252-0538-00L, Spring 2025

Shape Modeling and Geometry Processing

Surface Reconstruction

INTERACTIVE GEOMETRY LAB

March 5, 2025

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Geometry Acquisition Pipeline

Scanning
results in range images

⇨

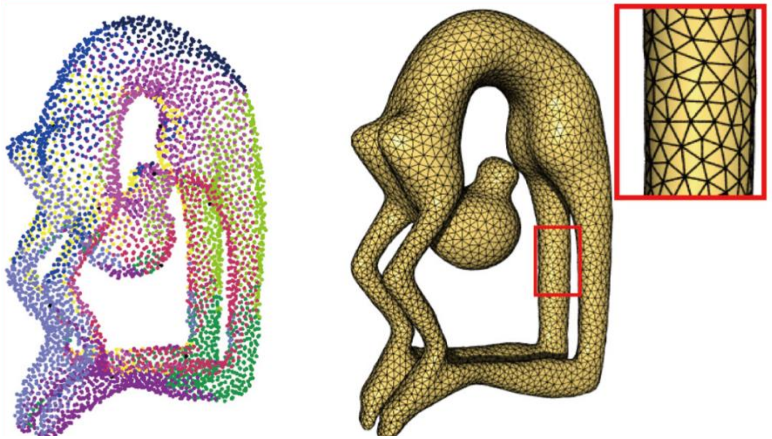
Registration
bring all range images to one coordinate system

⇨

Stitching/reconstruction
Integration of scans into a single mesh

⇨

Postprocess
Topological filtering
Geometric filtering
Remeshing
Compression



Digital Michelangelo Project

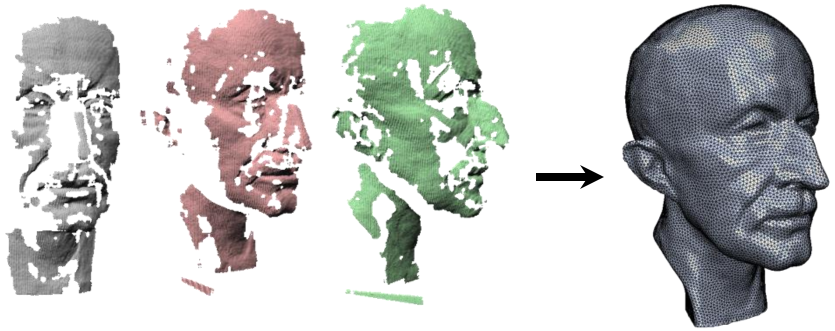
1G sample points → 8M triangles

4G sample points → 8M triangles

Input to Reconstruction Process

- **Input option 1**
  Just a set of 3D points, irregularly spaced
  - Need to estimate normals
    ➔ next class

- **Input option 2**
  Normals come from the range scans

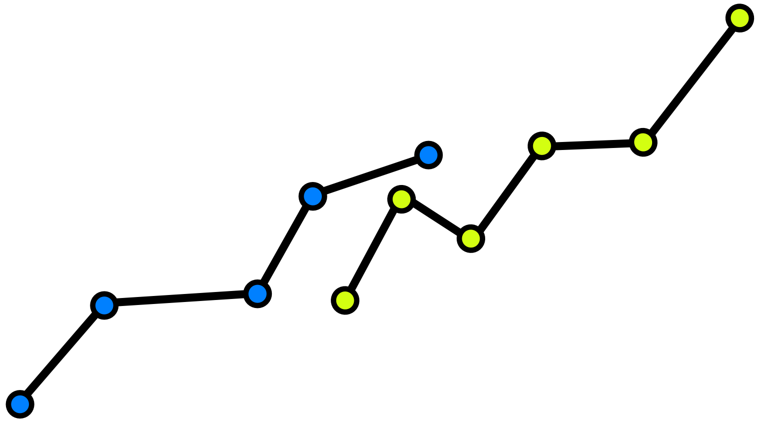


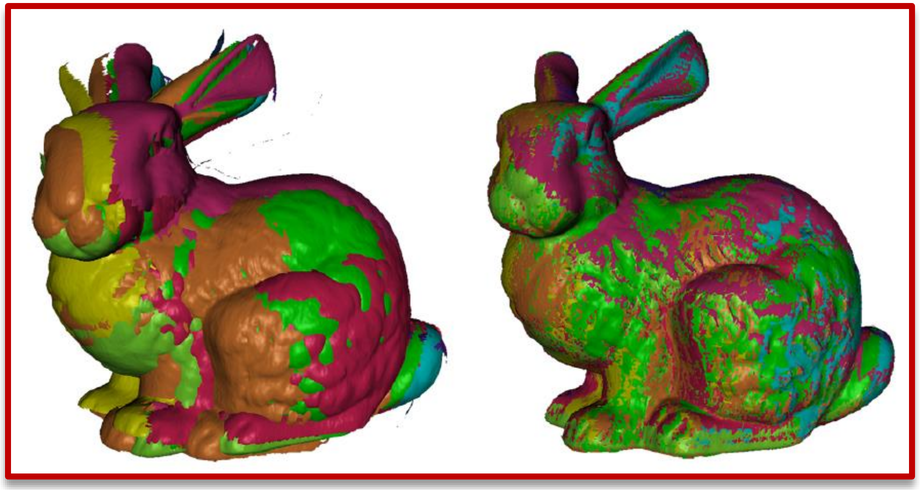
set of raw scans

reconstructed model

How to Connect the Dots?

- **Explicit reconstruction:**
  stitch the range scans together

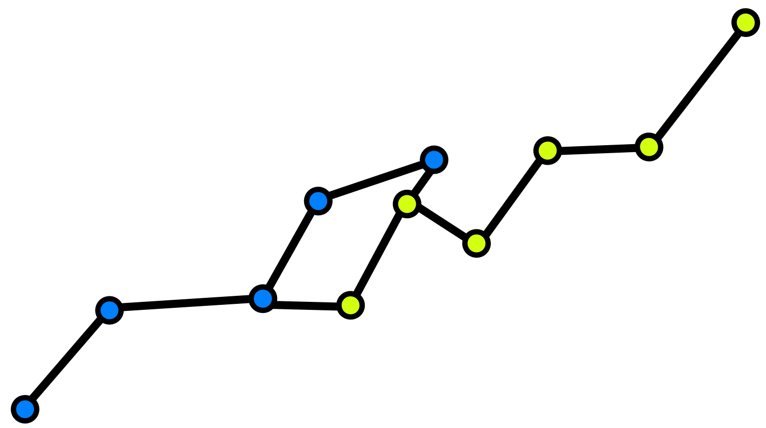




"Zippered Polygon Meshes from Range Images", Greg Turk and Marc Levoy, ACM SIGGRAPH 1994

How to Connect the Dots?

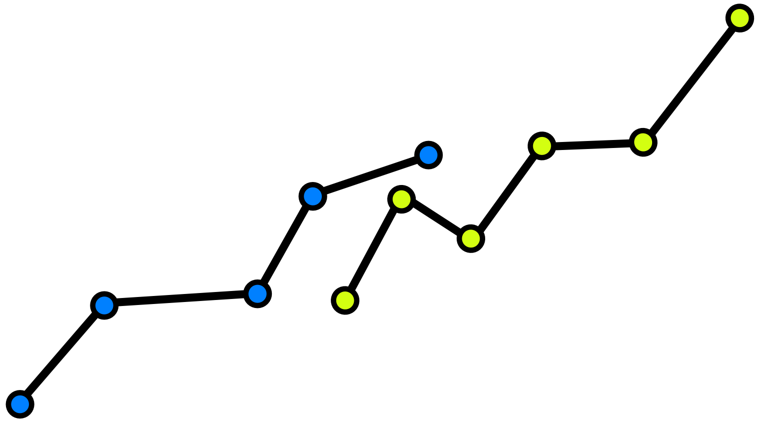
- **Explicit reconstruction:**
  stitch the range scans together



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

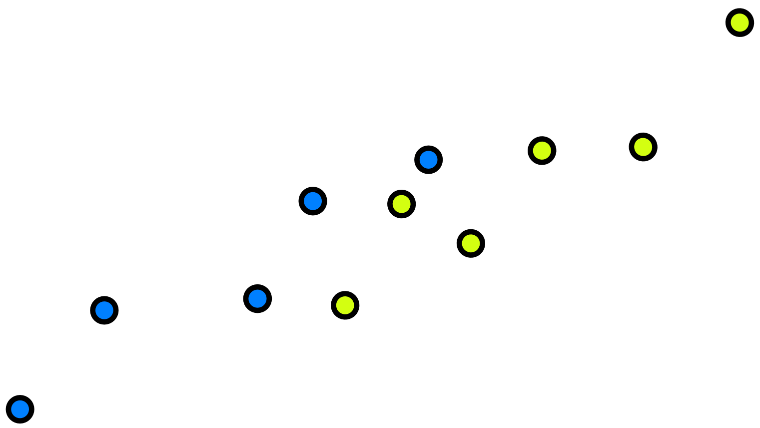
How to Connect the Dots?

- **Implicit reconstruction**: estimate a signed distance function (SDF); extract 0-level set mesh using Marching Cubes



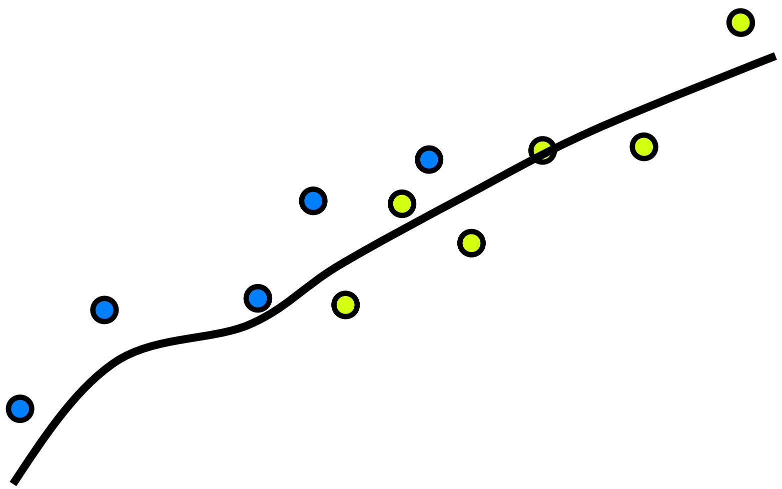
How to Connect the Dots?

- **Implicit reconstruction**: estimate a signed distance function (SDF); extract 0-level set mesh using Marching Cubes



How to Connect the Dots?

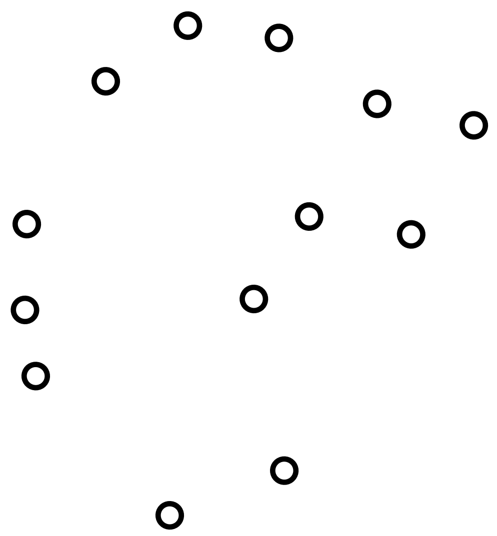
- **Implicit reconstruction**: estimate a signed distance function (SDF); extract 0-level set mesh using Marching Cubes



  - Approximation of input points
  - Watertight manifold results by construction

How to Connect the Dots?

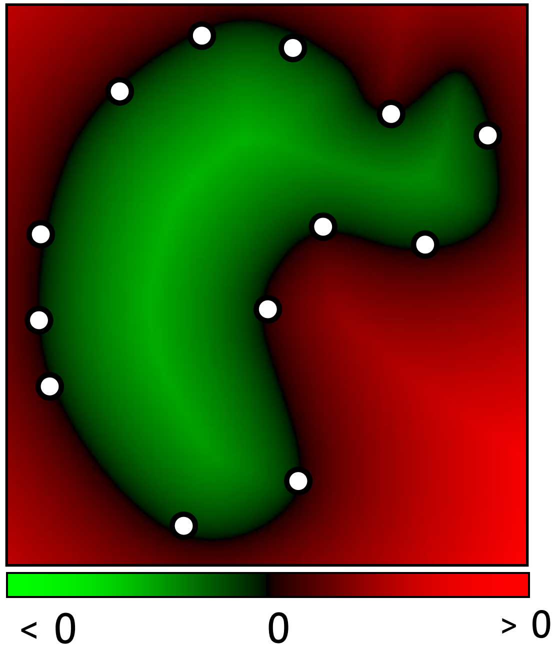
- **Implicit reconstruction**: estimate a signed distance function (SDF); extract 0-level set mesh using Marching Cubes



- Approximation of input points
- Watertight manifold results by construction

How to Connect the Dots?

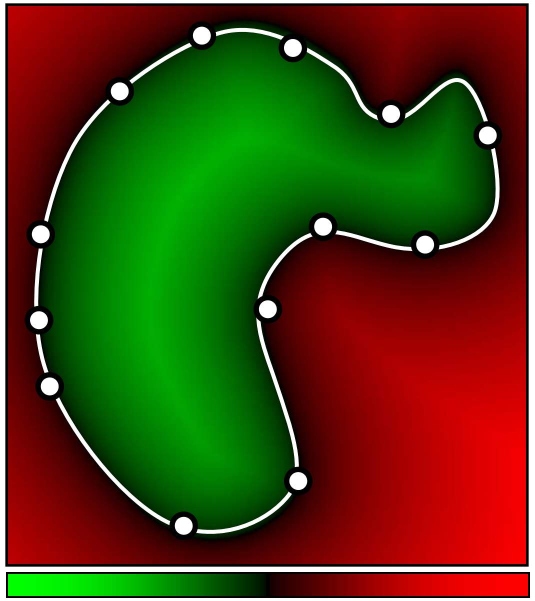
- **Implicit reconstruction**: estimate a signed distance function (SDF); extract 0-level set mesh using Marching Cubes



- Approximation of input points
- Watertight manifold results by construction

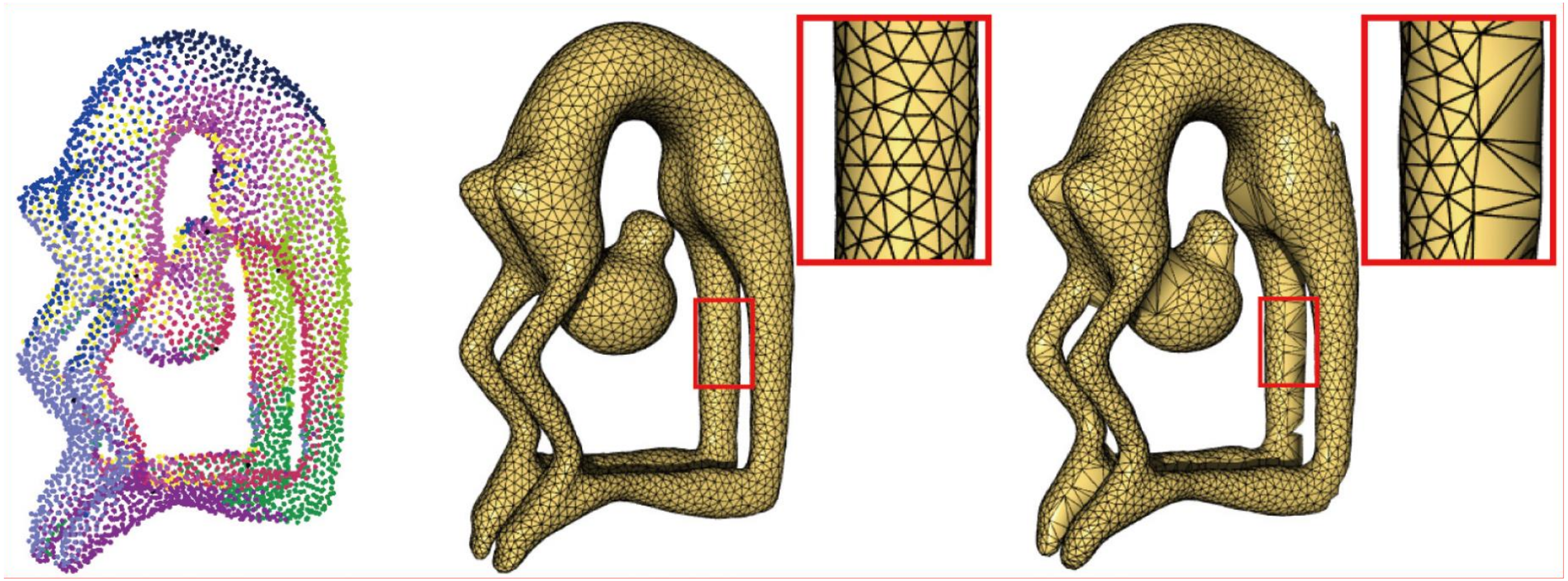
How to Connect the Dots?

