

HW2 Update

- Extend the deadline to 23:59pm, next Monday (Feb 21)
- Updated rubric (lowered requirement):

	ICP (>)	Learning (>)
2pt	Beat baseline	Beat baseline
3pt	40%	20%
4pt	60%	50%
5pt	80%	80%

L13: Analysis by Intrinsic

Hao Su

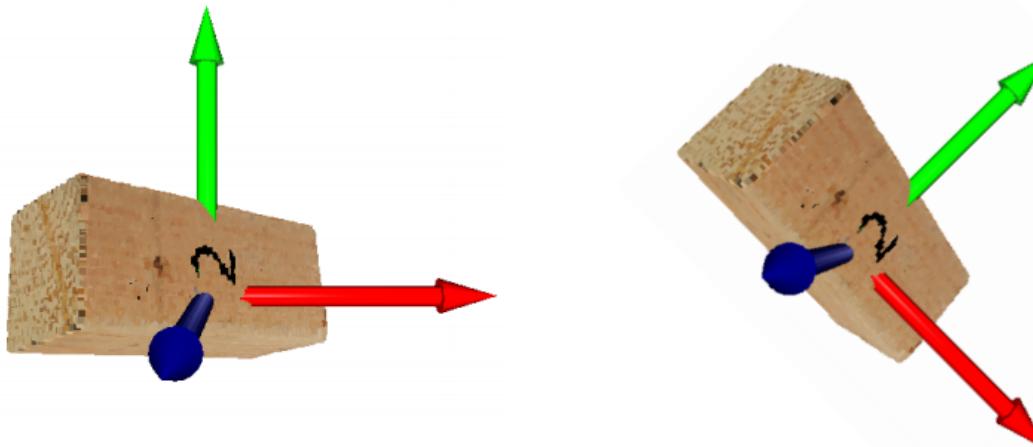
Thank Xiaoshuai Zhang for helping to prepare slides

Extrinsics v.s. Intrinsics

- Extrinsics: cares about the embedding of a surface
 - Looking at the surface **outside** the surface

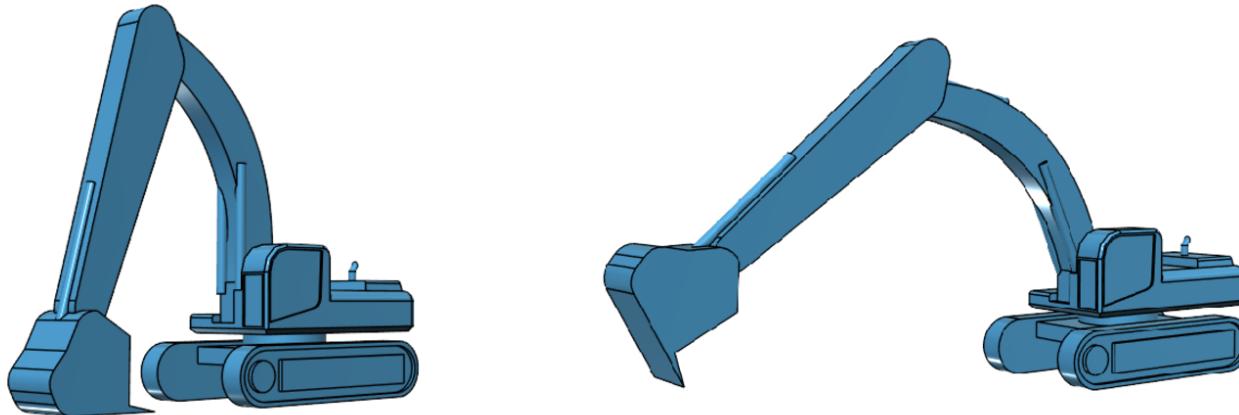
Extrinsics v.s. Intrinsics

- Extrinsics: cares about the embedding of a surface
 - Looking at the surface **outside** the surface
 - For example:
 - est. 6D pose of a rigid body



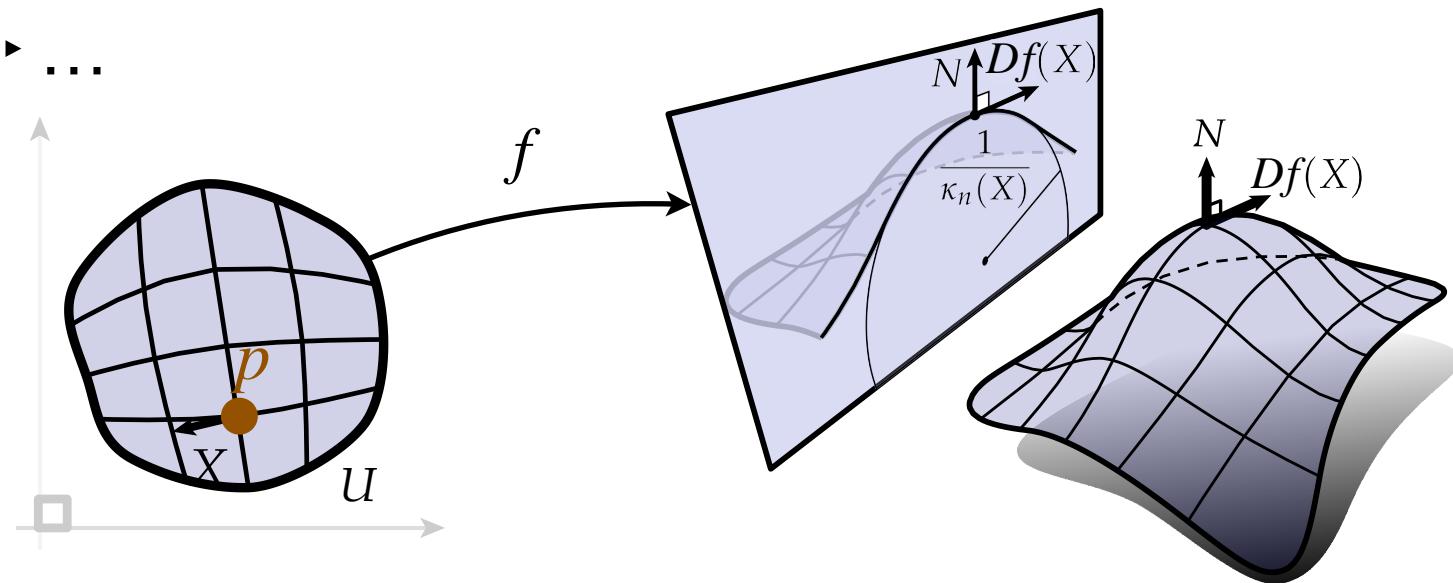
Extrinsics v.s. Intrinsics

- Extrinsics: cares about the embedding of a surface
 - Looking at the surface **outside** the surface
 - For example:
 - est. 6D pose of a rigid body
 - est. Articulation state of joints linking rigid bodies



Extrinsics v.s. Intrinsics

- Extrinsics: cares about the embedding of a surface
 - Looking at the surface **outside** the surface
 - For example:
 - est. 6D pose of a rigid body
 - est. Articulation state of joints linking rigid bodies
 - est. Normal curvature
 - ...



Extrinsics v.s. Intrinsics

- Extrinsics: cares about the embedding of a surface
 - Looking at the surface **outside** the surface
 - For example:
 - est. 6D pose of a rigid body
 - est. Articulation state of joints linking rigid bodies
 - est. Normal curvature
 - ...
- Intrinsics:
 - Looking at the surface **inside** the surface

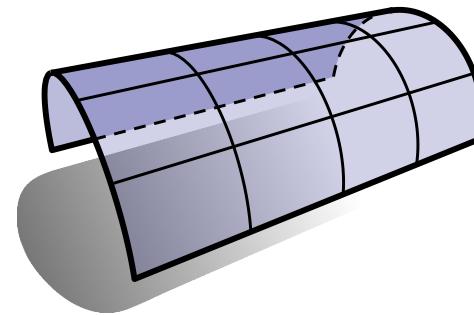
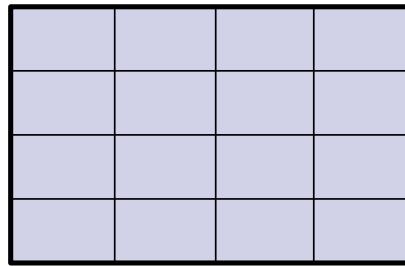
Extrinsics v.s. Intrinsics

- Extrinsics: cares about the embedding of a surface
 - Looking at the surface **outside** the surface
 - For example:
 - est. 6D pose of a rigid body
 - est. Articulation state of joints linking rigid bodies
 - est. Normal curvature
 - ...
- Intrinsics:
 - Looking at the surface **inside** the surface
 - **Geodesic distance** uniquely determines shape intrinsics

Recall: Gaussian and Mean Curvatures

Gaussian: $K := \kappa_1 \kappa_2$

Mean: $H := \frac{1}{2}(\kappa_1 + \kappa_2)$



$$\begin{aligned} K &= 0 \\ H &= 0 \\ \kappa_n(X) &\equiv 0 \end{aligned}$$

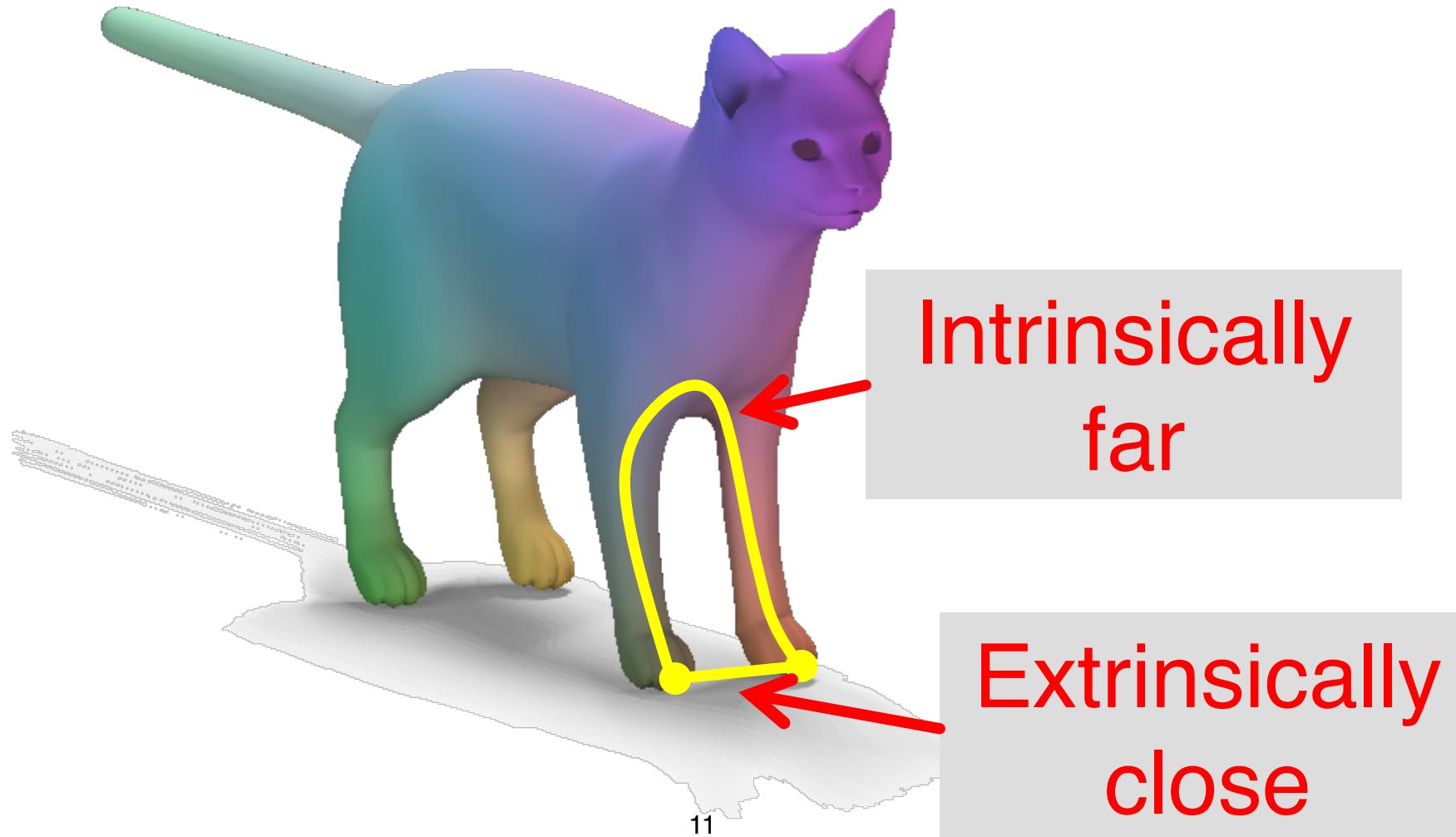
$$\begin{aligned} K &= 0 \\ H &\neq 0 \\ \kappa_n(X) &\not\equiv 0 \end{aligned}$$

Recall: Theorema Egregium

The Gaussian curvature of an embedded smooth surface in \mathbb{R}^3 is invariant under the local isometries.

