

## 6.2 Surface Reconstruction



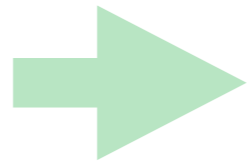
Hao Li

<http://cs621.hao-li.com>

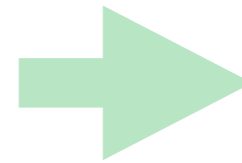
# Surface Reconstruction



physical  
model



captured  
point cloud



reconstructed  
model

# Input Data

## Set of irregular sample points

- with or without normals
- examples: multi-view stereo, union of range scan vertices



## Set of range scans

- each scan is a regular quad or tri-mesh
- normal vectors can be obtained through local connectivity



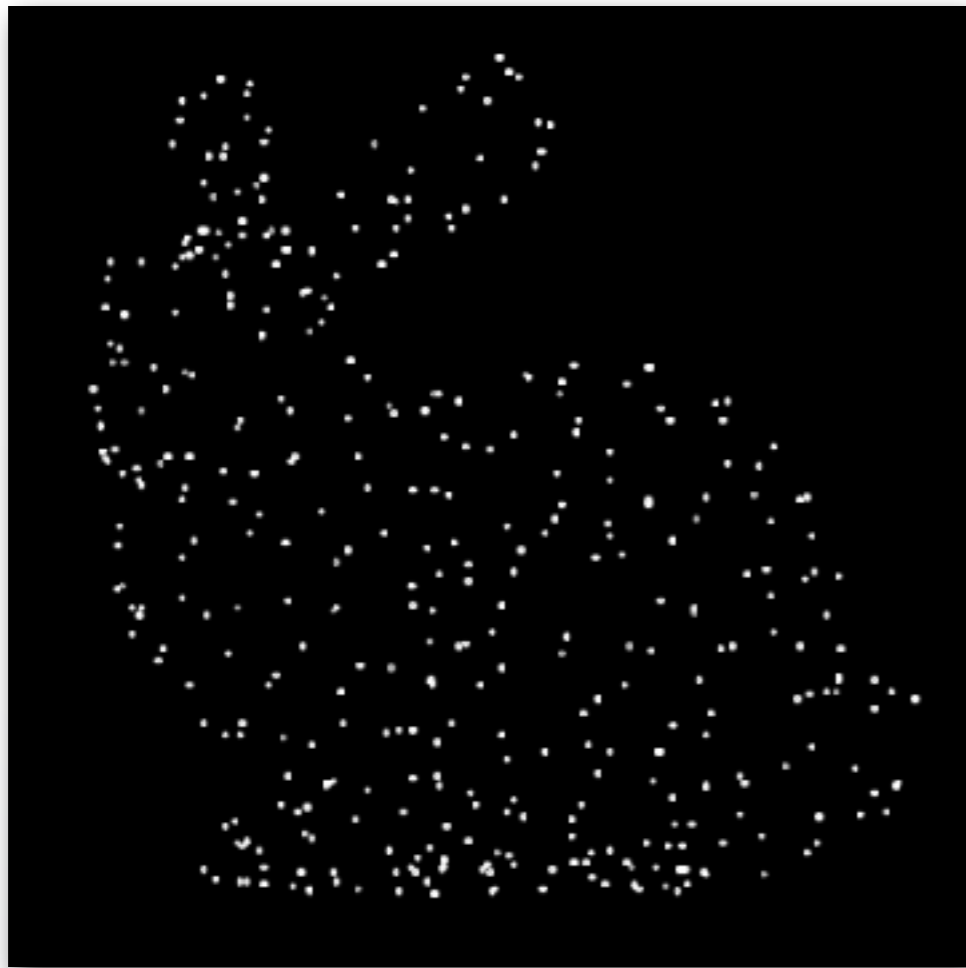
# Problem

Given a set of points  $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_n\}$  with  $\mathbf{p}_i \in \mathbb{R}^3$



# Problem

Find a manifold surface  $\mathcal{S} \subset \mathbb{R}^3$  which approximates  $\mathcal{P}$



# Two Approaches

## Explicit

Local surface  
connectivity estimation

Point interpolation

## Implicit

Signed distance function  
estimation

Mesh approximation

# Two Approaches

## Explicit

- Ball pivoting algorithm
- Delaunay triangulation
- Alpha shapes
- Zippering...

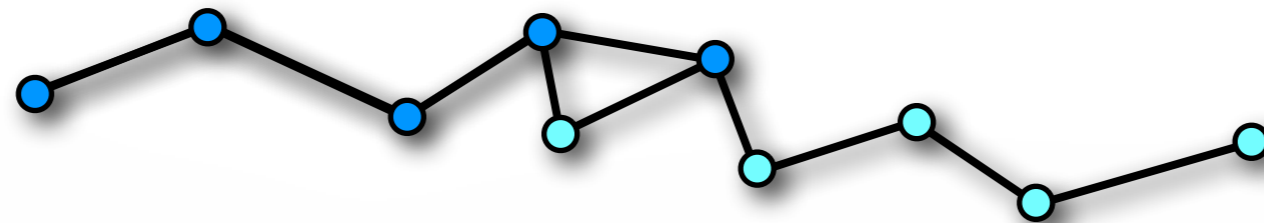
– Image space triangulation

## Implicit

- Distance from tangent planes
- SDF estimation via RBF
- ...

# Explicit Reconstruction

- Connect sample points by triangles
- Exact interpolation of sample points
- Bad for noisy or misaligned data
- Can lead to holes or non-manifold situations

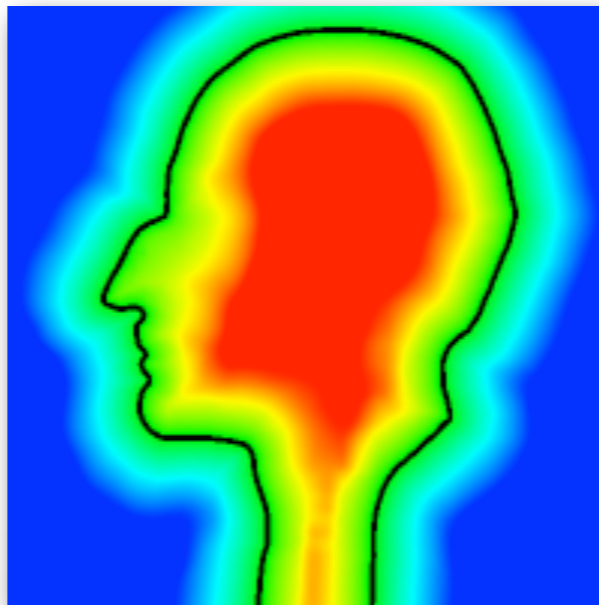




# Implicit Reconstruction

Given a set of points  $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_n\}$  with  $\mathbf{p}_i \in \mathbb{R}^3$   
Find a manifold surface  $\mathcal{S} \subset \mathbb{R}^3$  which approximates  $\mathcal{P}$

where  $\mathcal{S} = \{\mathbf{x} \mid d(\mathbf{x}) = 0\}$  with  $d(\mathbf{x})$  a signed distance function



# Data Flow

Point cloud

Signed distance function estimation

$d(\mathbf{x}) \downarrow$

Evaluation of distances on uniform grid

$d(\mathbf{i}), \mathbf{i} = [i, j, k] \in \mathbb{Z}^3 \downarrow$

Mesh extraction via marching cubes

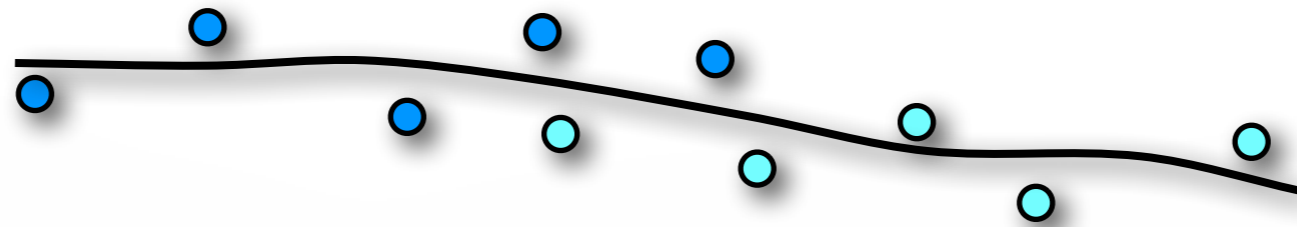
Mesh

# Implicit Surface Reconstruction Methods

Mainly differ in their signed distance function

# Implicit Reconstruction

- Estimate signed distance function (SDF)
- Extract Zero isosurface by Marching Cubes
- Approximation of input points
- Result is closed two-manifold surface

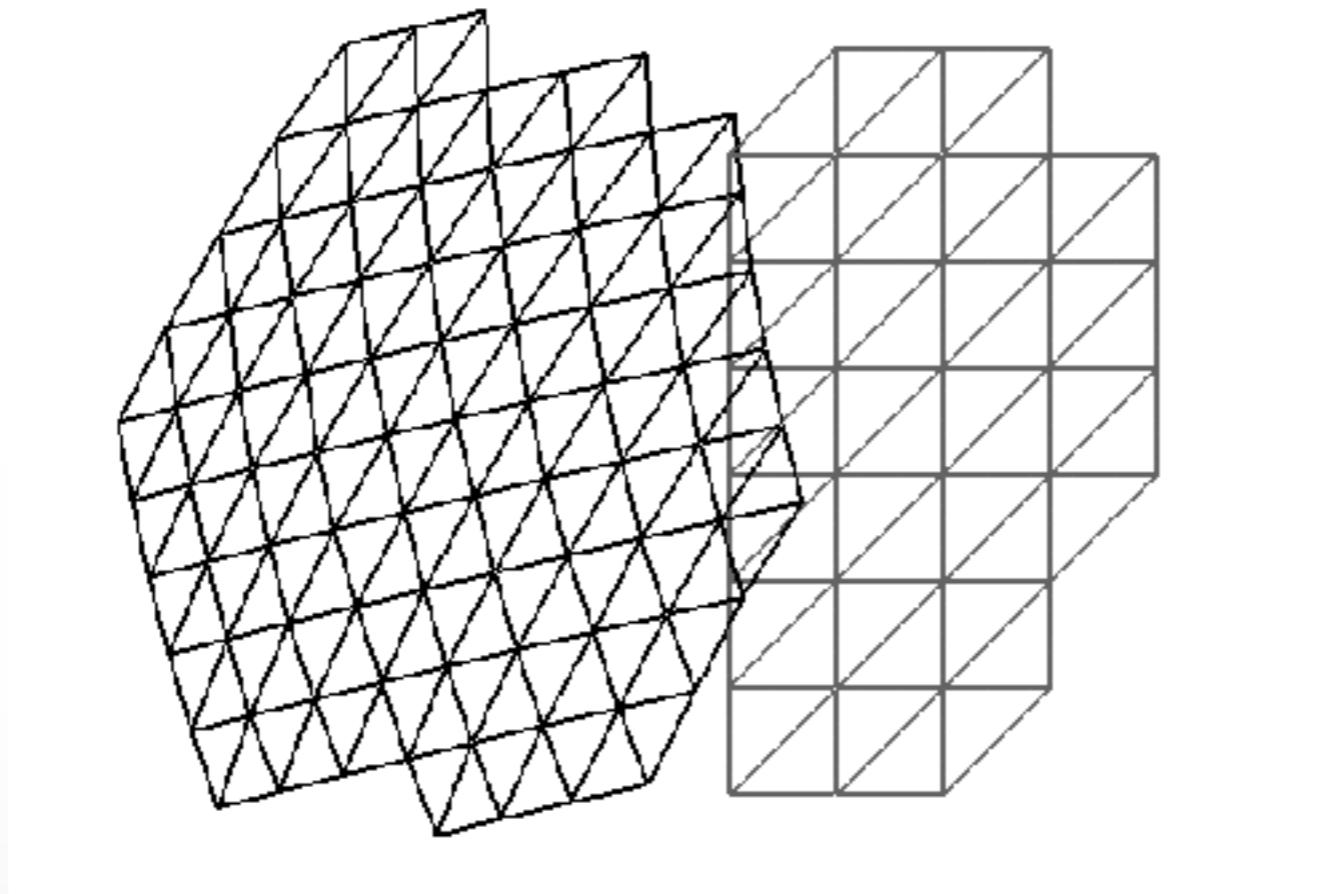


# Outline

- **Explicit Reconstruction**
  - Zippering range scans
- **Implicit Reconstruction**
  - SDF from point clouds
  - SDF from range scans
  - Poisson surface reconstruction

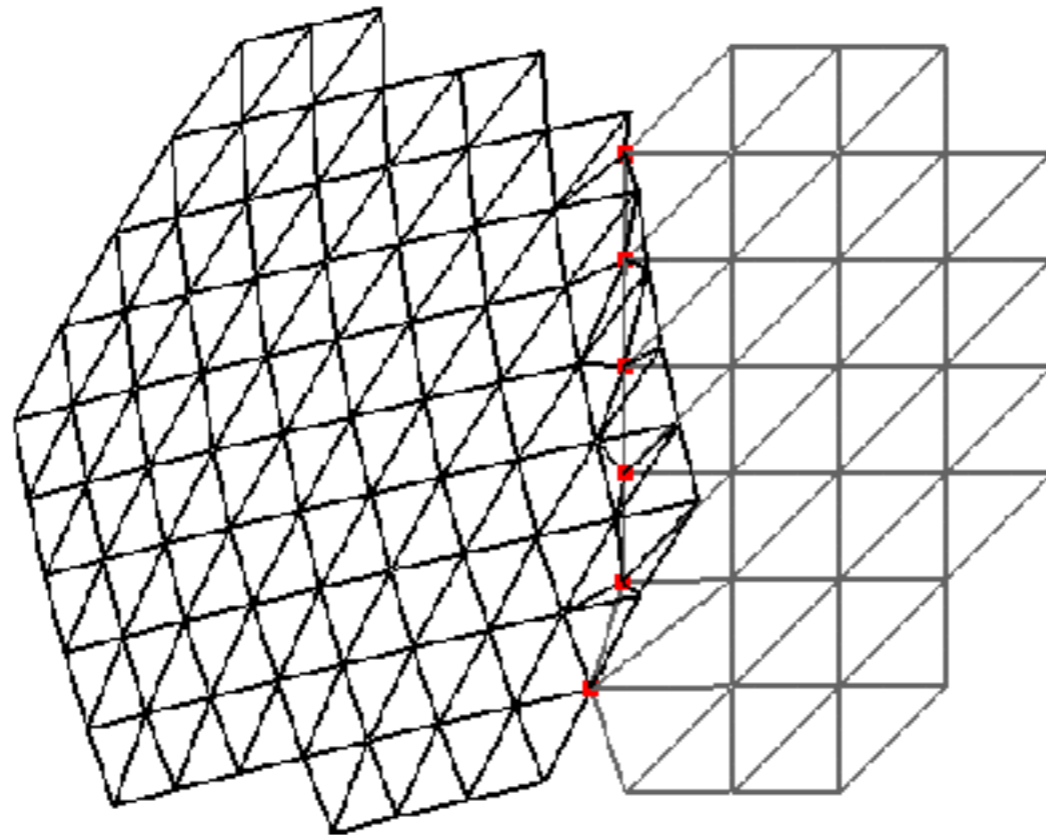
# Explicit Reconstruction

“Zipper” several scans to one single model



# Explicit Reconstruction

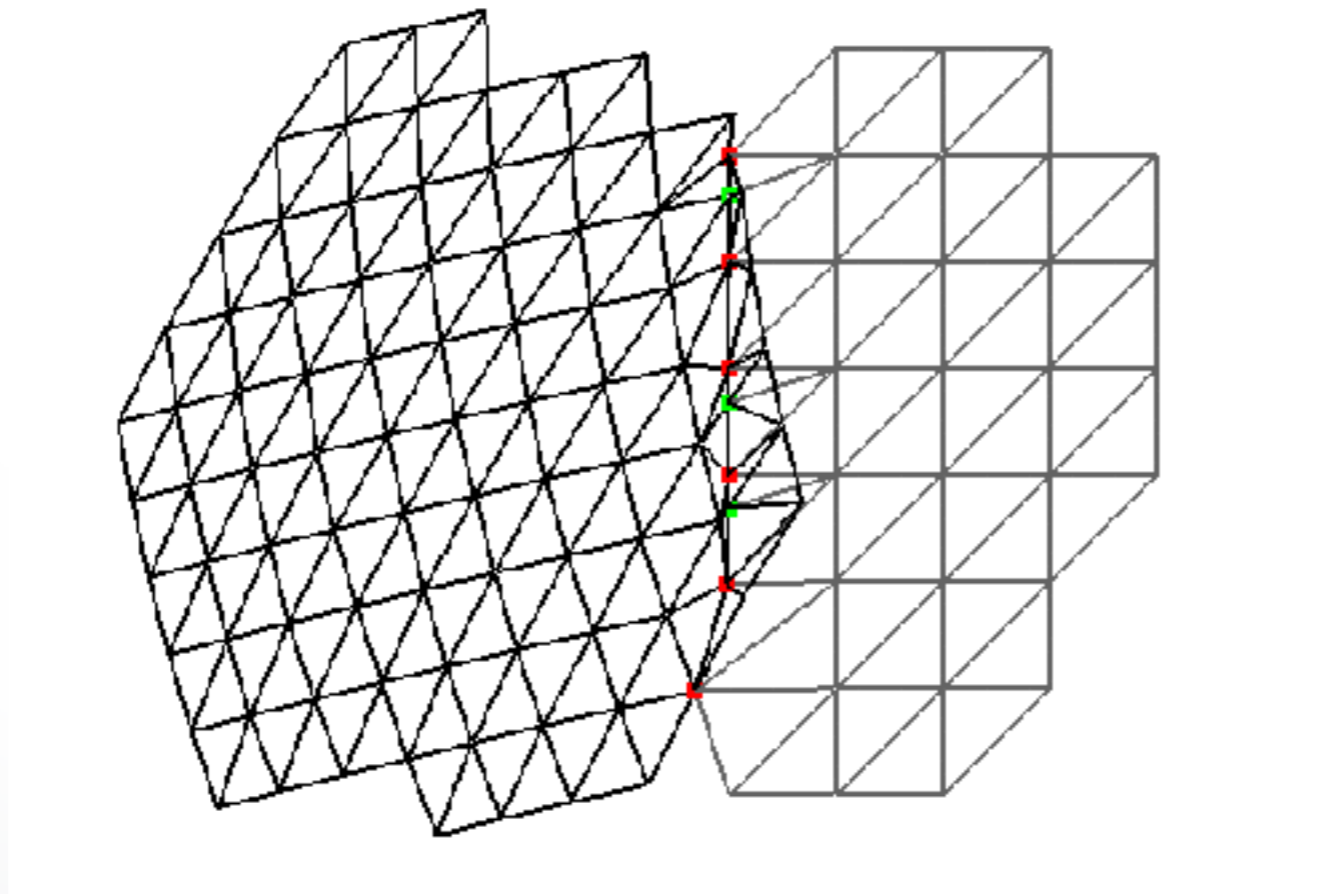
“Zipper” several scans to one single model