- **Implicit reconstruction**: estimate a signed distance function (SDF); extract 0-level set mesh using Marching Cubes



- Approximation of input points
- Watertight manifold results by construction

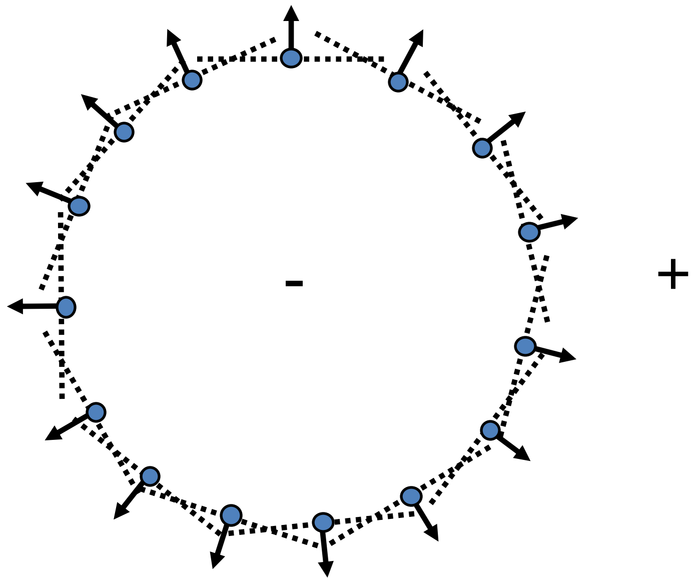
Implicit vs. Explicit



Input Implicit Explicit

SDF from Points and Normals

- Compute signed distance to the tangent plane of the closest point
- Normals help to distinguish between inside and outside

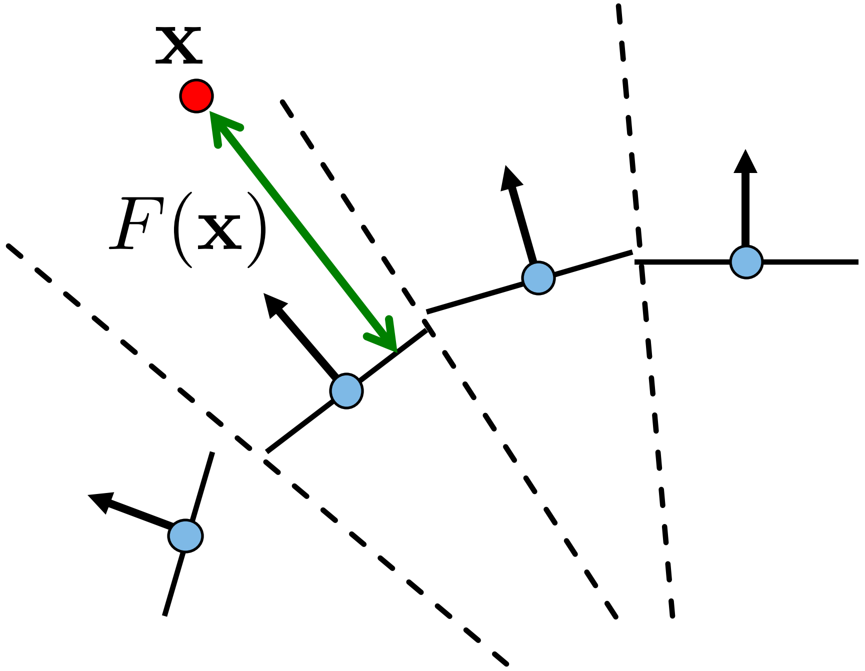


“Surface reconstruction from unorganized points”, Hoppe et al., ACM SIGGRAPH 1992
http://research.microsoft.com/en-us/um/people/hoppe/proj/recon/

SDF from Points and Normals

- Compute signed distance to the tangent plane of the closest point

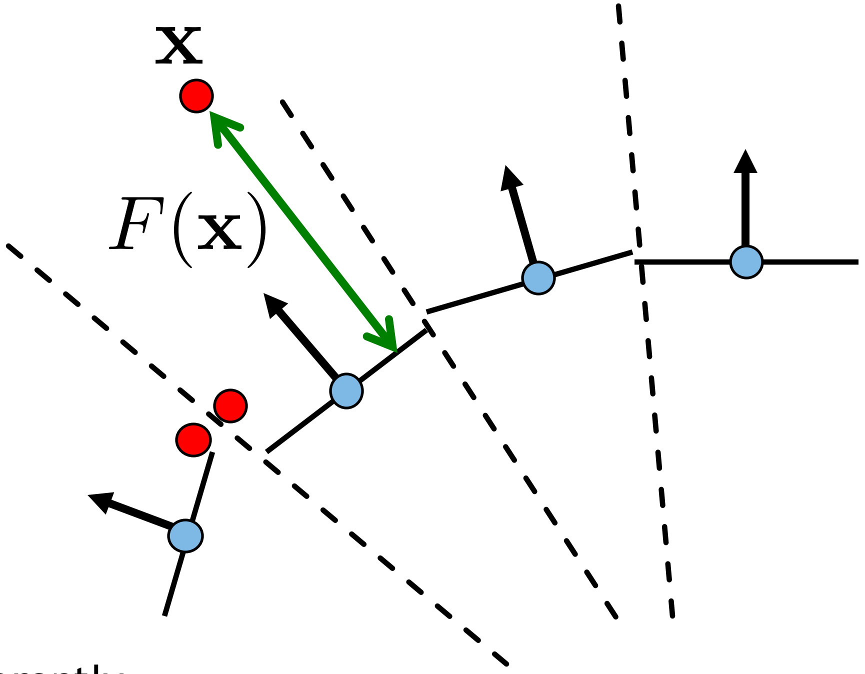
- Problem??



SDF from Points and Normals

- Compute signed distance to the tangent plane* of the closest point

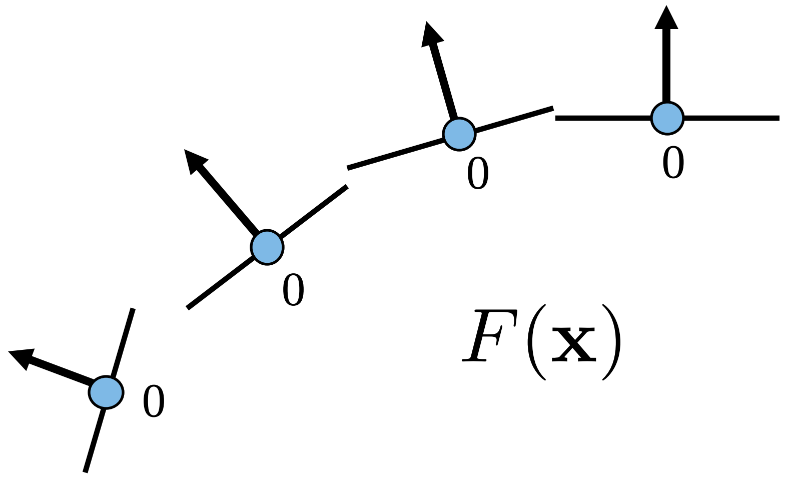
- The function will be discontinuous



* The Hoppe92 paper computes the tangent planes slightly differently (by PCA on k-nearest-neighbors of each data point, see next class), but the consequences are still the same.

Smooth SDF

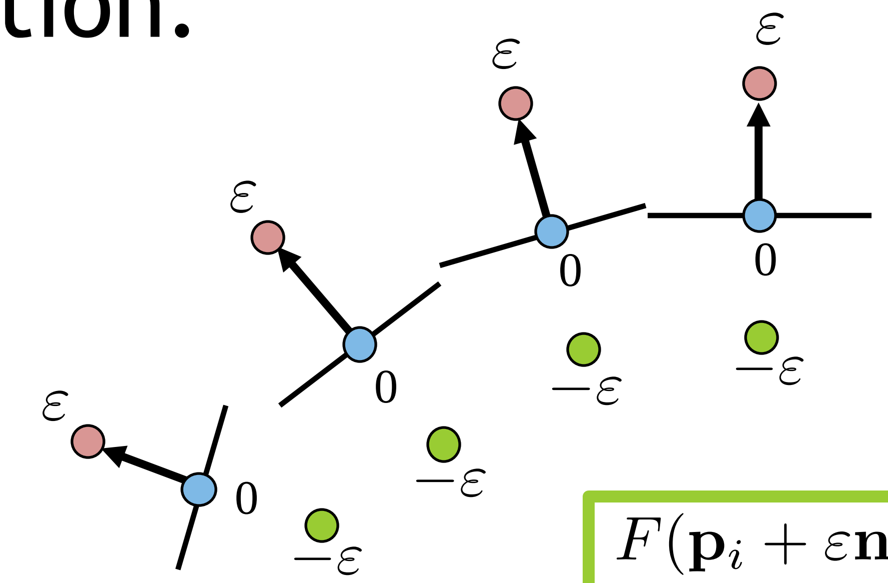
- Instead find a smooth formulation for F.
- Scattered data interpolation:
  - $F(\mathbf{p}_i) = 0$
  - F is smooth
  - Avoid trivial $F \equiv 0$



"Reconstruction and representation of 3D objects with radial basis functions", Carr et al., ACM SIGGRAPH 2001

Smooth SDF

- Instead find a smooth formulation for F.
- Scattered data interpolation:
  - $F(\mathbf{p}_i) = 0$
  - F is smooth
  - Avoid trivial $F \equiv 0$
- Add off-surface constraints



$$F(\mathbf{p}_i + \varepsilon \mathbf{n}_i) = \varepsilon$$
$$F(\mathbf{p}_i - \varepsilon \mathbf{n}_i) = -\varepsilon$$

Radial Basis Function Interpolation

- **RBF**: Weighted sum of shifted, smooth kernels

$$F(\mathbf{x}) = \sum_{m=0}^{N-1} w_m \, \varphi(\|\mathbf{x} - \mathbf{c}_m\|) + p(\mathbf{x})$$

Scalar coefficients
Unknowns

$$N = 3n$$

Smooth kernels
(basis functions)
centered at constrained points.
For example:

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

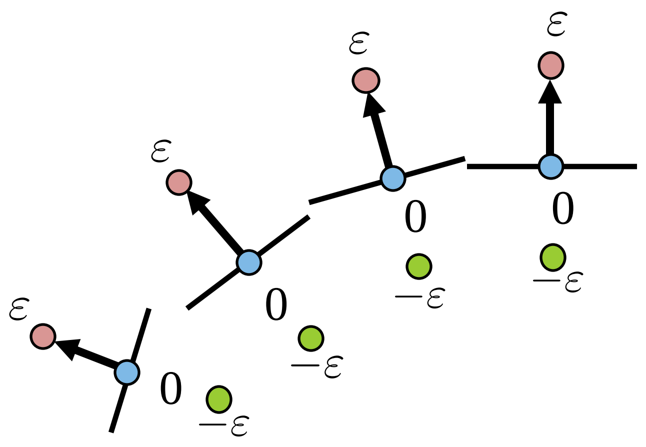
Linear polynomial with
unknown coeffs

$$p(\mathbf{x}) = a_0 + a_1 x + a_2 y + a_3 z$$

Radial Basis Function Interpolation

- Interpolate the constraints:

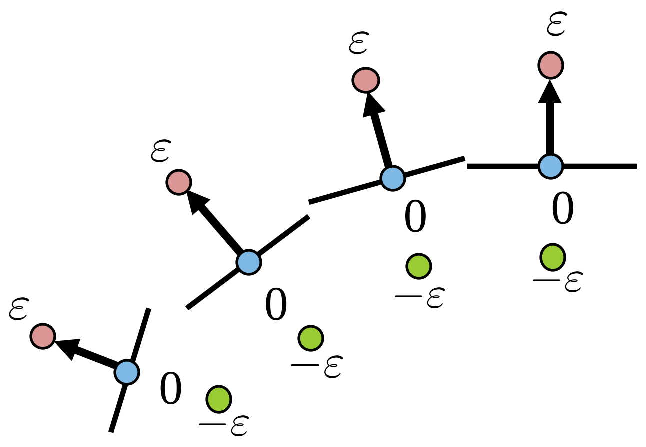
$$F(\mathbf{p}_i) = 0$$
$$F(\mathbf{p}_i + \varepsilon \mathbf{n}_i) = \varepsilon$$
$$F(\mathbf{p}_i - \varepsilon \mathbf{n}_i) = -\varepsilon$$



Radial Basis Function Interpolation

- Interpolate the constraints:

$$F(\mathbf{p}_i) = 0$$
$$F(\mathbf{p}_i + \varepsilon\mathbf{n}_i) = \varepsilon$$
$$F(\mathbf{p}_i - \varepsilon\mathbf{n}_i) = -\varepsilon$$



$$\{\mathbf{c}_{3i}, \mathbf{c}_{3i+1}, \mathbf{c}_{3i+2}\} := \{\mathbf{p}_i,\ \mathbf{p}_i + \varepsilon\mathbf{n}_i,\ \mathbf{p}_i - \varepsilon\mathbf{n}_i\}$$

$$d_m = \begin{cases} 0 & m = 3i \\ \varepsilon & m = 3i+1 \\ -\varepsilon & m = 3i+2 \end{cases}$$

$$\Rightarrow \quad F(\mathbf{c}_m) = d_m, \quad m = 0, \ldots, N-1$$

Radial Basis Function Interpolation

- Symmetric linear system to get the coeffs w_* and a_*:

$$A = \begin{pmatrix} \varphi(\|\mathbf{c}_0 - \mathbf{c}_0\|) & \dots & \varphi(\|\mathbf{c}_0 - \mathbf{c}_{N-1}\|) \\ \vdots & \ddots & \vdots \\ \vdots & & \vdots \\ \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_0\|) & \dots & \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_{N-1}\|) \end{pmatrix};$$

$$P \in \mathbb{R}^{N\times 4}, \text{ row } m \text{ of } P = (1,\ c_{x,m},\ c_{y,m},\ c_{z,m})$$

$$\begin{pmatrix} A & P \\ P^{\mathsf{T}} & 0 \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ \vdots \\ w_{N-1} \\ a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} d_0 \\ \vdots \\ \vdots \\ d_{N-1} \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- System size (N+4) x (N+4)
- Dense or sparse, depending on the kernel

