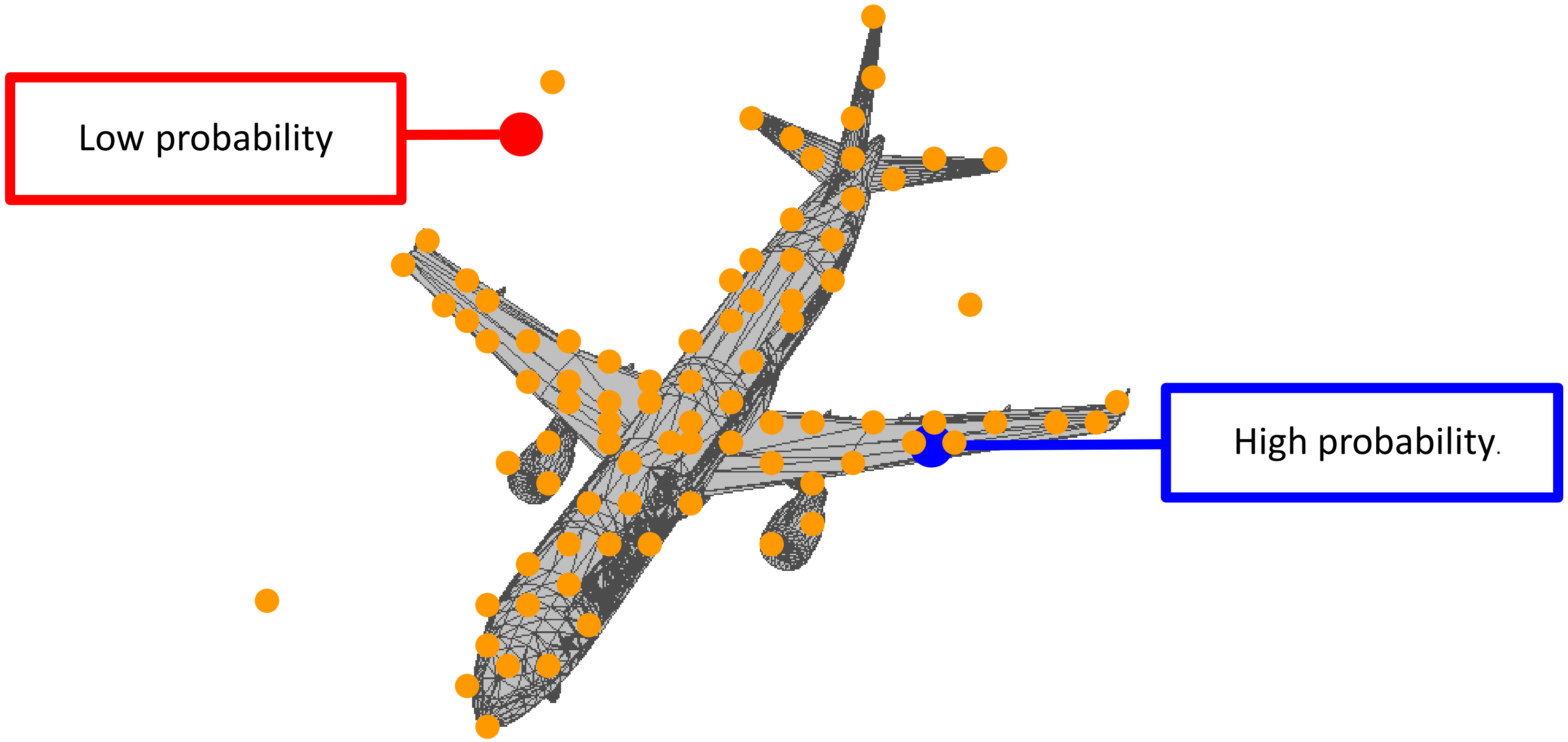
A Shape is a Distribution of 3D Points

Off-surface points have low probability.



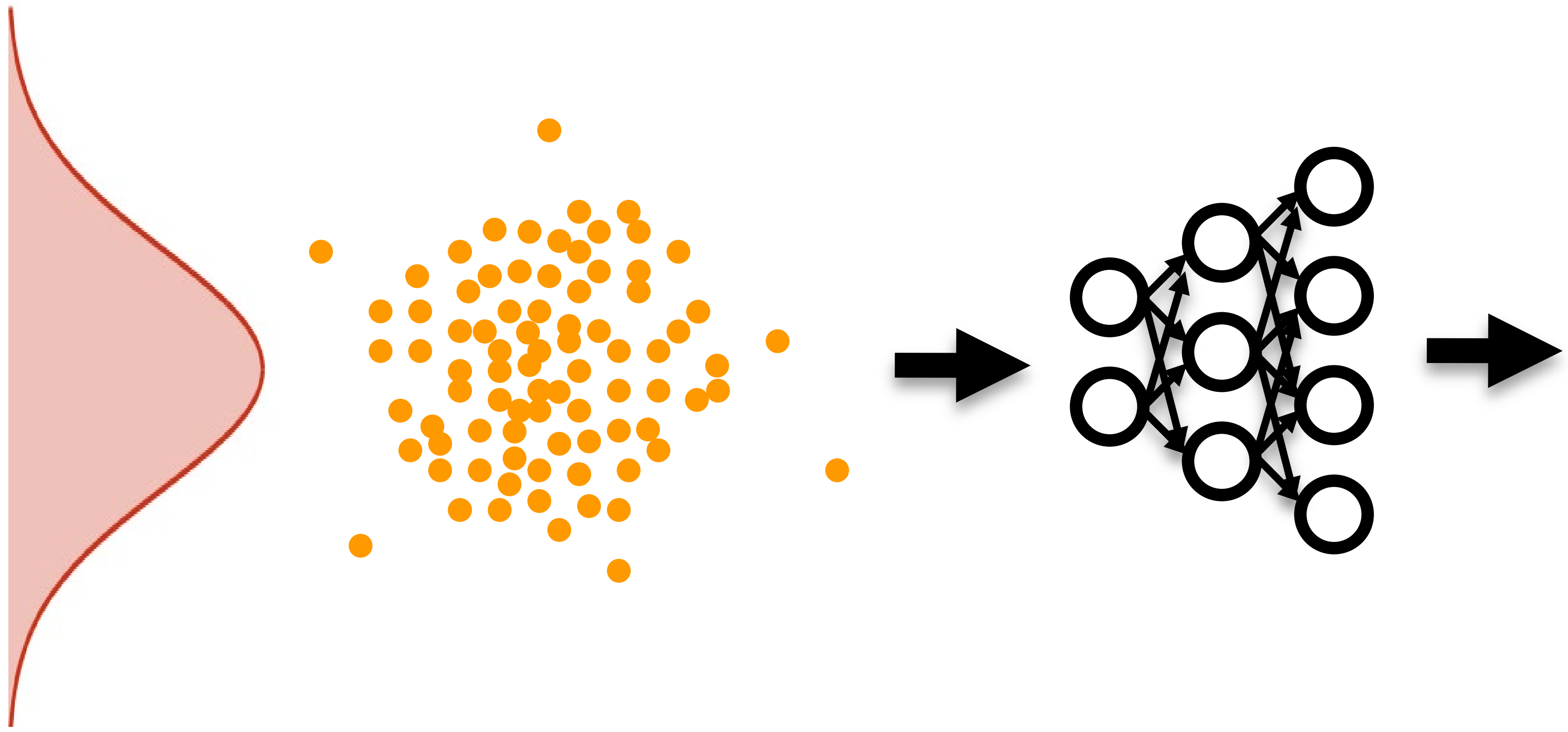
Near-surface points have high probability.

A Point Cloud Sampled from a Distribution

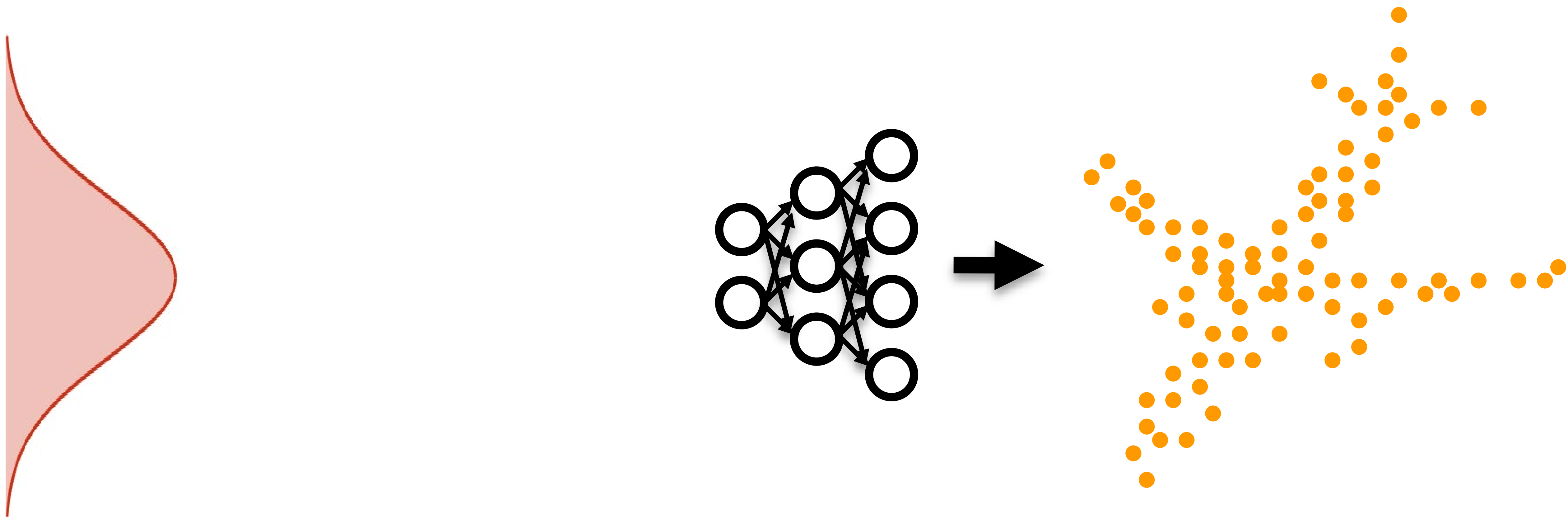


How to Model such a Distribution of 3D Points?

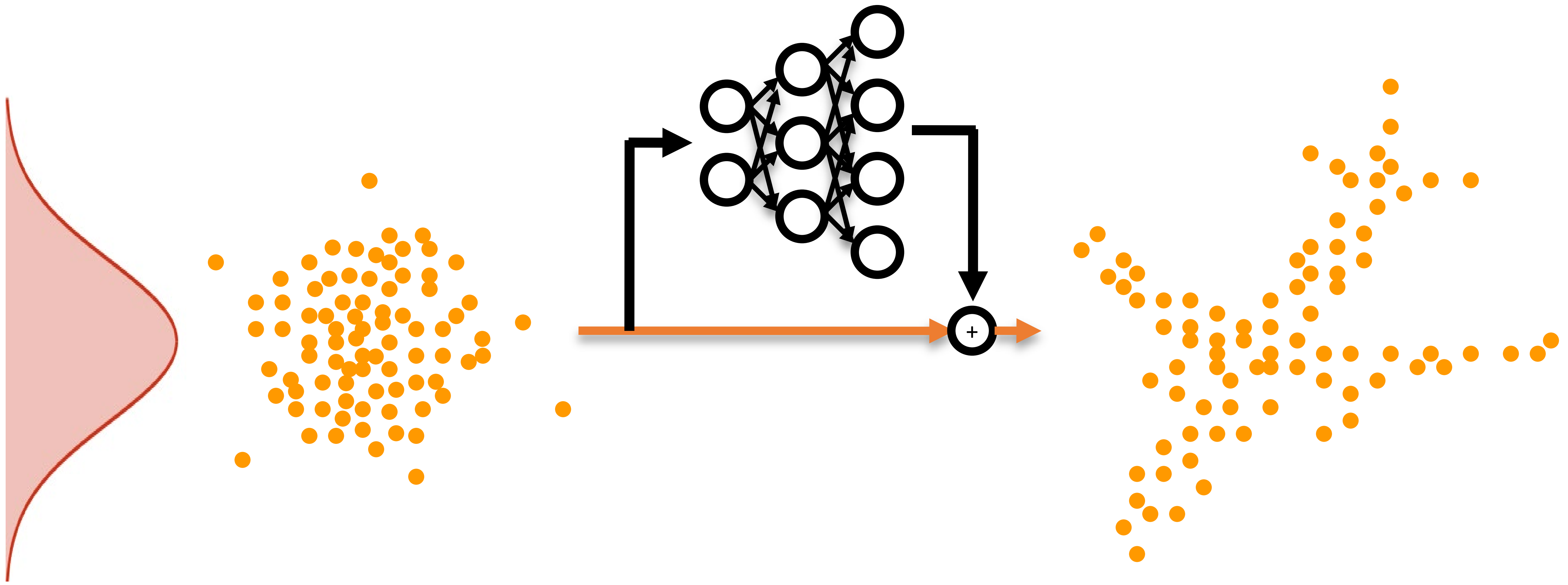
Transforming a 3D Gaussian PC to a Shape



Transforming a 3D Gaussian PC to a Shape



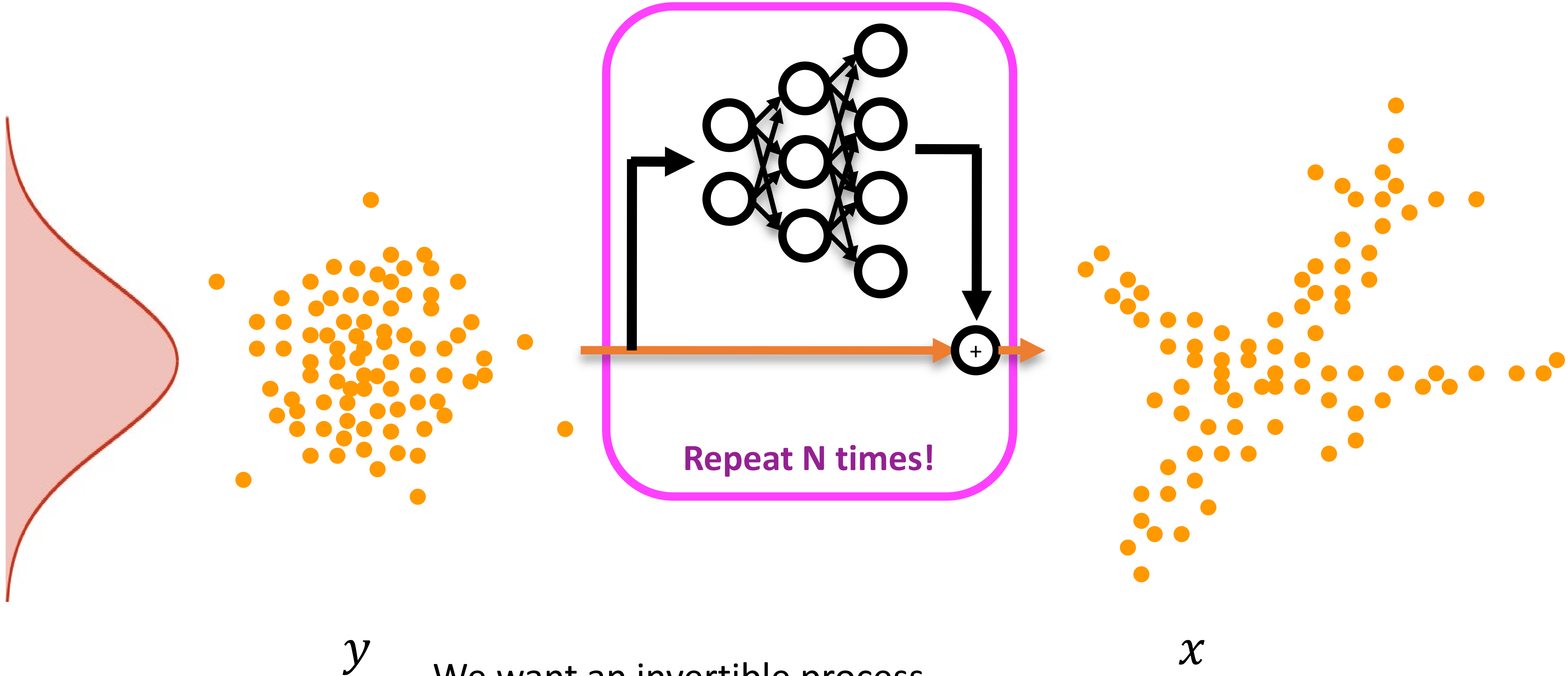
Transforming a 3D Gaussian PC to a Shape



y

x

Transforming a 3D Gaussian PC to a Shape



We want an invertible process,
transforming the point cloud *y* to *x* in a series of small steps

Invertible Transforms: from Discrete to Continuous

$$x = f_n \circ f_{n-1} \circ \cdots \circ f_1(y) \qquad y = f_1^{-1} \circ \cdots \circ f_n^{-1}(x)$$

discrete

$$\log P(x) = \log P(y) - \sum_{k=1}^{n} \log \left| \det \frac{\partial f_k}{\partial y_{k-1}} \right|$$

f_k Invertible neural network layers with simple Jacobians

continuous

$$\frac{\partial y(t)}{\partial t} = f(y(t), t)$$

$$x = y(t_0) + \int_{t_0}^{t_1} f(y(t), t) dt, \quad y(t_0) \sim P(y)$$

Neural ODEs

Instead of specifying a discrete sequence of hidden layers, we parametrize the derivative of the hidden state by a neural network.

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f\left(\mathbf{h}_t, \theta_t\right)$$

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta)$$

Neural Ordinary Differential Equations

Ricky T. Q. Chen*, Yulia Rubanova*, Jesse Bettencourt*, David Duvenaud
University of Toronto, Vector Institute
{rtqichen, rubanova, jessebett, duvenaud}@cs.toronto.edu

Abstract

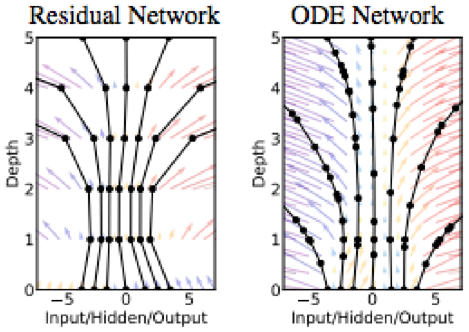
We introduce a new family of deep neural network models. Instead of specifying a discrete sequence of hidden layers, we parameterize the derivative of the hidden state using a neural network. The output of the network is computed using a black-box differential equation solver. These continuous-depth models have constant memory cost, adapt their evaluation strategy to each input, and can explicitly trade numerical precision for speed. We demonstrate these properties in continuous-depth residual networks and continuous-time latent variable models. We also construct continuous normalizing flows, a generative model that can train by maximum likelihood, without partitioning or ordering the data dimensions. For training, we show how to scalably backpropagate through any ODE solver, without access to its internal operations. This allows end-to-end training of ODEs within larger models.

1 Introduction

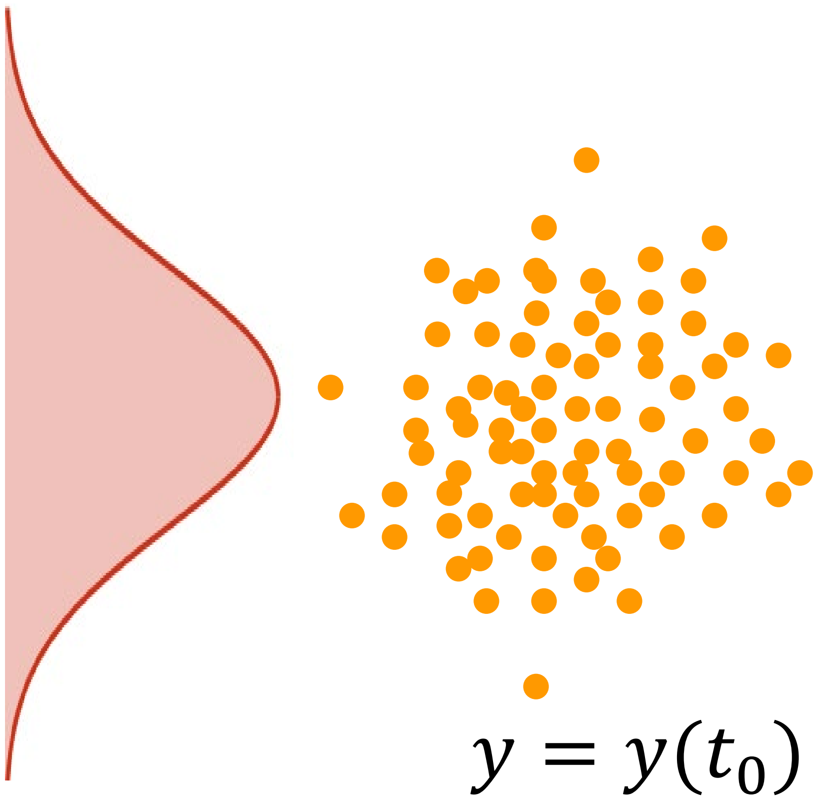
Models such as residual networks, recurrent neural network decoders, and normalizing flows build complicated transformations by composing a sequence of transformations to a hidden state:

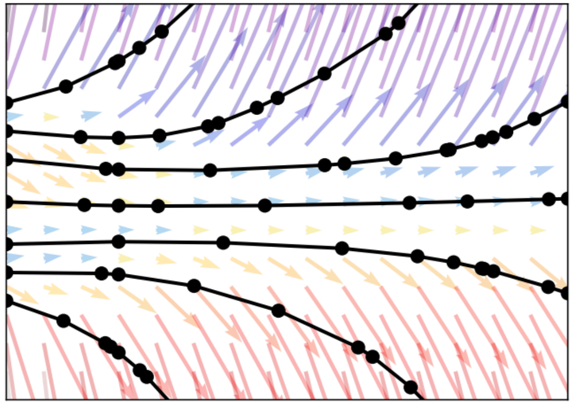
$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t) \qquad (1)$$

where $t \in \{0 \dots T\}$ and $\mathbf{h}_t \in \mathbb{R}^D$. These iterative updates can be seen as an Euler discretization of a continuous transformation (Lu et al., 2017; Haber and Ruthotto, 2017; Ruthotto and Haber, 2018).



Continuous Normalizing Flow (CNF)





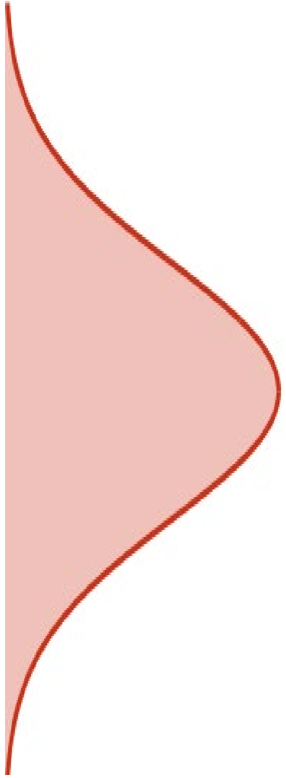
Point CNF

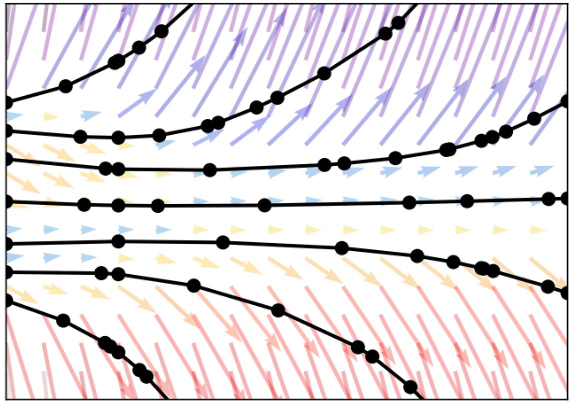
$$y = y(t_0)$$

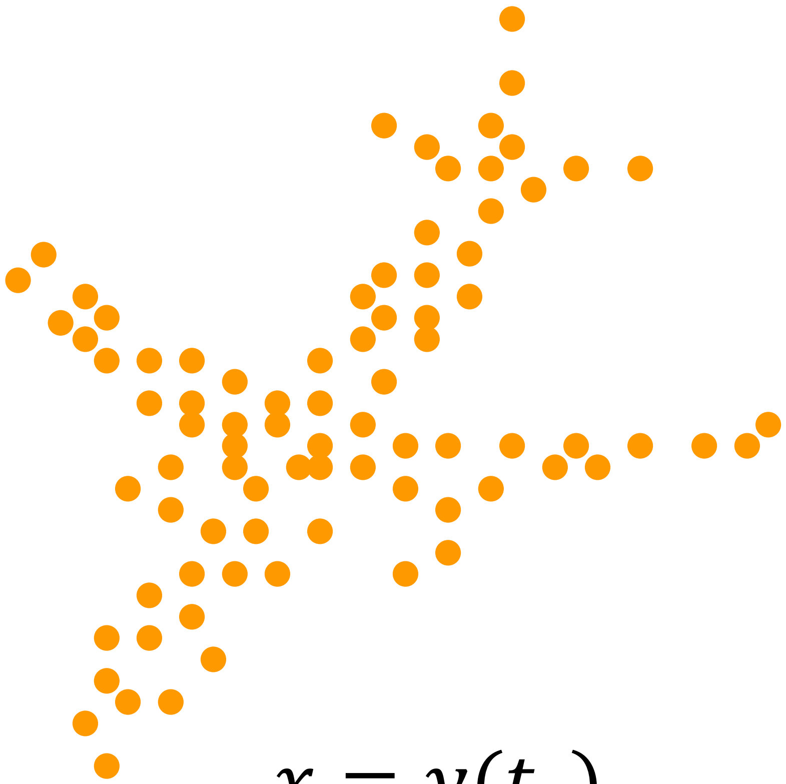
$$x = y(t_1)$$

$$x = y(t_1) = y + \int_{t_0}^{t_1} g_\theta(y(t), t)\,dt$$

Continuous Normalizing Flow (CNF)




Point CNF



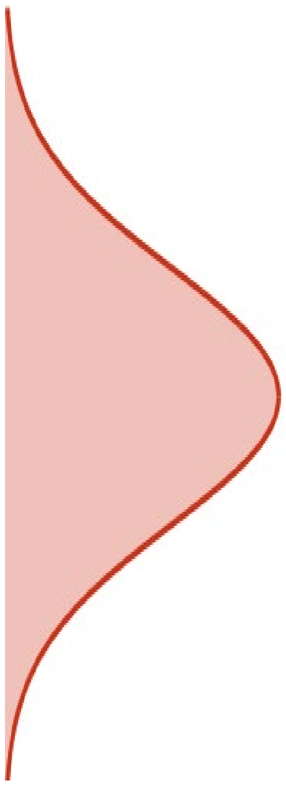
$y = y(t_0)$

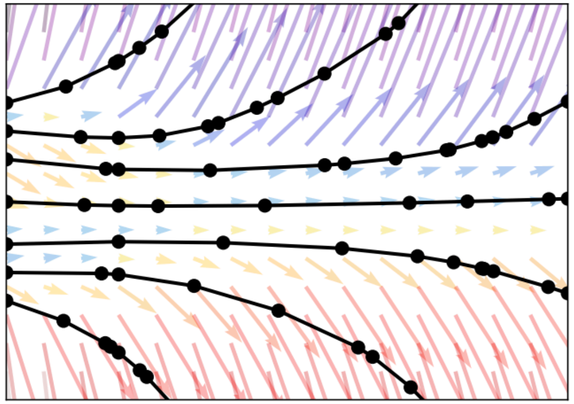
$x = y(t_1)$

$$x = y(t_1) = y + \int_{t_0}^{t_1} g_\theta(y(t), t)dt$$

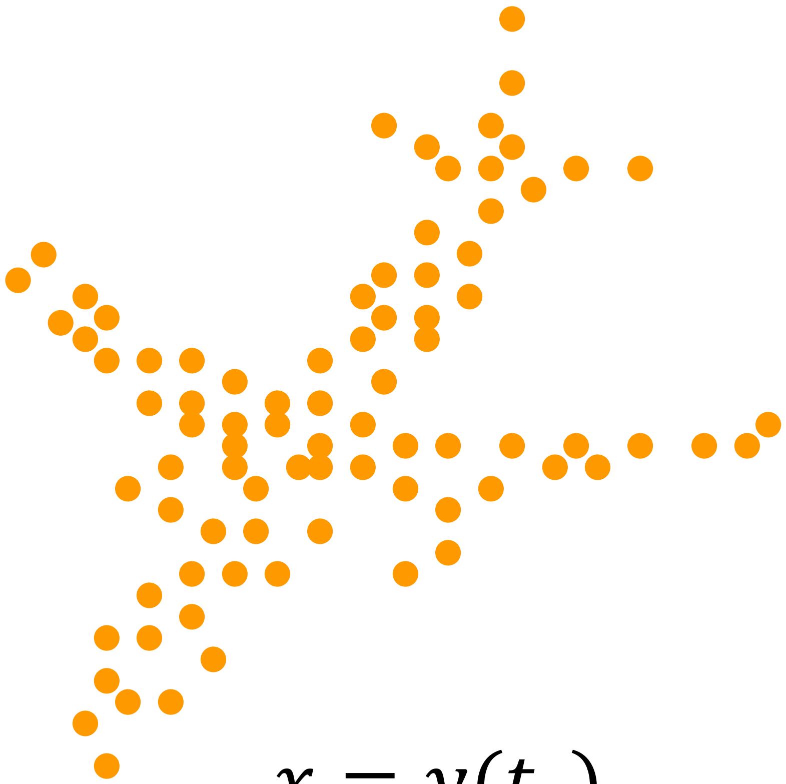
Can be inverted!

