Surface Reconstruction II

USTC, 2024 Spring

Qing Fang, fq1208@mail.ustc.edu.cn

https://qingfang1208.github.io/

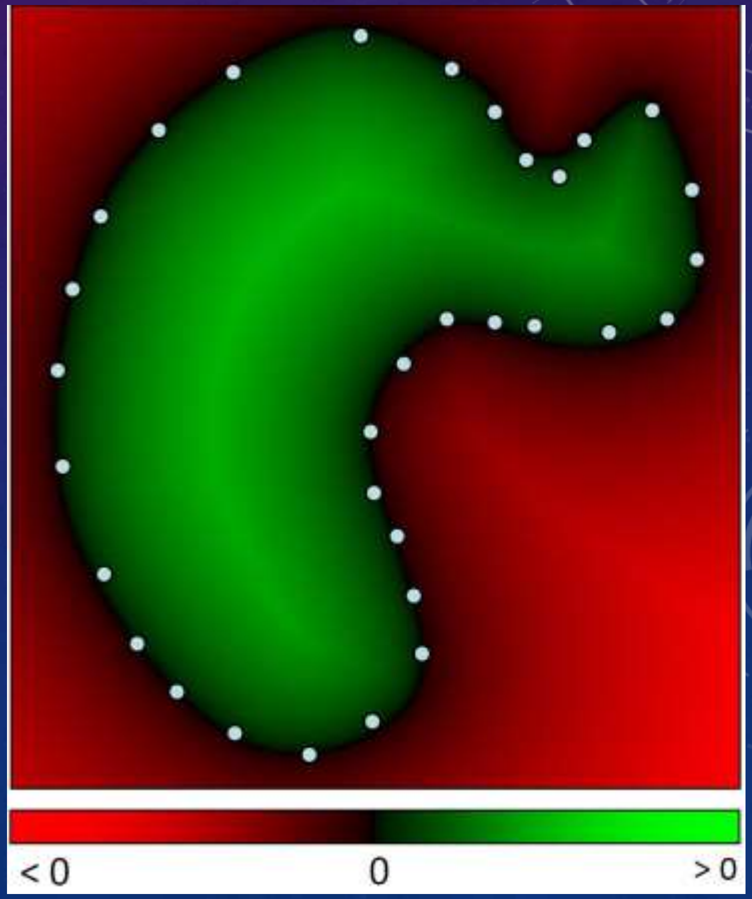
Reconstruction methods

- Explicit methods
  - VD and DT, Alpha shape, Zippering, …
- Implicit methods

Interior: $F(x) < 0$

Exterior: $F(x) > 0$

Surface: $F(x) = 0$



Implicit methods

- Two basic steps:

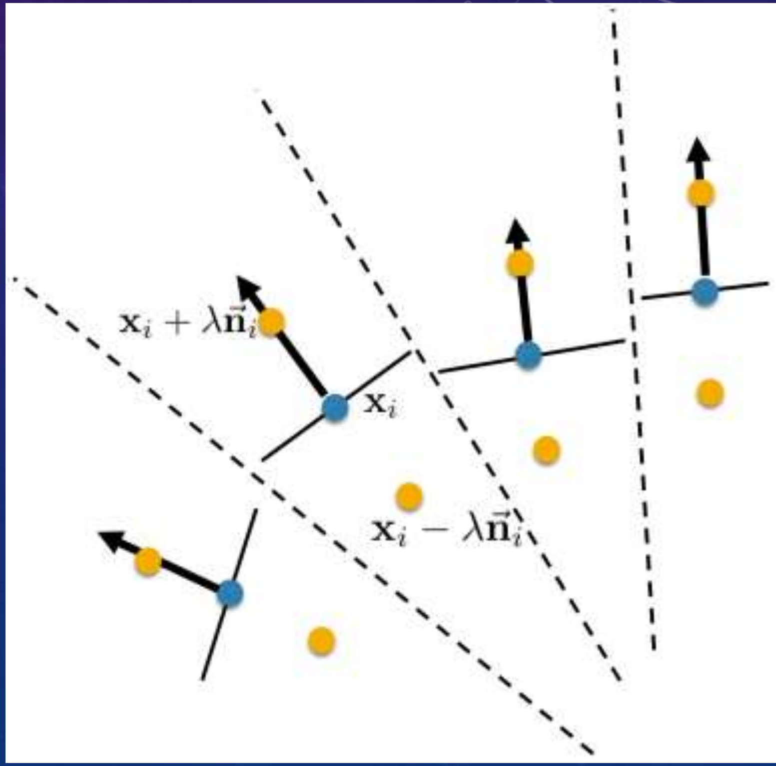
  1. Estimate an implicit field function from data

  2. Extract the zero iso-surface

$F(x_i) = 0$ not enough, may $F(x) \equiv 0$

Use normal to add off-surface points:

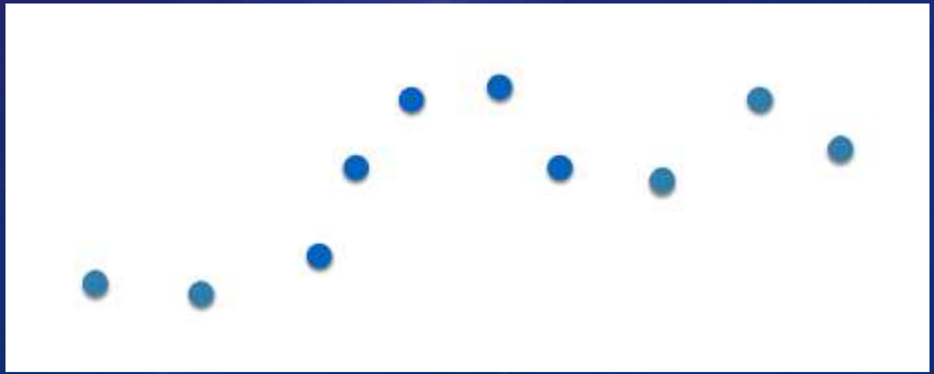
$$\begin{cases} F(x_i + \lambda n_i) > 0 \\ F(x_i - \lambda n_i) < 0 \end{cases}$$



Estimating normals

- Estimate the normal vector for each point

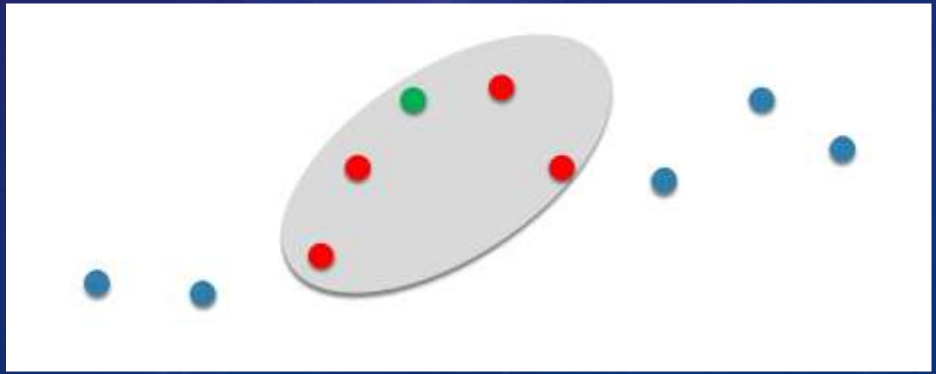
1. Extract the k-nearest neighbor point
2. Compute the best approximating tangent plane by covariance analysis
3. Compute the normal orientation



Estimating normals

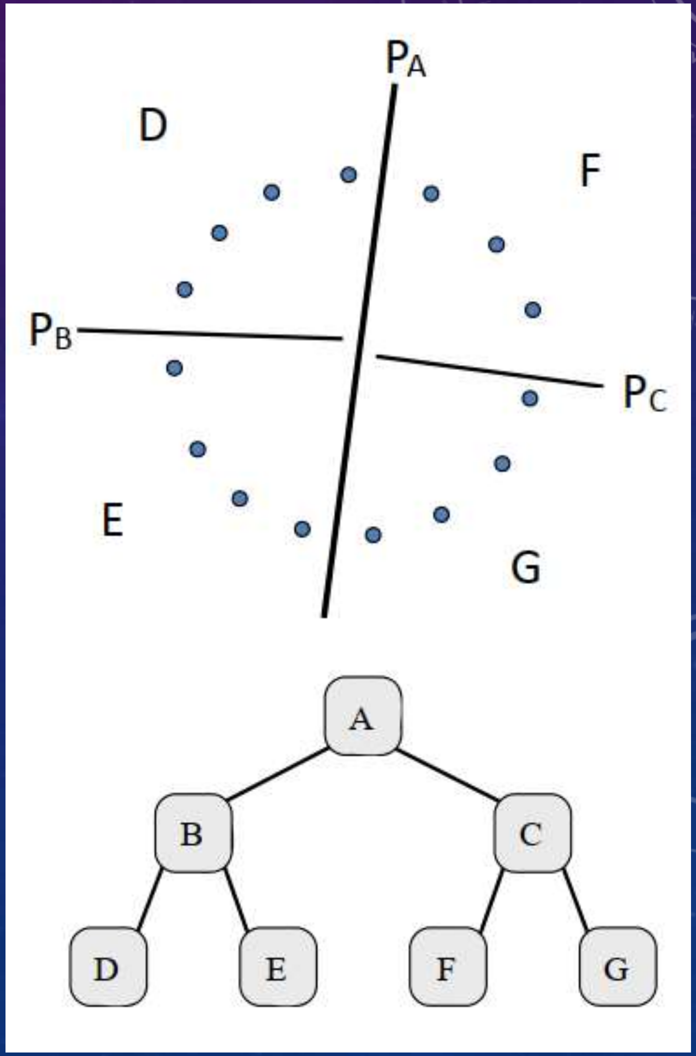
- Estimate the normal vector for each point

1. Extract the k-nearest neighbor point
2. Compute the best approximating tangent plane by covariance analysis
3. Compute the normal orientation



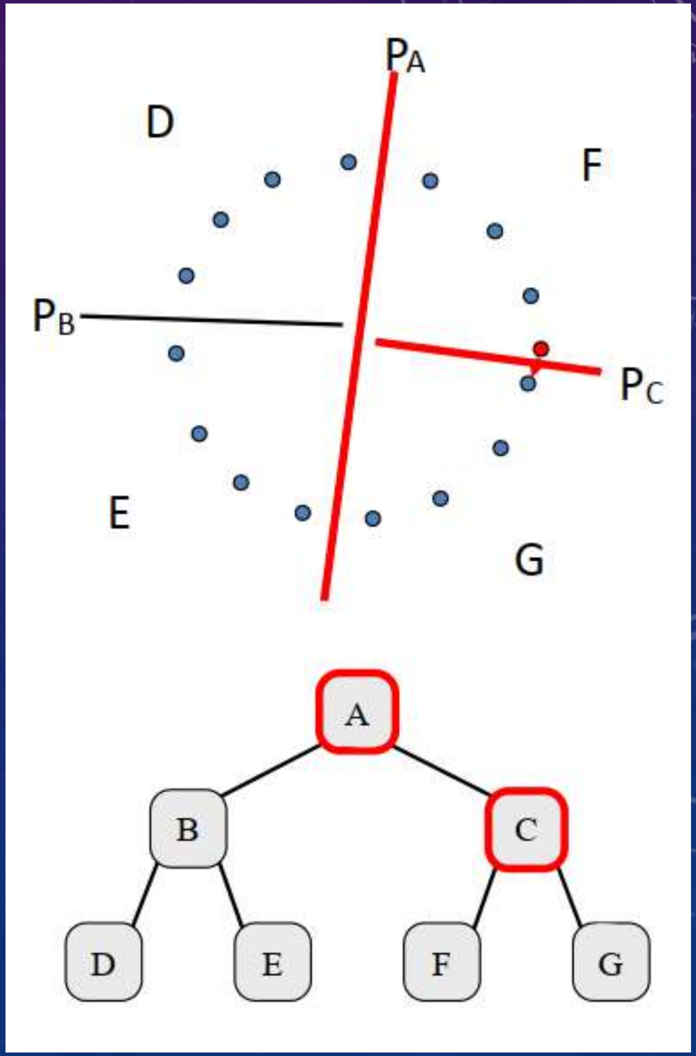
Local neighborhood

- Find k nearest neighbors (kNN) of a point
  - Brute force: O(n) complexity
- Use BSP tree
  - Binary space partitioning tree
  - Recursively partition 3D space by planes
  - Tree should be balanced, put plane at median
  - log(n) tree levels, complexity O(log n)