Radial Basis Function Interpolation

$$A = \begin{pmatrix} \varphi(\|\mathbf{c}_0 - \mathbf{c}_0\|) & \dots & \varphi(\|\mathbf{c}_0 - \mathbf{c}_{N-1}\|) \\ \vdots & \ddots & \vdots \\ \vdots & & \vdots \\ \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_0\|) & \dots & \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_{N-1}\|) \end{pmatrix};$$

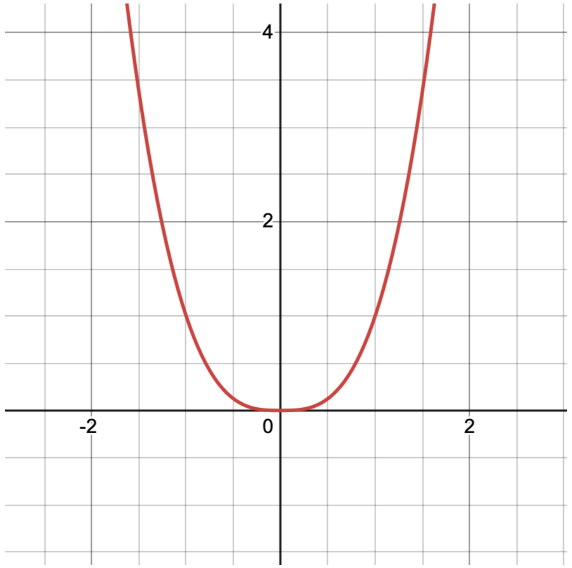
$$P \in \mathbb{R}^{N\times 4}, \text{ row } m \text{ of } P = (1,\ c_{x,m},\ c_{y,m},\ c_{z,m})$$

$$\begin{pmatrix} A & P \\ P^\mathsf{T} & 0 \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ \vdots \\ w_{N-1} \\ a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} d_0 \\ \vdots \\ \vdots \\ d_{N-1} \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$F(\mathbf{x}) = \sum_{m=0}^{N-1} w_m\, \varphi(\|\mathbf{x} - \mathbf{c}_m\|) + p(\mathbf{x})$$

RBF Kernels

- Triharmonic: $\varphi(r) = r^3$
  - Globally supported
  - Leads to dense symmetric linear system
  - C^2 smoothness
  - Works well for highly irregular sampling



RBF Kernels

- Polyharmonic
  - $\varphi(r) = r^k \log(r),\ \ k = 2, 4, 6 \ldots$
  - $\varphi(r) = r^k,\ \ k = 1, 3, 5 \ldots$
- Multiquadratic

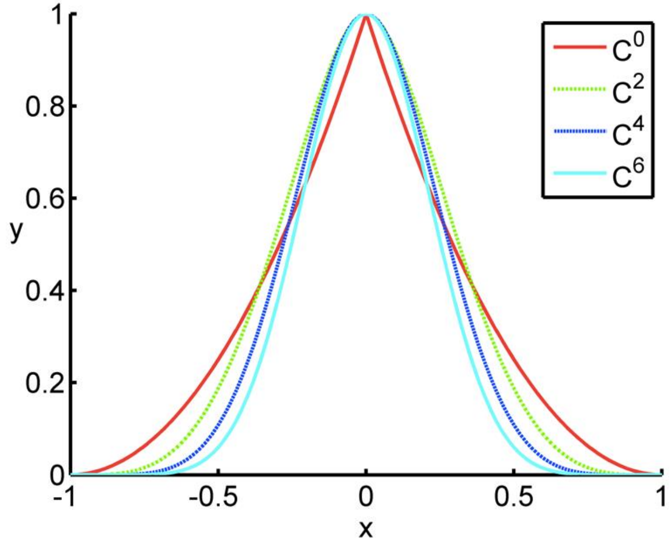
$$\varphi(r) = \sqrt{r^2 + \beta^2}$$

- Gaussian

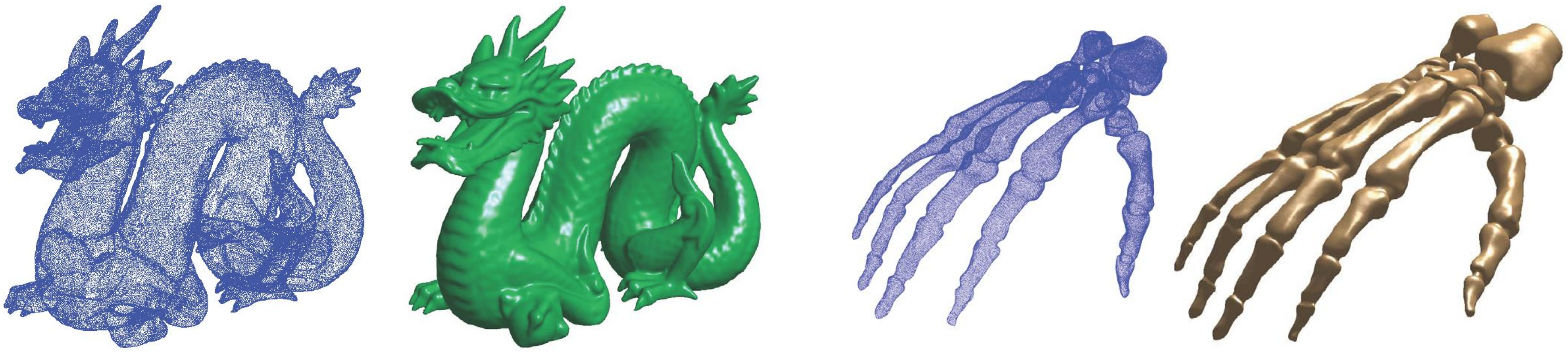
$$\varphi(r) = e^{-\beta r^2}$$

- B-Spline (compact support)

$$\varphi(r) = \text{piecewise-polynomial}(r)$$

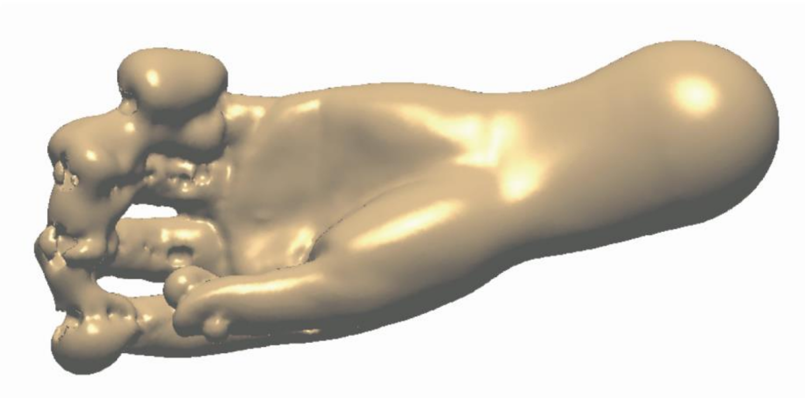


RBF Reconstruction Examples

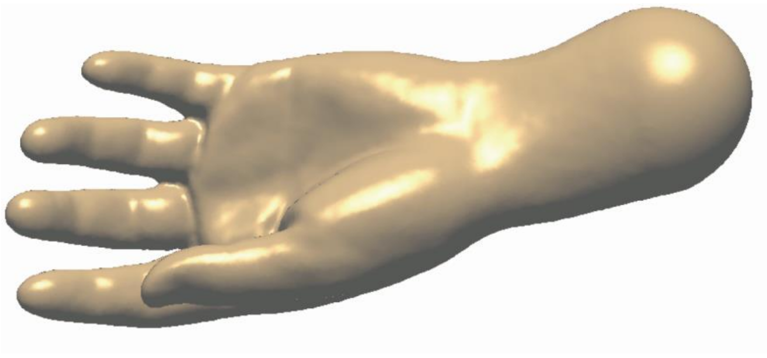


“Reconstruction and representation of 3D objects with radial basis functions”, Carr et al., ACM SIGGRAPH 2001

Off-Surface Points



Insufficient number/
badly placed off-surface points



Properly chosen off-surface points

“Reconstruction and representation of 3D objects with radial basis functions”, Carr et al., ACM SIGGRAPH 2001

Comparison of the various SDFs so far

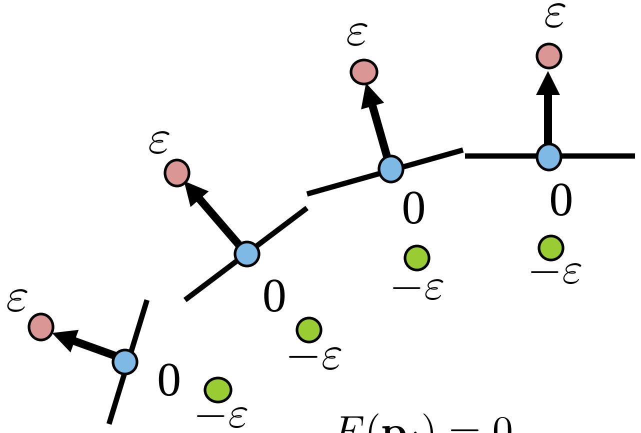
Distance to plane | Compact RBF | Global RBF Triharmonic

RBF – Discussion

- Global definition!

$$F(\mathbf{x}) = \sum_{m=0}^{N-1} w_m \, \varphi(\|\mathbf{x} - \mathbf{c}_m\|) + p(\mathbf{x})$$

- Global optimization of the weights, even if the basis functions are local

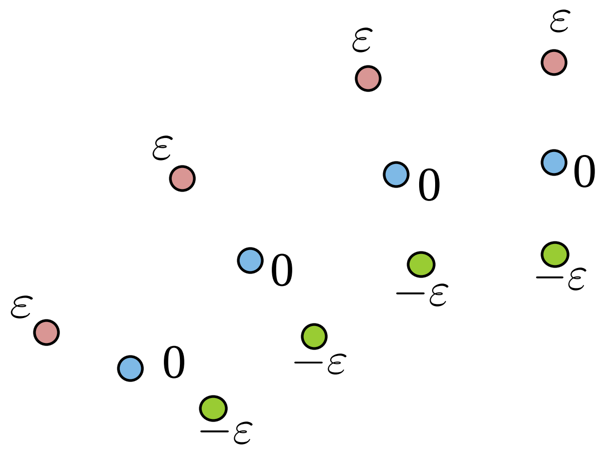


$$F(\mathbf{p}_i) = 0$$
$$F(\mathbf{p}_i + \varepsilon \mathbf{n}_i) = \varepsilon$$
$$F(\mathbf{p}_i - \varepsilon \mathbf{n}_i) = -\varepsilon$$

Moving Least Squares (MLS)

- Do purely **local** approximation of the SDF
- Weights change depending on where we are evaluating

- The beauty: the "stitching" of all local approximations, seen as one function $F(\mathbf{x})$, is smooth everywhere!
  - We get a **globally** smooth function but only do **local** computation

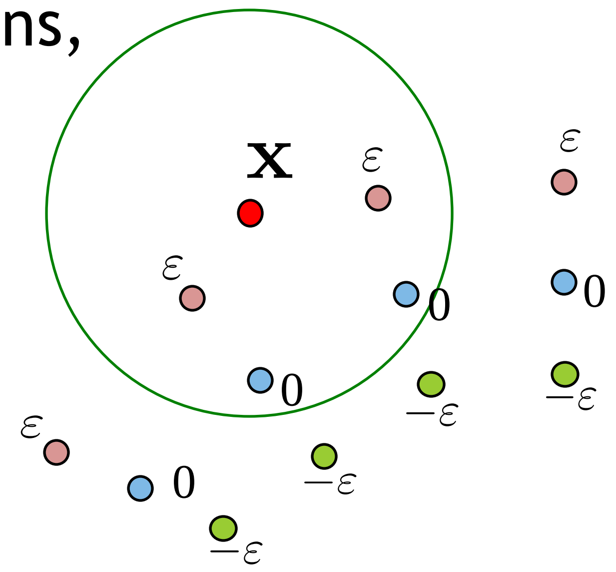


"Interpolating and Approximating Implicit Surfaces from Polygon Soup", Shen et al., ACM SIGGRAPH 2004
http://graphics.berkeley.edu/papers/Shen-IAI-2004-08/index.html

Moving Least Squares (MLS)

- Do purely **local** approximation of the SDF
- Weights change depending on where we are evaluating

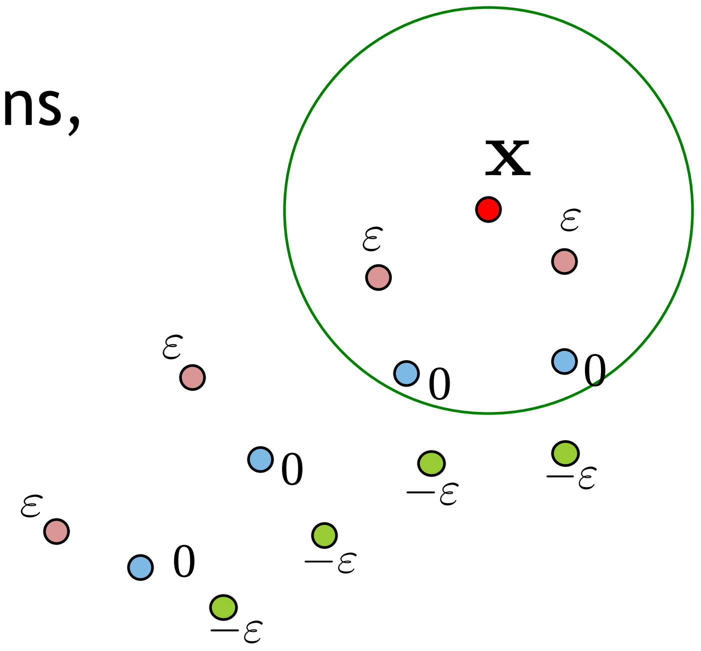
- The beauty: the “stitching” of all local approximations, seen as one function $F(\mathbf{x})$, is smooth everywhere!
  - We get a **globally** smooth function but only do **local** computation



Moving Least Squares (MLS)

- Do purely **local** approximation of the SDF
- Weights change depending on where we are evaluating

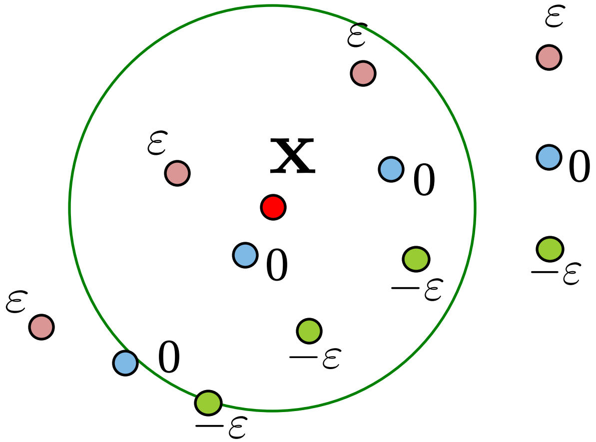
- The beauty: the "stitching" of all local approximations, seen as one function $F(\mathbf{x})$, is smooth everywhere!
  - We get a **globally** smooth function but only do **local** computation



Moving Least Squares (MLS)

- Do purely **local** approximation of the SDF
- Weights change depending on where we are evaluating