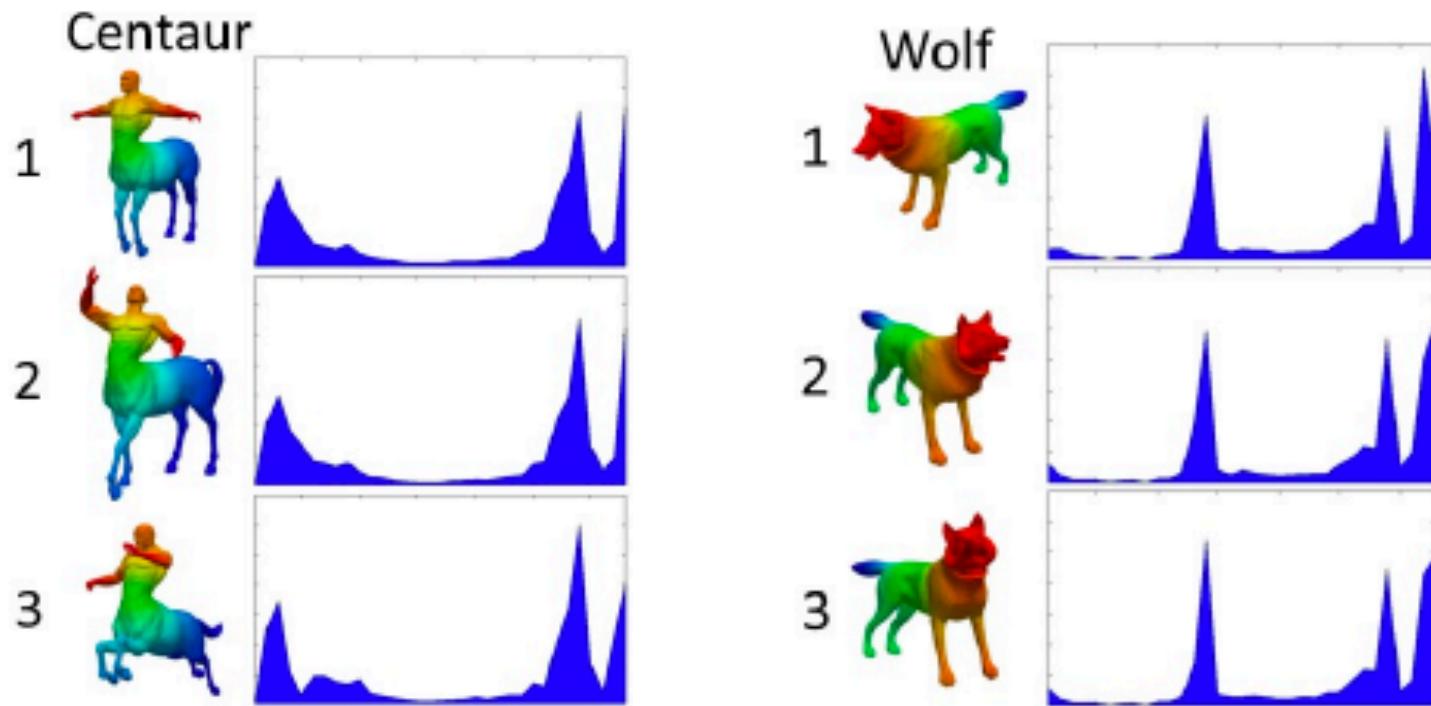
Gaussian Curvature is an Intrinsic Measurement of a Surface

Geodesic Distances



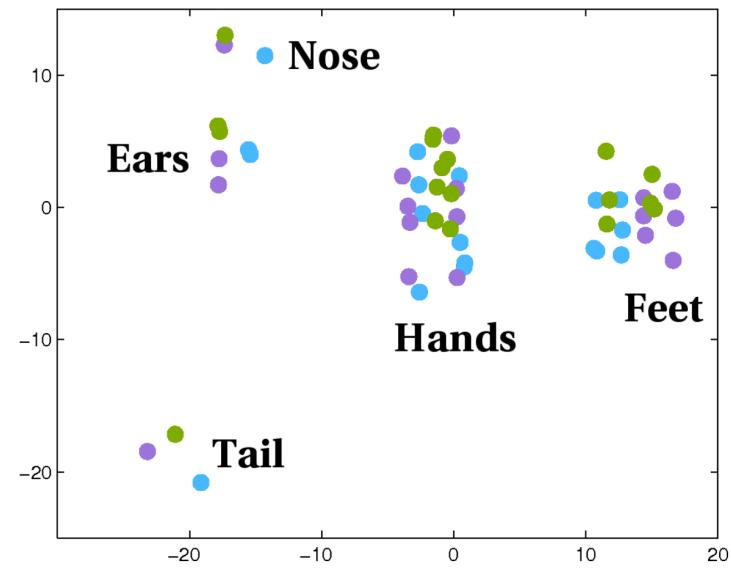
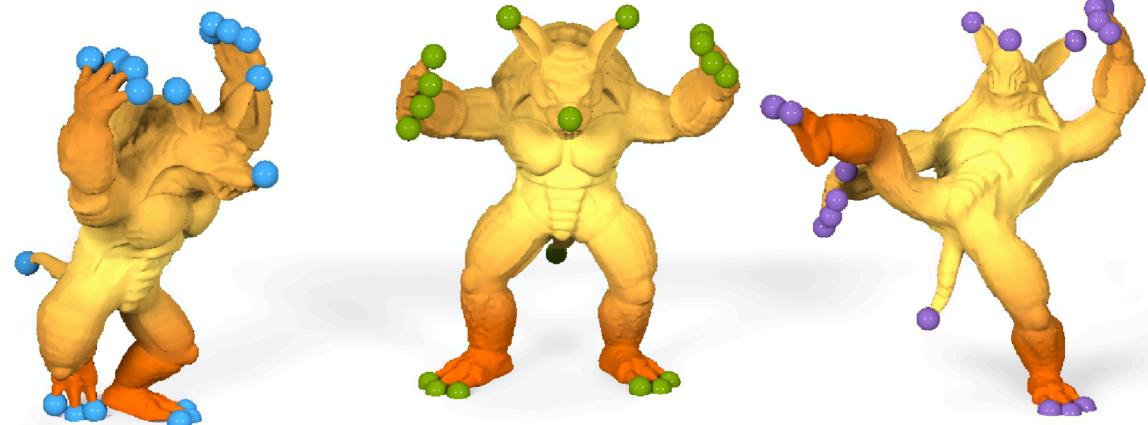
Applications: Shape Classification

- By classical method: Distribution of distances for point pairs randomly picked on the surface



Applications: Deformation-Invariant Point Feature

- By learning-based method: Extract some geodesic distance based features (intrinsic features)



Agenda

- Analytic Method for Computing Geodesics
- Learning to Predict Geodesics for Point Cloud
- Applications of Learning Point Cloud Geodesics
 - Normal Estimation
 - Mesh Reconstruction

Analytical Method for Computing Geodesics

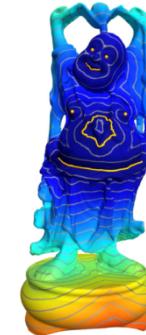
Related Queries



Single source



Multi-source

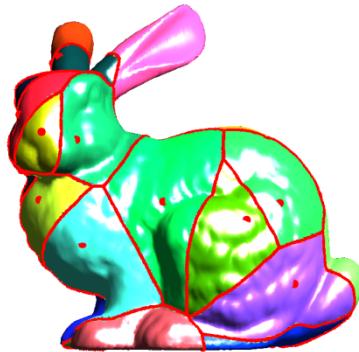


All-pairs

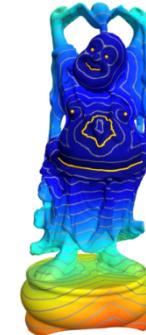
Related Queries



Single source

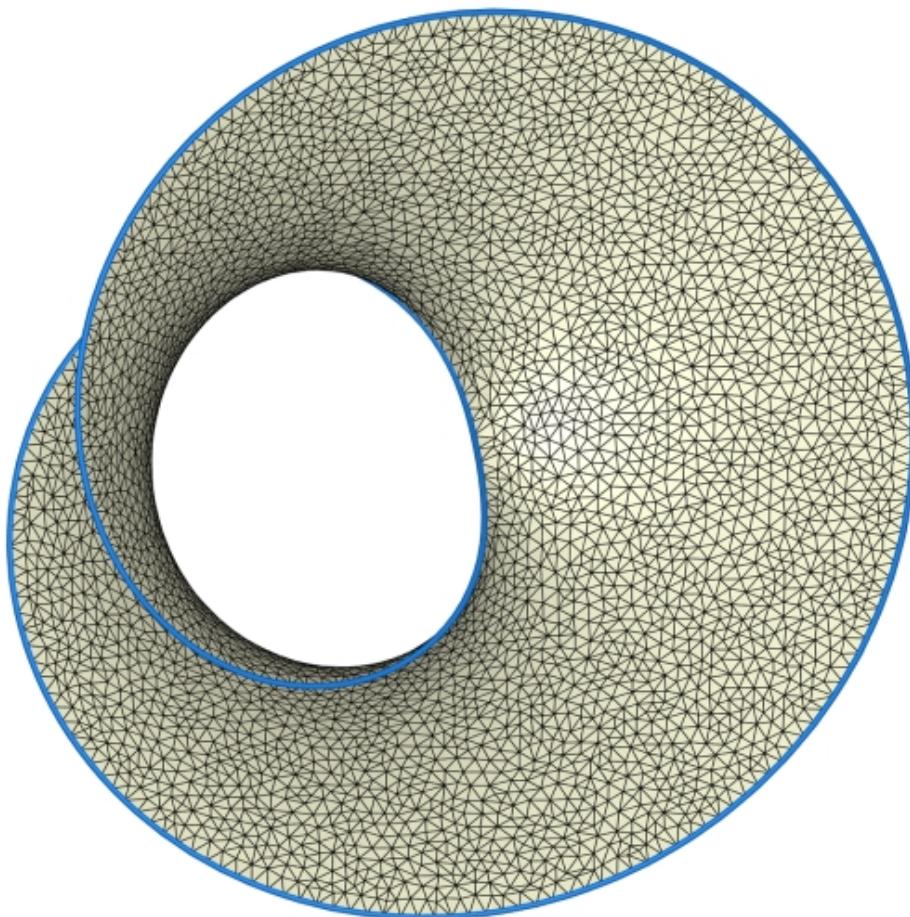


Multi-source



All-pairs

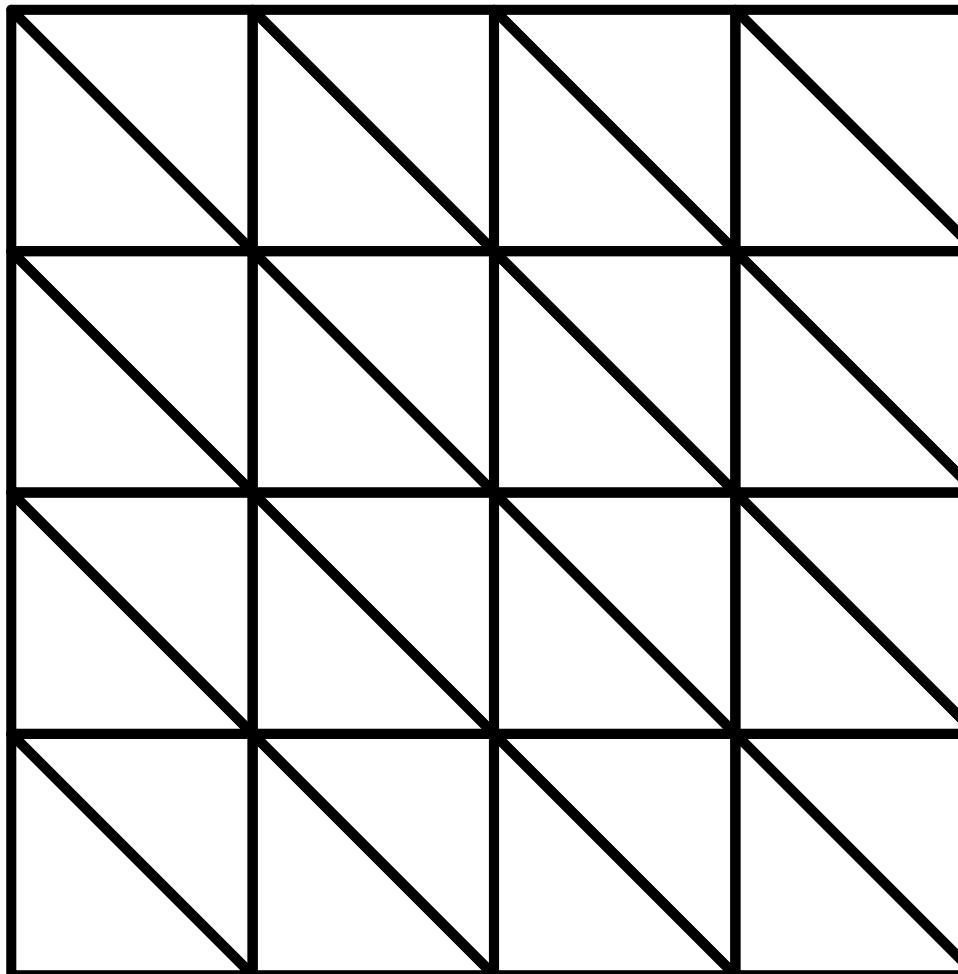
Geodesics on Meshes



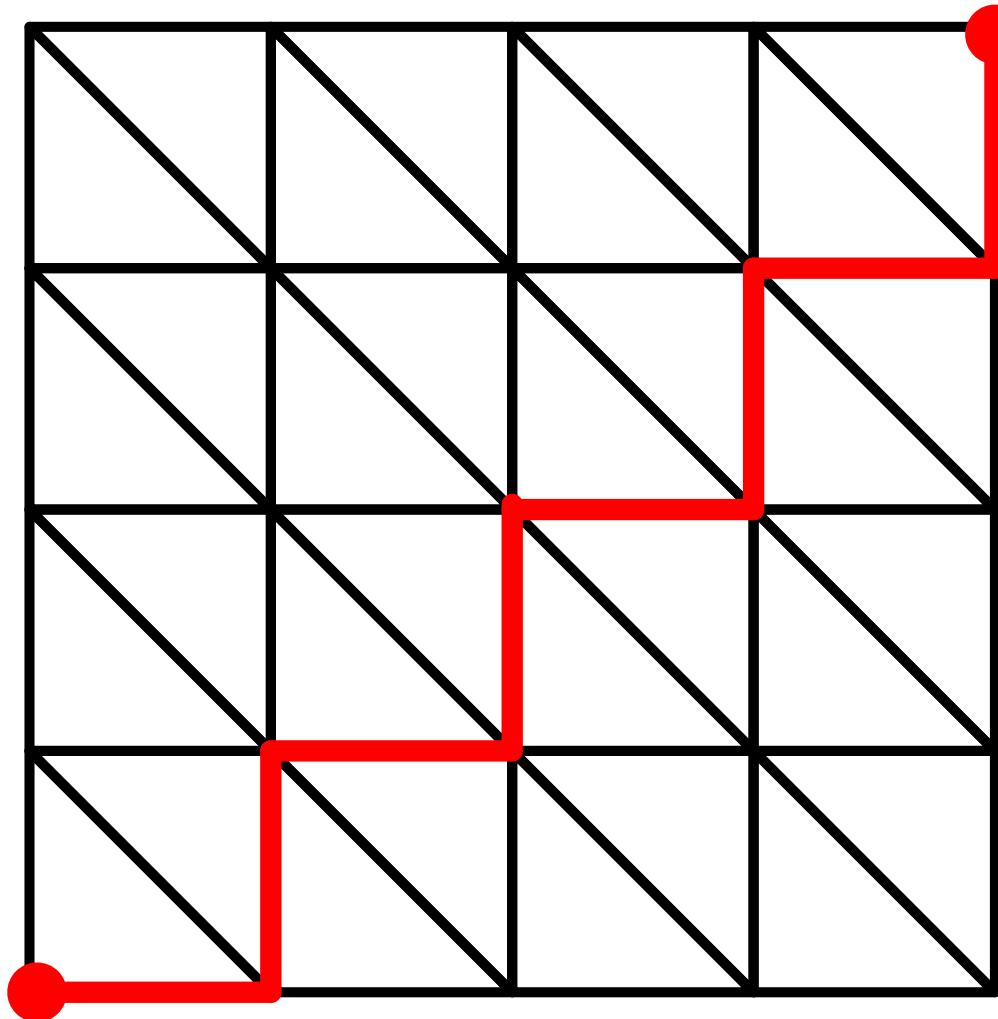
Approximate
geodesics as
paths along
edges