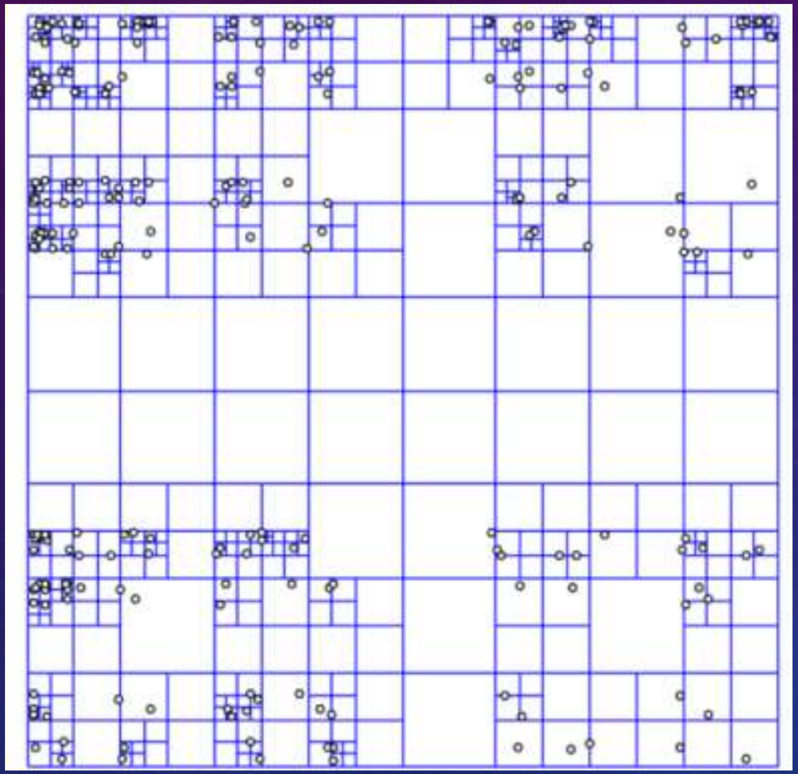


BSP Closest Points

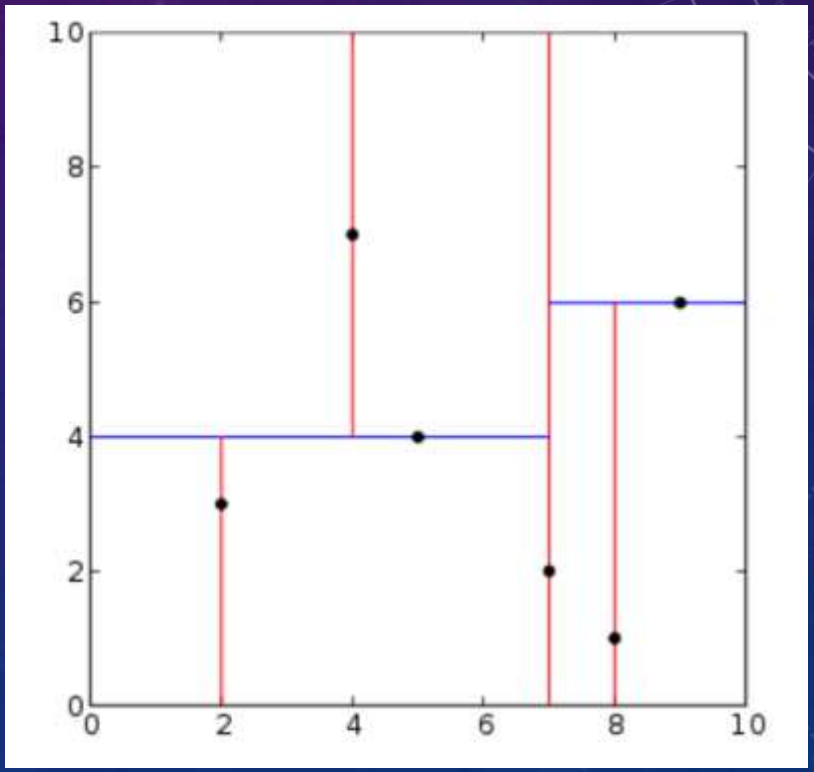
```
Node::dist(Point x, Scalar& dmin)
{
 if (leaf_node())
  for each sample point p[i]
   dmin = min(dmin, dist(x, p[i]));

 else
 {
  d = dist_to_plane(x);
  if (d < 0)
  {
   left_child->dist(x, dmin);
   if (|d| < dmin) right_child->dist(x, dmin);
  }
  else
  {
   right_child->dist(x, dmin);
   if (|d| < dmin) left_child->dist(x, dmin);
  }
 }
}
```



More Trees



Quad-tree (oct-tree)
Cells are squares (cubes)

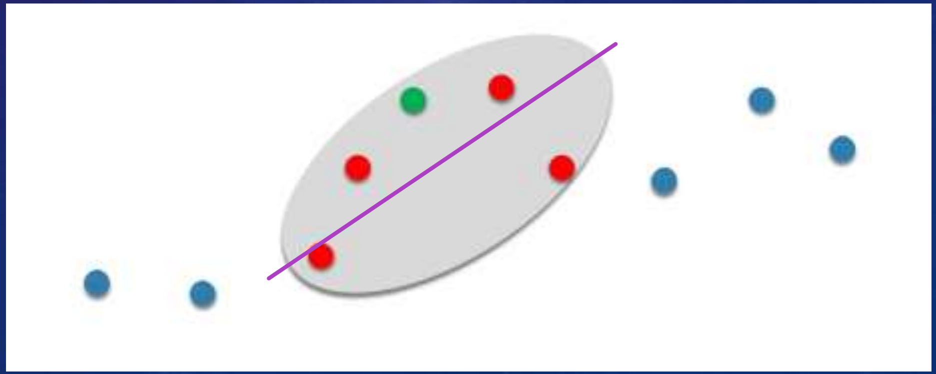


Kd-tree
Cells are axis-aligned boxes

Estimating normals

- Estimate the normal vector for each point

1. Extract the k-nearest neighbor point
2. Compute the best approximating tangent plane by covariance analysis
3. Compute the normal orientation



Principal component analysis

- Fit a plane with center c and normal $\vec{n}$ to a set of $\{x_1, \dots, x_k\}$

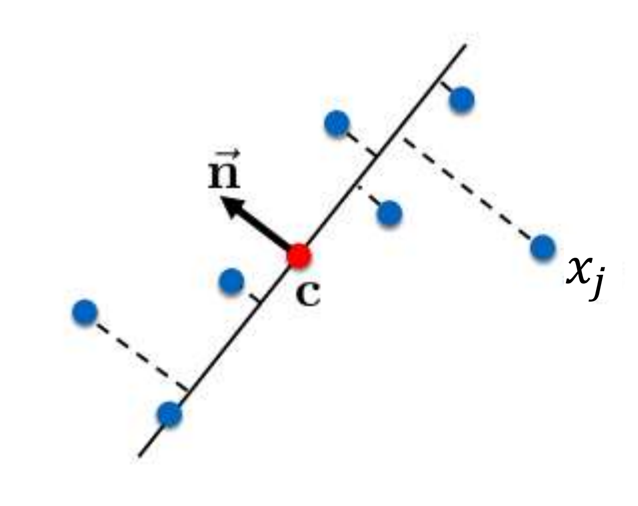
Minimize least squares error subject to normalization constraint

$$\min_{c,\vec{n}} \sum_{j=0}^{k} \left(\vec{n}^T (x_j - c)\right)^2$$

Close-form solution : let $c = \frac{1}{k} \sum_{j=0}^{k} x_j$

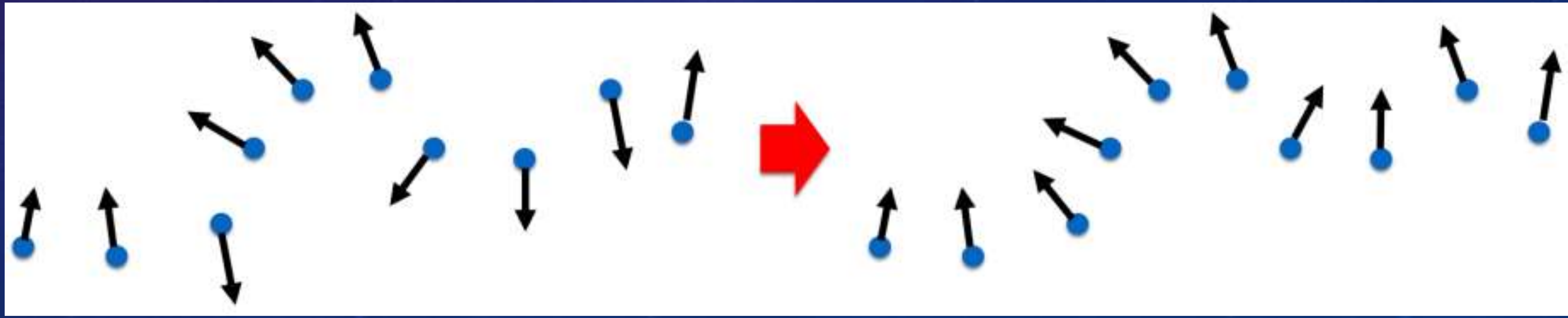
set $M = PP^T, P = [x_1 - c, \dots, x_k - c]$, then

$\vec{n}$ is the eigenvector of M with the smallest eigenvalue



Estimating normals

- Estimate the normal vector for each point
  1. Extract the k-nearest neighbor point
  2. Compute the best approximating tangent plane by covariance analysis
  3. Compute the normal orientation

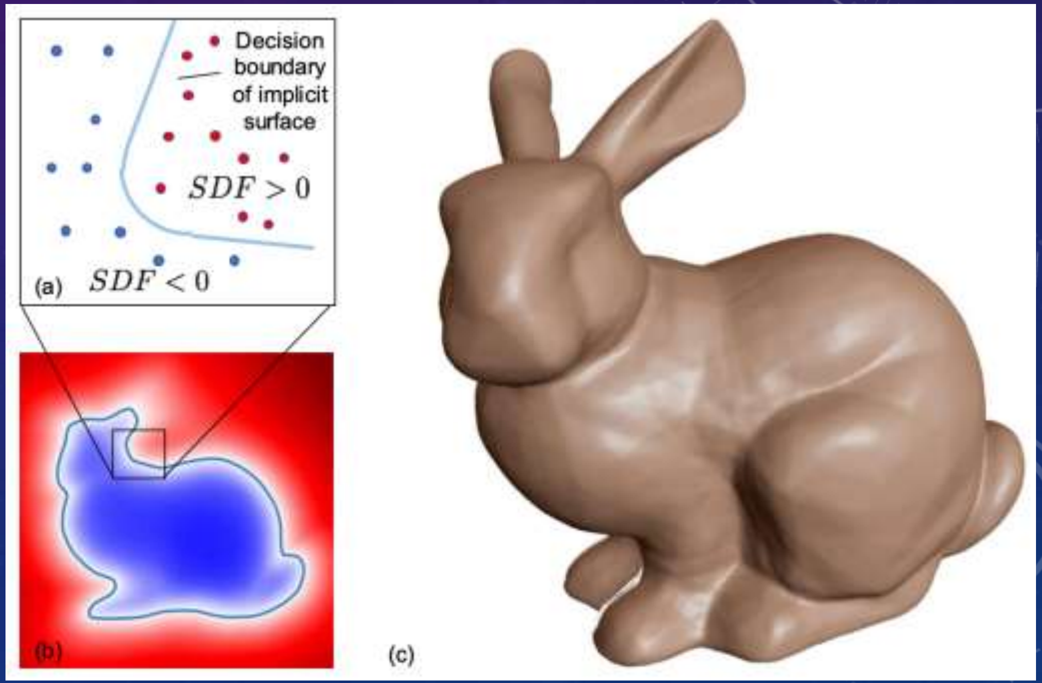


Normal Orientation

- Build graph connecting neighboring points
  - Edge (i, j) exists if $x_i \in kNN(x_j)$ or $x_j \in kNN(x_i)$
- Propagate normal orientation through graph
  - For edge (i, j) flip $\vec{n}_j$ if $\vec{n}_j^T \vec{n}_\mathrm{i} < 0$
  - Fails at sharp edges/corners
- Propagate along "safe" paths
  - Build a minimum spanning tree with weights $w_{ij} = 1 - |\vec{n}_j^T \vec{n}_\mathrm{i}|$

Reconstruction methods

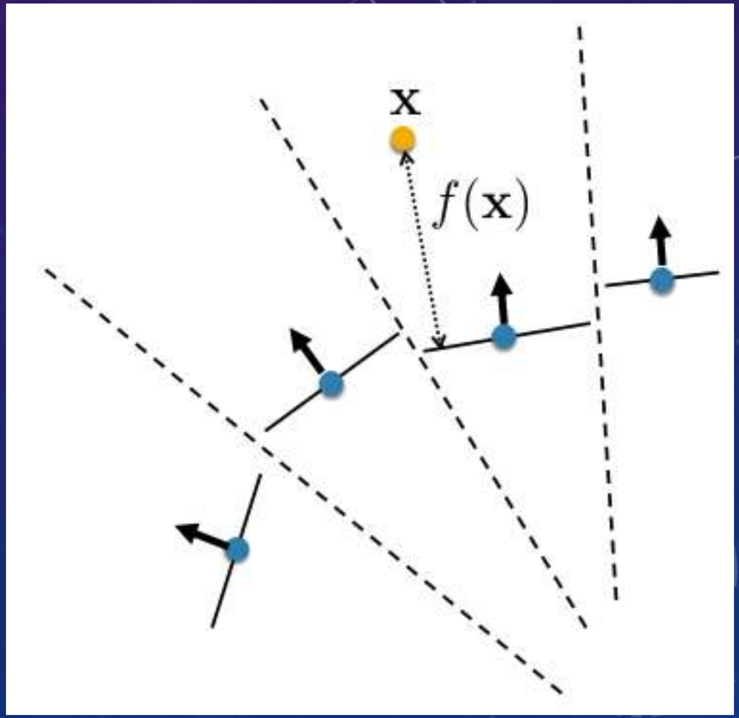
- Explicit methods
  - VD and DT, Alpha shape, Zippering, …
- Implicit methods (function)
  - Signed distance field