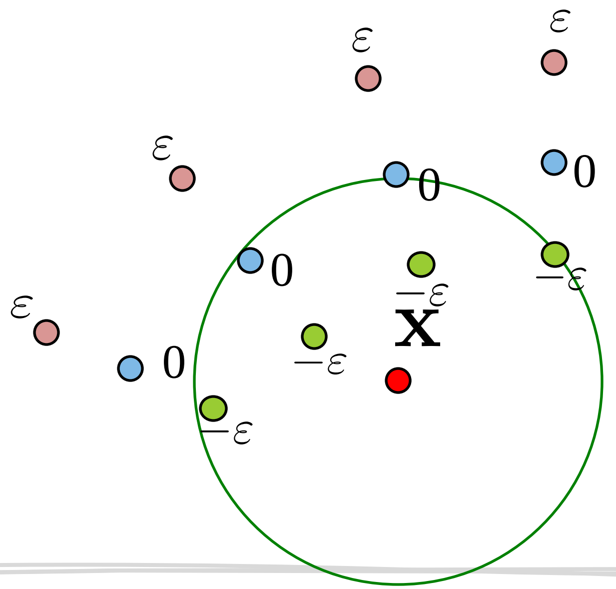
- The beauty: the “stitching” of all local approximations, seen as one function $F(\mathbf{x})$, is smooth everywhere!
  - We get a **globally** smooth function but only do **local** computation



Moving Least Squares (MLS)

- Do purely **local** approximation of the SDF
- Weights change depending on where we are evaluating

- The beauty: the "stitching" of all local approximations, seen as one function $F(\mathbf{x})$, is smooth everywhere!
  - We get a **globally** smooth function but only do **local** computation



Least-Squares Approximation

- Polynomial least-squares approximation
  - Choose degree, k

$$f \in \Pi_k^3 : f(x,y,z) = a_0 + a_1 x + a_2 y + a_3 z + a_4 x^2 + a_5 xy + \ldots + a_* z^k$$

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

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

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

$$\underset{f \in \Pi_k^3}{\operatorname{argmin}} \sum_{m=0}^{N-1} (f(\mathbf{c}_m) - d_m)^2 \quad \text{or:} \quad \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{m=0}^{N-1} \left(\mathbf{b}(\mathbf{c}_m)^\mathsf{T}\mathbf{a} - d_m\right)^2$$

MOVING Least-Squares Approximation

- Polynomial least-squares approximation
  - Choose degree, k

$$f \in \Pi_k^3 : f(x,y,z) = a_0 + a_1 x + a_2 y + a_3 z + a_4 x^2 + a_5 xy + \ldots + a_* z^k$$

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

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

  - Find $\mathbf{a}_\mathbf{x}$ that minimizes WEIGHTED sum of squared differences

$$f_\mathbf{x} = \underset{f \in \Pi_k^3}{\operatorname{argmin}} \sum_{m=0}^{N-1} \theta(\|\mathbf{x} - \mathbf{c}_m\|) \left(f(\mathbf{c}_m) - d_m\right)^2 \quad \text{or:} \quad \mathbf{a}_\mathbf{x} = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{m=0}^{N-1} \theta(\|\mathbf{x} - \mathbf{c}_m\|) \left(\mathbf{b}(\mathbf{c}_m)^\mathsf{T}\mathbf{a} - d_m\right)^2$$

MOVING Least-Squares Approximation

- Polynomial least-squares approximation
  - Choose degree, k

$$f \in \Pi_k^3 : f(x,y,z) = a_0 + a_1 x + a_2 y + a_3 z + a_4 x^2 + a_5 xy + \ldots + a_* z^k$$

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

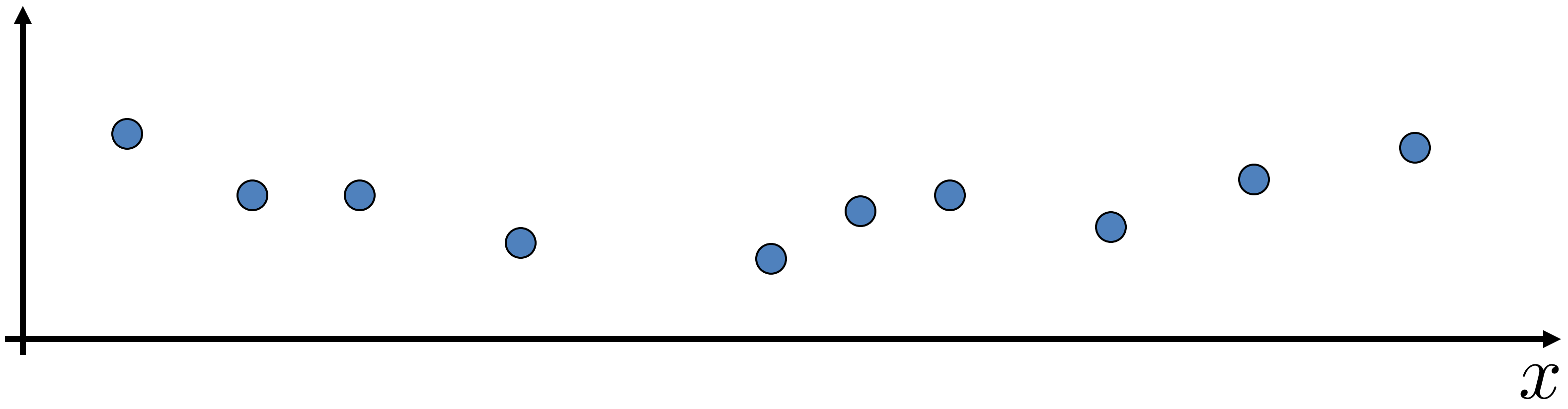
$$\mathbf{a} = (a_1, a_2, \ldots, a_*)^\mathsf{T}, \quad \mathbf{b}(\mathbf{x})^\mathsf{T} = (1, x, y, z, x^2, xy, \ldots, z^k)$$

  - Find $\mathbf{a_x}$ that minimizes WEIGHTED sum of squared differences
  - The value of the SDF is the obtained approximation evaluated at $\mathbf{x}$:

$$F(\mathbf{x}) = f_{\mathbf{x}}(\mathbf{x}) = \mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{a}_{\mathbf{x}}$$

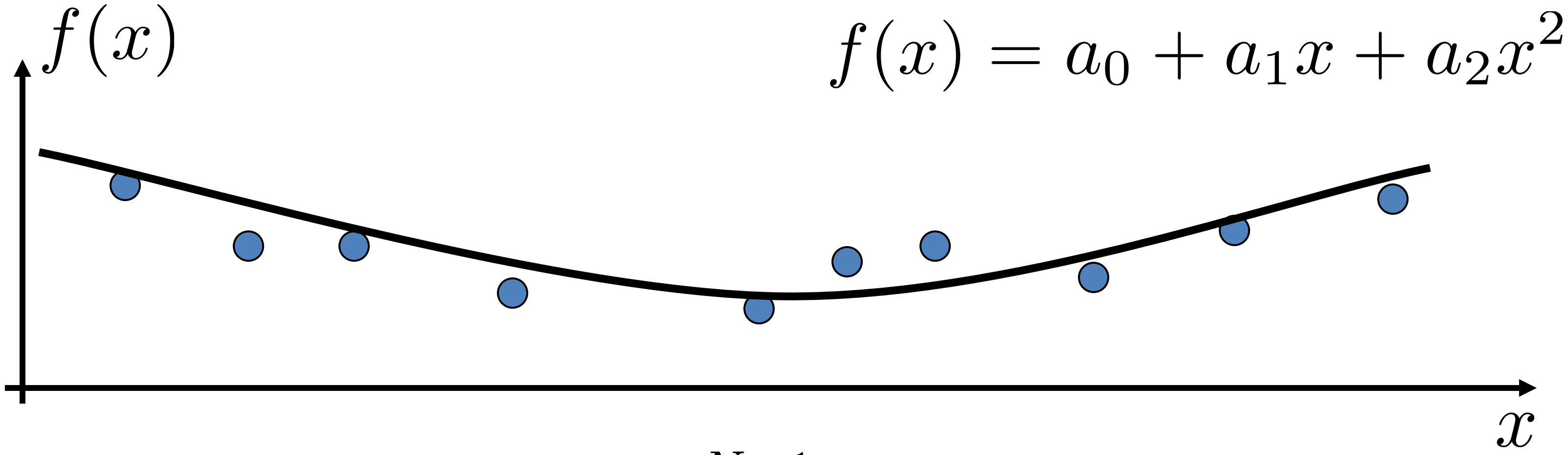
MLS – 1D Example

- Input values



MLS – 1D Example

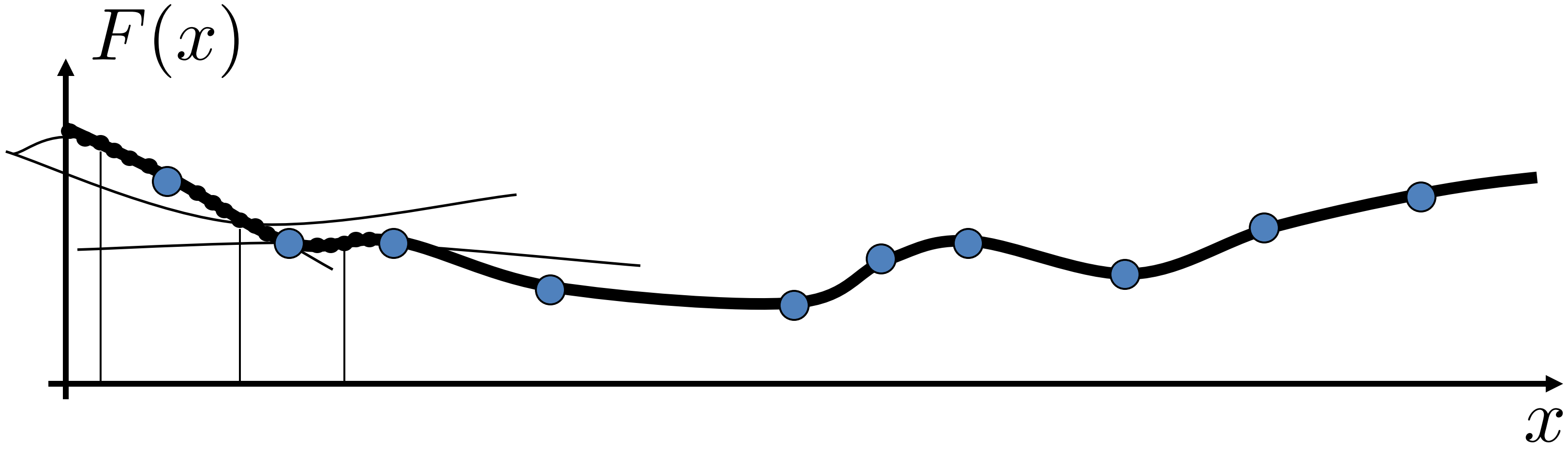
- Global approximation in Π_2^1



$$f = \underset{f \in \Pi_2^1}{\operatorname{argmin}} \sum_{m=0}^{N-1} \left(f(c_m) - d_m\right)^2$$

MLS – 1D Example

- MLS approximation using functions in Π_2^1

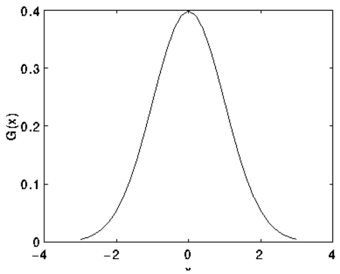


$$F(x) = f_x(x), \quad f_x = \underset{f \in \Pi_2^1}{\text{argmin}} \sum_{m=0}^{N-1} \theta(\|c_m - x\|) \left(f(c_m) - d_m\right)^2$$

Weight Functions

- Gaussian
  - h is a smoothing parameter

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



- Wendland function
  - Defined in $[0, h]$ and

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

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

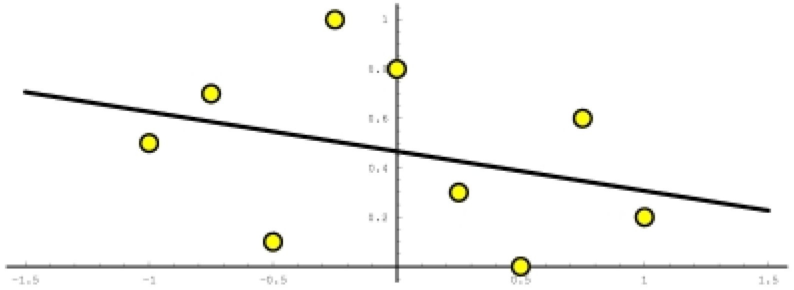
- “Singular” function

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

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

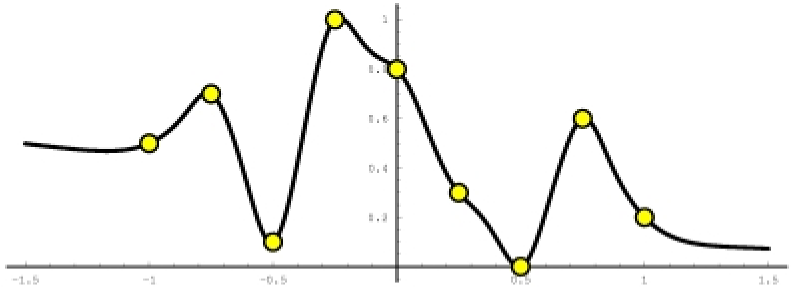
Dependence on Weight Function

- Global least squares with linear polynomials



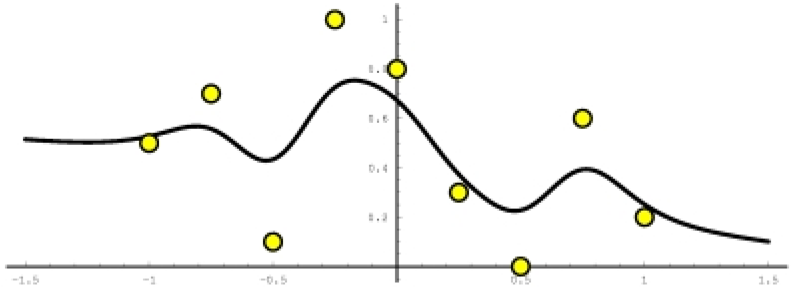
- MLS with (nearly) singular weight function

