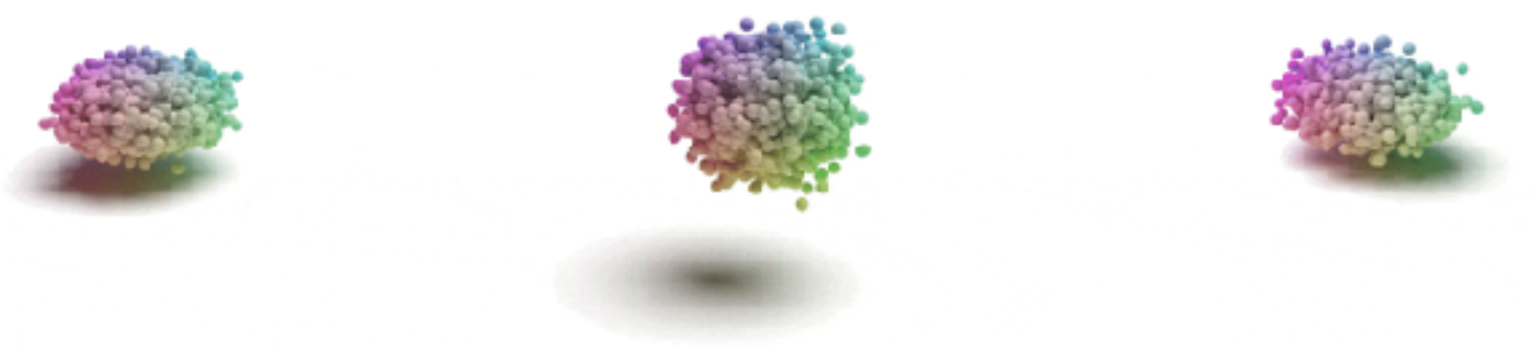
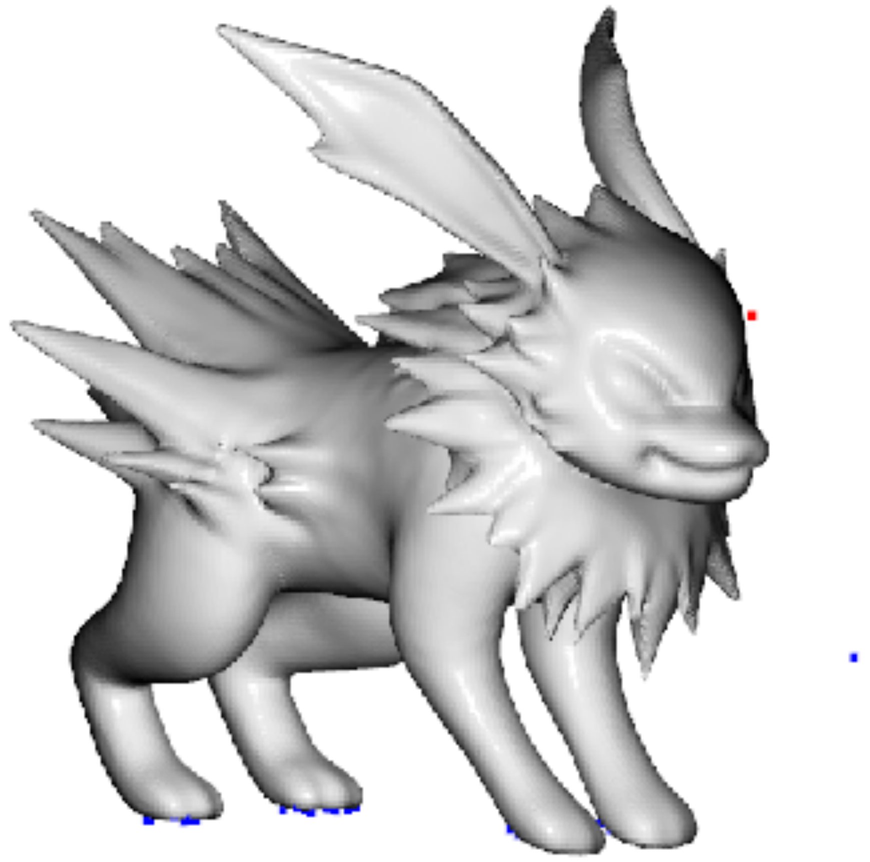
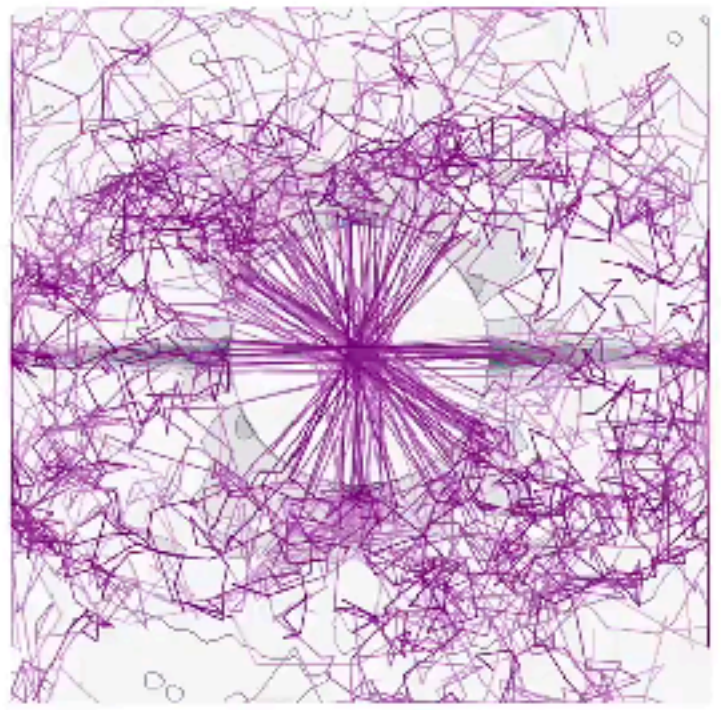


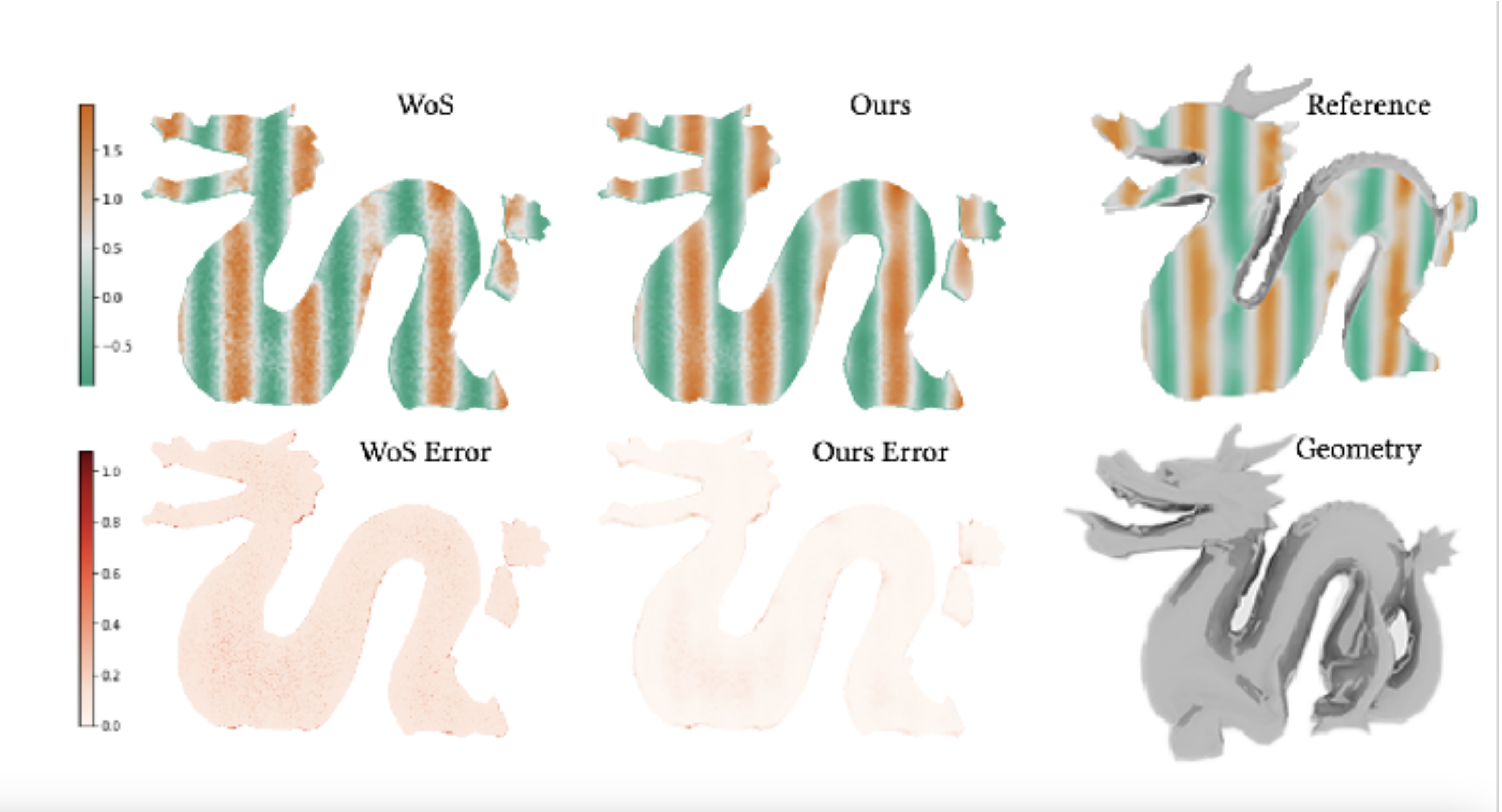


Shapes as Fields

Toward Geometry Processing without Discretization

Guandao Yang
Postdoc, Stanford





Guandao Yang

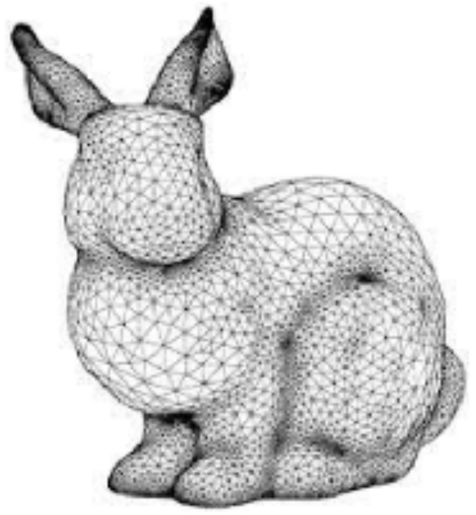
Postdoc @ Stanford

- Ph.D. from Cornell University, advised by Prof. Serge Belongie and Prof. Bharath Hariharan
- Research in the intersection of ML, CV, and CG.
- Collaborate with NVIDIA, Intel, Google, Magic Leap, and Adobe
- Hobbies: Trad climbing, piano

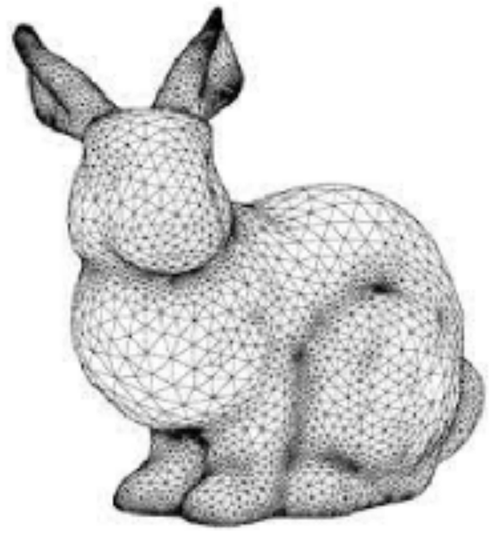
Geometry Processing

Creation, Manipulation, Analysis of Shapes

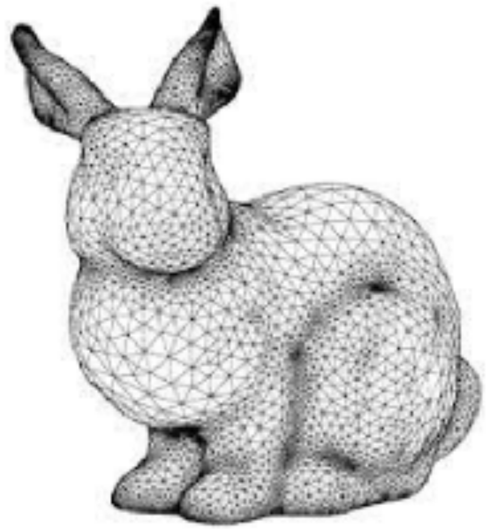
Creation

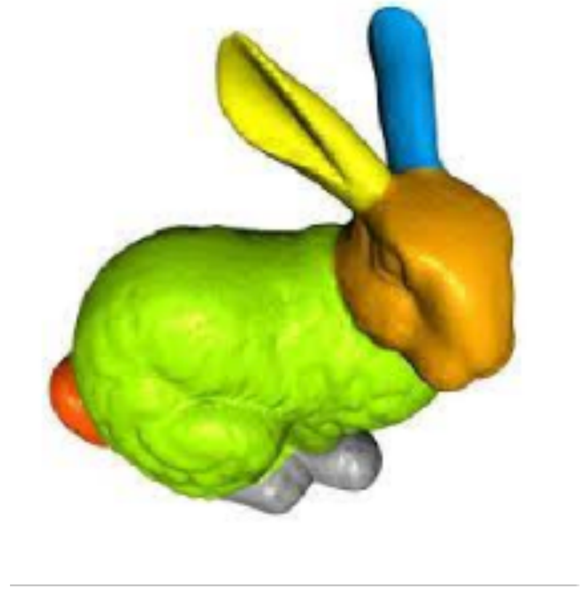


Manipulation



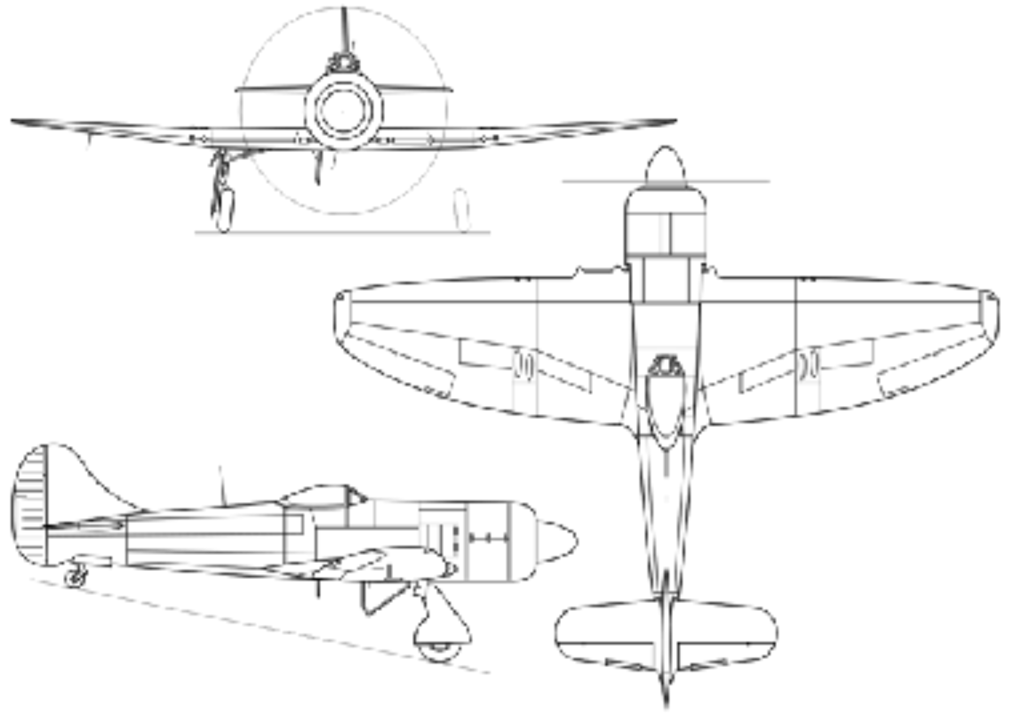
Analysis





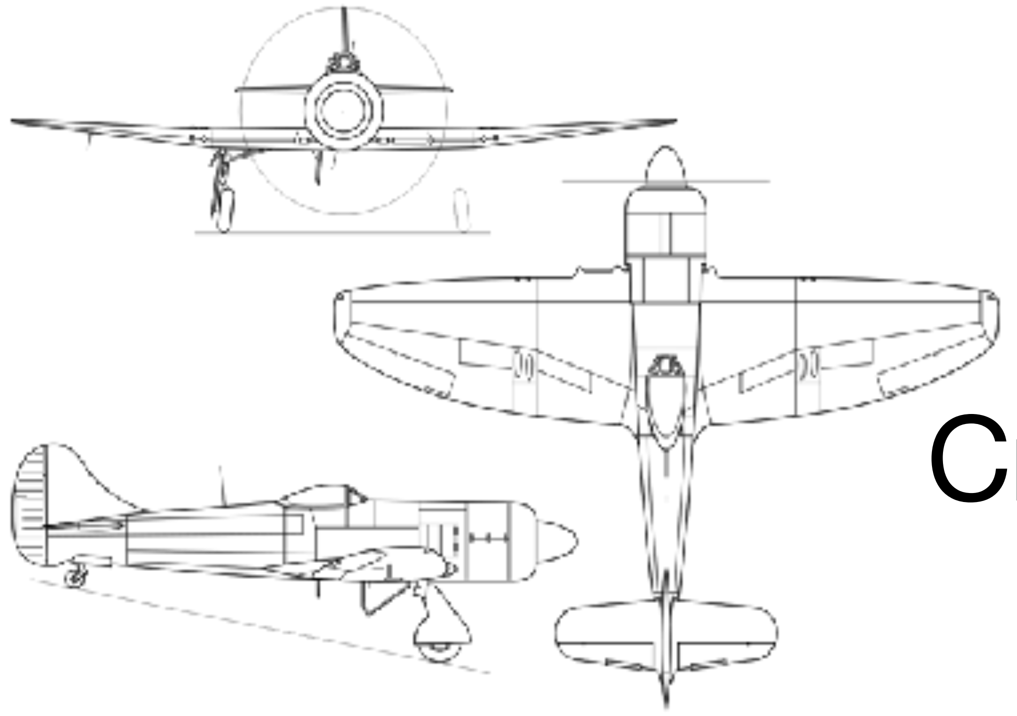
The Birth of a Digital Shape

Idea

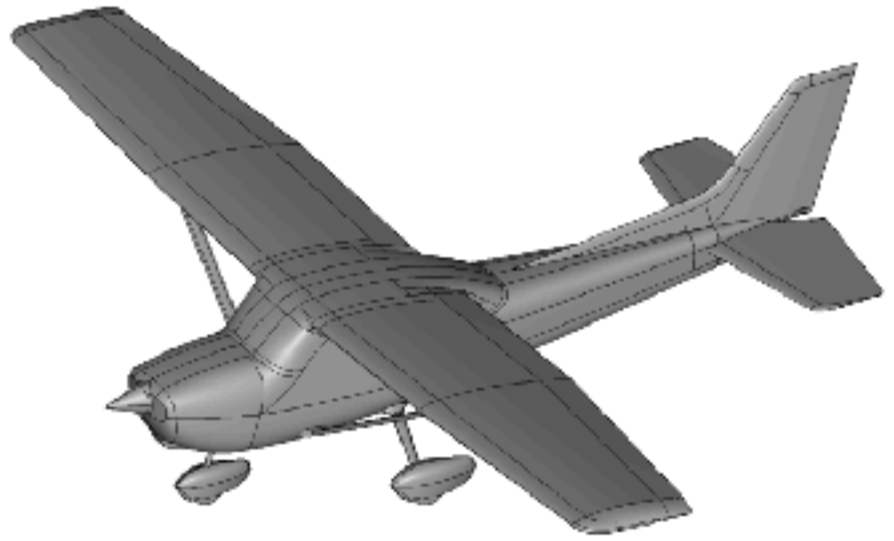


The Birth of a Digital Shape

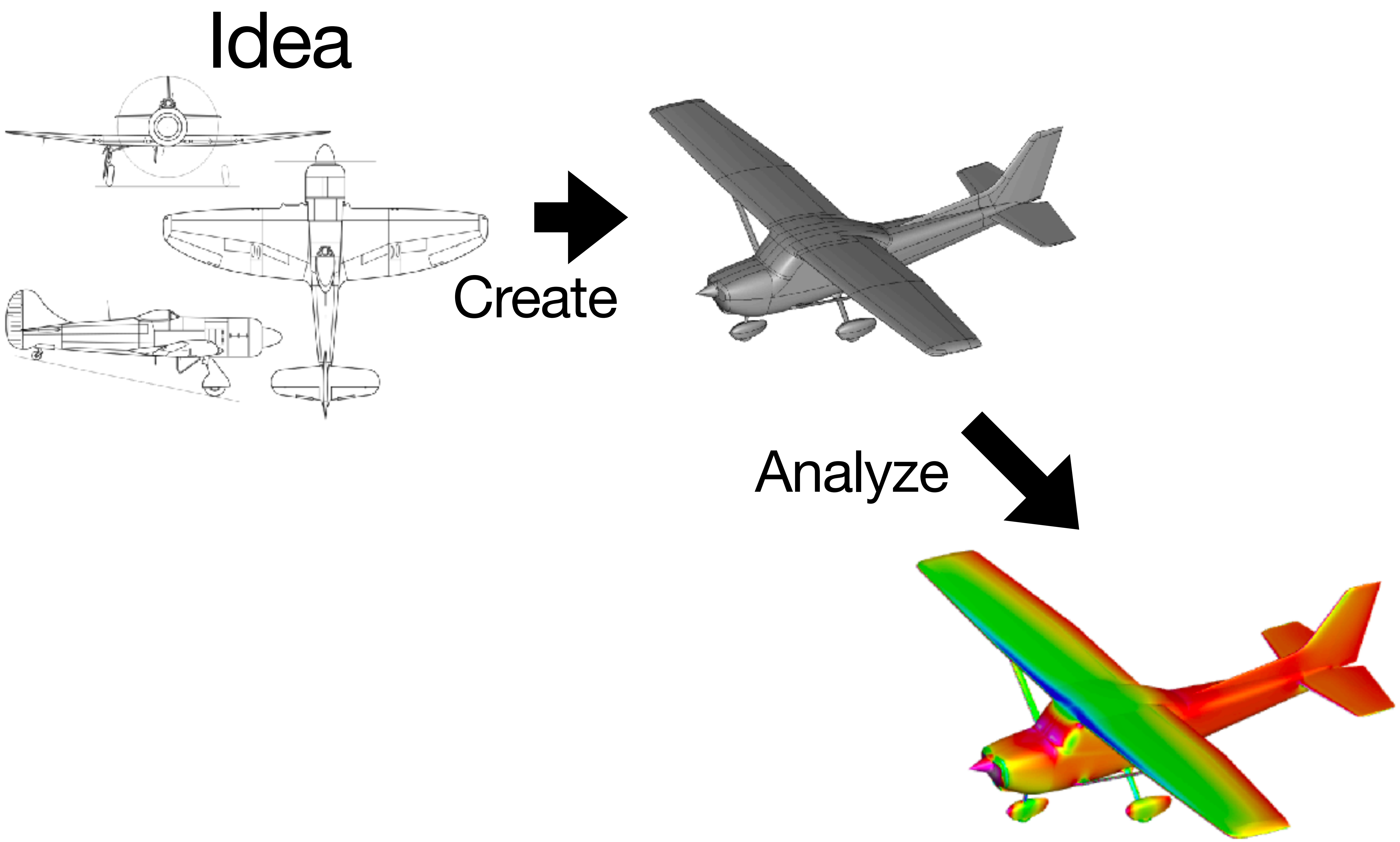
Idea



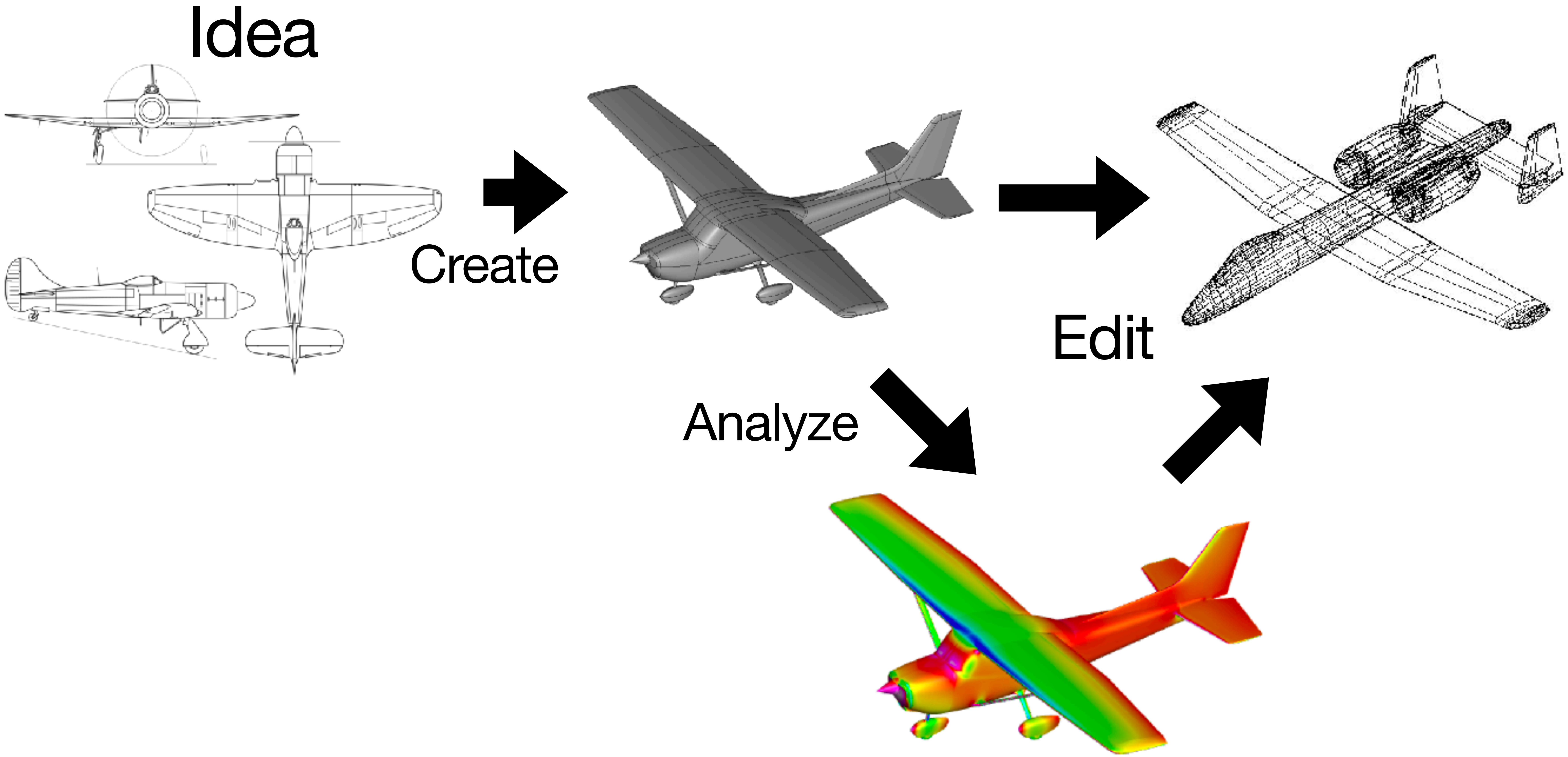
Create



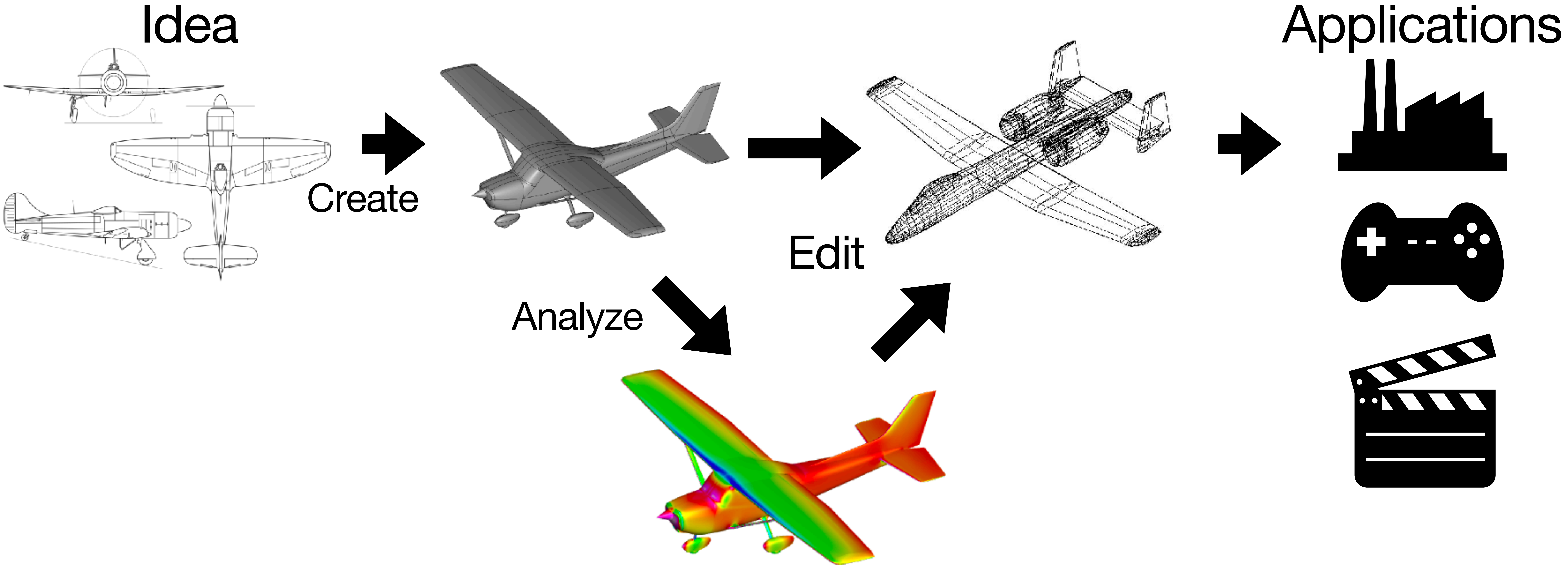
The Birth of a Digital Shape



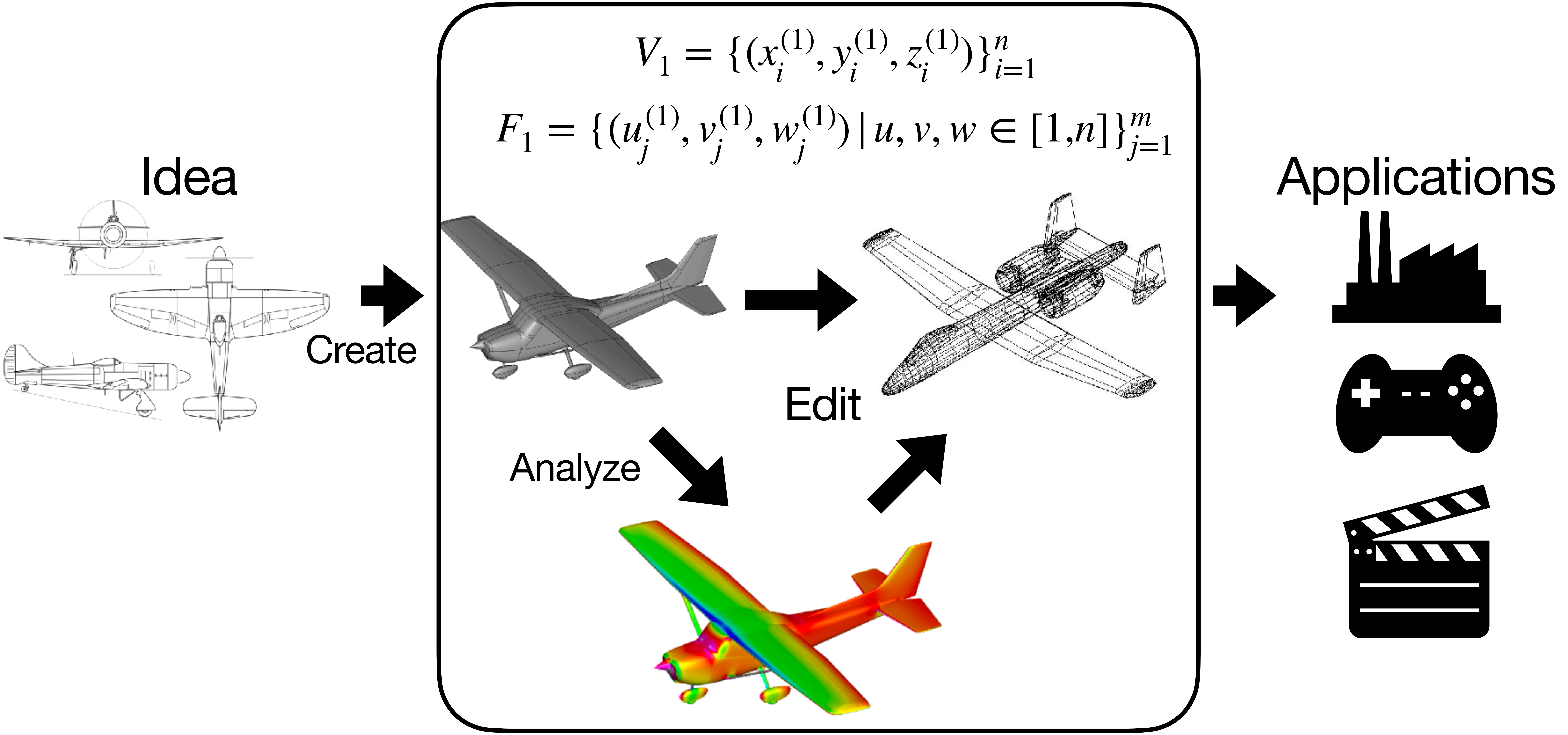
The Birth of a Digital Shape



The Birth of a Digital Shape

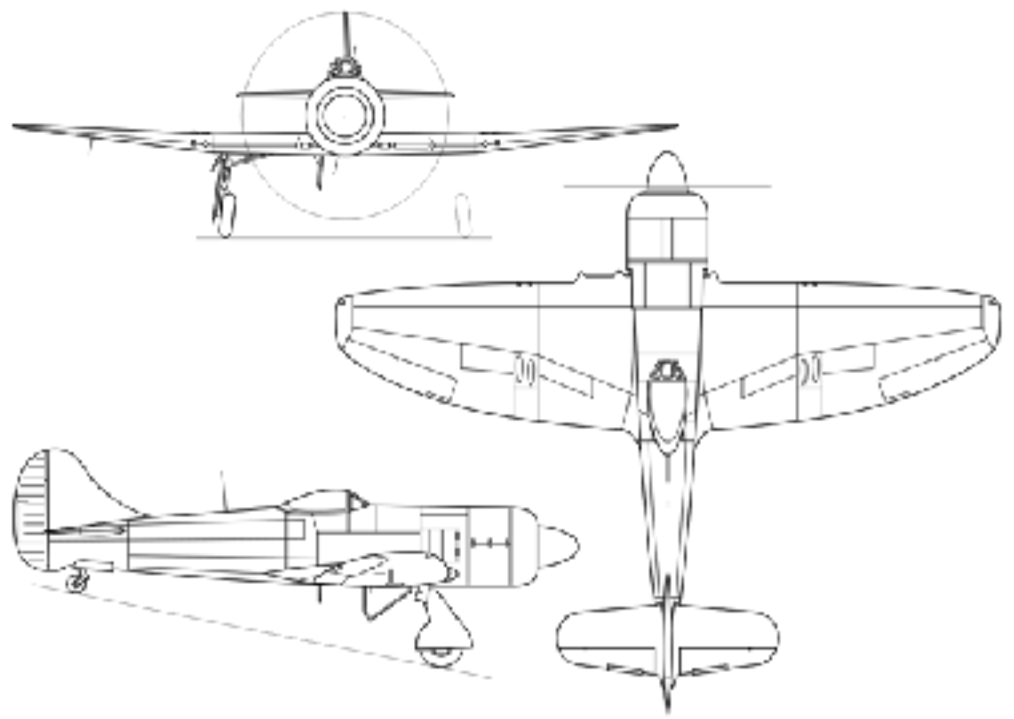


Geometry Processing is usually done with Mesh



Mesh Creation is Hard

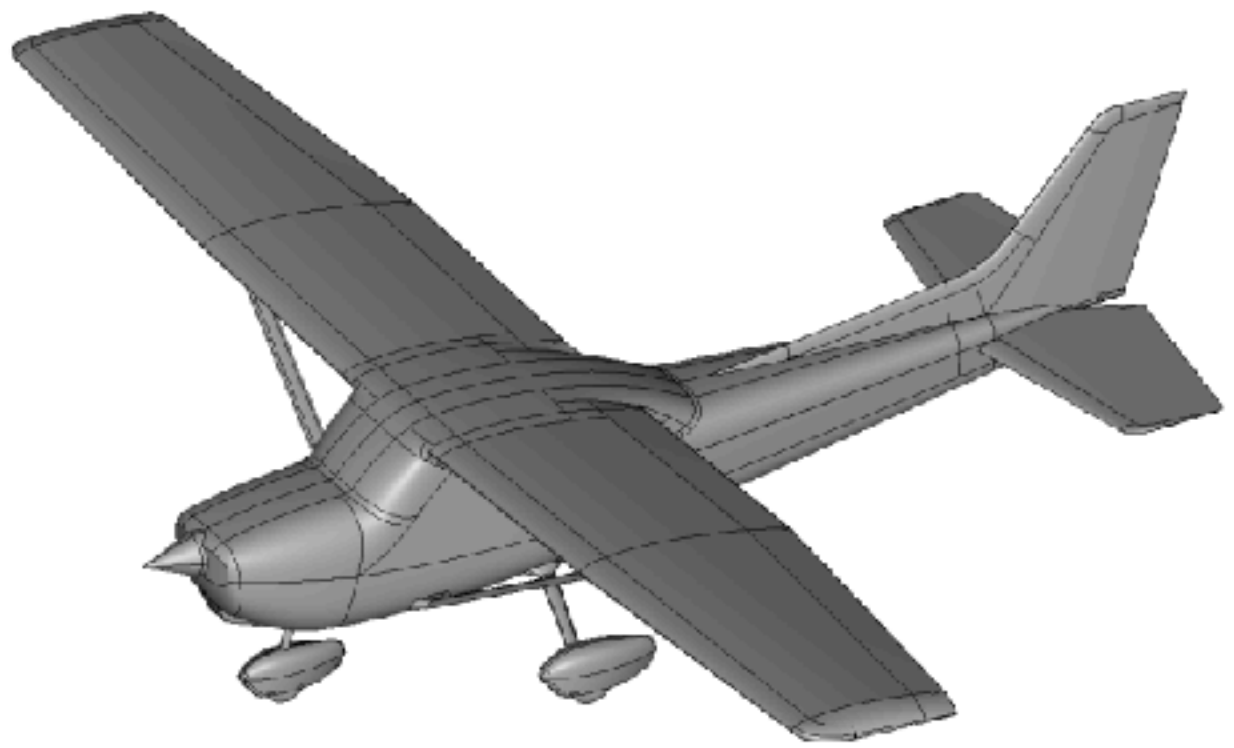
Point clouds,
Sketch,
Image, Ideas,
...

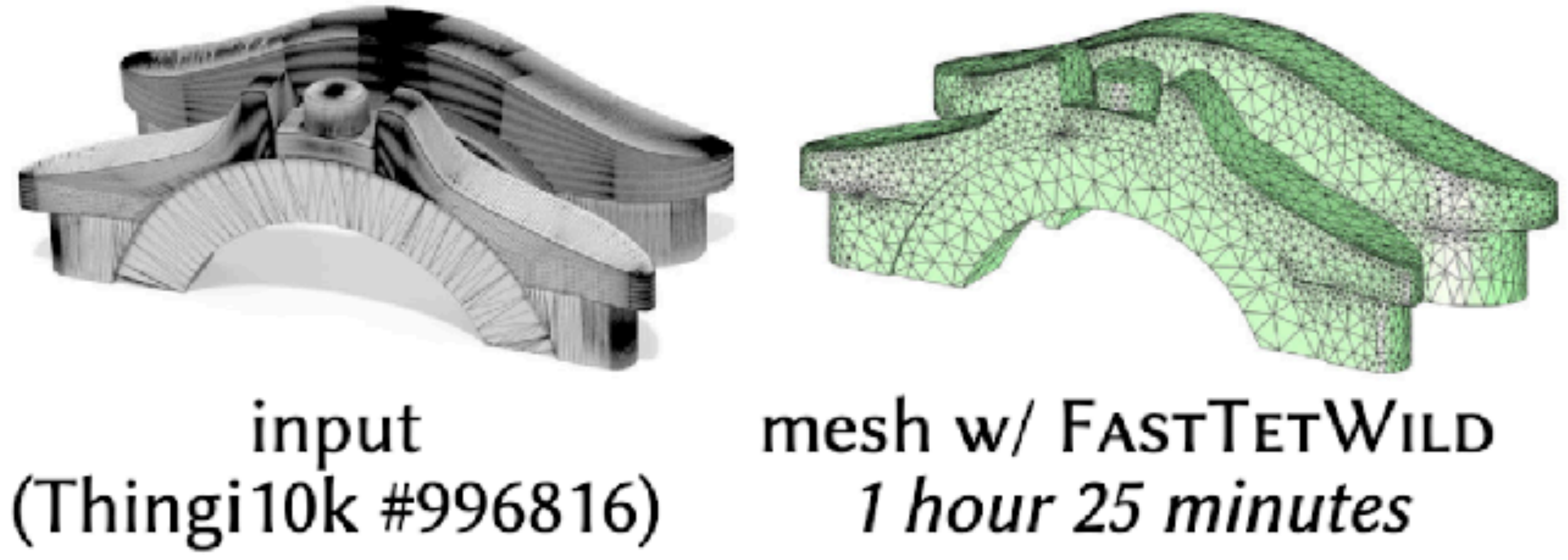


Create

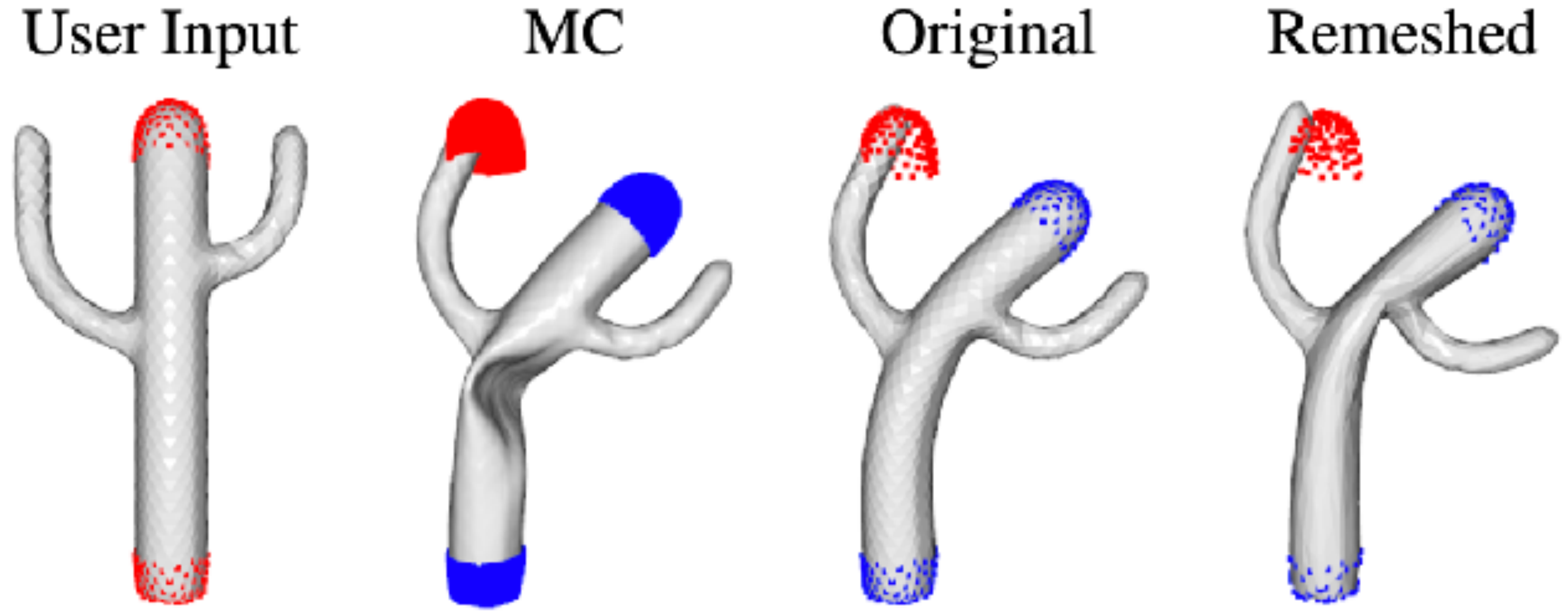
$$V_1 = \{(x_i^{(1)}, y_i^{(1)}, z_i^{(1)})\}_{i=1}^{n}$$

$$F_1 = \{(u_j^{(1)}, v_j^{(1)}, w_j^{(1)}) \,|\, u, v, w \in [1,n]\}_{j=1}^{m}$$





(Sawhney et. al., 2020, Sawhney et. al. 2022)



(Yang et. al., 2021)

"I hate meshes.
I cannot believe how hard this is.
Geometry is hard."

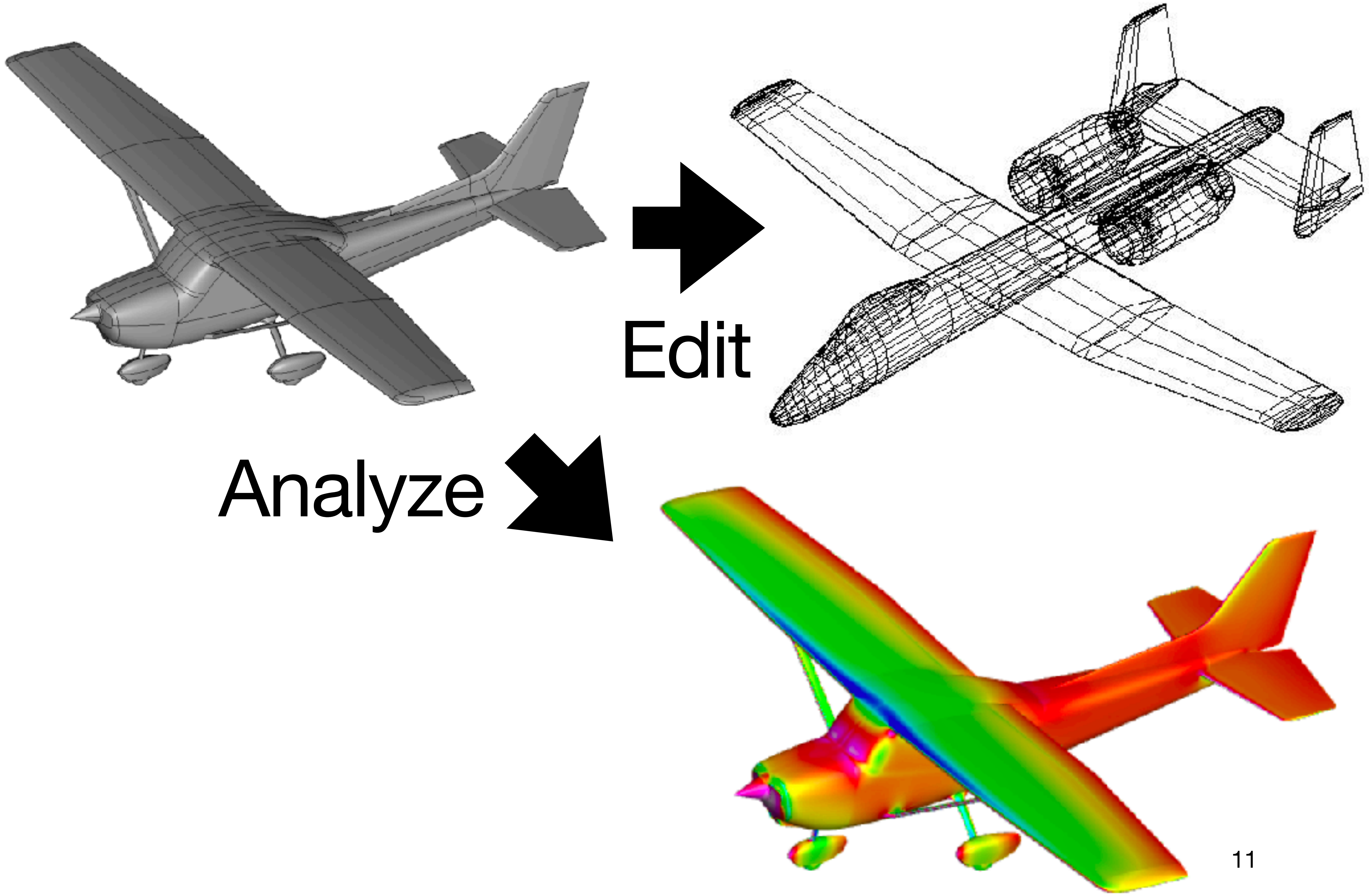
—David Baraff
Senior Research Scientist
Pixar Animation Studios

Credit to Prof. Keenan Crane, Monte Carlo Geometry Processing
https://www.youtube.com/watch?

Mesh Editing and Analysis are also Difficult

$$V_1 = \{(x_i^{(1)}, y_i^{(1)}, z_i^{(1)})\}_{i=1}^{n}$$

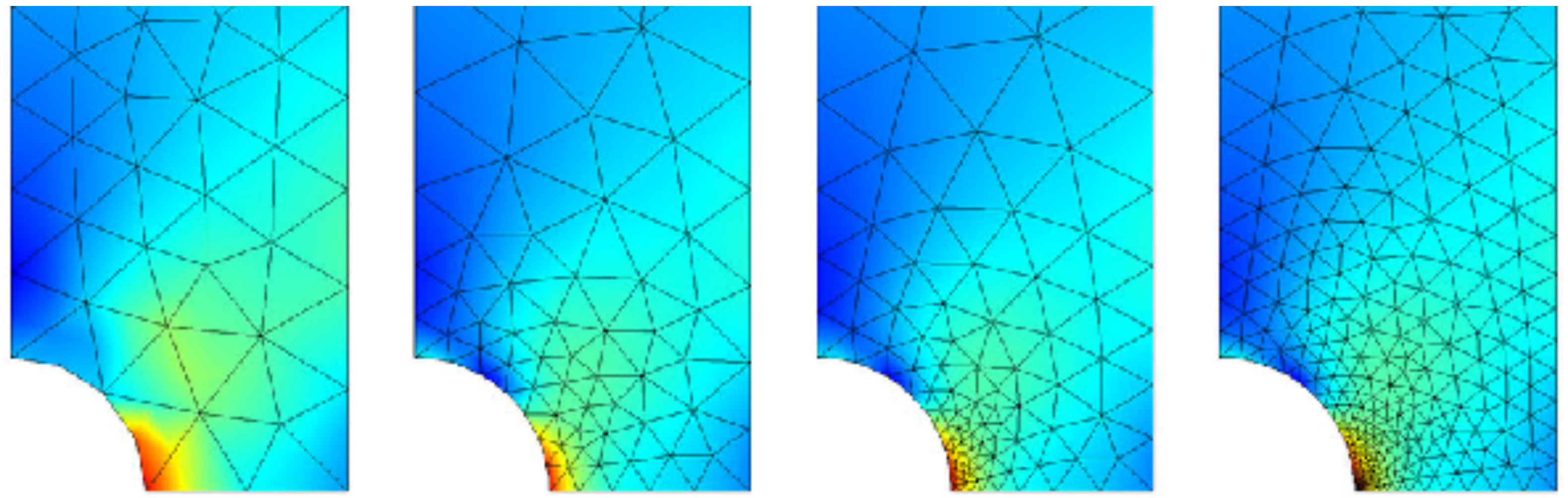
$$F_1 = \{(u_j^{(1)}, v_j^{(1)}, w_j^{(1)}) \,|\, u, v, w \in [1,n]\}_{j=1}^{m}$$

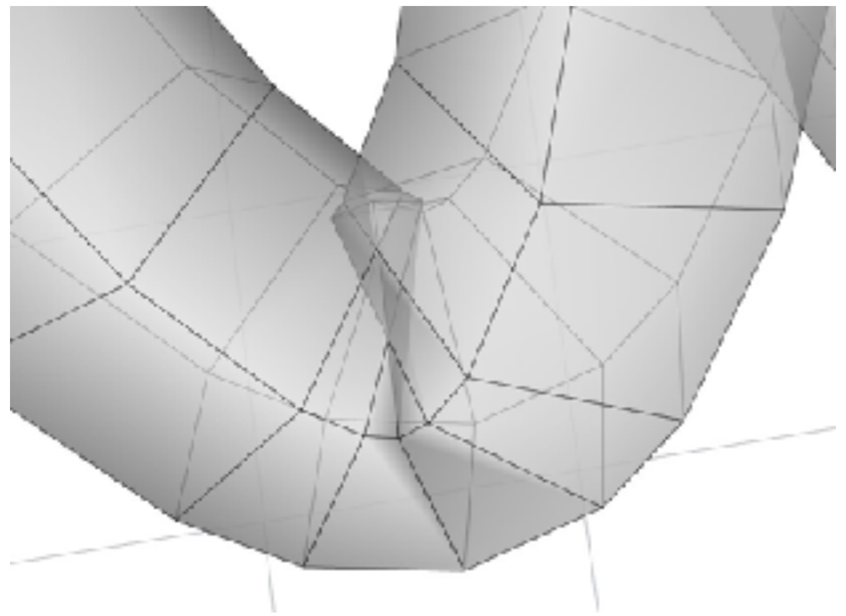


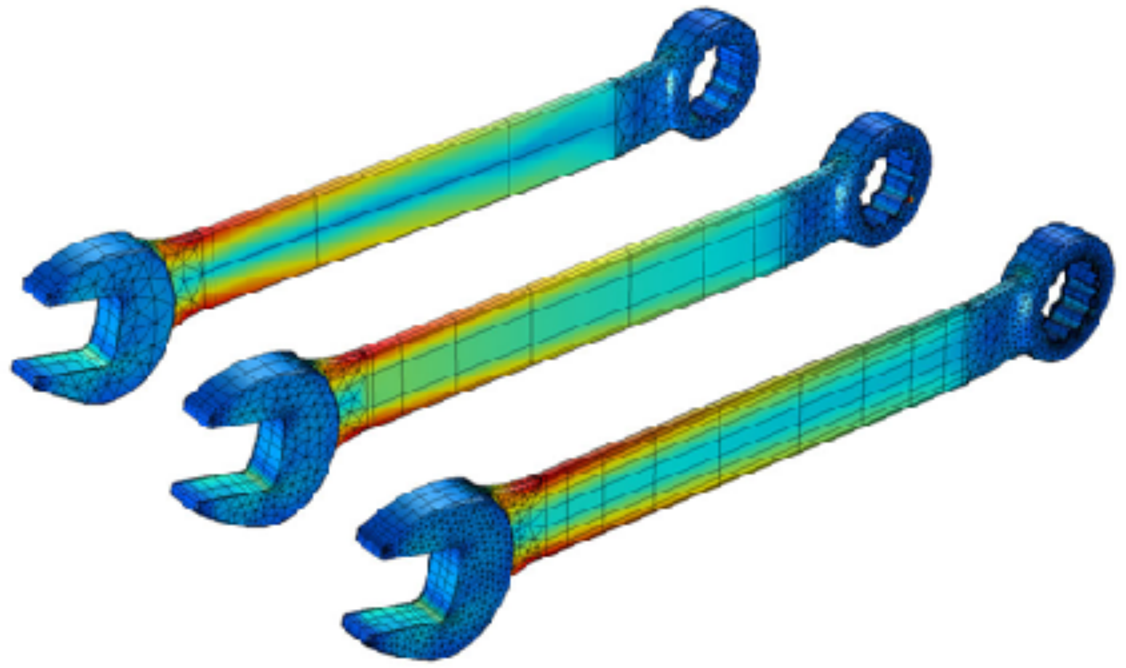
Topological changes

(Liu et. al., SIG 22')

Sensitive to discretization



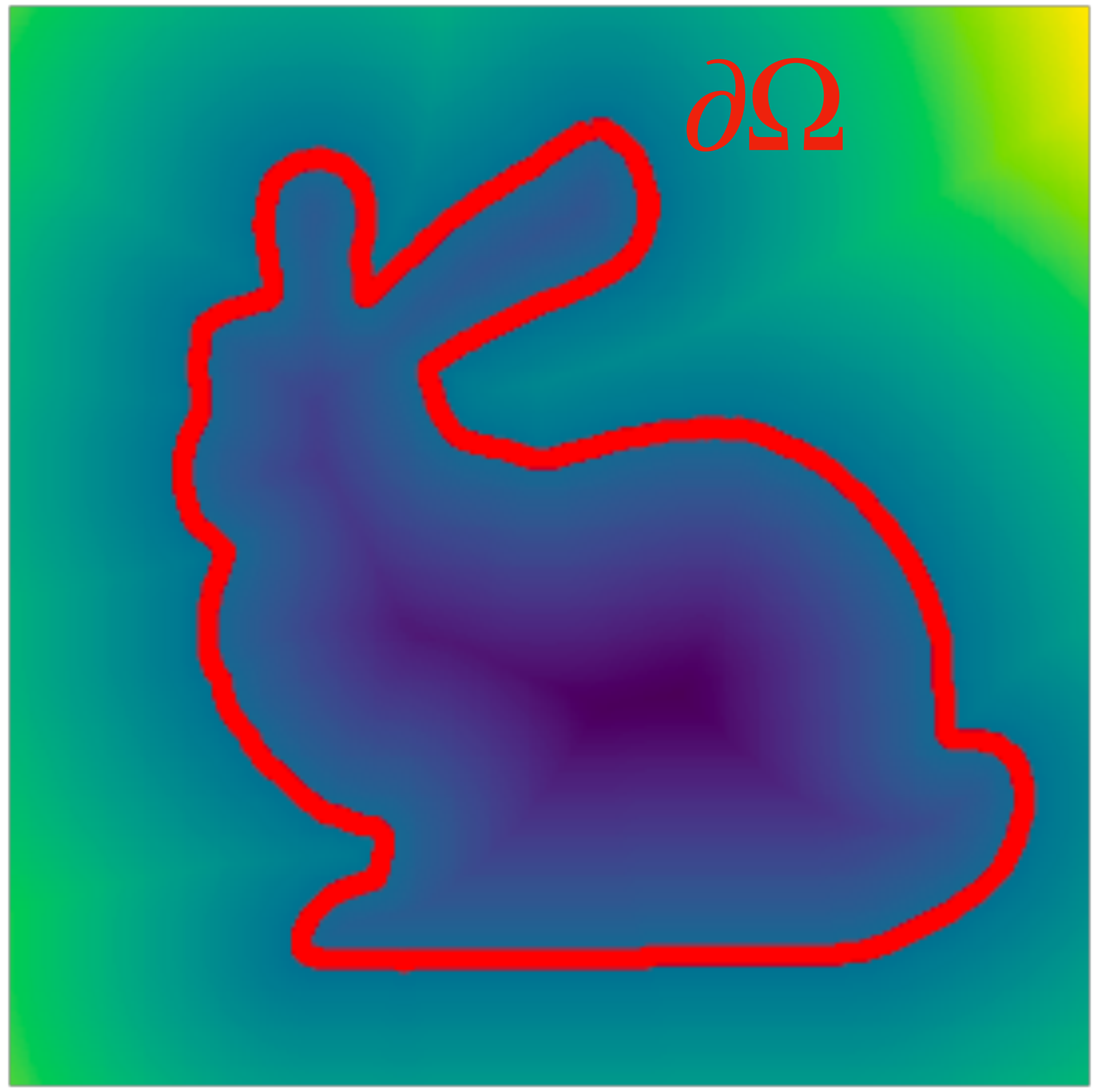




https://www.comsol.com/multiphysics/mesh-refinement

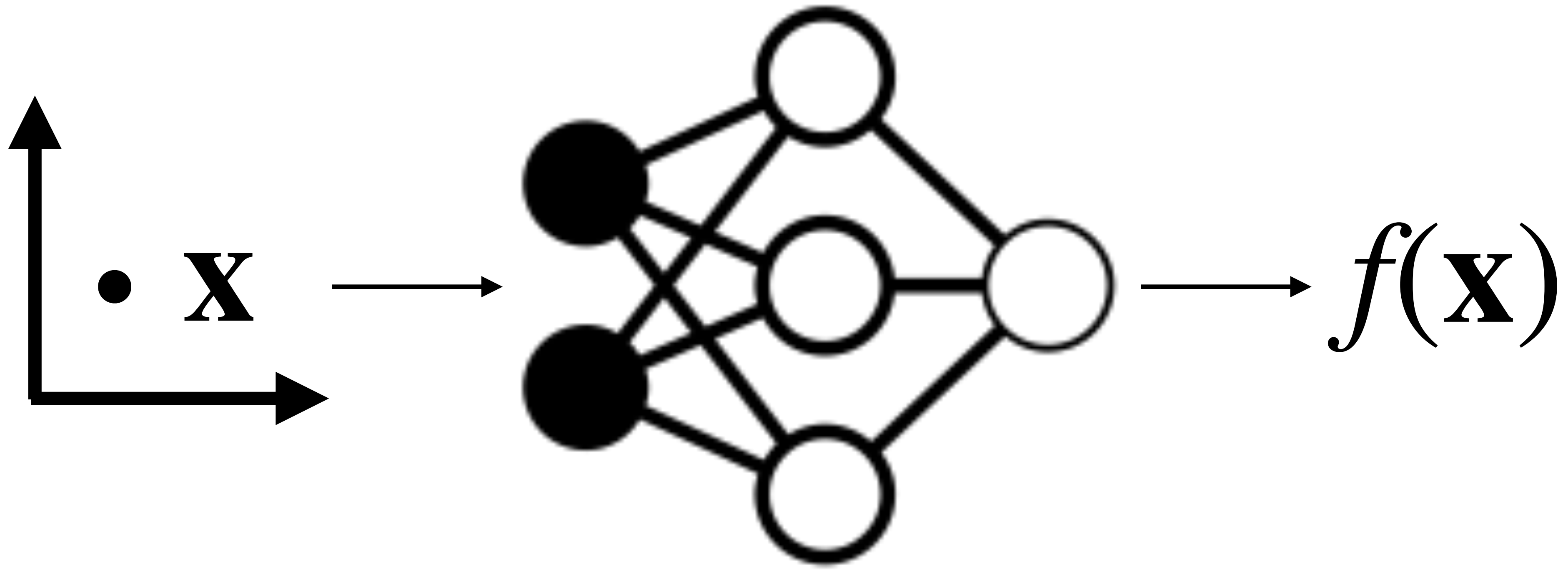
Can we use a different representation?

Neural Fields



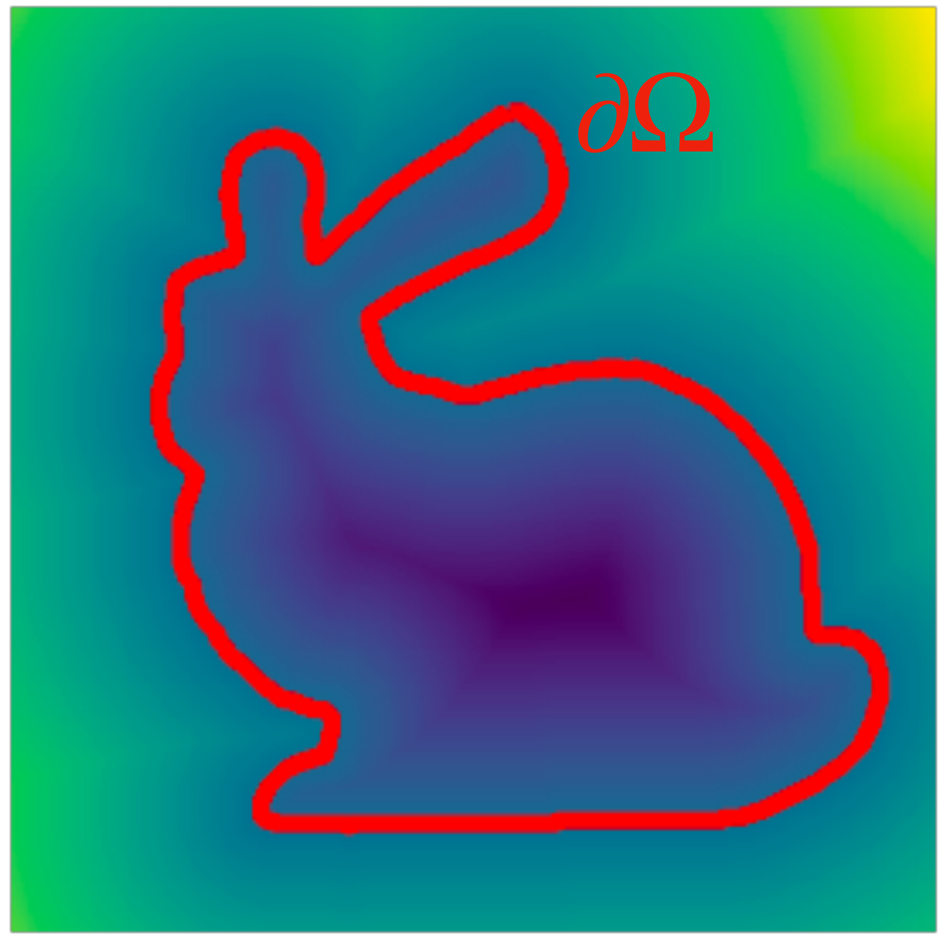
$$f : \mathbb{R}^3 \to \mathbb{R}$$

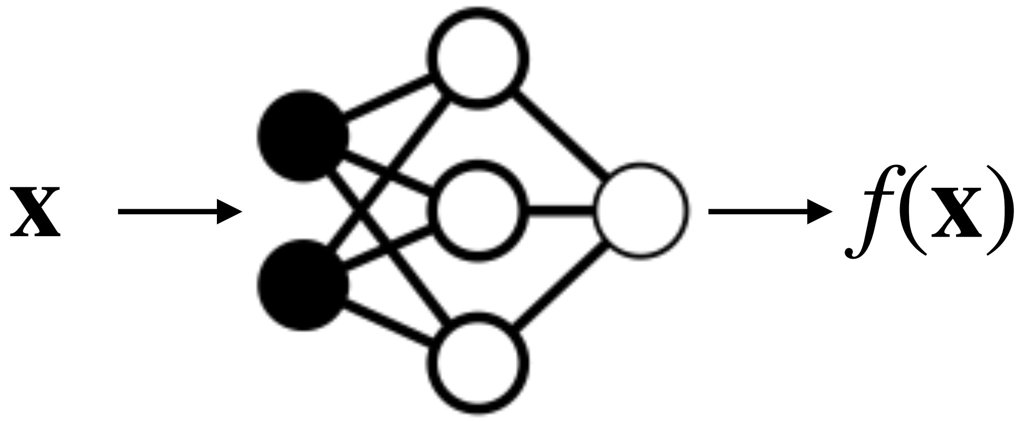
$$\partial\Omega = \{\mathbf{x} \mid f(\mathbf{x}) = c\}$$



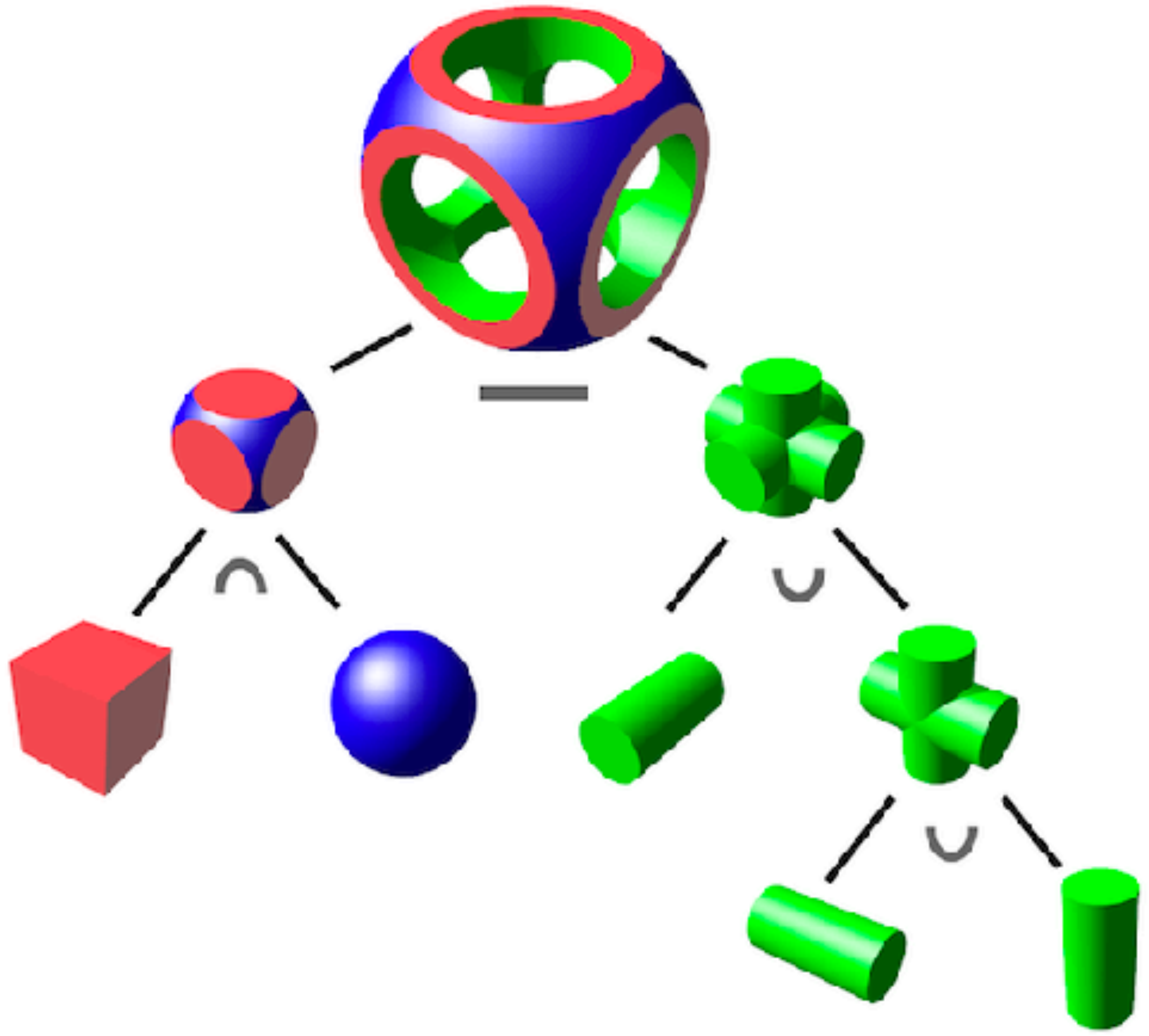
$$f_\theta(x) : \mathbb{R}^3 \to \mathbb{R}$$

Neural Fields





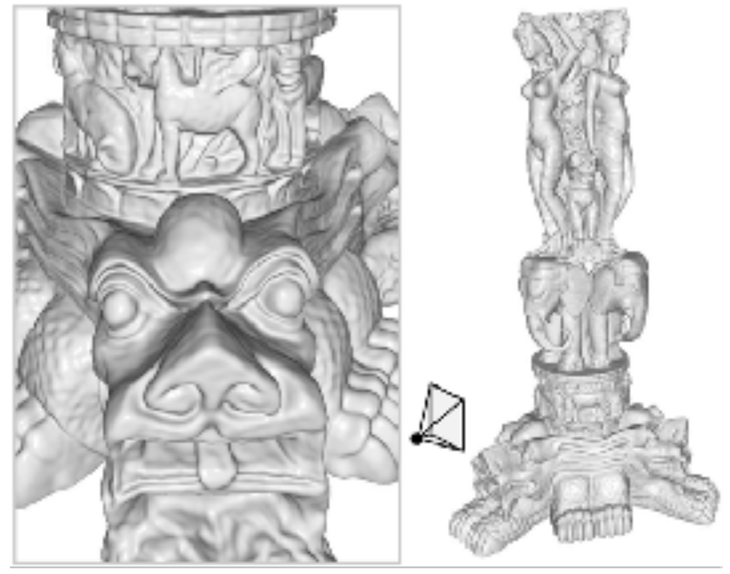
Neural Fields



Topological changes

Compact

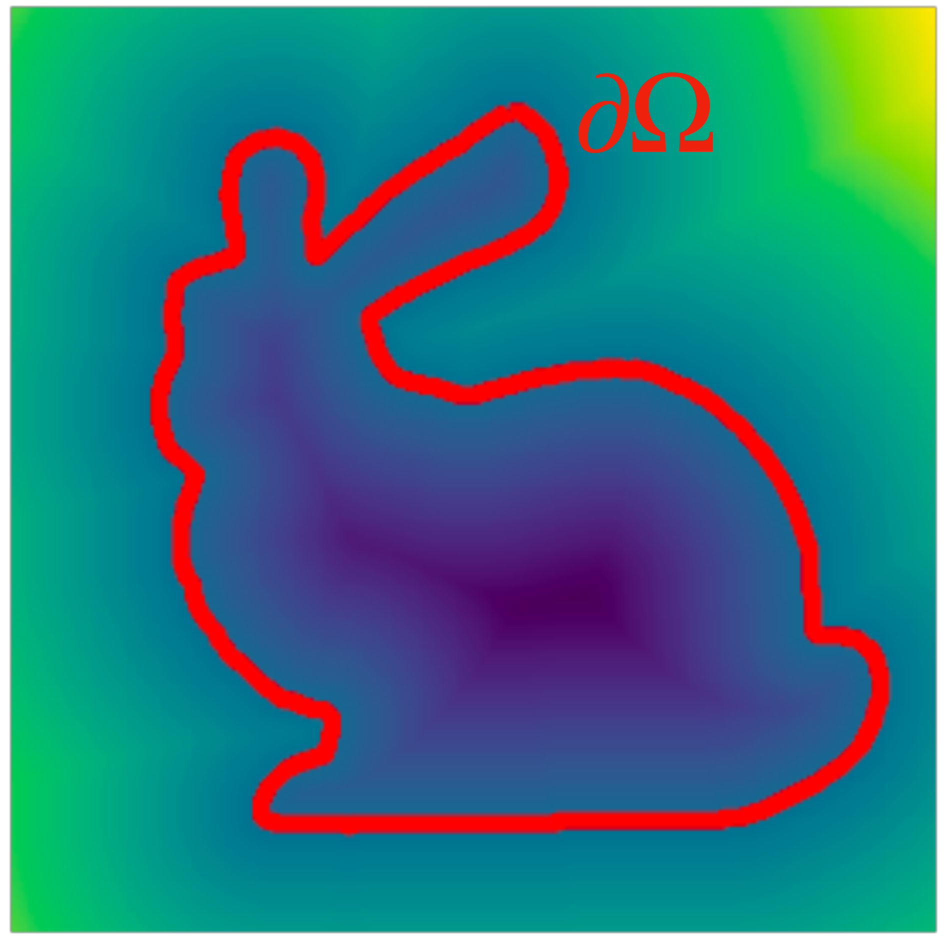
(Davies et al., 2020, Martel et al., 2021, Mescheder et al., 2021)

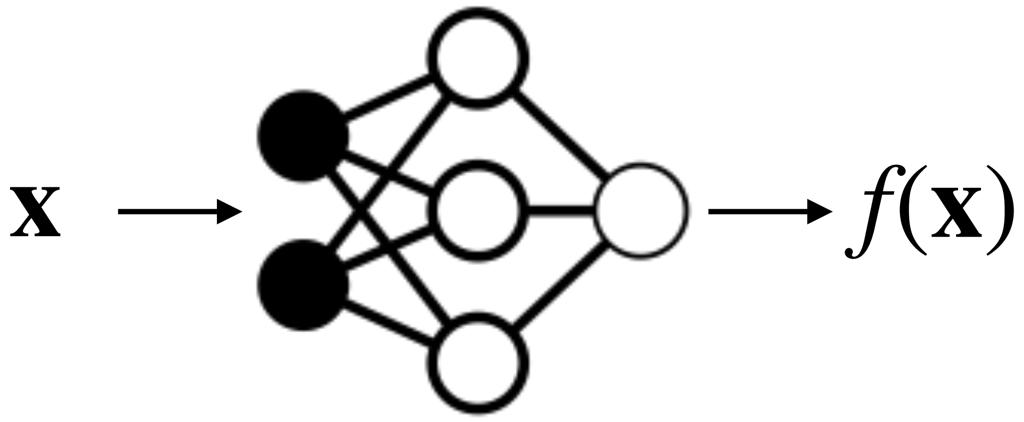


High fidelity

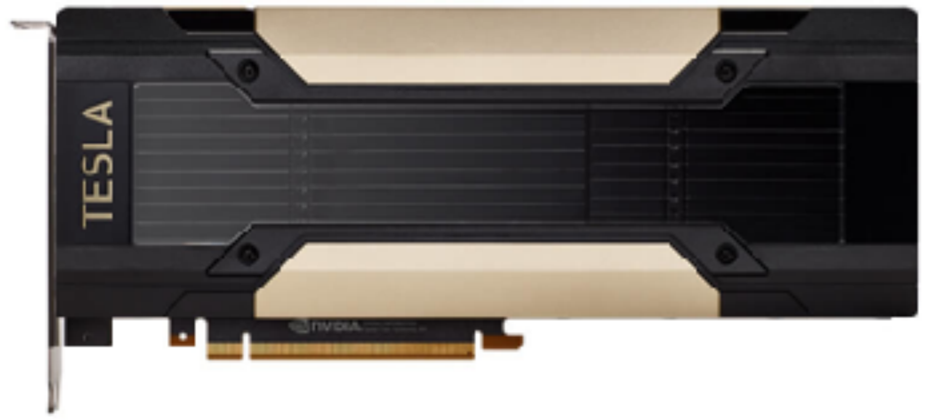
(Sitzmann et al., 2020, Martel et al., 2021, Park et al., 2019)

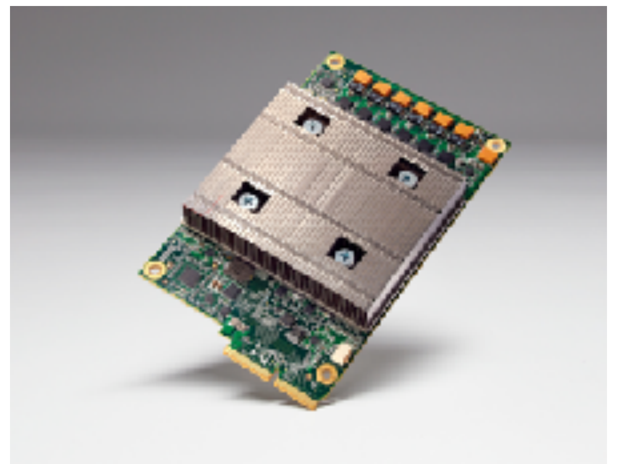
Neural Fields





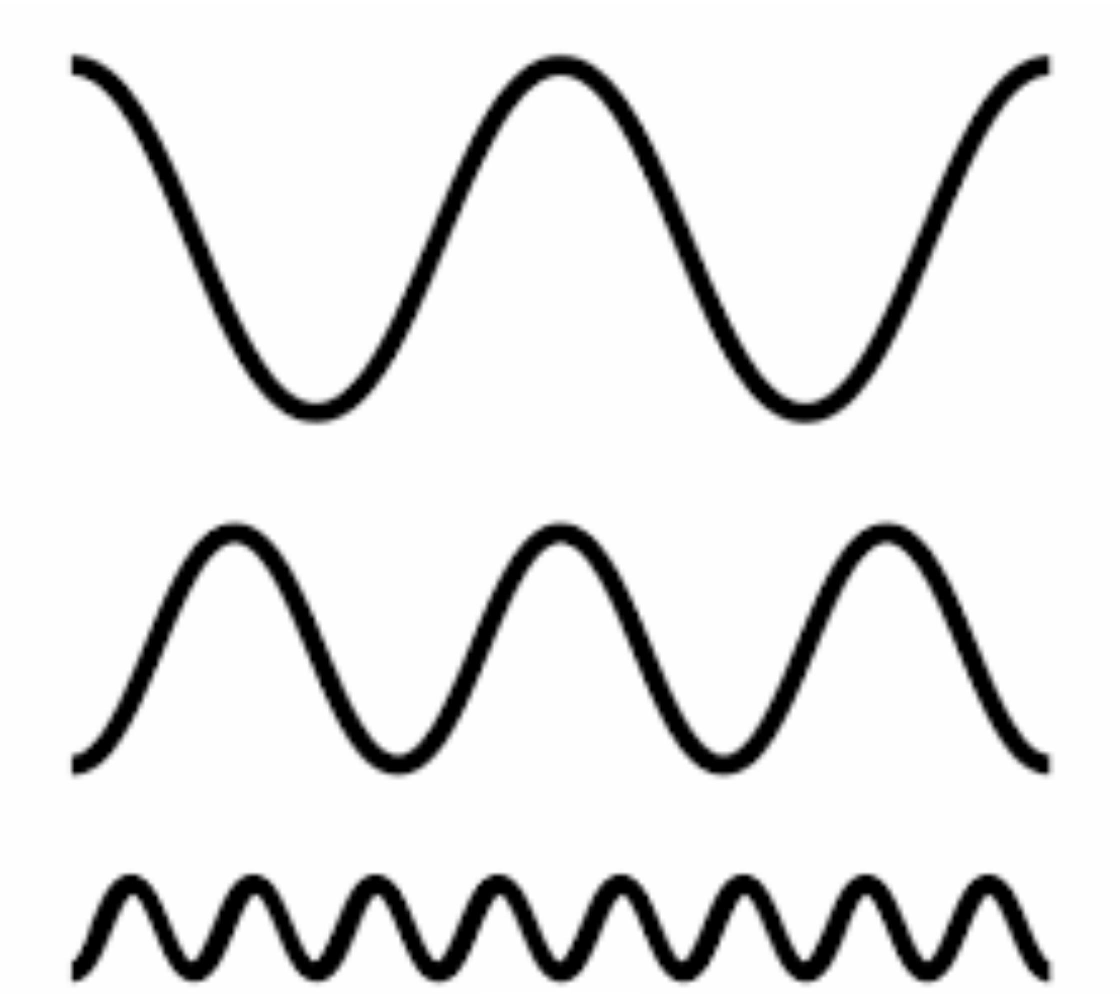
Neural Fields





Easy to optimize
(with Deep Learning Frameworks)

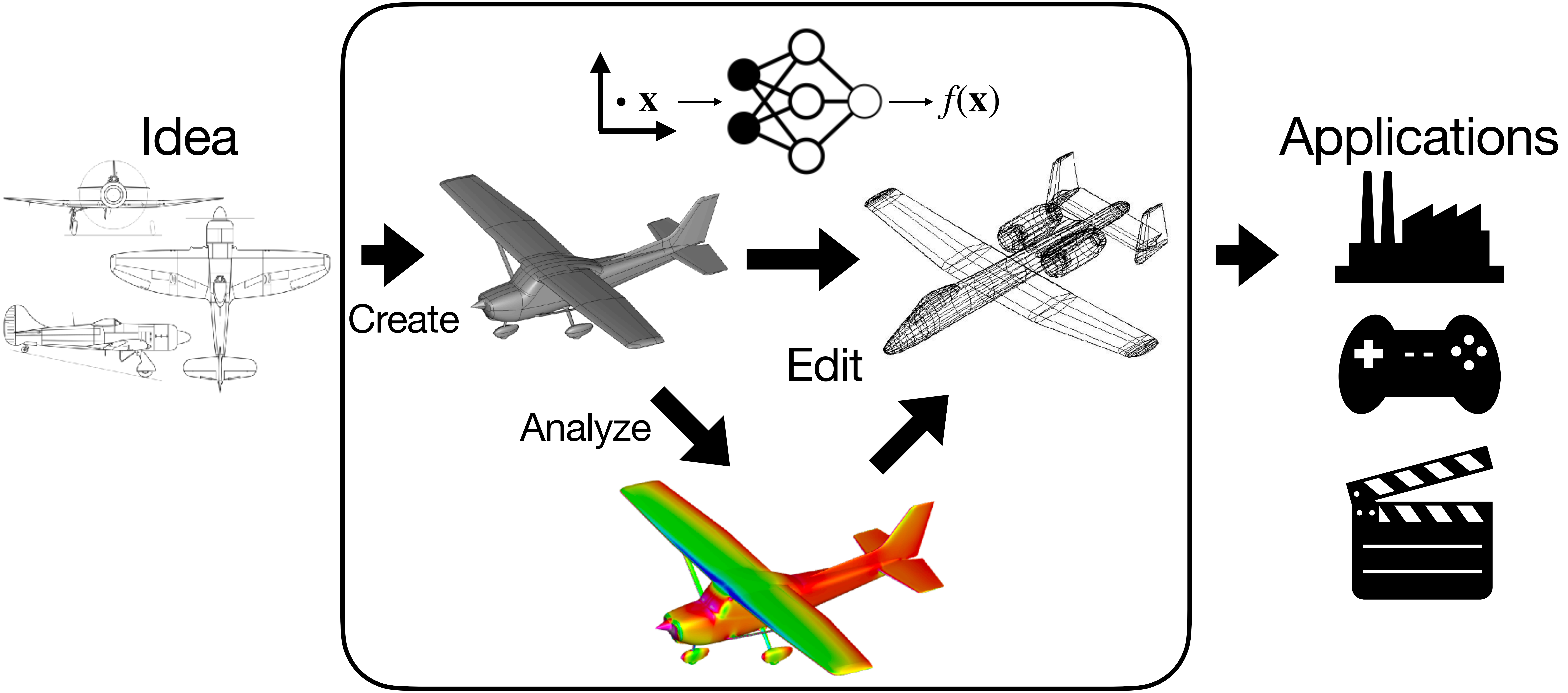
(Sitzmann et al., 2020,
Lindel et al., 2021, Tancik et al., 2022)