SDF from tangent plane

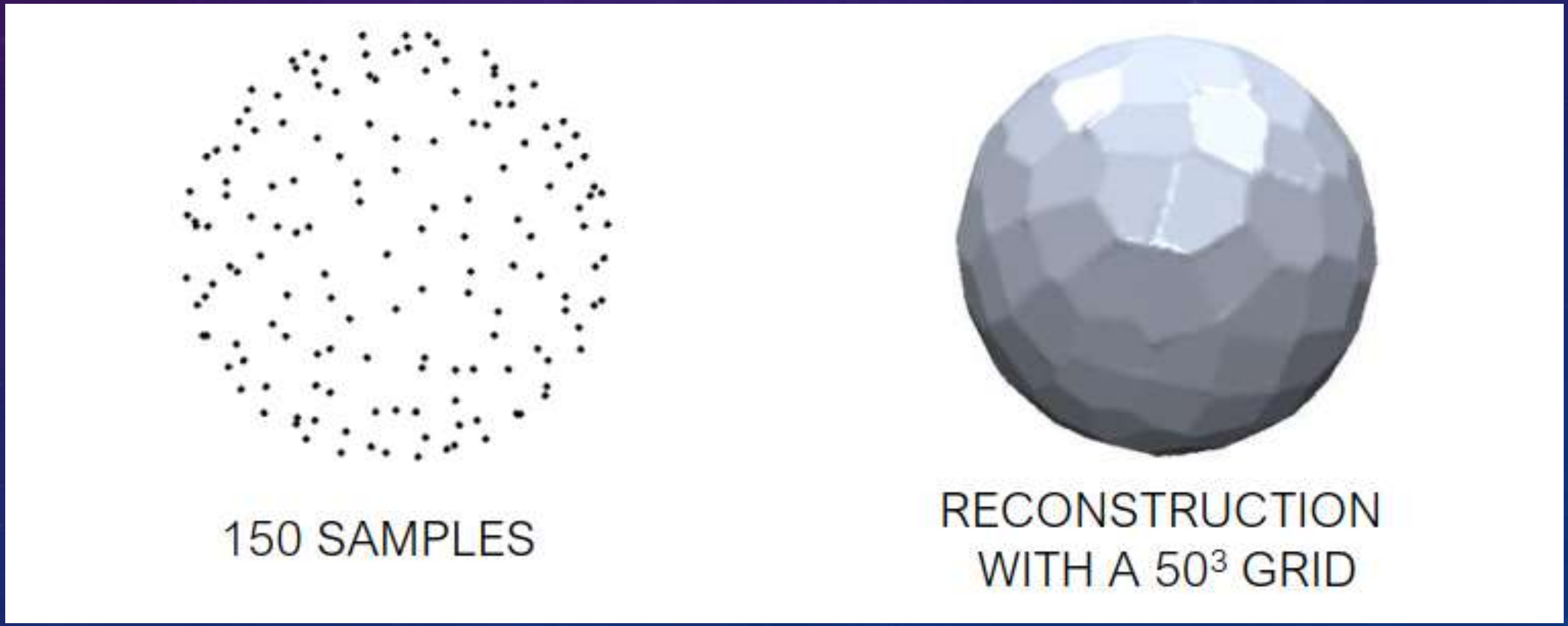
- Signed distance from tangent planes
  - Points and normals determine local tangent planes
  - Use distance from closest point's tangent plane

$$\begin{cases} F(x_i) = 0 \\ F(x_i + \lambda n_i) = \lambda \\ F(x_i - \lambda n_i) = -\lambda \end{cases}$$



SDF from tangent plane

- Simple and efficient, but SDF is not continuous

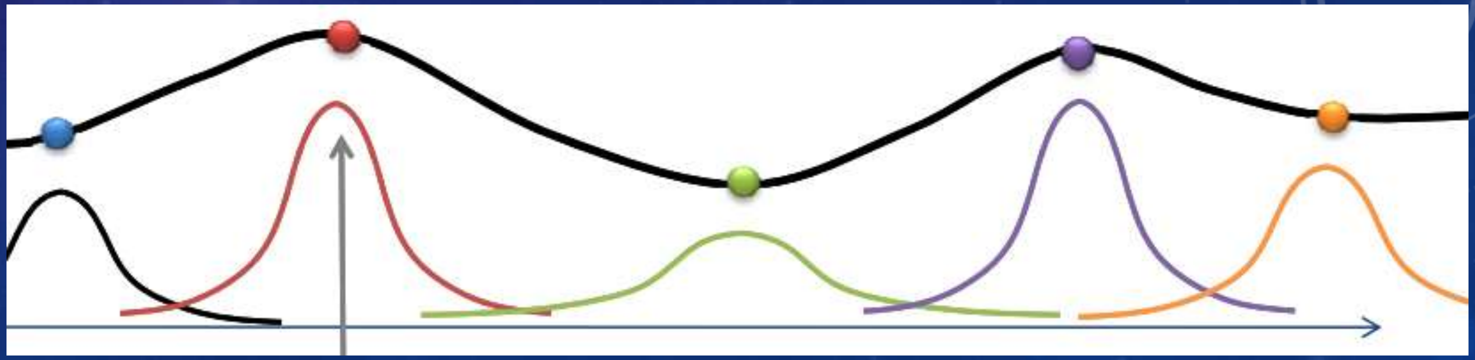


Smooth SDF Approximation

- Use radial basis functions (RBFs) to implicitly represent surface
  - Function such that the value depends only on the distance from the origin or from a center
  - Sum of radial basis functions used to approximate a function

$$F(x) = \sum_i w_i \psi_i(x)$$

$$\psi_i(x) = \psi(x - c_i)$$



Smooth SDF Approximation

- Solving equations: $2n$ equations , $2n$ variables

The on- and off-surface points are the centers c_i, then

$$F(x) = \sum_{i=1}^{n} w_i \psi(\|x - x_i\|) + \sum_{i=N}^{2n} w_i \psi(\|x - (x_i + \epsilon \vec{n}_i)\|)$$

$$\begin{cases} F(x_j) = \sum_i w_i \psi_i(x_j) = \sum_i w_i \psi_i(\|x_j - c_i\|) = 0 \\ F(x_j + \epsilon \vec{n}_j) = \sum_i w_i \psi_i(\|x_j + \epsilon \vec{n}_j - c_i\|) = \epsilon \end{cases}$$

Smooth SDF Approximation

- Solving equations: $2n$ equations , $2n$ variables

$$
\underbrace{\begin{pmatrix}
\phi(\|\mathbf{x}_1 - \mathbf{x}_1\|) & \cdots & \phi(\|\mathbf{x}_1 - (\mathbf{x}_n + \varepsilon \mathbf{n}_n)\|) \\
\vdots & & \vdots \\
\phi(\|(\mathbf{x}_n + \varepsilon \mathbf{n}_n) - \mathbf{x}_1\|) & \cdots & \phi(\|(\mathbf{x}_n + \varepsilon \mathbf{n}_n) - (\mathbf{x}_n + \varepsilon \mathbf{n}_n)\|)
\end{pmatrix}}_{\mathbf{K}}
\underbrace{\begin{pmatrix} w_1 \\ \vdots \\ w_n \\ \vdots \\ w_{2n} \end{pmatrix}}_{\mathbf{w}}
=
\underbrace{\begin{pmatrix} 0 \\ \vdots \\ 0 \\ \varepsilon \\ \vdots \\ \varepsilon \end{pmatrix}}_{\mathbf{d}}
$$

RBF Basis Functions

- Wendland basis functions $\psi(r) = \left(1 - \frac{r}{\sigma}\right)_+^4 (\frac{4r}{\sigma} + 1)$
  - Compactly supported in $[0, \sigma]$
  - Leads to Leads to sparse, symmetric positive-definite linear
  - SDF C^2 is smooth
  - But surface is not necessarily fair
  - Not suited for highly irregular sampling

RBF Basis Functions

- Triharmonic basis functions $\psi(r) = r^3$
  - Globally supported function
  - Leads to dense linear system
  - SDF C^2 is smooth
  - Provably optimal fairness
  - Works well for irregular sampling

Comparison

SDF From tangent plane

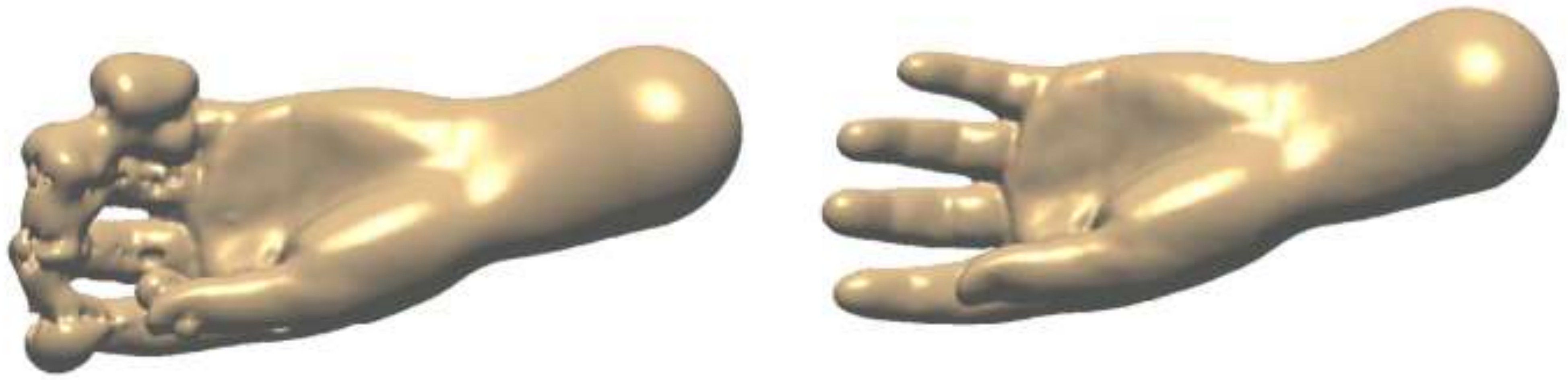
RBF Wendland

RBF Triharmonic

Other Radial Basis Functions

- Polyharmonic spline
  - $\psi(r) = r^k \log(r), k = 2,4,6, \ldots$
  - $\psi(r) = r^k, k = 1,3,5, \ldots$
- Multiquadratic $\psi(r) = \sqrt{r^2 + \beta^2}$
- Gaussian $\psi(r) = e^{-\beta r^2}$
- B-Spline (compact support) $\psi(r) = \text{piecewise-poly}(r)$

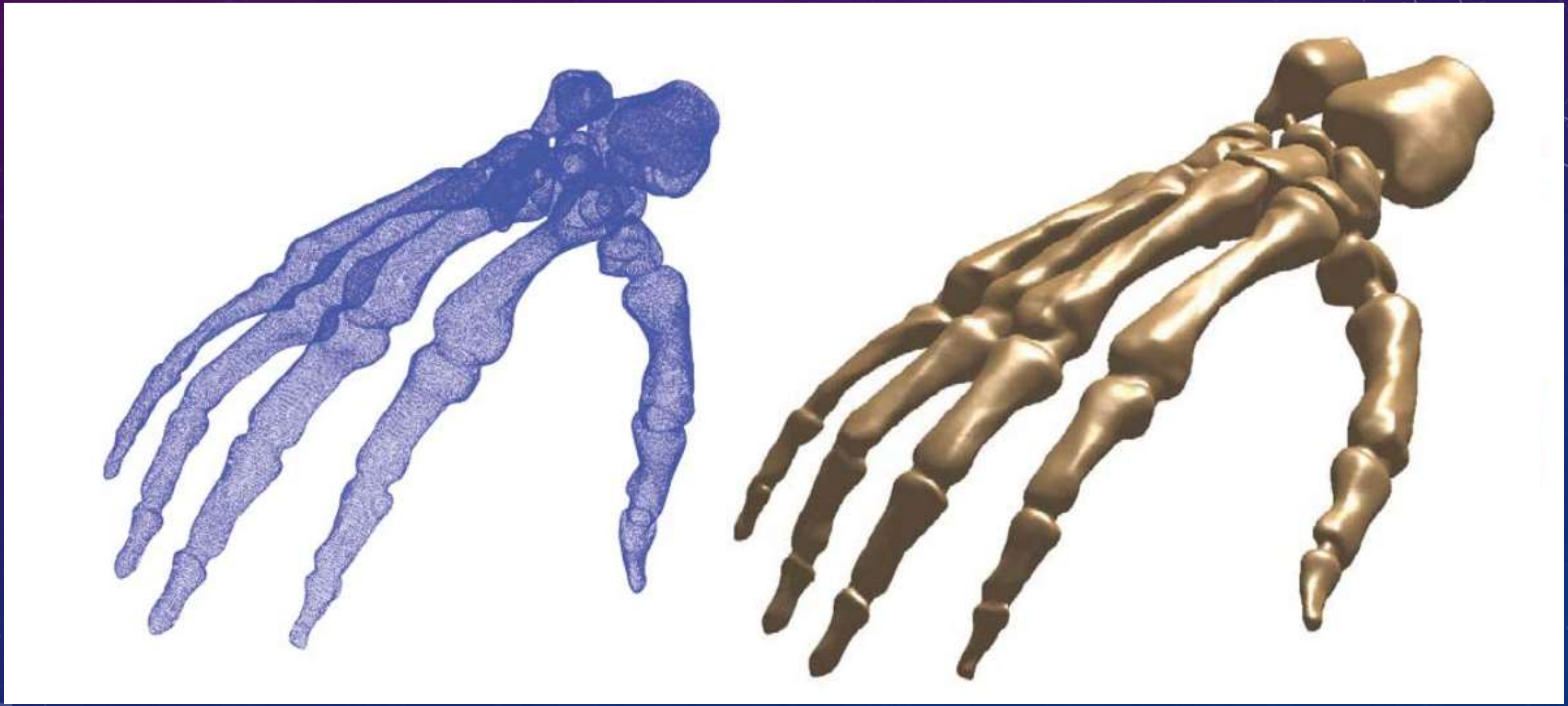
How Big is ϵ?