CNF is Invertible





Point CNF

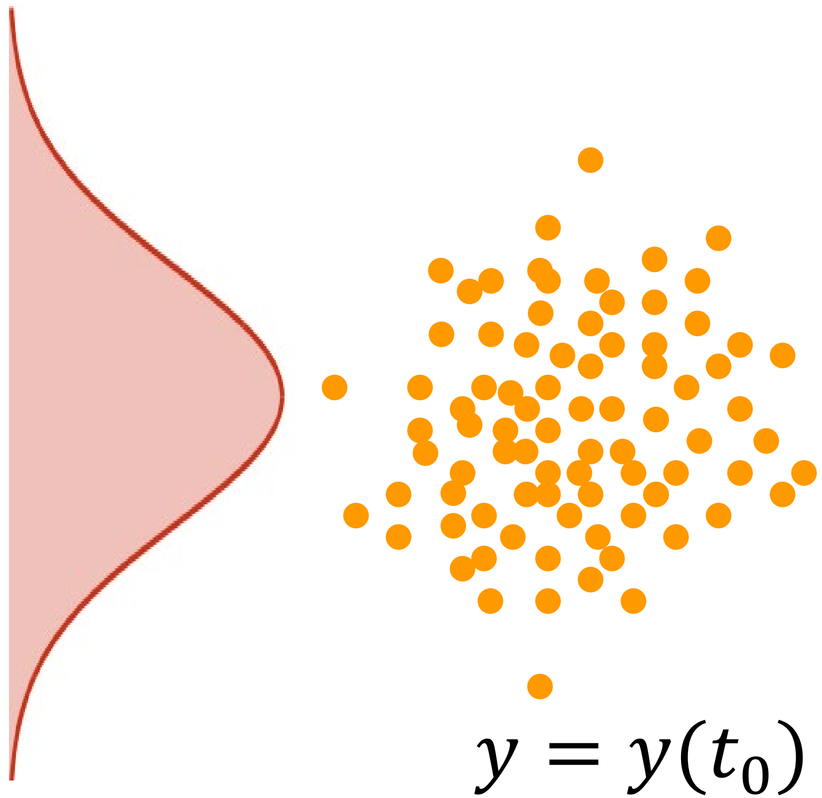


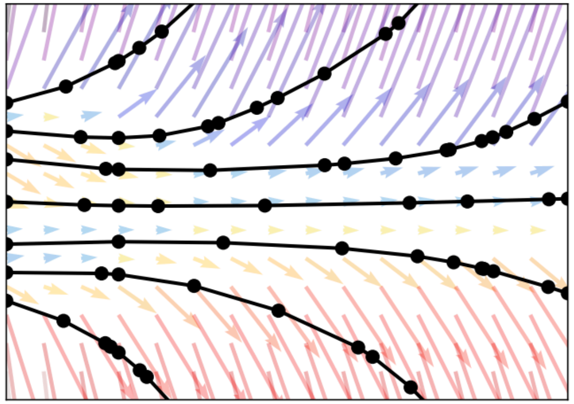
$y = y(t_0)$

$x = y(t_1)$

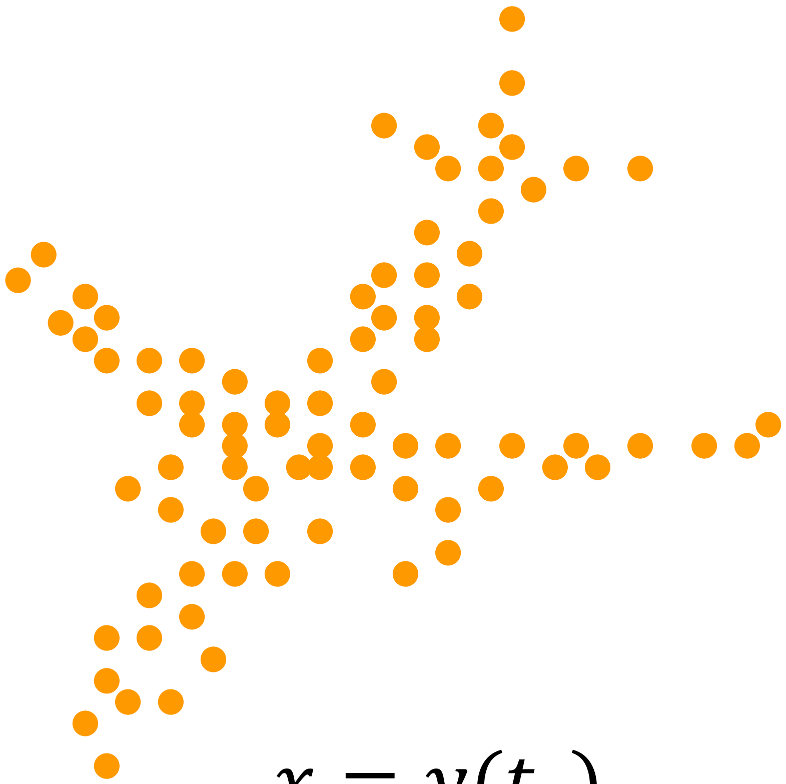
$$y = y(t_0) = x + \boxed{\int_{t_1}^{t_0}} g_\theta(y(t), t)dt$$

CNF is Invertible





Point CNF

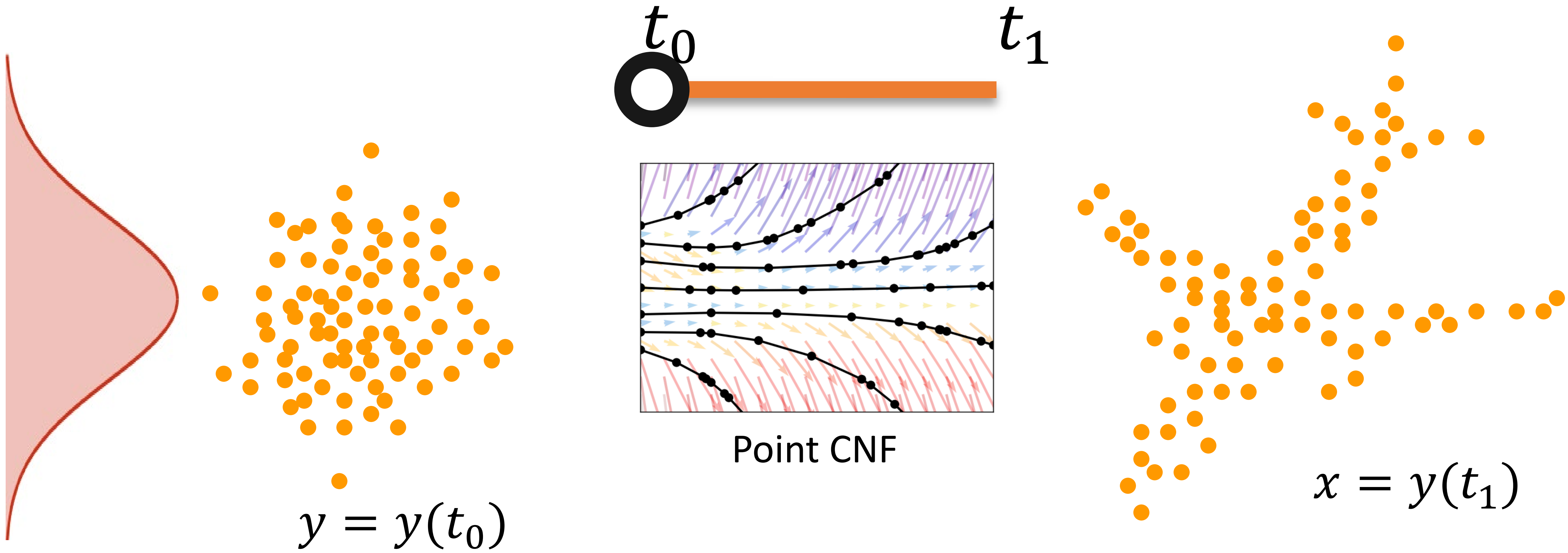


$y = y(t_0)$

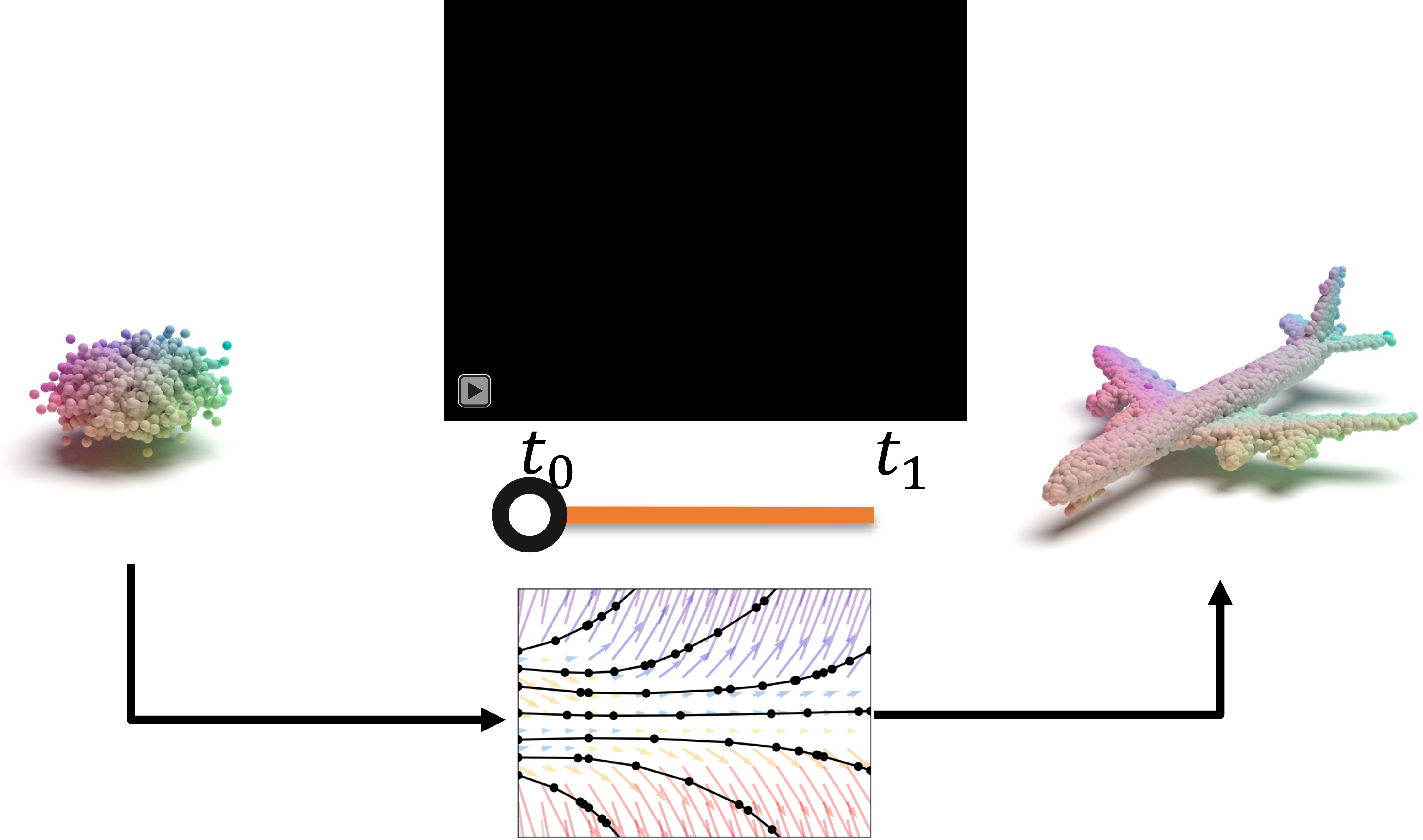
$x = y(t_1)$

$$y = y(t_0) = x + \boxed{\int_{t_1}^{t_0}} g_\theta(y(t), t)dt$$

Change of Variable Formula

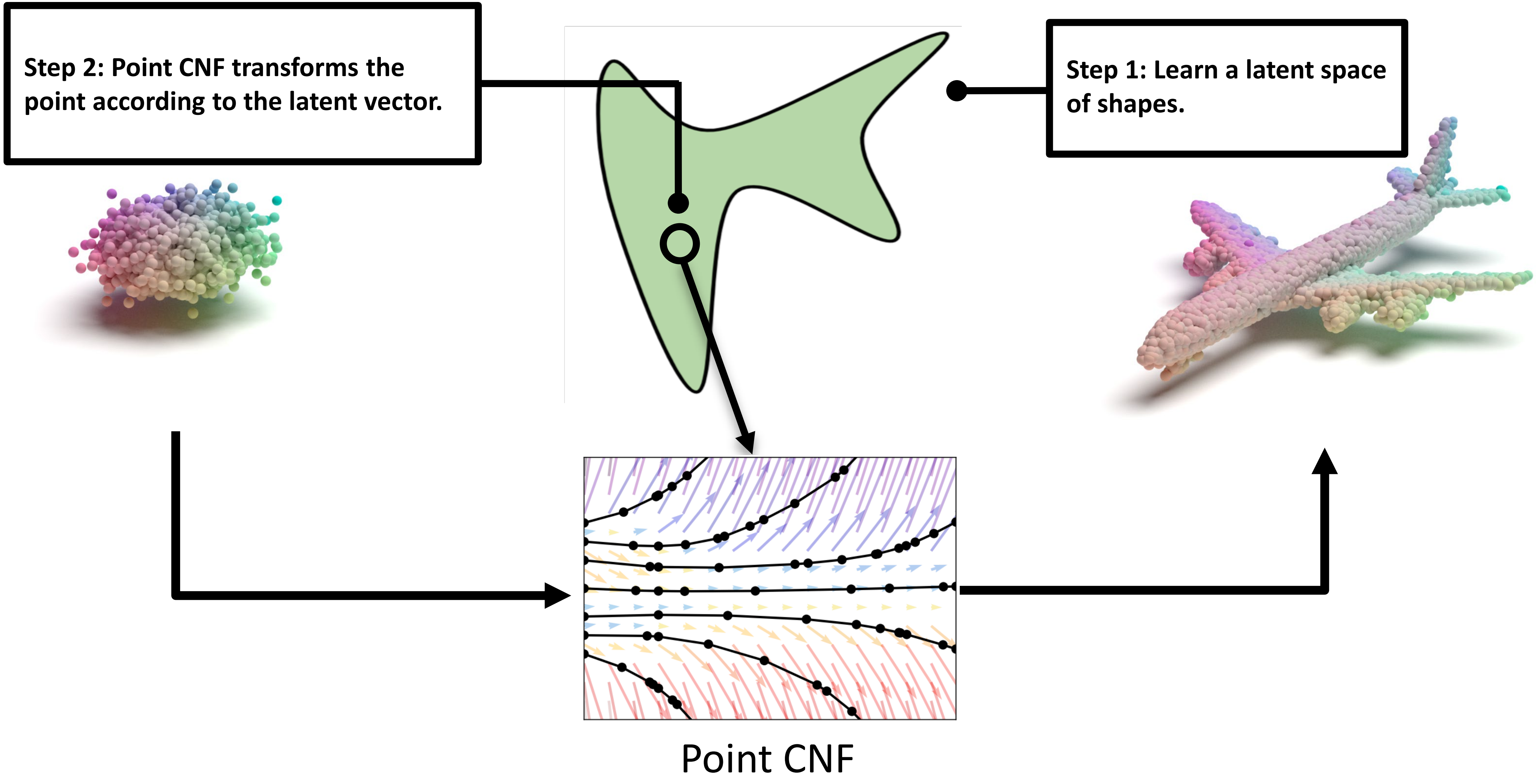


$$\log P(x) = \log P\left(x + \int_{t_1}^{t_0} g_\theta(y(t), t)\,dt\right) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial g_\theta(x(t), t)}{\partial x(t)}\right) dt$$

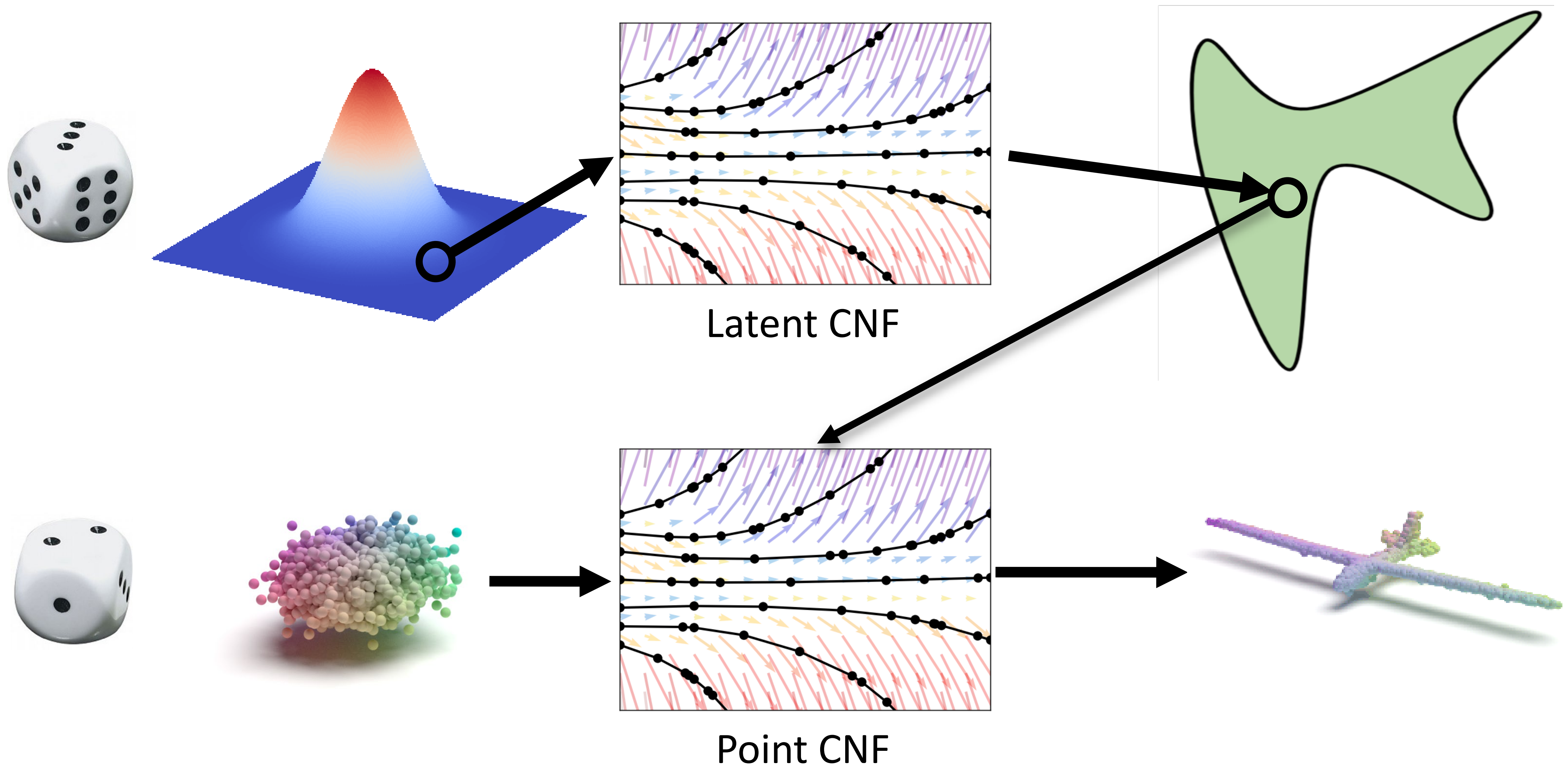


Point CNF

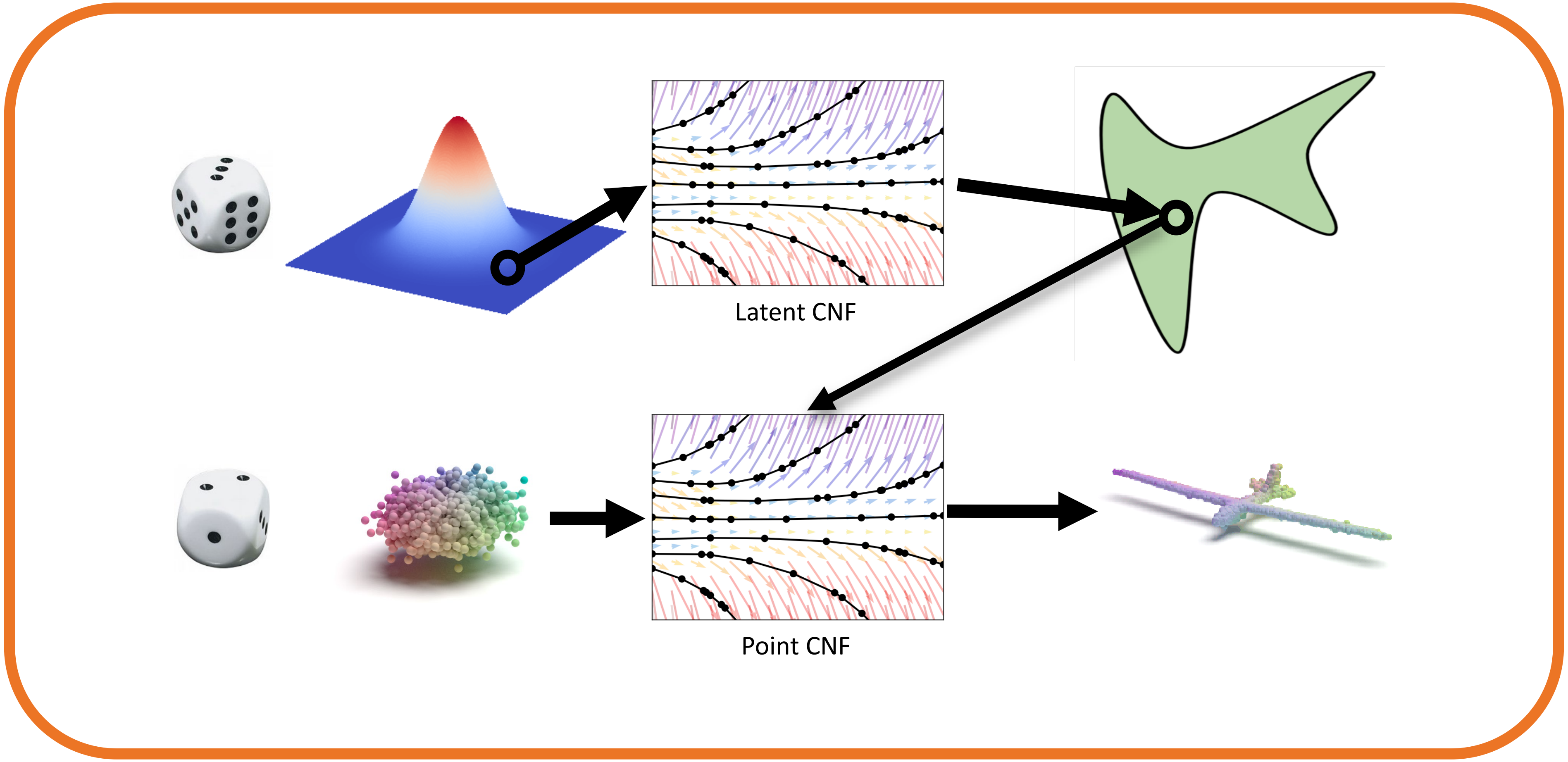
How to Sample Multiple Shapes?



Sample a Novel Shape

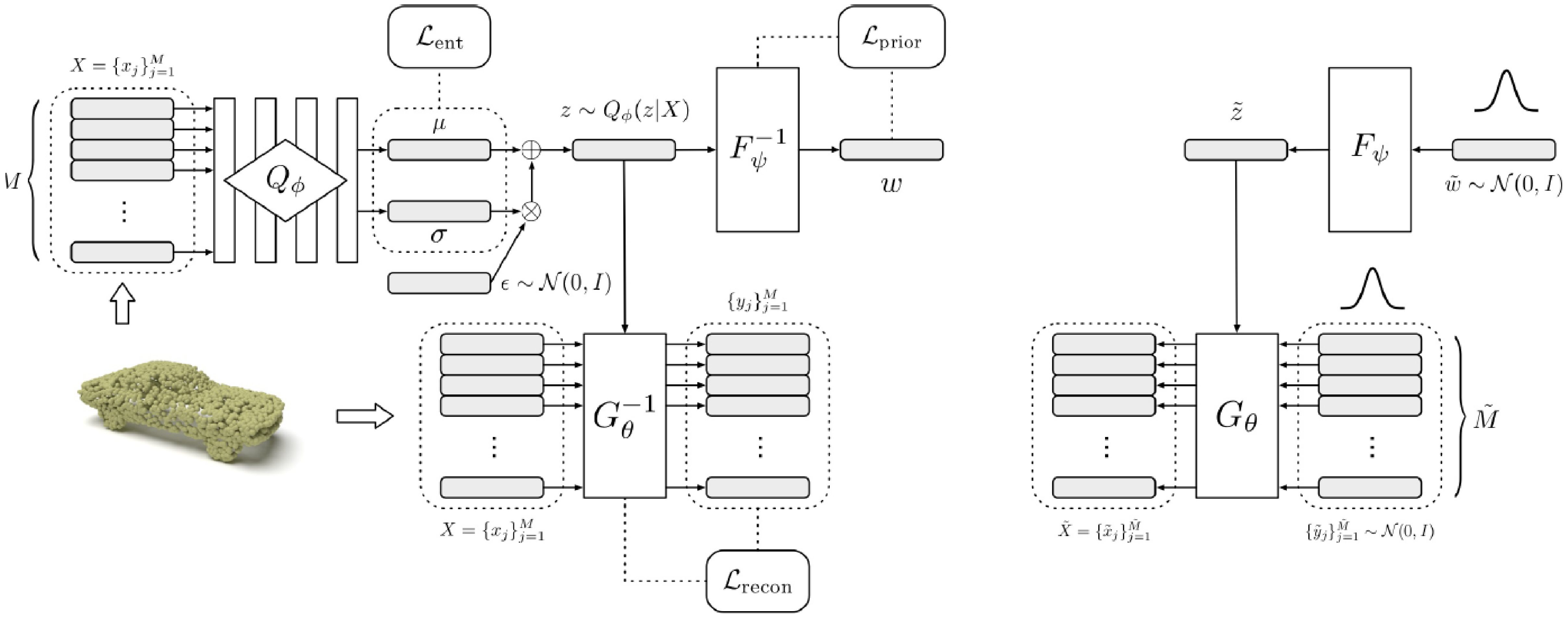


The Entire PointFlow Architecture

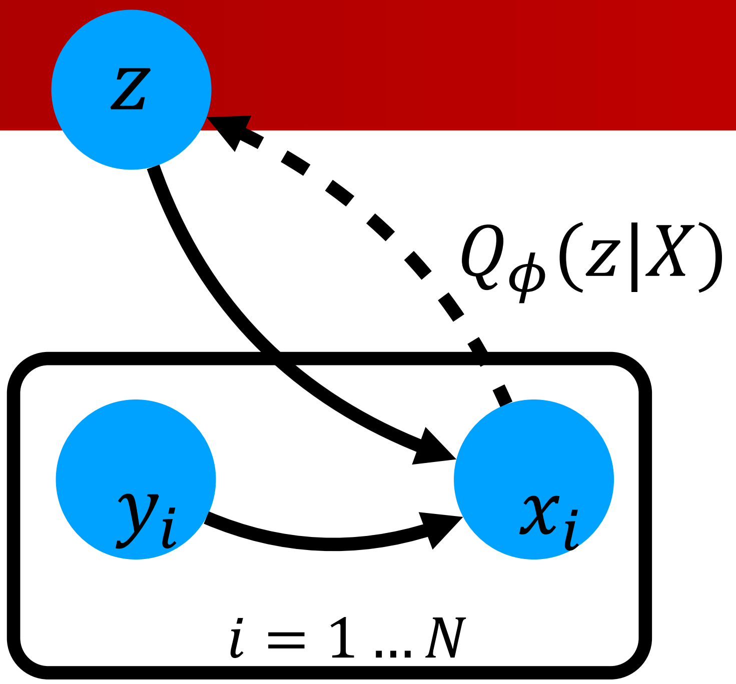


Everything is invertible

VAE Architecture



VAE Framework

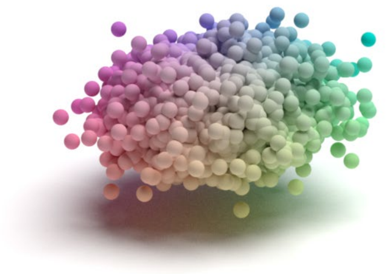


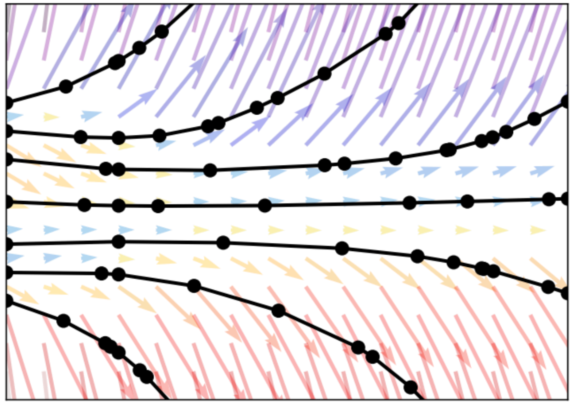
$$\log P_\theta(X) \geq \underbrace{\mathbb{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)] - D_{KL}(Q_\phi(z|X)||P_\psi(z))}_{\text{ELBO}}$$

$$= \underbrace{\mathbb{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)]}_{\text{Reconstruction Loss}} - \underbrace{\mathbb{E}_{Q_\phi(z|X)}[\log Q_\phi(z|X)]}_{\text{Regularization Loss}} + \underbrace{\mathbb{E}_{Q_\phi(z|X)}[\log P_\psi(z)]}_{\text{Prior Loss}}$$

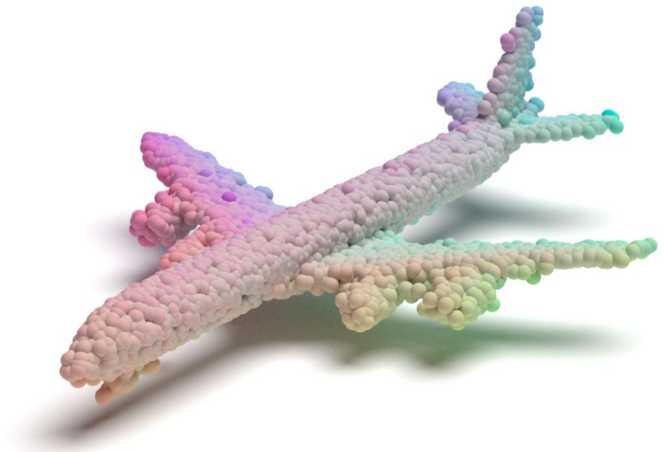
VAE Framework - Reconstruction Loss

$$\underbrace{\mathbb{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)]}_{\textbf{Reconstruction Loss}} - \mathbb{E}_{Q_\phi(z|X)}[\log Q_\phi(z|X)] + \mathbb{E}_{Q_\phi(z|X)}[\log P_\psi(z)]$$



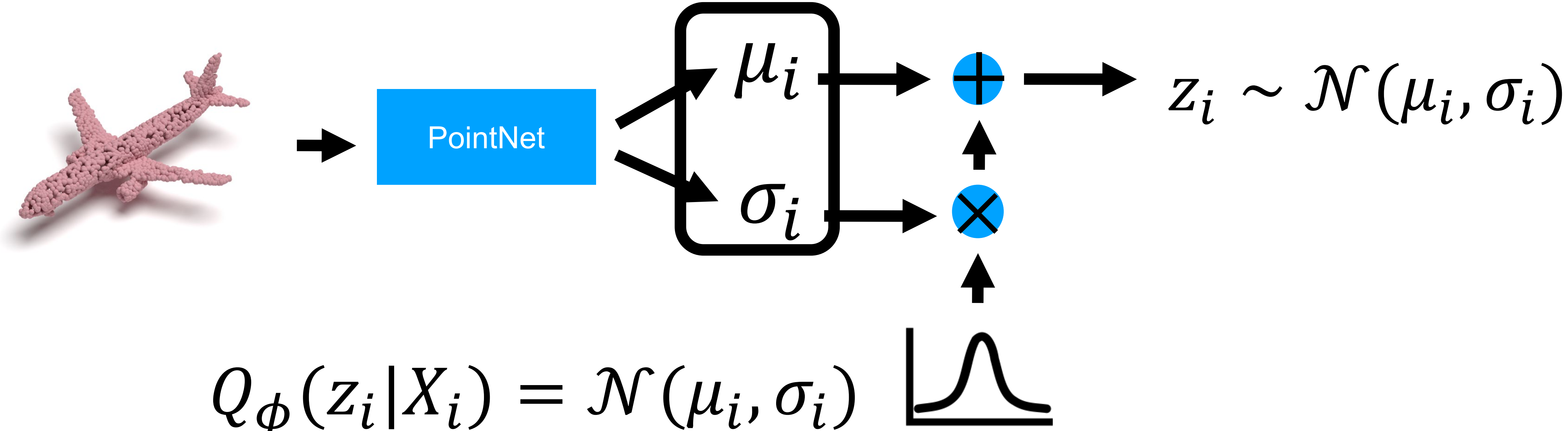


Point CNF



VAE Framework - Regularization Loss

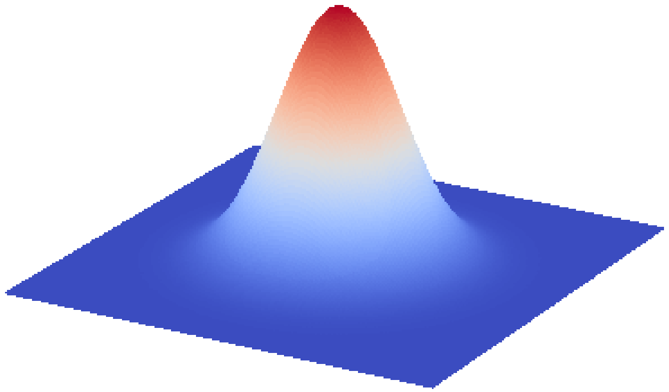
$$\mathbb{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)] - \underbrace{\mathbb{E}_{Q_\phi(z|X)}[\log Q_\phi(z|X)]}_{\textbf{Regularization Loss}} + \mathbb{E}_{Q_\phi(z|X)}[\log P_\psi(z)]$$

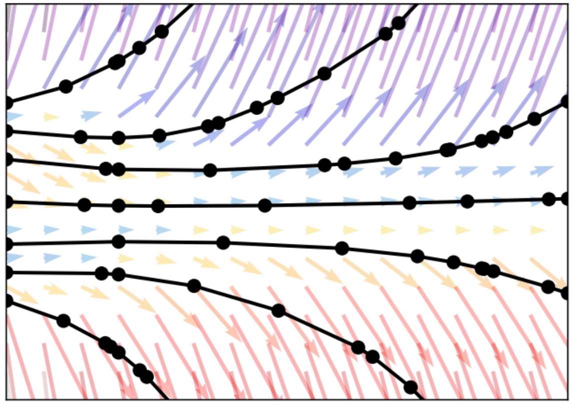


$$Q_\phi(z_i|X_i) = \mathcal{N}(\mu_i, \sigma_i)$$

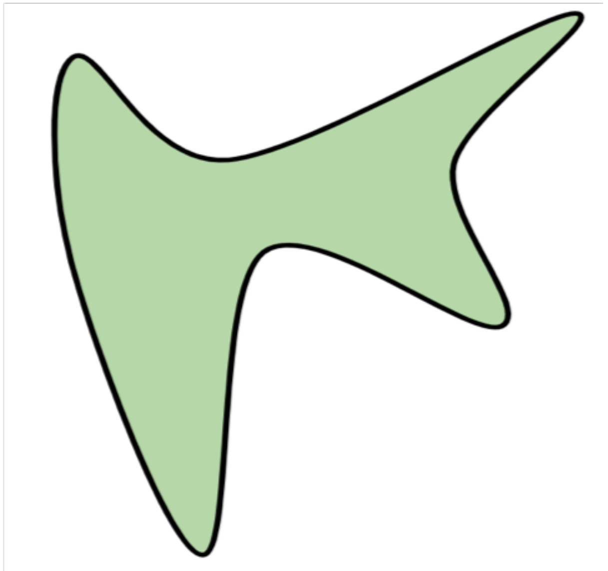
VAE Framework - Prior Loss

$$\mathbb{E}_{Q_\phi(z|X)}[\log P_\theta(X|z)] - \mathbb{E}_{Q_\phi(z|X)}[\log Q_\phi(z|X)] + \underbrace{\mathbb{E}_{Q_\phi(z|X)}[\log P_\psi(z)]}_{\text{Prior Loss}}$$