Project & insert boundary vertices

# Explicit Reconstruction

“Zipper” several scans to one single model

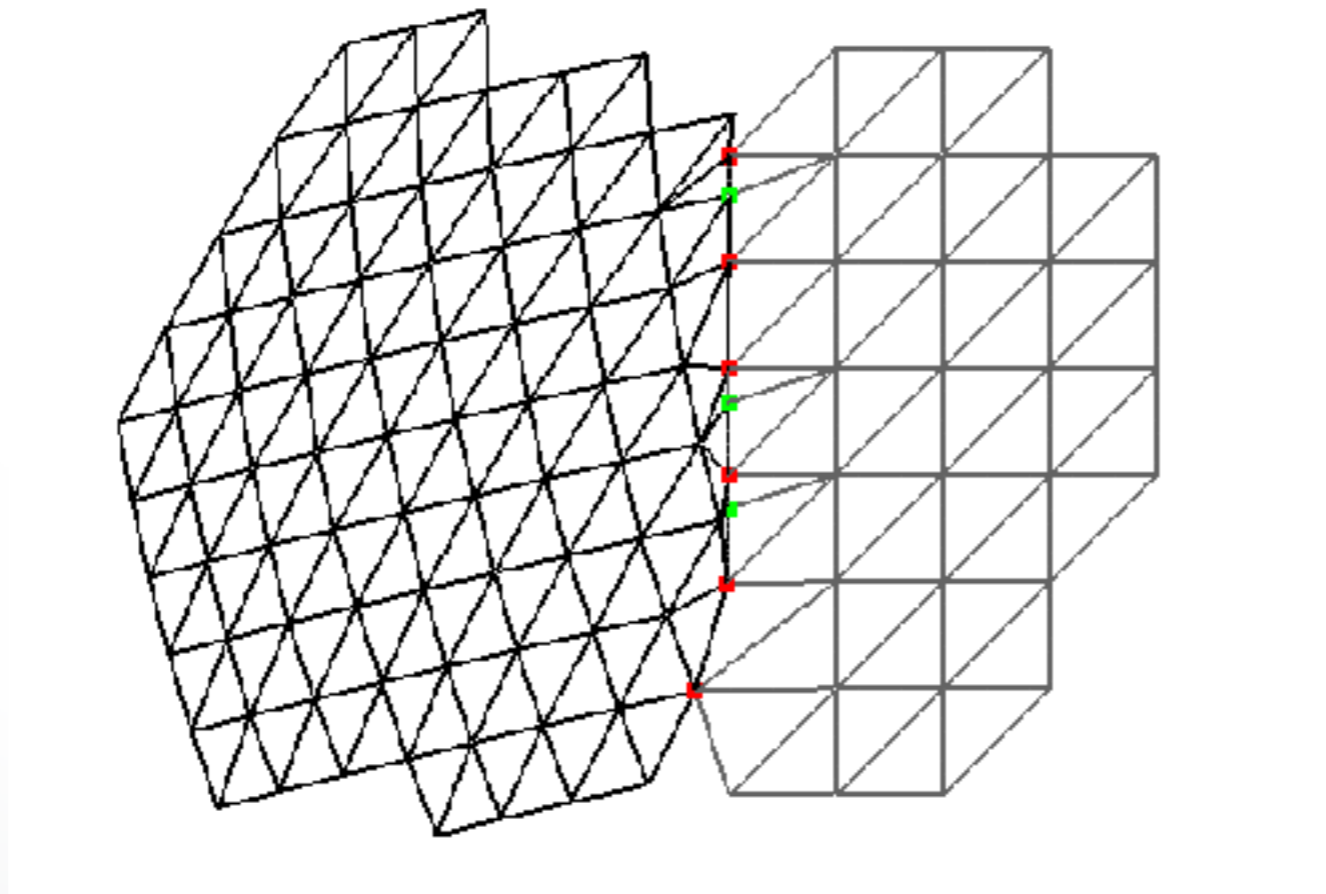


Intersect boundary edges



# Explicit Reconstruction

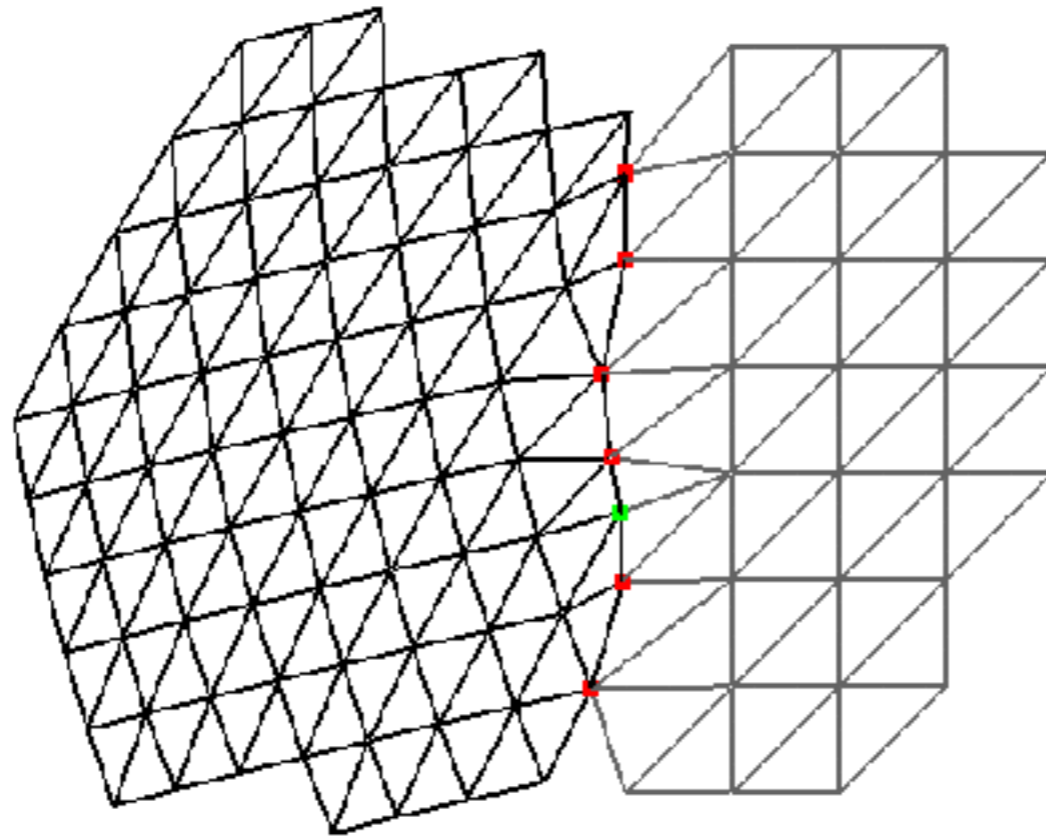
“Zipper” several scans to one single model



Discard overlap region

# Explicit Reconstruction

“Zipper” several scans to one single model

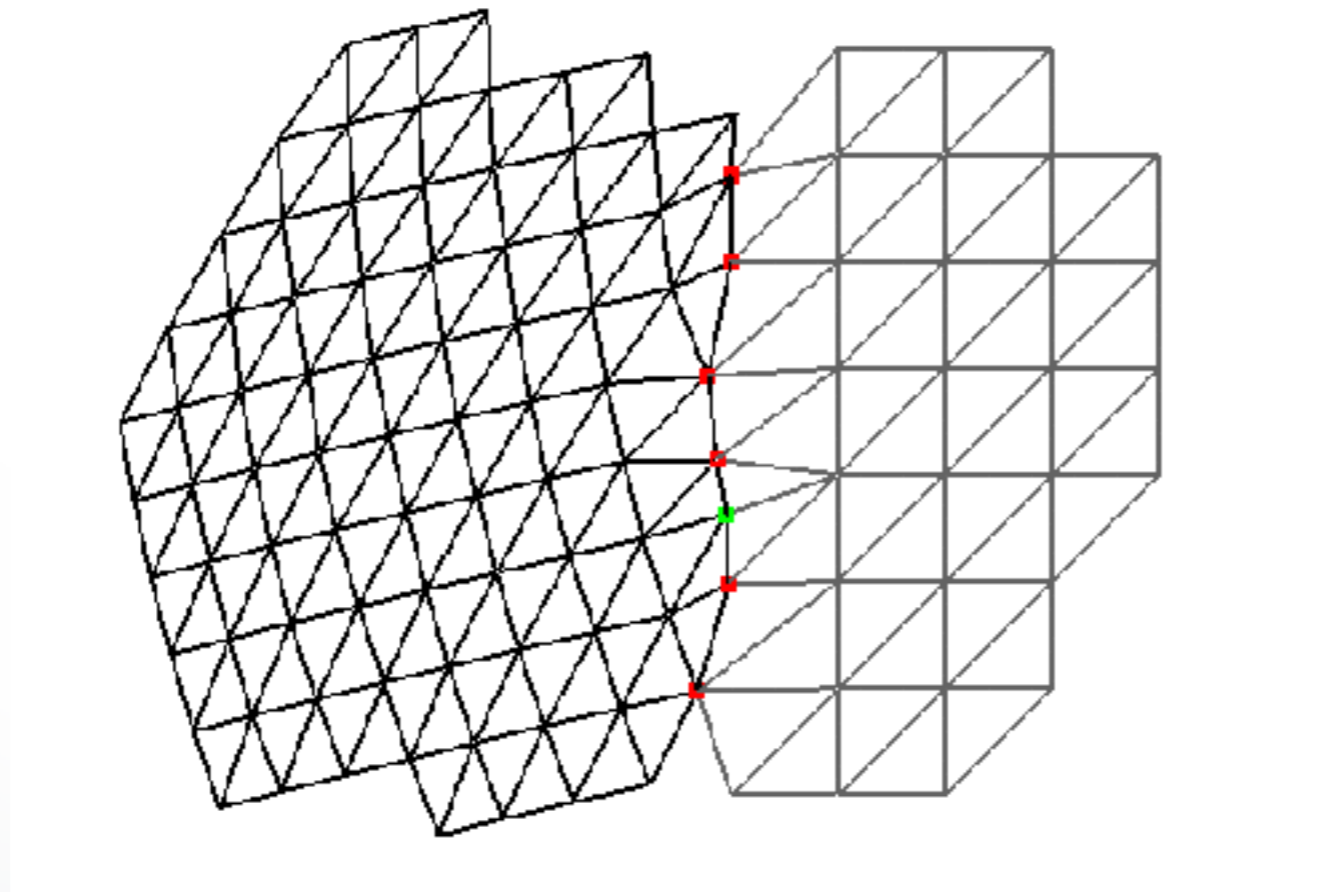


Locally optimize triangulation

# Explicit Reconstruction

“Zipper” several scans to one single model

Problems for intricate geometries...



explicit



implicit



input model

# Mesh Zippering Summary

## Pros:

- Preserves regular structure of each scan
- No additional data structures

## Cons:

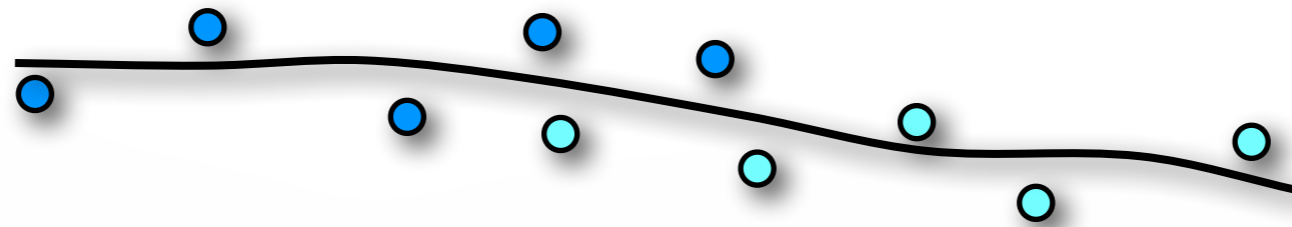
- Zippering can be numerically difficult
- Problems with complex, noisy, incomplete data

# Outline

- **Explicit Reconstruction**
  - Zippering range scans
- **Implicit Reconstruction**
  - **SDF from point clouds**
  - SDF from range scans
  - Poisson surface reconstruction

# Implicit Reconstruction

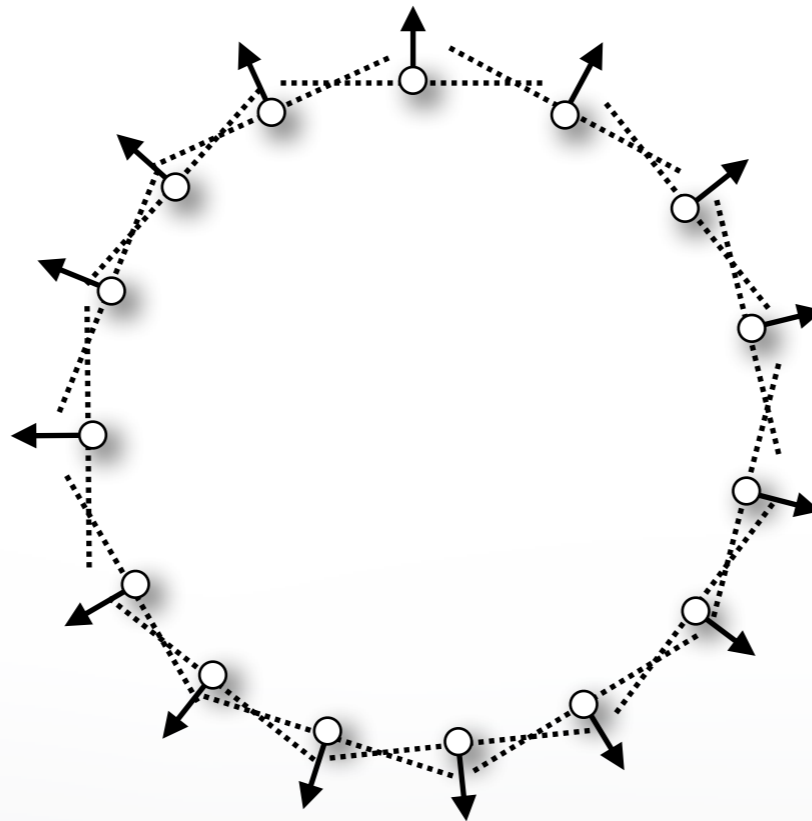
- Estimate signed distance function (SDF)
- Extract Zero isosurface by Marching Cubes
- Approximation of input points
- Watertight manifold by construction



# Signed Distance Function

## Construct SDF from point samples

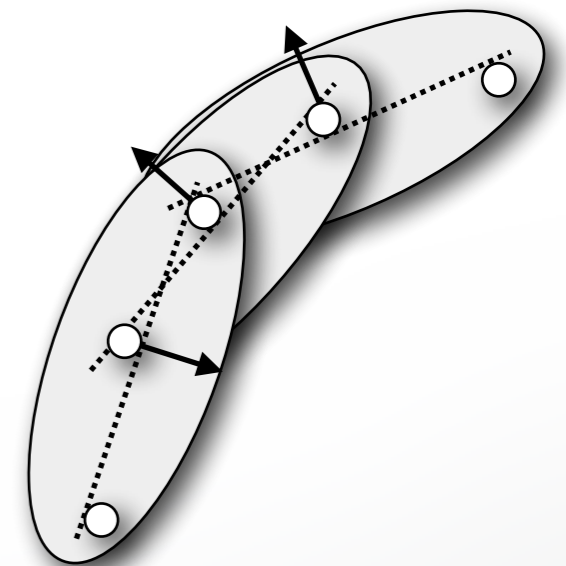
- Distance to points is not enough
- Need inside/outside information
- Requires normal vectors



# Normal Estimation

## Find normal $\mathbf{n}_i$ for each sample point $\mathbf{p}_i$

- Examine local neighborhood for each point
  - Set of  $k$  nearest neighbors
- Compute best approximating tangent plane
  - Covariance analysis
- Determine normal orientation
  - Minimal Spanning Tree propagation

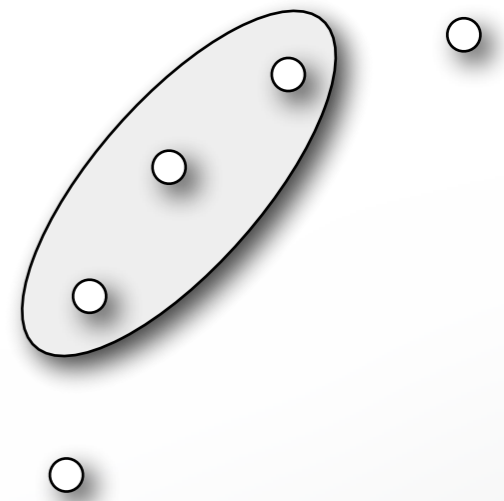




# Normal Estimation

Find normal  $\mathbf{n}_i$  for each sample point  $\mathbf{p}_i$

- Examine local neighborhood for each point
  - Set of  $k$  nearest neighbors
- Compute best approximating tangent plane
  - Covariance analysis
- Determine normal orientation
  - Minimal Spanning Tree propagation