Without normal length validation

With normal length validation

RBF reconstruction examples



Complexity Issues

- Solve the linear system for RBF weights
  - Hard to solve for large number of samples
- Compactly supported RBFs
  - Sparse linear system, efficient solvers
- Adaptative RBF fitting
  - Start with a few RBFs only
  - Add more RBFs in region of large error

Reconstruction methods

- Explicit methods
  - VD and DT, Alpha shape, Zippering, …
- Implicit methods (function)
  - Signed distance field
  - Moving least square

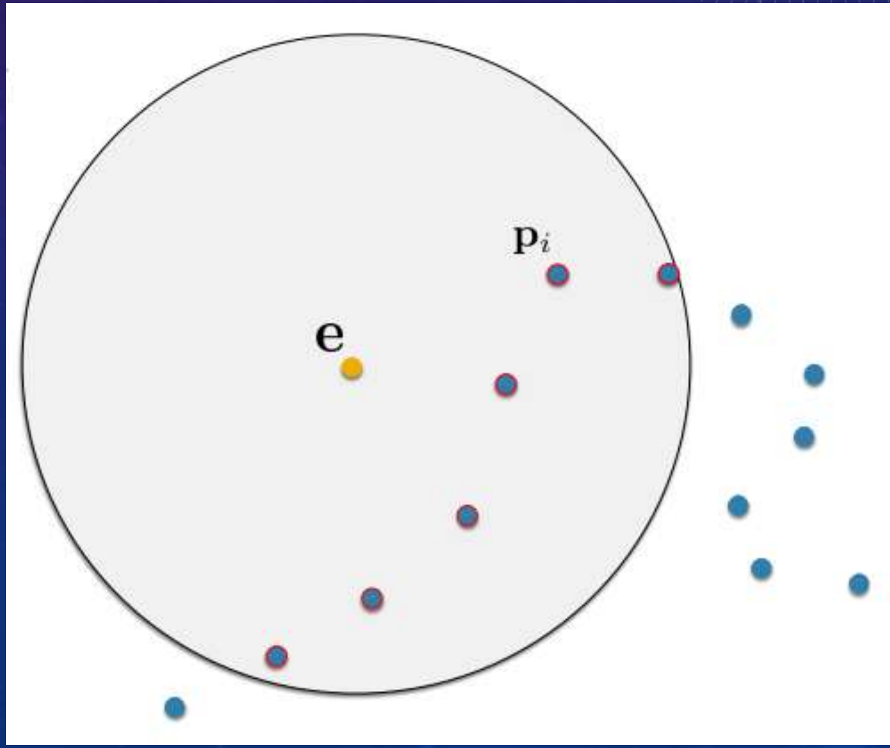
Moving Least Square

- Approximates a smooth surface from irregularly sampled points
- Create a local estimate of the surface at every point in space
- Implicit function is computed by local approximations
- Projection operator that projects points onto the MSL surface

Moving Least Square

- How to project e on the surface defined by the input

  1. Get Neighborhood of e



Moving Least Square

- How to project e on the surface defined by the input

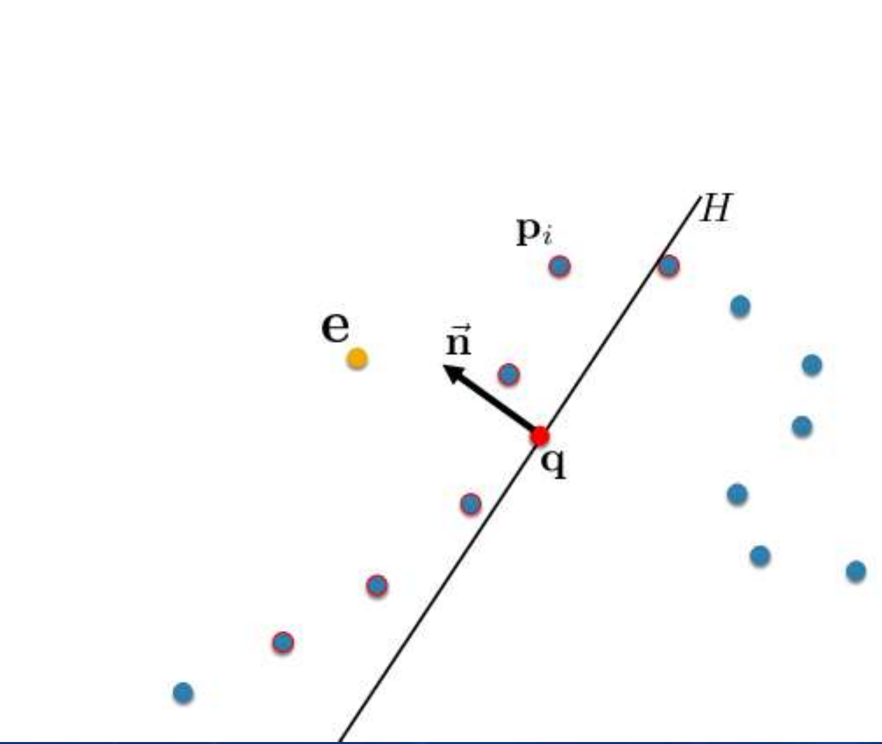
1. Get Neighborhood of e
2. Find a local reference plane

$$H = \{x \in \mathbb{R}^3 | \vec{n}^T(p_i - q) = 0\}$$

Minimizing the energy

$$\sum_i \left(\vec{n}^T(p_i - q)\right)^2 \theta(\|p_i - q\|)$$

θ: Smooth, positive, and monotonically decreasing weight function



Moving Least Square

- How to project e on the surface defined by the input

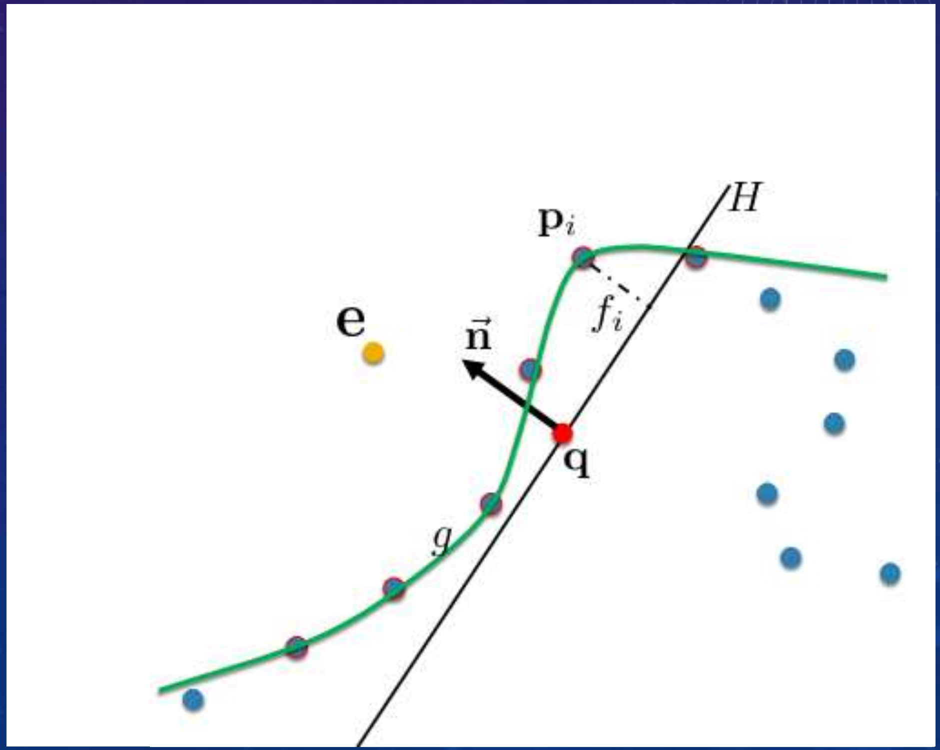
1. Get Neighborhood of e
2. Find a local reference plane
3. Find a polynomial approximation