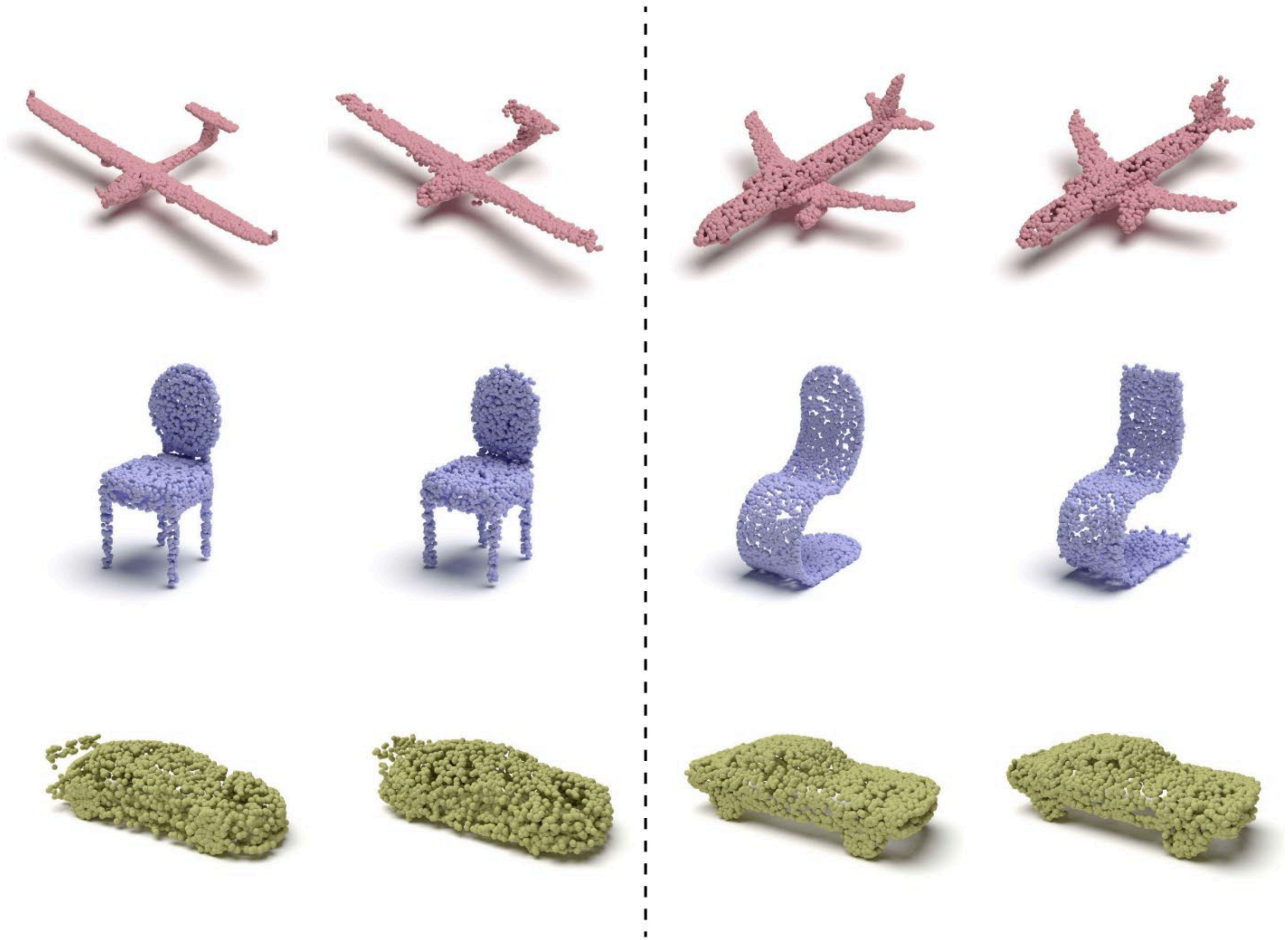




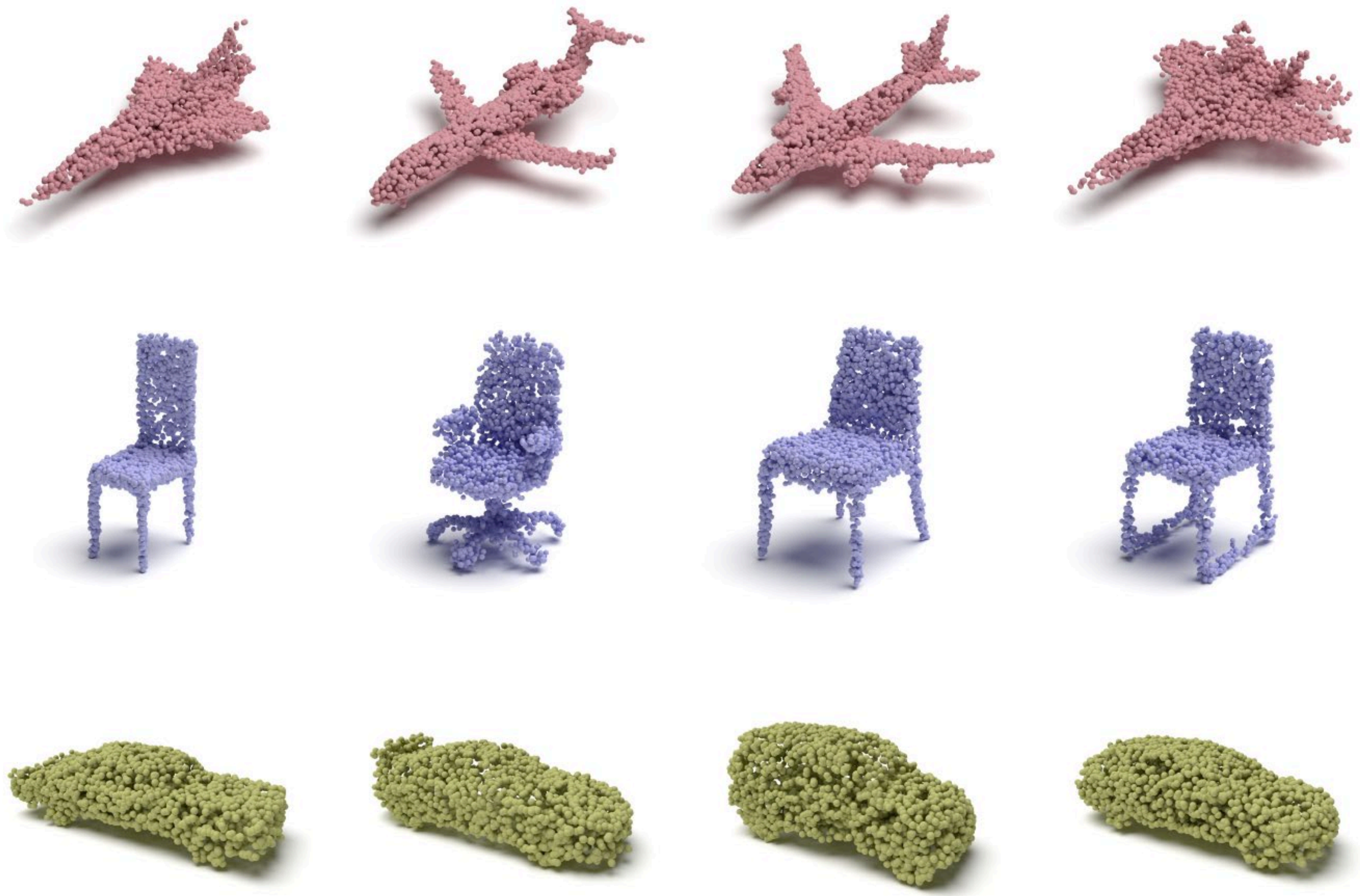
Latent CNF



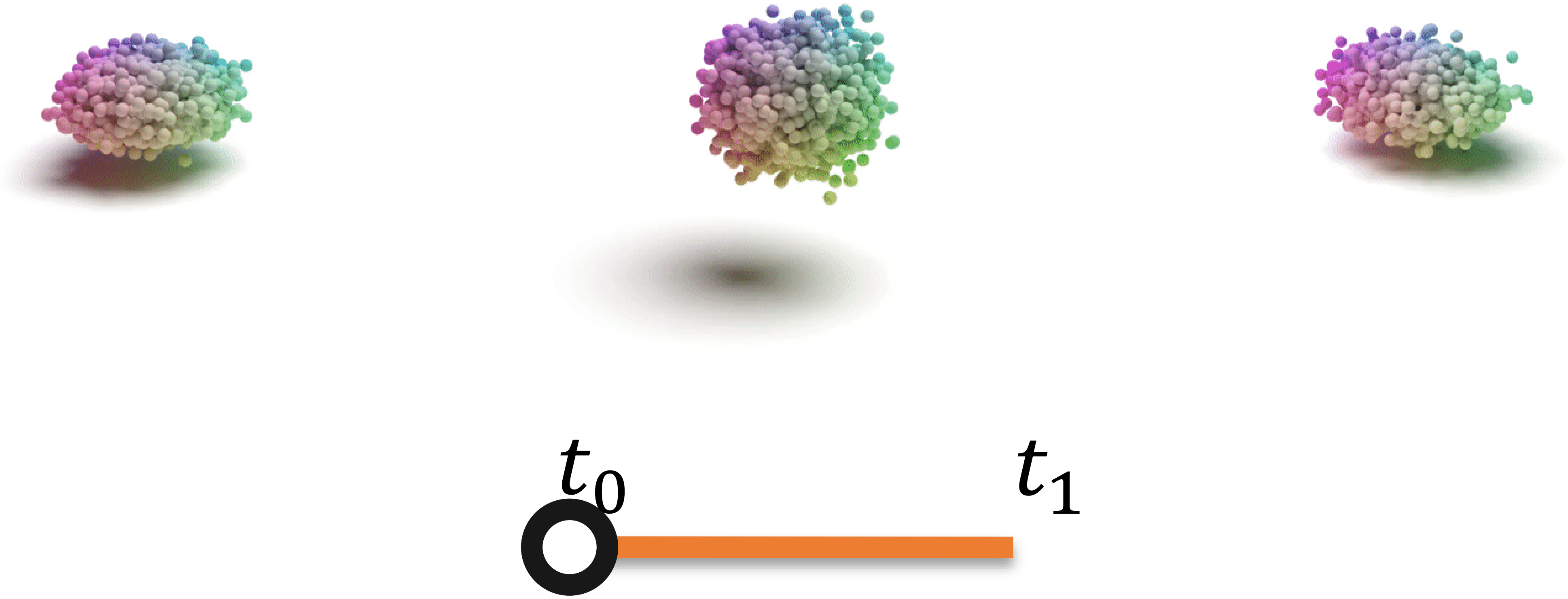
Auto-encoding Results



Generation Results

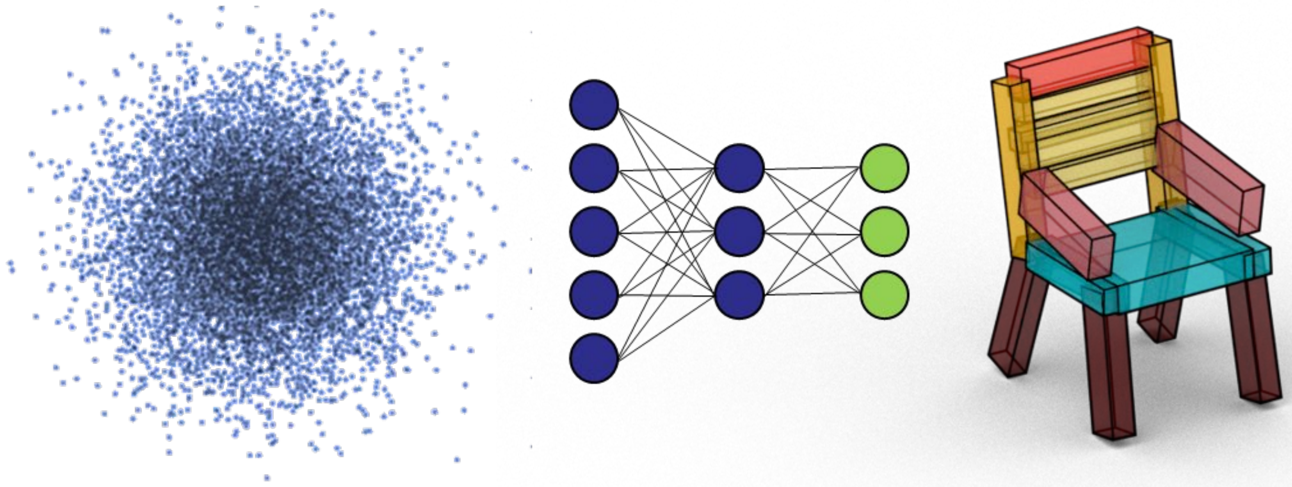


Visualization of Point Transformations



Structured Shape Representations

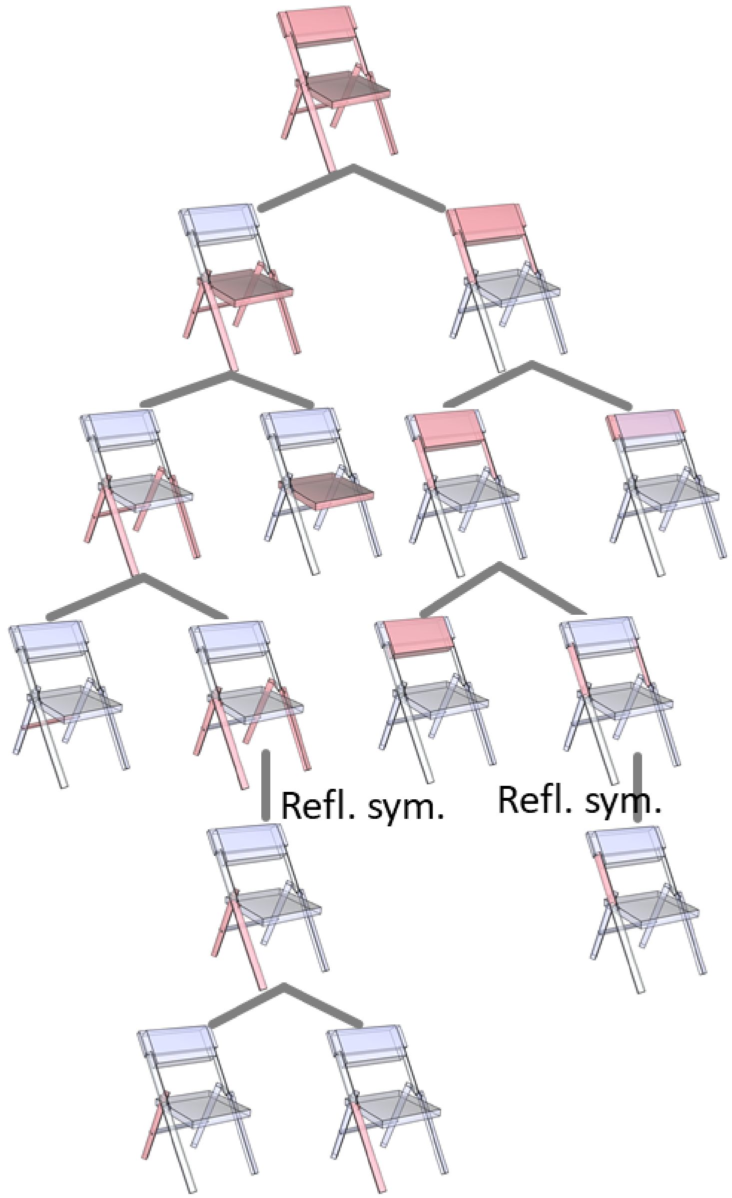
Structured Shape Representations



Example of
Neural Shape Generator:
GRASS

GRASS: Generative Recursive Autoencoders for Shape Structures. Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, Leonidas Guibas. Siggraph 2017.

Shapes Naturally Have Compositional Structure

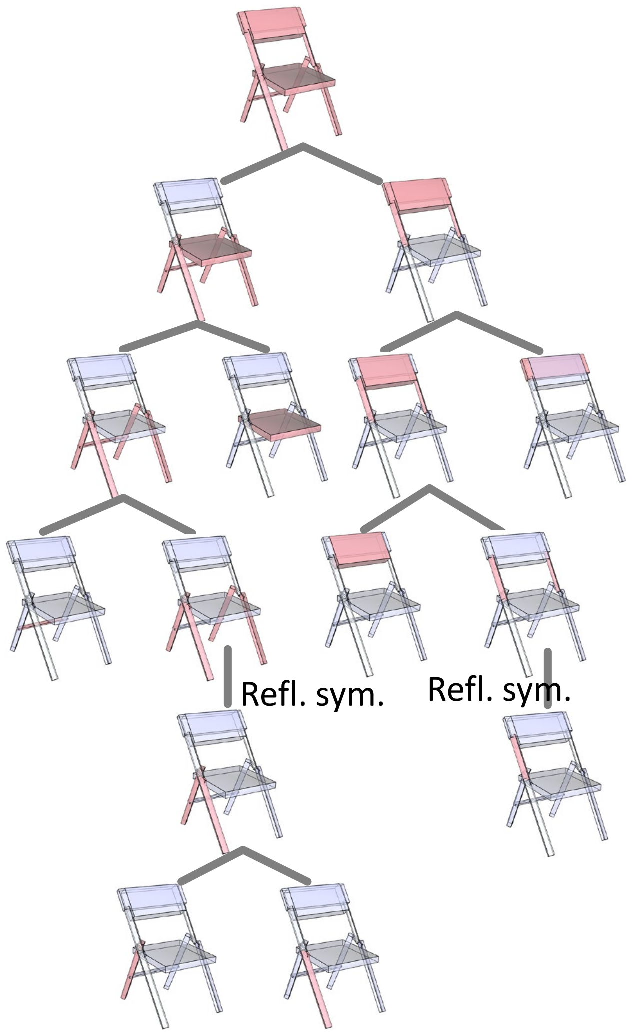


These reflect

- part-subpart hierarchies
- groupings based on type
- groupings based on adjacency
- groupings based on symmetry

GRASS: Generative Network Over Unlabeled Part Layouts

- GRASS factorizes a shape into a hierarchical layout of simplified parts, plus fine-grained part geometries
- Weakly supervised:
  - ✓ segments or parts
  - ✗ labels
  - ✗ manually curated “ground truth” hierarchies
- Structure-aware: learns a generative distribution over informative shape structures

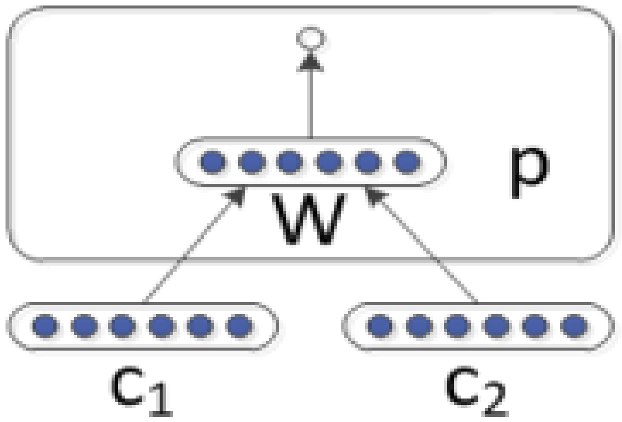


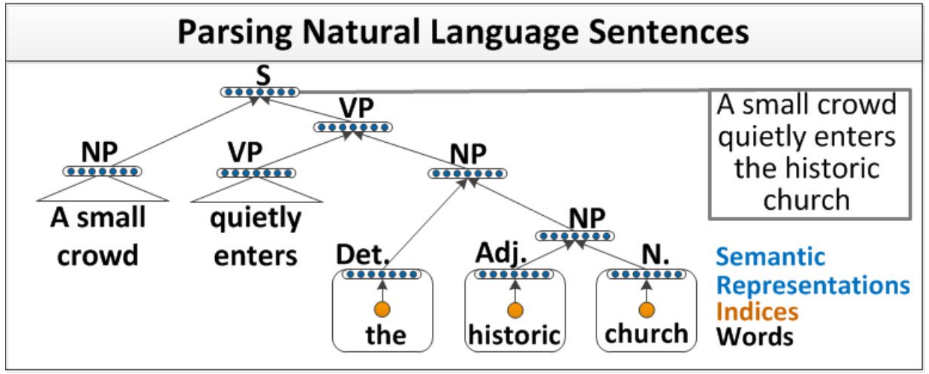
Three Challenges

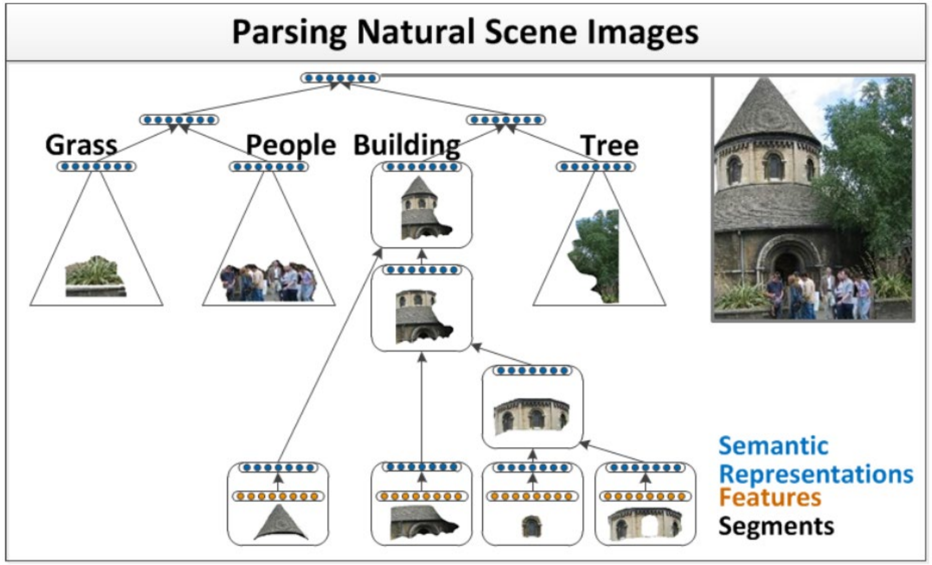
- **Challenge 1:** Ingest and generate arbitrary complexity part layouts with a fixed-dimensional network
- **Challenge 2:** Map a layout invertibly to a fixed-D code ("Shape DNA") that implicitly captures adjacency, symmetry and hierarchy
- **Challenge 3:** Capture both structure and fine geometry details

Recursive Neural Network (RvNN)

- Repeatedly merge two nodes into one
- Each node has an n-D feature vector, computed recursively
- $p = f(W[c_1;c_2] + b)$ f a non-linearity

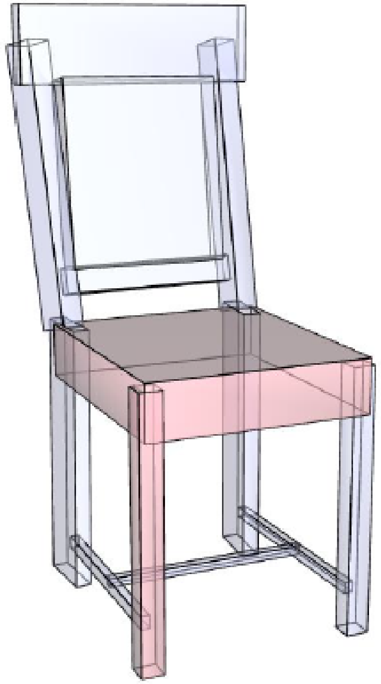






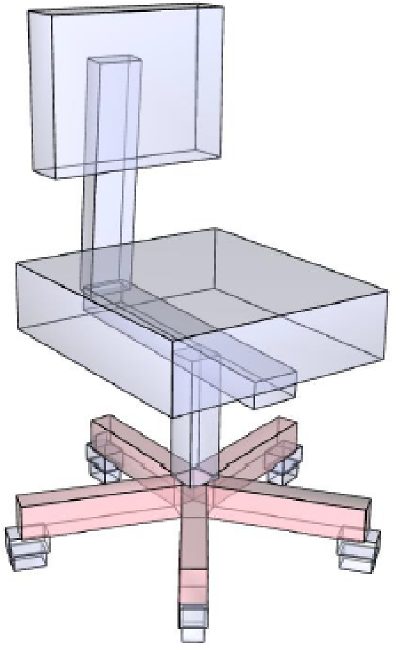
Socher et al. 2011

Different Types of Merges, Varying Cardinalities

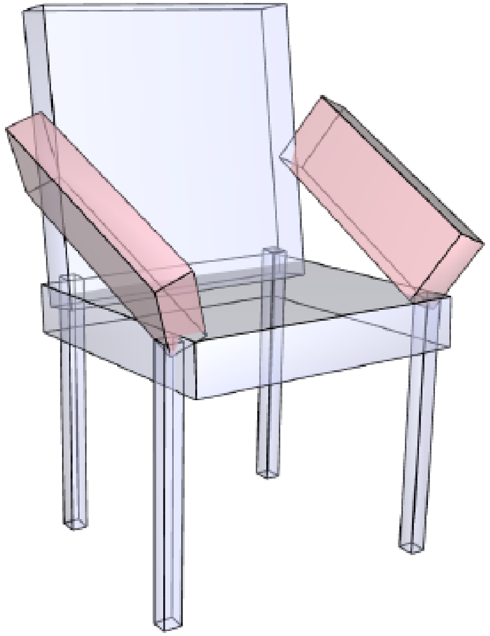


Adjacency

Translational symmetry



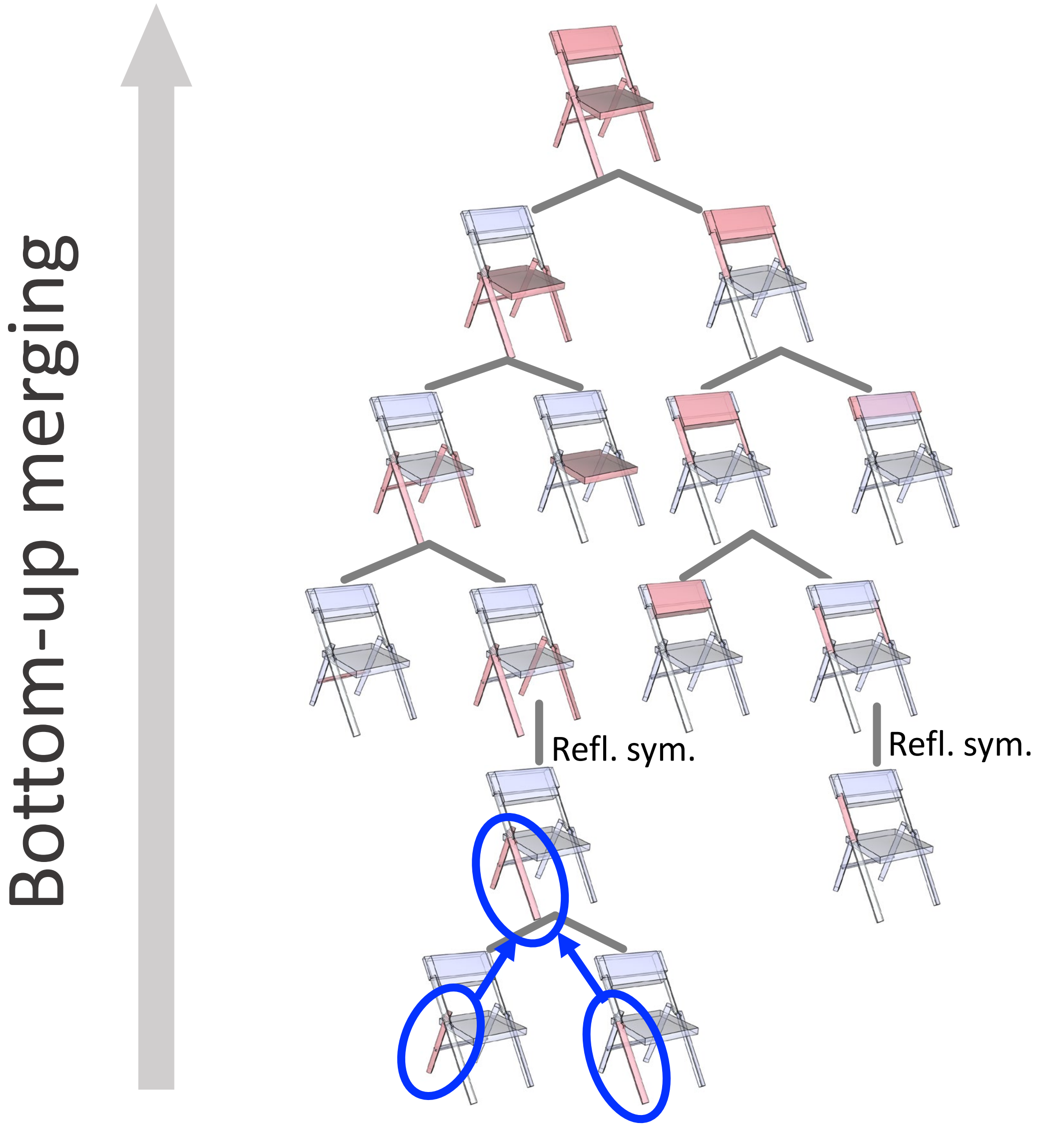
Rotational symmetry

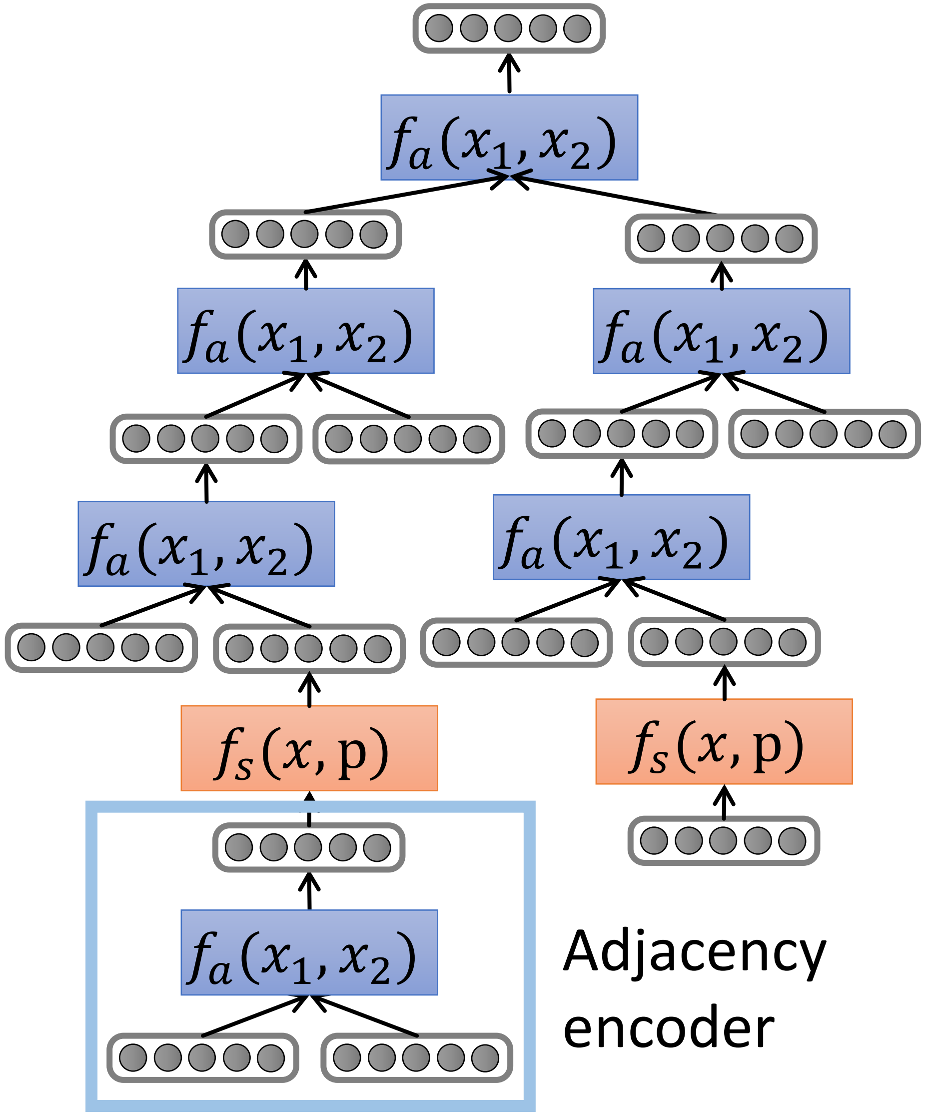


Reflectional symmetry

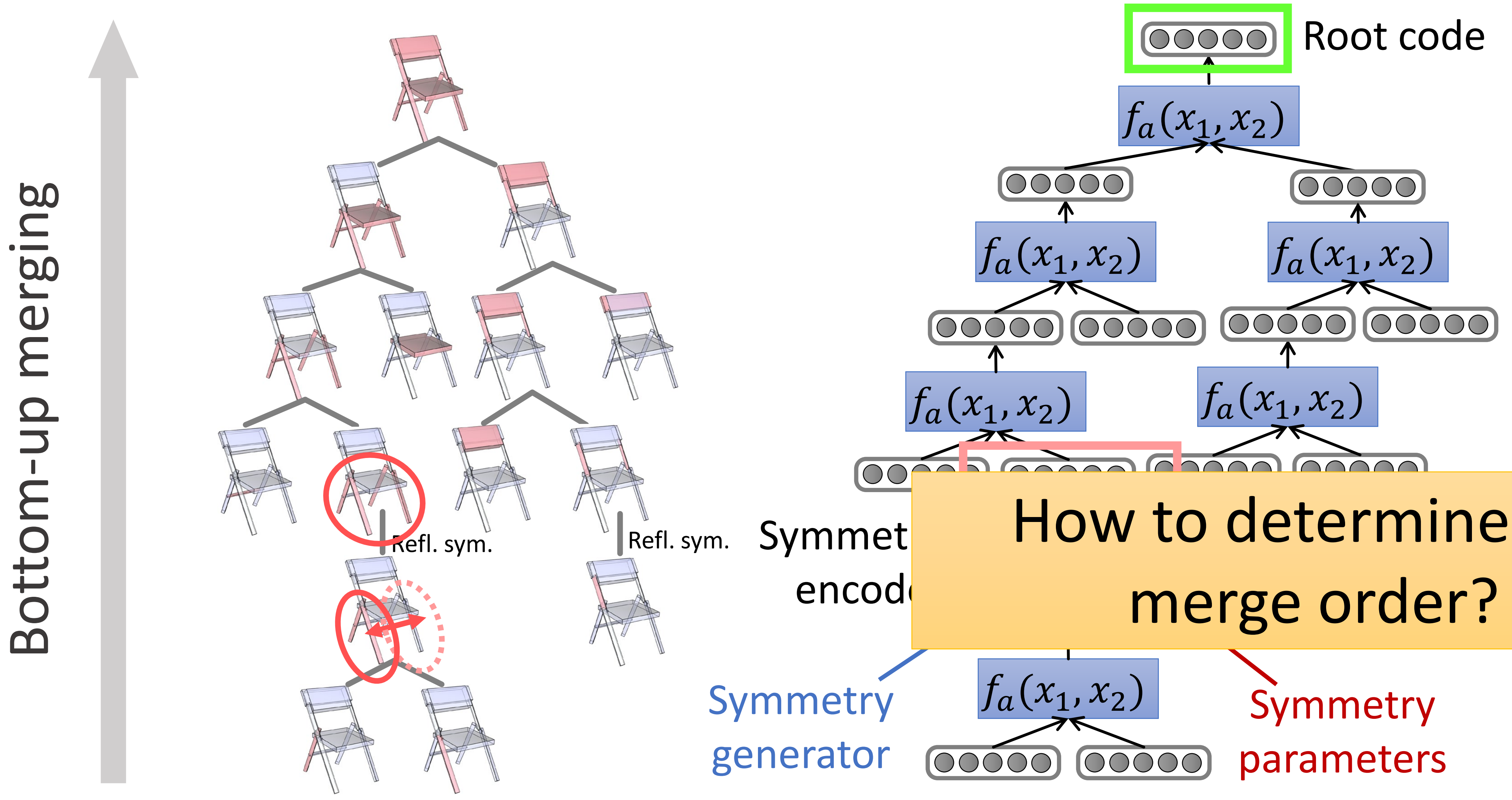
- How to encode them to the same code space?
- How to decode them appropriately, given just a code?