Meshes are graphs

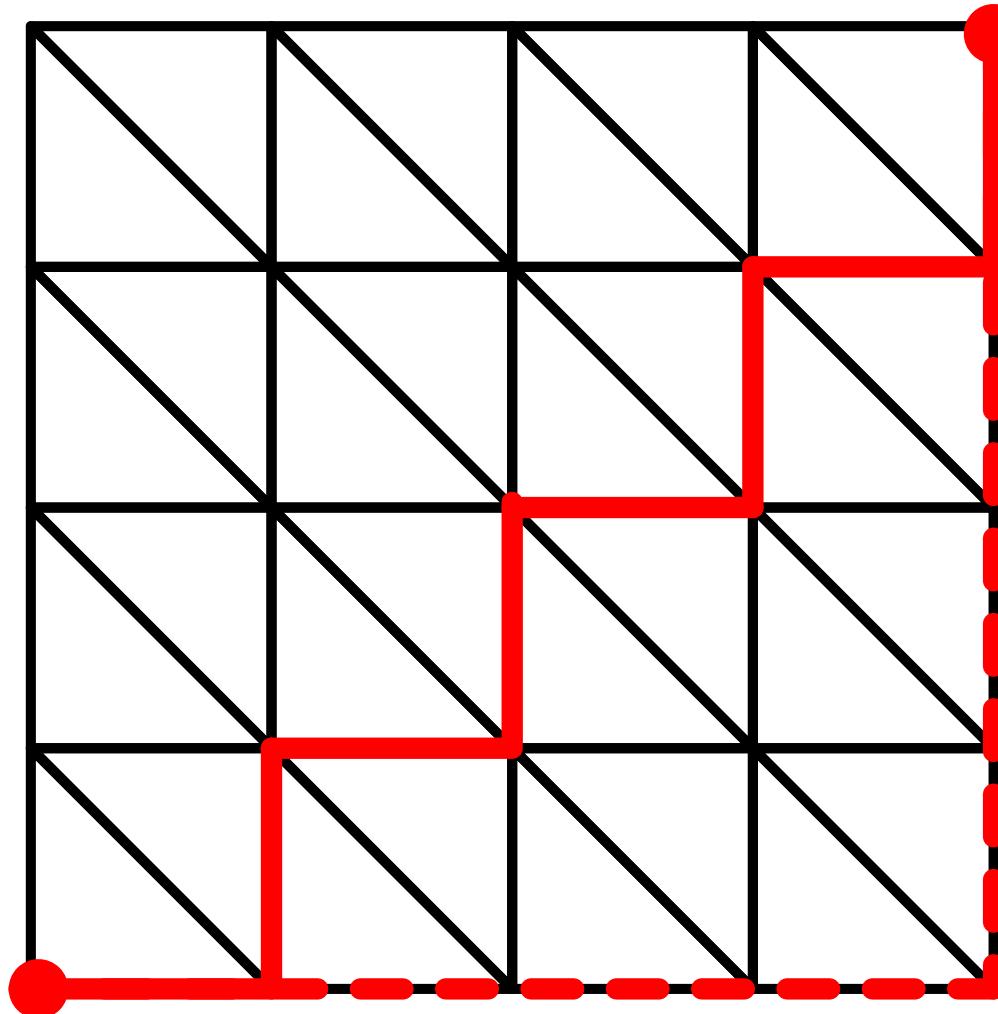
Can We Use Shortest Path Algorithms to Compute Geodesics?



Can We Use Shortest Path Algorithms to Compute Geodesics?



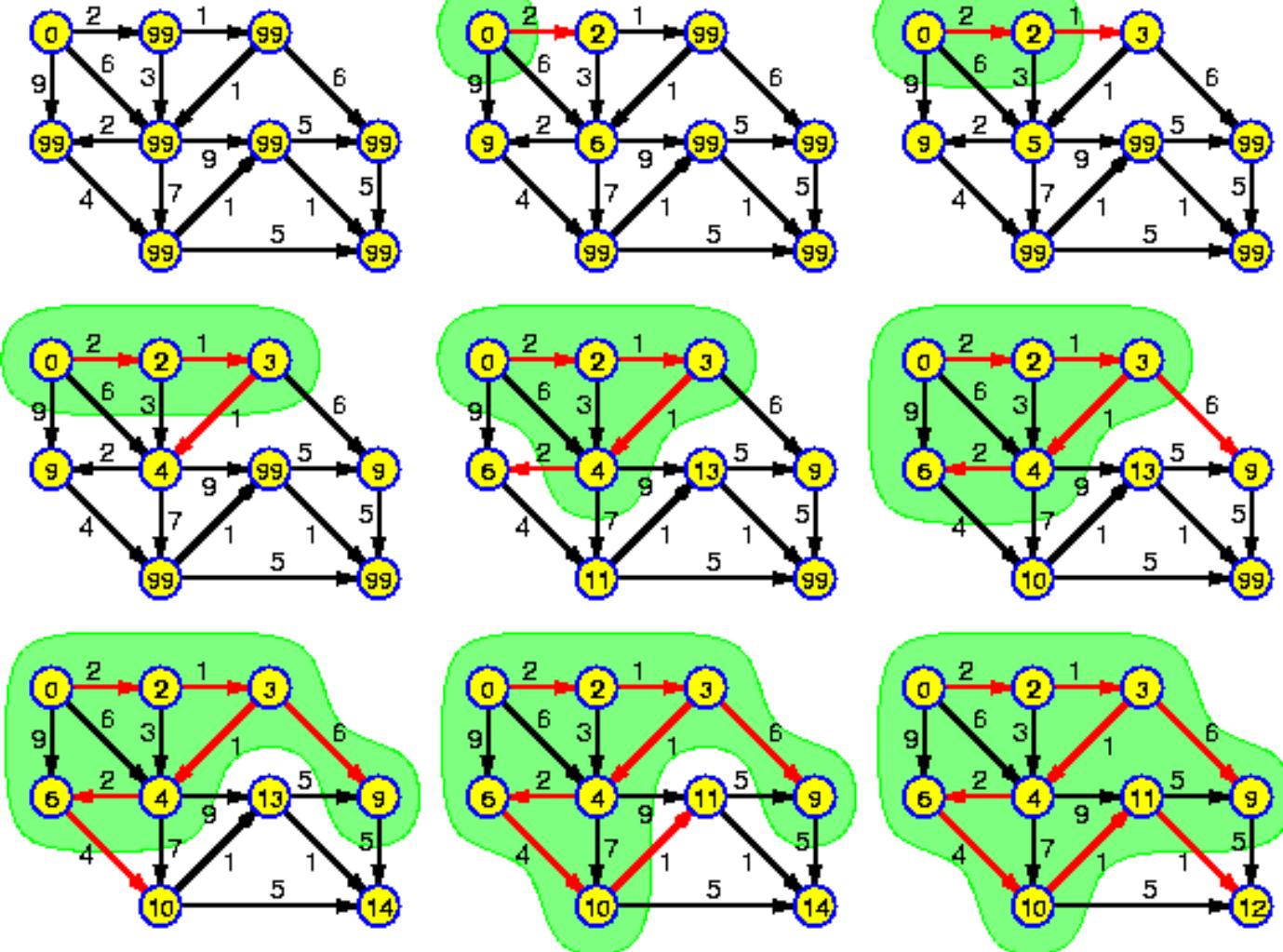
Can We Use Shortest Path Algorithms to Compute Geodesics?



Fast Marching Algorithm

Dijkstra's algorithm, modified to
approximate geodesic distances.

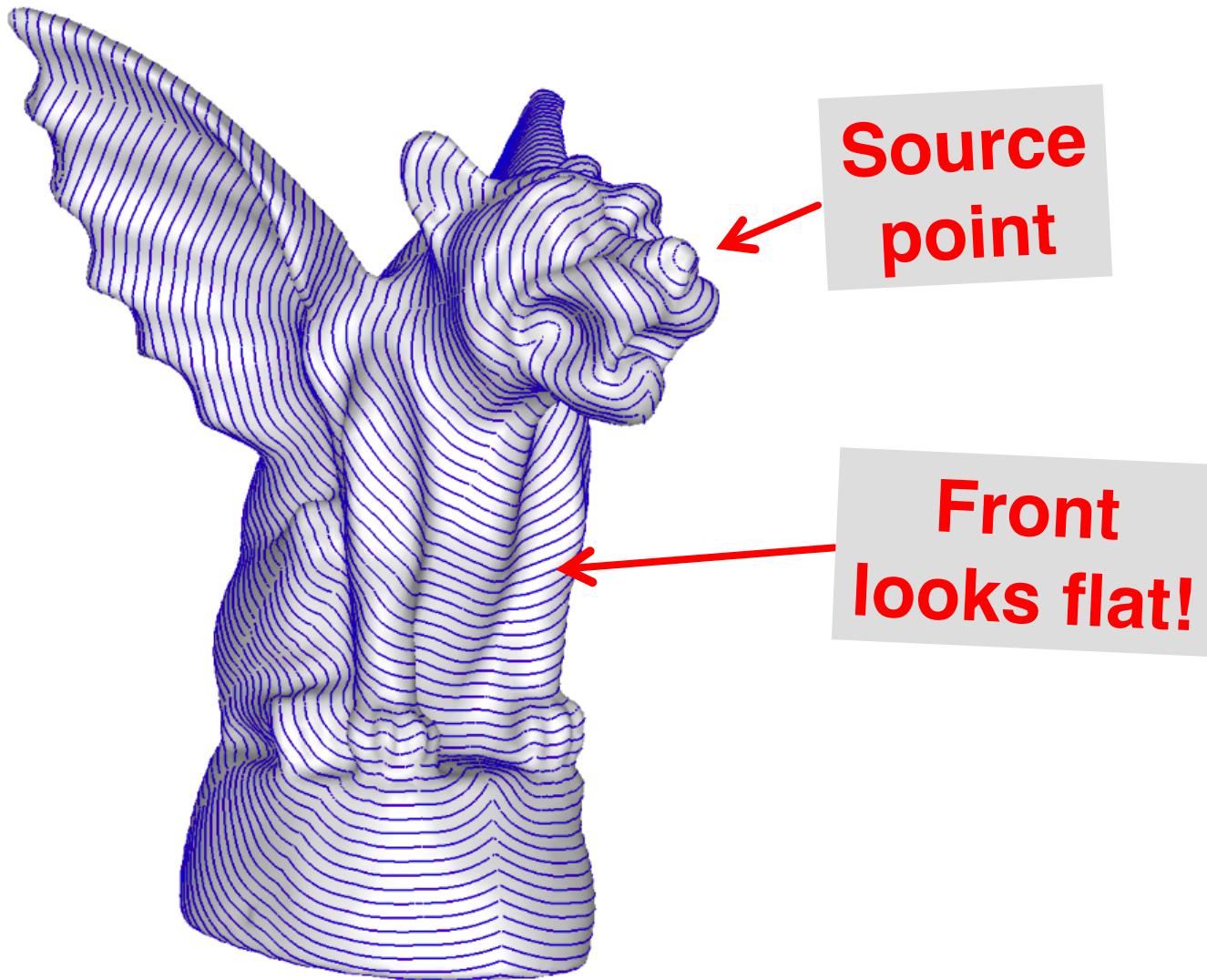
Review of Dijkstra's Algorithm



Key Idea of Dijkstra's Algorithm

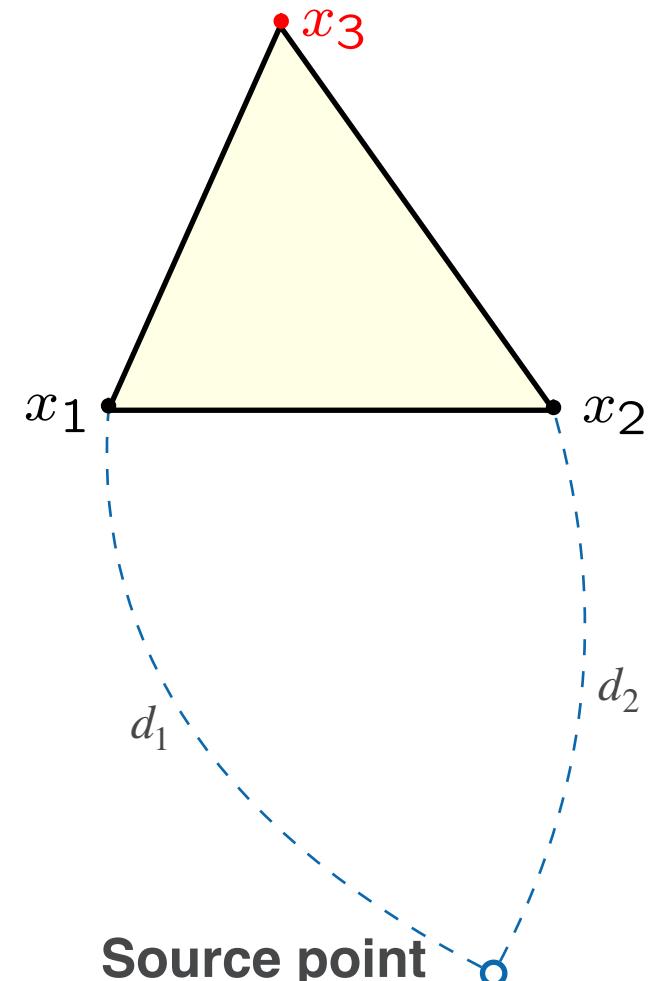
- Maintain a frontier set of vertices with the shortest distance from the source
- Propagate to the neighborhood

Fast Marching Algorithm



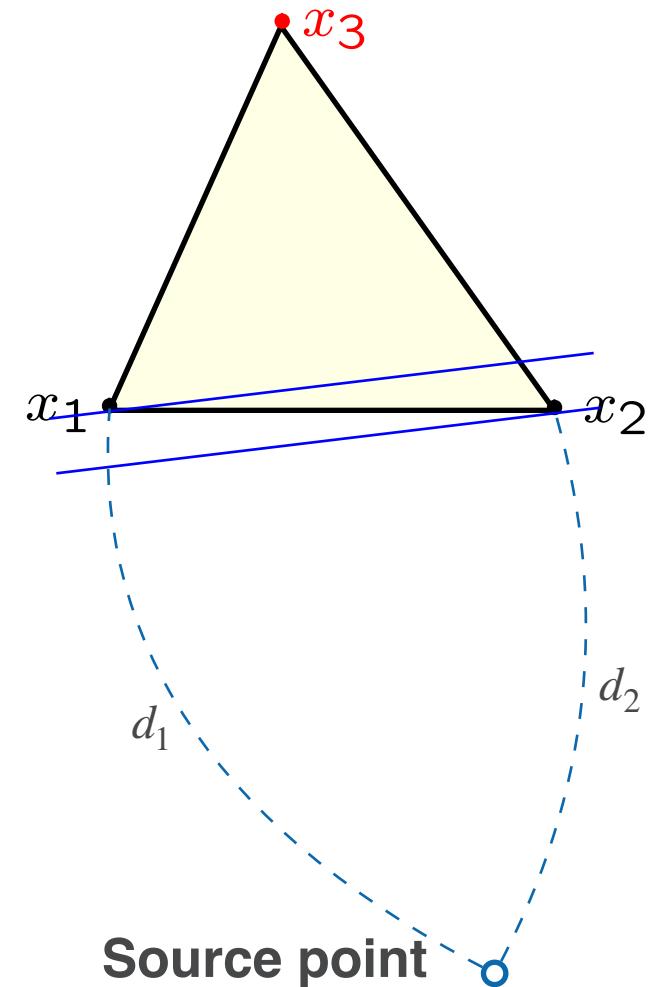
Fast Marching Algorithm

- At x_1 and x_2 stores the shortest paths d_1 and d_2
- Question: shortest path d_3 at x_3 ?