# Normal Estimation

## Find closest point of a query point

- Find closest point of a query point
  - Brute force:  $O(n)$  complexity

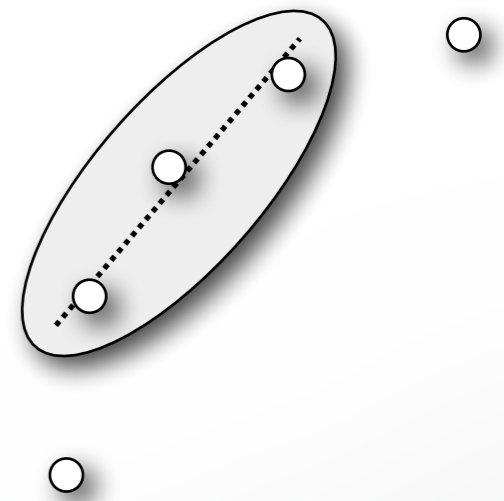
## Use Hierarchical BSP tree

- Binary space partitioning tree (general version of kD-tree)
- Recursively partition 3D space by planes
- Tree should be balanced, put plane at median
- $\log(n)$  tree levels, complexity  $\log(n)$

# Normal Estimation

## Find normal $\mathbf{n}_i$ for each sample point $\mathbf{p}_i$

- Examine local neighborhood for each point
  - Set of  $k$  nearest neighbors
- **Compute best approximating tangent plane**
  - Covariance analysis
- Determine normal orientation
  - Minimal Spanning Tree propagation



# Plane Fitting

Fit a plane with center  $\mathbf{c}$  and normal  $\mathbf{n}$  to a set of points  $\{\mathbf{p}_1, \dots, \mathbf{p}_m\}$

Minimize least squares error

$$E(\mathbf{c}, \mathbf{n}) = \sum_{i=1}^m (\mathbf{n}^T (\mathbf{p}_i - \mathbf{c}))^2$$

Subject to non-linear constraint

$$\|\mathbf{n}\| = 1$$

# Plane Fitting

## Reformulate error function

$$\begin{aligned} E(\mathbf{c}, \mathbf{n}) &= \sum_{i=1}^m (\mathbf{n}^T (\mathbf{p}_i - \mathbf{c}))^2 \\ &= \sum_{i=1}^m (\mathbf{n}^T \hat{\mathbf{p}}_i)^2 \quad (\text{with } \hat{\mathbf{p}}_i := \mathbf{p}_i - \mathbf{c}) \\ &= \sum_{i=1}^m \hat{\mathbf{p}}_i^T \mathbf{n} \mathbf{n}^T \hat{\mathbf{p}}_i \quad (\text{version 1}) \\ &= \sum_{i=1}^m \mathbf{n}^T \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^T \mathbf{n} \quad (\text{version 2}) \end{aligned}$$

# Determine $\mathbf{c}$ from version 1

Derivative of  $E(\mathbf{c}, \mathbf{n})$  w.r.t.  $\mathbf{c}$  has to vanish

$$\frac{\partial E(\mathbf{c}, \mathbf{n})}{\partial \mathbf{c}} = \sum_{i=1}^m -2 \mathbf{n} \mathbf{n}^T \hat{\mathbf{p}}_i = -2 \mathbf{n} \mathbf{n}^T \sum_{i=1}^m \hat{\mathbf{p}}_i \stackrel{!}{=} 0$$

This is only possible for

$$\sum_{i=1}^m \hat{\mathbf{p}}_i = 0 \quad \Rightarrow \quad \mathbf{c} = \frac{1}{m} \sum_{i=1}^m \mathbf{p}_i$$

Plane center is barycenter of points  $\mathbf{p}_i$

# Determine $\mathbf{n}$ from version 2

Represent  $\mathbf{n}$  in basis  $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$

$$\mathbf{n} = \alpha_1 \mathbf{e}_1 + \alpha_2 \mathbf{e}_2 + \alpha_3 \mathbf{e}_3$$

Since  $\mathbf{n}$  has unit length we get

$$1 = \mathbf{n}^T \mathbf{n} = \alpha_1^2 + \alpha_2^2 + \alpha_3^2$$

Insert into energy formulation

$$\mathbf{n}^T \mathbf{C} \mathbf{n} = \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_2 + \alpha_3^2 \lambda_3 \geq \alpha_1^2 \lambda_3 + \alpha_2^2 \lambda_3 + \alpha_3^2 \lambda_3 = \lambda_3$$

Minimum is achieved for  $\alpha_1 = \alpha_2 = 0, \alpha_3 = 1 \Rightarrow \mathbf{n} = \mathbf{e}_3$

# Principal Component Analysis

Plane center is barycenter of points

$$\mathbf{c} = \frac{1}{m} \sum_{i=1}^m \mathbf{p}_i$$

Normal is eigenvector w.r.t. smallest eigenvalue of covariance matrix

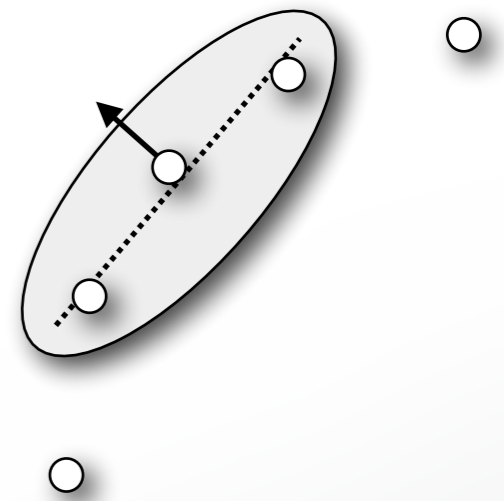
$$\mathbf{C} = \sum_{i=1}^m (\mathbf{p}_i - \mathbf{c})(\mathbf{p}_i - \mathbf{c})^T$$



# Normal Estimation

## Find normal $\mathbf{n}_i$ for each sample point $\mathbf{p}_i$

- Examine local neighborhood for each point
  - Set of  $k$  nearest neighbors
- Compute best approximating tangent plane
  - Covariance analysis
- **Determine normal orientation**
  - Minimal Spanning Tree propagation



# Normal Orientation

## Riemannian graph connects neighboring points

- Edge  $(ij)$  exists if  $\mathbf{p}_i \in k\text{NN}(\mathbf{p}_j)$  or  $\mathbf{p}_j \in k\text{NN}(\mathbf{p}_i)$

## Propagate normal orientation through graph

- For neighbors  $\mathbf{p}_i, \mathbf{p}_j$  Flip  $\mathbf{n}_j$  if  $\mathbf{n}_i^\top \mathbf{n}_j < 0$
- Fails at sharp edges/corners

## Propagate along “save” paths (parallel normals)

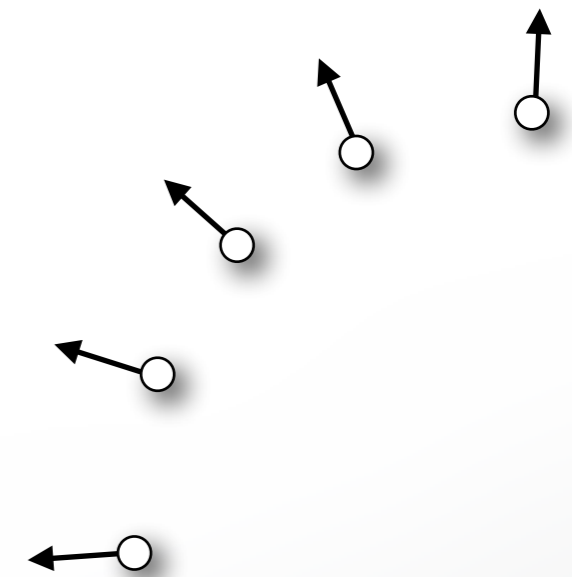
- Minimum spanning tree with angle-based edge weights

$$w_{ij} = 1 - |\mathbf{n}_i^\top \mathbf{n}_j|$$

# Normal Estimation

## Find normal $\mathbf{n}_i$ for each sample point $\mathbf{p}_i$

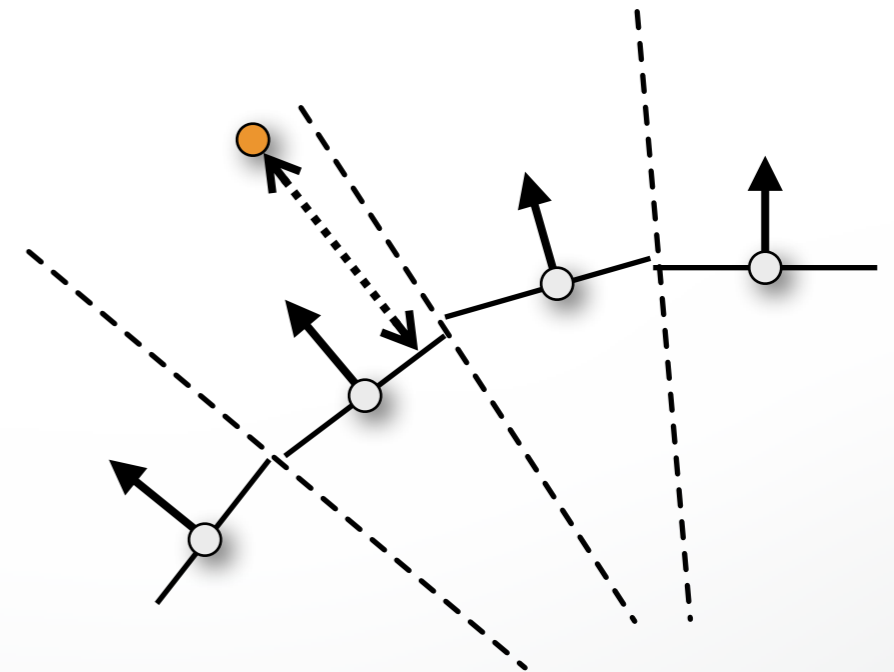
- Examine local neighborhood for each point
  - Set of  $k$  nearest neighbors
- Compute best approximating tangent plane
  - Covariance analysis
- **Determine normal orientation**
  - Minimal Spanning Tree propagation



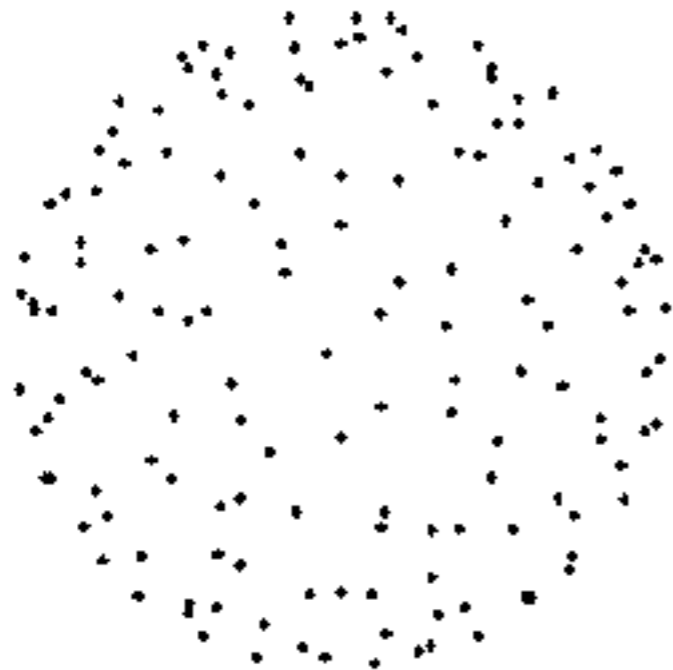
# Normal Estimation

## Distance from tangent planes [Hoppe 92]

- Points + normals determine local tangent planes
- Use distance from closest point's tangent plane
- Linear approximation in Voronoi cell
- Simple and efficient, but SDF is only  $C^{-1}$



# Hoppe '92 Reconstruction



150 samples



reconstruction  
on  $50^3$  grid

# Smooth SDF Approximation

## Scattered data interpolation problem

- On-surface constraints  $\text{dist}(\mathbf{p}_i) = 0$
- Avoid trivial solution  $\text{dist} \equiv 0$
- Off-surface constraints  $\text{dist}(\mathbf{p}_i + \mathbf{n}_i) = 1$

## Radial basis functions (RBFs)

- Well suited for smooth interpolation
- Sum of shifted, weighted kernel functions

$$\text{dist}(\mathbf{x}) = \sum_i w_i \cdot \varphi(\|\mathbf{x} - \mathbf{c}_i\|)$$

# RBF Interpolation

Interpolate on- and off-surface constraints

$$\text{dist}(\mathbf{x}_j) = \sum_{i=1}^n w_i \cdot \varphi(\|\mathbf{x}_j - \mathbf{c}_i\|) \stackrel{!}{=} d_j, \quad j = 1, \dots, n$$

Choose centers  $\mathbf{c}_i$  as constrained points  $\mathbf{x}_i$

Solve symmetric linear system for weights  $w_i$

$$\begin{pmatrix} \varphi(\|\mathbf{x}_1 - \mathbf{x}_1\|) & \cdots & \varphi(\|\mathbf{x}_1 - \mathbf{x}_n\|) \\ \vdots & \ddots & \vdots \\ \varphi(\|\mathbf{x}_n - \mathbf{x}_1\|) & \cdots & \varphi(\|\mathbf{x}_n - \mathbf{x}_n\|) \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix}$$

# RBF Interpolation

## Wendland basis functions

$$\varphi(r) = \left(1 - \frac{r}{\sigma}\right)_+^4 \left(4\frac{r}{\sigma} + 1\right)$$

- Compactly supported in  $[0, \sigma]$
- Leads to sparse, symm. pos. def. linear system
- Resulting SDF is  $\mathcal{C}^2$  smooth
- But surface is not necessarily fair
- Not suited for highly irregular sampling



# Comparison



Hoppe '92



Compact RBF  
Wendland  $C^2$

# RBF Basis Functions

## Triharmonic basis functions

$$\phi(r) = r^3$$

- Globally supported function
- Leads to dense linear system
- SDF is  $\mathcal{C}^2$  smooth
- Provably optimal fairness (see smoothing lecture)

$$\int_{\mathbb{R}^3} \left( \frac{\partial^3 \text{dist}}{\partial x \partial x \partial x} \right)^2 + \left( \frac{\partial^3 \text{dist}}{\partial x \partial x \partial y} \right)^2 + \dots + \left( \frac{\partial^3 \text{dist}}{\partial z \partial z \partial z} \right)^2 dx dy dz \rightarrow \min$$

- Works well for irregular sampling

# Comparison



Hoppe '92



Compact RBF  
Wendland  $C^2$



Global RBF  
Triharmonic

# Complexity Considerations

## Solve the linear system for RBF weights

- Hard to solve for large number of samples

## Compactly supported RBFs

- Sparse linear system
- Efficient CG or sparse Cholesky solver (later...)

## Greedy RBF fitting [Carr01]

- Start with a few RBFs only
- Add more RBFs in region of large error

# SDF From Points

## Pros:

- Result is a closed 2-manifold surface
- Suitable for noisy input data

## Cons:

- Solve linear system of RBF weights
- Result is uniformly over-tessellated → mesh decimation
- Can contain poorly shaped triangles → remeshing

# Outline

- **Explicit Reconstruction**
  - Zippering range scans
- **Implicit Reconstruction**
  - SDF from point clouds
  - **SDF from range scans**
  - Poisson surface reconstruction

# Weighted Average of SDFs

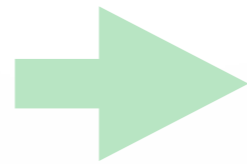
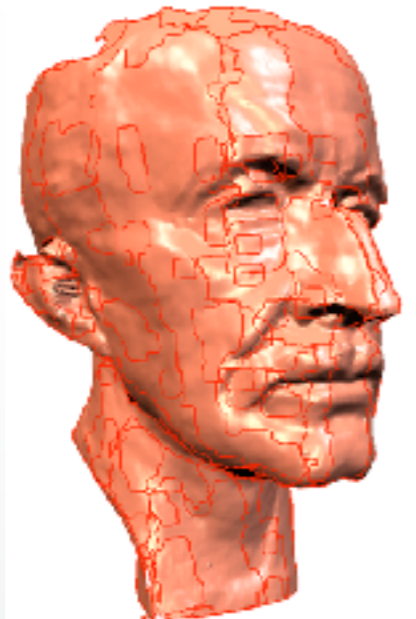
**Individual SDFs of each scan:**  $d_i(\mathbf{x})$

- Distance along scanner's line of sight

**Respective weighting functions:**  $w_i(\mathbf{x})$

- Take scanning angle into account

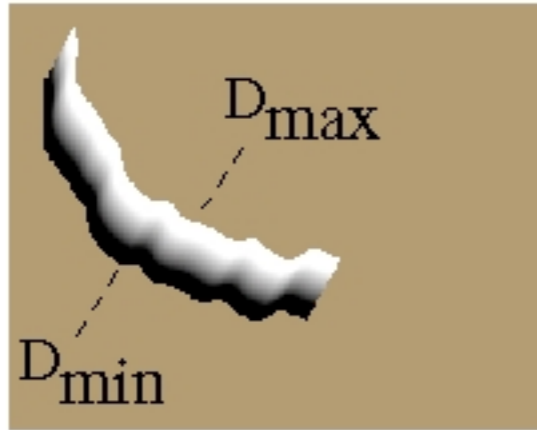
**Global SDF as weighted average**



$$D(\mathbf{x}) = \frac{\sum_i w_i(\mathbf{x}) d_i(\mathbf{x})}{\sum_i w_i(\mathbf{x})}$$

# Weighted Average of SDFs

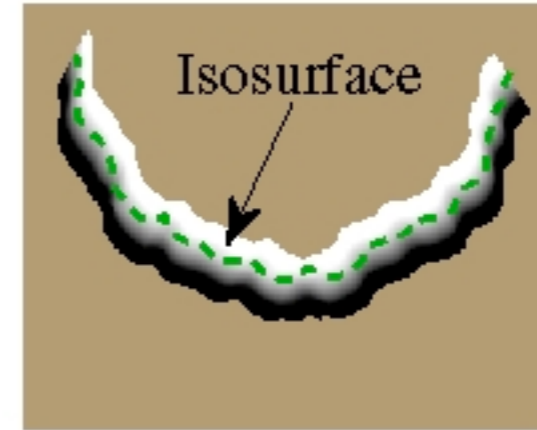
SDFs



$d_1$

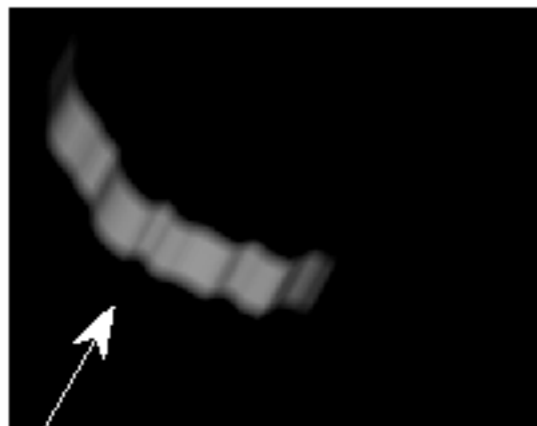


$d_2$



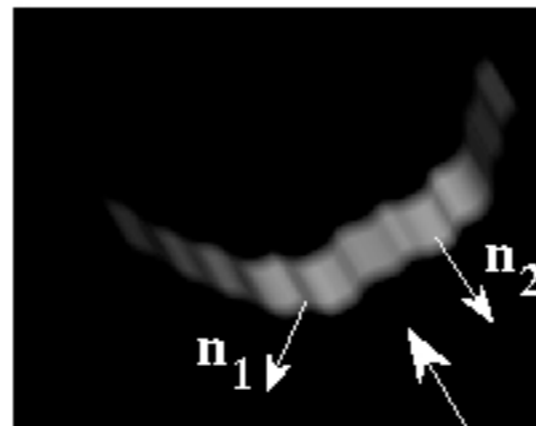
$(w_1d_1+w_2d_2)/(w_1+w_2)$

Weight Functions



Sensor

$w_1$



Sensor

$w_2$



$w_1+w_2$

[Curless, Levoy96]