Recursively Merging Parts





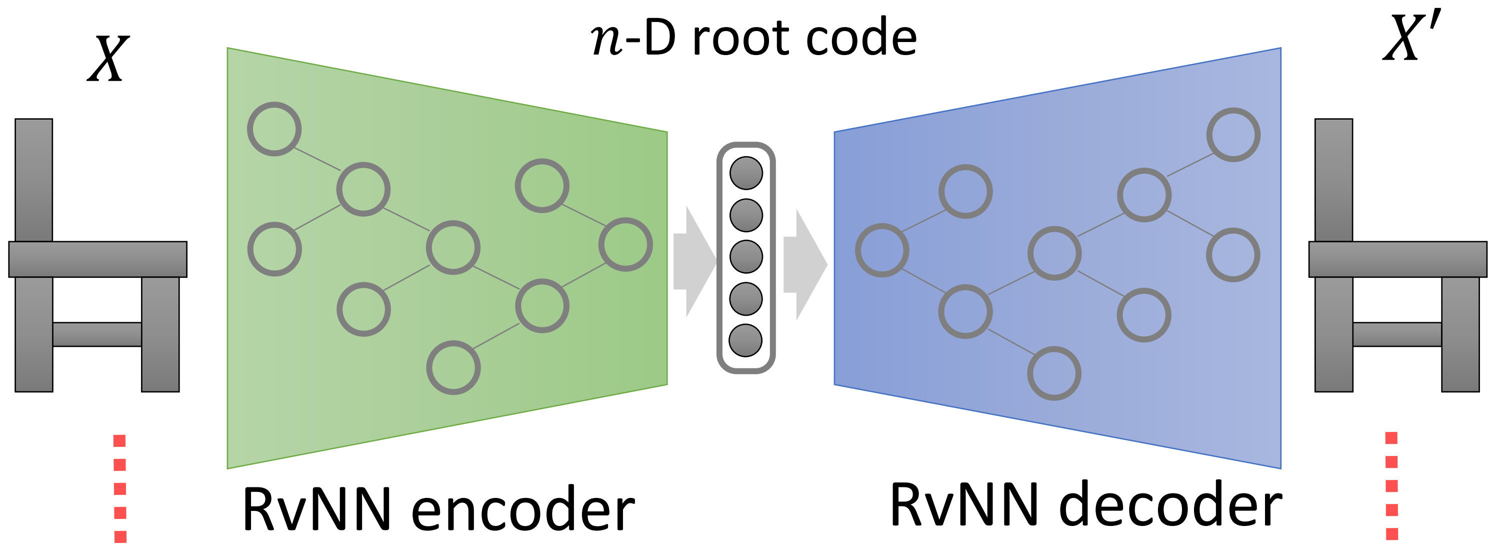
Recursively Merging Parts



How to determine the merge order?

Training with Reconstruction loss

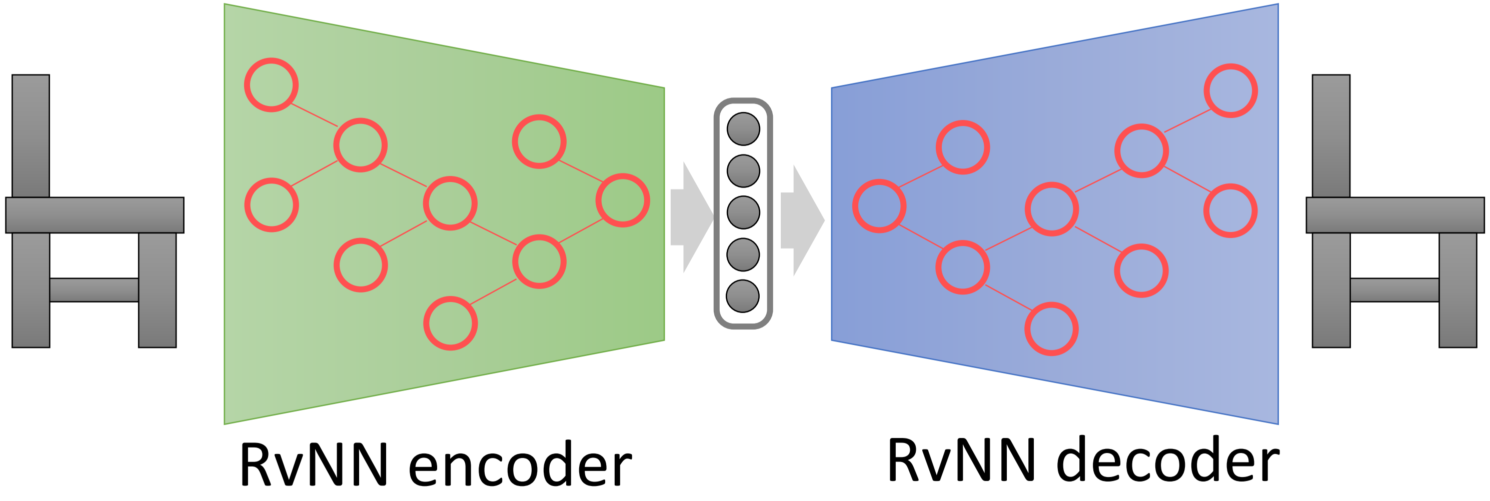
- Learn weights from a variety of randomly sampled merge orders for each box structure



$$L = \|X - X'\|_2$$

In Training

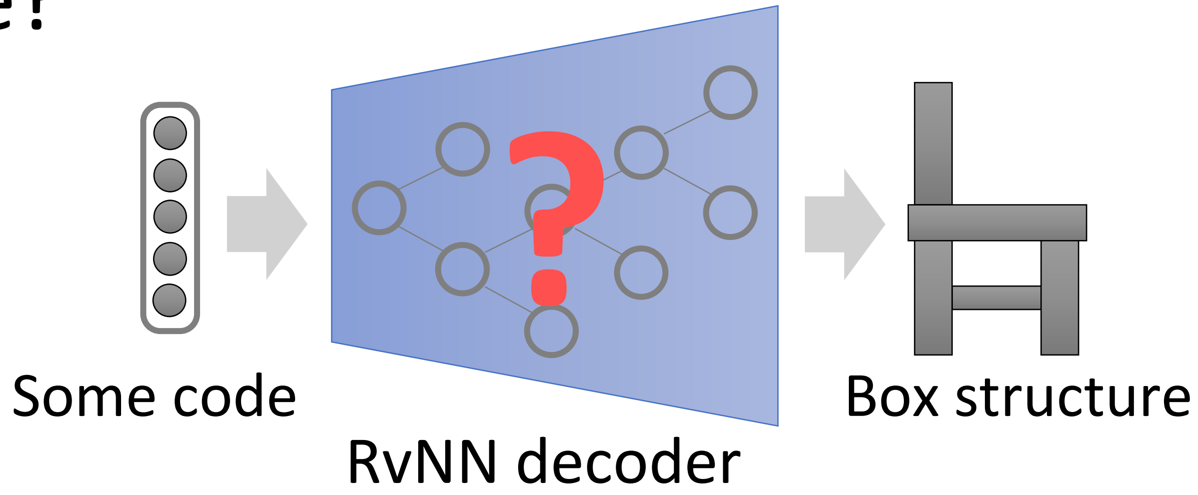
- Encoding: Given a box structure, determine the merge order as:
  - The hierarchy that gives the lowest reconstruction error



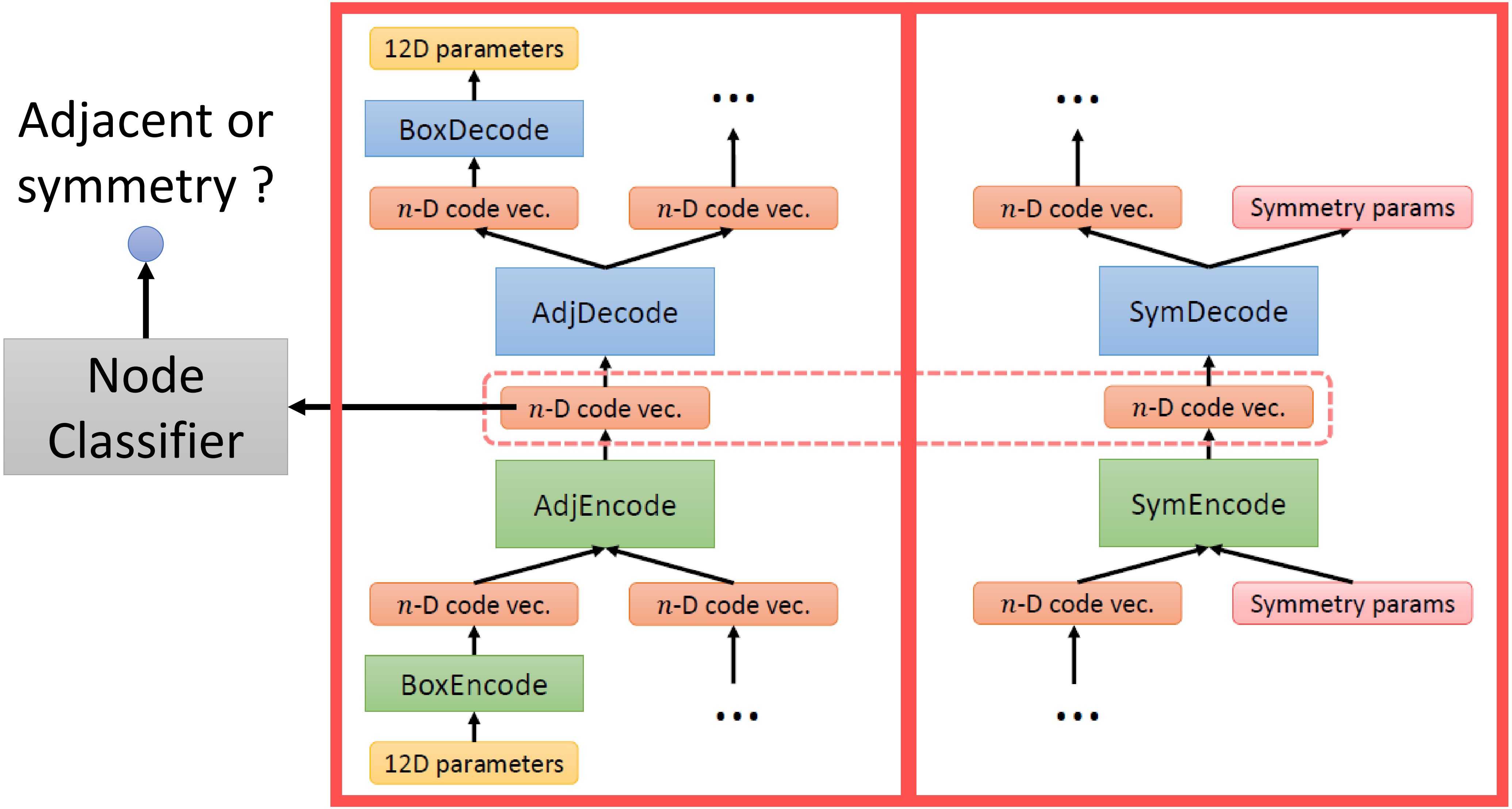
In Testing

- Encoding: Given a box structure, determine the merge order as:
  - The hierarchy that gives the lowest reconstruction error

- Decoding: Given an arbitrary code, how to generate the corresponding structure?



How to Know what Type of Encoder to Use?



Making the Network Generative

- Variational Auto-Encoder (VAE): Learn a distribution that approximates the data distribution of true 3D structures

$$P(X) \approx P_{gt}(X)$$

- Marginalize over a latent "DNA" code

$$\text{maximize } P(X) = \int \underbrace{P(X|z;\theta)}_{\text{Likelihood}} P(z)dz$$

Parameters

Variational Bayes Formulation ELBO

maximize $P(X) = \int P(X|z;\theta)P(z)dz$

maximize $\boxed{E_{z\sim Q}\left[\log P(X|z)\right]} - \boxed{\mathcal{D}\left[Q(z|X)\|P(z)\right]}$

z should reconstruct X, given that it was drawn from $Q(z|X)$

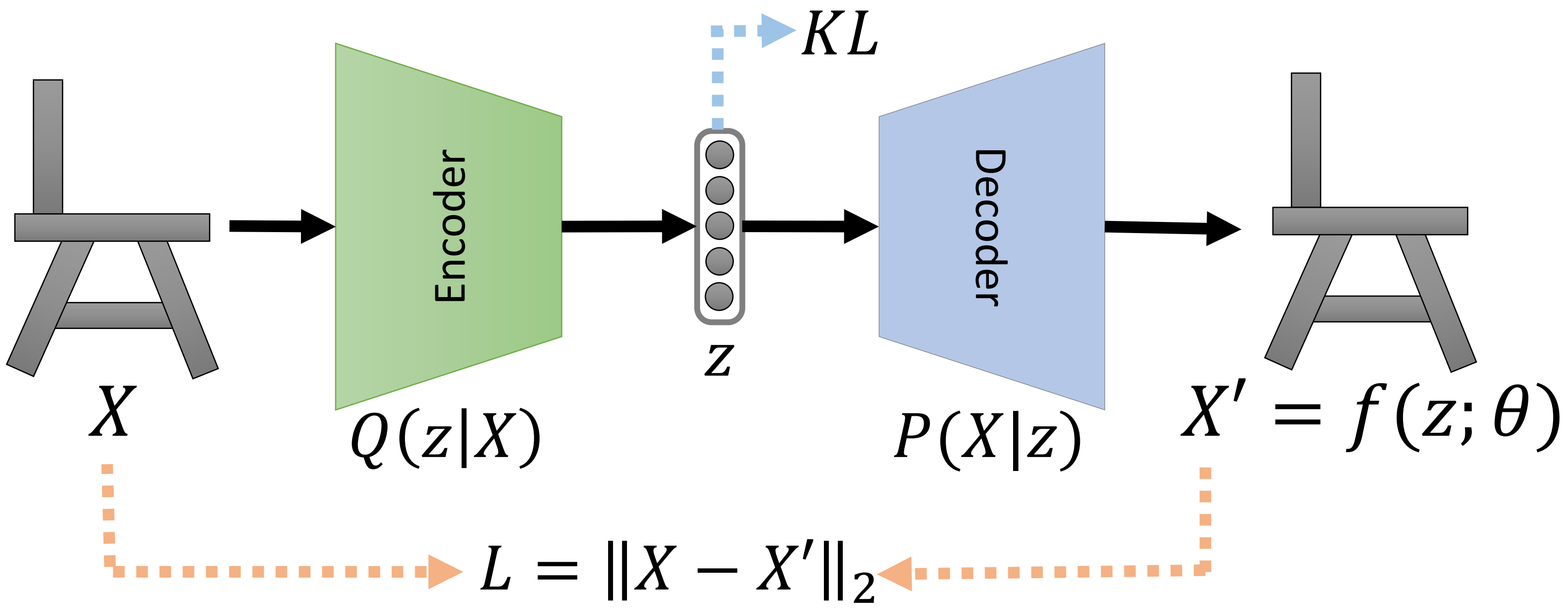
Assuming z's follow a normal distribution

Evidence Lower Bound

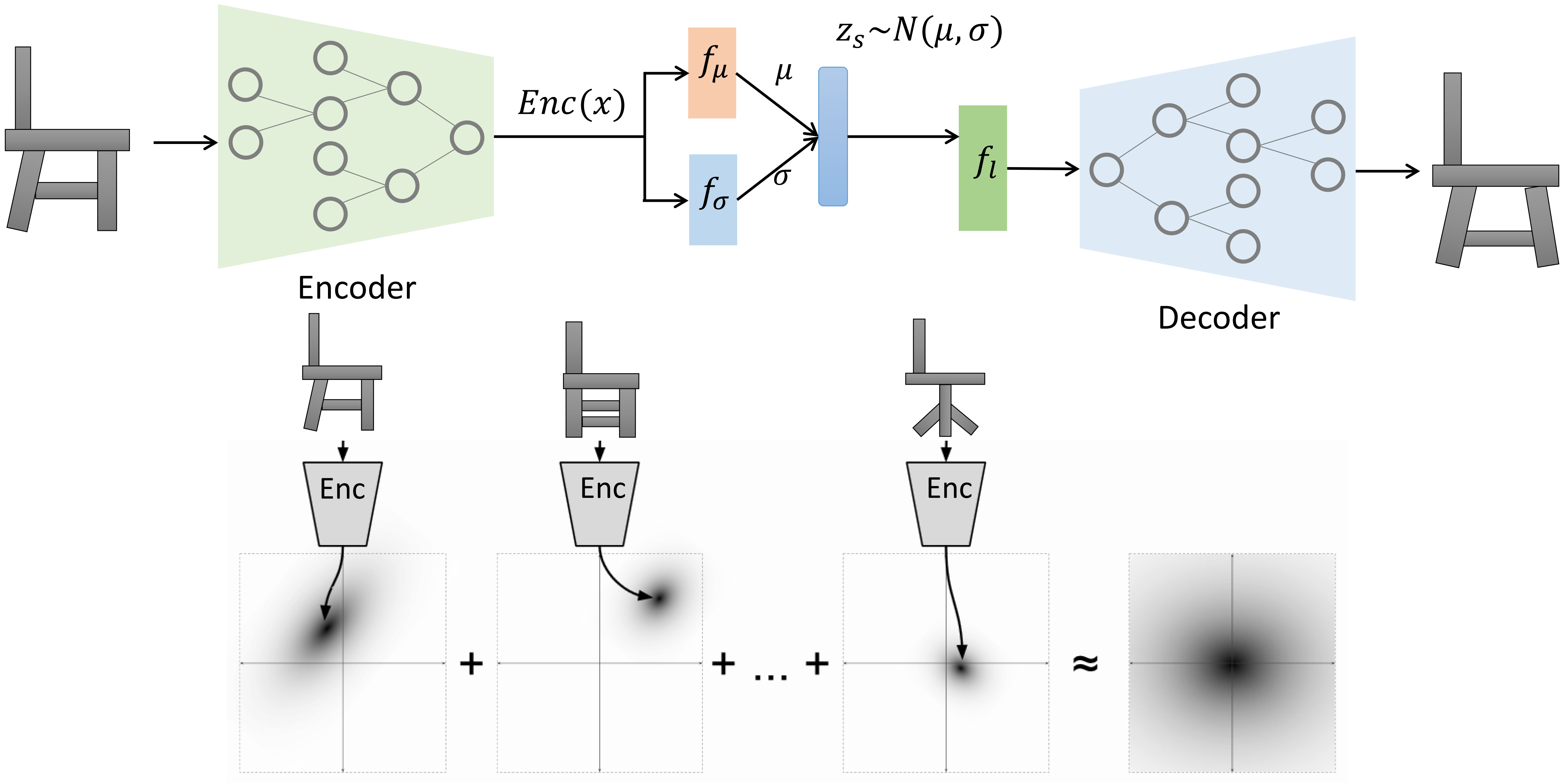
Variational Autoencoder (VAE)

maximize $\boxed{E_{z\sim Q}\left[\log P(X|z)\right]} - \boxed{\mathcal{D}\left[Q(z|X)\,\|\,P(z)\right]}$

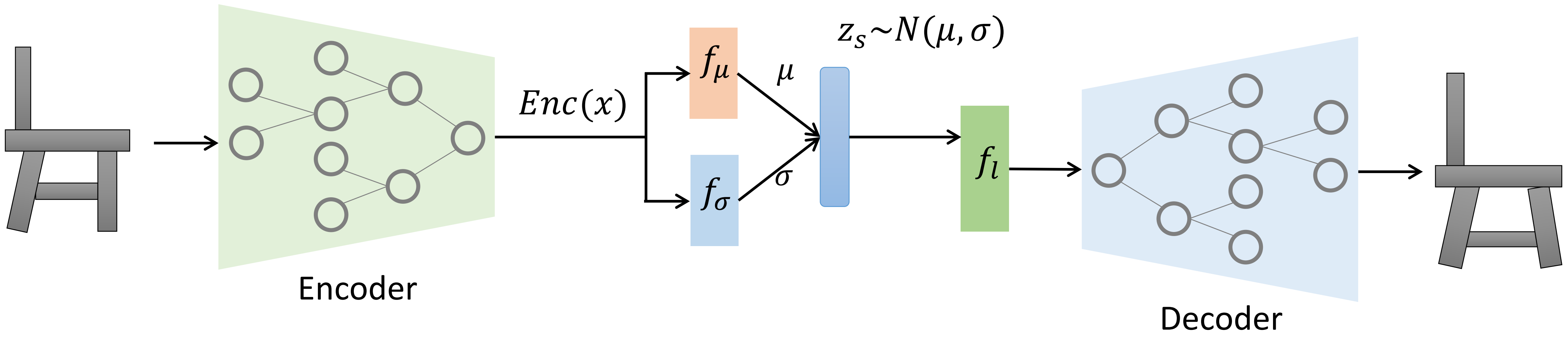
Reconstruction loss   KL divergence loss

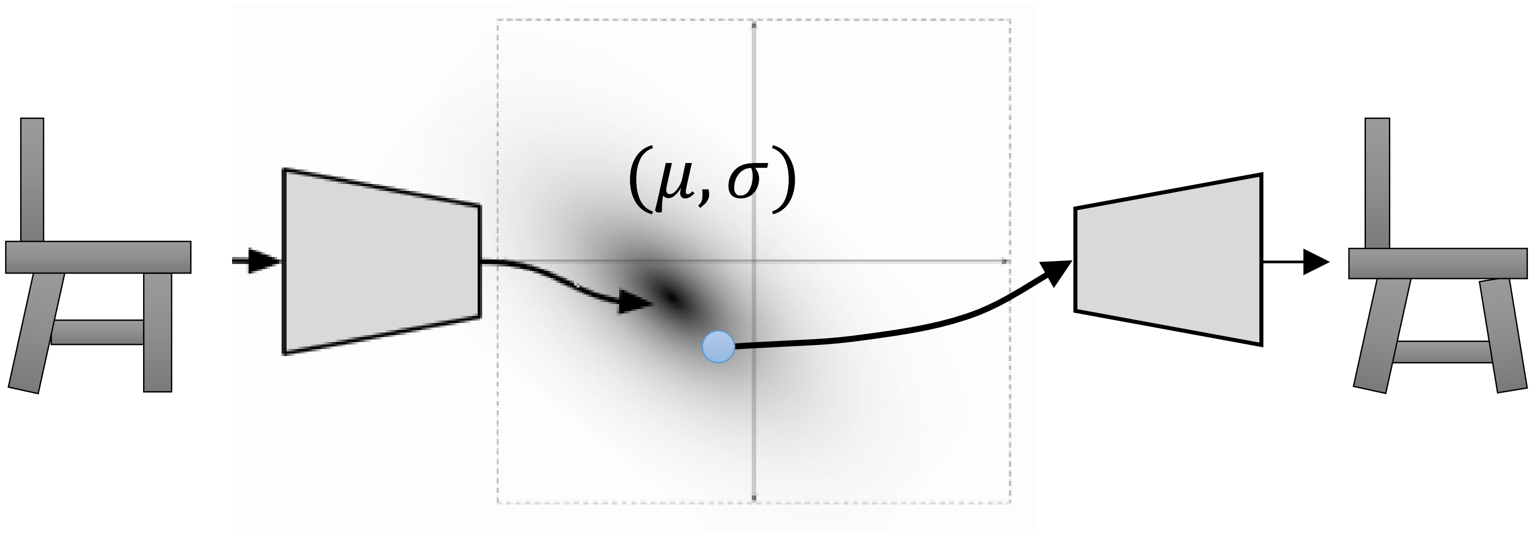


Variational Autoencoder (VAE)

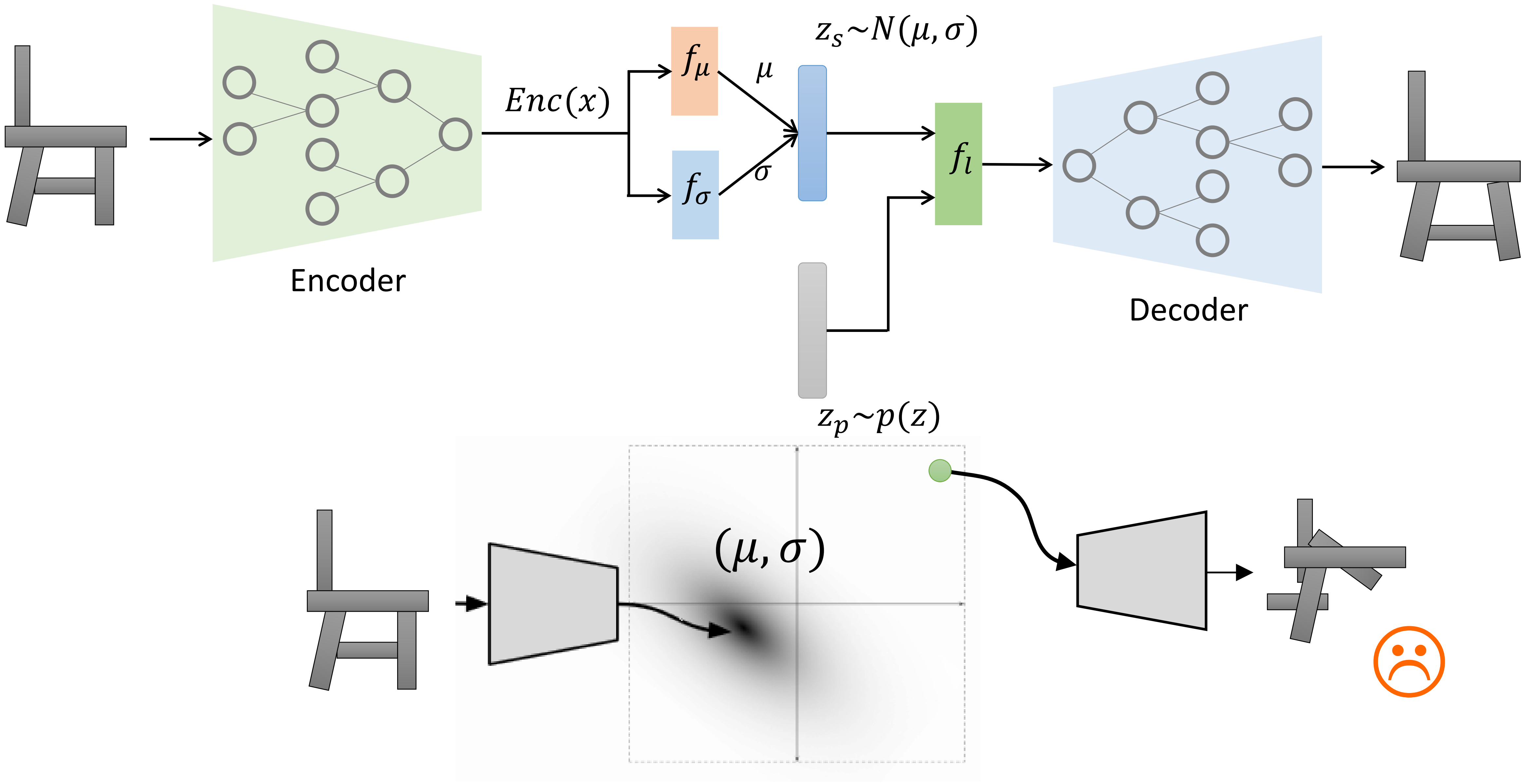


Sampling Near μ is Robust

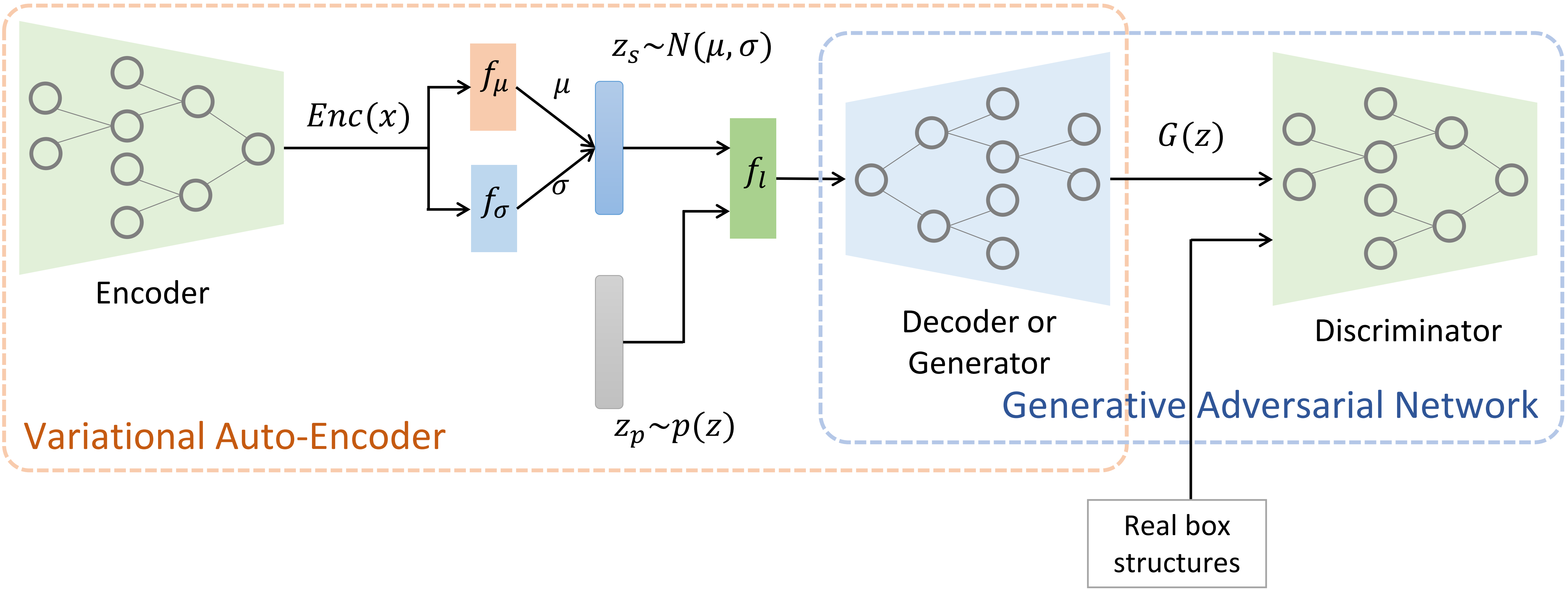




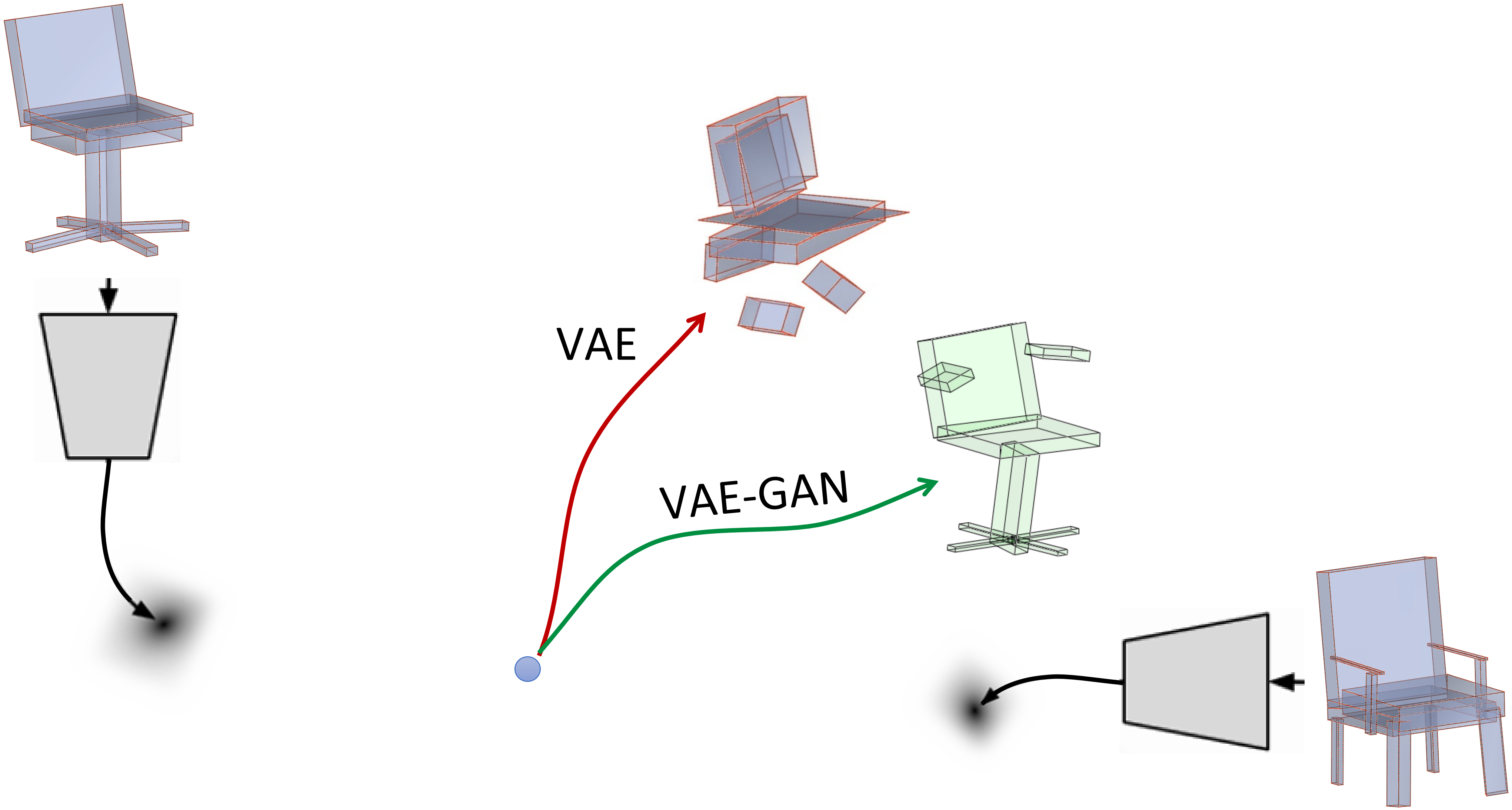
Sampling Far Away from μ?