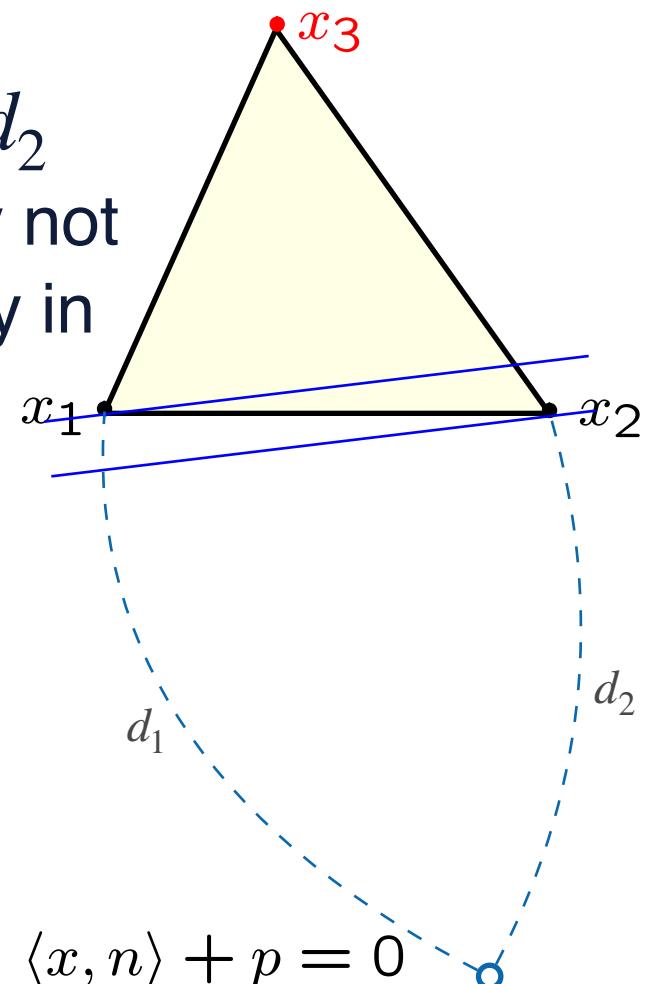
Fast Marching Algorithm

- Idea: Planar front approximation ($\because |x_1x_2| \approx 0$)

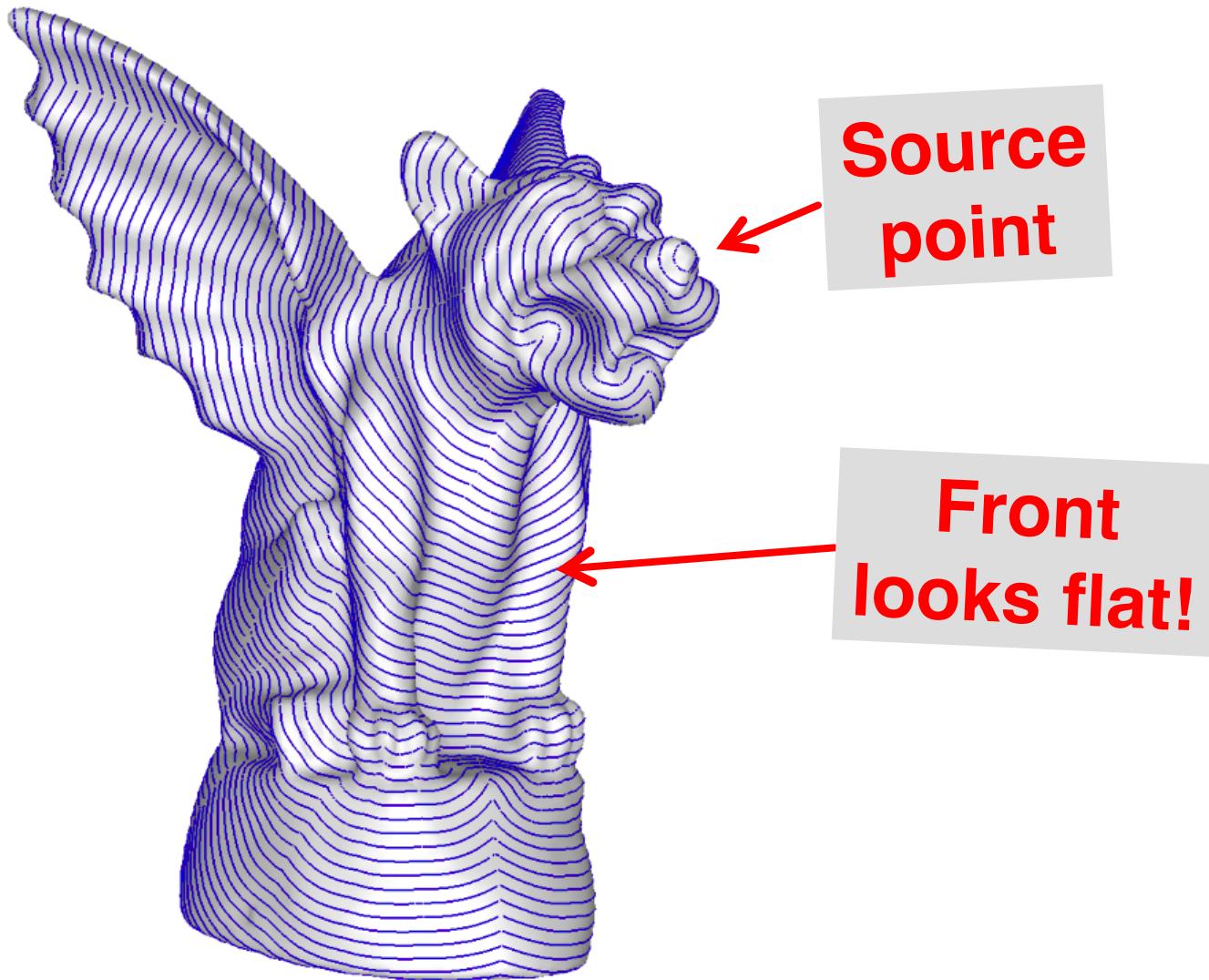


Fast Marching Algorithm

- Planar front approximation
 - Find a point in the plane of $\triangle x_1x_2x_3$ according to the vertex-plane distance d_1 and d_2
 - Issue: $|x_1x_2|$, d_1 , and d_2 may not conform to triangular inequality in the $\triangle x_1x_2x_3$ plane

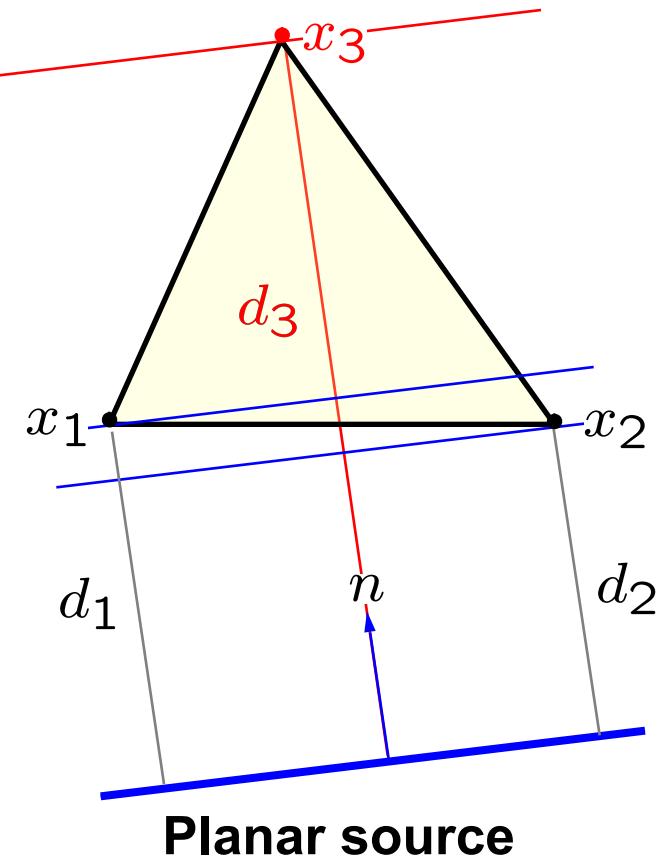
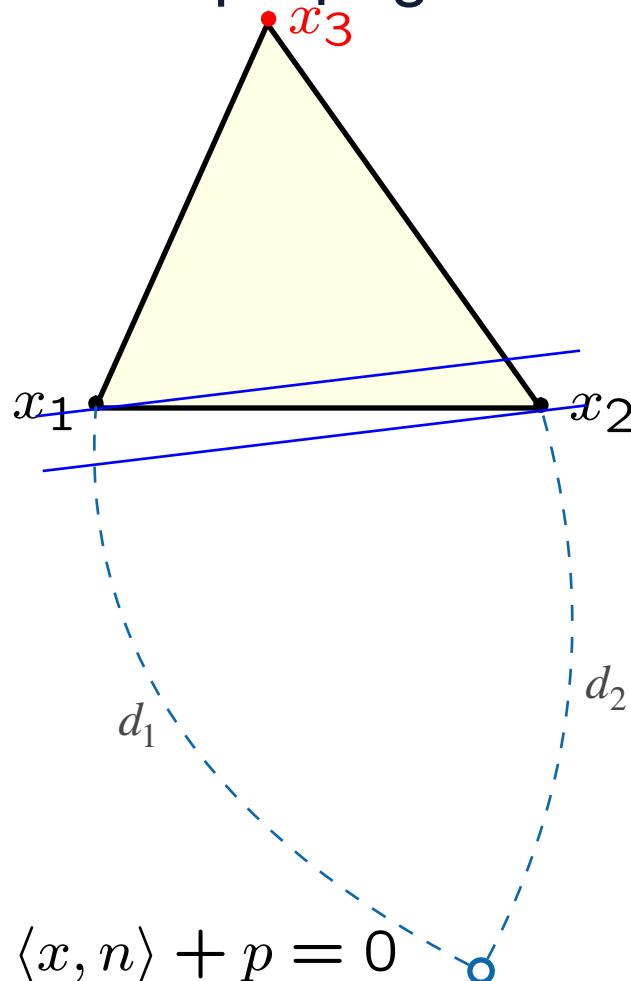


Trick: Every front looks flat!



Fast Marching Algorithm

- Change of view: point source → planar source
- Front propagates from **every point on the source**



Fast Marching Update Steps

- Front hits x_1 at time d_1
- Hits x_2 at time d_2
- When does the front arrive x_3 ?

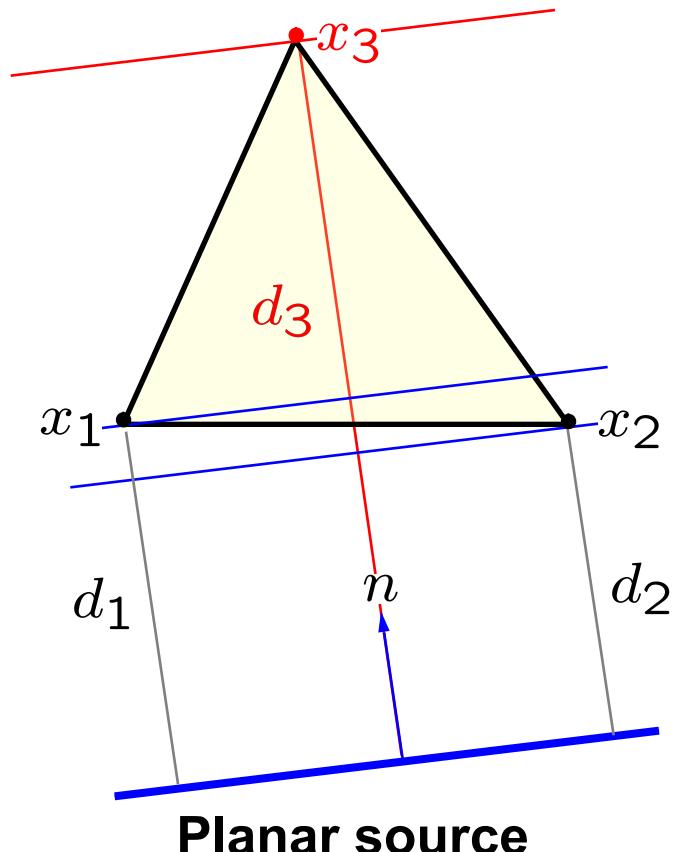
- Step 1: Model **wave front** propagation from a planar source

$$\langle x, n \rangle + p = 0$$

n : unit propagation direction

p : source offset

- Step 2: Compute the distance from x_3 to the plane



$$\langle x, n \rangle + p = 0$$

Practical Implementation

Fast Exact and Approximate Geodesics on Meshes

Vitaly Surazhsky
University of Oslo

Tatiana Surazhsky
University of Oslo

Danil Kirсанов
Harvard University

Steven J. Gortler
Harvard University

Hugues Hoppe
Microsoft Research

Abstract

The computation of geodesic paths and distances on triangle meshes is a common operation in many computer graphics applications. We present several practical algorithms for computing such geodesics from a source point to one or all other points efficiently. First, we describe an implementation of the exact “single source, all destination” algorithm presented by Mitchell, Mount, and Papadimitriou (MMP). We show that the algorithm runs much faster in practice than suggested by worst case analysis. Next, we extend the algorithm with a merging operation to obtain computationally efficient and accurate approximations with bounded error. Finally, to compute the shortest path between two given points, we use a lower-bound property of our approximate geodesic algorithm to efficiently prune the frontier of the MMP algorithm, thereby obtaining an exact solution even more quickly.

Keywords: shortest path, geodesic distance.

1 Introduction

In this paper we present practical methods for computing both exact and approximate shortest (i.e. geodesic) paths on a triangle mesh. These geodesic paths typically cut across faces in the mesh and are therefore not found by the traditional graph-based Dijkstra algorithm for shortest paths.

The computation of geodesic paths is a common operation in many computer graphics applications. For example, parameterizing a mesh often involves cutting the mesh into one or more charts (e.g. [Krishnamurthy and Levoy 1996]). The result generally has less distortion if the cuts are geodesic. Geodesic mesh into charts, as done in [Katz et al.

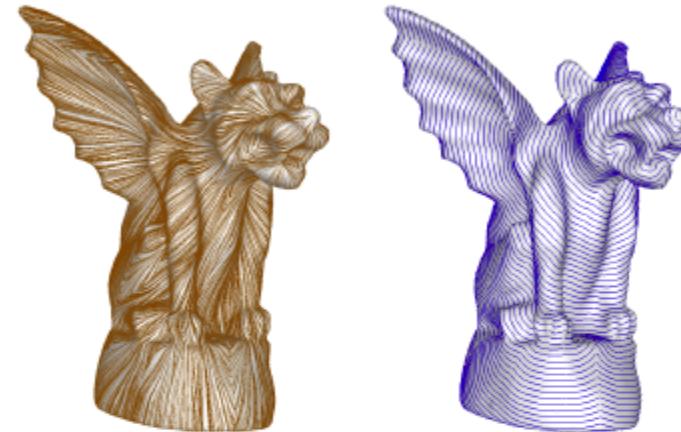


Figure 1: Geodesic paths from a source vertex, and isolines of the geodesic distance function.