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



- MLS with approximating weight function

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



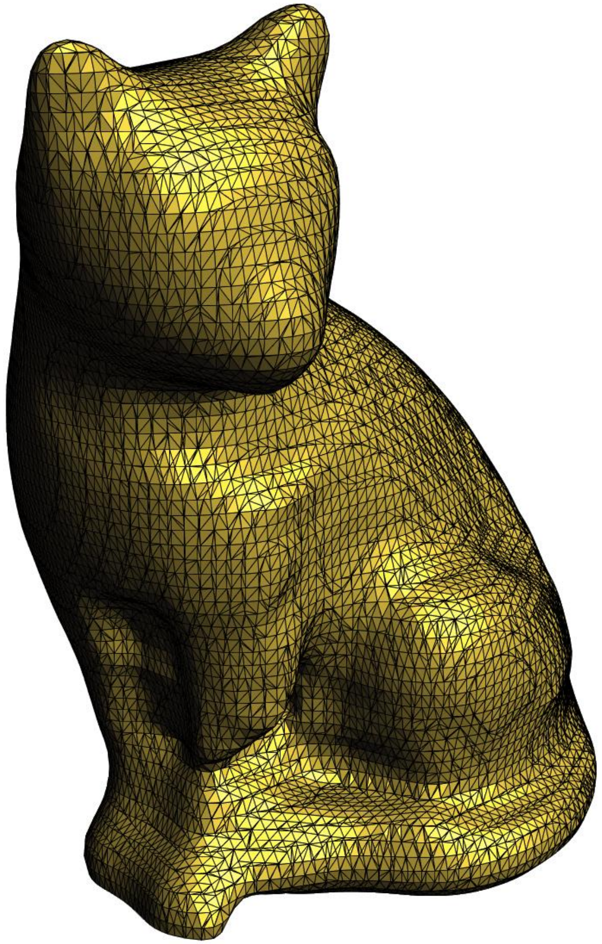
Dependence on Weight Function

- 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

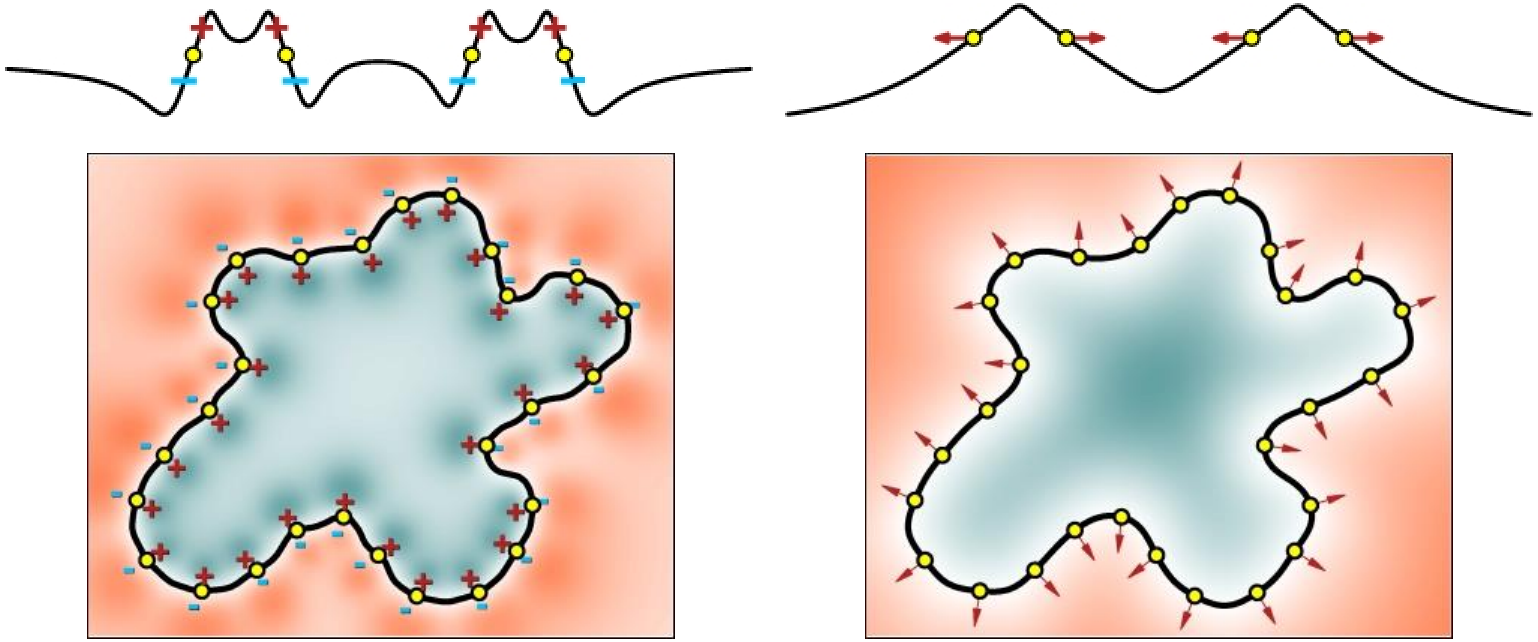
$$F(\mathbf{x}) = f_{\mathbf{x}}(\mathbf{x}), \quad f_{\mathbf{x}} = \underset{f \in \Pi_k^d}{\operatorname{argmin}} \sum_{m=0}^{N-1} \theta(\|\mathbf{c}_m - \mathbf{x}\|)\left(f(\mathbf{c}_m) - d_m\right)^2$$

Example: Reconstruction



MLS SDF – Possible Improvement

- Point constraints vs. true normal constraints



- Details: see [Shen et al. SIGGRAPH 2004] and Ex2

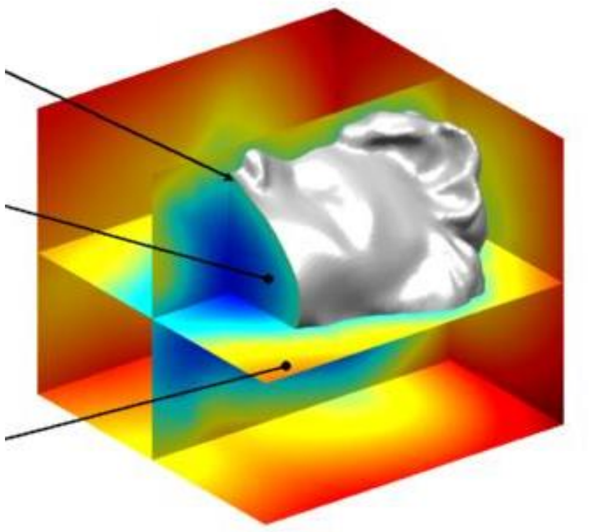
Global RBF vs. Local MLS

- RBF:
  - sees the whole data set, can make for very smooth surfaces
  - global (dense) system to solve – expensive

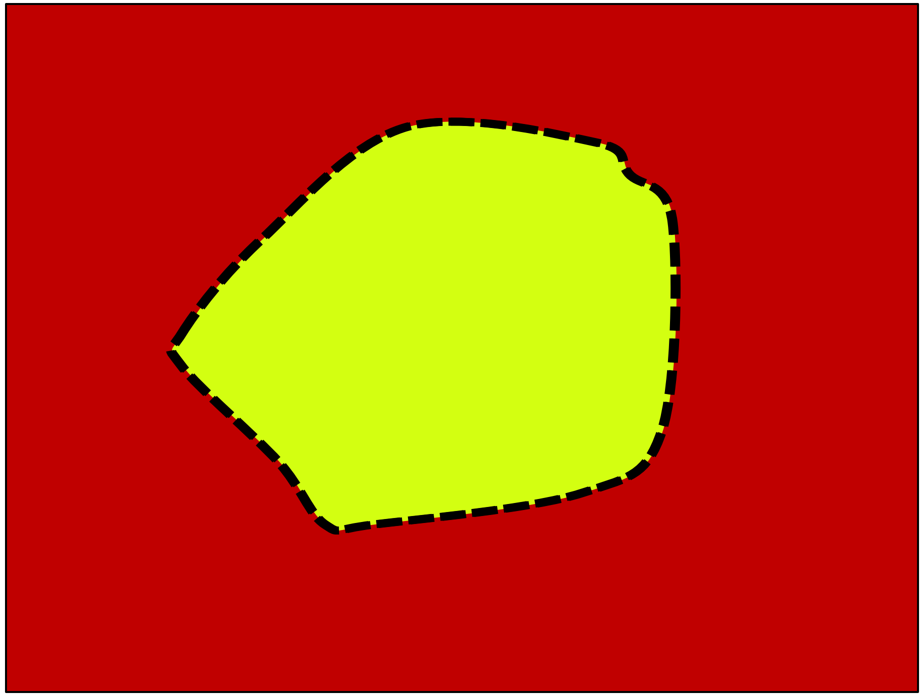
- MLS:
  - sees only a small part of the dataset, can get confused by noise
  - local linear solves – cheap

Extracting the Surface

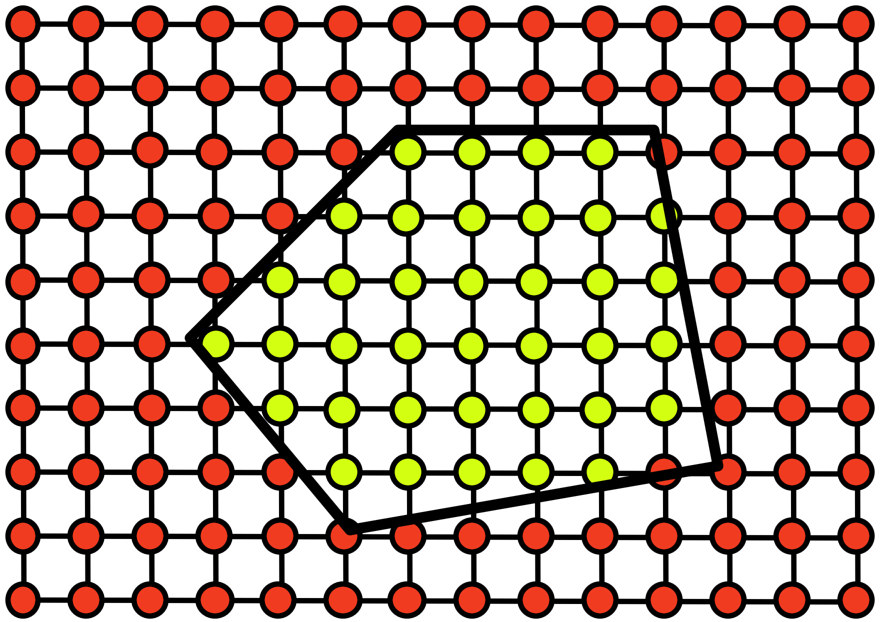
- Wish to compute a manifold mesh of the level set



Sample the SDF

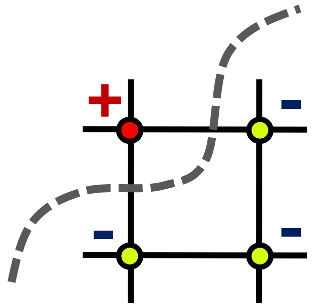


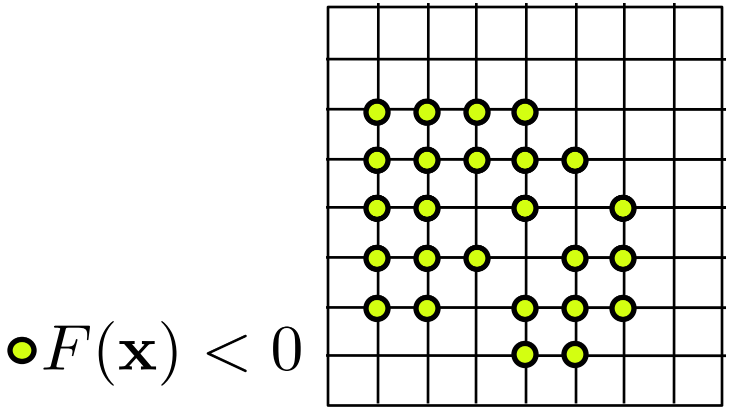
Sample the SDF

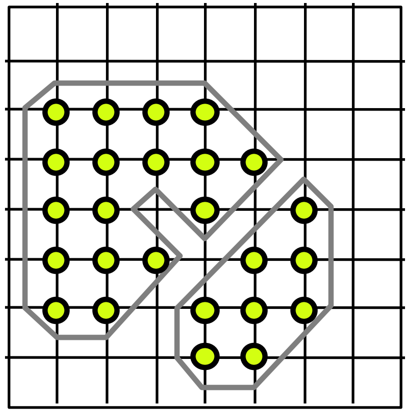


Tessellation

- Want to approximate an implicit surface with a mesh
- Can‘t explicitly compute all the roots
  - Sampling the level set is difficult (root finding)
- Solution: find approximate roots by trapping the implicit surface in a grid (lattice)

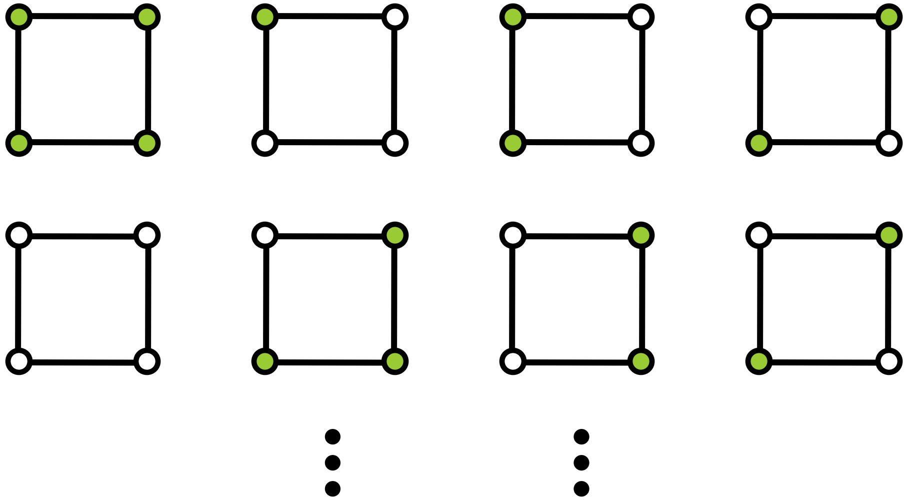






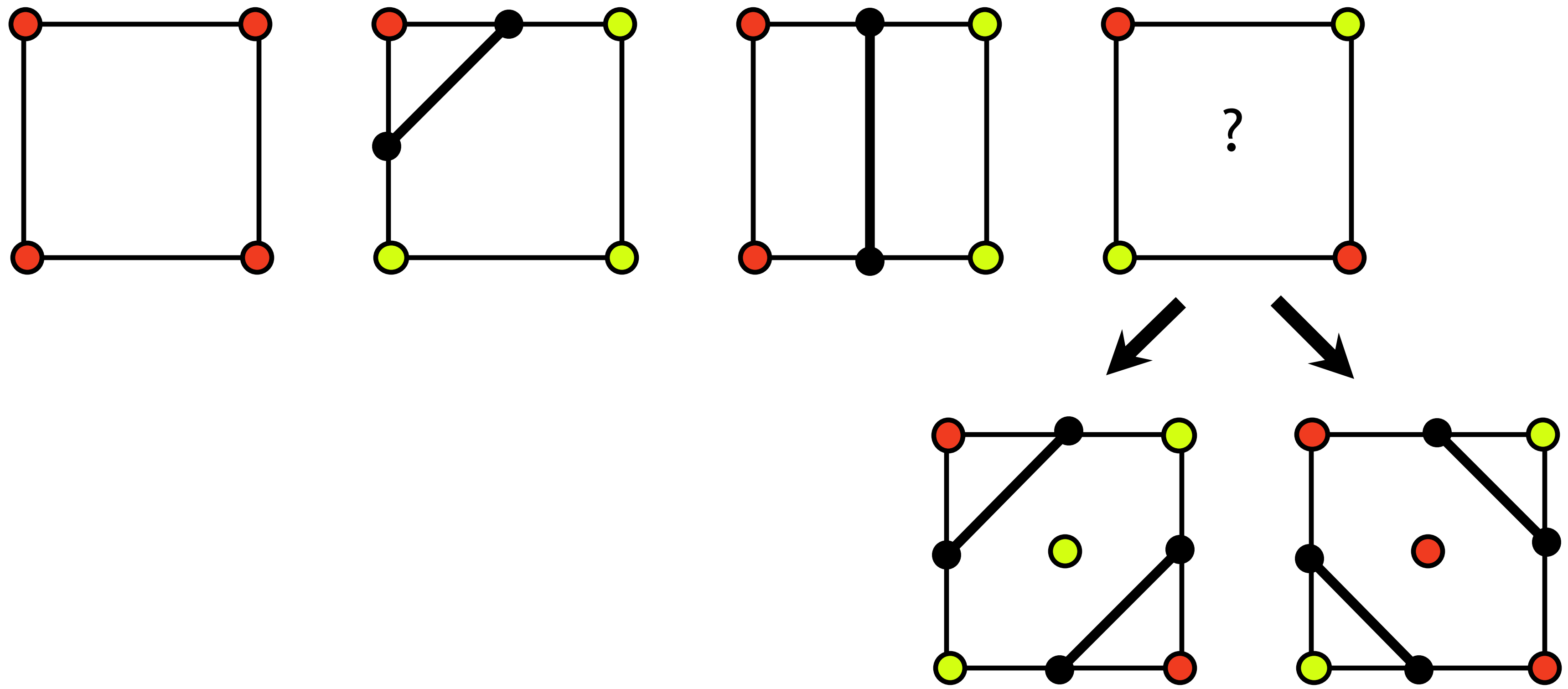
Marching Squares

- 16 different configurations in 2D
- 4 equivalence classes (up to rotational and reflection symmetry + complement)