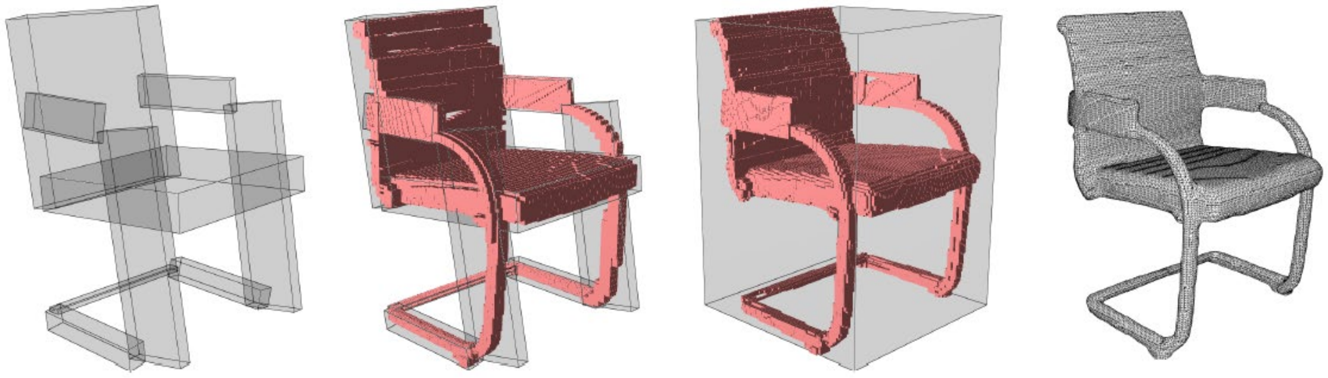
Adversarial Training: VAE-GAN

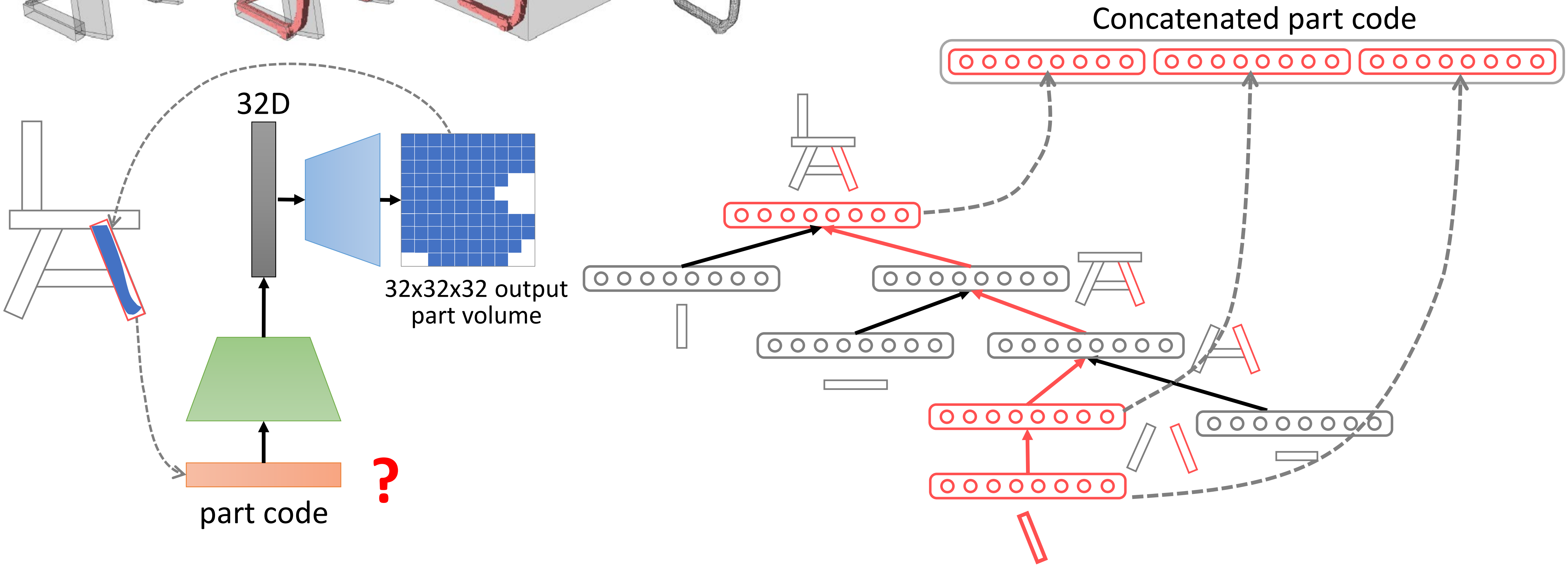


Benefit of Adversarial Training

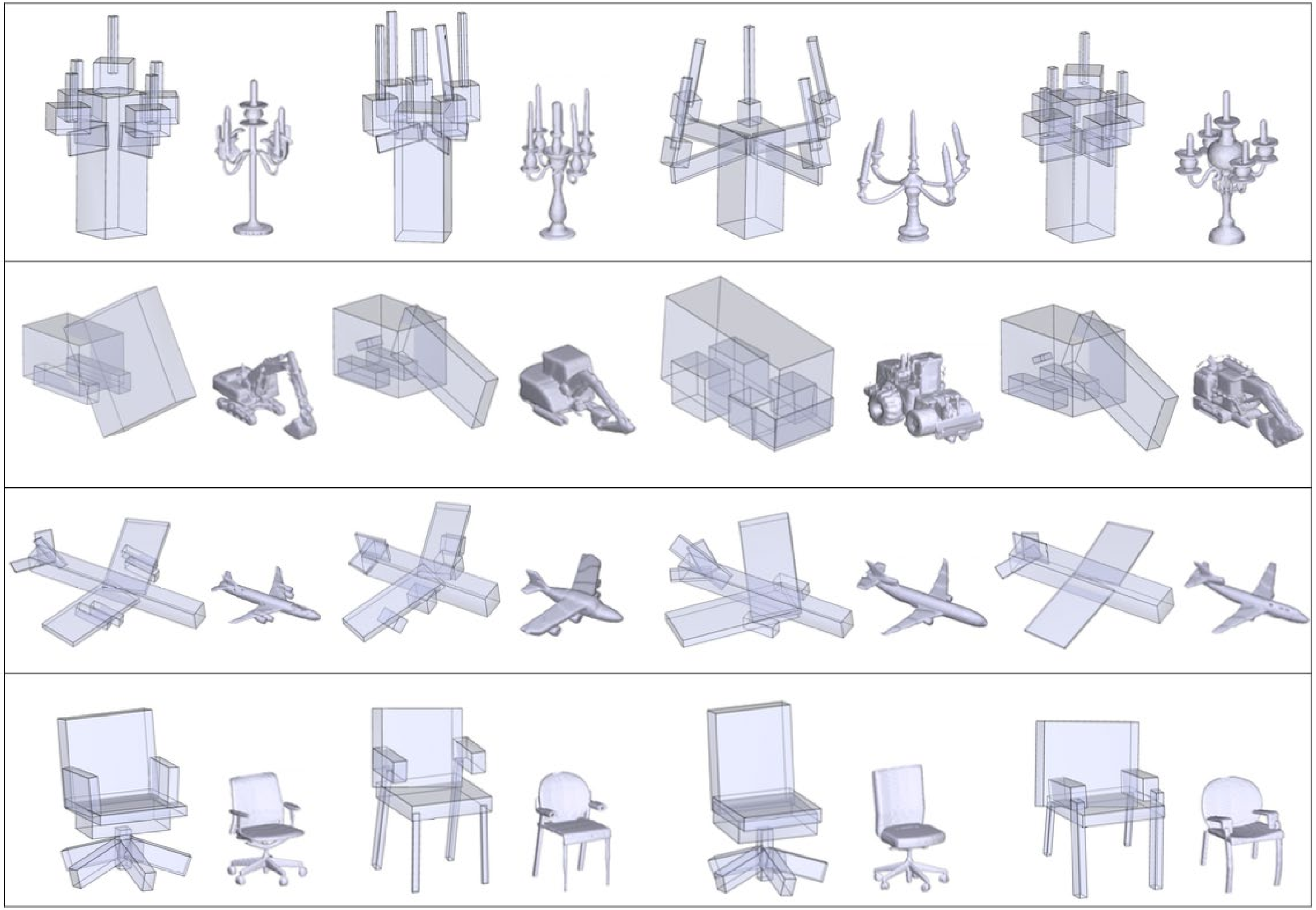


Voxelized Part Geometry Synthesis

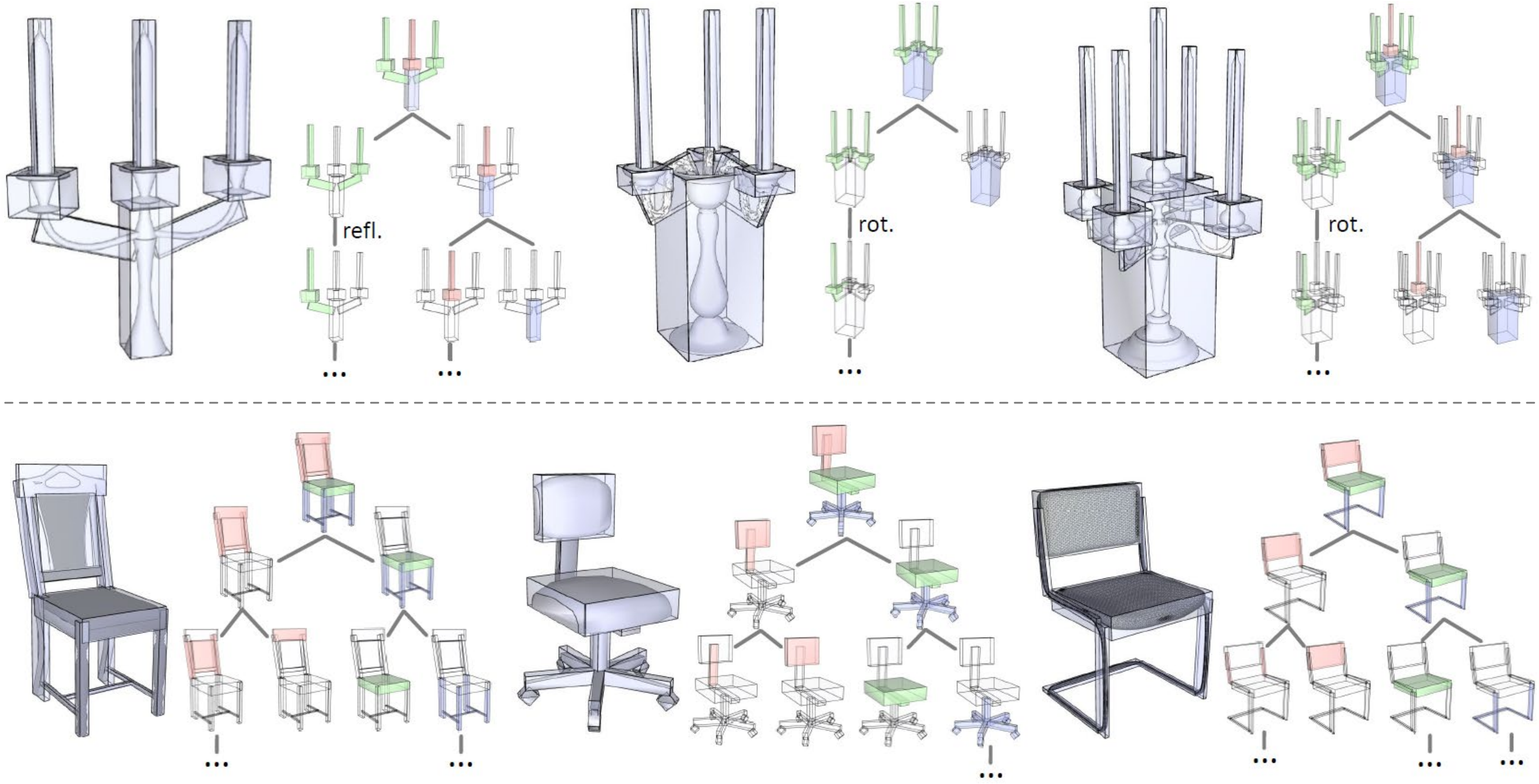




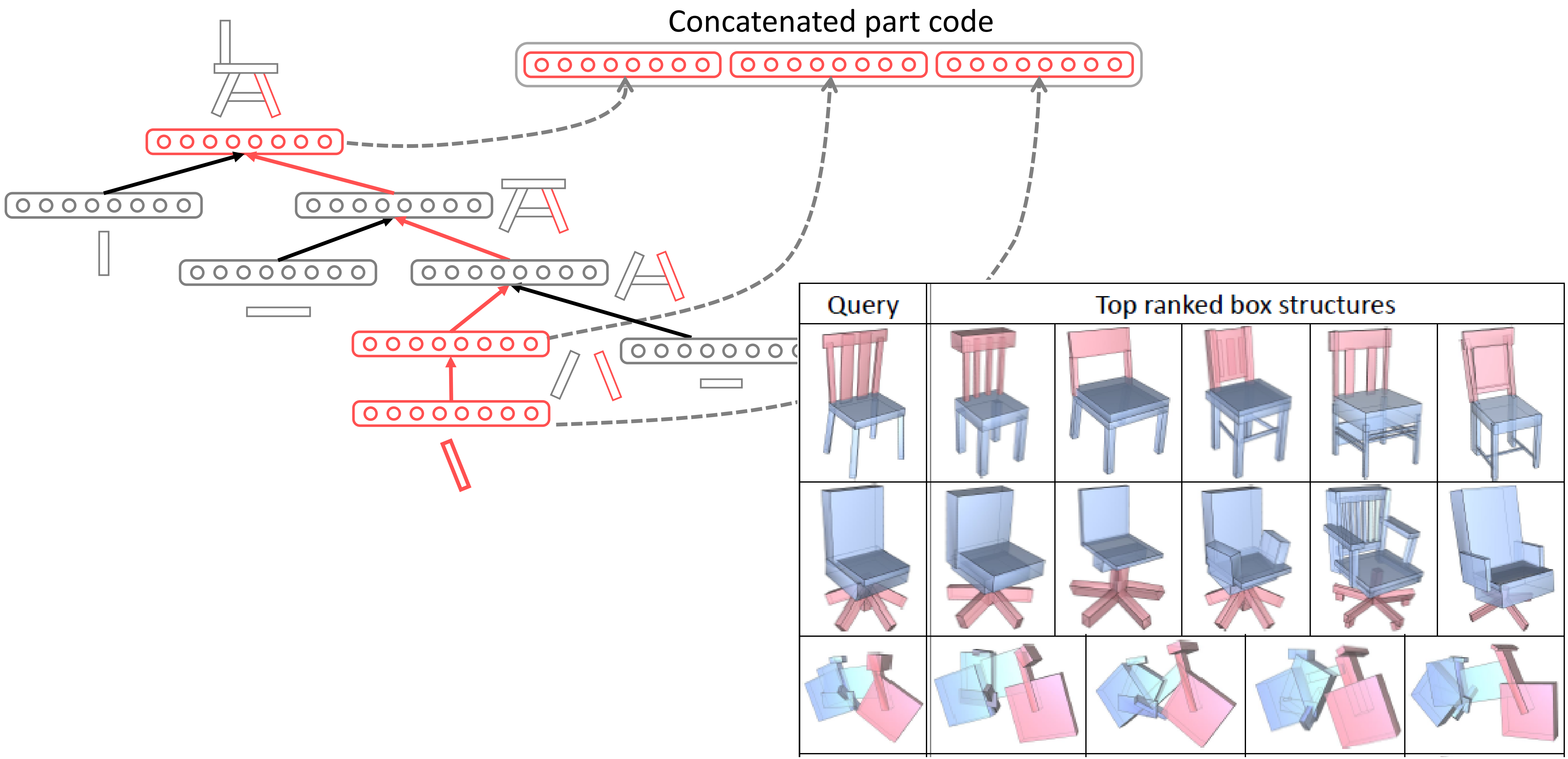
Results: Shape Synthesis



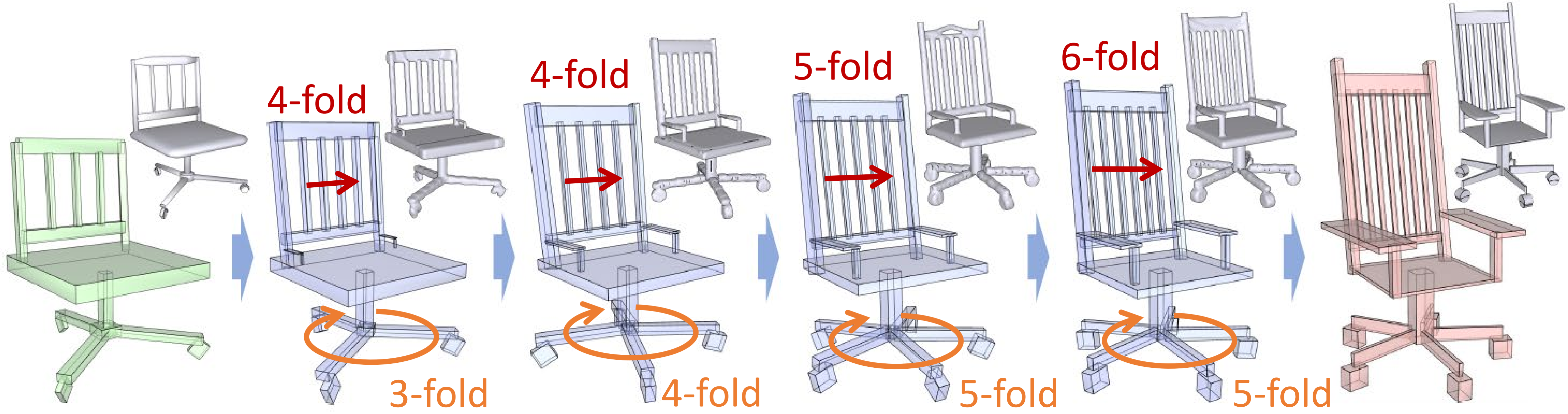
Results: Inferring Consistent Hierarchies



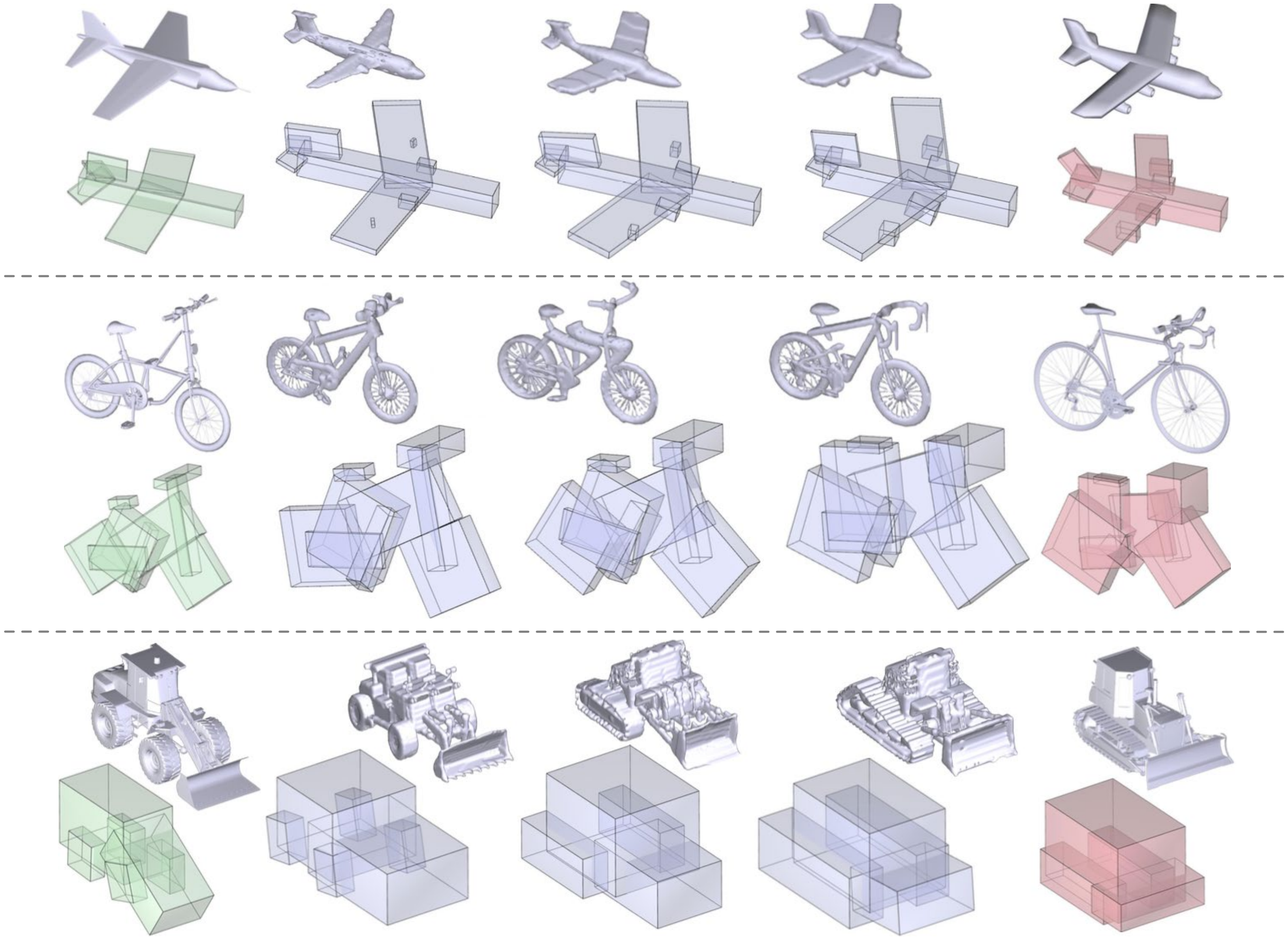
Results: Shape Retrieval



Results: Shape Interpolation



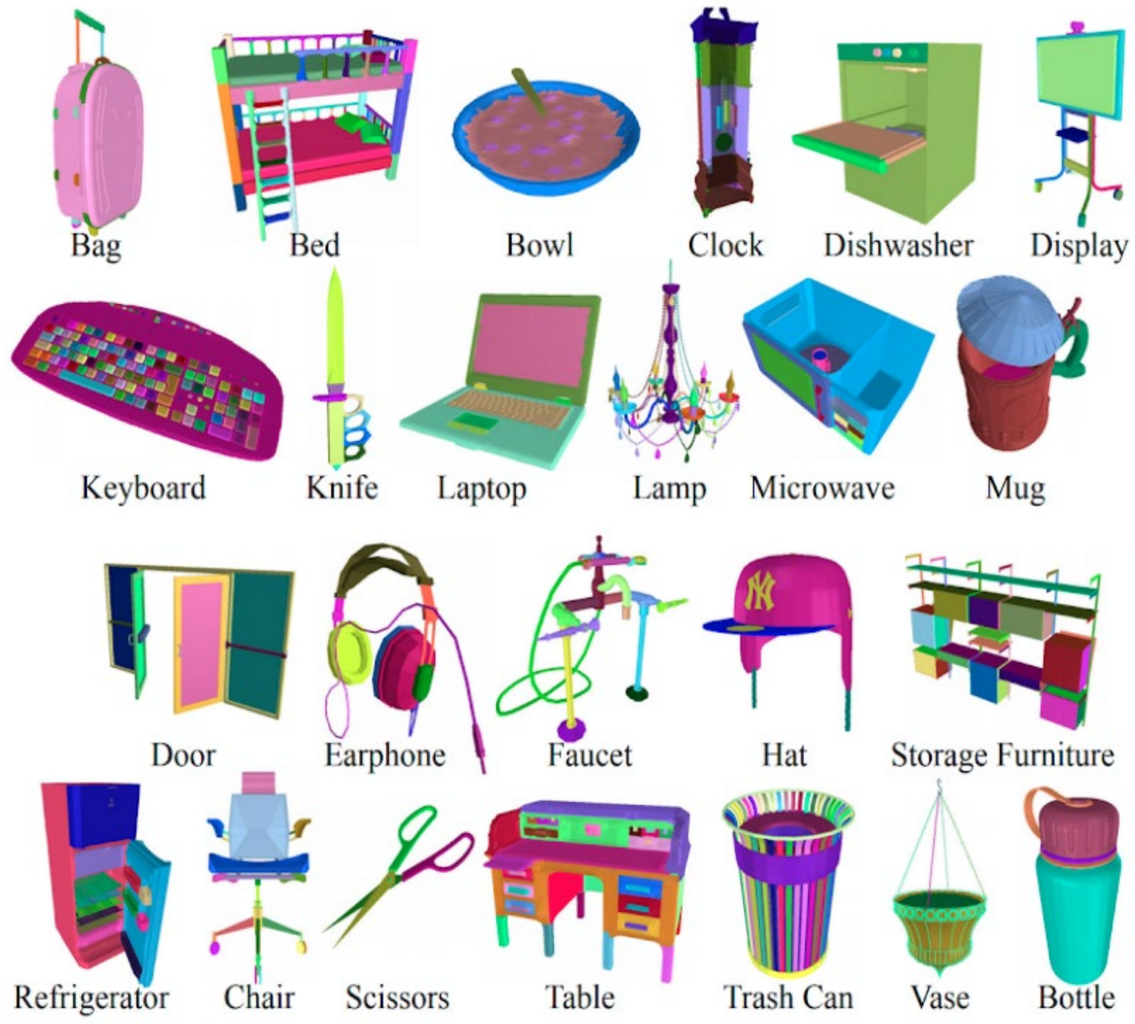
Results: Shape Interpolation



Exploting PartNet: StructureNet

StructureNet: Hierarchical Graph Networks for 3D Shape Generation. Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, Leonidas J. Guibas. Siggraph Asia 2019.