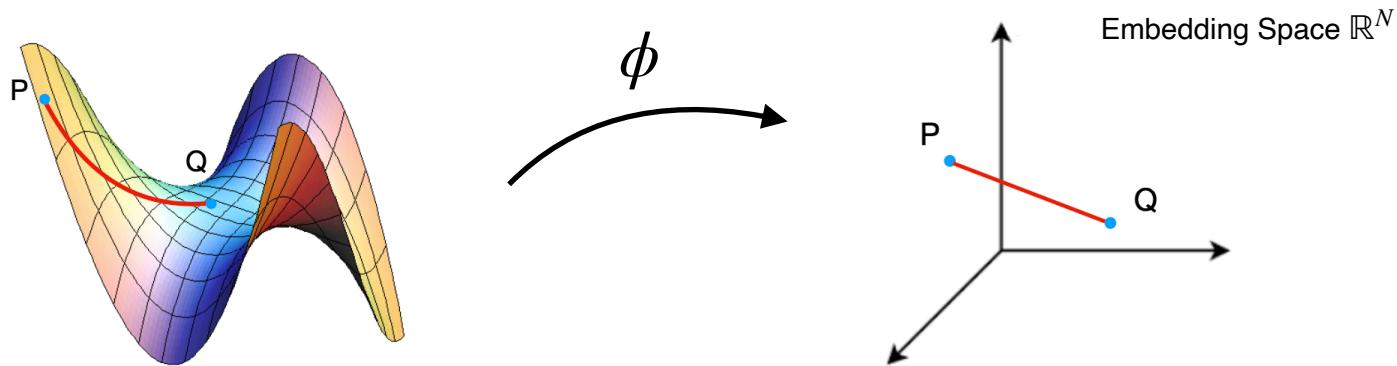
tance function over the edges, the implementation is actually practical even though, to our knowledge, it has never been done previously. We demonstrate that the algorithm’s worst case running time of $O(n^2 \log n)$ is pessimistic, and that in practice, the algorithm runs in sub-quadratic time. For instance, we can compute the exact geodesic distance from a source point to all vertices of a 400K-triangle mesh in about one minute.

Approximation algorithm We extend the algorithm with a merging operation to obtain computationally efficient and accurate approximations with *bounded* error. In practice, the algorithm runs in

<http://code.google.com/p/geodesic/>

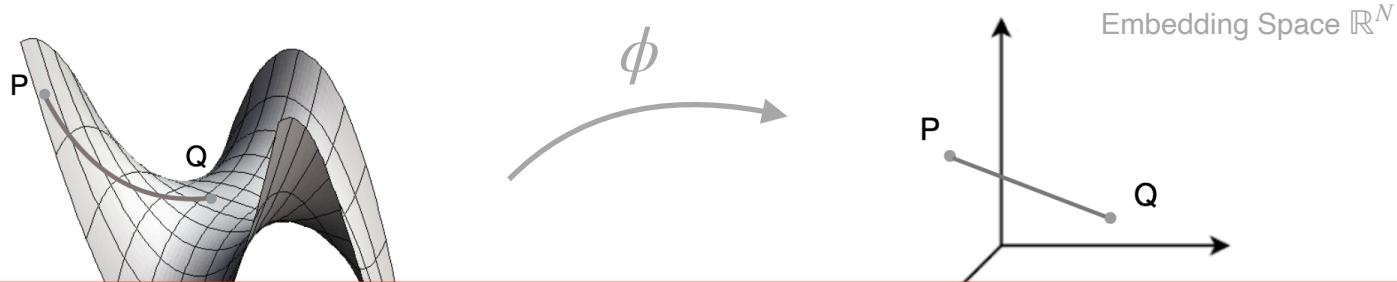
Learning to Predict Geodesics

Metric Learning Formulation



- Given points P, Q on the surface, what is $d_G(P, Q)$?
- Metric learning:
$$d_G(P, Q) \approx \|\phi(P) - \phi(Q)\|_p$$
- ϕ : embedding function (e.g., MLP)
- Usually, $\dim(\phi(\cdot)) \gg 3$ and $p = 2$

Metric Learning Formulation



**Q: Could “ \approx ” become “ $=$ ” with
 $\dim(\phi(\cdot)) < \infty$?**

- Metric learning.

$$d_G(P, Q) \approx \|\phi(P) - \phi(Q)\|_p$$

- ϕ : embedding function (e.g., MLP)
- Usually, $\dim(\phi(\cdot)) \gg 3$ and $p = 2$

Counter-Example

- Recall our $SO(2)$ example:
 - Distance preservation would preserve topological structure
 - Circle has different topology as line segments
 - So $SO(2)$ cannot be embedded into Euclidean space without distortion
- Some results of the embedding for general metric space to come

Some Result of Embedding

General Metrics in ℓ_p

Distortion of Embedding

- Metrics:
 - $d(x, y) \geq 0$
 - $d(x, x) = 0$ and $\forall y \neq x, d(x, y) > 0$
 - $d(x, y) + d(y, z) \geq d(x, z)$
- Assume $\phi : S \mapsto T$ and $d_S(\cdot, \cdot)$ and $d_T(\cdot, \cdot)$
 - $\text{expansion}(f) = \max_{x,y} \frac{d_T(\phi(x), \phi(y))}{d_S(x, y)}$
 - $\text{contraction}(f) = \max_{x,y} \frac{d_S(x, y)}{d_T(\phi(x), \phi(y))}$
 - $\text{distortion}(f) = \text{expansion} \times \text{contraction}$

A Well-Known Result

Theorem (Bourgain, 1985).

Let (X, d) be a metric space on n points. Then,

$$(X, d) \xrightarrow{O(\log n)} \ell_p^{O(\log^2 n)}$$

Remarks:

- **Any n -point** metric space (X, d) can be embedded in ℓ_p with distortion $O(\log n)$
- To accommodate more points, one needs higher dim space
- This bound can be shown tight

Consequence

- We cannot expect that a finite dimensional ℓ_p space would approximate any metric
- To compromise, we usually require the approximation to have higher precision for closer points:
 - Closer points need to have precise distances
 - Far away points just need to be known far away
- Recall the hinge loss for point feature learning in the *bottom-up instance segmentation* lecture:

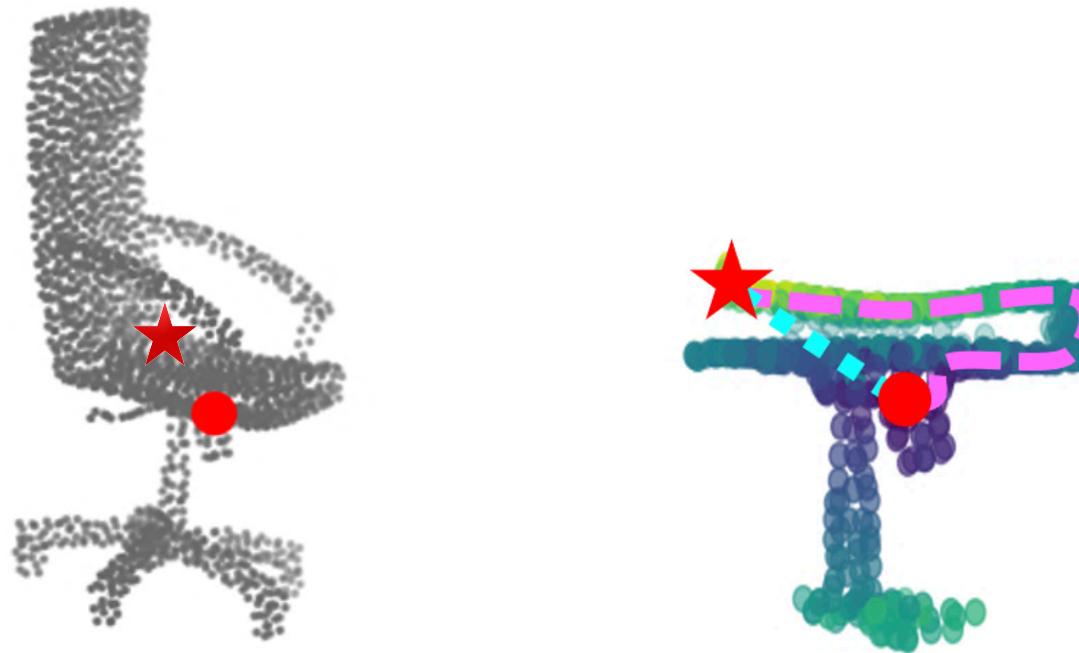
$$L_{ij} = \max(0, K - \|F_i - F_j\|)$$

Learning to Predict Geodesics

Example work: GeoNet

Geodesics for Point Clouds

- No connectivity information

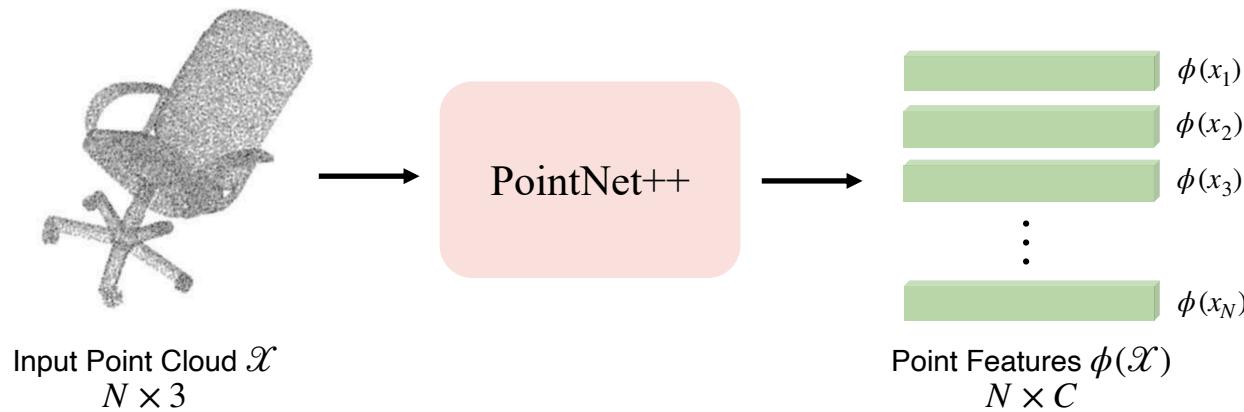


Task Formulation

- Input: Point Cloud $\mathcal{X} = \{x_i\}_{i=1}^N$
- Output: Geodesic distance to K Euclidean neighbors for each point

GeoNet: Geodesic Distance Regression

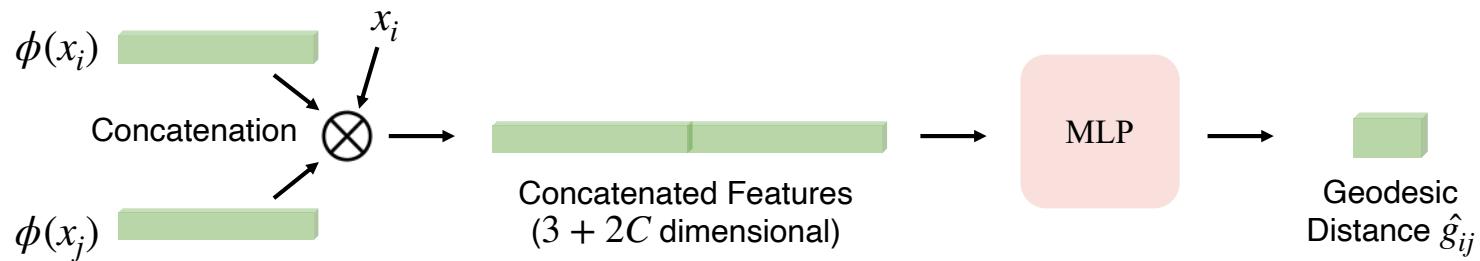
Step 1: Feature Extraction



GeoNet: Geodesic Distance Regression

Step 2: Metric Learning

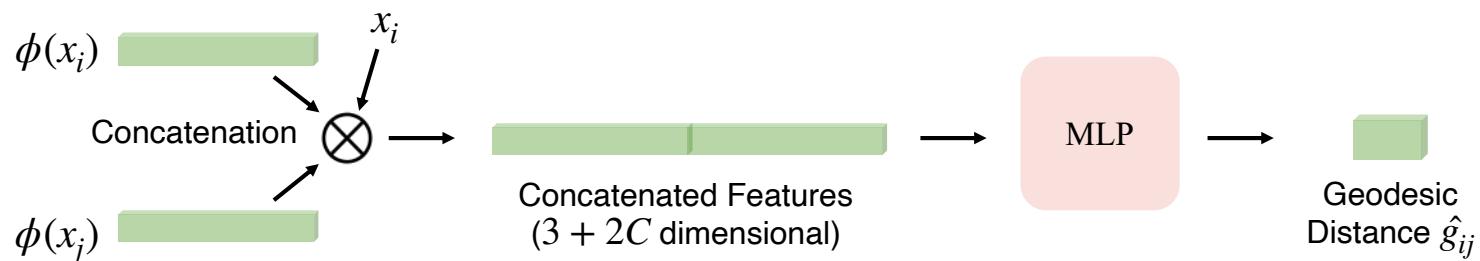
For each x_i and its neighbor x_j :



GeoNet: Geodesic Distance Regression

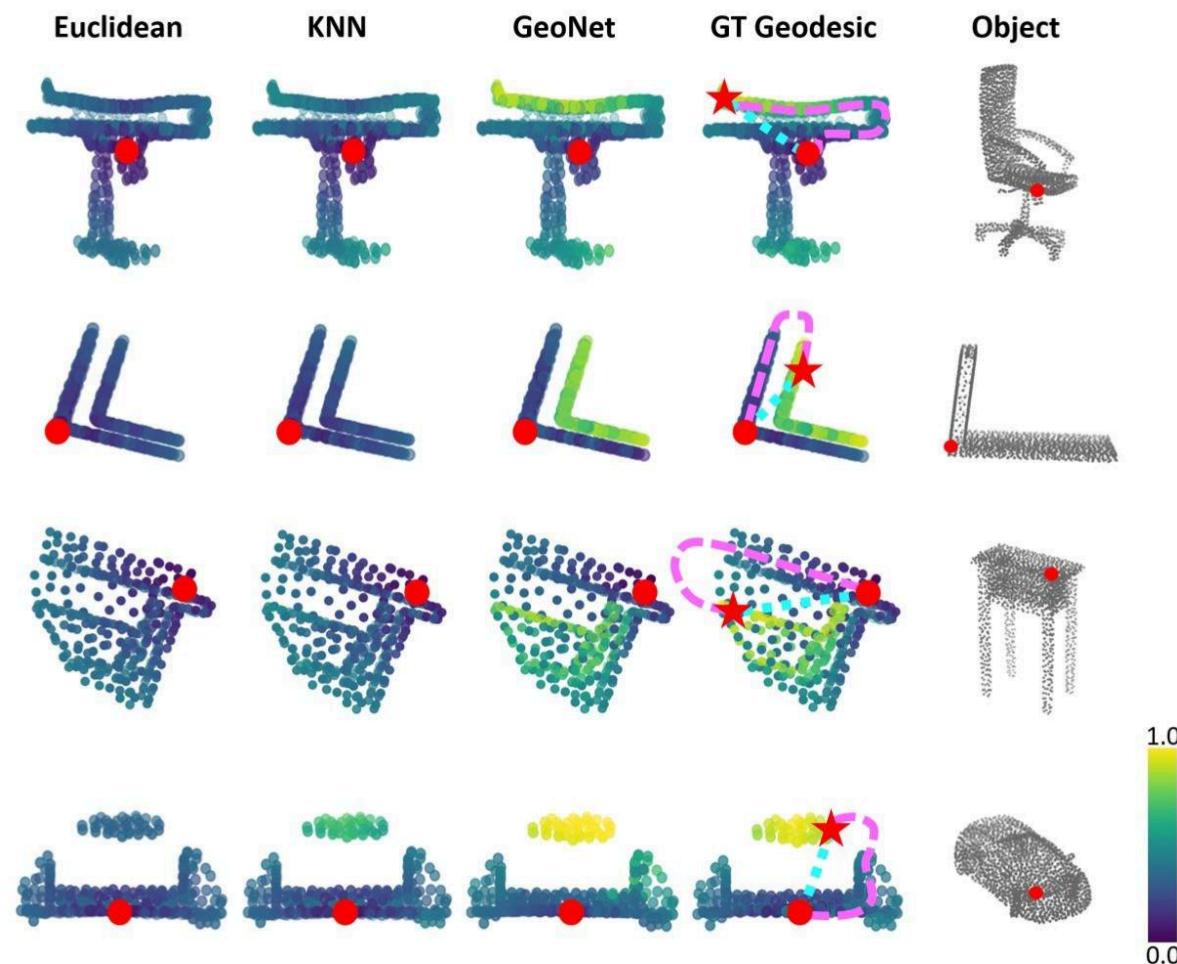
Step 2: Metric Learning

For each x_i and its neighbor x_j :