Continuous, avoid explicit discretization, and easy access to gradient

(Yang et al., 2022)

Geometry Processing with Neural Fields

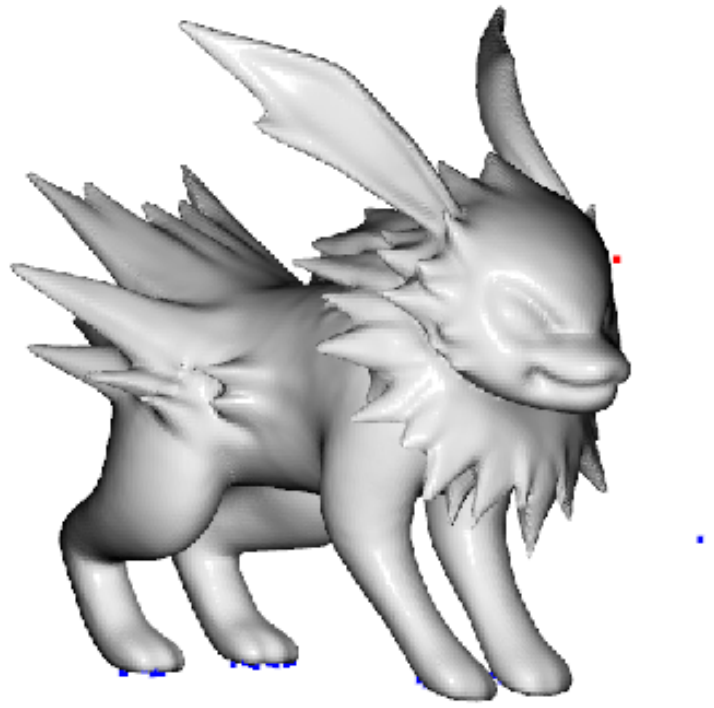


Today's Agenda

Synthesis

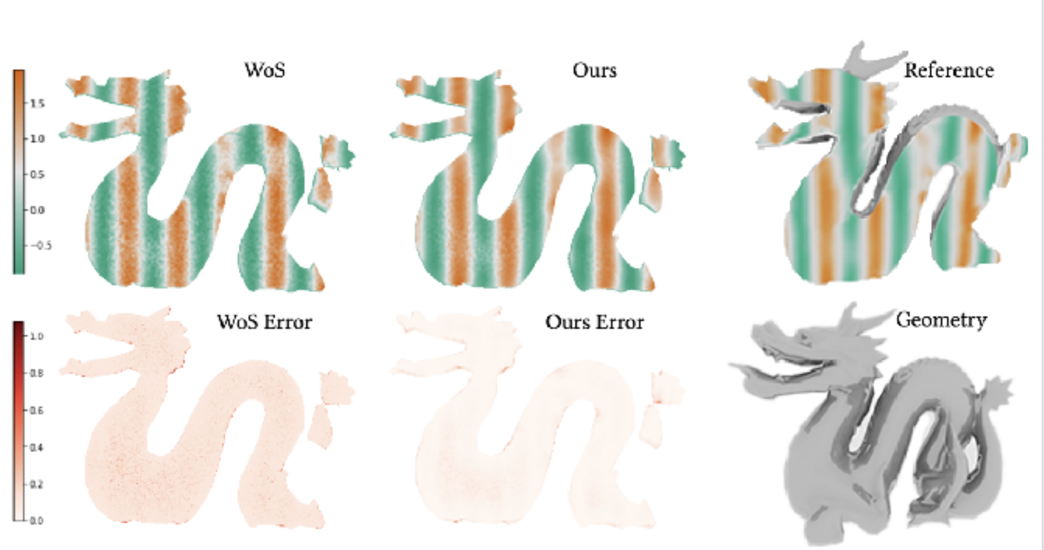
PointFlow (ICCV 2019)

ShapeGF (ECCV 2020)

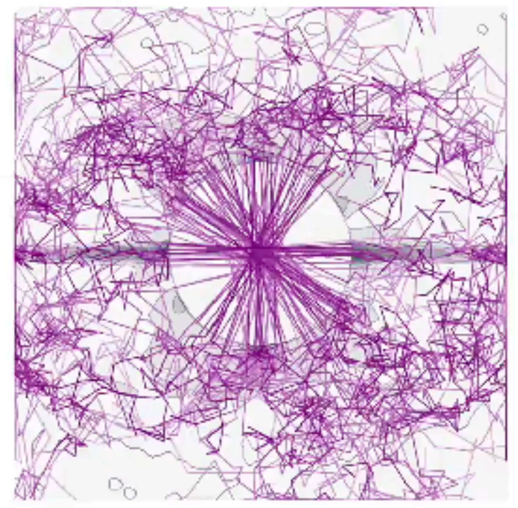


NFGP (NeurIPS 2021)

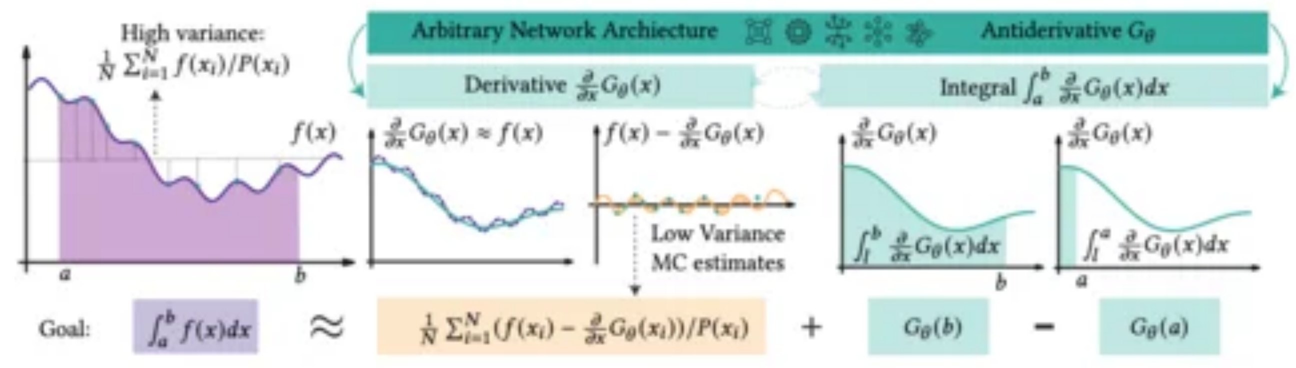
Analysis



Neural cache (SIG Asia 2023)



Symmetry (SIG Asia 2024)



NCV (SIGRAPH 2024)

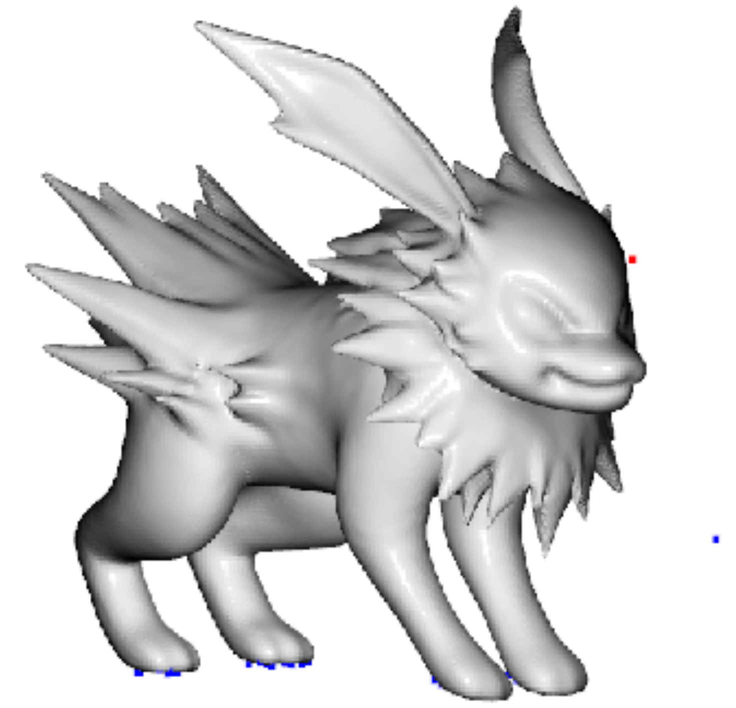
2019 2020 2021 2023 2024

Synthesis - Generation

Synthesis

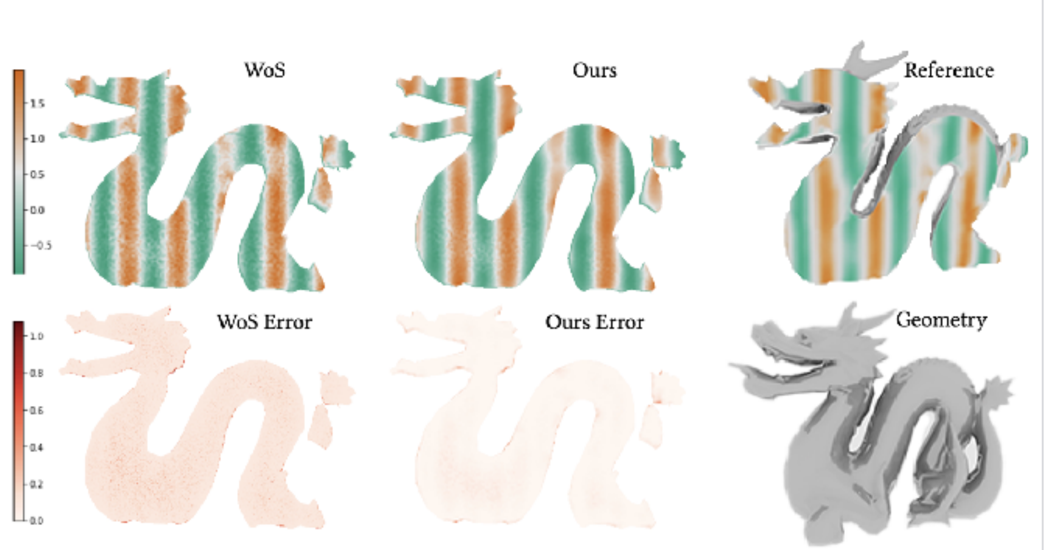
PointFlow (ICCV 2019)

ShapeGF (ECCV 2020)

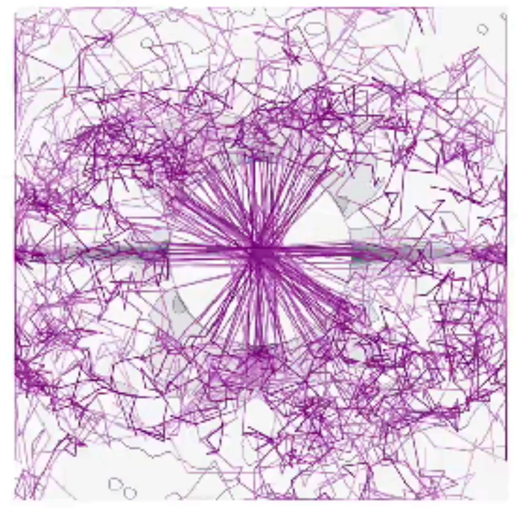


NFGP (NeurIPS 2021)

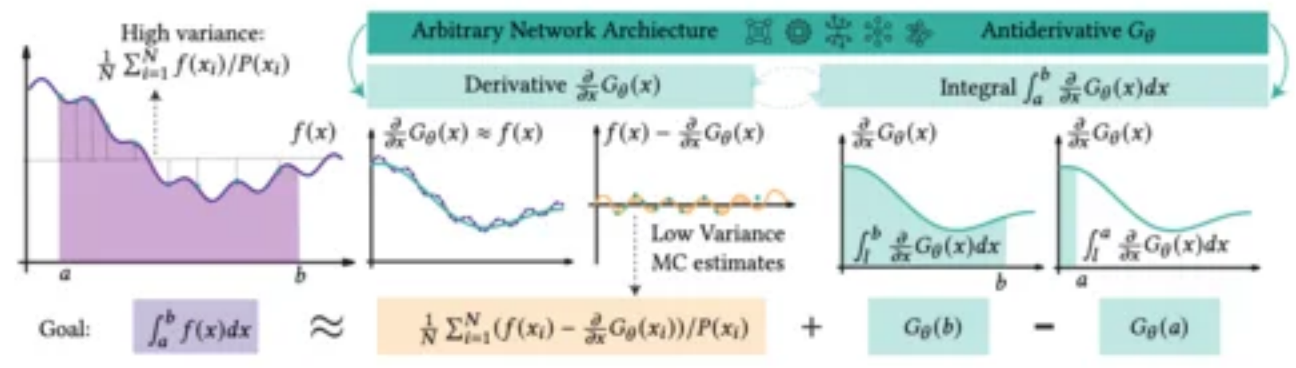
Analysis



Neural cache (SIG Asia 2023)



Symmetry (SIG Asia 2024)



NCV (SIGRAPH 2024)

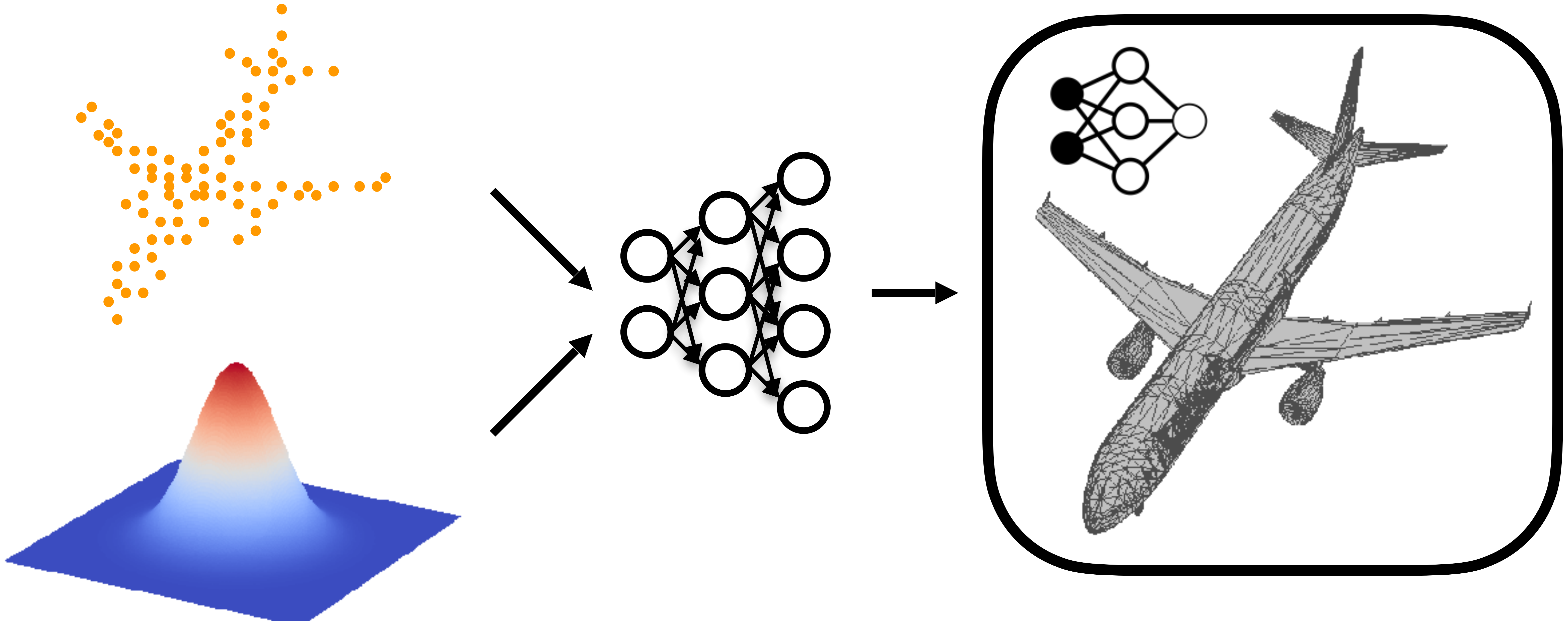
2019 2020 2021 2023 2024

Problem Set-up: Shape Generation

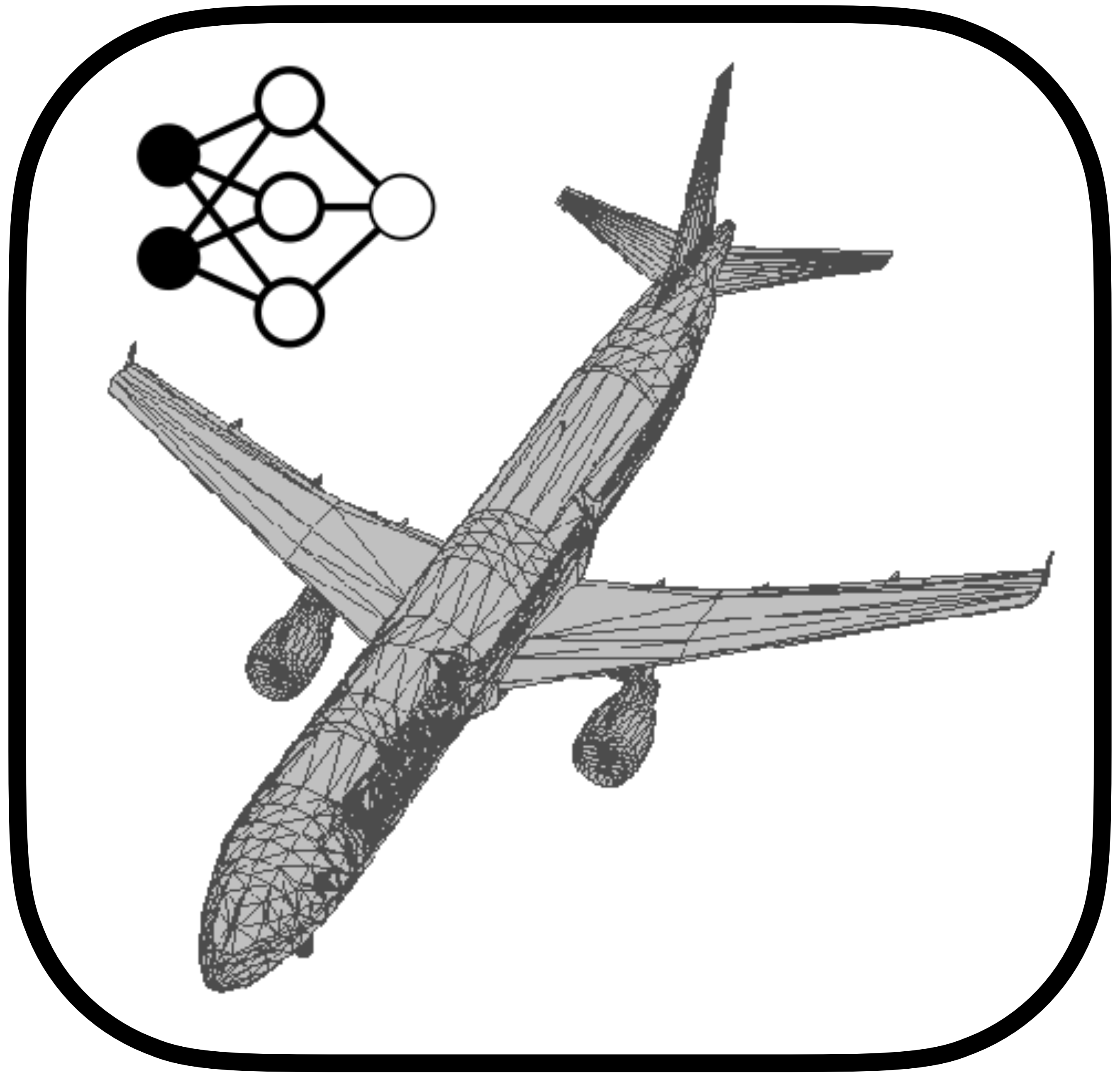
Training

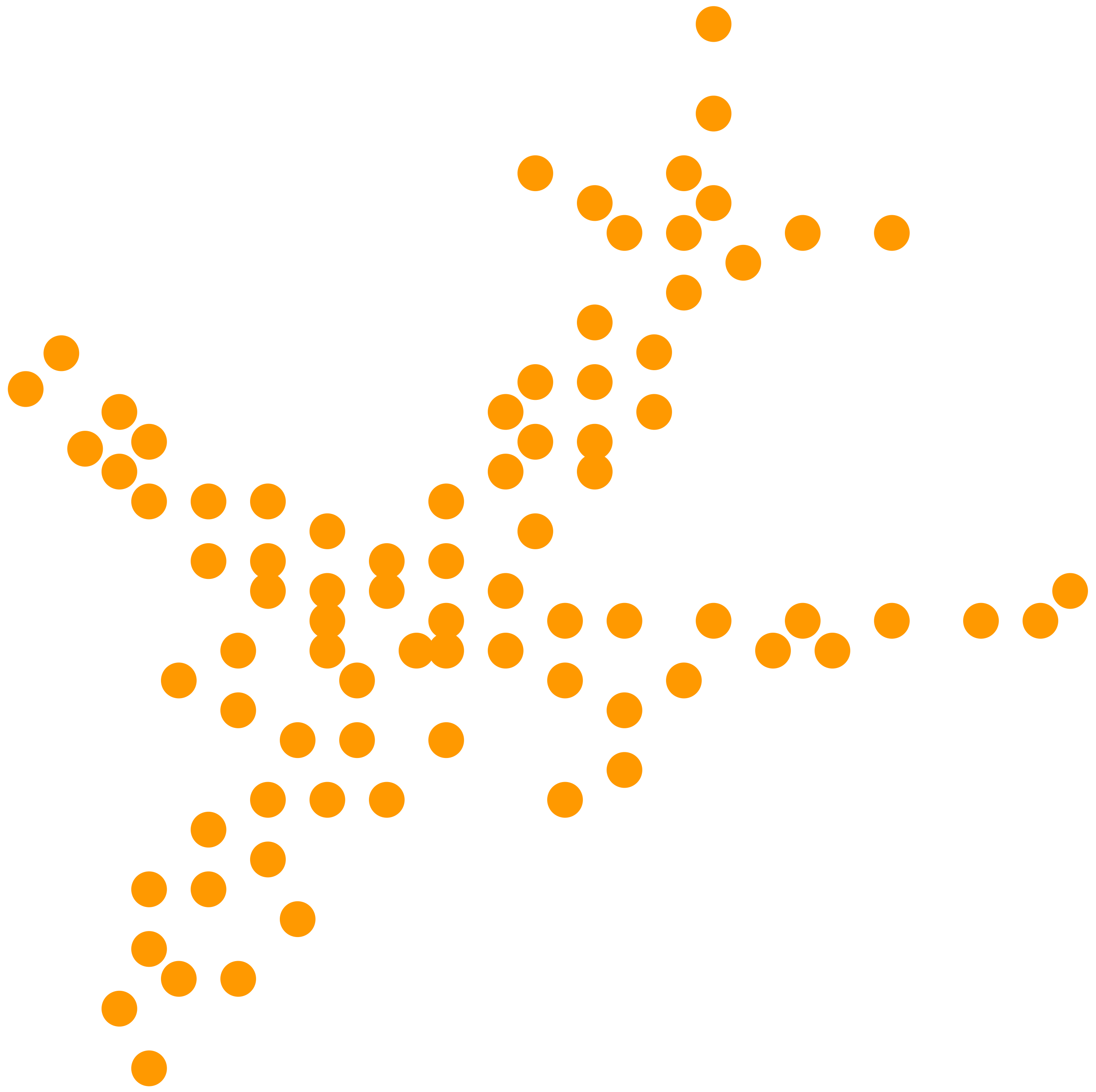
(A collection of 3D point clouds)

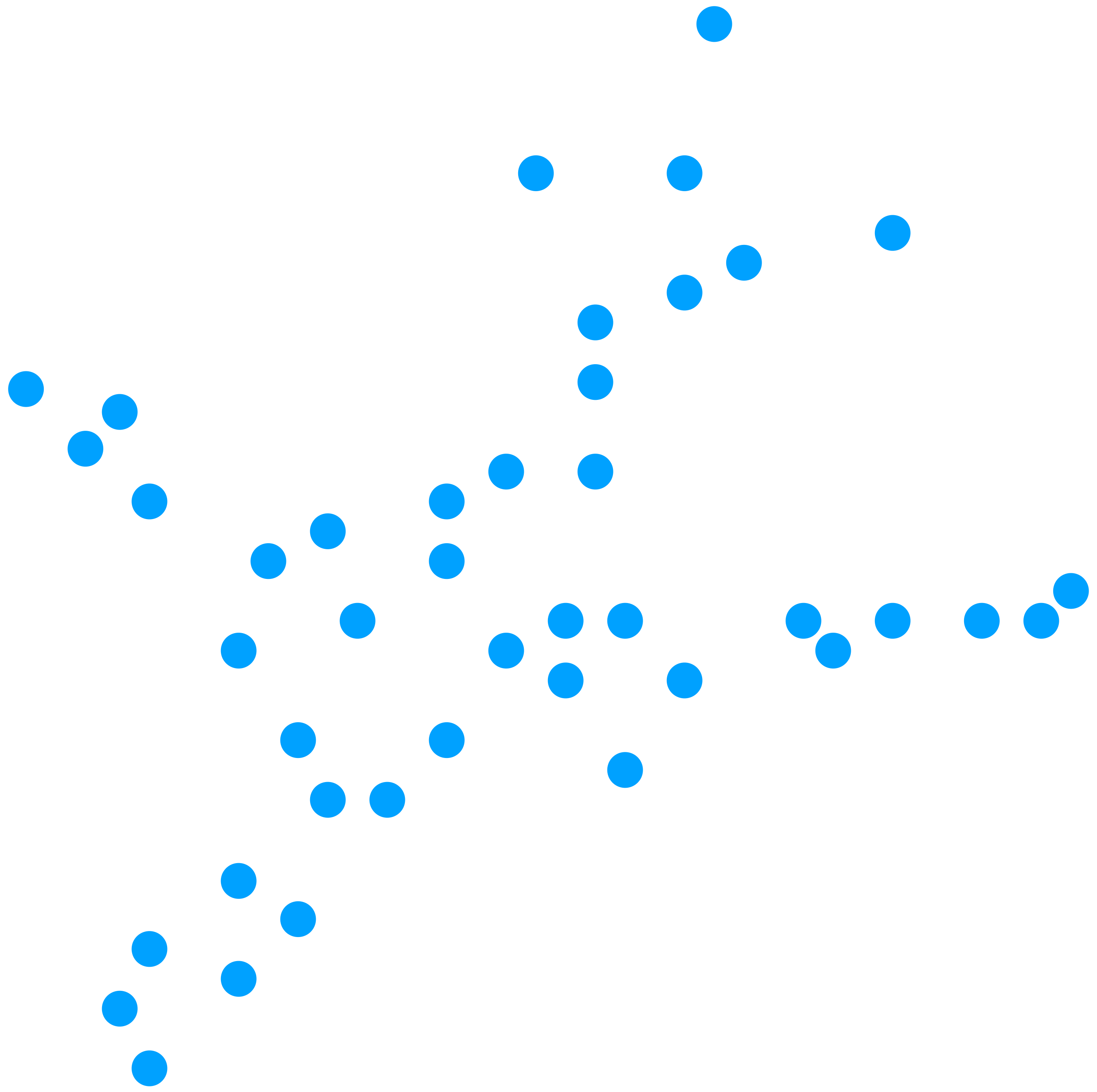
Testing



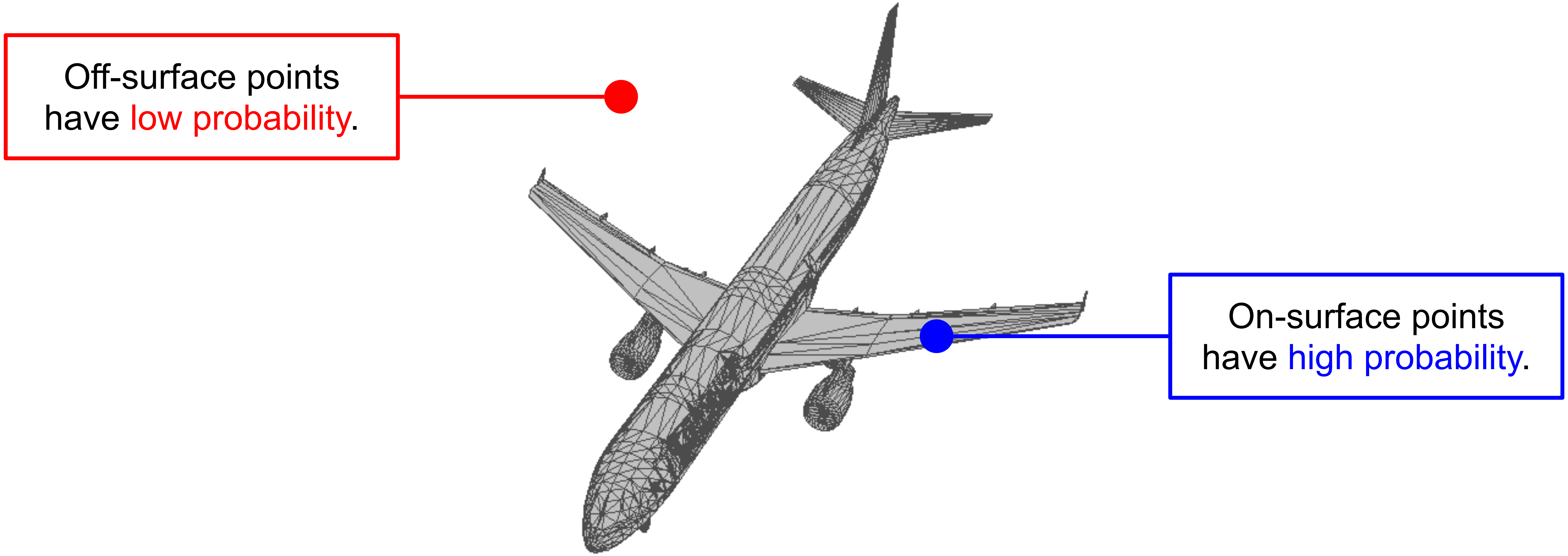
Representation for Arbitrary Size Point Clouds





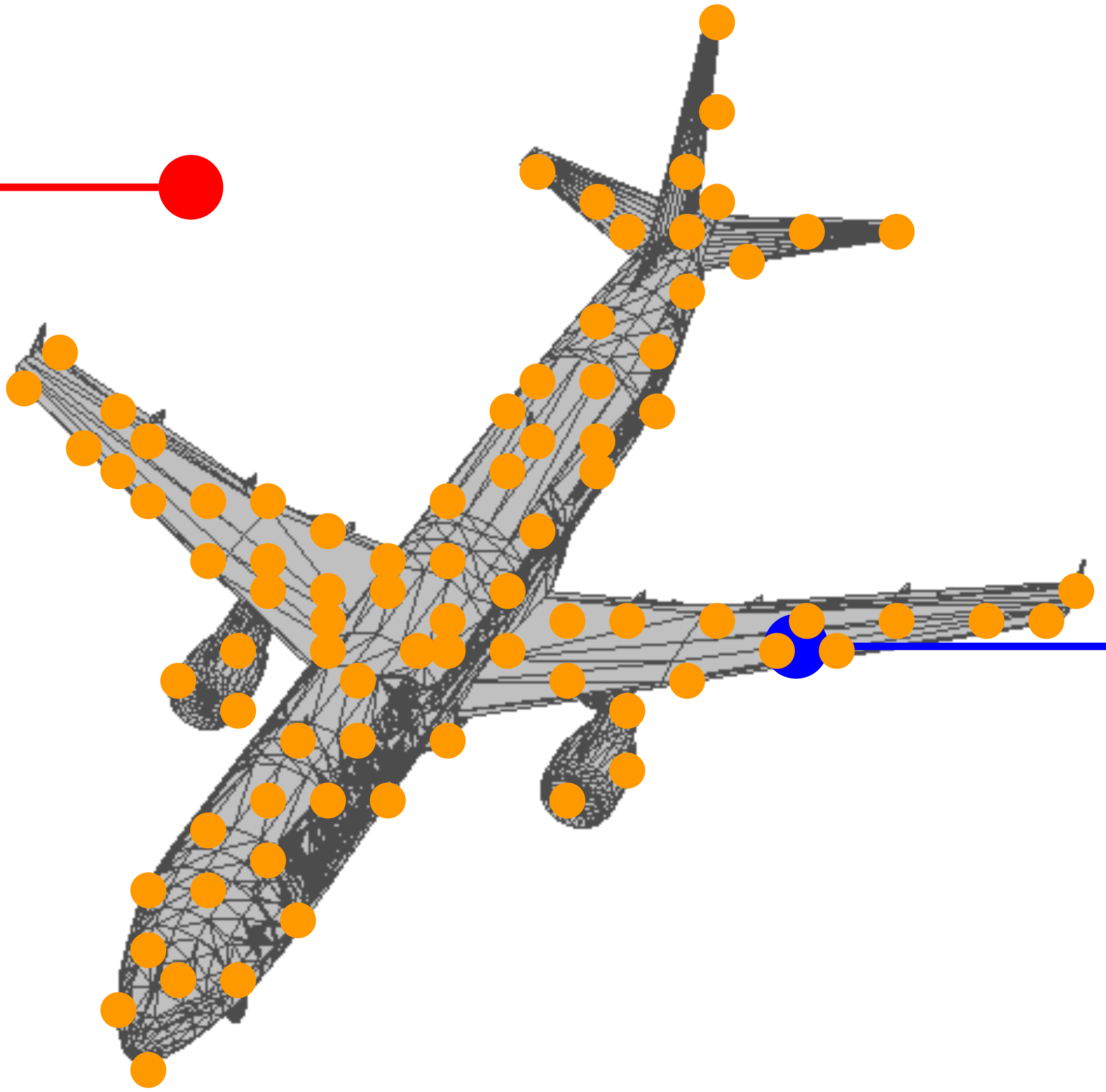


Each shape is a distribution of 3D points.
(i.e. shape as a 3D density field)



Point cloud is a sampled from such distribution

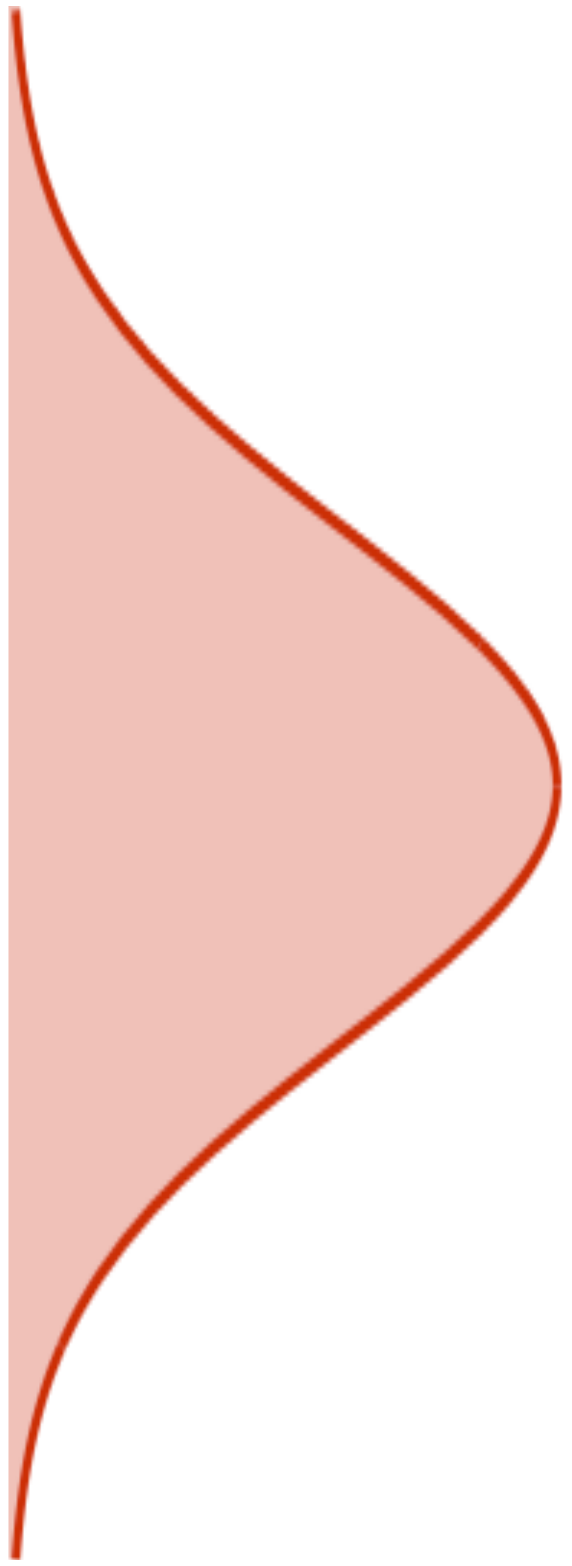
Off-surface points have low probability.

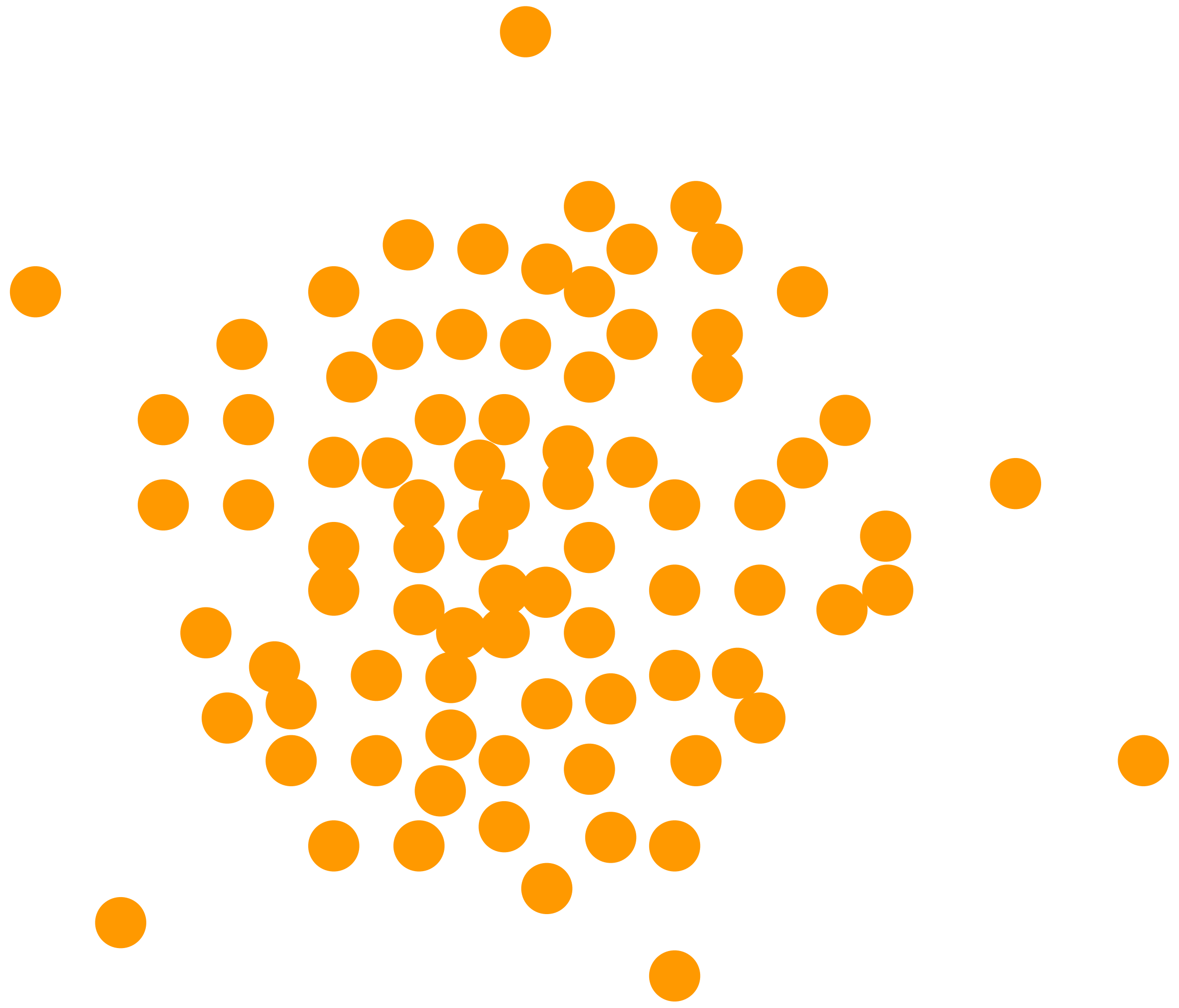


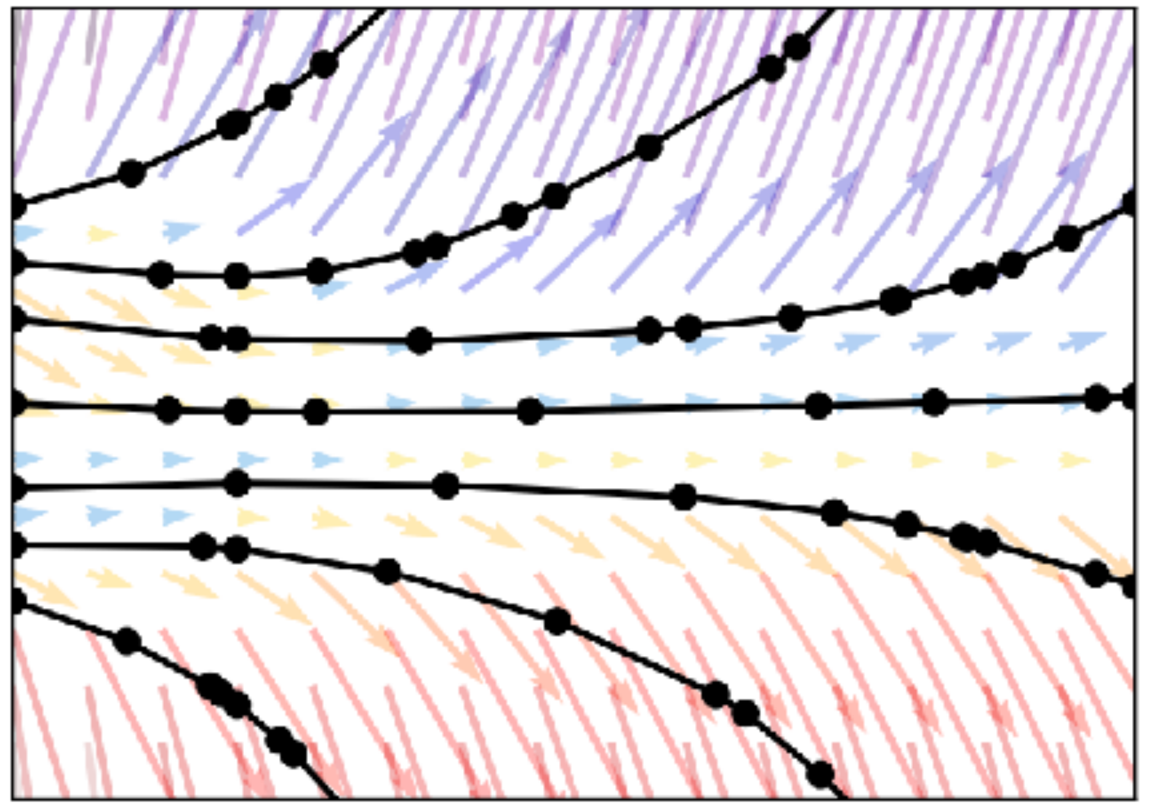
On-surface points have high probability.

Transforming a Gaussian to a Shape

Transforming a Gaussian to a Shape

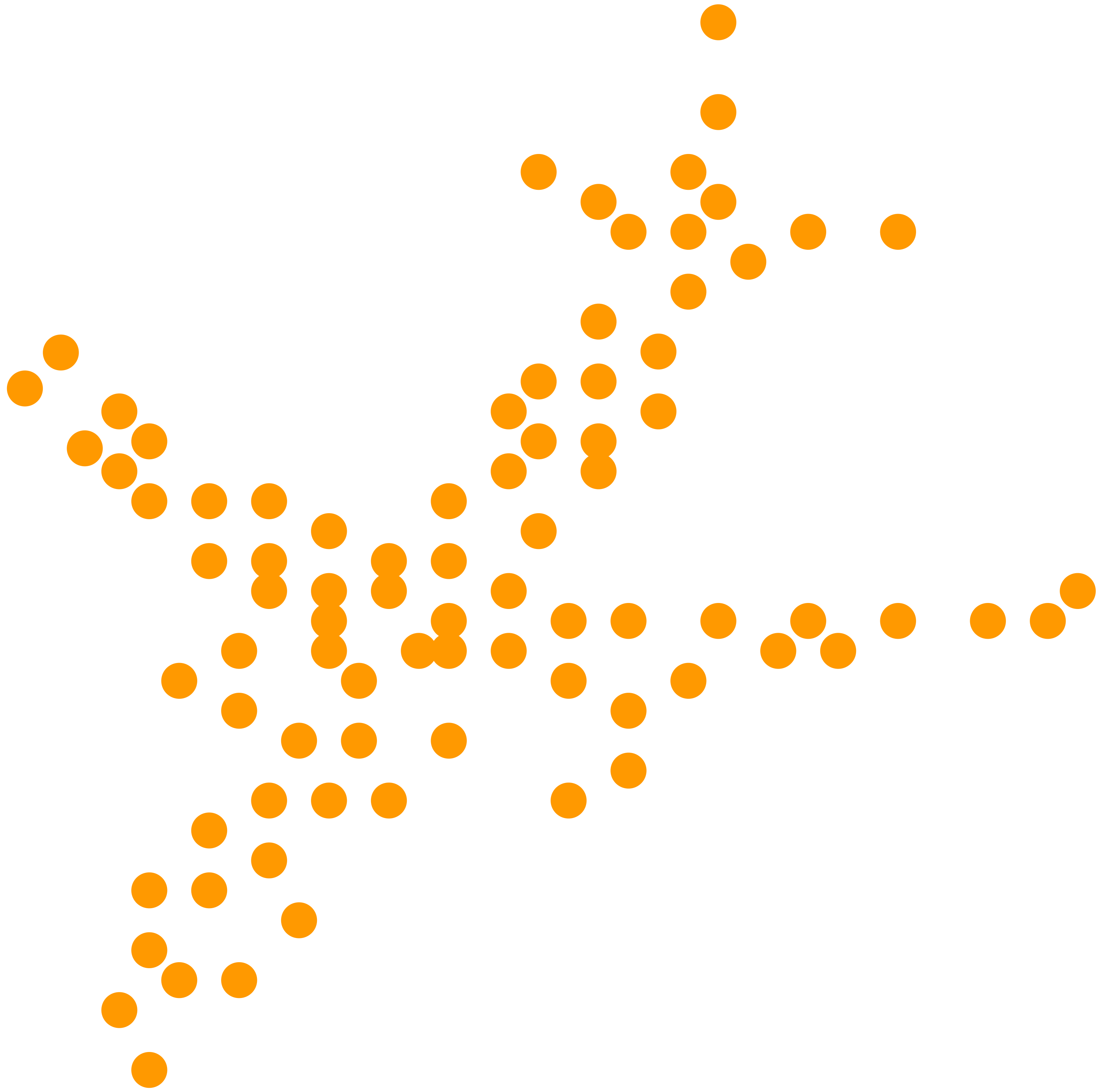






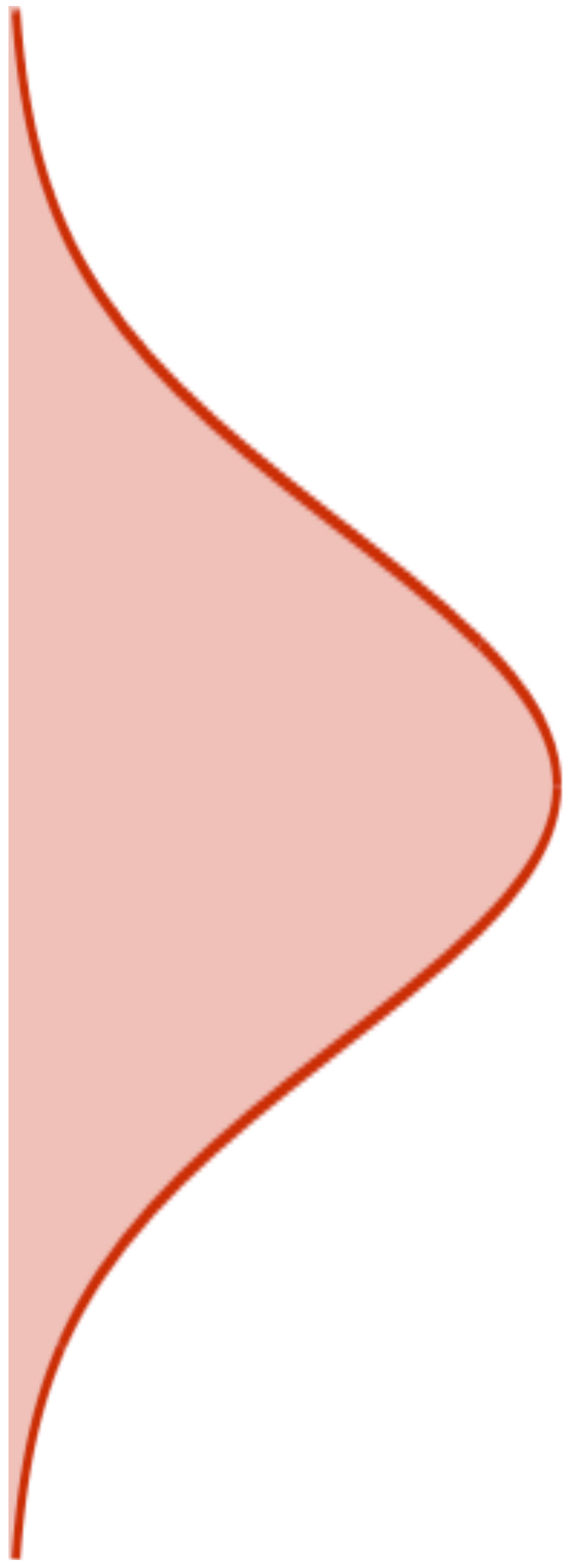
CNF

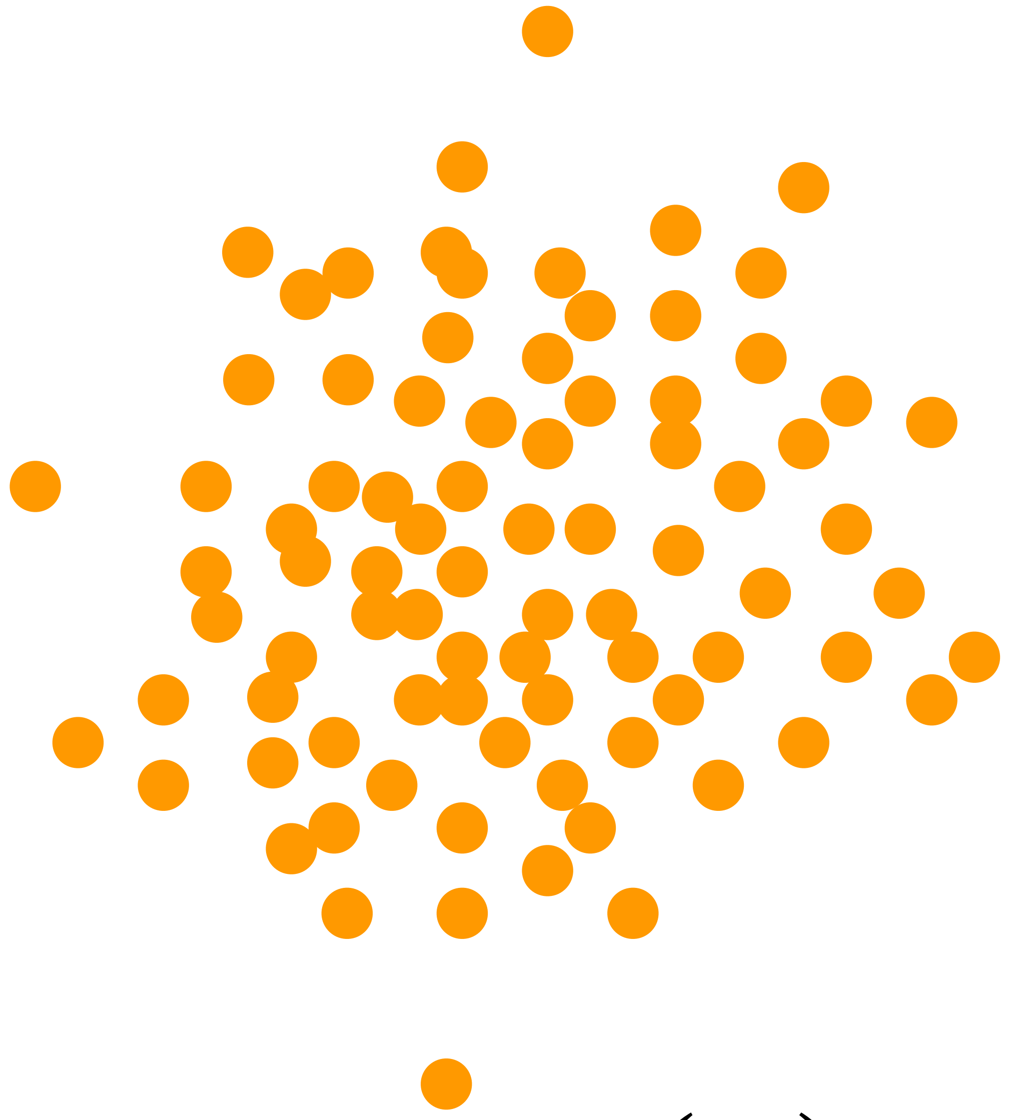
(Chen et al., 2018)

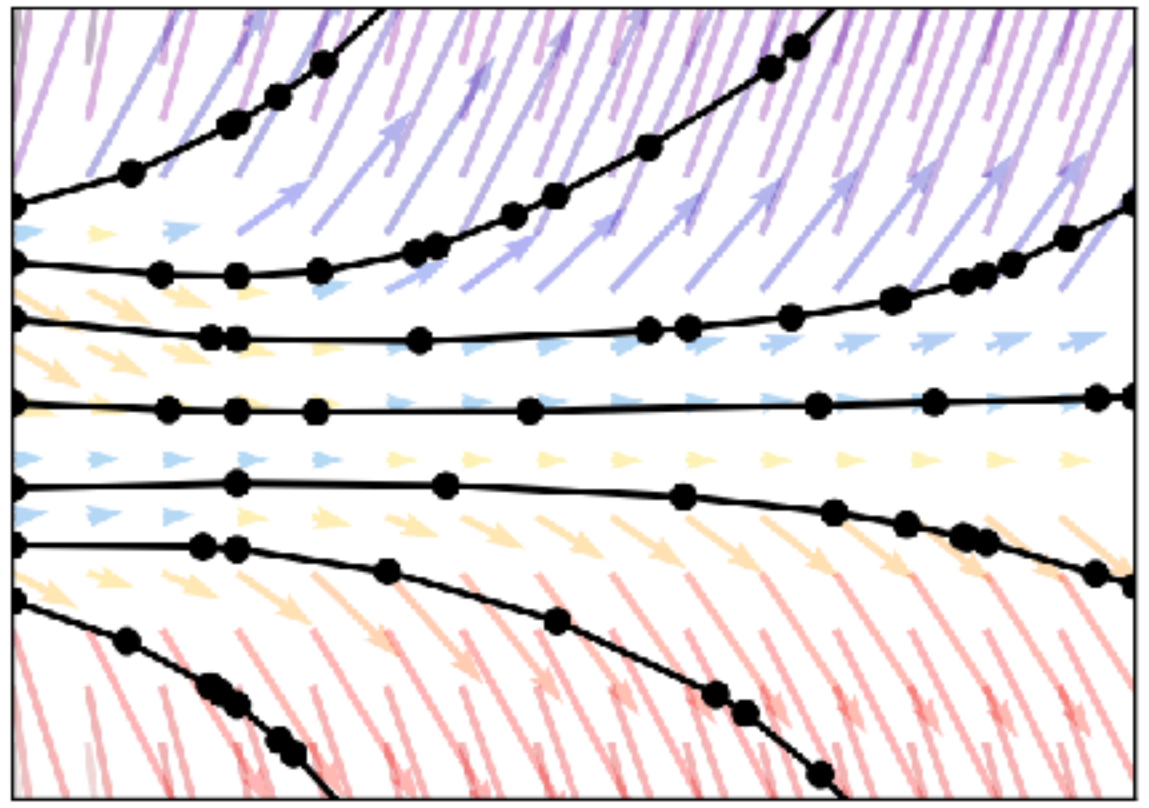


Continuous Normalizing Flow

t_0 t_1







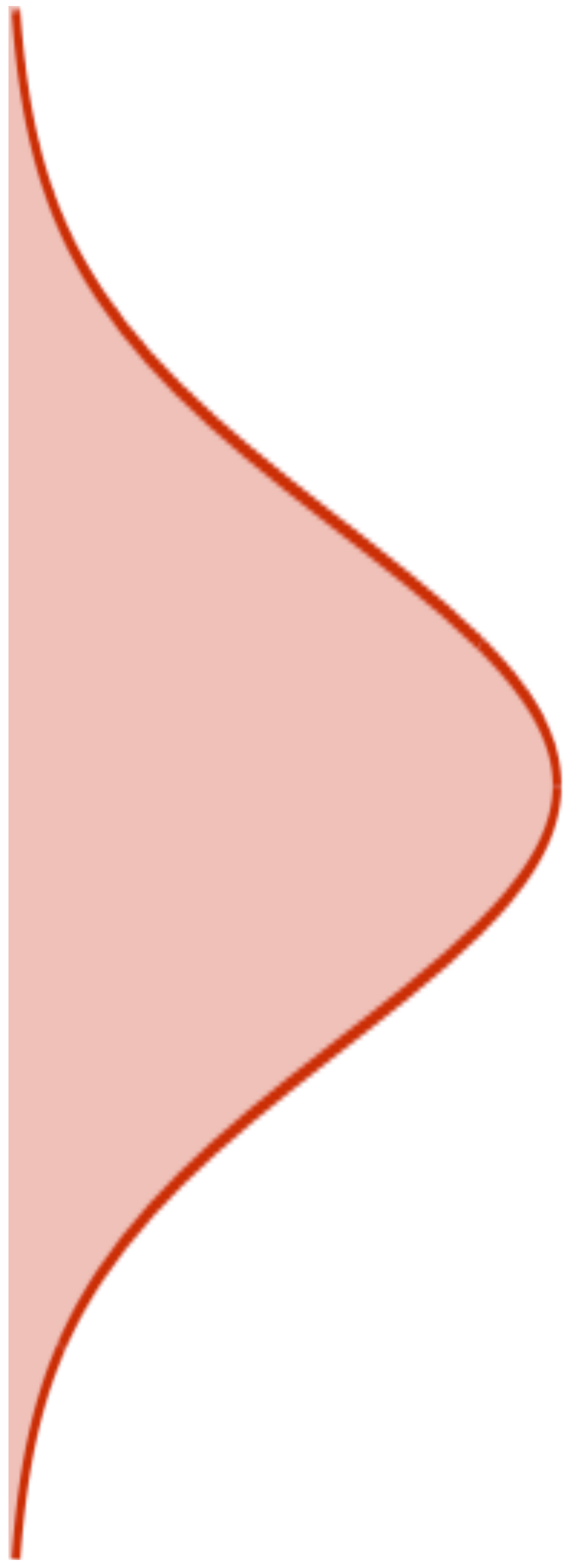
CNF

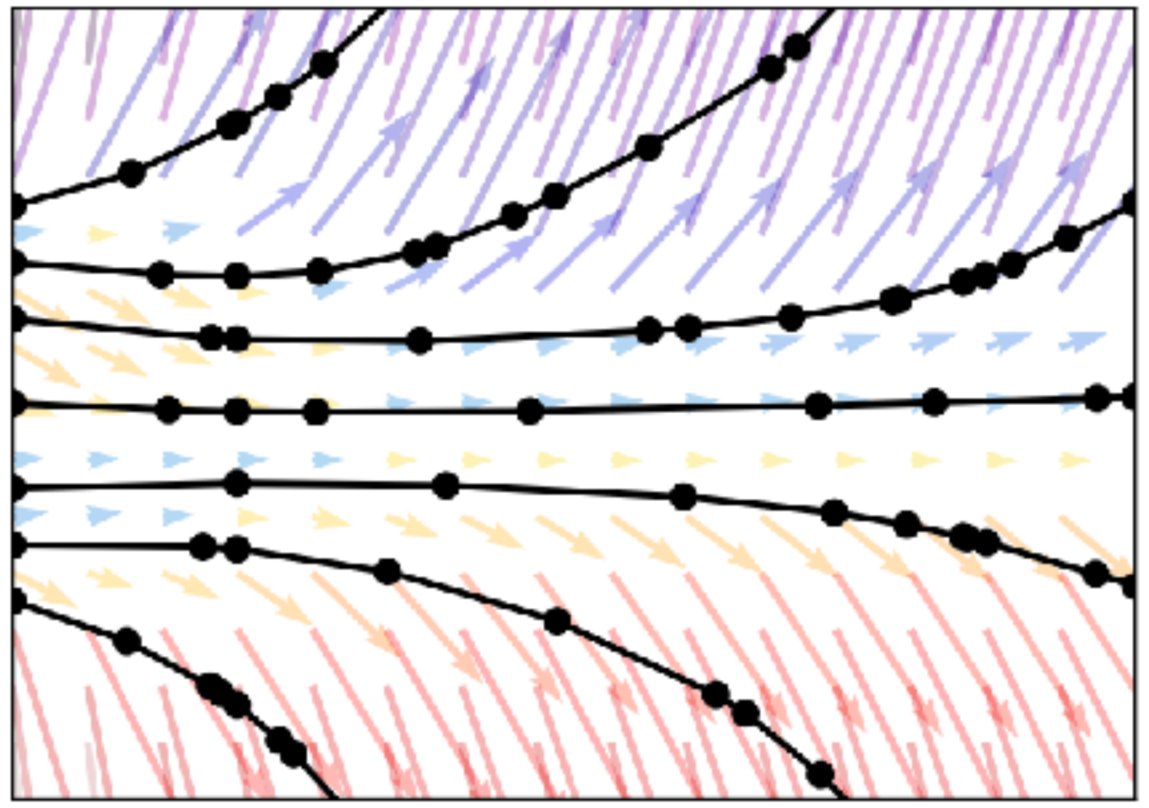
$y = y(t_0)$

$x = y(t_1)$

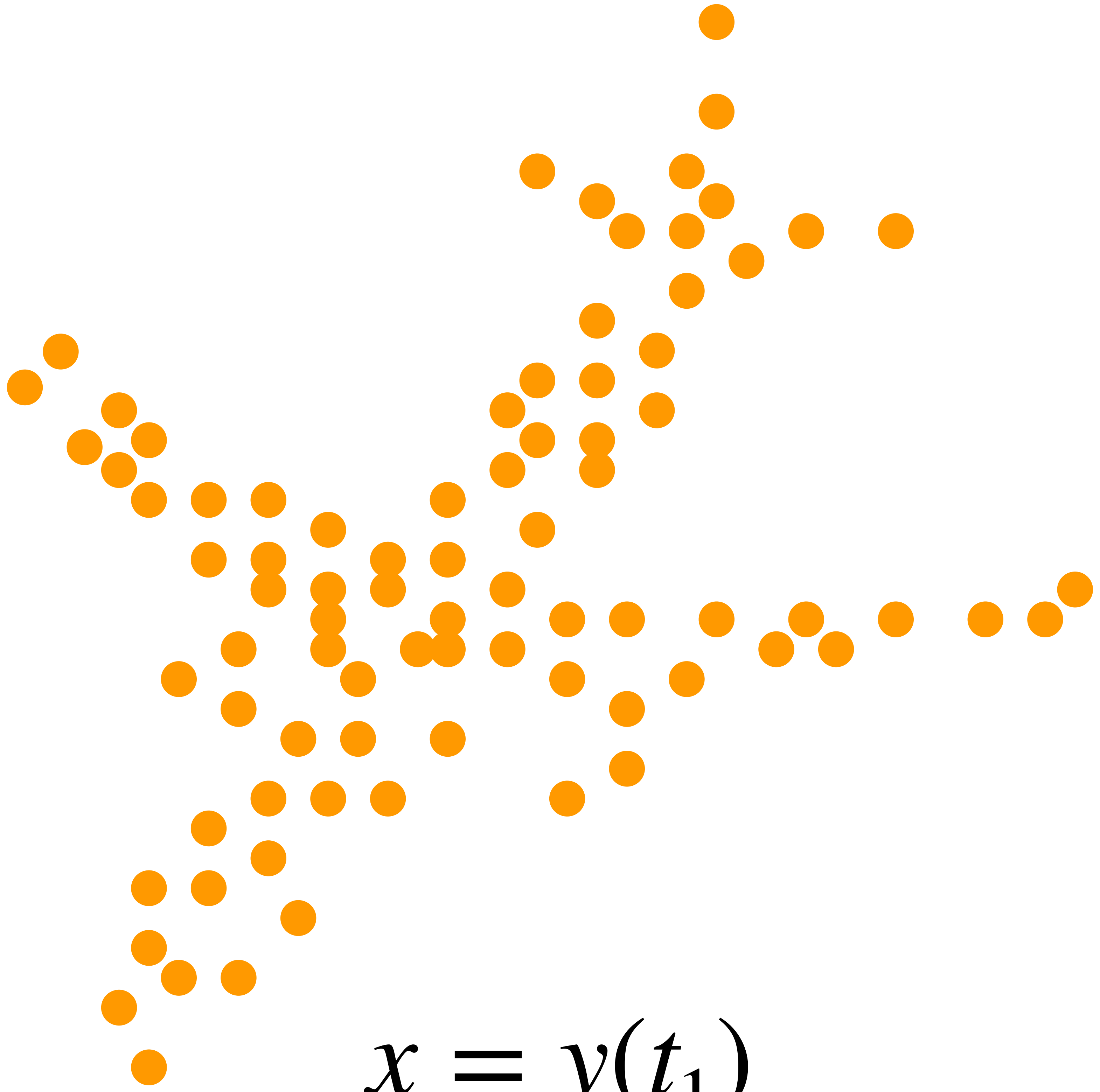
$$x = y(t_1) = y + \int_{t_0}^{t_1} g_\theta(y(t), t)dt$$

Continuous Normalizing Flow





CNF

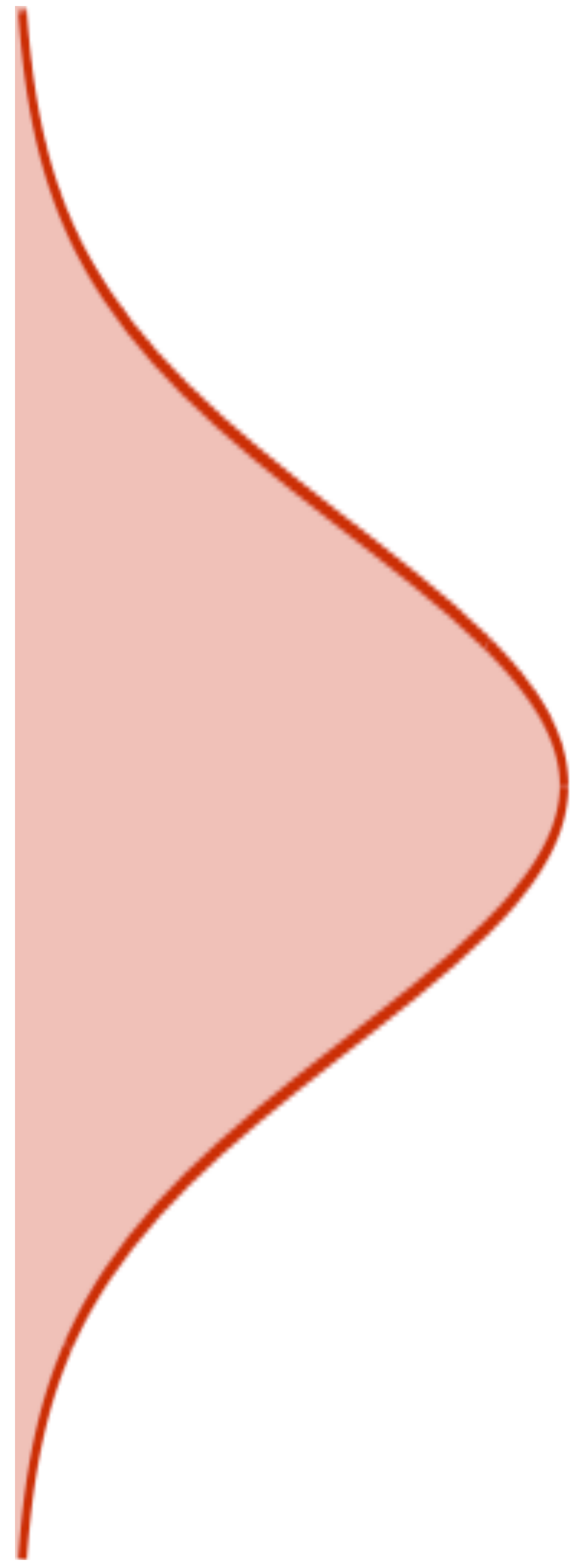


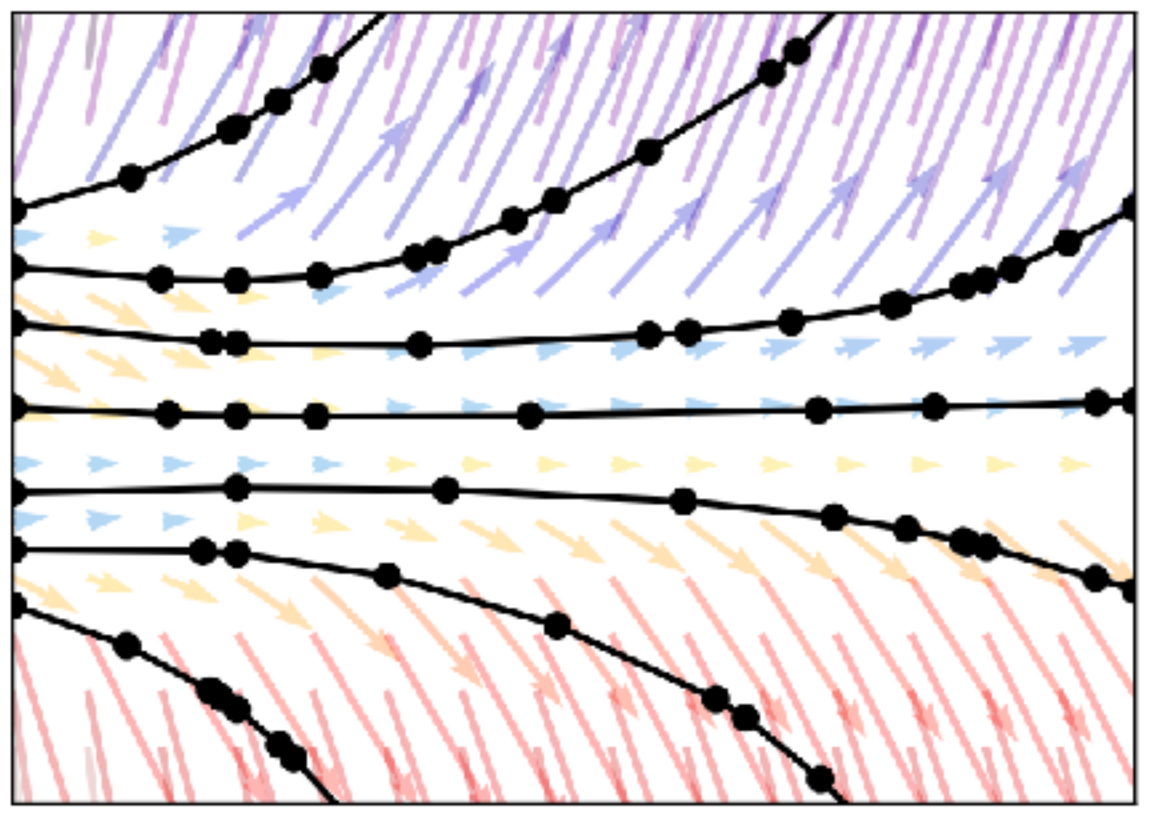
$y = y(t_0)$

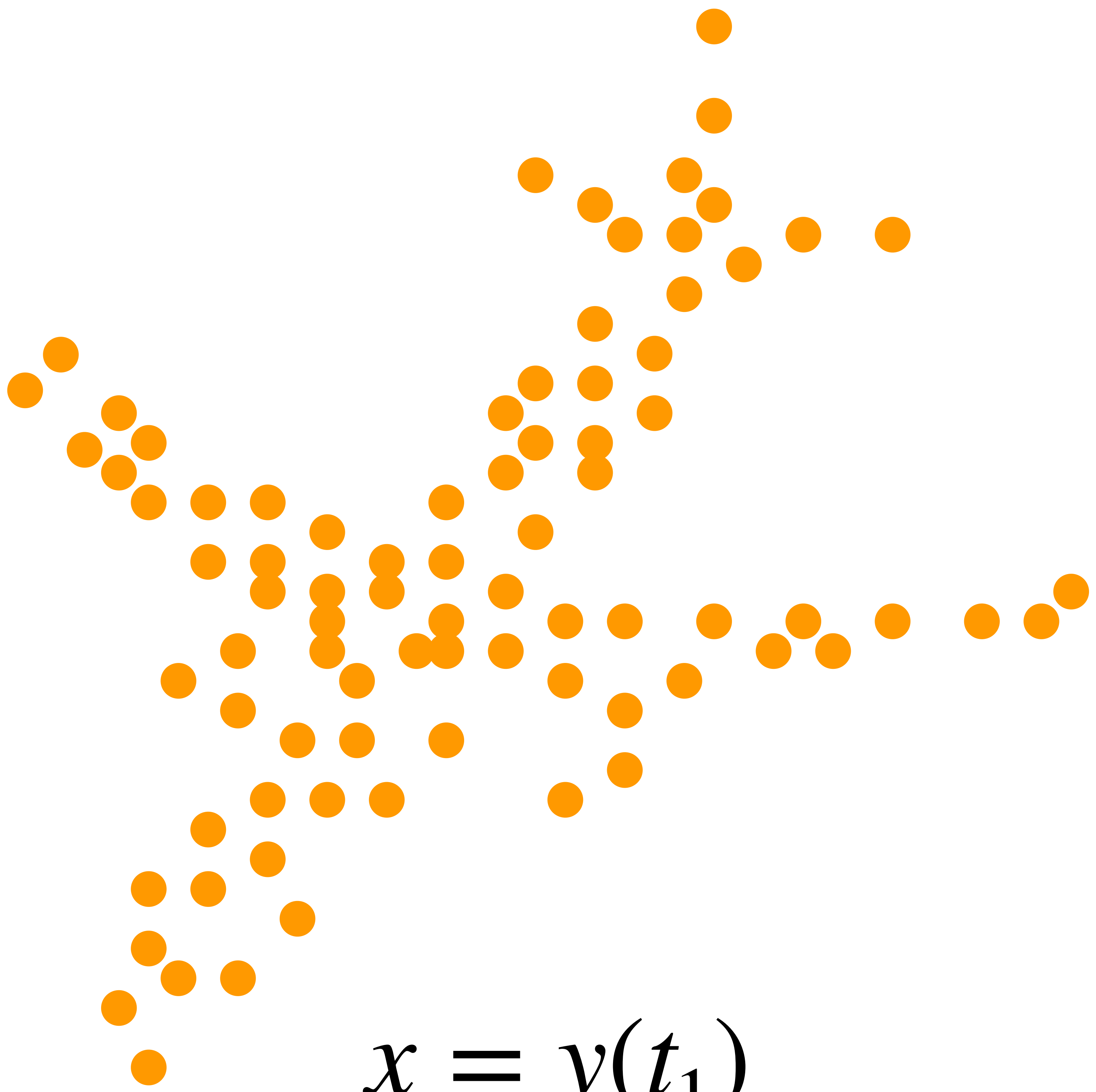
$x = y(t_1)$

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

CNF is invertible




CNF



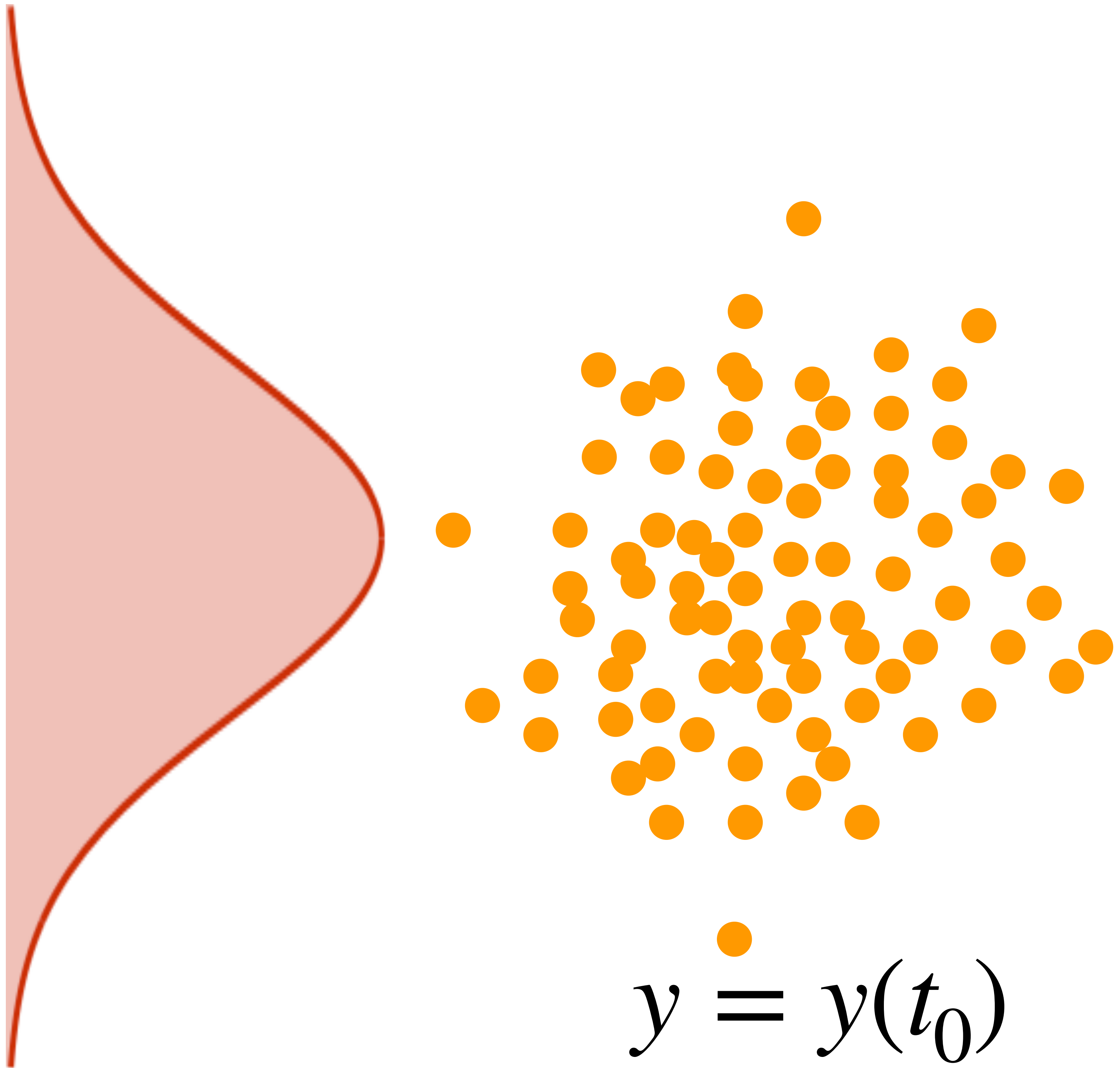
$y = y(t_0)$

$x = y(t_1)$

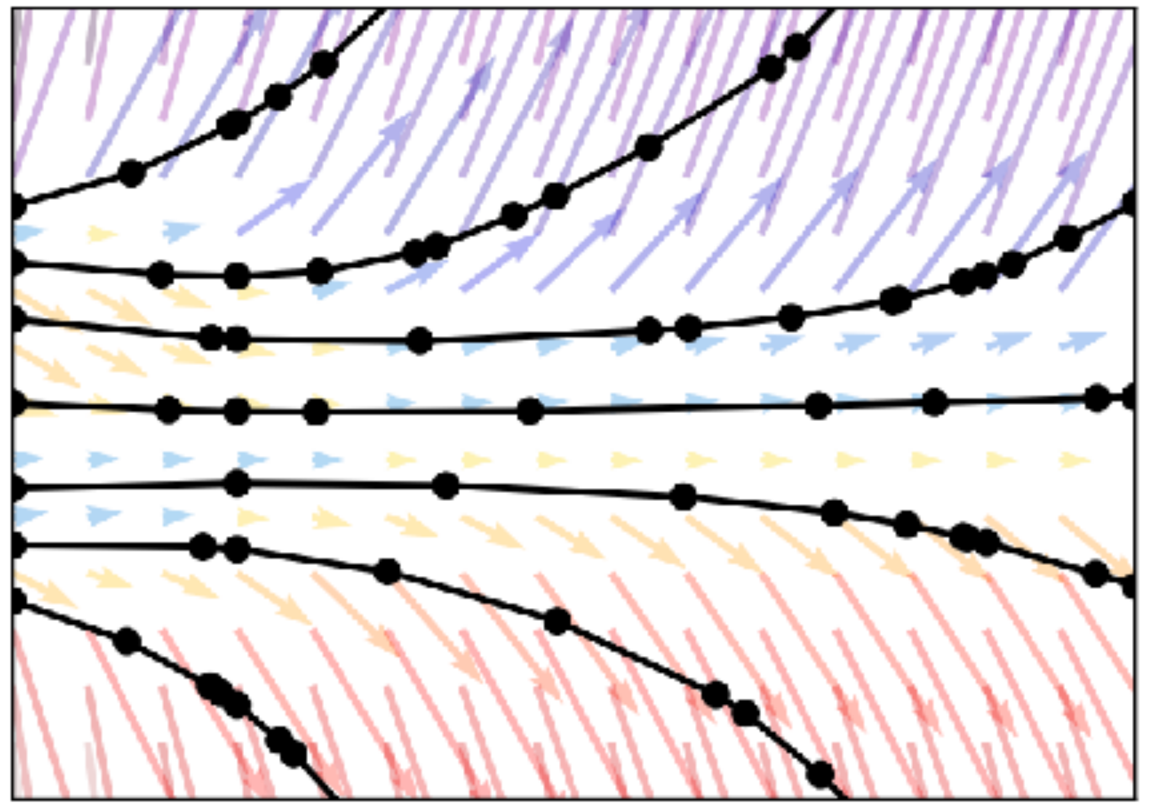
$$y = y(t_0) = x + \boxed{\int_{t_1}^{t_0}} g_\theta(y(t), t)dt$$

CNF is invertible

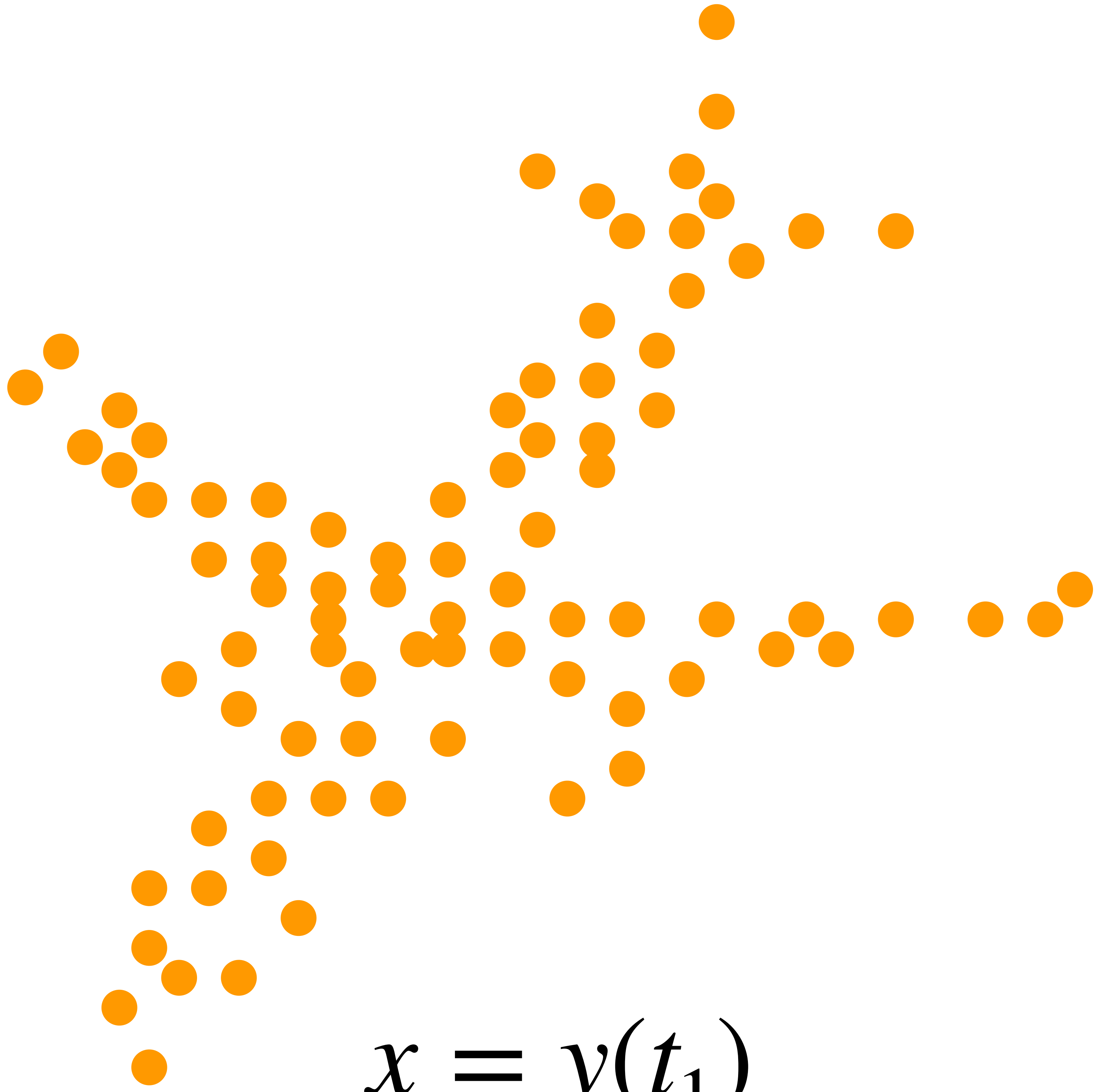
t_0 t_1



$y = y(t_0)$



CNF

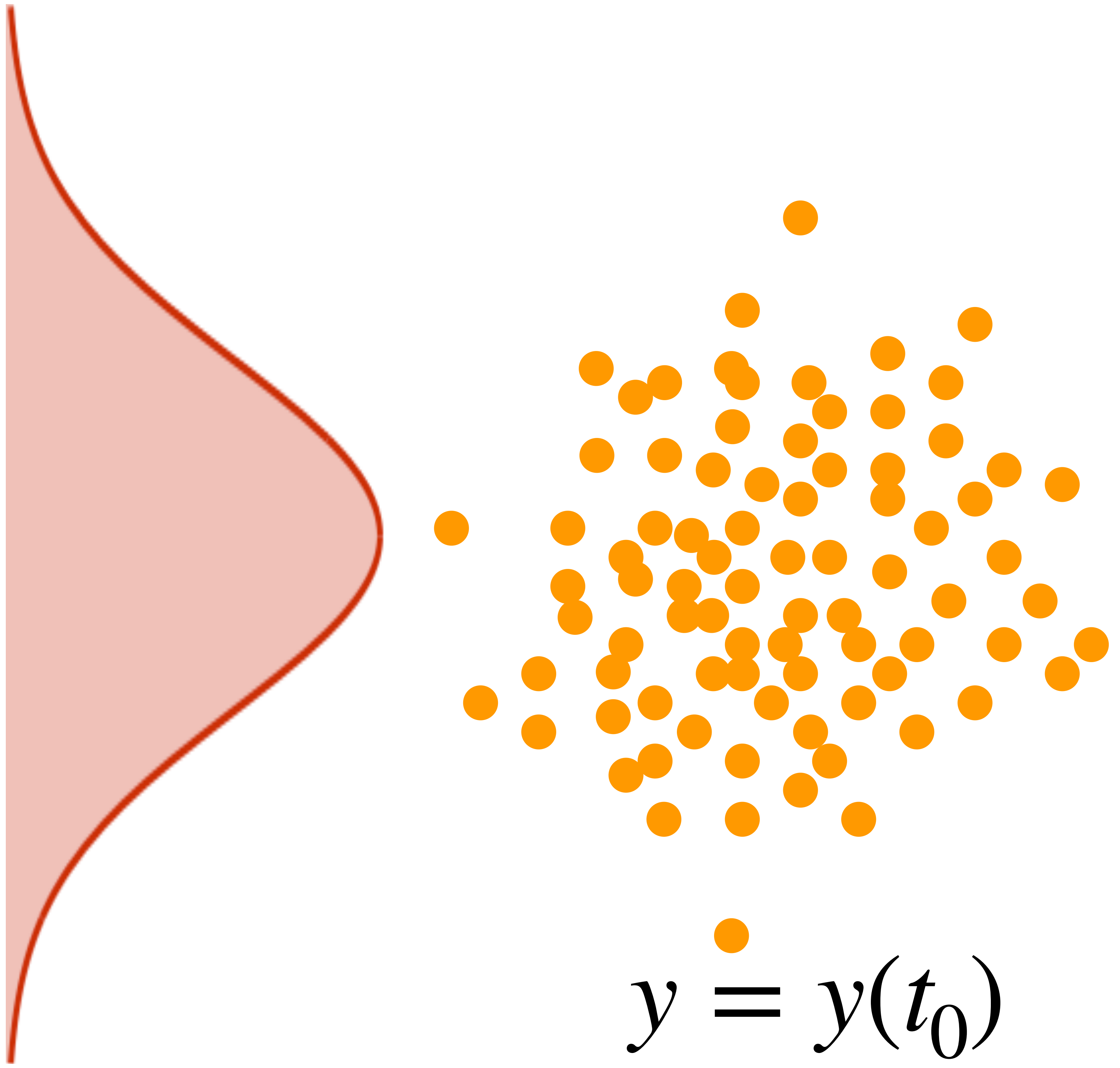


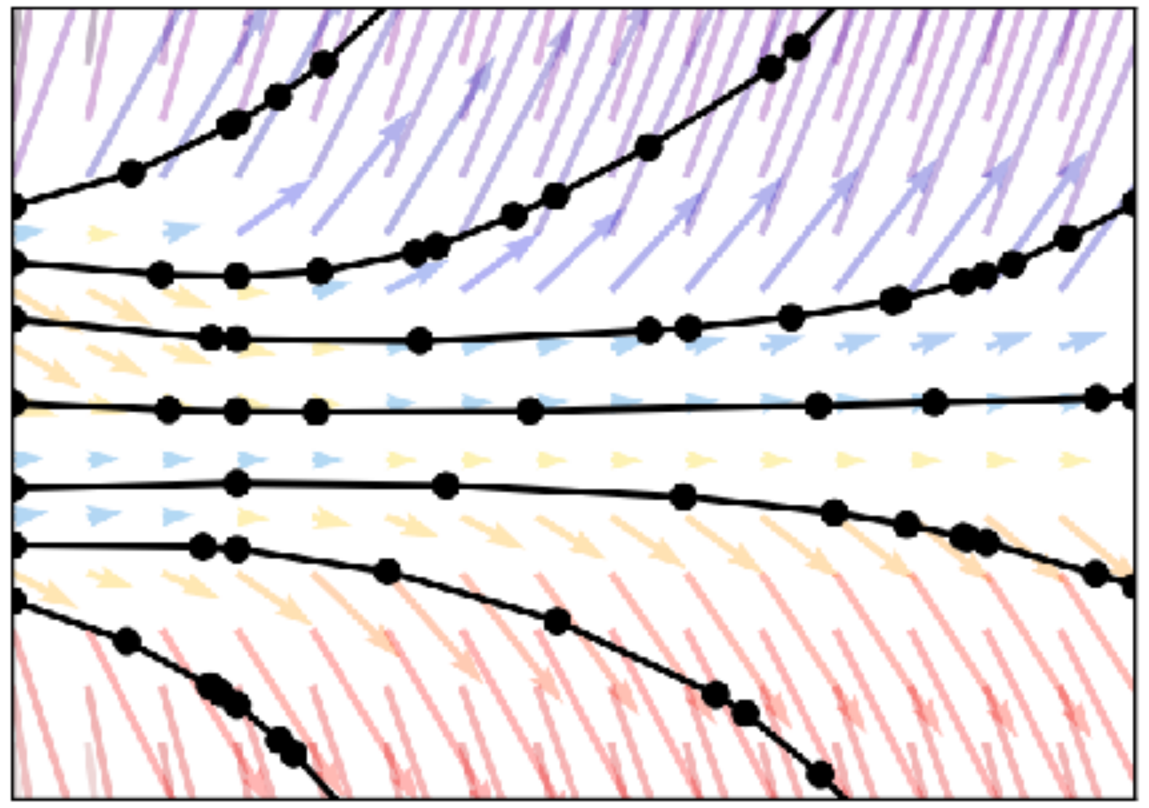
$x = y(t_1)$

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

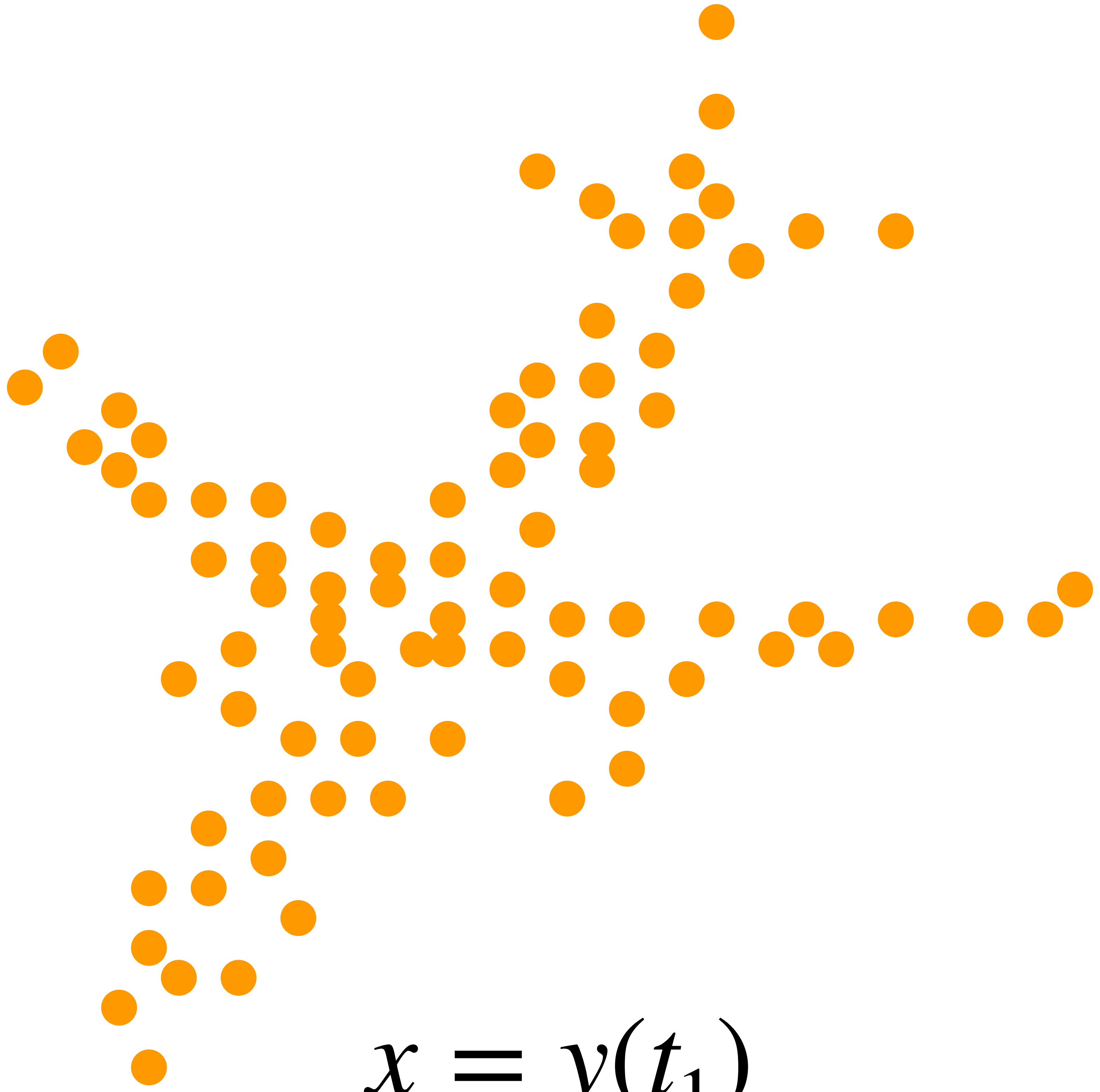
Change of Variable Formula

t_0 t_1





CNF



$y = y(t_0)$ $x = y(t_1)$

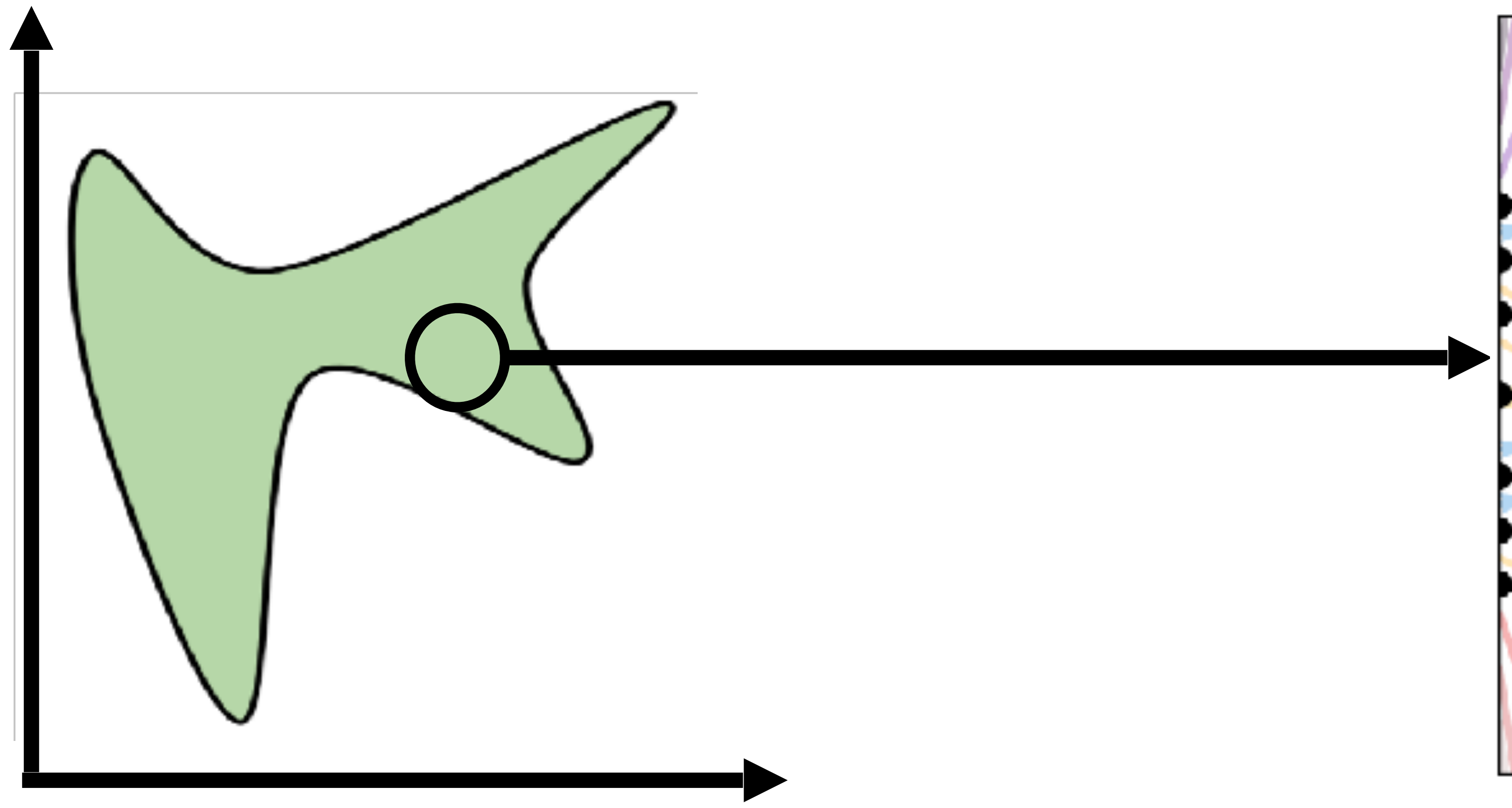
$$\log P(x) = \log P\left(x + \int_{t_1}^{t_0} g_\theta(y(t), t)dt\right) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial g_\theta(x(t), t)}{\partial x(t)}\right) dt$$

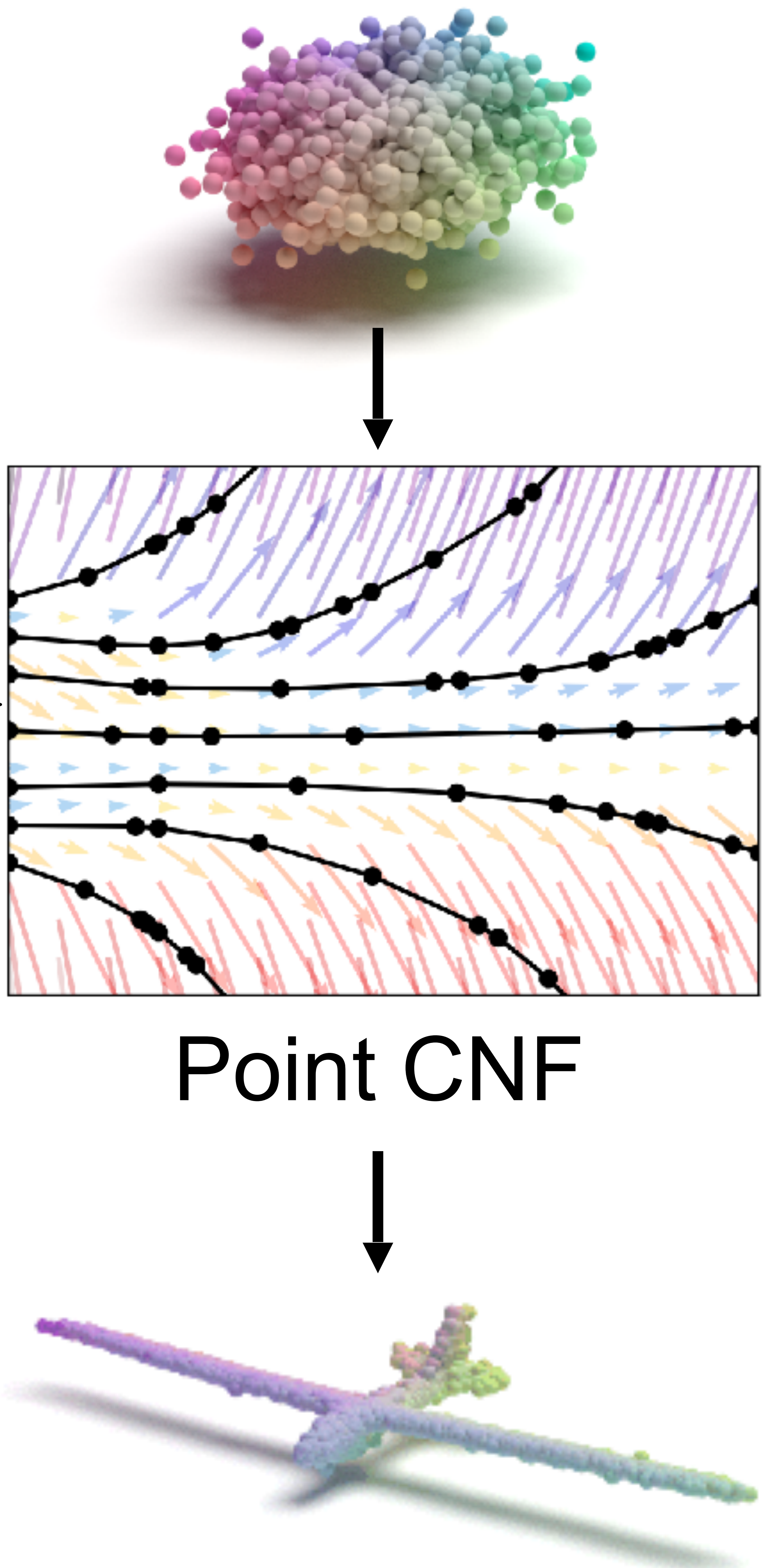
Encoding multiple shapes

$$g_\theta : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}^3$$

$$\downarrow$$

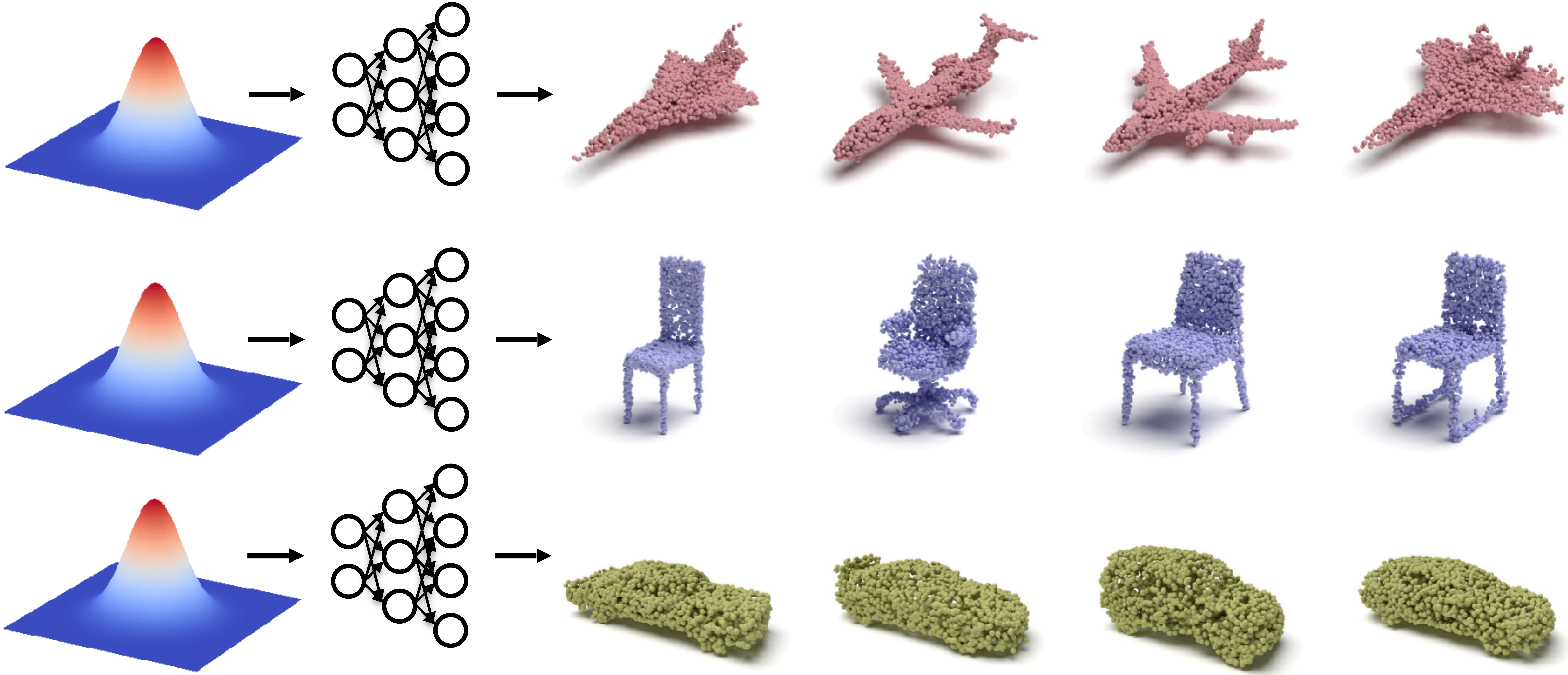
$$g_\theta : \mathbb{R}^3 \times \mathbb{R} \times \boxed{\mathbb{R}^{128}} \to \mathbb{R}^3$$



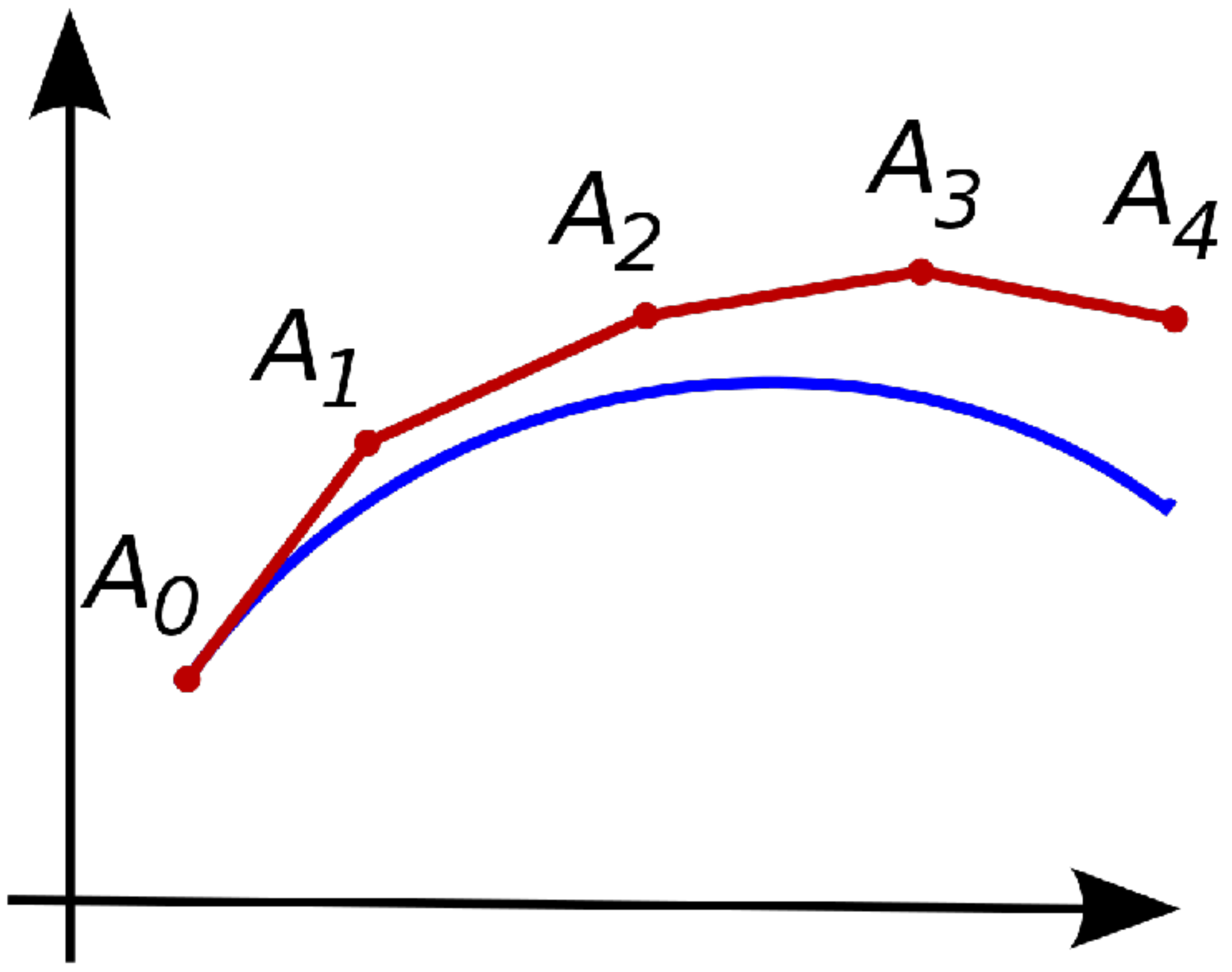


$$\log P(x\,|\,z) = \log P\left(x + \int_{t_1}^{t_0} g_\theta(y(t), t, \boxed{z})\, dt\right) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial g_\theta(x(t), t, \boxed{z})}{\partial x(t)}\right) dt$$

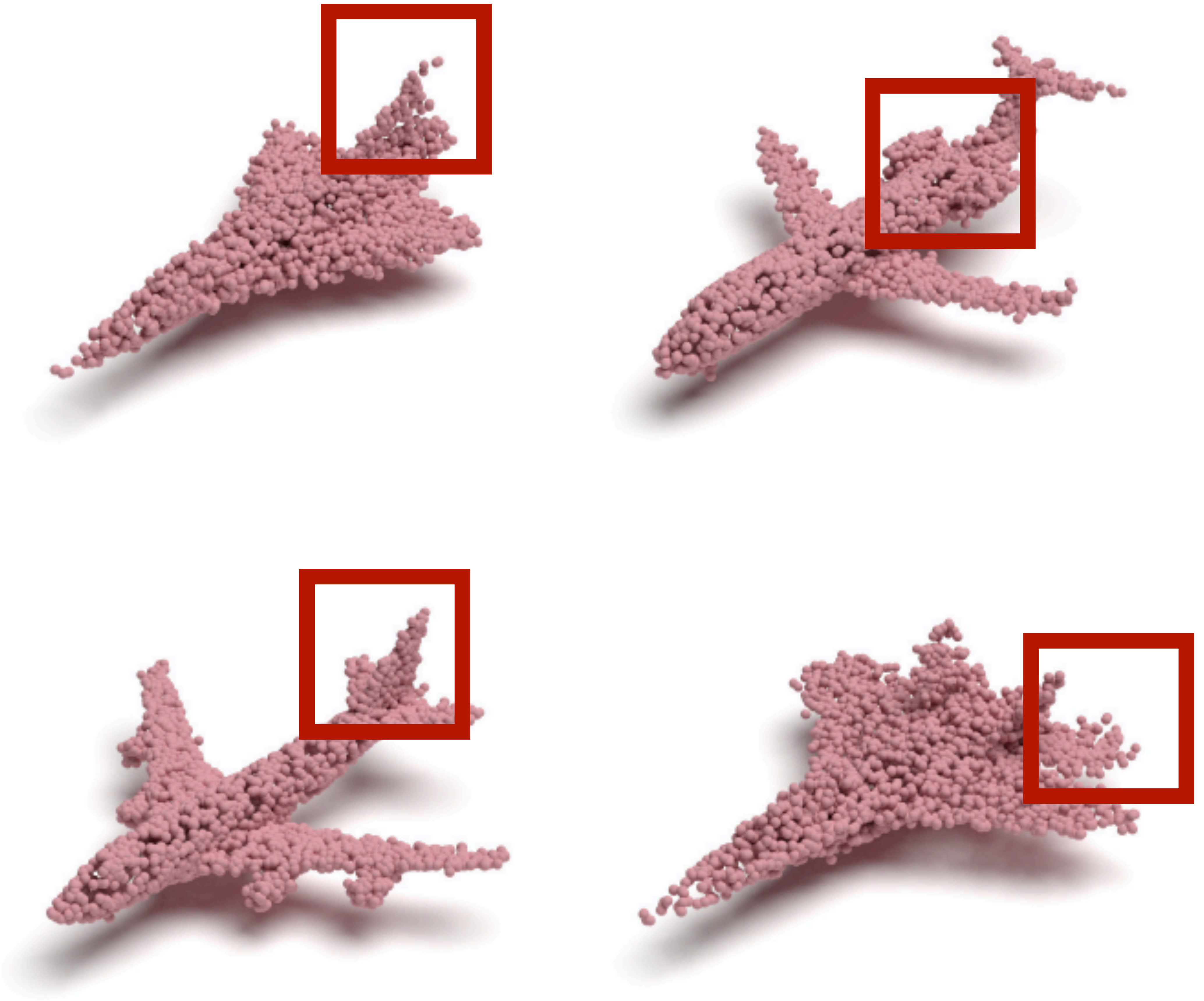
Generation Results



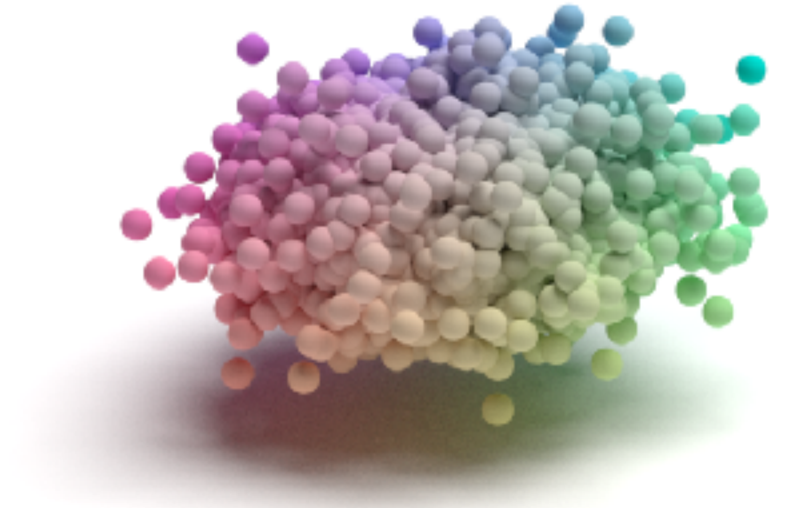
Limitation

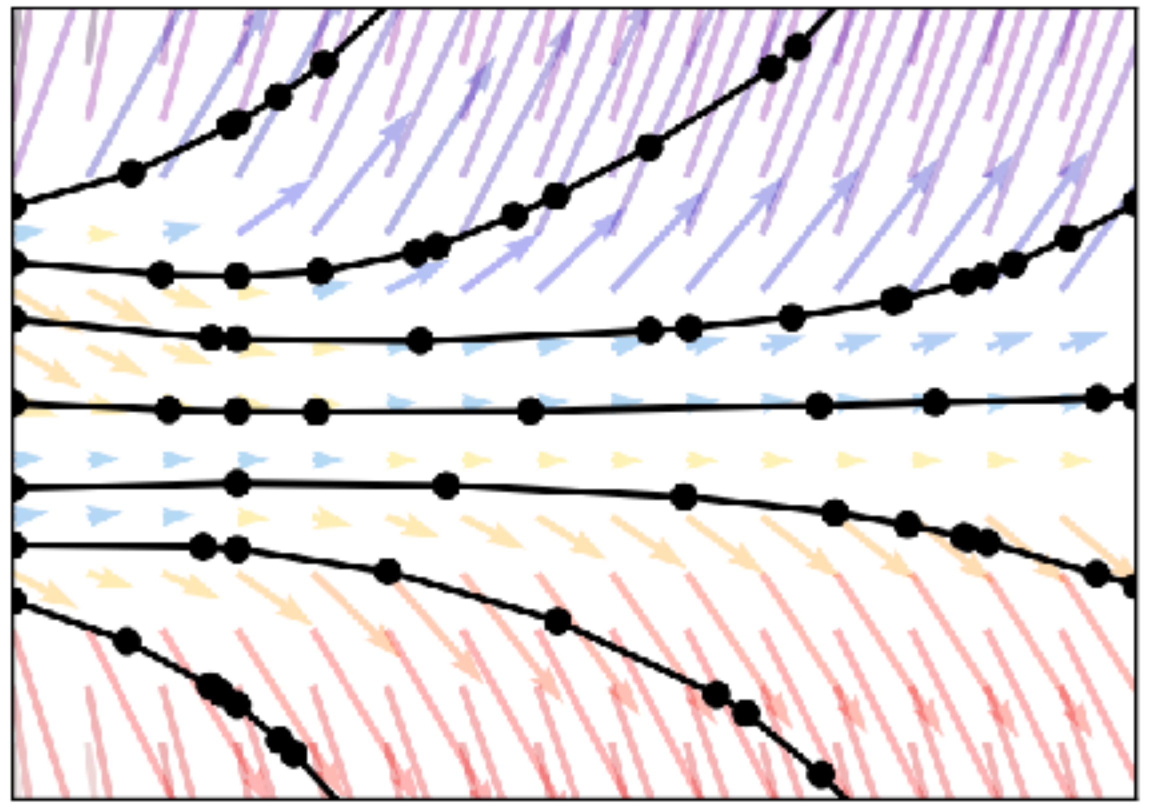


Slow training time

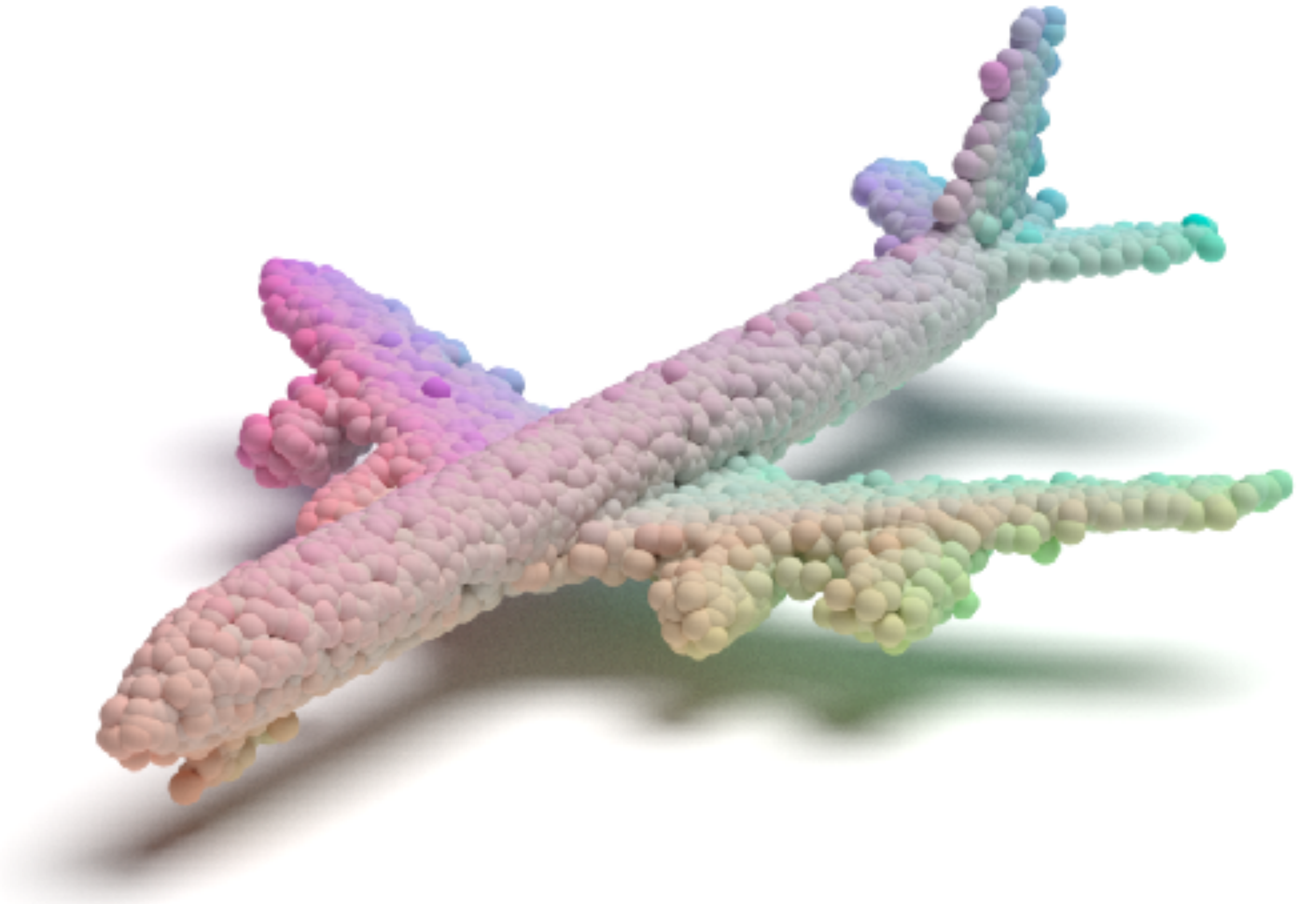


Modeling a normalized distribution is hard





Point CNF

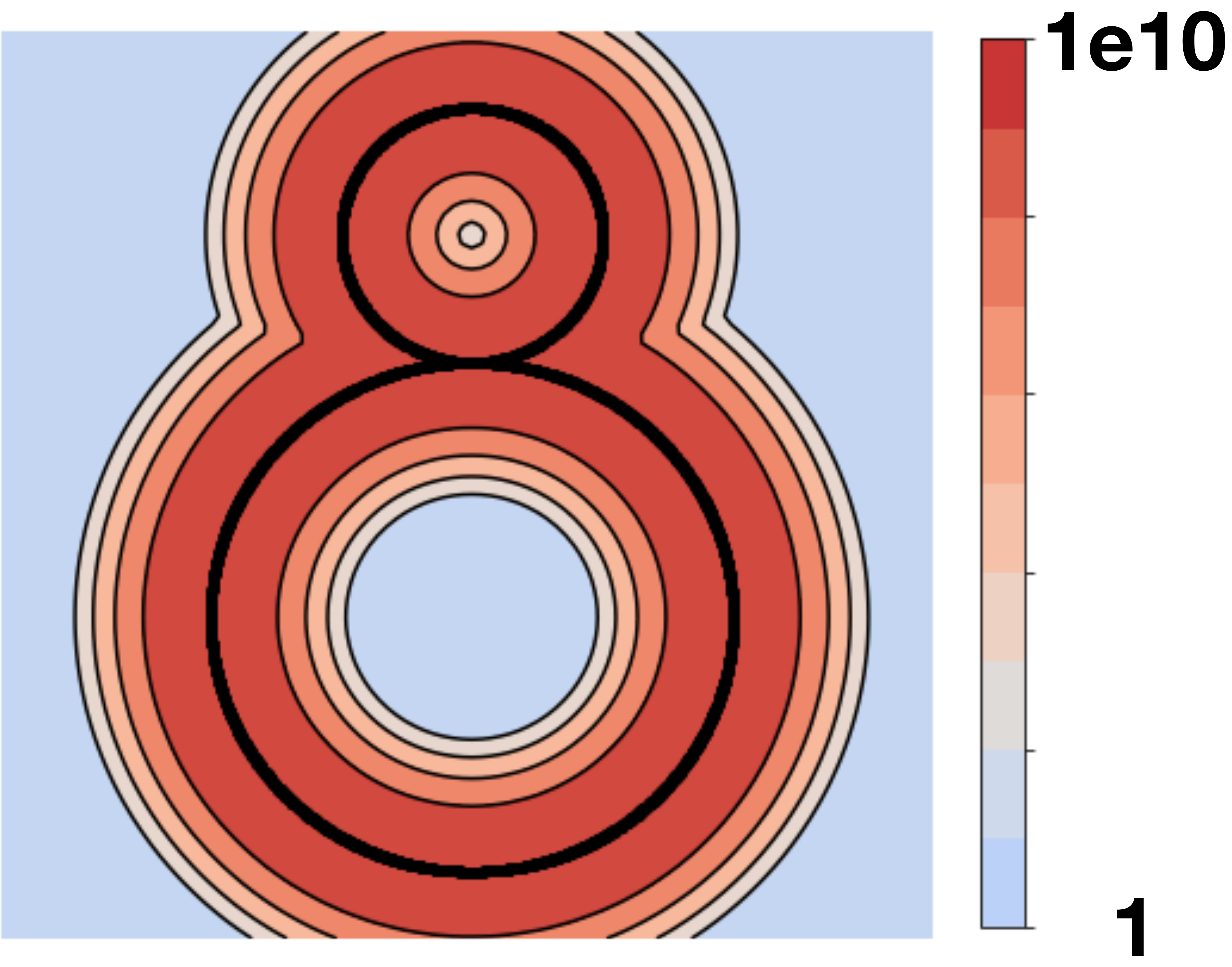


$$\log P(x) = \log P\left(\boxed{x + \int_{t_1}^{t_0} g_\theta(y(t), t)dt}\right) \boxed{- \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial g_\theta(x(t), t)}{\partial x(t)}\right) dt}$$

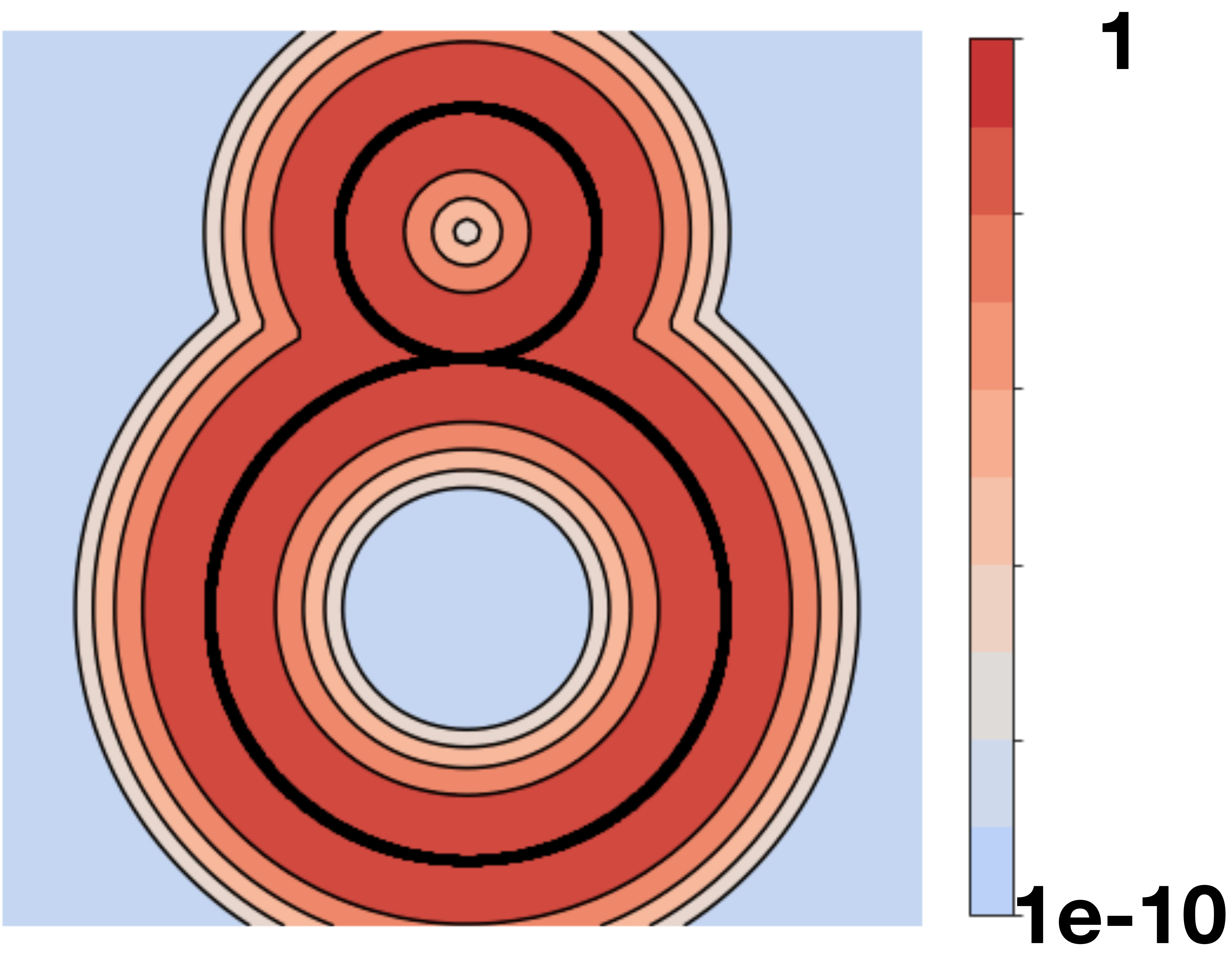
Invertible
(Restricted)

Normalizing
(Slow, create noise)

Each shape is an unnormalized 3D density fields.



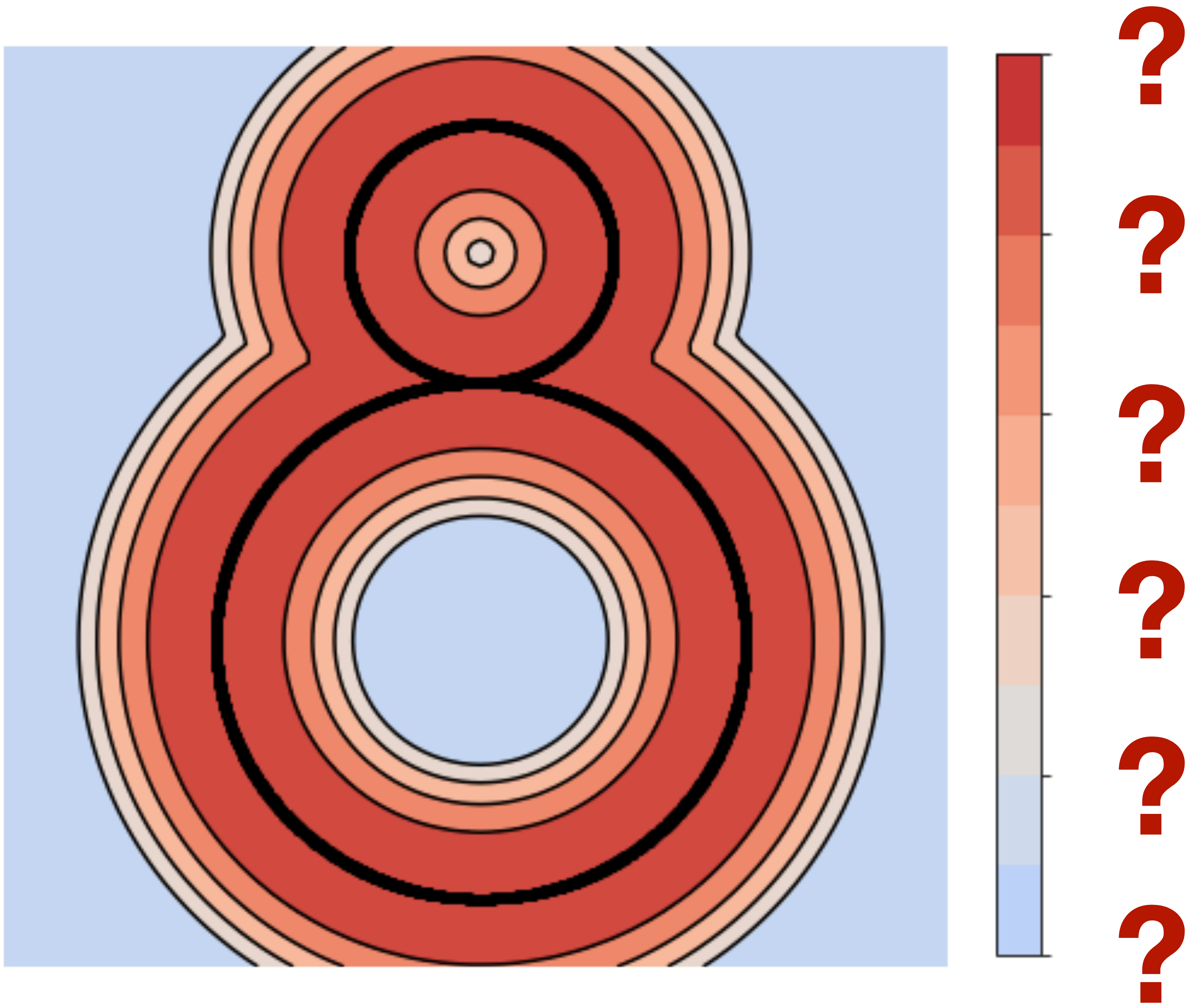
Density field



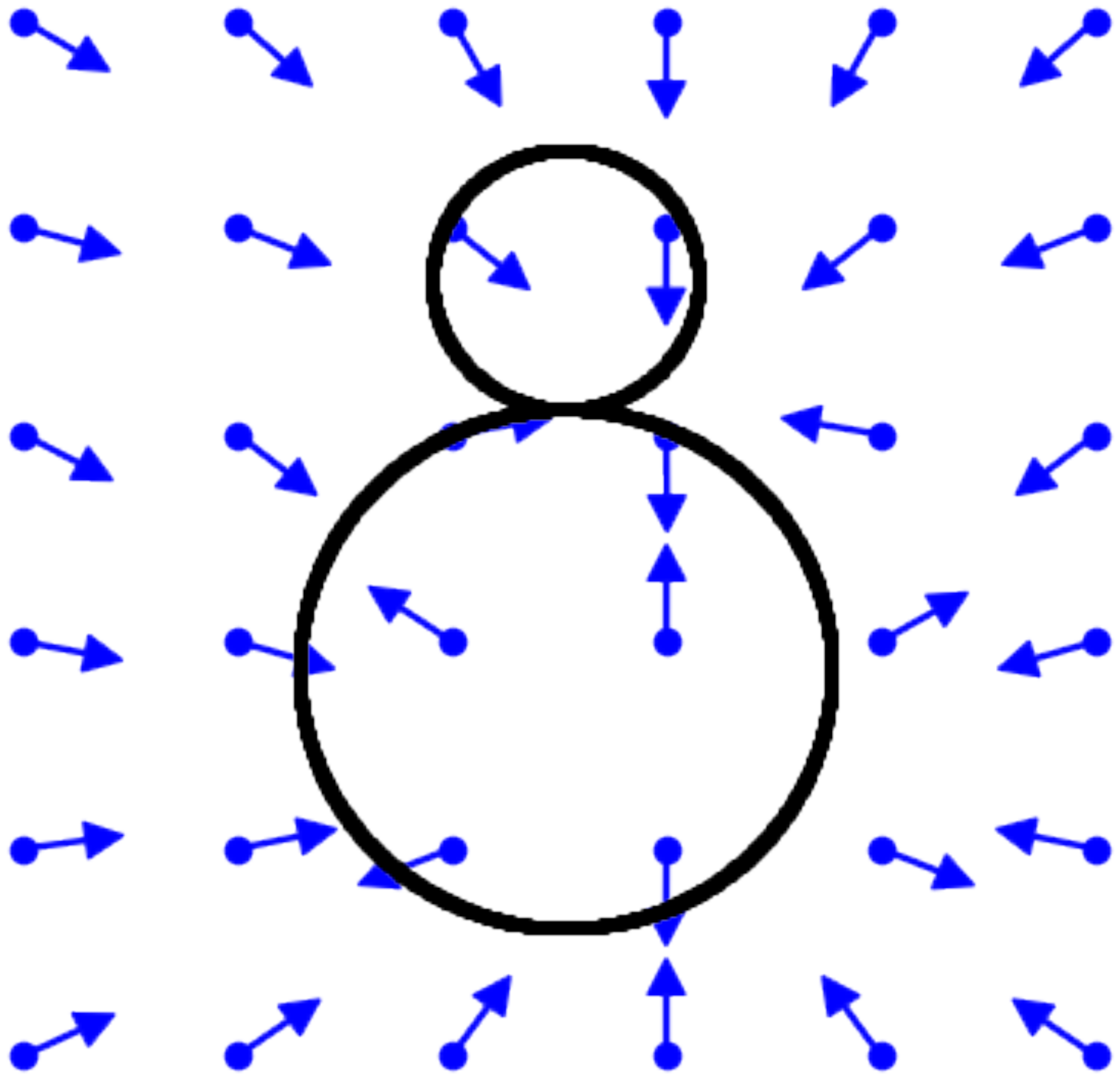
Density field

Different scale, SAME SHAPE

Representing an unnormalized 3D density field

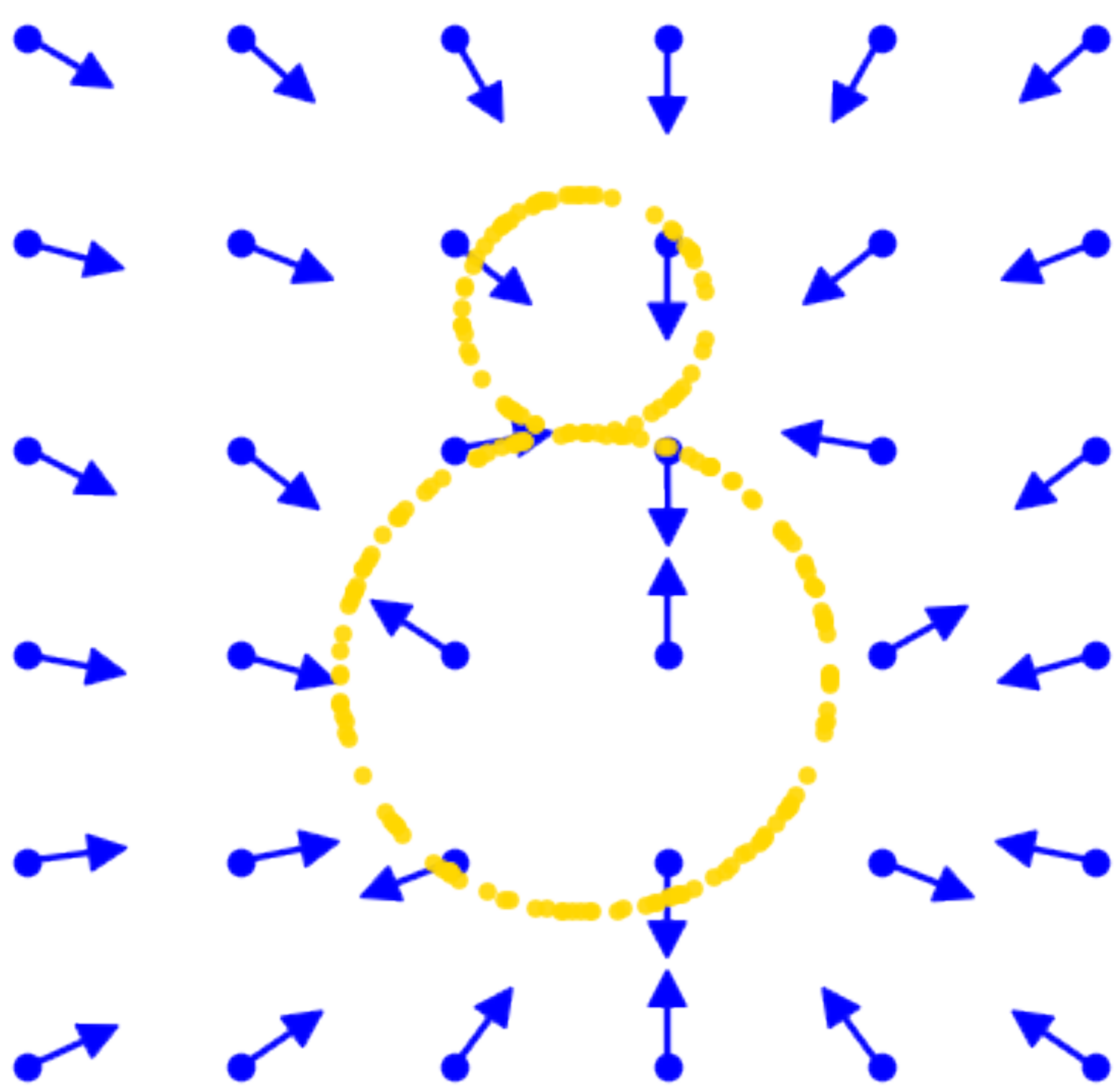


Unnormalized density field

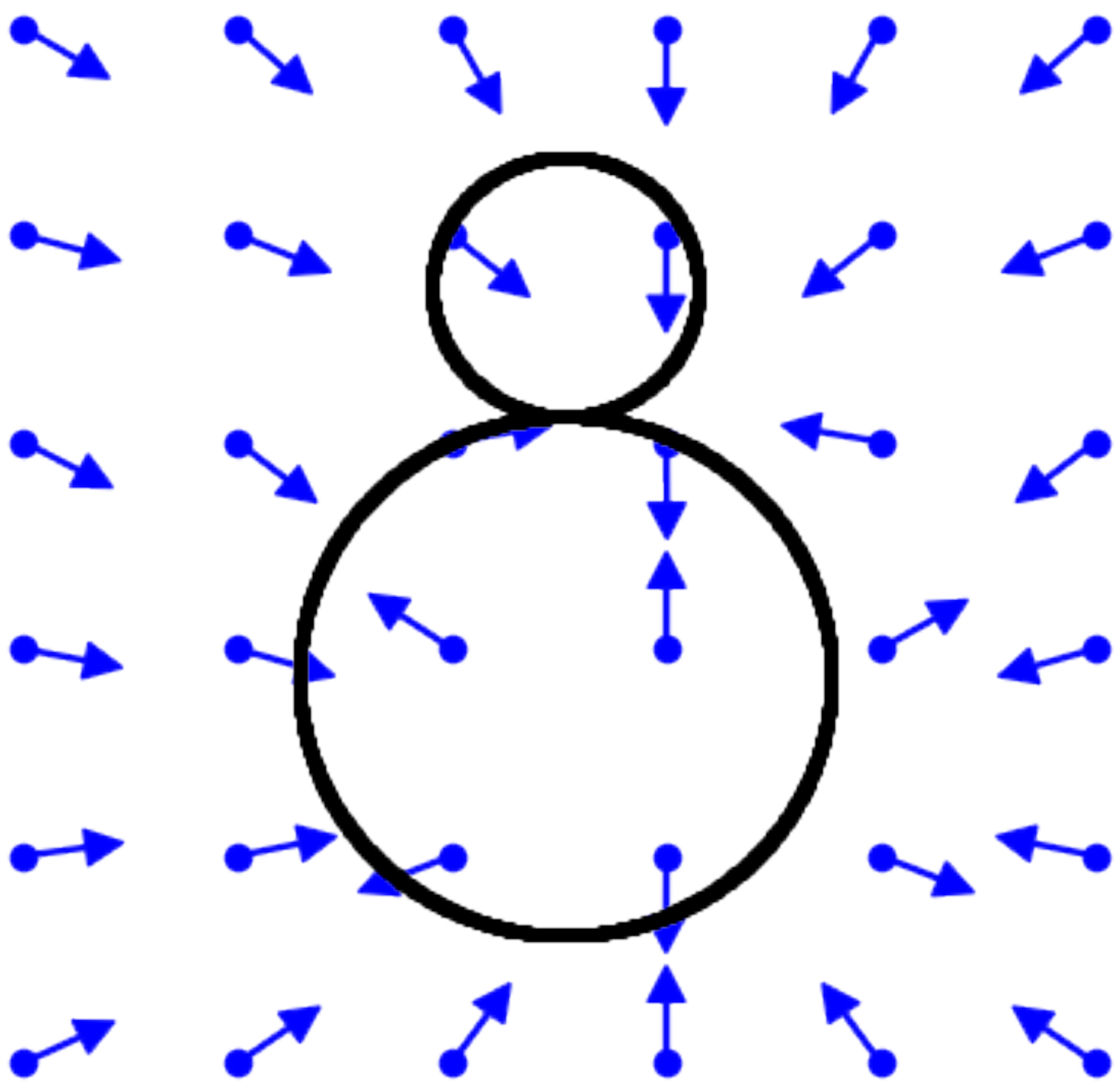


Gradient field

Representing an unnormalized 3D density field.

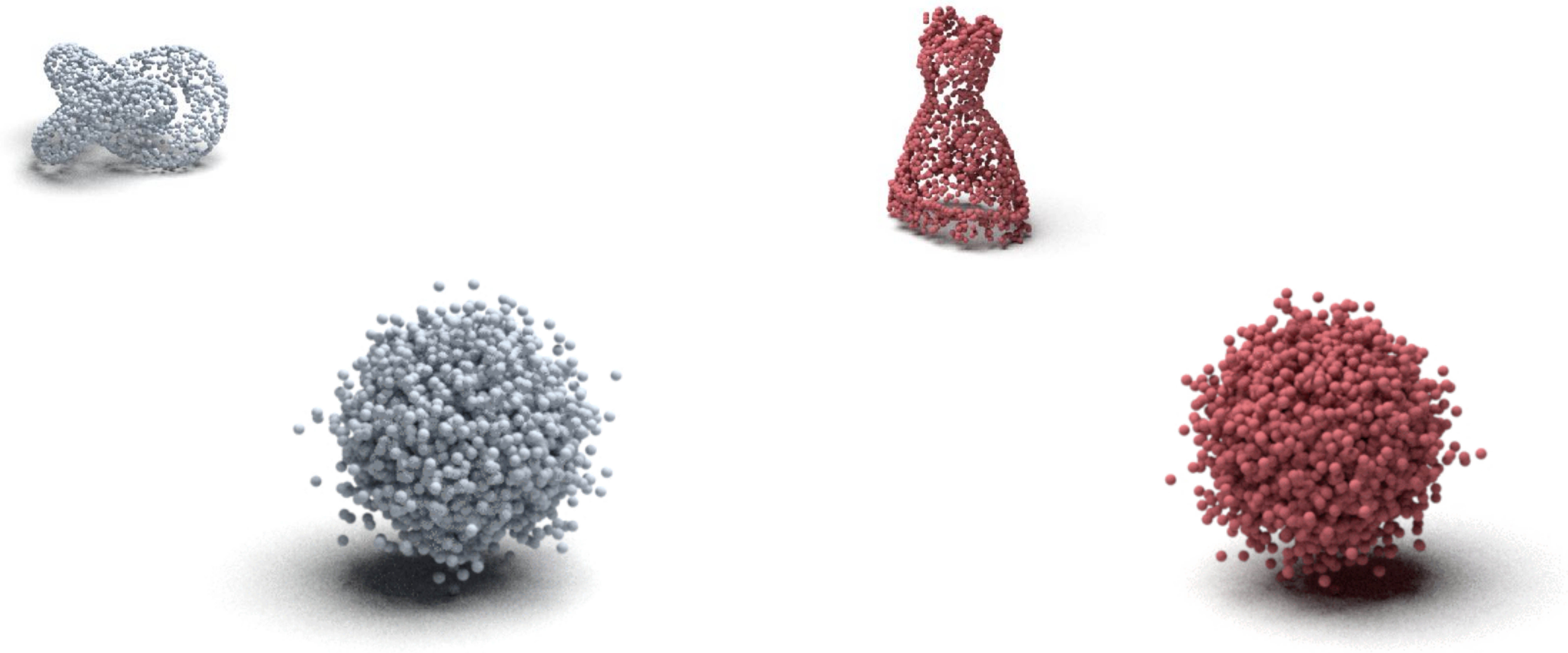


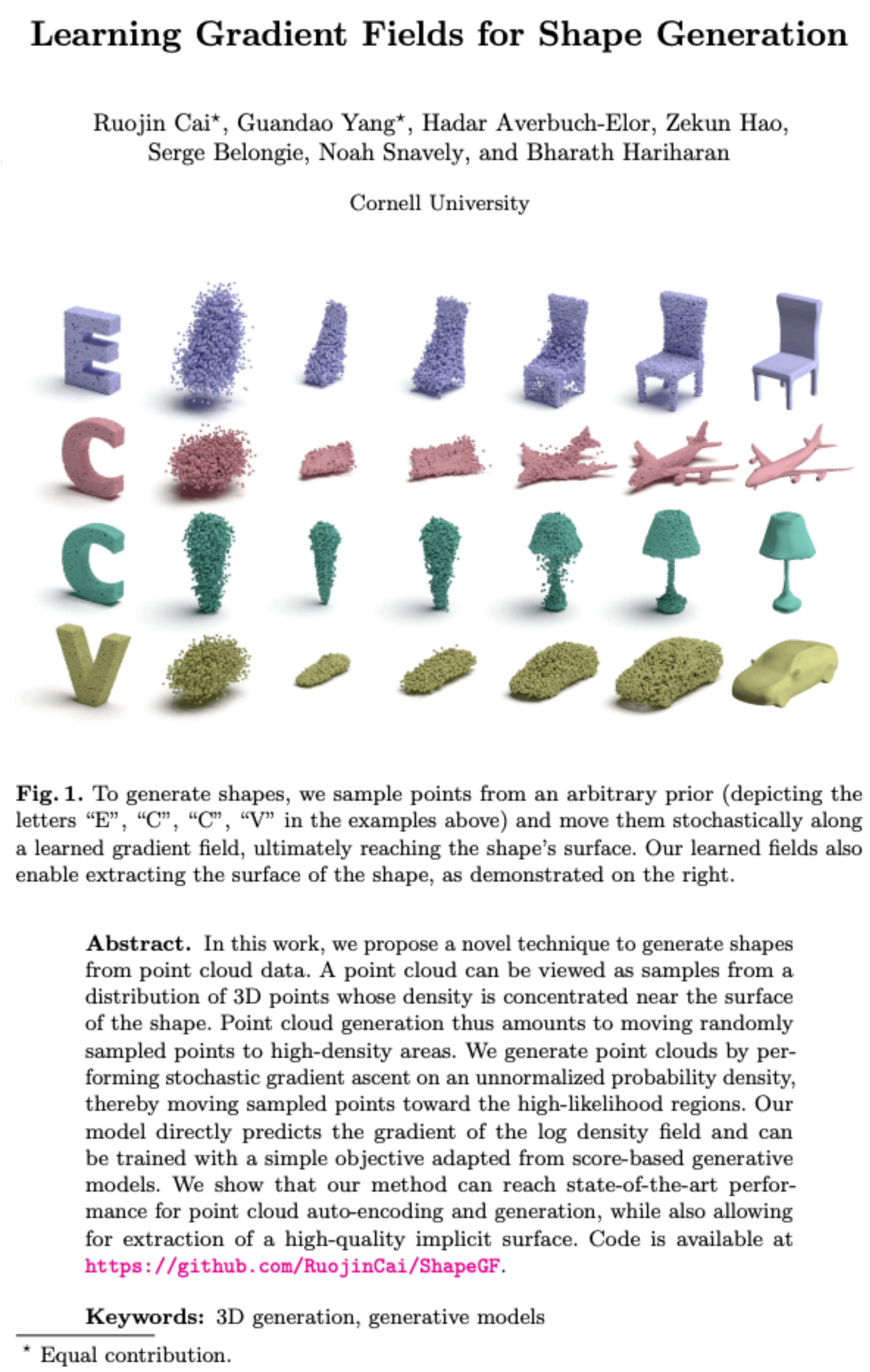
Point cloud creation as stochastic gradient ascend



Gradient field

Complicated topologies and non-watertight mesh




Learning Gradient Fields for Shape Generation

Ruojin Cai*, Guandao Yang*, Hadar Averbuch-Elor, Zekun Hao,
Serge Belongie, Noah Snavely, and Bharath Hariharan

Cornell University

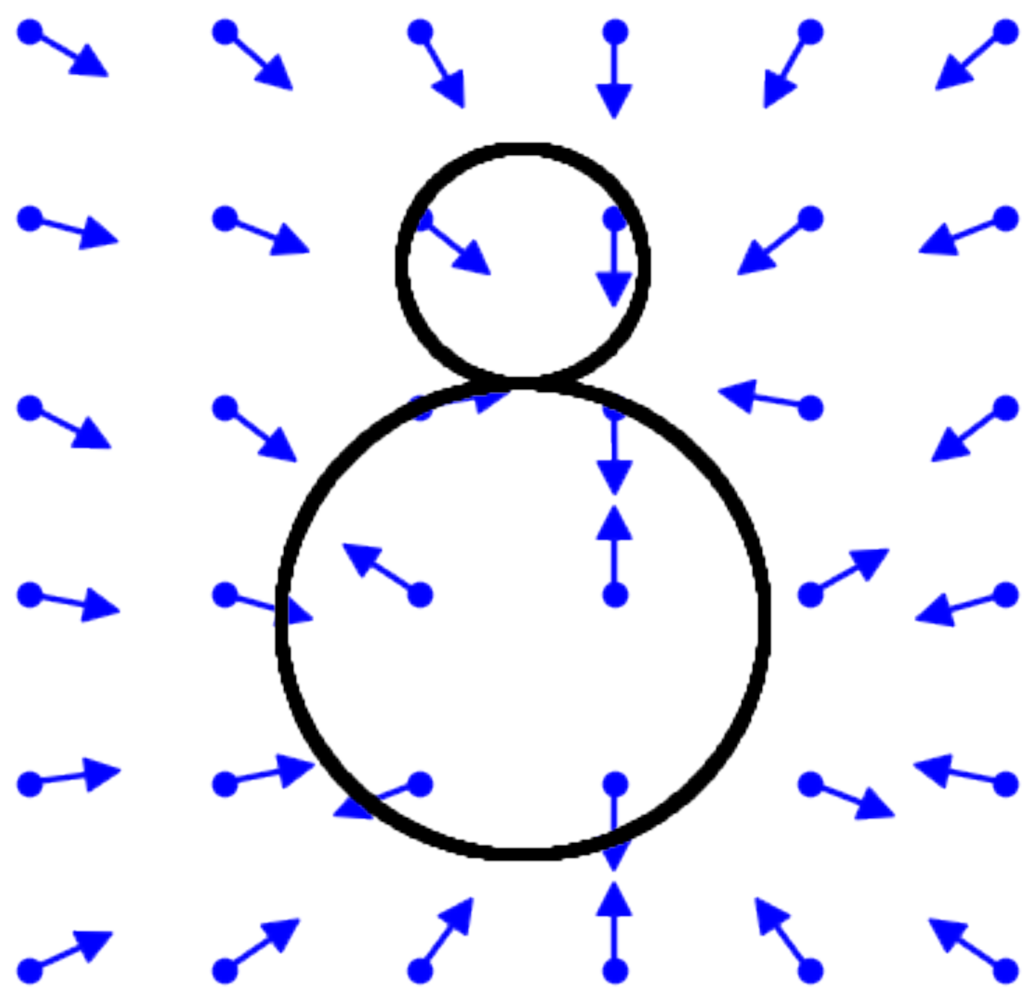
Fig. 1. To generate shapes, we sample points from an arbitrary prior (depicting the letters "E", "C", "C", "V" in the examples above) and move them stochastically along a learned gradient field, ultimately reaching the shape's surface. Our learned fields also enable extracting the surface of the shape, as demonstrated on the right.

Abstract. In this work, we propose a novel technique to generate shapes from point cloud data. A point cloud can be viewed as samples from a distribution of 3D points whose density is concentrated near the surface of the shape. Point cloud generation thus amounts to moving randomly sampled points to high-density areas. We generate point clouds by performing stochastic gradient ascent on an unnormalized probability density, thereby moving sampled points toward the high-likelihood regions. Our model directly predicts the gradient of the log density field and can be trained with a simple objective adapted from score-based generative models. We show that our method can reach state-of-the-art performance for point cloud auto-encoding and generation, while also allowing for extraction of a high-quality implicit surface. Code is available at https://github.com/RuojinCai/ShapeGF.

Keywords: 3D generation, generative models

* Equal contribution.

(Cai et. al., 2020)




Ruojin Cai


Zekun Hao

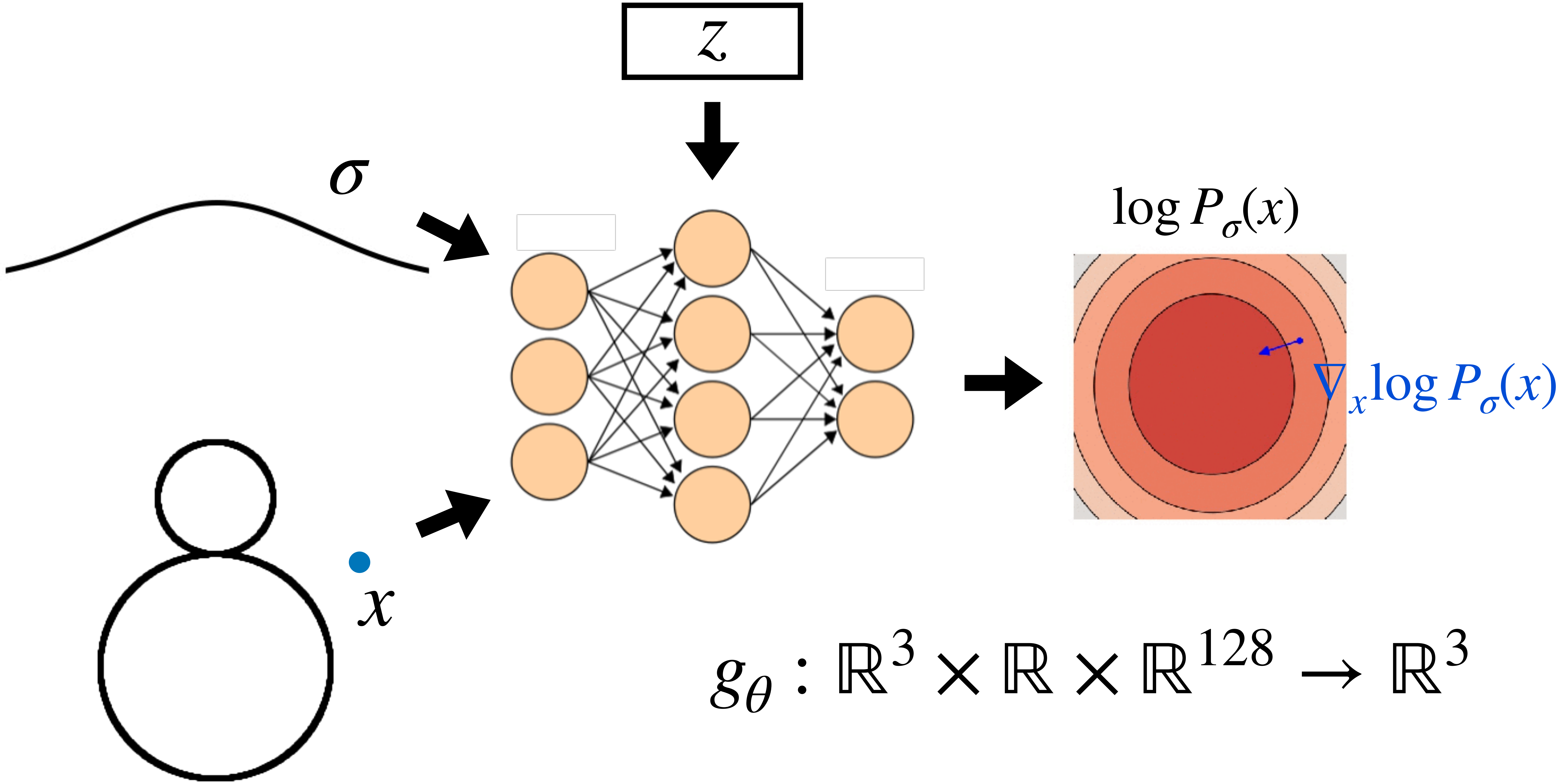

Hadar Averbunch-Elor


Noah Snavely


Serge Belongie

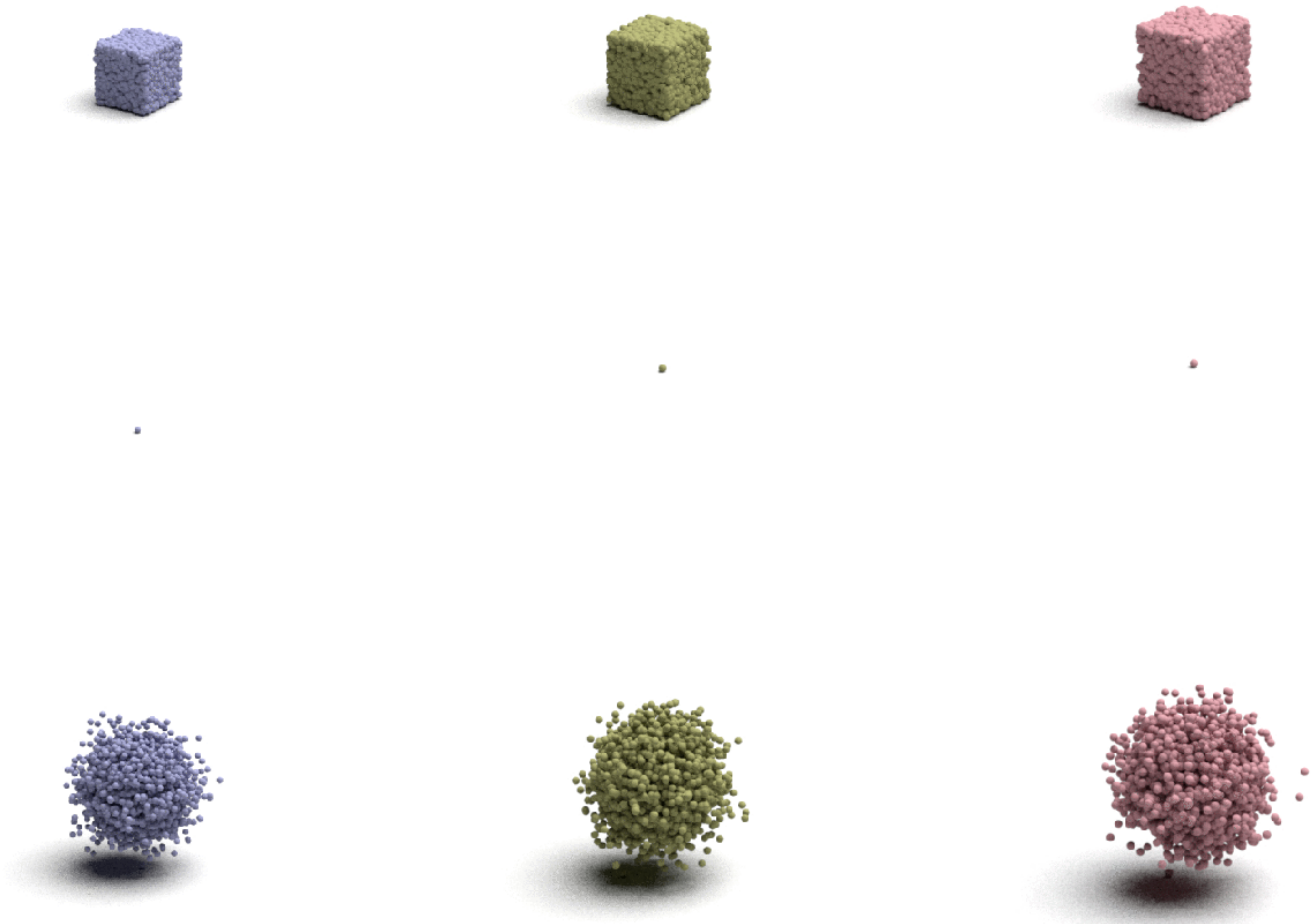

Bharath Hariharan

Learning a conditional neural gradient field

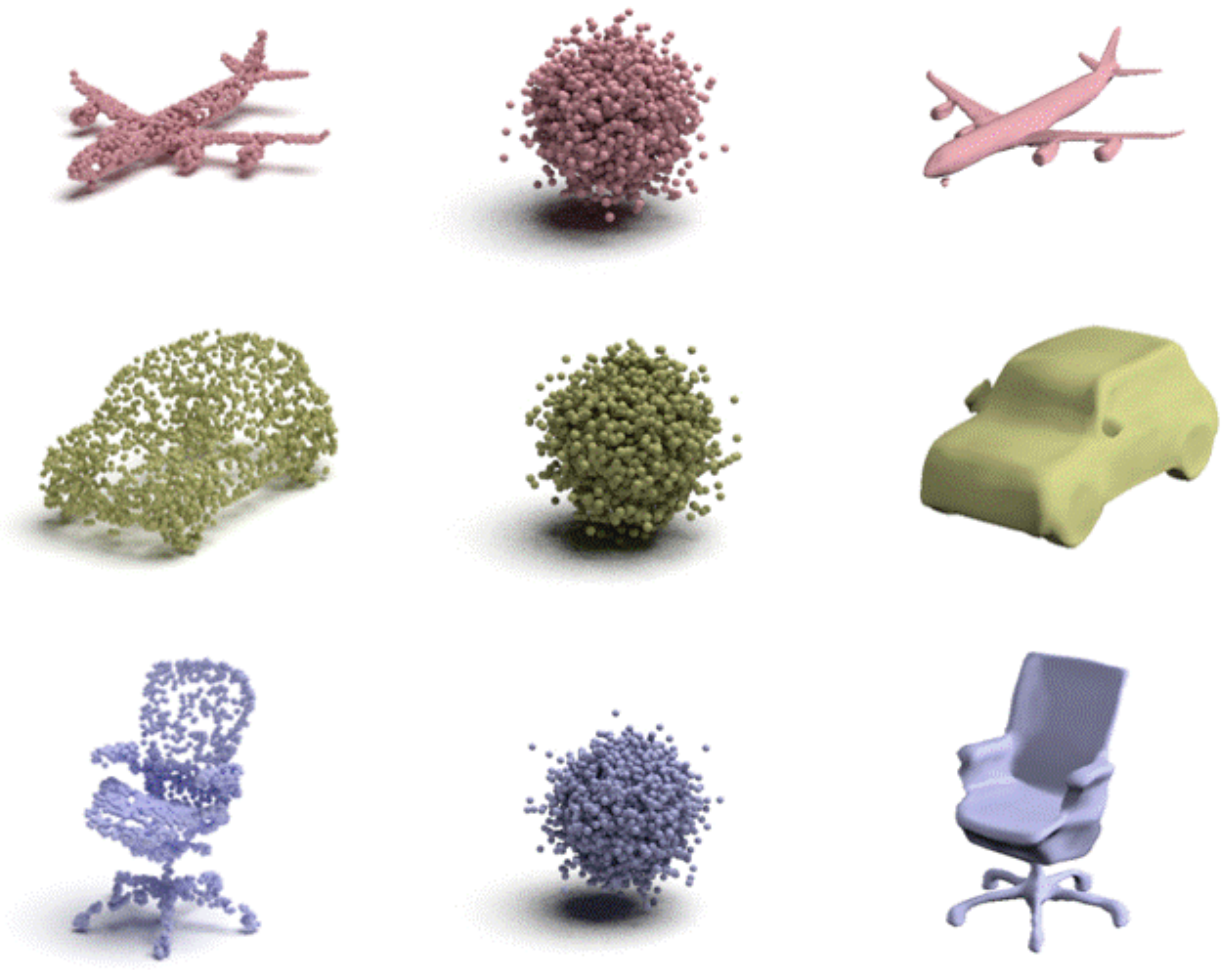


$$g_\theta : \mathbb{R}^3 \times \mathbb{R} \times \mathbb{R}^{128} \rightarrow \mathbb{R}^3$$

Generation results



Auto-encoding and surface extraction



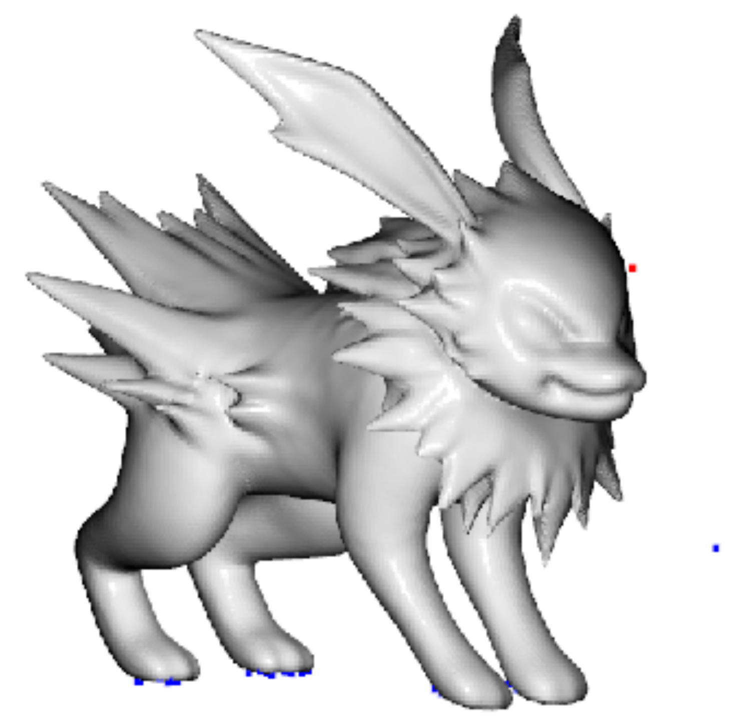
Input **Point cloud** **Surface**

Synthesis - Editing

Synthesis

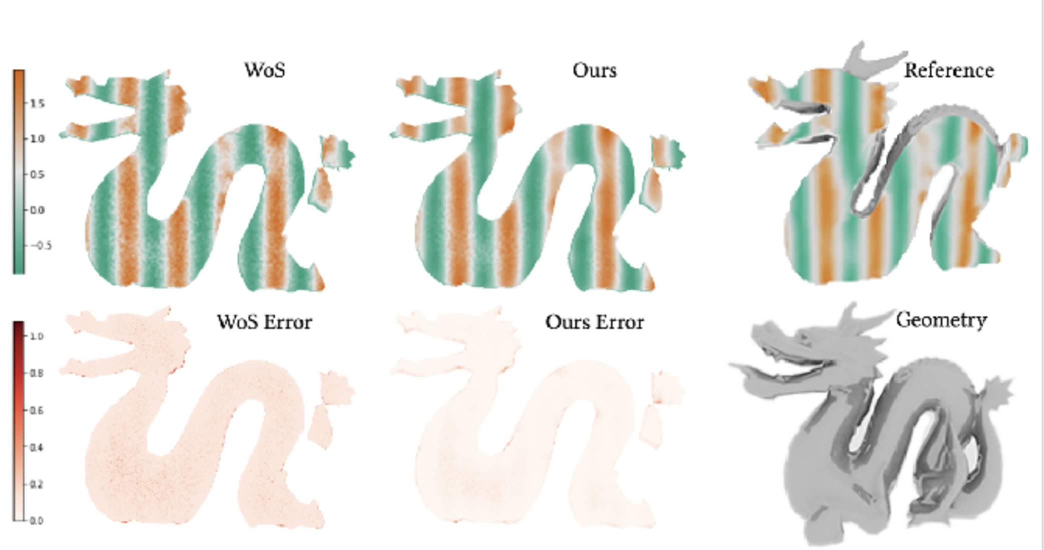
PointFlow (ICCV 2019)

ShapeGF (ECCV 2020)

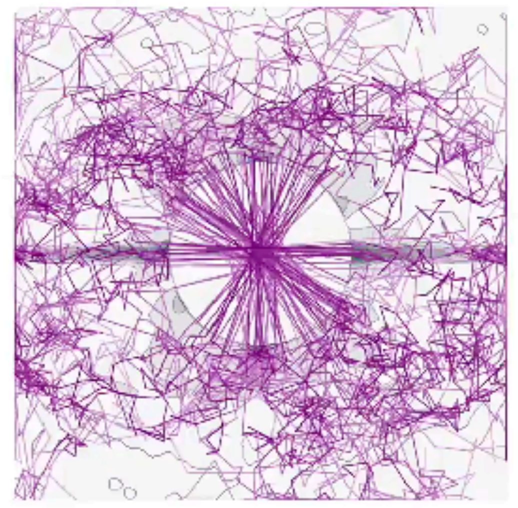


NFGP (NeurIPS 2021)

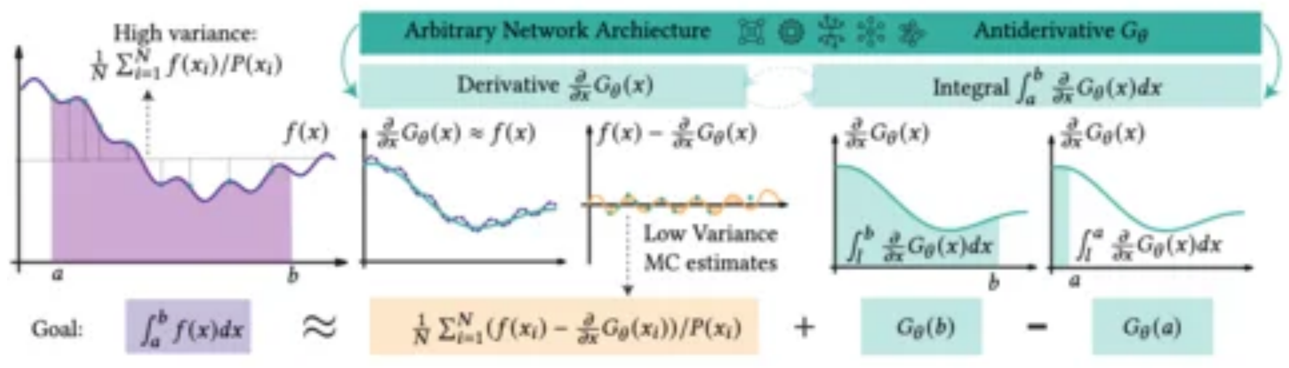
Analysis



Neural cache (SIG Asia 2023)



Symmetry (SIG Asia 2024)



NCV (SIGRAPH 2024)

2019 2020 2021 2023 2024

Elastic Deformation

Computer Graphics, Volume 21, Number 4, July 1987

Elastically Deformable Models

Demetri Terzopoulos[†]
John Platt[‡]
Alan Barr[‡]
Kurt Fleischer[†]

[†]Schlumberger Palo Alto Research, 3340 Hillview Avenue, Palo Alto, CA 94304
[‡]California Institute of Technology, Pasadena, CA 91125

Abstract: The theory of elasticity describes deformable materials such as rubber, cloth, paper, and flexible metals. We employ elasticity theory to construct differential equations that model the behavior of non-rigid curves, surfaces, and solids as a function of time. Elastically deformable models are active: they respond in a natural way to applied forces, constraints, ambient media, and impenetrable obstacles. The models are fundamentally dynamic and realistic animation is created by numerically solving their underlying differential equations. Thus, the description of shape and the description of motion are unified.

Keywords: Modeling, Deformation, Elasticity, Dynamics, Animation, Simulation

CR categories: G.1.8—Partial Differential Equations; I.3.5—Computational Geometry and Object Modeling (Curve, Surface, Solid, and Object Representations); I.3.7—Three-Dimensional Graphics and Realism

1. Introduction

Methods to formulate and represent instantaneous shapes of objects are central to computer graphics modeling. These methods have been particularly successful for modeling rigid objects whose shapes do not change over time. This paper develops an approach to modeling which incorporates the physically-based dynamics of flexible materials into the purely geometric models which have been used traditionally. We propose models based on elasticity theory which conveniently represent the shape and motion of deformable materials, especially when these materials interact with other physically-based computer graphics objects.

1.1. Physical Models versus Kinematic Models

Most traditional methods for computer graphics modeling are kinematic; that is, the shapes are compositions of geometrically or algebraically defined primitives. Kinematic models are passive because they do not interact with each other or with external forces. The models are either stationary or are subjected to motion according to prescribed trajectories. Expertise is required to create natural and pleasing dynamics with passive models.

As an alternative, we advocate the use of active models in computer graphics. Active models are based on principles of mathematical physics [5]. They react to applied forces (such as gravity), to constraints (such as linkages), to ambient media (such as viscous fluids), or to impenetrable obstacles (such as supporting surfaces) as one would expect real, physical objects to react.

This paper develops models of deformable curves, surfaces, and solids which are based on simplifications of elasticity theory. By simulating physical properties such as tension and rigidity, we can model static shapes exhibited by a wide range of deformable objects, including string, rubber, cloth, paper, and flexible metals. Furthermore, by including physical properties such as mass and damping, we can simulate the dynamics of these objects. The simulation involves numerically solving the partial differential equations that govern the evolving shape of the deformable object and its motion through space.

The dynamic behavior inherent to our deformable models significantly simplifies the animation of complex objects. Consider the graphical representation of a coiled telephone cord. The traditional approach has been to represent the instantaneous shape of the cord as a mesh assembly of bicubic spline patches or polygons. Making the cord move plausibly is a nontrivial task. In contrast, our deformable models can provide a physical representation of the cord which exhibits natural dynamics as it is subjected to external forces and constraints.

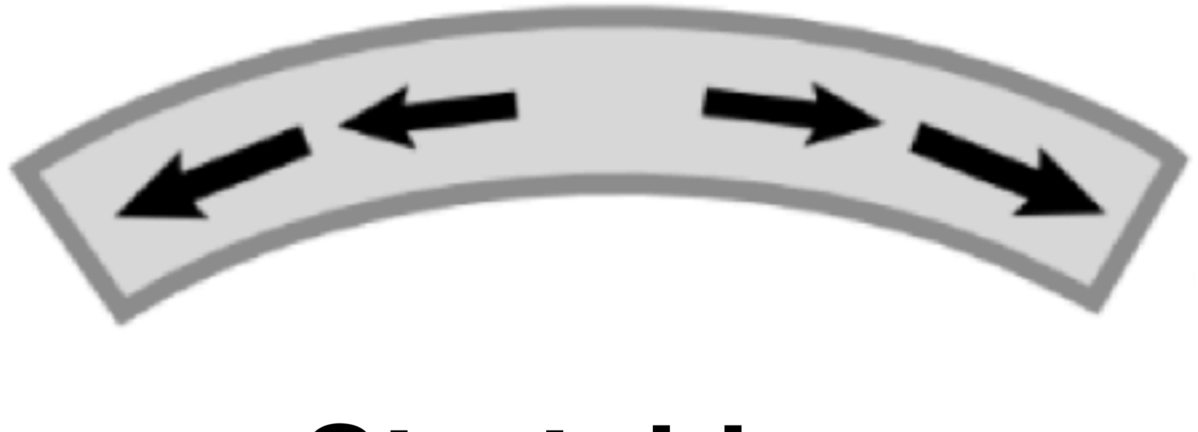
1.2. Outline

The remainder of the paper develops as follows: Section 2 discusses the connections of our work to other physical models in computer graphics. Section 3 gives differential equations of motion describing the dynamic behavior of deformable models under the influence of external forces. Section 4 contains an analysis of deformation and defines deformation energies for curve, surface, and solid models. Section 5 lists various external forces that can be applied to deformable models to produce animation. Section 6 describes our implementation of deformable models. Section 7 presents simulations illustrating the application of deformable models. Section 8 discusses our work in progress.

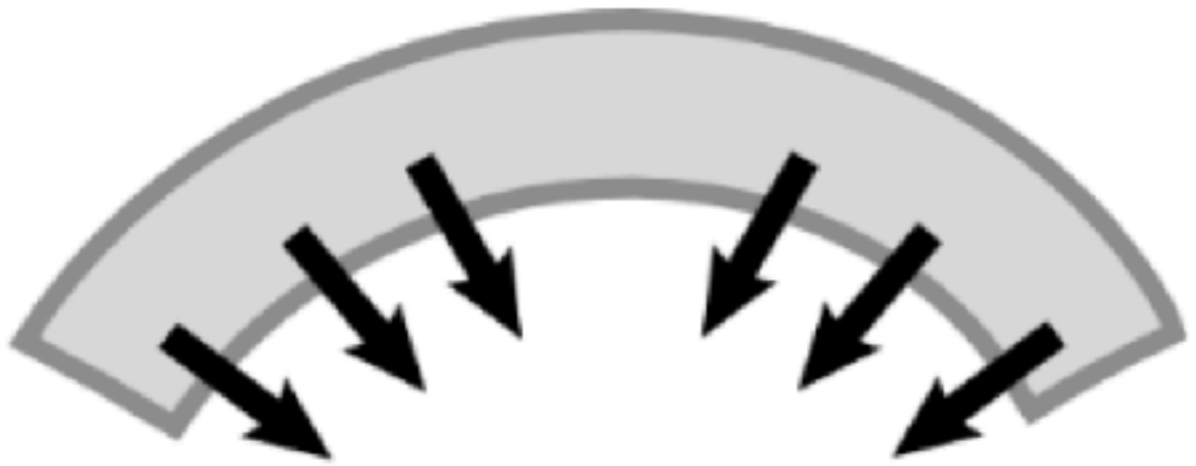
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1987 ACM-0-89791-227-6/87/007/0205 $00.75

Terzopoulos et. al., 1987;
Sorkine and Alexa, 2007;
Levi and Gotsman, 2015

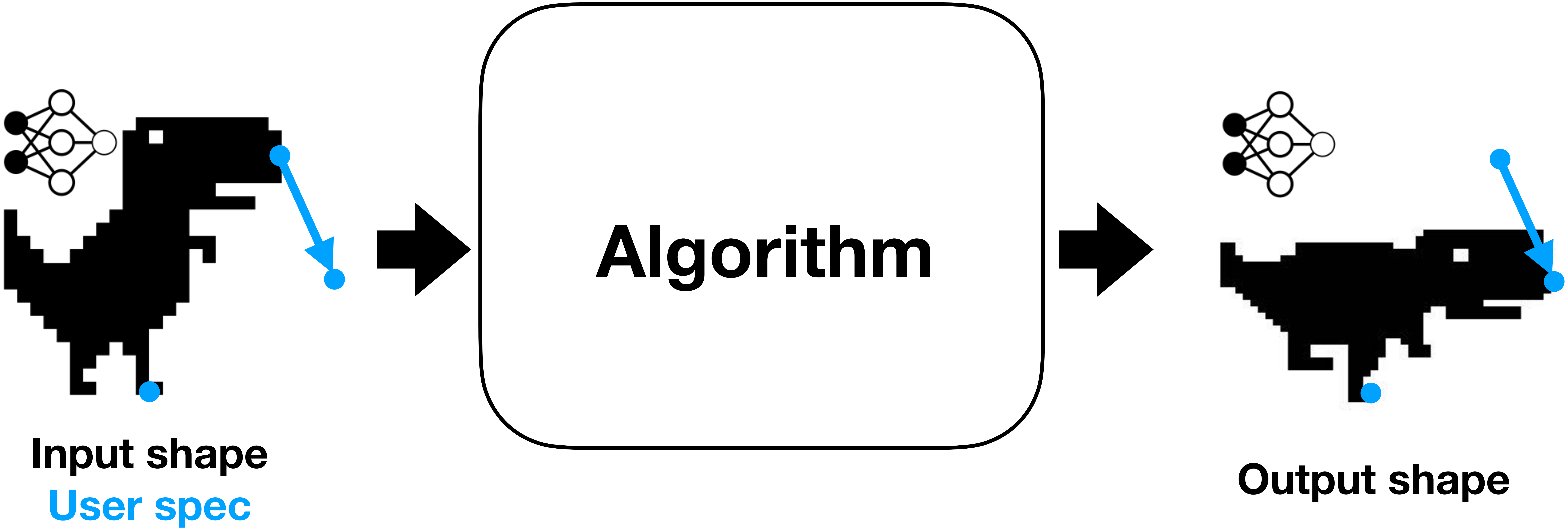


Stretching

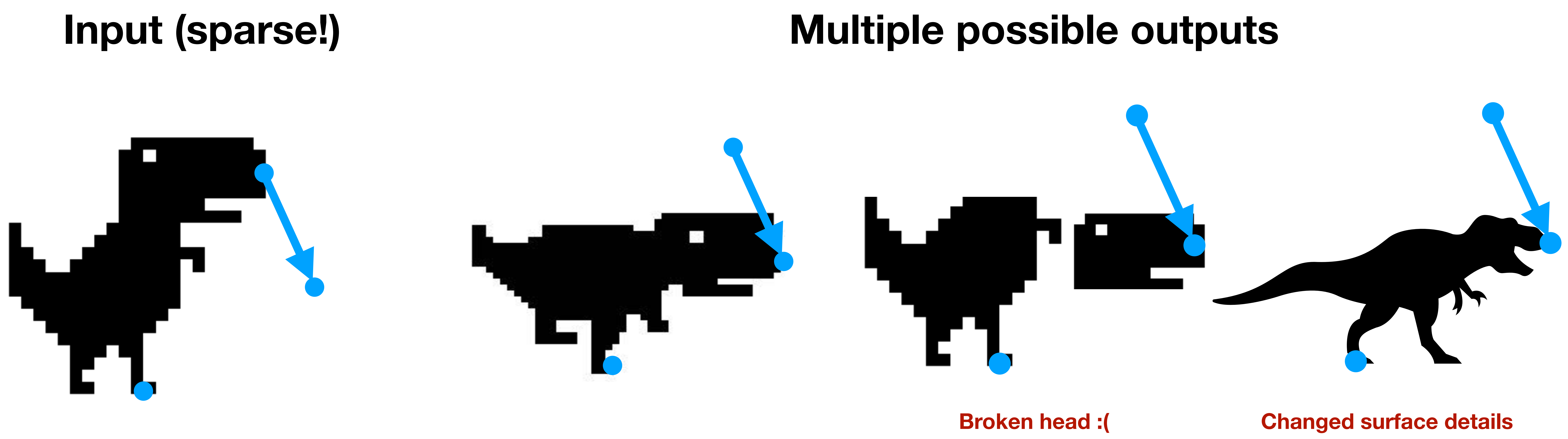


Bending

Elastic Deformation with Neural Fields

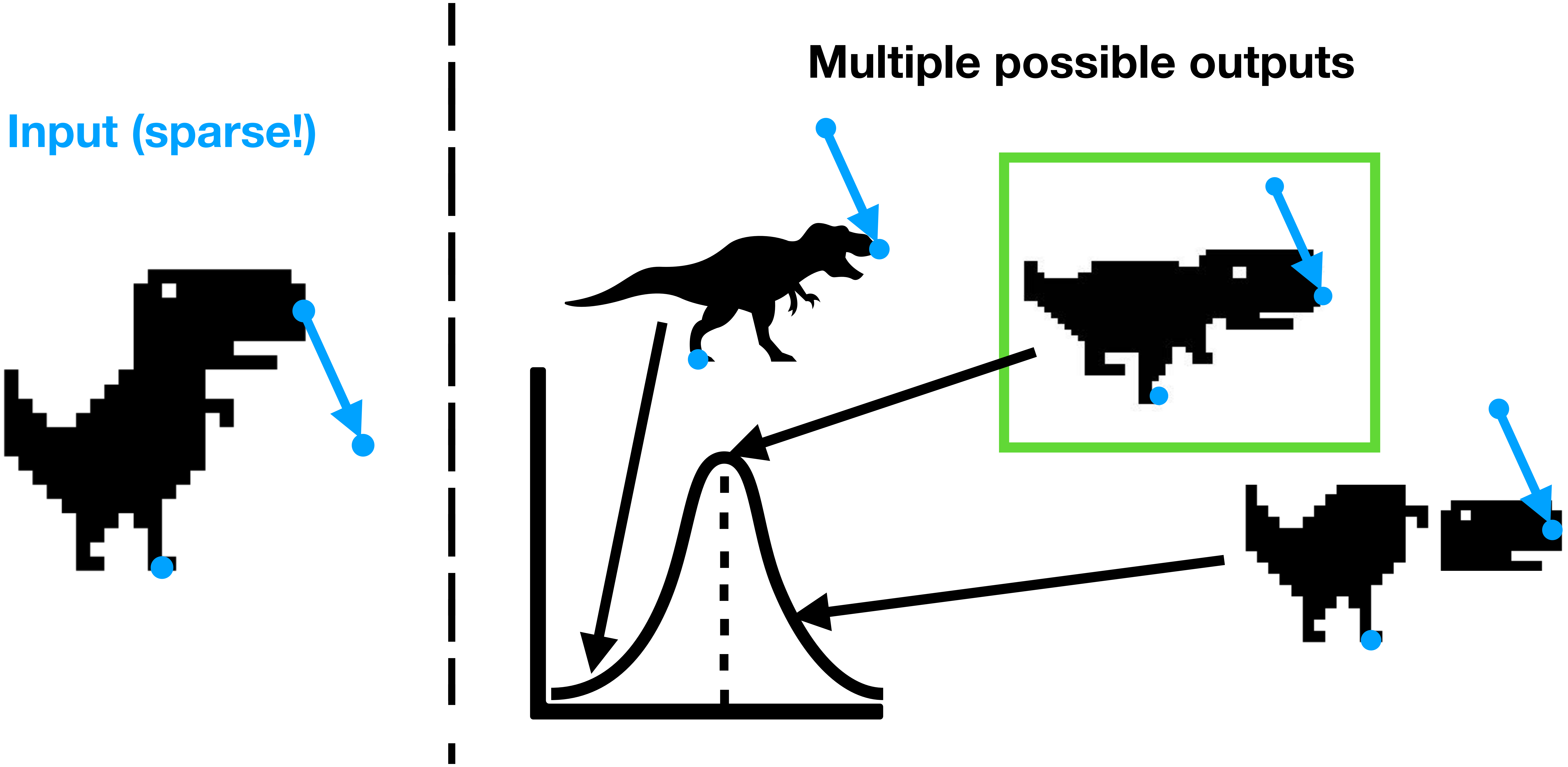


Elastic Deformation is under constrained



Editing algorithm requires prior knowledge

Shape Priors for Editing



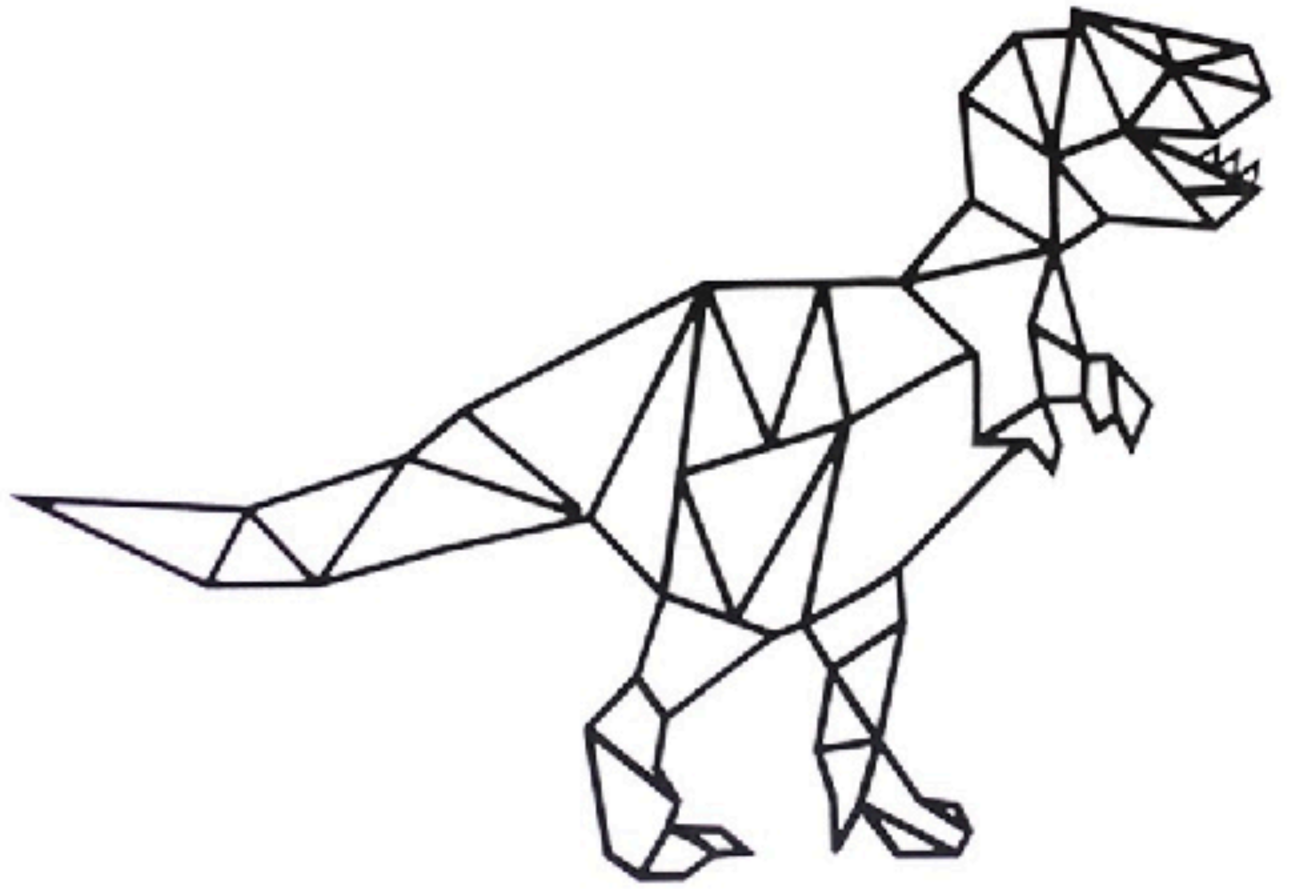
Quantify Priors

$$f : (u, v) \mapsto (x, y, z)$$

Normal $\quad \mathbf{n} = \dfrac{f_u \times f_v}{\| f_u \times f_v \|}$

Curvature $\quad \mathbf{II} = \begin{bmatrix} f_{uu}^T \mathbf{n} & f_{uv}^T \mathbf{n} \\ f_{vu}^T \mathbf{n} & f_{vv}^T \mathbf{n} \end{bmatrix} \quad \kappa = \dfrac{|\,\mathbf{II}\,|}{|\,\mathbf{I}\,|}$

Quantify Prior - Mesh



$$V = \{(x_i, y_i, z_i)\}_{i=1}^{n}$$
$$F = \{(u, v, w) \mid 1 \leq u, v, w, \leq n\}$$

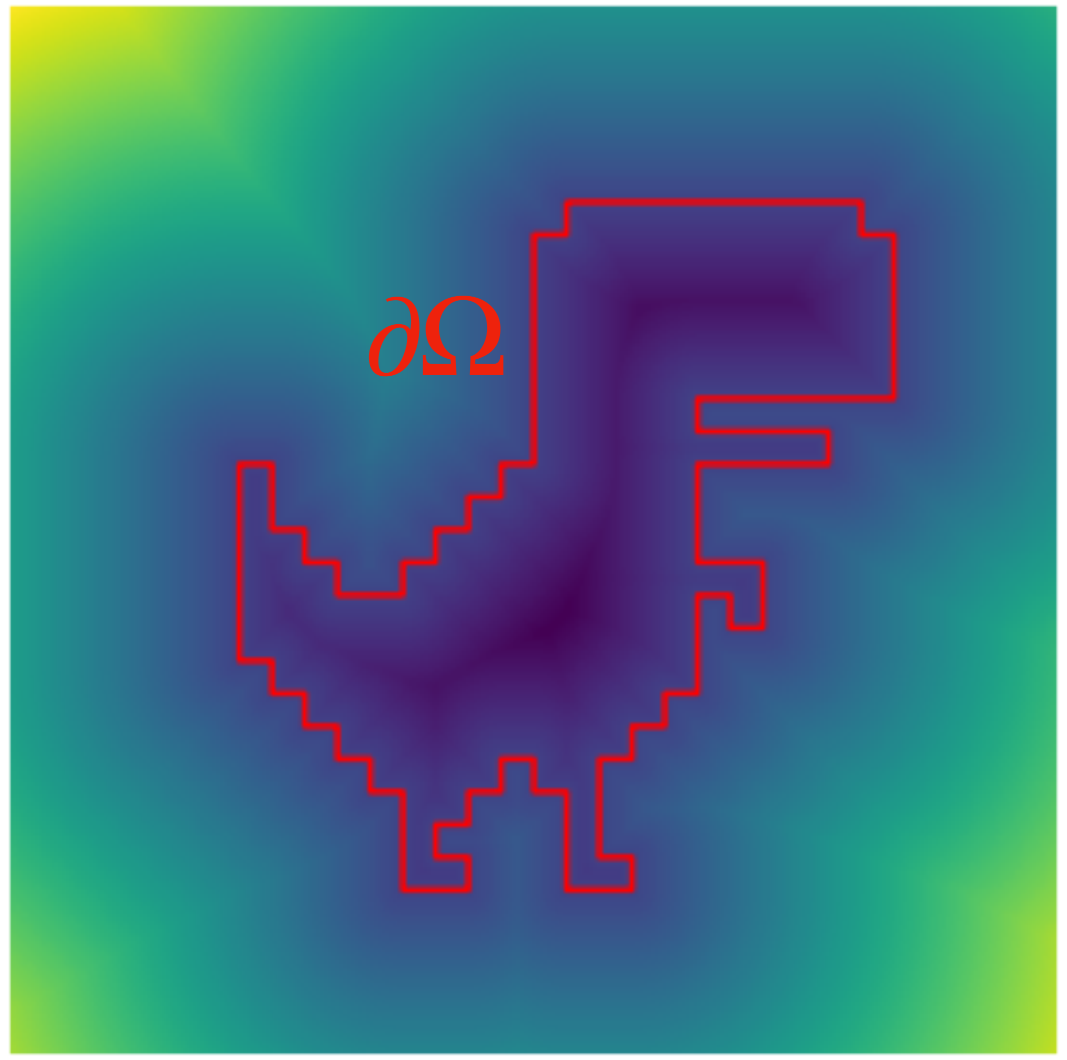
Normal

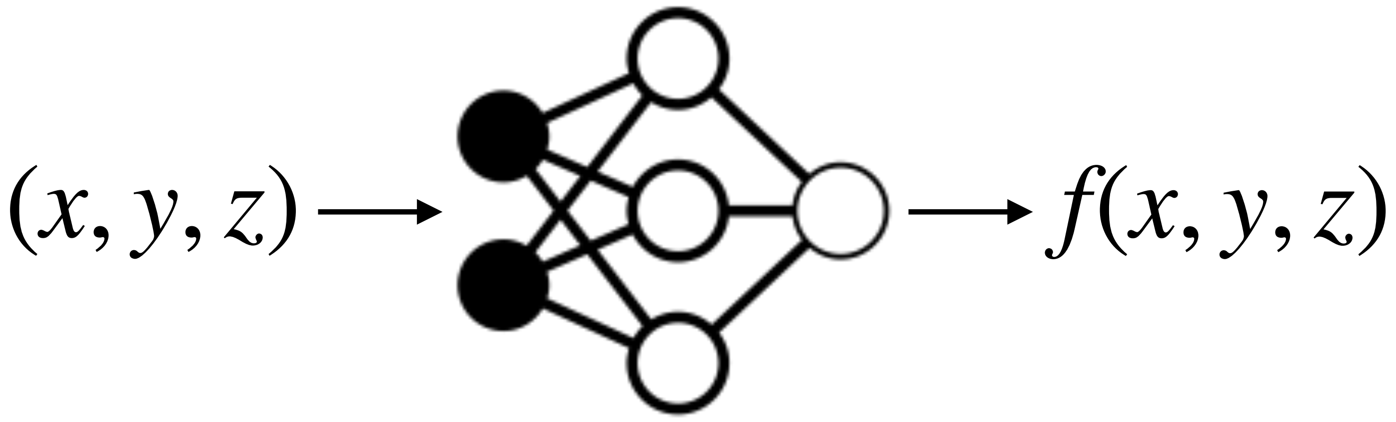
$$\mathbf{n}_{i,j,k} = \frac{(V_j - V_i) \times (V_k - V_i)}{|(V_j - V_i) \times (V_k - V_i)|}$$

Curvature

$$\kappa_i = \frac{1}{2} \sum_{j \in \mathcal{N}(i)} (\cot \alpha_{ij} + cot\beta_{ij})(V_i - V_j)$$

Quantify Prior - Neural Fields





$$\partial\Omega = \{(x, y, z) \mid f(x, y, z) = 0\}$$

Normal

Curvature

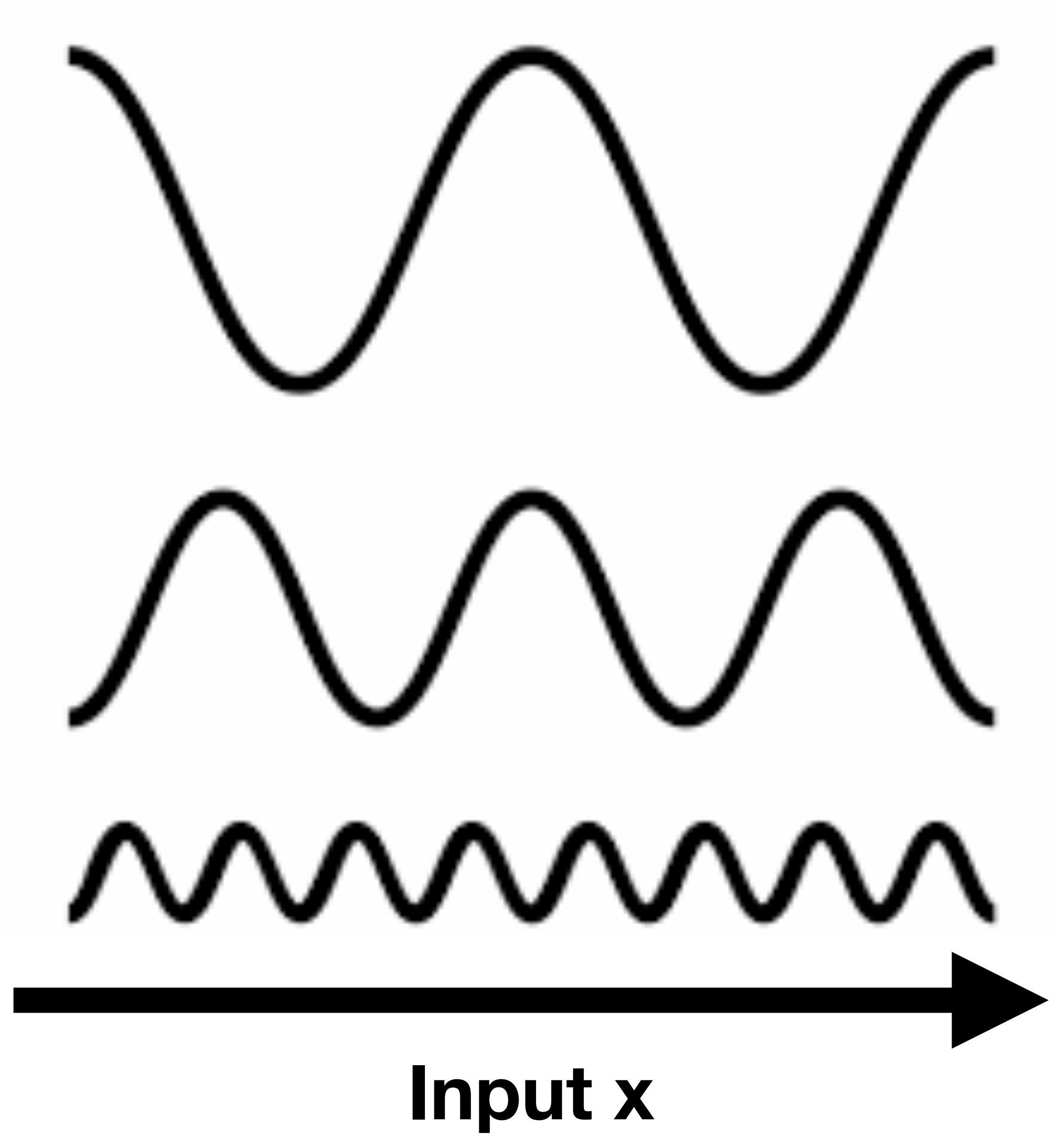
?

Differentiability of Neural Fields

PyTorch

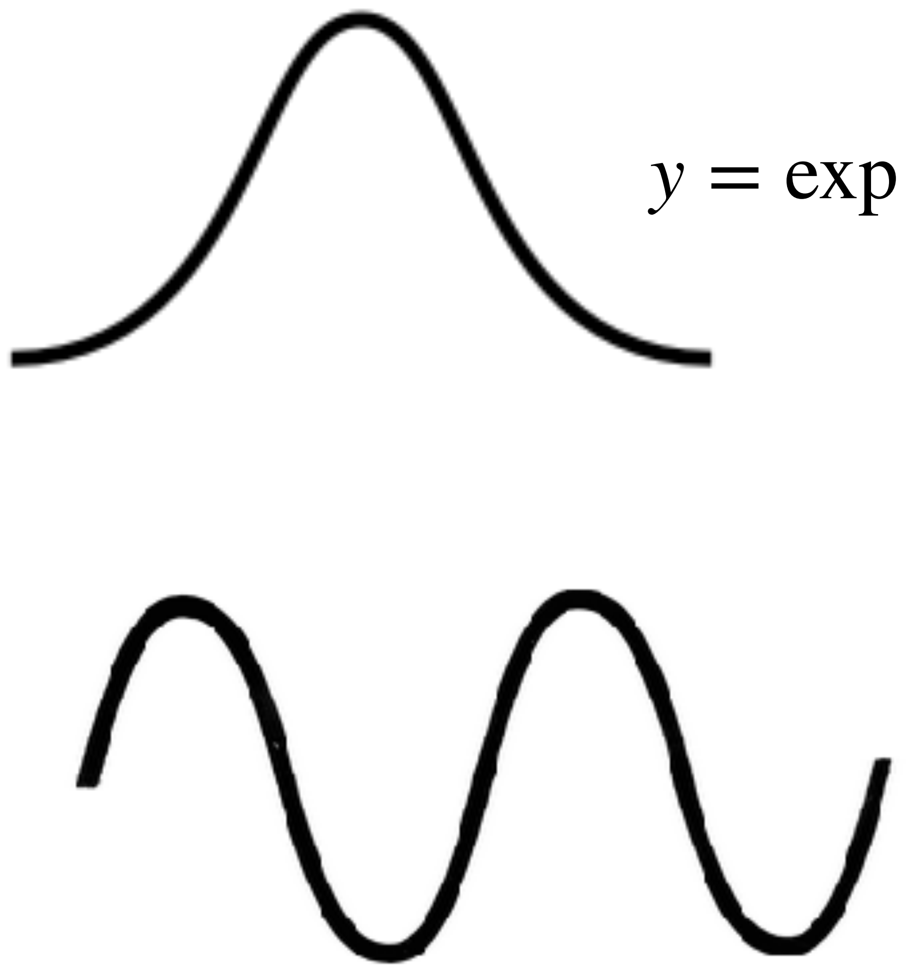
TensorFlow

AutoGrad Frameworks



Positional Encoding

(Tannic et al., 2020, Mildenhall et al., 2020)



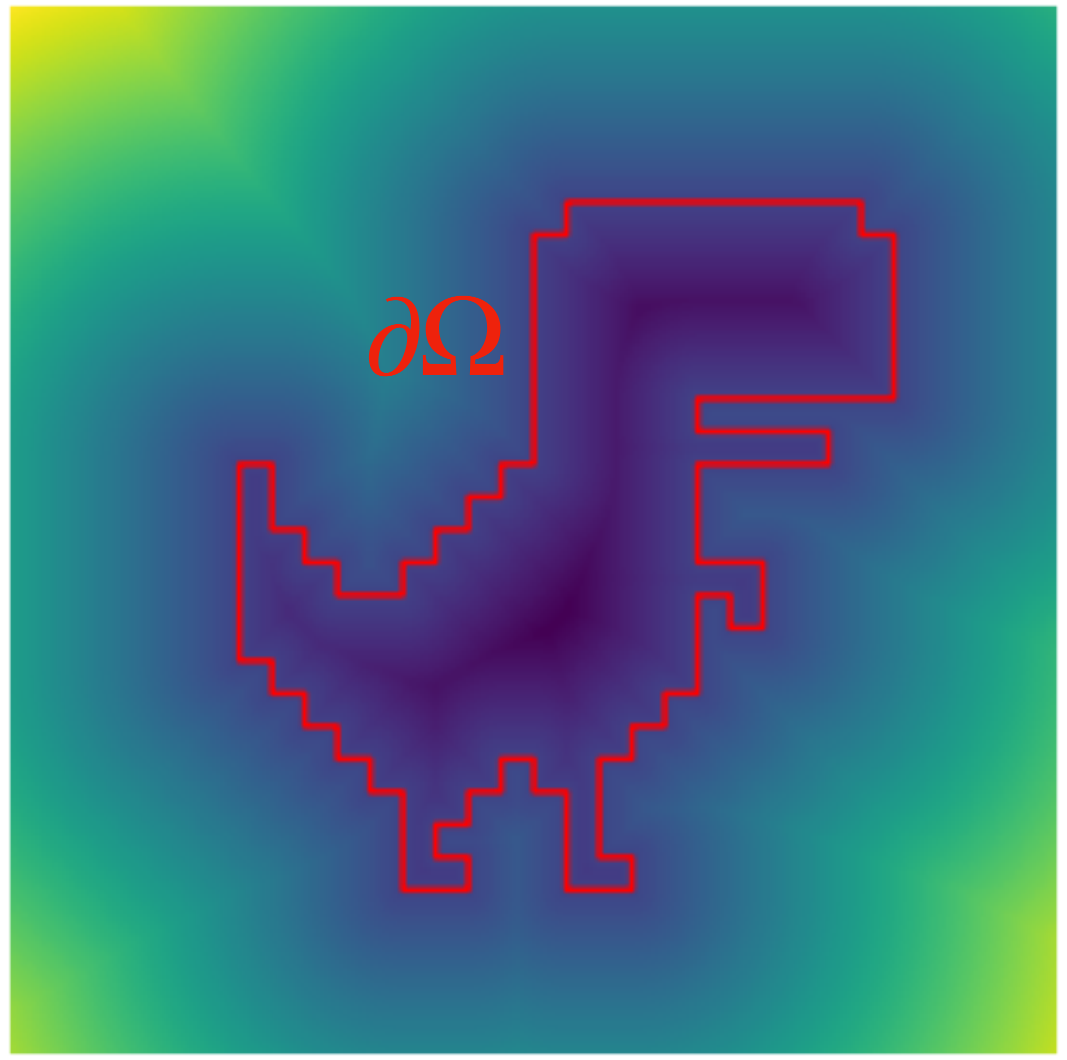
$$y = \exp\left(-\frac{(x-\mu)^2}{\sigma}\right)$$

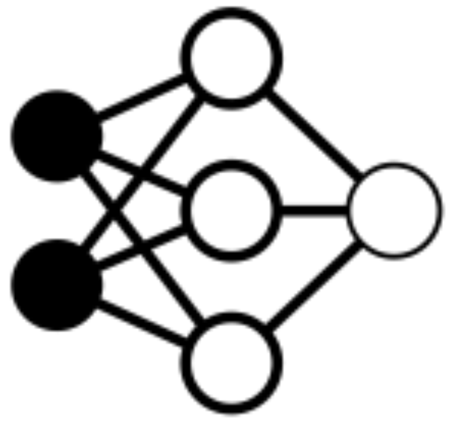
$$y = \sin(x)$$

Smooth Activation

(Sitzmann et al., 2020, Chng et al., 2022, Ramasinghe et al. 2022, Zheng et al., 2021, Fathony et al., 2021)

Quantify Prior - Neural Fields



$(x, y, z) \longrightarrow$  $\longrightarrow f(x, y, z)$

$$\partial\Omega = \{(x, y, z) \mid f(x, y, z) = 0\}$$

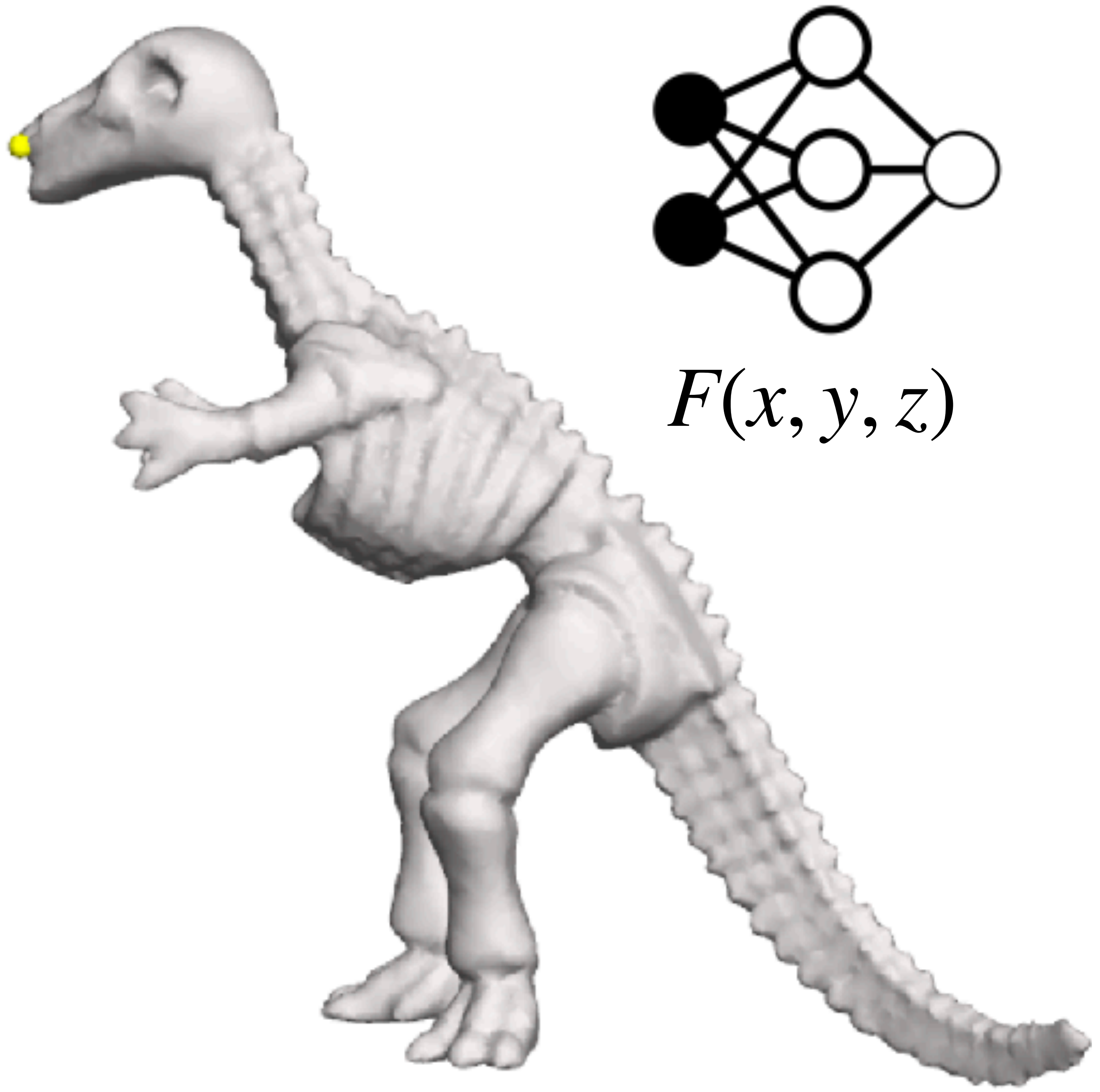
Normal

$$\mathbf{n}(\mathbf{x}) = \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}$$

Curvature

$$\kappa(\mathbf{x}) = -\frac{1}{2}\nabla \cdot \left(\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}\right)$$

Problem Setup

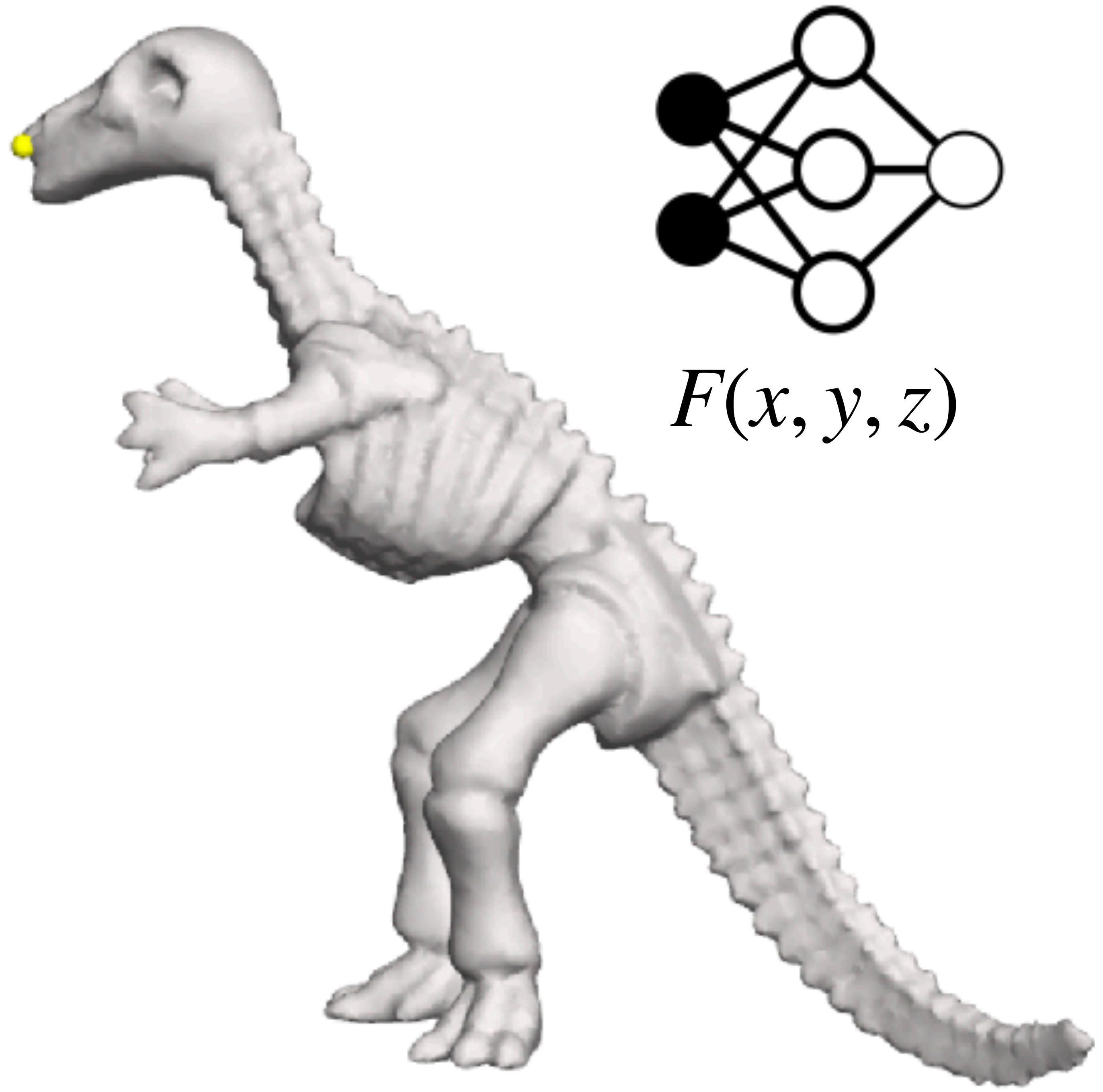


$F(x, y, z)$

$$F(\mathbf{x}) \approx \sigma(\mathbf{x}) \min_{\mathbf{p} \in \partial\Omega} \|\mathbf{p} - \mathbf{x}\|$$

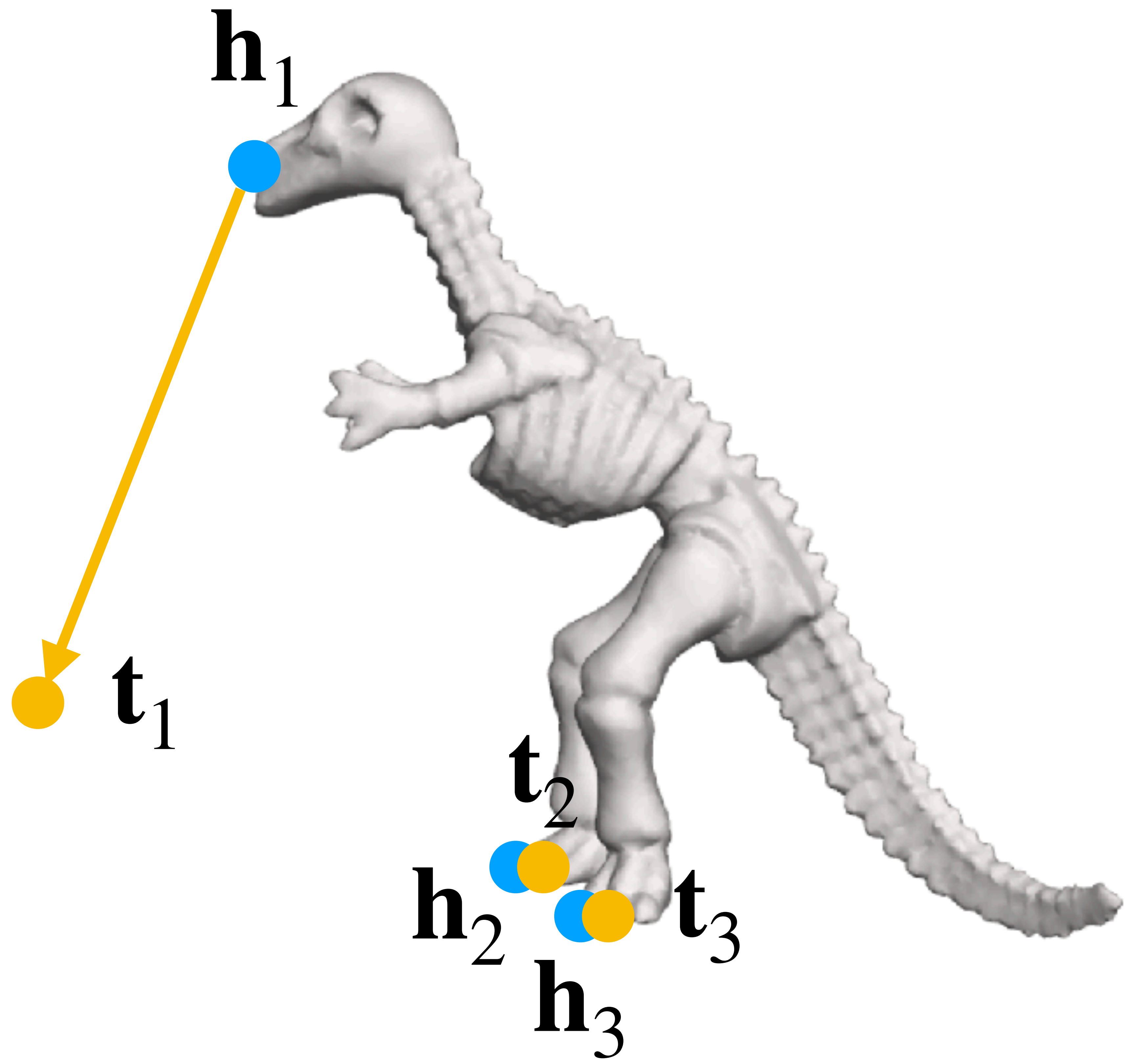
$$\sigma(\mathbf{x}) = \begin{cases} -1 & \textbf{if } \mathbf{x} \in \Omega \\ 1 & \textbf{otherwise} \end{cases}$$

Problem Setup



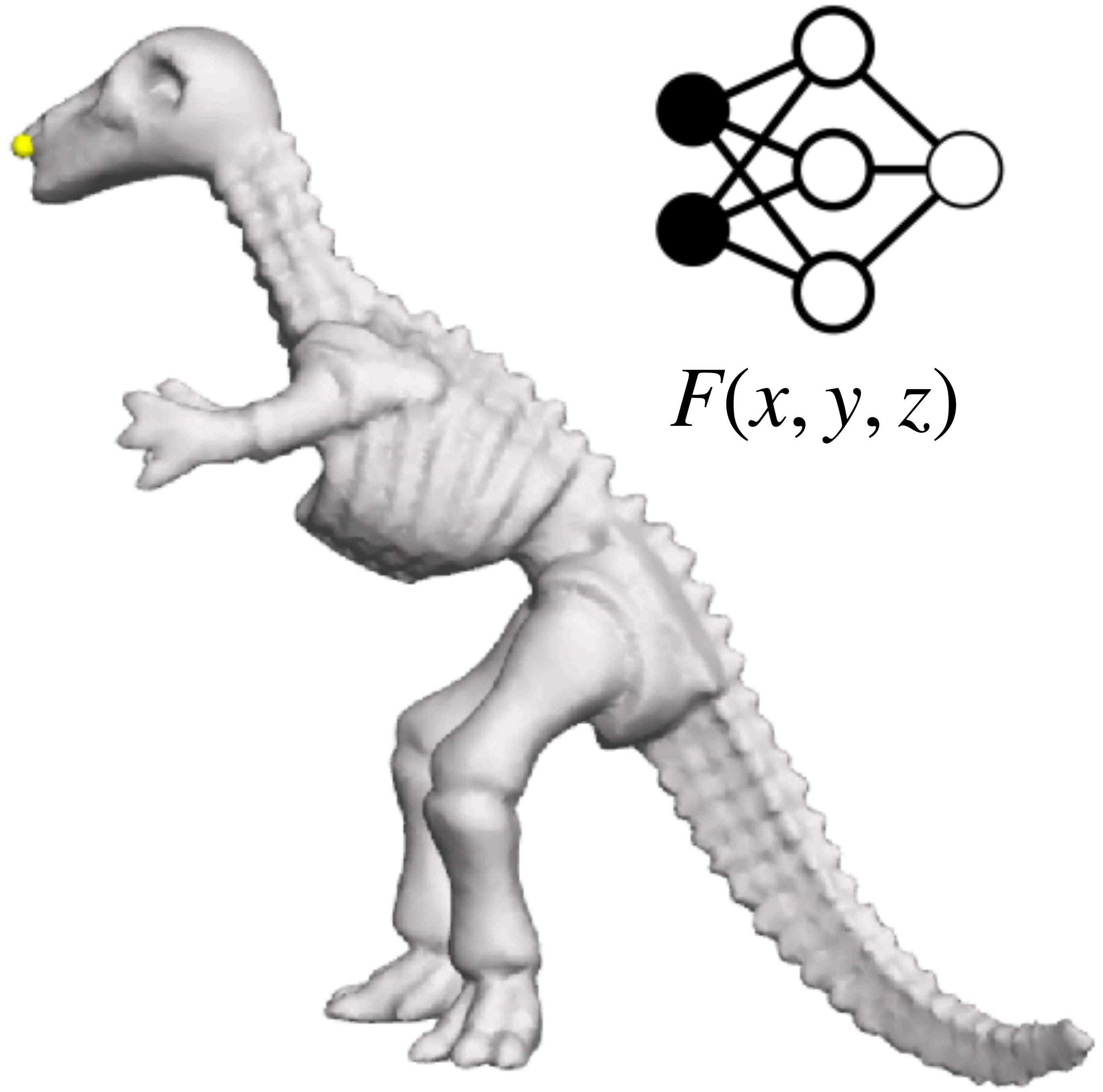
$$F(\mathbf{x}) \approx \sigma(\mathbf{x}) \min_{\mathbf{p}\in\partial\Omega} \|\mathbf{p} - \mathbf{x}\|$$

$$\sigma(\mathbf{x}) = \begin{cases} -1 & \textbf{if } \mathbf{x} \in \Omega \\ 1 & \textbf{otherwise} \end{cases}$$



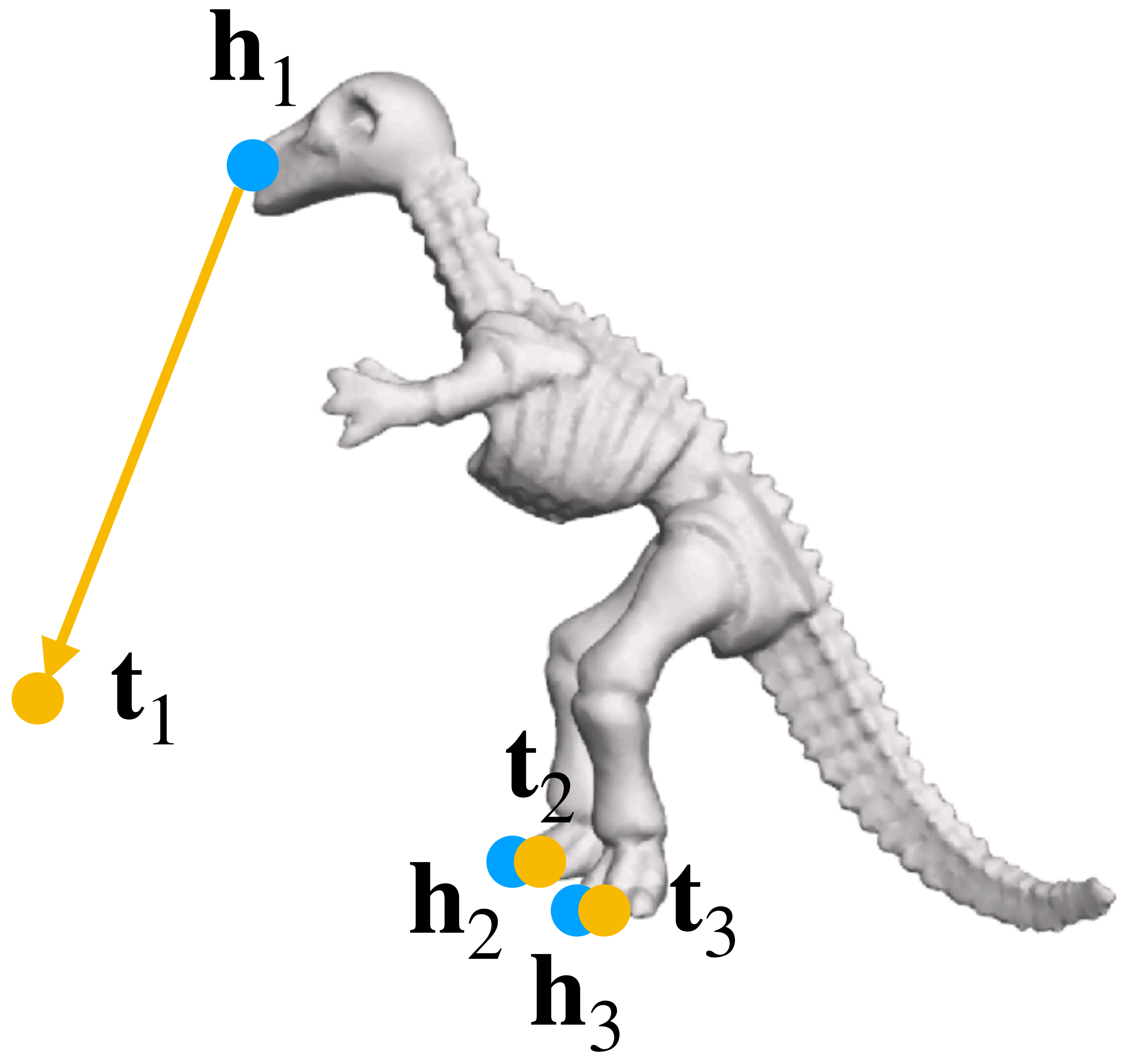
User specification

Problem Setup

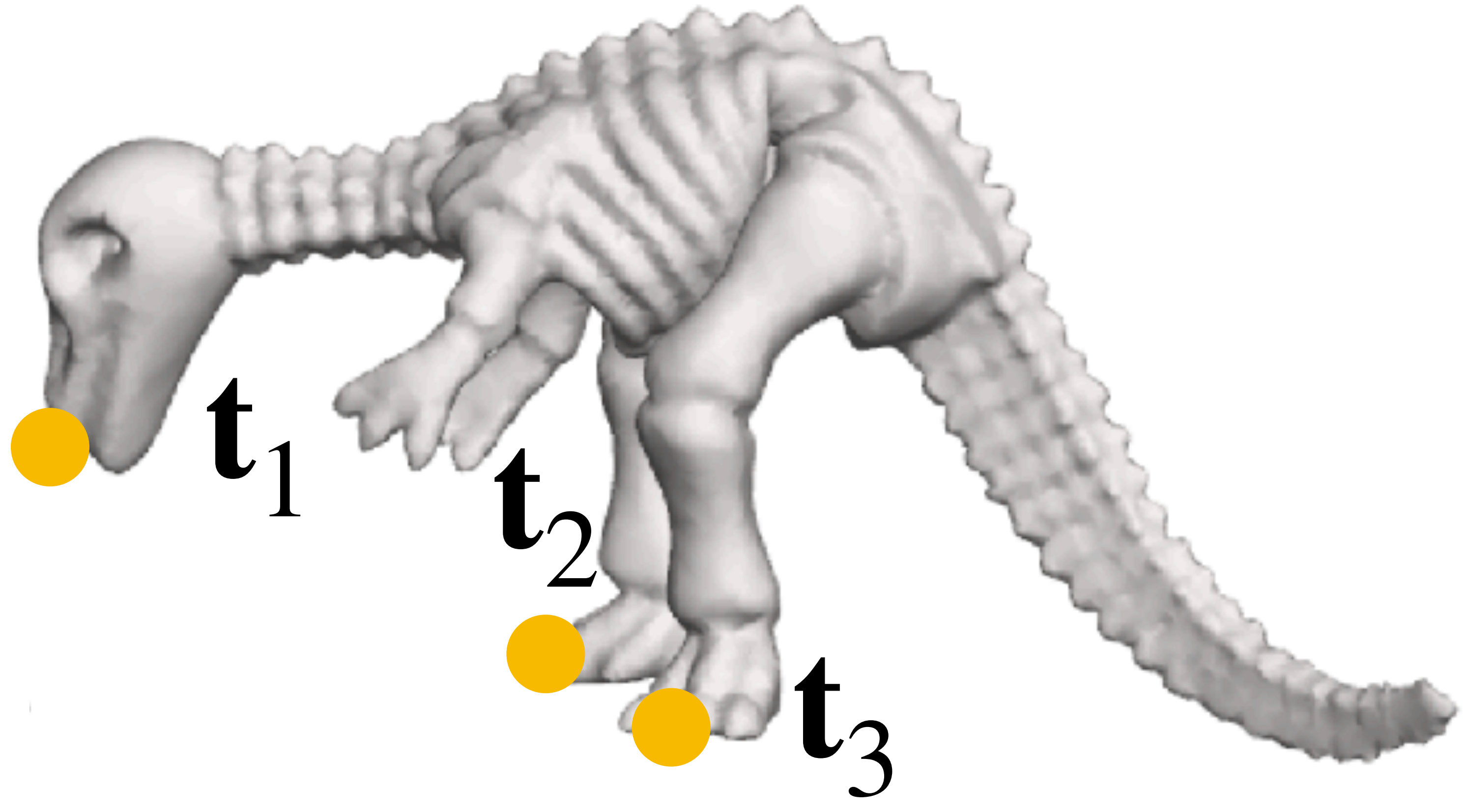


$$F(\mathbf{x}) \approx \sigma(\mathbf{x}) \min_{\mathbf{p} \in \partial\Omega} \|\mathbf{p} - \mathbf{x}\|$$

$$\sigma(\mathbf{x}) = \begin{cases} -1 & \textbf{if } \mathbf{x} \in \Omega \\ 1 & \textbf{otherwise} \end{cases}$$



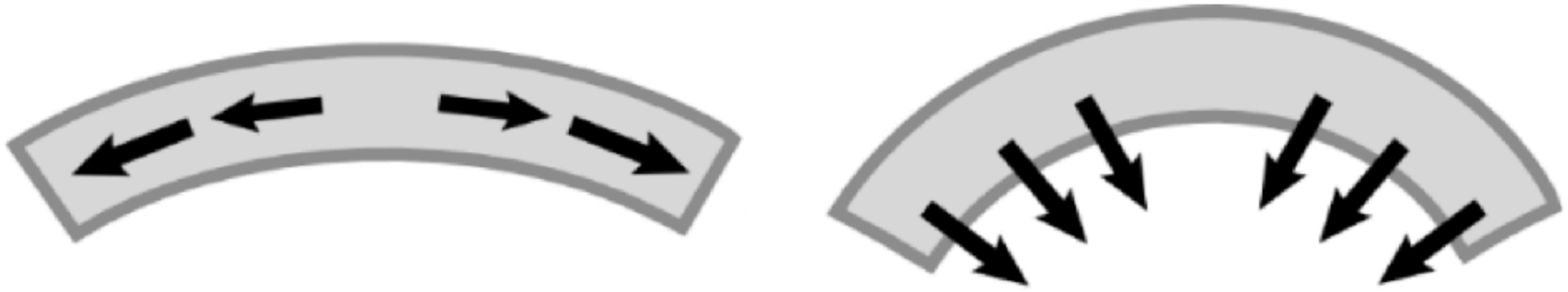
User specification



Desired Output

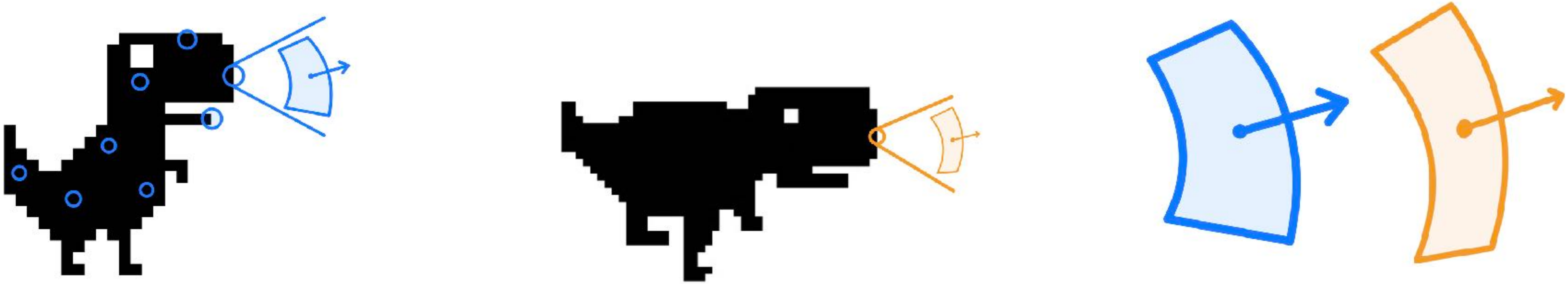
Natural Elastic Deformation

Elastic Deformation



Stretching

Bending



1. Sampling

2. Correspondences

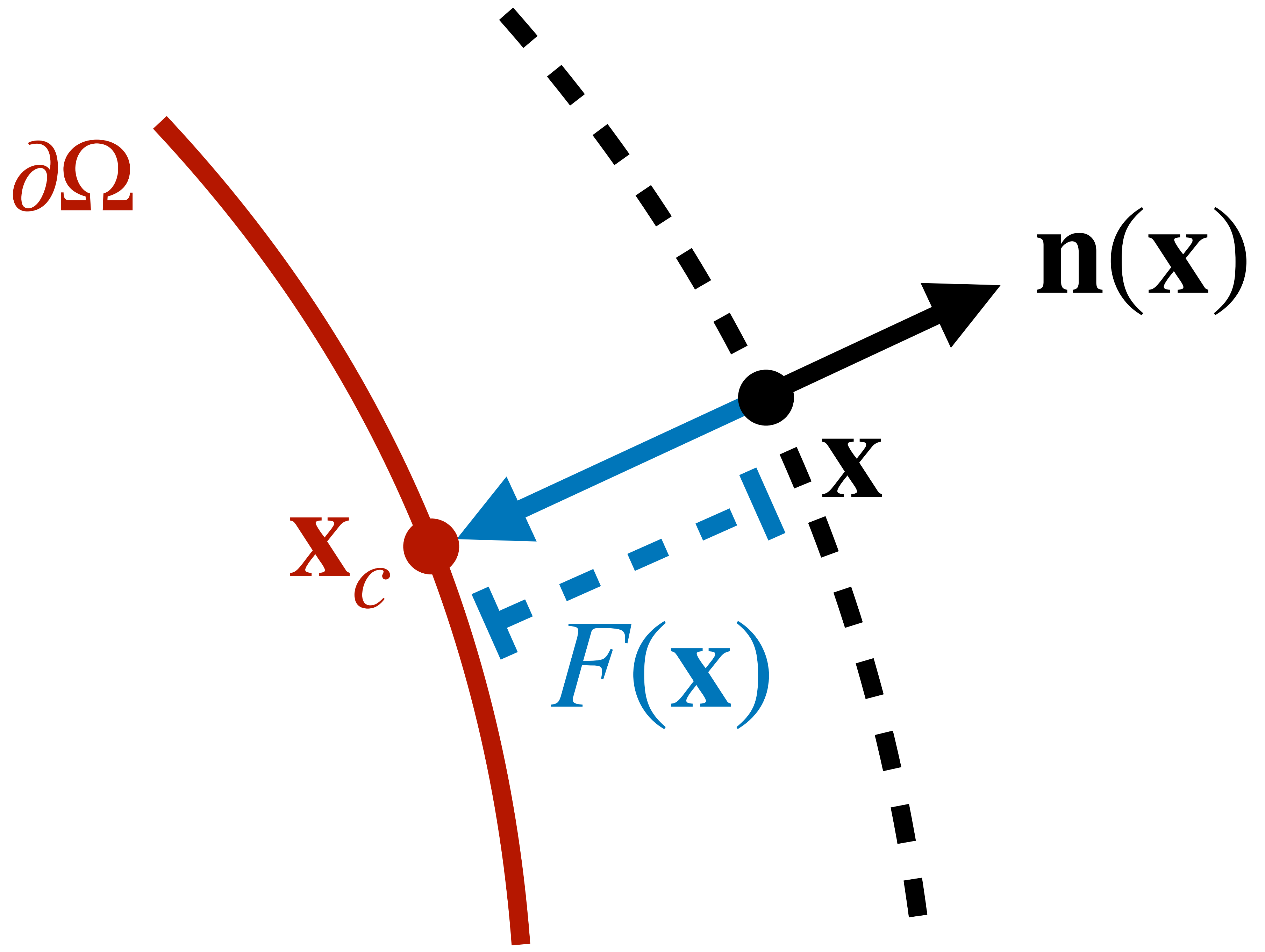
3. Comparing

Step 1: Sampling

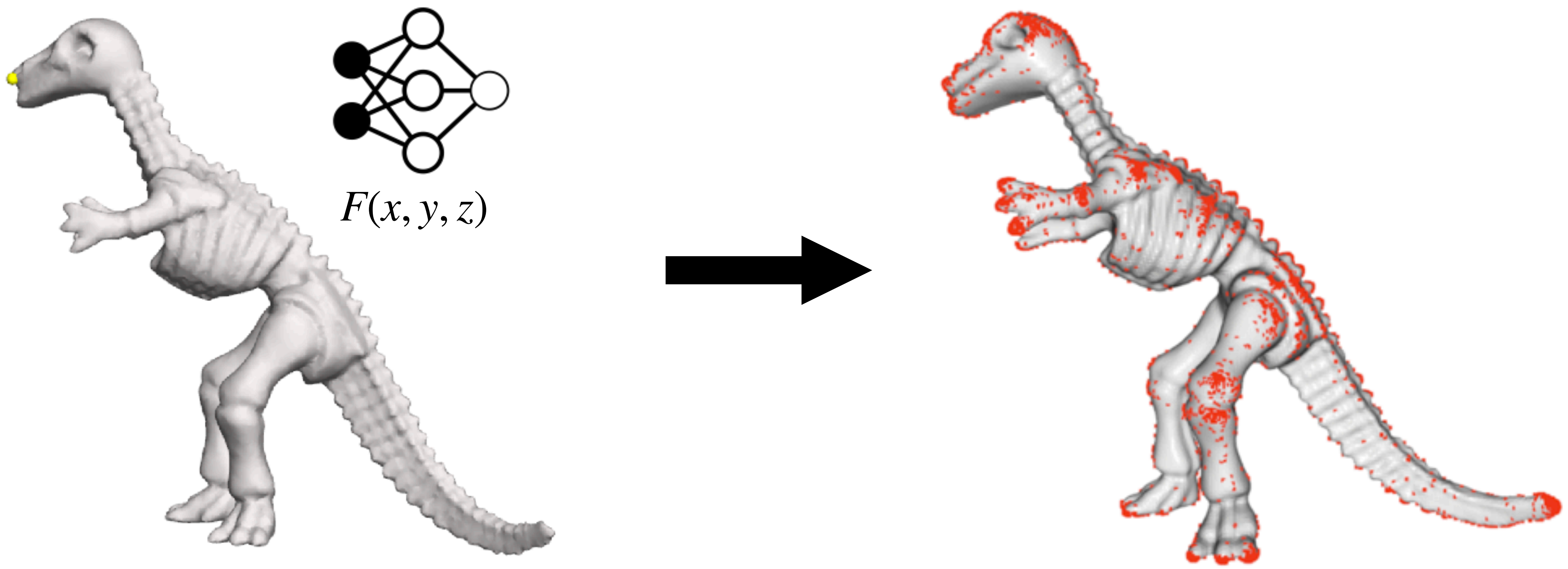
$$F(\mathbf{x}) \approx SDF(\mathbf{x})$$

$$\mathbf{x}_c = arg \min_{\mathbf{p} \in \partial\Omega} |\mathbf{p} - \mathbf{x}|$$

$$= \mathbf{x} - F(\mathbf{x})\mathbf{n}(\mathbf{x})$$



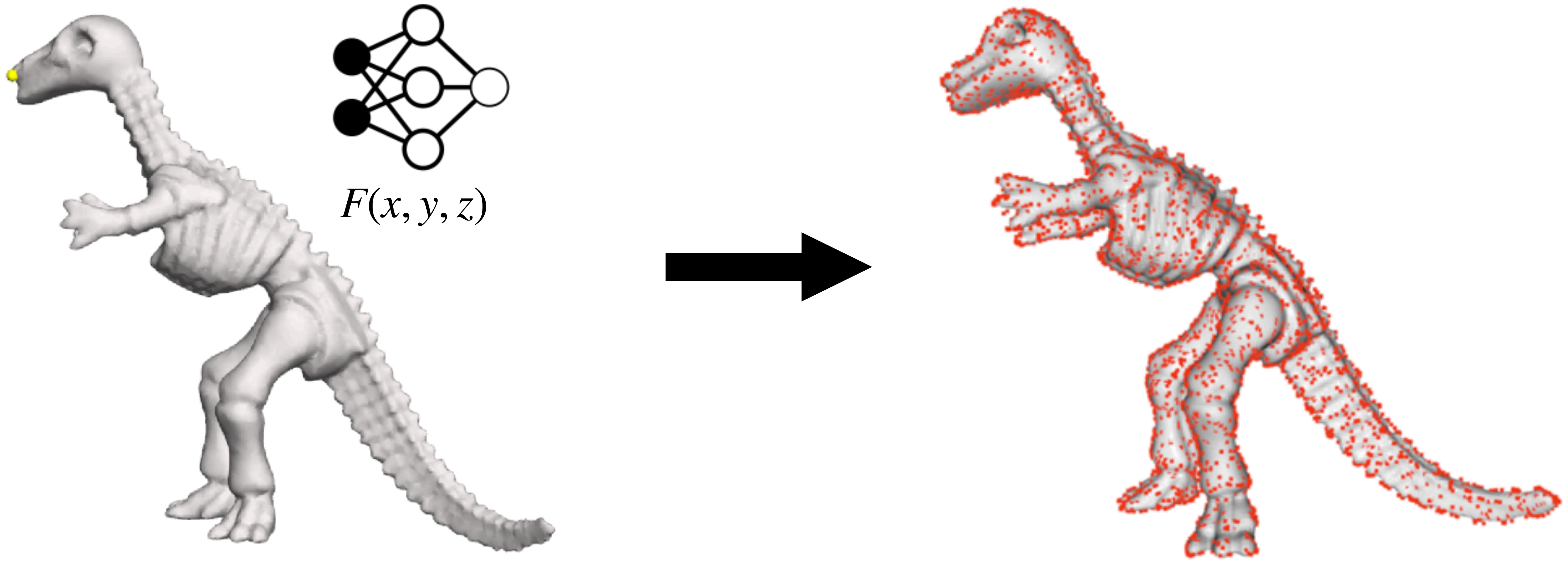
Step 1: Sampling



$$\mathbf{x}_0 \sim U([-1,1]^3)$$

$$\hat{\mathbf{x}} = \mathbf{x}_0 - F(\mathbf{x}_0)\mathbf{n}(\mathbf{x}_0)$$

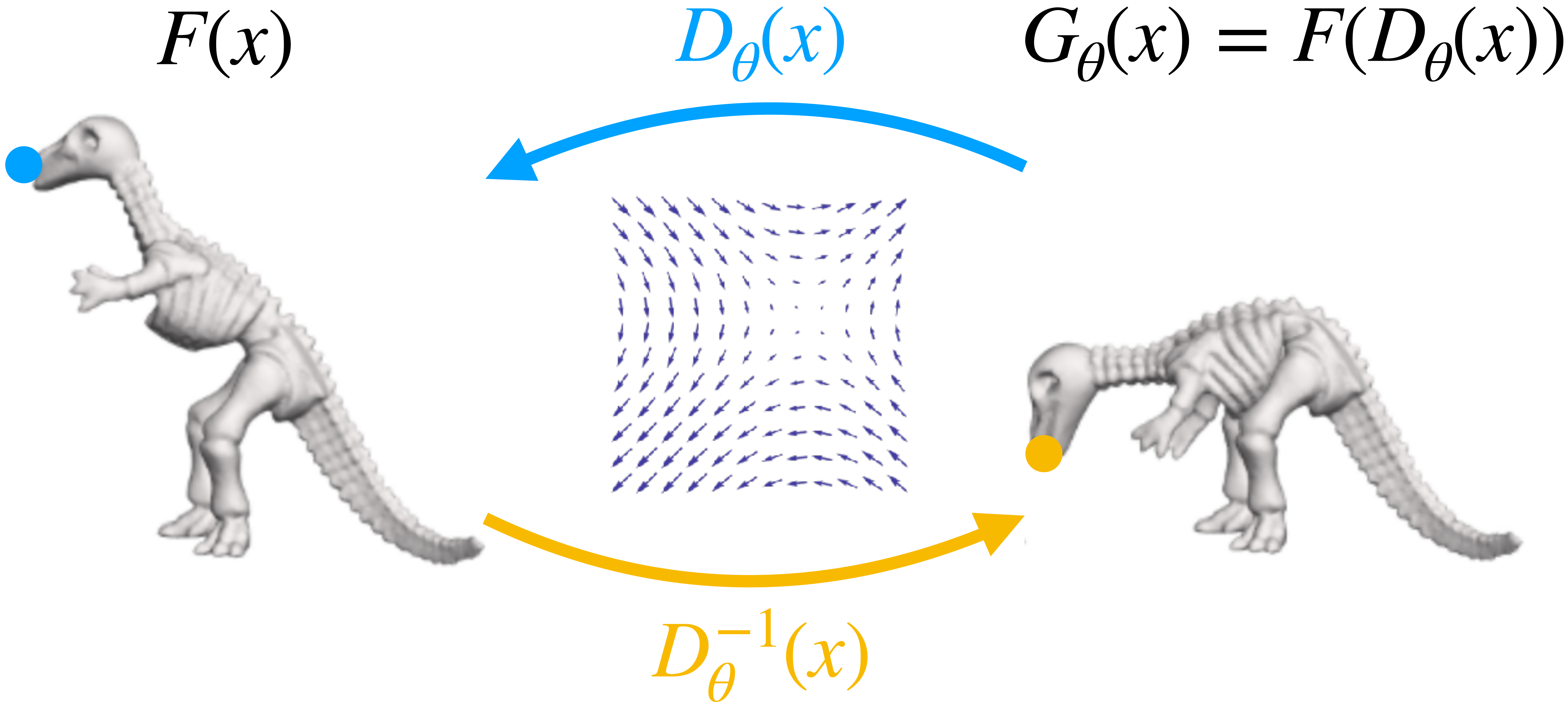
Step 1: Sampling



$$\mathbf{x}_0 \sim \{\mathbf{x} \,|\, \mathbf{x} \in U([-1,1]^3),\ F(\mathbf{x}) < \tau\}$$

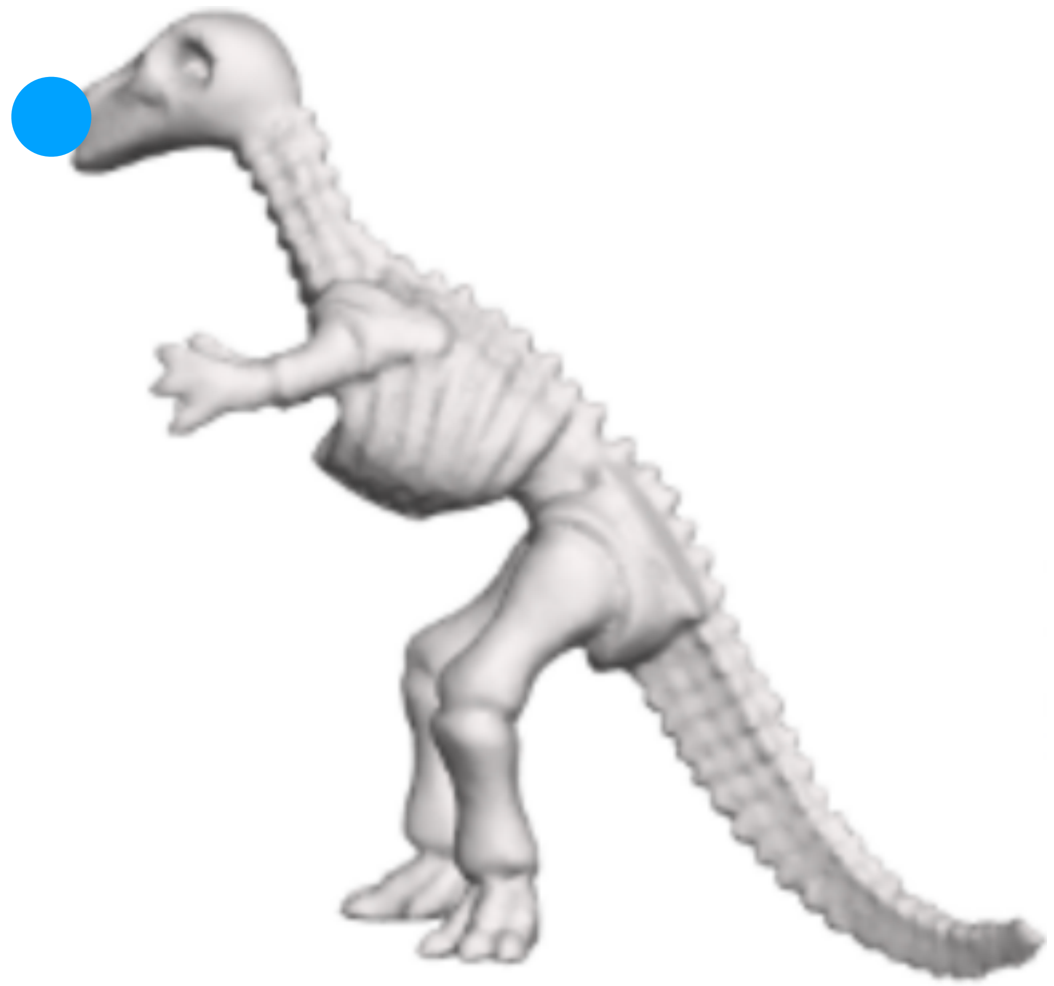
$$\hat{\mathbf{x}} = \mathbf{x}_0 - F(\mathbf{x}_0)\mathbf{n}(\mathbf{x}_0)$$

Step 2: Correspondences

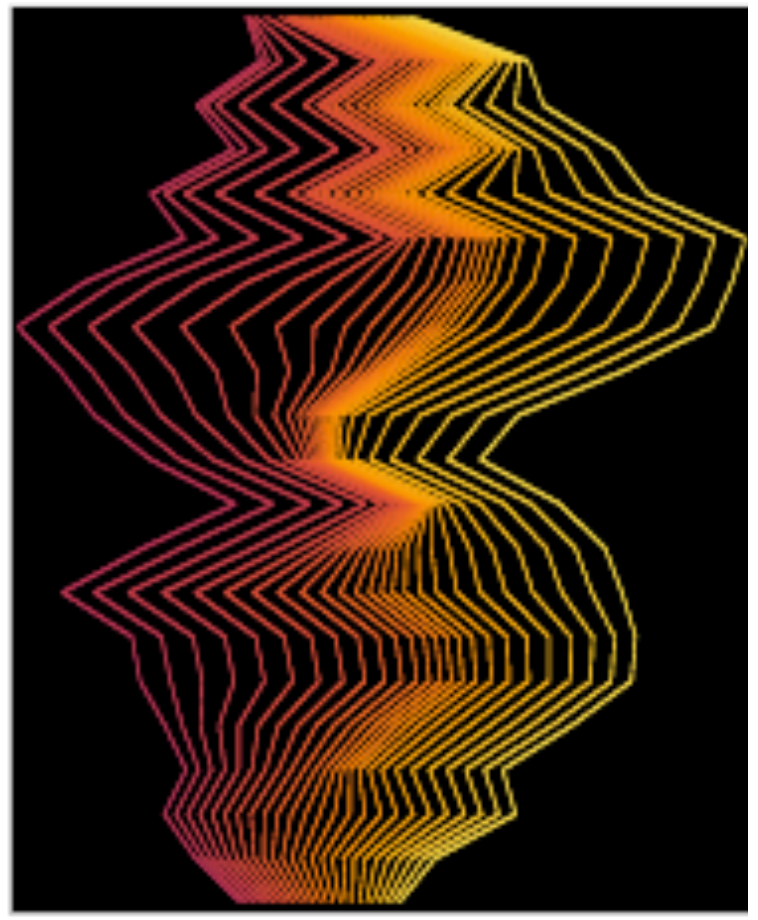


Step 2: Correspondences

$F(x)$

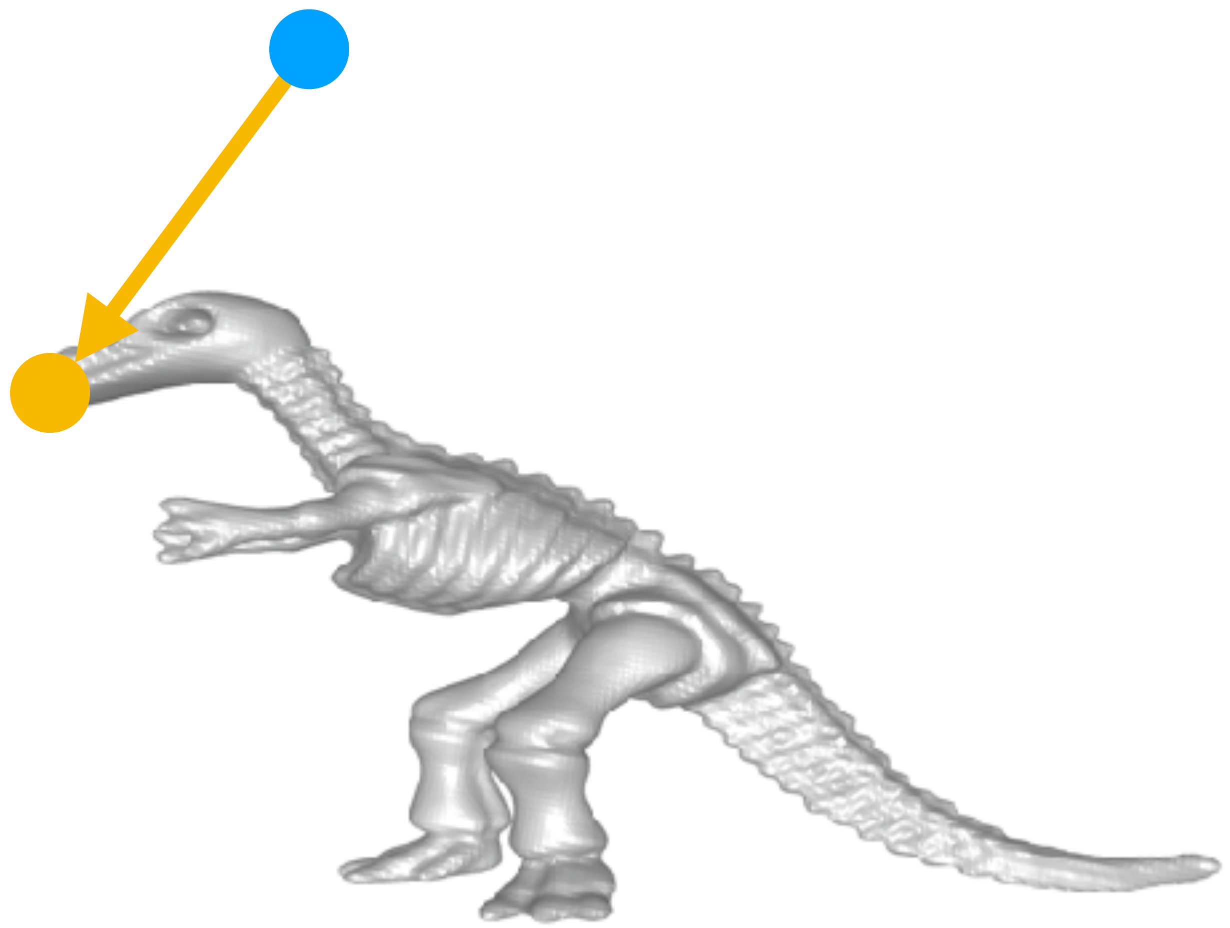


$D_\theta(x) = x + g_\theta(x)$



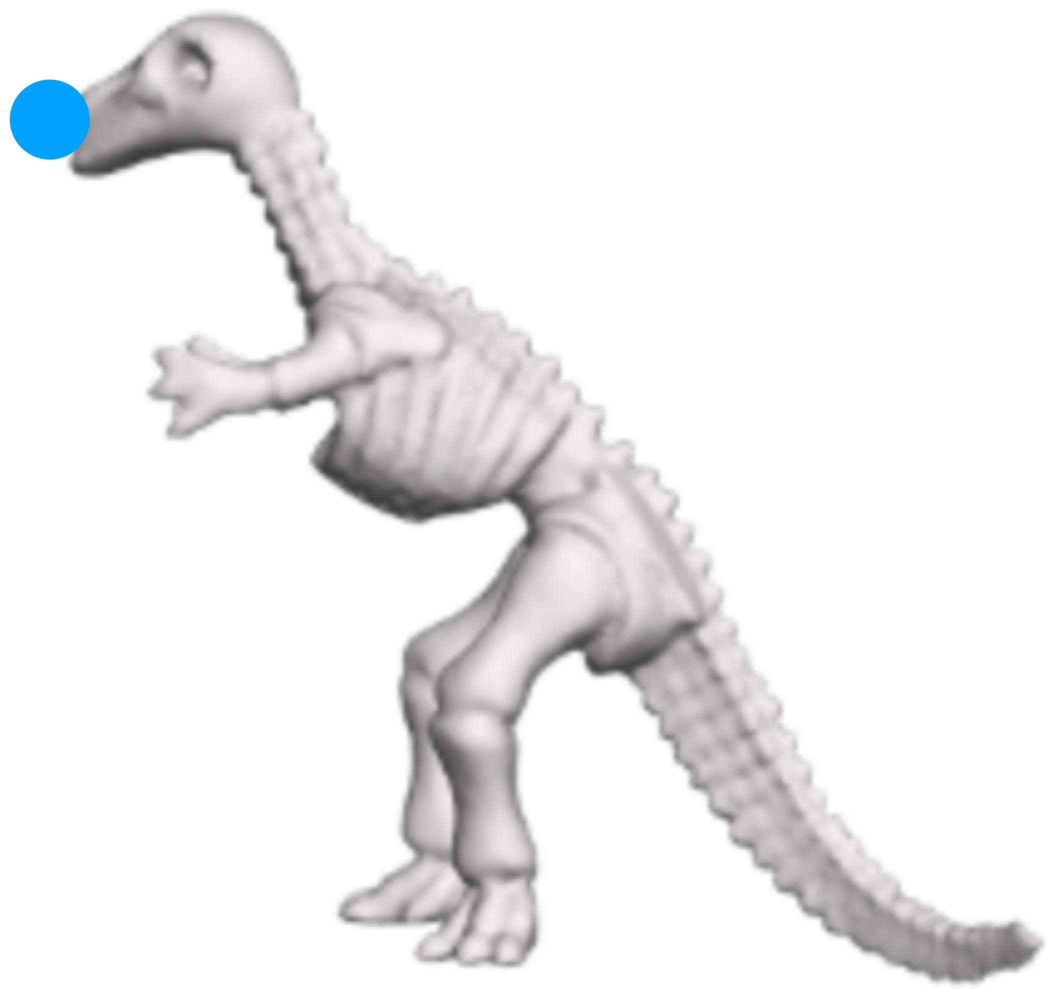
Invertible ResNet
(Behrmann *et. al.*, 2019)

$G_\theta(x) = F(D_\theta(x))$

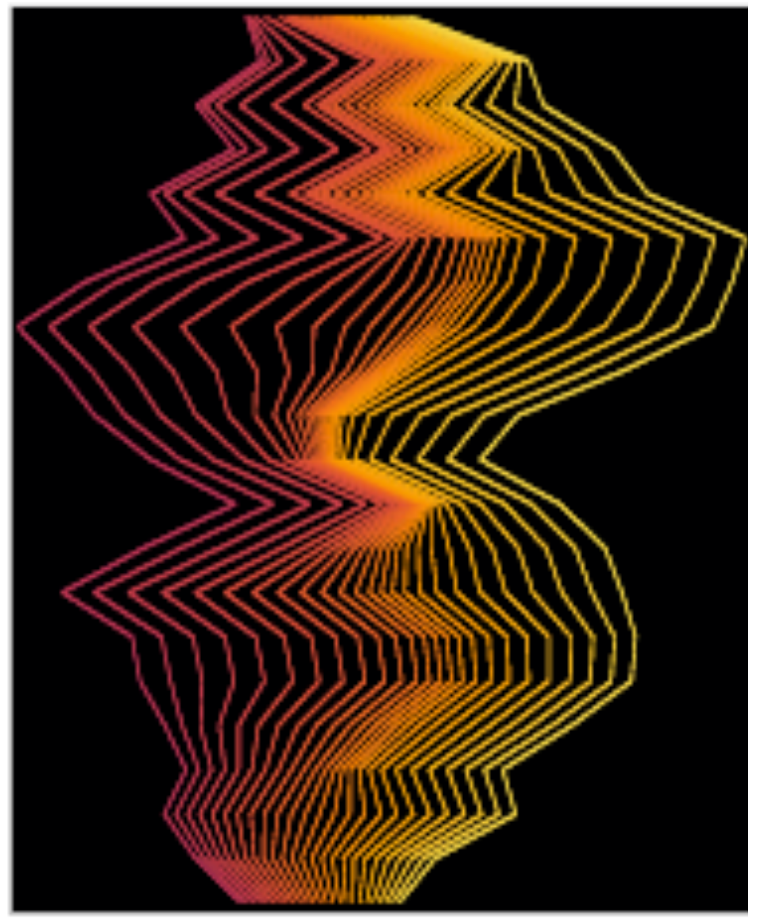


Step 2: Correspondences

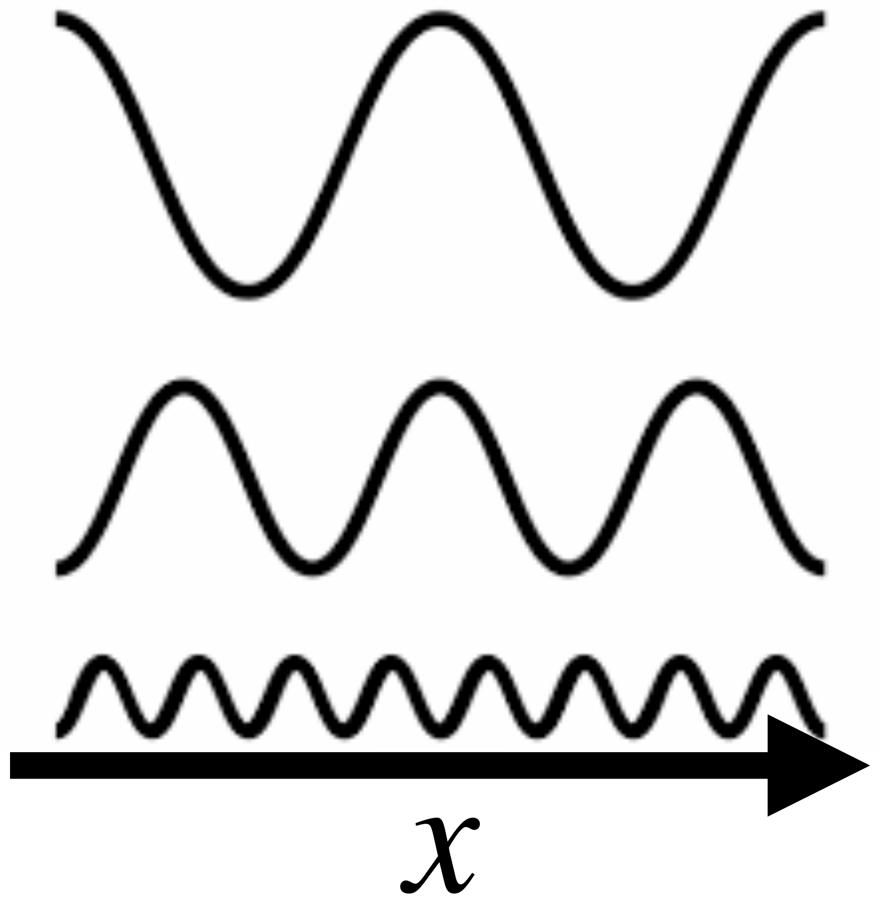
$F(x)$



$D_\theta(x) = x + g_\theta(x)$



Invertible ResNet
(Behrman et. al., 2019)



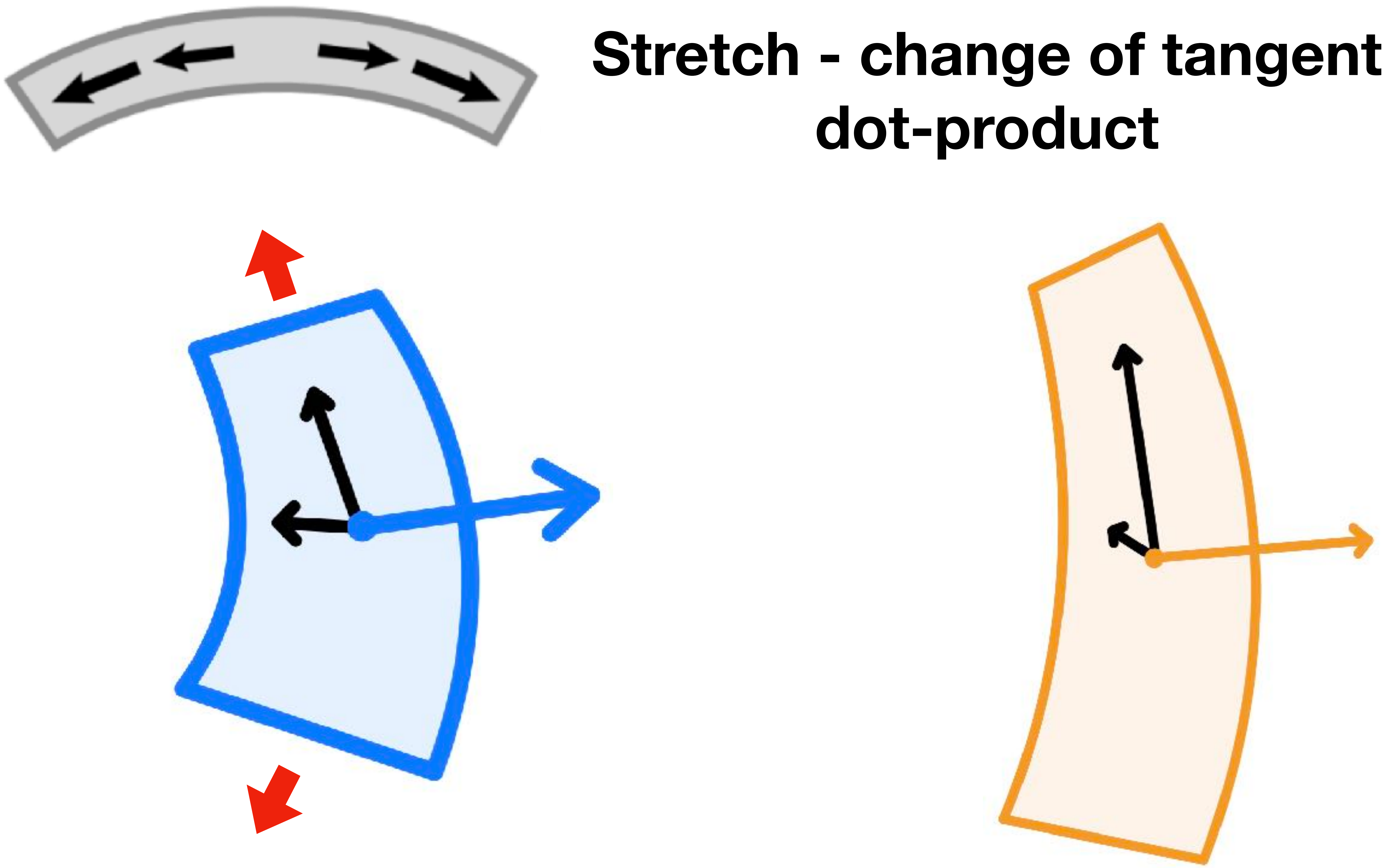
Lip-bounded
Positional Encoding

$G_\theta(x) = F(D_\theta(x))$

Step 3: Comparing

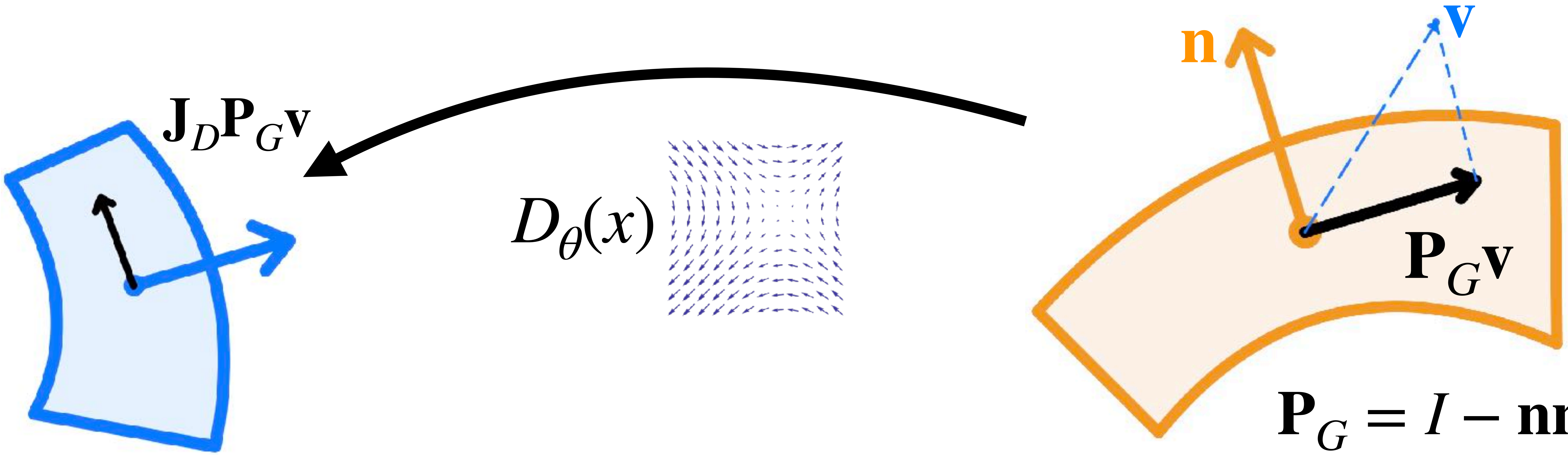
Step 3: Comparing

Stretch - change of tangent dot-product



Step 3: Comparing

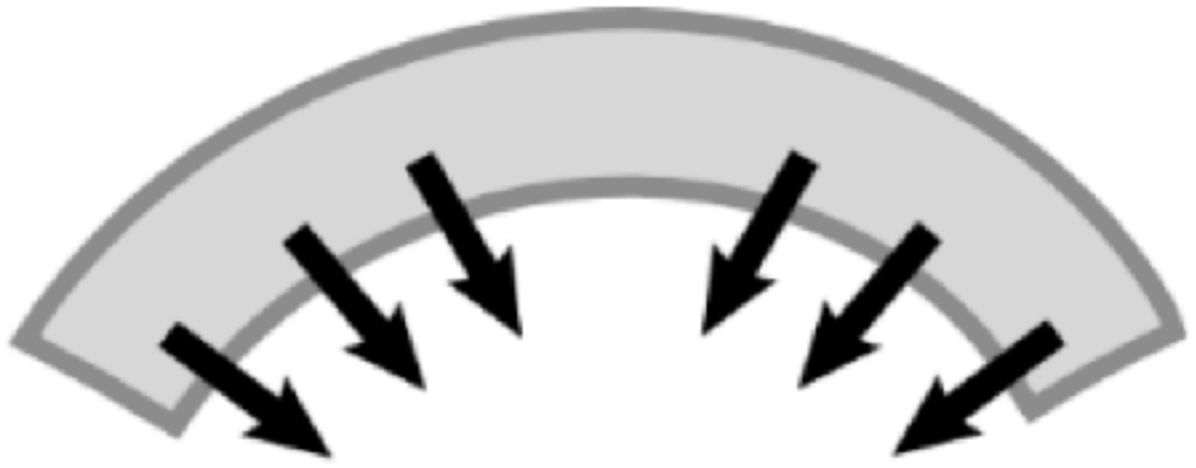
Stretch - change of tangent dot-product



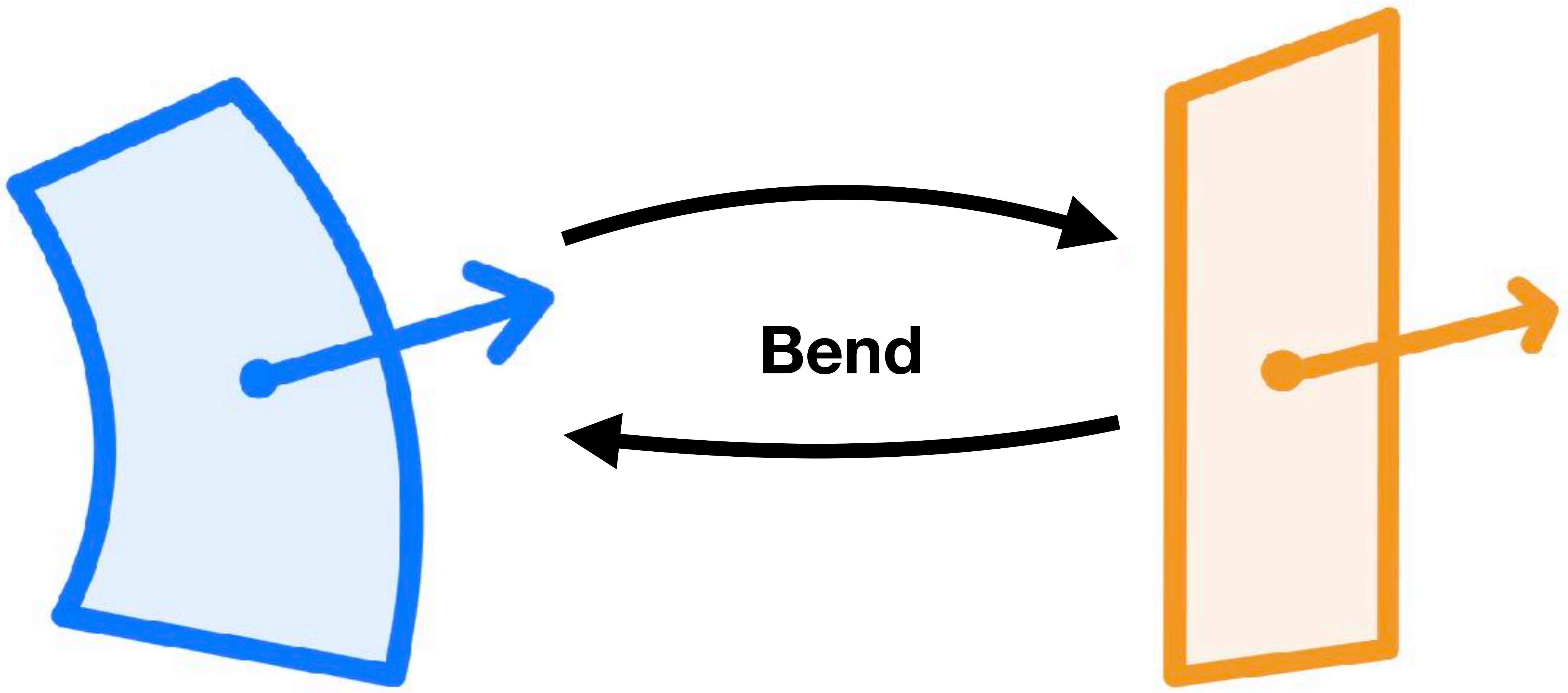
$$\mathbf{P}_G = I - \mathbf{n}\mathbf{n}^T$$

$$\mathcal{L}_s = \int \left\| \mathbf{P}_G^T \left(\mathbf{I} - \mathbf{J}_D^T \mathbf{J}_D \right) \mathbf{P}_G \right\|_F dx$$

Step 3: Comparing



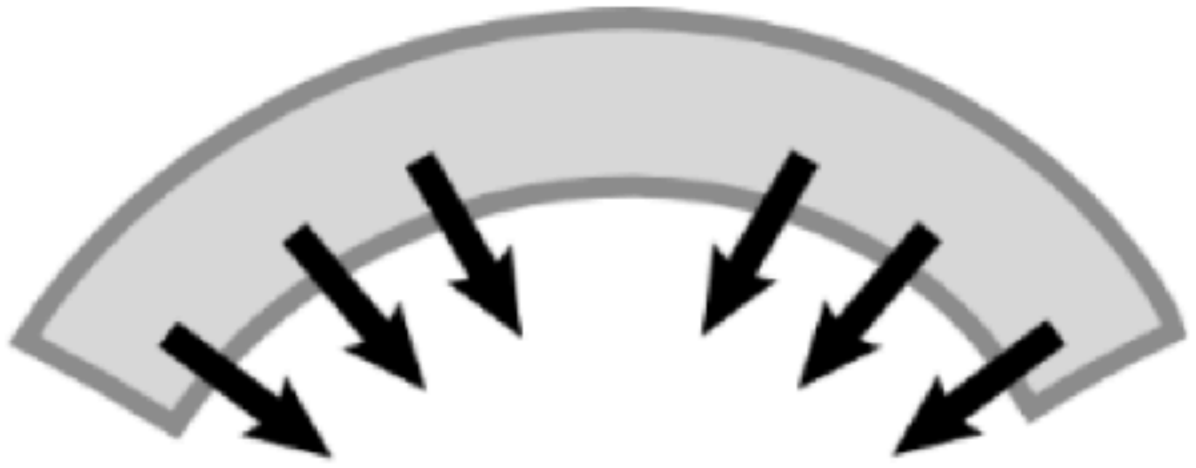
Bending - change of curvature



More curved

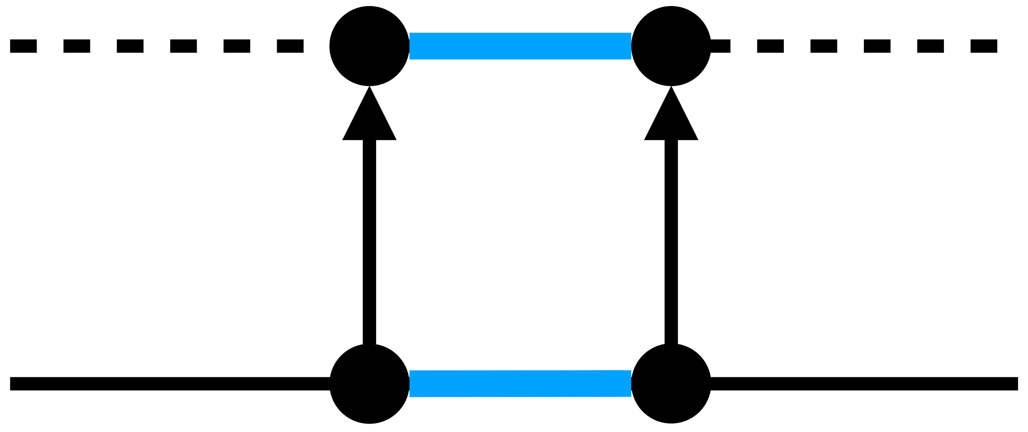
Less curved

Step 3: Comparing

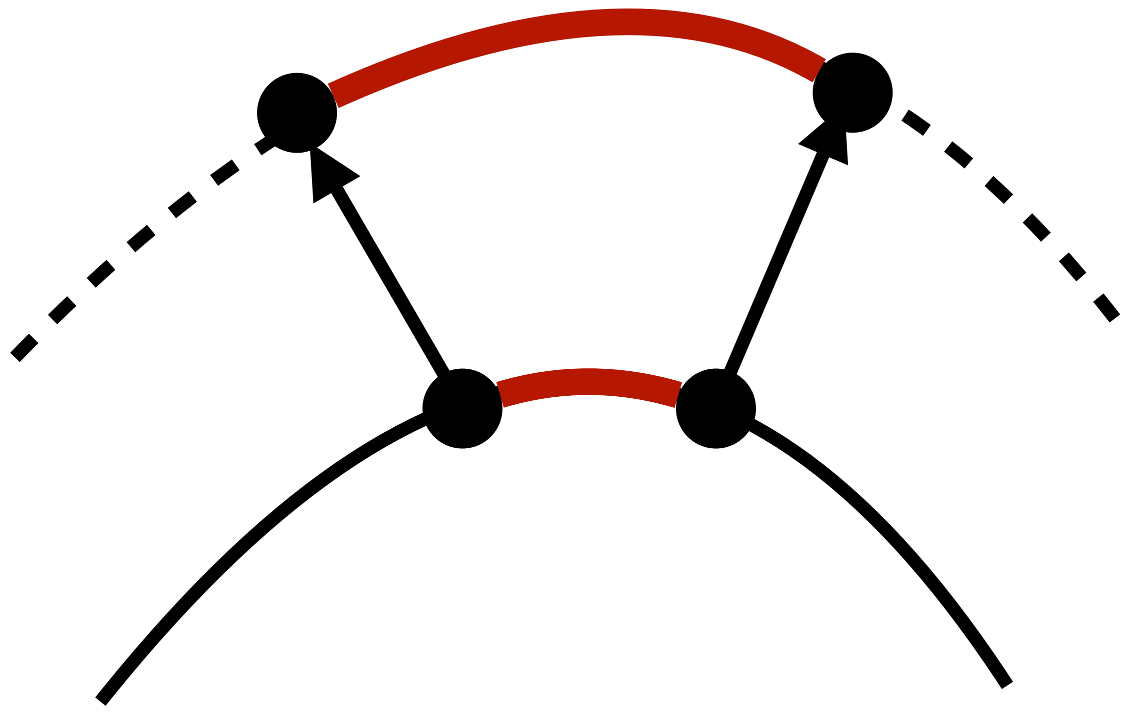


Bending - change of curvature

Curvature - change along normal direction



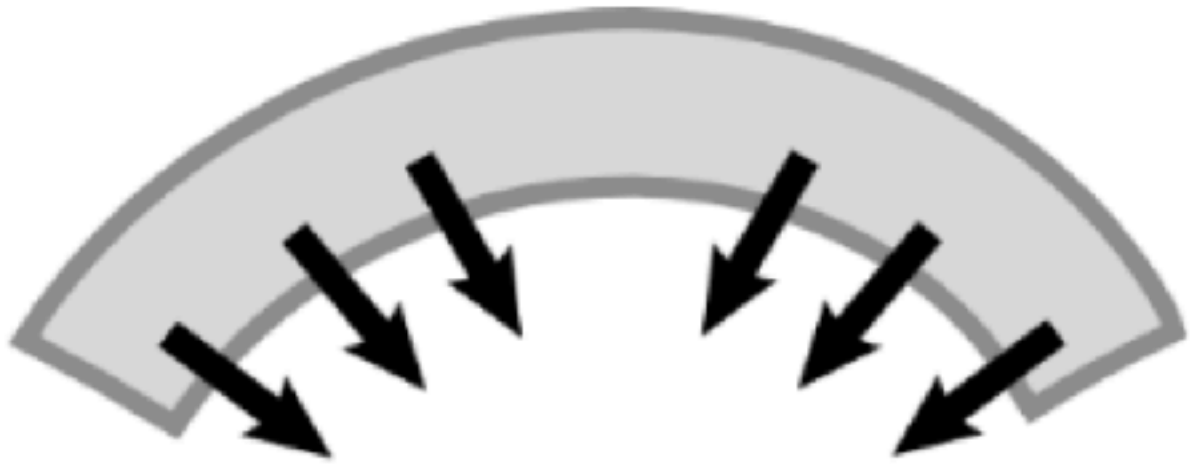
Low curvature
Little change



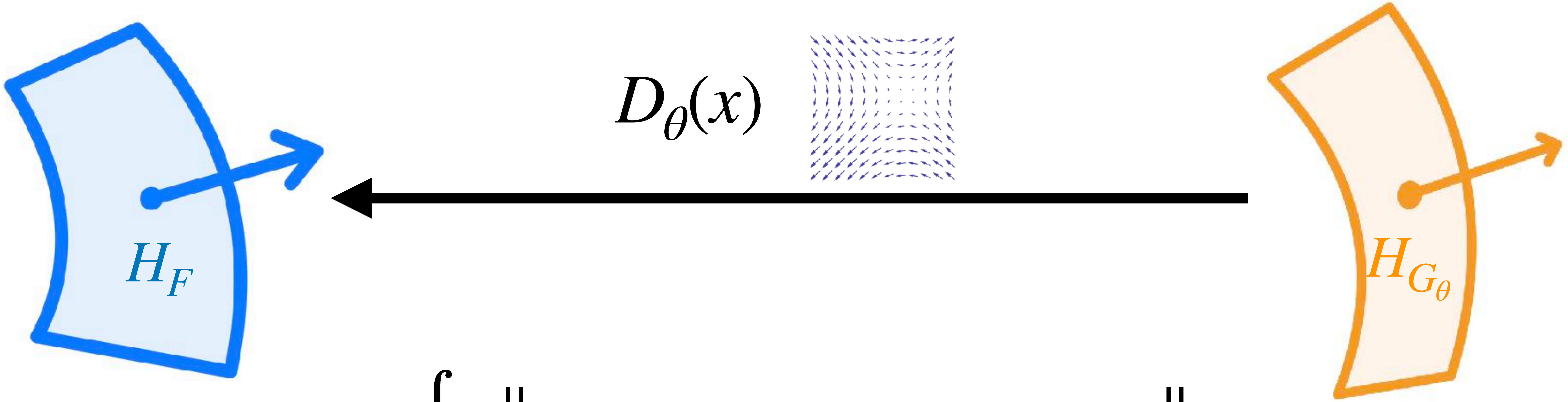
High curvature
Large change

$$H_f = \begin{bmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{yx} & f_{yy} & f_{yz} \\ f_{zx} & f_{zy} & f_{zz} \end{bmatrix}$$

Step 3: Comparing



Bending - change of curvature



$$\mathscr{L}_b = \int_x \left\| \mathbf{P}_G^T \left(\mathbf{H}_G - \mathbf{J}_D^T \mathbf{H}_F \mathbf{J}_D \right) \mathbf{P}_G \right\|_F dx$$

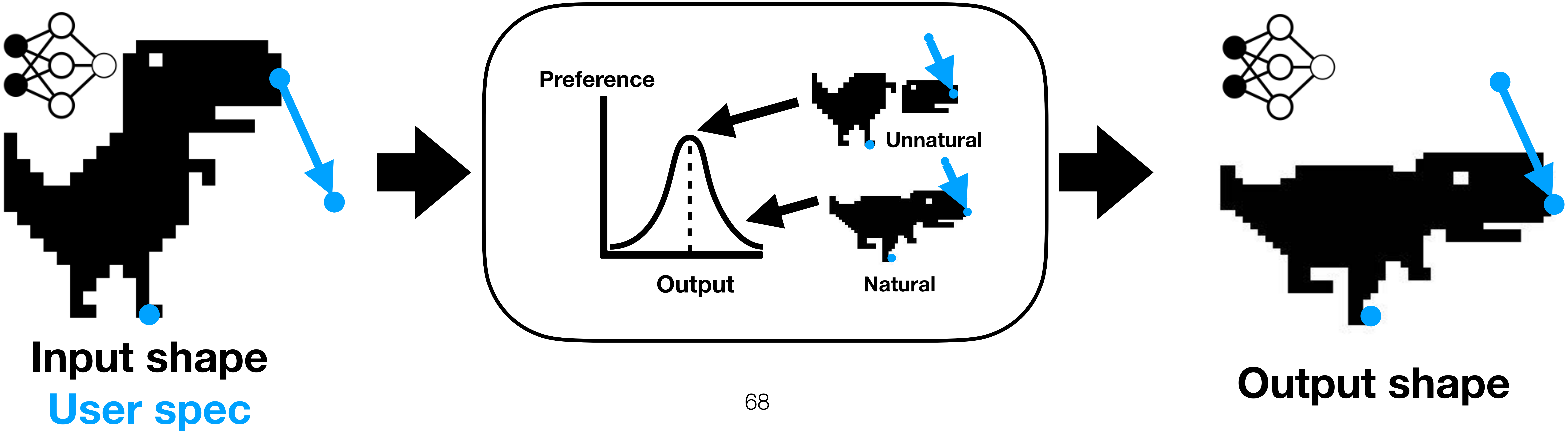
Final Objective

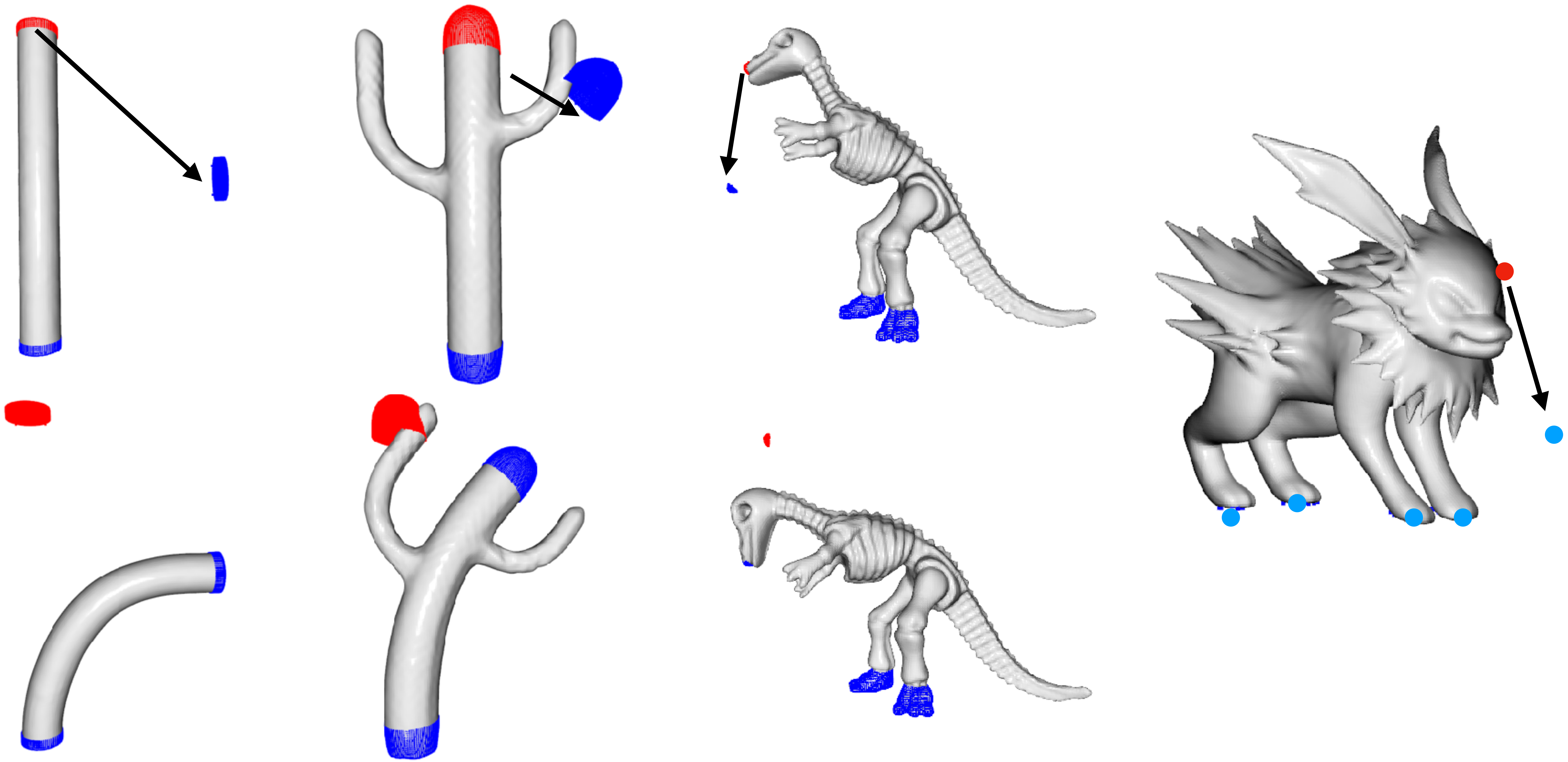
$$\mathcal{L}_s = \int_x \left\| \mathbf{P}_G^T \left(\mathbf{I} - \mathbf{J}_D^T \mathbf{J}_D\right) \mathbf{P}_G \right\|_F dx$$

$$\mathcal{L}_{spec} = \sum_i \max(\tau, |D_\theta(t_i) - h_i|)$$

$$\mathcal{L}_b = \int_x \left\| \mathbf{P}_G^T \left(\mathbf{H}_G - \mathbf{J}_D^T \mathbf{H}_F \mathbf{J}_D\right) \mathbf{P}_G \right\|_F dx$$

$$\min_{\theta} \; k_s\mathcal{L}_s + k_b\mathcal{L}_b + k_c\mathcal{L}_{spec}$$





Analysis - Solving PDEs

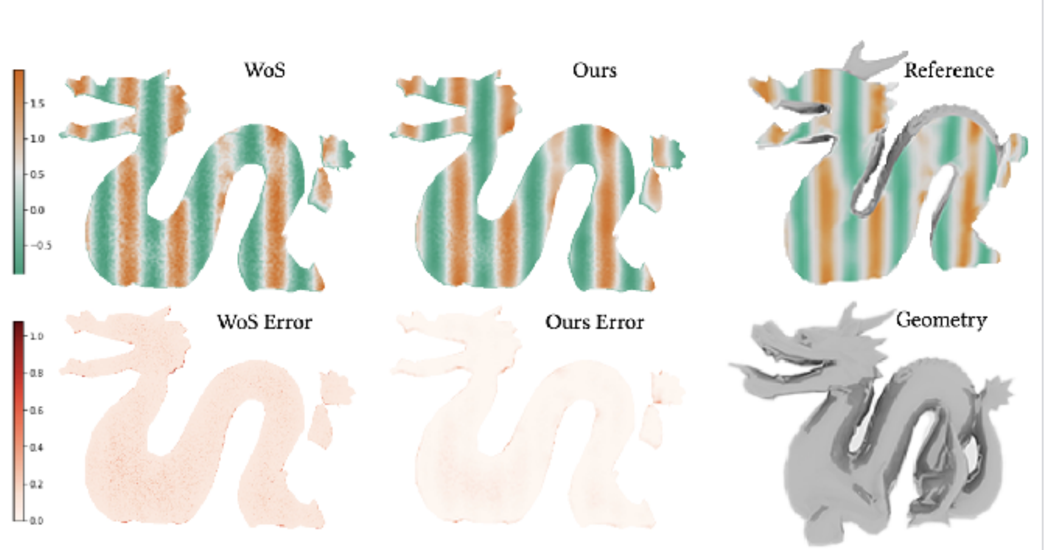
Synthesis

PointFlow (ICCV 2019)

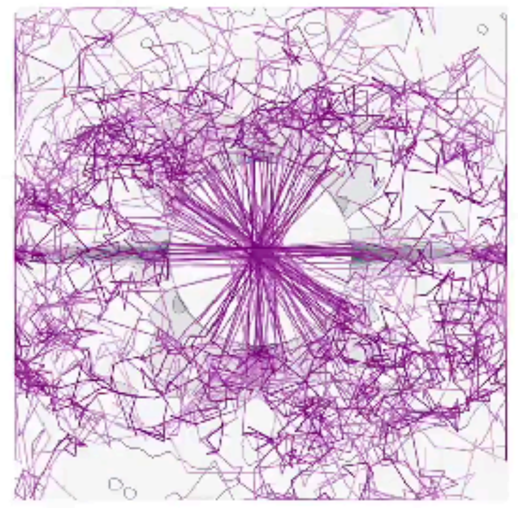
ShapeGF (ECCV 2020)

NFGP (NeurIPS 2021)

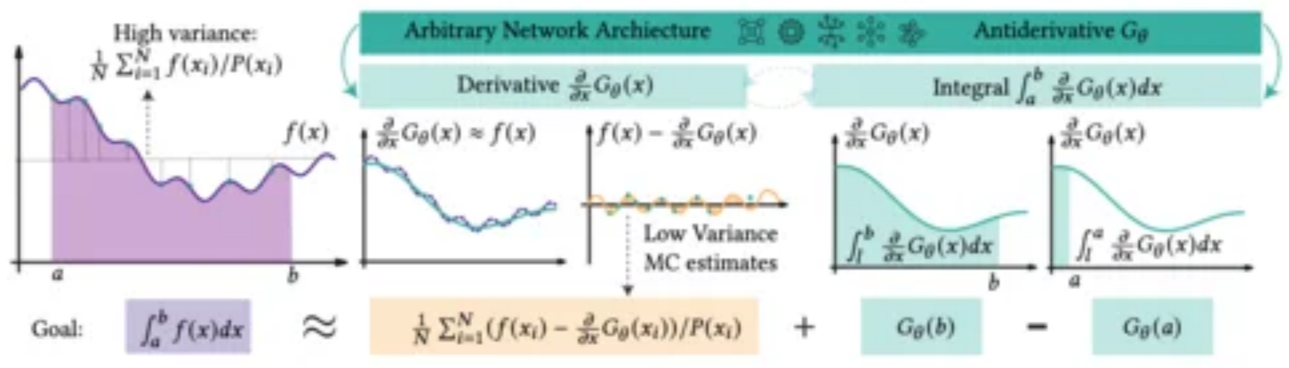
Analysis



Neural cache (SIG Asia 2023)



Symmetry (SIG Asia 2024)



NCV (SIGRAPH 2024)

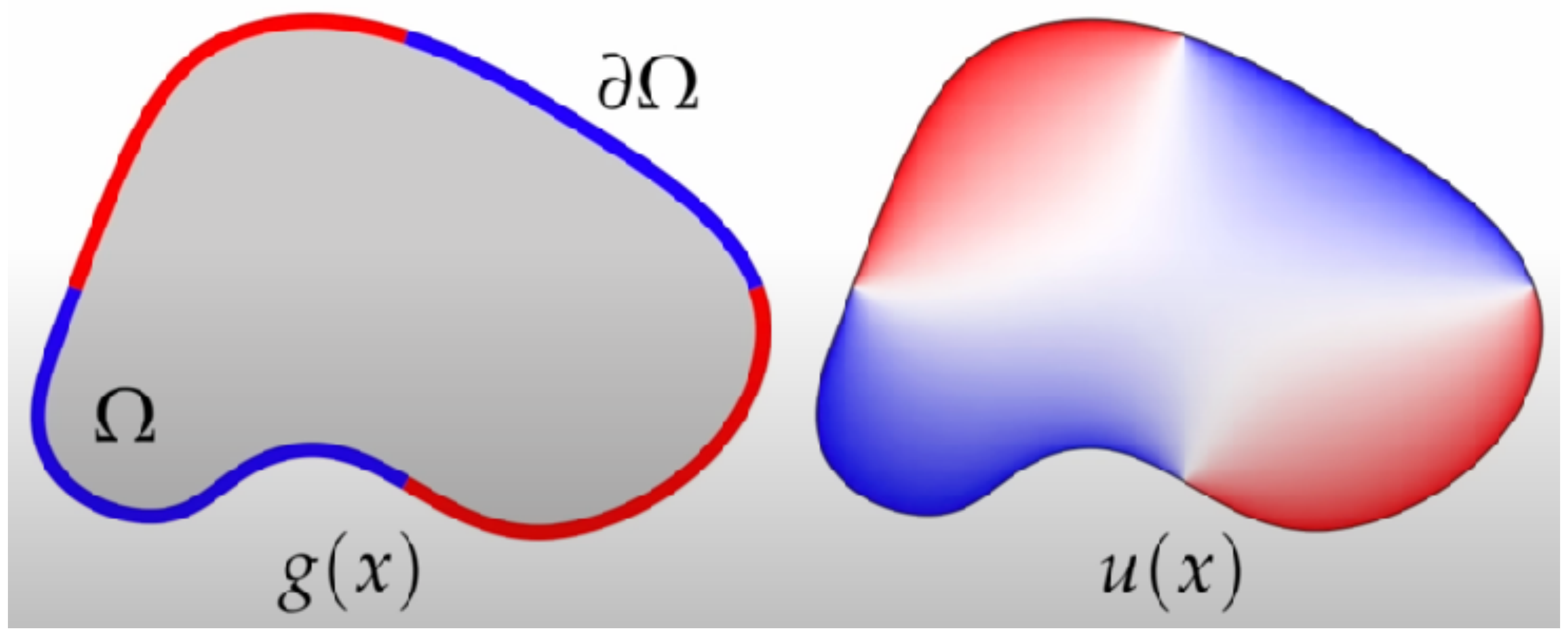
2019 2020 2021 2023 2024

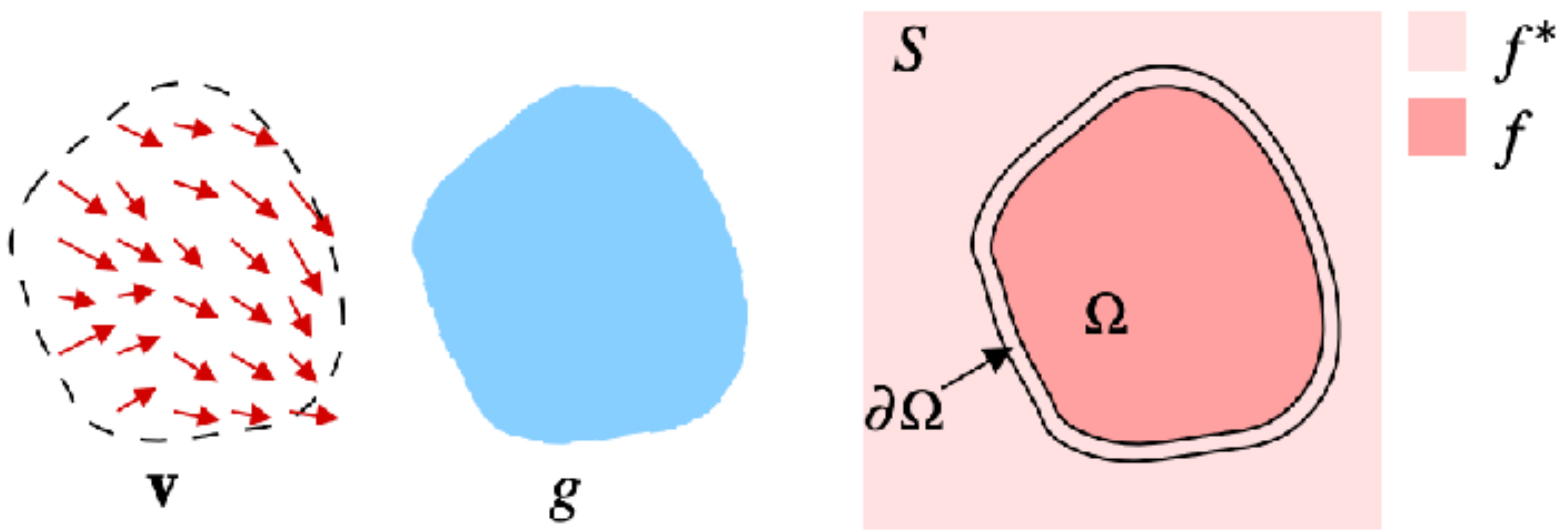
Partial Differential Equations are Important

Equations that involve partial derivatives.

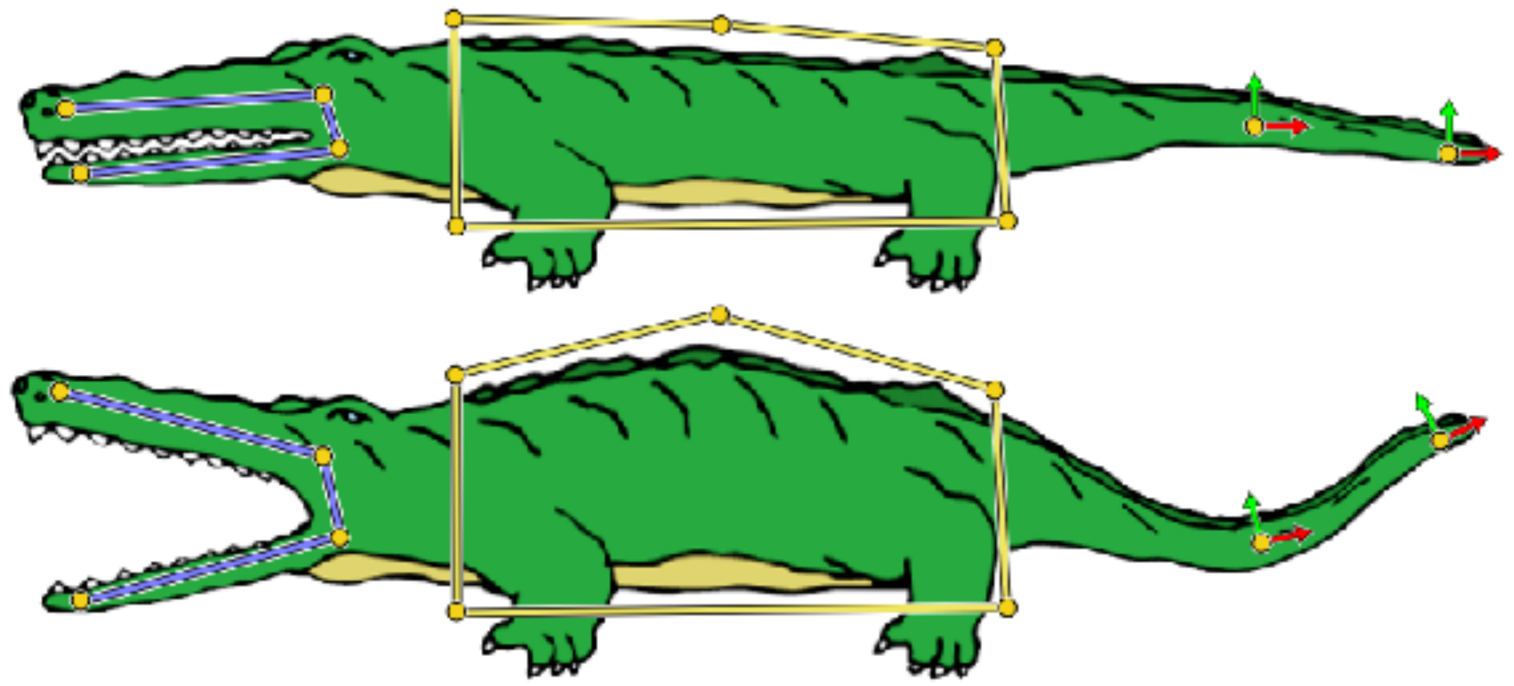
e.g. Laplace Equation

$$\begin{aligned} \Delta u &= 0 \quad \text{on } \Omega \\ u &= g \quad \text{on } \partial\Omega \end{aligned}$$

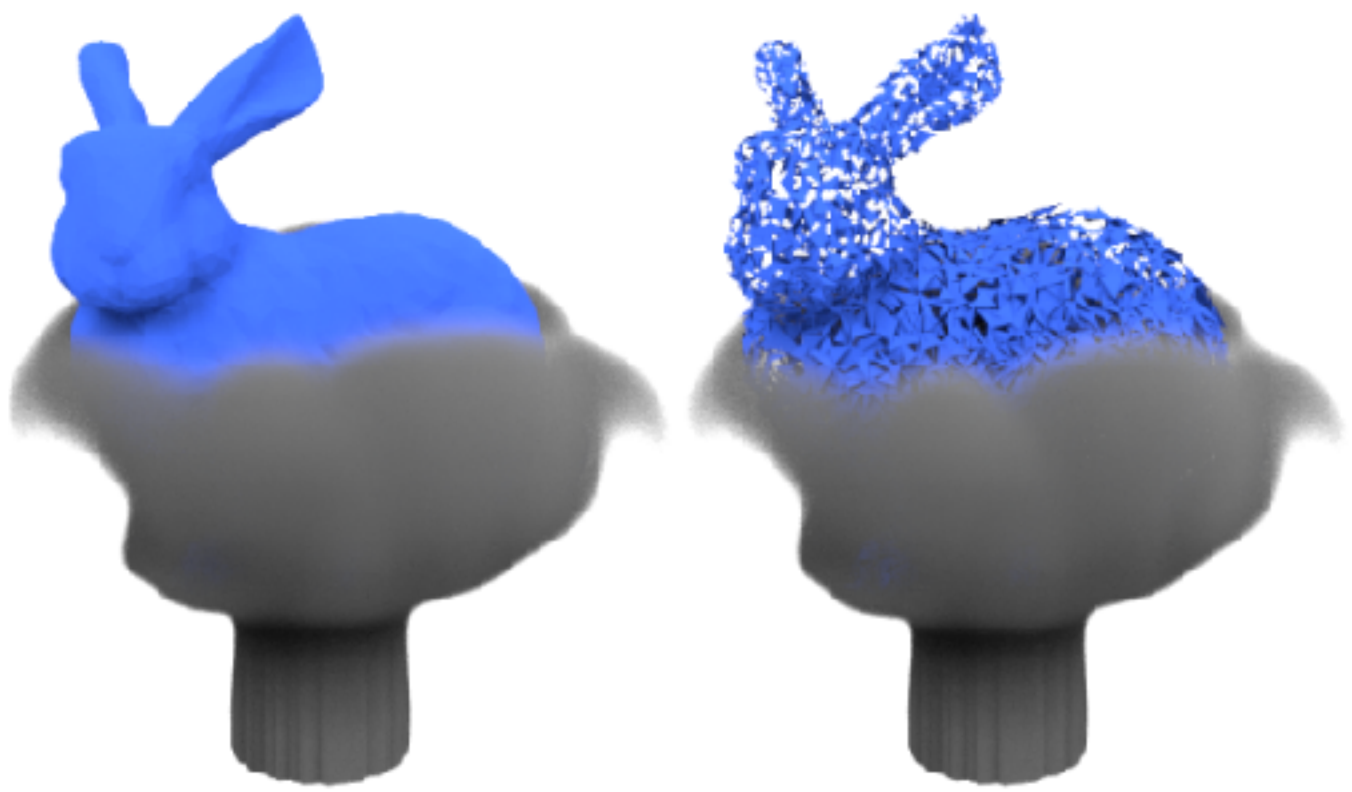




(Poisson eq) Image Editing
(Perez, Gangnet, and Blake, 2012)

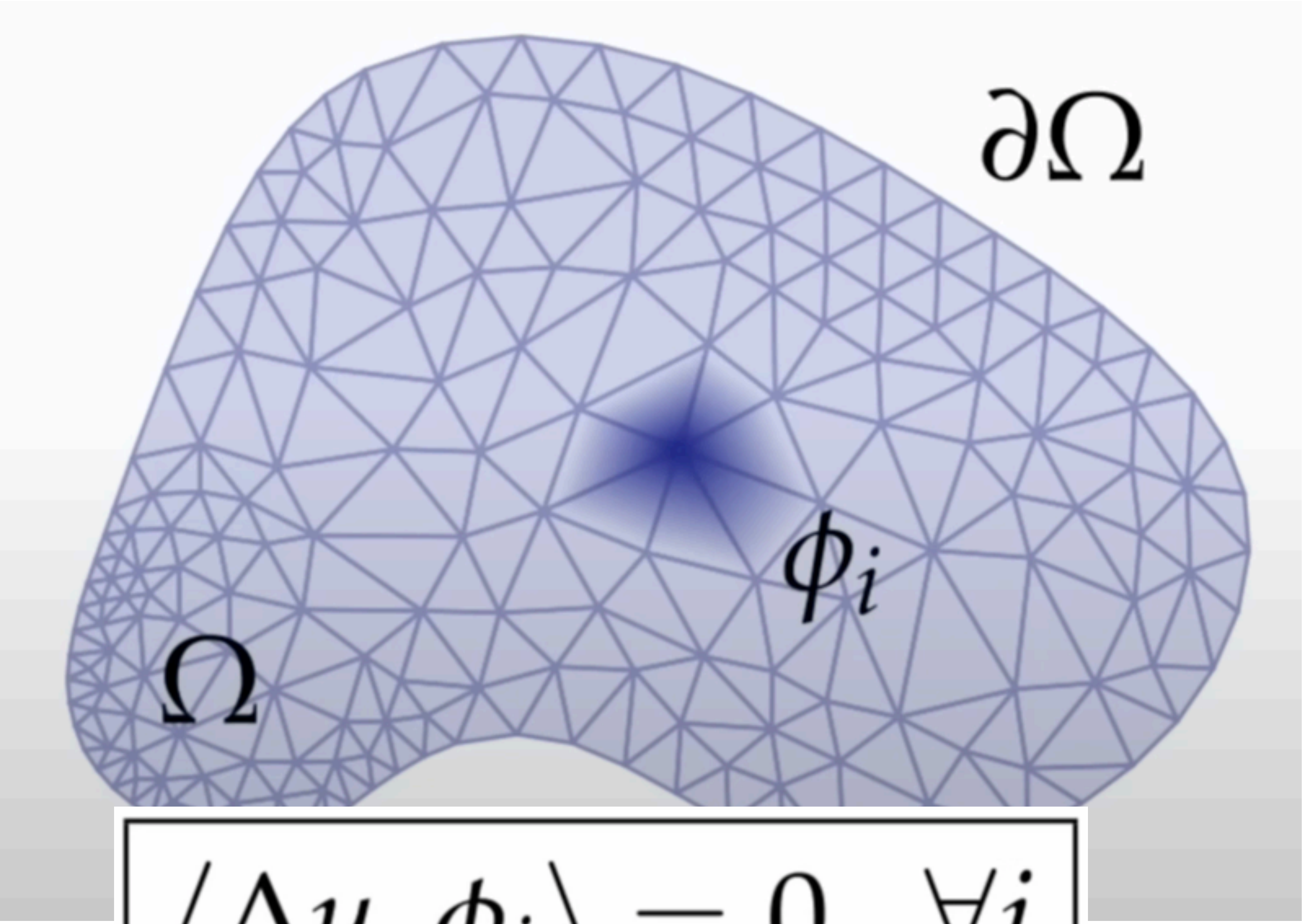


(Biharmonic equation) Deformation
(Jacobson et. al, 2011)



(Navier-Stokes) Fluid Simulation
(Rioux-Lavoie et. al, 2022)

Solving PDEs - Finite-element Method

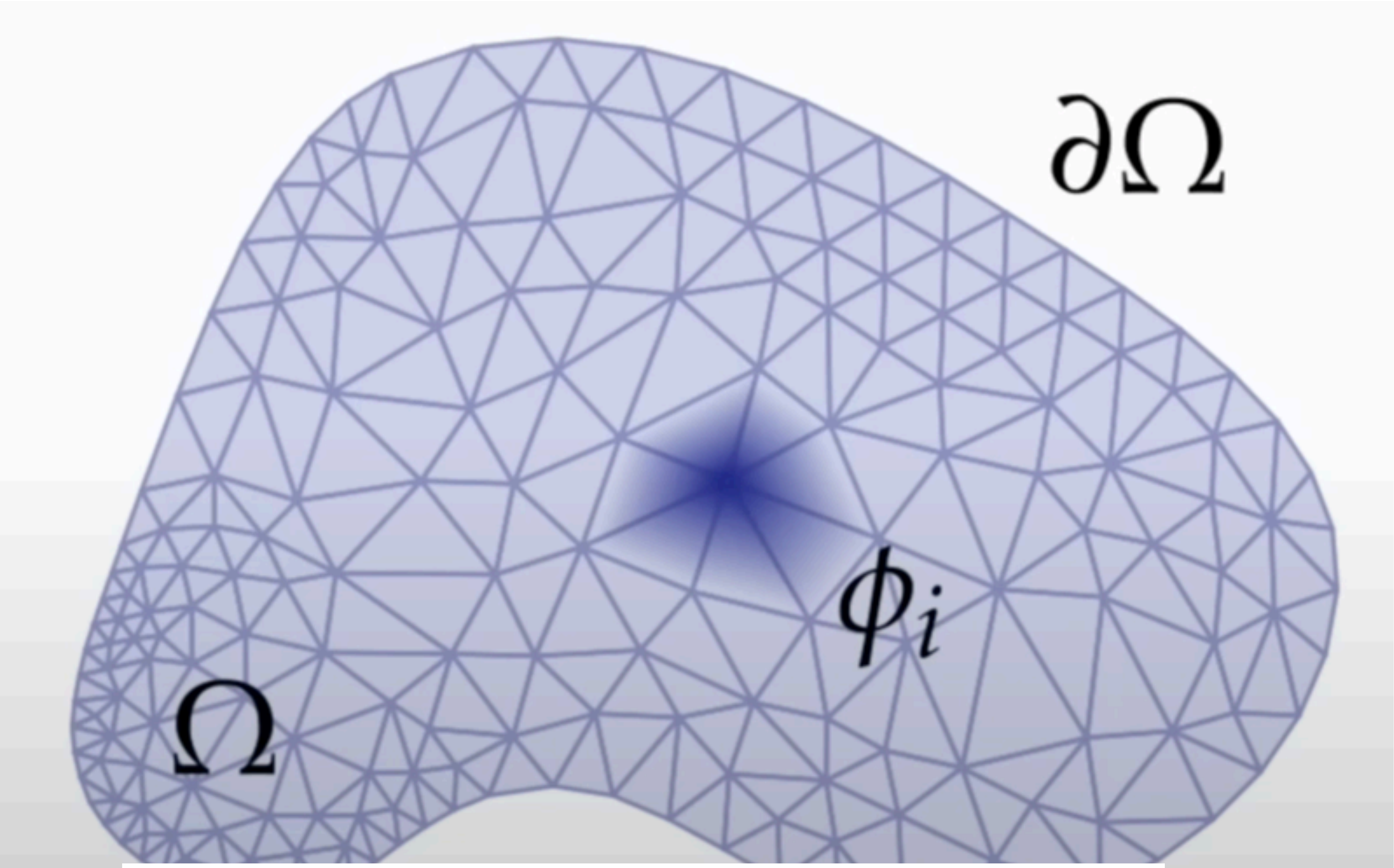


$$\boxed{\langle \Delta u, \phi_i \rangle = 0, \ \forall i}$$

Figure Credit: Keenan Crane

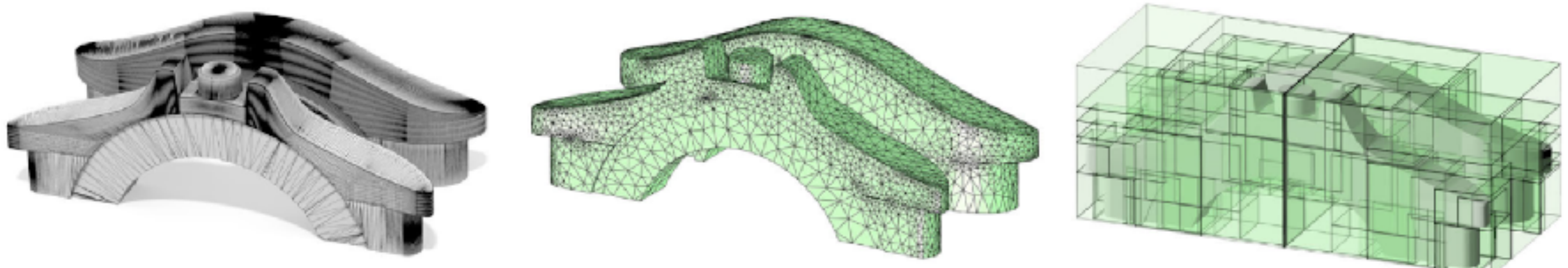
Solving PDEs - Finite-element Method

Discretization can be difficult.

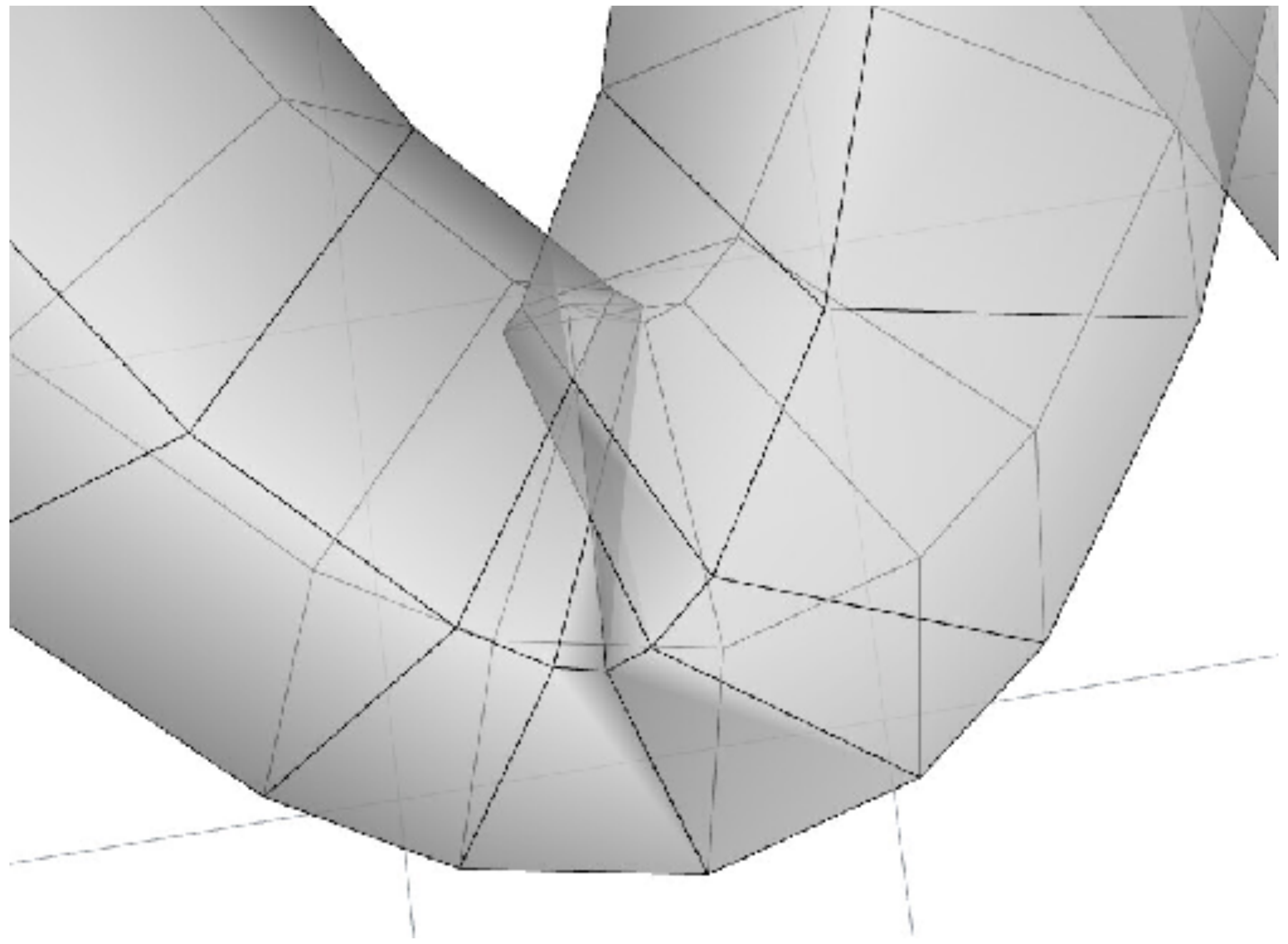


$$\langle \Delta u, \phi_i \rangle = 0, \ \forall i$$

Figure Credit: Keenan Crane



input (Thingi10k #996816) | mesh w/ FastTetWild *1 hour 25 minutes* | build BVH for WoS *< 1 second*



Bruno Levy @BrunoLevy01 · 9/15/23

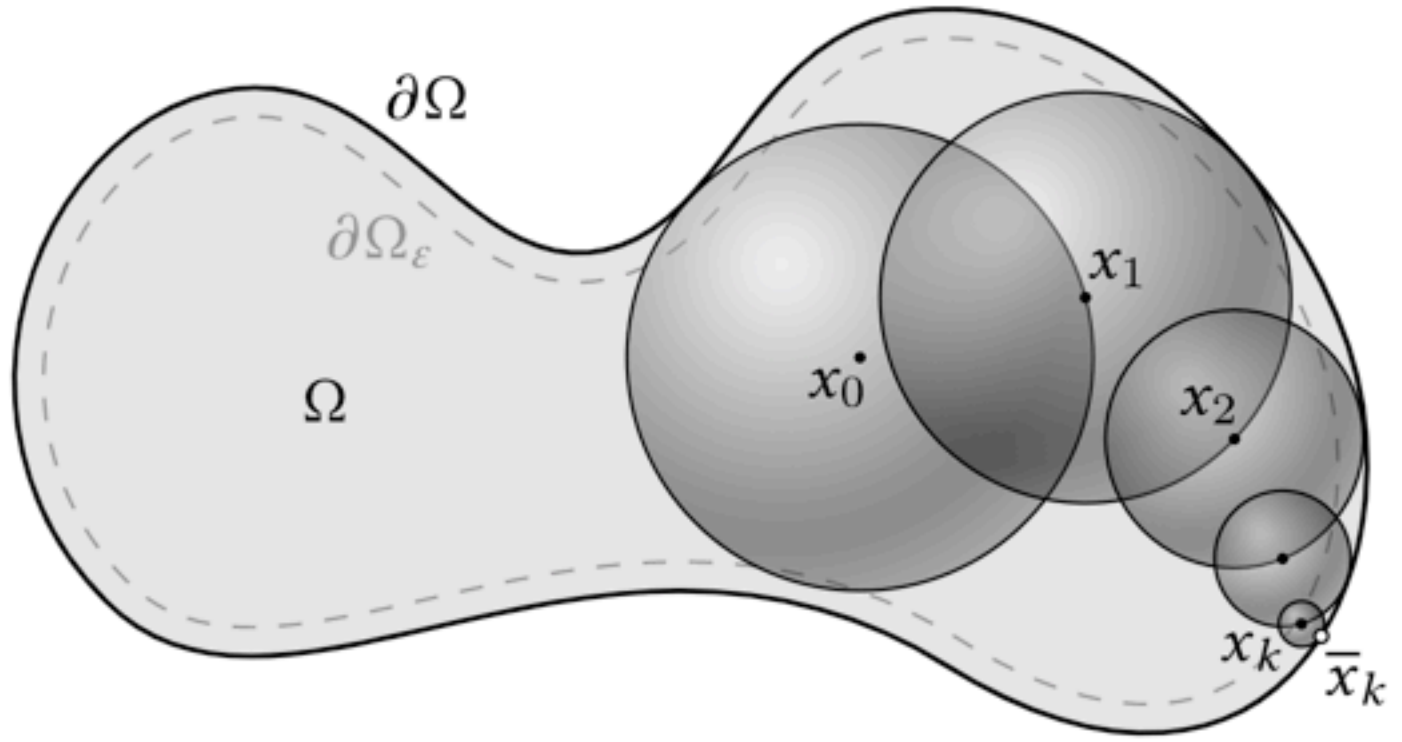
1/N

If you do mesh processing, a probably know this mesh, right ?

Else let me introduce Thingi10K #996816, that I like to call "the nemesis". This innocent-looking mesh has the power to stress your mesh intersection code much further than you may imagine.

2 · 12 · 97 · 13.1K

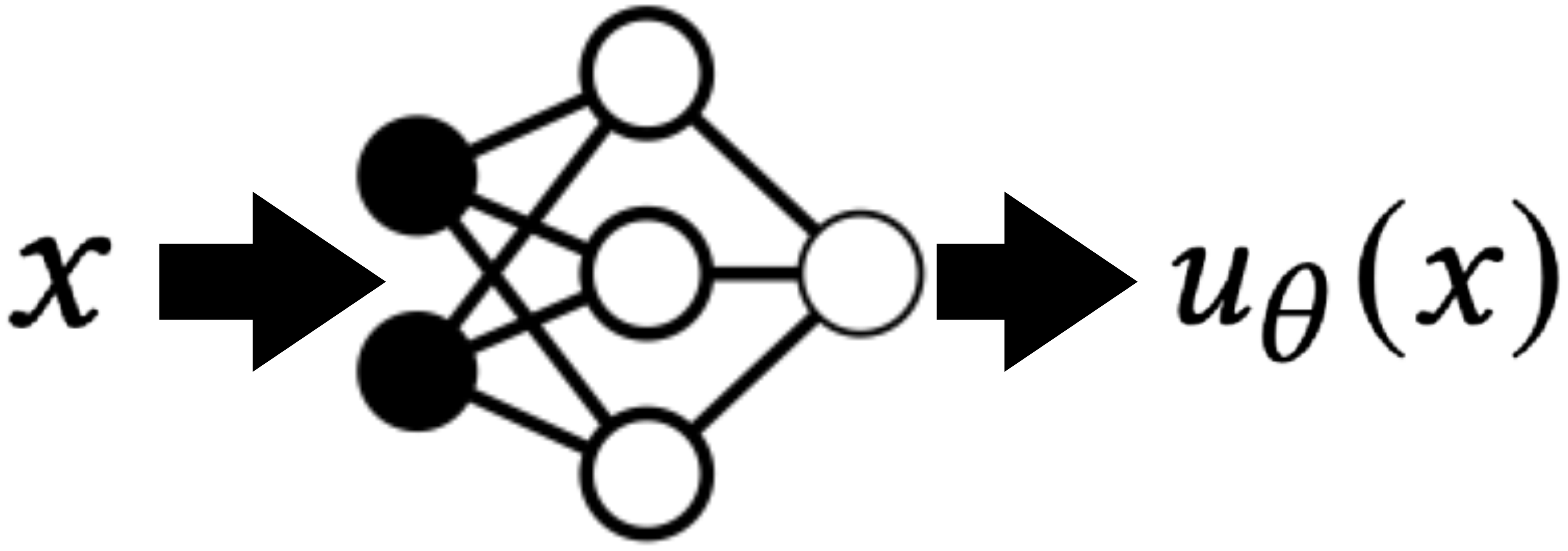
Can we solve PDEs without Discretization?



$$\hat{u}(x) = \begin{cases} g(\bar{x}) & \text{if } d_\Omega(x) < \epsilon \\ \hat{u}(y_i),\ y_i \sim \mathcal{U}_{\partial B(x)} & \text{otherwise} \end{cases}$$

Derive an integral solution for the PDE; estimate the integral by Monte Carlo method.

(Shawney and Crane, 2020)

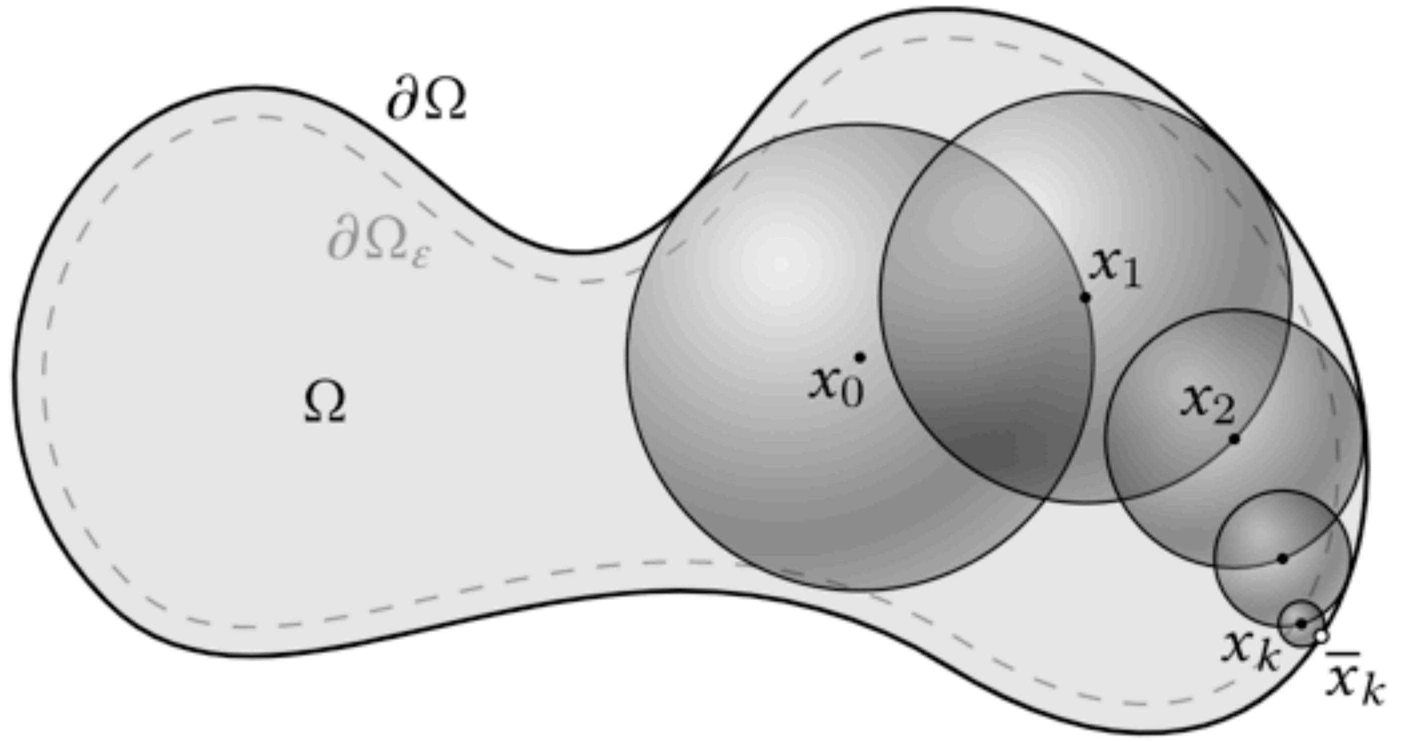


$$\mathcal{L}(\theta) = \int_\Omega |u_\theta(x) - f(x)|^2\, dx + \int_{\partial\Omega} |u_\theta(x) - g(x)|^2\, dx$$

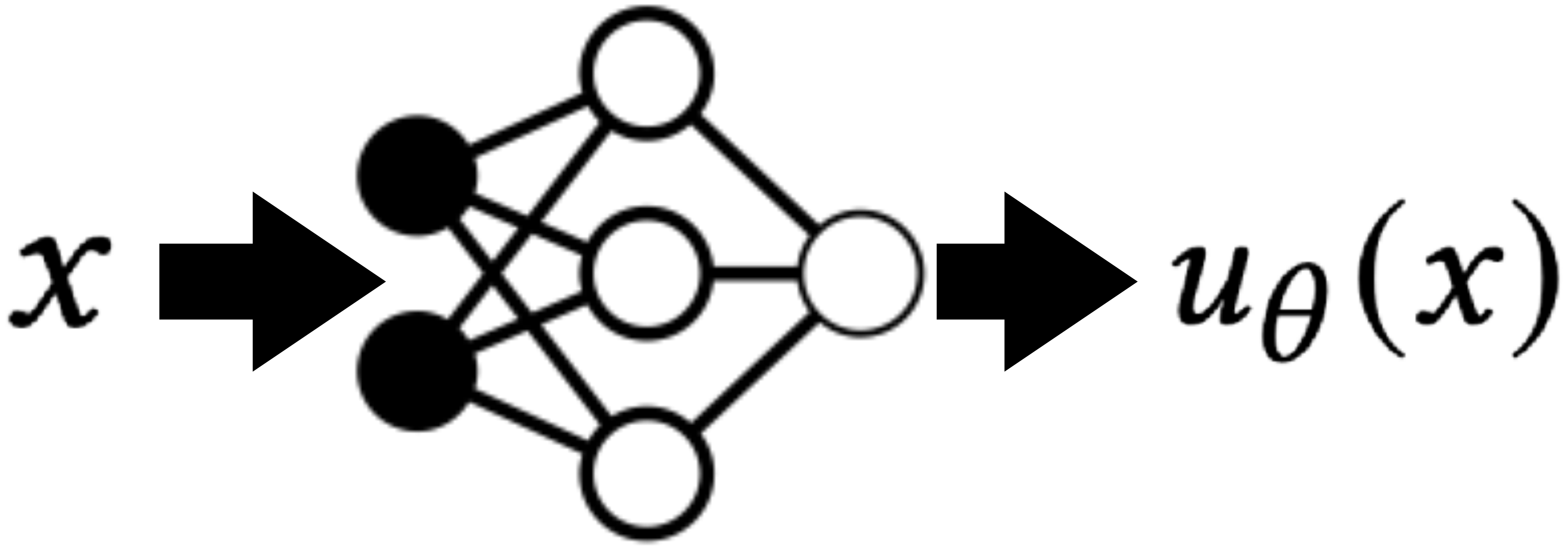
Neural network represent the mapping from spatial coordinate to the PDE solutions; train with losses to enforce PDE constraints.

(Raissi et. al., 2019, Sitzmann et. al., 2020)

Can we solve PDEs without Discretization?



$$\hat{u}(x) = \begin{cases} g(\bar{x}) & \text{if } d_\Omega(x) < \epsilon \\ \hat{u}(y_i),\ y_i \sim \mathcal{U}_{\partial B(x)} & \text{otherwise} \end{cases}$$



$$\mathcal{L}(\theta) = \int_\Omega |u_\theta(x) - f(x)|^2 \, dx + \int_{\partial\Omega} |u_\theta(x) - g(x)|^2 \, dx$$

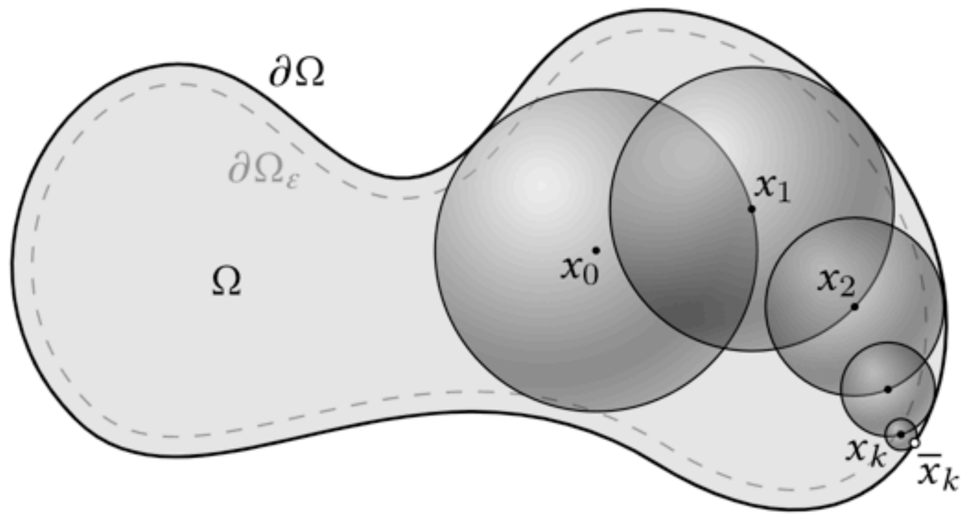
Unbiased (accurate)

High variance (slow)

Biased (inaccurate)

Low-variance (fast)

Can we solve PDEs without Discretization?

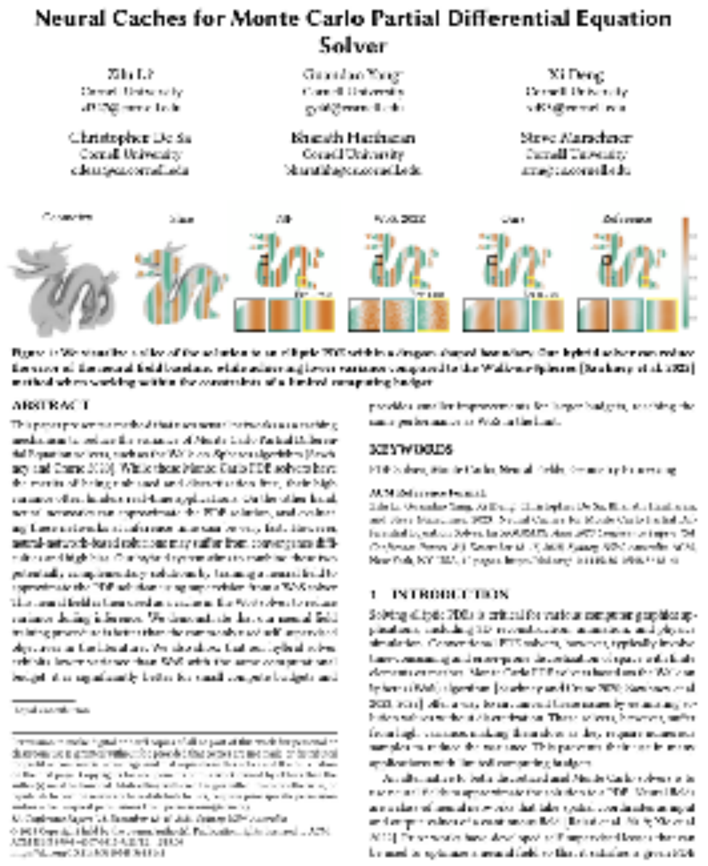


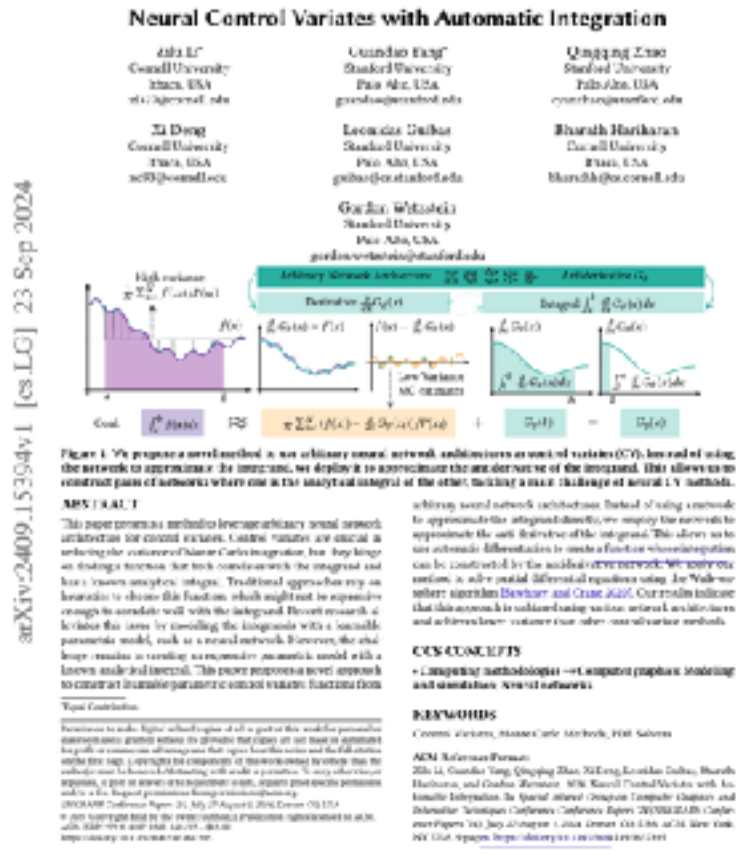
$$\hat{u}(x) = \begin{cases} g(\bar{x}) & \text{if } d_\Omega(x) < \epsilon \\ \hat{u}(y_i),\ y_i \sim \mathcal{U}_{\partial B(x)} & \text{otherwise} \end{cases}$$

Unbiased (accurate)

High variance (slow)

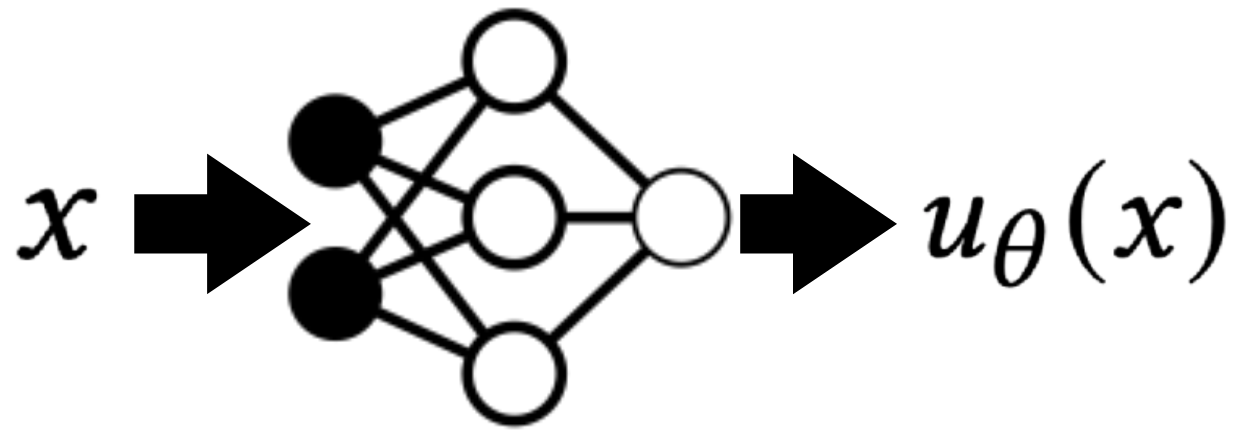
Our hypothesis: hybrid methods are better!


Neural Caches for Monte Carlo Partial Differential Equation Solver


Neural Control Variates with Automatic Integration

arXiv:2409.15394v1 [cs.LG] 23 Sep 2024

Accurate + Fast



$$\mathcal{L}(\theta) = \int_\Omega |u_\theta(x) - f(x)|^2\,dx + \int_{\partial\Omega} |u_\theta(x) - g(x)|^2\,dx$$

Biased (inaccurate)

Low-variance (fast)


Zilu Li


Xi Deng


Qingqing Zhao


Chris De Sa


Bharath Hariharan

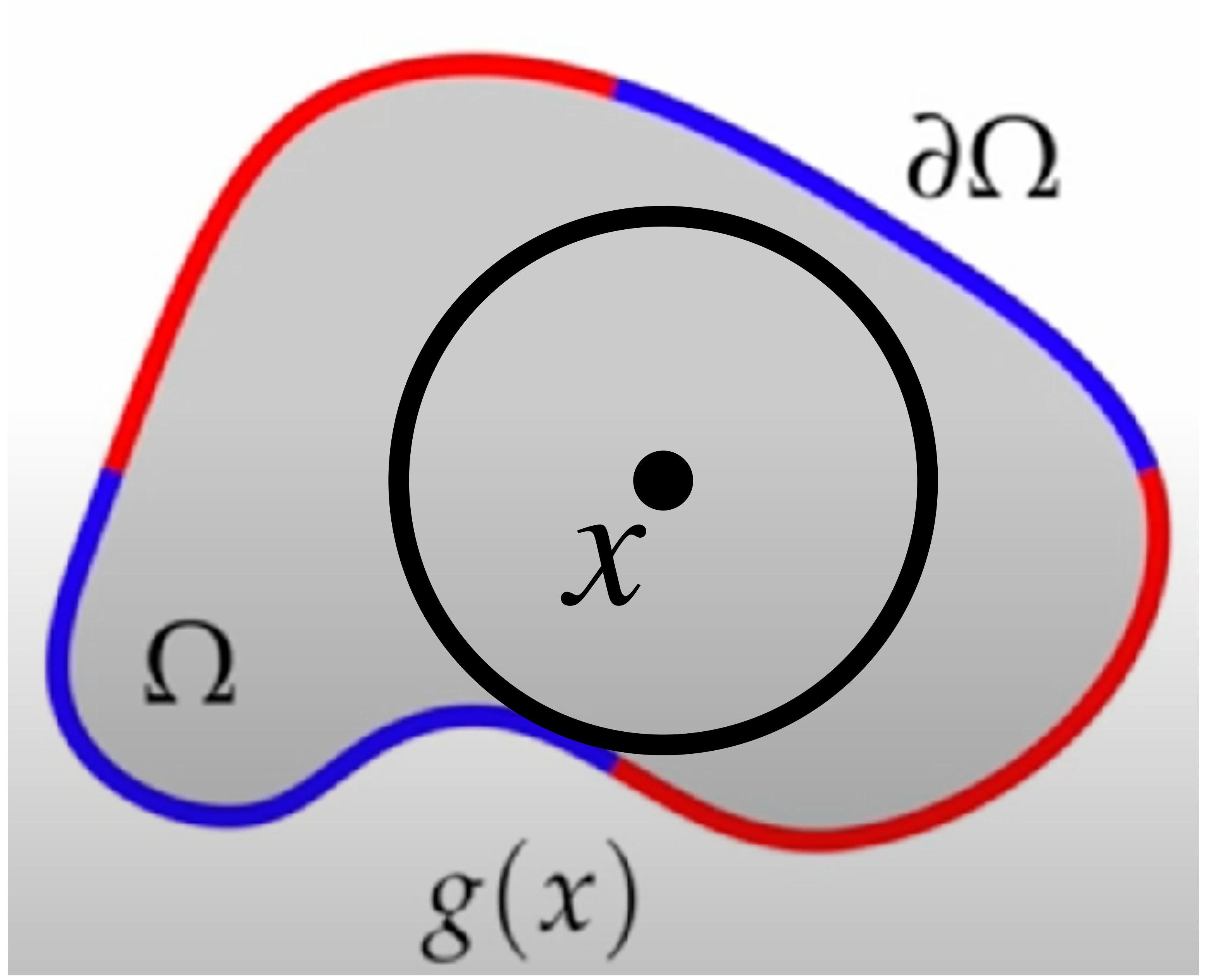

Leonidas Guibas


Steve Marschner


Gordon Wetzstein

Monte Carlo Solver for Laplace Equation

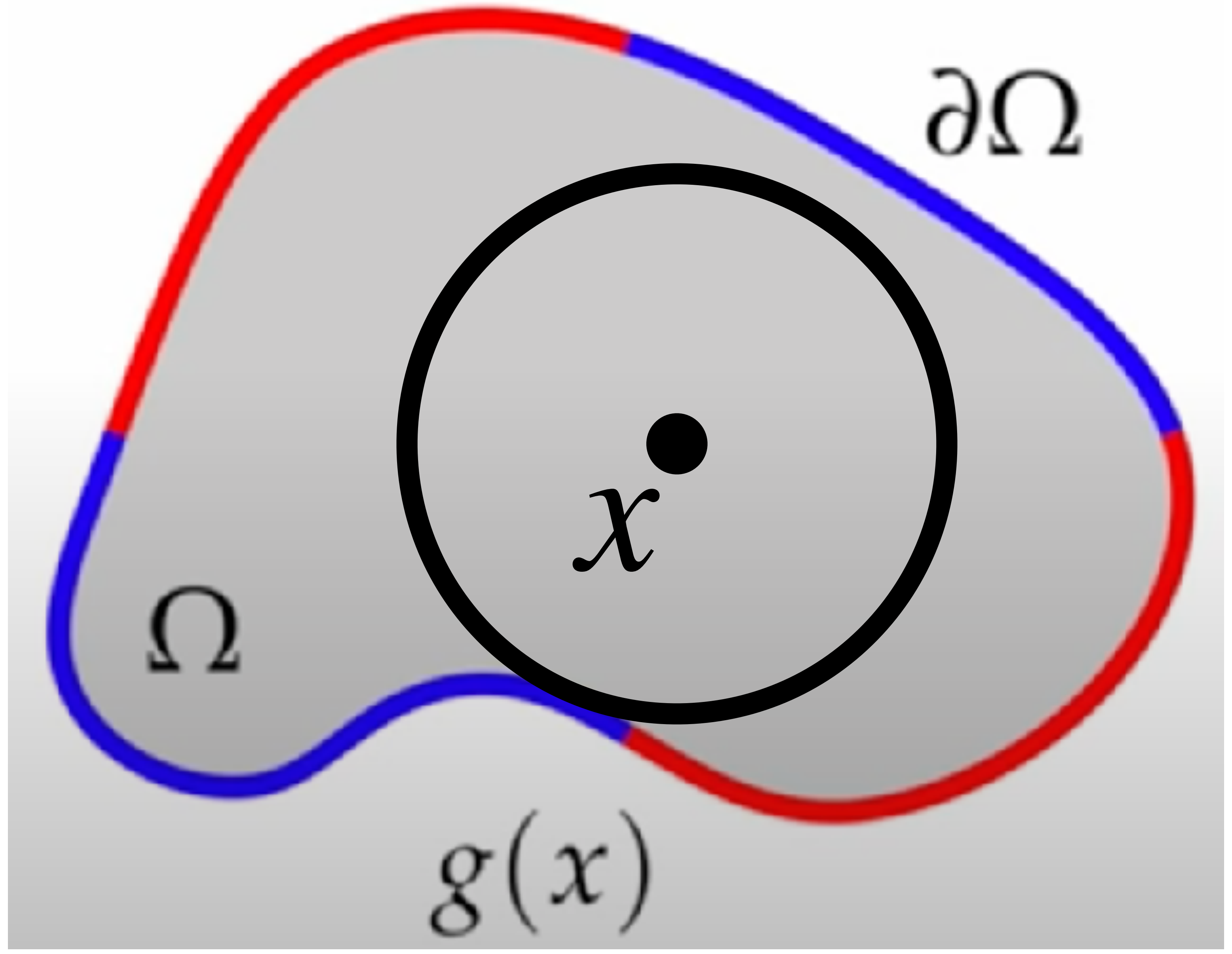
$$\boxed{\begin{aligned} \Delta u &= 0 \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega. \end{aligned}}$$



$$u(x) = \frac{1}{|\partial B(x)|} \int_{\partial B(x)} u(y)dy$$

Monte Carlo Solver for Laplace Equation

$$\boxed{\begin{aligned} \Delta u &= 0 \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega. \end{aligned}}$$

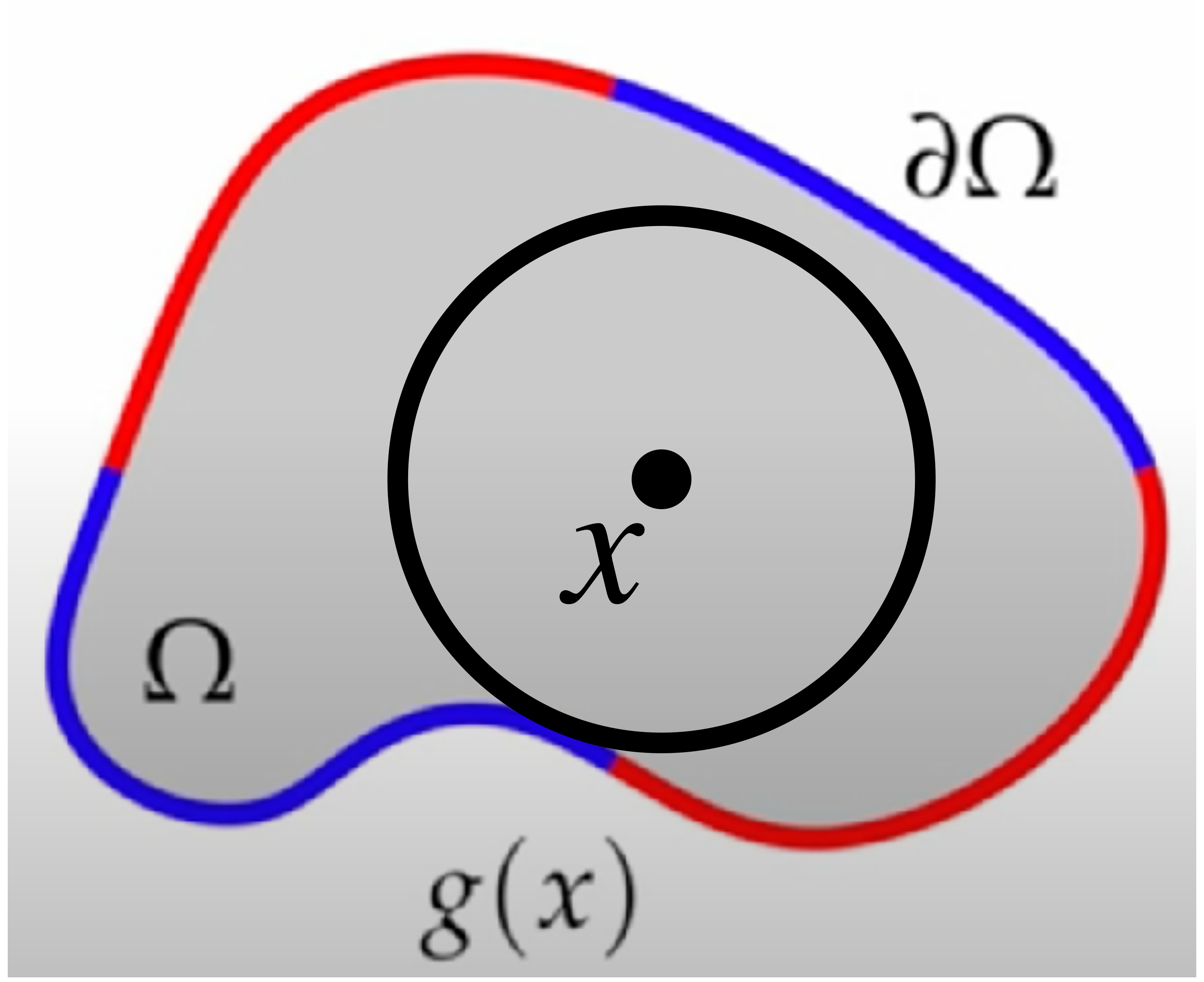


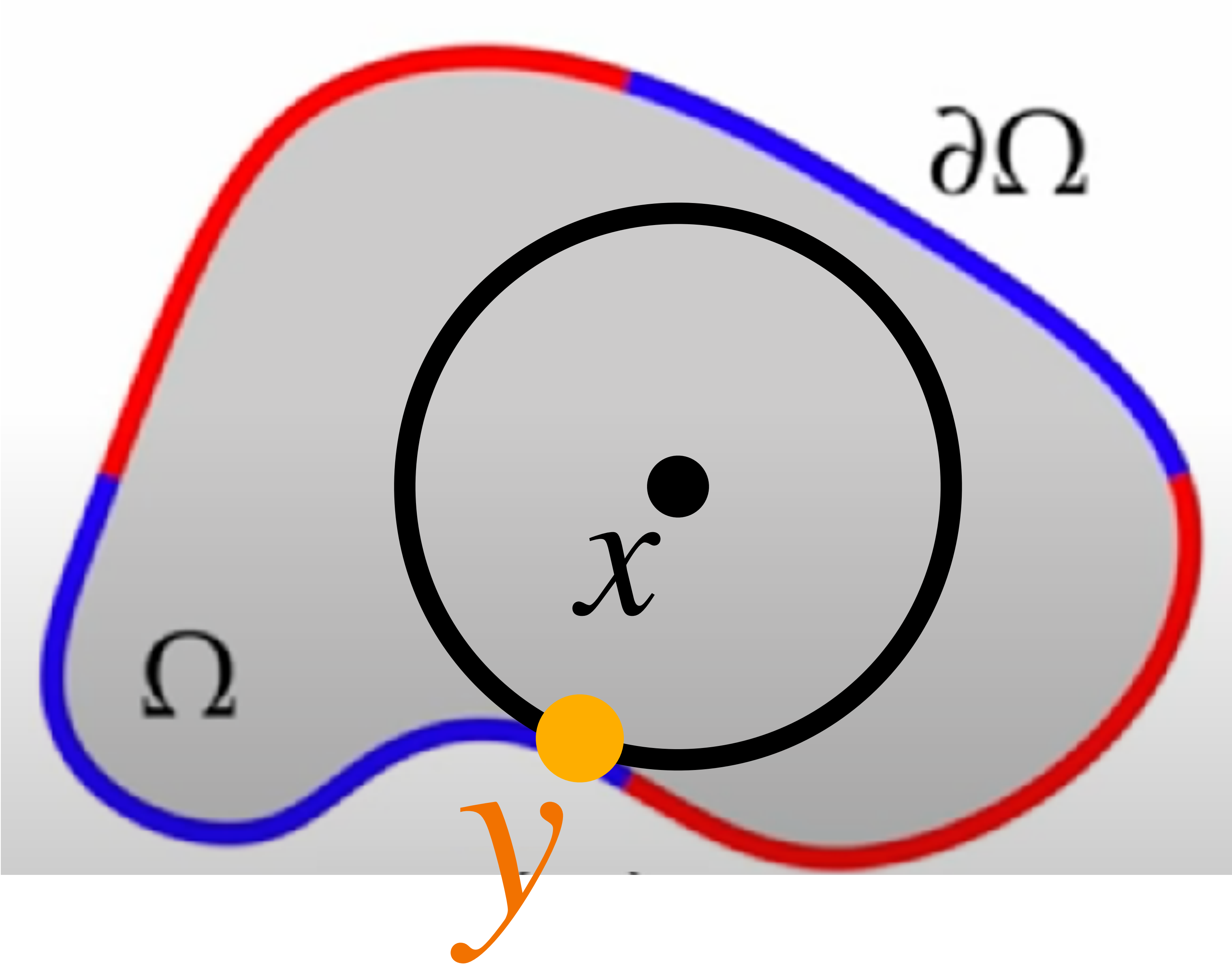
$$u(x) = \frac{1}{|\partial B(x)|} \int_{\partial B(x)} u(y)dy$$

$$\hat{u}(x) = \begin{cases} g(\bar{x}) & \text{if } d_\Omega(x) < \epsilon \\ \hat{u}(y_i),\ y_i \sim \mathcal{U}_{\partial B(x)} & \text{otherwise} \end{cases}$$

Monte Carlo Solver for Laplace Equation

$$\boxed{\begin{aligned} \Delta u &= 0 \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega. \end{aligned}}$$

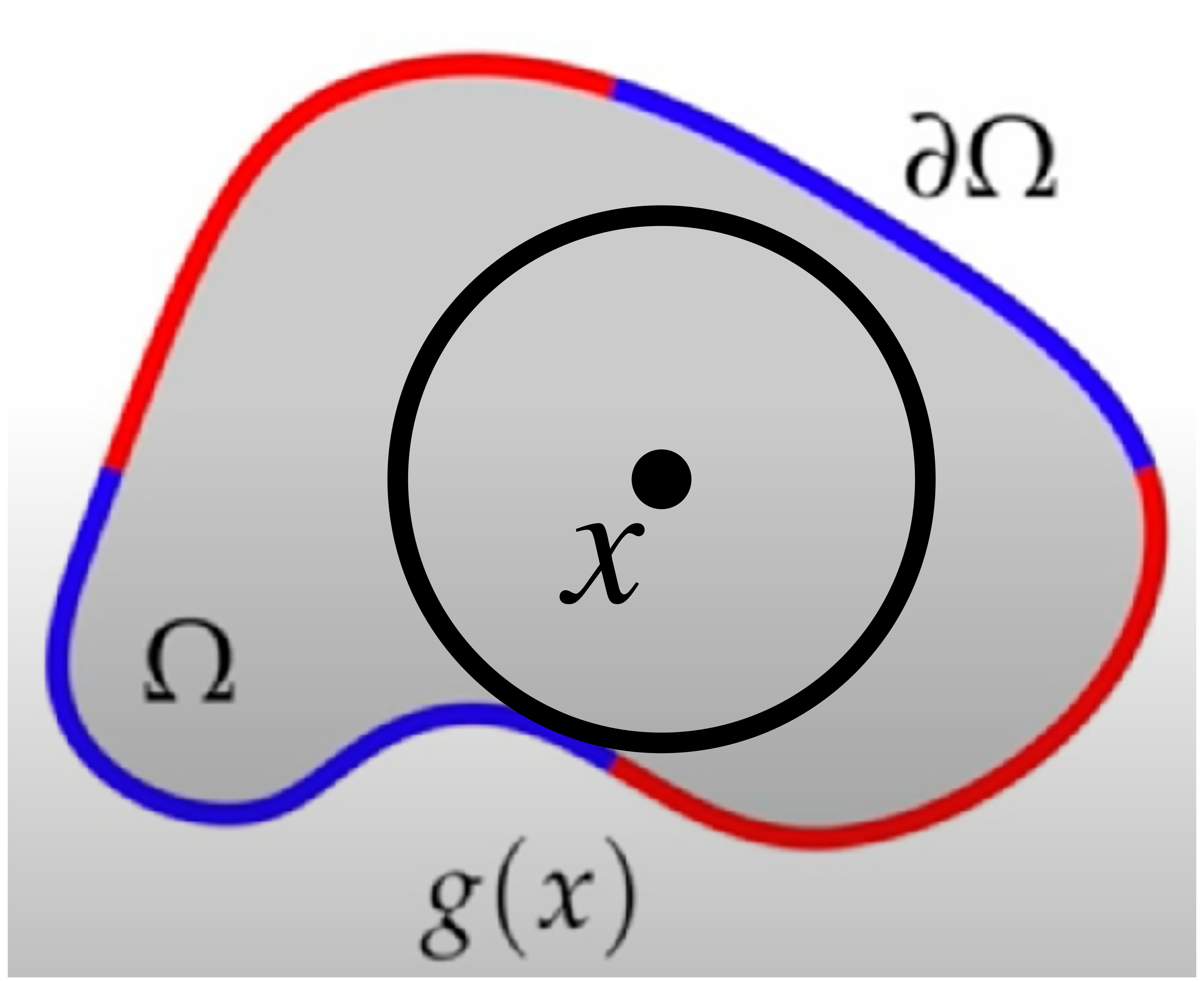


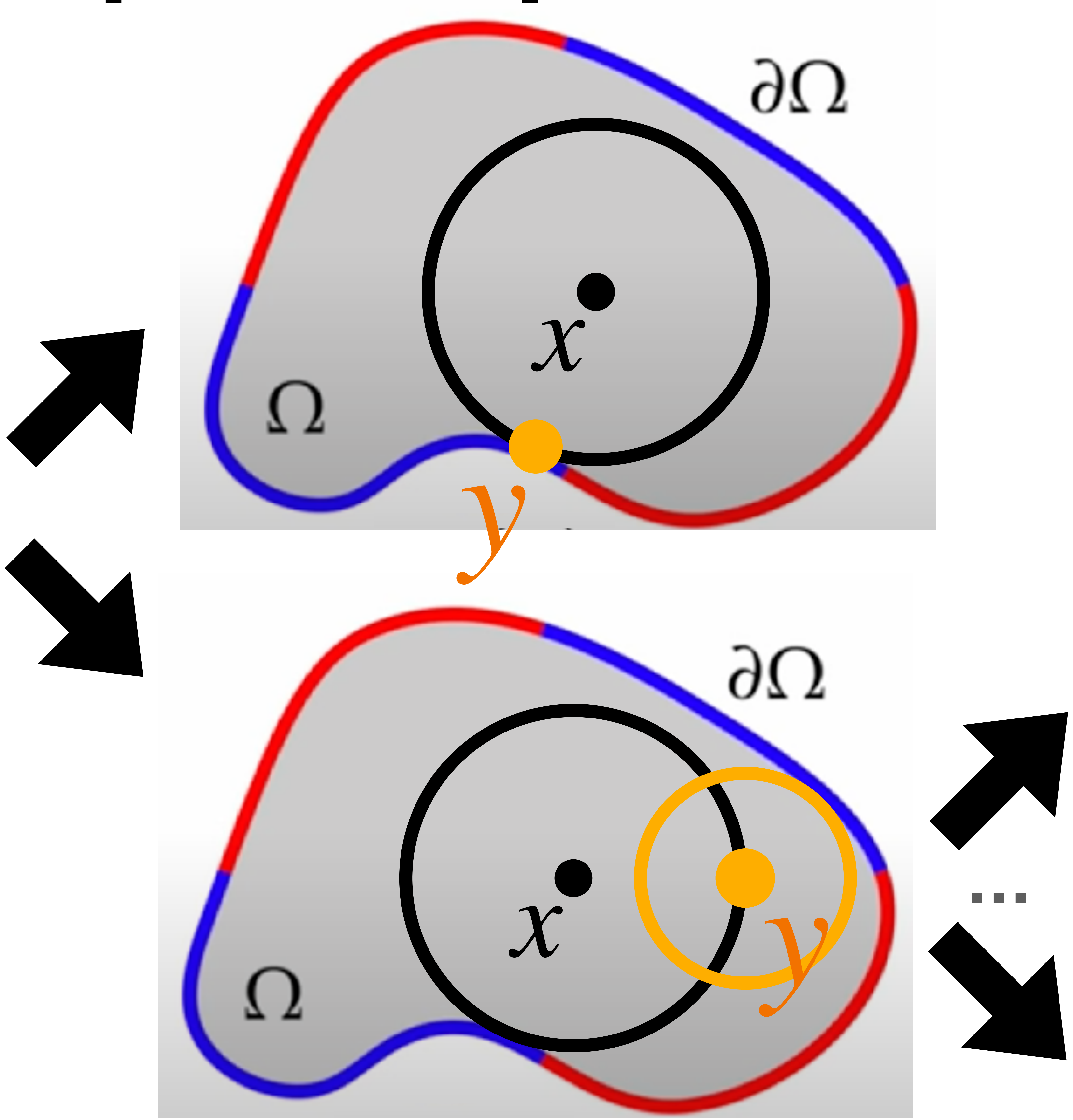


$$\hat{u}(x) = \begin{cases} g(\bar{x}) & \text{if } d_\Omega(x) < \epsilon \\ \hat{u}(y_i),\ y_i \sim \mathcal{U}_{\partial B(x)} & \text{otherwise} \end{cases}$$

Monte Carlo Solver for Laplace Equation

$$\boxed{\begin{aligned} \Delta u &= 0 \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega. \end{aligned}}$$

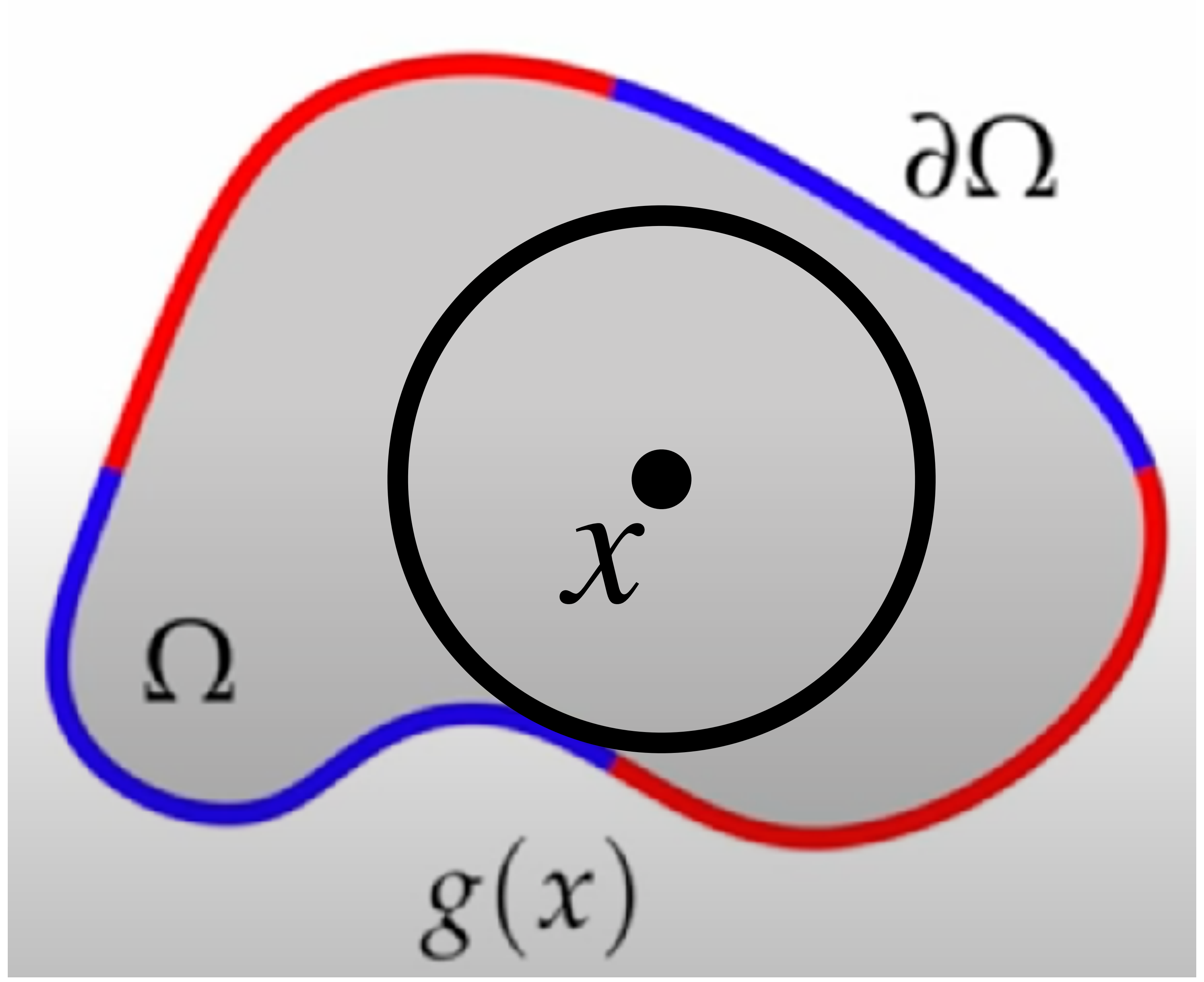


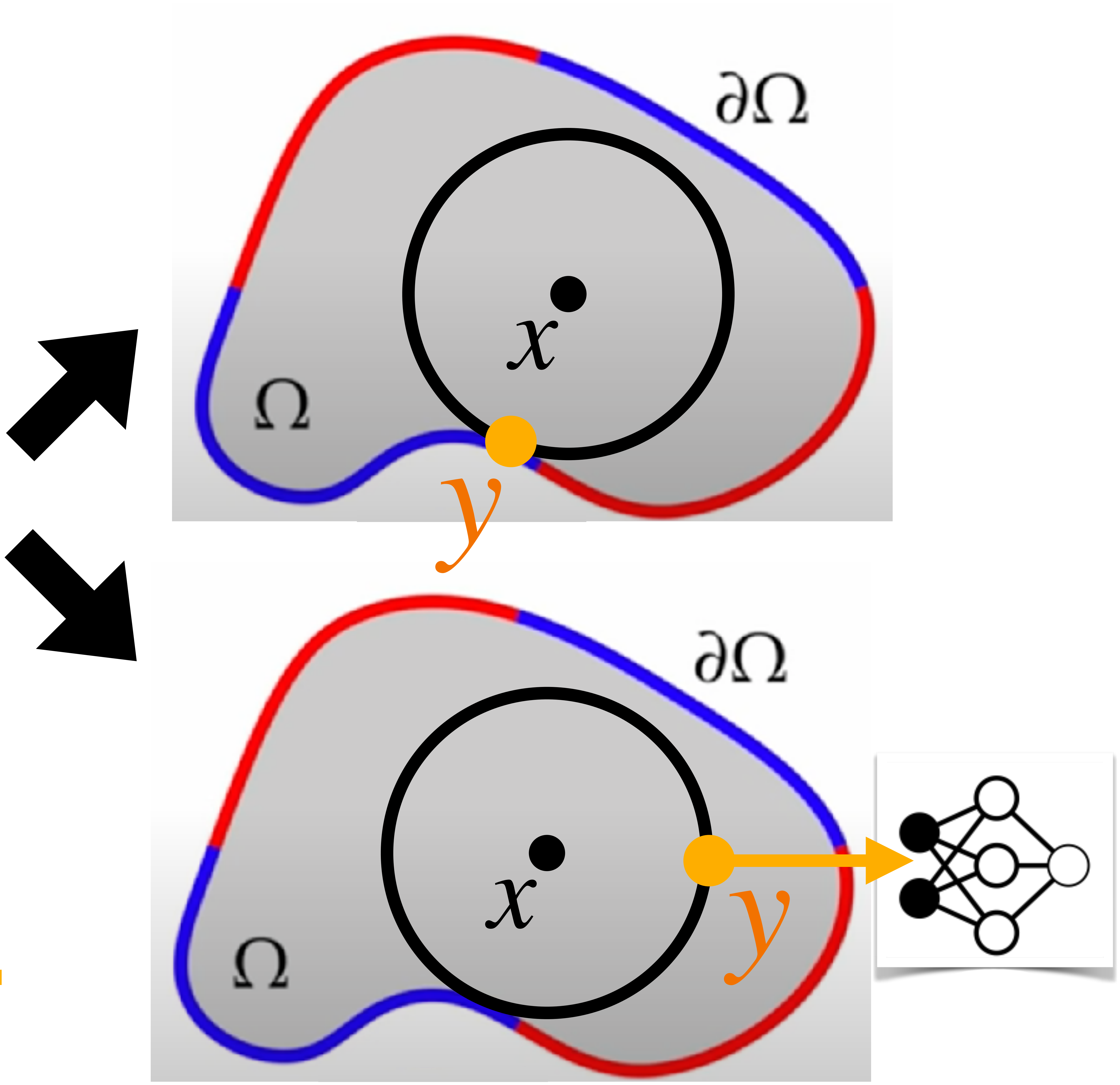


$$\hat{u}(x) = \begin{cases} g(\bar{x}) & \text{if } d_\Omega(x) < \epsilon \\ \hat{u}(y_i),\ y_i \sim \mathcal{U}_{\partial B(x)} & \text{otherwise} \end{cases}$$

Our Method

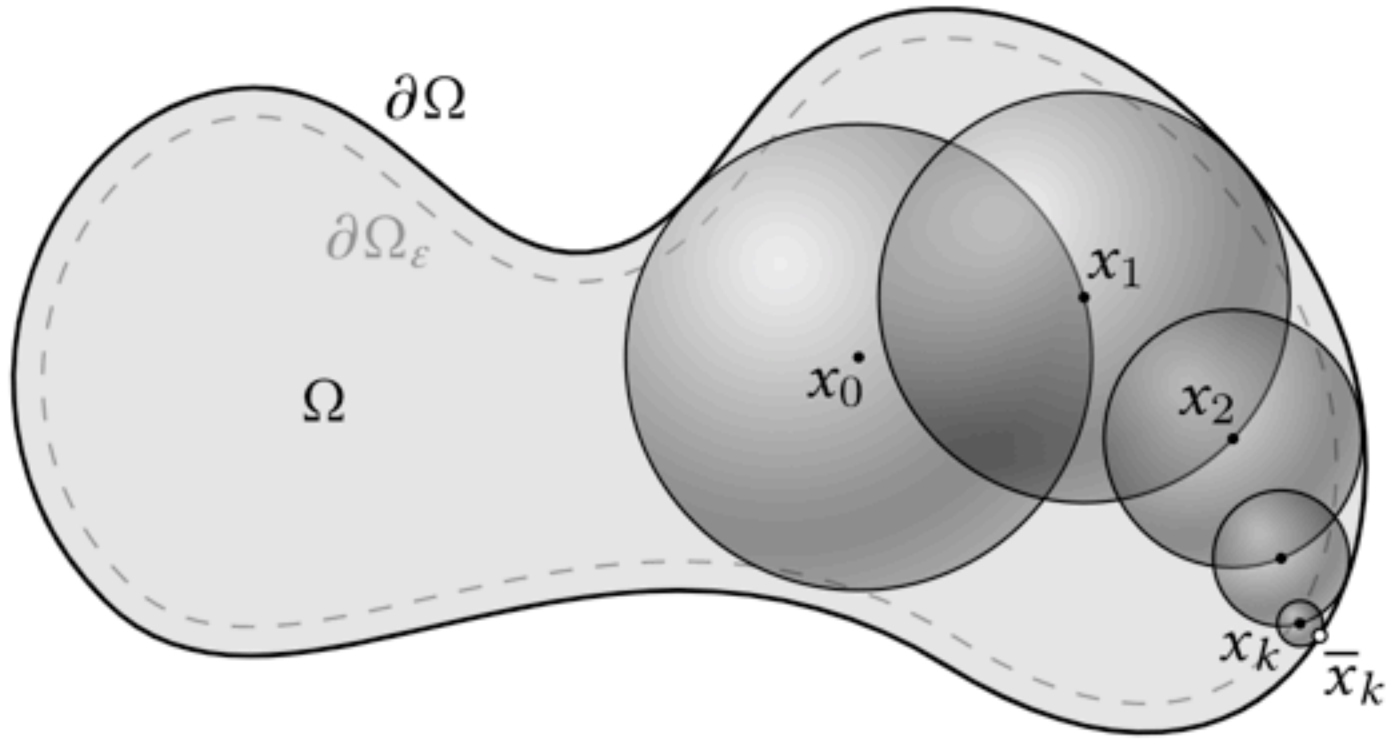
$$\boxed{\begin{aligned} \Delta u &= 0 \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega. \end{aligned}}$$





$$\hat{u}_{\theta,n}(x) = \begin{cases} g(\bar{x}) & \text{if } d_\Omega(x) < \epsilon \\ u_\theta(x) & \text{if } n = 0 \\ \hat{u}_{\theta,n-1}(y_i),\ y_i \sim \mathcal{U}_{\partial B(x)} & \text{otherwise} \end{cases}$$

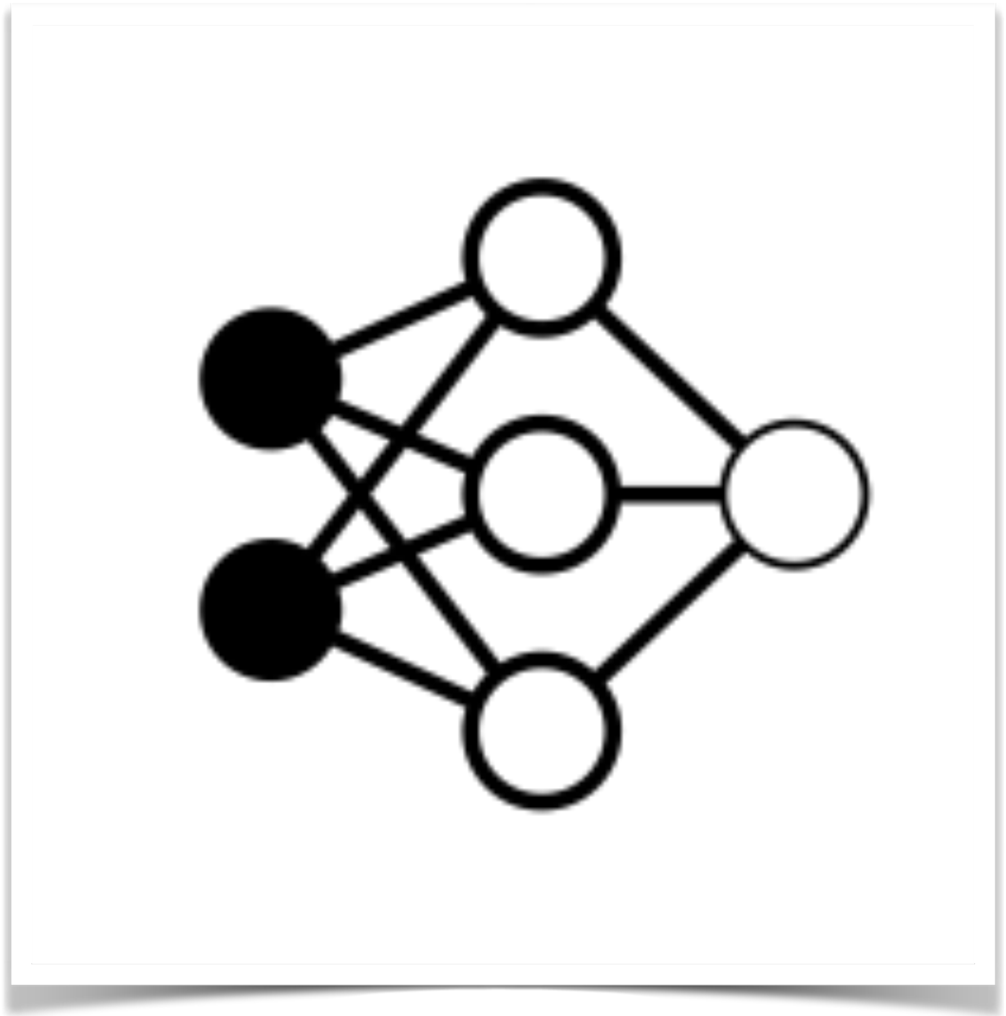
Our method - training



$$(x_0^{(1)}, \hat{u}(x_0^{(1)}))$$

$$\dots$$

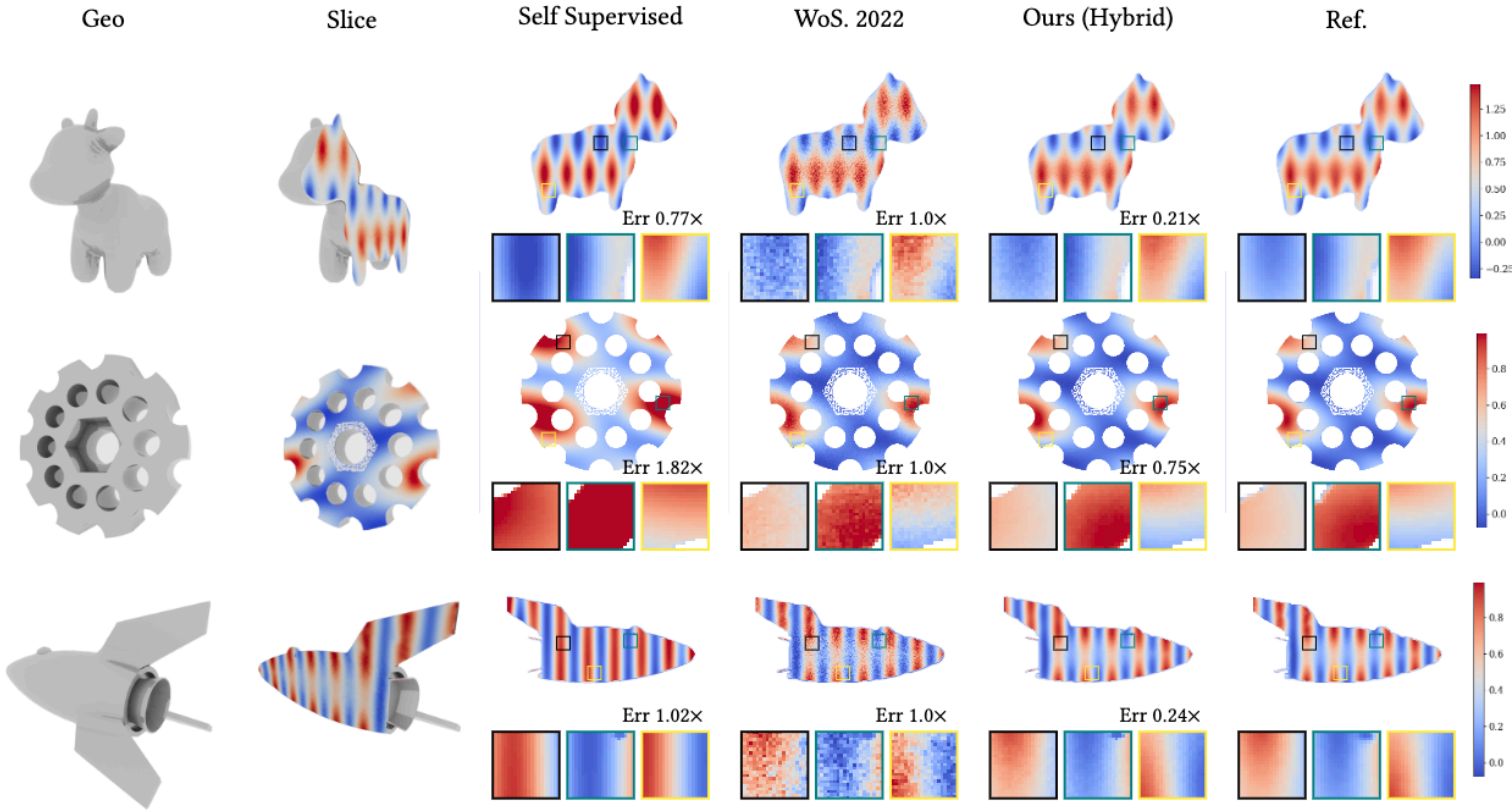
$$(x_0^{(n)}, \hat{u}(x_0^{(n)}))$$



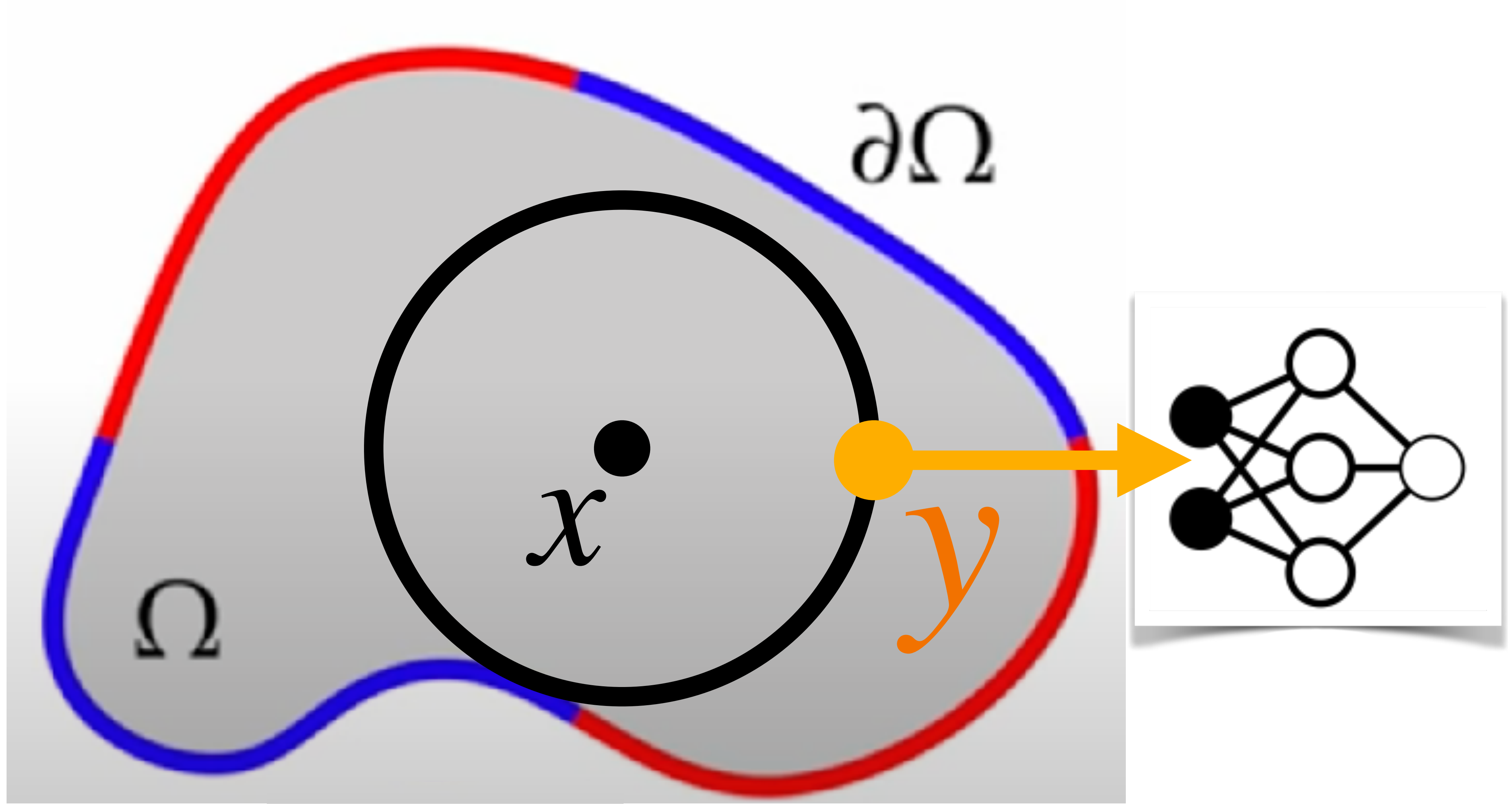
$$y_i^{(k+1)} = (ky_i^{(k)} + \hat{u}(x_i))/(k+1)$$

$$\mathcal{L}_t(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\|u_\theta(x_i) - y_i^{(t)}\right\|^2$$

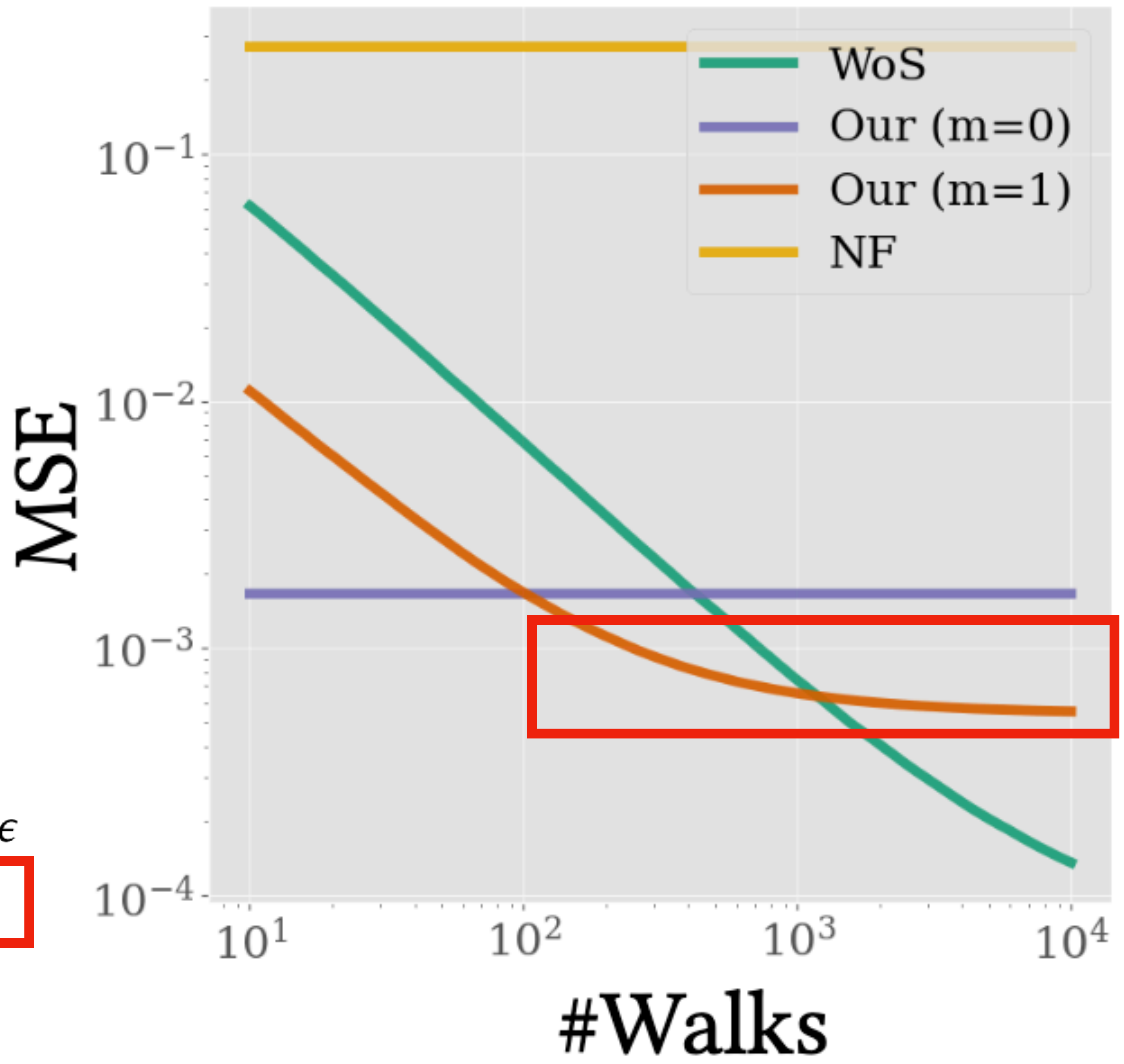

Cow Scene
Training
Inference
Time (min)
25
20
15
10
5
0
WoS
Ours
WoS
Ours
WoS
Ours
256 x 256
512 x 512
1024 x 1024
Resolution
Bunny Scene
Training
Inference
200
175
150
125
100
75
50
25
0
WoS
Ours
WoS
Ours
WoS
Ours
10x faster
256 x 256
512 x 512
1024 x 1024
Resolution


Geo
Slice
Self Supervised
WoS. 2022
Ours (Hybrid)
Ref.
Err 0.77×
Err 1.0×
Err 0.21×
Err 1.82×
Err 1.0×
Err 0.75×
Err 1.02×
Err 1.0×
Err 0.24×

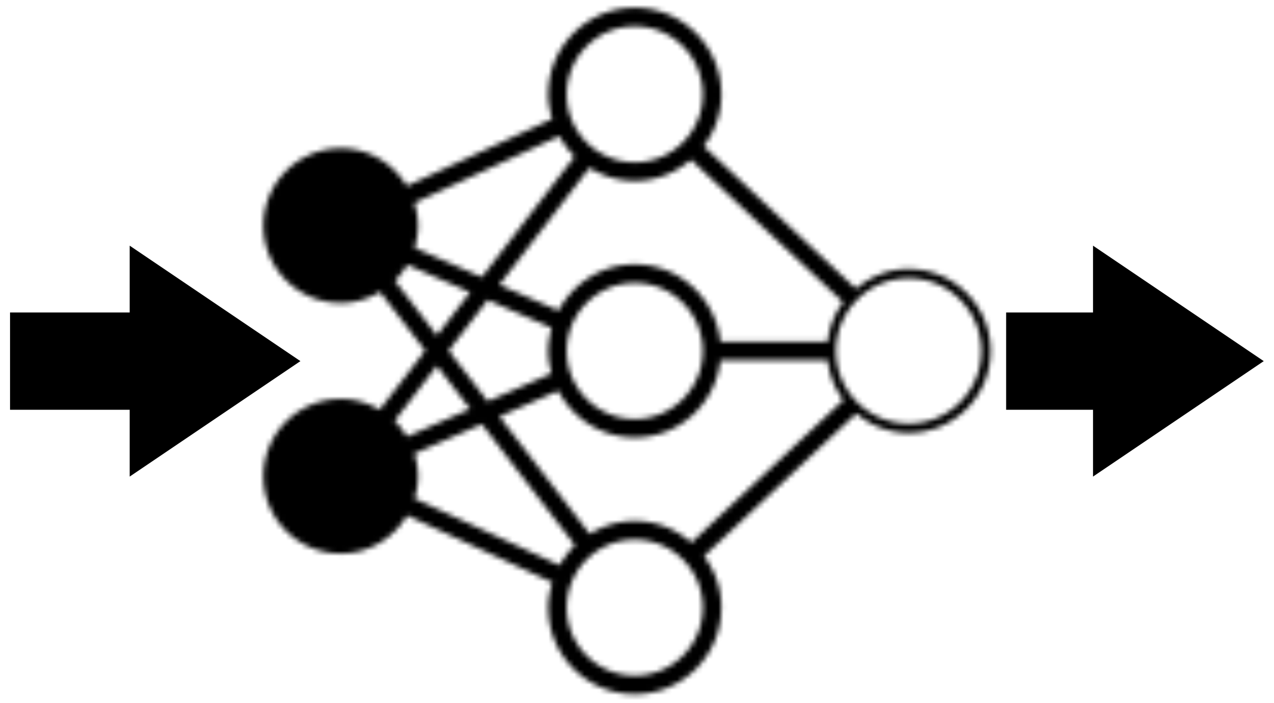
Limitation - Bias



$$\hat{u}_{\theta,n}(x) = \begin{cases} g(\bar{x}) & \text{if } d_\Omega(x) < \epsilon \\ \boxed{u_\theta(x) \quad \text{if } n = 0} & \\ \hat{u}_{\theta,n-1}(y_i),\ y_i \sim \mathcal{U}_{\partial B(x)} & \text{otherwise} \end{cases}$$

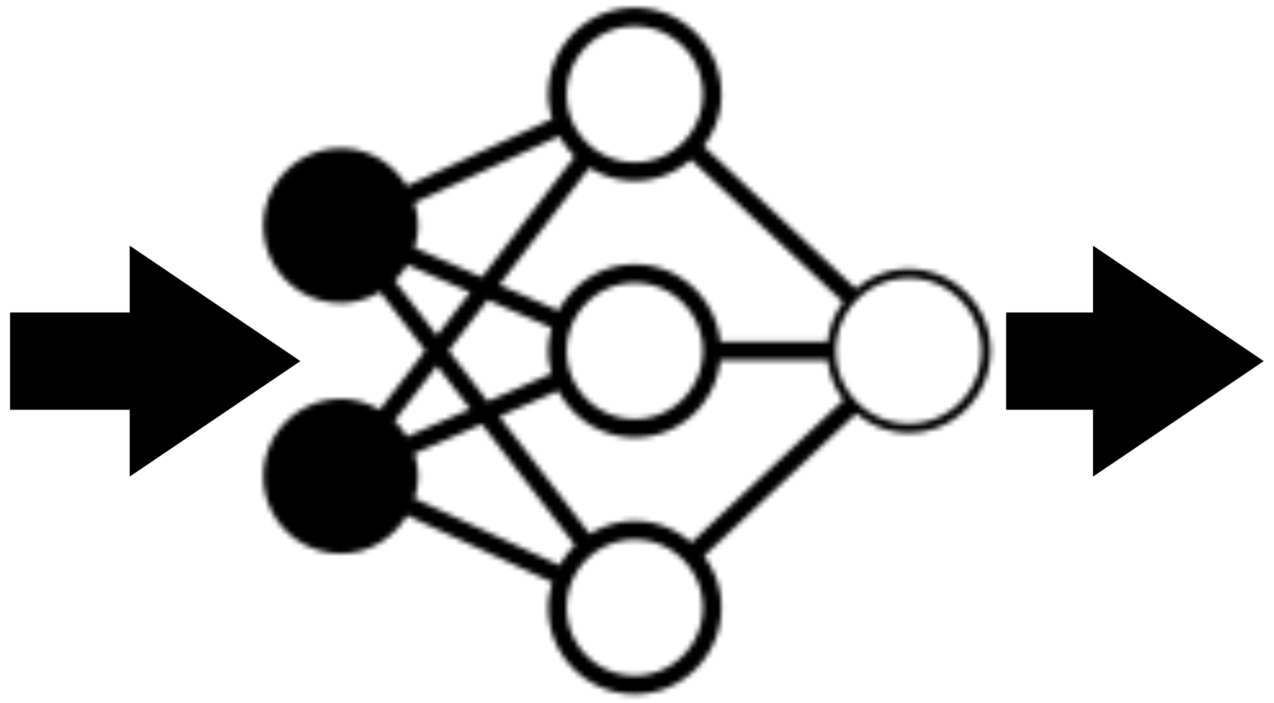


Neural field is a biased estimator for Integral

x  $u_\theta(x) \boxed{\approx} \int_{y\in\partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy$

Solution: Control Variates

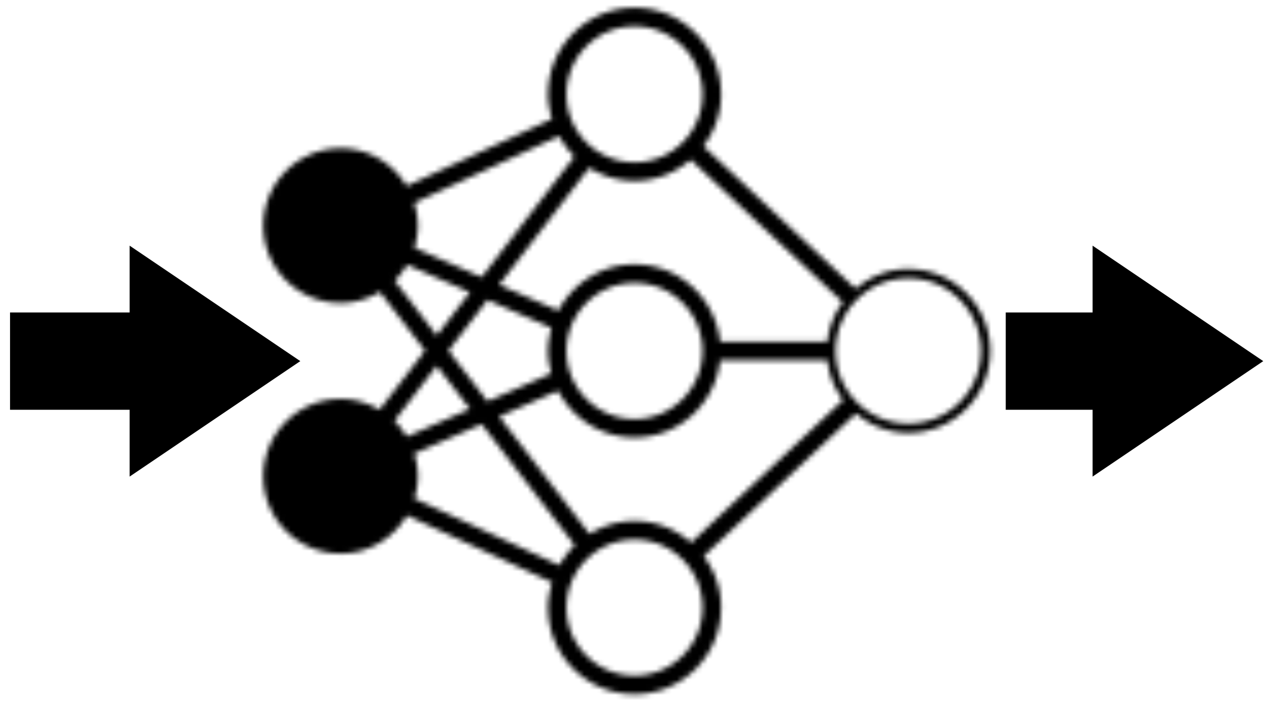
x



$$u_\theta(x) \boxed{\approx} \int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy$$

$$\int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy = {\color{blue} u_\theta(x) - u_\theta(x)} + \int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy$$

Solution: Control Variates

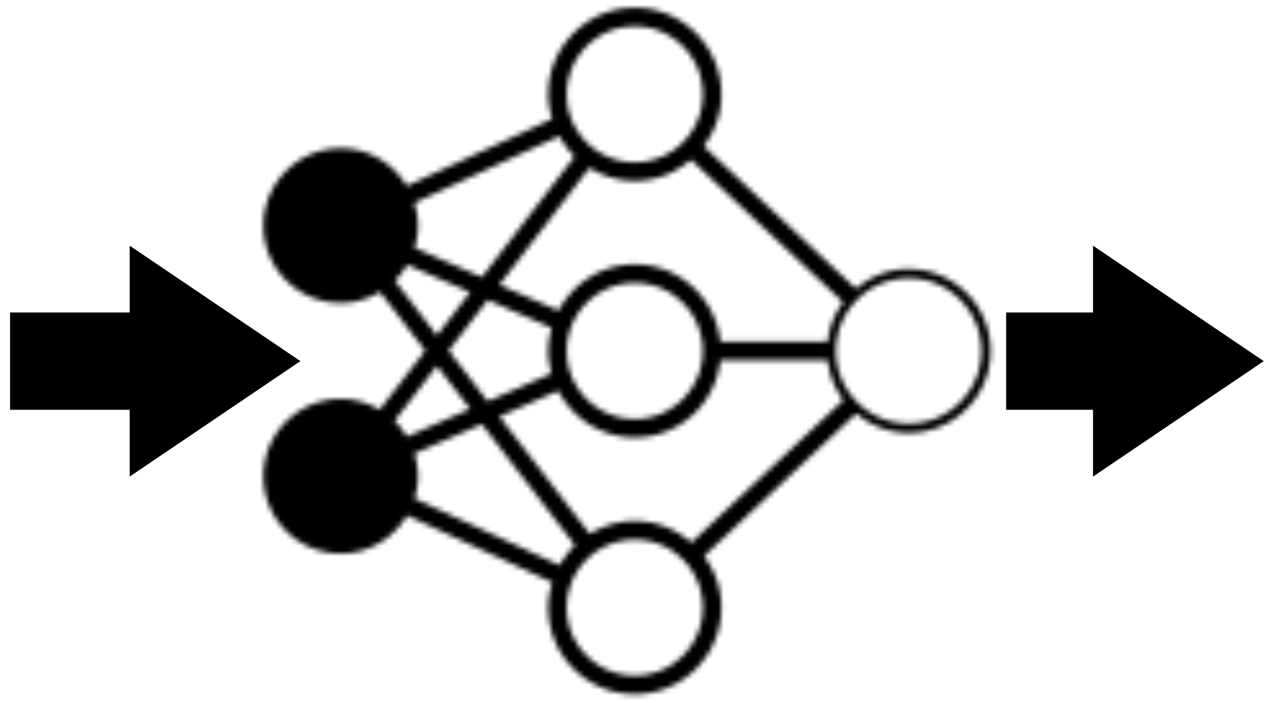
x  $u_\theta(x) \boxed{\approx} \int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy$

$$\int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy = u_\theta(x) - u_\theta(x) + \int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy$$

$$= u_\theta(x) + \int_{y \in \partial\Omega_r(x)} \frac{u(y) - v_\theta(y)}{2\pi r} dy$$

$$\boxed{u_\theta(x) = \int_{y \in \partial\Omega_r(x)} \frac{v_\theta(y)}{2\pi r} dy}$$

Solution: Control Variates

x  $u_\theta(x) \boxed{\approx} \int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy$

$$\begin{aligned}
\int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy &= u_\theta(x) - u_\theta(x) + \int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy \\
&= u_\theta(x) + \int_{y \in \partial\Omega_r(x)} \frac{u(y) - v_\theta(y)}{2\pi r} dy \\
&= u_\theta(x) + \mathbb{E}_{y \in \mathcal{U}[\partial\Omega_r(x)]} \left[u(y) - v_\theta(y) \right]
\end{aligned}$$

Solution: Control Variates

Two requirements:

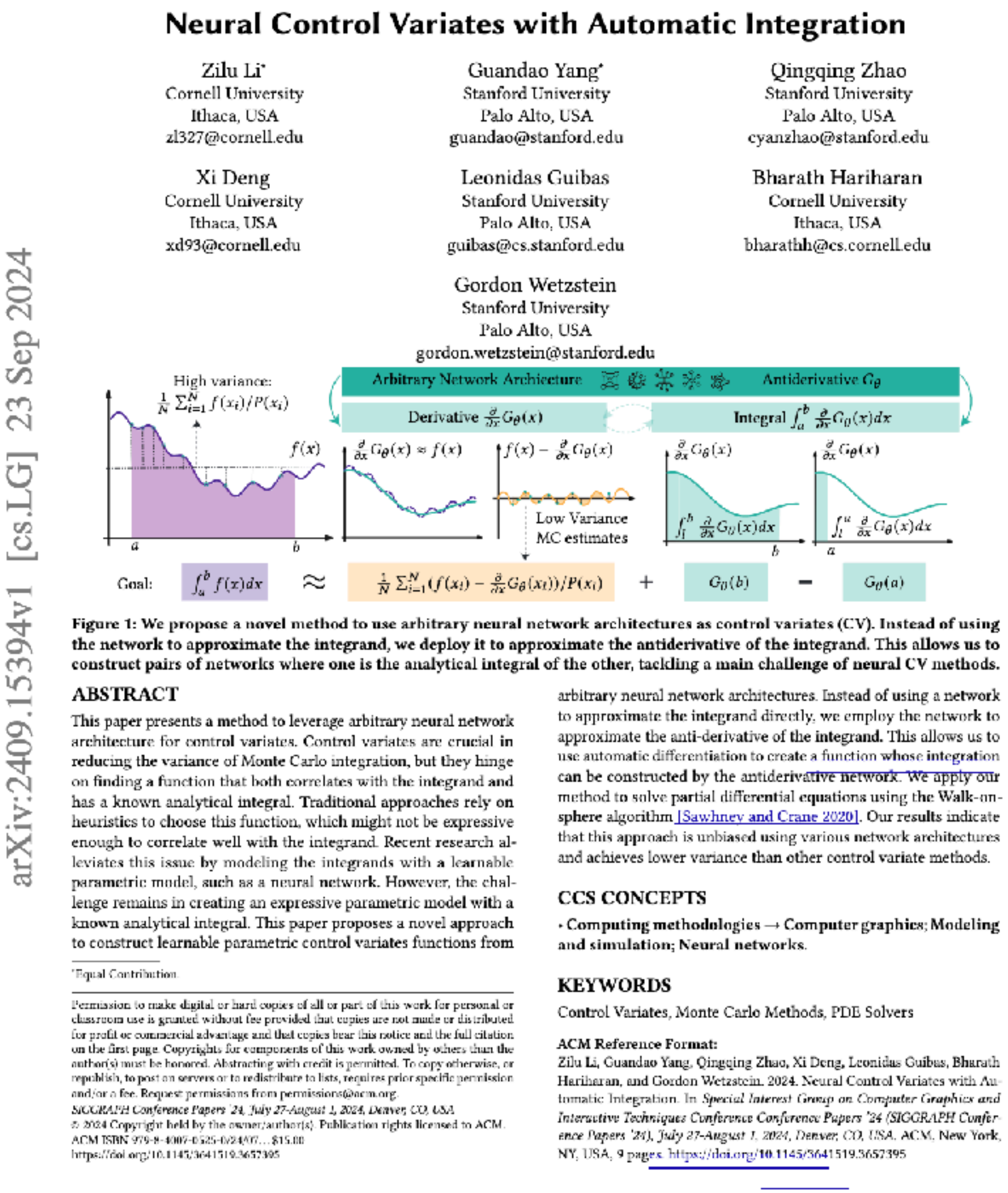
$$\int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy = u_\theta(x) - u_\theta(x) + \int_{y \in \partial\Omega_r(x)} \frac{u(y)}{2\pi r} dy$$

$$= u_\theta(x) + \int_{y \in \partial\Omega_r(x)} \frac{u(y) - v_\theta(y)}{2\pi r} dy$$

$$= u_\theta(x) + \mathbb{E}_{y \in \mathcal{U}[\partial\Omega_r(x)]} \left[u(y) - v_\theta(y) \right]$$

$$\boxed{u_\theta(x) = \int_{y \in \partial\Omega_r(x)} \frac{v_\theta(y)}{2\pi r} dy}$$

$$\boxed{\mathbb{V}\left[u(y) - v_\theta(y)\right] <<< \mathbb{V}\left[u(y)\right]}$$



Neural Control Variates with Automatic Integration

Zilu Li* · Cornell University · Ithaca, USA · zl327@cornell.edu

Guandao Yang* · Stanford University · Palo Alto, USA · guandao@stanford.edu

Qingqing Zhao · Stanford University · Palo Alto, USA · cyanzhao@stanford.edu

Xi Deng · Cornell University · Ithaca, USA · xd93@cornell.edu

Leonidas Guibas · Stanford University · Palo Alto, USA · guibas@cs.stanford.edu

Bharath Hariharan · Cornell University · Ithaca, USA · bharathh@cs.cornell.edu

Gordon Wetzstein · Stanford University · Palo Alto, USA · gordon.wetzstein@stanford.edu

Figure 1: We propose a novel method to use arbitrary neural network architectures as control variates (CV). Instead of using the network to approximate the integrand, we deploy it to approximate the antiderivative of the integrand. This allows us to construct pairs of networks where one is the analytical integral of the other, tackling a main challenge of neural CV methods.