PartNet: Part Segmentation Annotation



https://partnet.cs.stanford.edu/

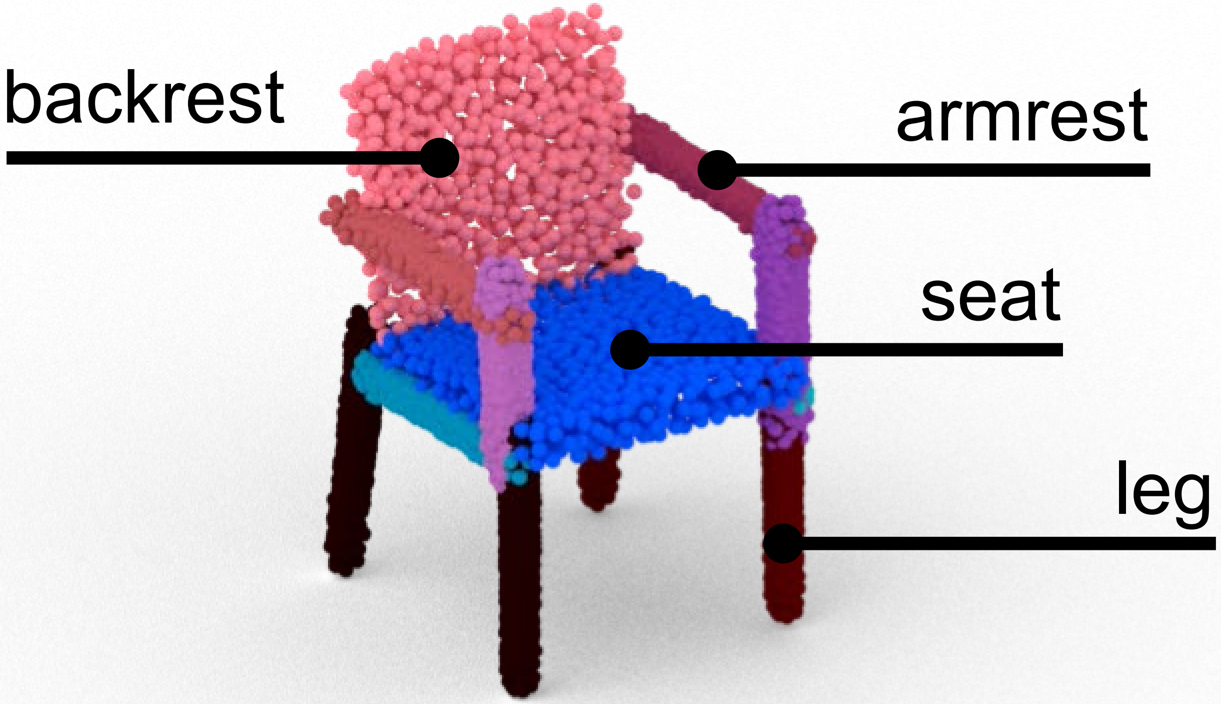
- Synthetic, 3D Shapes
- Based upon ShapeNet
- 573,585 Part Instances
- 26,671 Objects, 24 Categories

- Part Segmentations
  - Fine-grained
  - Hierarchical
  - Instance-level

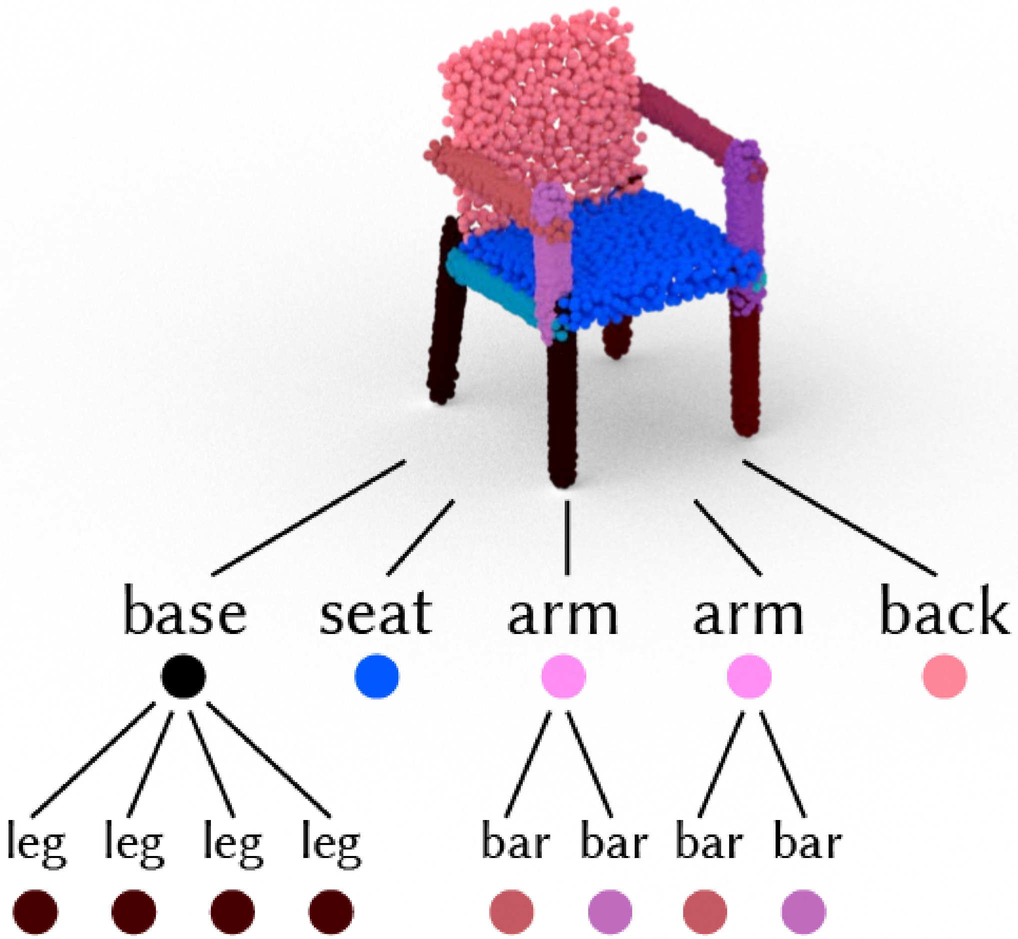
Mo et al., "**PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding**", CVPR 2019

Point Cloud Geometry and Structure

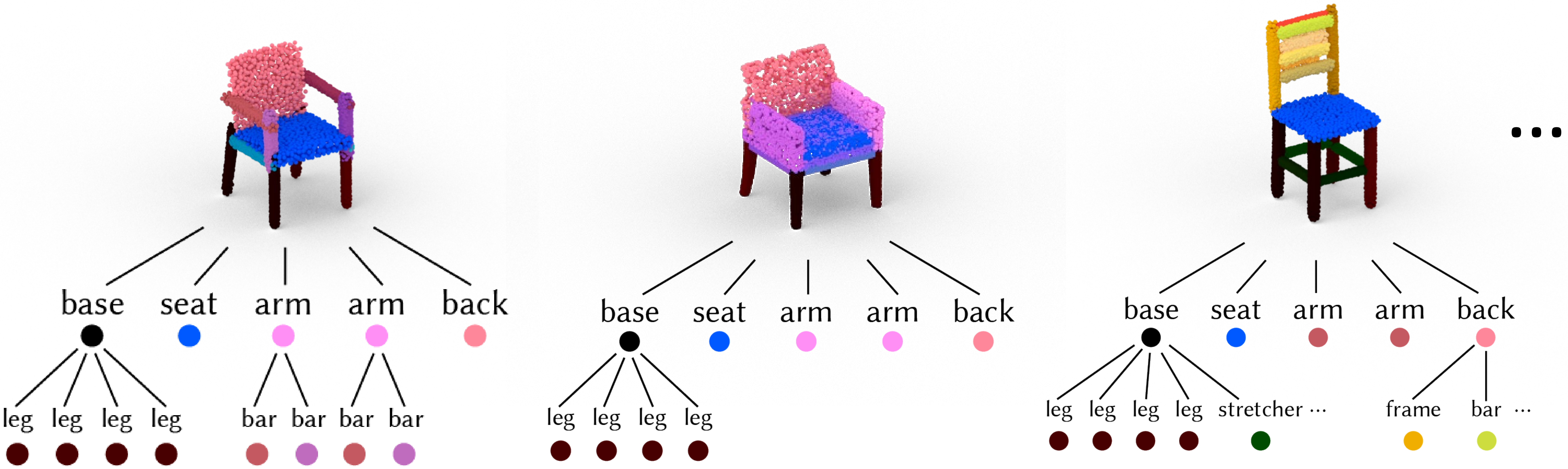
Geometry and Structure



Structure: Part Hierarchy

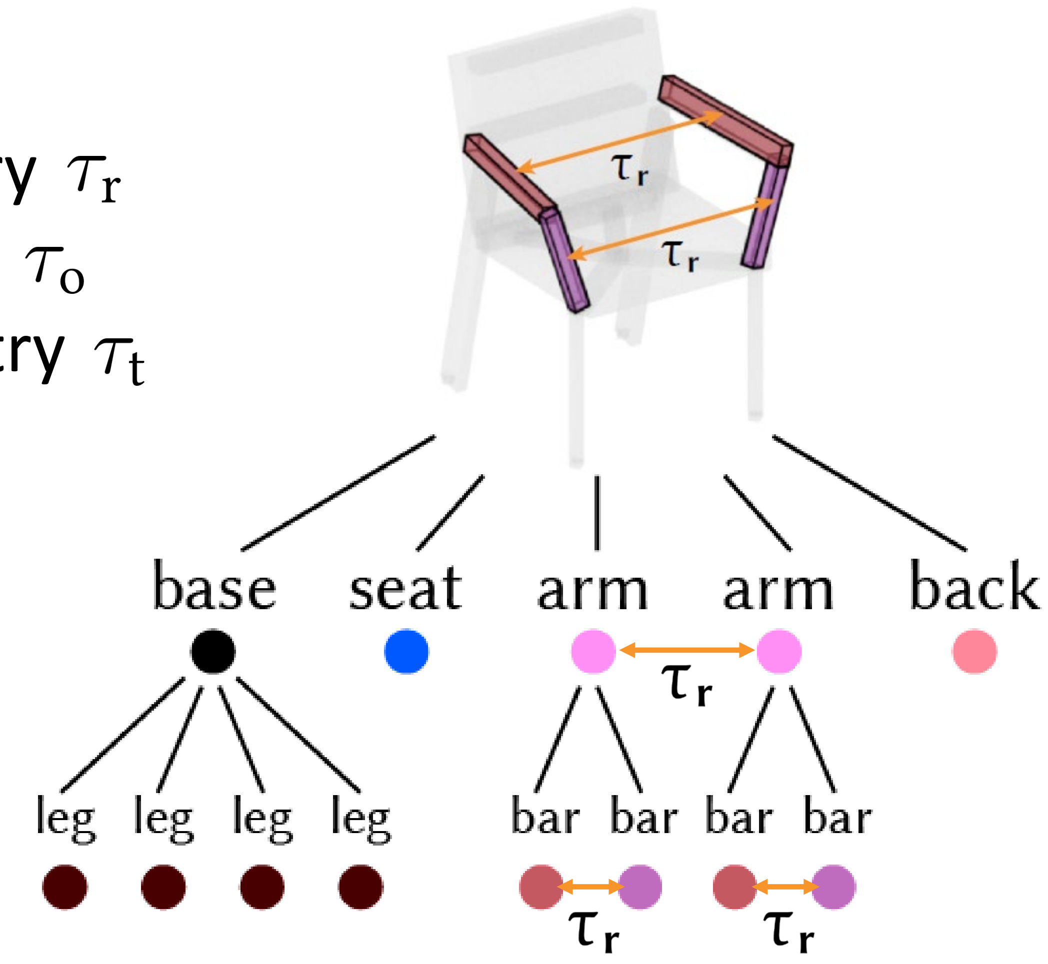


Structural Consistency

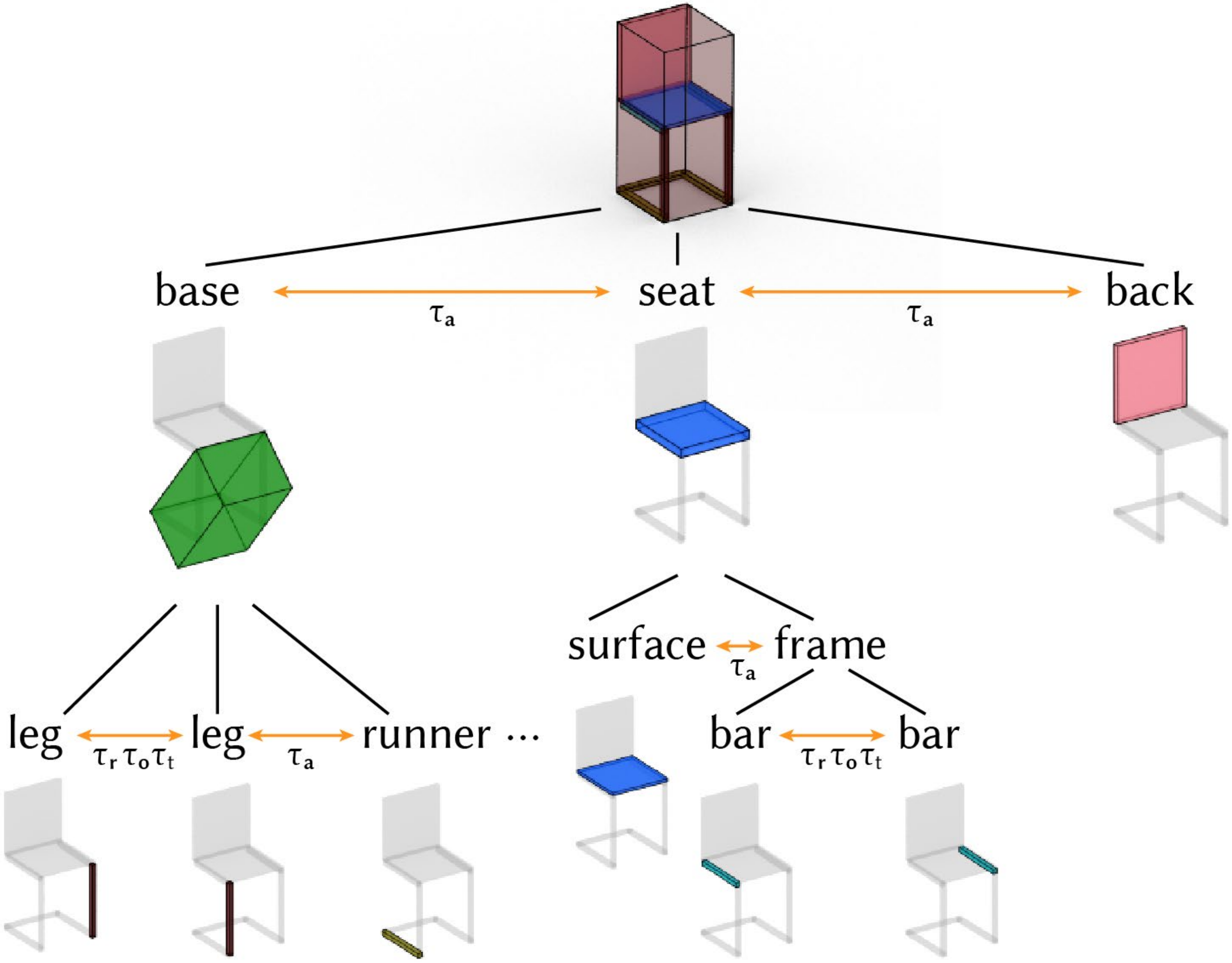


Object Representation: Sibling Relationships

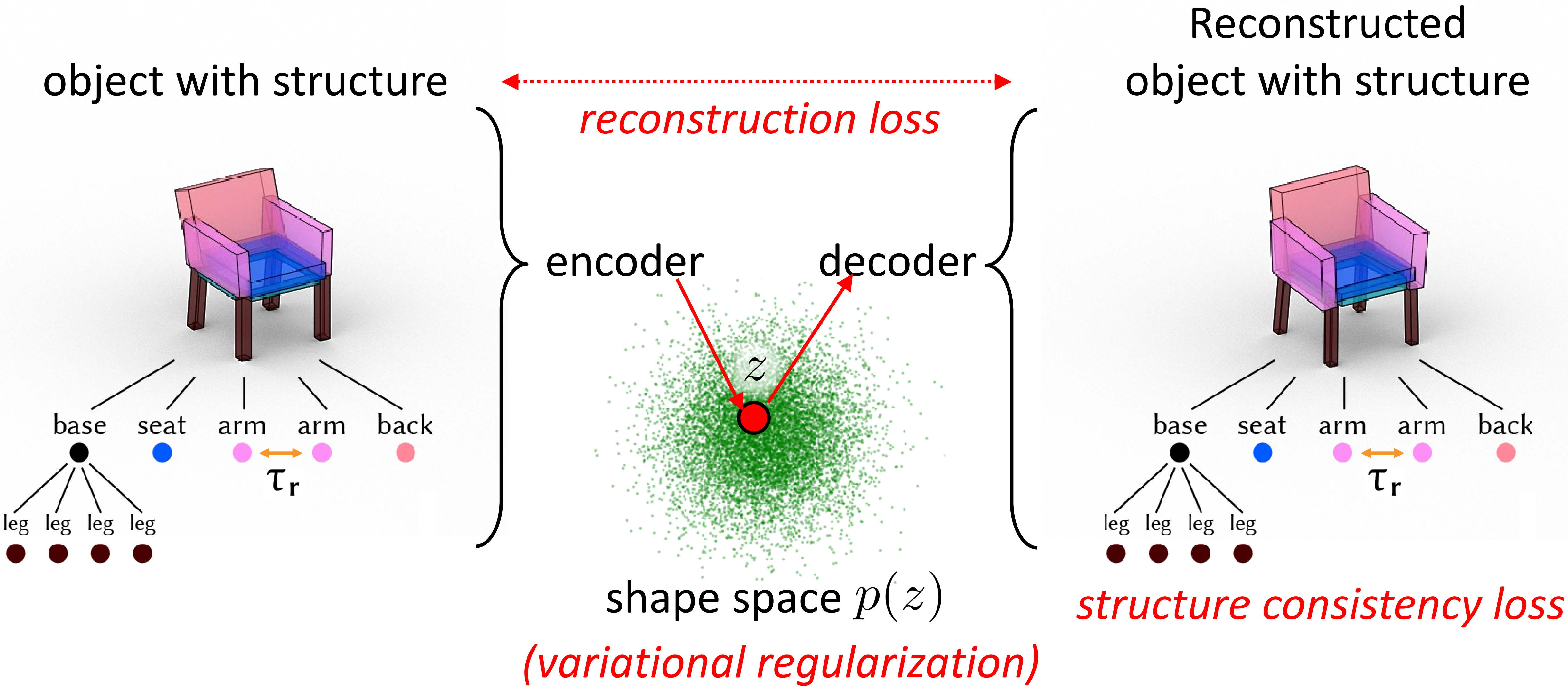
Reflectional Symmetry τ_r
Rotational Symmetry τ_o
Translational Symmetry τ_t
Adjacency τ_a