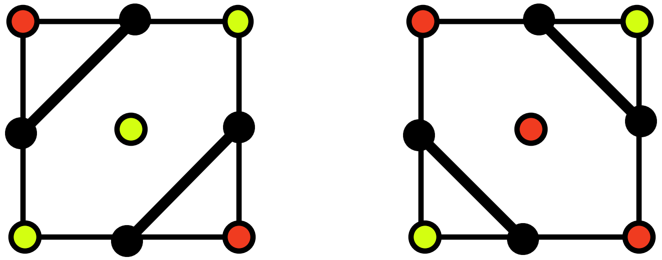
Tessellation in 2D

- 4 equivalence classes (up to rotational and reflection symmetry + complement)

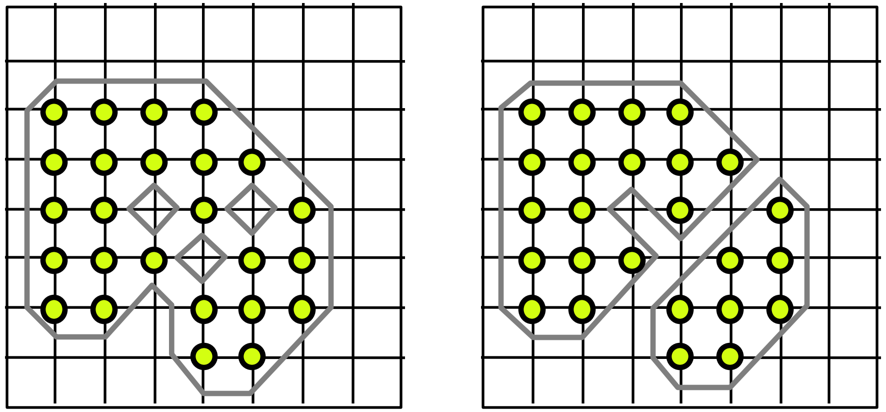


Tessellation in 2D

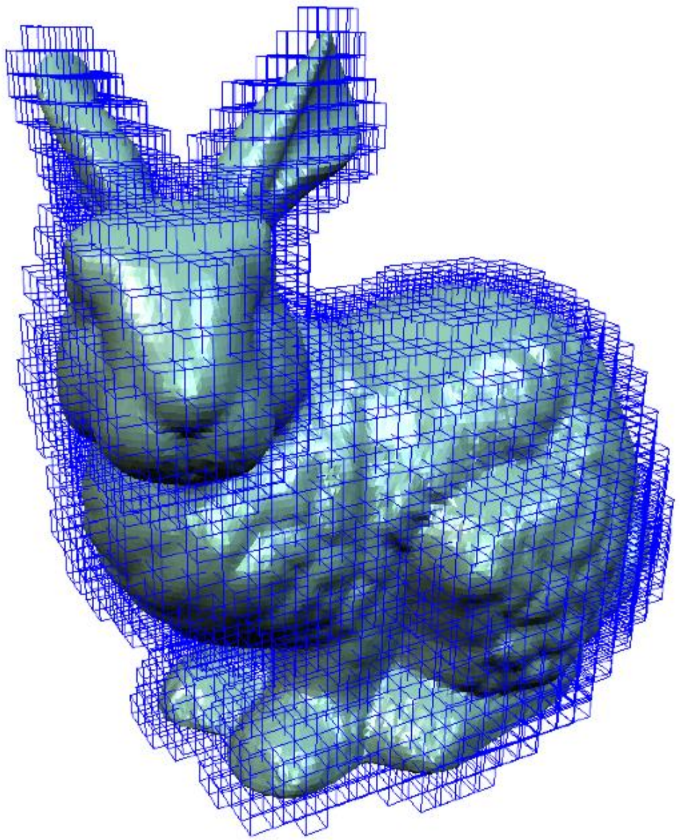
- Case 4 is ambiguious:

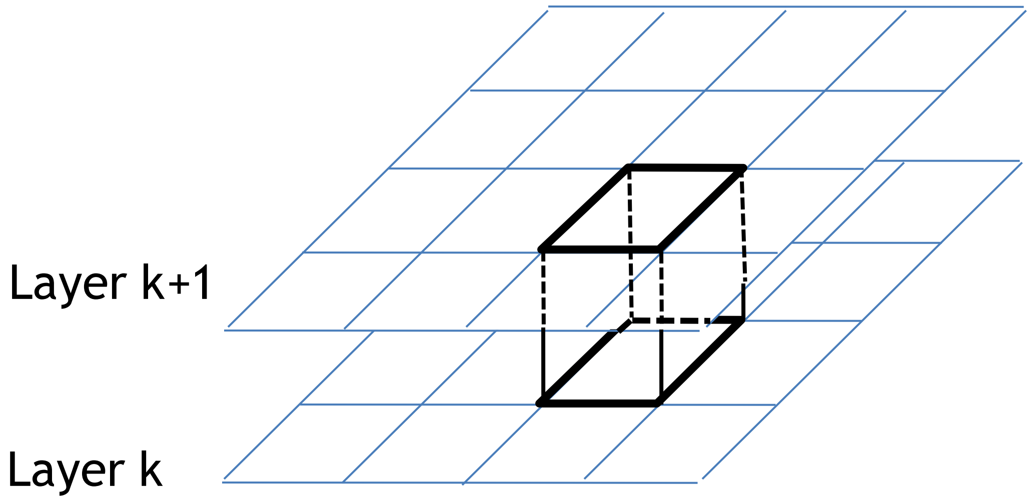


- Always pick consistently to avoid problems with the resulting mesh



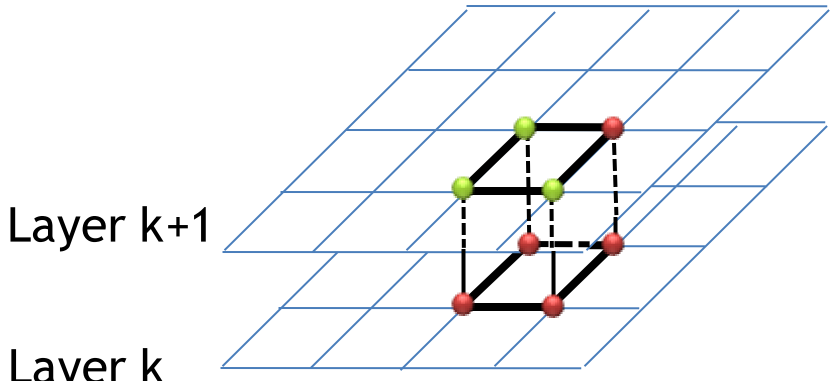
3D: Marching Cubes





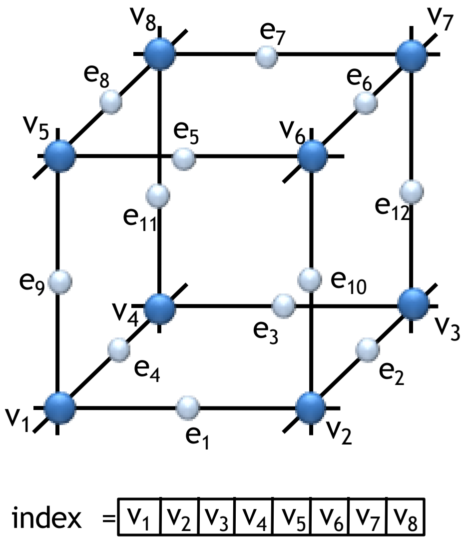
Marching Cubes

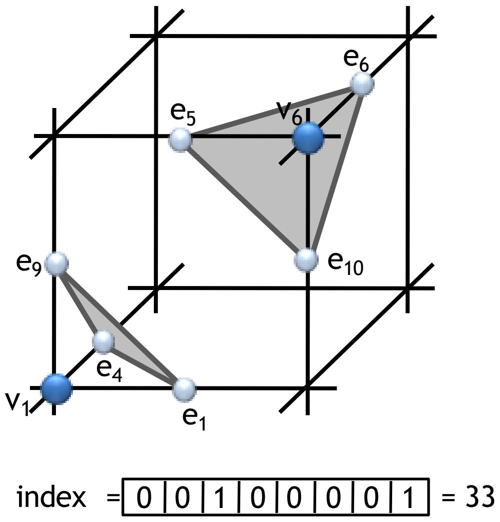
- Marching Cubes (Lorensen and Cline 1987)
  1. Load 4 layers of the grid into memory
  2. Create a cube whose vertices lie on the two middle layers
  3. Classify the vertices of the cube according to the implicit function (inside, outside or on the surface)



Marching Cubes

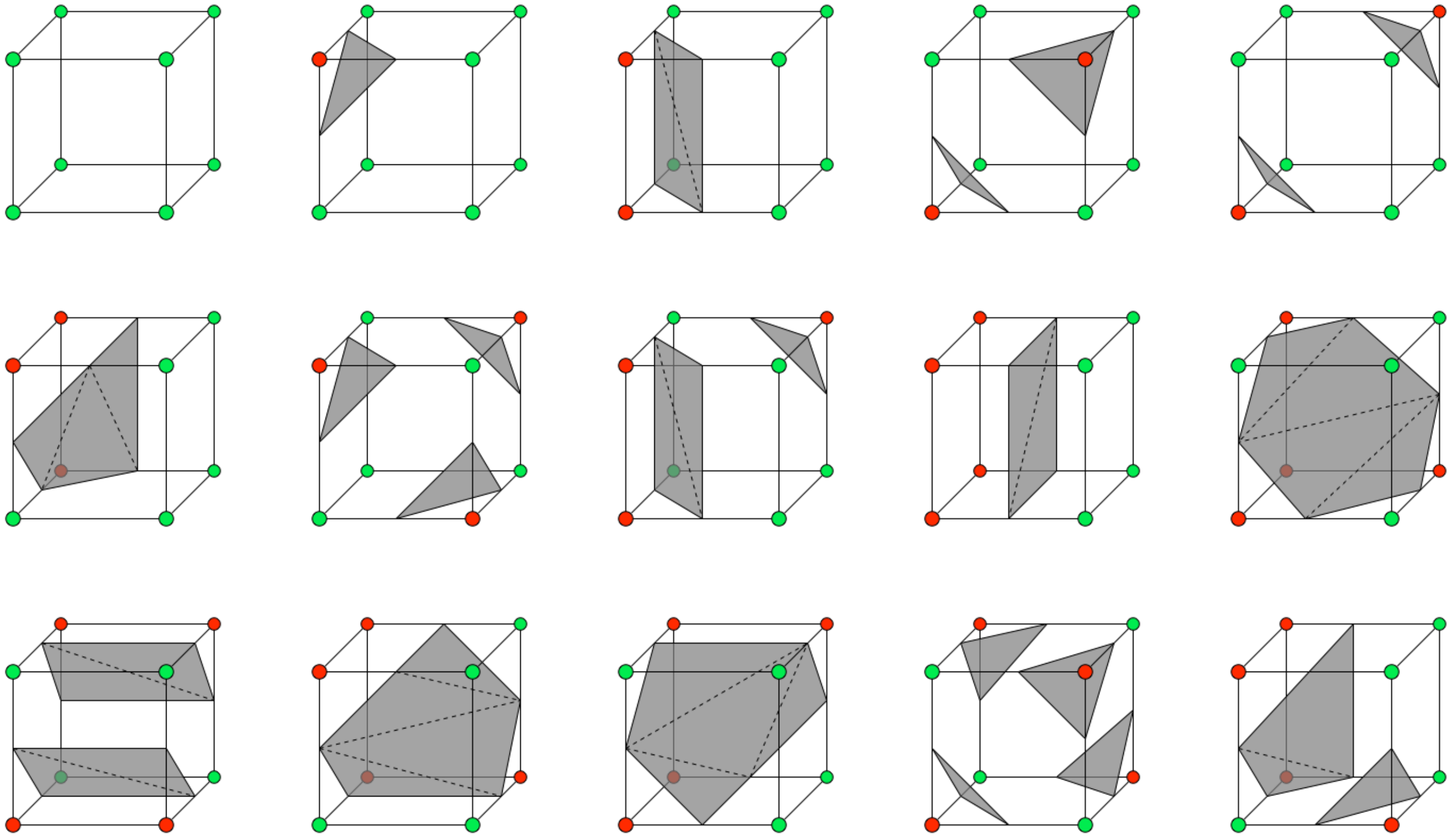
4. Compute case index. We have $2^8 = 256$ cases (0/1 for each of the eight vertices) – can store as 8 bit (1 byte) index.





Marching Cubes

- Unique cases (by rotation, reflection and complement)

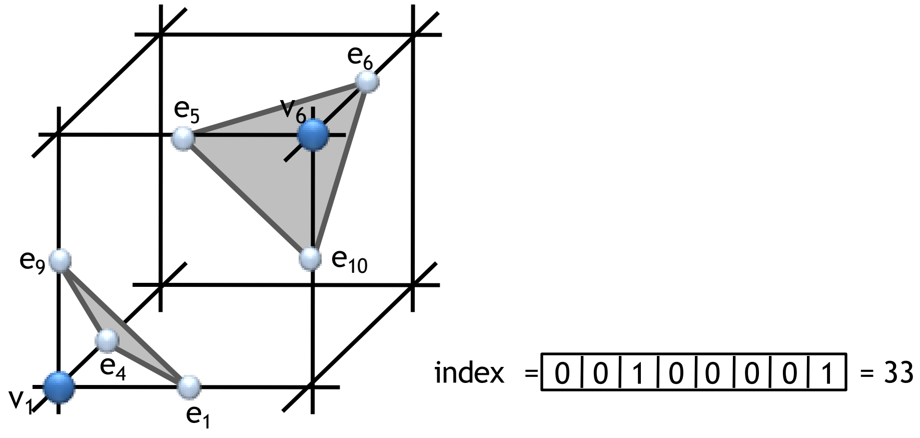


Tessellation

3D – Marching Cubes

5. Using the case index, retrieve the connectivity in the look-up table

- Example: the entry for index 33 in the look-up table indicates that
  - the cut edges are e_1; e_4; e_5; e_6; e_9 and e_{10} ;
  - the output triangles are (e_1; e_9; e_4) and (e_5; e_{10}; e_6).