ABSTRACT

This paper presents a method to leverage arbitrary neural network architecture for control variates. Control variates are crucial in reducing the variance of Monte Carlo integration, but they hinge on finding a function that both correlates with the integrand and has a known analytical integral. Traditional approaches rely on heuristics to choose this function, which might not be expressive enough to correlate well with the integrand. Recent research alleviates this issue by modeling the integrands with a learnable parametric model, such as a neural network. However, the challenge remains in creating an expressive parametric model with a known analytical integral. This paper proposes a novel approach to construct learnable parametric control variates functions from arbitrary neural network architectures. Instead of using a network to approximate the integrand directly, we employ the network to approximate the anti-derivative of the integrand. This allows us to use automatic differentiation to create a function whose integration can be constructed by the antiderivative network. We apply our method to solve partial differential equations using the Walk-on-sphere algorithm [Sawhney and Crane 2020]. Our results indicate that this approach is unbiased using various network architectures and achieves lower variance than other control variate methods.

*Equal Contribution.

SIGGRAPH Conference Papers '24, July 27-August 1, 2024, Denver, CO, USA

ACM ISBN 979-8-4007-0525-0/24/07…$15.00
https://doi.org/10.1145/3641519.3657395

CCS CONCEPTS

• **Computing methodologies → Computer graphics; Modeling and simulation; Neural networks.**

KEYWORDS

Control Variates, Monte Carlo Methods, PDE Solvers

ACM Reference Format:
Zilu Li, Guandao Yang, Qingqing Zhao, Xi Deng, Leonidas Guibas, Bharath Hariharan, and Gordon Wetzstein. 2024. Neural Control Variates with Automatic Integration. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24), July 27-August 1, 2024, Denver, CO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3641519.3657395

$$\boxed{u_\theta(x) = \int_{y \in \partial\Omega_r(x)} \frac{v_\theta(y)}{2\pi r} dy}$$

$$\boxed{\mathbb{V}\left[u(y) - v_\theta(y)\right] <<< \mathbb{V}\left[u(y)\right]}$$

Zilu Li

Xi Deng

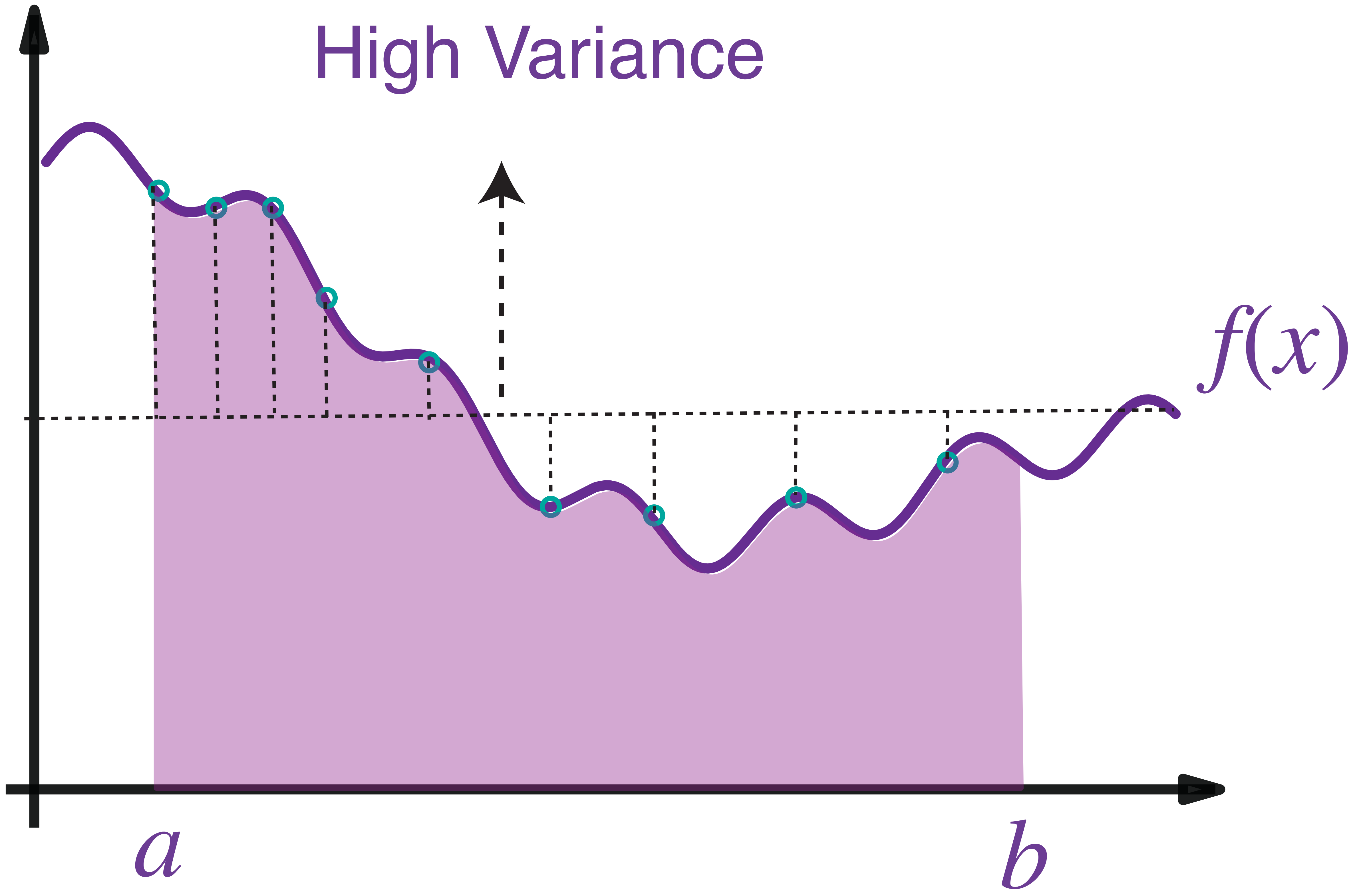
Qingqing Zhao

Bharath Hariharan

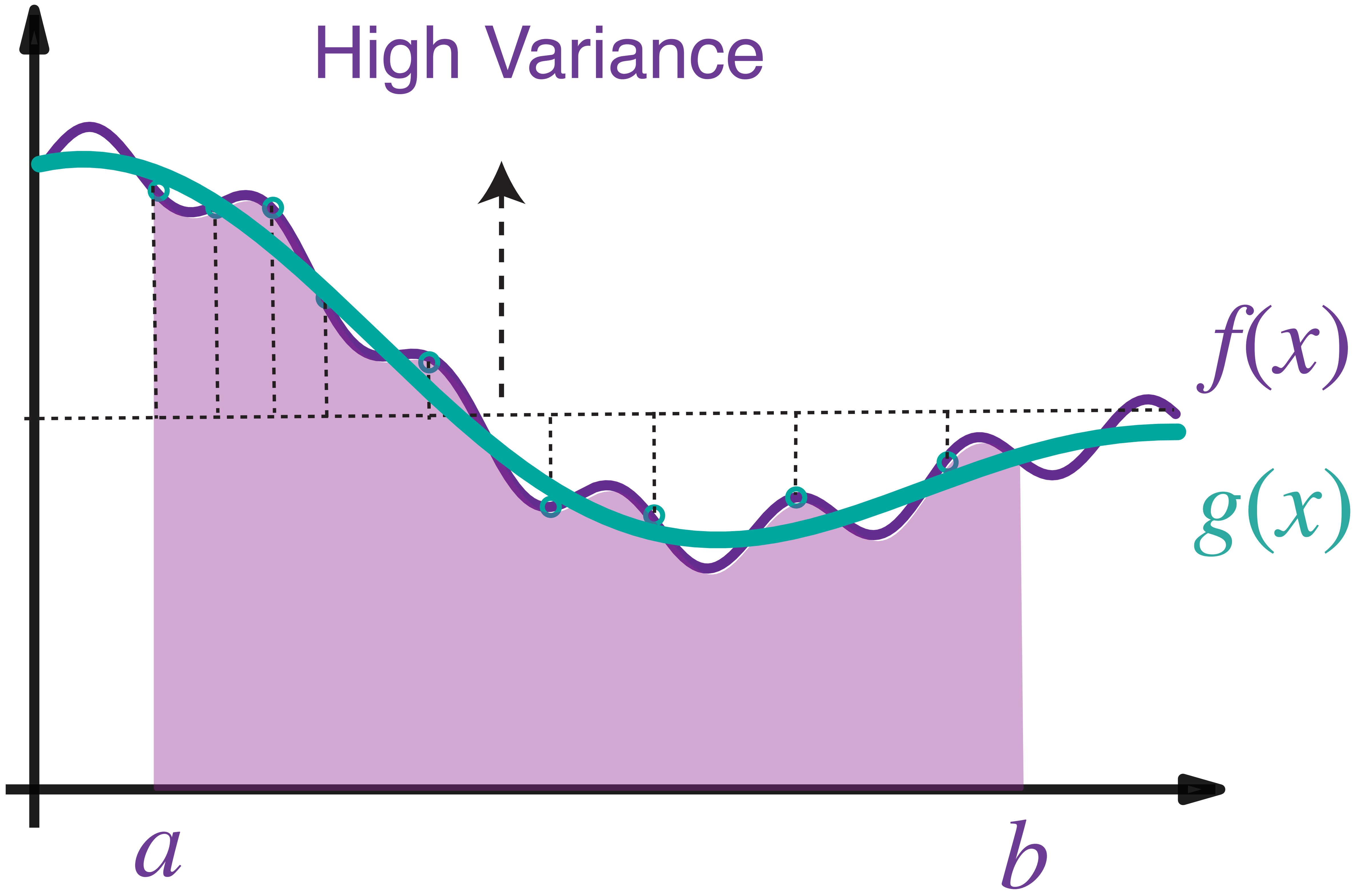
Leonidas Guibas

Gordon Wetzstein

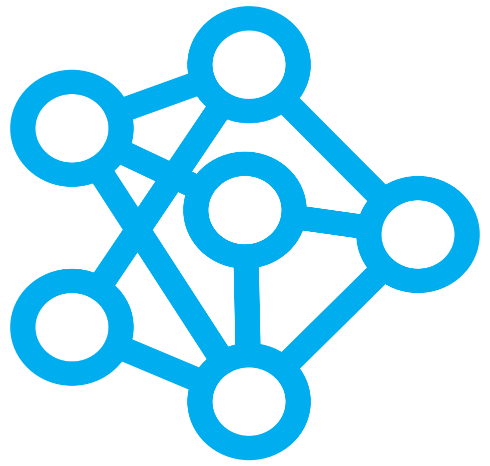
1D Example - estimating $\int_a^b f(x)dx$

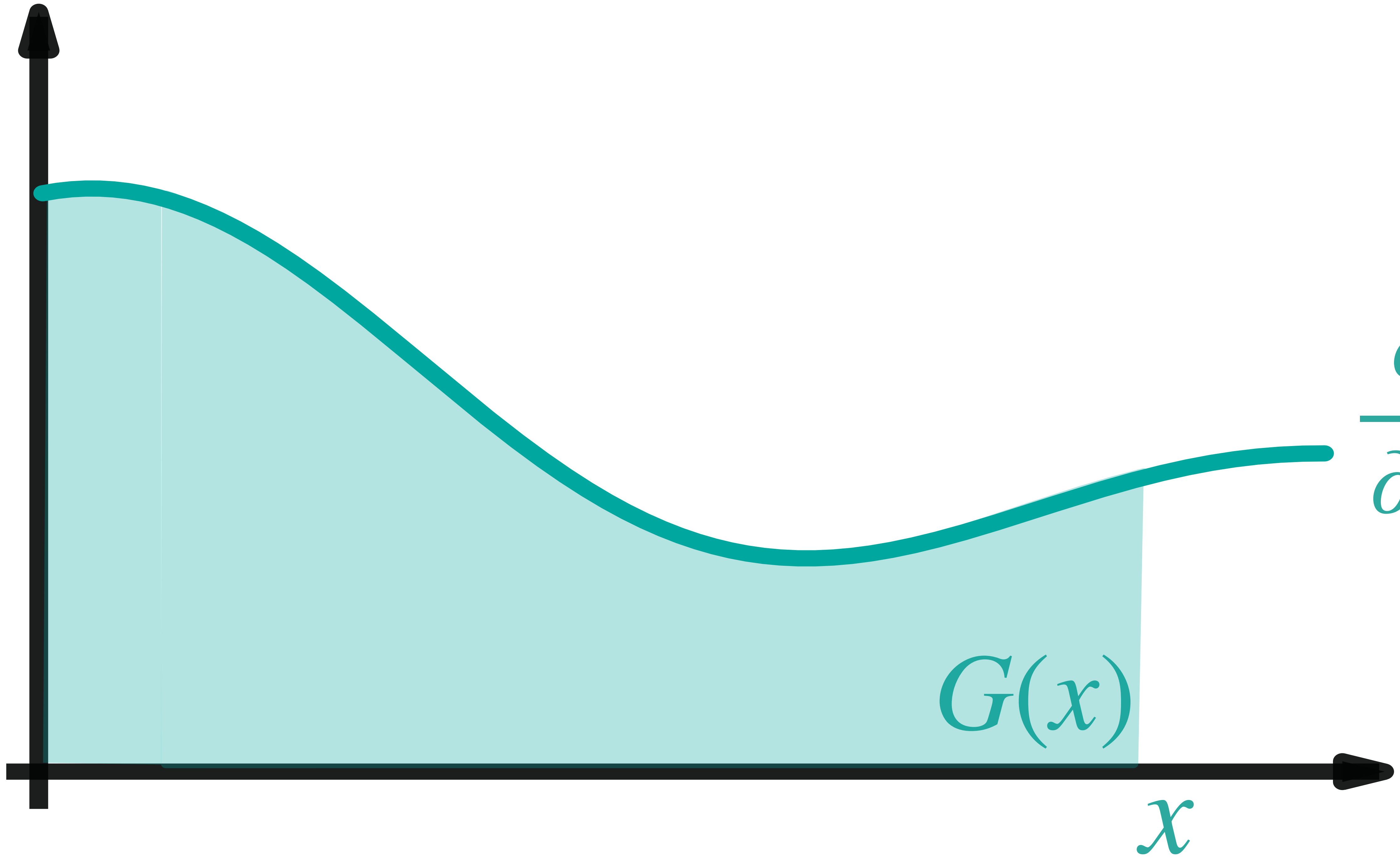


1D Example - estimating $\int_a^b f(x)dx$



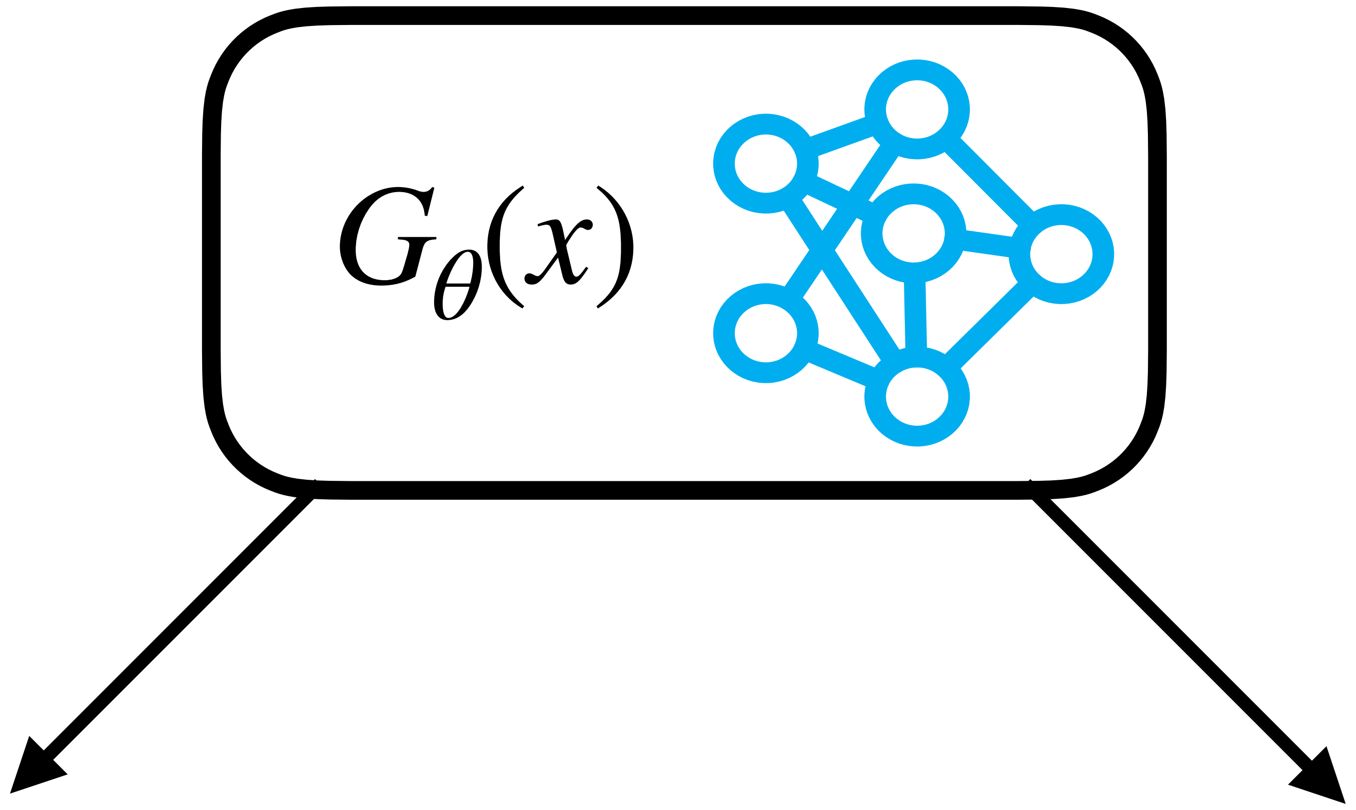
Instantiate the Network to Approximate Indefinite Integral

$G_\theta :$ $\mathbb{R} \rightarrow \mathbb{R}$  $\approx$ Indefinite Integral $\int^{a} f(x)dx$

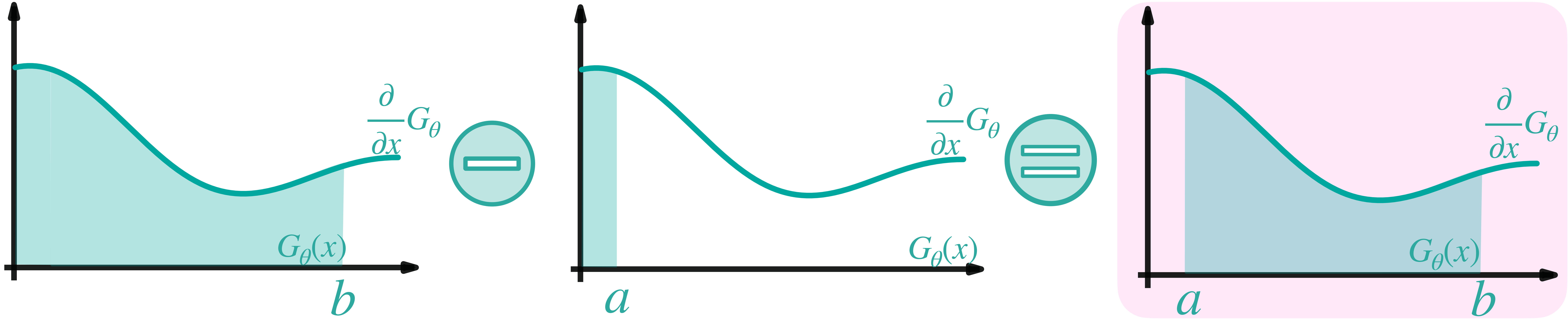


$$\frac{\partial}{\partial x} G_\theta \approx f(x)$$

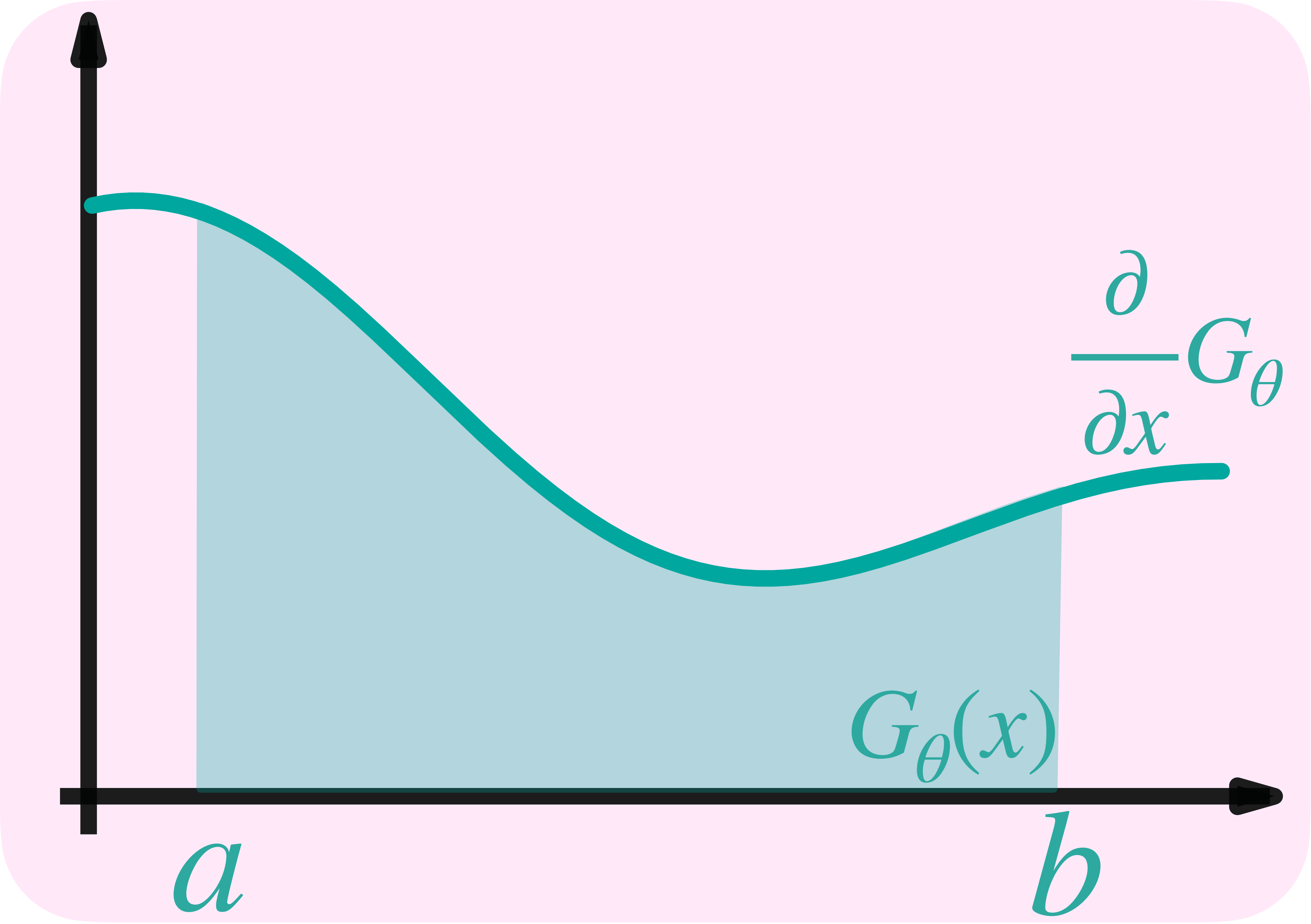
Approximating the Definite Integral



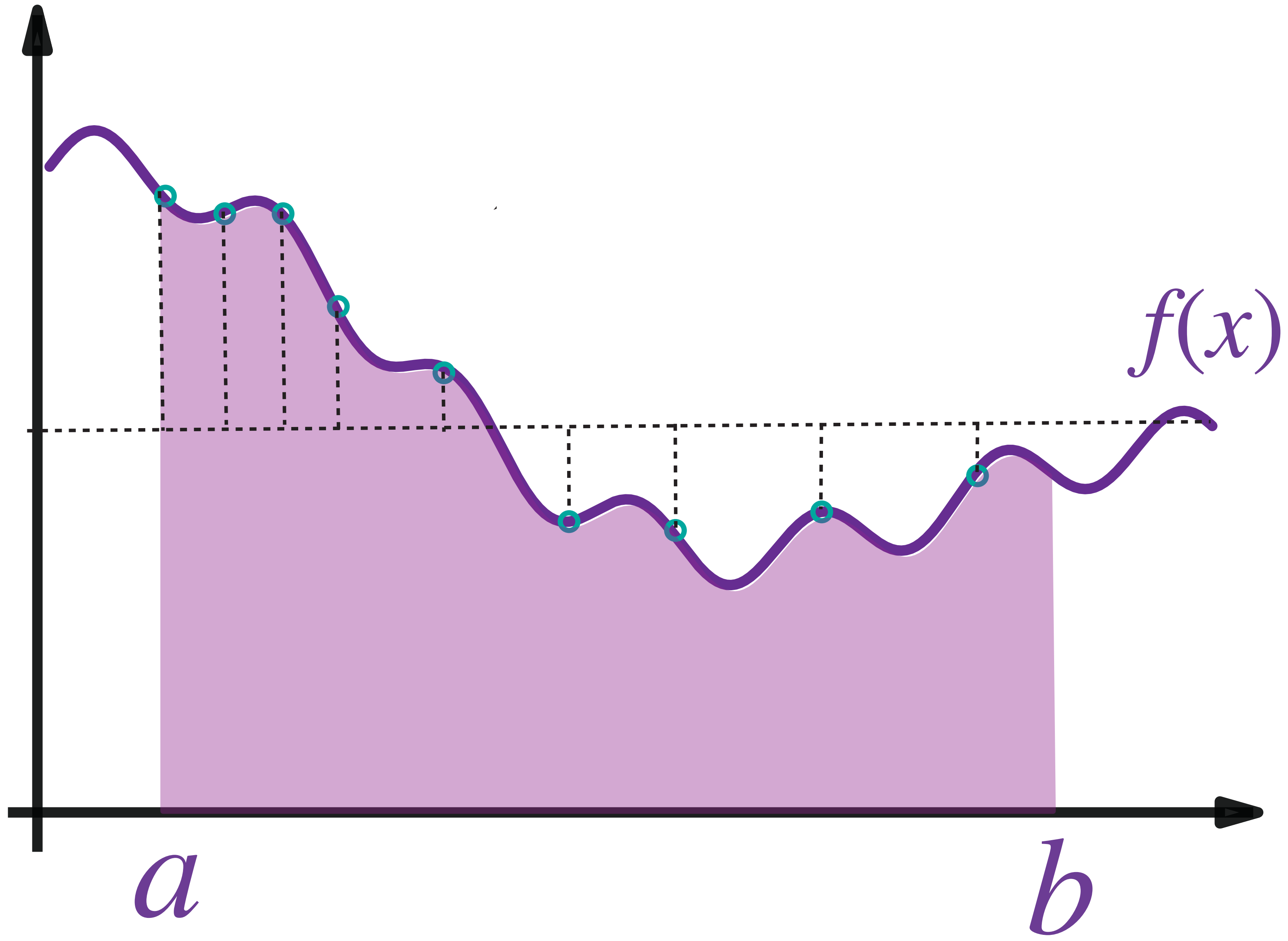
$$G_\theta(b) \quad - \quad G_\theta(a) \quad = \quad \int_a^b \frac{\partial}{\partial x} G_\theta(x)dx$$



Approximating the Definite Integral



$\approx$



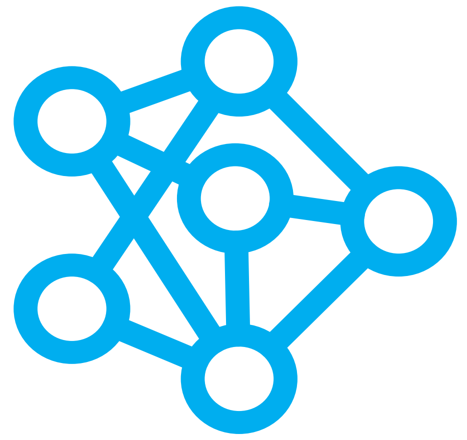
$$\int_a^b \frac{\partial}{\partial x}G_\theta(x)dx$$

$$\int_a^b f(x)dx$$

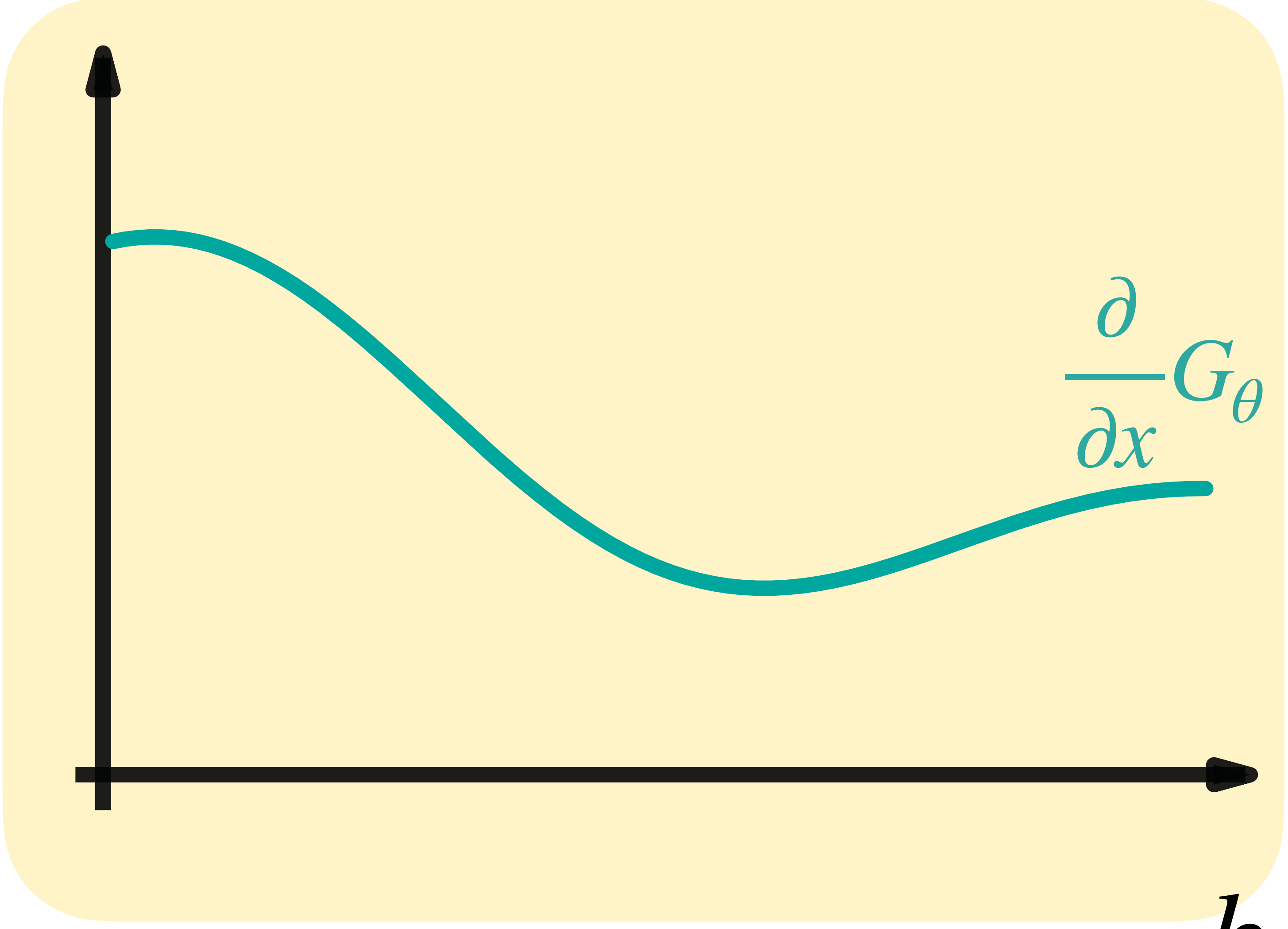
Computing the Integrand $g(x)$

$$u_\theta(x) = \int_{y \in \partial\Omega_r(x)} \frac{v_\theta(y)}{2\pi r} dy$$

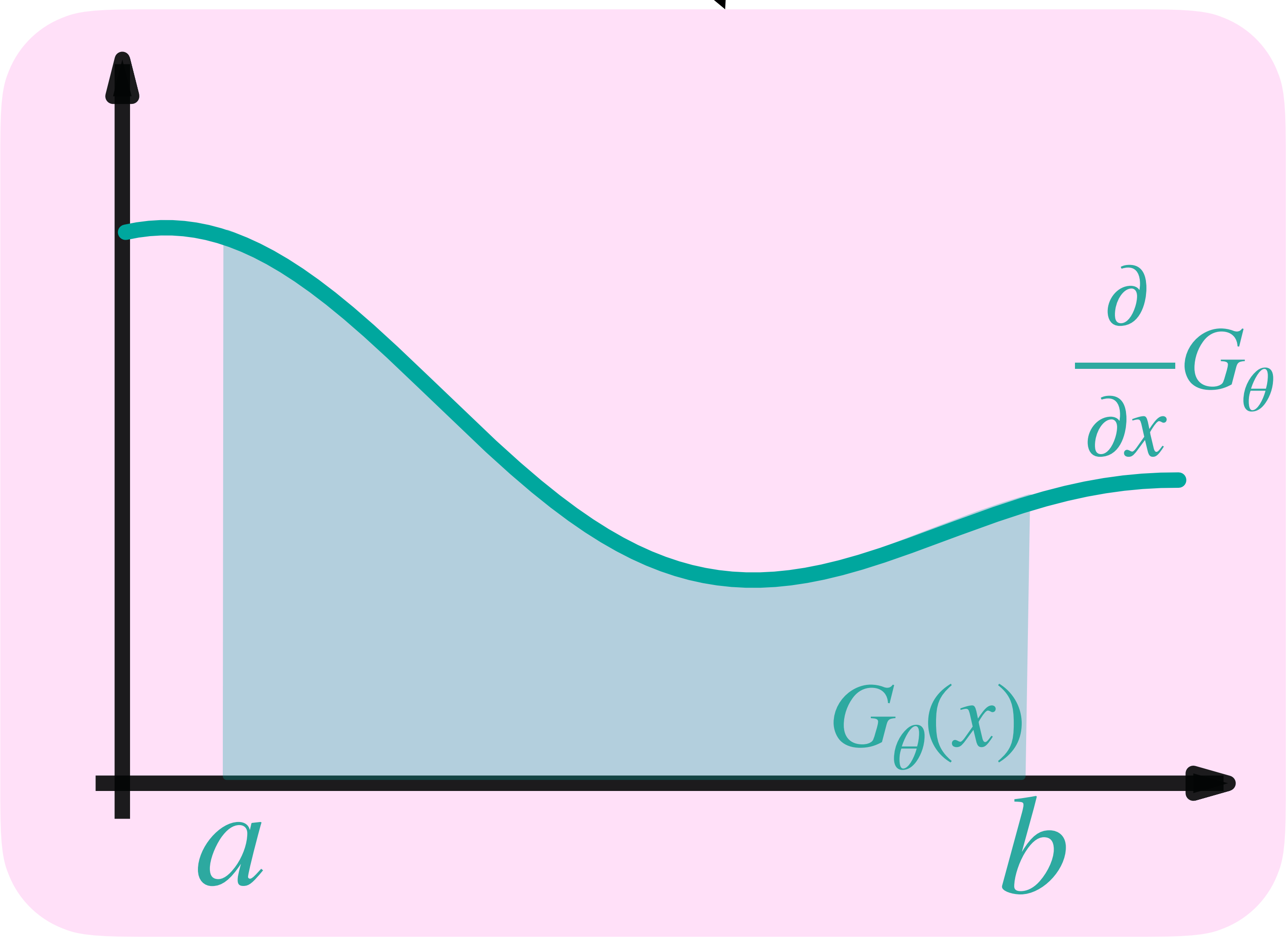
$G_\theta(x)$



$\partial/\partial x$



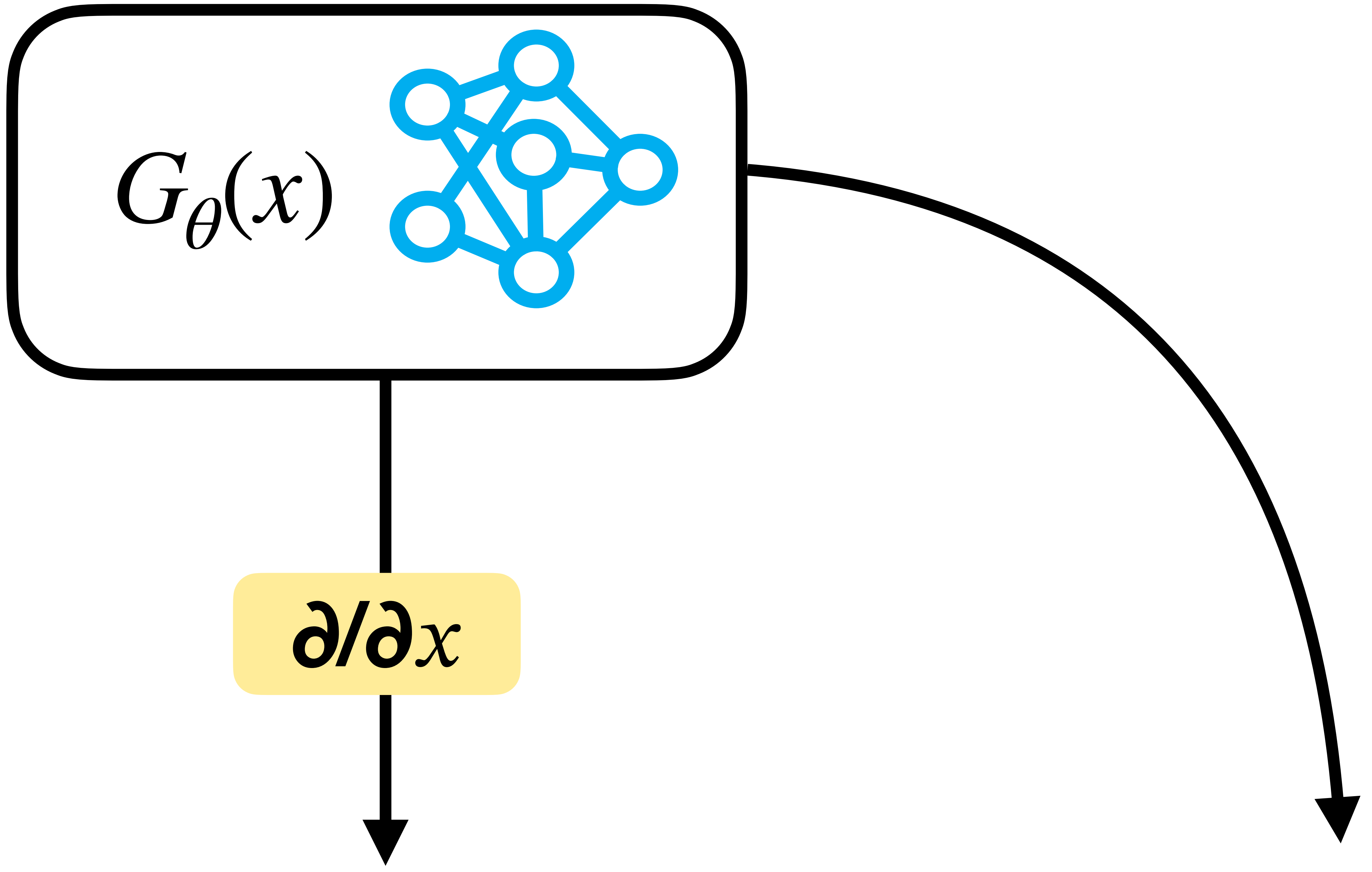
$$g_\theta(x) = \frac{\partial}{\partial x} G_\theta$$



$$\int_a^b \frac{\partial}{\partial x} G_\theta(x) dx = G_\theta(b) - G_\theta(a)$$

(LINDELL, D. et al., 2021)

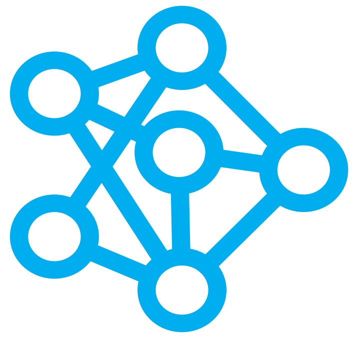
Constructing Control Variates



$$\int_a^b f(x)dx = \int_a^b \left(f(x) - \frac{\partial}{\partial x}G_\theta(x)\right)dx + G_\theta(b) - G_\theta(a)$$

Training

$$\mathbb{V}\left[u(y) - v_\theta(y)\right] <<< \mathbb{V}\left[u(y)\right]$$

$G_\theta(x)$ 

$\partial/\partial x$

$$\int_a^b f(x)dx = \int_a^b \left(f(x) - \frac{\partial}{\partial x}G_\theta(x)\right)dx + G_\theta(b) - G_\theta(a)$$

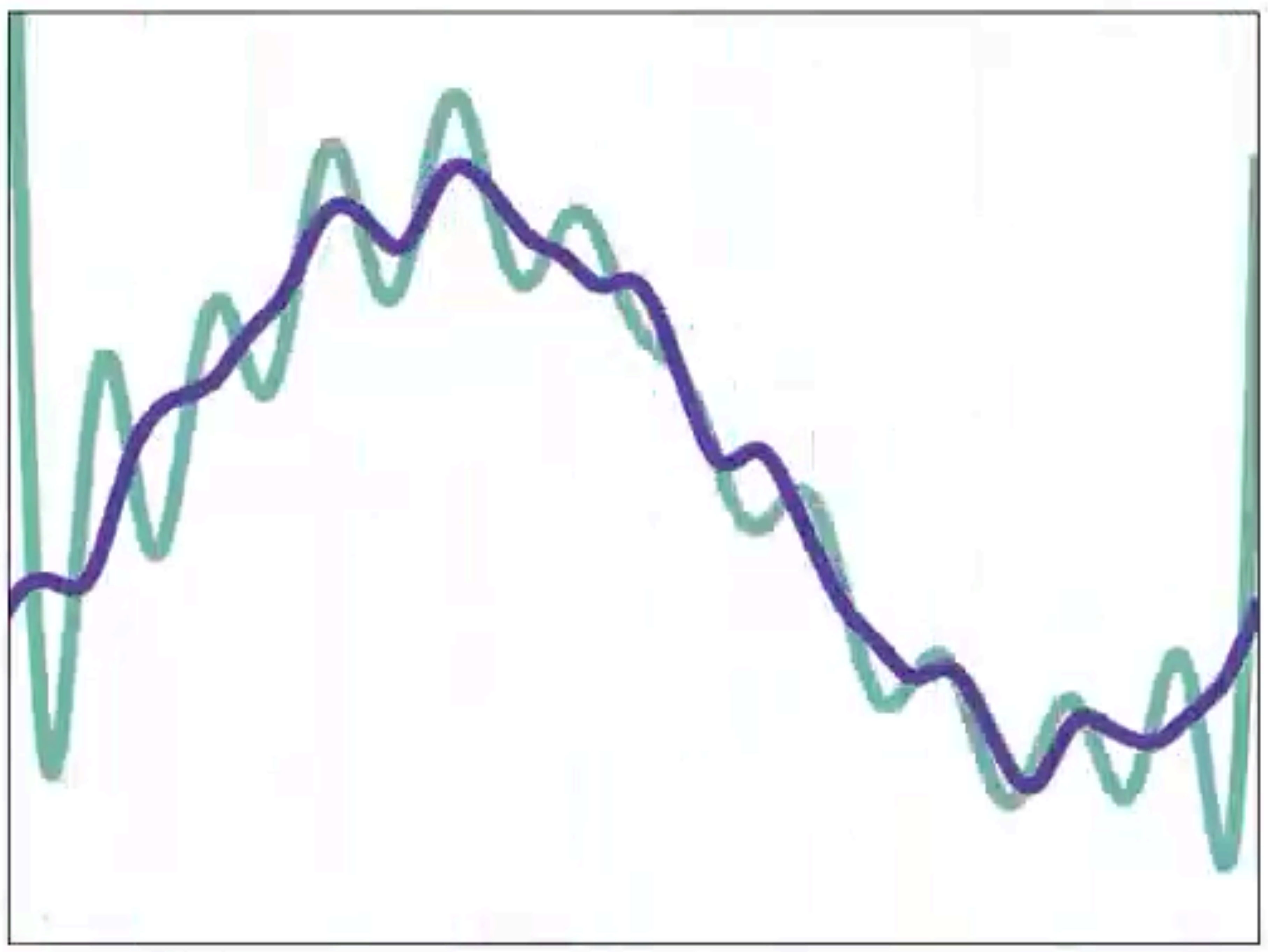
Objective: Minimizing the Variance

$$\mathbb{V} = \int_a^b \left(f(x) - \frac{\partial}{\partial x}G_\theta(x)\right)^2 dx - \left(G_\theta(a) - G_\theta(b) - \int_a^b f(x)dx\right)^2$$

Training the Derivative Network

Objective: Minimizing the Variance

—: $f(x)$ —: $\frac{\partial}{\partial x} G_\theta$

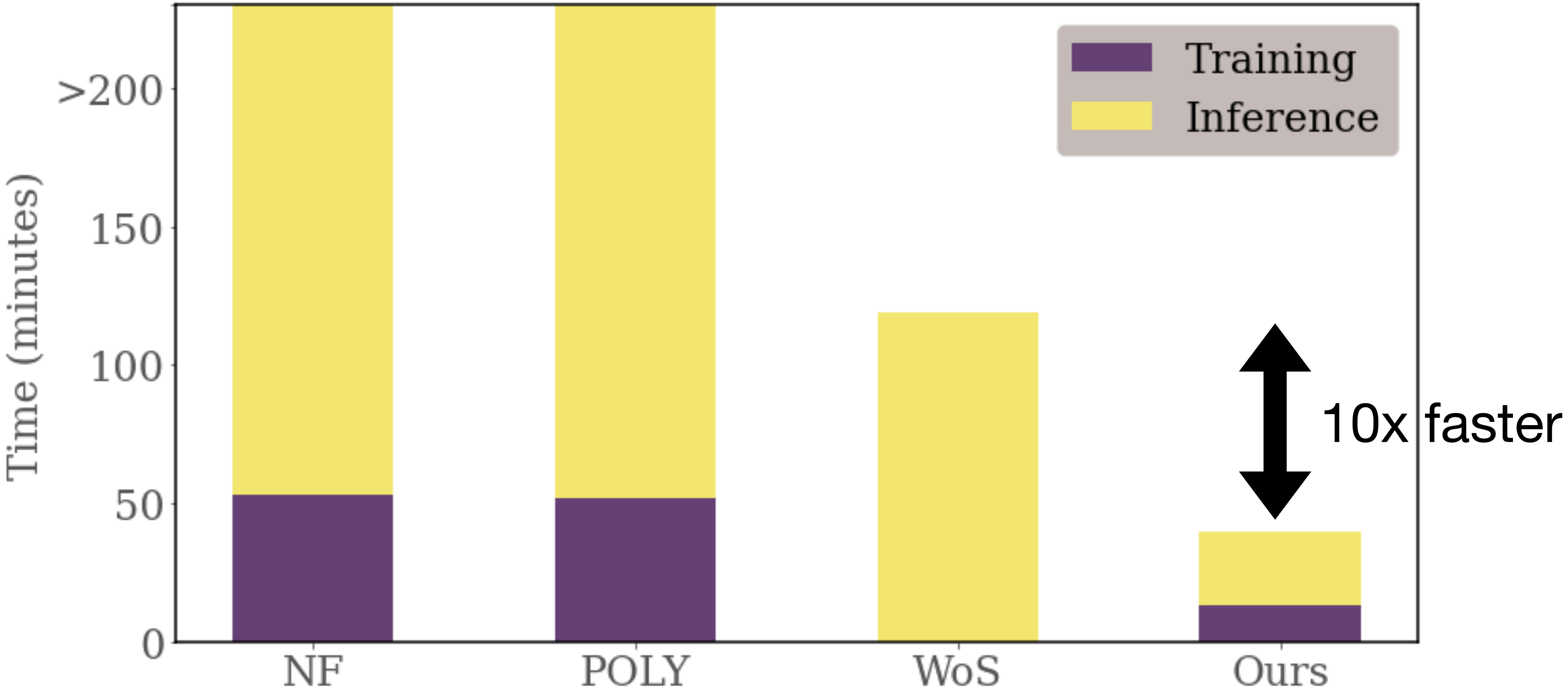


$Var(f-g)$


NF
Var 14.67 x
POLY
Var 1.24 x
WoS
Var 1.00 x
Ours
Var 0.30 x

Our estimator is faster for high resolution

Computational Time Breakdown to Create a 1024 Resolution Image

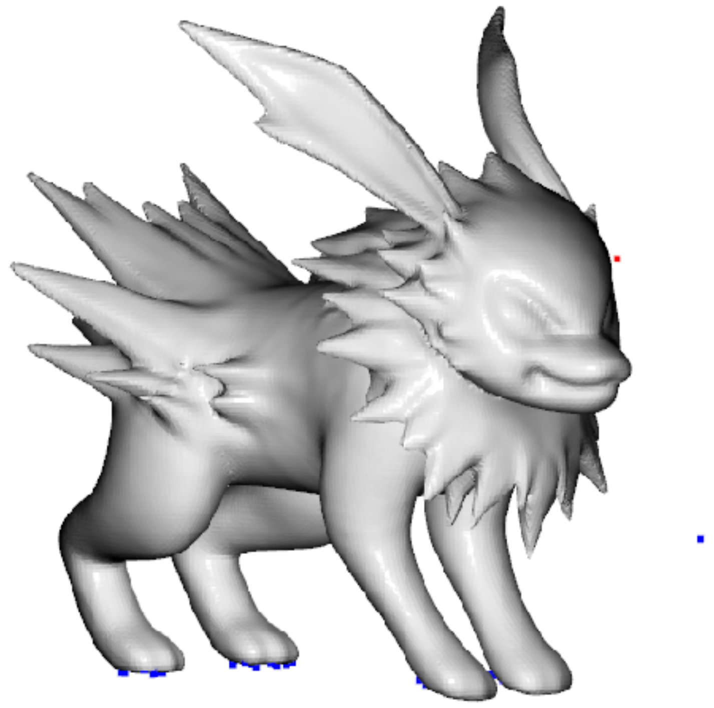


Analysis - Symmetry Detection

Synthesis

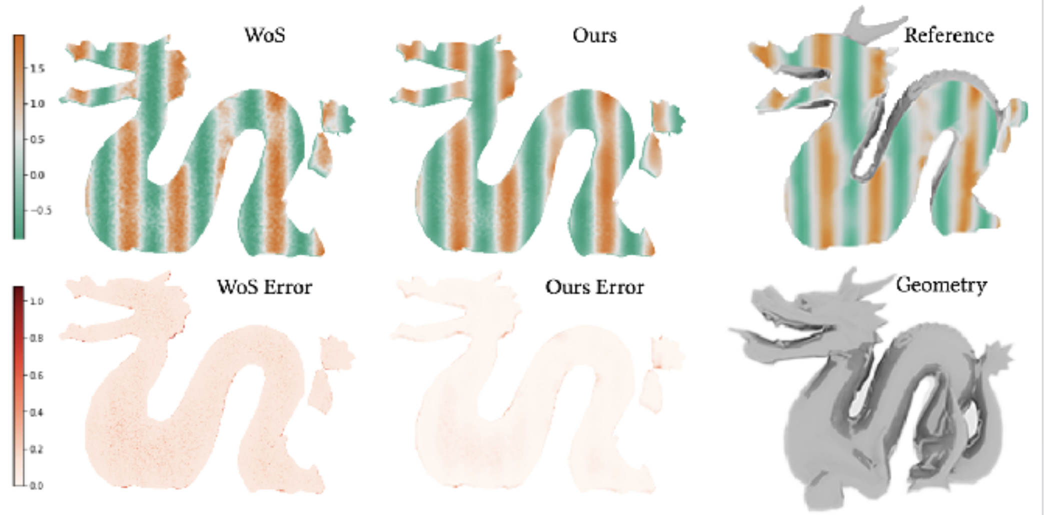
PointFlow (ICCV 2019)

ShapeGF (ECCV 2020)

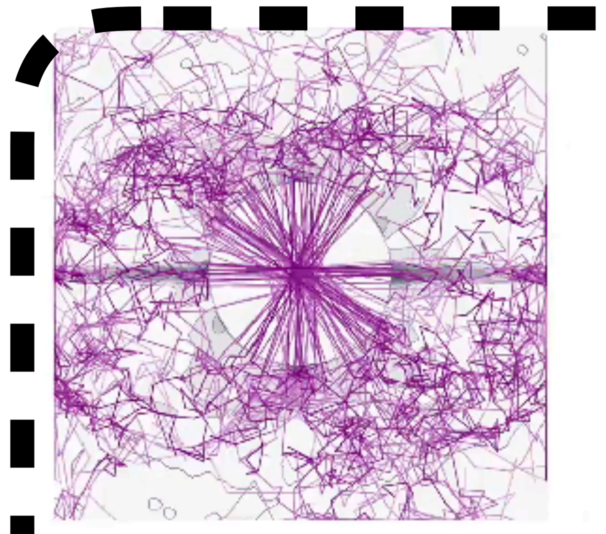


NFGP (NeurIPS 2021)

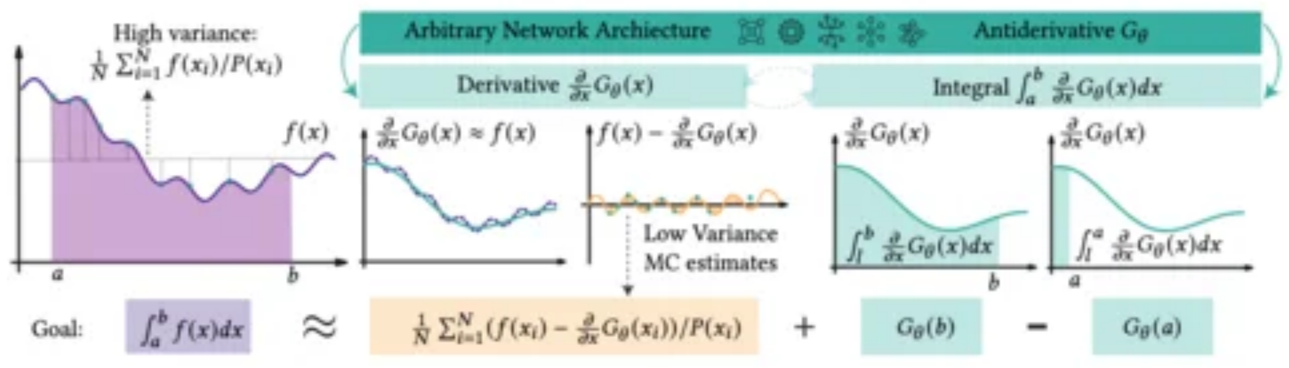
Analysis



Neural cache (SIG Asia 2023)



Symmetry (SIG Asia 2024)

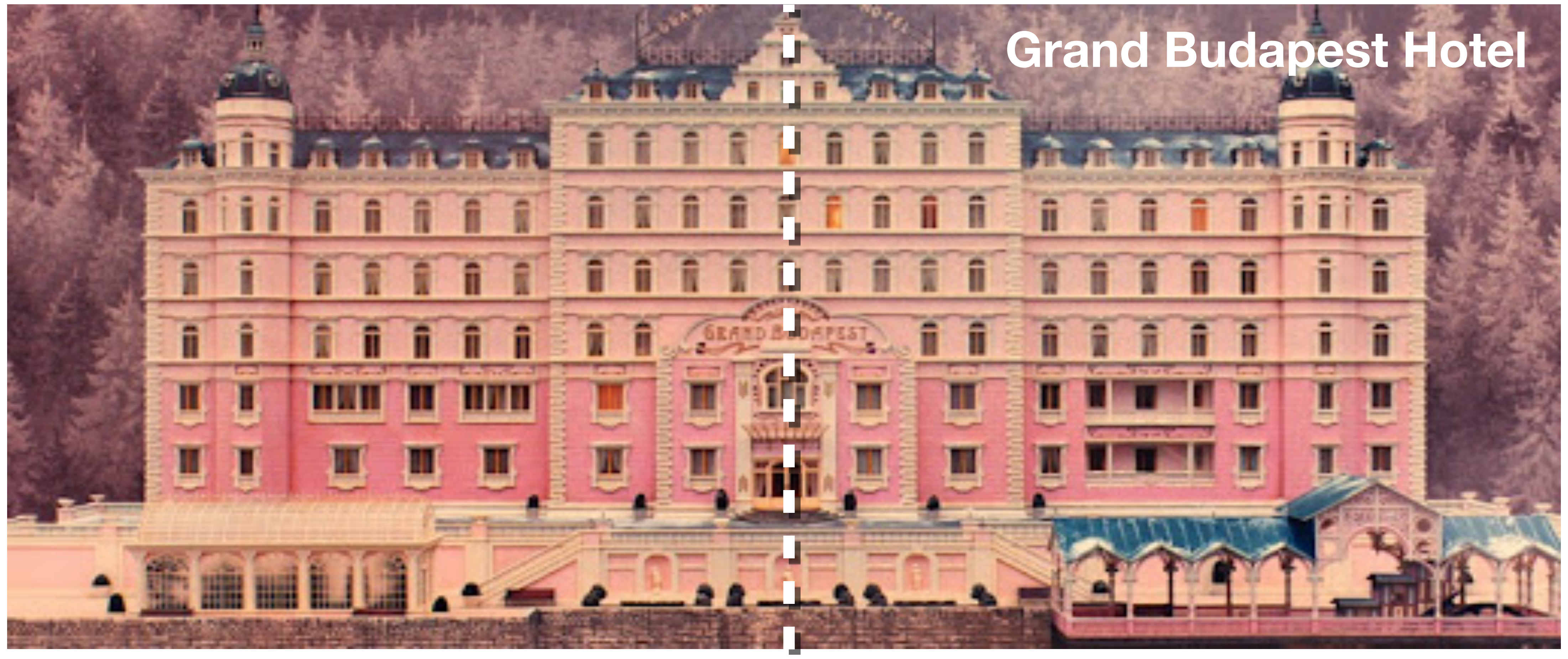


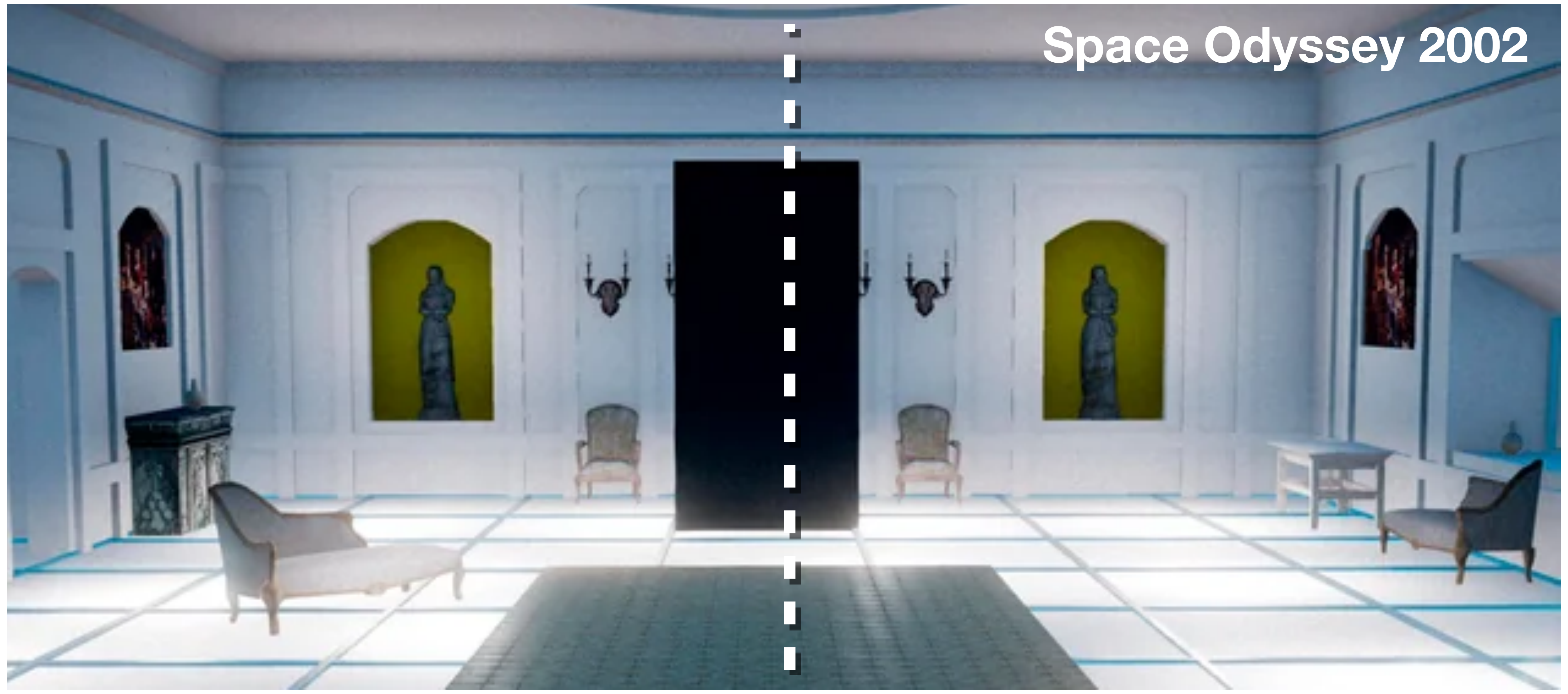
NCV (SIGRAPH 2024)

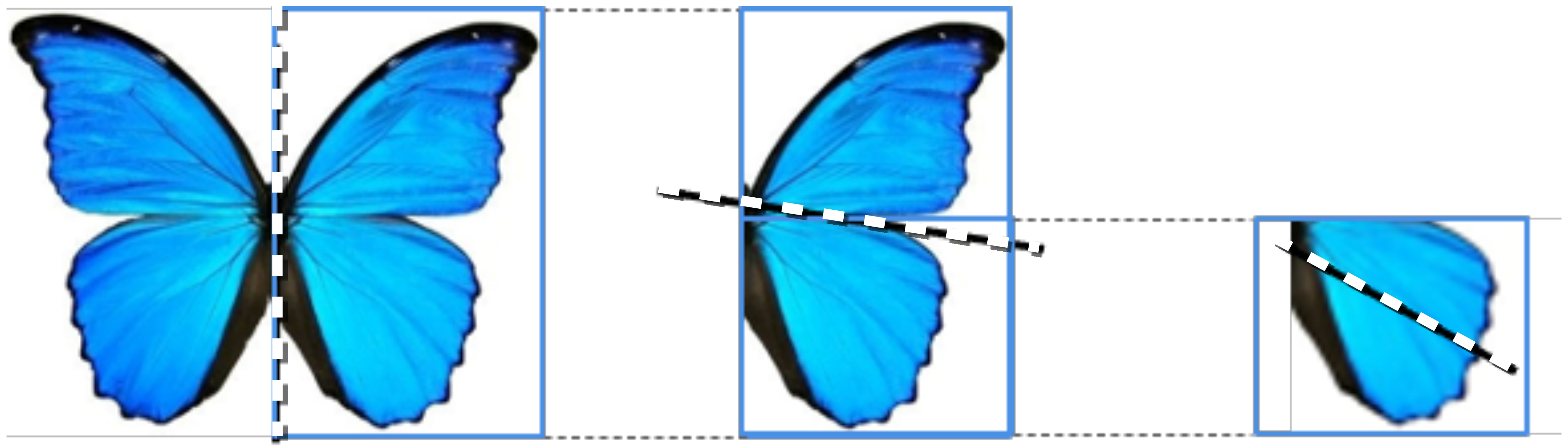
2019 2020 2021 2023 2024

Symmetry is ubiquitous

How do we detect symmetry of a shape?

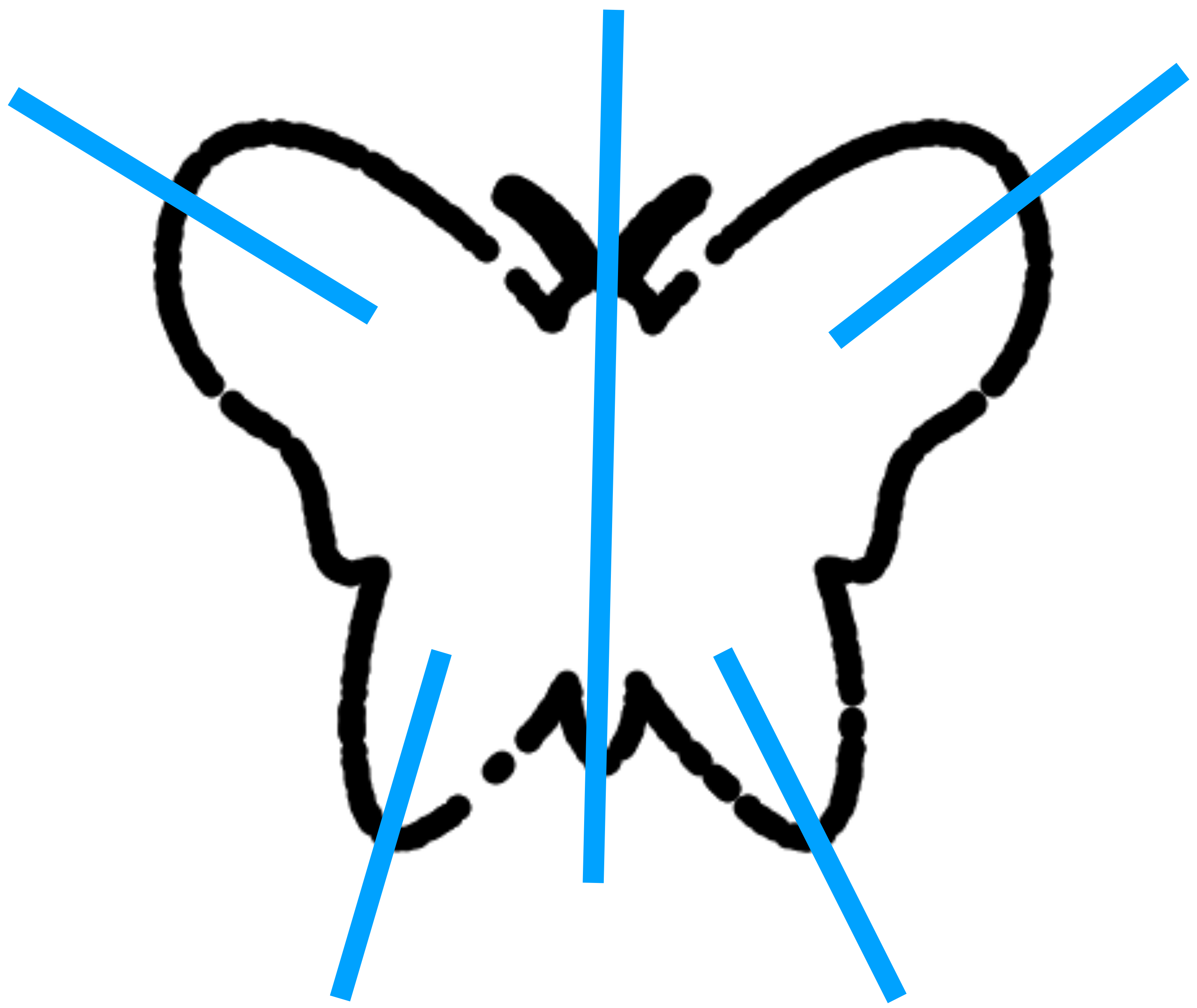






How do we detect symmetry of a shape?

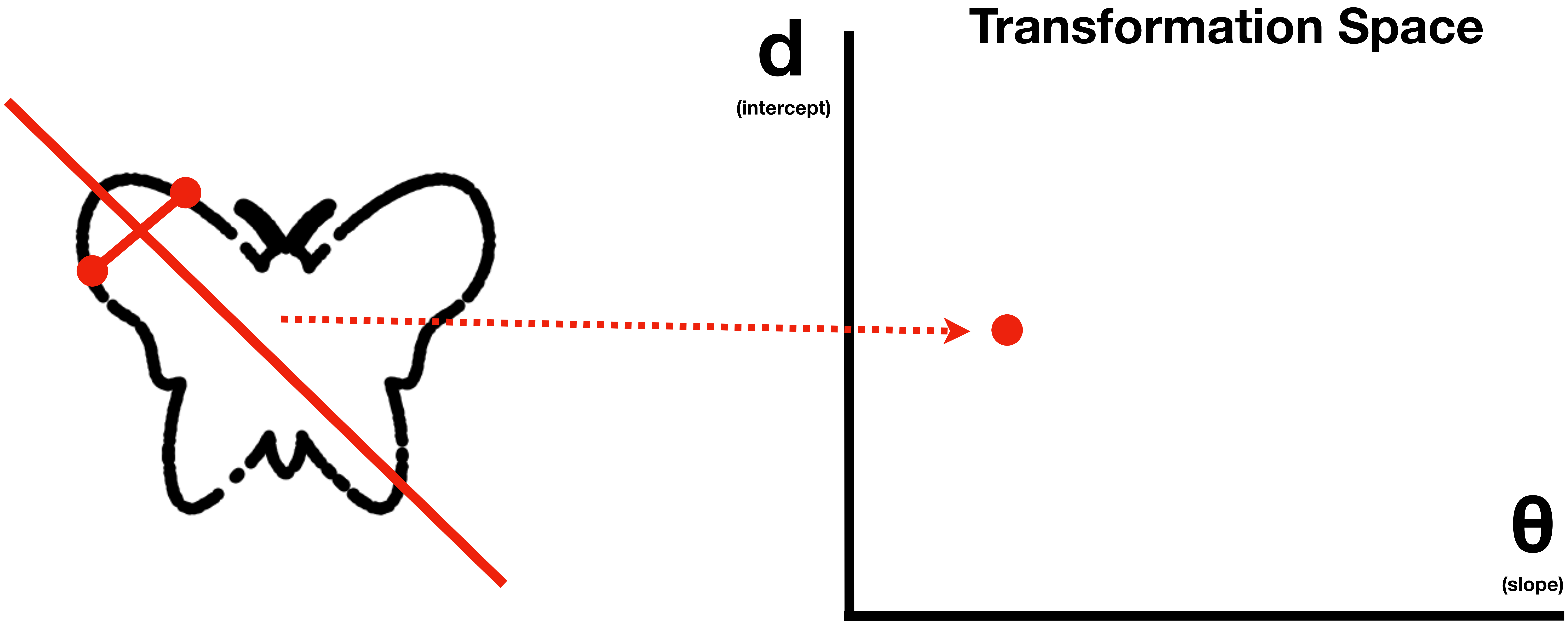
IDEA: Voting



(Mitra et al. 2006)

How do we detect symmetry of a shape?

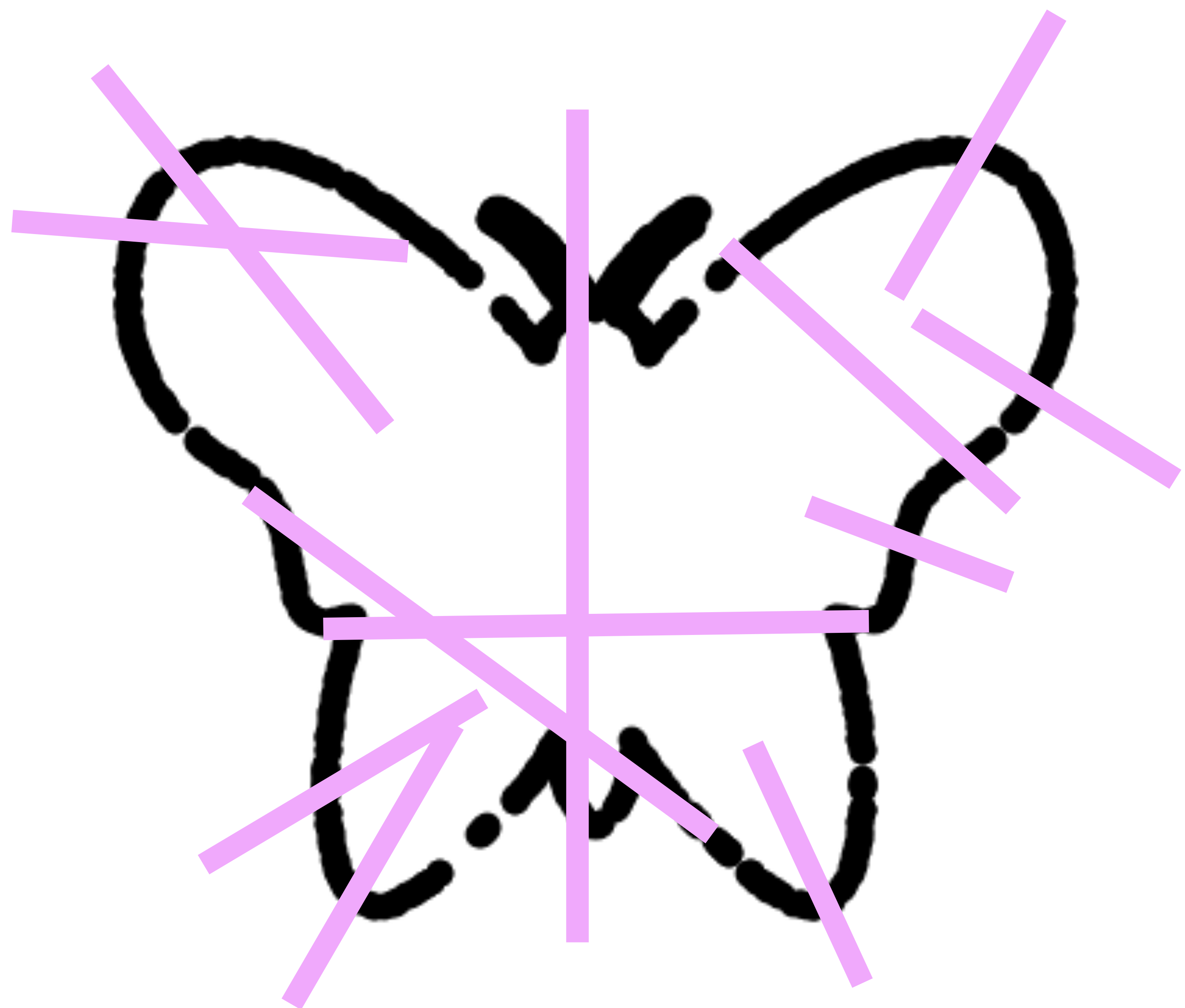
IDEA: Voting

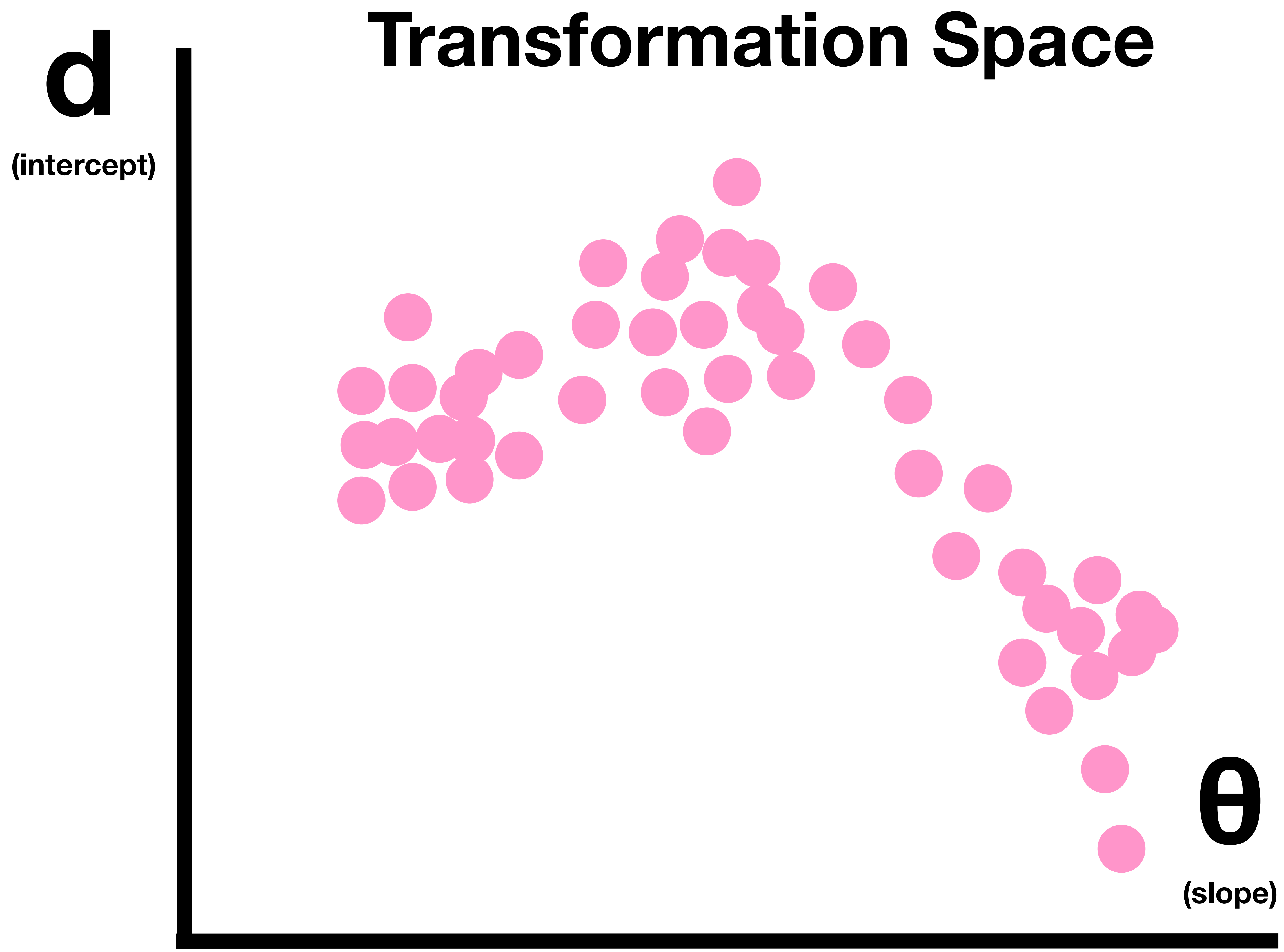


(Mitra et al. 2006)

How do we detect symmetry of a shape?

IDEA: Voting

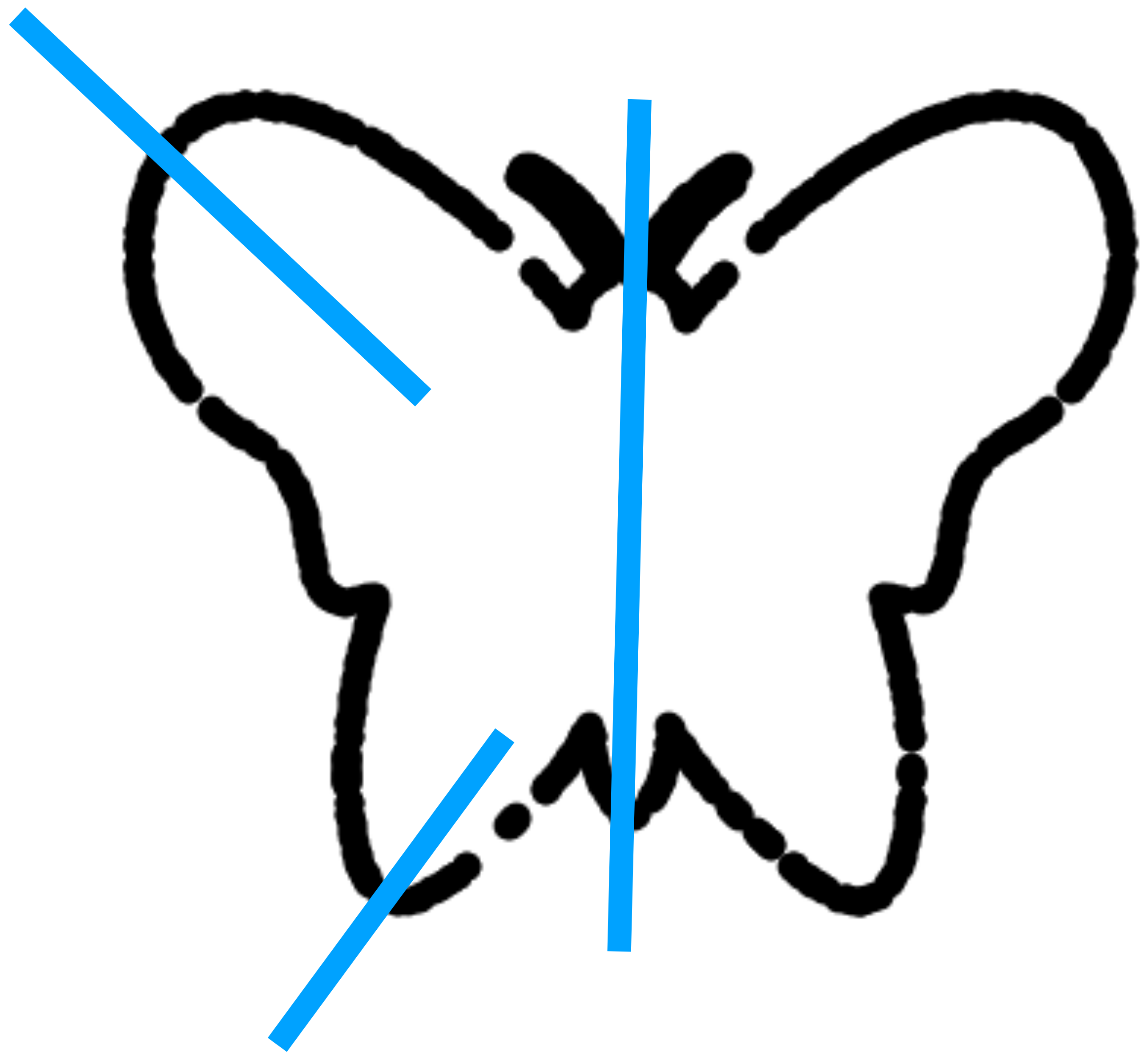


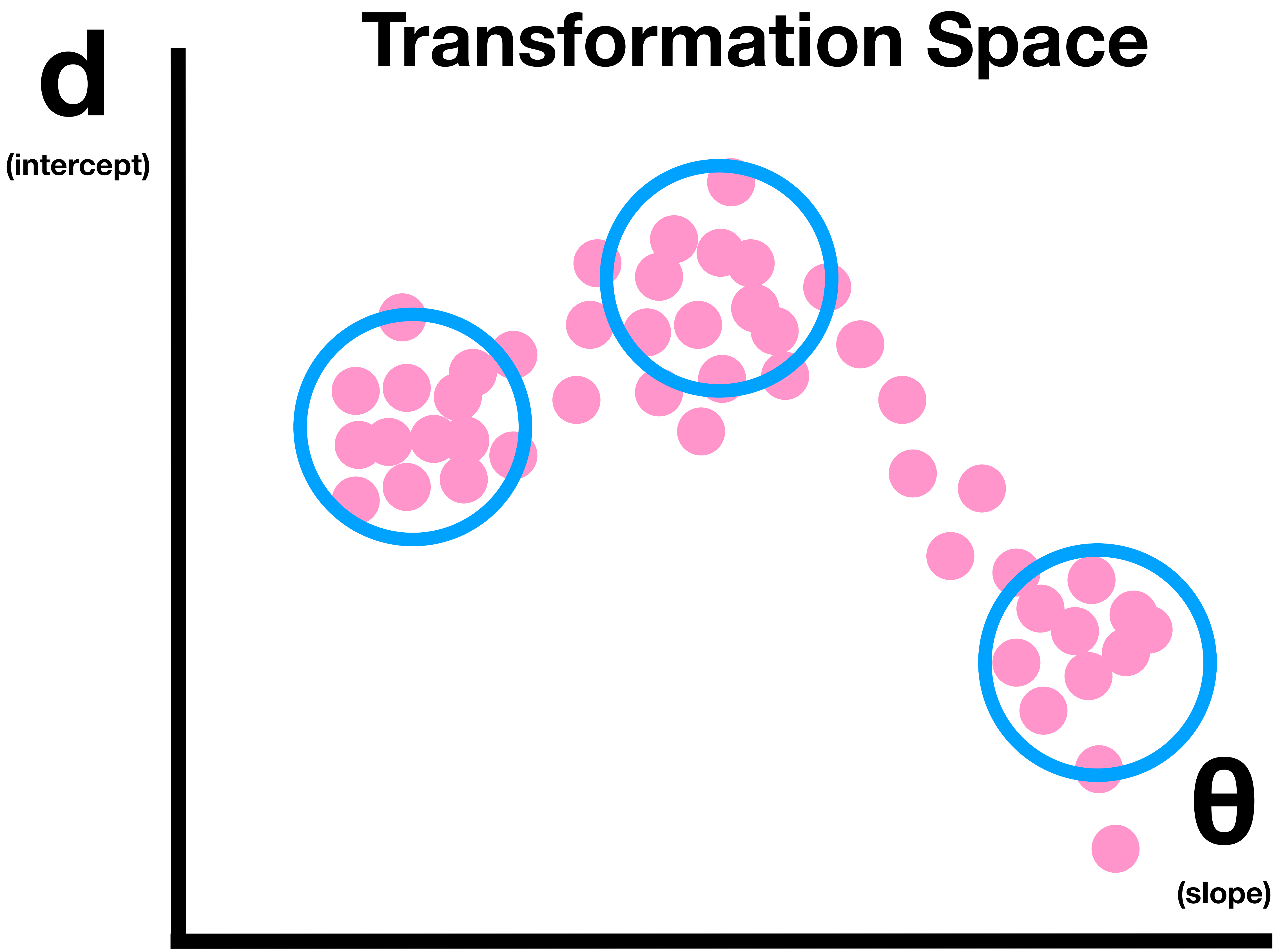


(Mitra et al. 2006)

How do we detect symmetry of a shape?

IDEA: Voting

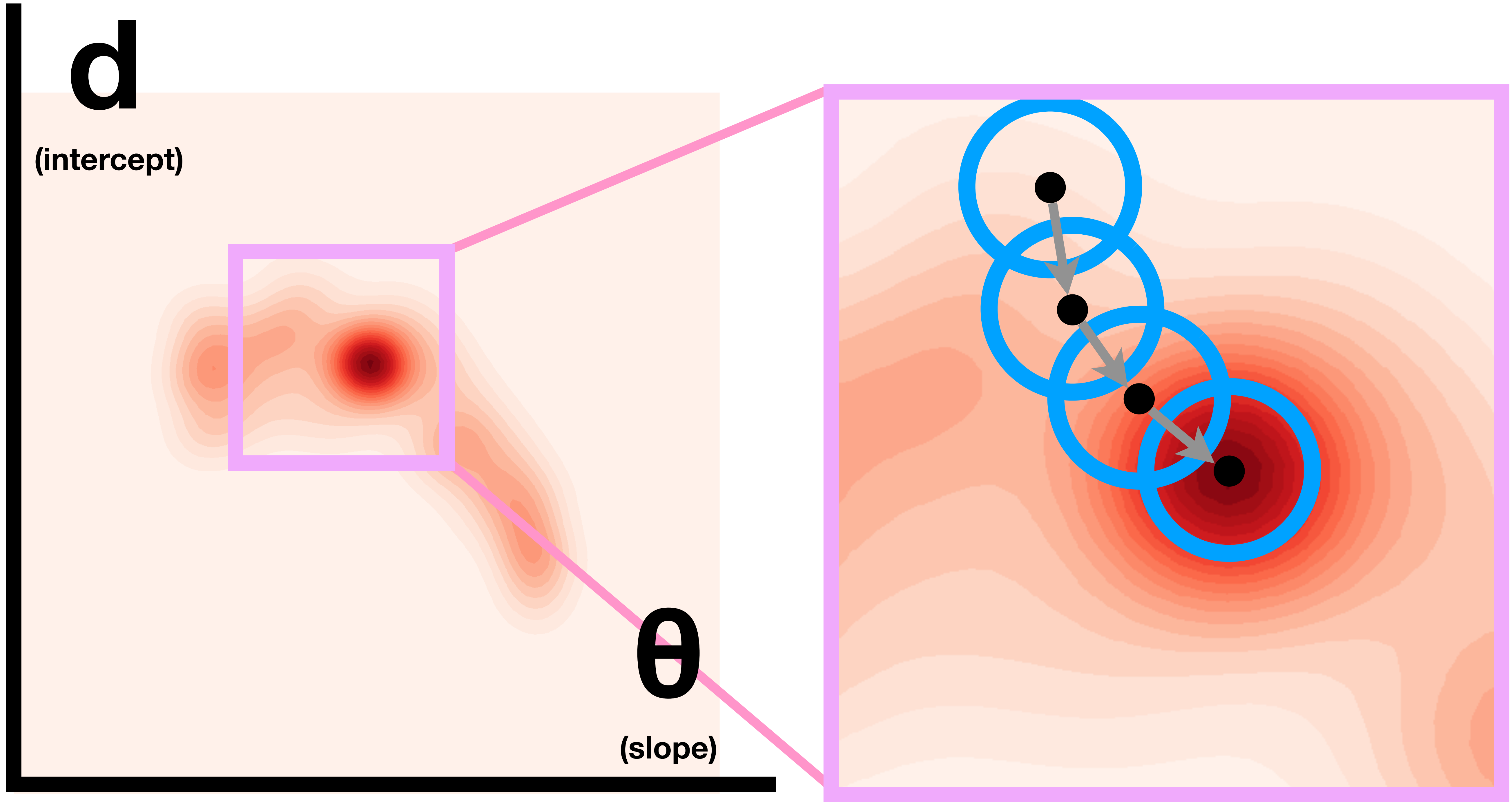




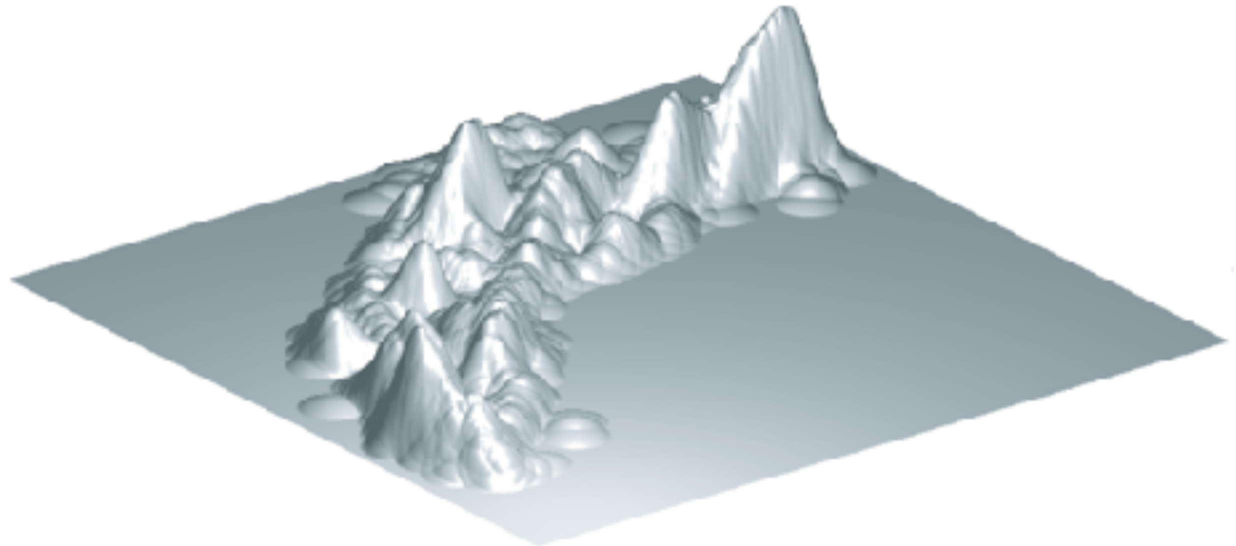
(Mitra et al. 2006)

How do we detect symmetry of a shape?

Prior works - mean shift to seek mode



$$p(T) = \frac{1}{|N(T)|h^d} \sum_{T_i \in N(T)}^{n} K\left(\frac{T - T_i}{h}\right)$$

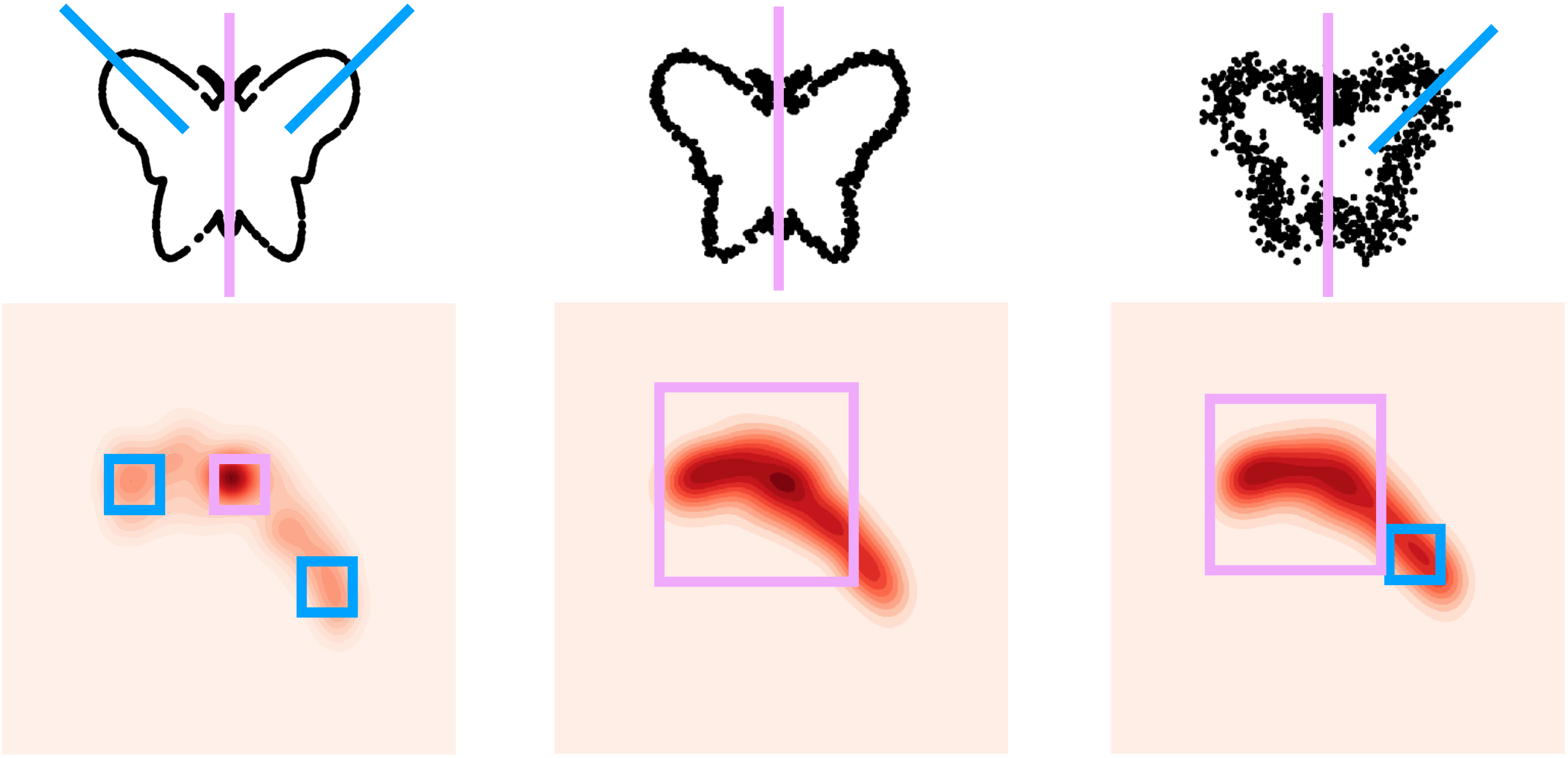


$$T^{(k+1)} \leftarrow \frac{\sum_{T' \in N_k} K((T' - T^{(k)})h^{-1})T'}{\sum_{T' \in N_k} K((T' - T^{(k)})h^{-1})}$$

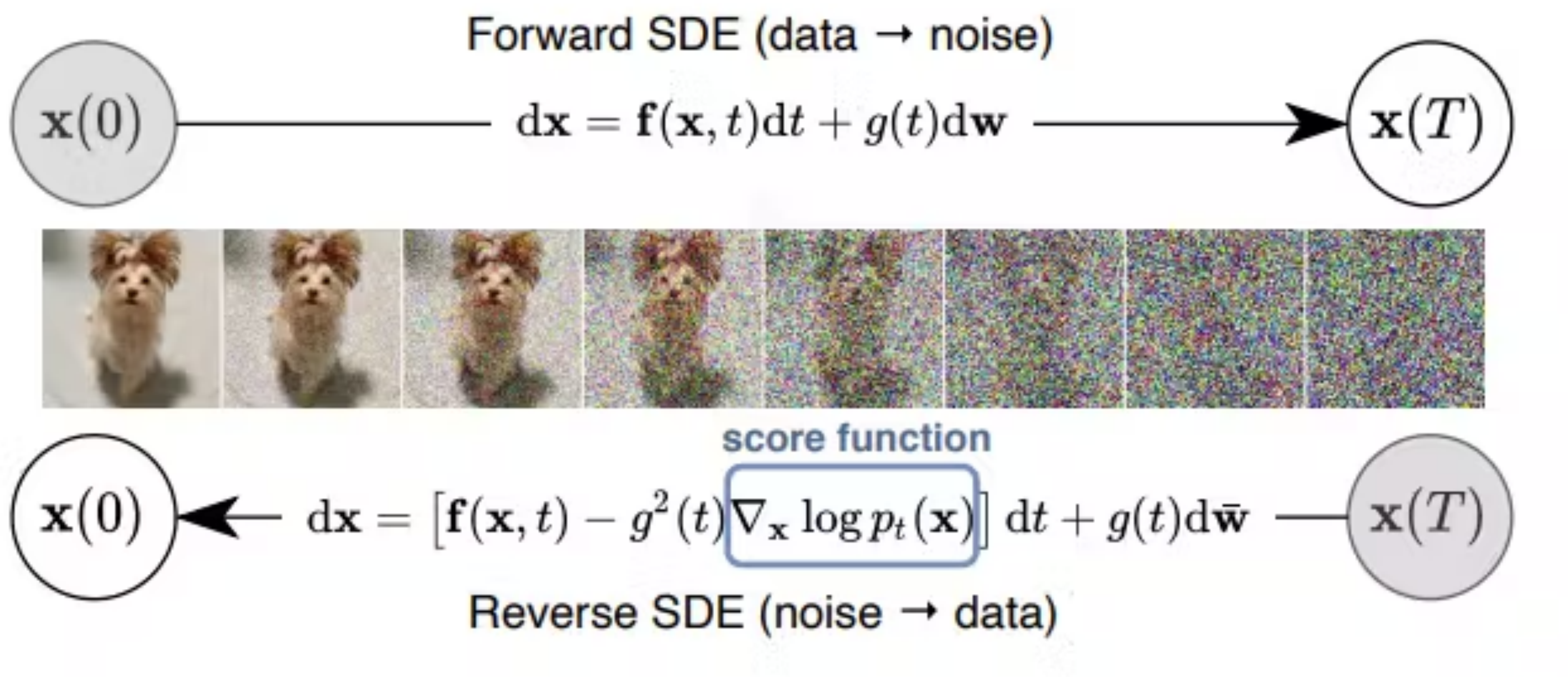
(Mitra et al. 2006)

How do we detect symmetry?

Prior works limitation: unable to handle noisy shape

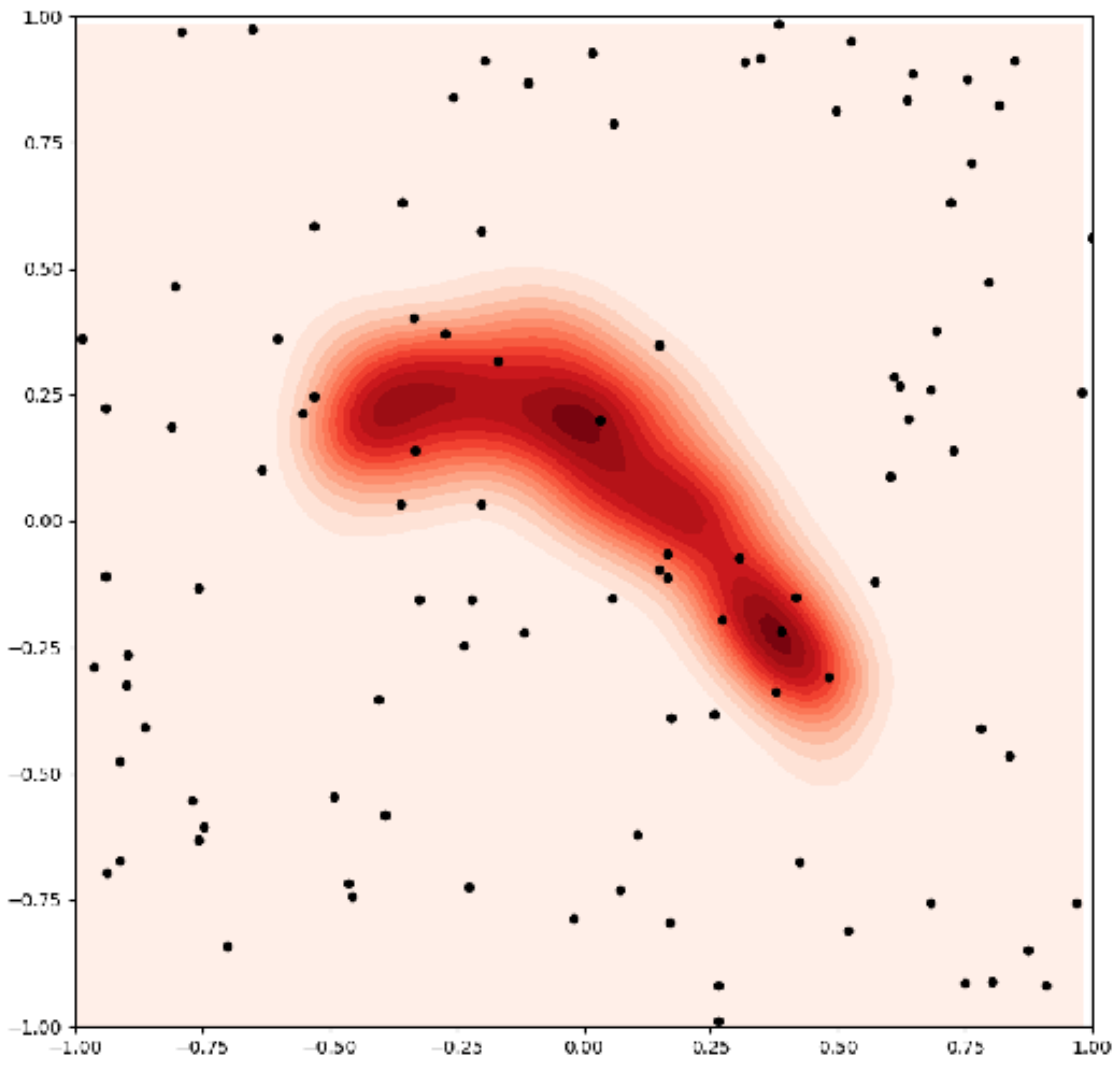


Other mode-seeking algorithms?



(Song et al., 2020)

Langevin



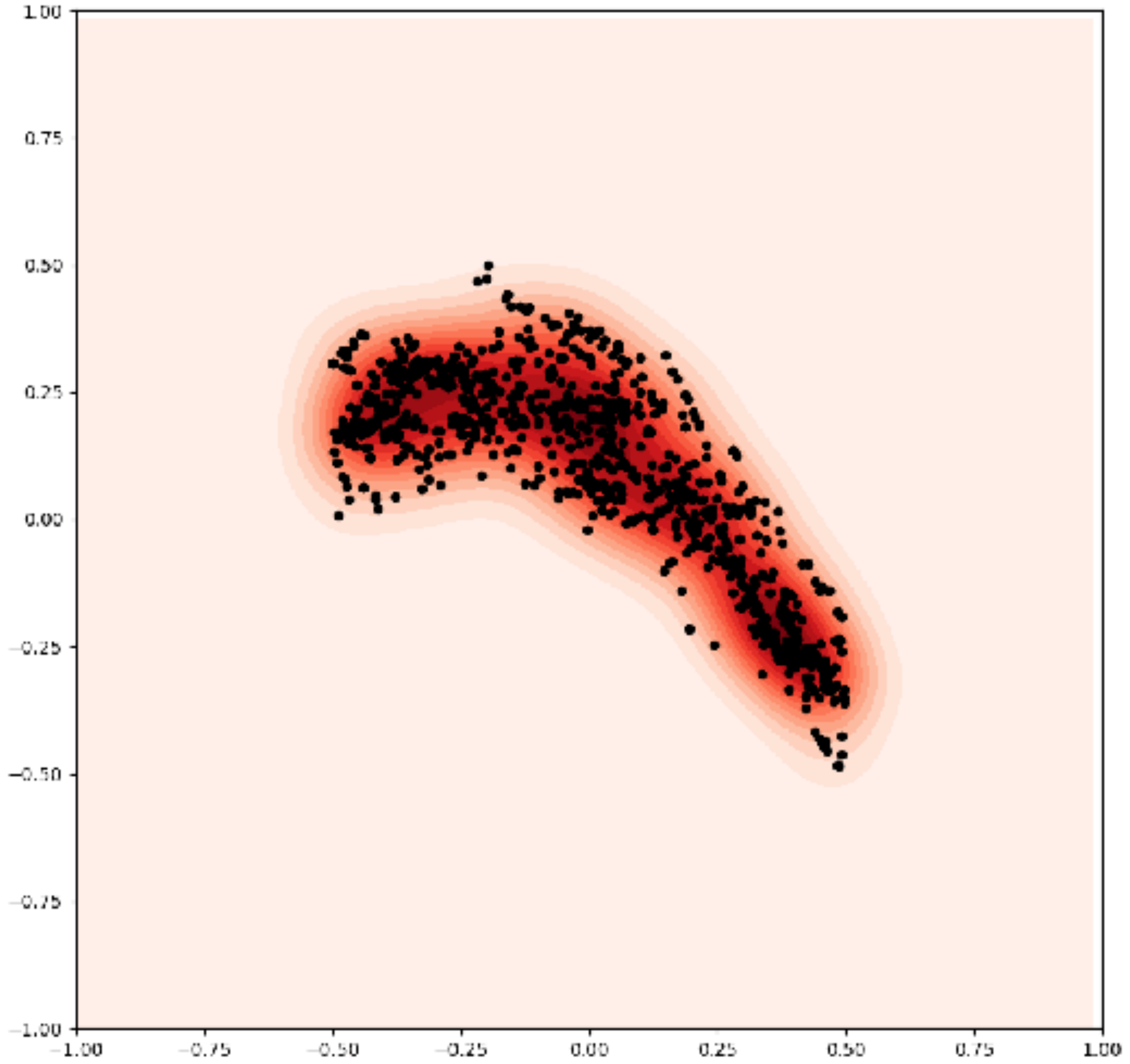
$$P_\sigma(x) = \int p_{\text{data}}(y)\mathcal{N}(x; y, \sigma^2 I)dy \approx \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|\mathcal{X}|}\mathcal{N}(x; x_i, \sigma^2 I)$$

$$\nabla_x \log P_\sigma(x) \approx \left(\frac{\sum_{y\in\mathcal{X}}\mathcal{N}(x; y, \sigma^2 I)\cdot y}{\sum_{y\in\mathcal{X}}\mathcal{N}(x; y, \sigma^2 I)} - x\right)\sigma^{-2}$$

$$x^{(t+1)} \leftarrow x^{(t)} + \alpha_t \nabla_x \log P_{\sigma_t}(x^{(t)}) + \sqrt{2\alpha_t}\beta_t \epsilon_t$$

Mean shift is a form of Langevin without Stochasticity

Meanshift



$$p(T) = \frac{1}{|N(T)|h^d} \sum_{T_i \in N(T)}^{n} K\left(\frac{T - T_i}{h}\right)$$

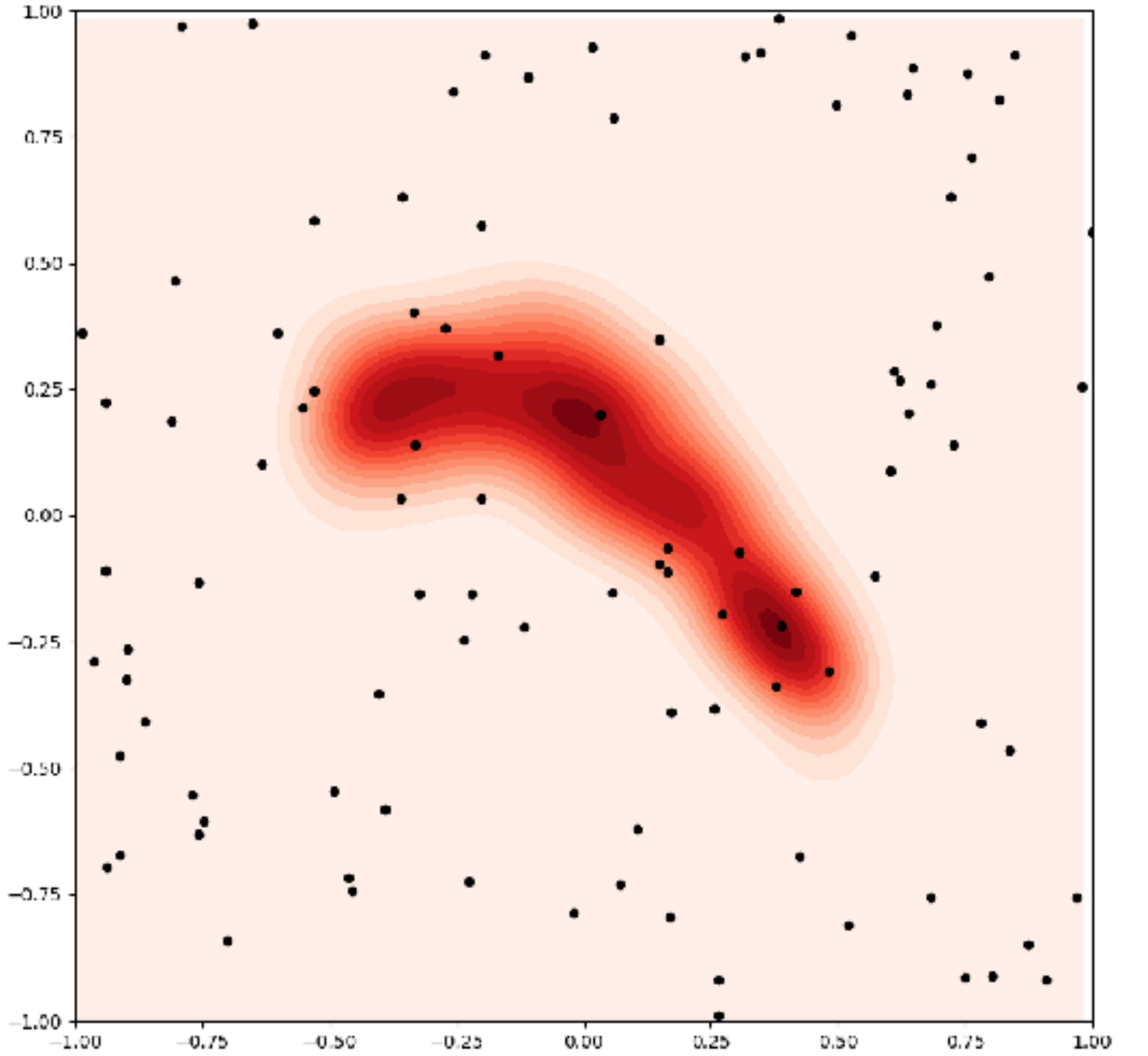
$$T^{(k+1)} \leftarrow \frac{\sum_{T' \in N_k} K((T' - T^{(k)})h^{-1})T'}{\sum_{T' \in N_k} K((T' - T^{(k)})h^{-1})}$$

$$P_\sigma(x) = \int P_{\text{data}}(y)\mathcal{N}(x; y, \sigma^2 I)dy \approx \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|\mathcal{X}|}\mathcal{N}(x; x_i, \sigma^2 I)$$

$$\nabla_x \log P_\sigma(x) \approx \left(\frac{\sum_{y \in \mathcal{X}} \mathcal{N}(x; y, \sigma^2 I) \cdot y}{\sum_{y \in \mathcal{X}} \mathcal{N}(x; y, \sigma^2 I)} - x\right)\sigma^{-2}$$

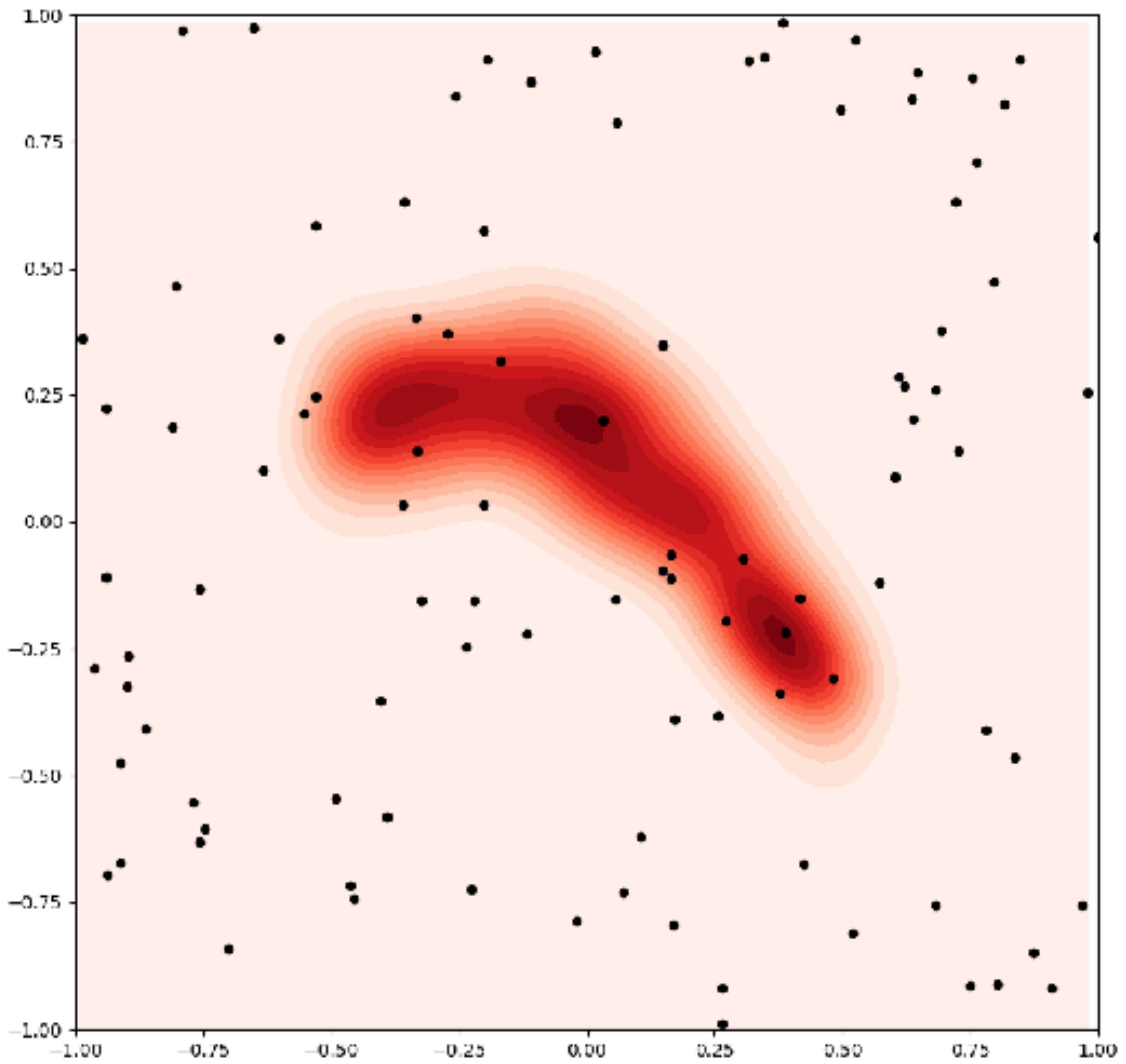
$$x^{(t+1)} \leftarrow x^{(t)} + \alpha_t \nabla_x \log P_{\sigma_t}(x^{(t)}) + \cancel{\sqrt{2\alpha_t}\beta_t \epsilon_t}$$

Langevin



Our method: Langevin with Stochasticity

Langevin

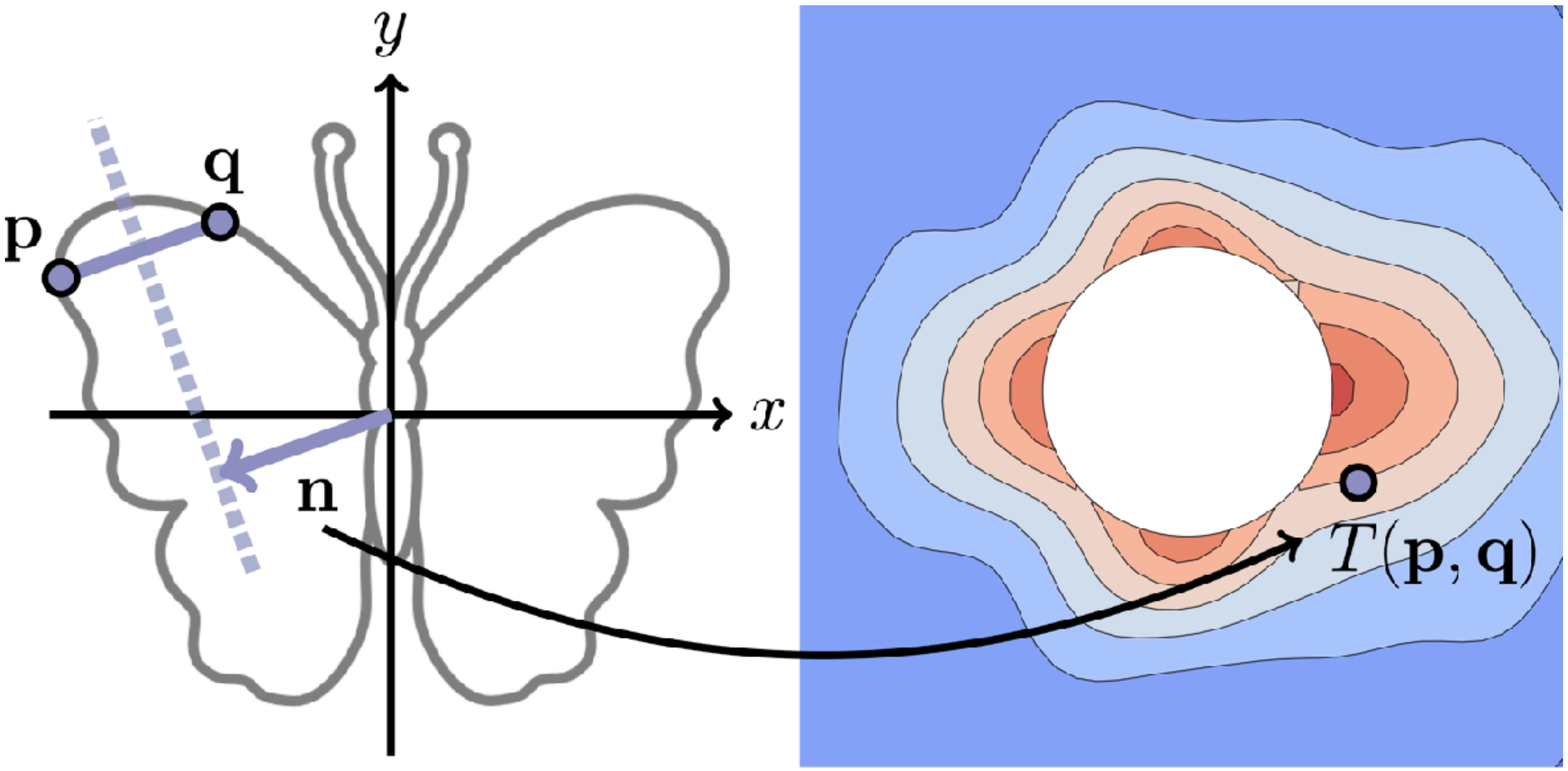


$$P_\sigma(x) = \int P_{\text{data}}(y)\mathcal{N}(x; y, \sigma^2 I)dy \approx \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|\mathcal{X}|}\mathcal{N}(x; x_i, \sigma^2 I)$$

$$\nabla_x \log P_\sigma(x) \approx \left(\frac{\sum_{y\in\mathcal{X}}\mathcal{N}(x; y, \sigma^2 I)\cdot y}{\sum_{y\in\mathcal{X}}\mathcal{N}(x; y, \sigma^2 I)} - x\right)\sigma^{-2}$$

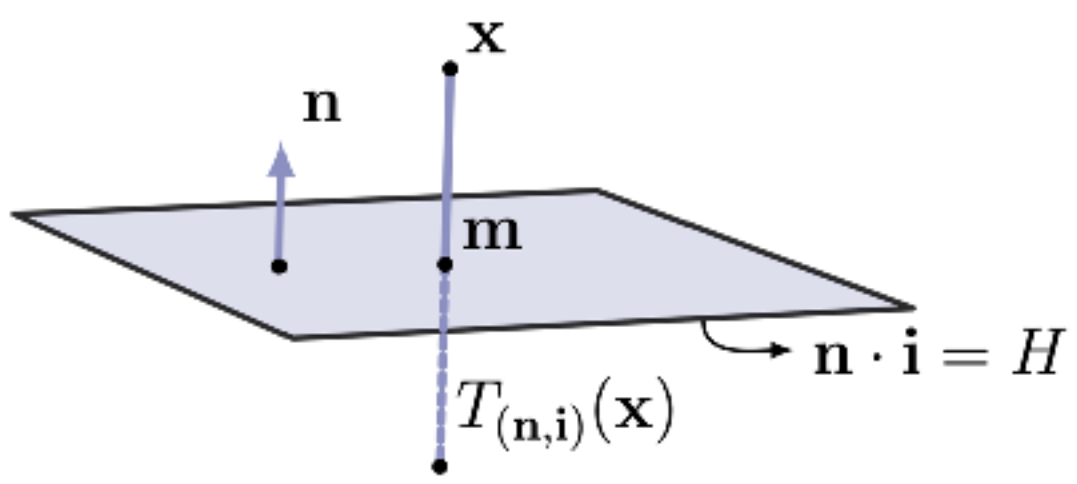
$$x^{(t+1)} \leftarrow x^{(t)} + \alpha_t \nabla_x \log P_{\sigma_t}(x^{(t)}) + \sqrt{2\alpha_t}\beta_t\epsilon_t$$

Step 1: Create Transformation Space



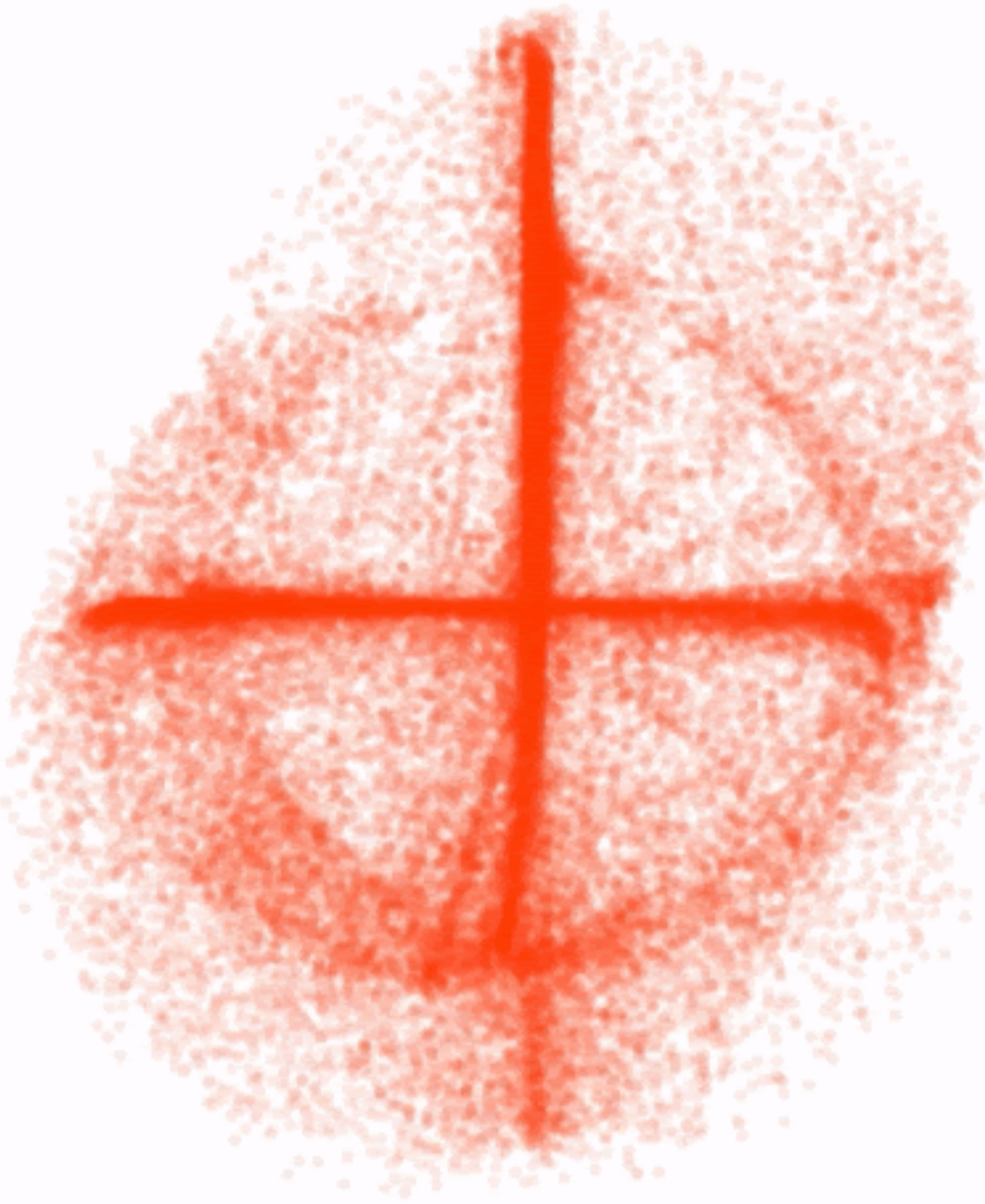
$$T(\mathbf{p}, \mathbf{q}) = n(\mathbf{p}, \mathbf{q}) \cdot (\mathrm{sign}(l(\mathbf{p}, \mathbf{q})) \cdot k + l(\mathbf{p}, \mathbf{q}))$$

Transformation Space in 3D



Raw 3D shape

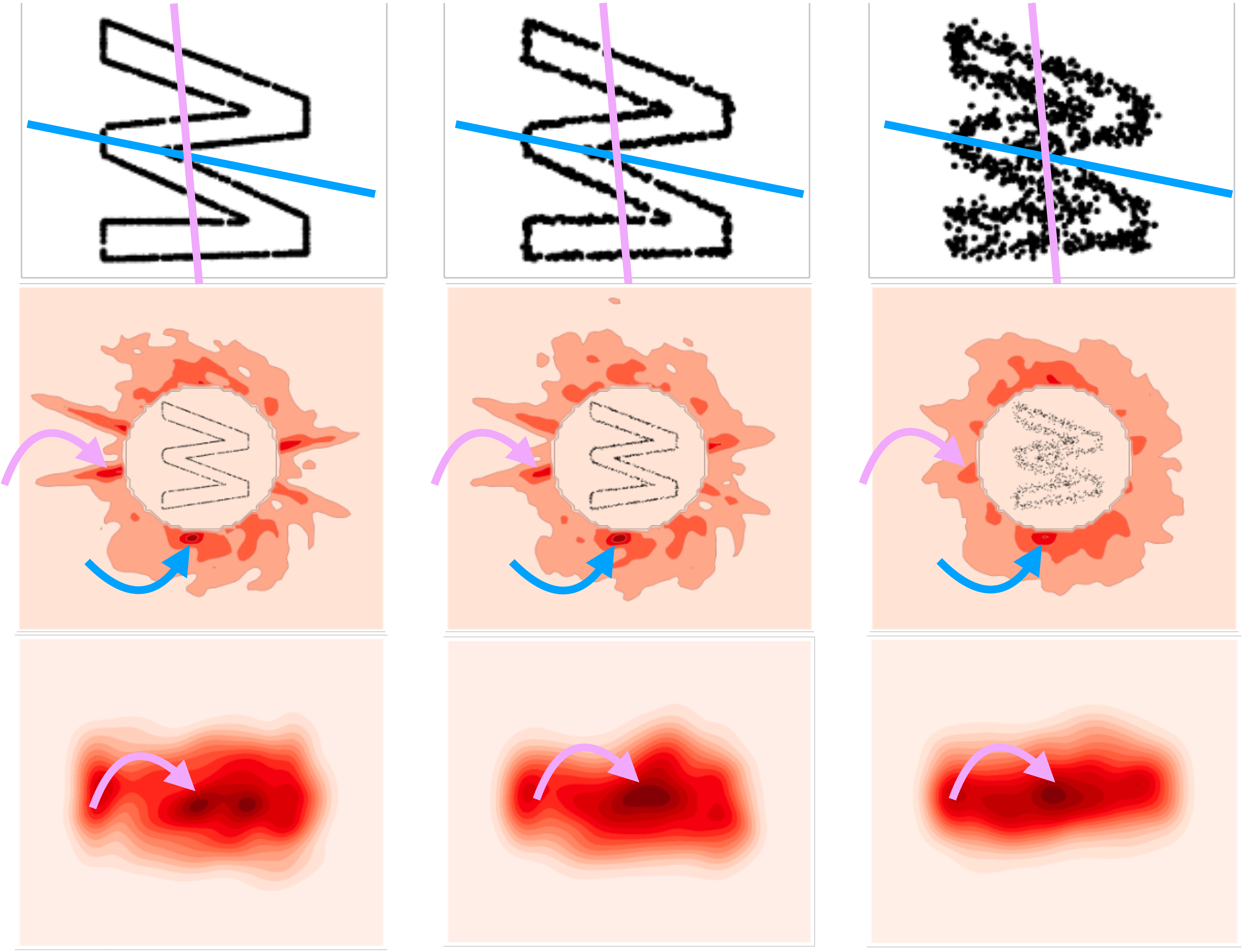
3D transformation space



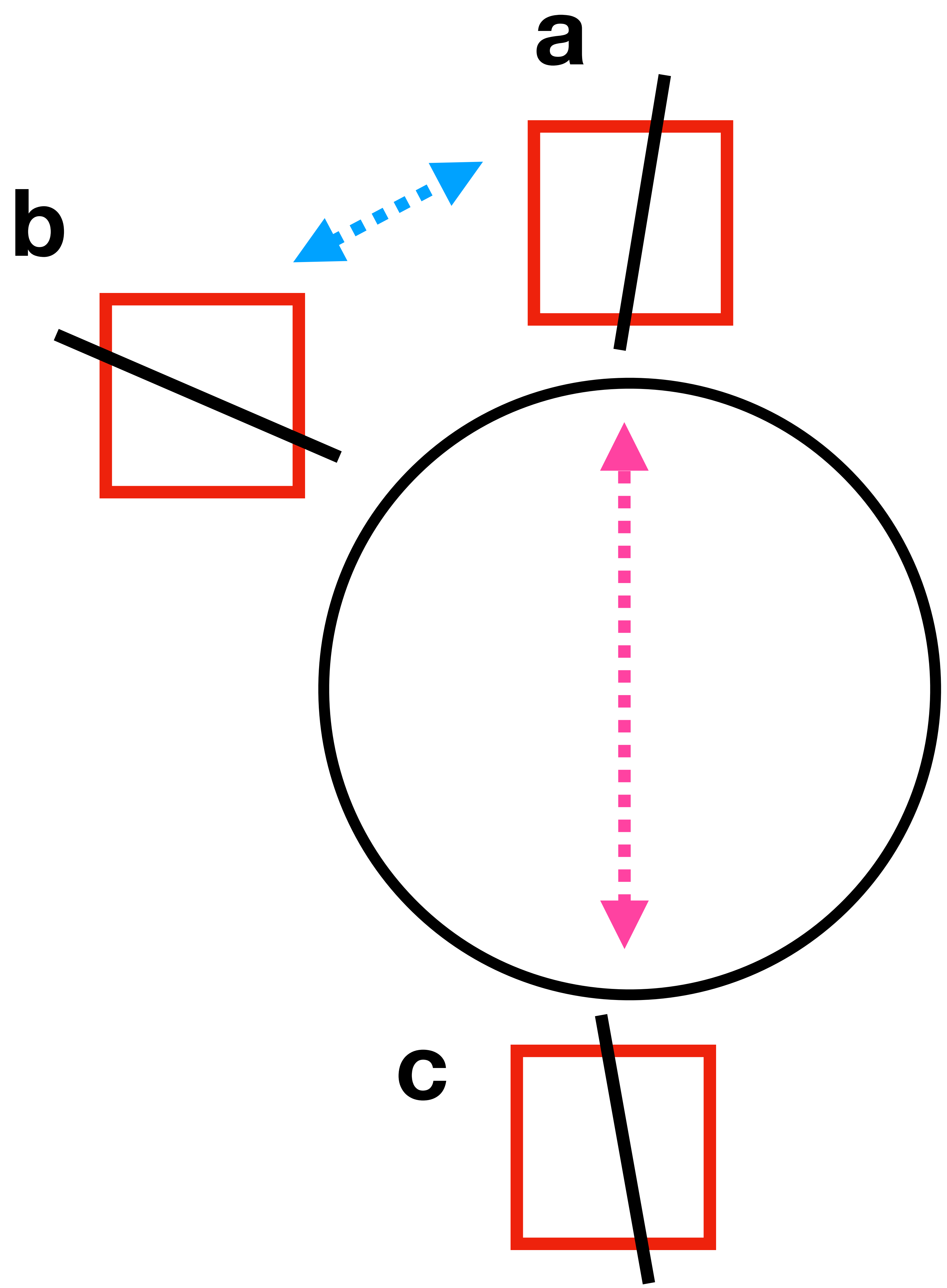
Geometry
Key symmetry

Ours

Mitra *et al.* 06



Step 2: Define Distance Function

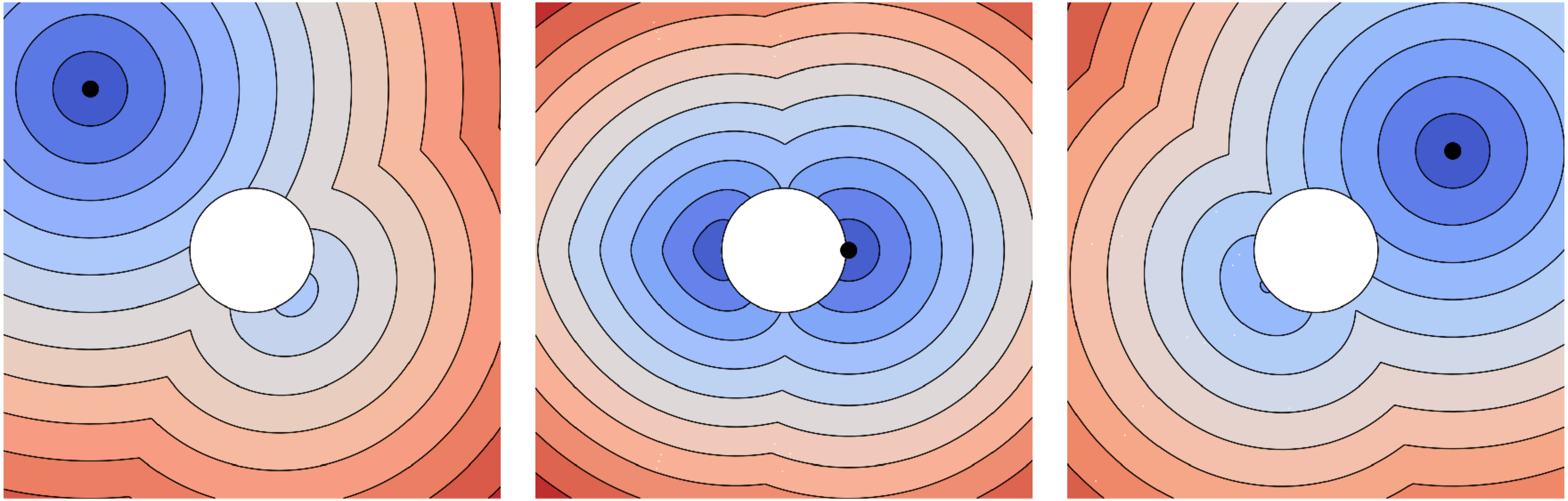


dist(a,b) > dist(a,c)

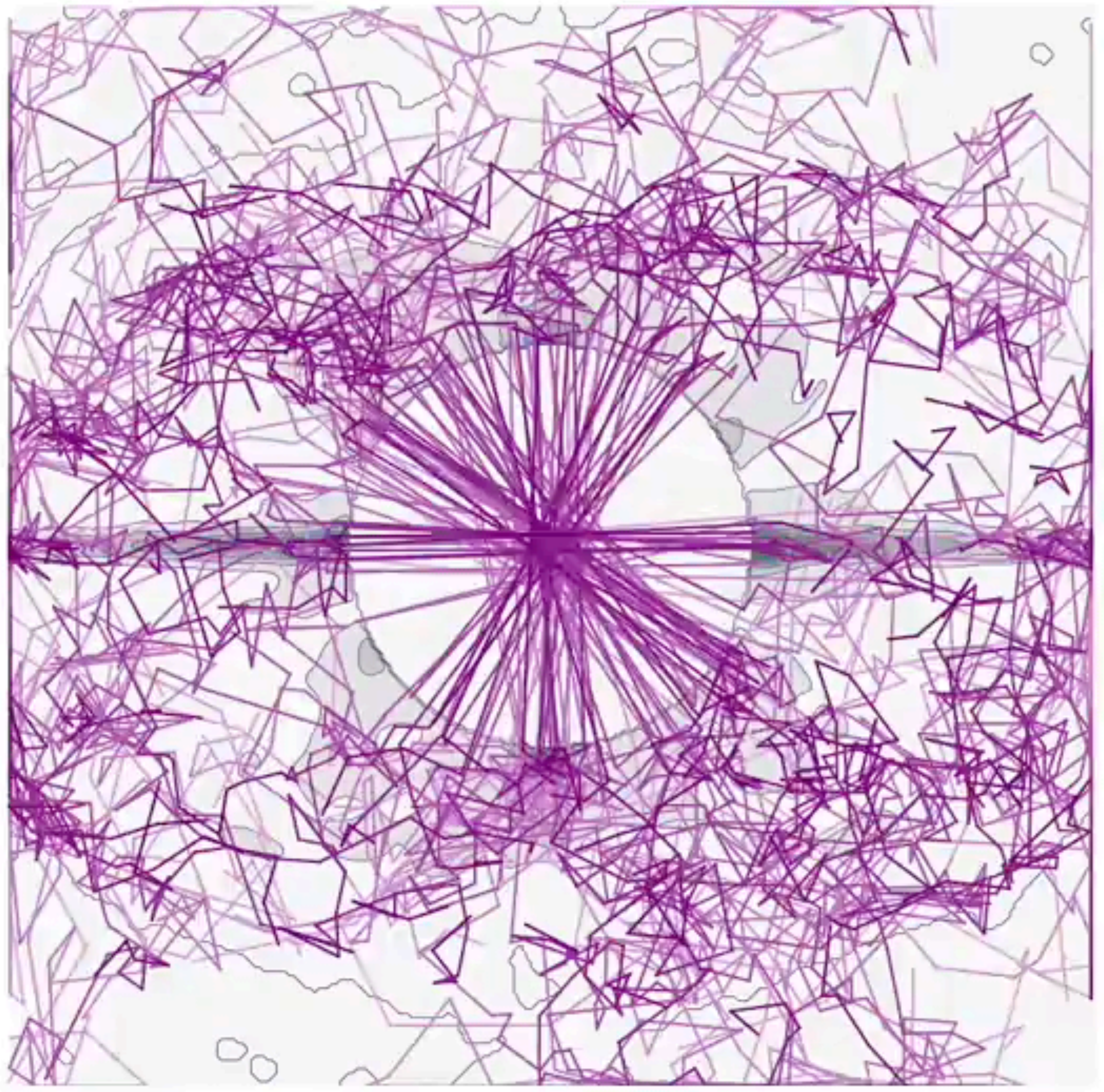
$$\mathbf{d}(x,y) = \min \begin{cases} \min_z \|x - z\| + \|y + z\| \\ \min_r \int_0^1 \mathrm{valid}(r(t))|r'(t)|dt \end{cases}$$

Step 2: Define Distance Function

$$\mathbf{d}(x, y) = \min \begin{cases} \min_z \|x - z\| + \|y + z\| \\ \min_r \int_0^1 \text{valid}(r(t))|r'(t)|dt \end{cases}$$

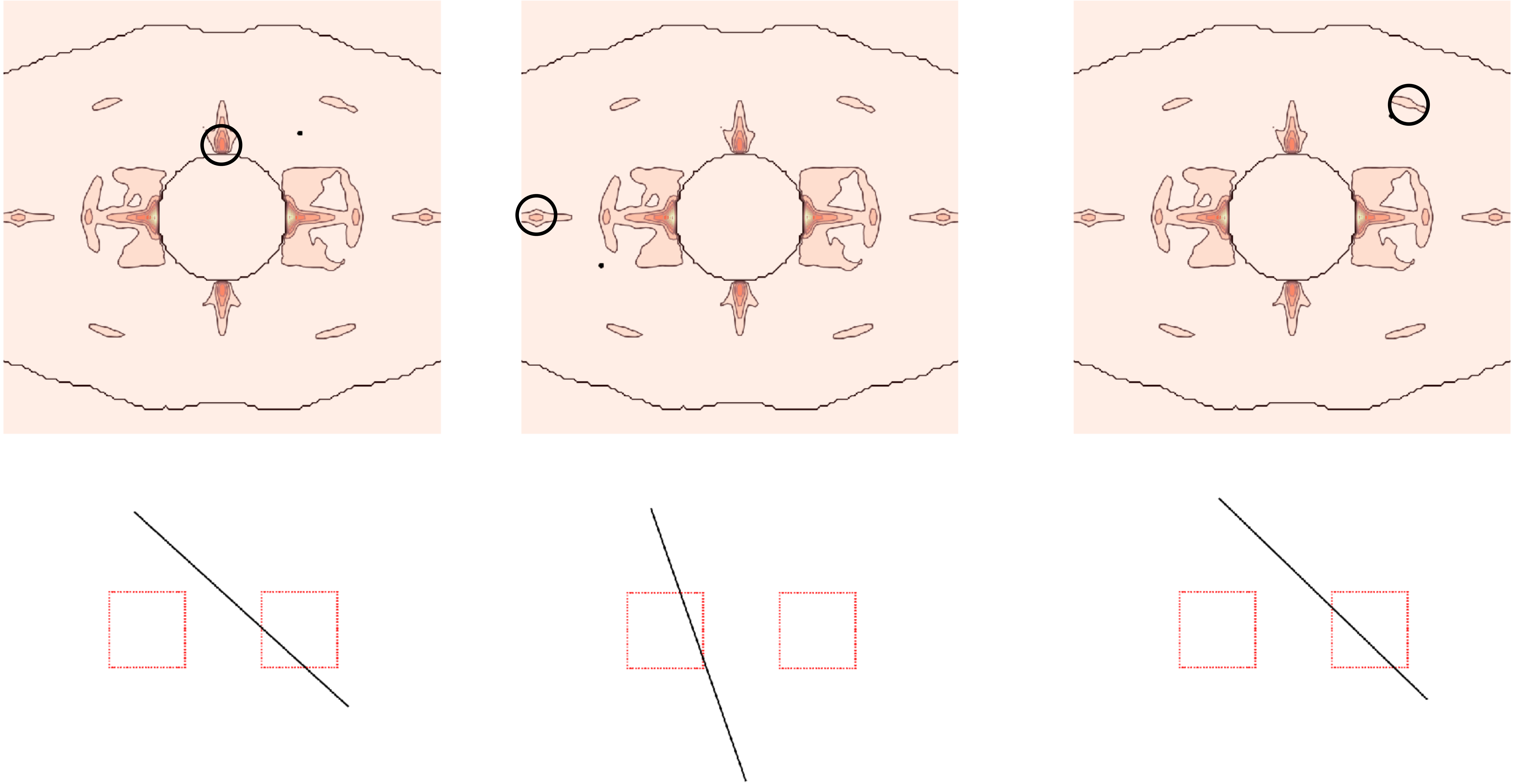


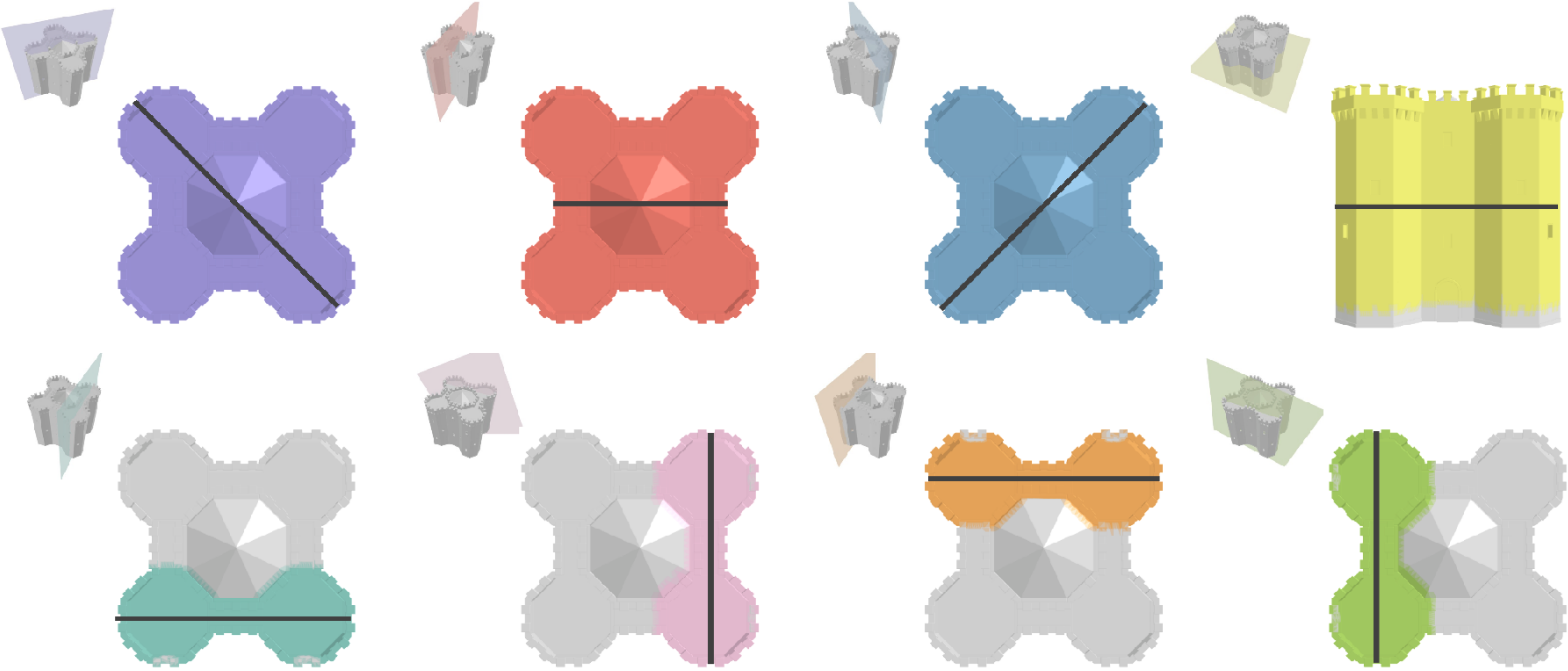
Step 3: Walking in the Transformation Space

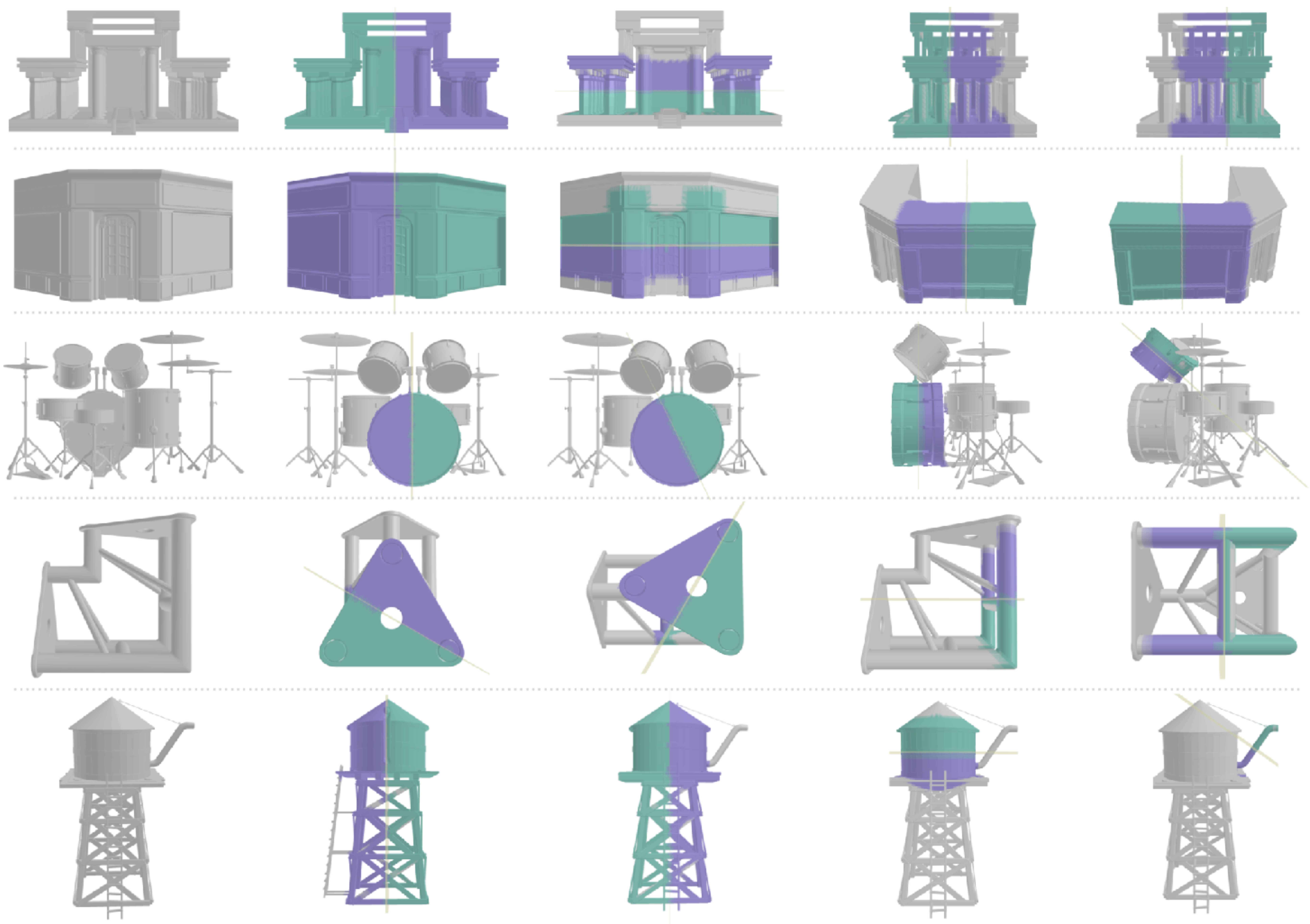


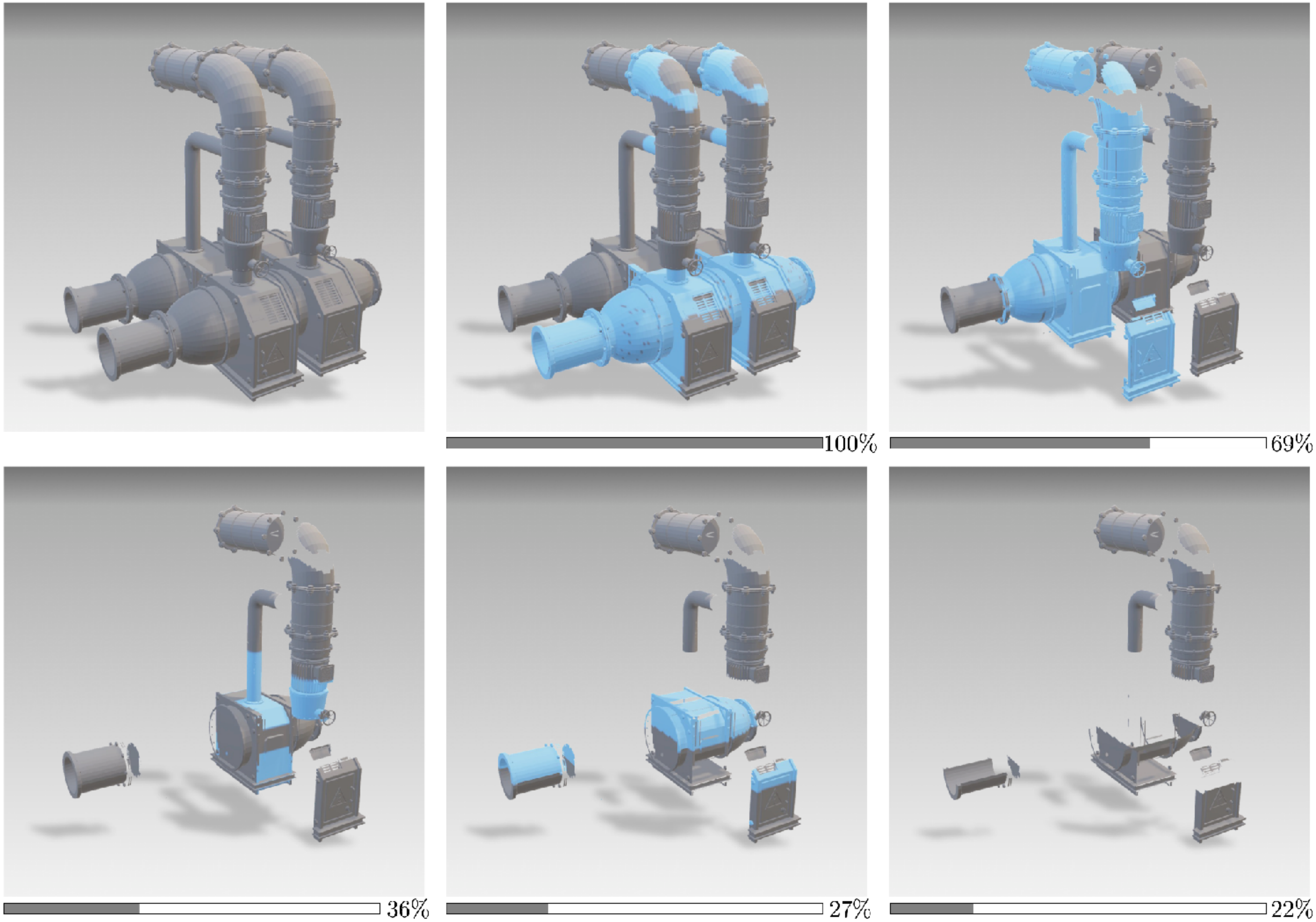
Global symmetries

Local symmetries

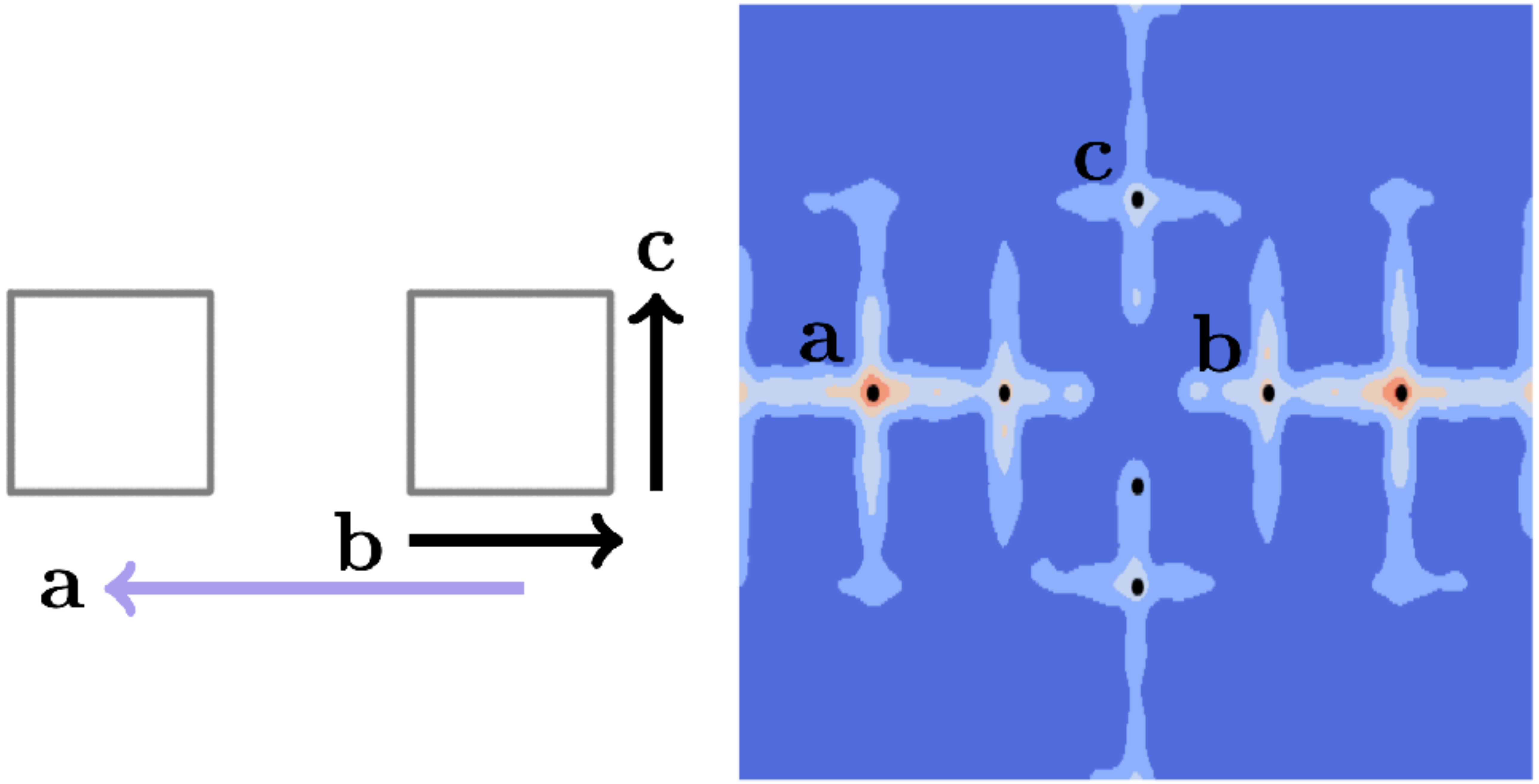


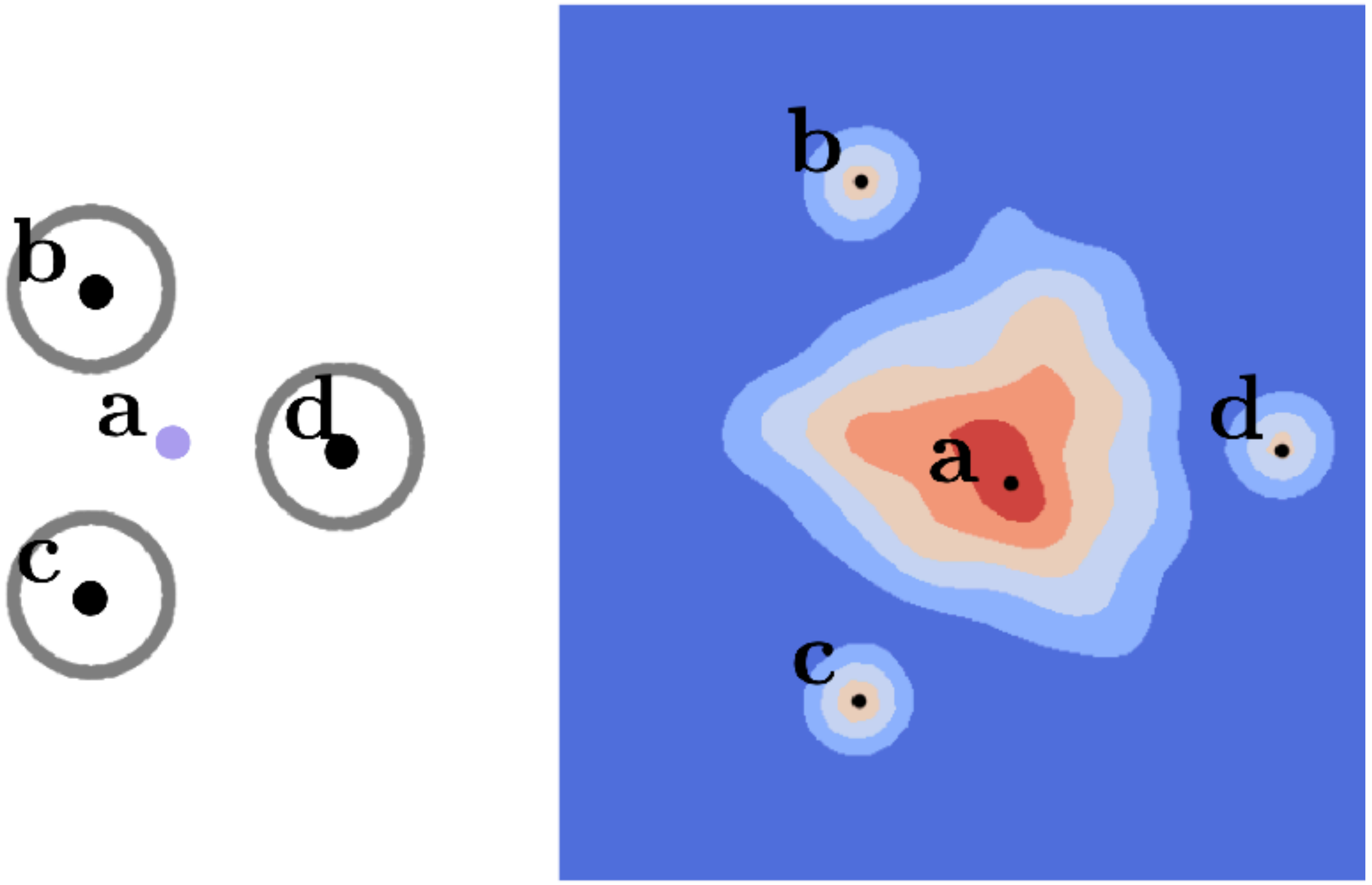





100%
69%
36%
27%
22%

Additional symmetry types



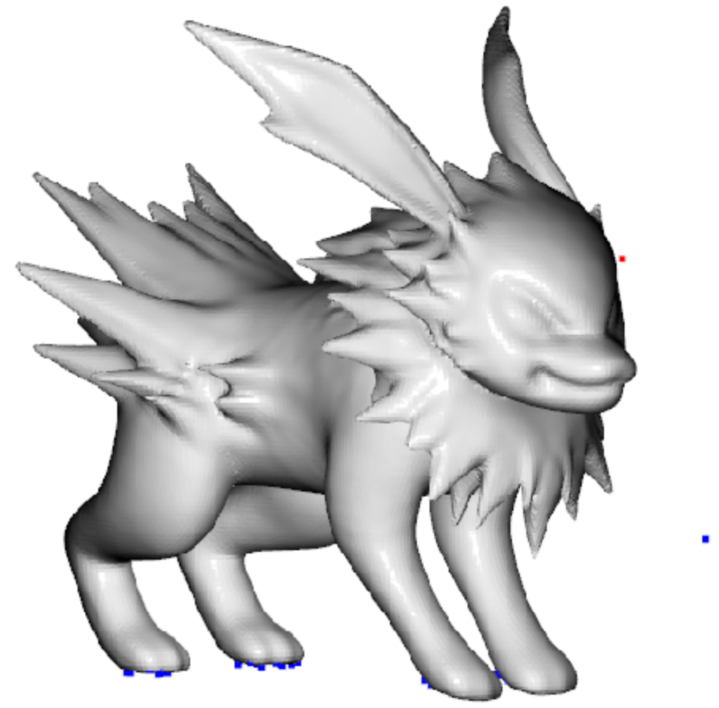


Analysis - Symmetry Detection

Synthesis

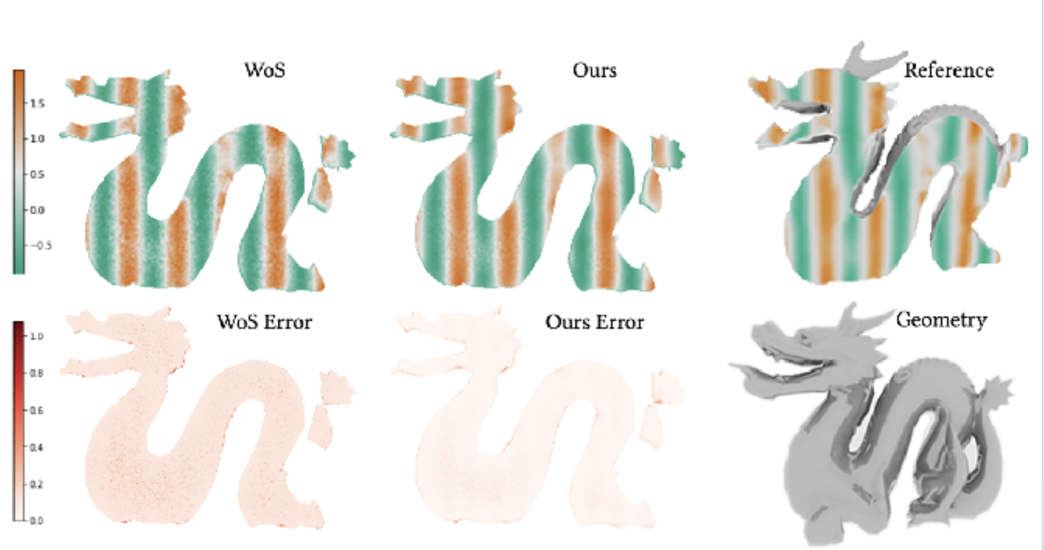
PointFlow (ICCV 2019)

ShapeGF (ECCV 2020)

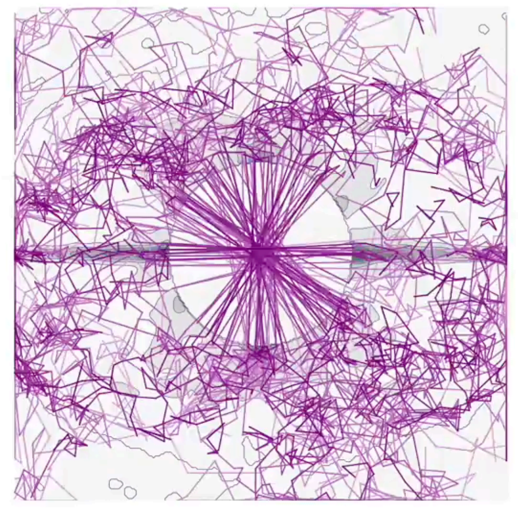


NFGP (NeurIPS 2021)

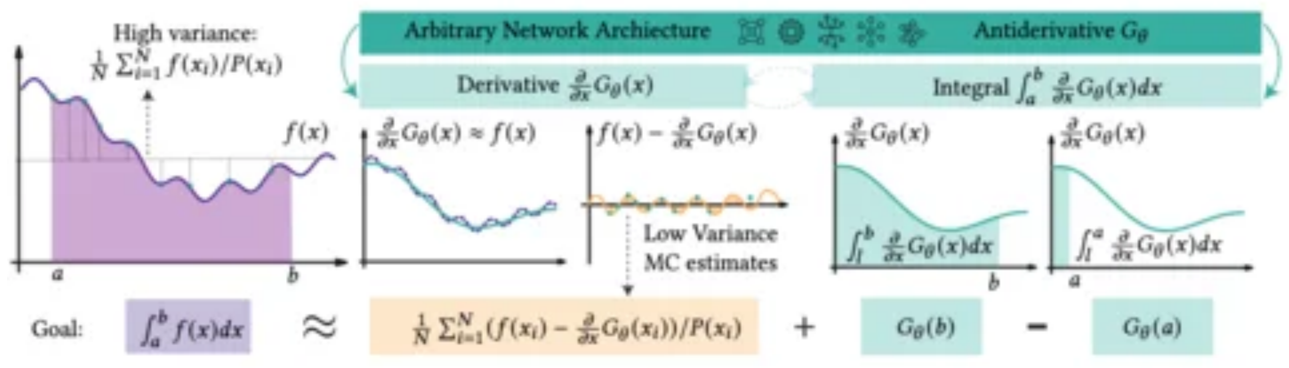
Analysis



Neural cache (SIG Asia 2023)



Symmetry (SIG Asia 2024)



NCV (SIGRAPH 2024)

2019 2020 2021 2023 2024

Thank you all!


Ruojin Cai


Zilu Li


Jihyeon Je


Qingqing Zhao


Xi Deng


Jack Liu


Shengqu Cai


Boyang Deng


Zekun Hao


Xun Huang


Ming-Yu Liu


Hadar Averbunch-Elor


Or Litany


Chris De Sa


Noah Snavely


Steve Marschner


Vladlen Koltun


Leonidas Guibas


Gordon Wetzstein


Bharath Hariharan


Serge Belongie