$$g: H \rightarrow \mathbb{R}^3$$

Minimizing the energy

$$\sum_i (g(x_i, y_i) - f_i)^2 \theta(\|p_i - q\|)$$

(x_i, y_i): 2D coordinate of the projection on H

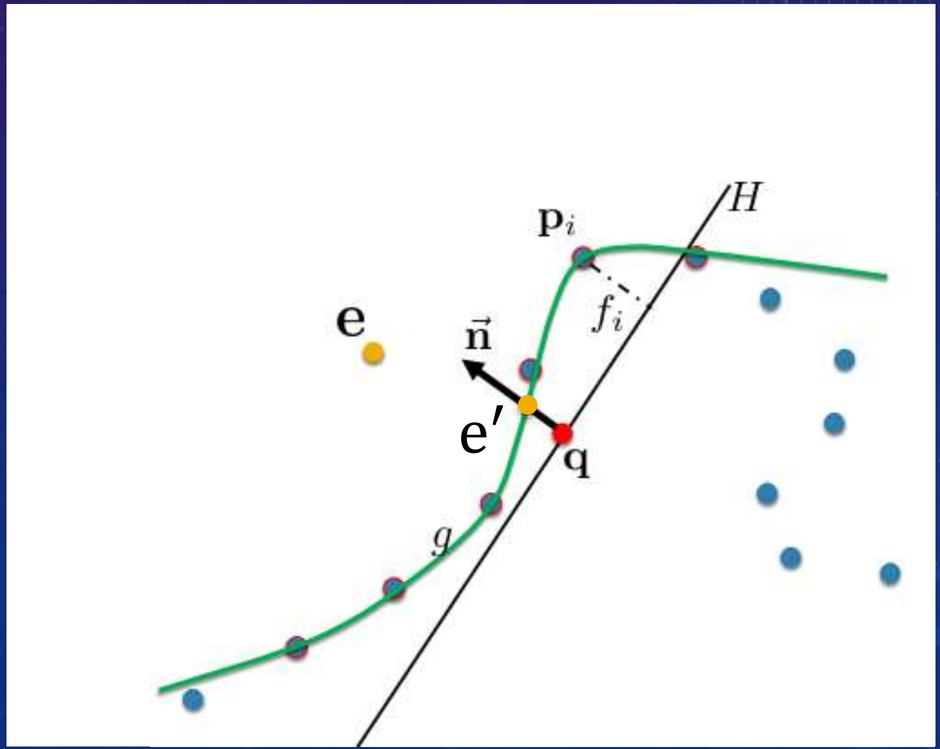


Moving Least Square

- How to project e on the surface defined by the input

1. Get Neighborhood of e
2. Find a local reference plane
3. Find a polynomial approximation
4. Projection of e

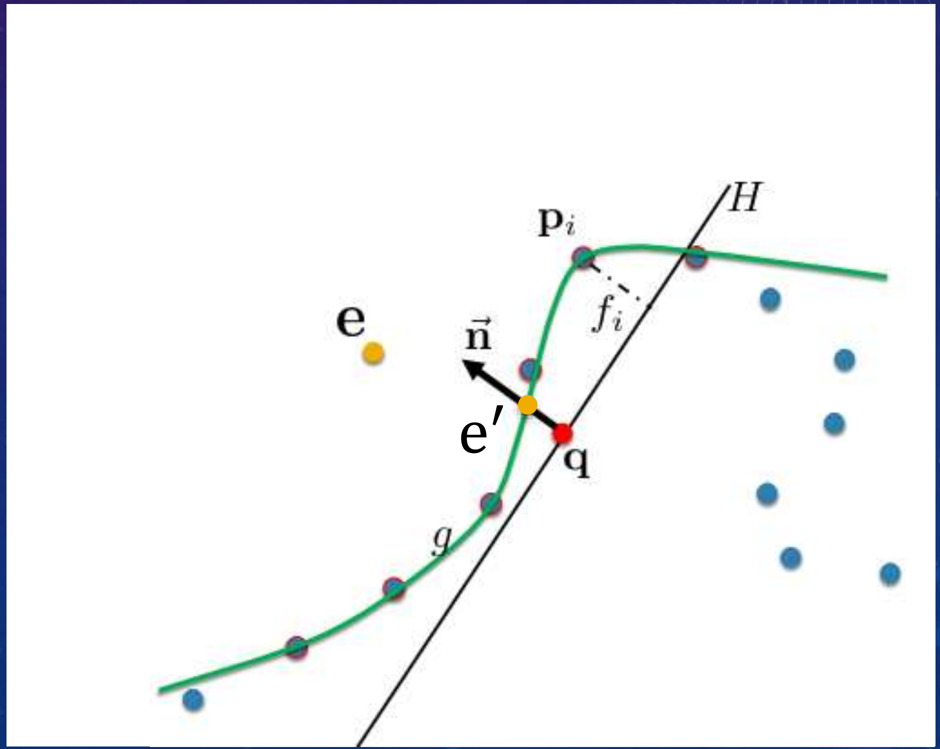
$$e' = q + g(0,0)\vec{n}$$



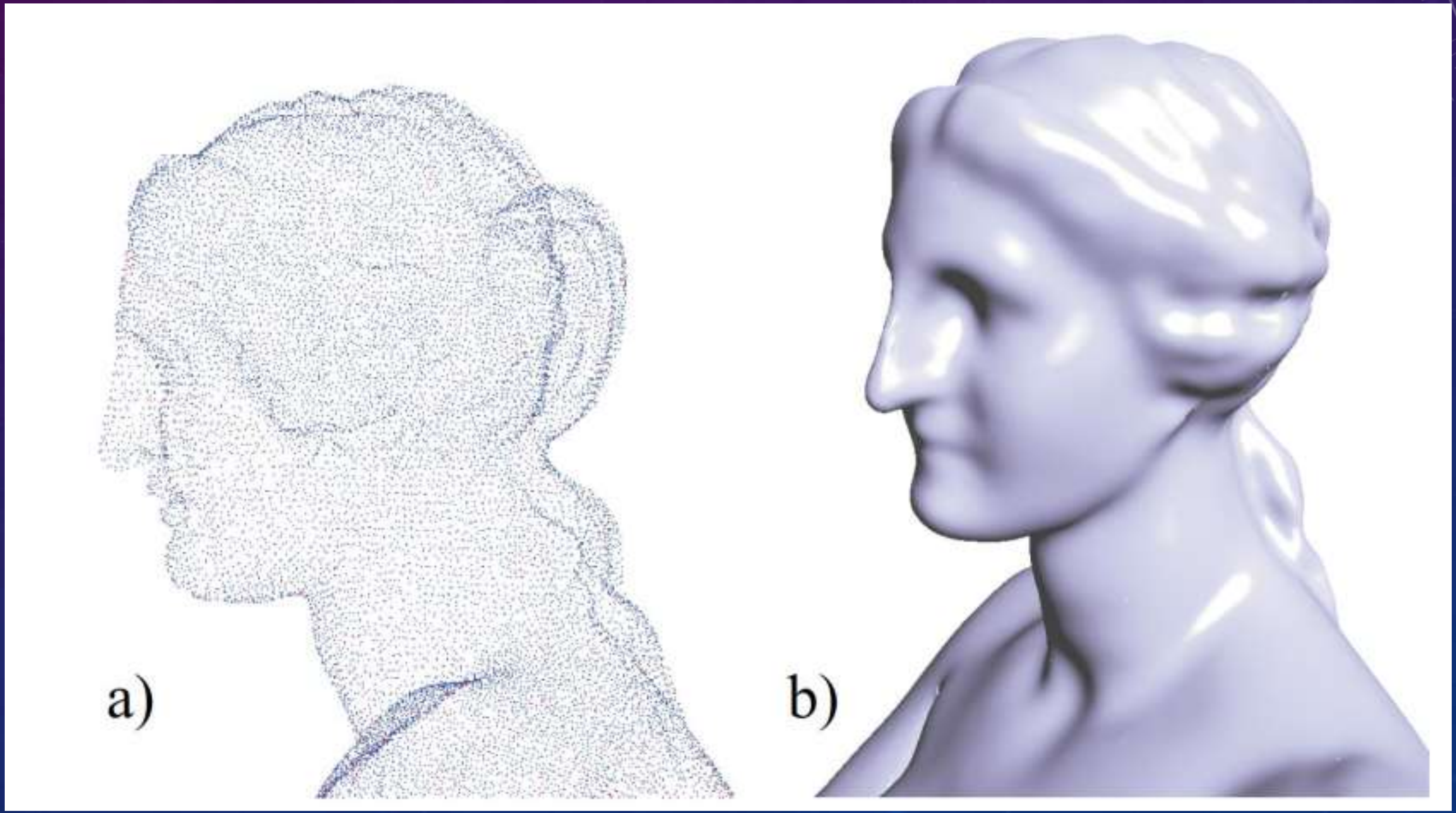
Moving Least Square

- How to project e on the surface defined by the input

1. Get Neighborhood of e
2. Find a local reference plane
3. Find a polynomial approximation
4. Projection of e
5. Iterate if $g(0,0) > \epsilon$



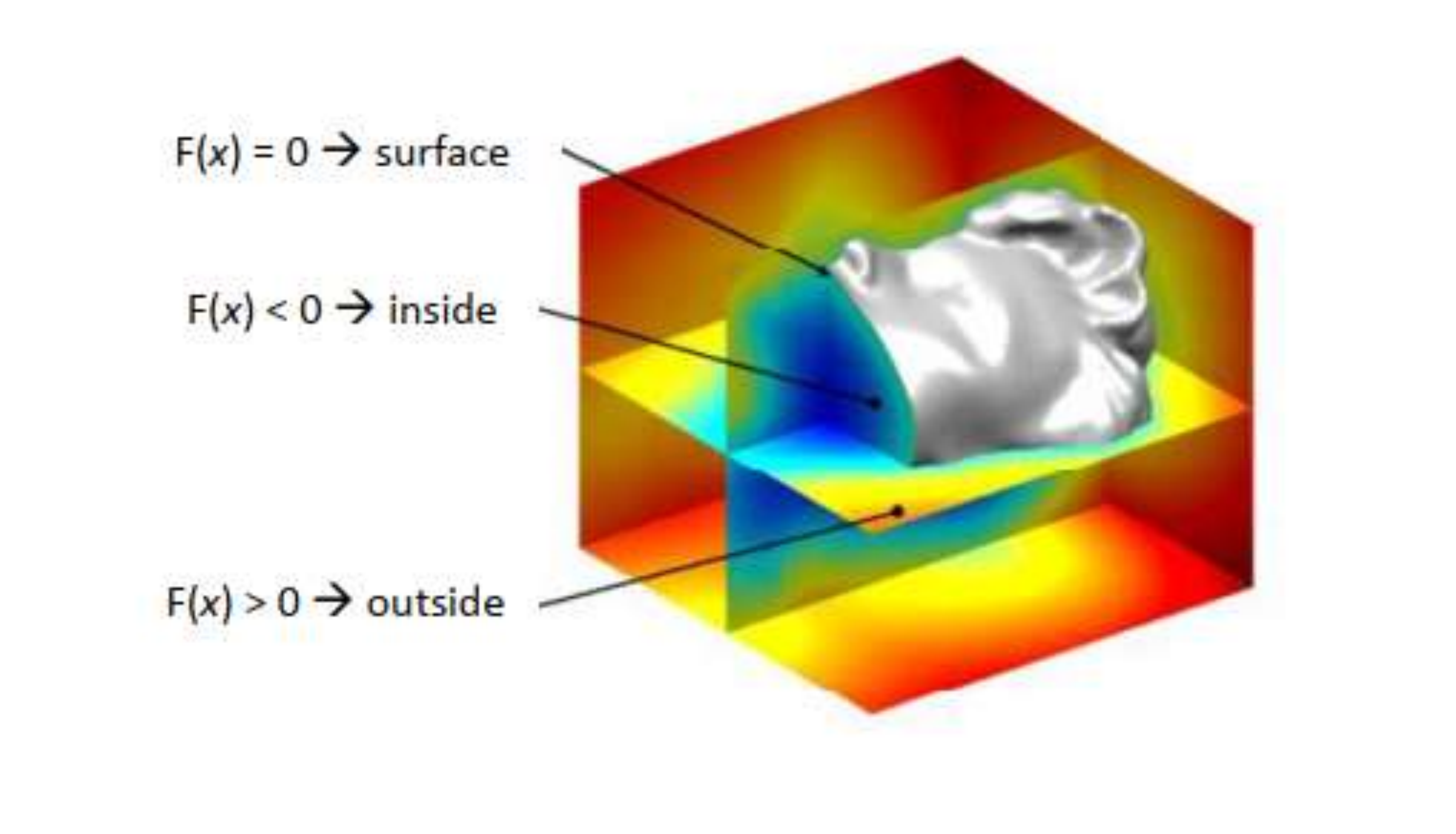
Moving Least Square



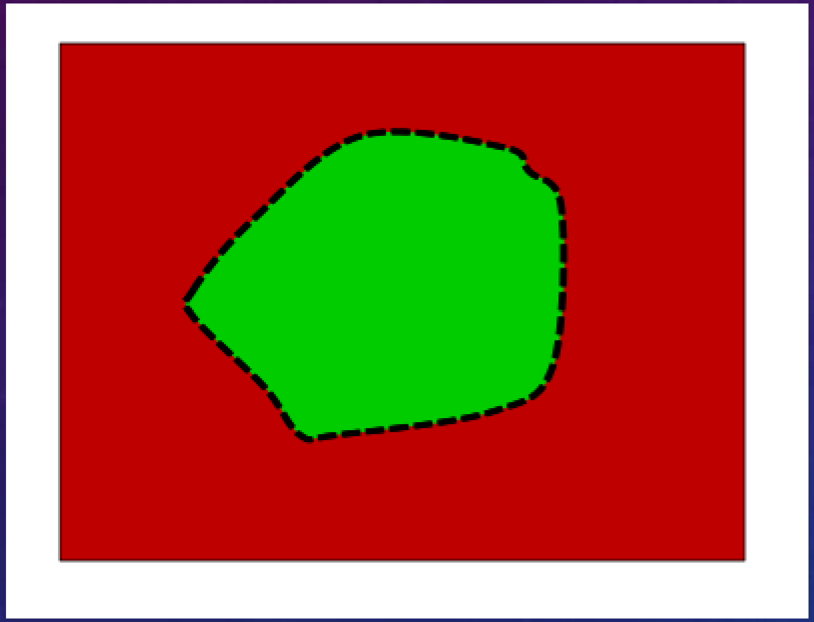
Reconstruction methods

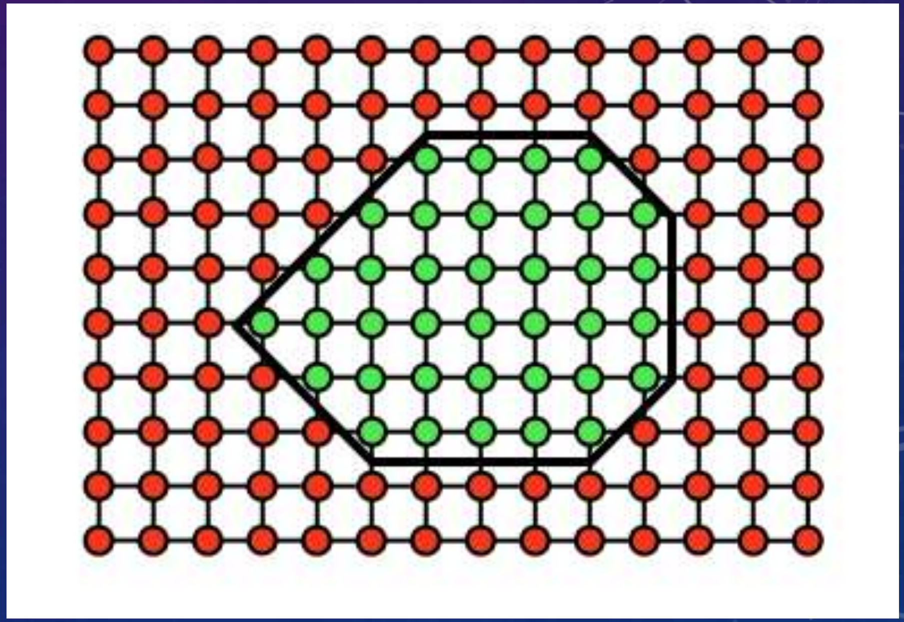
- Explicit methods
  - VD and DT, Alpha shape, Zippering, …
- Implicit methods (function)
  - Signed distance field
  - Moving least square
  - Poisson surface reconstruction (assignment 2)