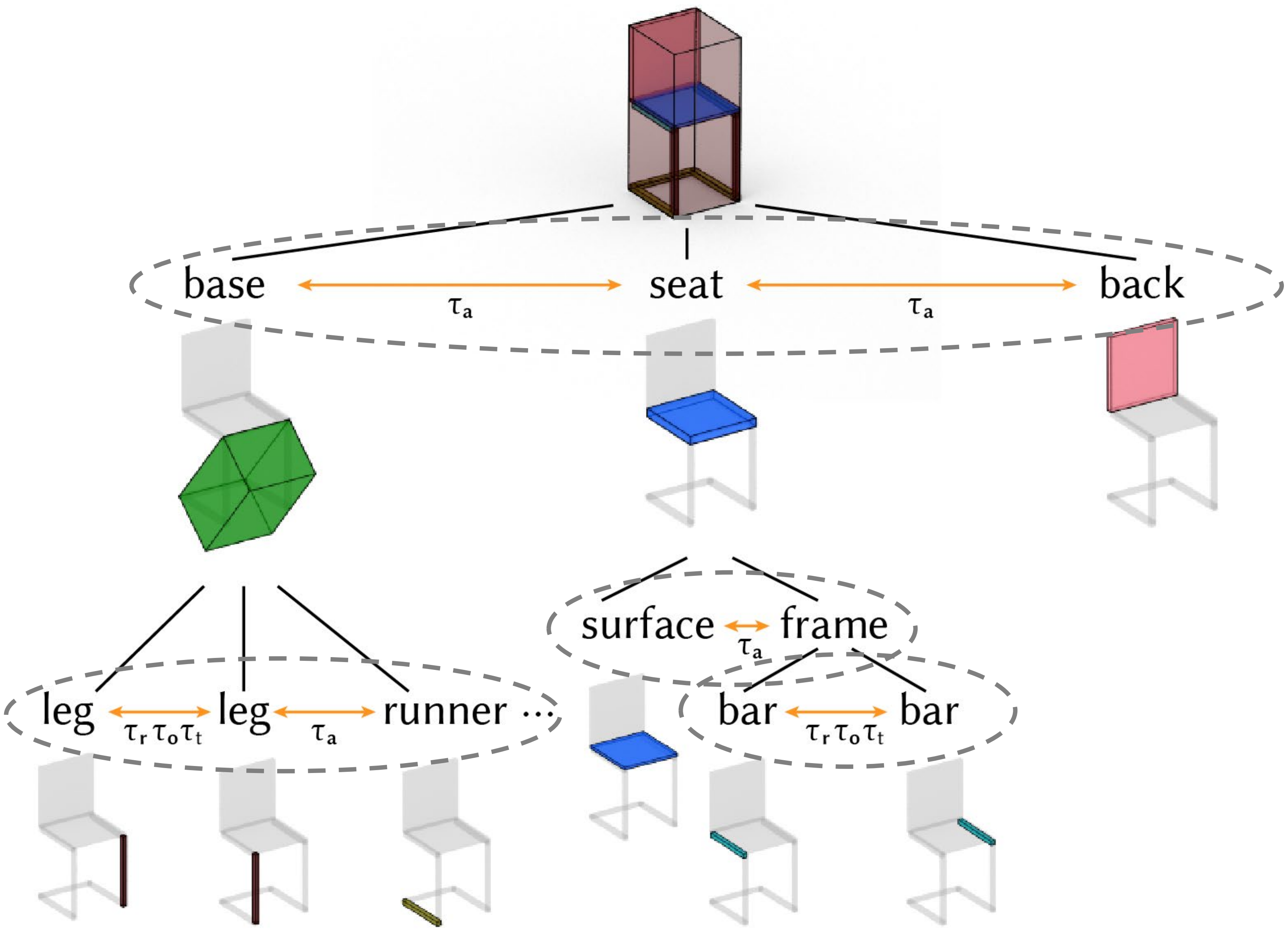
Object Representation: Example



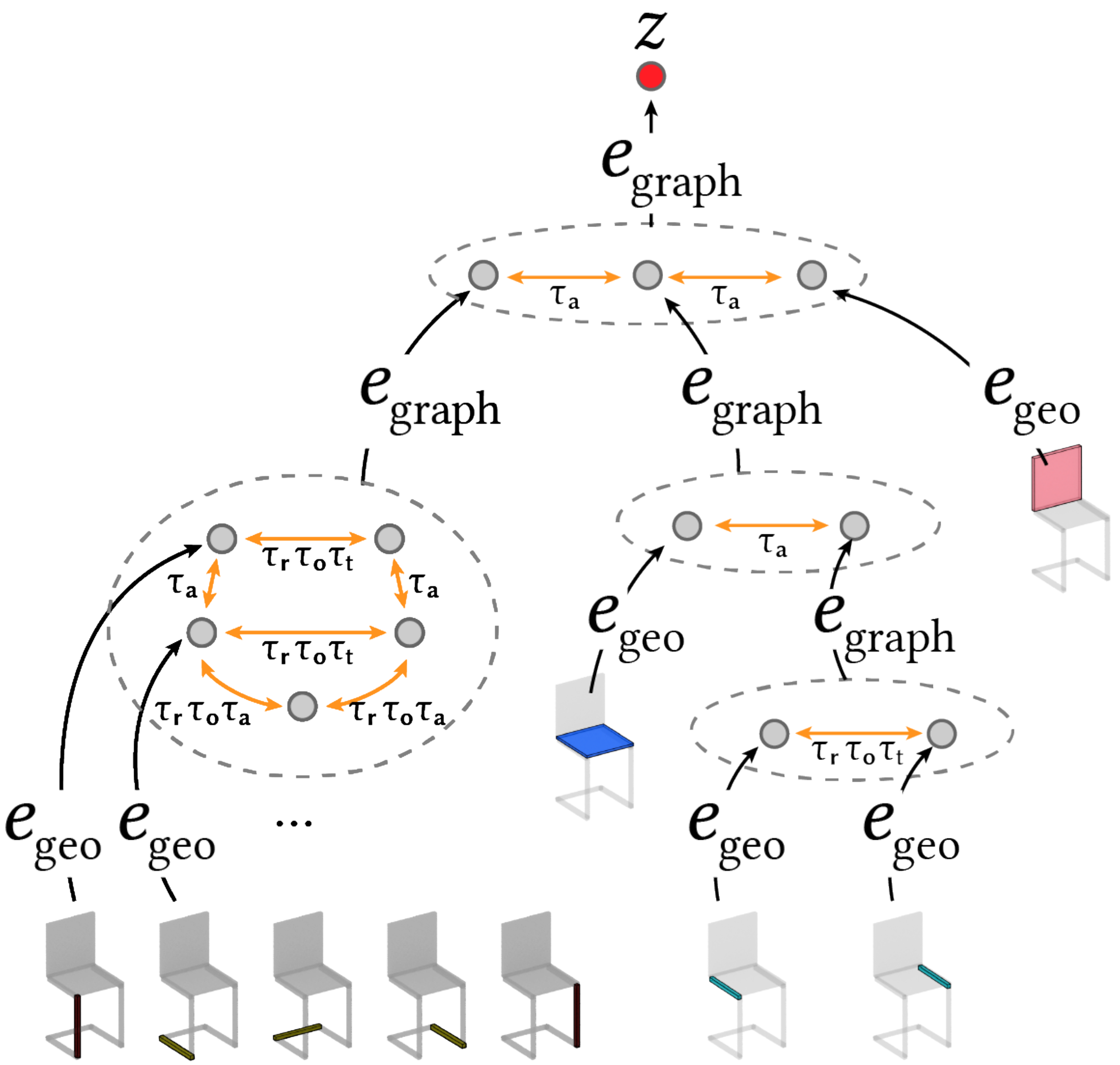
Architecture Overview: VAE Training

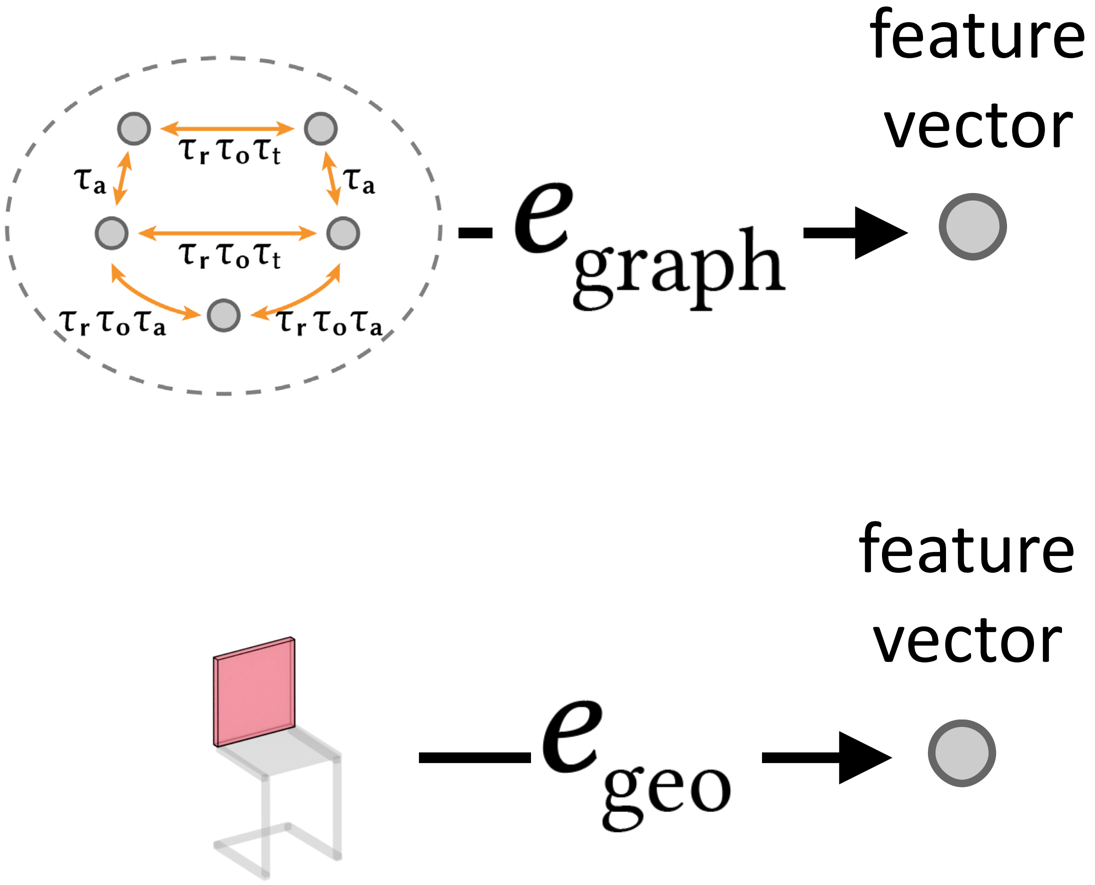


A Hierarchy of Graphs

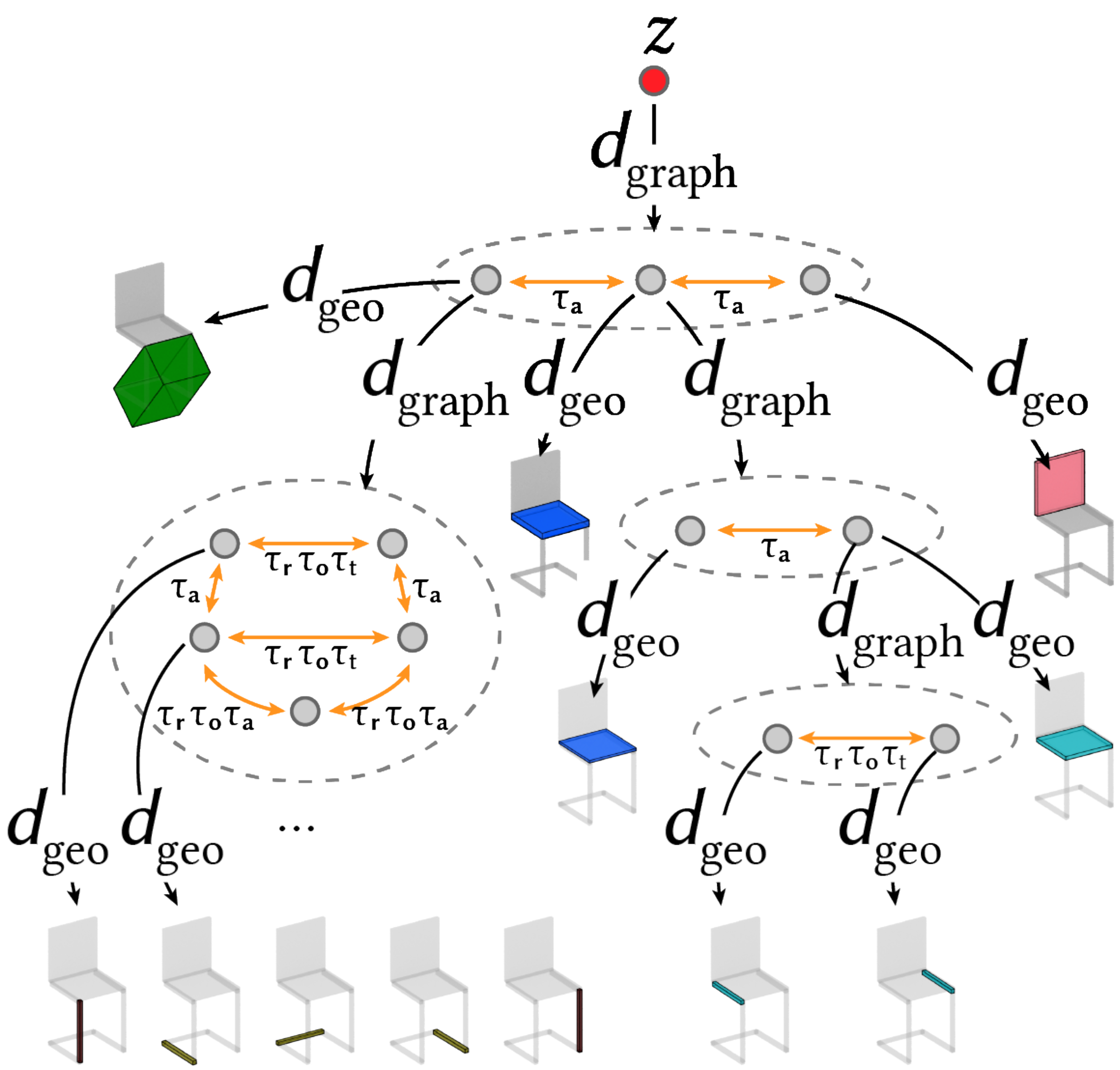


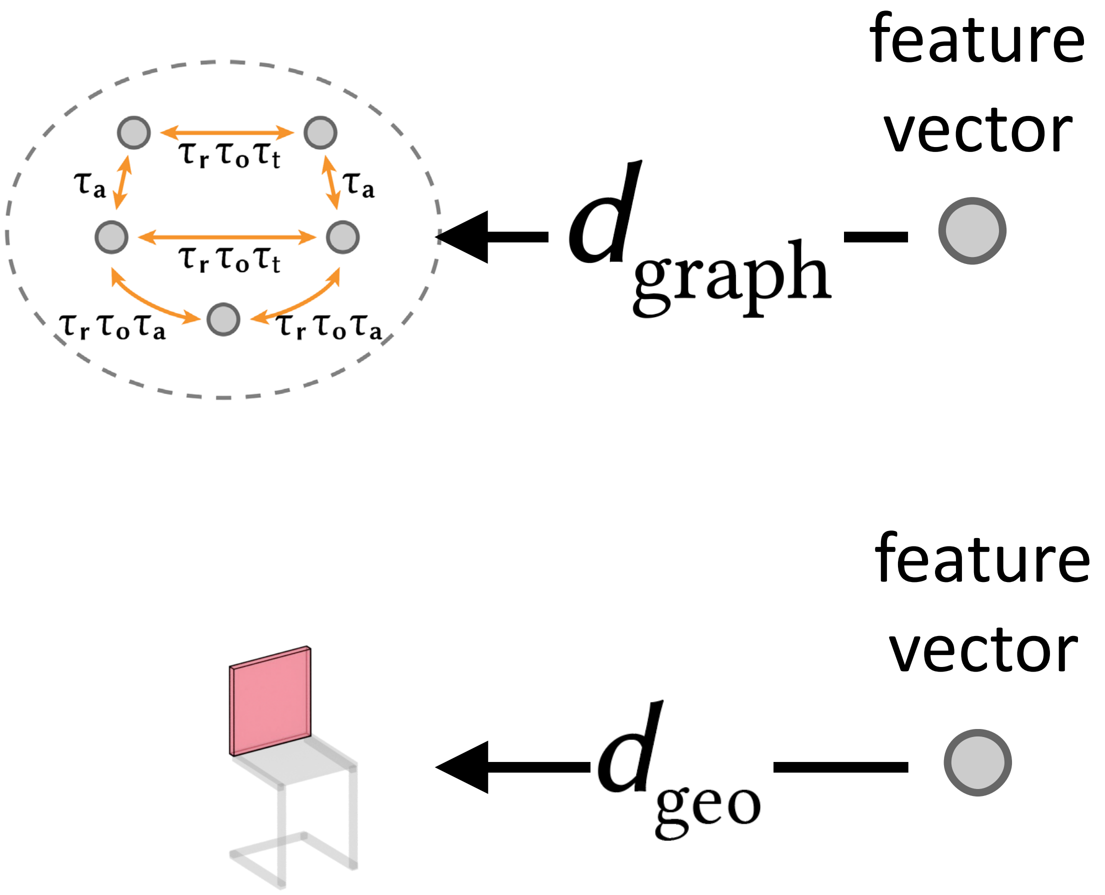
Hierarchical Graph Encoder



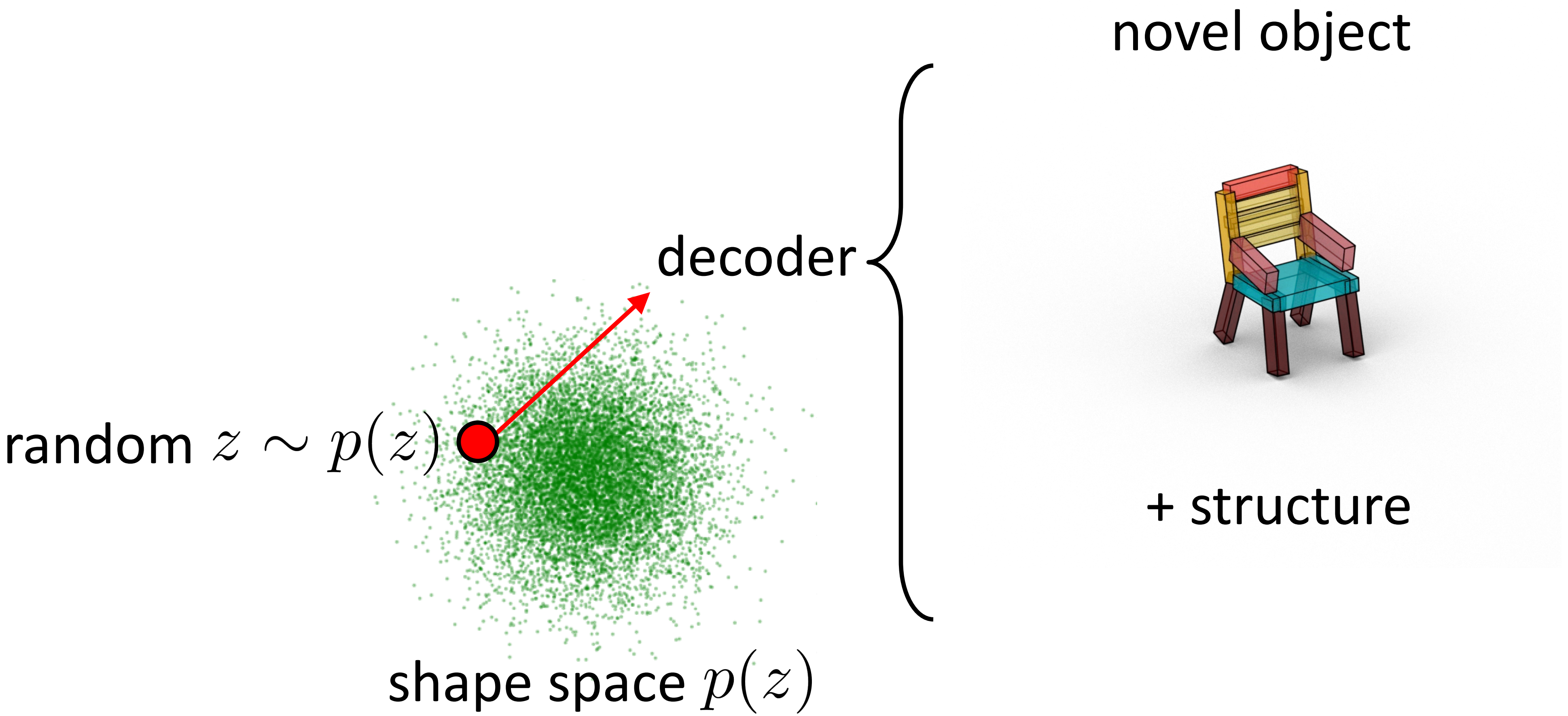


Hierarchical Graph Decoder





Application 1: Generation



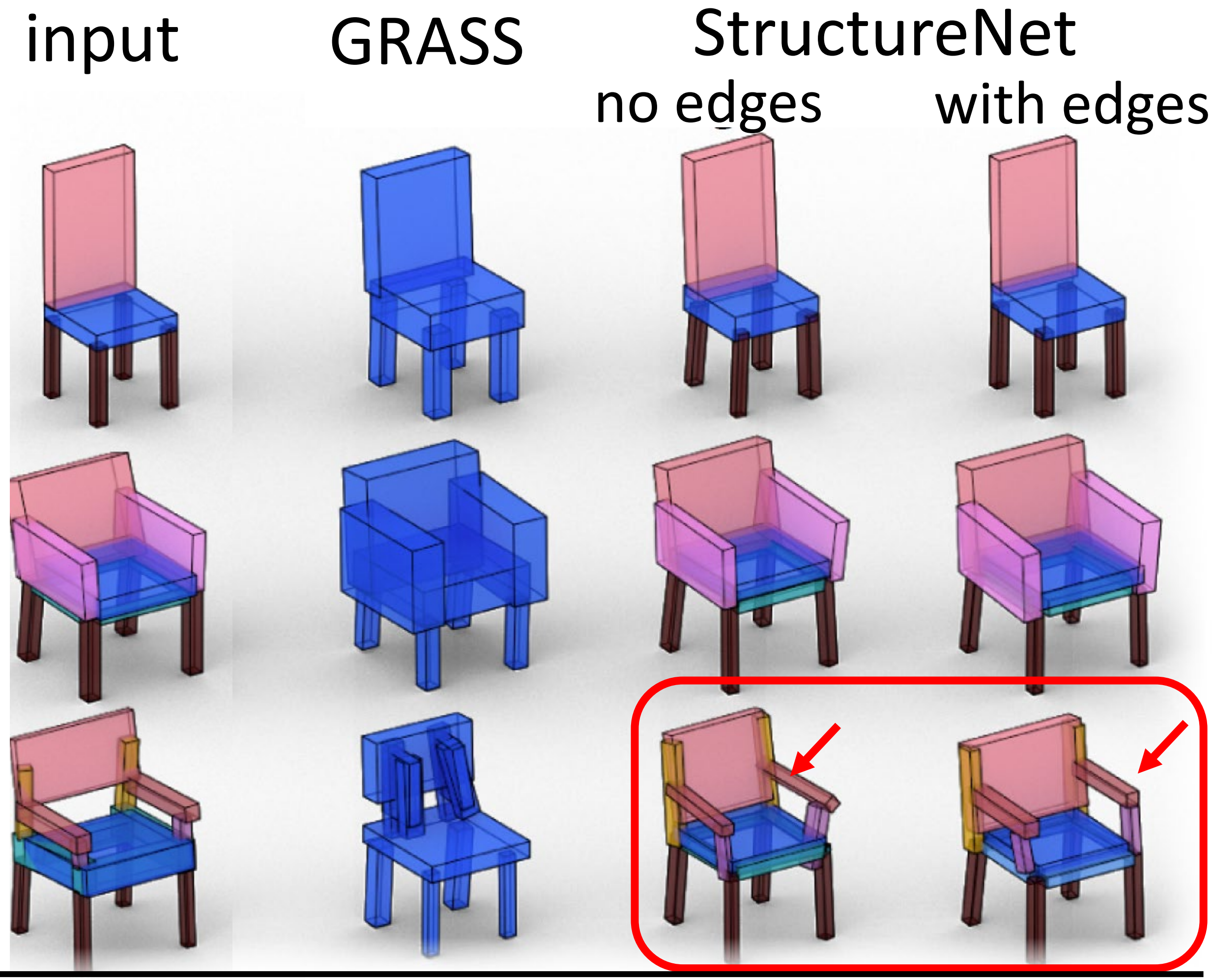
Generation

Generation

Novelty

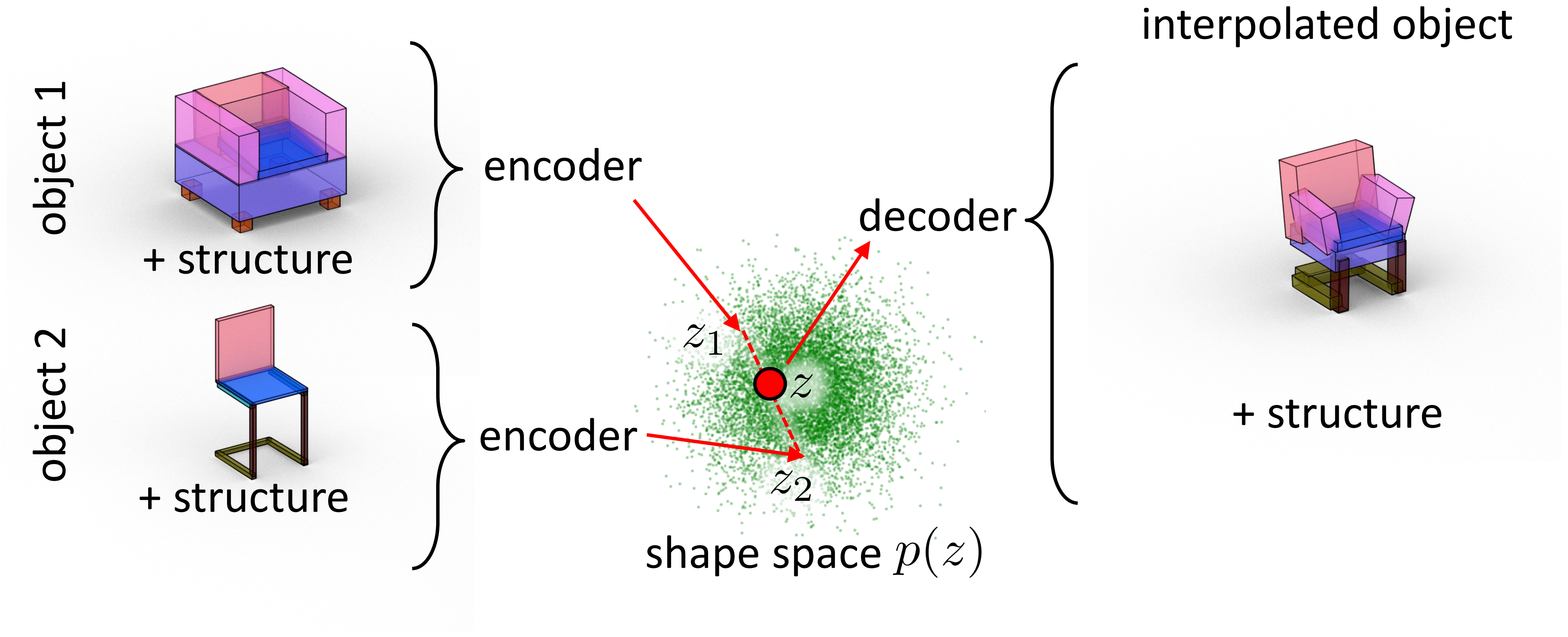
generated closest training samples

Comparison to GRASS

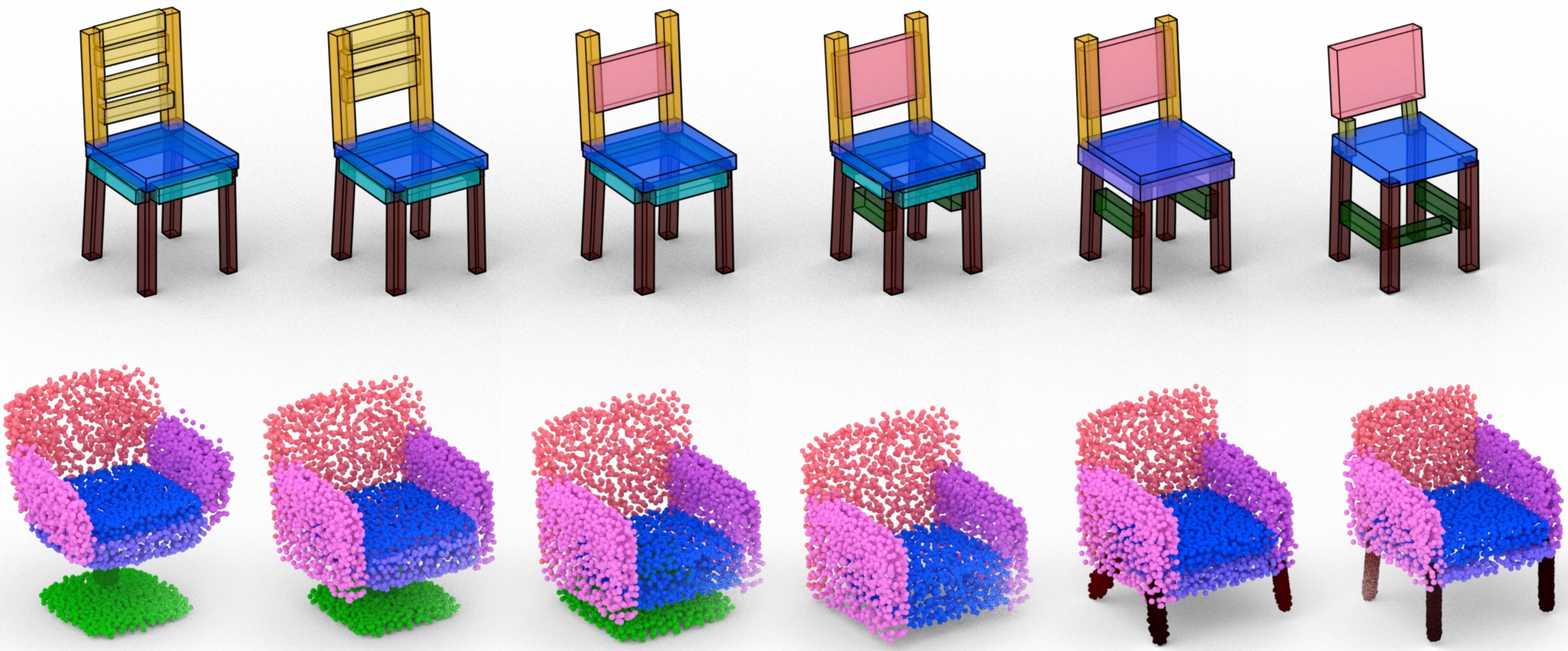


	GRASS	StructureNet no edges	StructureNet with edges
relative quality	0.788	0.984	1.0
relative coverage	0.818	0.989	1.0

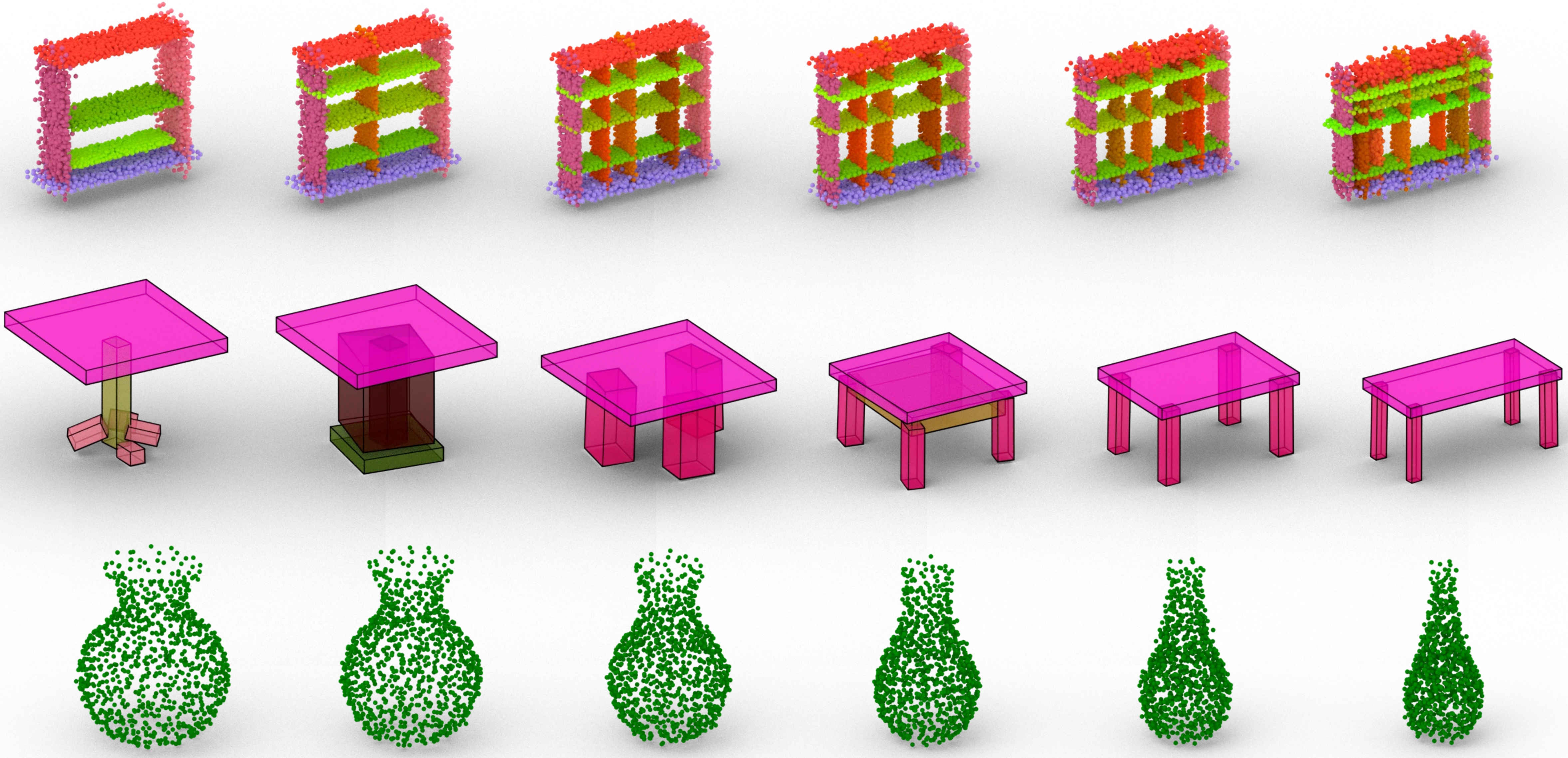
Application 2: Interpolation



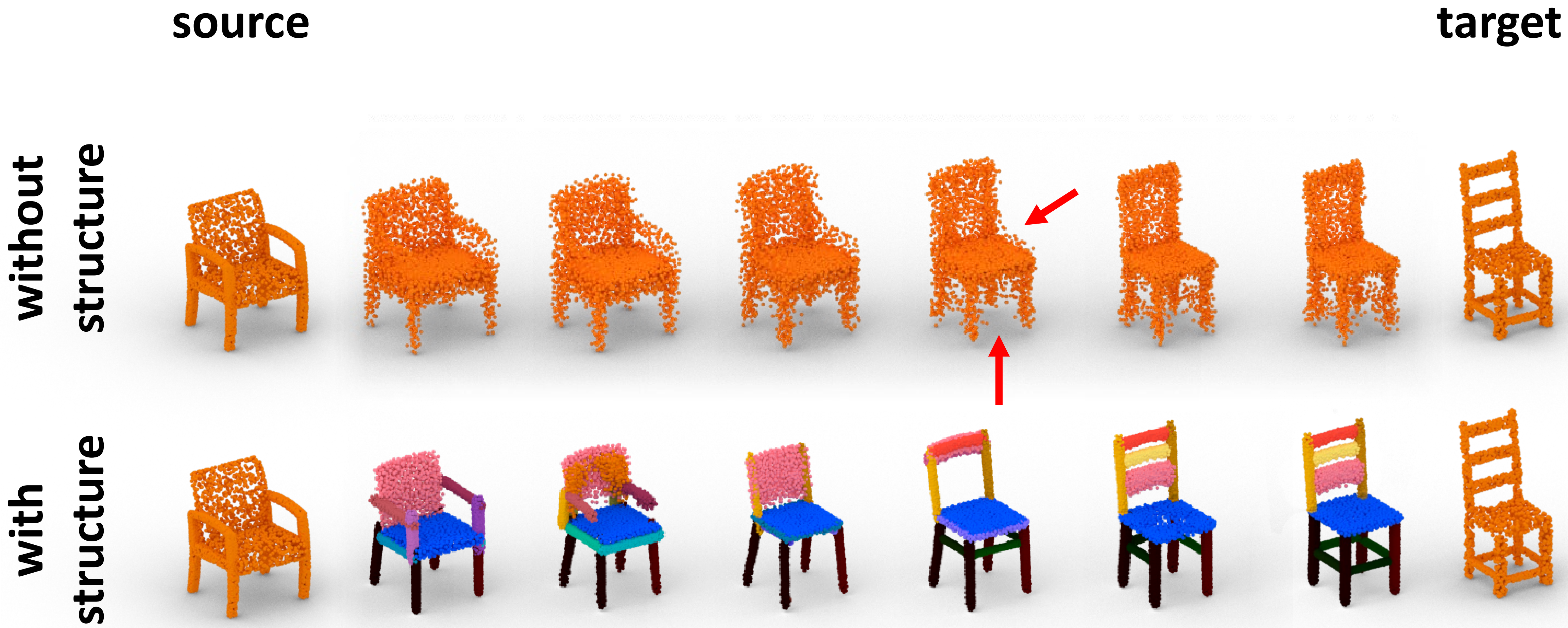
Interpolation



Interpolation



With vs. Without Structure



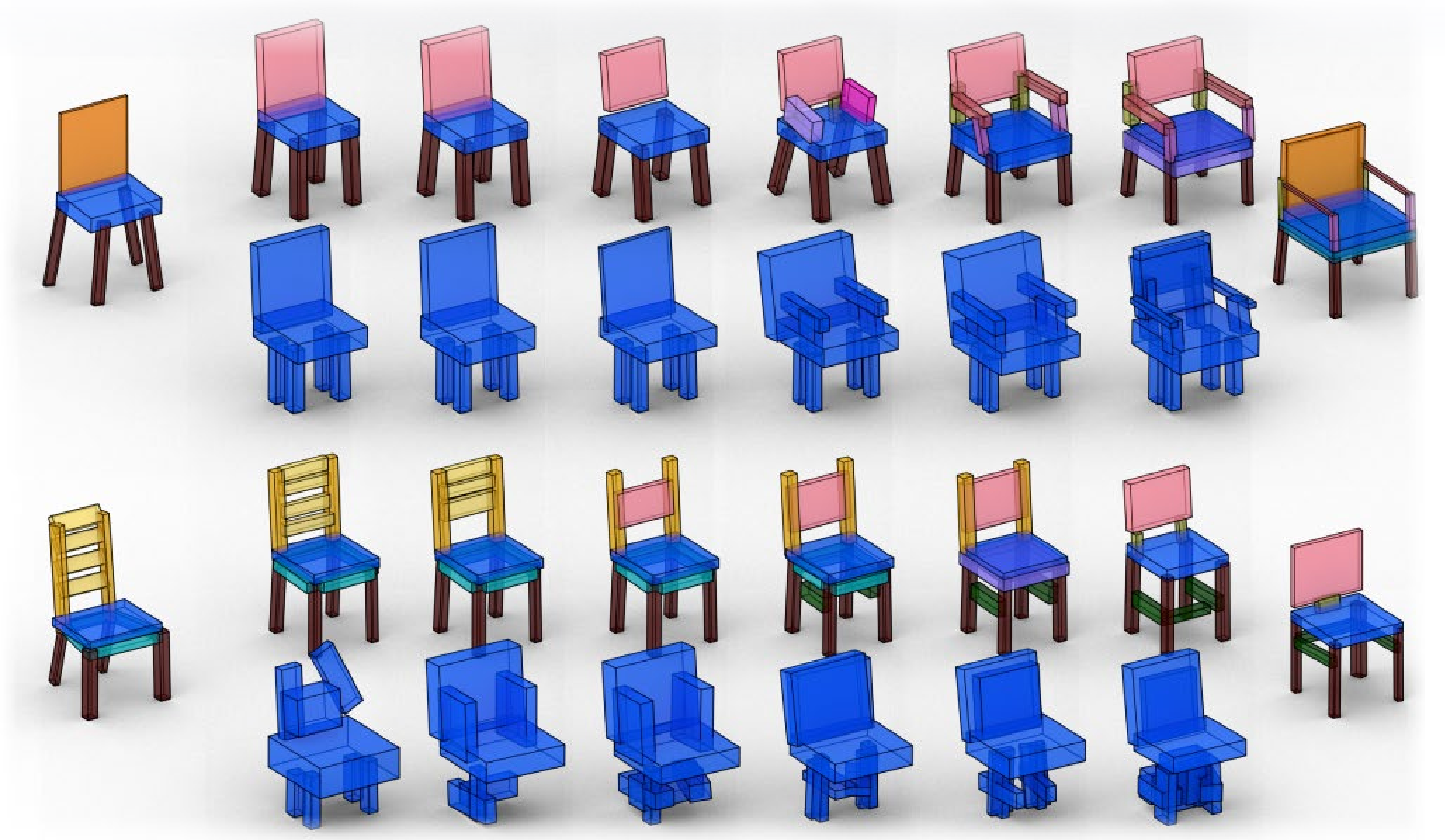
Interpolation StructureNet vs. GRASS

StructureNet

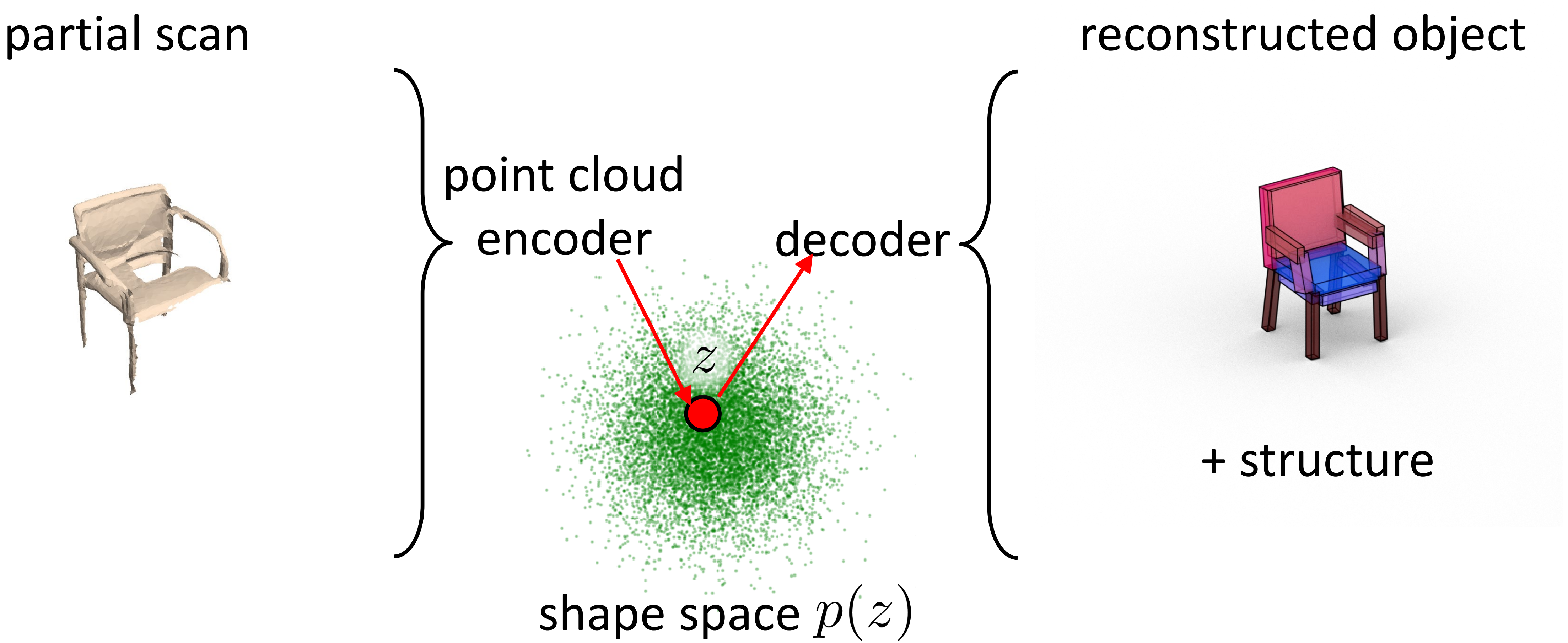
GRASS

StructureNet

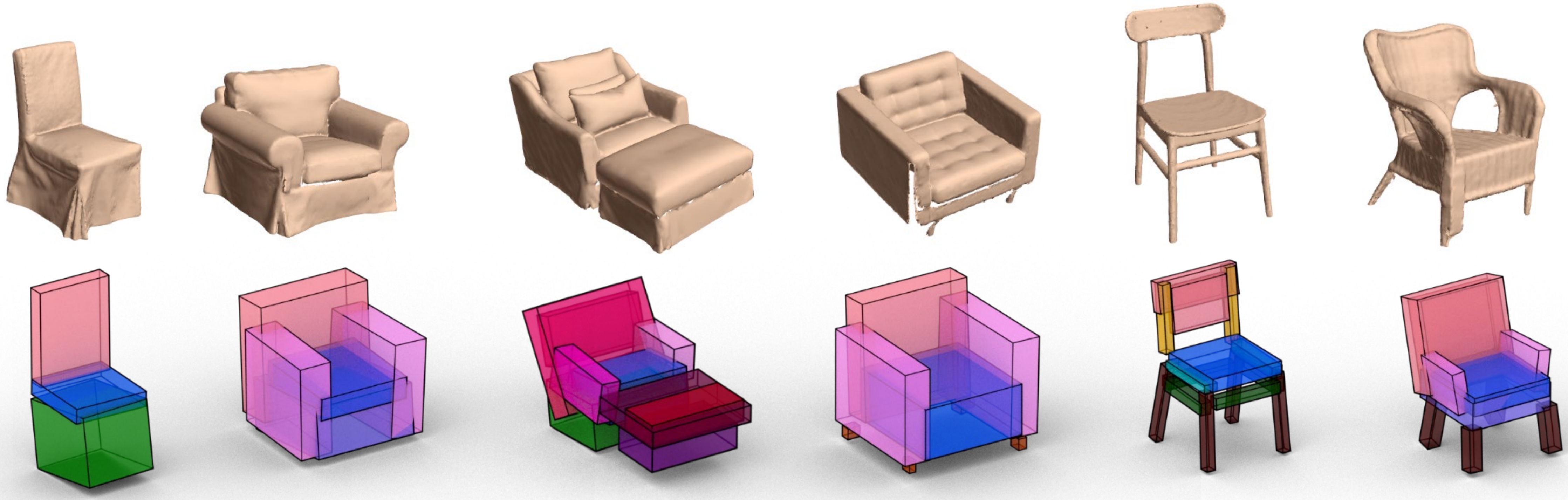
GRASS



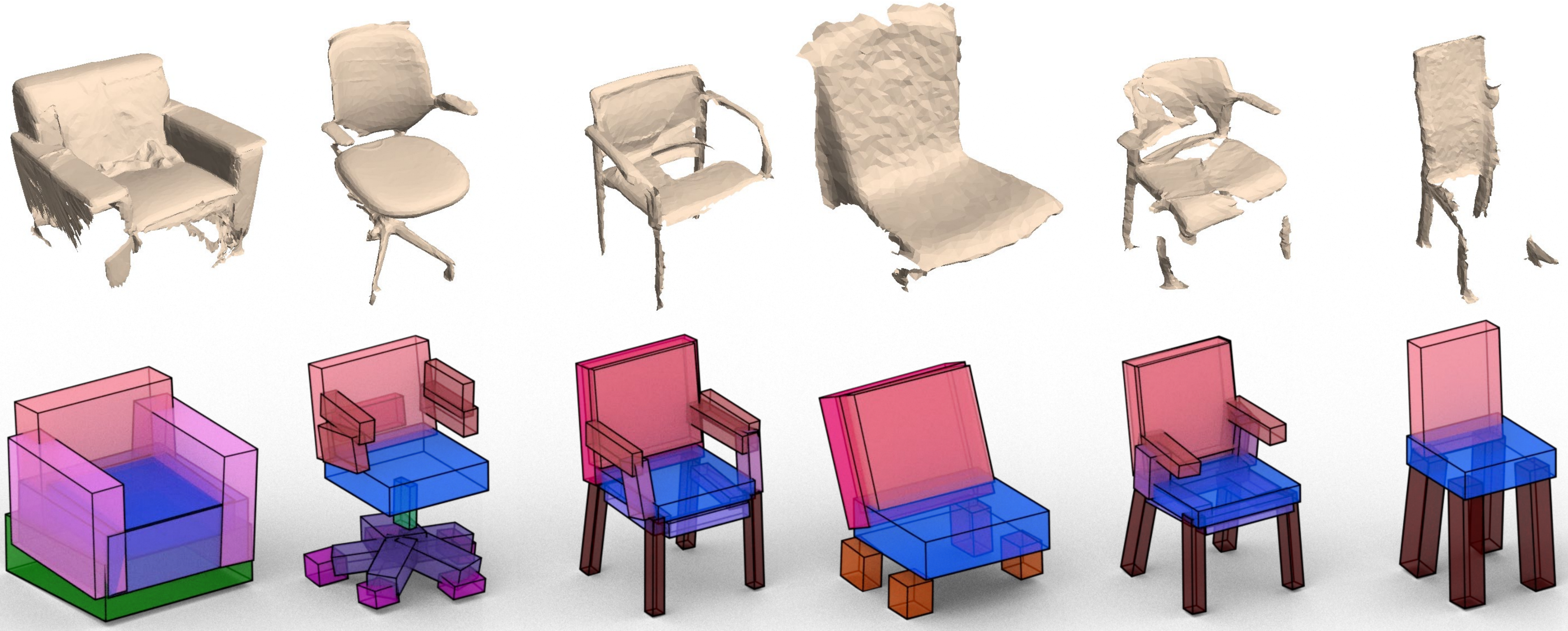
Application 3: Scan Abstraction



Abstraction of Full Scans



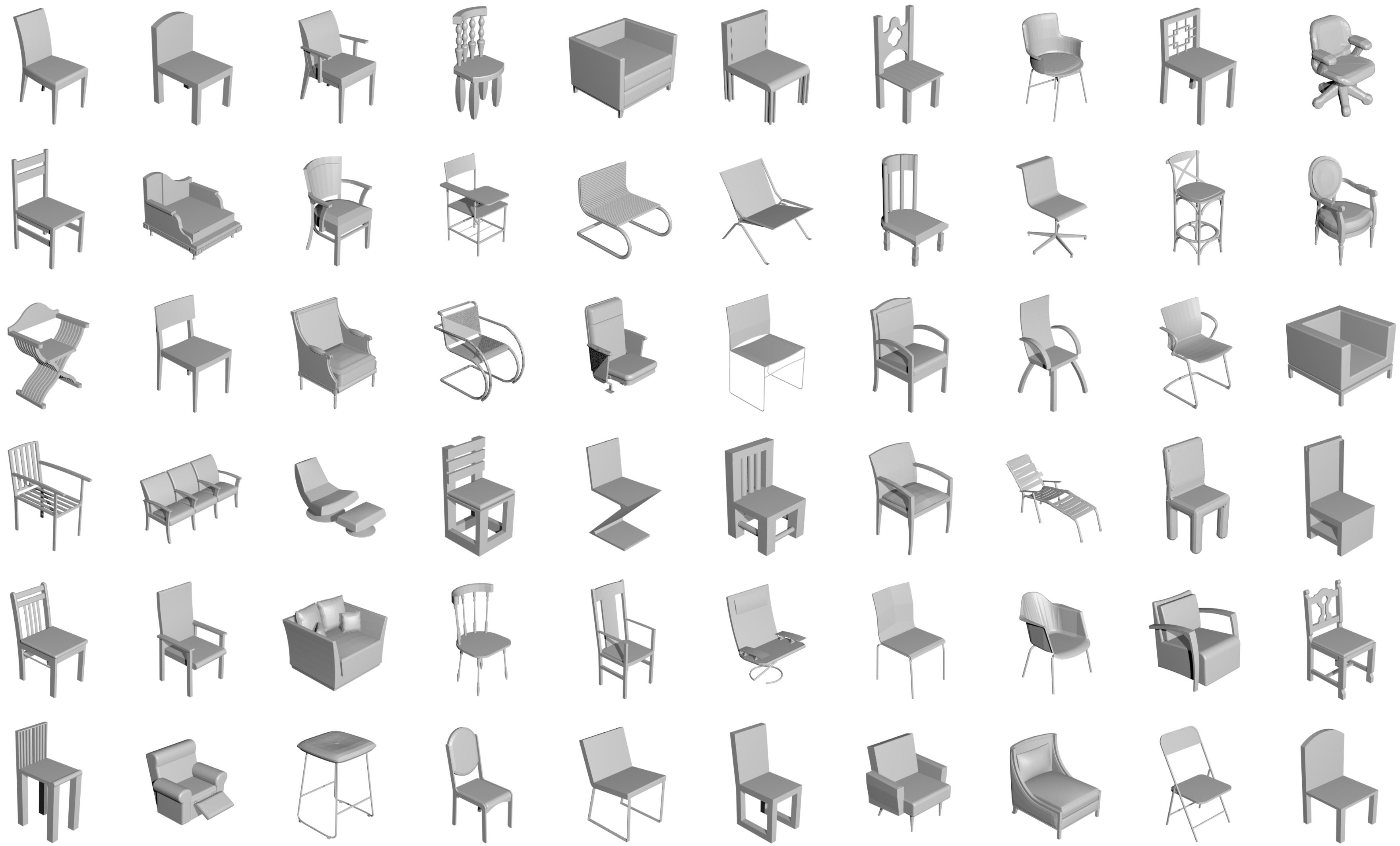
Abstraction of Partial Scans



Shape Abstraction via Volumetric Primitives

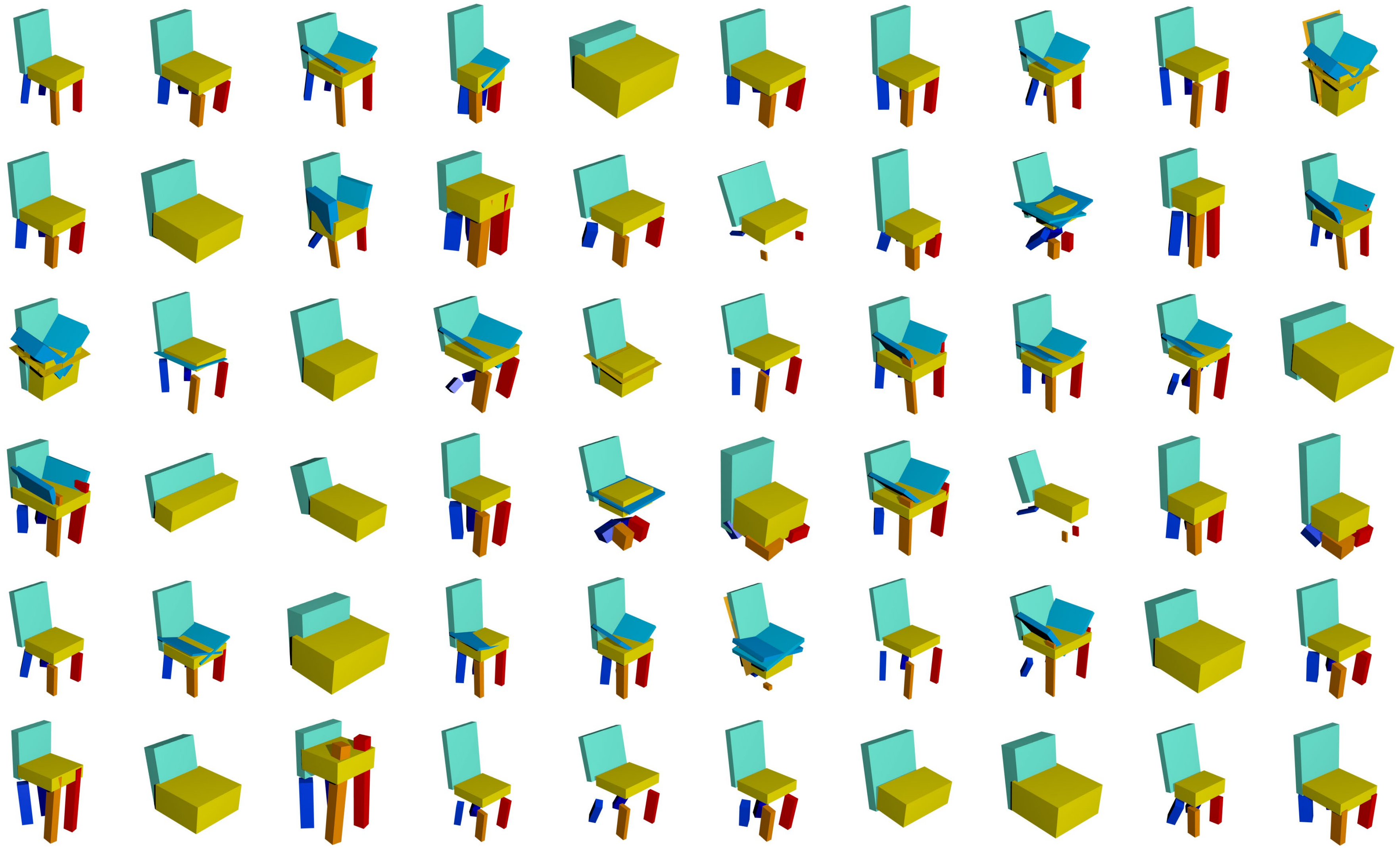
Tulsiani, Shubham, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning Shape Abstractions by Assembling Volumetric Primitives. CVPR 2017.

Input



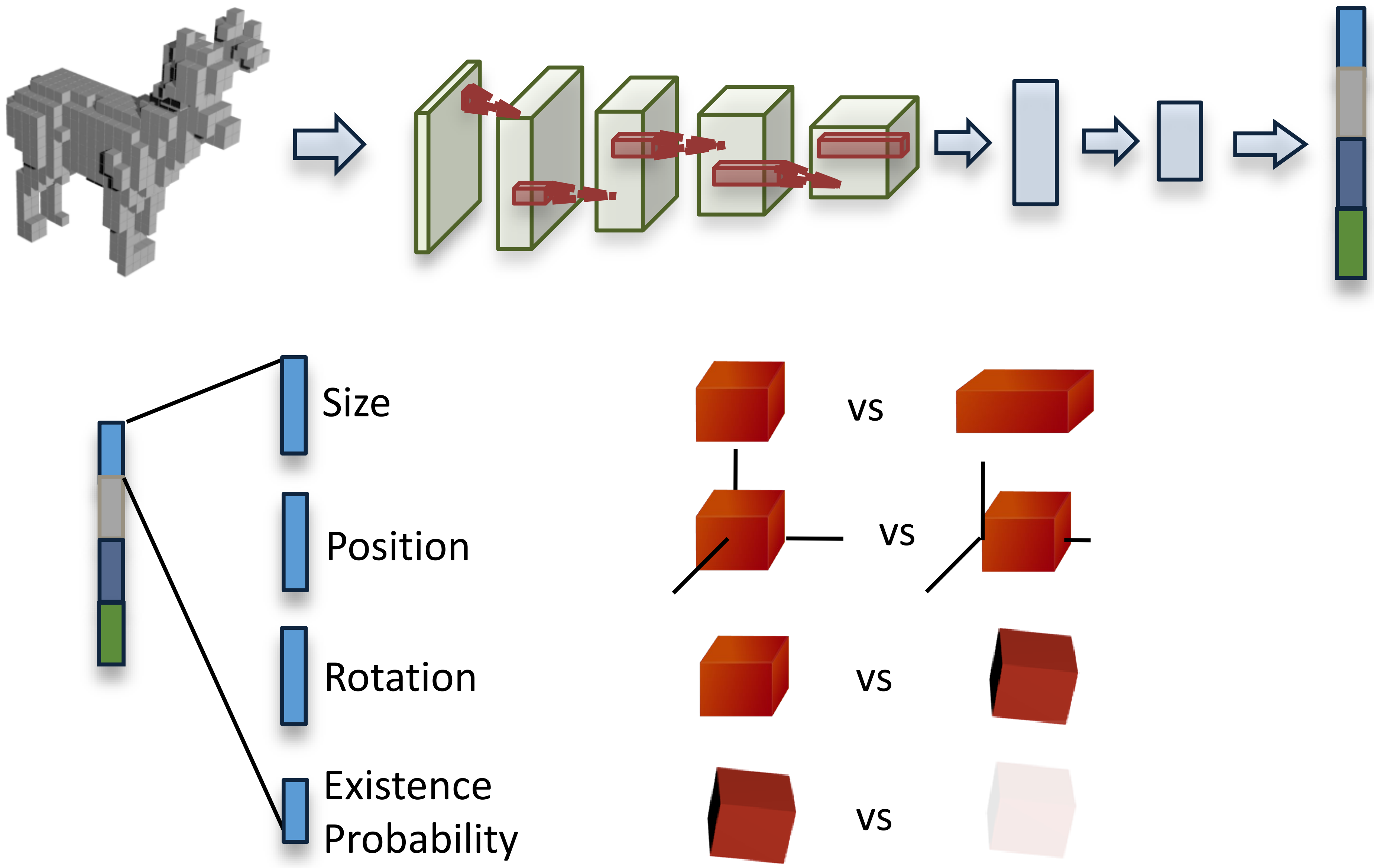
Shape Collection

Output

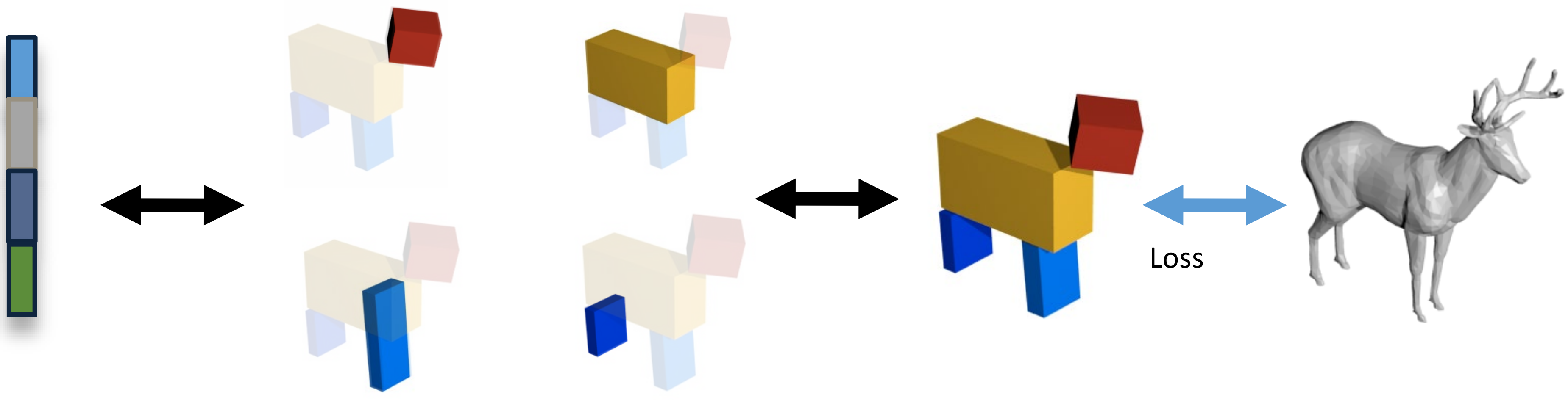


Unsupervised Consistent Abstractions

Approach

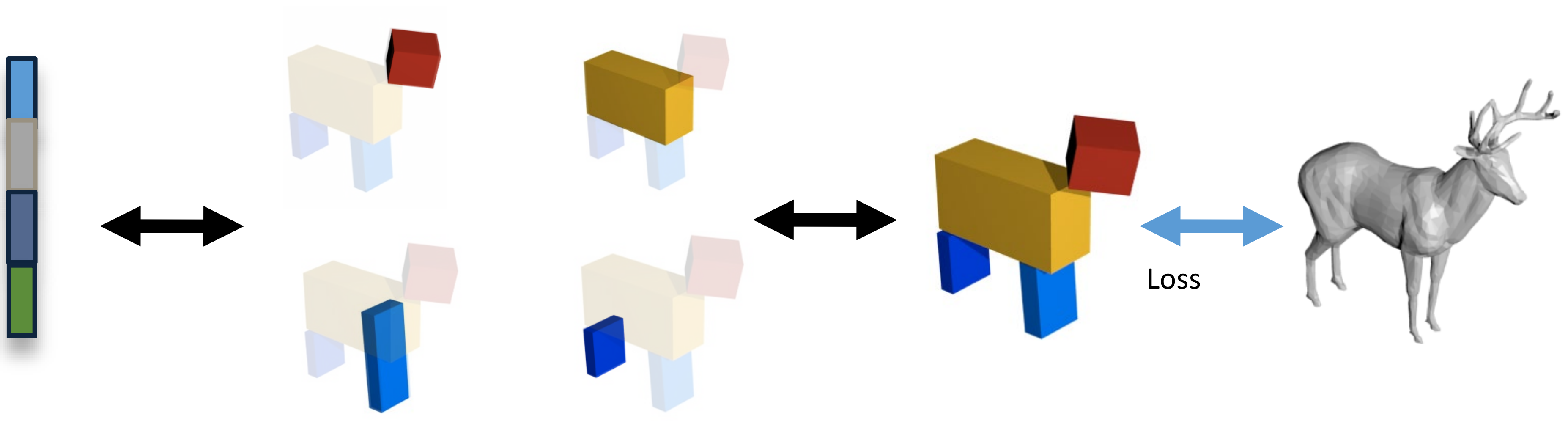


Unsupervised Loss Function



The predicted parameters are trained with a loss that tries to minimize the distance between assembled boxes and ground-truth mesh

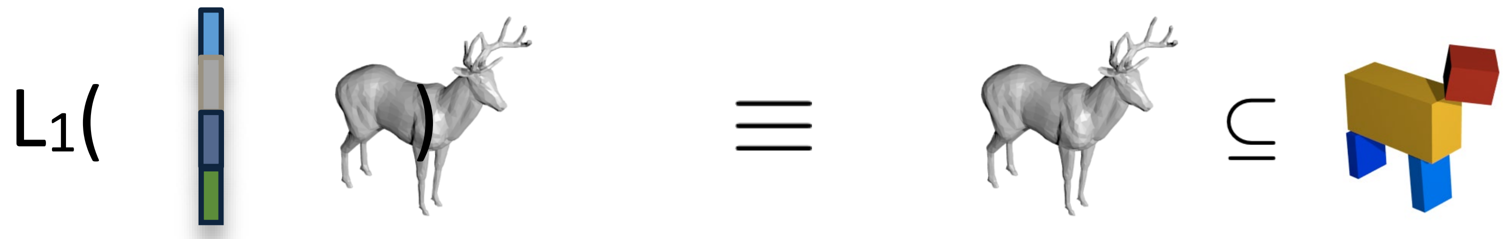
Unsupervised Loss Function

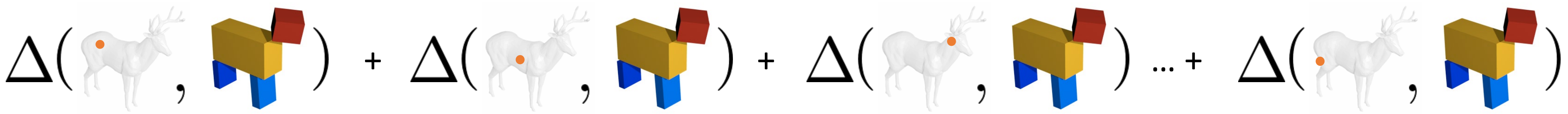


Coverage Loss : $O \subseteq \bigcup_m \bar{P}_m$

Consistency Loss : $\bigcup_m \bar{P}_m \subseteq O$

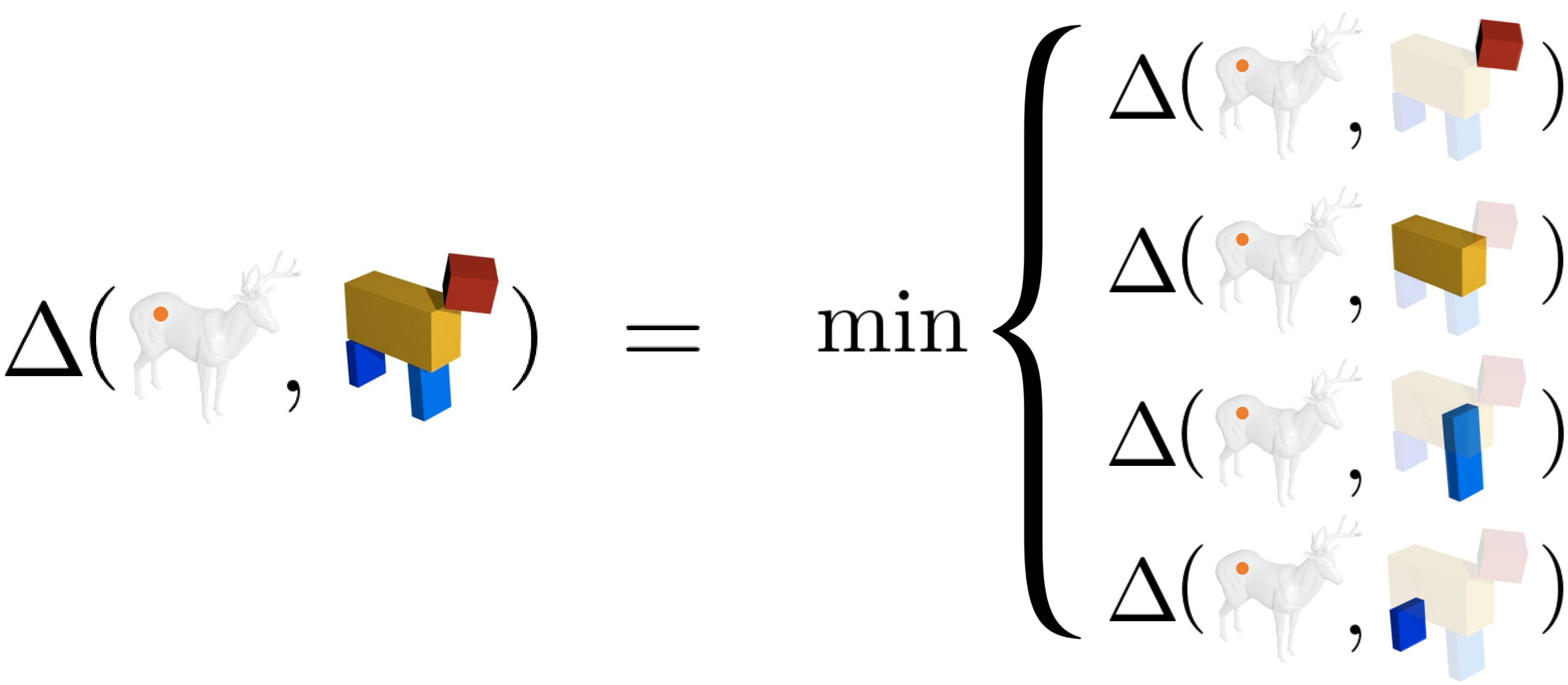
Coverage Loss





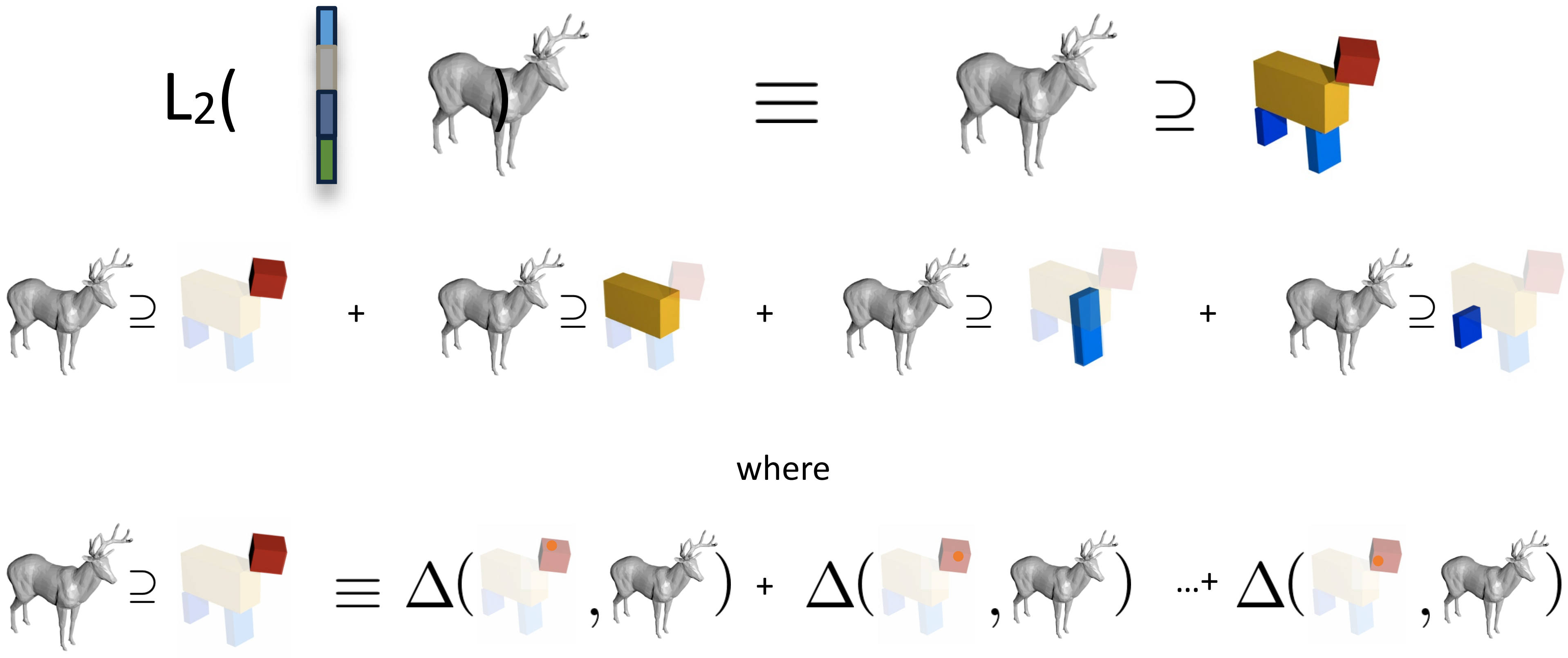
$$L_1(\{(z_m, q_m, t_m)\}, O) = \mathbb{E}_{p \sim S(O)} \|\mathcal{C}(p; \bigcup_m \bar{P}_m)\|^2$$

Coverage Loss



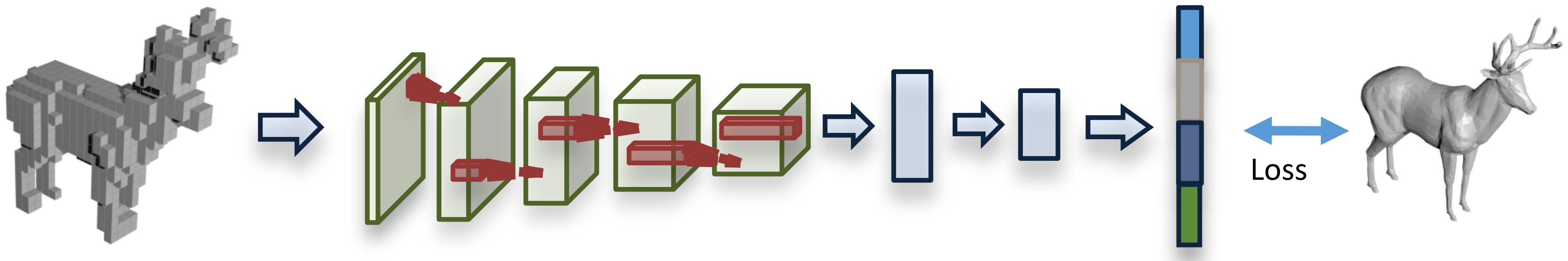
$$\mathcal{C}(p; \bigcup_m \bar{P}_m) = \min_m \mathcal{C}(p; \bar{P}_m)$$

Consistency Loss



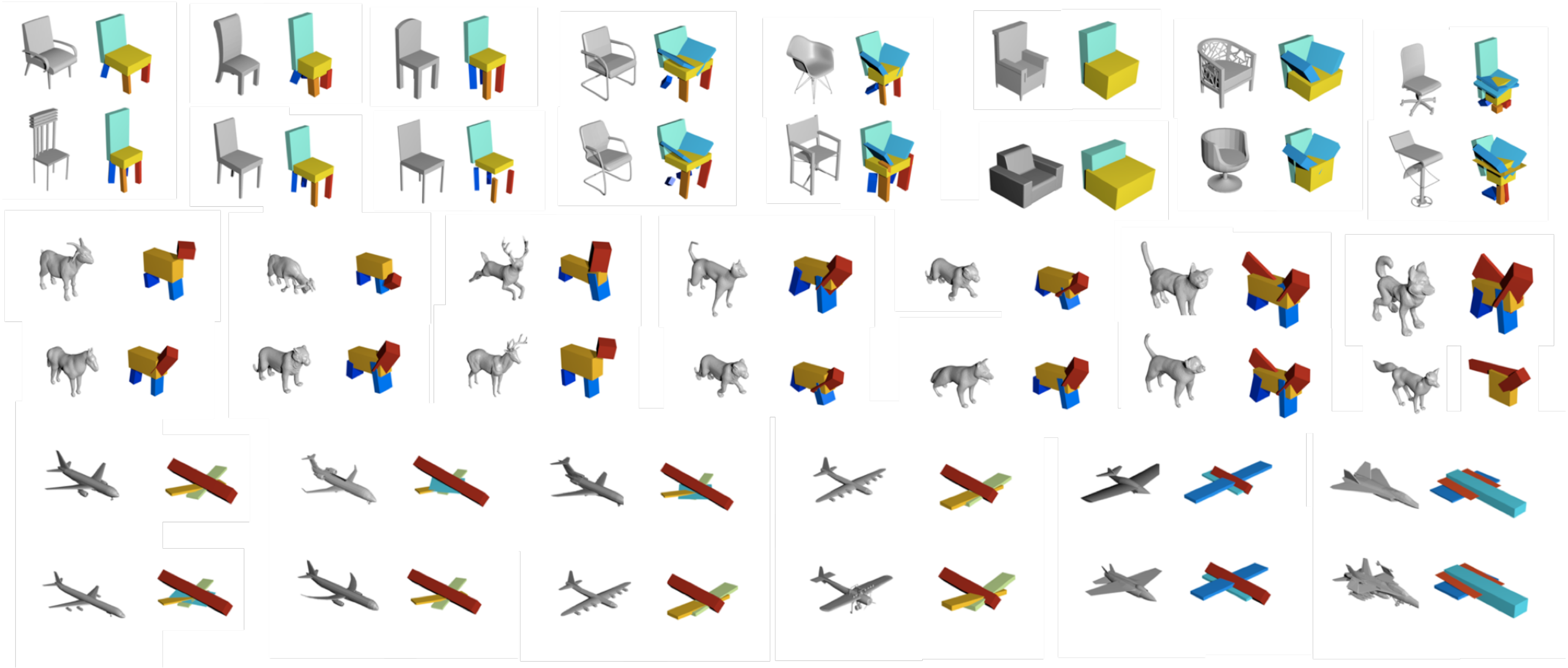
$$L_2(\{(z_m, q_m, t_m)\}, O) = \sum_m \mathbb{E}_{p \sim S(\bar{P}_m)} \|\mathcal{C}(p; O)\|^2$$

Approach Summary

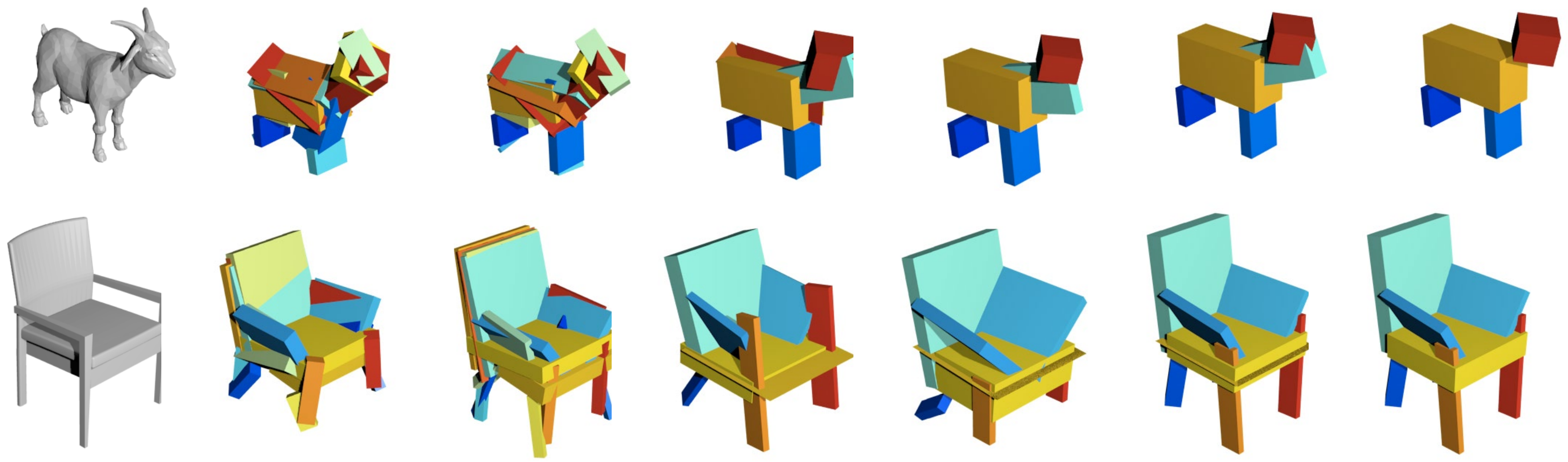


We train a CNN to predict primitive parameters such that the assembled shape is similar to the underlying object

Results



Analysis



Shapes become more parsimonious as training progresses (due to our parsimony reward)

Unsupervised Parsing

Projection of the predicted primitives onto the original shape.
We visualize the parsing by coloring each point according to the assigned primitive.
We see that similar parts e.g. airplane wings, chair seat, etc. are consistently colored.

That's All