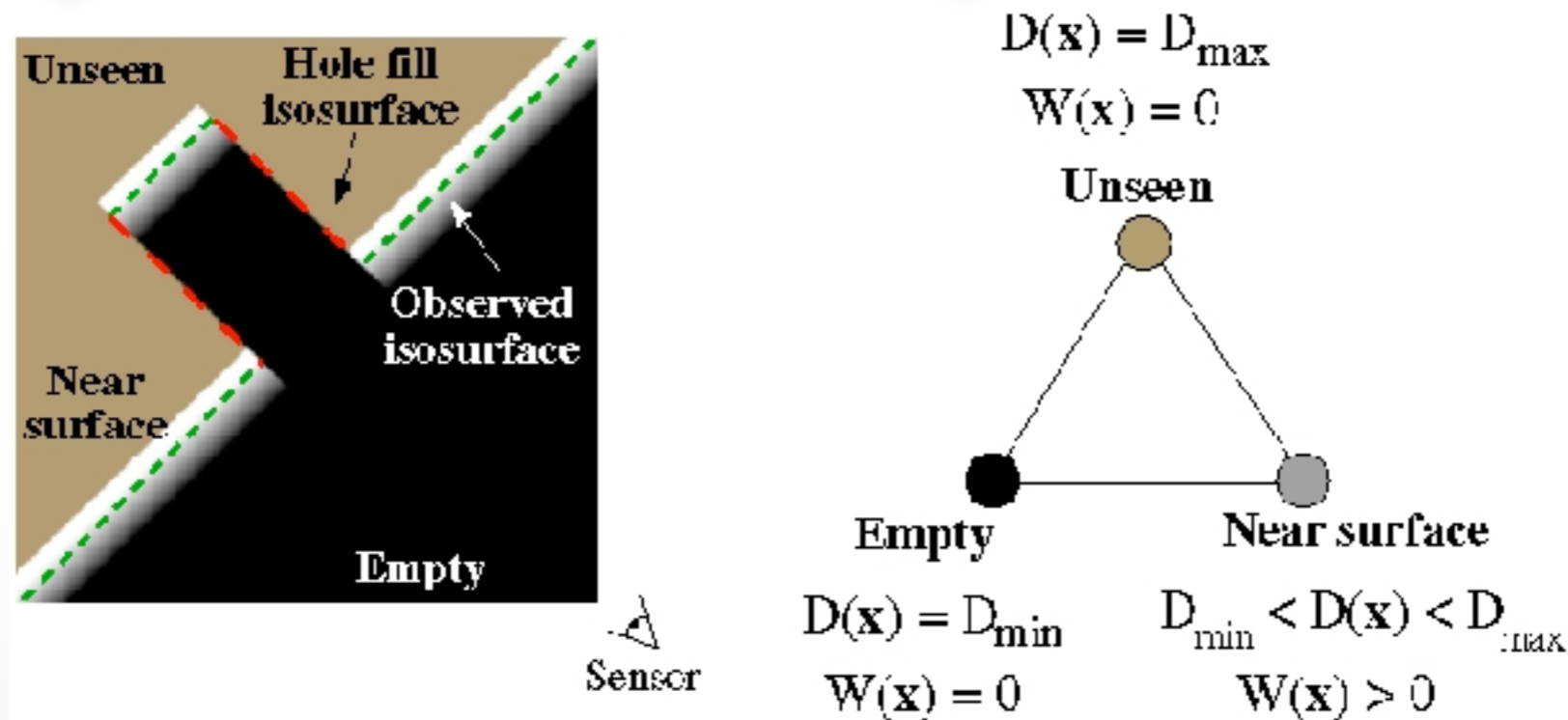


# Automatic Hole Filling

## Classify grid voxel into three states

- Empty: Between scanner and surface (space carving)
- Unseen: Behind surface
- Near surface: Close to scanned surface

## Marching Cubes automatically fill holes



[Curless, Levoy96]

# Volumetric Reconstruction

Happy Buddha: from original to hardcopy



Photograph of original model

Photograph of painted original

Range surface from one scan

Reconstruction before hole-filling

Reconstruction after hole-filling

Hardcopy

[Curless, Levoy96]

# Digital Michelangelo Project



1G sample points  $\rightarrow$  8M triangles



4G sample points  $\rightarrow$  8M triangles

# SDF From Range Scans

## Pros:

- Result is a closed 2-manifold surface
- Can take scanning information into account

## Cons:

- Result is uniformly over-tesselated → mesh decimation
- Can contain poorly shaped triangles → remeshing

# References

## Reconstruction from point sets

- Hoppe et al.: Surface Reconstruction from Unorganized Points, SIGGRAPH 1992
- Carr etl a.: Reconstruction and representation of 3D objects with radial basis functions, SIGGRAPH 2001

## Reconstruction of range scans

- Curless, Levoy: A Volumetric Method for Building Complex Models from Range Images, SIGGRAPH 1996.
- Levoy et al.: Digital Michelangelo Project: 3D Scanning of Large Statues, SIGGRAPH 2000.

# Outline

- **Explicit Reconstruction**
  - Zippering range scans
- **Implicit Reconstruction**
  - SDF from point clouds
  - SDF from range scans
  - **Poisson surface reconstruction**

# Poisson Surface Reconstruction

- **Michael Kazhdan**, M. Bolitho, and H. Hoppe, SGP 2006
- Source Code available at:
  - <http://www.cs.jhu.edu/~misha/>
- Implementation included in Meshlab

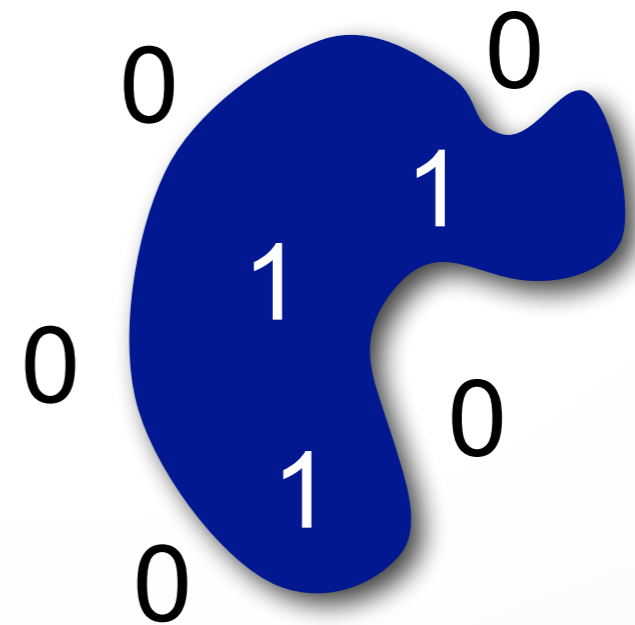


# Poisson Surface Reconstruction

## Indicator Function

- reconstruct the surface by solving for the indicator function of the shape

$$\chi_M(p) = \begin{cases} 1 & \text{if } p \in M \\ 0 & \text{if } p \notin M \end{cases}$$



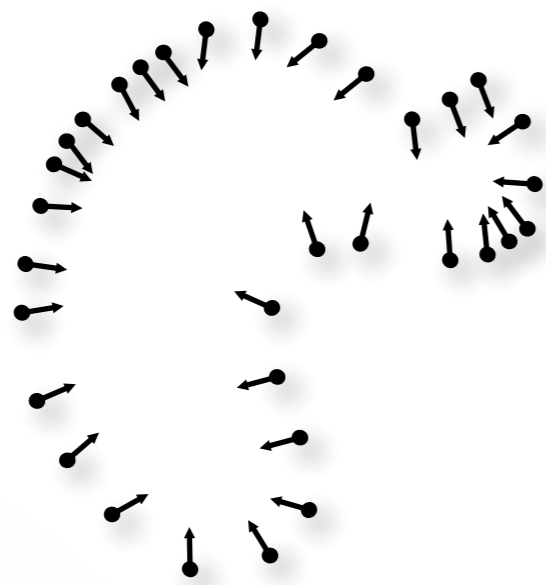
Indicator function

$\chi_M$



# Challenge

How to construct the indicator function?



Oriented points

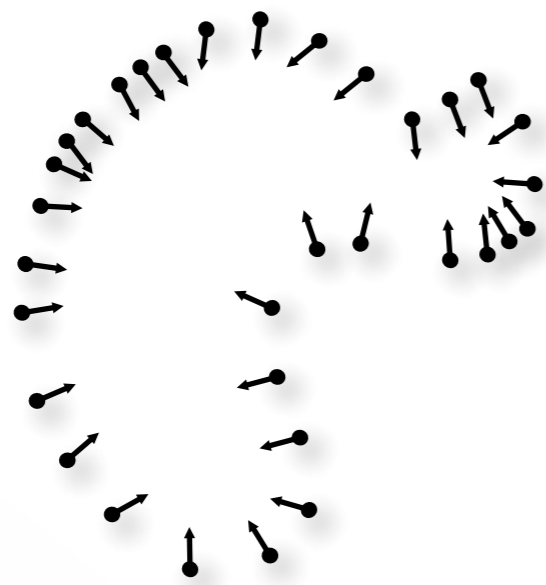


Indicator function

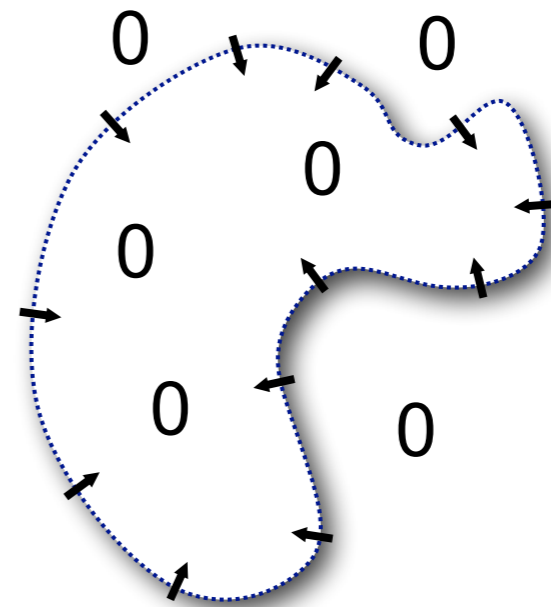
$$\chi_M$$

# Gradient Relationship

There is a relationship between the normal field and gradient of indicator function



Oriented points



Indicator gradient

$$\nabla\chi_M$$

# Integration

Represent the points by a vector field  $\vec{V}$

Find the function  $\chi$  whose gradient best approximates  $\vec{V}$

$$\min_{\chi} \|\nabla\chi - \vec{V}\|$$

# Integration as a Poisson Problem

Represent the points by a vector field  $\vec{V}$

Find the function  $\chi$  whose gradient best approximates  $\vec{V}$

$$\min_{\chi} \|\nabla\chi - \vec{V}\|$$

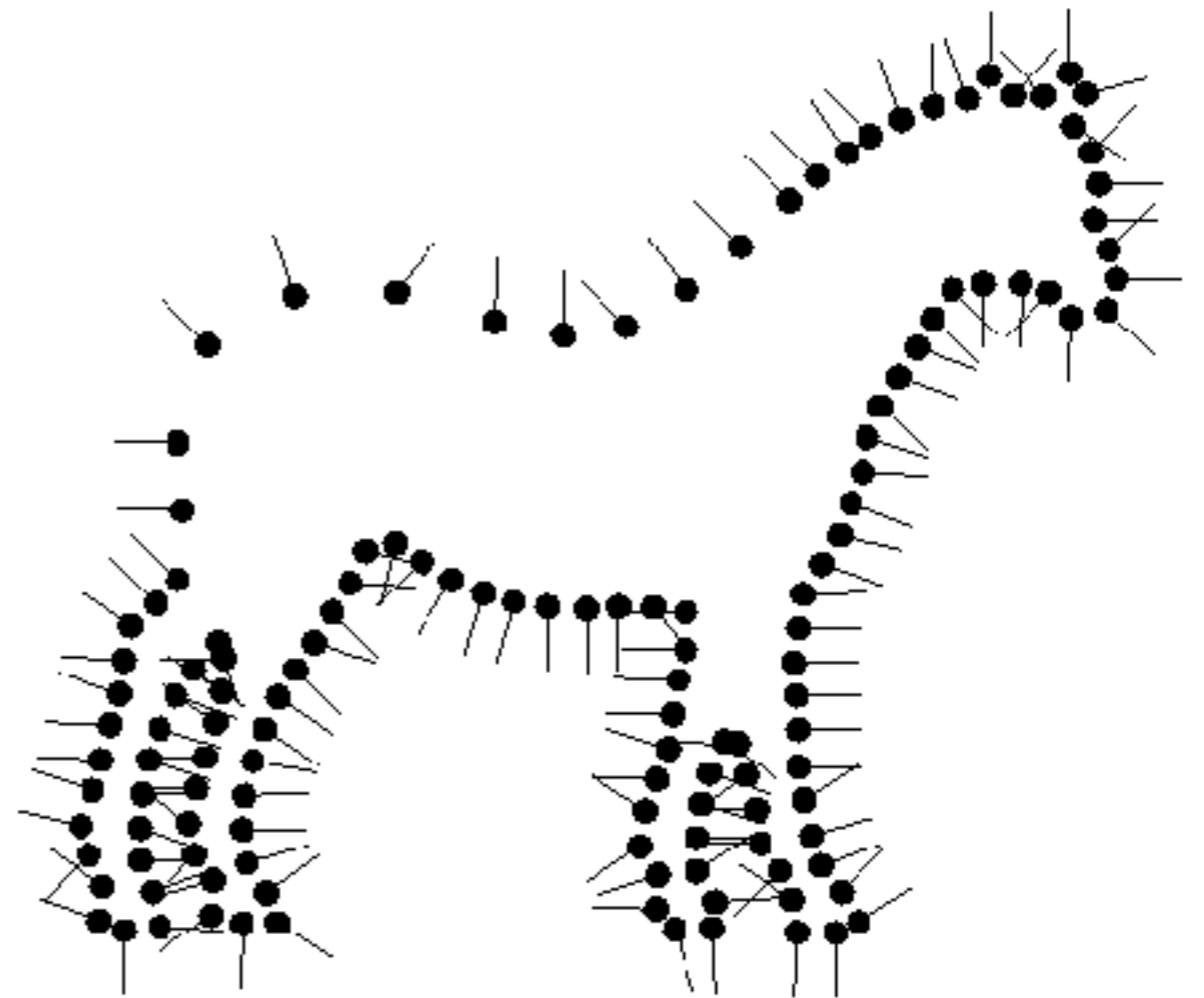
Applying the divergence operator, we can transform this into a Poisson problem:

$$\nabla \times (\nabla\chi) = \nabla \times \vec{V} \iff \Delta\chi = \nabla \times \vec{V}$$

# Implementation: Adaptive Octree

## Given the Points:

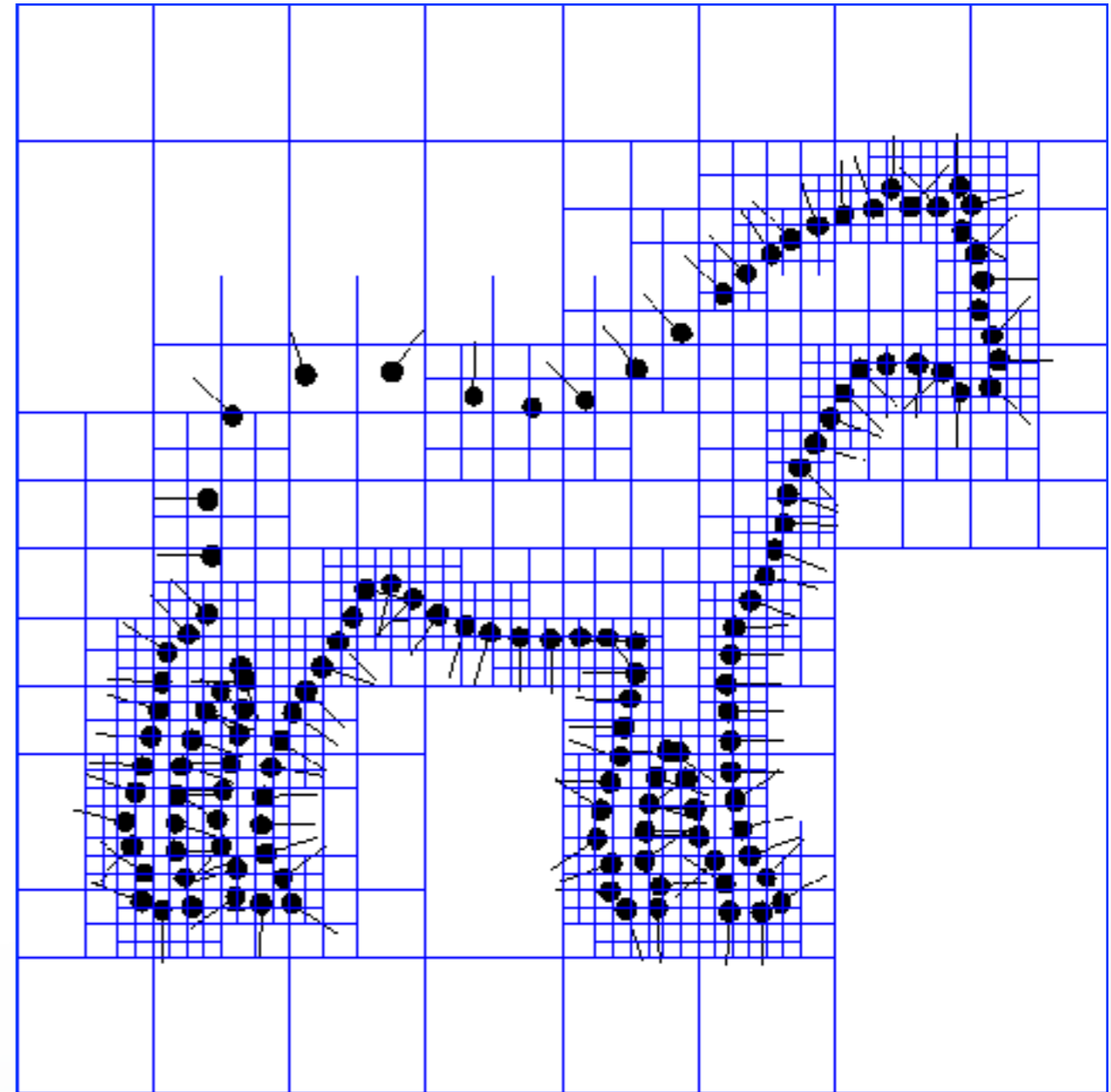
- Set Octree
- Compute vector field
- Compute indicator function
- Extract iso-surface



# Implementation: Adaptive Octree

## Given the Points:

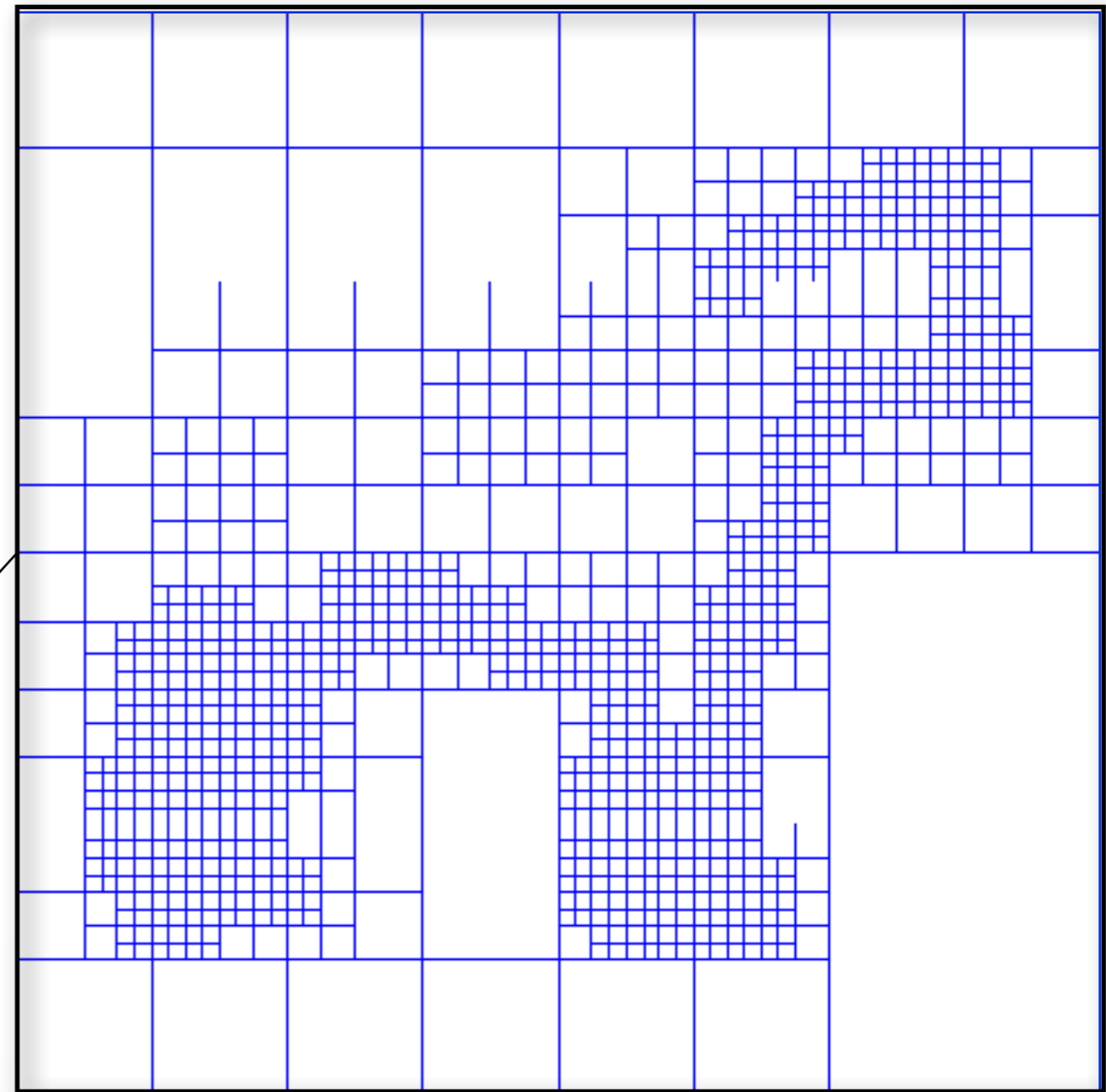
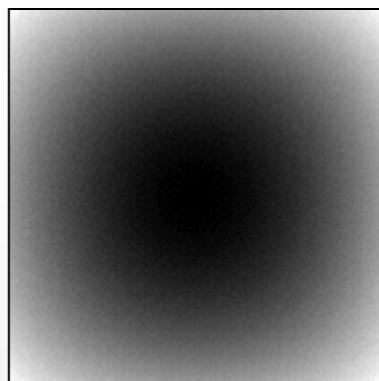
- **Set Octree**
- Compute vector field
- Compute indicator function
- Extract iso-surface



# Implementation: Vector Field

## Given the Points:

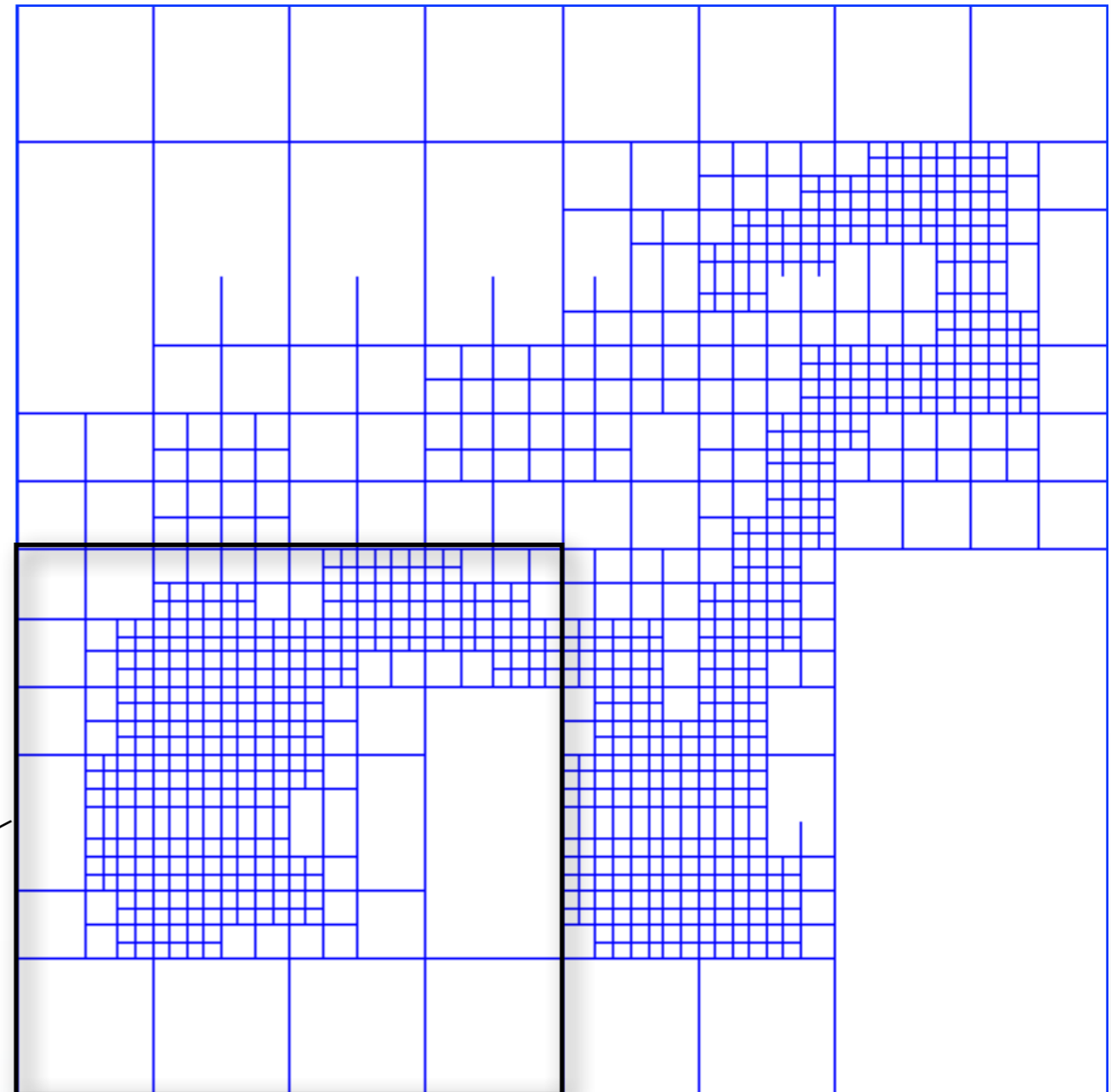
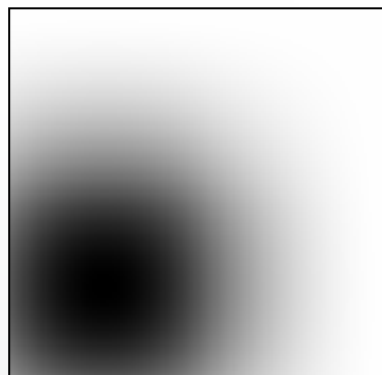
- Set Octree
- **Compute vector field**
  - **Define a function space**
  - Splat the samples
- Compute indicator function
- Extract iso-surface



# Implementation: Vector Field

## Given the Points:

- Set Octree
- **Compute vector field**
  - **Define a function space**
  - Splat the samples
- Compute indicator function
- Extract iso-surface

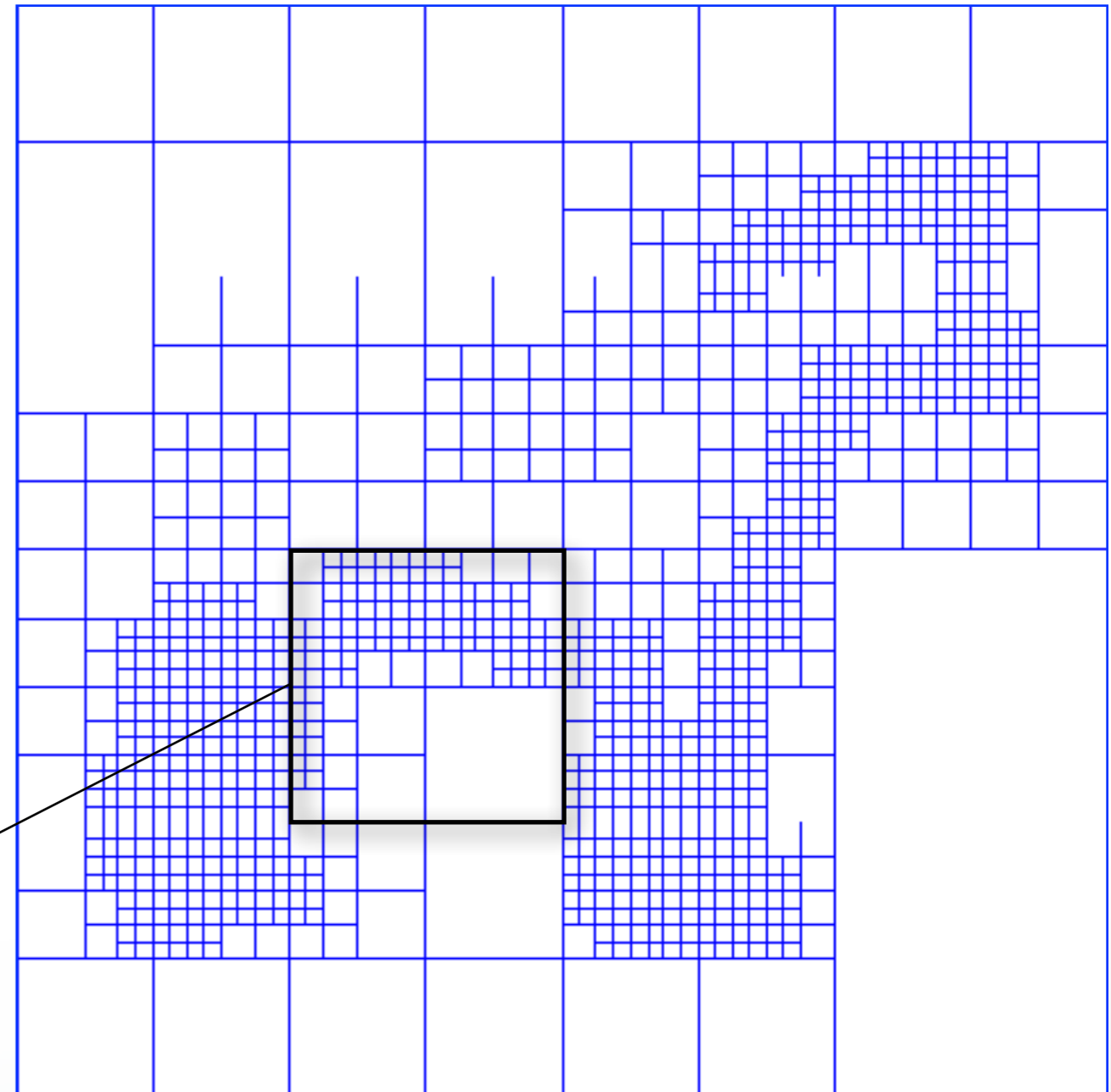
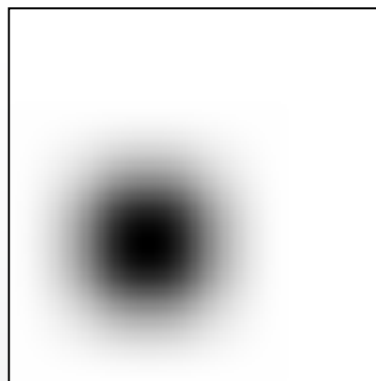




# Implementation: Vector Field

## Given the Points:

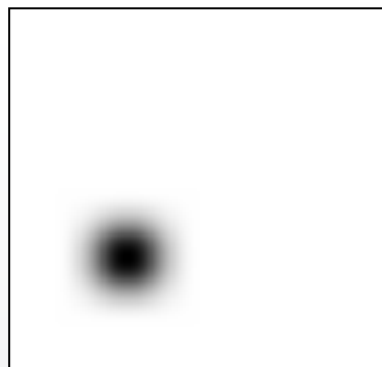
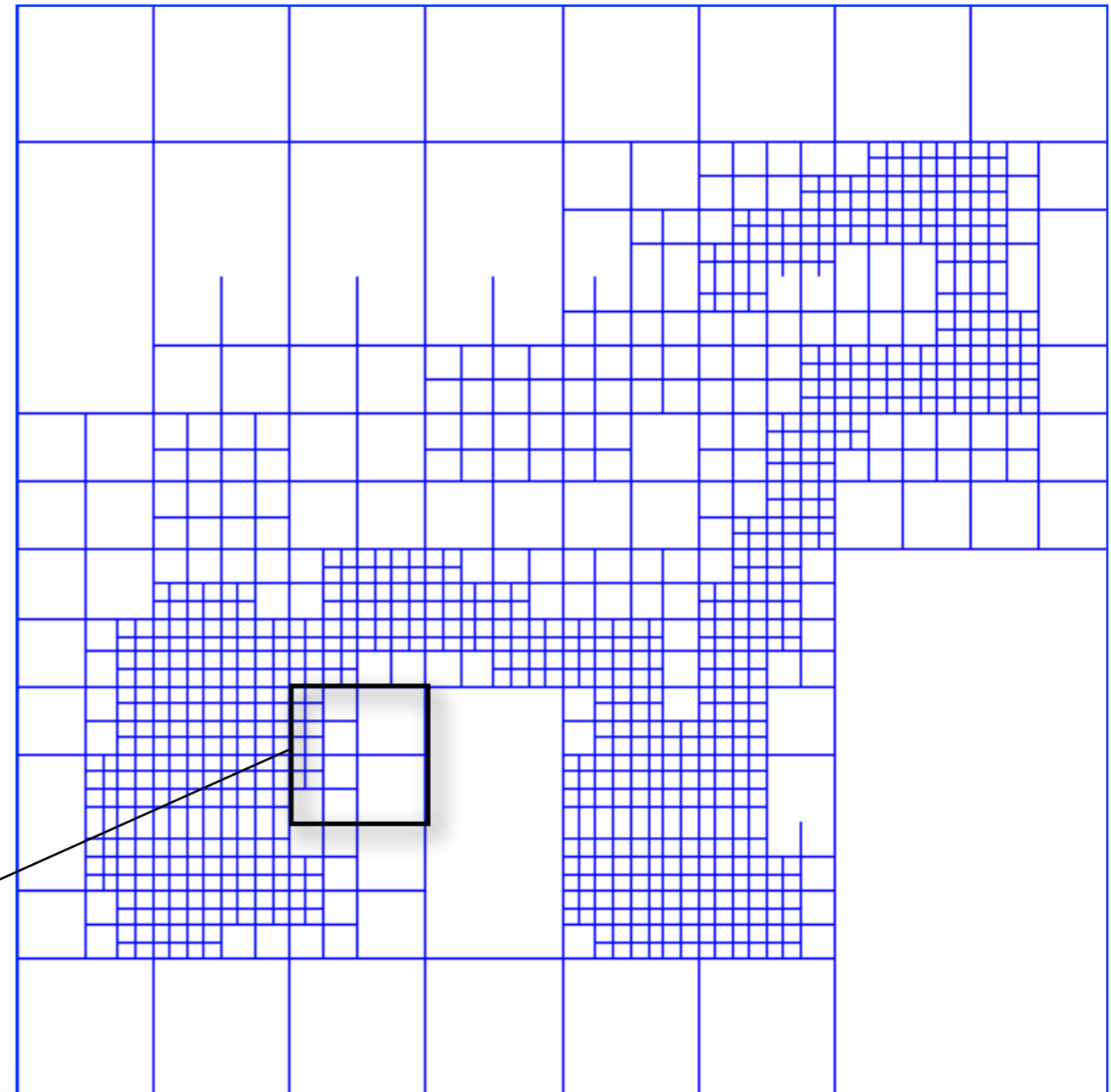
- Set Octree
- **Compute vector field**
  - **Define a function space**
  - Splat the samples
- Compute indicator function
- Extract iso-surface



# Implementation: Vector Field

## Given the Points:

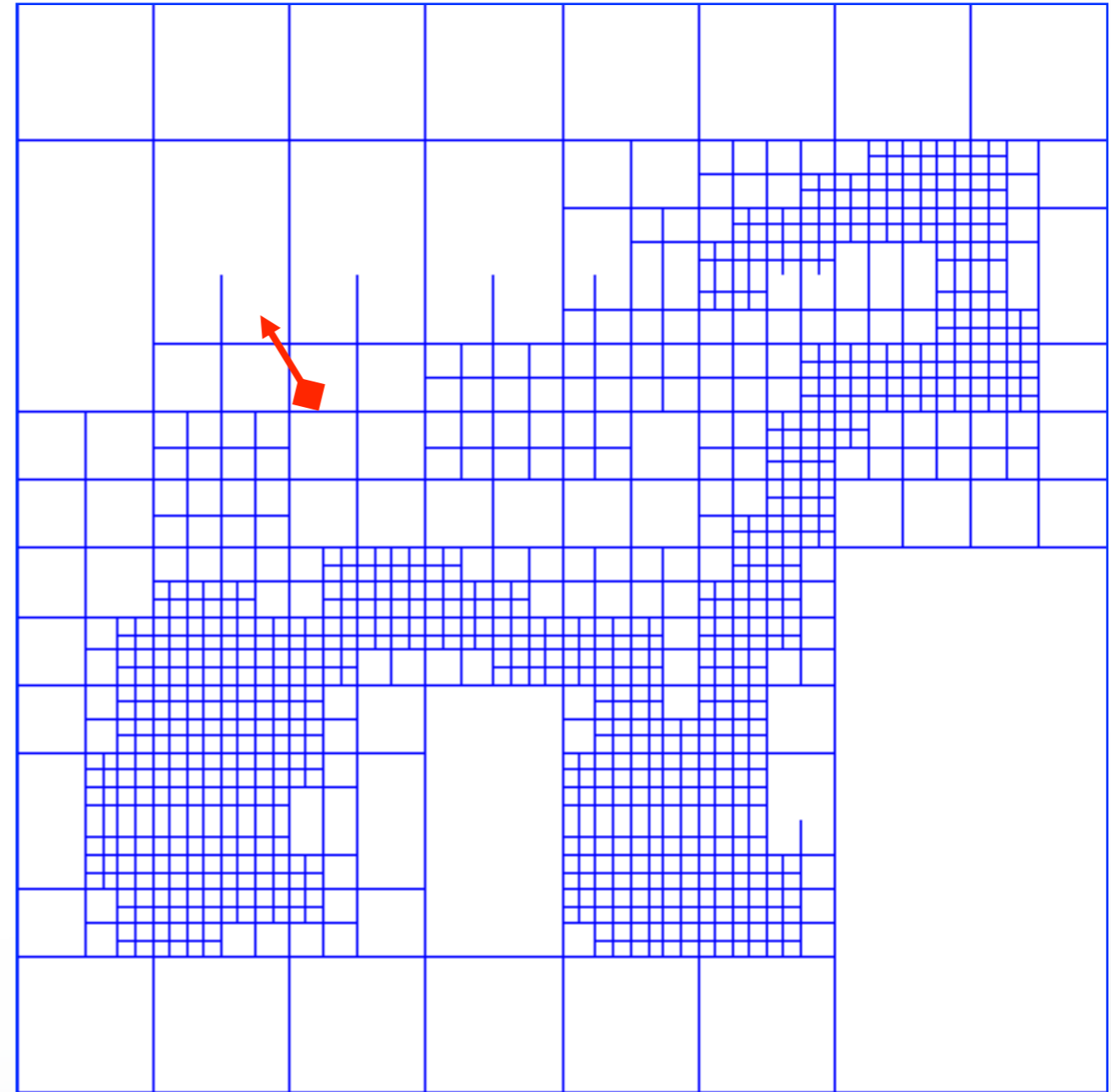
- Set Octree
- **Compute vector field**
  - **Define a function space**
  - Splat the samples
- Compute indicator function
- Extract iso-surface



# Implementation: Vector Field

## Given the Points:

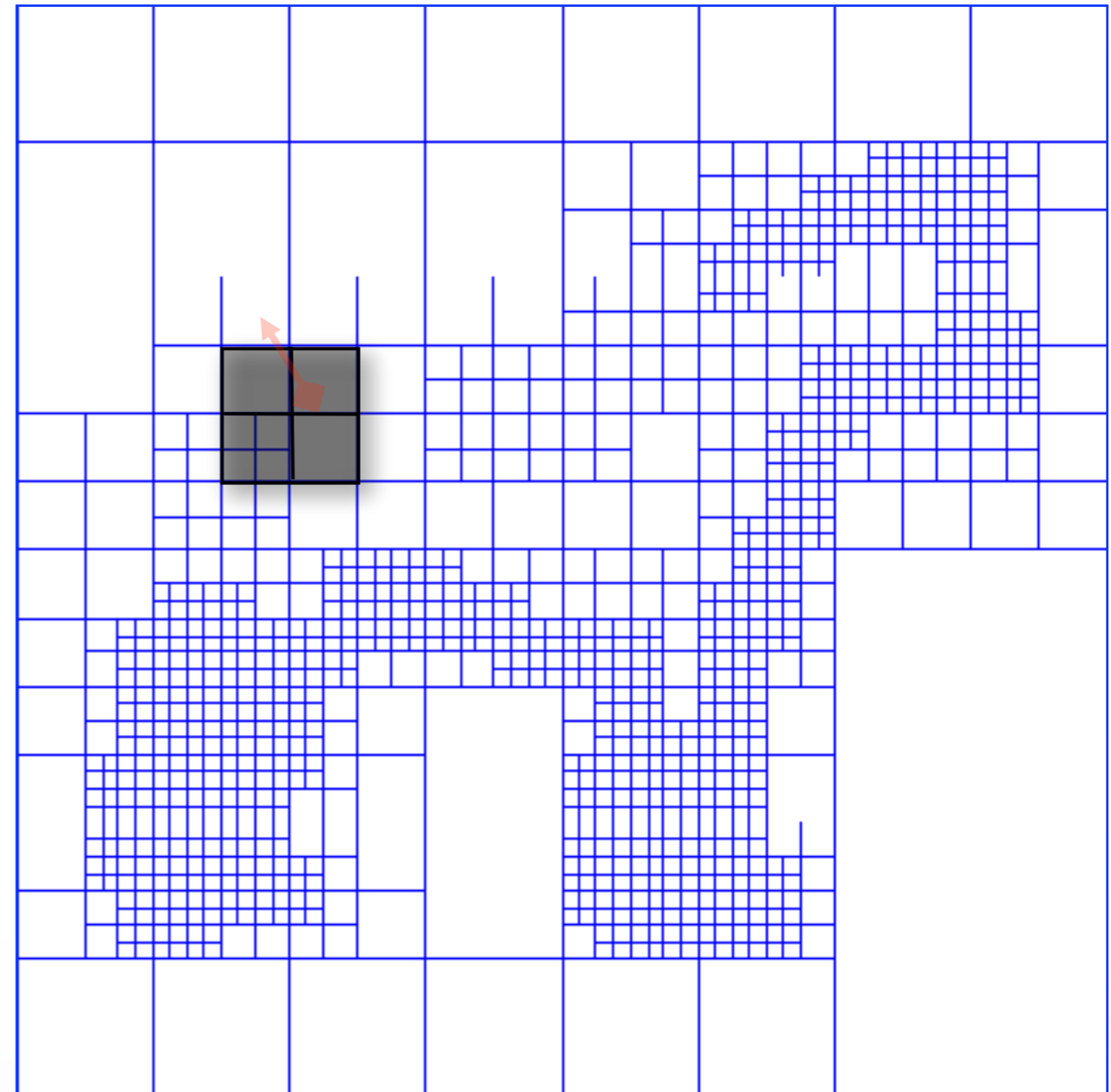
- Set Octree
- **Compute vector field**
  - Define a function space
  - **Splat the samples**
- Compute indicator function
- Extract iso-surface



# Implementation: Vector Field

## Given the Points:

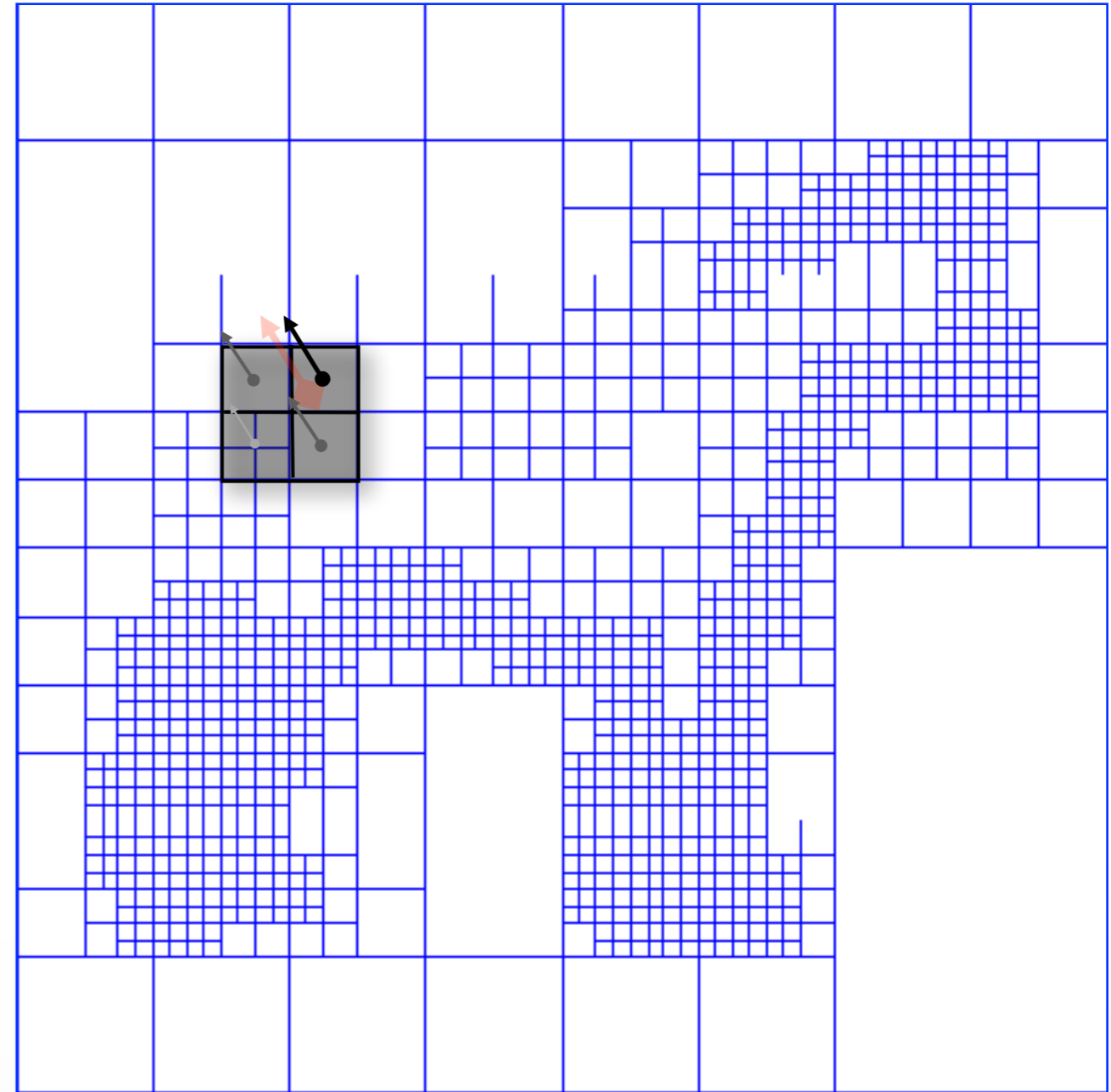
- Set Octree
- **Compute vector field**
  - Define a function space
  - **Splat the samples**
- Compute indicator function
- Extract iso-surface



# Implementation: Vector Field

## Given the Points:

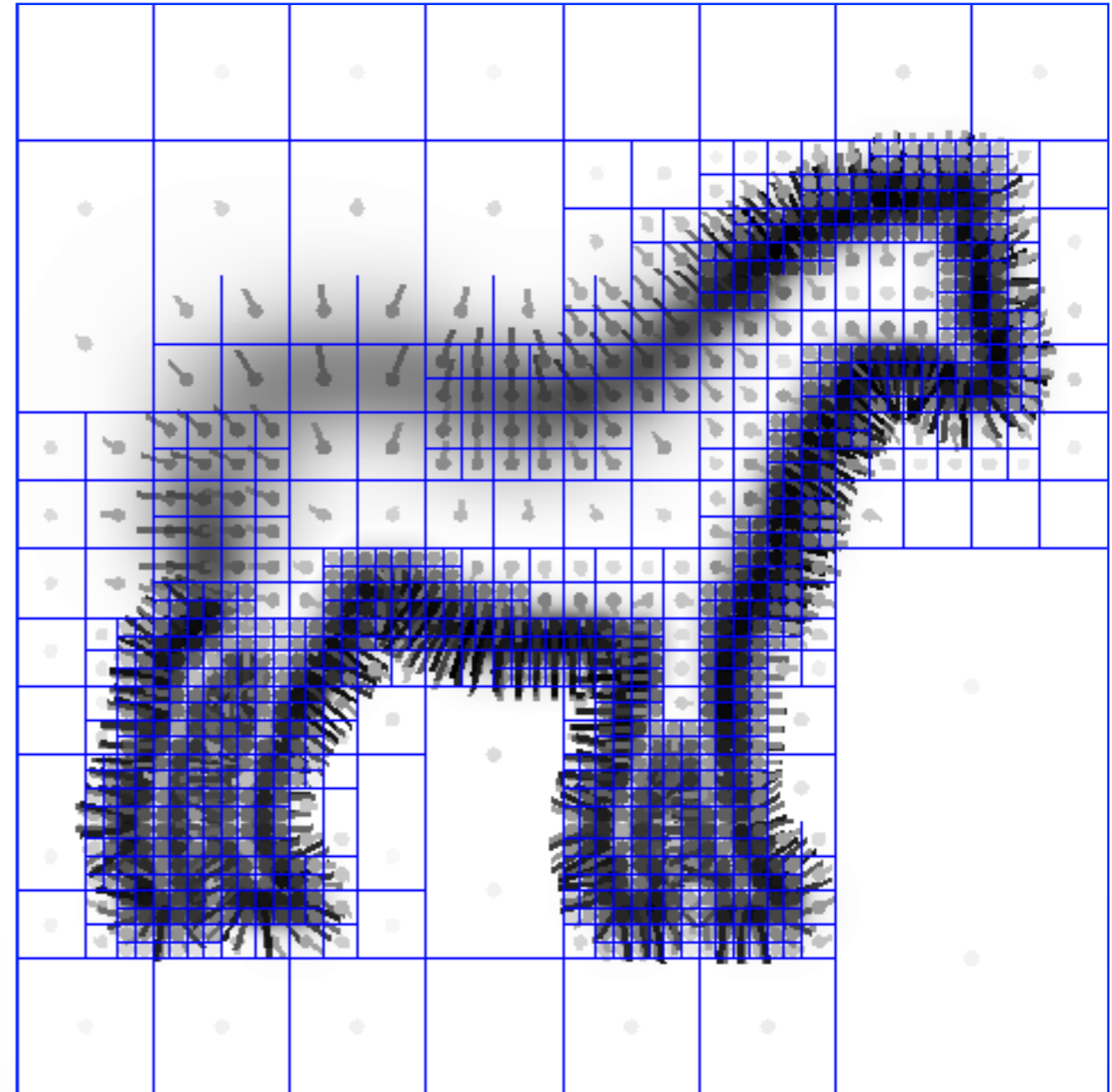
- Set Octree
- **Compute vector field**
  - Define a function space
  - **Splat the samples**
- Compute indicator function
- Extract iso-surface



# Implementation: Vector Field

## Given the Points:

- Set Octree
- **Compute vector field**
  - Define a function space
  - **Splat the samples**
- Compute indicator function
- Extract iso-surface



# Implementation: Indicator Function

## Given the Points:

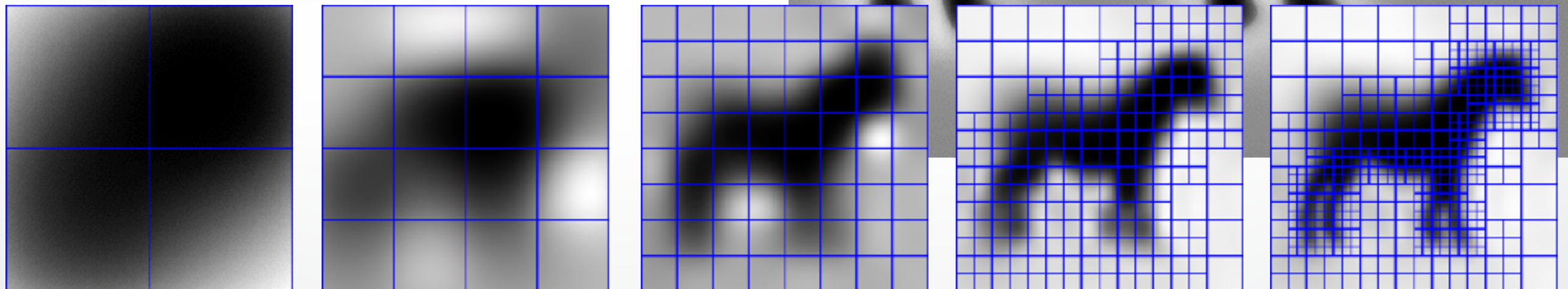
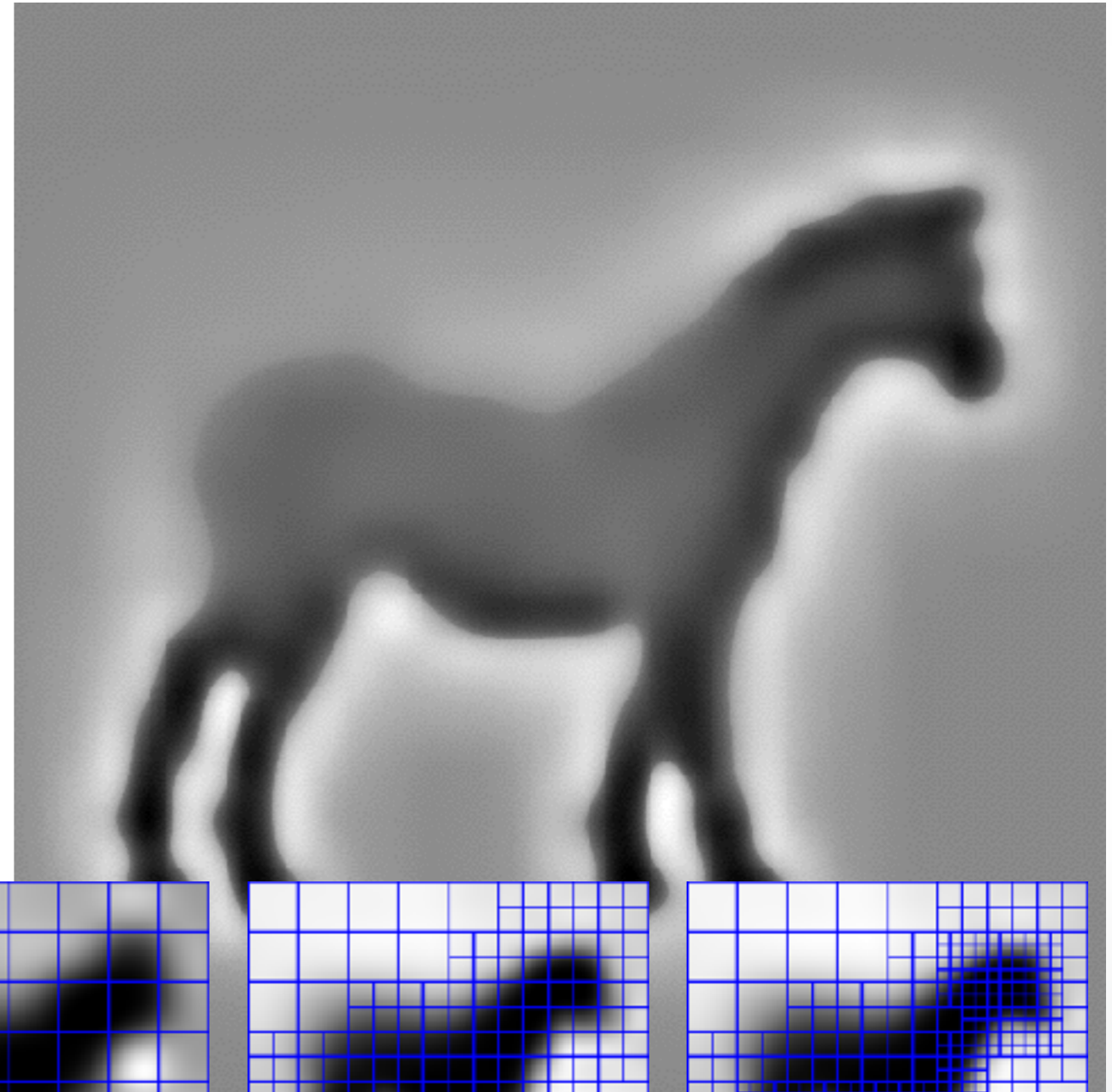
- Set Octree
- Compute vector field
- **Compute indicator function**
  - **Compute divergence**
  - Solve Poisson Equation
- Extract iso-surface



# Implementation: Indicator Function

## Given the Points:

- Set Octree
- Compute vector field
- **Compute indicator function**
  - Compute divergence
  - **Solve Poisson Equation**
- Extract iso-surface

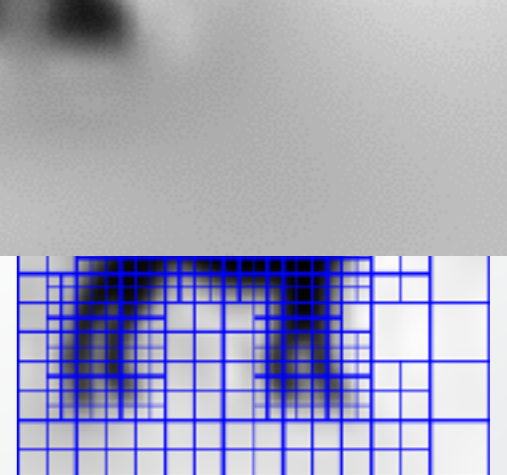
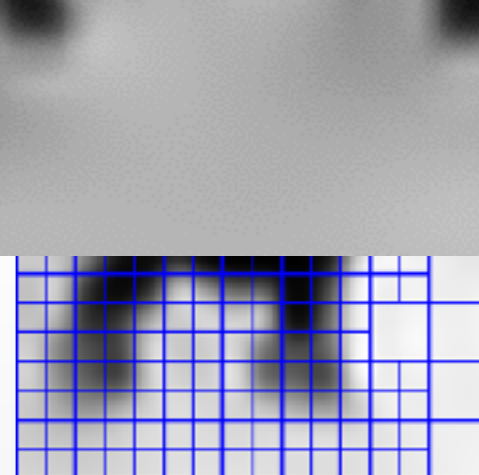
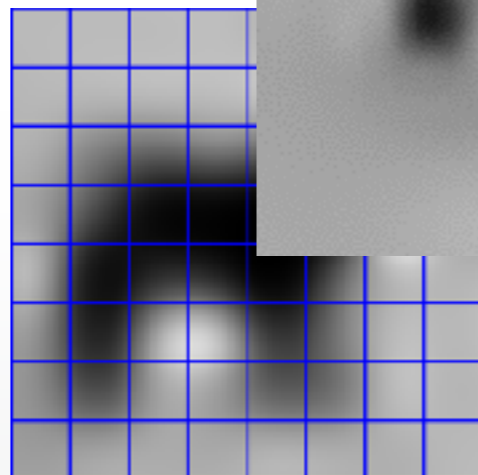
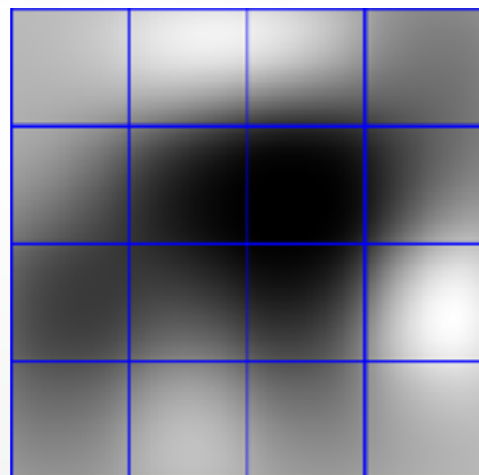
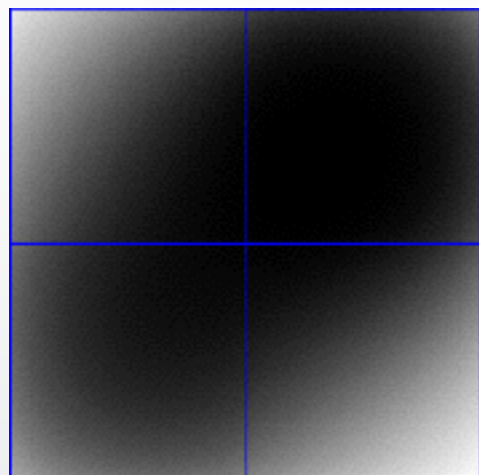




# Implementation: Indicator Function

## Given the Points:

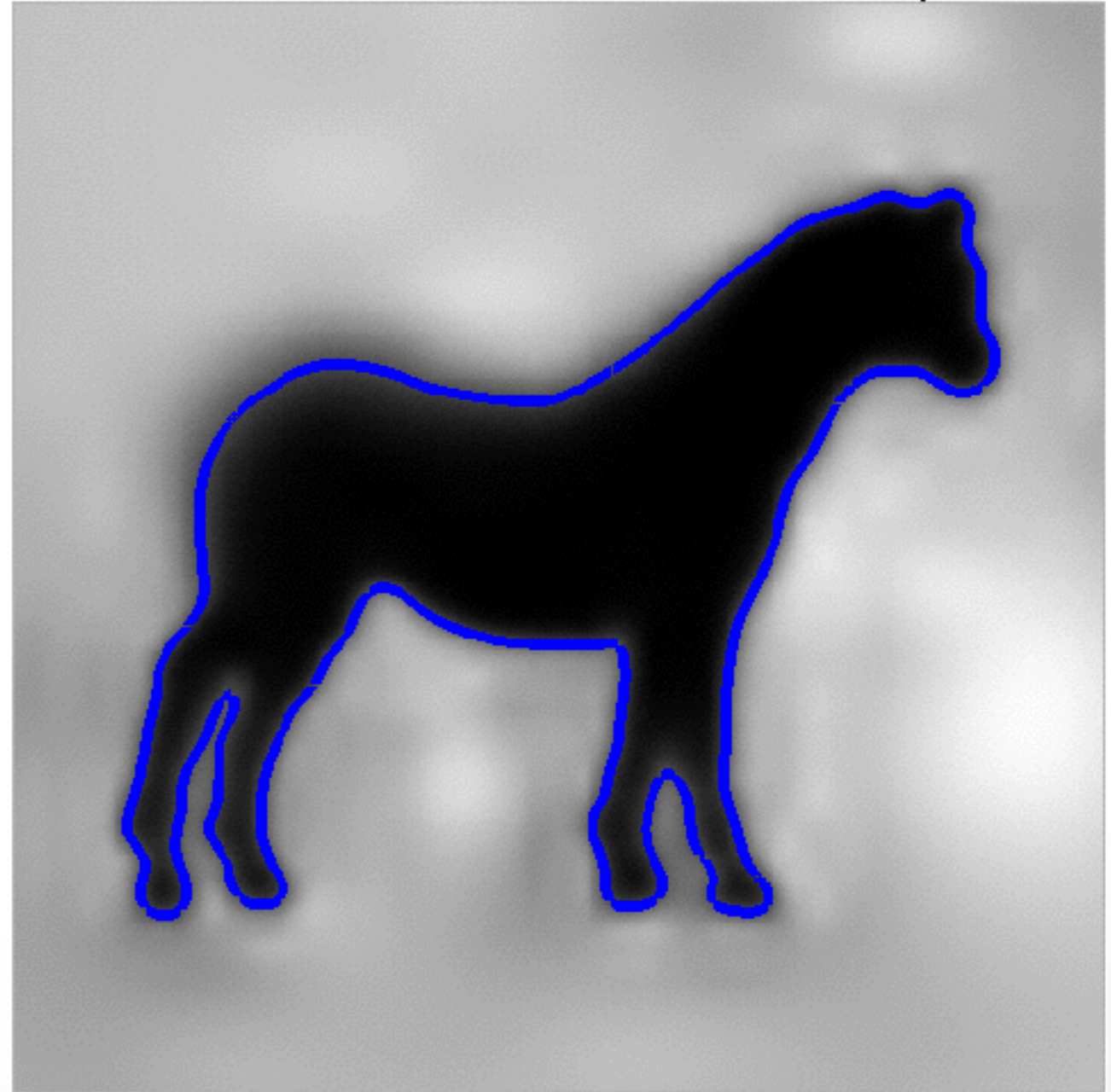
- Set Octree
- Compute vector field
- **Compute indicator function**
  - Compute divergence
  - **Solve Poisson Equation**
- Extract iso-surface



# Implementation: Iso-Surface

## Given the Points:

- Set Octree
- Compute vector field
- Compute indicator function
- **Extract iso-surface**

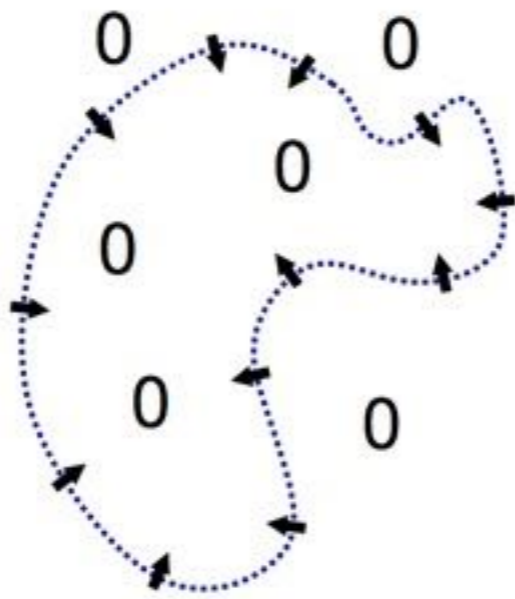


# Summary



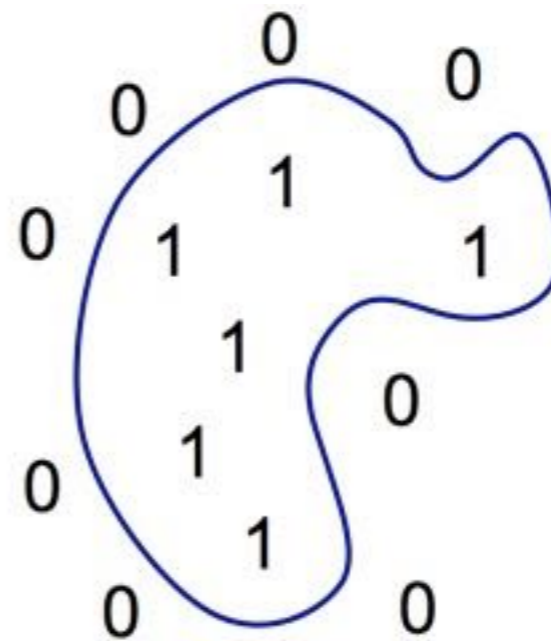
Oriented points

$$\vec{V}$$



Indicator gradient

$$\nabla \chi_M$$



Indicator function

$$\chi_M$$



Surface

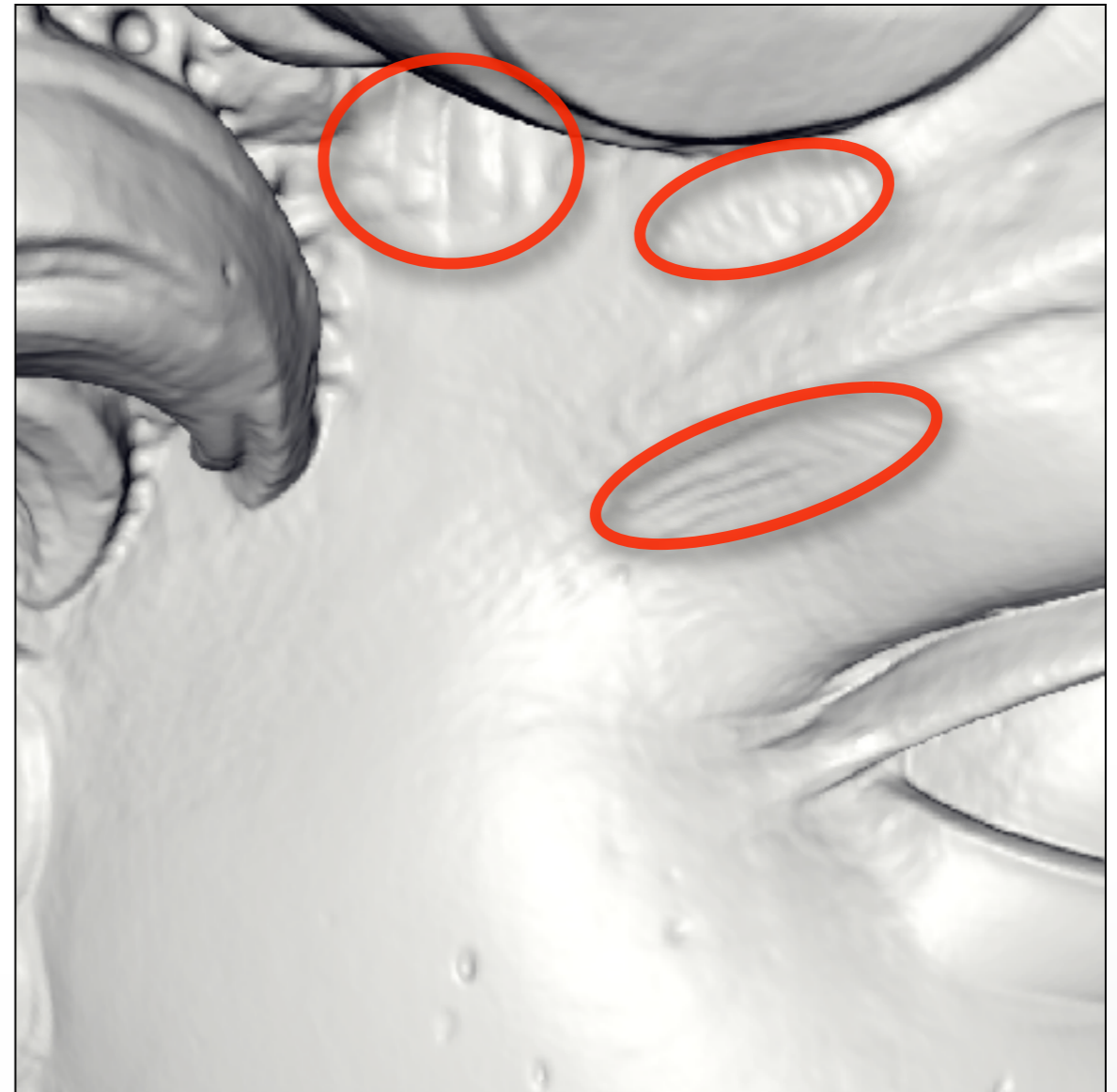
$$\partial M$$

# Michelangelo's David

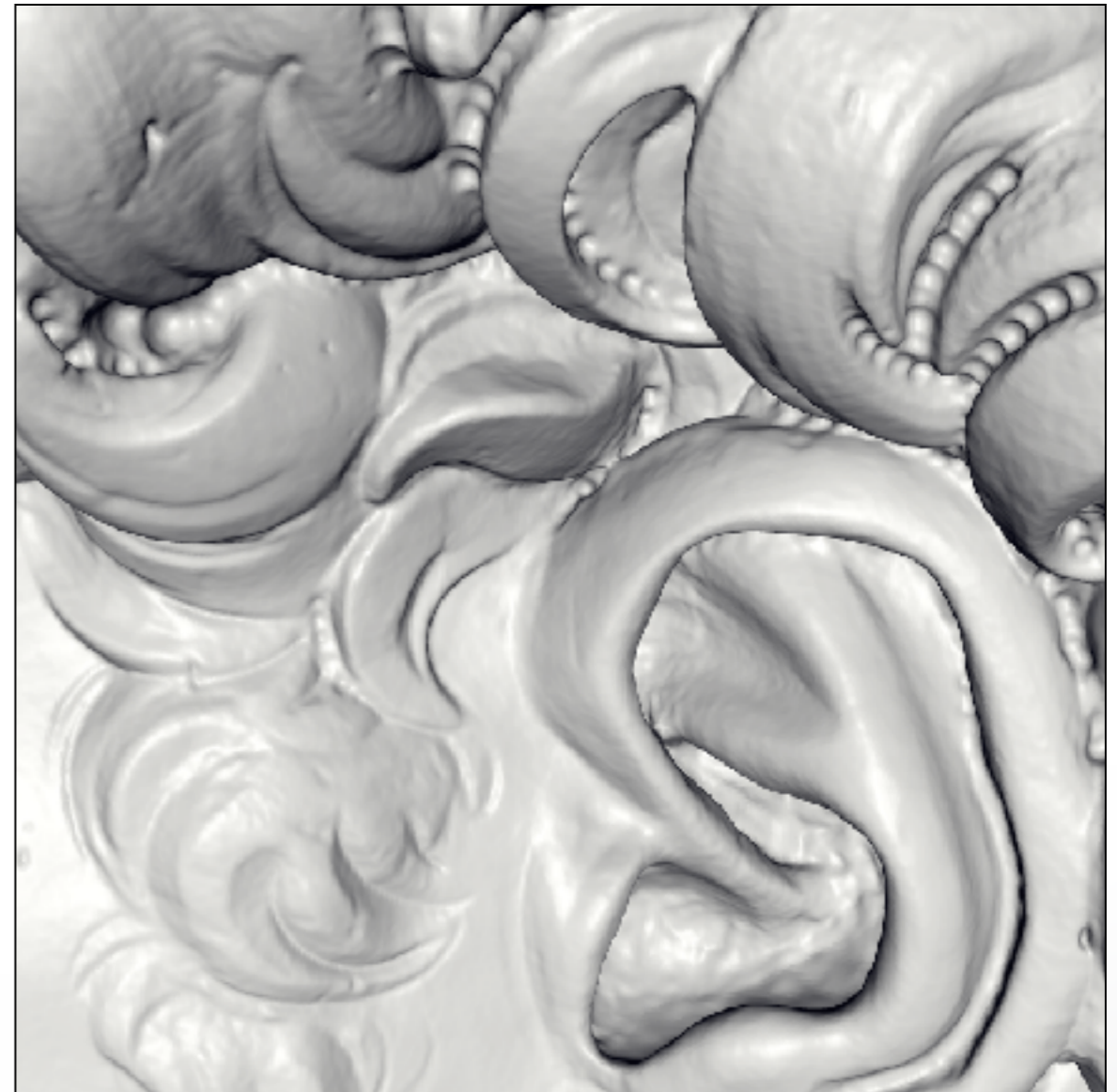


- 215 million data points from 1000 scans
- 22 million triangle reconstruction
- Compute Time: 2.1 hours
- Peak Memory: 6600MB

# David – Chisel marks



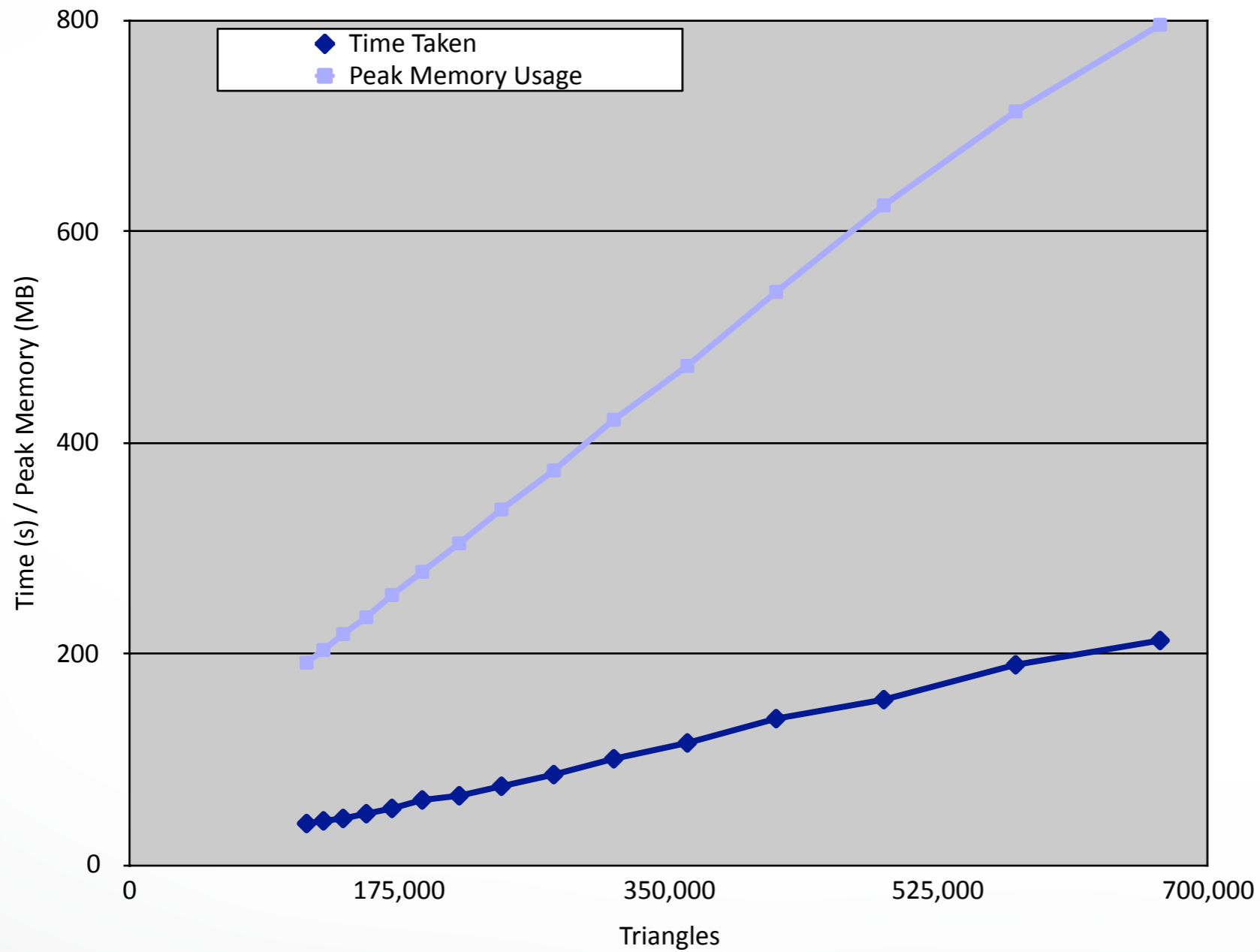
# David – Drill marks



# David – Drill marks



# Scalability – Buddha Model





# Stanford Bunny



**Power Crust**



**FastRBF**



**MPU**



**VRIP**

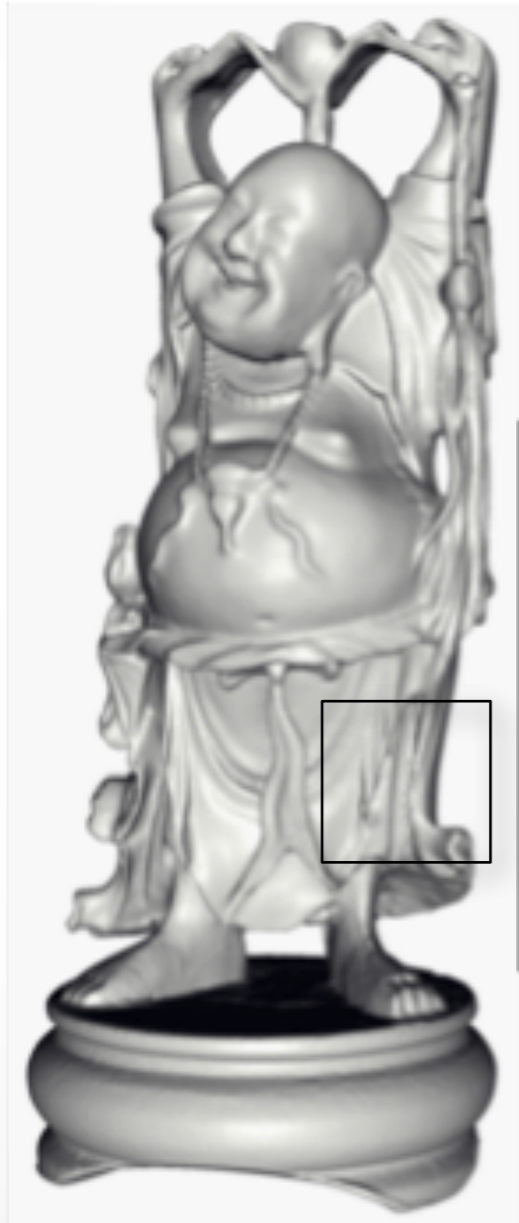


**FFT Reconstruction**

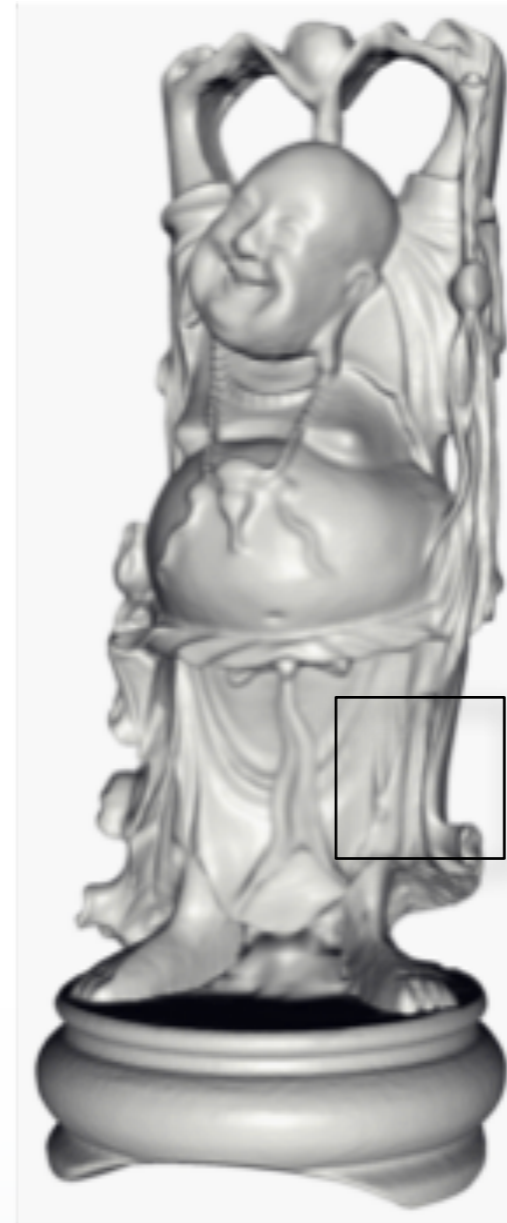
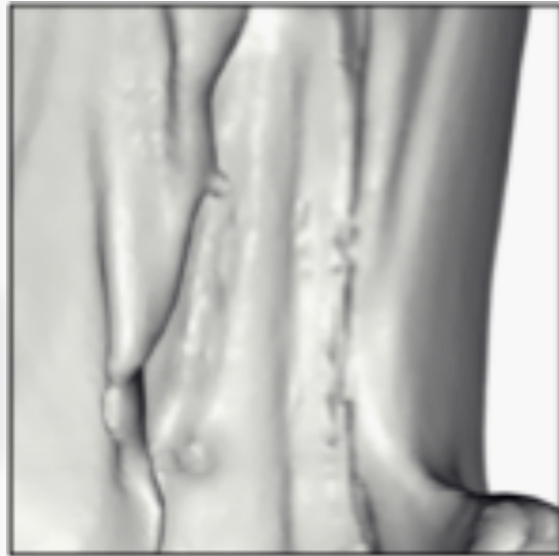


**Possion Reconstruction**

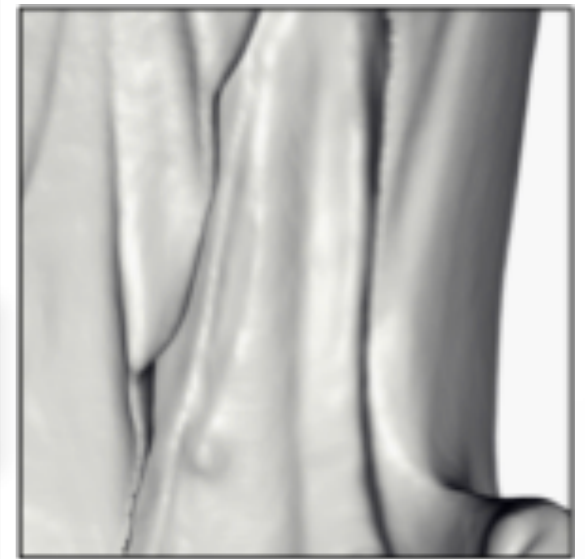
# VRIP Comparison



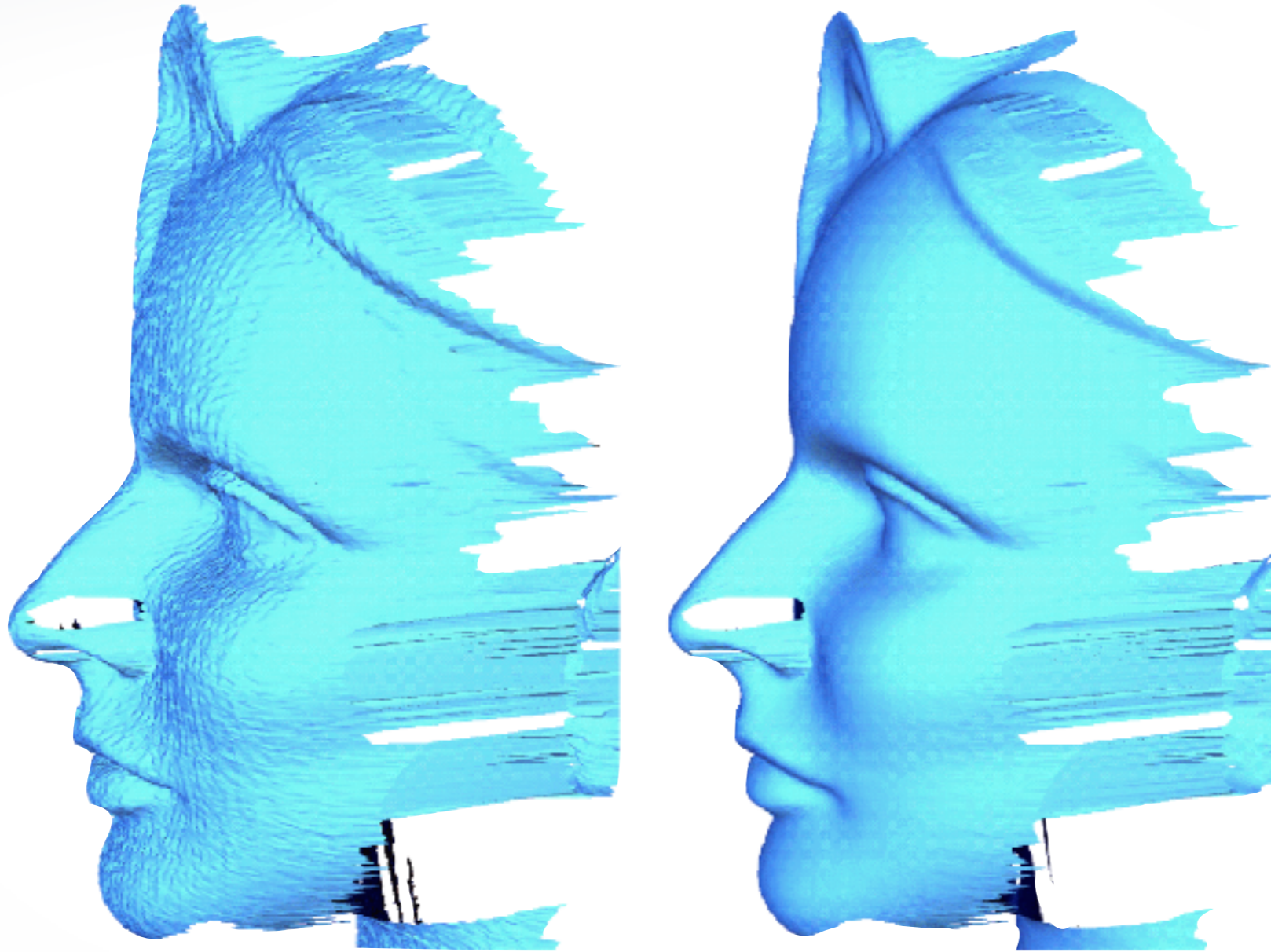
**VRIP**



**Poisson Reconstruction**



# Next Time



**Surface Smoothing**

<http://cs621.hao-li.com>

# Thanks!