Extracting the Surface

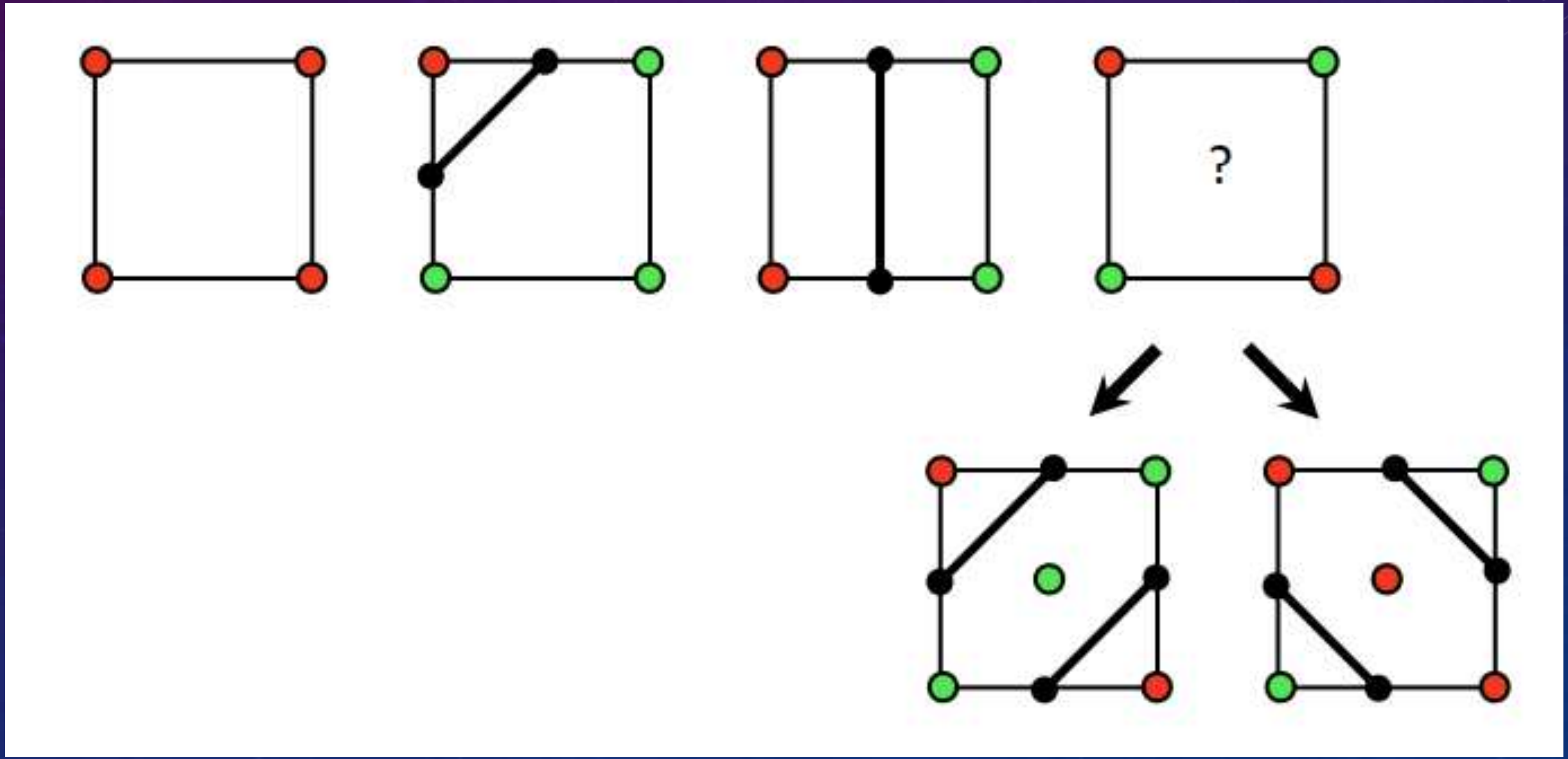


Sample the SDF





2D: Marching Squares

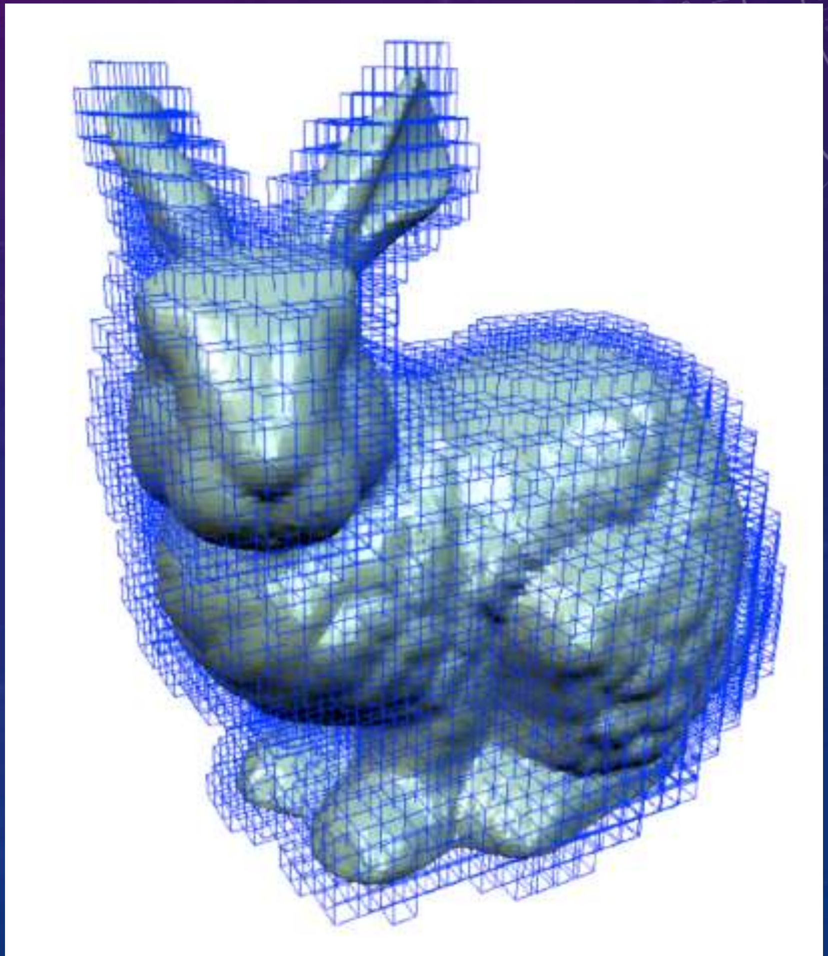


3D: Marching Cubes

- Classify grid nodes as inside/outside
- Classify cell: 2^8 configurations
- Linear interpolation along edges
- Look-up table for patch configuration
  - Disambiguation more complicated

Marching Cubes

- Cell classification:
  - Inside
  - Outside
  - Intersecting



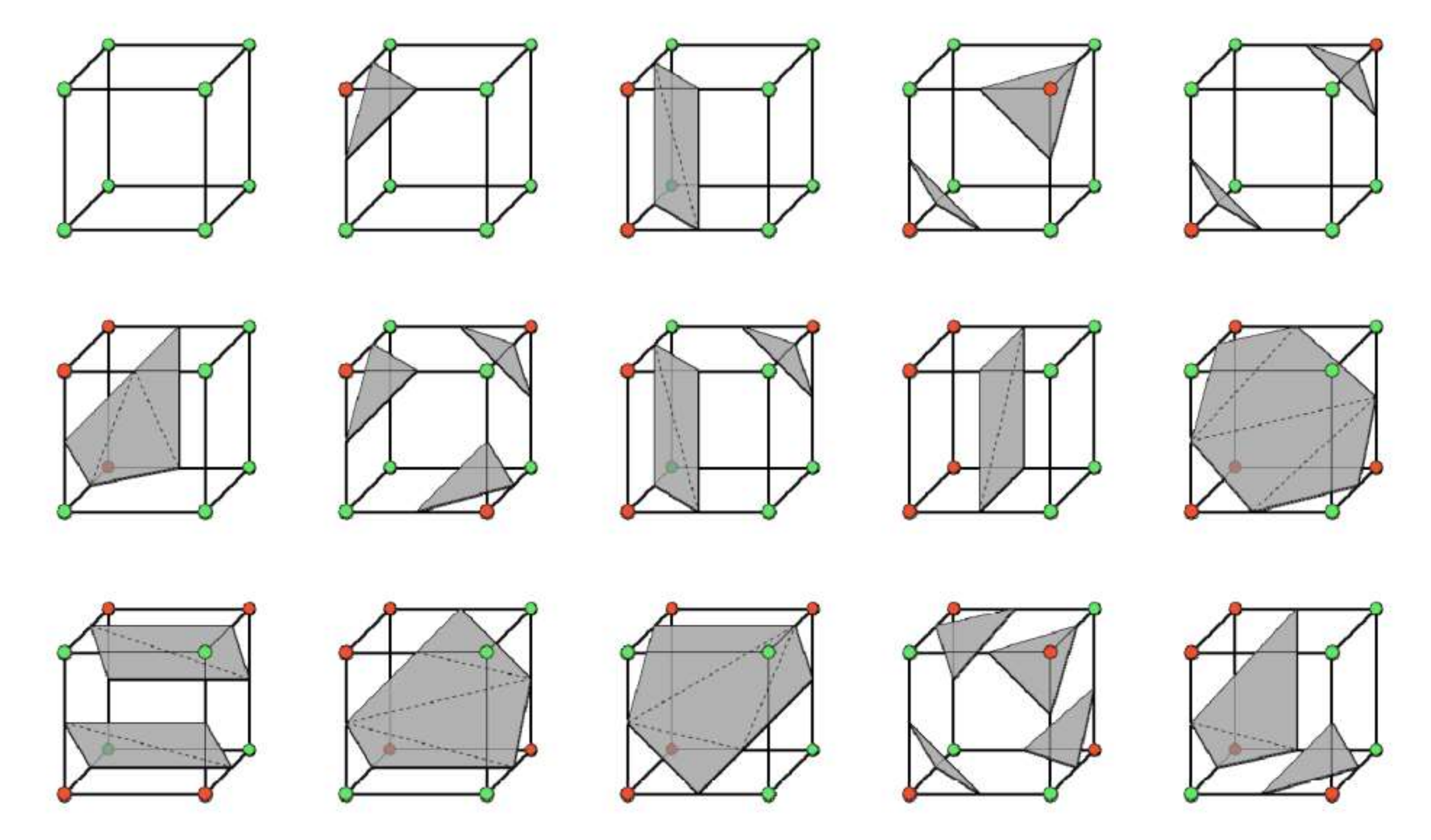
Marching Cubes

- Cases:

  256→15

Considering:

- Inversion
- Rotation



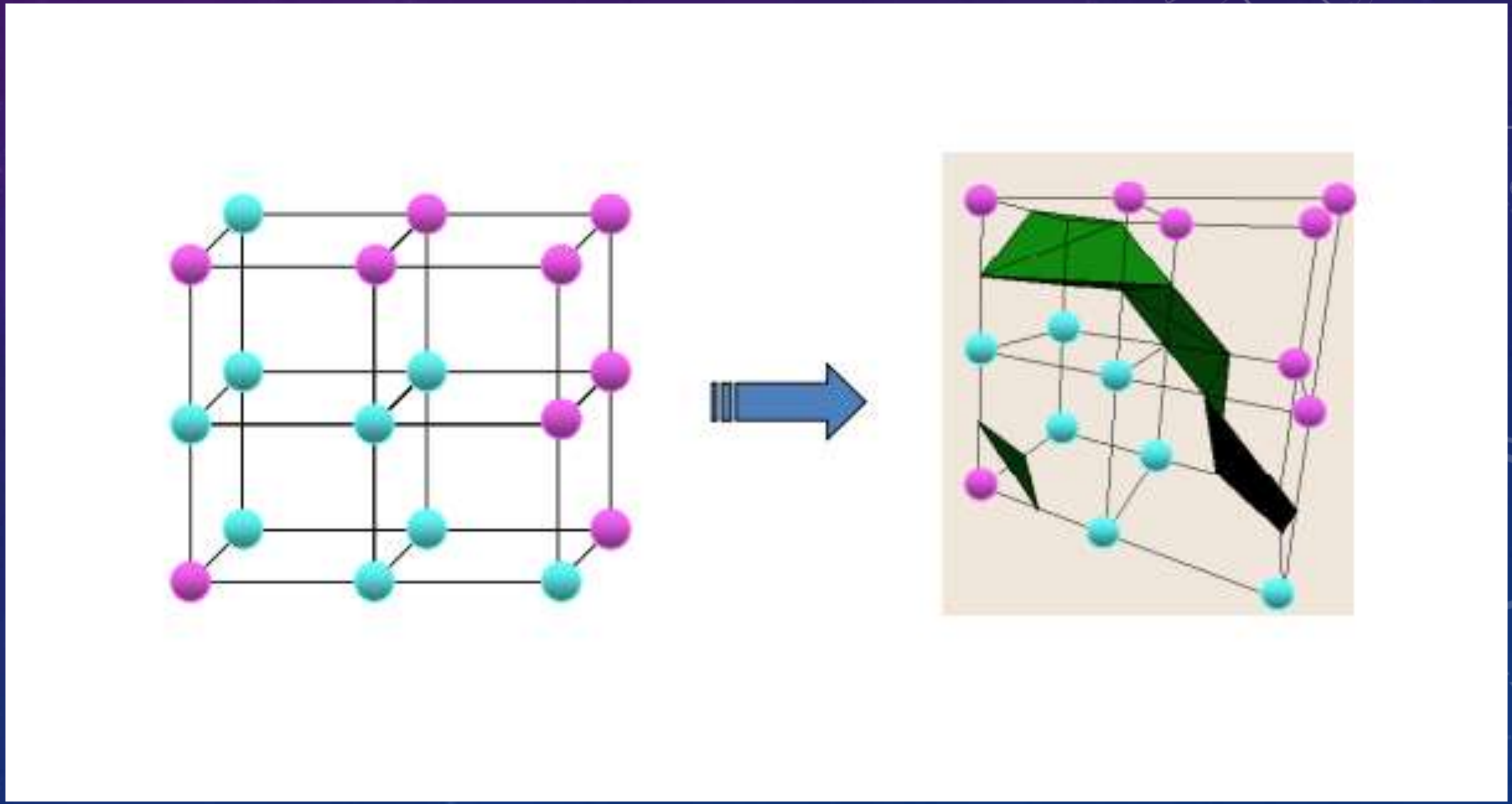
Marching Cubes

- Cases:

  256→15

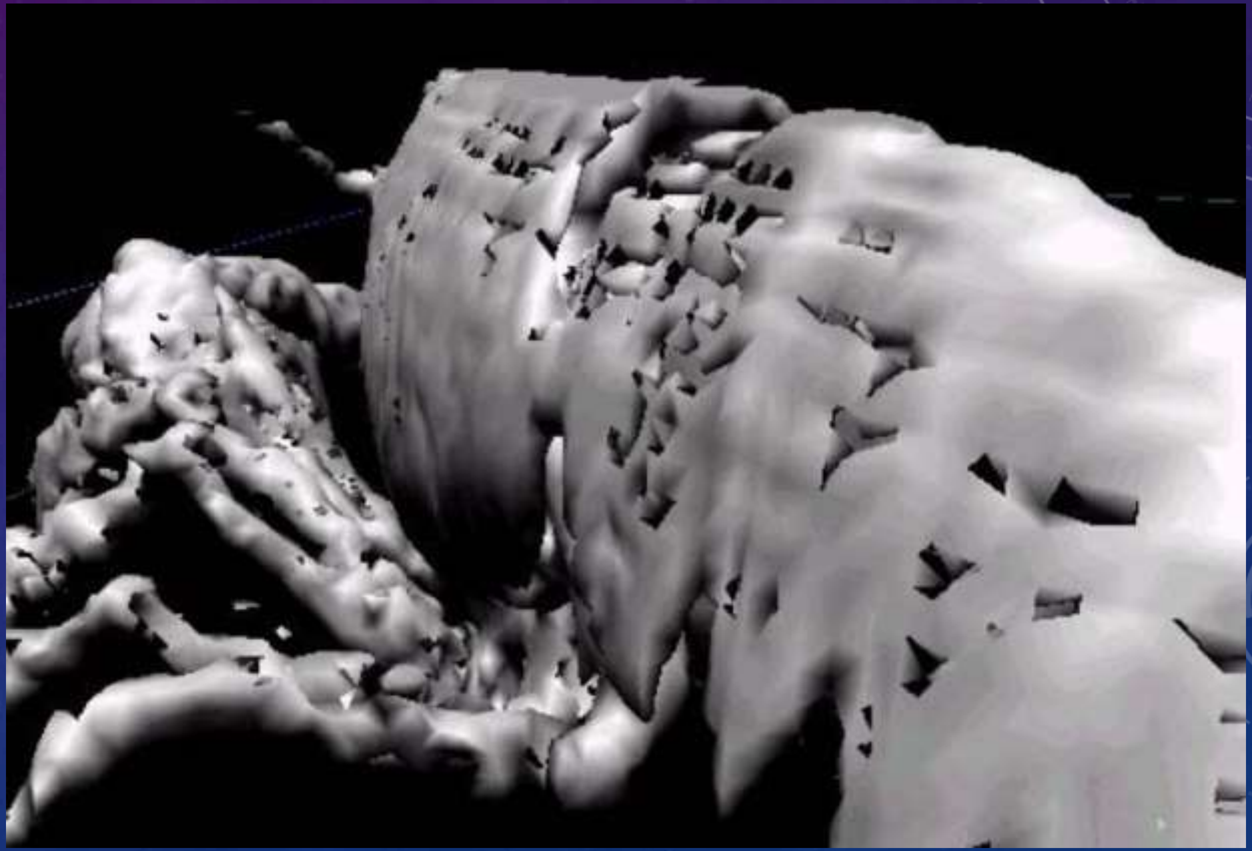
  Considering:

  - Inversion
  - Rotation

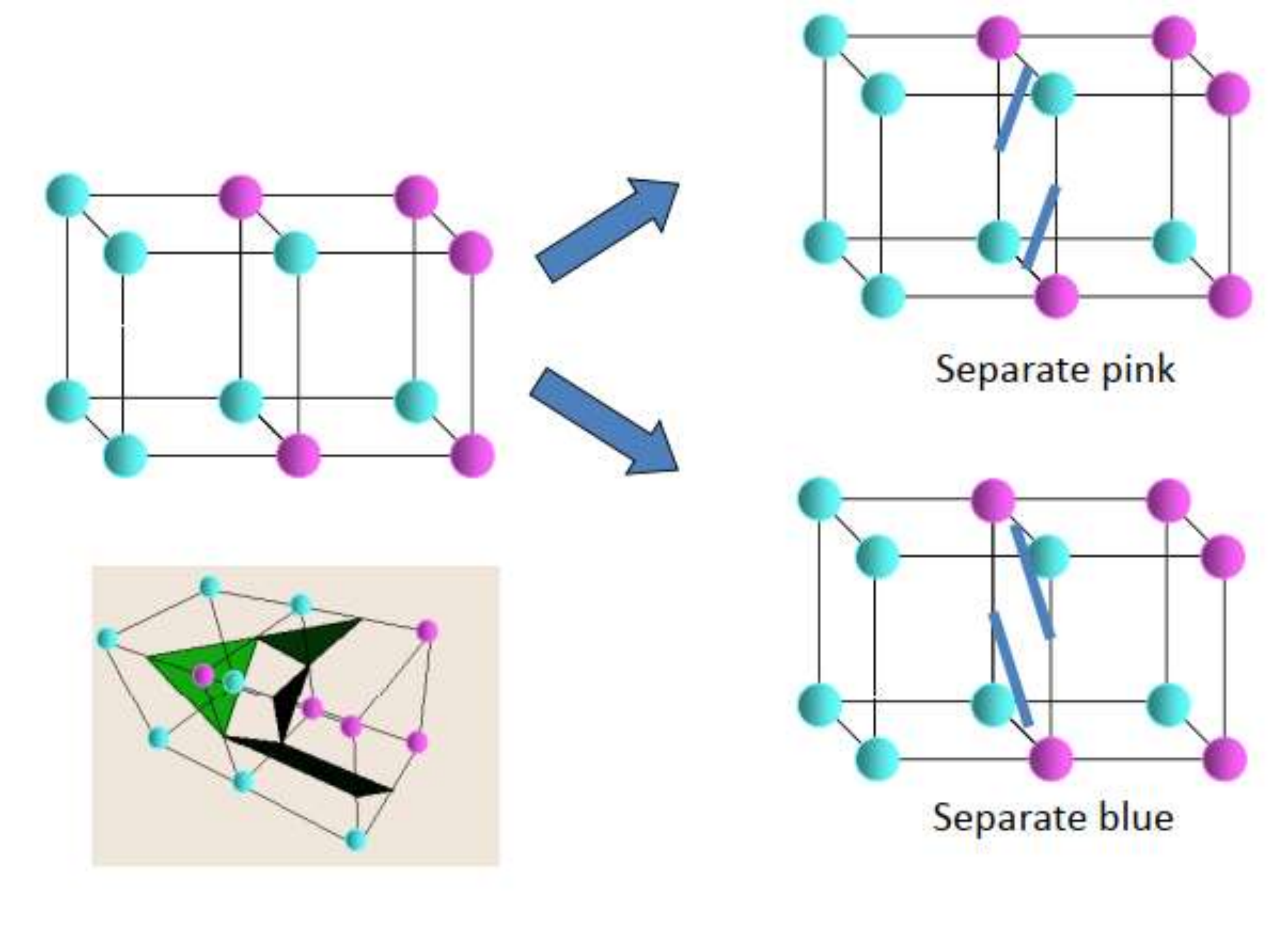


Marching Cubes problems

- Ambiguity
  - Holes
- Generates HUGE meshes
  - Millions of polygons

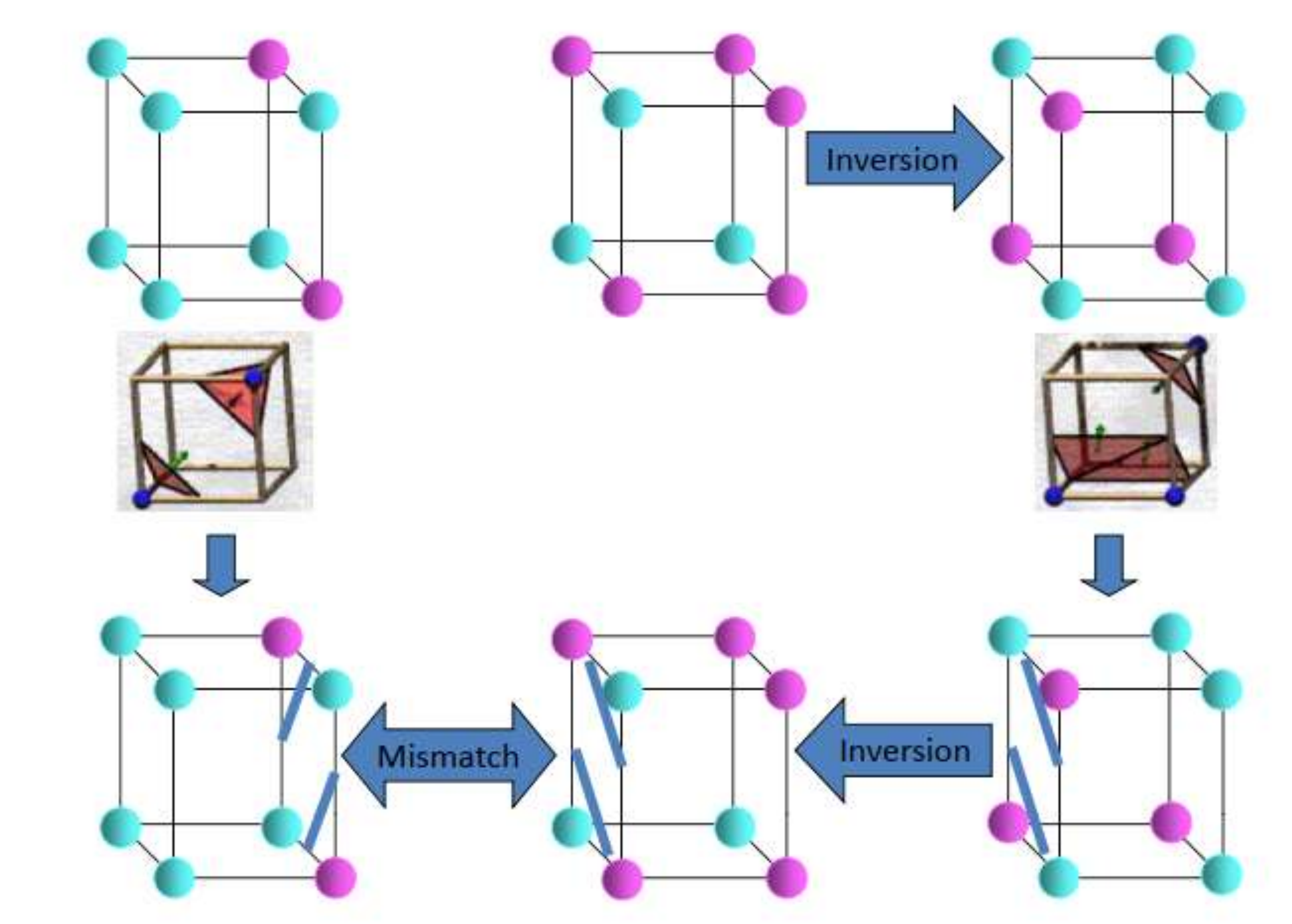


Ambiguity



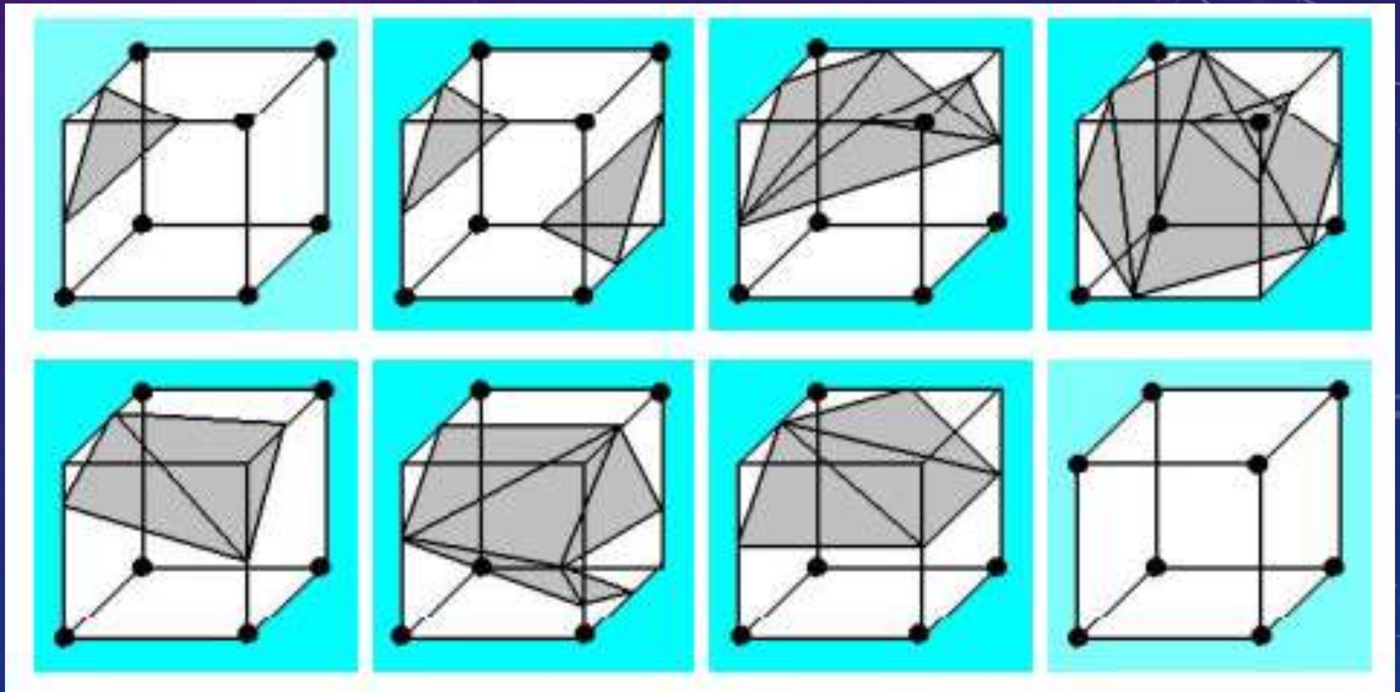
Inversion problem

- Inversion → mismatch
- 15 cases → 23 cases
    - Rotation only
    - Always separate same color
    - Ambiguous faces triangulated consistently



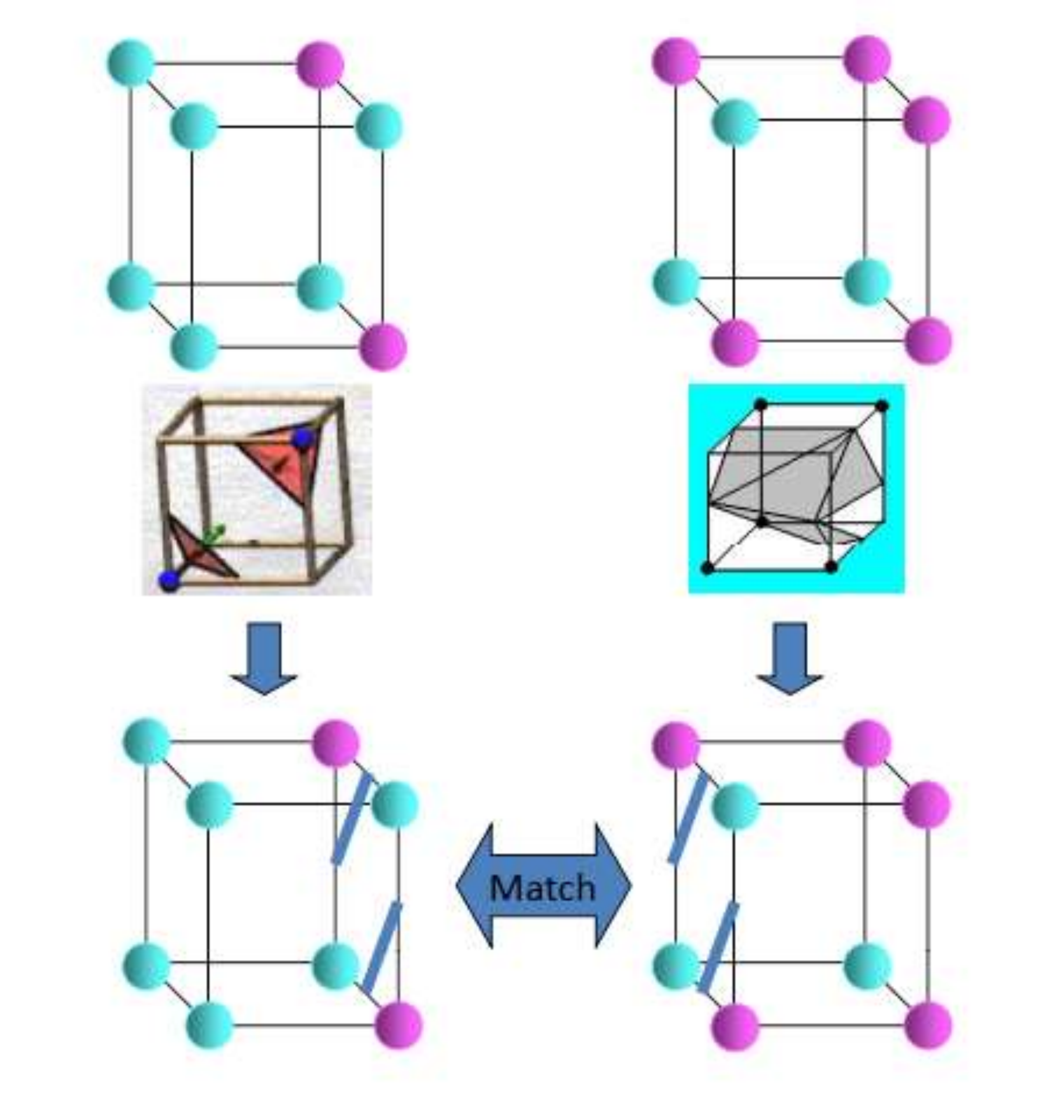
Ambiguity Solution

- Inversion → mismatch
- 15 cases → 23 cases
- 8 new cases



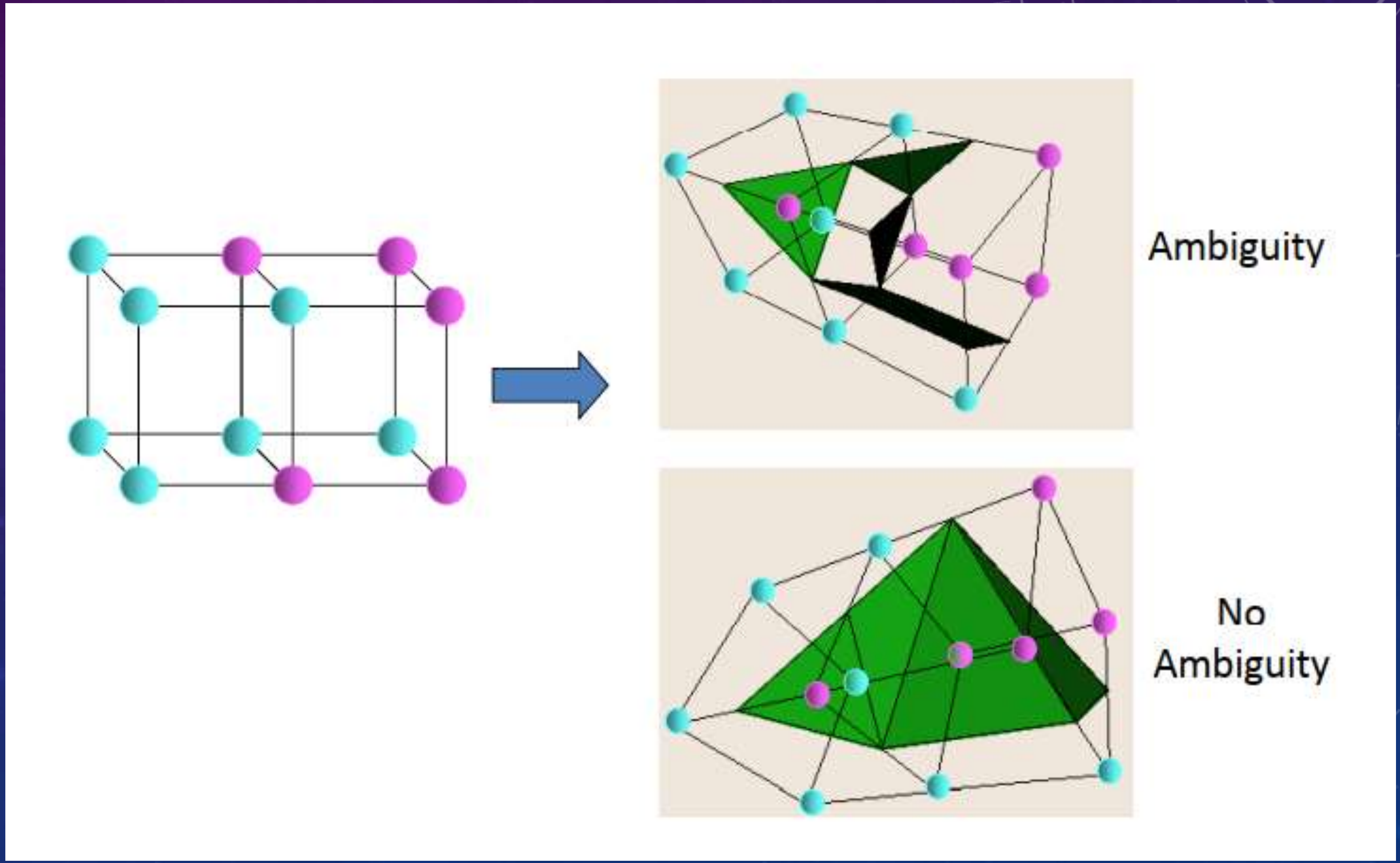
Ambiguity Solution

- Inversion → mismatch
- 15 cases → 23 cases
- 8 new cases

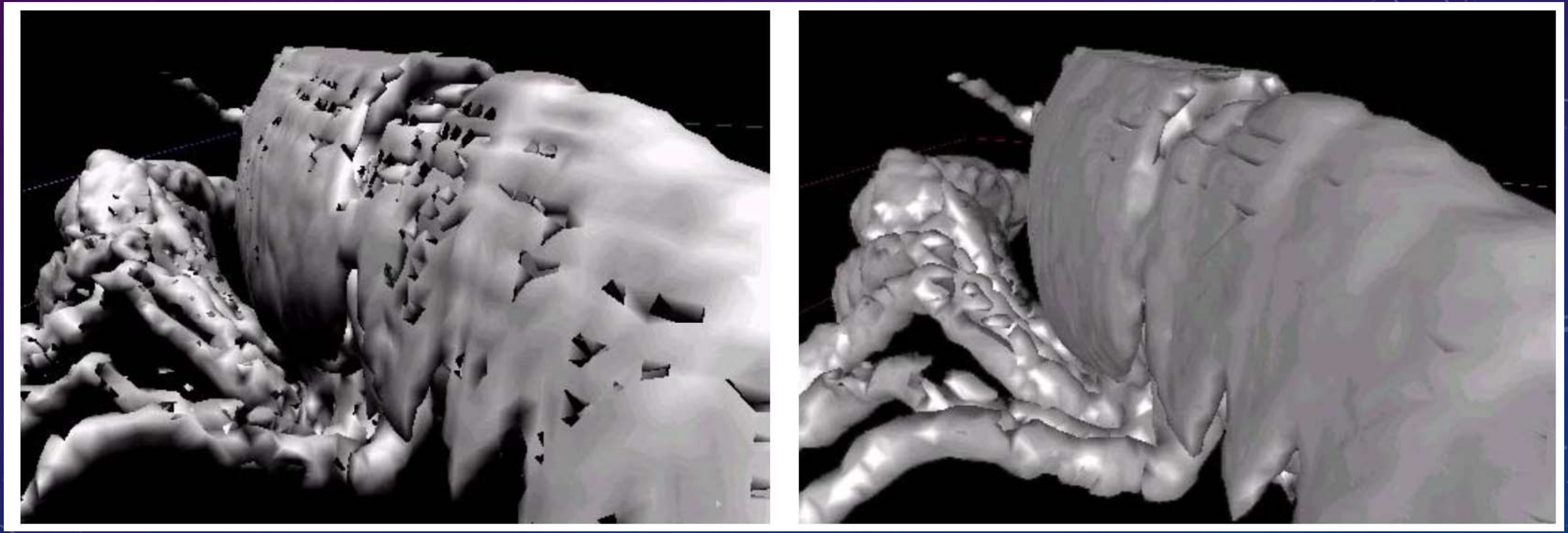


Ambiguity Solution

- Inversion → mismatch
- 15 cases → 23 cases
- 8 new cases



Ambiguity V.S. No Ambiguity



Marching Cubes Issues

- Grid not adaptive
- Many polygons required to represent small features