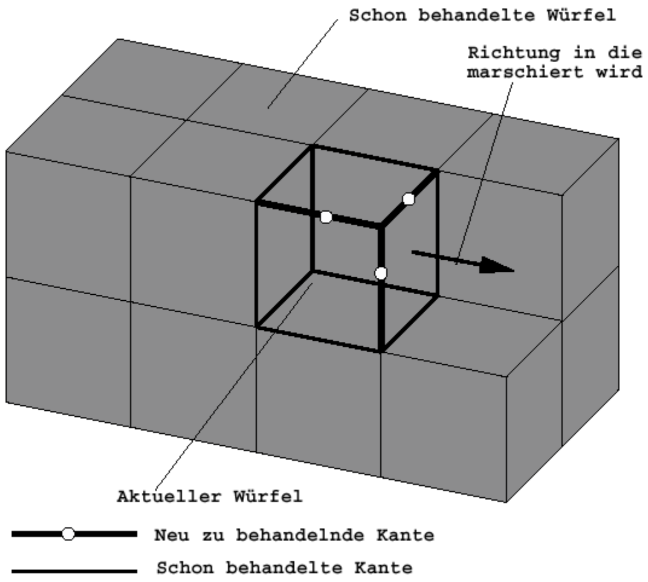
Marching Cubes

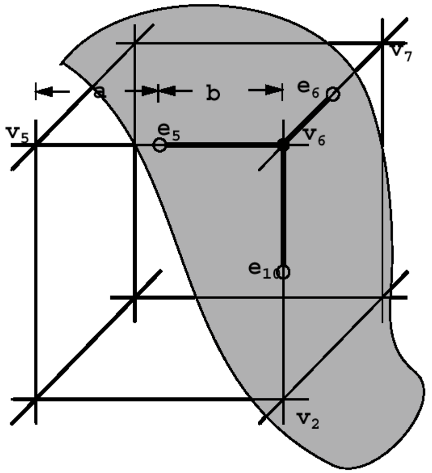
6. Compute the position of the cut vertices by linear interpolation:

$$\mathbf{v}_s = t\mathbf{v}_a + (1-t)\mathbf{v}_b$$

$$t = \frac{F(\mathbf{v}_b)}{F(\mathbf{v}_b) - F(\mathbf{v}_a)}$$

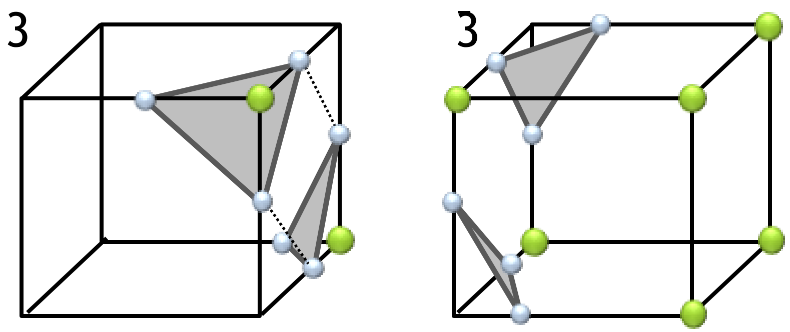
7. Move to the next cube

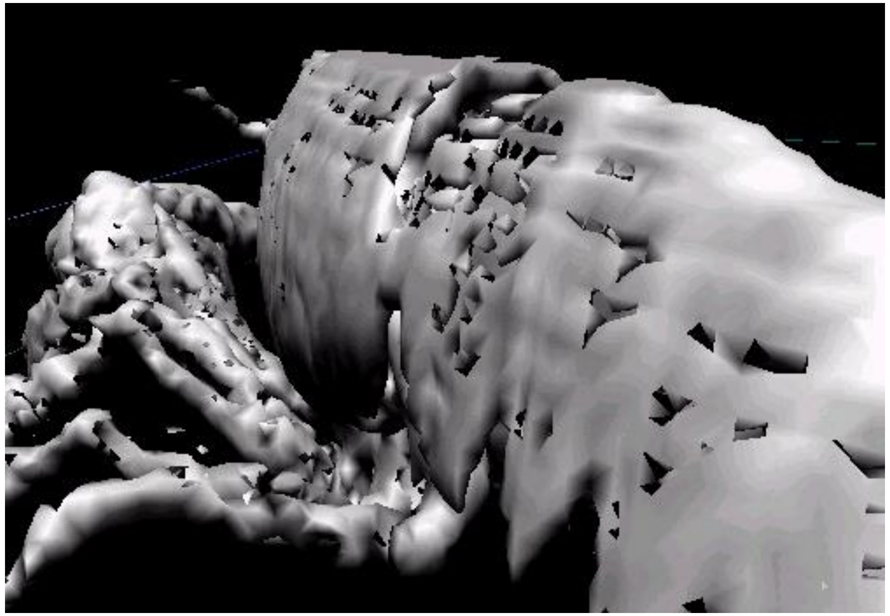




Marching Cubes – Problems

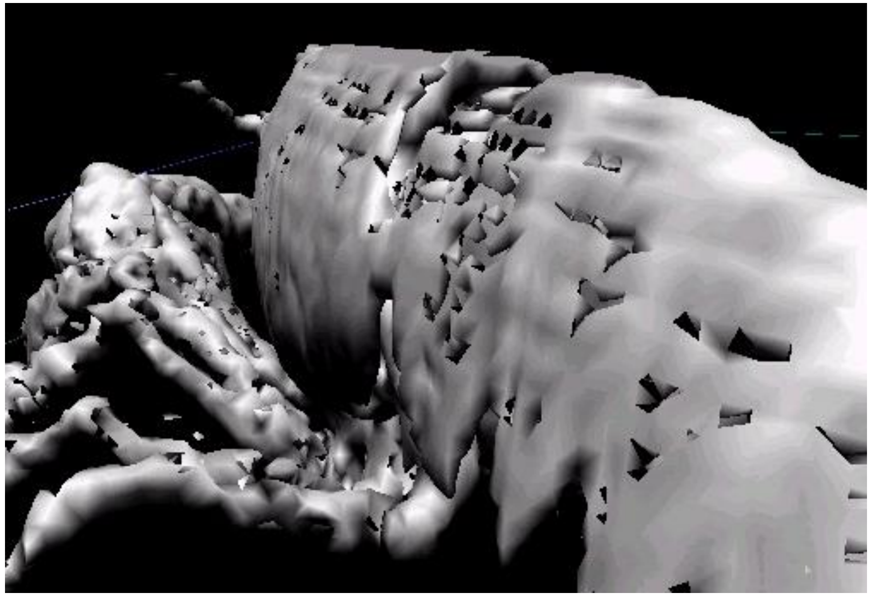
- Have to make consistent choices for neighboring cubes – otherwise get holes

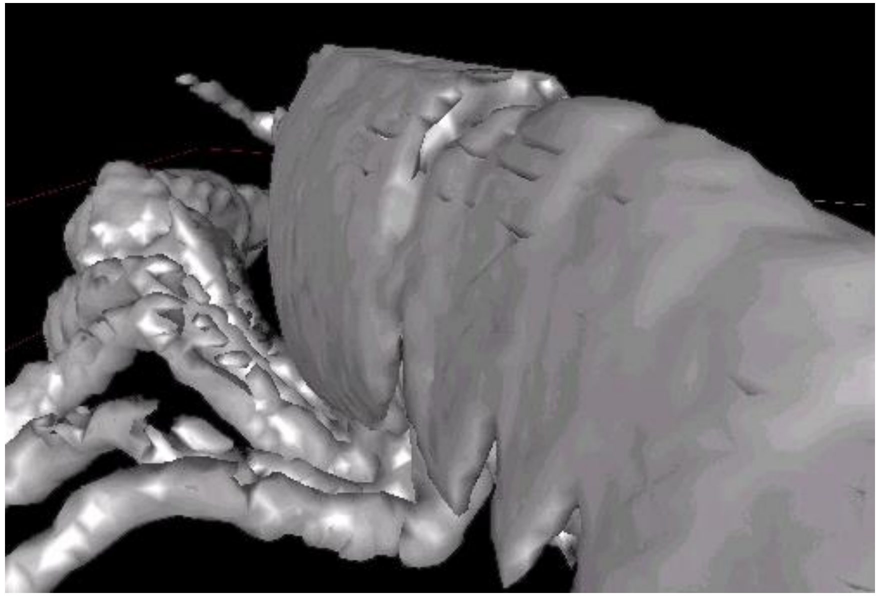




Marching Cubes – Problems

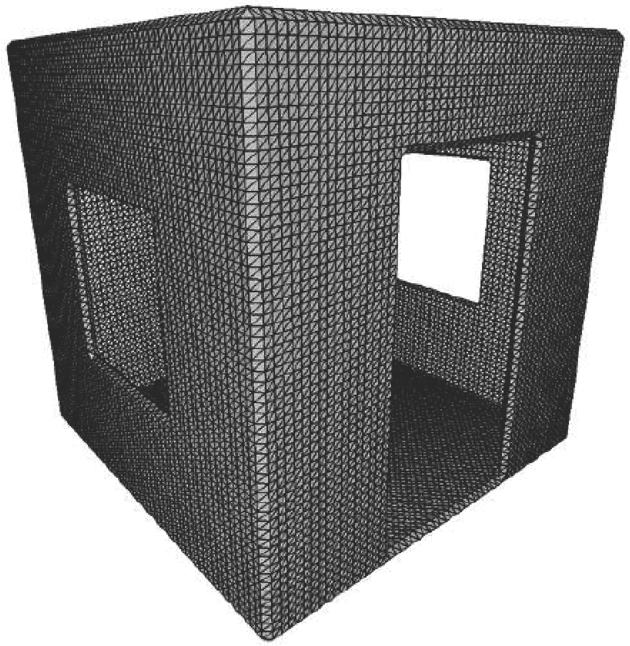
- Resolving ambiguities


Ambiguity


No Ambiguity

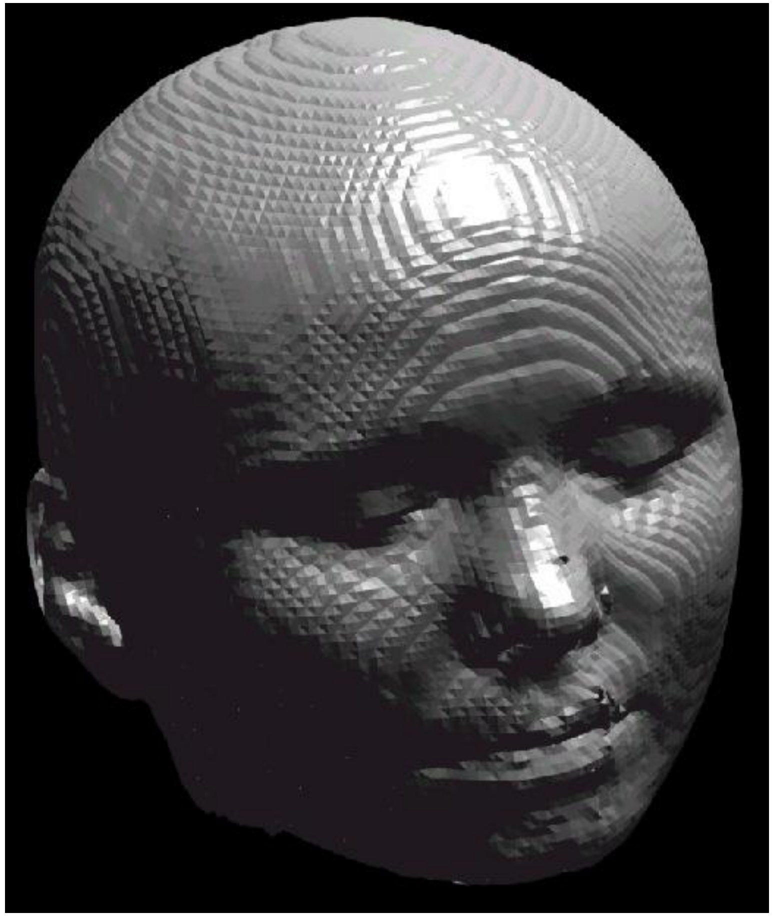
Marching Cubes – Problems

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



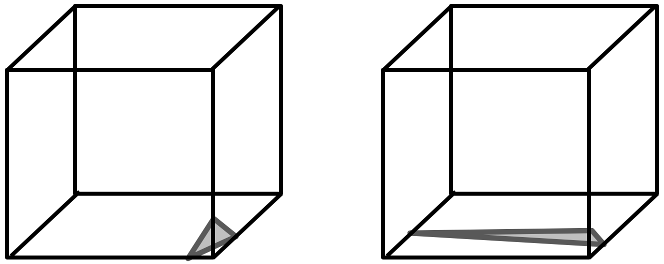
Images from: "Dual Marching Cubes: Primal Contouring of Dual Grids" by Schaeffer et al.

Marching Cubes – Problems



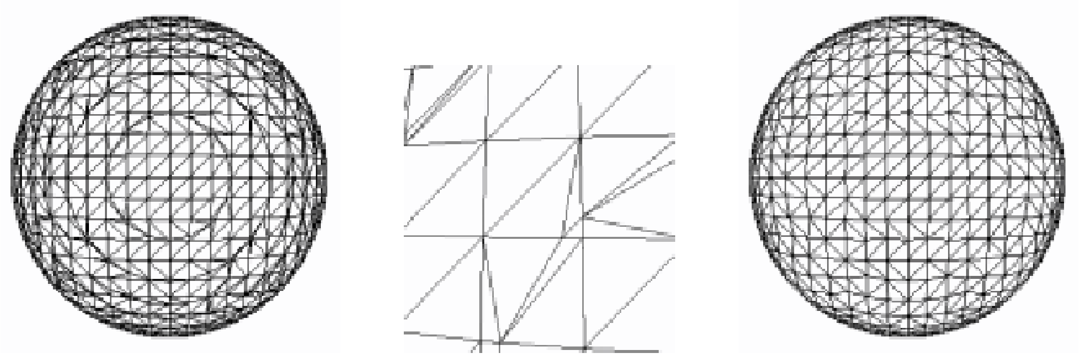
Marching Cubes – Problems

- Problems with short triangle edges
  - When the surface intersects the cube close to a corner, the resulting tiny triangle doesn‘t contribute much area to the mesh
  - When the intersection is close to an edge of the cube, we get skinny triangles (bad aspect ratio)
- Triangles with short edges waste resources but don‘t contribute to the surface mesh representation



Grid Snapping

- Solution: threshold the distances between the created vertices and the cube corners
- When the distance is smaller than d_{snap} we snap the vertex to the cube corner
- If more than one vertex of a triangle is snapped to the same point, we discard that triangle altogether



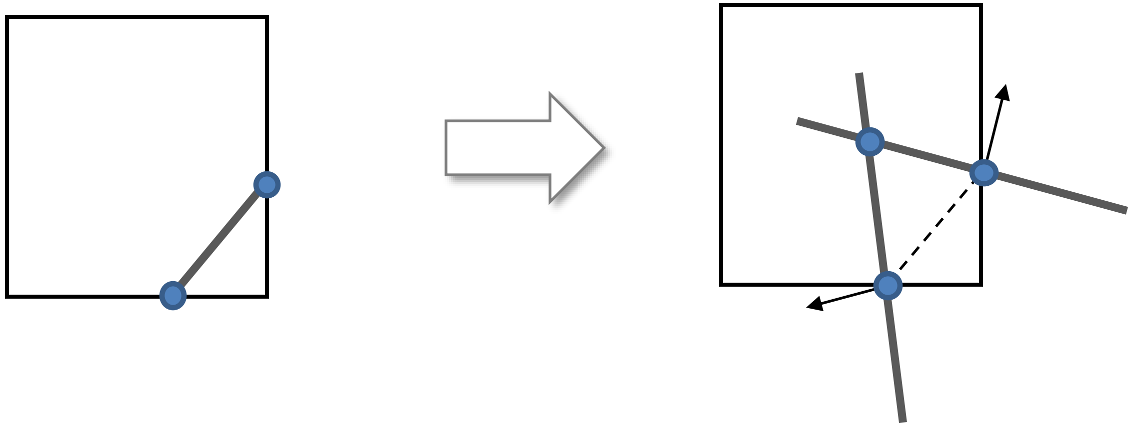
Grid Snapping

- With grid snapping one can obtain significant reduction of space consumption

d_{snap}	0	0,1	0,2	0,3	0,4	0,46	0,495
Vertices	1446	1398	1254	1182	1074	830	830
Reduction (%)	0	3,3	13,3	18,3	25,7	42,6	42,6

