



Lecture 8: Surface Completion

Li Yi

Apr 10, 2024

Syllabus

Syllabus for 3D Visual Computing Spring 2025

	Geometric Foundation			Apr 17 (Week 9)	Suface Reconstruction	Project Announced
Feb 20 (Week 1)	Curve and Surface				3D Analysis	
Feb 27 (Week 2)	3D Representations	Hw1 Out		Apr 24 (Week 10)	3D Backbone Networks	Hw2 Due;
Mar 6 (Week 3)	3D Transformations			May 1 (Week 11)	Holiday	
	3D Reconstruction and Synthesis			May 8 (Week 12)	3D Backbone Networks II	In-Class Midterm
Mar 13 (Week 4)	Learning-Based SFM and SLAM			May 15 (Week 13)	3D Detection and Segmentation	
Mar 20 (Week 5)	Learning-Based Multi-view Stereo	Hw1 Due; Hw 2 Out		May 22 (Week 14)	Pose Estimation	
Mar 27 (Week 6)	NeRF			May 29 (Week 15)	Intrinsic Shape Understanding	
Apr 3 (Week 7)	Single Image to 3D			Jun 5 (Week 16)	Shape Deformation	
Apr 10 (Week 8)	3D Generative Models			Jun 12 (Week 17)		Project Due

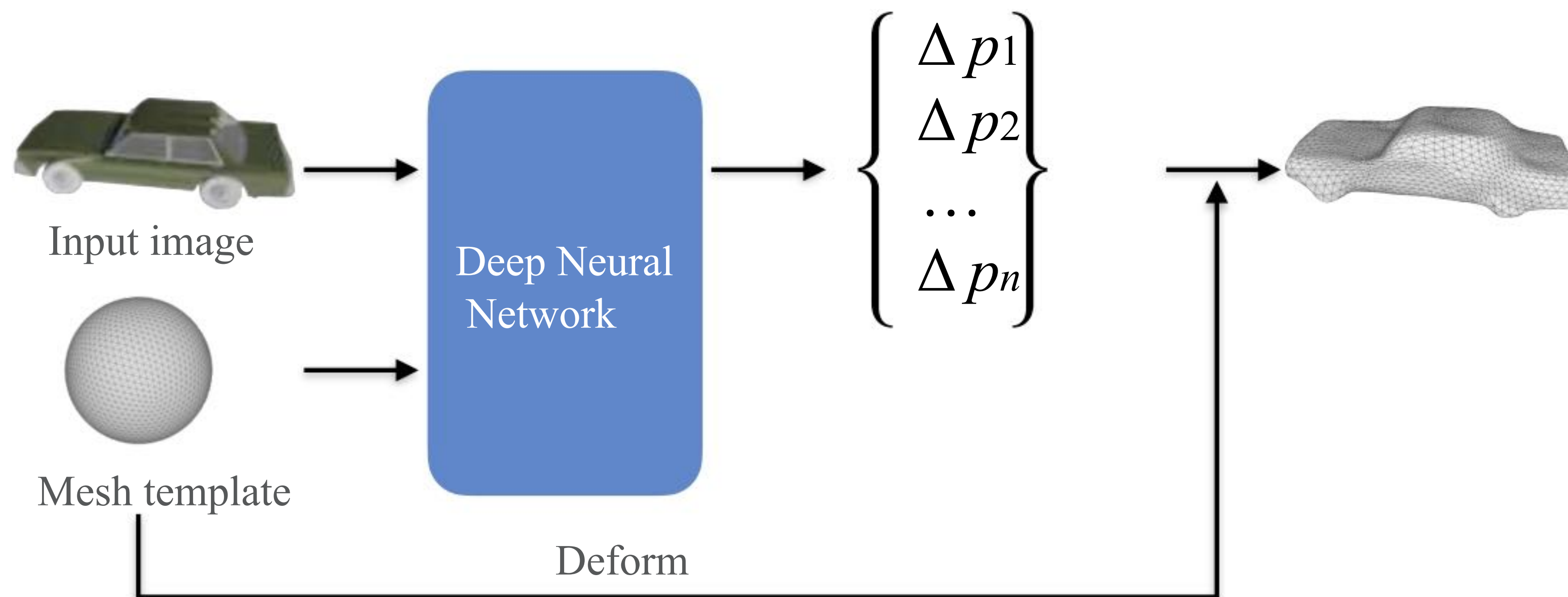
Recap

- Introduction to the task of single-image to 3D
- Synthesis-for-Learning Pipeline
- Single-image to Point Cloud
- *Single-image to Mesh*

Editing-based Mesh Modeling

- Key idea: starting from an established mesh and modify it to become the target shape

For example, deformation-based modeling:

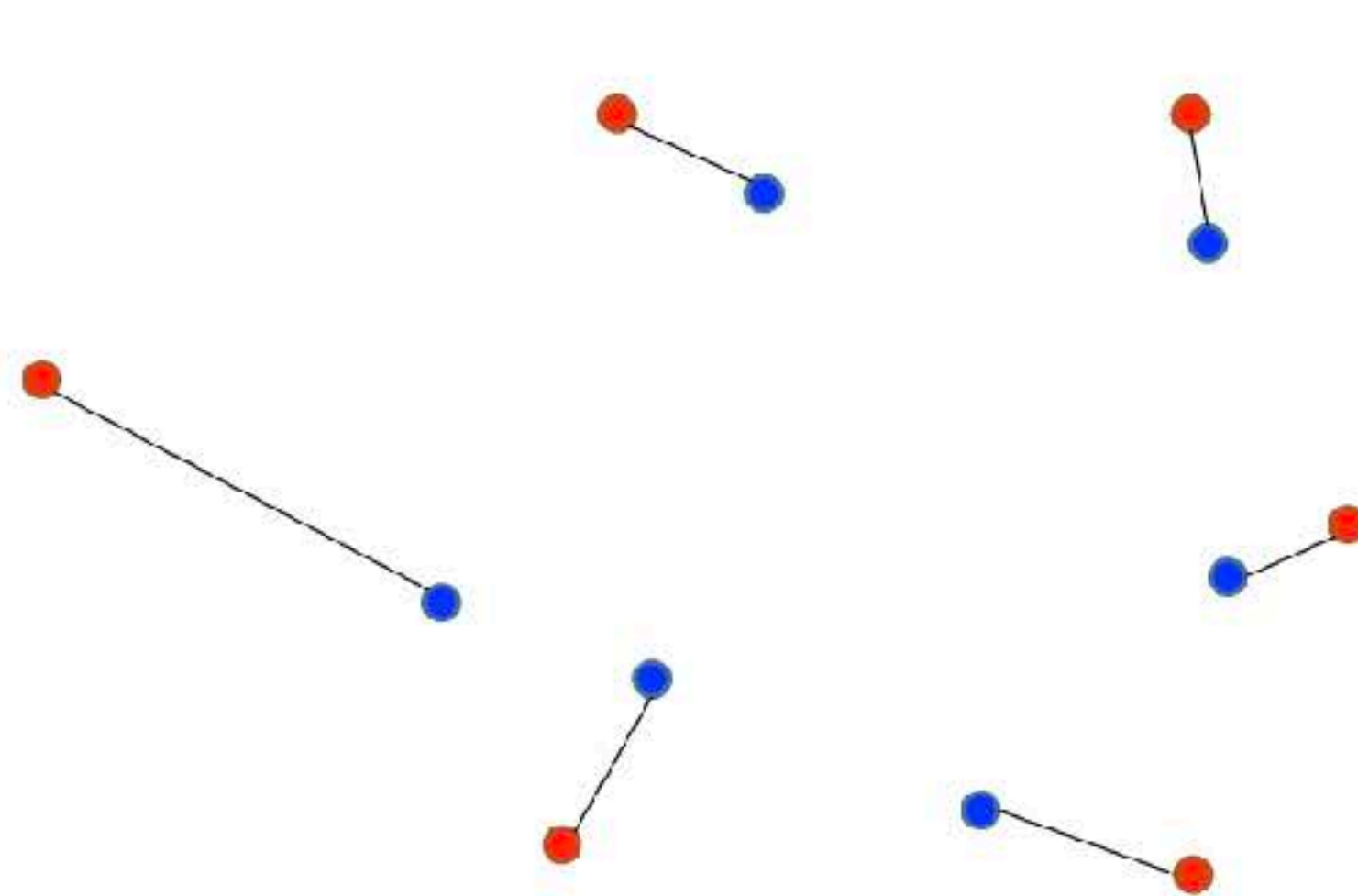


Some Example Losses for Mesh Editing

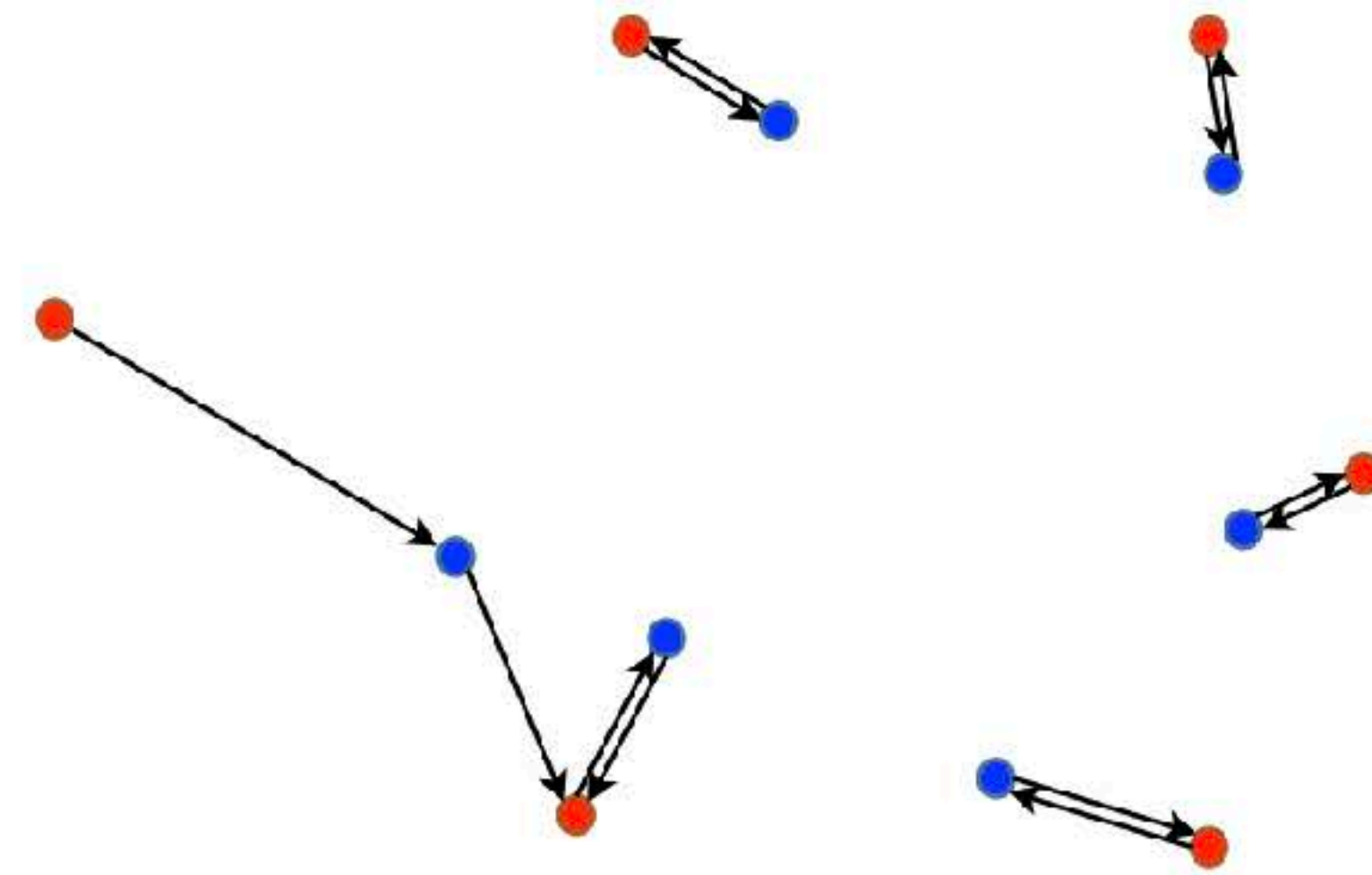
- Vertices distance.
 - Vertices point set distance.
- Uniform vertices distribution.
 - Edge length regularizer.
- Mesh surface smoothness.
- Normal Loss.

Loss I: Set Distance between Vertices

- Vertices are a set of points
- Recall the metrics for point clouds



Earth Mover's distance



Chamfer distance

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \rightarrow S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2 \quad d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2$$

Loss II: Uniform Vertices Distribution

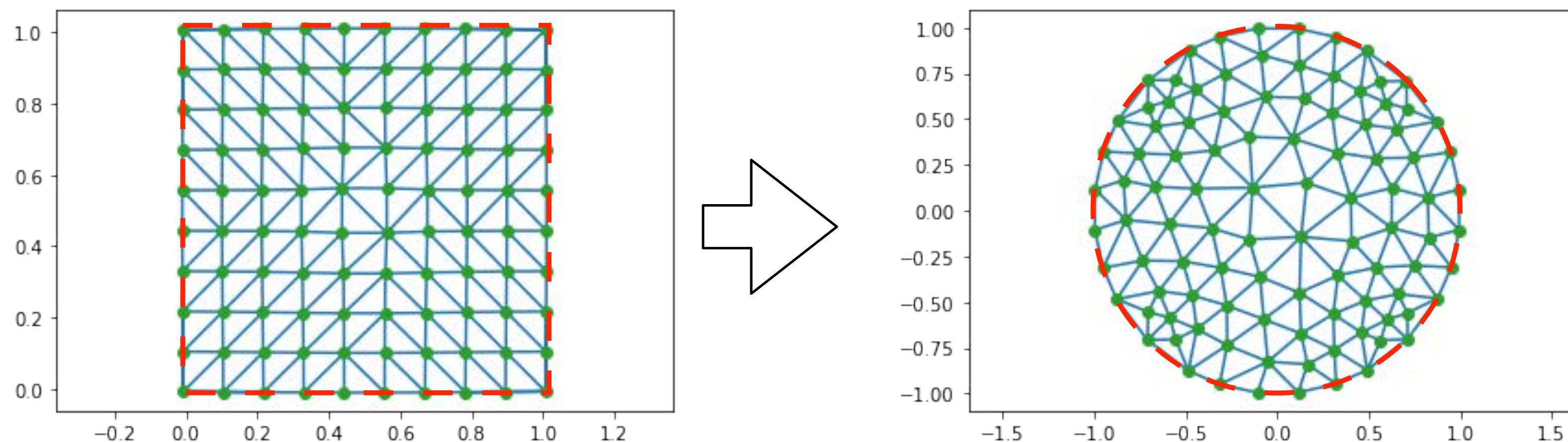
- Penalizes the flying vertices and overlong edges to guarantee the high quality of recovered 3D geometry
- Encourage equal edge length between vertices

$$L_{\text{unif}} = \sum_p \sum_{k \in N(p)} \|p - k\|_2^2$$

Loss II: Uniform Vertices Distribution

$$L_{\text{unif}} = \sum_p \sum_{k \in N(p)} \|p - k\|_2^2$$

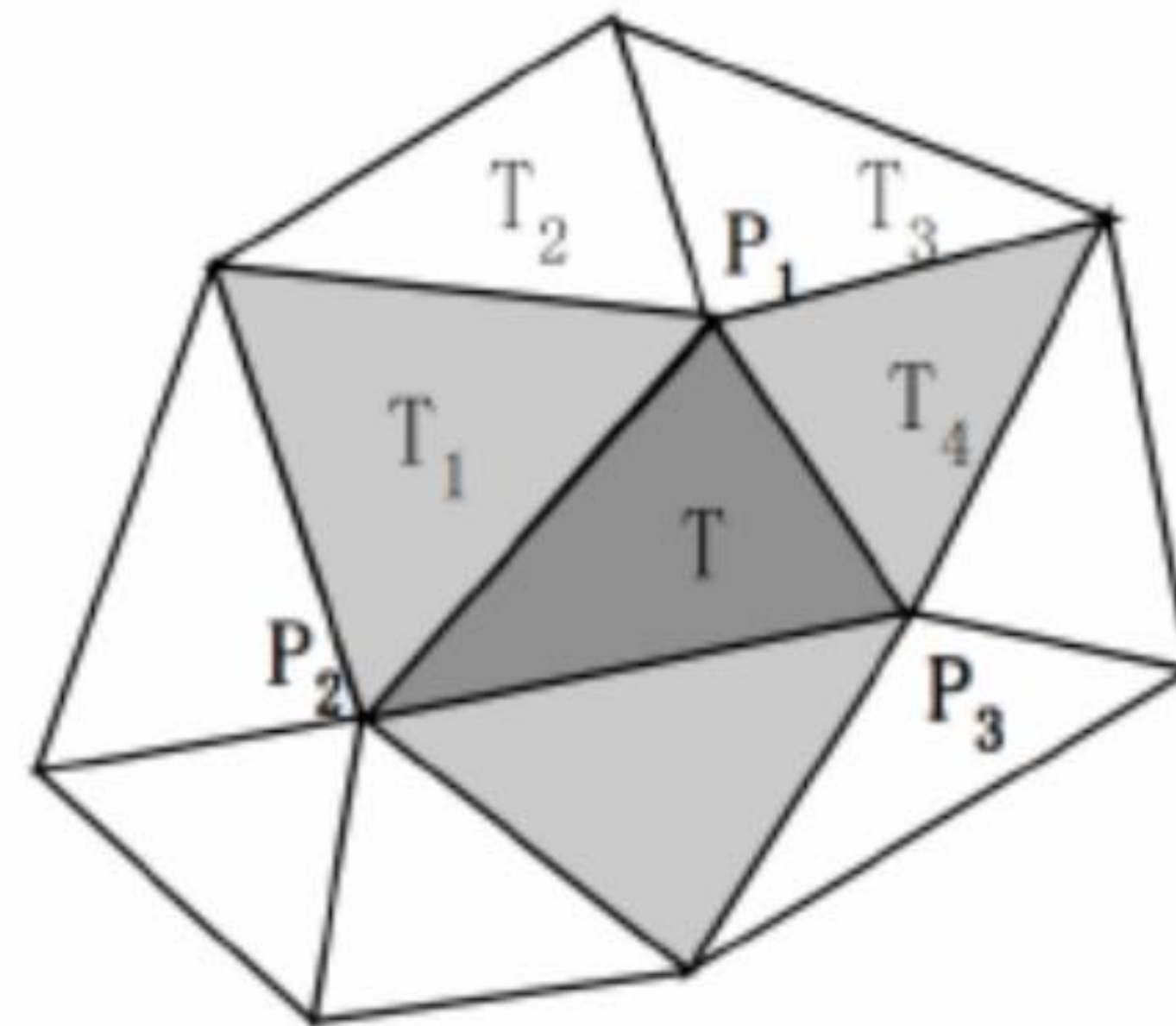
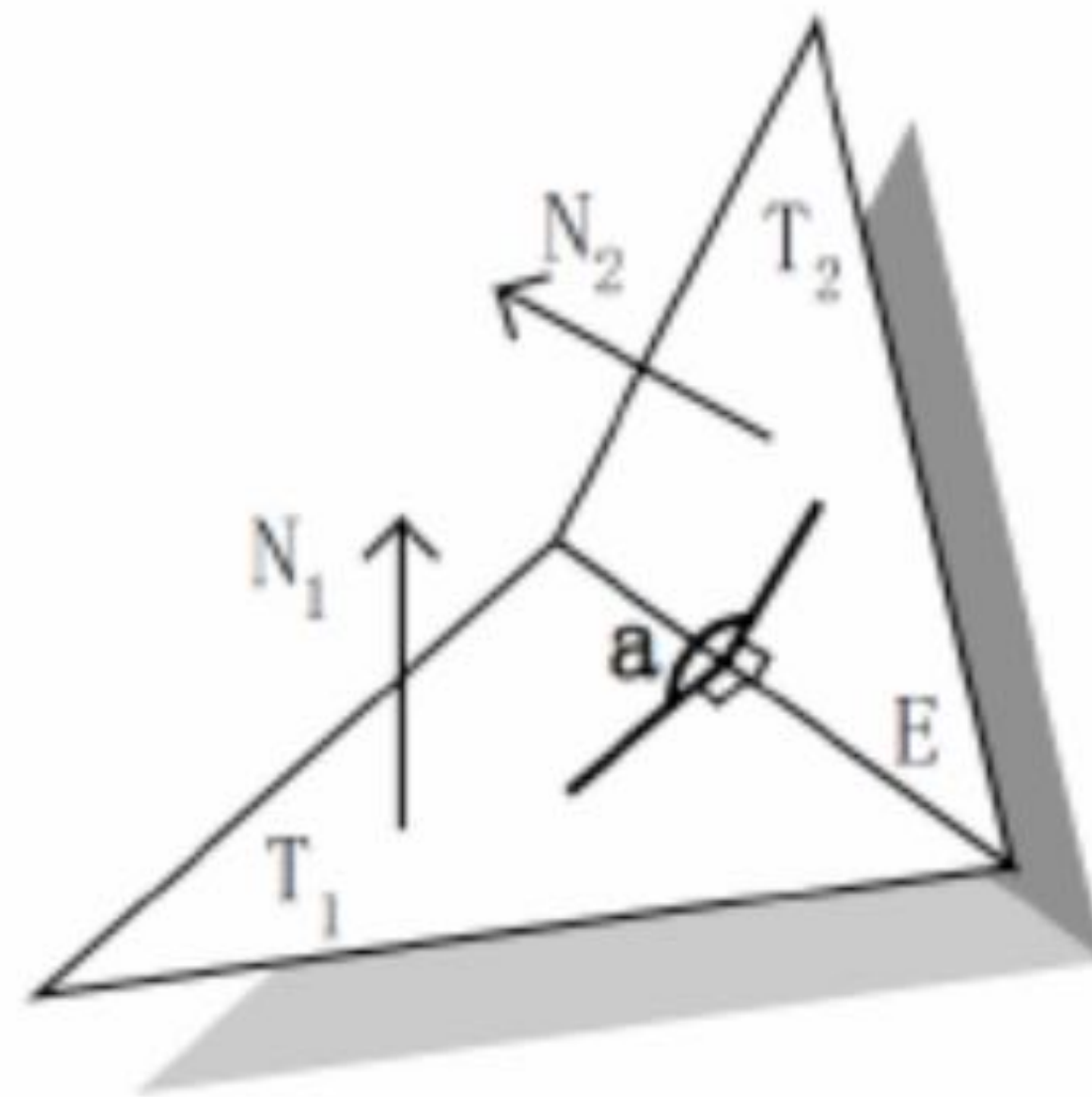
Effect of minimizing l when fixing topology and setting boundary points to the new positions



Loss III: Mesh Smoothness

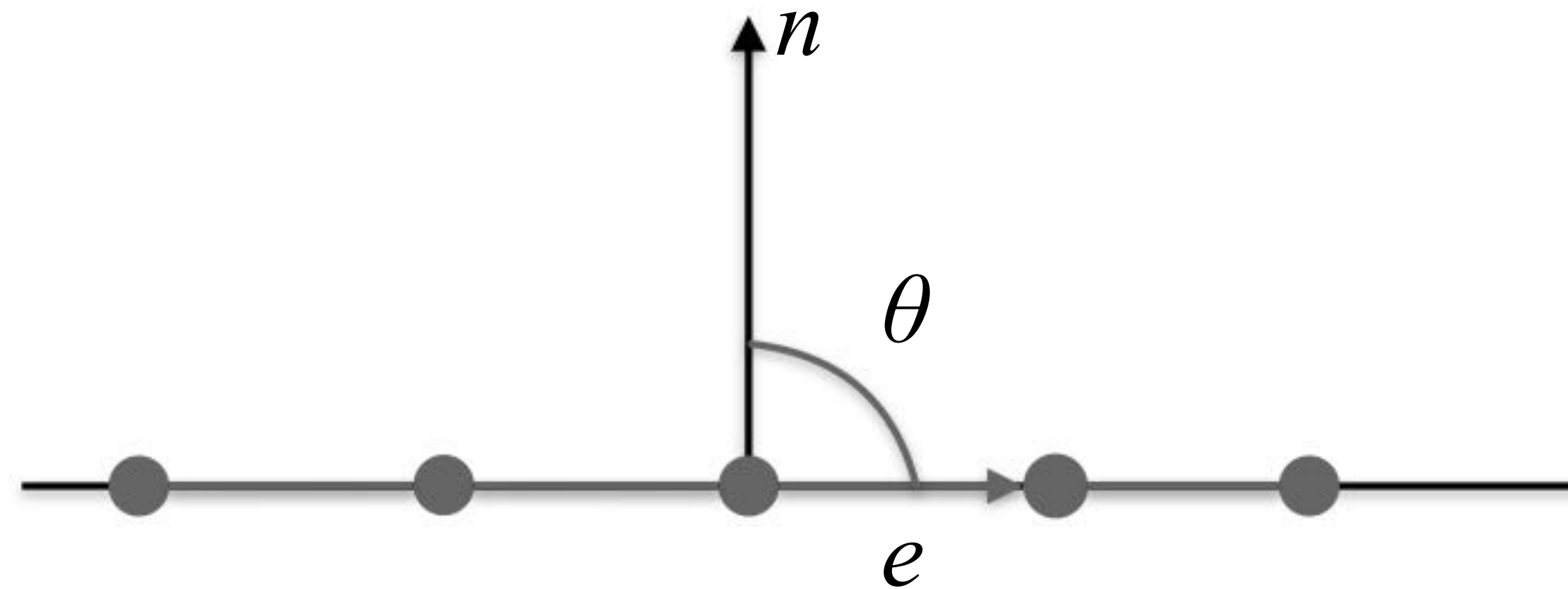
- L_{smooth} encourages that intersection angles of faces are close to 180 degrees.

$$L_{smooth} = \sum_i (\cos \theta_i + 1)^2$$



Loss IV: Normal Loss

- **Key assumption:** vertices within a local neighborhood lie on the same tangent plane.
- Regularize edge to be perpendicular to the underlying groundtruth vertex normal

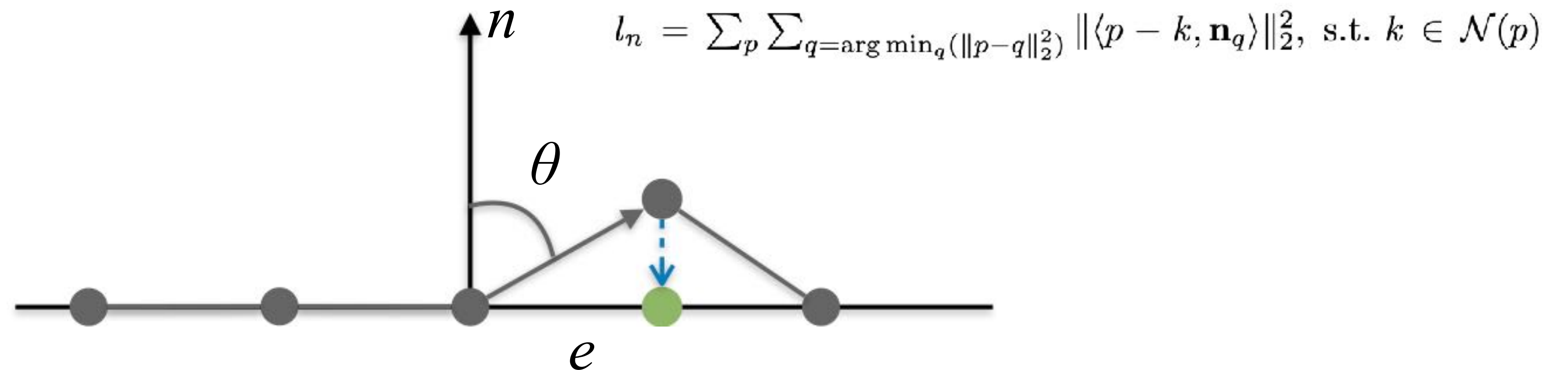


Loss IV: Normal Loss

- But how to find the vertices normal?
- One approach: use the nearest ground truth point normal as current vertex normal.

Loss IV: Normal Loss

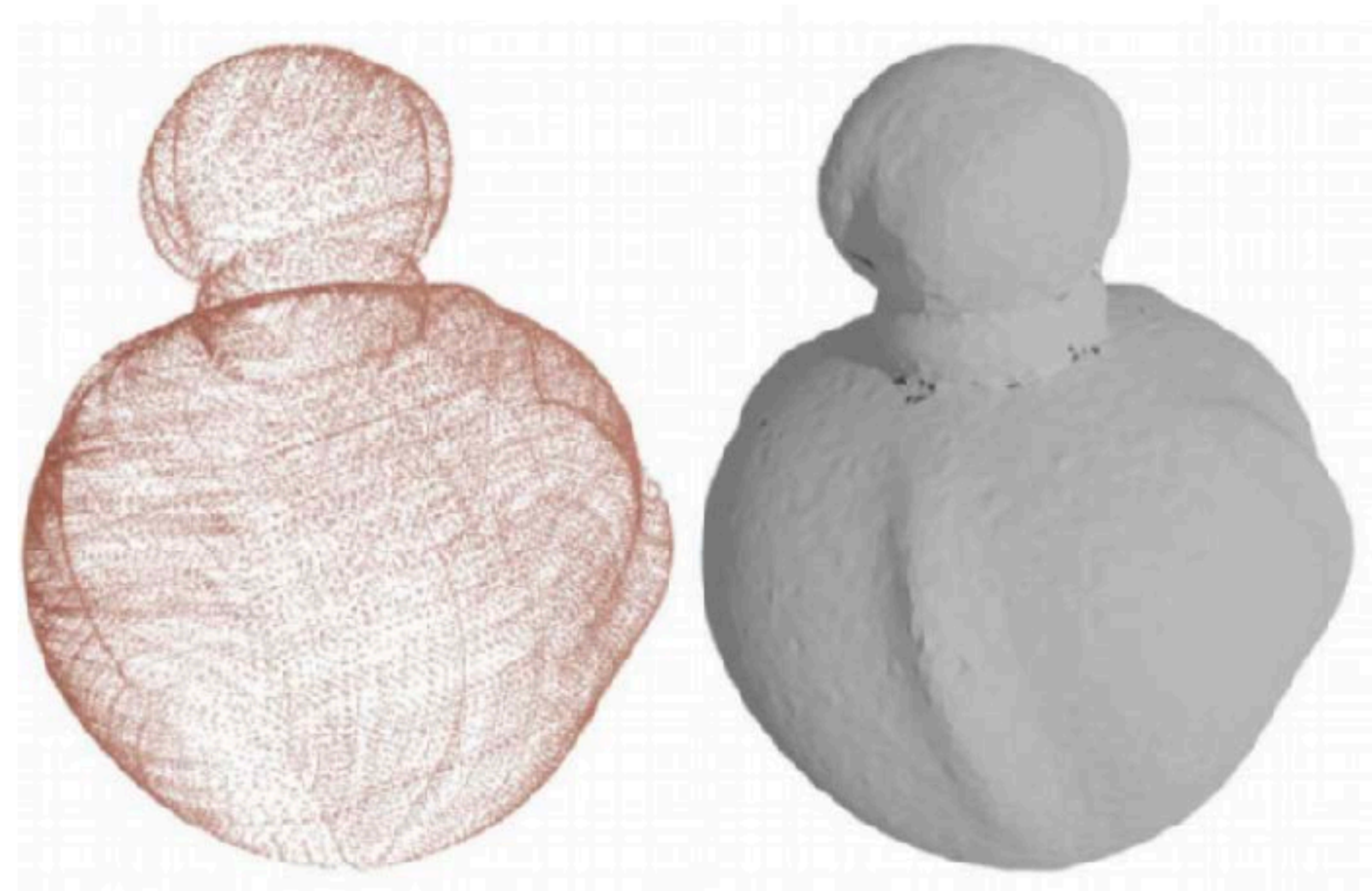
- But how to find the vertices normal?
- One approach: use the nearest ground truth point normal as current vertex normal.
- Penalize the edge direction to perpendicular to vertex normal.



Recap

- Synthesis-for-learning pipeline leverages easy-to-obtain synthetic data for challenging 3D visual understanding tasks
- Single image to 3D point cloud is possible with properly defined set metric (EMD and CD)
- Natural ambiguity in single image to 3D
- Single image to mesh can be achieved through template deformation
- Mesh reconstruction requires more regularizations

Today's Focus



Surface Reconstruction

Outline

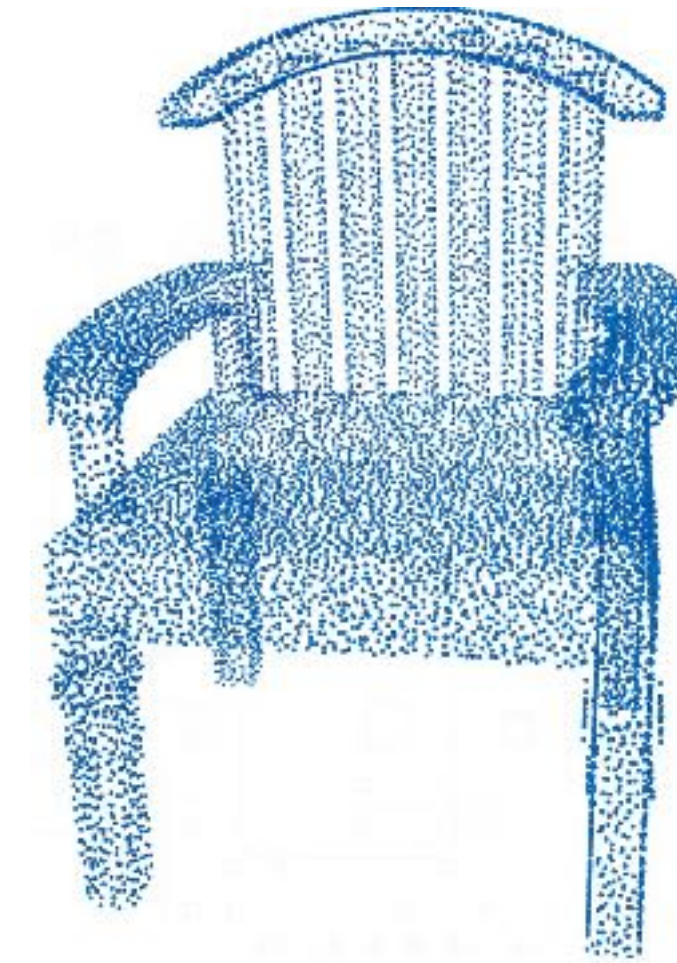
- Problem Definition
- Explicit Algorithms
- Implicit Algorithms

Outline

- *Problem Definition*
- Explicit Algorithms
- Implicit Algorithms

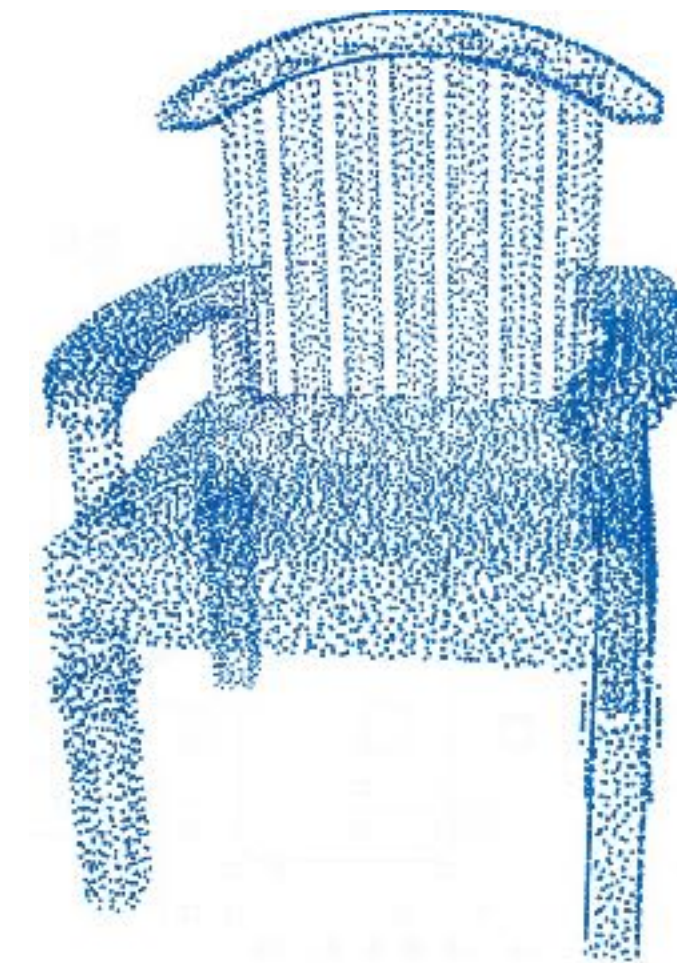
Problem Definition

- Input: point cloud (with or without normals)
- Output: triangle mesh



Two Basic Families

- Explicit algorithms
 - Directly connect the input points with triangles, e.g.,
 - ball-pivoting algorithm
 - extrinsic-intrinsic ratio algorithm
- Implicit algorithms
 - Approximate the input points by implicit field functions $S = \{x : F(x) = 0\}$
 - Then extract iso-surfaces, e.g.,
 - poisson surface reconstruction
 - reconstruction with RBF

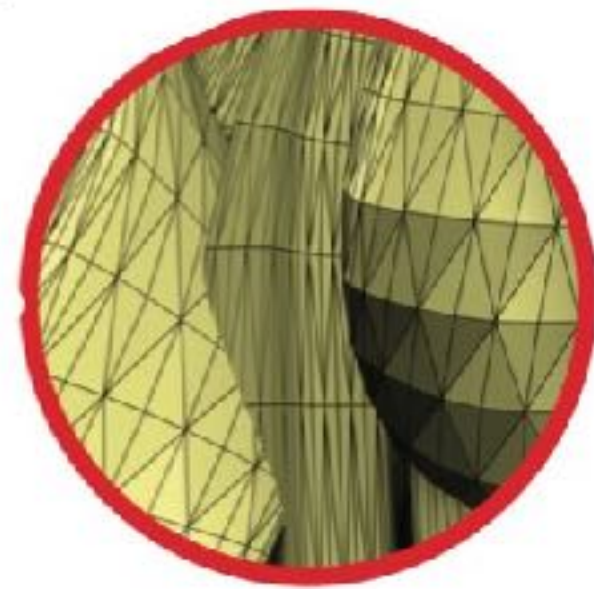


Some Desired Properties of the Algorithm

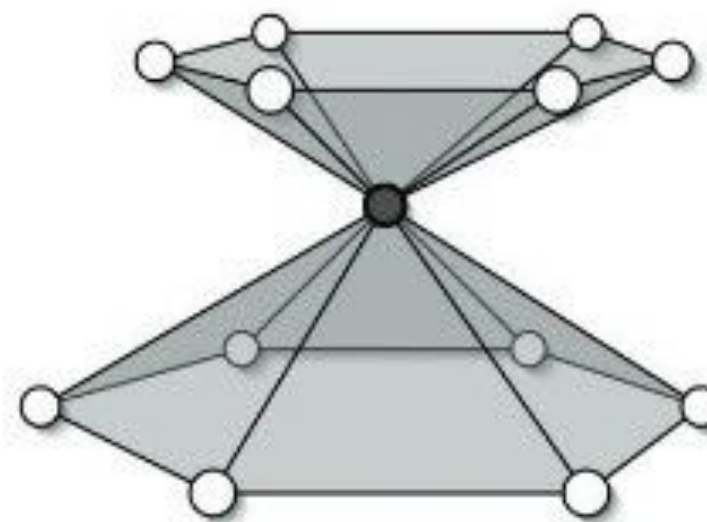
- Fast: The input point cloud may be large. We expect the computation to be fast.
- Robust: May recover the underlying surface structure even when the input point cloud is noisy
- Output mesh is desired to satisfy some geometric constraints

Geometric Constraint: Manifold

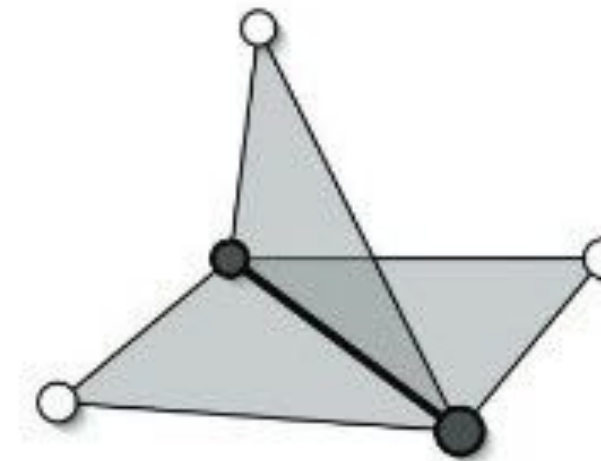
- A mesh is **manifold** if it does not contain:
 - self intersection
 - non-manifold edge (has more than 2 incident faces)
 - non-manifold vertex (one-ring neighborhood is not connected after removing the vertex)



self intersection



non-manifold vertex

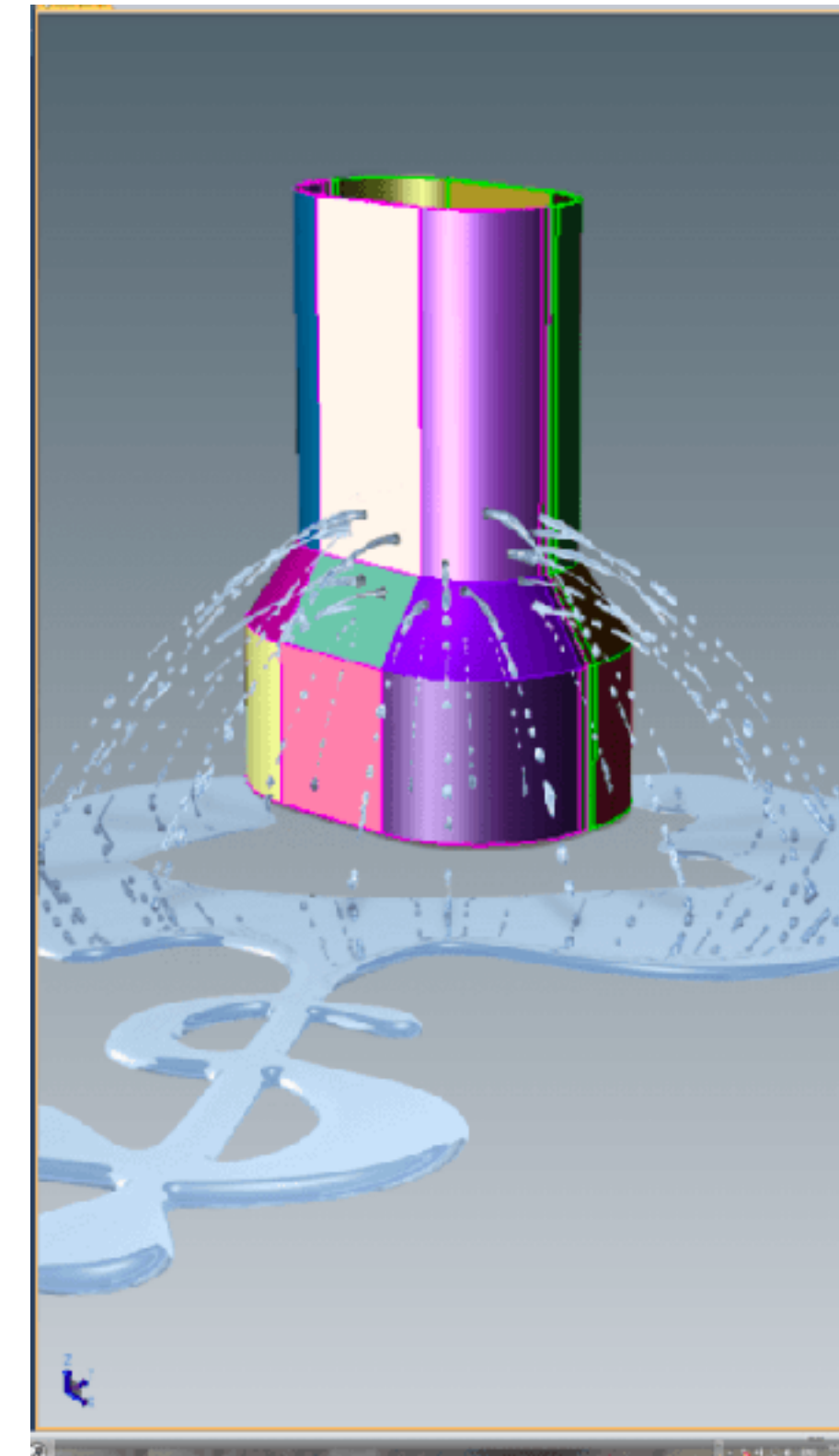


non-manifold edge

- A useful property for many subsequent geometry processing pipelines
 - e.g., to add texture maps and ...

Geometric Constraint: Watertight

- A **manifold** mesh is **watertight** if each edge has **exactly** two incident faces, i.e., no boundary edges.
- Defines the interior, hence the volume of a solid object
- Required by many physical- simulation algorithms:
 - Estimate mass from density
 - Collision between objects
 - Force simulation
 - ...



Outline

- Problem Definition
- *Explicit Algorithms*
- Implicit Algorithms

Ball-Pivoting Algorithm

- Input: a point cloud and a hyper-parameter ρ

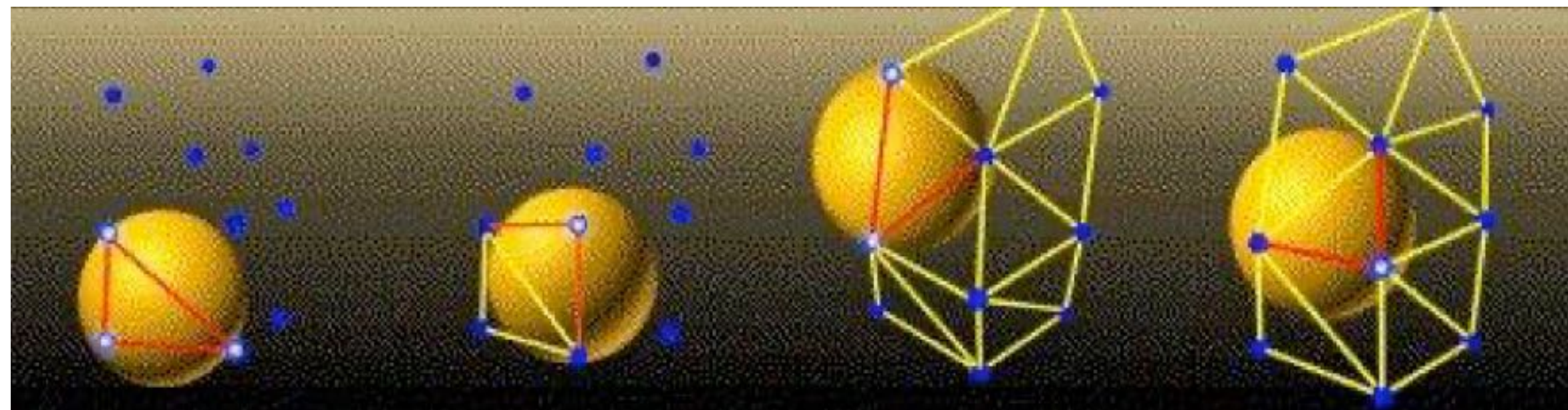
Ball-Pivoting Algorithm

- Input: a point cloud and a hyper-parameter ρ
- Assumption:
 - input points are dense enough that a ball of radius ρ cannot pass through the surface without touching the points.



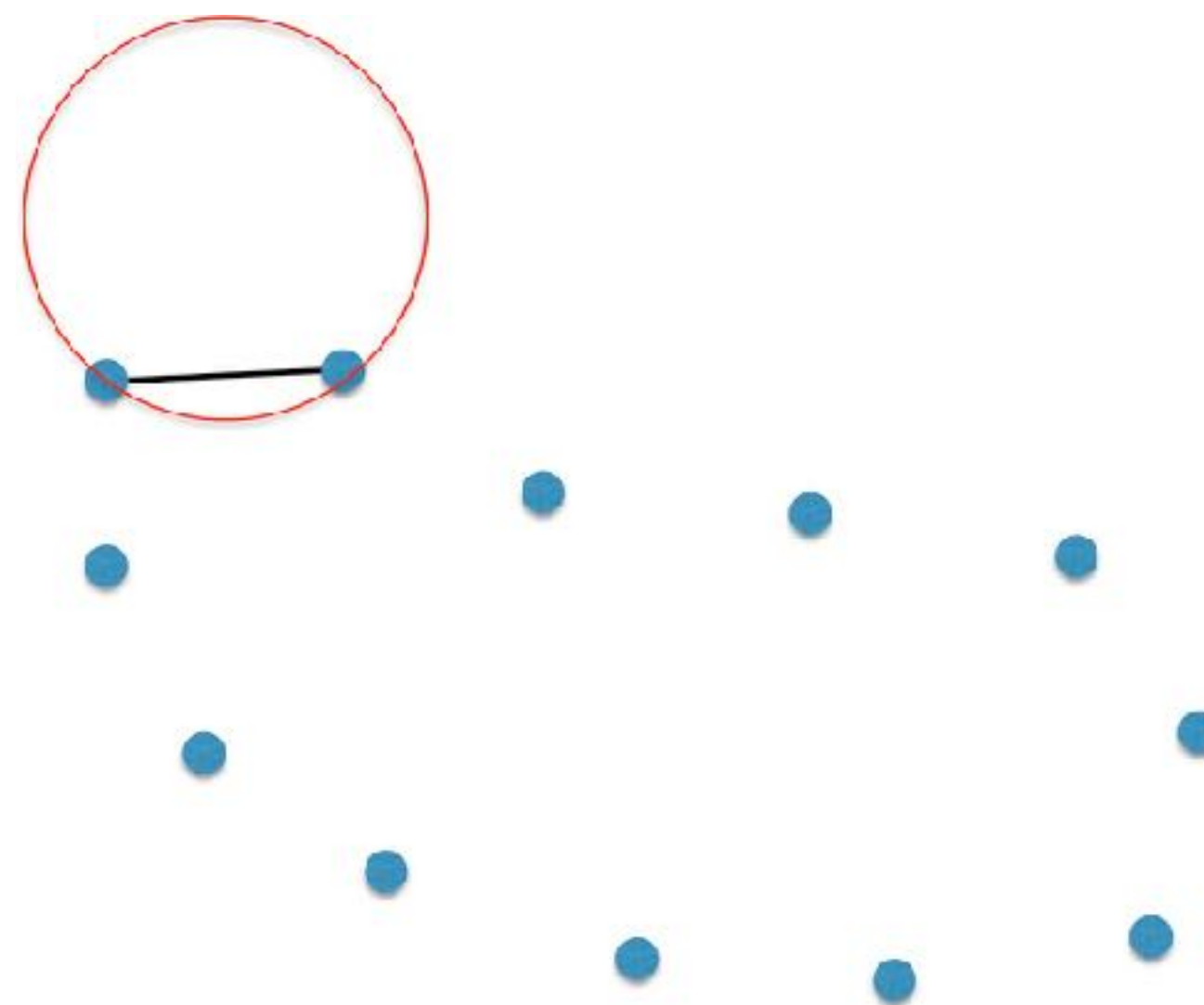
Ball-Pivoting Algorithm

- Input: a point cloud and a hyper-parameter ρ
- Assumption:
 - input points are dense enough that a ball of radius ρ cannot pass through the surface without touching the points.
- Principle for face formation:
 - three points form a triangle if a ball of radius ρ touches them without containing any other points.



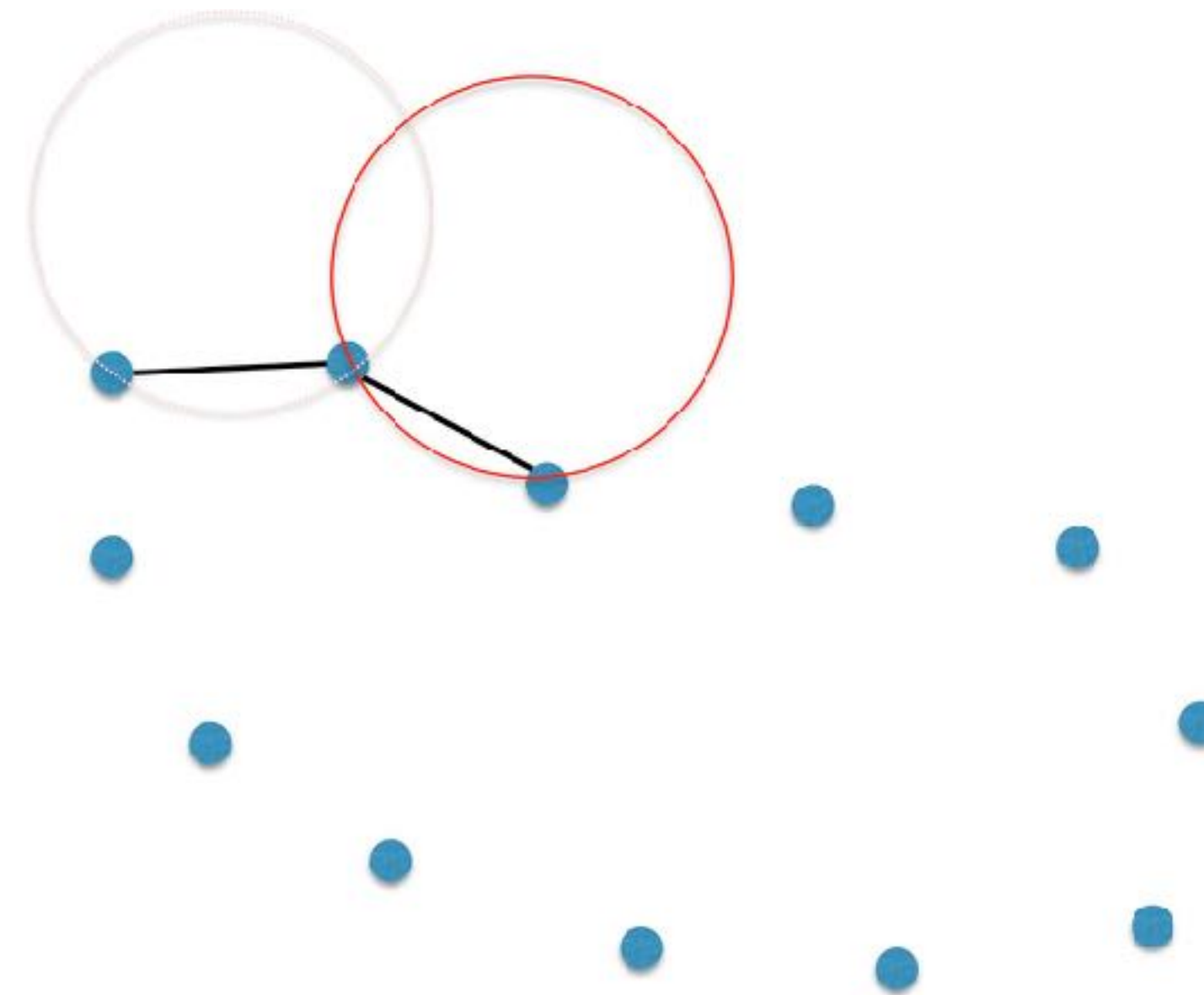
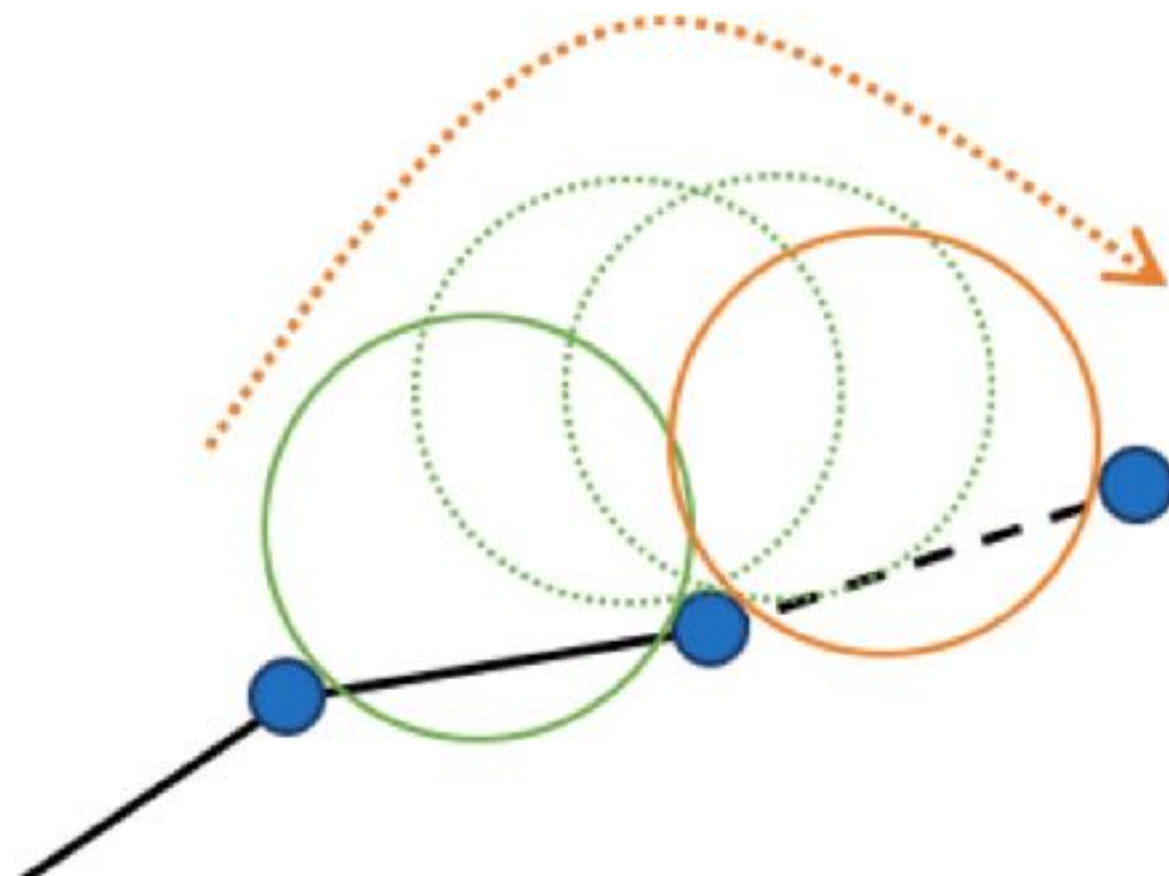
Ball-Pivoting Algorithm (2D)

- Starting with a corner point and a ρ -ball
- Verify potential edges (triangles) in the ρ -neighborhood by the previous principle



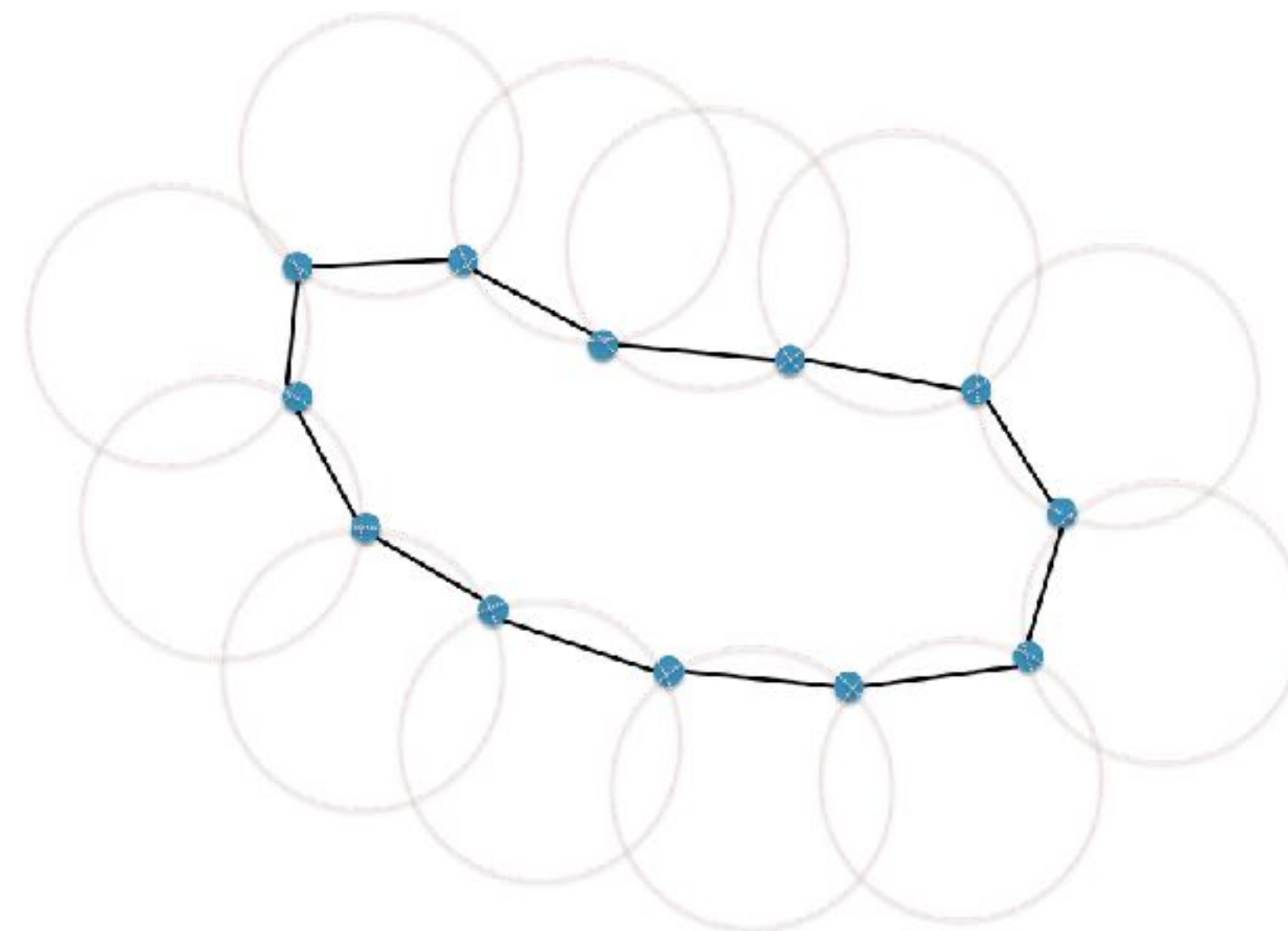
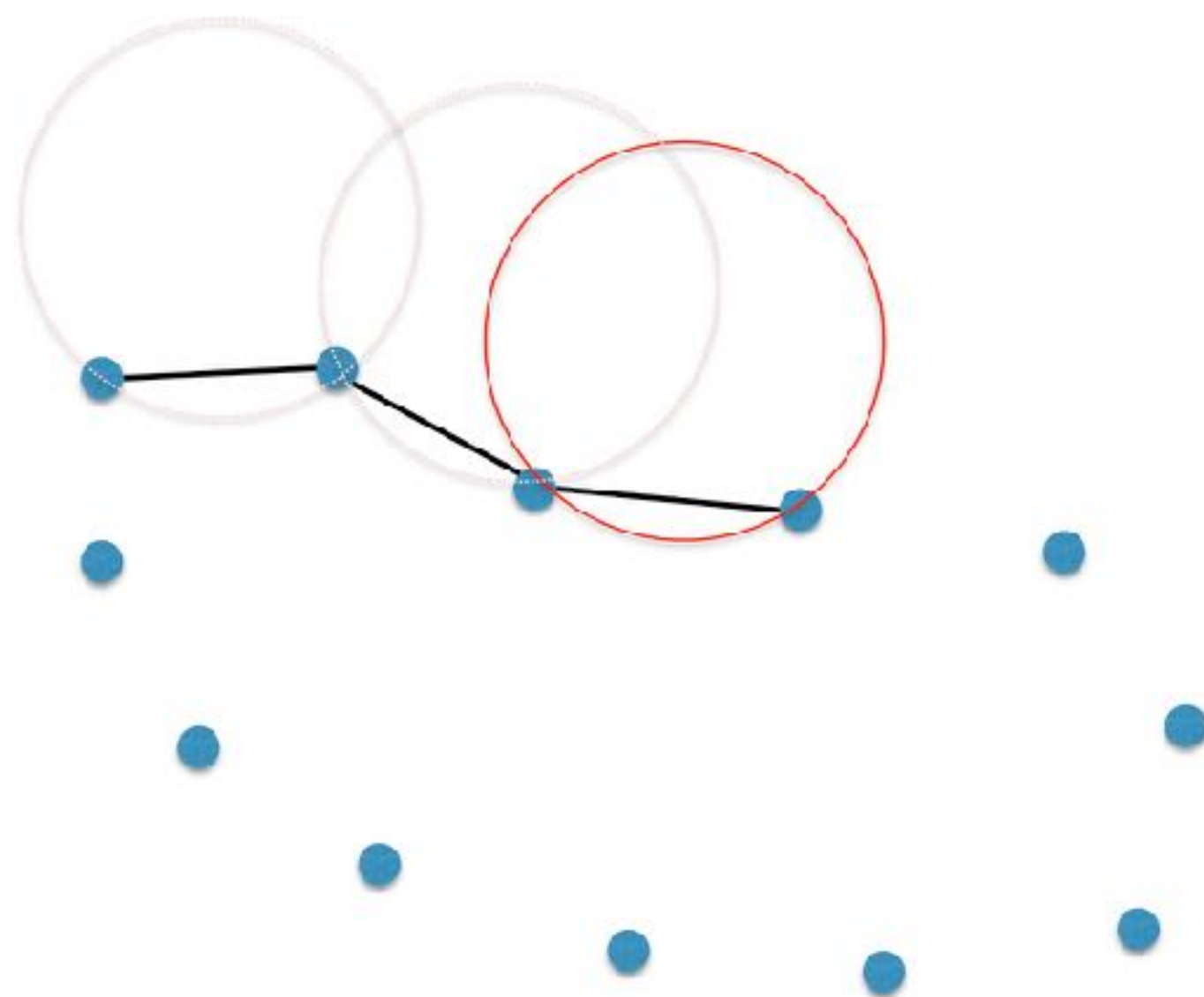
Ball-Pivoting Algorithm (2D)

- The ball pivots around an edge (triangles) until it touches another point, forming another triangle.



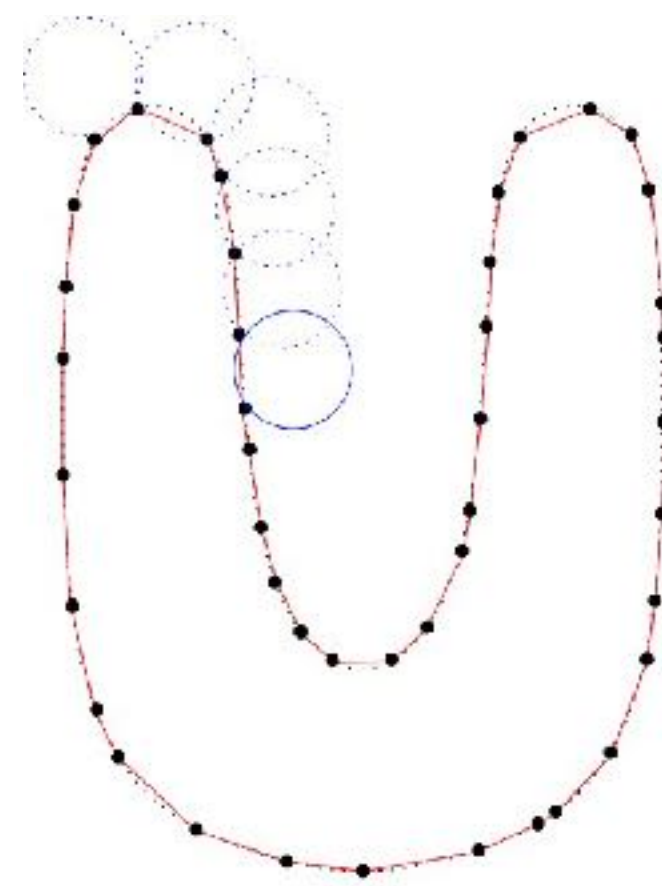
Ball-Pivoting Algorithm (2D)

- The process continues until all reachable edges have been tried
- Then starts from another seed triangle, until all points have been considered.

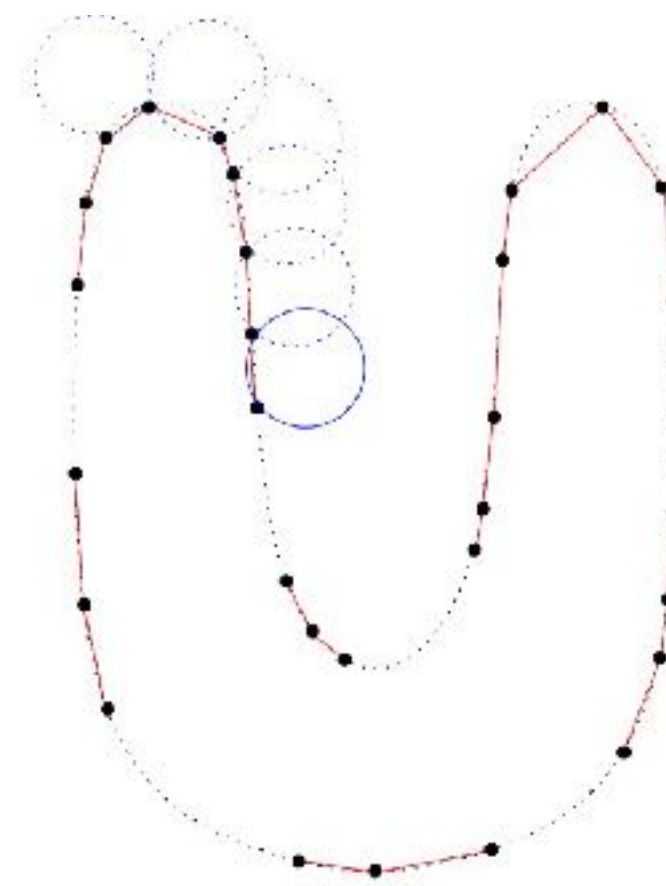


Radius ρ Matters

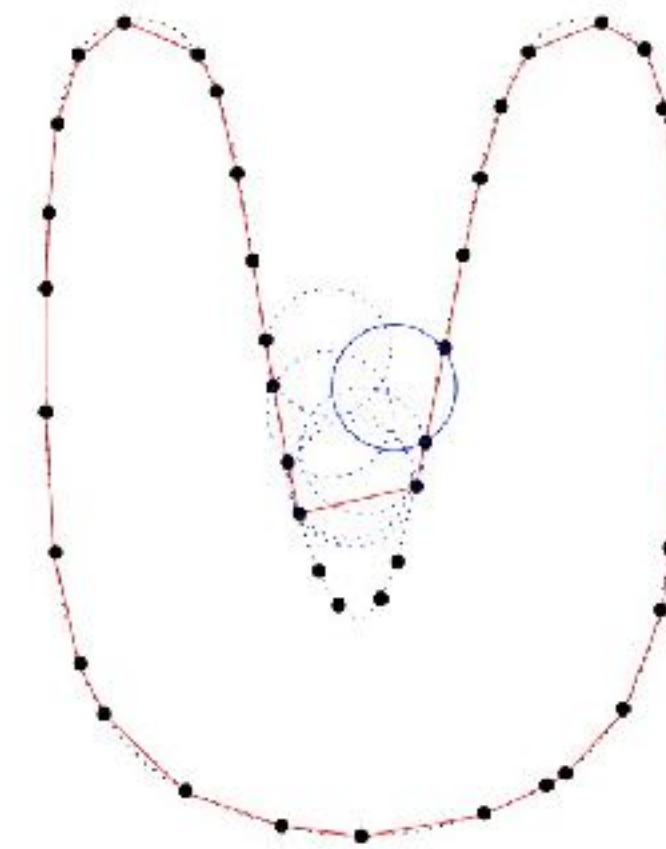
- Appropriate radius (a)



(a)



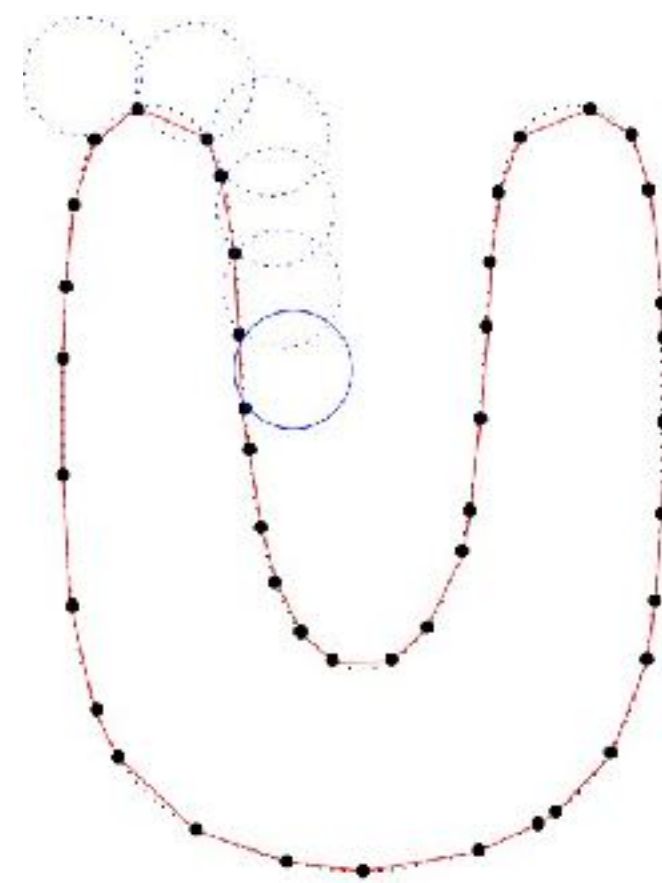
(b)



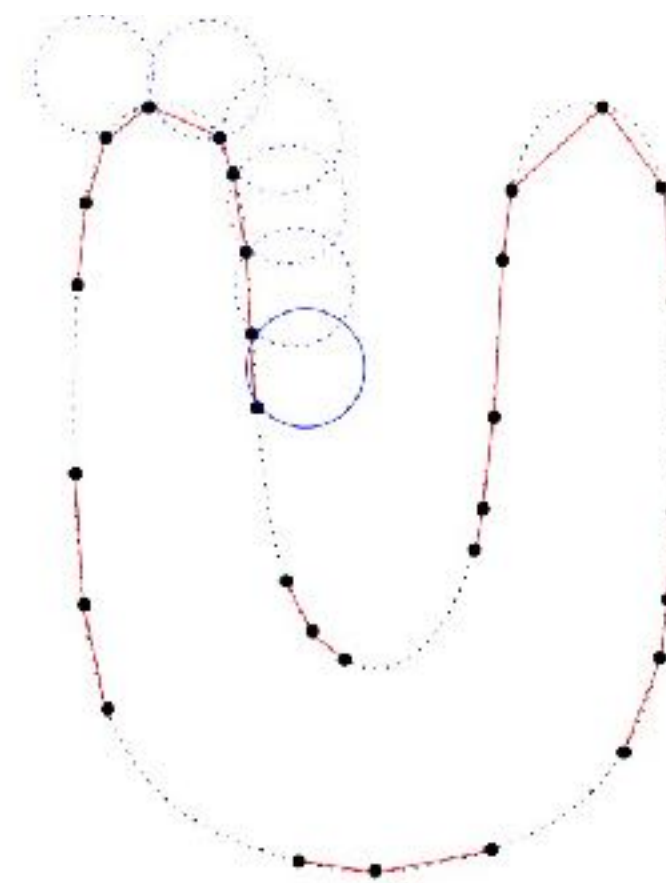
(c)

Radius ρ Matters

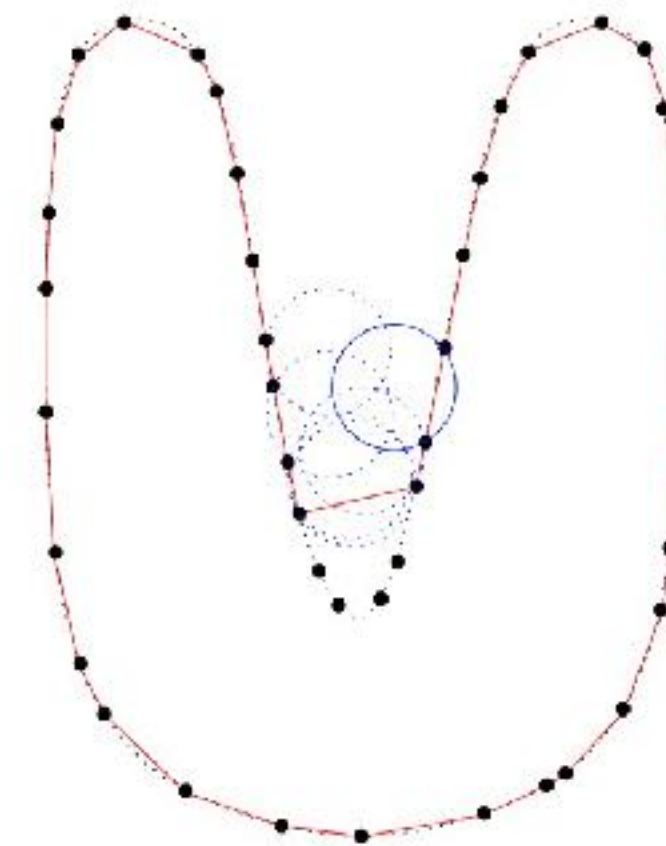
- Appropriate radius (a)
- Radius too small: some of the edges will not be created, leaving holes. (b)



(a)



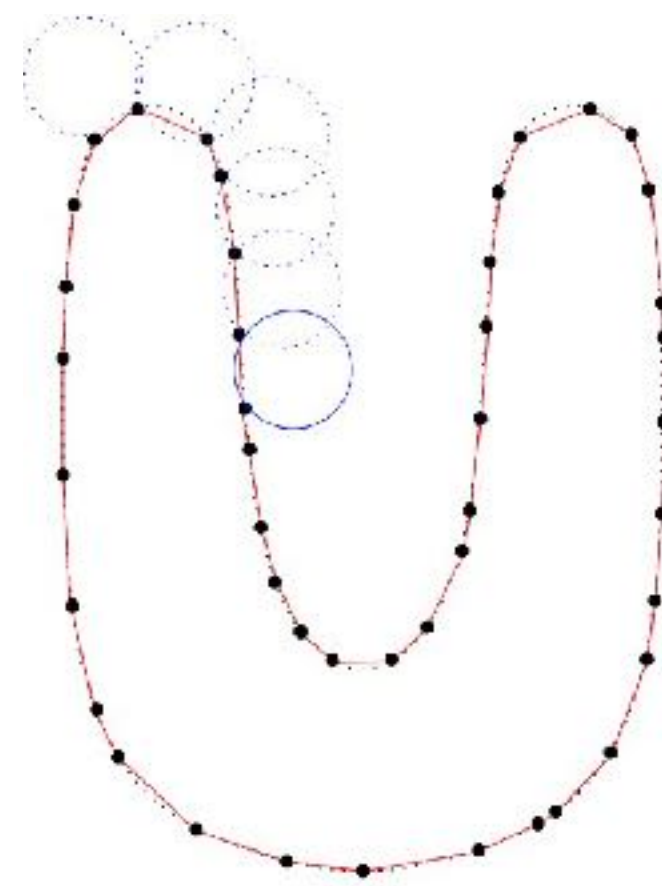
(b)



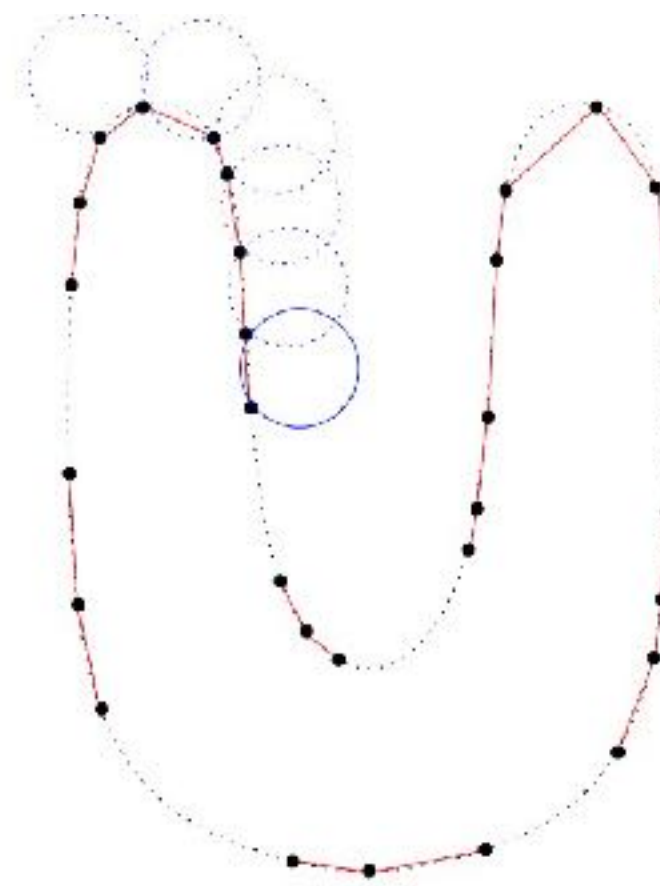
(c)

Radius ρ Matters

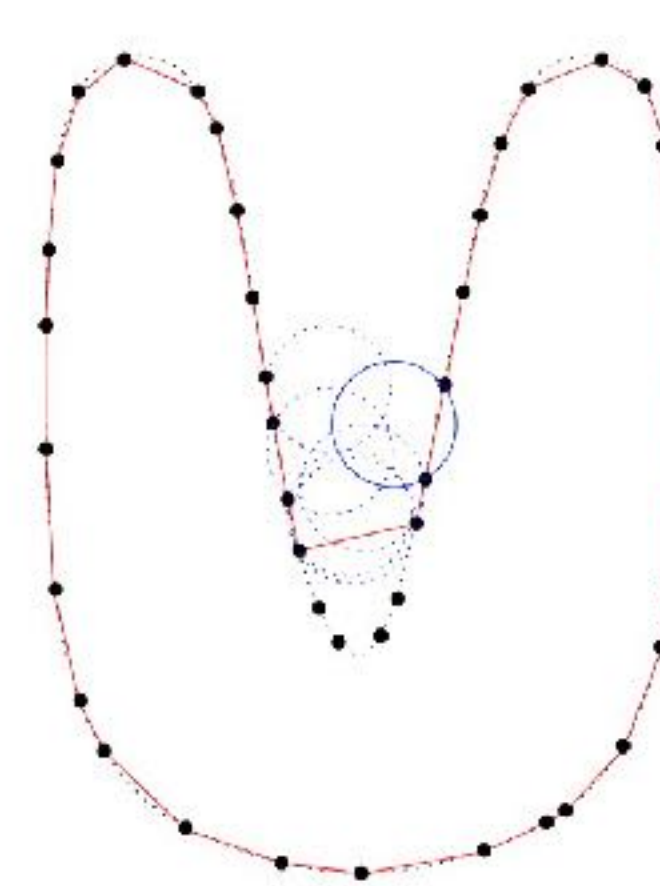
- Appropriate radius (a)
- Radius too small: some of the edges will not be created, leaving holes. (b)
- Large radius: some of the points will not be reached (when the curvature of the manifold is larger than $1/\rho$) (c)



(a)



(b)



(c)

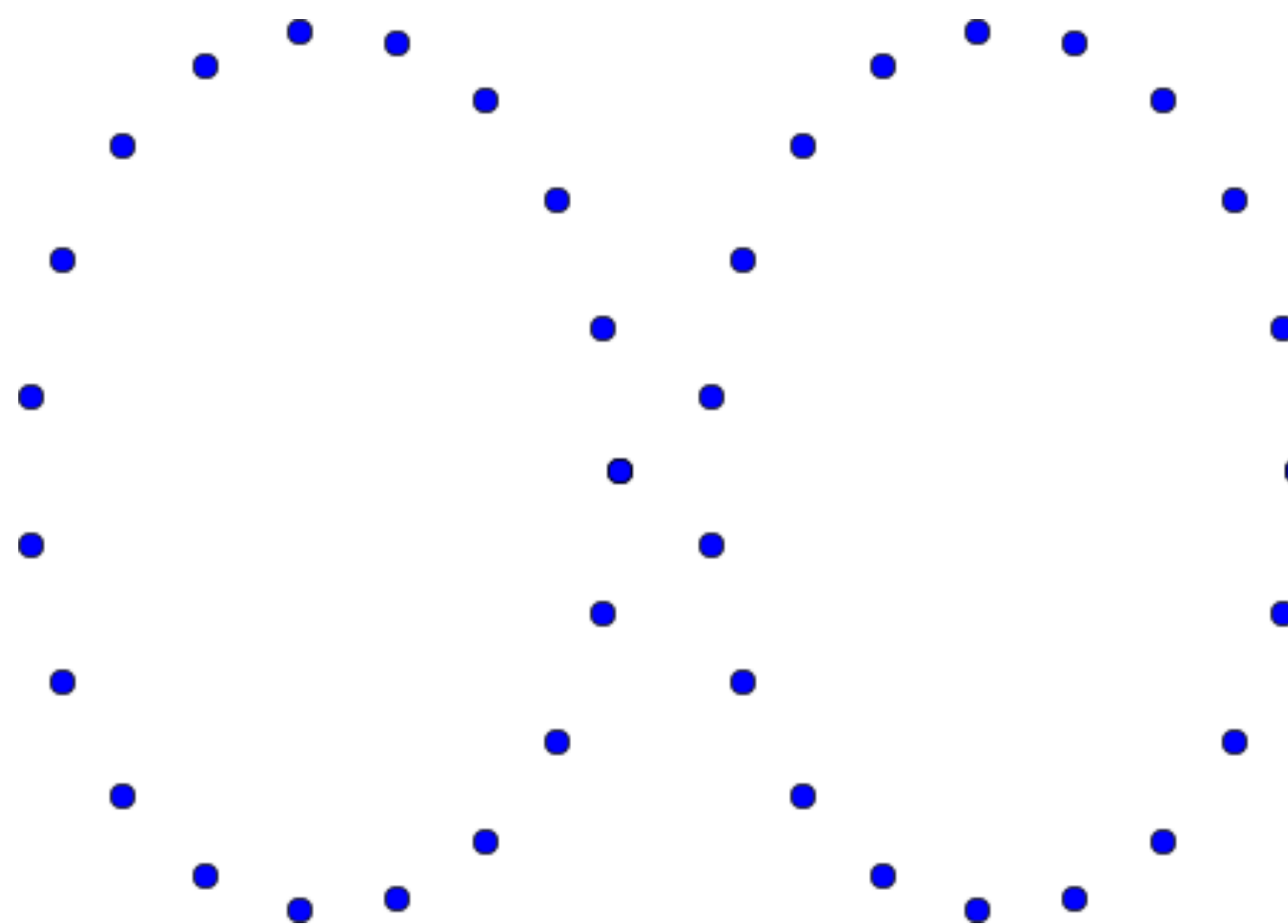
Iterative Approach

- Using multiple radius, iteratively connects the points.
- Small Radius capture high frequencies.
- Large Radius close holes.



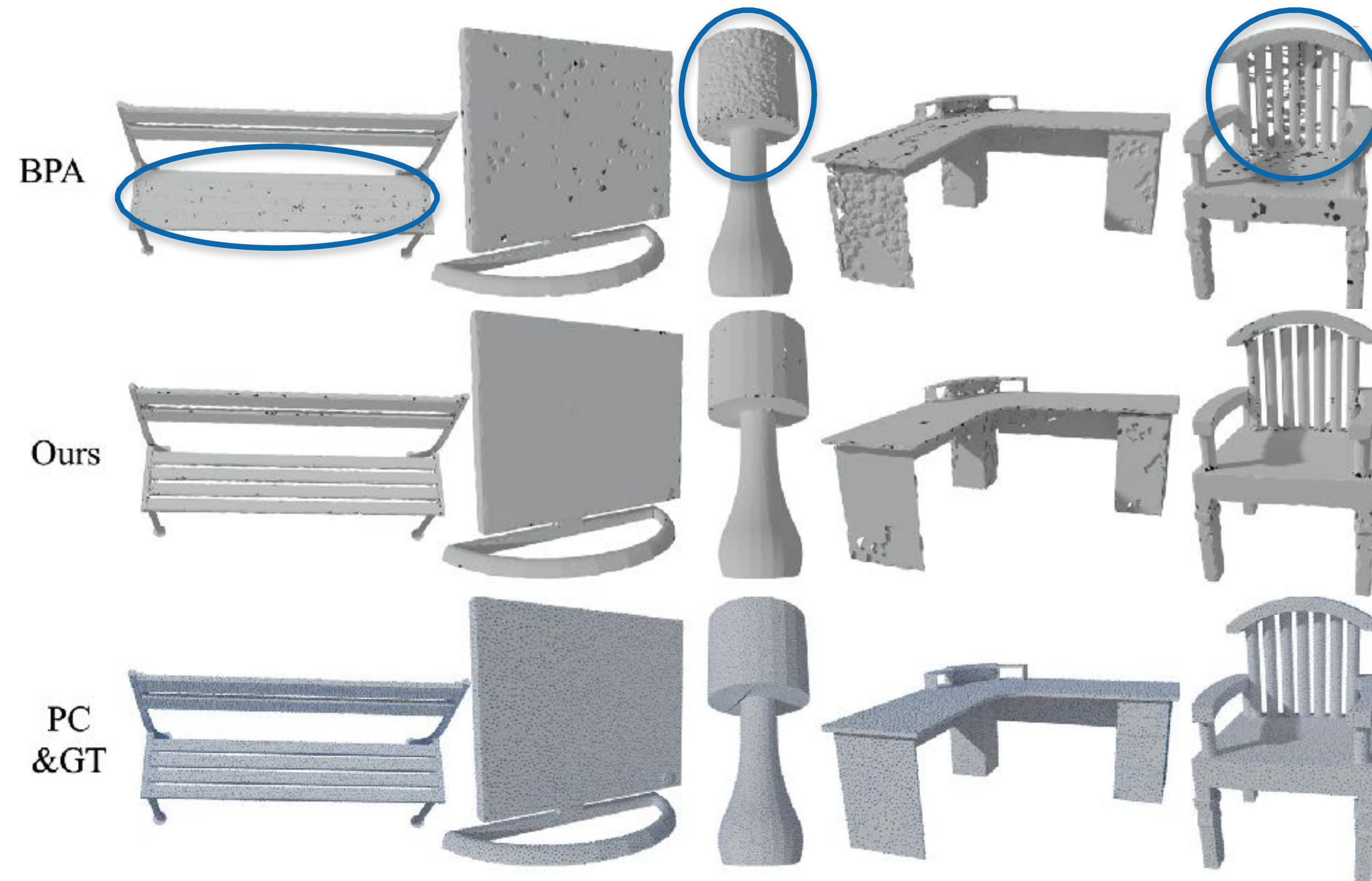
Ambiguous Structures

- Sometimes, defining a rule for structure estimation is hard
 - e.g., we tend to interpret the following point cloud as two disjoint ellipses; however, no ρ value allows us to separate them



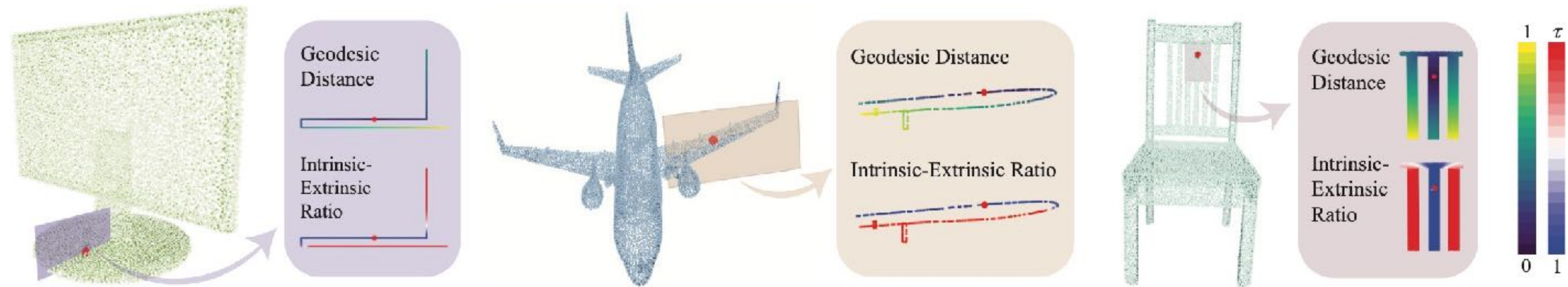
Ambiguous Structures

- Traditional Rule-based methods cannot handle ambiguous structures (e.g., thin structures & adjacent parts).

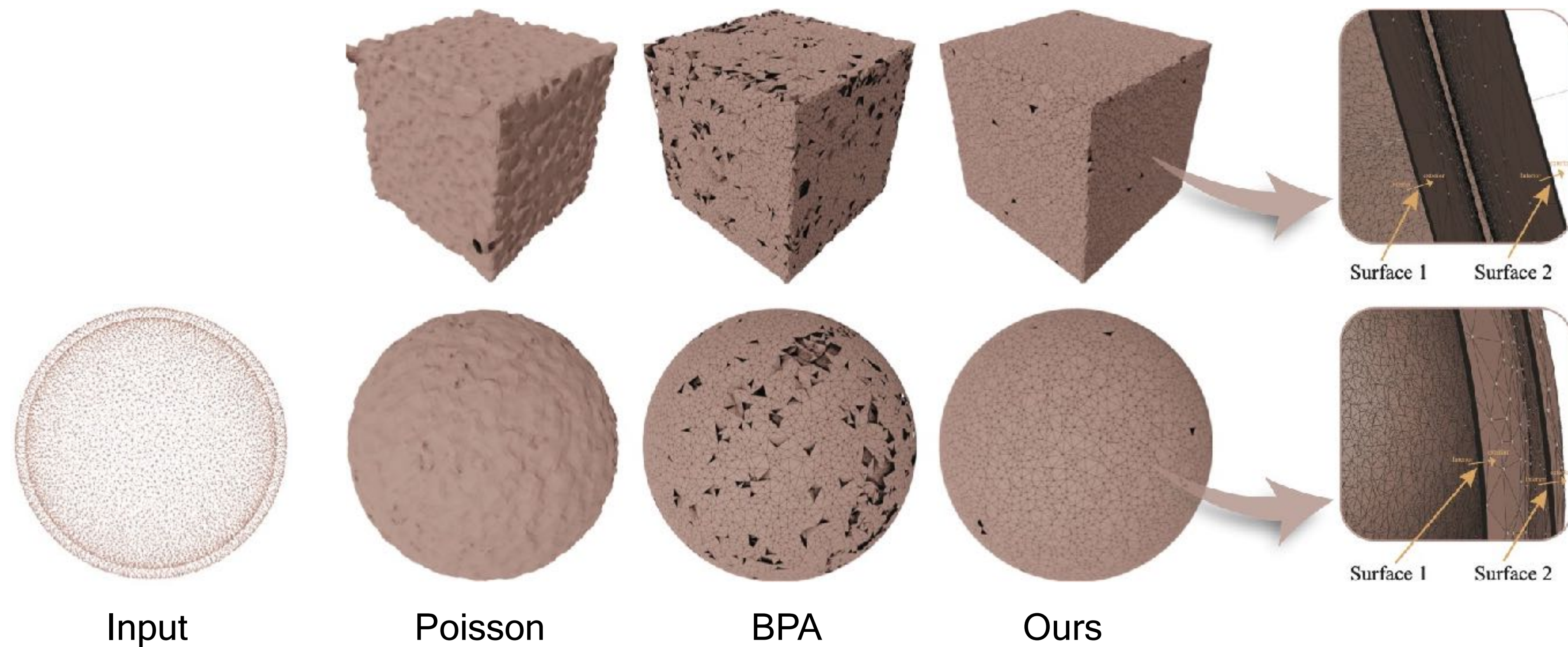


Learning-Based Method

- Train a network to filter out incorrect connections.
- Utilize the Intrinsic-Extrinsic Ratio to guide the training.



Learning-Based Method



Pros and Cons

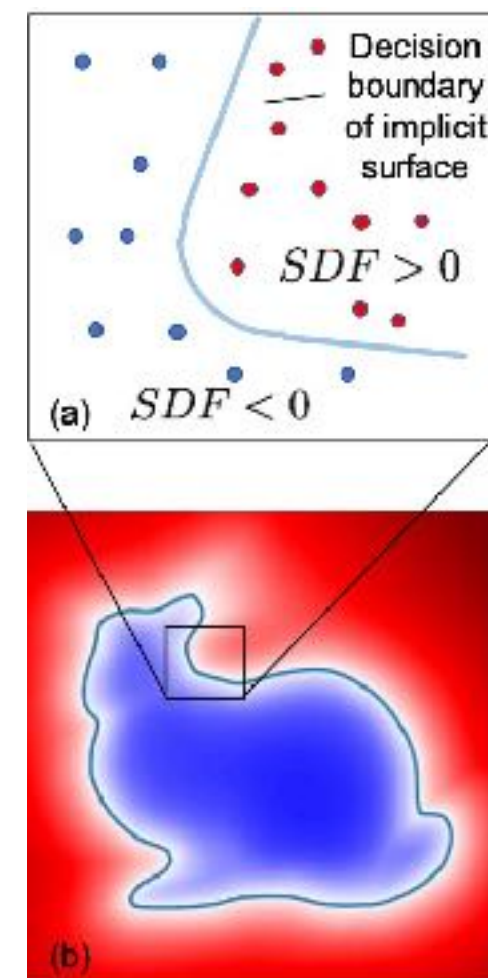
- Pros:
 - Linear complexity (fast)
 - No dependence on normals
- Cons:
 - Can lead to non-manifold situations, and no water-tight guarantee
- Regarding robustness:
 - Learning can improve the robustness
 - However, current learning-based method would still not work when the sampling density is low

Outline

- Problem Definition
- Explicit Algorithms
- *Implicit Algorithms*

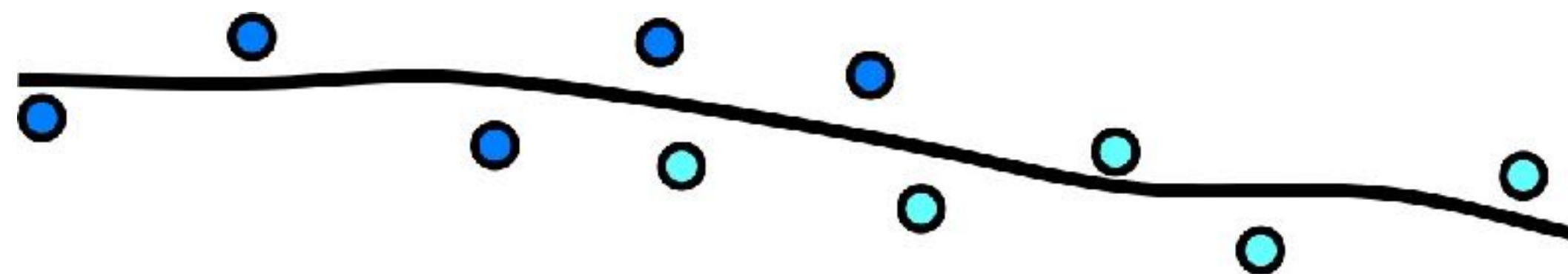
Implicit Field Function

- Interior: $F(x, y, z) < 0$
- Exterior: $F(x, y, z) > 0$
- Surface: $F(x, y, z) = 0$ (zero set, zero iso-surface)
- Example implementation:
 - SDF: $F(x, y, z) = \text{distance to the surface}$



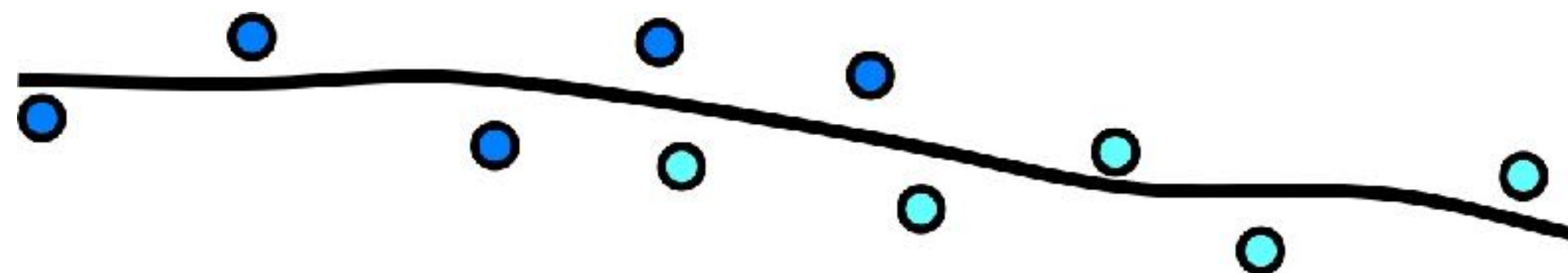
Implicit Meshing Algorithm

- Two basic steps:
 1. Estimate an implicit field function from data
 2. Extract the zero iso-surface



Implicit Meshing Algorithm

- Two basic steps:
 1. **Estimate an implicit field function from data**
 2. Extract the zero iso-surface



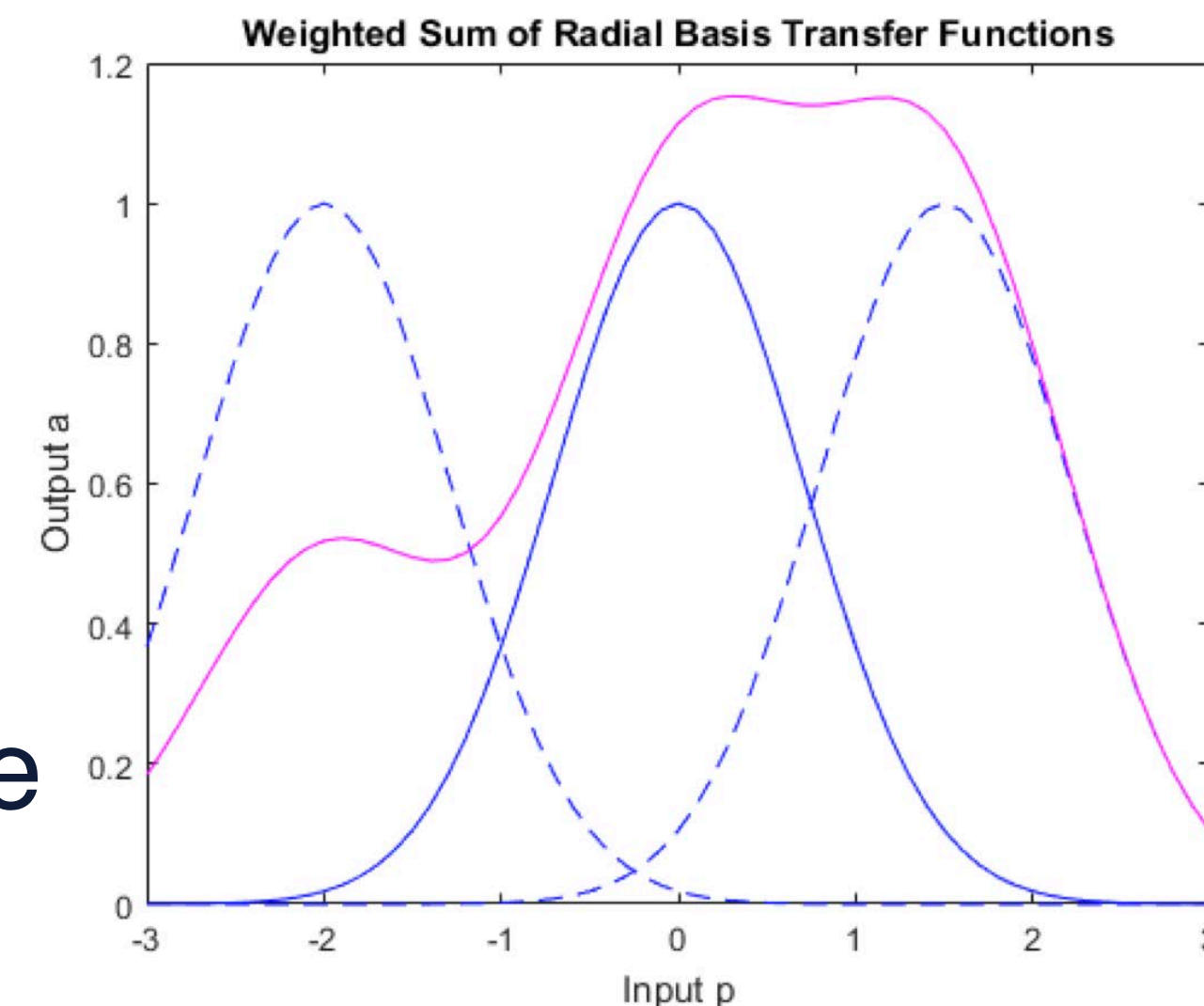
Radial Basis Functions

- Radial basis functions (RBF) $\phi_c(\mathbf{x})$: function value depends only on the distance from a center point c
- Use a weighted sum of radial basis functions to approximate the shape:

$$\phi_c(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{c}\|)$$

$$f(\mathbf{x}) = \sum_{i=1}^n \omega_i \phi(\|\mathbf{x} - \mathbf{x}_i\|) + p(x)$$

where p is a polynomial of low degree



Constraints: Avoiding Non-Trivial Solutions

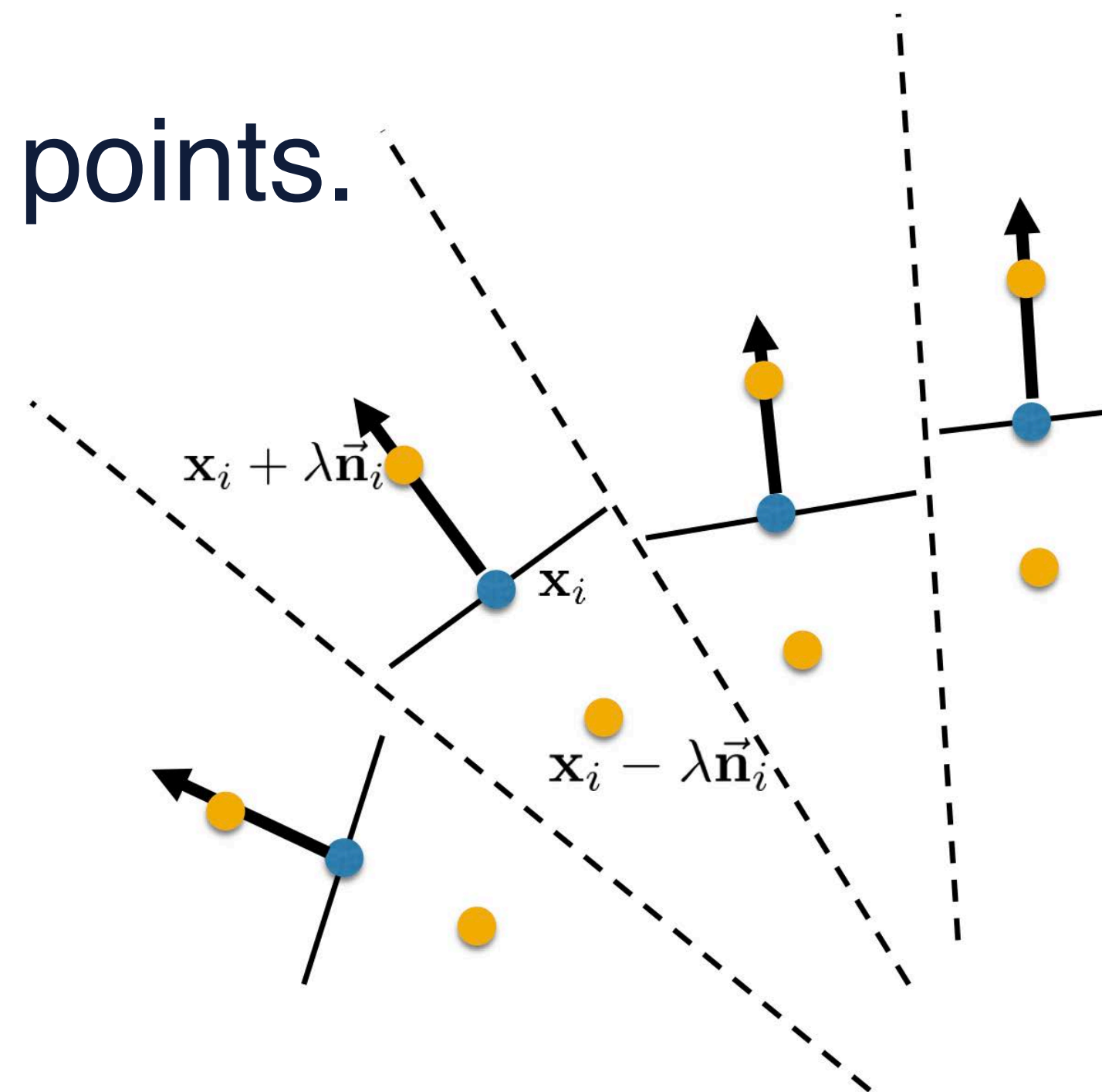
- Only force input points \mathbf{x}_i to have zero value
 $f(\mathbf{x}_i) = 0$ is not enough, it may get the trivial solution
 $f(\mathbf{x}) \equiv 0$

- Use normal to add off-surface points.

$$f(\mathbf{x}_i) = 0$$

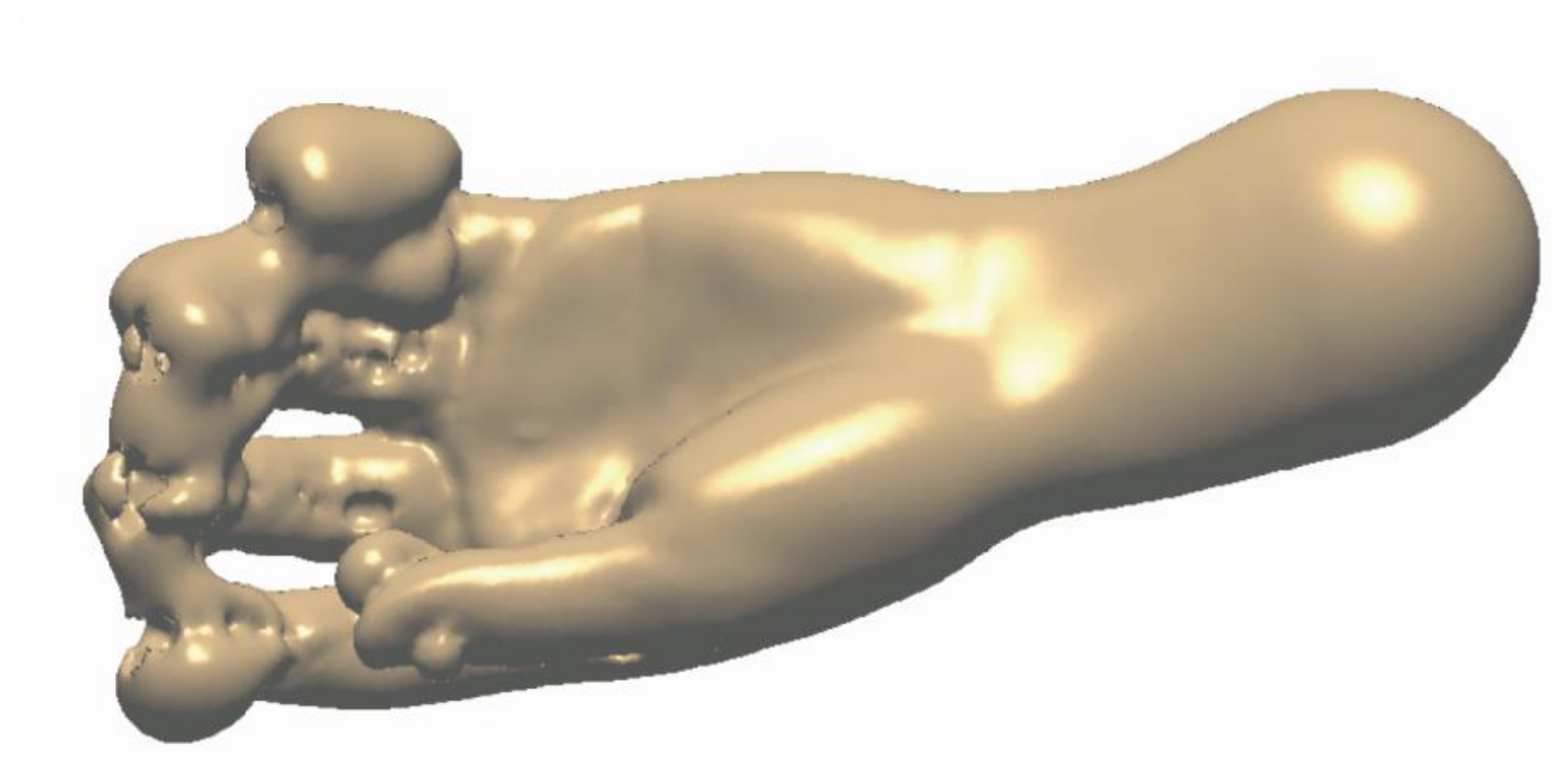
$$f(\mathbf{x}_i + \lambda \vec{\mathbf{n}}_i) = \lambda$$

$$f(\mathbf{x}_i - \lambda \vec{\mathbf{n}}_i) = -\lambda$$

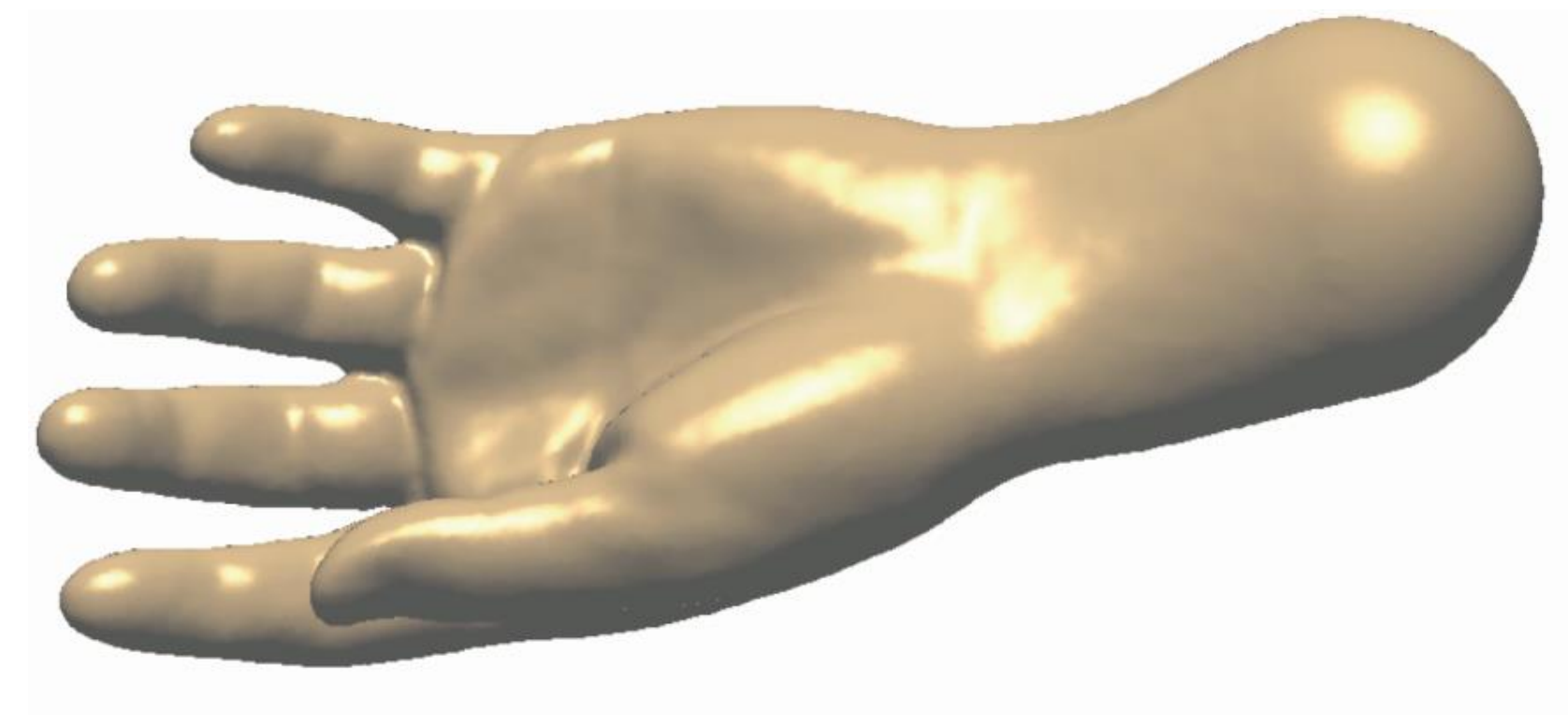


Off-surface Points

- Select an offset such that off-surface points do not intersect other parts of the surface
- Adaptive offset: the off-surface point is constructed so that the closest point is the surface point that generated it



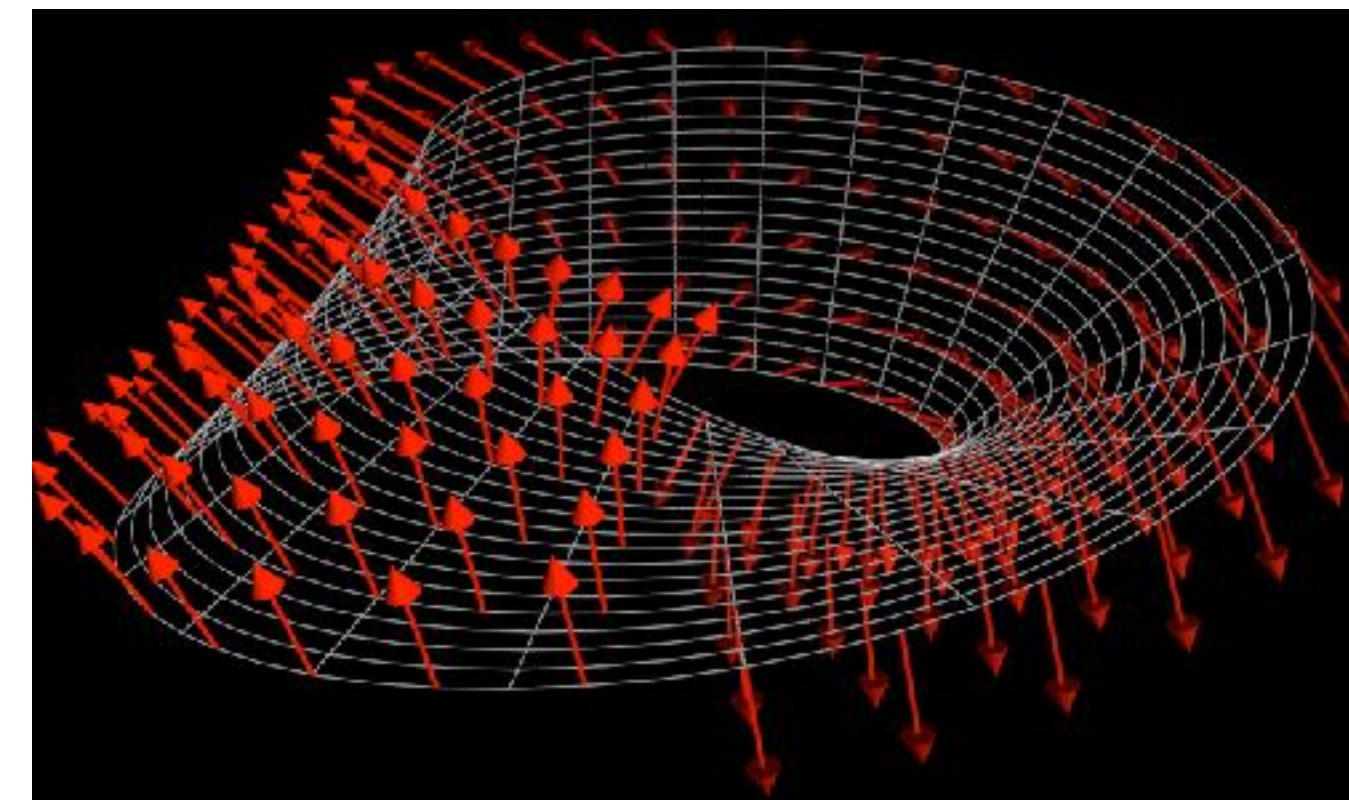
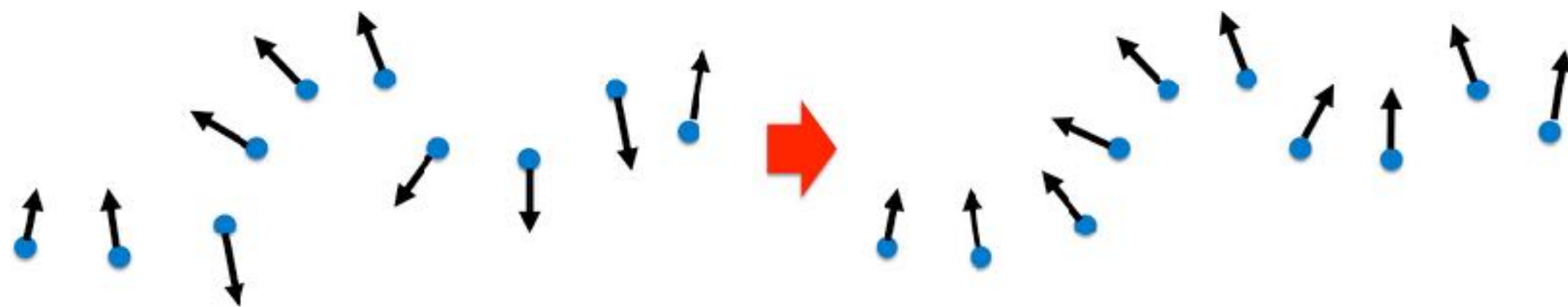
FIXED OFFSET



ADAPTIVE OFFSET

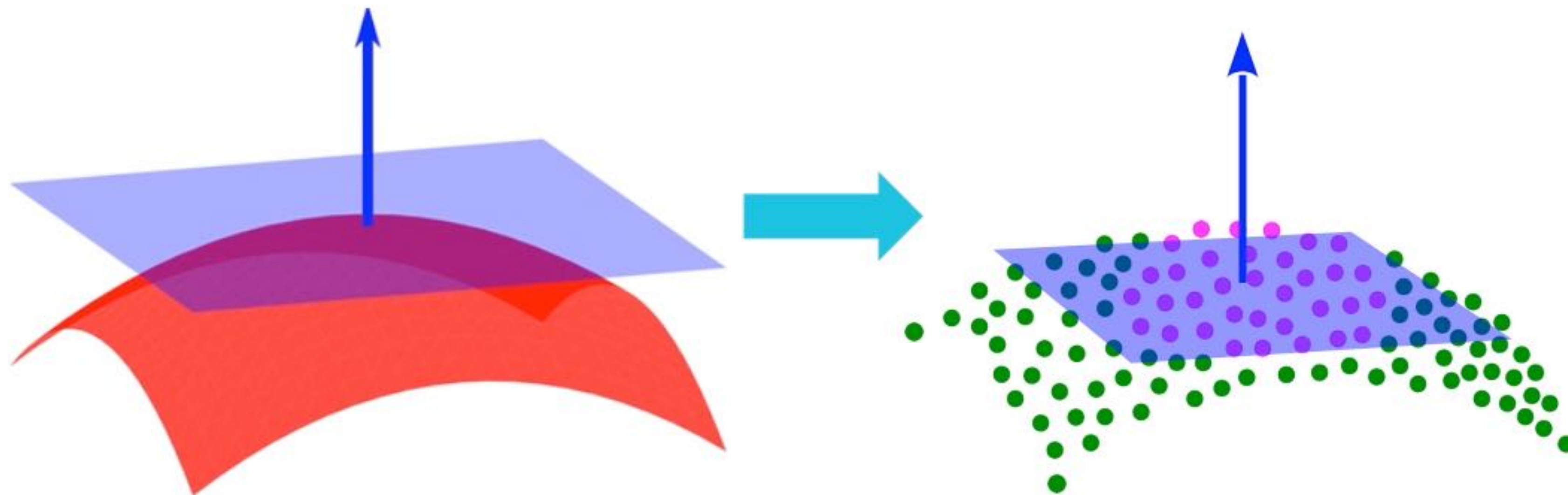
Consistent Normals are Required

- We just assumed consistent normals
- Normal is typically required to build watertight meshes
- However, obtaining consistent normal orientation is non-trivial.



Recall: Normal Computation in Point Cloud

- The normal of a point cloud can be computed through PCA over a local neighborhood



Determine Normal Orientation

- Build graph connecting neighboring points
 - Edge (ij) exists if $\mathbf{p}_i \in \text{kNN}(\mathbf{p}_j)$ or $\mathbf{p}_j \in \text{kNN}(\mathbf{p}_i)$
- Propagate normal orientation through graph
 - For neighbors $\mathbf{p}_i, \mathbf{p}_j$: Flip \mathbf{n}_j if $\mathbf{n}_i^T \mathbf{n}_j < 0$
 - Fails at sharp edges/corners
- Propagate along “safe” paths (parallel normals)
 - Minimum spanning tree with angle-based edge weights
 $w_{ij} = 1 - |\mathbf{n}_i^T \mathbf{n}_j|$

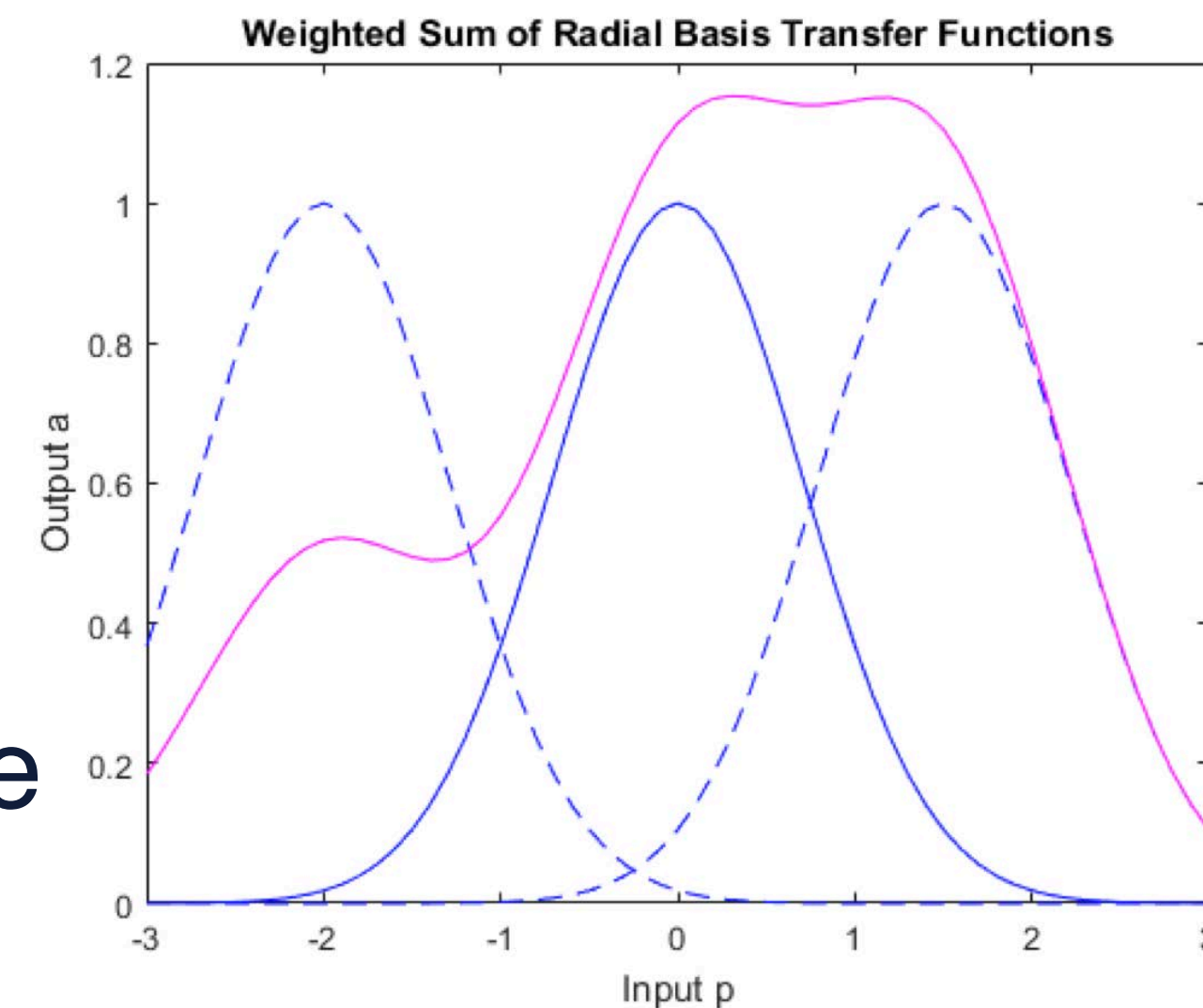
Radial Basis Functions

- Radial basis functions (RBF) $\phi_c(\mathbf{x})$: function value depends only on the distance from a center point c
- Use a weighted sum of radial basis functions to approximate the shape:

$$\phi_c(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{c}\|)$$

$$f(\mathbf{x}) = \sum_{i=1}^n \omega_i \phi(\|\mathbf{x} - \mathbf{x}_i\|) + p(x)$$

where p is a polynomial of low degree



Estimate Parameters

- Variables:
 - $n + l$ variables on ω_i (RBF coef.) and c_i (polynomial coef.)
- Solve a linear system of $3n + l$ equations
 - $3n$: from the point, inside, and outside
 - l : additional constraints to guarantee the smoothness and integrability of f

$$\begin{pmatrix} A & P \\ P^\top & 0 \end{pmatrix} \begin{pmatrix} \omega \\ c \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix}$$

$$\begin{aligned} A_{i,j} &= \phi(|x_i - x_j|), & i, j &= 1, \dots, N, \\ P_{i,j} &= p_j(x_i), & i &= 1, \dots, N, \quad j = 1, \dots, \ell. \end{aligned}$$

Implementation Details

- Triharmonic basis functions: $\phi(r) = r^3$
 - Need its extrapolation ability
 - Should **not** use RBF with compact or local support (e.g., Gaussian density)
- Polynomial: third-order is practically good

Implementation Details

- Do not need to use all the input data points as RBF centers
 - Use a greedy algorithm to select a subset of points
- Noisy data
 - Exact interpolation?
 - Treat the linear equation as solving a linear square problem and add a smoothness term

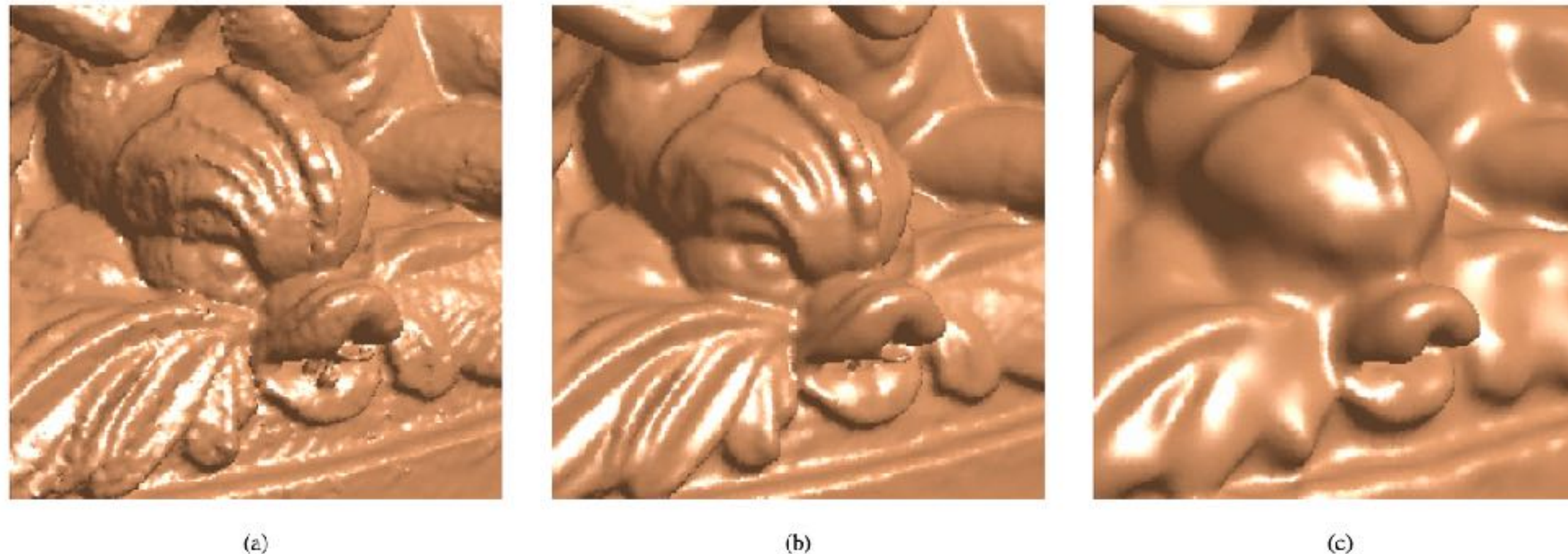


Figure 9: (a) Exact fit, (b) medium amount of smoothing applied (the RBF approximates at data points), (c) increased smoothing.

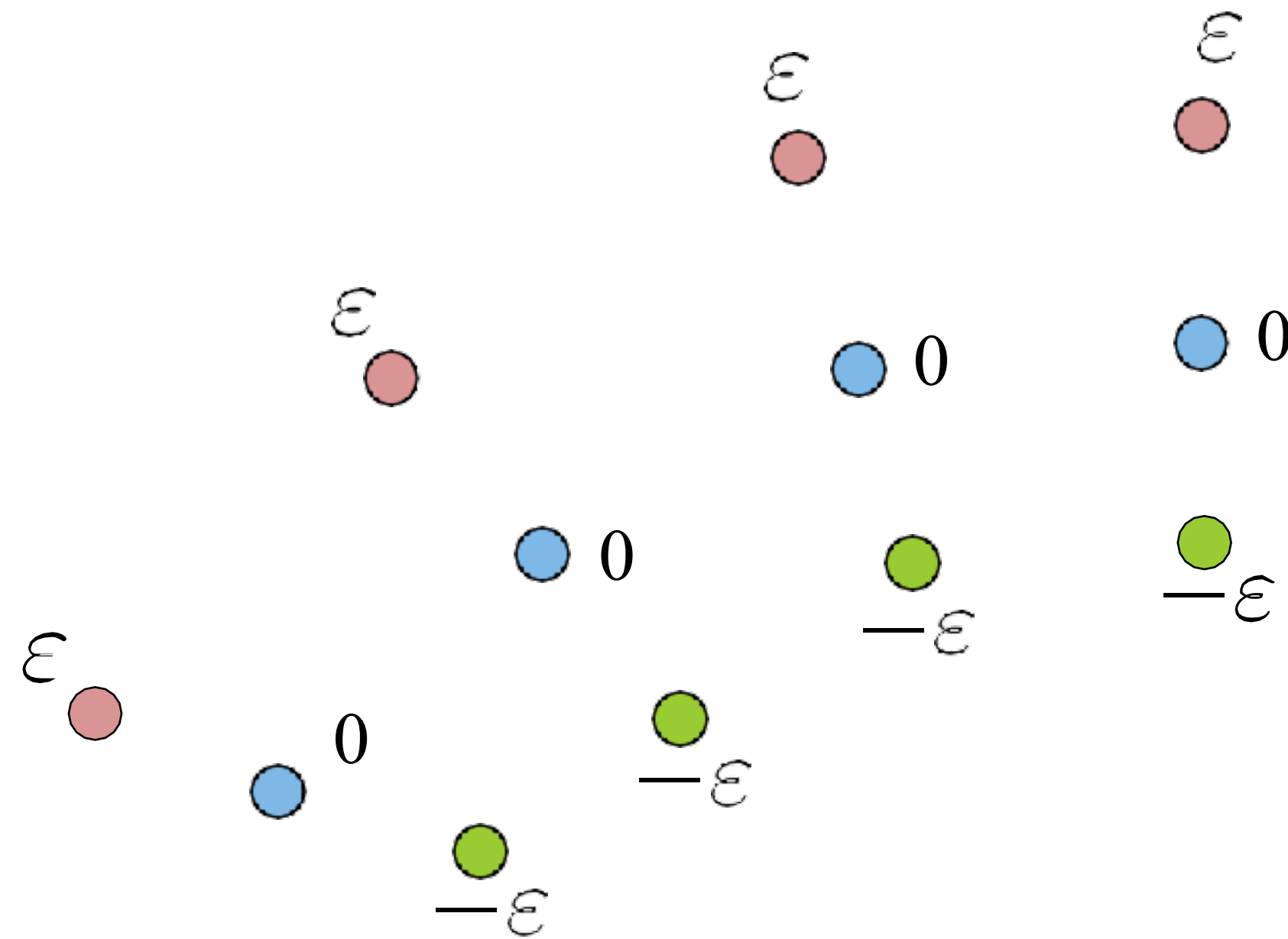
RBF Discussion

- Pros: Global definition
 - Single function
 - Globally optimal
- Cons: Global definition
 - Global optimization – slow
 - Why is global better?

Moving Least Squares (MLS)

Do purely **local** approximation of the SDF

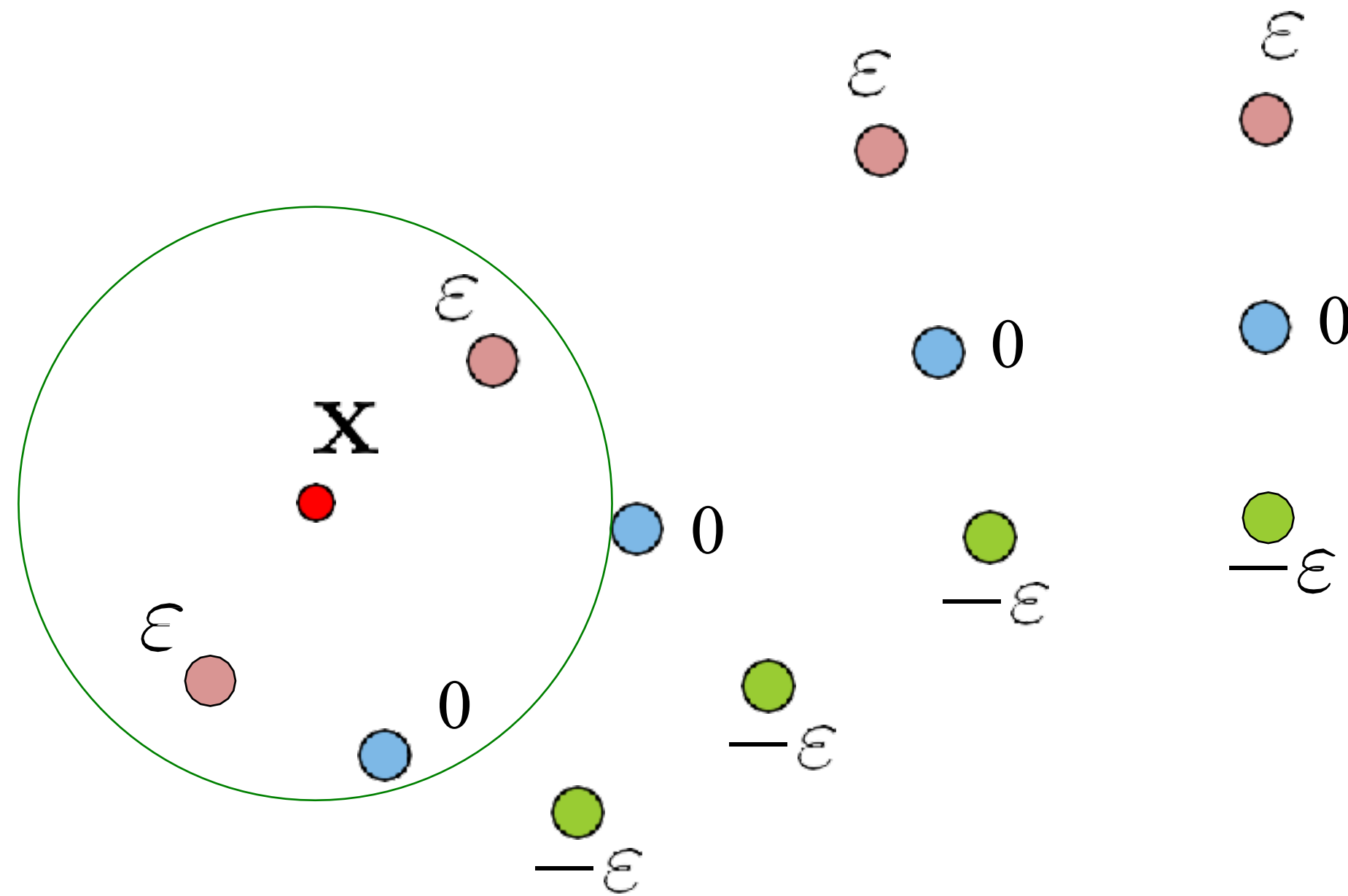
Weights change depending on where we are evaluating



Moving Least Squares (MLS)

Do purely **local** approximation of the SDF

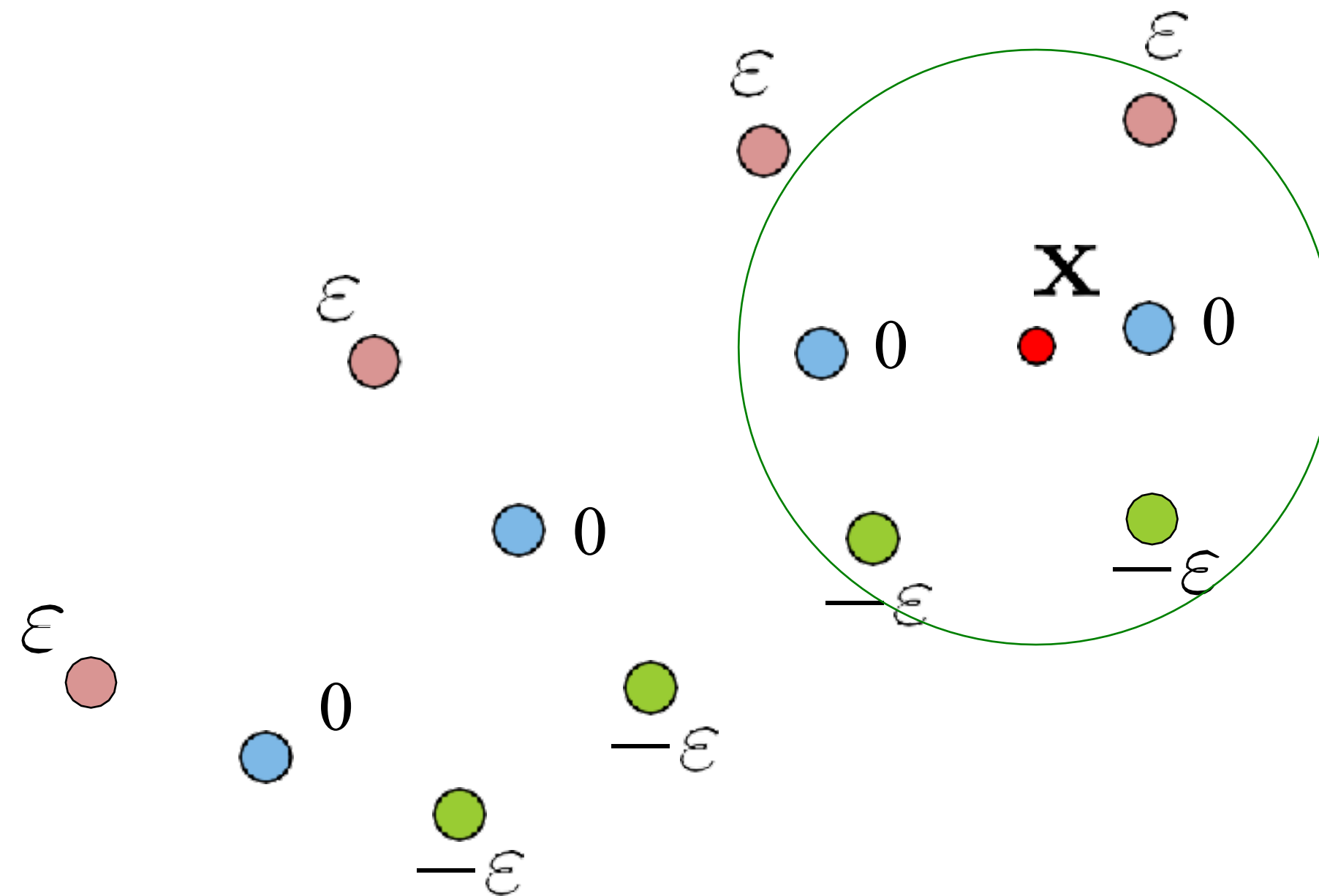
Weights change depending on where we are evaluating



Moving Least Squares (MLS)

Do purely **local** approximation of the SDF

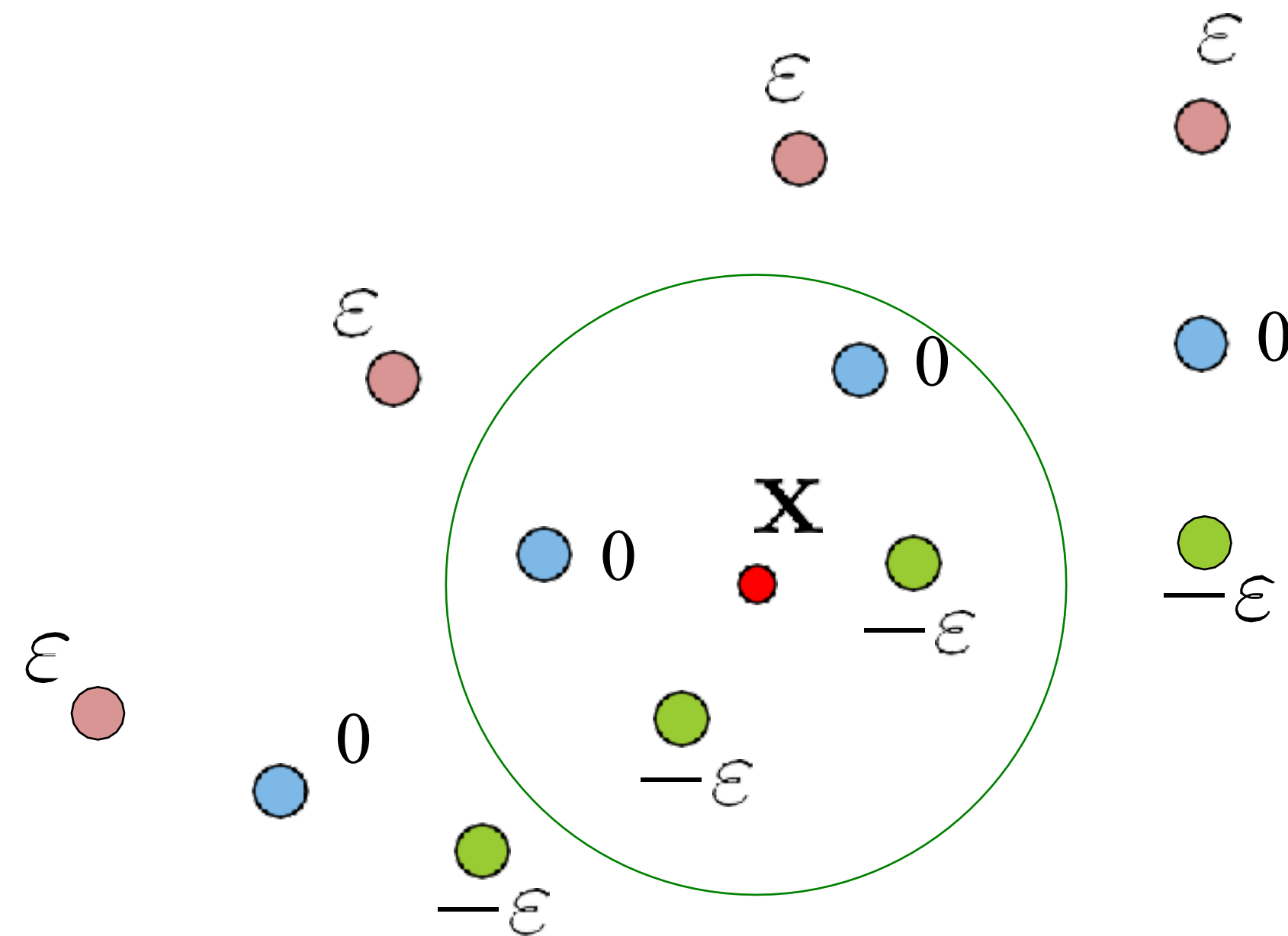
Weights change depending on where we are evaluating



Moving Least Squares (MLS)

Do purely **local** approximation of the SDF

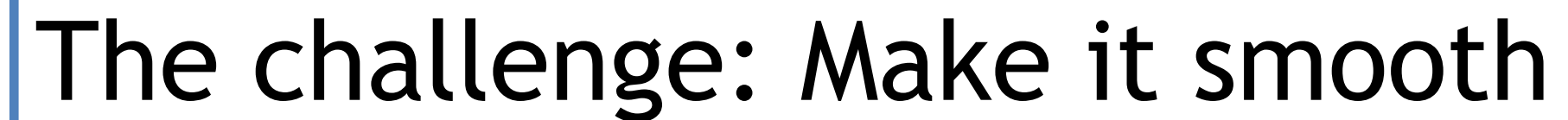
Weights change depending on where we are evaluating



Moving Least Squares (MLS)

Do purely **local** approximation of the SDF

Weights change depending on where we are evaluating



The challenge: Make it smooth

Least-Squares Approximation

Polynomial least-squares approximation

$$f \in \Pi_k^3 : f(x, y, z) = a_0 + a_1x + a_2y + a_3z + a_4x^2 + a_5xy + \dots + a_*z^k$$

$$f(\mathbf{x}) = \mathbf{b}(\mathbf{x})^T \mathbf{a}$$

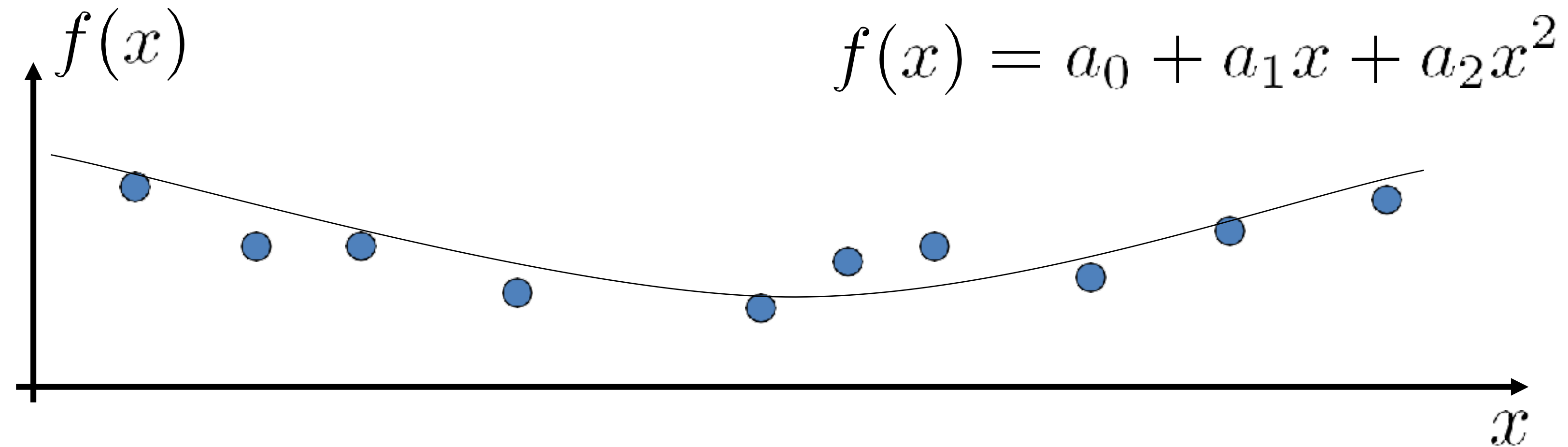
$$\mathbf{a} = (a_1, a_2, \dots, a_*)^T, \quad \mathbf{b}(\mathbf{x})^T = (1, x, y, z, x^2, xy, \dots, z^k)$$

Find \mathbf{a} that minimizes sum of squared differences

$$\operatorname{argmin}_{\mathbf{a}} \sum_{m=0}^{N-1} (\mathbf{b}(\mathbf{c}_m)^T \mathbf{a} - d_m)^2$$

MLS – 1D Example

- Global approximation in Π_2^1



$$\operatorname{argmin}_{\mathbf{a}} \sum_{m=0}^{N-1} (\mathbf{b}(\mathbf{c}_m)^T \mathbf{a} - d_m)^2$$

Moving Least-Squares Approximation

Polynomial least-squares approximation

$$f \in \Pi_k^3 : f(x, y, z) = a_0 + a_1x + a_2y + a_3z + a_4x^2 + a_5xy + \dots + a_*z^k$$

$$f(\mathbf{x}) = \mathbf{b}(\mathbf{x})^T \mathbf{a}$$

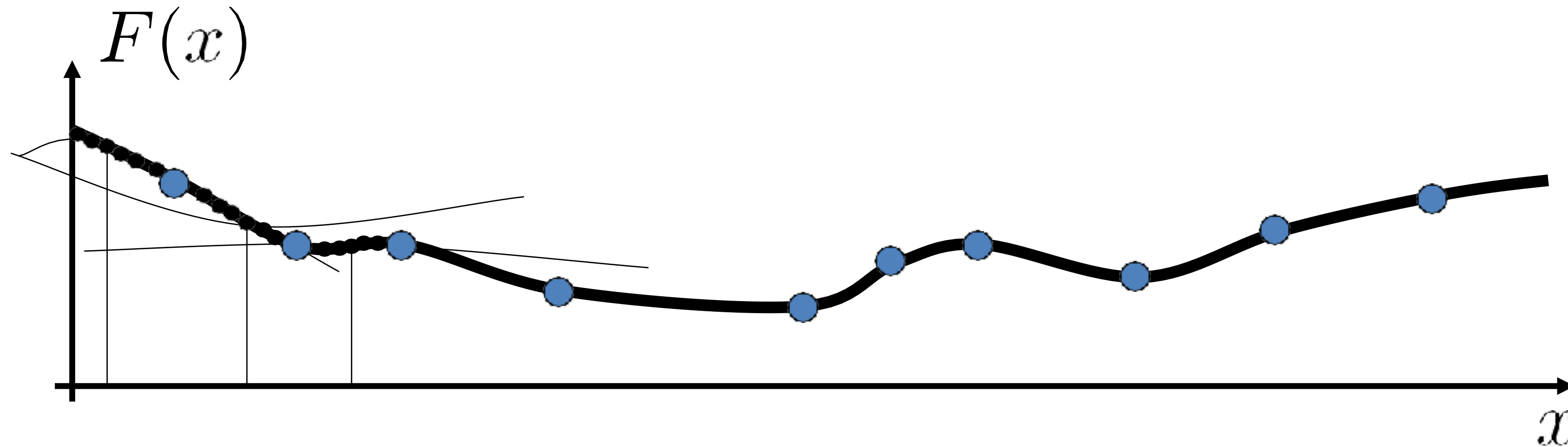
$$\mathbf{a} = (a_1, a_2, \dots, a_*)^T, \quad \mathbf{b}(\mathbf{x})^T = (1, x, y, z, x^2, xy, \dots, z^k)$$

Find \mathbf{a} that minimizes **weighted** sum of squared differences

$$\mathbf{a}_{\mathbf{x}} = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{m=0}^{N-1} \theta(\|\mathbf{x} - \mathbf{c}_m\|) (\mathbf{b}(\mathbf{c}_m)^T \mathbf{a} - d_m)^2$$

MLS – 1D Example

- MLS approximation using functions in Π_2^1



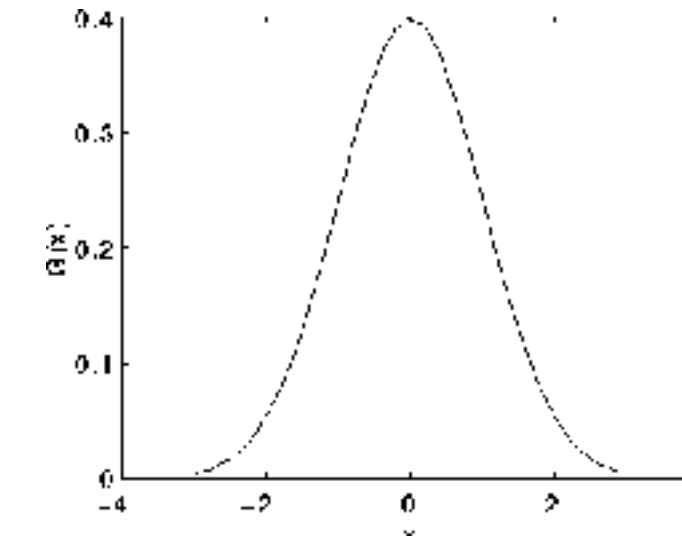
$$F(x) = f_x(x), \quad f_x = \operatorname{argmin}_{f \in \Pi_2^1} \sum_{m=0}^{N-1} \theta(\|c_m - x\|) (f(c_m) - d_m)^2$$

Weight Functions

Gaussian

$$\theta(r) = e^{-\frac{r^2}{h^2}}$$

h is a smoothing parameter



Wendland function

$$\theta(r) = (1 - r/h)^4 (4r/h + 1)$$

Defined in $[0, h]$ and

$$\theta(0) = 1, \quad \theta(h) = 0, \quad \theta'(h) = 0, \quad \theta''(h) = 0$$

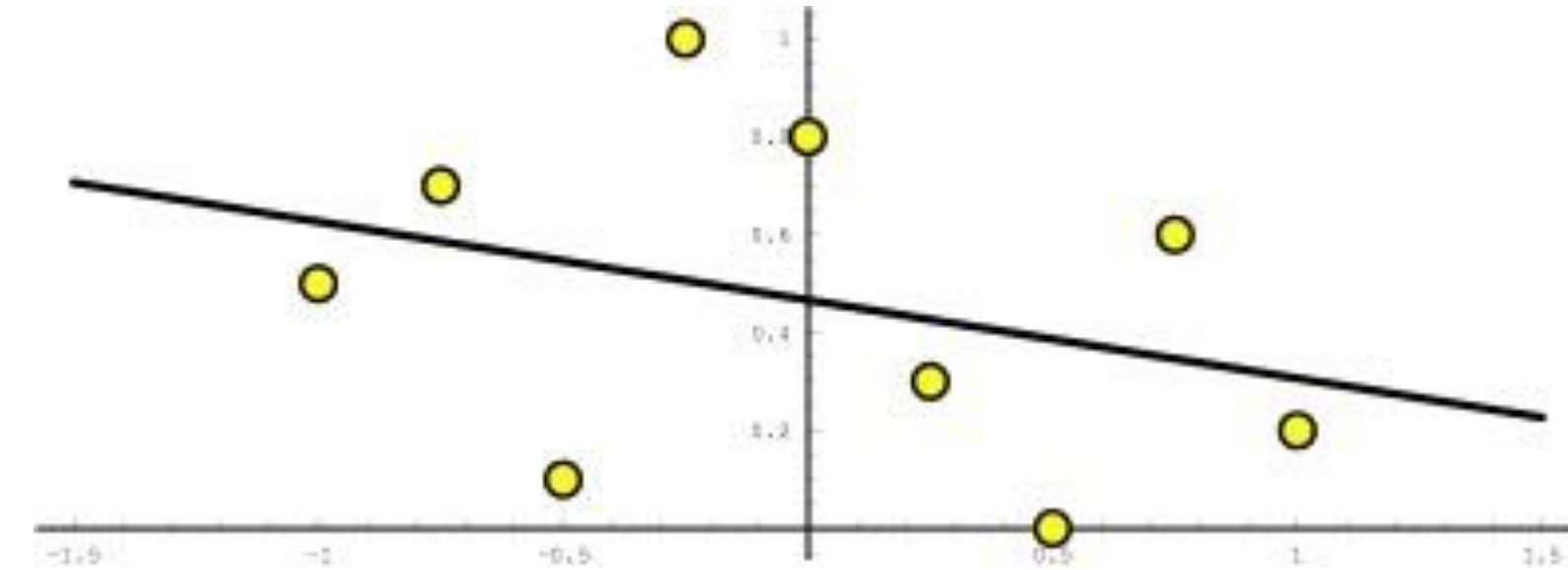
Singular function

$$\theta(r) = \frac{1}{r^2 + \varepsilon^2}$$

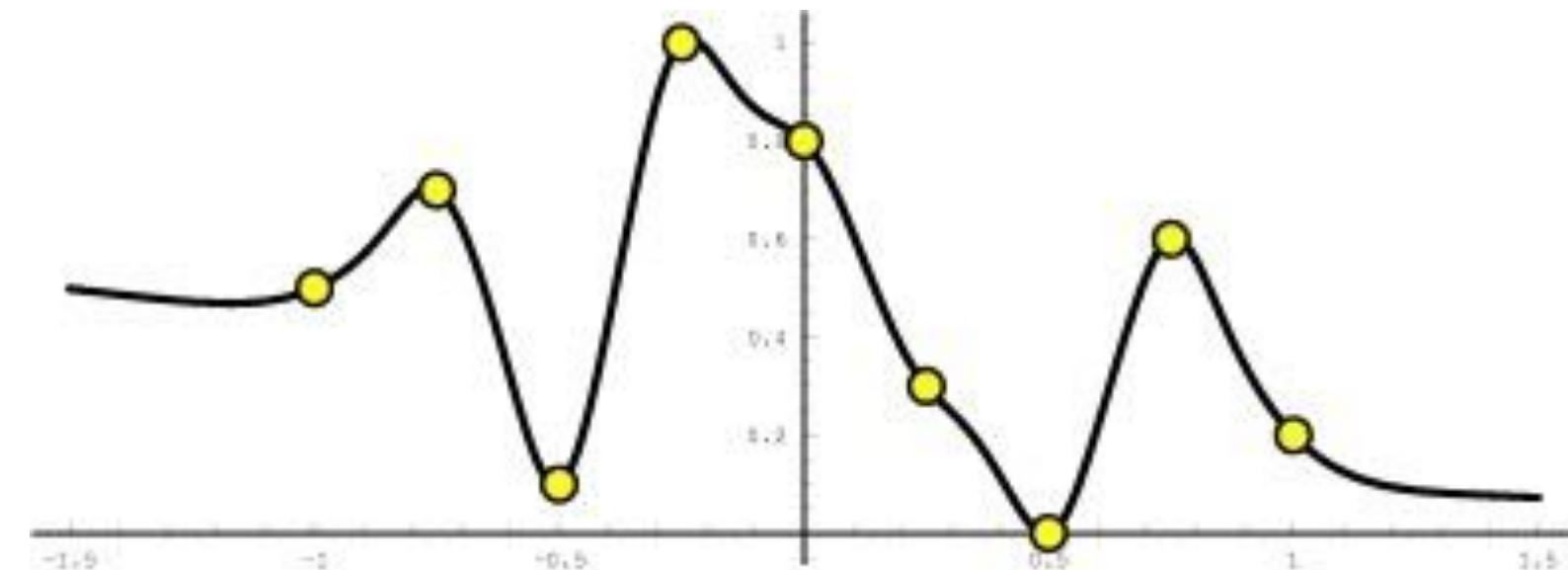
For small ε , weights are large near $r=0$ (interpolation)

Dependence on Weight Function

Global least squares with linear basis

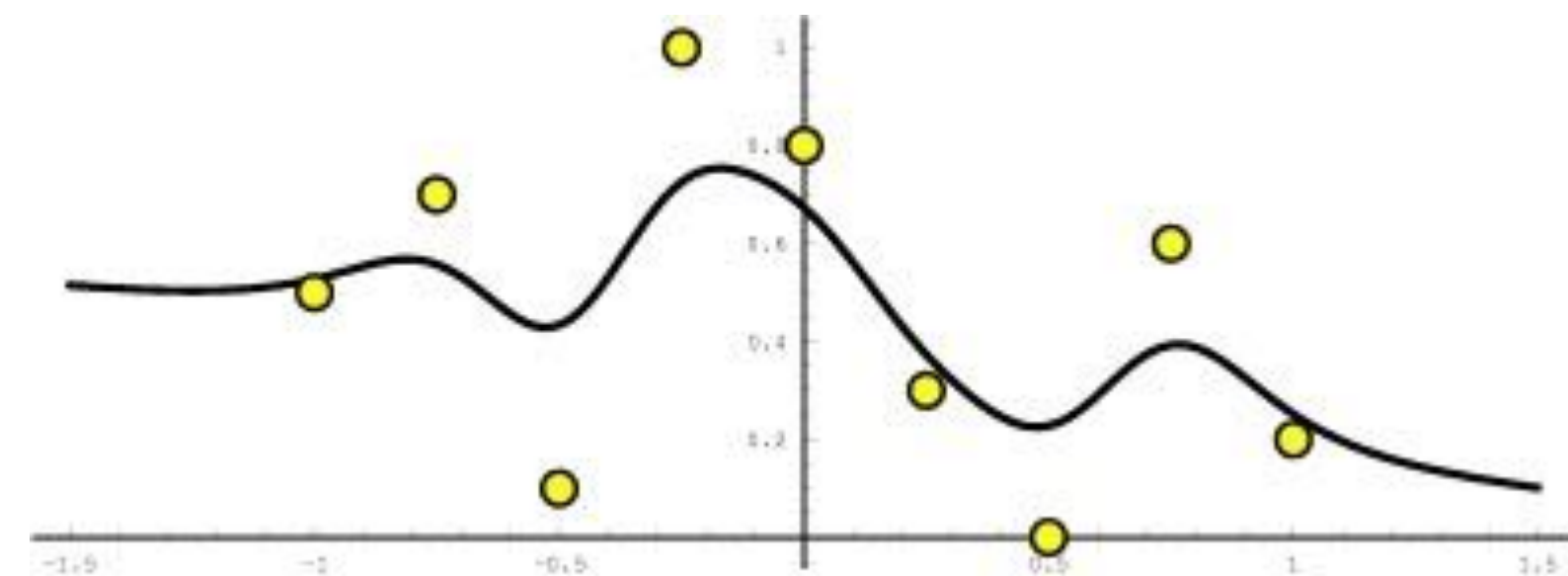


MLS with (nearly) singular weight function $\theta(r) = \frac{1}{r^2 + \varepsilon^2}$



MLS with approximating weight function

$$\theta(r) = e^{-\frac{r^2}{h^2}}$$



Dependence on Weight Function

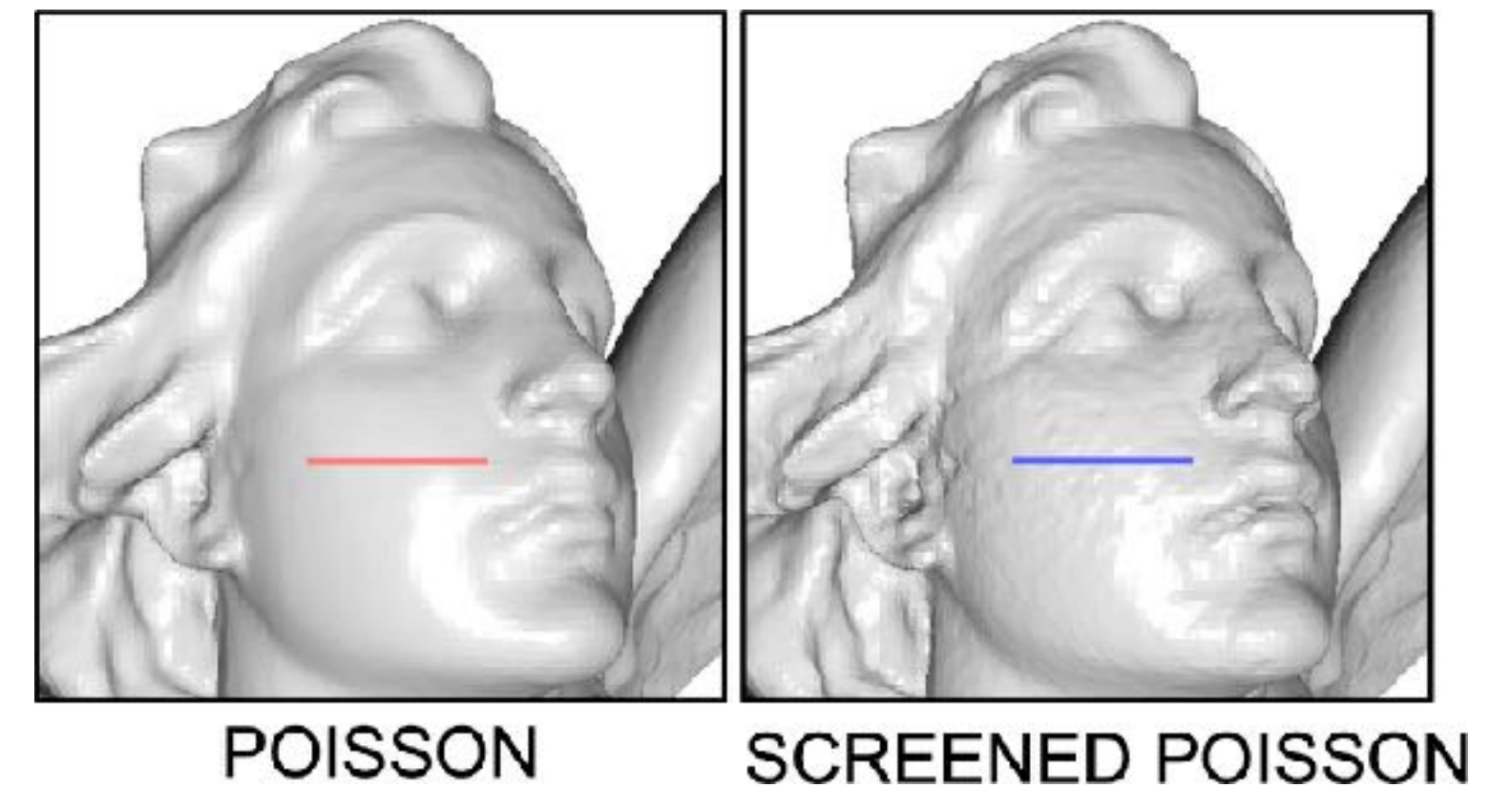
The MLS function F is continuously differentiable if and only if the weight function θ is continuously differentiable

In general, F is as smooth as θ

$$F(\mathbf{x}) = f_{\mathbf{x}}(\mathbf{x}), \quad f_{\mathbf{x}} = \operatorname{argmin}_{f \in \Pi_k^d} \sum_{m=0}^{N-1} \theta(\|\mathbf{c}_m - \mathbf{x}\|) (f(\mathbf{c}_m) - d_m)^2$$

More than RBF and MLS

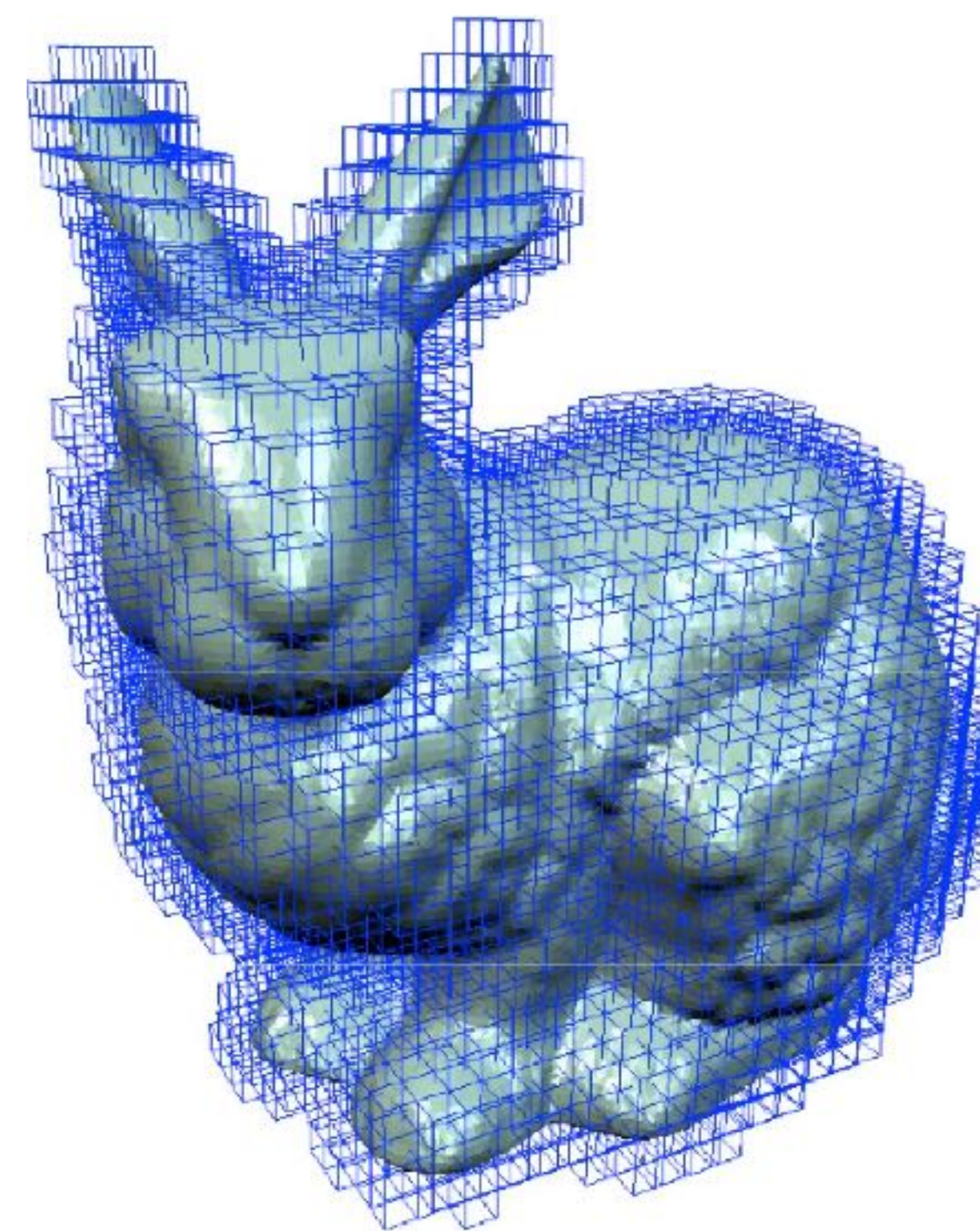
- Kazhdan M, Bolitho M, Hoppe H. “**Poisson surface reconstruction.**” ESGP, 2006.
 - Robust to noise, adapt to the sampling density
 - Over-smoothing
- Kazhdan M, Hoppe H. “**Screened poisson surface reconstruction.**” ToG, 2013.
 - Sharper reconstruction, faster
 - But it assumes clean data



Implicit Meshing Algorithm

- Two basic steps:
 1. Estimate an implicit field function from data
 - 2. Extract the zero iso-surface**

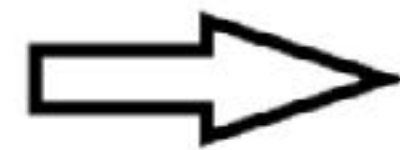
Input: a signed distance field
(Implicitly assumed knowing the
inside/outside of the shape,
often needs to be estimated with
normal information)



Classical Solution: 2D Marching Square

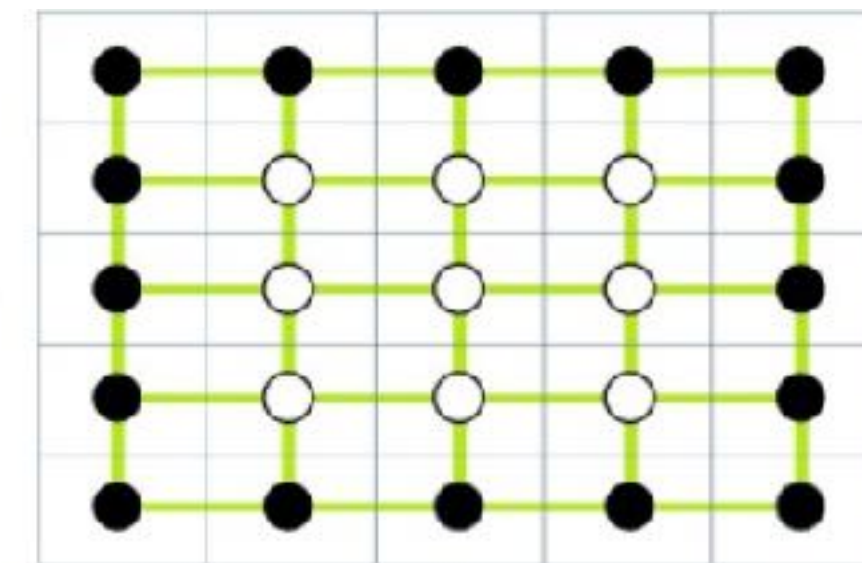
1	1	1	1	1
1	2	3	2	1
1	3	3	3	1
1	2	3	2	1
1	1	1	1	1

Threshold
with iso-value

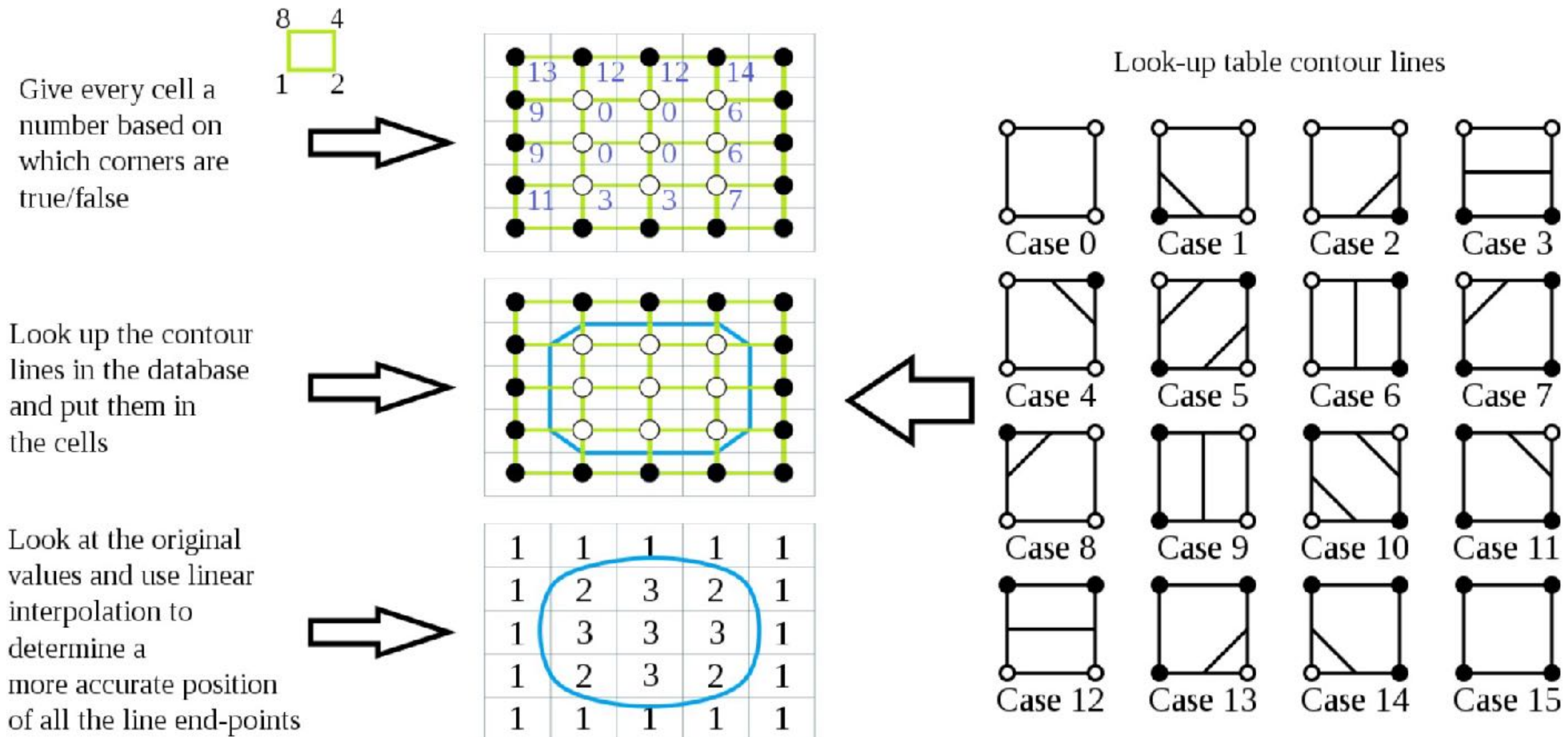


0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Binary image
to cells

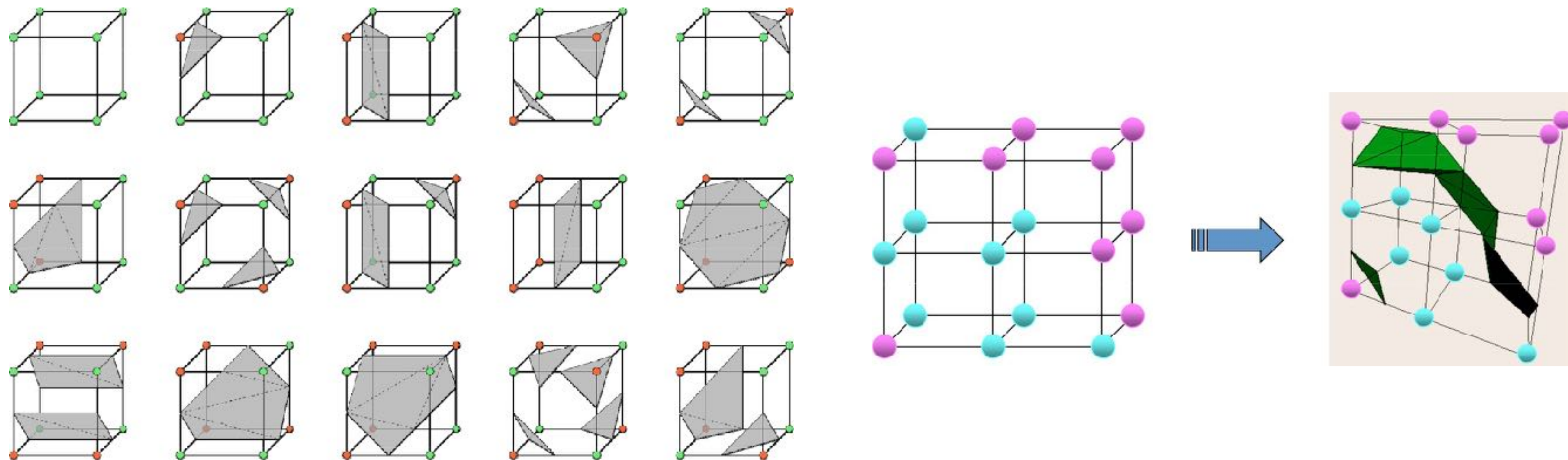


Classical Solution: 2D Marching Square



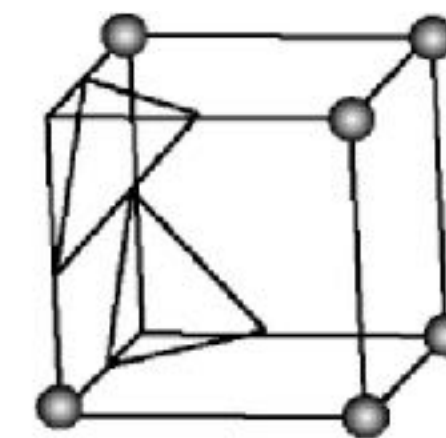
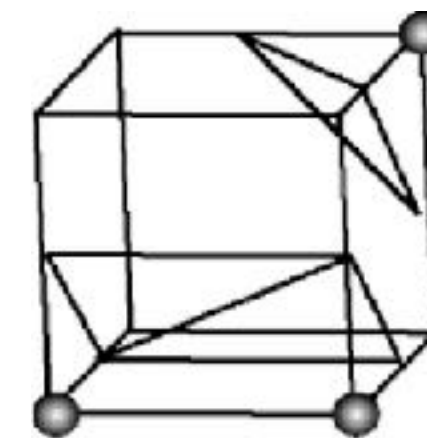
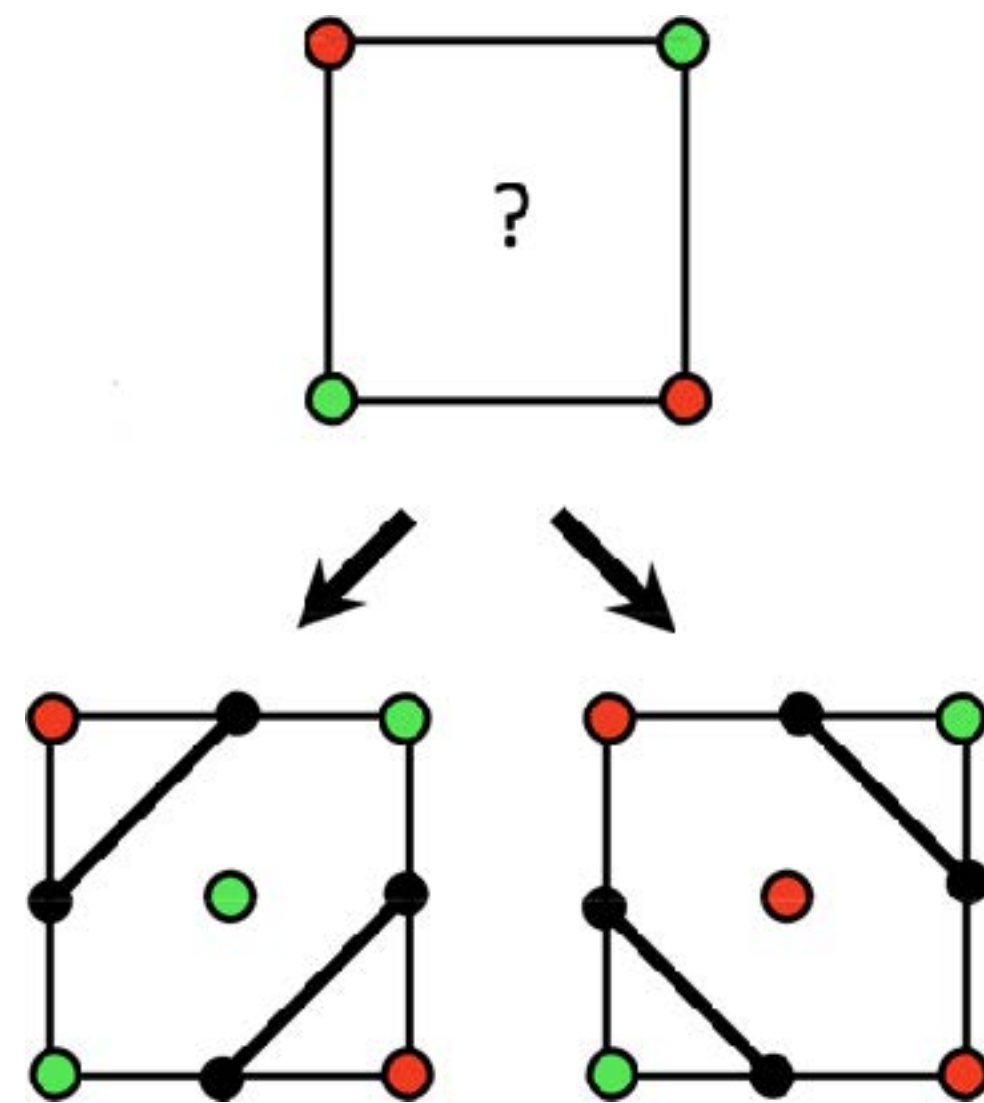
Classical Solution: 3D Marching Cube

- $2^8 = 256$ cases
- The first published version exploits rotation and inversion, and only considers 15 unique cases:



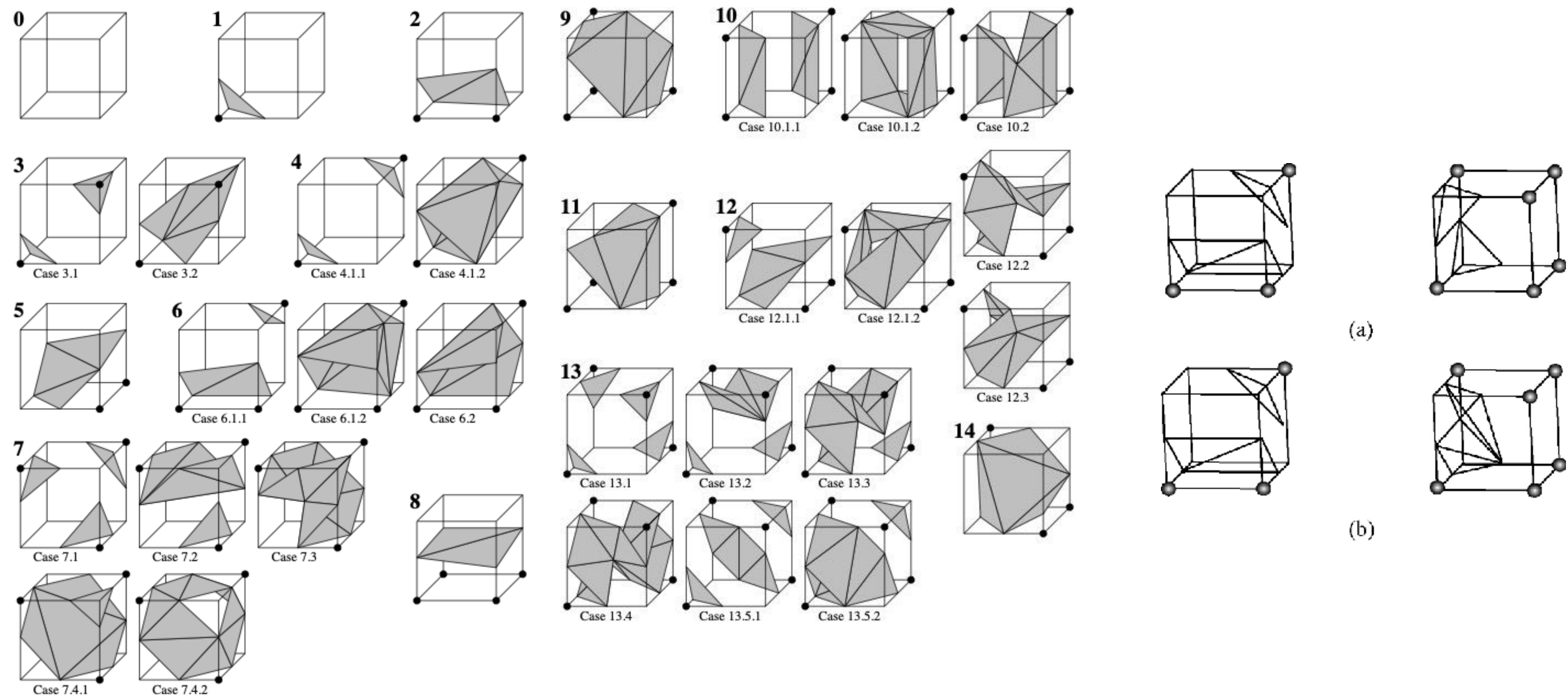
Ambiguity

- Ambiguity leads to holes:



Solution to Ambiguity

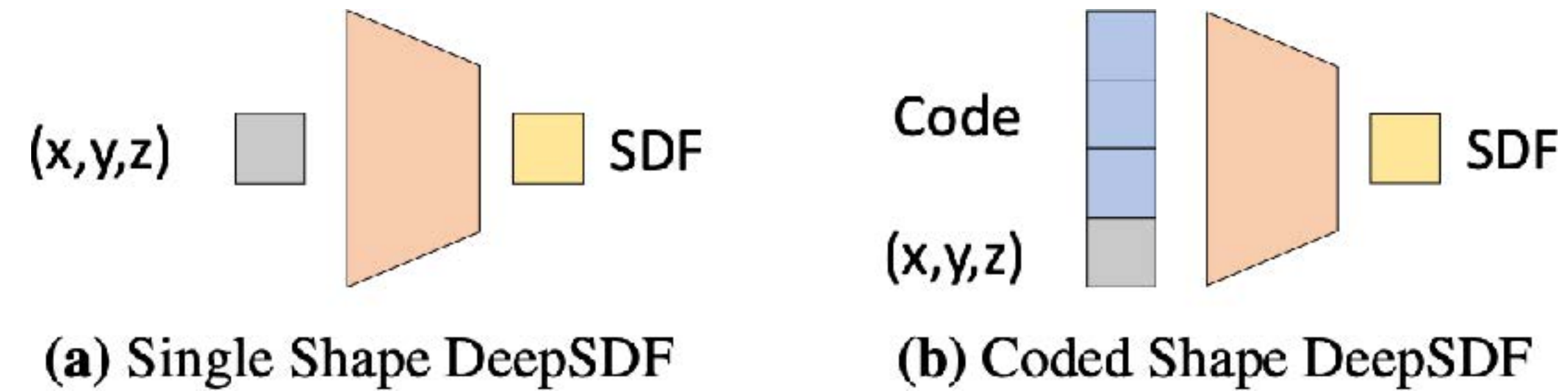
- Considering more cases in the look-up table by watching larger context:



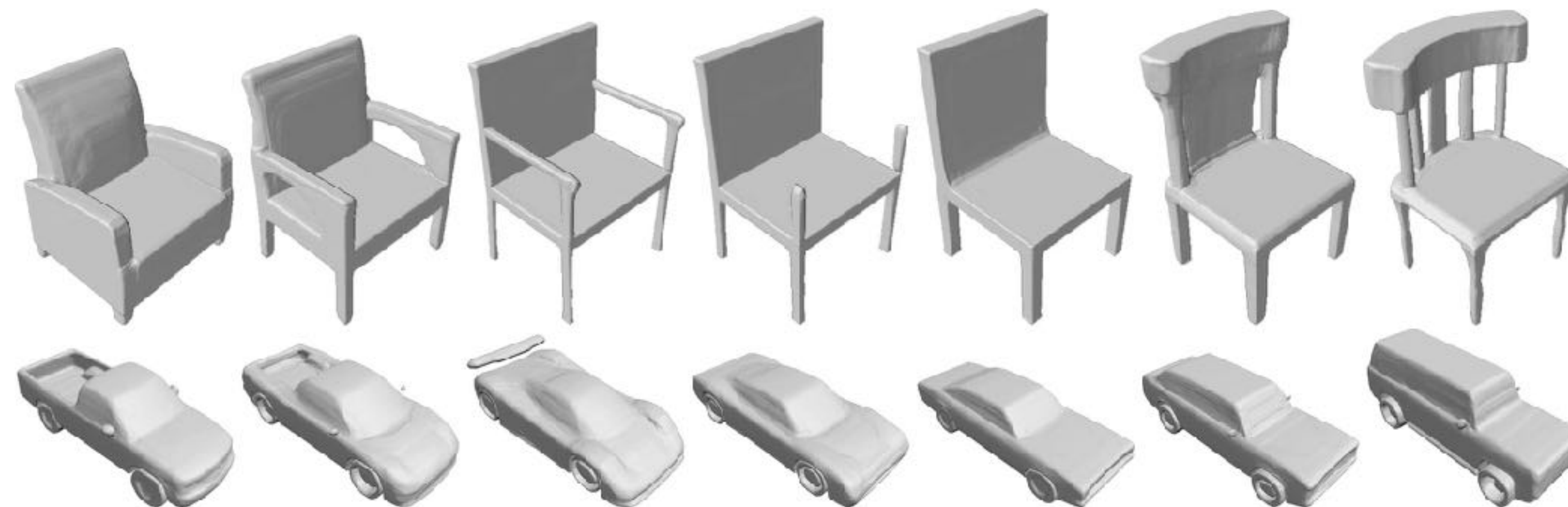
Comparison

	Explicit meshing (e.g., ball-pivoting)	Implicit meshing (e.g., RBF, MLS, Poission)
Sensitive to normals	No	Yes
Watertight manifold	No	Yes, in most cases
Complexity	Linear	<ul style="list-style-type: none">• Large-scale equations to estimate implicit function• Marching cubes• Dense voxelization

Use Neural Network to Approximate Implicit Field Function



- (a) use the network to overfit a single shape
- (b) use a latent code to represent a shape, so that the network can be used for multiple shapes



Learning-Based Marching Cube

Deep Marching Cubes: Learning Explicit Surface Representations

Yiyi Liao^{1,2}

Simon Donné^{1,3}

Andreas Geiger^{1,4}

¹Autonomous Vision Group, MPI for Intelligent Systems and University of Tübingen

²Institute of Cyber-Systems and Control, Zhejiang University

³imec - IPI - Ghent University

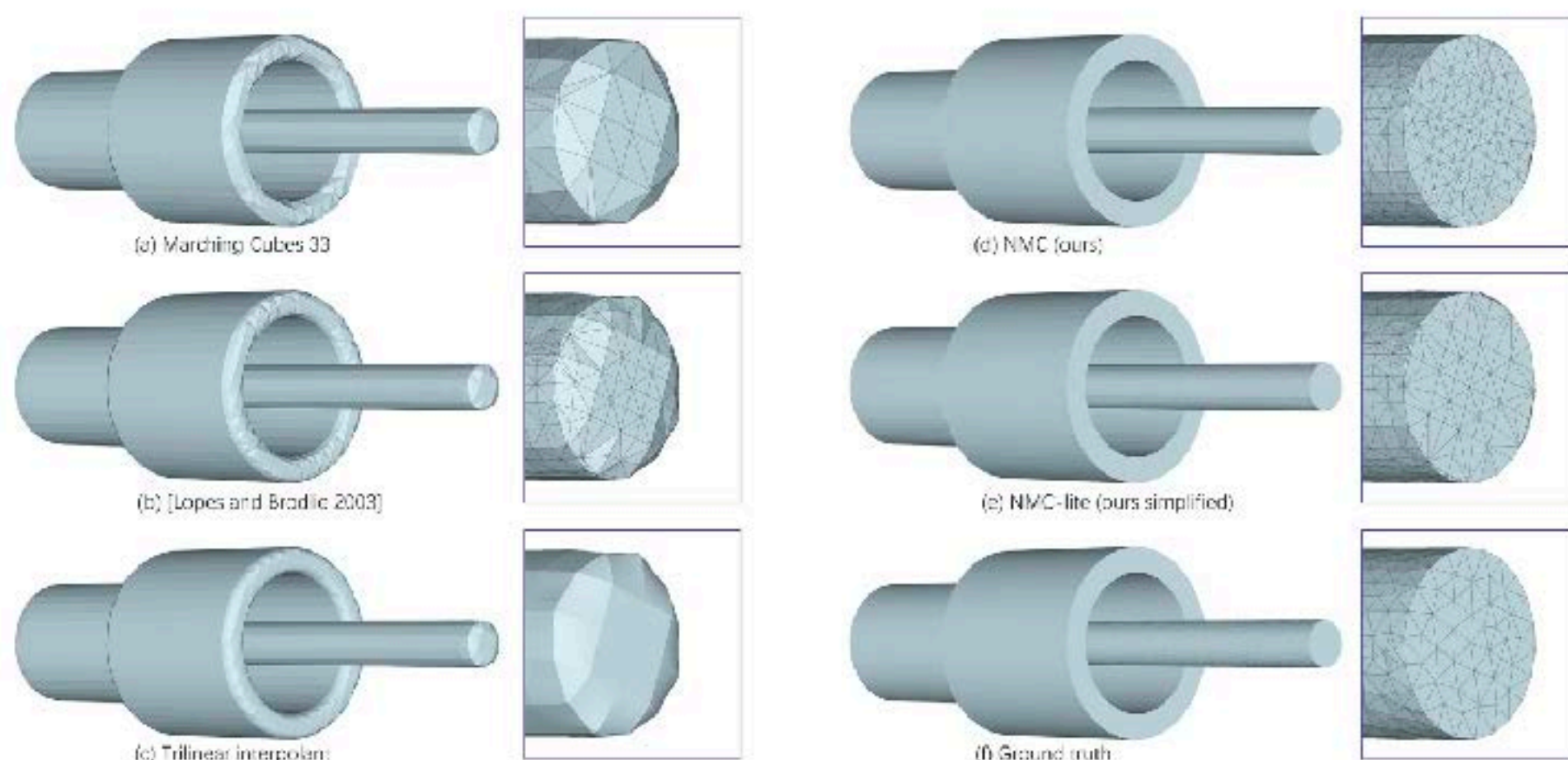
⁴CVG Group, ETH Zürich

{yiyi.liao, simon.donne, andreas.geiger}@tue.mpg.de

Neural Marching Cubes

ZHIQIN CHEN, Simon Fraser University, Canada

HAO ZHANG, Simon Fraser University, Canada



Neural Dual Contouring

ZHIQIN CHEN, Simon Fraser University, Canada

ANDREA TAGLIASACCHI, Google Research, University of Toronto, Canada

THOMAS FUNKHOUSER, Google Research, USA

HAO ZHANG, Simon Fraser University, Canada

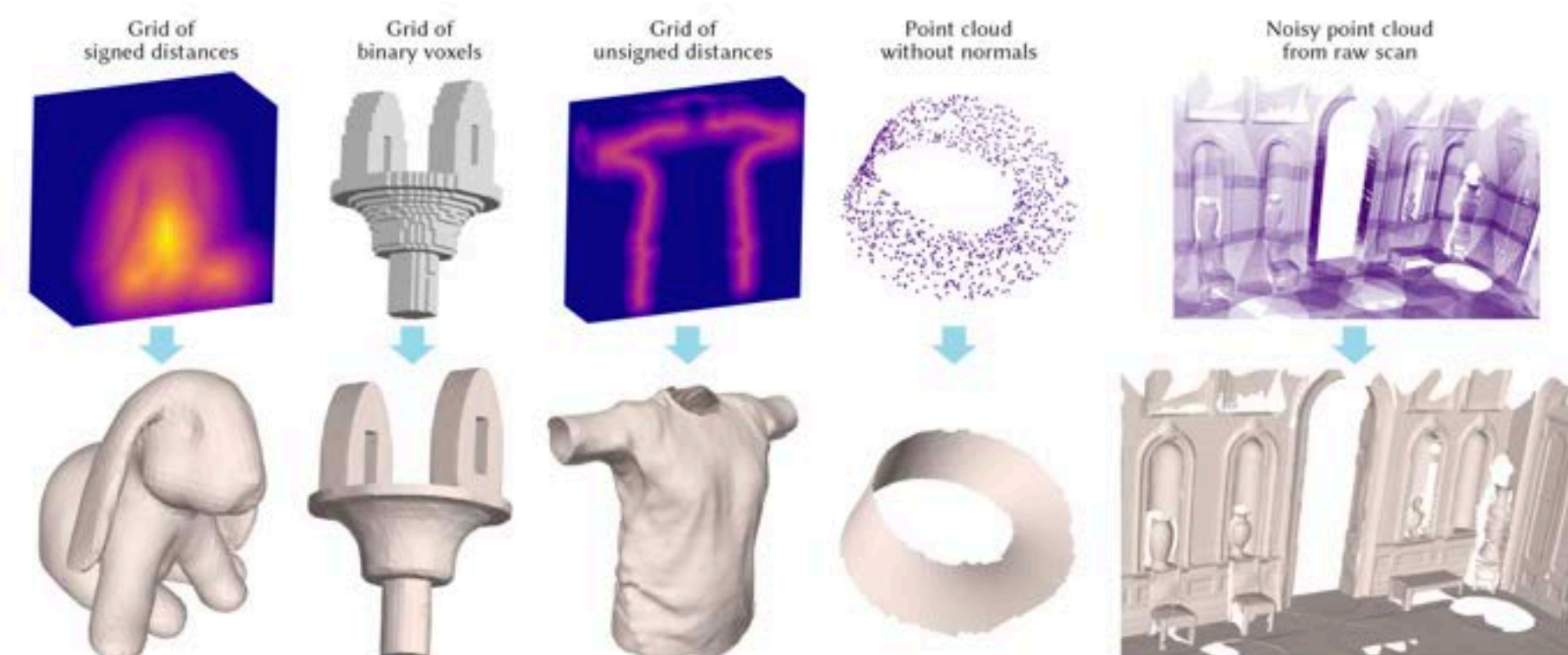


Fig. 1. Neural dual contouring (NDC) is a unified data-driven approach that learns to reconstruct meshes (bottom) from a variety of inputs (top): signed or unsigned distance fields, binary voxels, non-oriented point clouds, and noisy raw scans. Trained on CAD models, NDC generalizes to a broad range of shape types: CAD models with sharp edges, organic shapes, open surfaces for clothes, scans of indoor scenes, and even the non-orientable Möbius strip.

Consistent Orientation?

- Require the ground-truth ***signed*** implicit functions during training (e.g., signed distance, occupancy)
- However, 3D raw data, such as point cloud or triangle soup, are not necessarily consistently oriented
 - e.g., getting normal orientation for ShapeNet data is not easy

Sign Agnostic Learning of Shapes from Raw Data

- Unsigned distance is easy to obtain
 - Distance to the point cloud & triangle soup
- Learn ***signed*** distance from ***unsigned distance*** groundtruth
 - Require a special loss function

Sign Agnostic Learning

$$\text{loss}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x} \sim D_{\chi}} \tau \left(f(\mathbf{x}; \boldsymbol{\theta}), h_{\chi}(\mathbf{x}) \right)$$

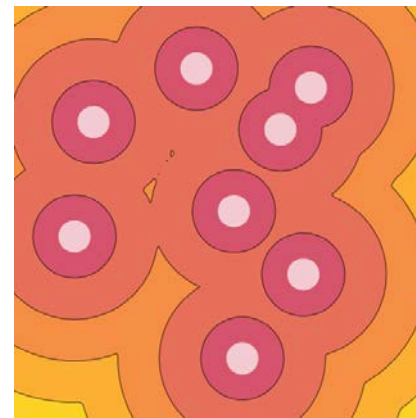
- $\chi \subset \mathbb{R}^3$: input raw data (*e.g.*, a point cloud or a triangle soup)
- $f(\mathbf{x}; \boldsymbol{\theta}) : \mathbb{R}^3 \times \mathbb{R}^m \rightarrow \mathbb{R}$: learned signed function
- D_{χ} : distribution of the training samples defined by χ
- $h_{\chi}(\mathbf{x})$: some *unsigned* distance measure to χ
- $\tau : \mathbb{R} \times \mathbb{R}_+ \rightarrow \mathbb{R}$: a similarity function

Sign Agnostic Learning

$$\text{loss}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x} \sim D_{\chi}} \tau \left(f(\mathbf{x}; \boldsymbol{\theta}), h_{\chi}(\mathbf{x}) \right)$$

- $h_{\chi}(\mathbf{x})$: some *unsigned* distance measure to χ

$$h_2(\mathbf{z}) = \min_{\mathbf{x} \in \chi} \|\mathbf{z} - \mathbf{x}\|_2$$



$$h_0(z) = \begin{cases} 0 & z \in \chi \\ 1 & z \notin \chi \end{cases}$$



- $\tau : \mathbb{R} \times \mathbb{R}_+ \rightarrow \mathbb{R}$: a similarity function

$$\tau_{\ell}(a, b) = ||a| - b|^{\ell}$$

Two Local Minima

$$\text{loss}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x} \sim D_\chi} \tau \left(f(\mathbf{x}; \boldsymbol{\theta}), h_\chi(\mathbf{x}) \right)$$

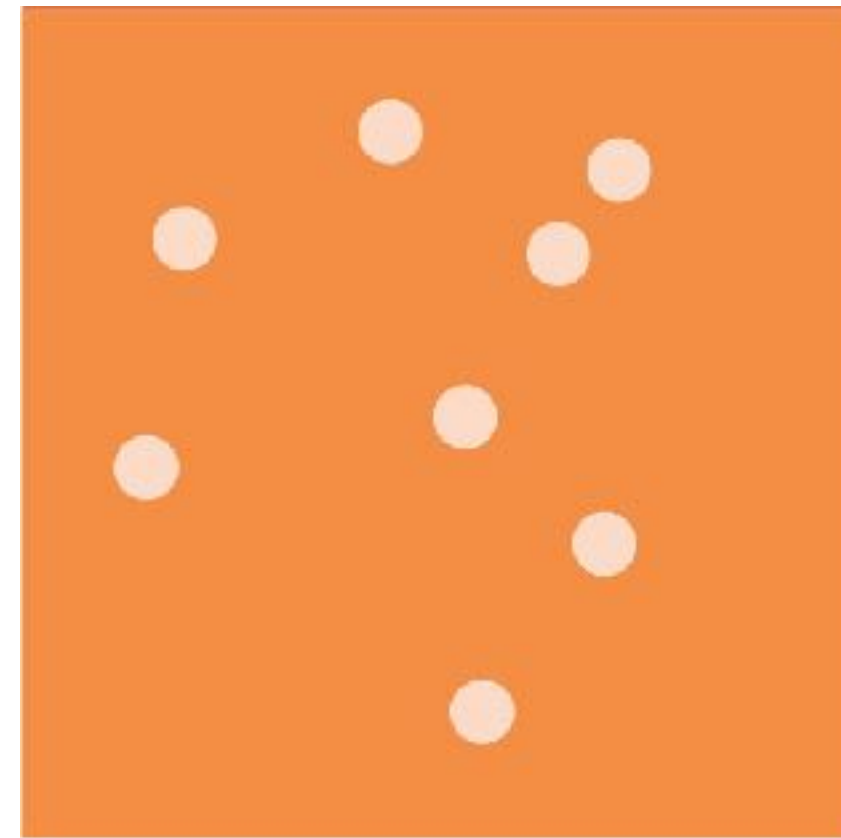
- f is an ***unsigned*** function that resembles $h_\chi(\mathbf{x})$
- f is a ***signed*** function and $|f|$ resembles $h_\chi(\mathbf{x})$
- We prefer the second case to use marching cube

Two Local Minima

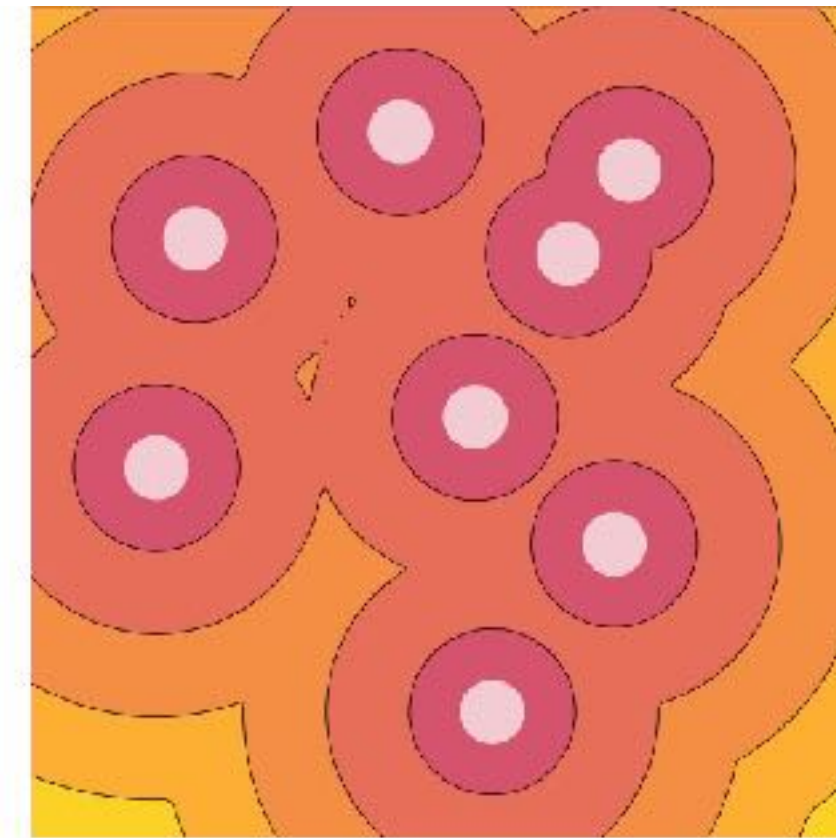
- Pick a special weight initialization θ^0 so that $f(x; \theta^0) \approx \varphi(\|x\| - r)$, where $\varphi(\|x\| - r)$ is the signed distance function to an r -radius sphere
 - $f > 0$ if $\|x\| > r$
 - $f < 0$ if $\|x\| < r$
- Under such an initialization, f is not easy to converge to the unsigned local minima.

2D Results

Unsigned
function as
supervision

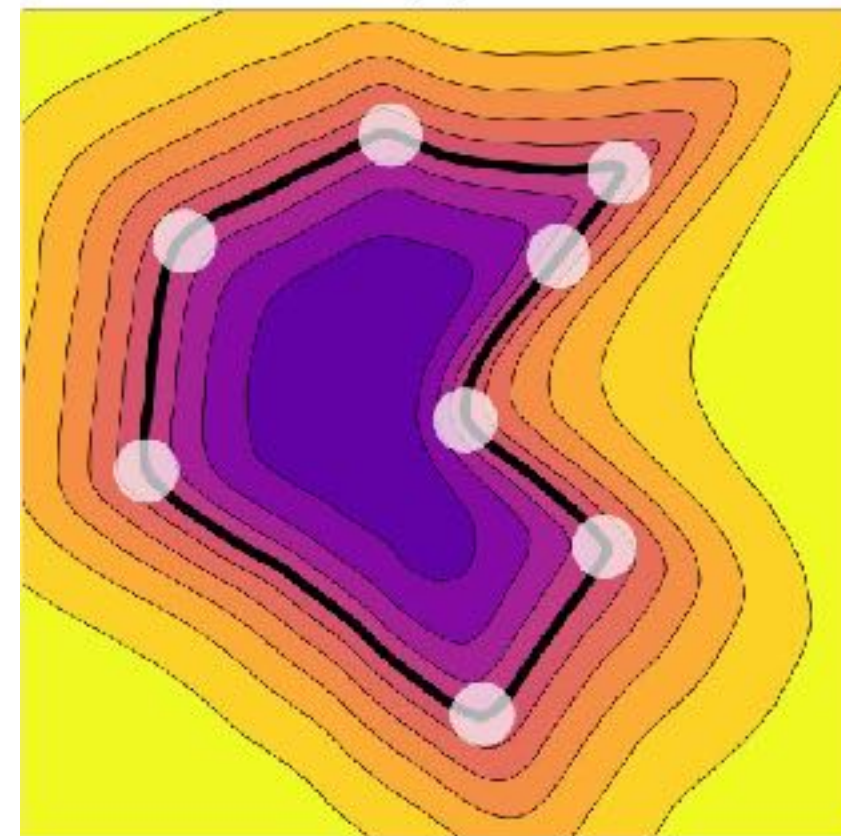


(a)

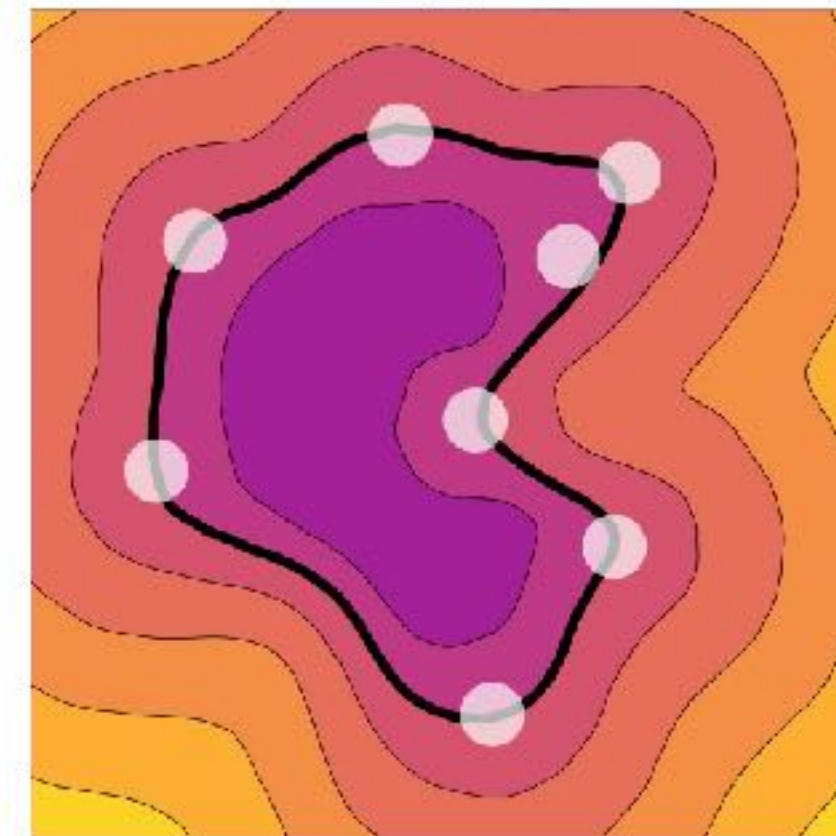


(b)

Learned
signed function



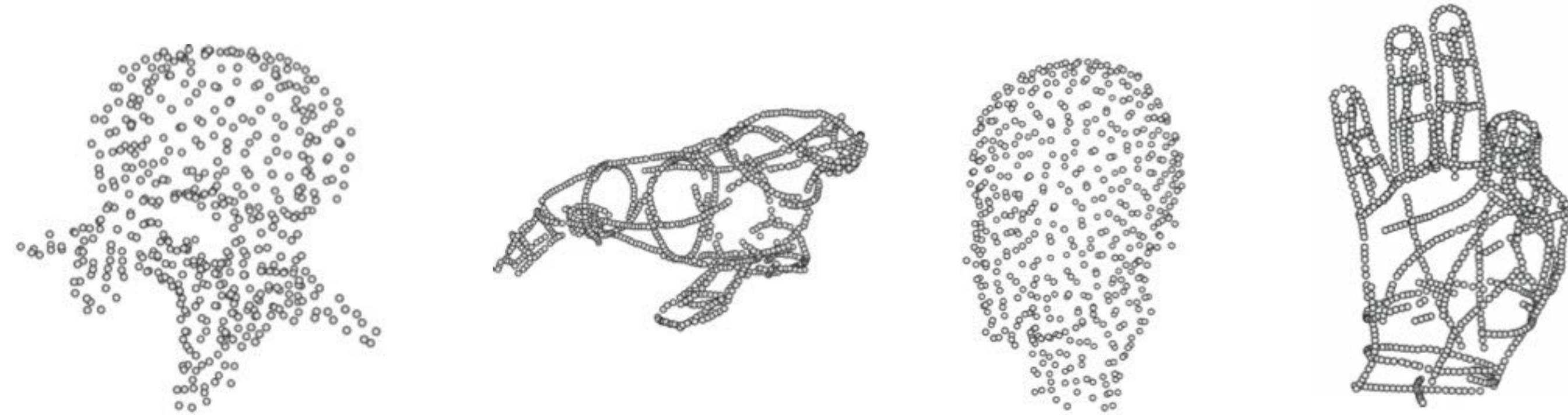
(c)



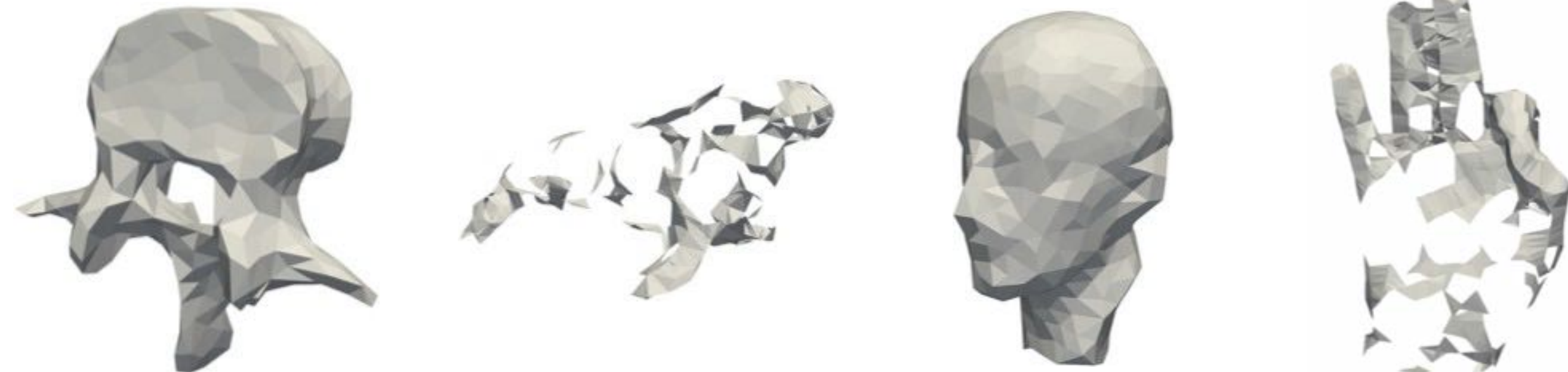
(d)

Surface Reconstruction Results from Raw Data

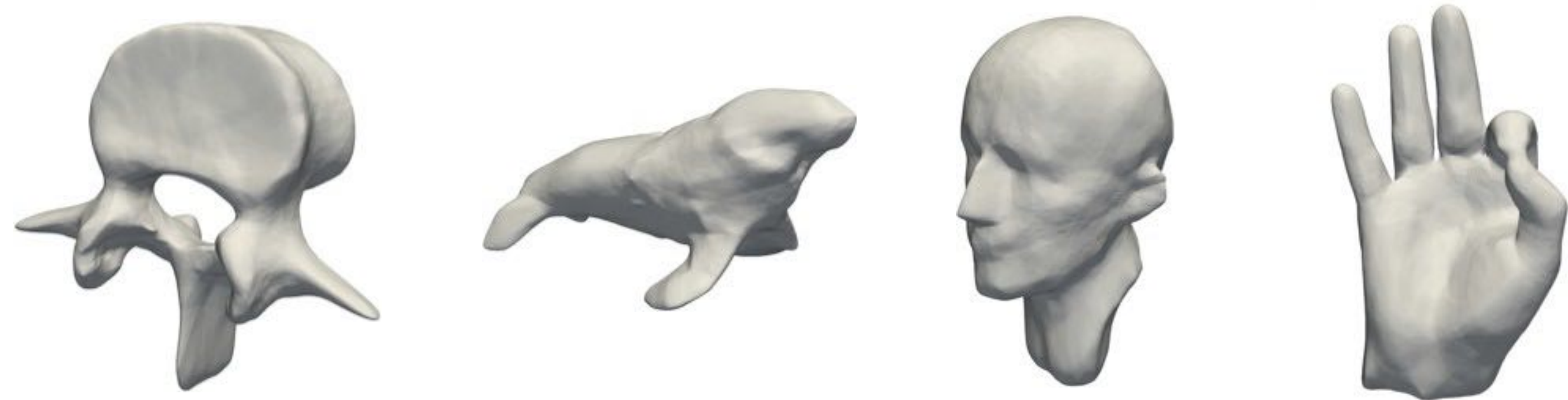
Raw
Point Cloud



Ball-Pivoting
Algorithm



SAL



Next time

- AIGC in 3D

a sculpture of a rooster.



a robotic dog. a robot in the shape of a dog.



a pile of crab is seasoned and well cooked.



Point cloud diffusion model