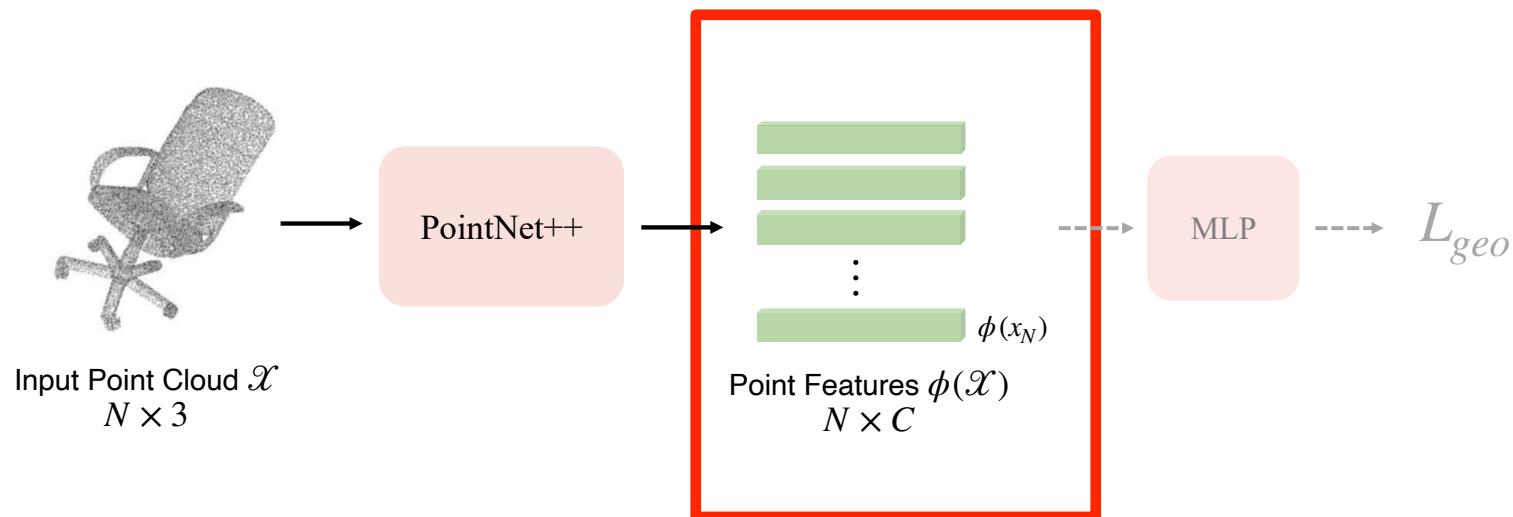
$$L_{geo} = \sum_{x_i \in \mathcal{X}} \sum_{x_j \in NN(x_i)} |g_{ij} - \hat{g}_{ij}|$$

Geodesics Regression



Geodesics Induced Features

- Geodesics induced features are useful for downstream tasks



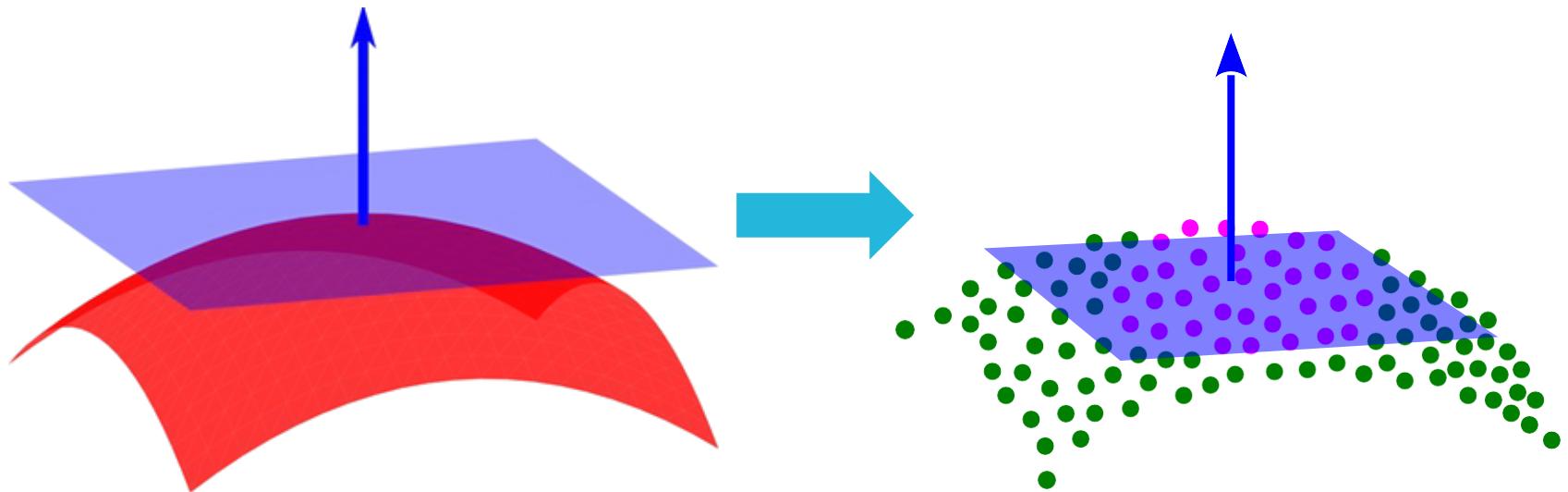
Applications

- Normal Estimation
- Mesh reconstruction

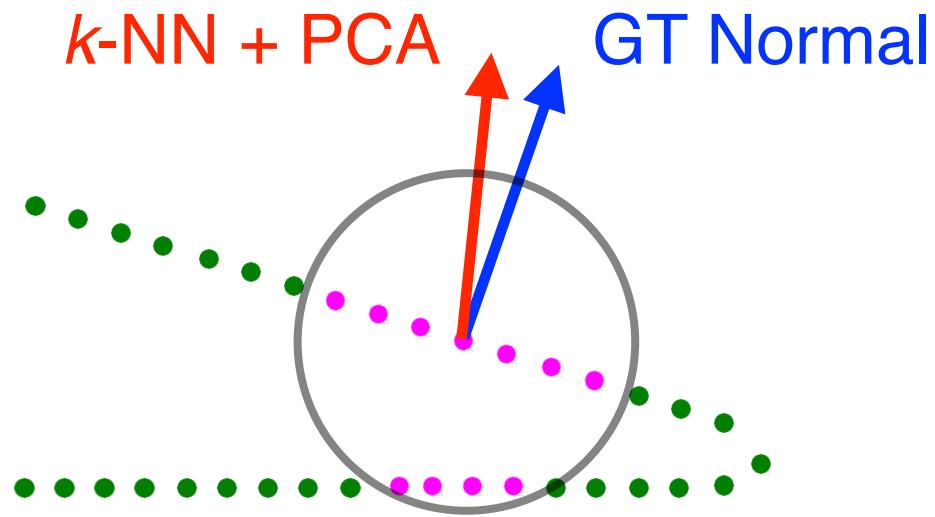
Recall: Normal Estimation

- Plane-fitting: find the plane that best fits the neighborhood of a point of interest

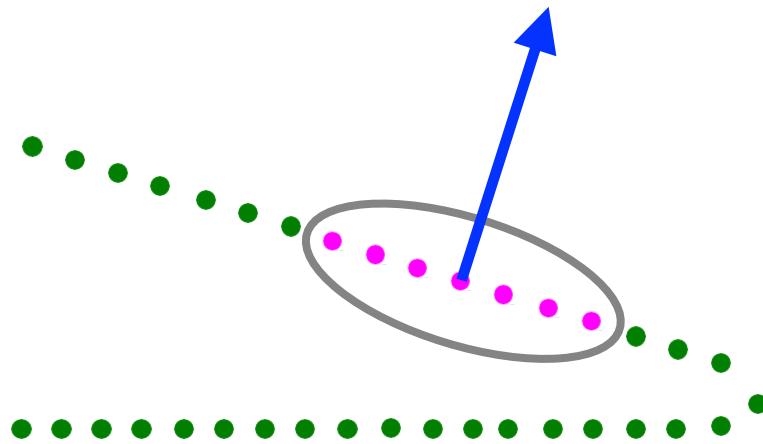
- Let $M = \sum_i (x_i - \bar{x})(x_i - \bar{x})^T$ and $\bar{x} = \frac{1}{n} \sum_i x_i$,
- n : the last principal component of M



Normal Estimation

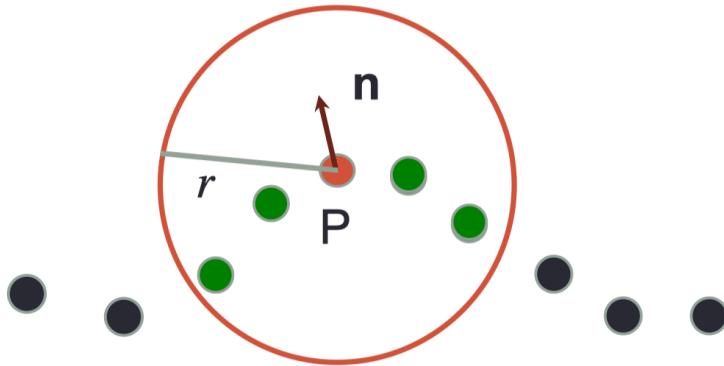


Normal Estimation



$d_G(u, v)$ is large although $d_E(u, v)$ is small!
⇒ Use geodesic neighborhoods for PCA

Normal Estimation



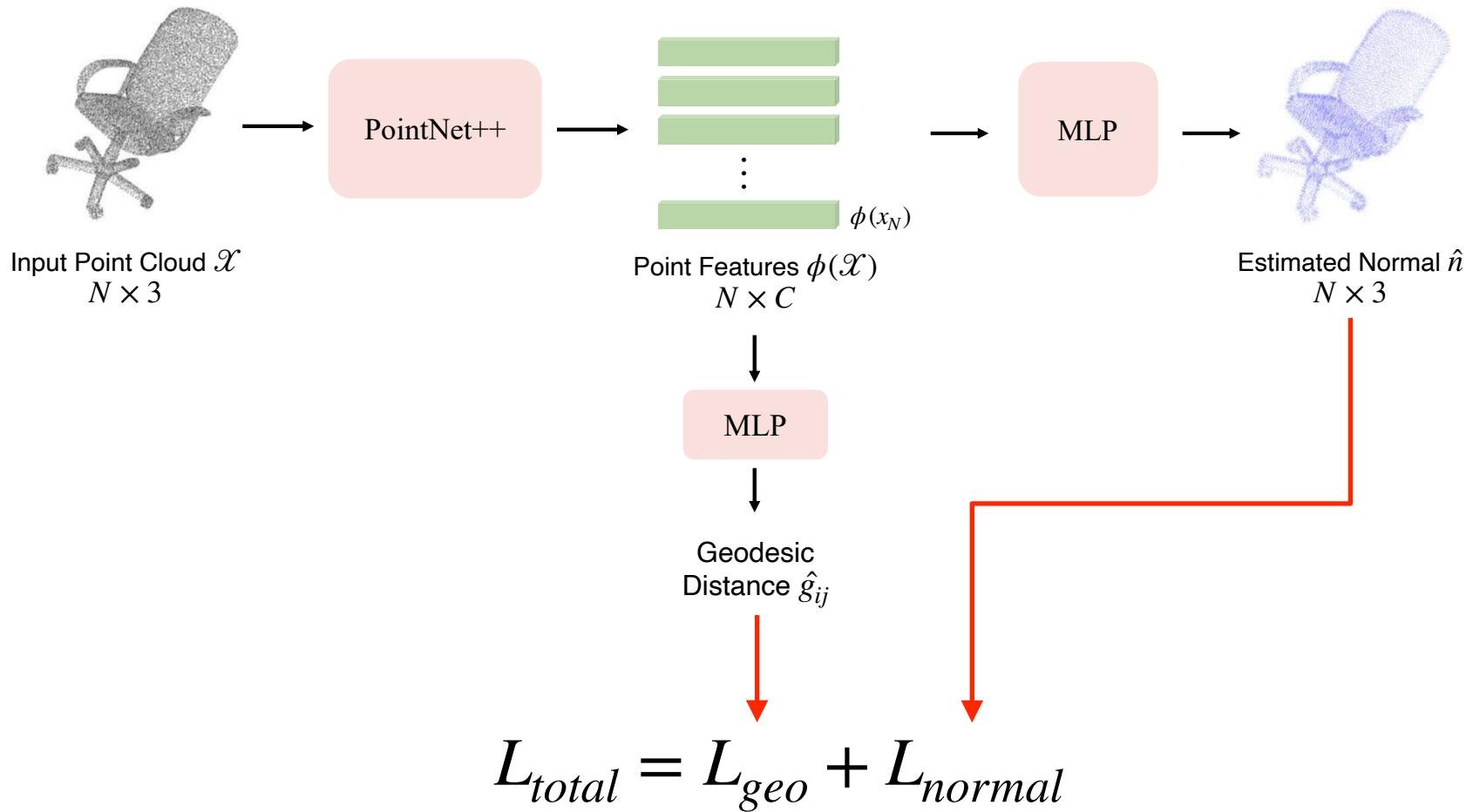
Critical parameter: number of neighbors k .

For unevenly sampled point cloud, we typically use all points inside a ball of radius r

Tricky to choose in practice!

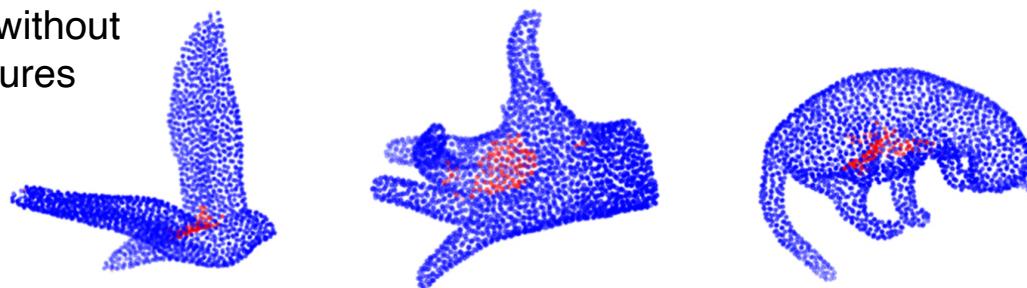
How about **predicting the normal**,
without PCA and estimation of r or k ?

Learning to Regress Normals

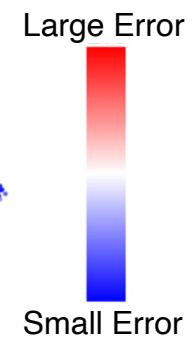


Error Patterns

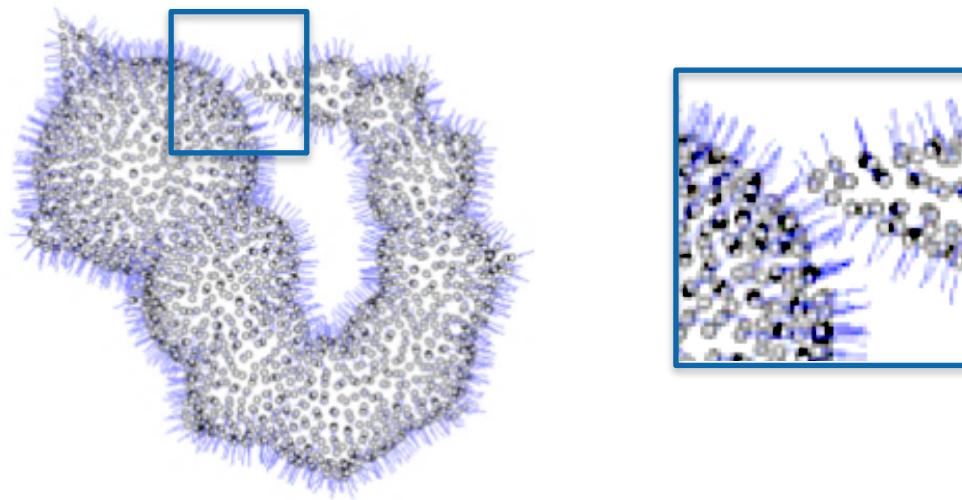
PointNet++ without
GeoNet features



PointNet++ with
GeoNet features



Learning to Regress Normals



Advantage of Learning-based Normal Estimation

- Can be robust to point cloud sampling strategy
- Without the need to determine neighborhood size
- Get orientation consistent normals (in classical methods, this step is highly non-trivial)

Applications

- Normal Estimation
- Mesh Reconstruction

Mesh Reconstruction

- Input: Point Cloud
- Output: Polygon Mesh

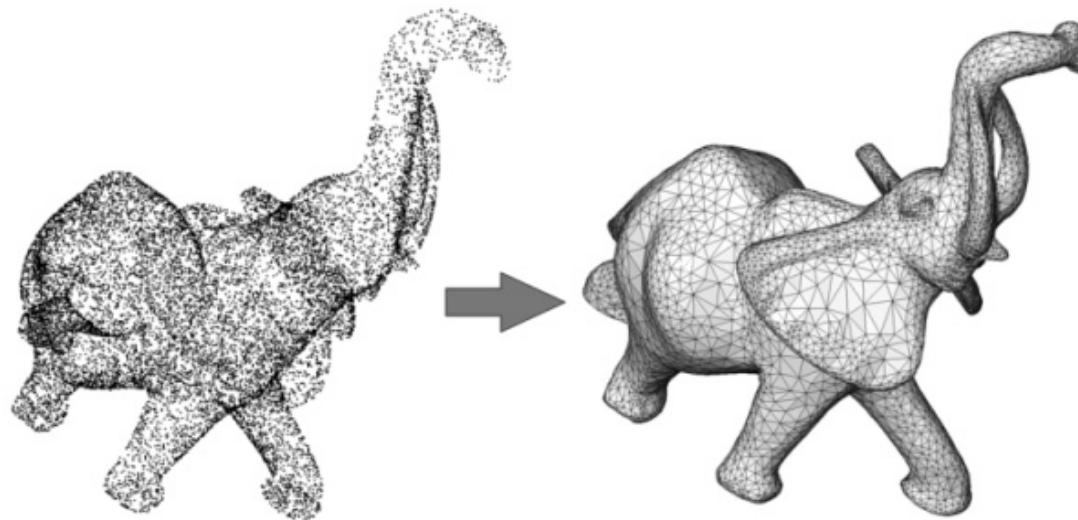


Figure courtesy of Pierre Alliez,
Laurent Saboret, Gaël Guennebaud

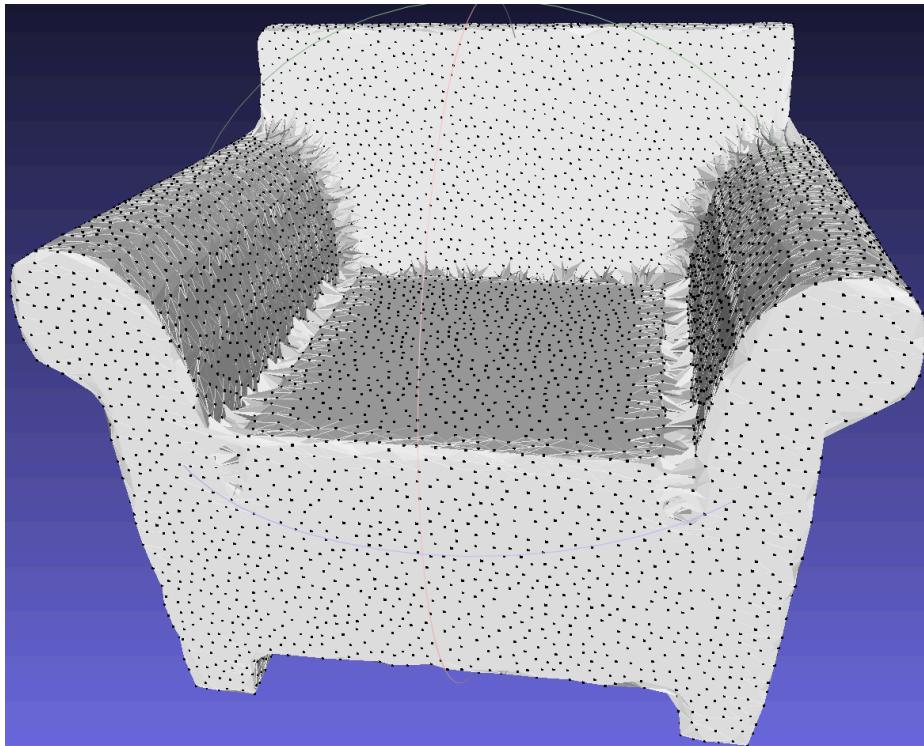
A Simple K -NN Approach

- For each point x_i we try to greedily add all triangles formed by x_i and its neighbors:

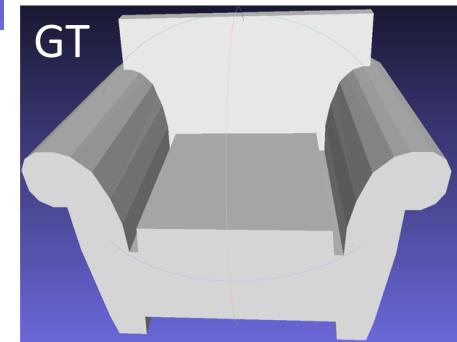
$\chi = \{x_i\}^n$ is the input point cloud
 $S = \{\}$ is the output mesh triangles

```
for  $i = 1$  to  $n$ :  
    find  $K$  nearest neighbors  $\{p_l\}_{l=1}^K$  of  $x_i$ :  
    for  $j = 1$  to  $K$ :  
        for  $k = j + 1$  to  $K$ :  
            triangle  $\triangle = (x_i, N_j, N_k)$   
            if  $\triangle$  is not in  $S$  and  $S$  is manifold after adding  $\triangle$ :  
                add  $\triangle$  to  $S$ 
```

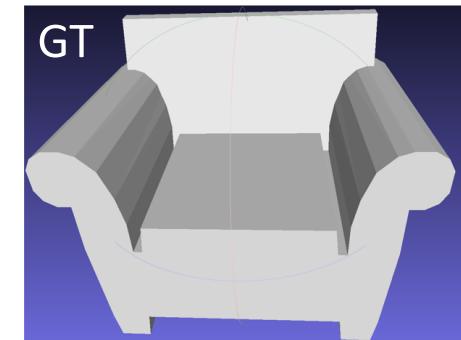
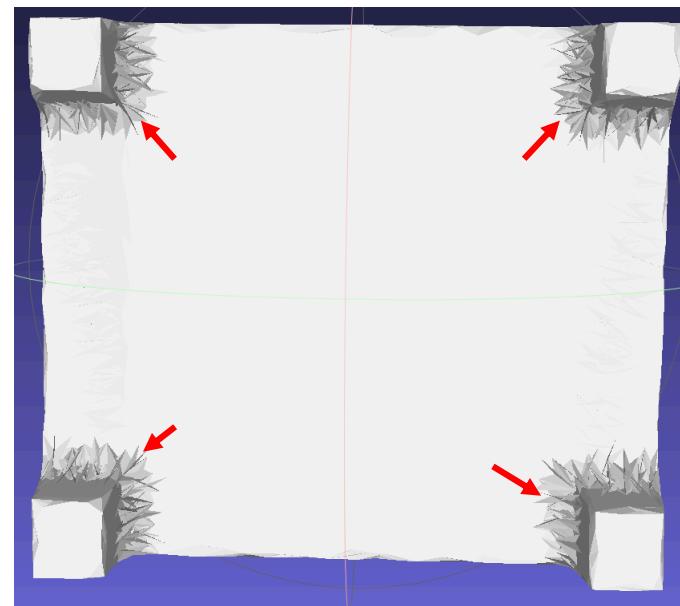
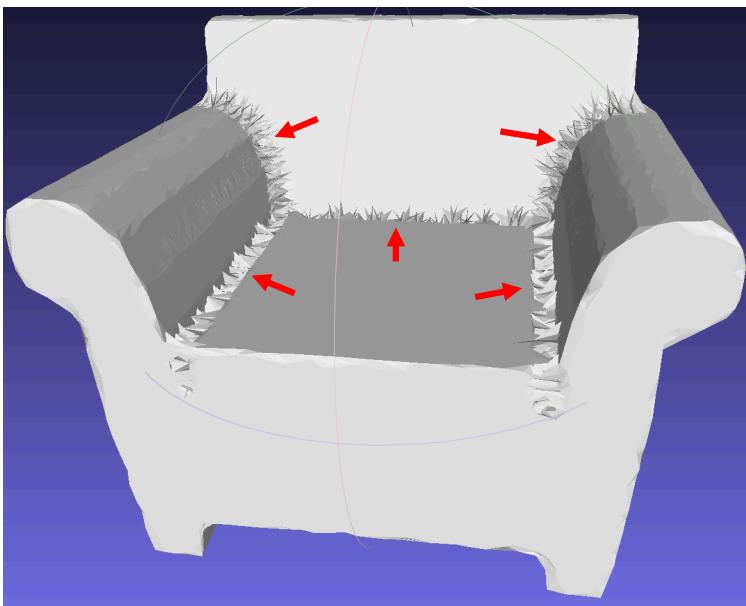
Results not bad...



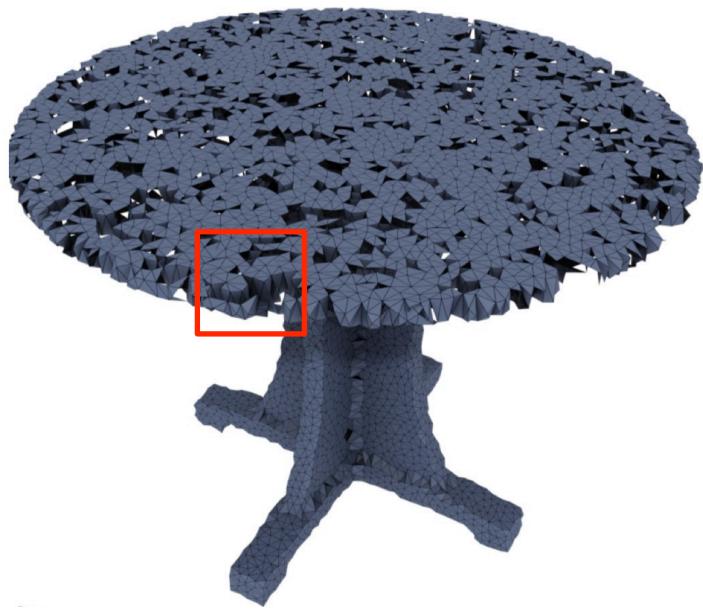
$K = 10$



But has issues at boundaries



and some bad triangles



- Triangles formed by connecting the upper surface and the bottom surface of the tabletop (intrinsically far)

and some bad triangles



**Q: Do we have a certificate
of good triangles?**



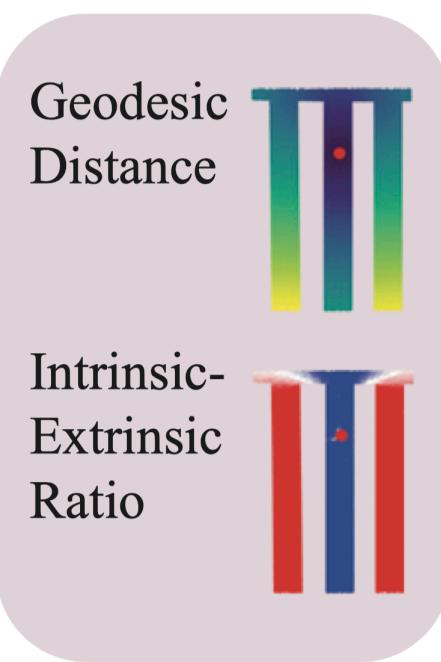
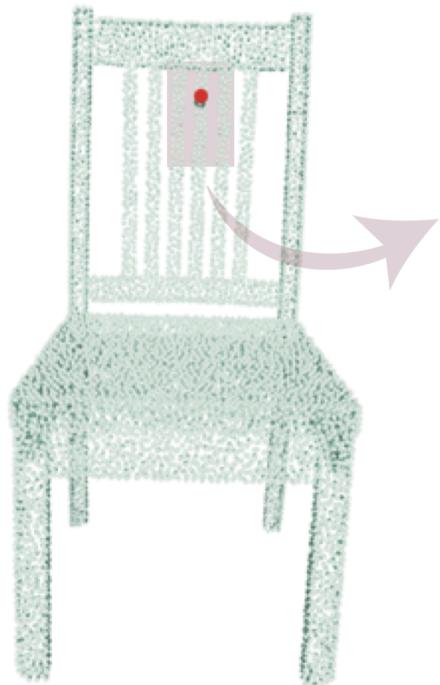
- Triangles formed by connecting the upper surface and the bottom surface of the tabletop (intrinsically far)

Intrinsic-Extrinsic Ratio

Given two vertices u, v on the surface, the intrinsic-extrinsic ratio is defined as:

$$\text{IER}(u, v) = \frac{d_G(u, v)}{d_E(u, v)}$$

Visualization



Intrinsic-Extrinsic Ratio for Triangles

For a triangle $\triangle uvw$:

$$\text{IER}(\triangle uvw) = \frac{d_G(u, v) + d_G(v, w) + d_G(w, u)}{d_E(u, v) + d_E(v, w) + d_E(w, u)}$$

If point u, v, w is **close enough**, $\text{IER}(\triangle uvw) \leq 1 + \varepsilon$
 \Rightarrow (very likely) Triangle is on the surface

Intrinsic-Extrinsic Ratio for Triangles

For a triangle $\triangle uvw$:

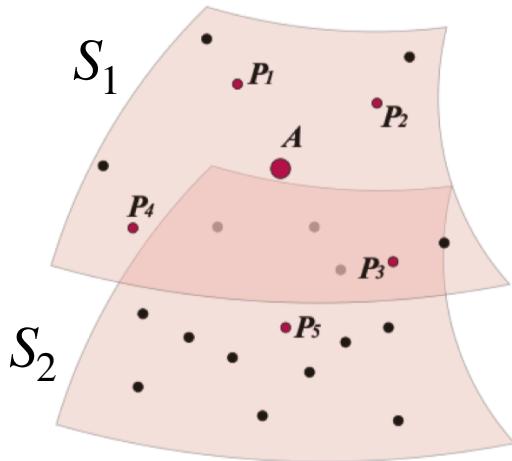
$$\text{IER}(\triangle uvw) = \frac{d_G(u, v) + d_G(v, w) + d_G(w, u)}{d_E(u, v) + d_E(v, w) + d_E(w, u)}$$

If point u, v, w is **close enough**, $\text{IER}(\triangle uvw) \leq 1 + \varepsilon$
 \Rightarrow (very likely) Triangle is on the surface

We learn to predict this quantity!

Meshing Point Clouds with Intrinsic-Extrinsic Ratio Guidance

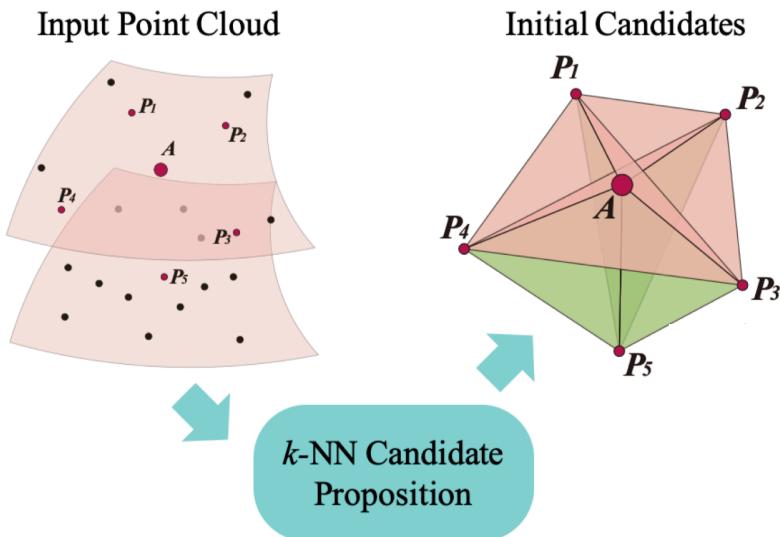
Input Point Cloud



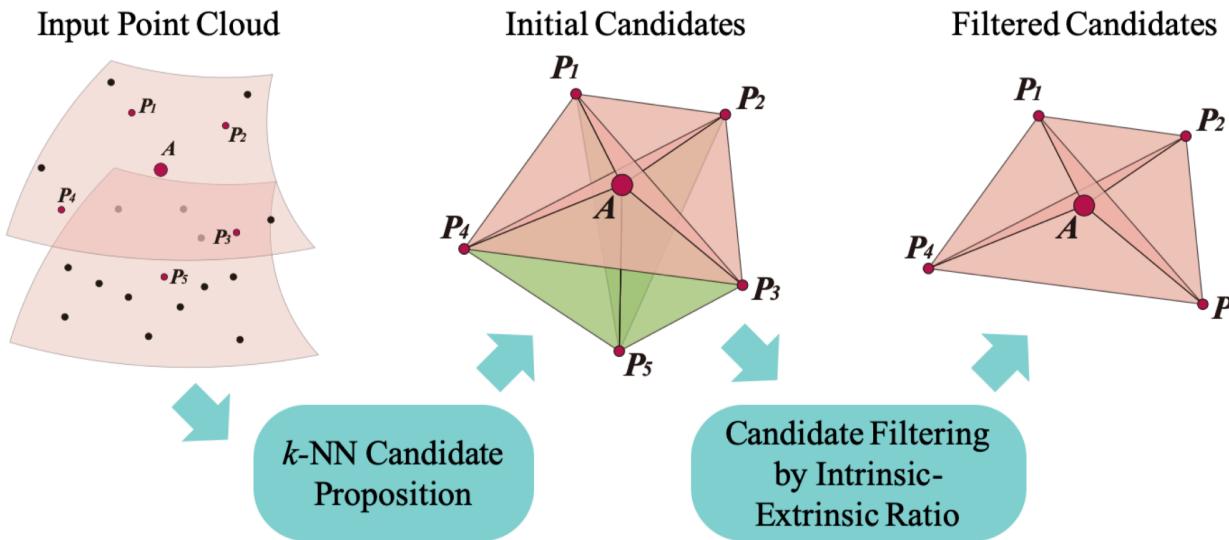
Case Study:

- Input point cloud is sampled from two closed thin surfaces S_1 and S_2
- $\{P_i\}$ are Euclidean nearest neighbors of A
 - $\{A, P_1, P_2, P_3, P_4\} \subset S_1$
 - $P_5 \in S_2$

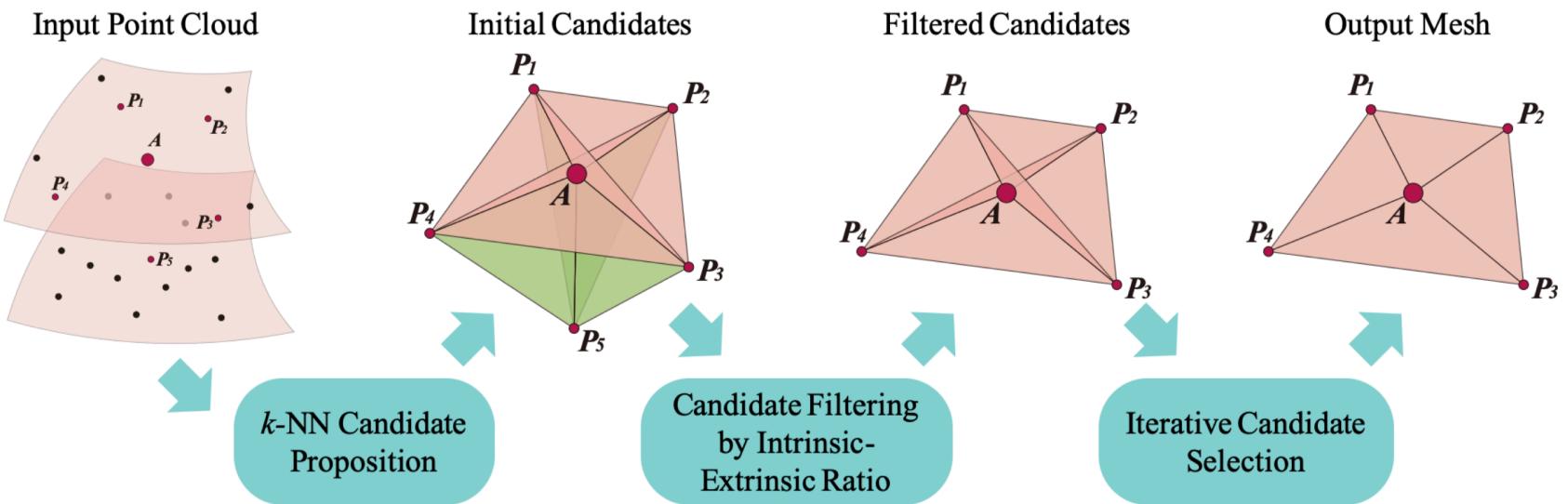
Meshing Point Clouds with Intrinsic-Extrinsic Ratio Guidance



Meshing Point Clouds with Intrinsic-Extrinsic Ratio Guidance

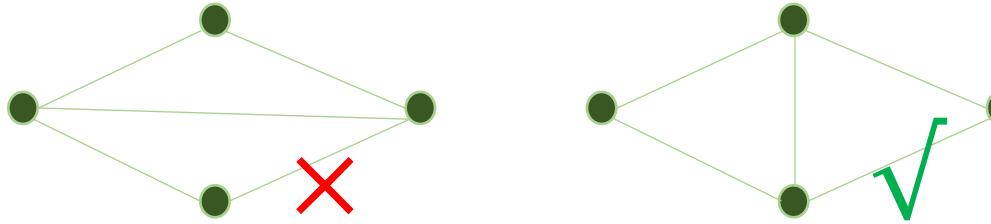


Meshing Point Clouds with Intrinsic-Extrinsic Ratio Guidance



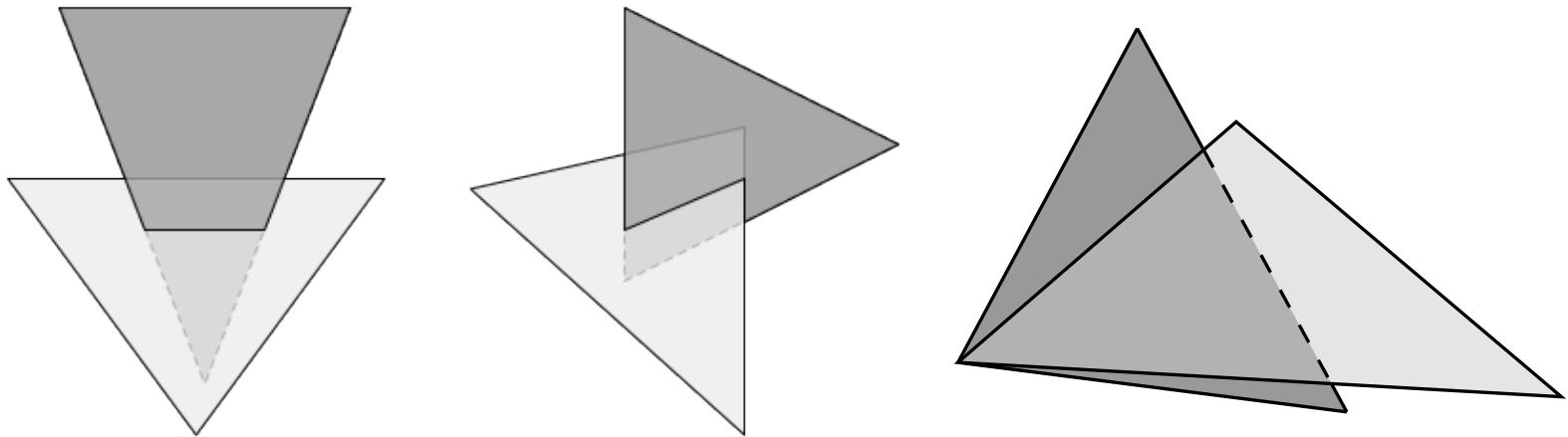
Iterative Candidate Selection

- The added triangle will not break manifold properties
 - No edge has more than two incident faces
 - No intersection between triangles
- Prefer equilateral triangles:



Measured by the ratio of longest edge to shortest edge

Triangle Collision Detection



Guigue P, Devillers O. Fast and robust triangle-triangle overlap test using orientation predicates. Journal of Graphics Tools. 2003 Jan 1;8(1):25-32.

Code: https://github.com/erich666/jgt-code/tree/master/Volume_08/Number_1/Guigue2003

Meshing Results

k -NN

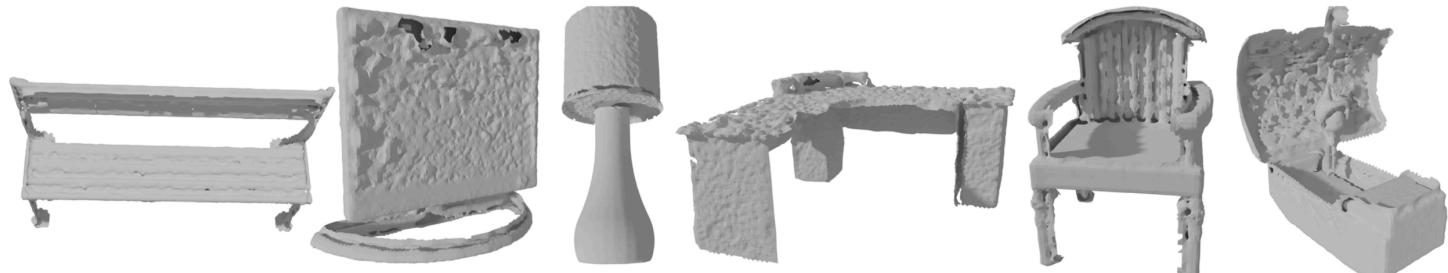


k -NN
with IER

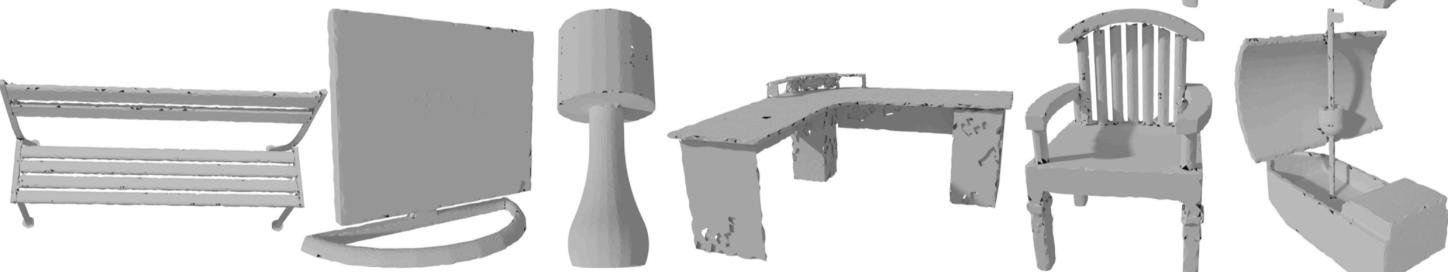


Robustness

Poisson



k -NN
with IER



Ground-
Truth