Sharp Corners and Features

- Kobbelt et al. SIGGRAPH 2001 „Feature sensitive surface extraction from volume data“
  - Evaluate the normals (use gradient of F)
  - When they significantly differ, create an additional vertex



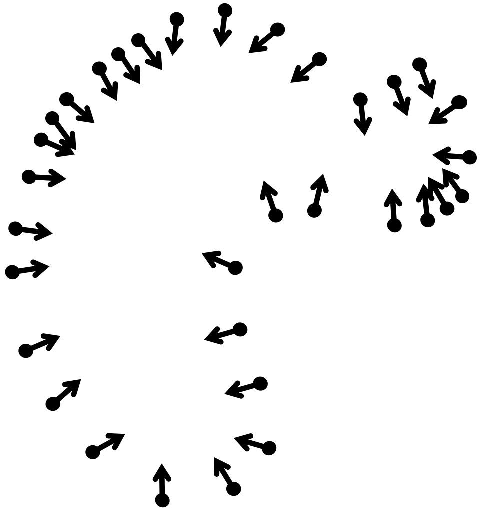
Poisson Surface Reconstruction

- Very popular modern method, code available:
  M. Kazhdan, M. Bolitho and H. Hoppe, Symposium on Geometry Processing 2006, follow-up in 2013, 2020…
  http://www.cs.jhu.edu/~misha/Code/PoissonRecon/

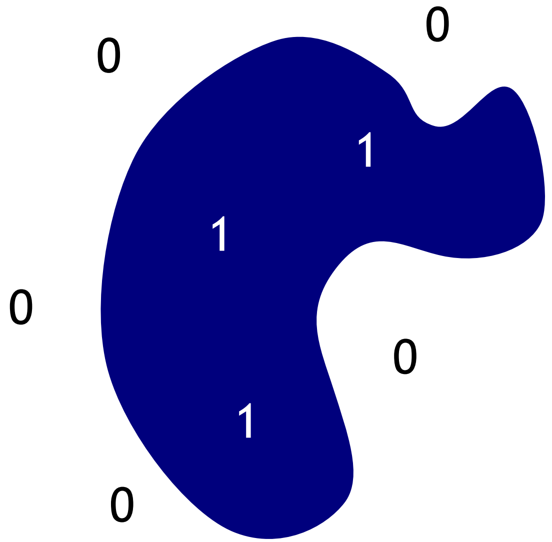
- Global fitting of an *indicator function* using a PDE
  - Robust to noise, sparse, computationally tractable

- You will try out the code in Ex2 and compare with MLS results

Poisson Surface Reconstruction



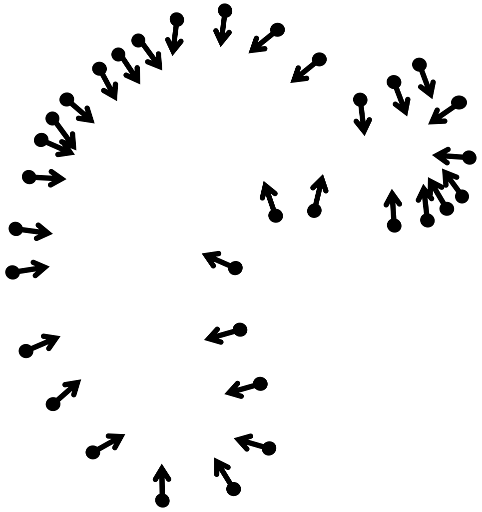
Oriented points



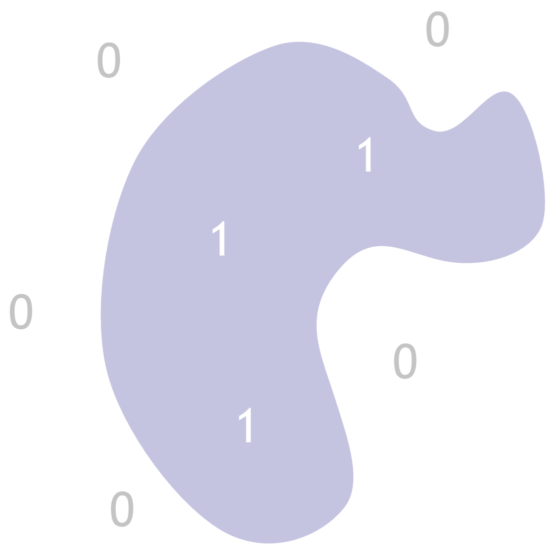
Indicator function

$\chi_{\mathcal{M}}$

Poisson Surface Reconstruction



Oriented points

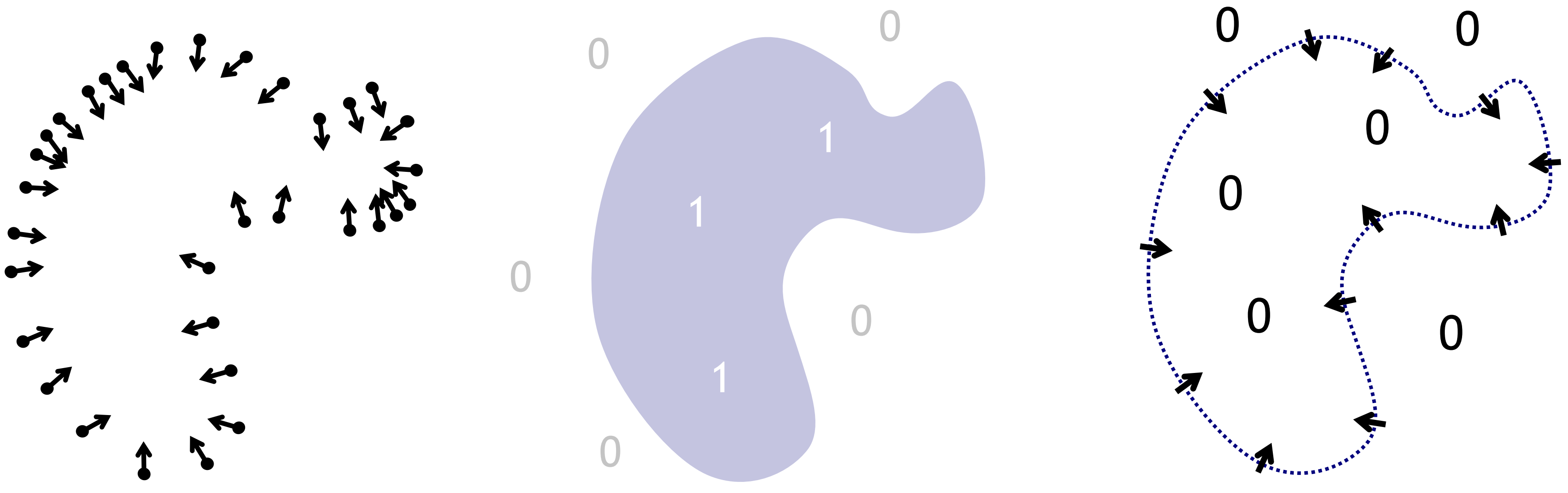


Indicator function

$\chi_{\mathcal{M}}$

We don't know the indicator function ☹

Poisson Surface Reconstruction



Oriented points

Indicator function

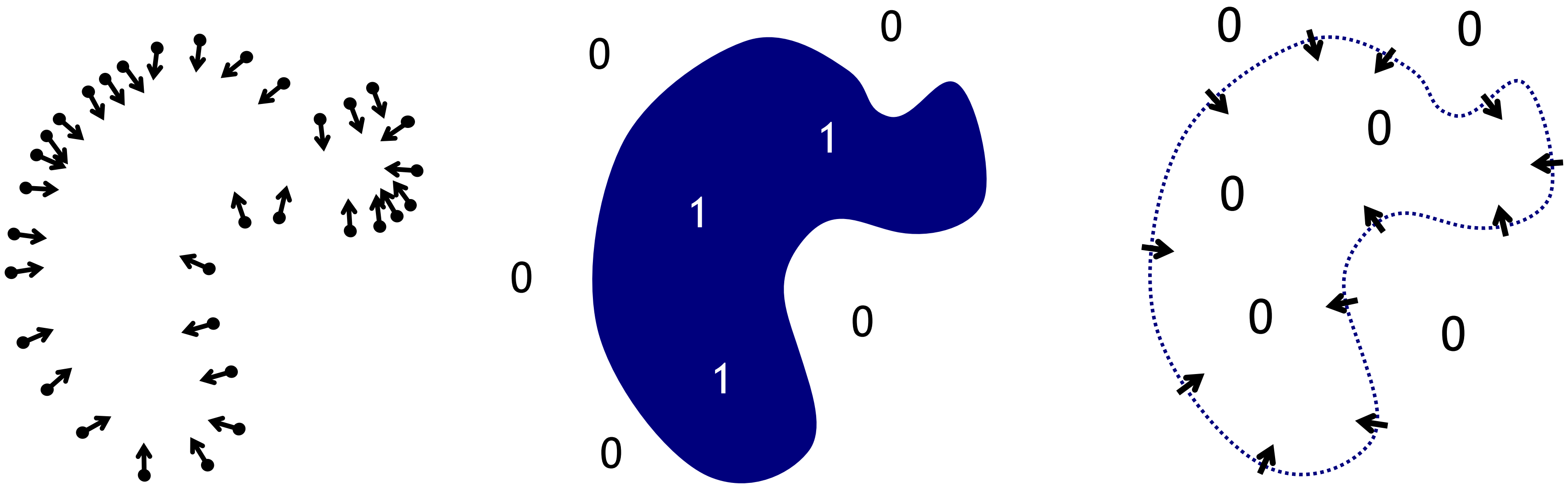
$\chi_{\mathcal{M}}$

Indicator gradient

$\nabla \chi_{\mathcal{M}}$

But we can estimate its gradient! ☺

Poisson Surface Reconstruction



Oriented points

Indicator function

$$\chi_{\mathcal{M}}$$

Indicator gradient

$$\nabla \chi_{\mathcal{M}}$$

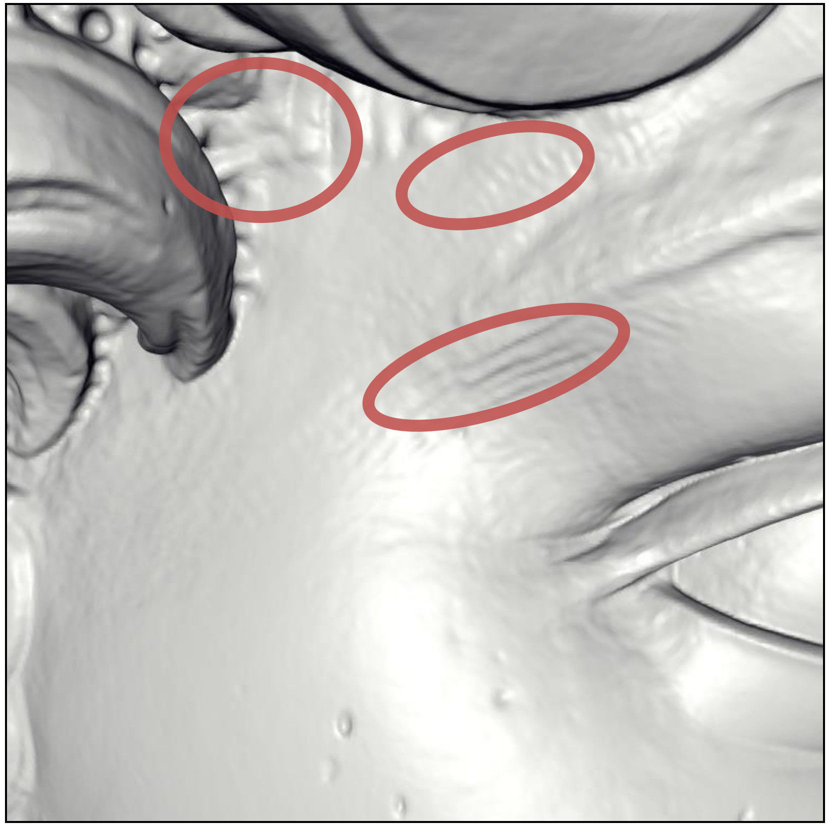
Reconstruct χ by solving the Poisson equation

$$\Delta \chi_{\mathcal{M}} = \mathrm{div} \nabla \chi_{\mathcal{M}}$$

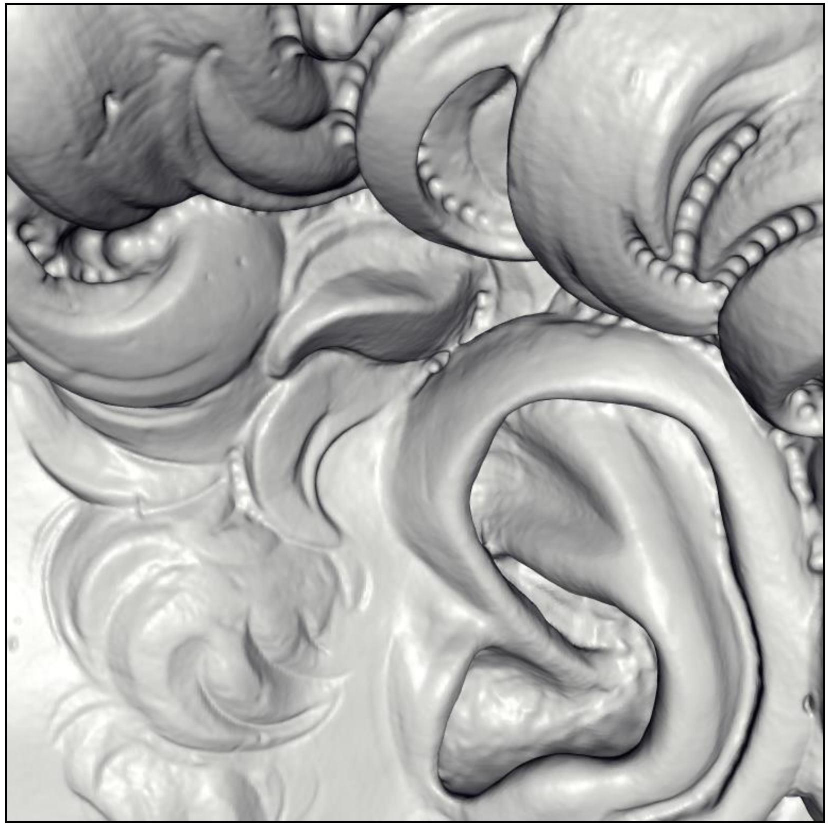
Michelangelo's David

- 215M data points from 1000 scans
- 22M triangle reconstruction
- Compute time: 2.1 hours
  (this was in year 2006)
- Peak memory: 6600 MB

David – Chisel marks



David – Drill Marks



David – Eye

Thank You

INTERACTIVE GEOMETRY LAB

March 5, 2025

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich