Lecture III: Surface Reconstruction

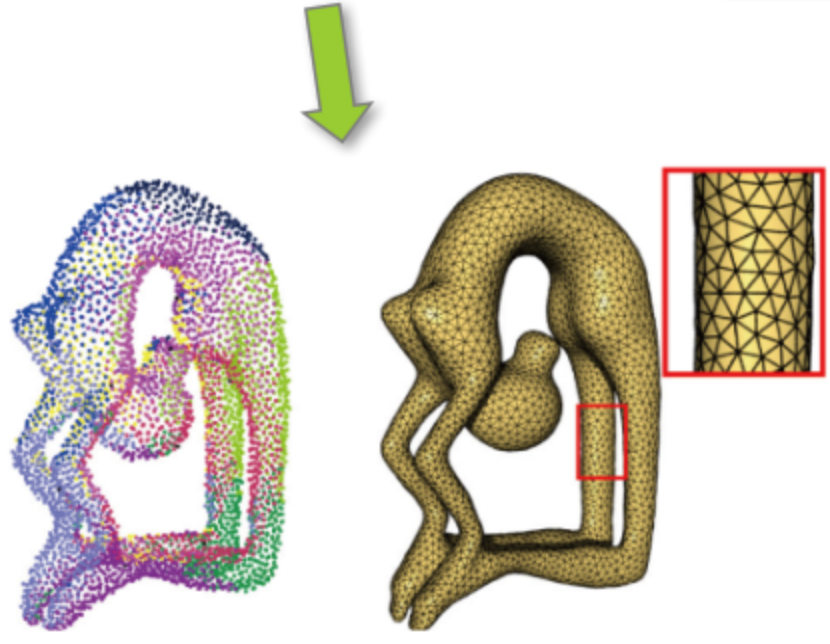
Geometry Acquisition Pipeline

Scanning:
Range images

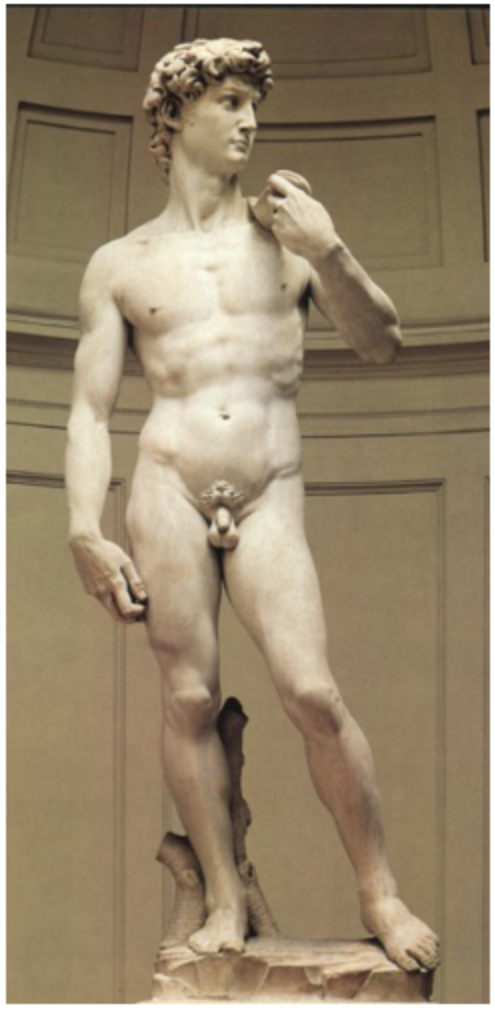
Registration:
Align range images

Stitching/ Reconstruction:
Create mesh from scans

Postprocess:
- Topological and geometric filtering
- Remeshing
- Compression



Digital Michelangelo Project



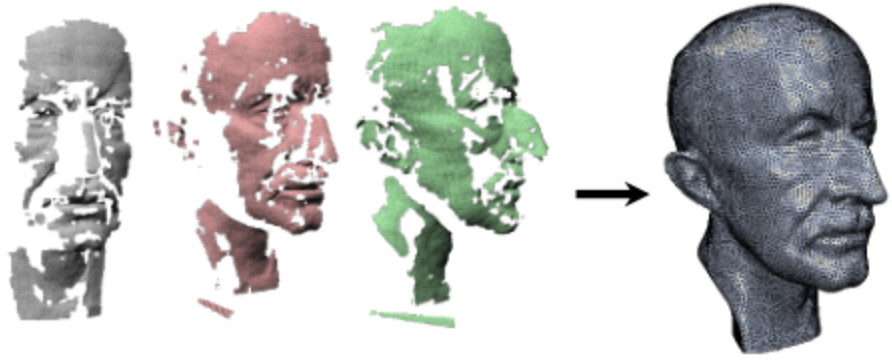
1G sample points → 8M triangles

4G sample points → 8M triangles

Input to Reconstruction Process

- Input option 1: just a set of 3D points, irregularly spaced
  - Need to estimate normals

- Input option 2: normals come from the range scans

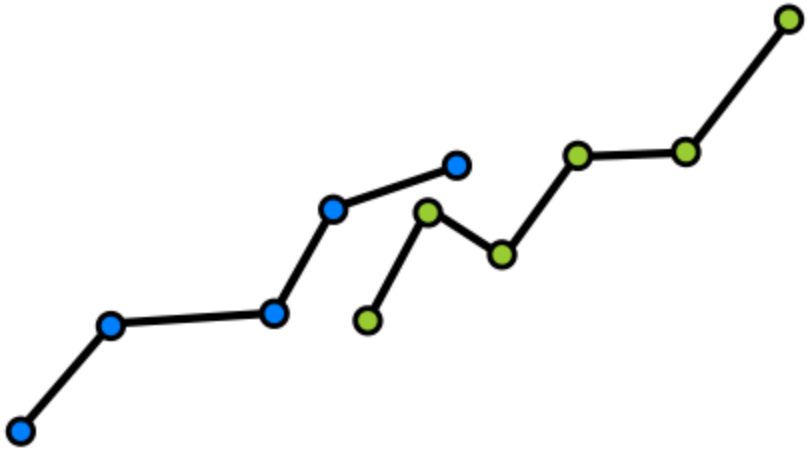


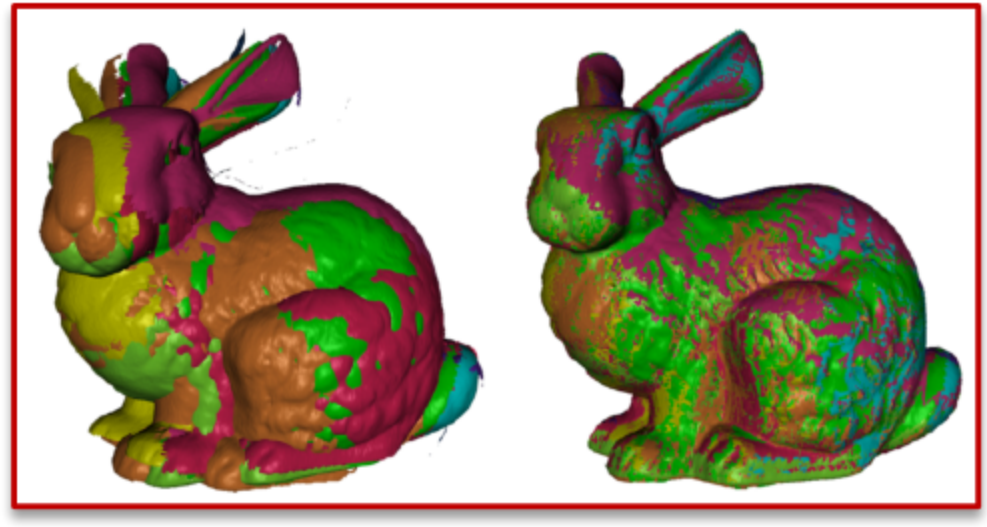
set of raw scans

reconstructed model

How to Connect the Dots?

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

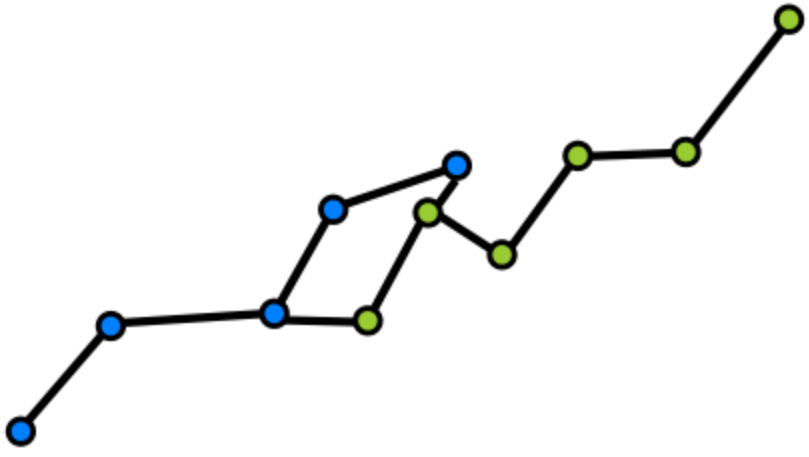




"Zippered Polygon Meshes from Range Images", Greg Turk and Marc Levoy, ACM SIGGRAPH 1994

How to Connect the Dots?

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



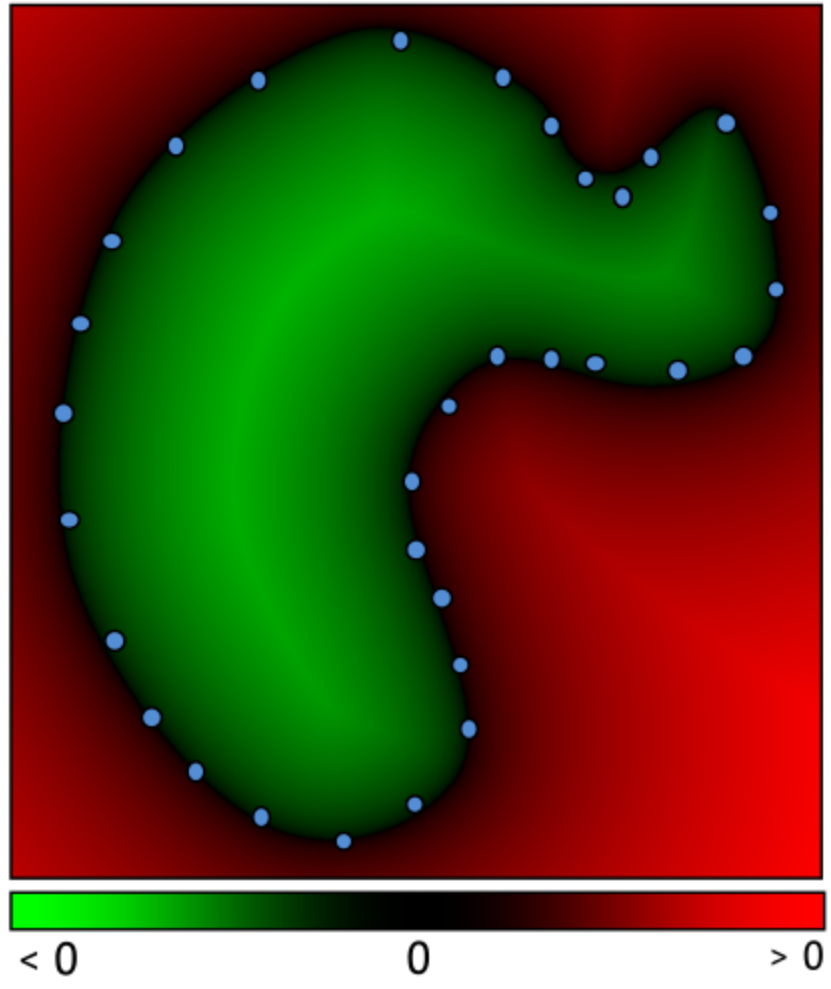
- Connect sample points by triangles.
- Exact interpolation of sample points.
- Bad for noisy or misaligned data.
- Can lead to holes or non-manifold situations.

Implicit Function Approach

- Compute a function

$$f : \mathbb{R}^3 \rightarrow \mathbb{R}$$

  - Value > 0: outside.
  - Value < 0: inside.



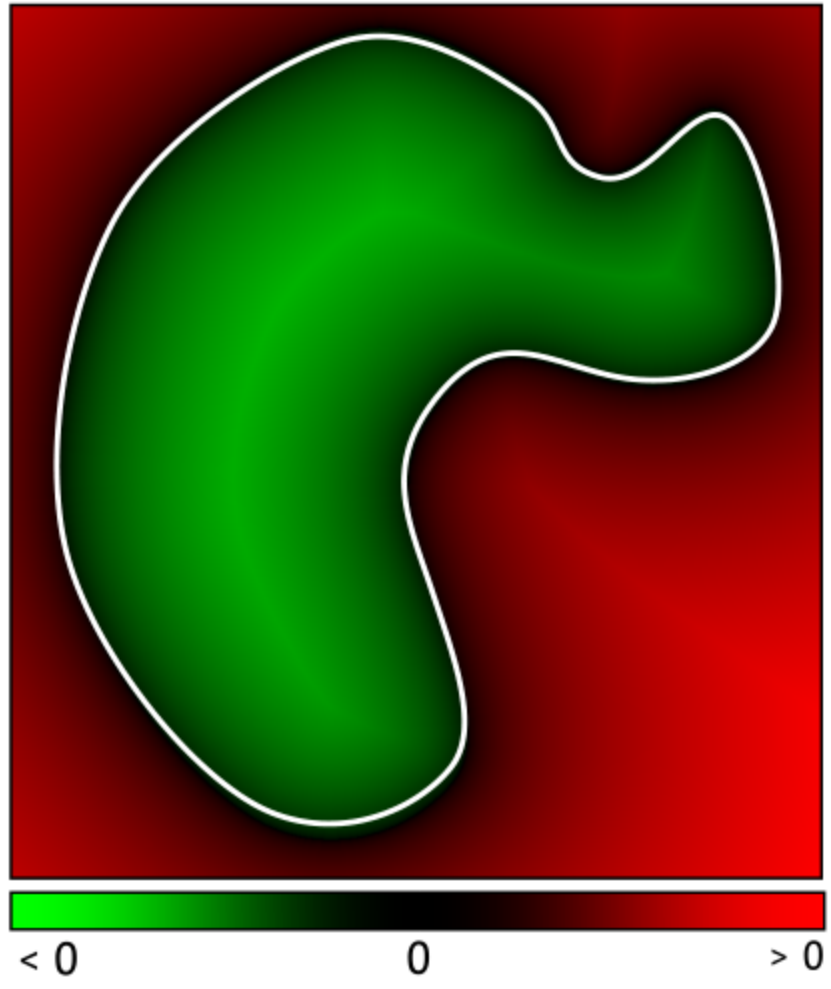
Implicit Function Approach

- Compute a function

$$f : \mathbb{R}^3 \rightarrow \mathbb{R}$$

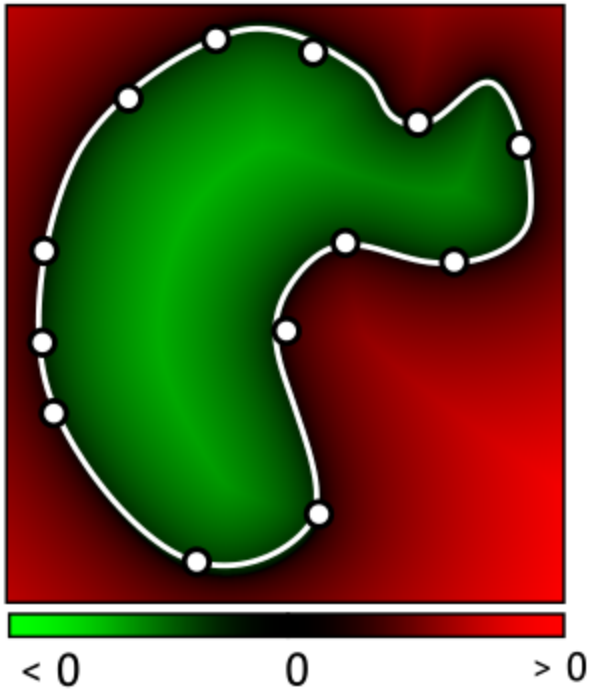
  - Value > 0: outside.
  - Value < 0: inside.
    Extract the zero-set

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

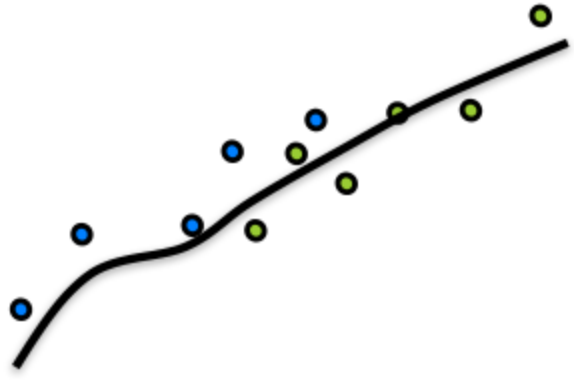


How to Connect the Dots?

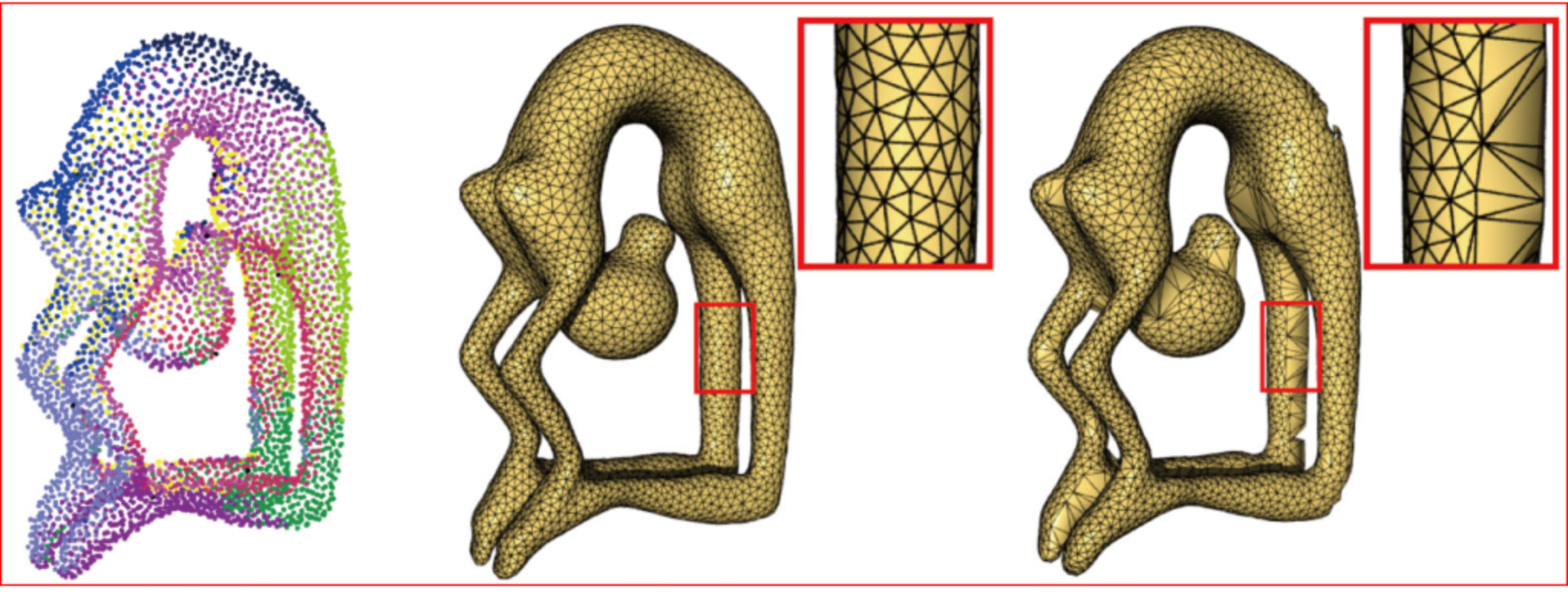
- **Implicit reconstruction**: estimate this signed distance function (SDF); extract zero-level set mesh using Marching Cubes.



- Approximation of input points.
- Watertight manifold results by construction.



Implicit vs. Explicit



Input

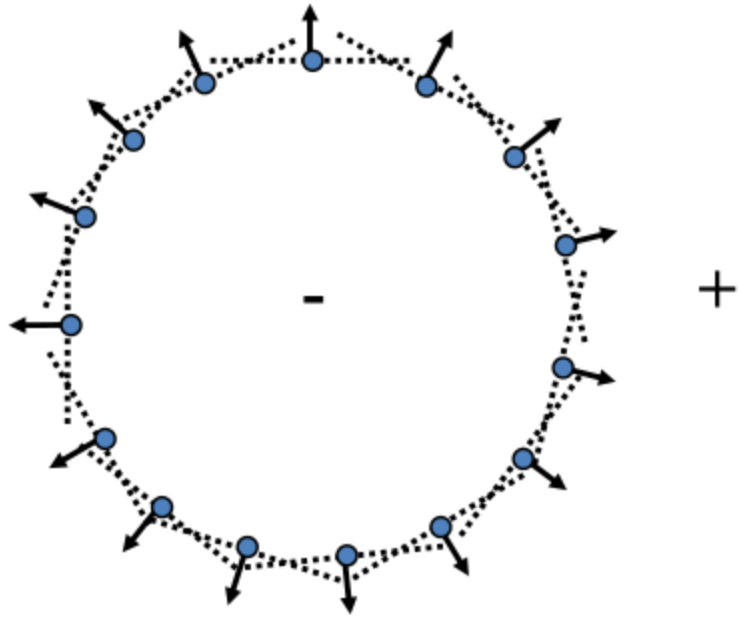
Implicit

Explicit

SDF from Points and Normals

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

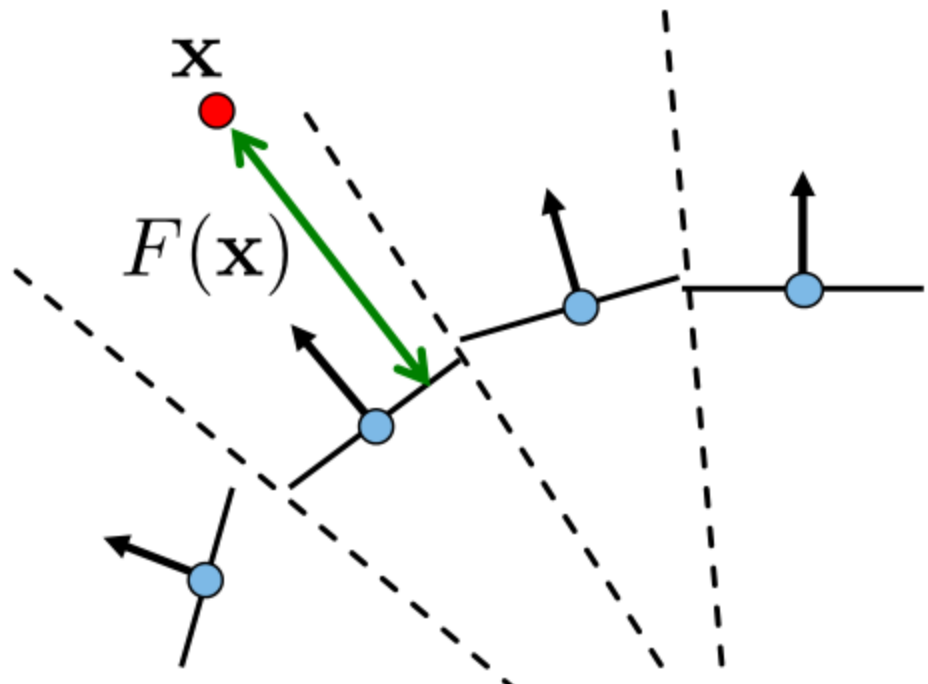
- Normals help to distinguish between inside and outside.



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

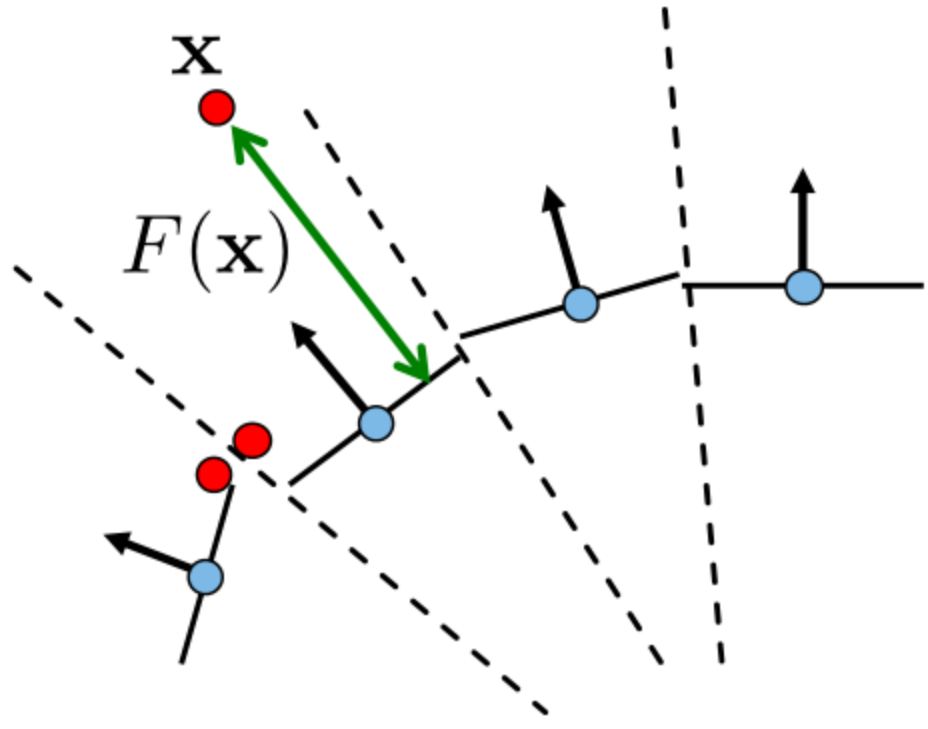
SDF from Points and Normals

- Compute signed distance to the tangent plane of the closest point.
- Problem?



SDF from Points and Normals

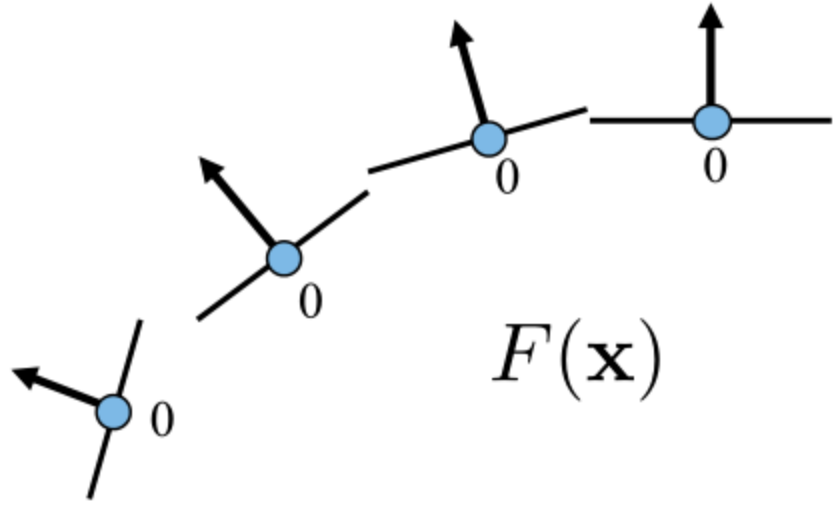
- Compute signed distance to the tangent plane* of the closest point.
- The function is discontinuous!



* 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.
- Requiring:
  - $F(p_i) = 0$ (interpolating)
  - F is smooth.
  - Avoid trivial solution $F \equiv 0$.

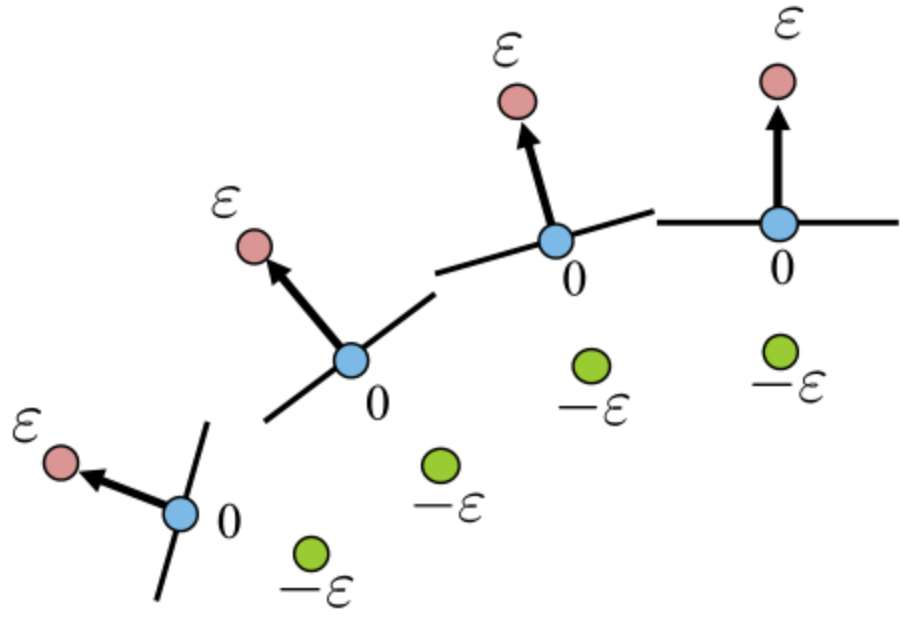


"Reconstruction and representation of 3D objects with radial basis functions", Carr et al., ACM SIGGRAPH 2001

Smooth SDF

- Requiring:
  - $F(p_i) = 0$ (interpolating)
  - F is smooth.
  - Avoid trivial solution $F \equiv 0$.

- Add off-surface (positive and negative normal) constraints.



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

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

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

Radial Basis Function Interpolation

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

$$F(x) = \sum_{i=0}^{N-1} w_i \varphi(\|x - c_i\|)$$

Scalar weights
Unknowns

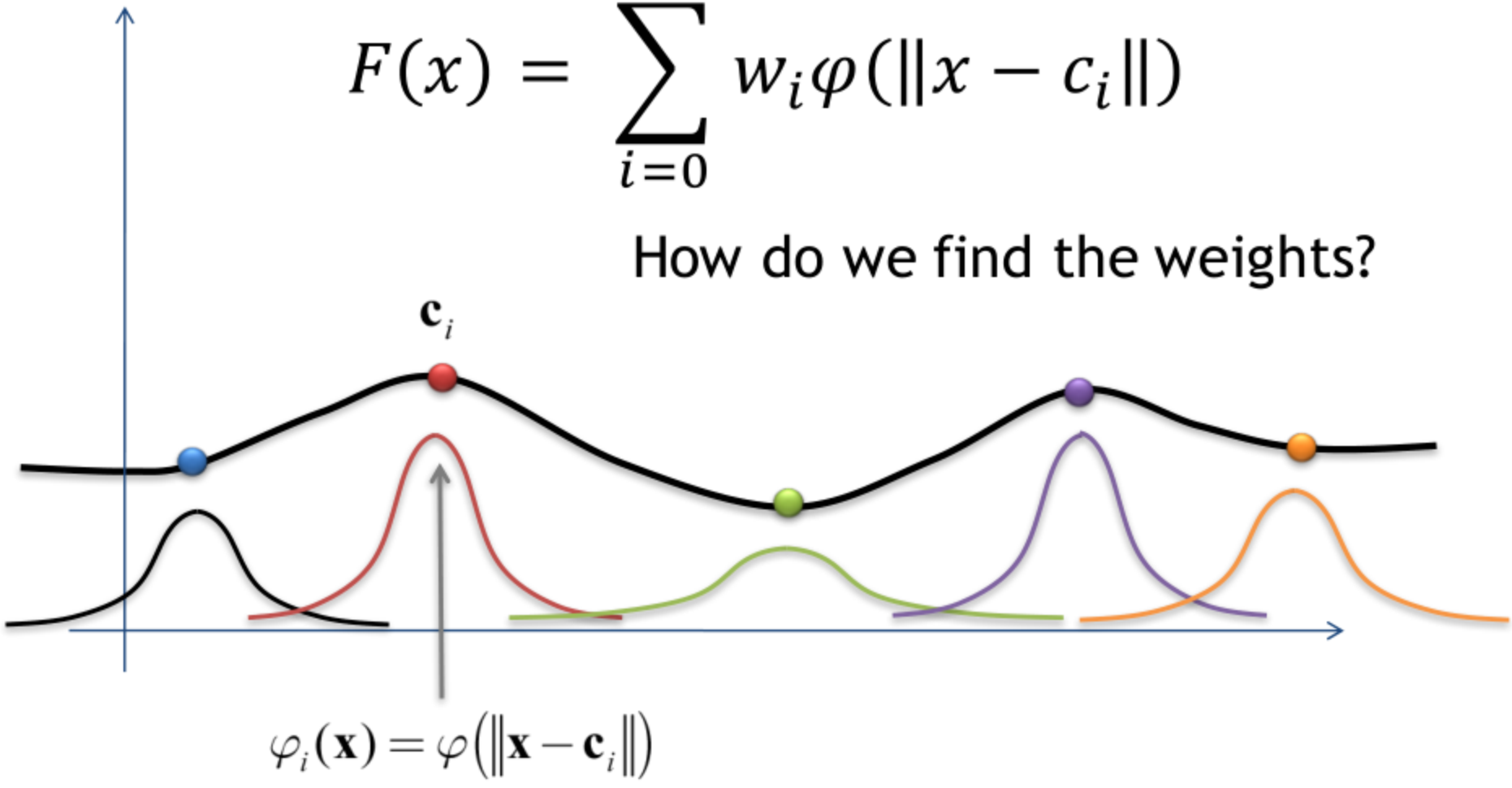
Smooth kernels (basis functions) centered at constrained points.
For example:
$\varphi(r) = r^3$

$N = 3n$

Radial Basis Function Interpolation

$$F(x) = \sum_{i=0}^{N-1} w_i \varphi(\|x - c_i\|)$$

How do we find the weights?



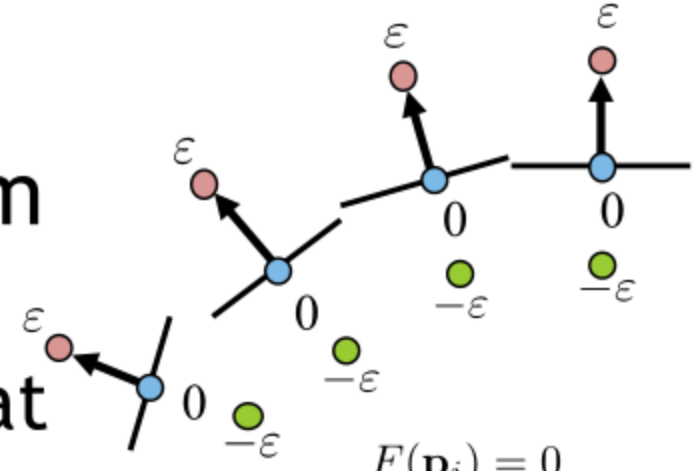
Radial Basis Function Interpolation

- Interpolate the constraints:

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

$$\forall j = 0, \ldots, N-1, \quad \sum_{i=0}^{N-1} w_i\, \varphi(\|\mathbf{c}_j - \mathbf{c}_i\|) = d_j$$

- In words: we know the desired function value (sum of RBF) at each center.
  - We solve for the weights that do that!



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

Radial Basis Function Interpolation

- Interpolate the constraints:

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

  - Symmetric linear system to get the weights:

$$\begin{pmatrix} \varphi(\|\mathbf{c}_0 - \mathbf{c}_0\|) & \cdots & \varphi(\|\mathbf{c}_0 - \mathbf{c}_{N-1}\| \\ \vdots & \ddots & \vdots \\ \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_0\|) & \cdots & \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_{N-1}\|) \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_{N-1} \end{pmatrix} = \begin{pmatrix} d_0 \\ \vdots \\ d_{N-1} \end{pmatrix}$$

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

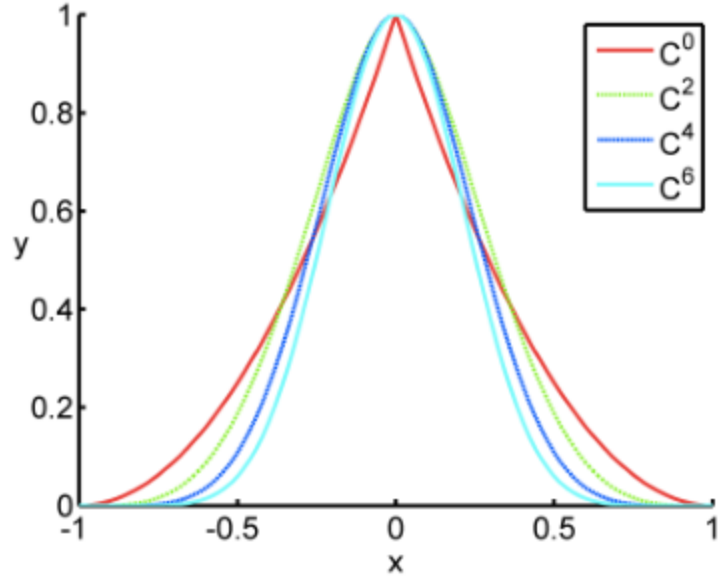
- Polyharmonic spline
  - $\varphi(r) = r^k \log(r),\ \ k = 2, 4, 6 \ldots$
  - $\varphi(r) = r^k,\ \ k = 1, 3, 5 \ldots$

- Multiquadratic

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

- Gaussian

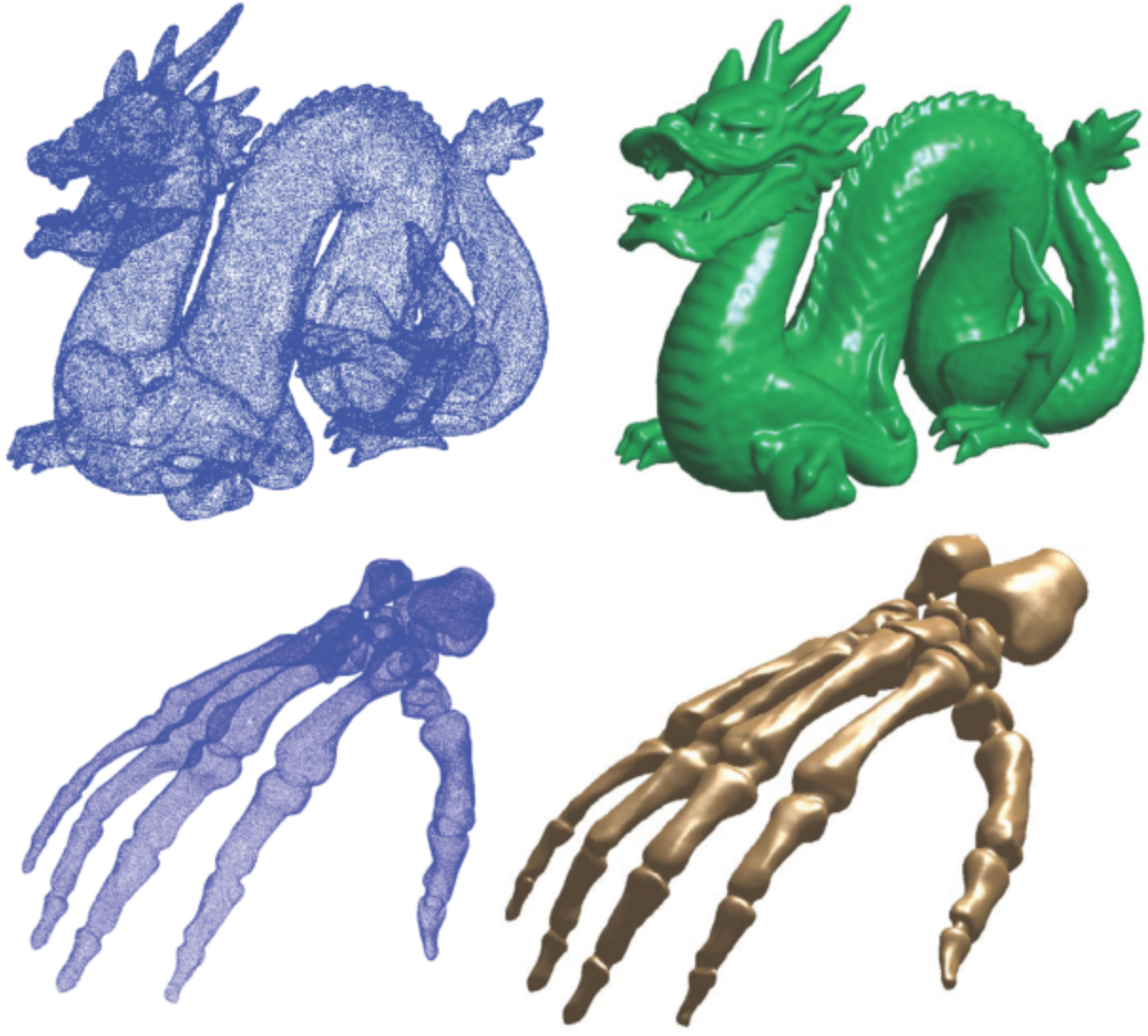
$$\varphi(r) = e^{-\beta r^2}$$



- 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

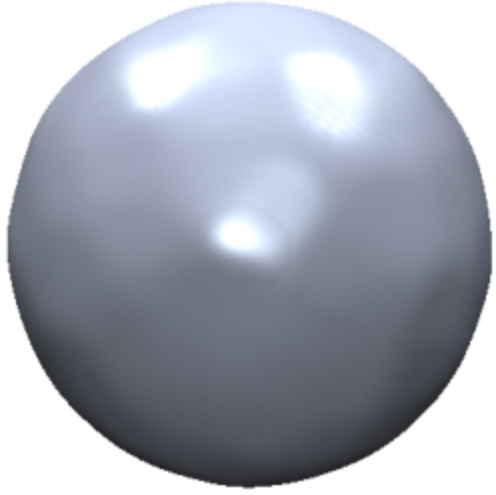
Insufficient number/
badly placed off-surface points

Properly chosen off-surface points

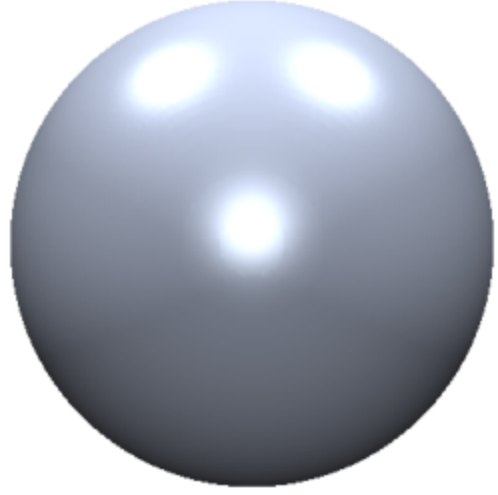
"Reconstruction and representation of 3D objects with radial basis functions", Carr et al., ACM SIGGRAPH 2001

Comparison of the various SDFs so far

Distance
to plane



Compact RBF



Global RBF
Triharmonic

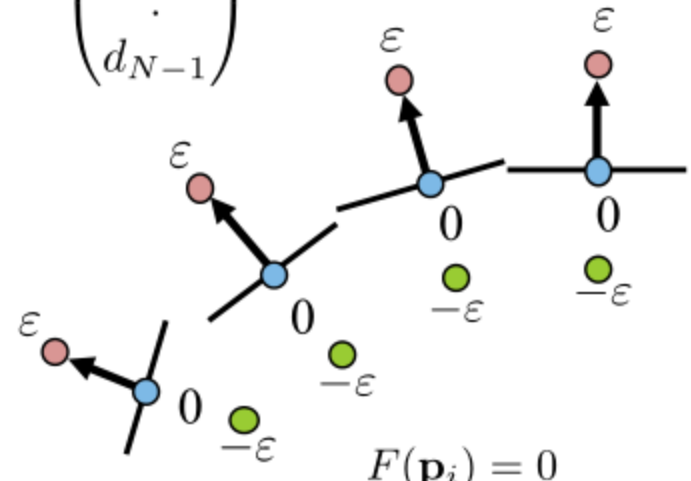
RBF - Discussion

- Global definition! $F(\mathbf{x}) = \sum_{i=0}^{N-1} w_i \, \varphi(\|\mathbf{x} - \mathbf{c}_i\|)$

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

$$\begin{pmatrix} \varphi(\|\mathbf{c}_0 - \mathbf{c}_0\|) & \cdots & \varphi(\|\mathbf{c}_0 - \mathbf{c}_{N-1}\| \\ \vdots & \ddots & \vdots \\ \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_0\|) & \cdots & \varphi(\|\mathbf{c}_{N-1} - \mathbf{c}_{N-1}\|) \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_{N-1} \end{pmatrix} = \begin{pmatrix} d_0 \\ \vdots \\ d_{N-1} \end{pmatrix}$$

- Global optimization of weights.
  - Even if the basis functions are local!

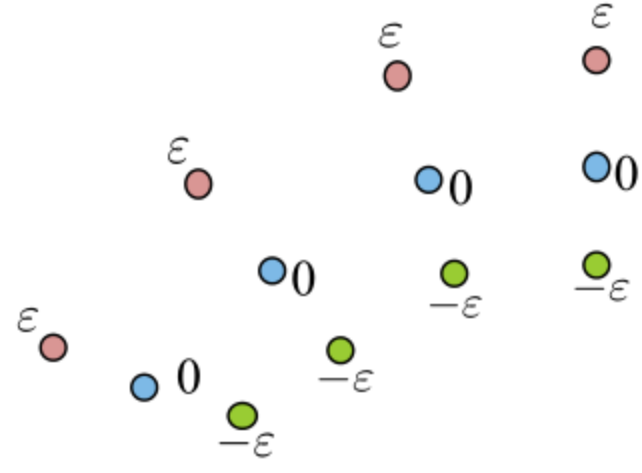


$F(\mathbf{p}_i) = 0$
$F(\mathbf{p}_i + \varepsilon\mathbf{n}_i) = \varepsilon$
$F(\mathbf{p}_i - \varepsilon\mathbf{n}_i) = -\varepsilon$

Moving Least Squares (MLS)

- Approximate the SDF **locally.**
- Weights vary according to position.

- **Advantage:** the blending of all local approximations as one function $F(\mathbf{x})$, is smooth everywhere!
  - **Globally** smooth function with **local** computation.

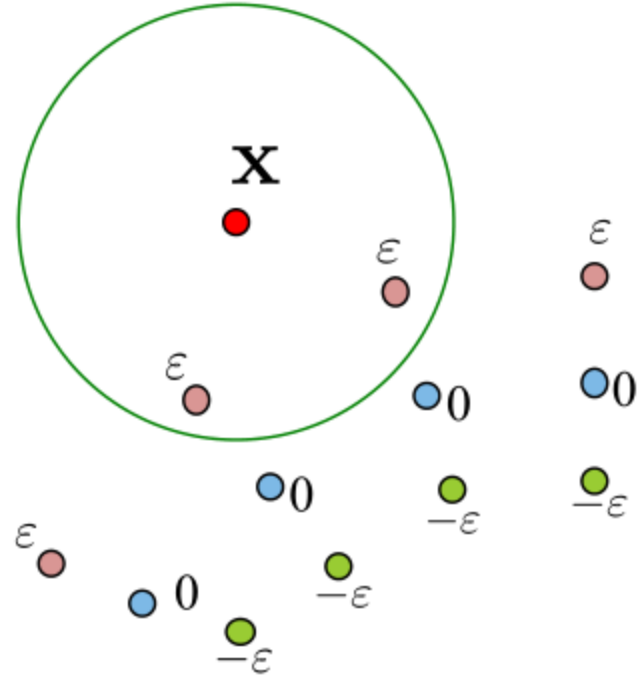


"Interpolating and Approximating Implicit Surfaces from Polygon Soup", Shen et al., ACM SIGGRAPH 2004
http://graphics.berkeley.edu/papers/Shen-IAI-2004-08/index.html

Moving Least Squares (MLS)

- Approximate the SDF **locally.**
- Weights vary according to position.

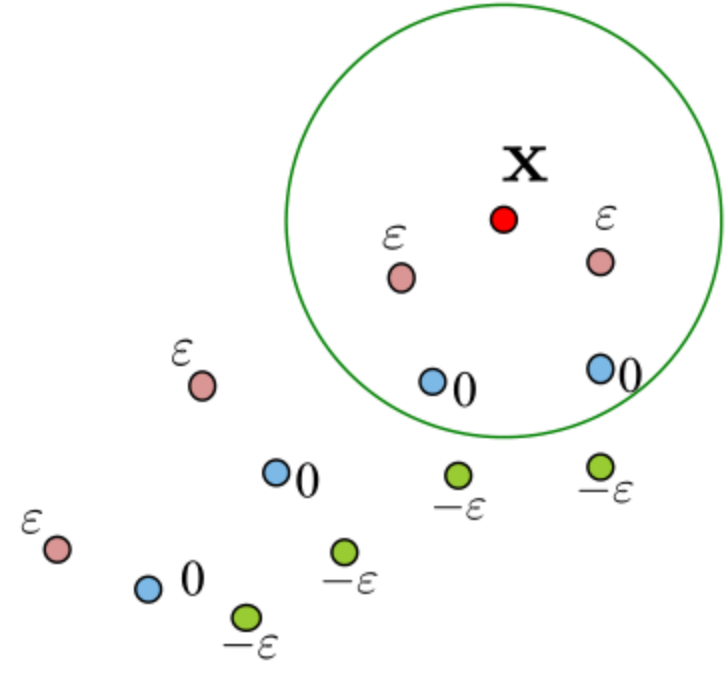
- **Advantage:** the blending of all local approximations as one function $F(\mathbf{x})$, is smooth everywhere!
  - **Globally** smooth function with **local** computation.



Moving Least Squares (MLS)

- Approximate the SDF **locally.**
- Weights vary according to position.

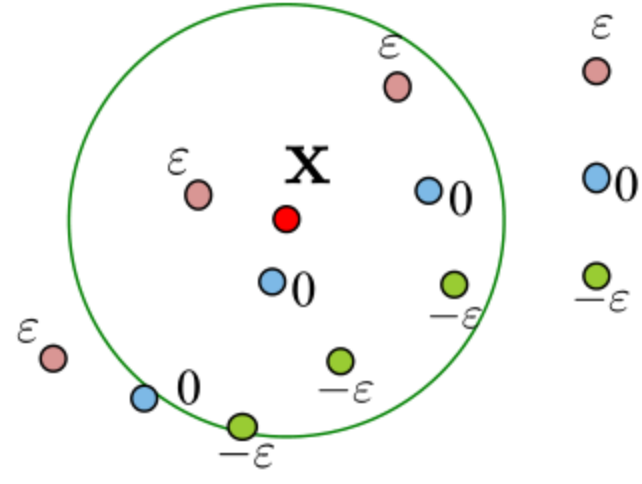
- **Advantage:** the blending of all local approximations as one function $F(\mathbf{x})$, is smooth everywhere!
  - **Globally** smooth function with **local** computation.



Moving Least Squares (MLS)

- Approximate the SDF **locally.**
- Weights vary according to position.

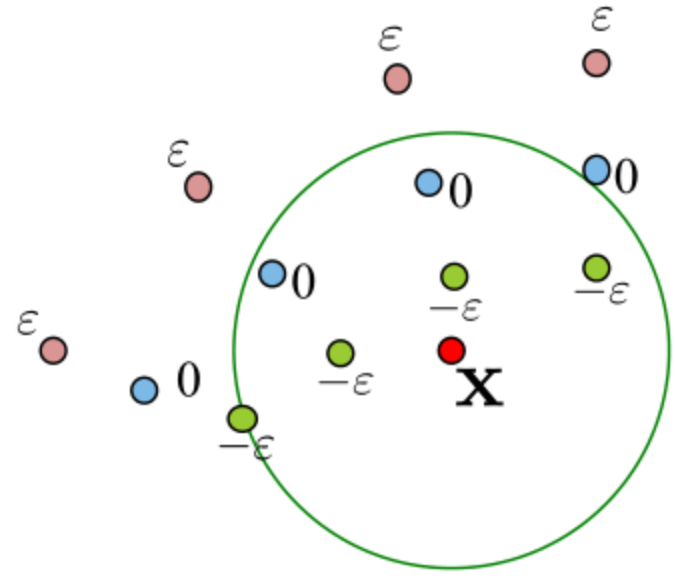
- **Advantage:** the blending of all local approximations as one function $F(\mathbf{x})$, is smooth everywhere!
  - **Globally** smooth function with **local** computation.



Moving Least Squares (MLS)

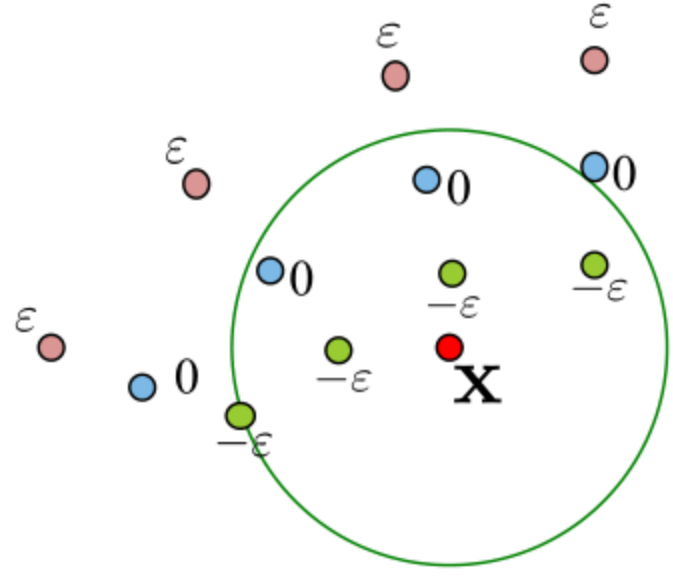
- Approximate the SDF **locally.**
- Weights vary according to position.

- **Advantage:** the blending of all local approximations as one function $F(\mathbf{x})$, is smooth everywhere!
  - **Globally** smooth function with **local** computation.



Moving Least Squares (MLS)

- Essence of algorithm:
  - Approximating function $F(x)$ by a local polynomial $f(x)$.
  - Estimate $p(x)$ on-the-fly for each point $x \in \mathbb{R}^3$.
  - As in RBF: interpolates the points as-much-as-possible.
  - $f(x)$ is chosen as the best polynomial approximating F_i.
  - Approximation cares more about points close to x.



Approximating Polynomial

- Using 3D polynomial of degree *k*
  - Or: $f \in \Pi_k^3$.

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

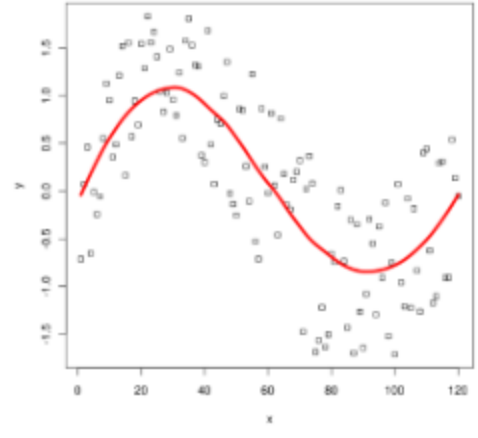
- In vector notation:

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

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

Weighted Polynomial Regression

- Input: sample set $\{d_i\}, 0 \leq i \leq N-1$.
- Importance weights per sample: θ_i.
- Output: poly. Coefficients a_i.
- Best approximation:



$$a = argmin \sum_{0}^{N-1} (f(c_i) - d_i)^2$$

Constant per sample

Our variables

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

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

MOVING Least-Squares Approximation

- Moving least squares: weights depend on position!

$$f_{\mathbf{x}} = \underset{f \in \Pi_k^3}{\operatorname{argmin}} \sum_{i=0}^{N-1} \theta(\|\mathbf{x} - \mathbf{c}_i\|) \left(f(\mathbf{c}_i) - d_i\right)^2 \text{ or:}$$

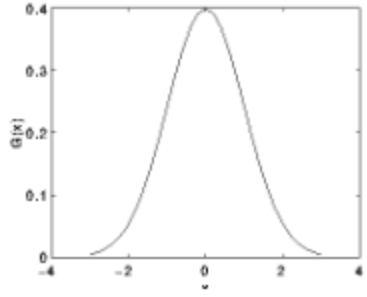
$$\mathbf{a}_{\mathbf{x}} = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{i=0}^{N-1} \theta(\|\mathbf{x} - \mathbf{c}_i\|) \left(\mathbf{b}(\mathbf{c}_i)^T \mathbf{a} - d_i\right)^2$$

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

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

Weight Functions

- Gaussian $\theta(r) = e^{-\frac{r^2}{h^2}}$
  - h is a smoothing parameter



- Wendland function $\theta(r) = (1 - r/h)^4(4r/h + 1)$
  - Defined in $[0, h]$ and

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

- Singular function

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

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

"Argmin"? Linear Least Squares

- Minimize an energy of the sort:

$$r = argmin(A_{n\times m} r_n - \mathrm{b}_m)^2$$

  - Matrix form of our equations.

- We need: $\frac{d(Ar-b)^2}{dr} = 0$ (critical point).

$$(Ar-b)^2 = (Ar-b)^T(Ar-b) =$$

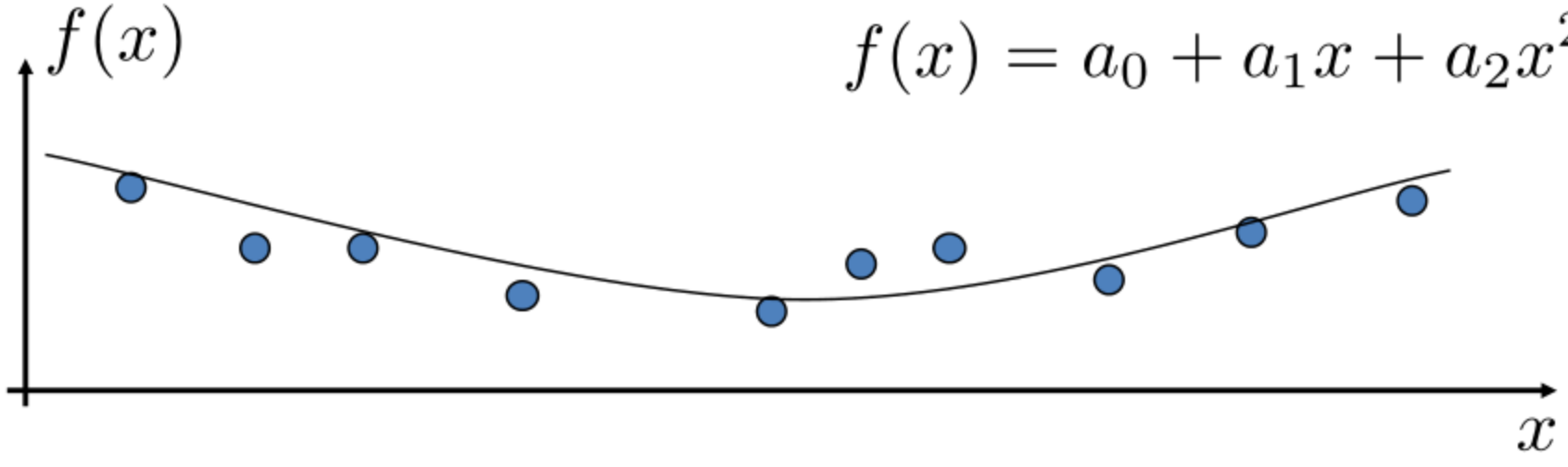
$$r^T A^T Ar - 2rA^T b + const.$$

$$\frac{d(Ar-b)^2}{dr} = \boxed{2A^T Ar - 2A^T b} = 0$$

Linear equation in *r*!

MLS – 1D Example

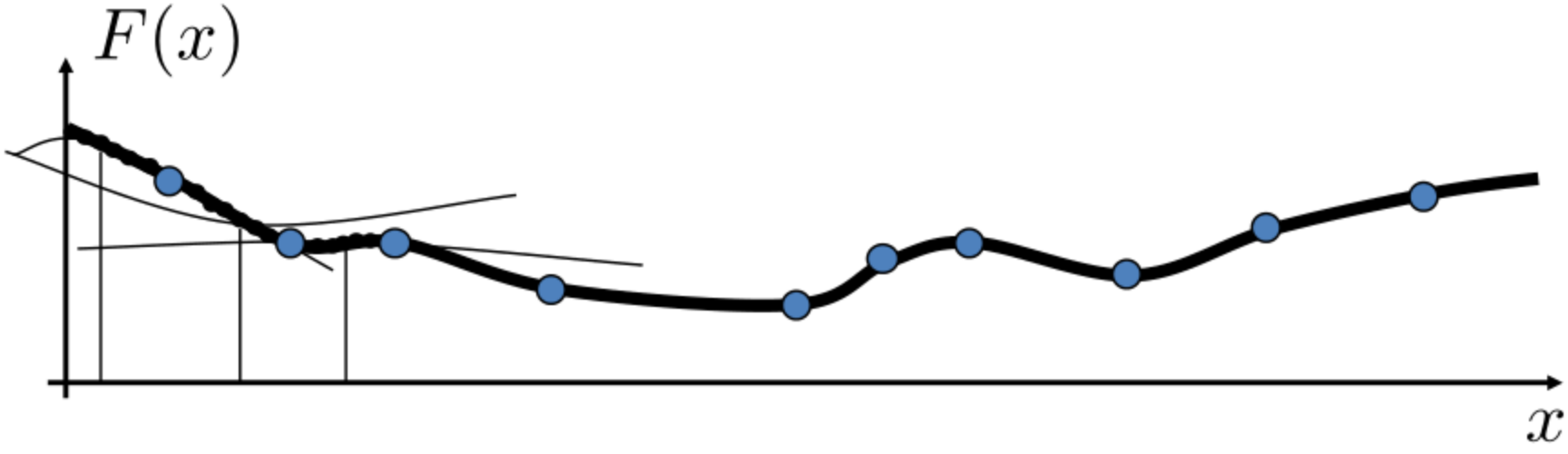
- Global approximation in Π_2^1



$$f = \underset{f \in \Pi_2^1}{\operatorname{argmin}} \sum_{i=0}^{N-1} \left(f(c_i) - d_i \right)^2$$

MLS – 1D Example

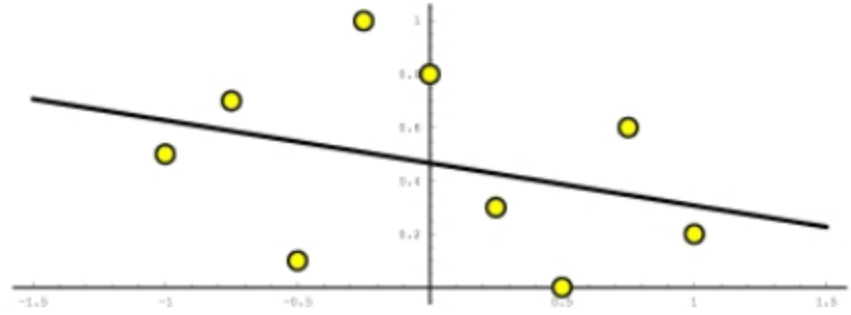
- MLS approximation using functions in Π_2^1



$$F(x) = f_x(x), \quad f_x = \underset{f \in \Pi_2^1}{\operatorname{argmin}} \sum_{i=0}^{N-1} \theta(\|c_i - x\|) \left(f(c_i) - d_i\right)^2$$

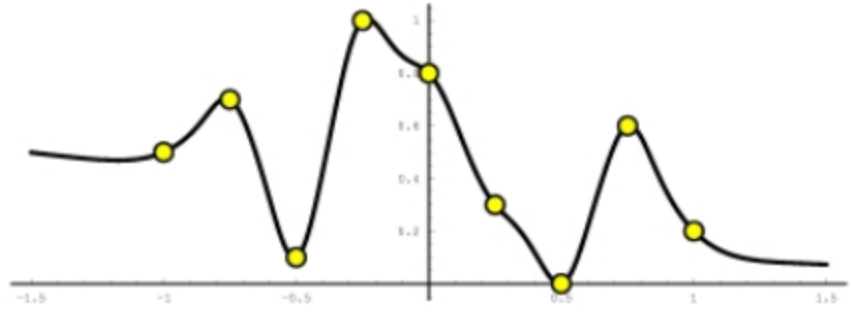
Dependence on Weight Function

- Global least squares with linear basis



- MLS with (nearly) singular weight function

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



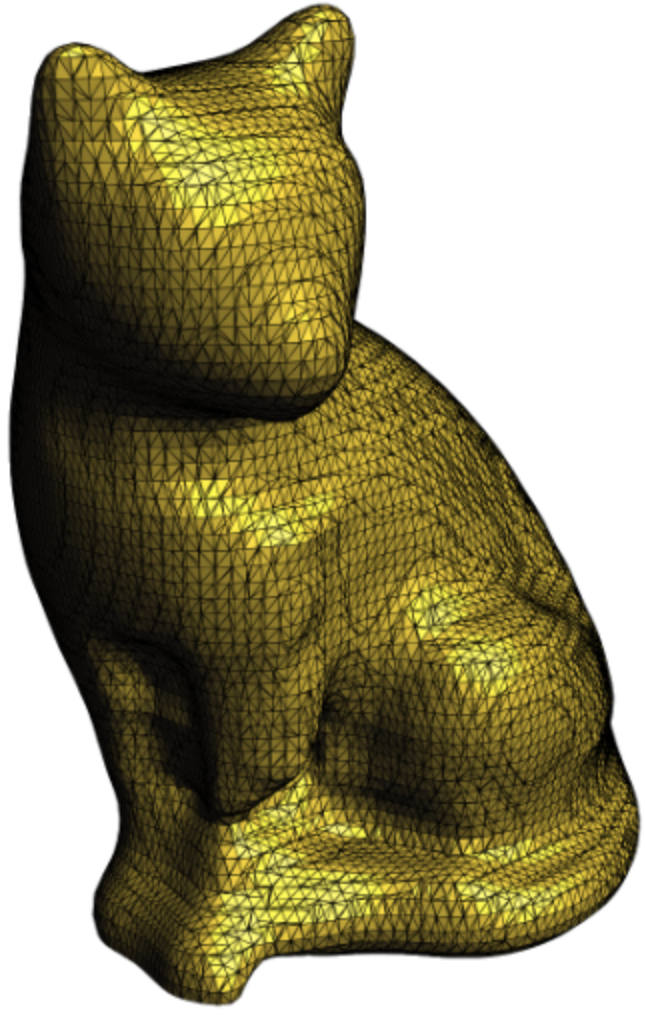
- MLS with approximating weight function $\theta(r) = e^{-\frac{r^2}{h^2}}$

Dependence on Weight Function

- The MLS function F is continuously differentiable iff the weight function θ is continuously differentiable.
- In general, F is as smooth as θ.

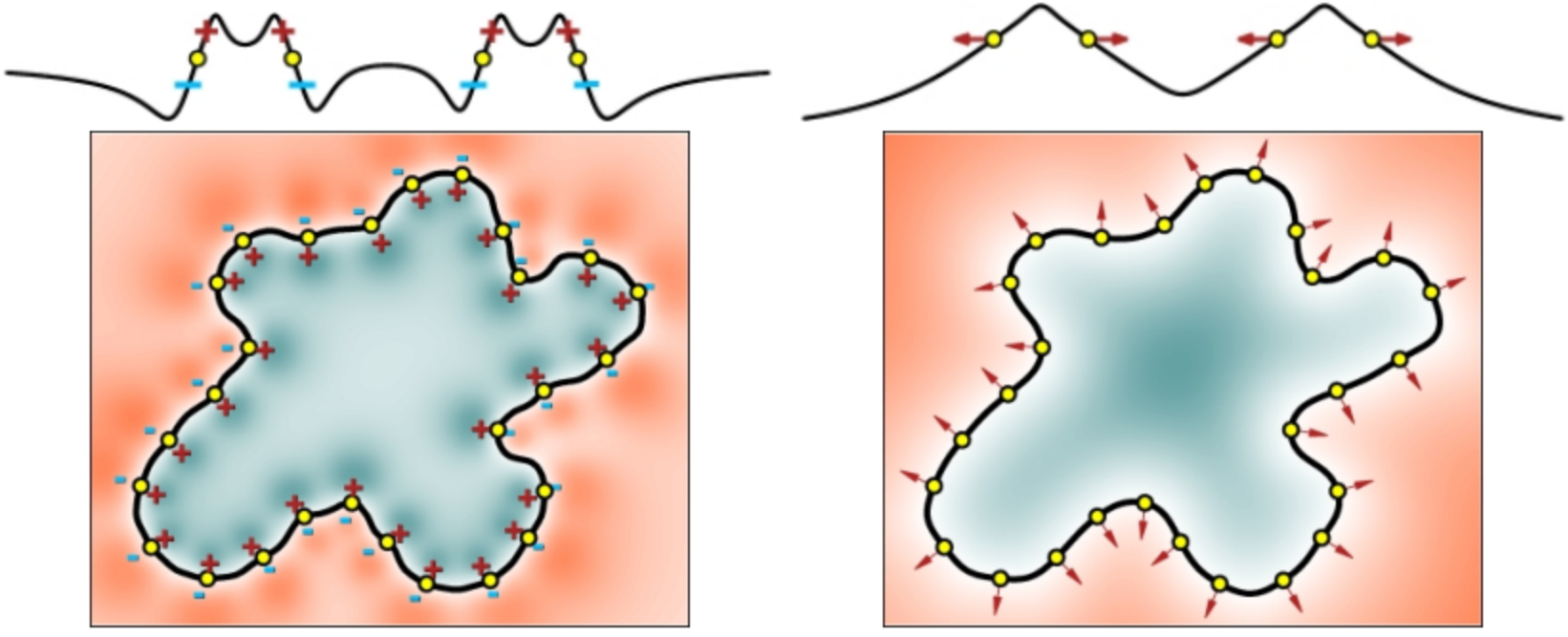
$$F(\mathbf{x}) = f_{\mathbf{x}}(\mathbf{x}), \quad f_{\mathbf{x}} = \underset{f \in \Pi_k^d}{\operatorname{argmin}} \sum_{i=0}^{N-1} \theta(\|\mathbf{c}_i - \mathbf{x}\|) \left(f(\mathbf{c}_i) - d_i\right)^2$$

Example: Reconstruction



MLS SDF – Possible Improvement

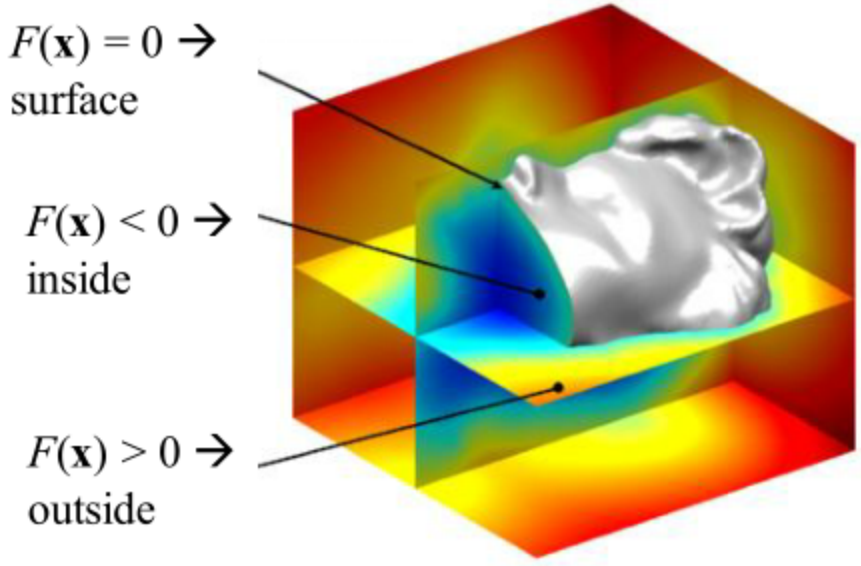
- Point constraints vs. true normal constraints



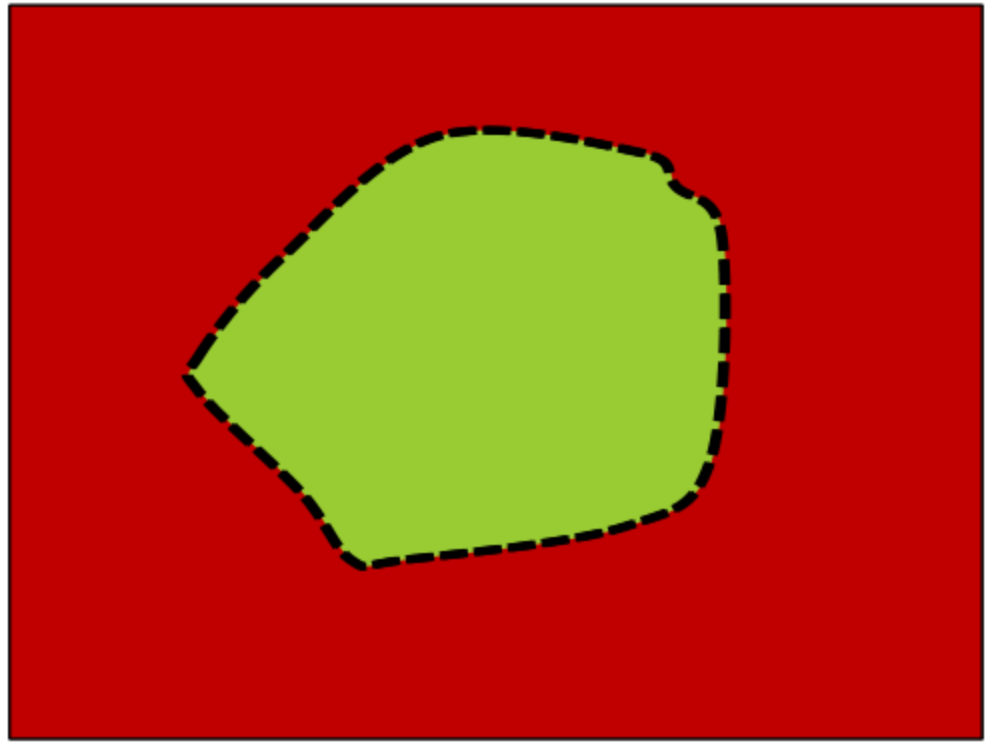
- Details: see [Shen et al. SIGGRAPH 2004]
  - …and the assignment in Ex1.

Extracting the Surface

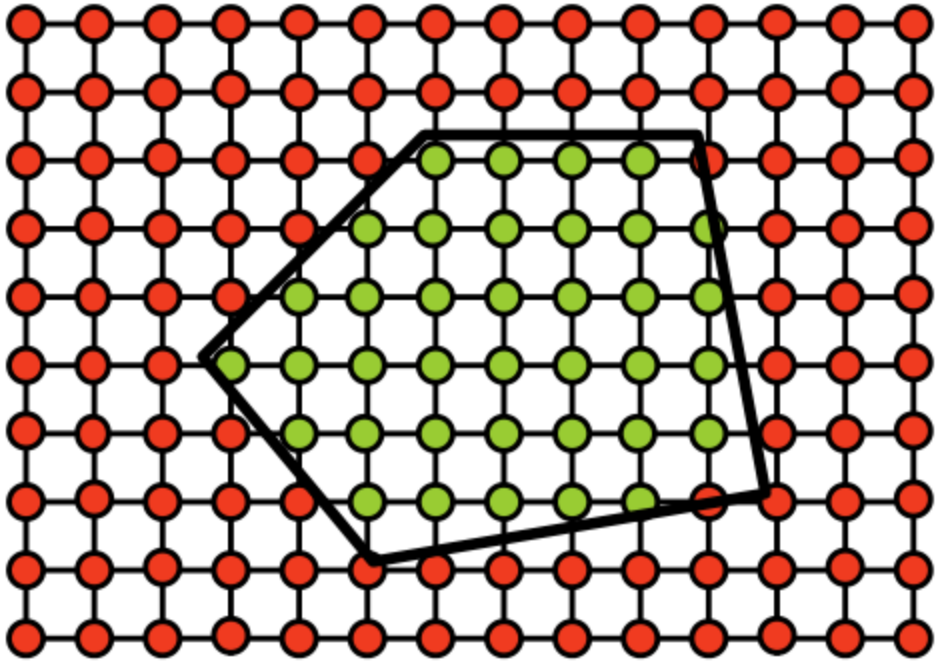
- How do we make a manifold mesh of the SDF?



Sample the SDF

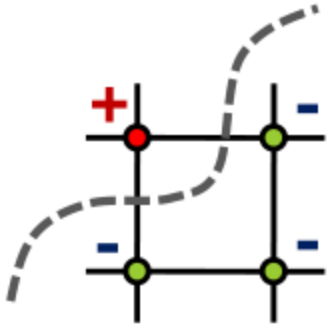


Sample the SDF

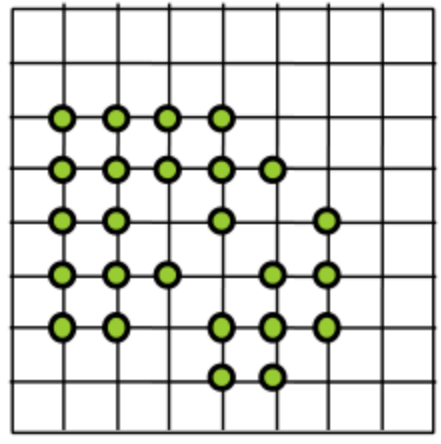


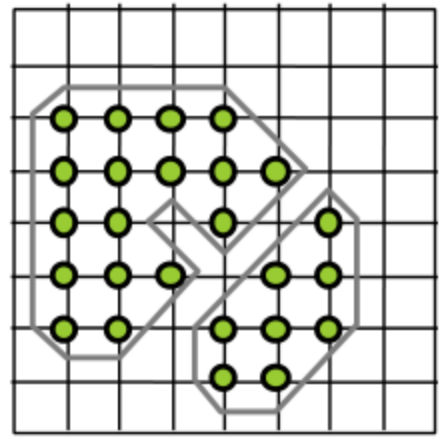
Tessellation

- Idea: sample and assume function is linear within the sampled grid.
- Zero set passes only where different signs exist in a grid cell.



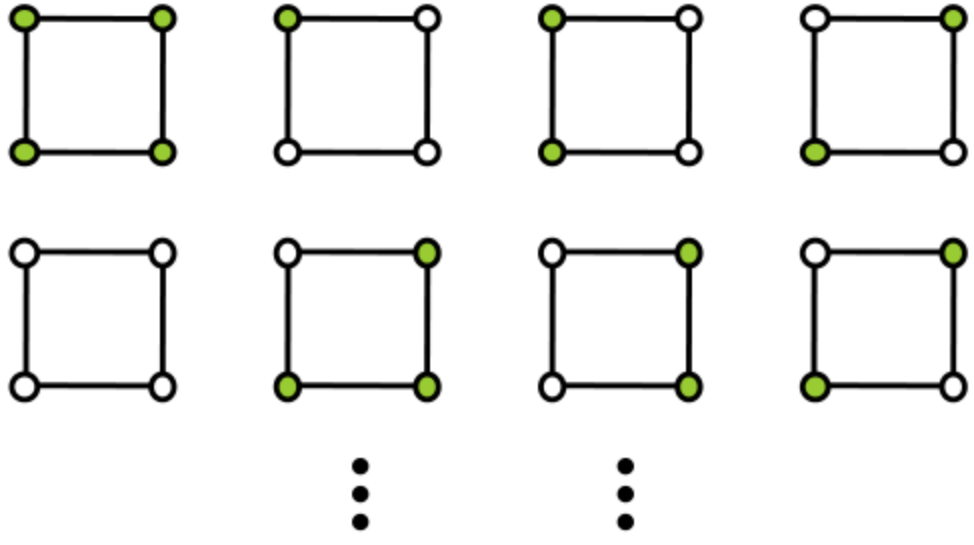
$F(\mathbf{x}) < 0$





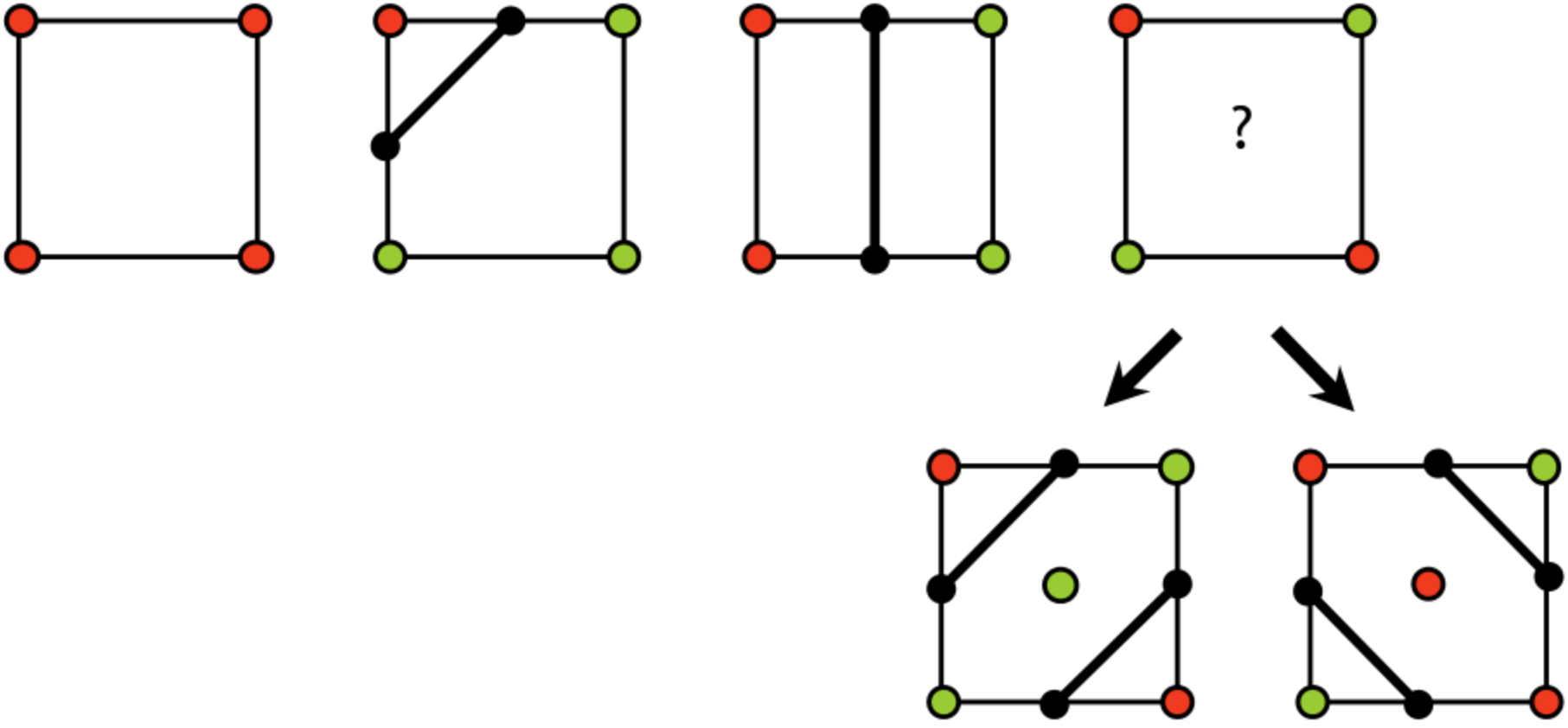
2D Marching Squares

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



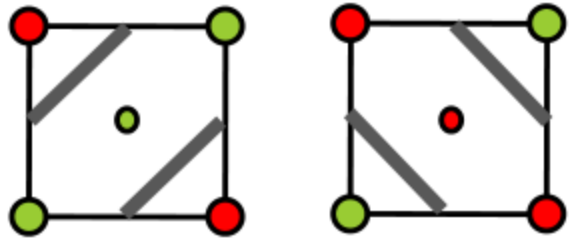
Tessellation in 2D

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

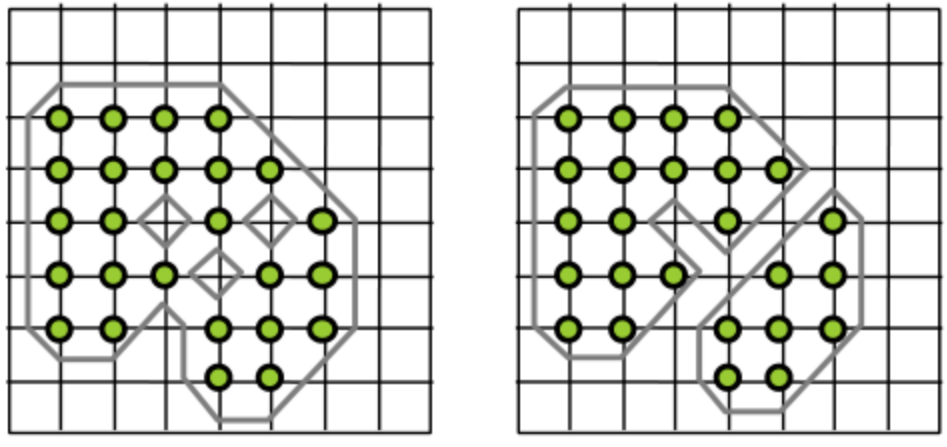


Tessellation in 2D

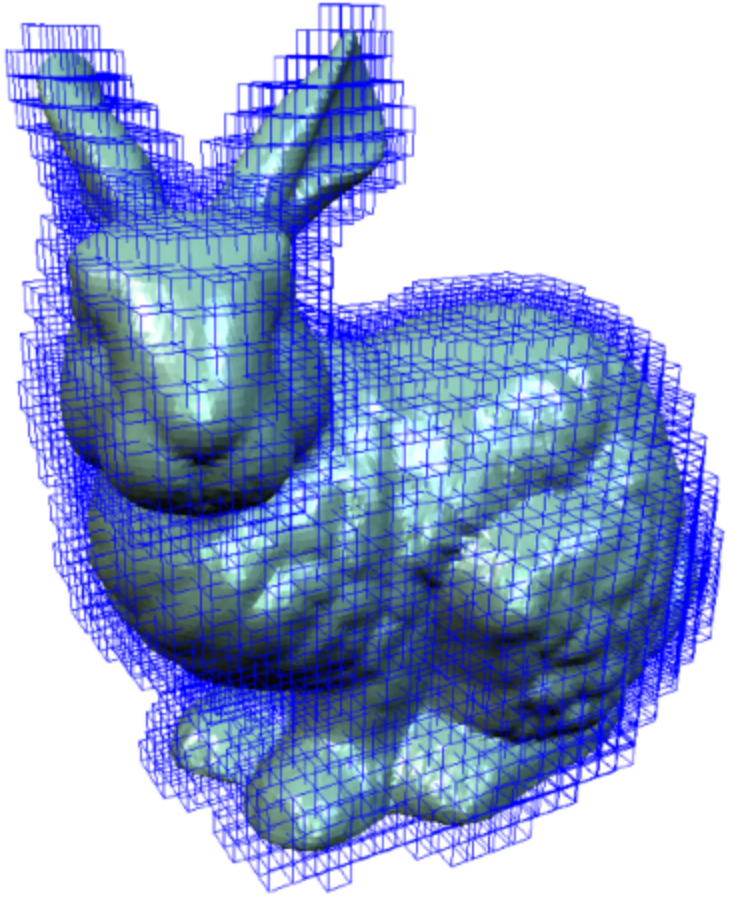
- Case 4 is ambiguious:

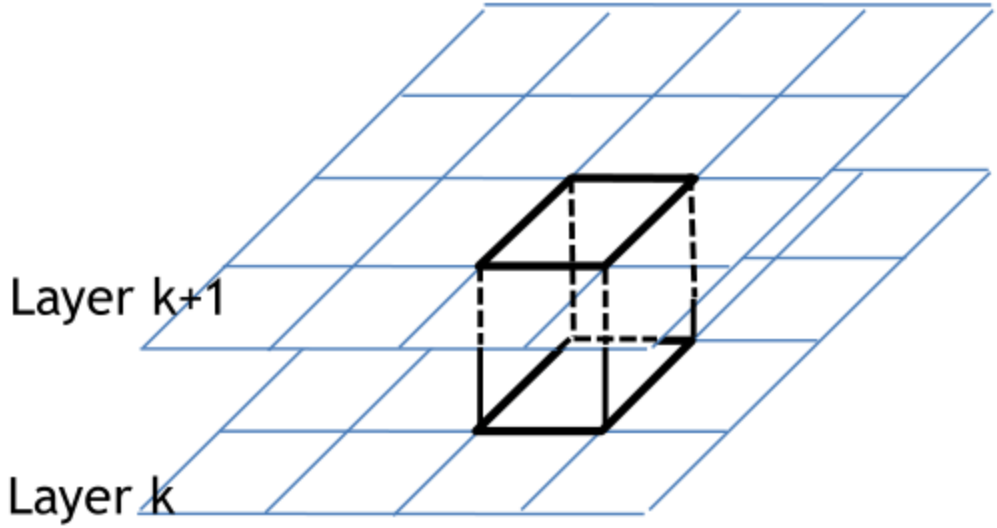


- Choose arbitrarilty; but consistently.
- To avoid problems with the resulting mesh.



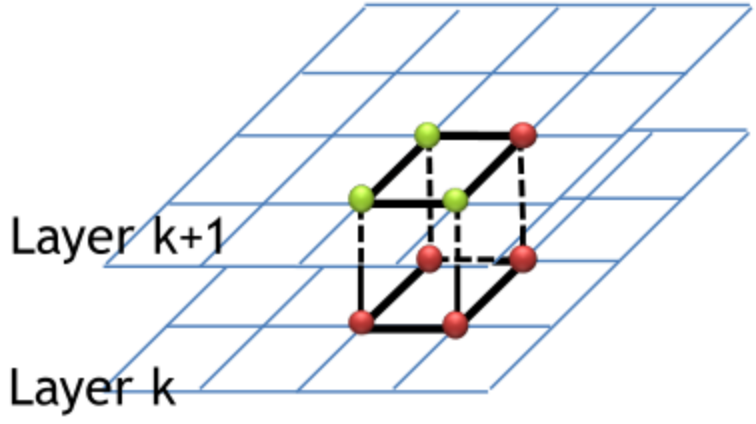
3D: Marching Cubes





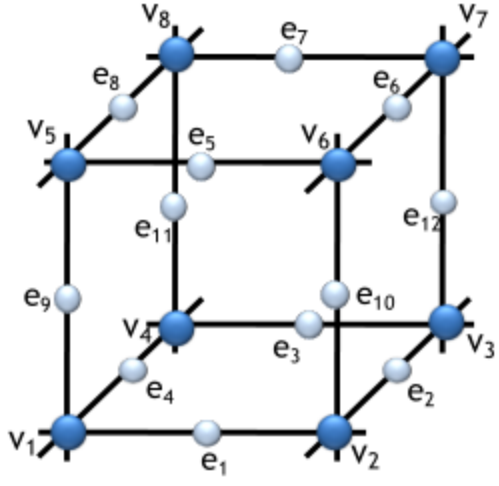
Marching Cubes

- 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).

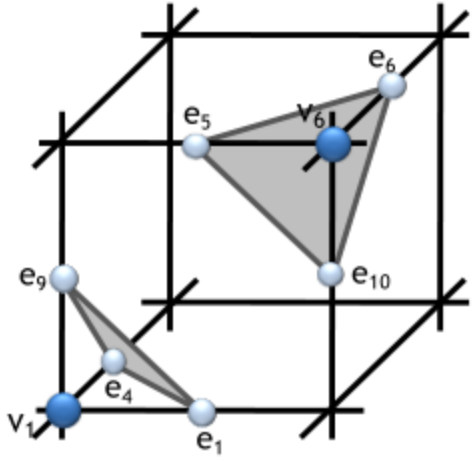


Marching Cubes

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.



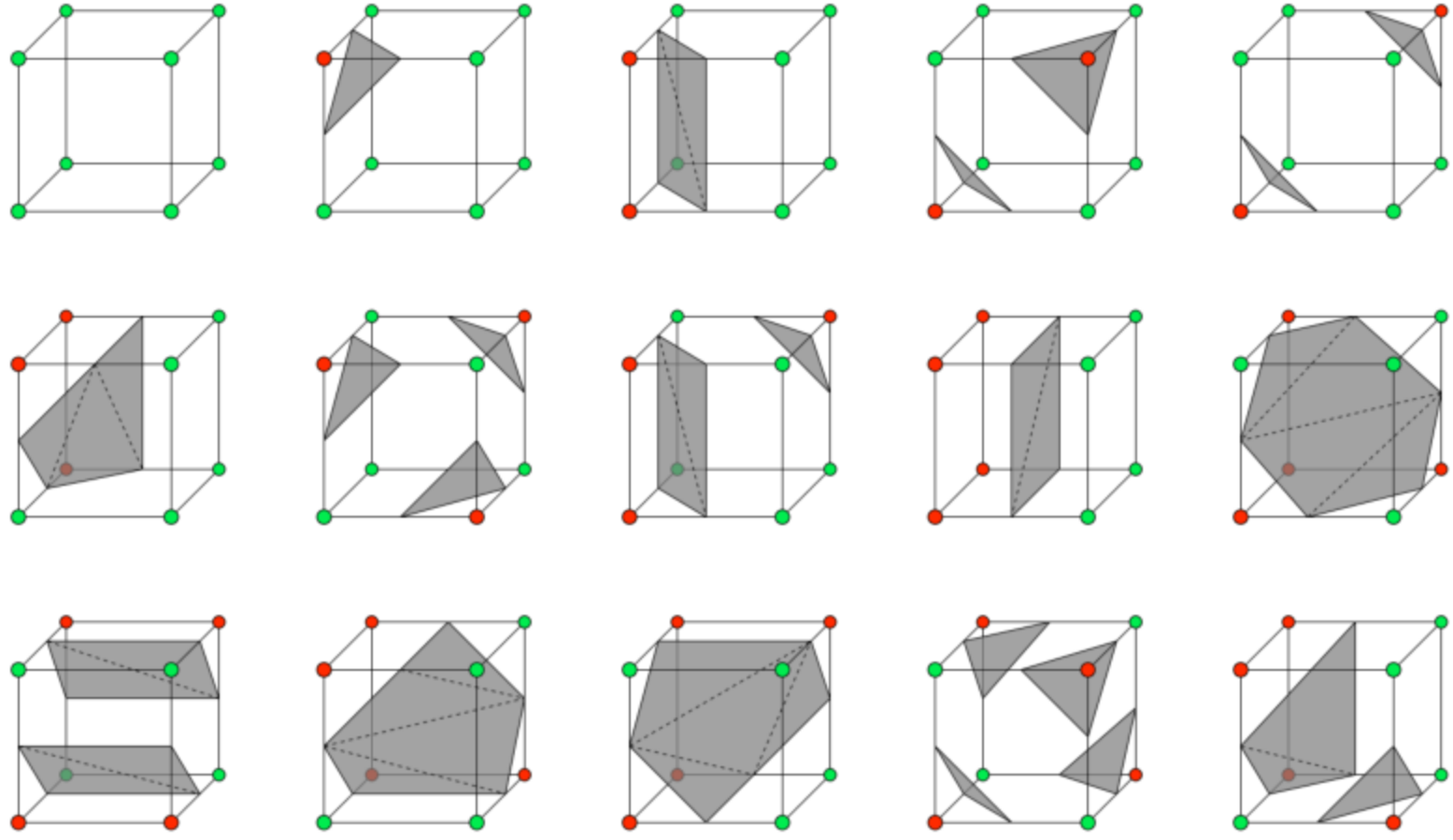
index = | v_1 | v_2 | v_3 | v_4 | v_5 | v_6 | v_7 | v_8 |



index = | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | = 33

Marching Cubes

- Unique cases (by rotation, reflection and complement)

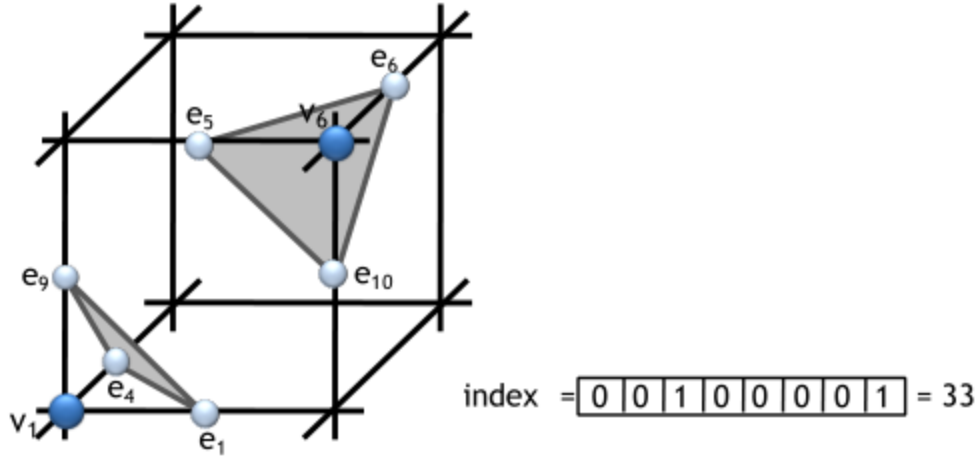


Tessellation

3D – Marching Cubes

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

- Example: the entry for index 33 indicates:
  - Cut edges are e_1; e_4; e_5; e_6; e_9 and e_{10} ;
  - Output triangles are (e_1; e_9; e_4) and (e_5; e_{10}; e_6).



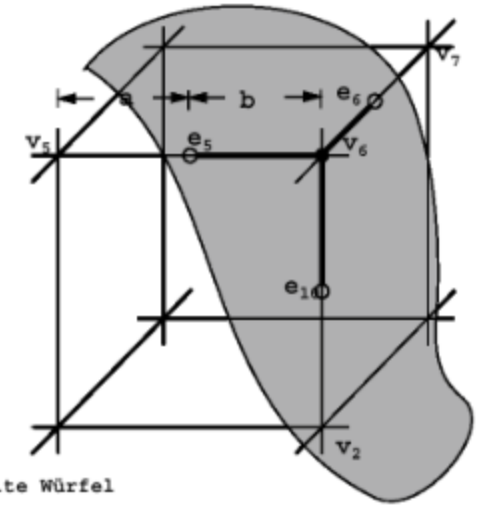
Marching Cubes

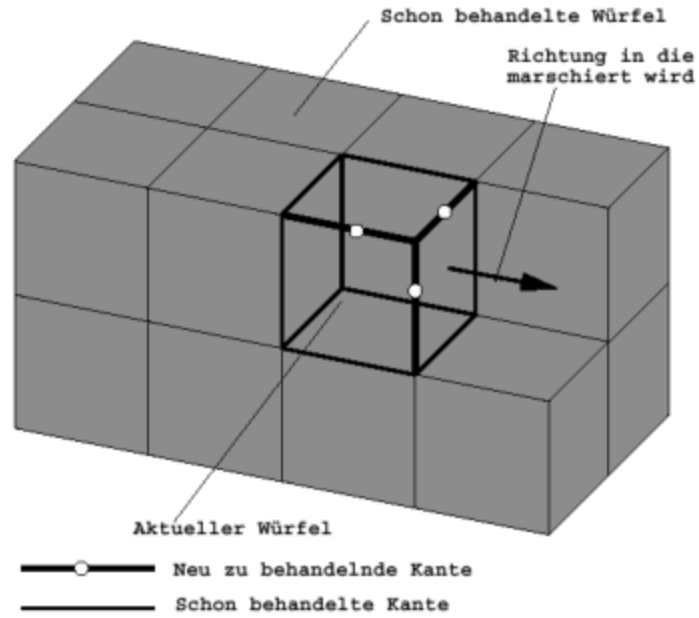
6. Compute the position of the cut vertices by linear interpolation:

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

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

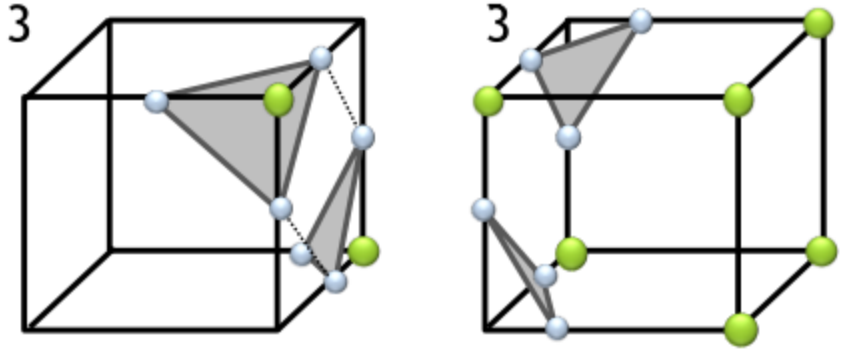
7. Move to the next cube

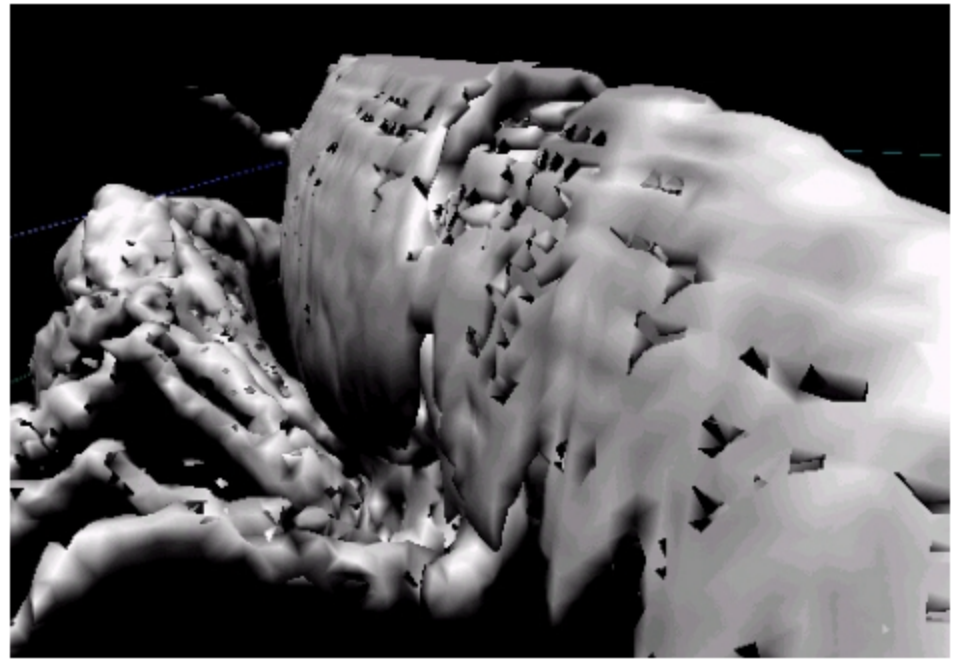




Marching Cubes – Problems

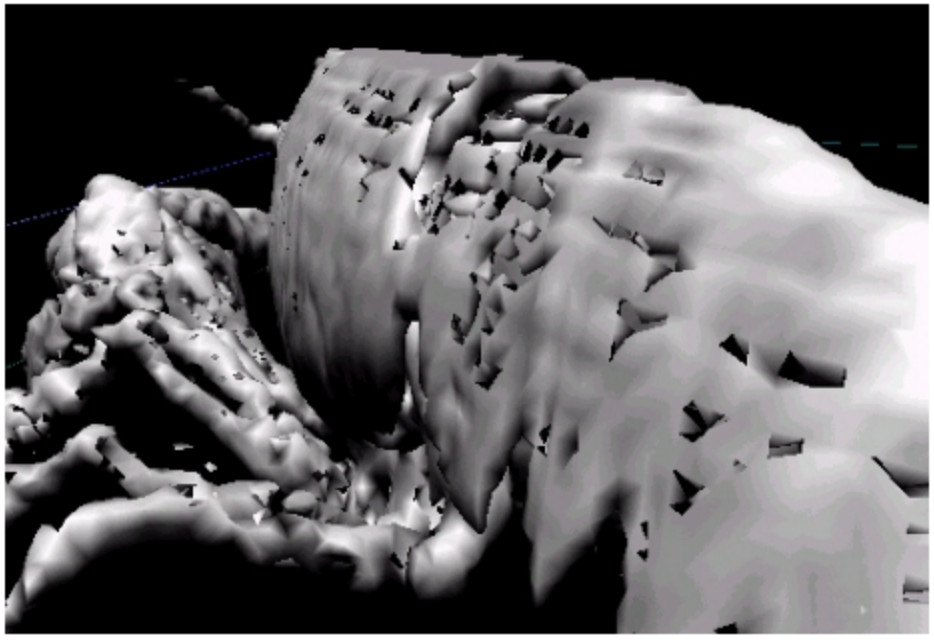
- Inconsistent choices for neighboring cubes ➔ holes.



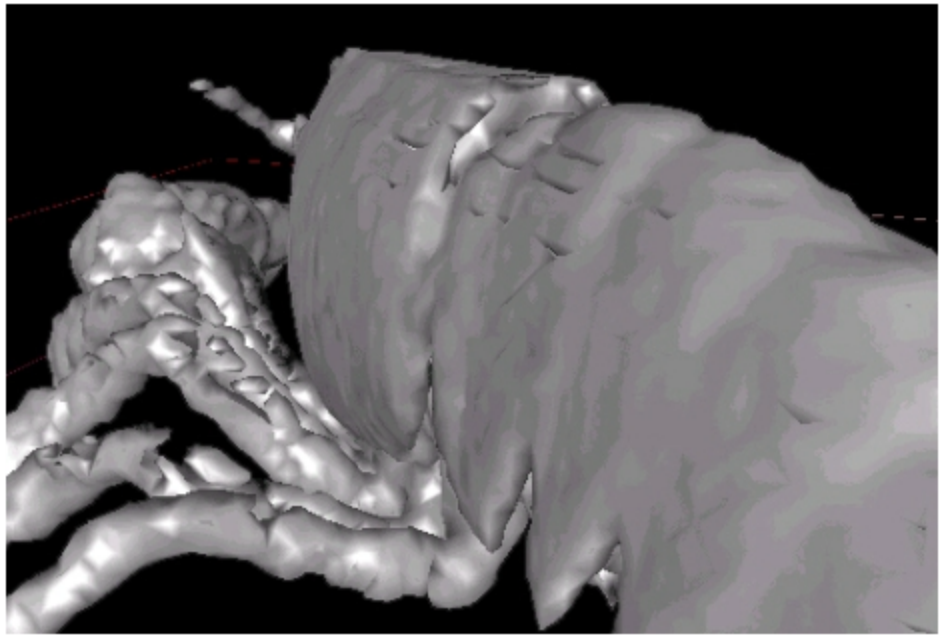


Marching Cubes – Problems

- Resolving ambiguities



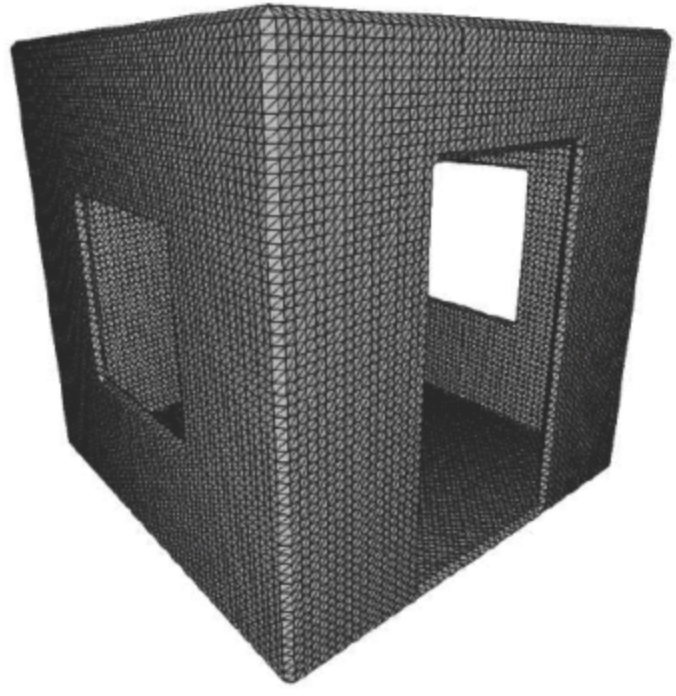
Ambiguity



No Ambiguity

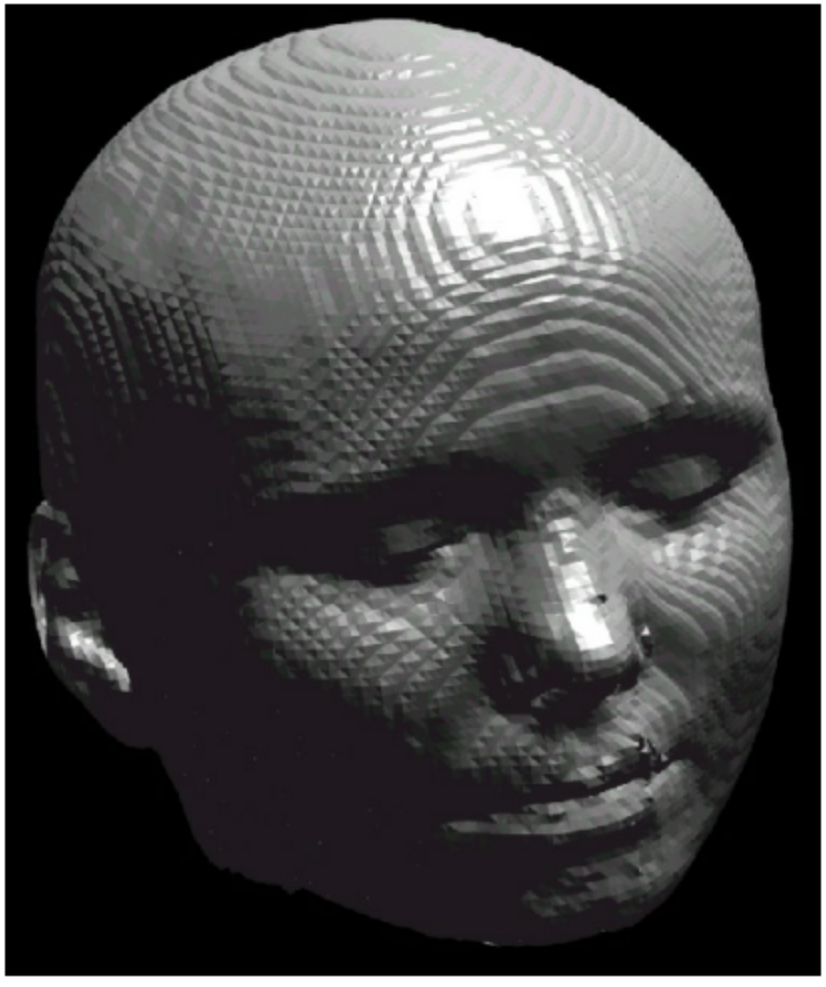
Marching Cubes – Disadvantages

- Grid not adaptive.
- Many polygons are used regardless of the geometric feature size.



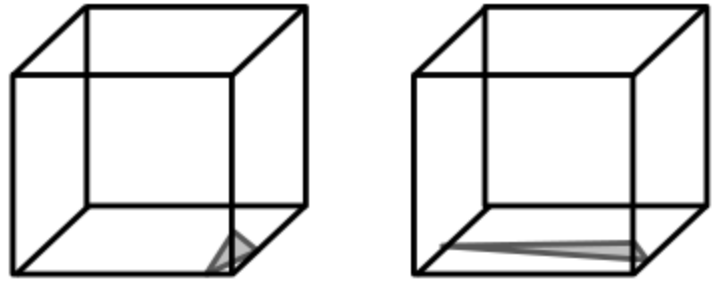
Images from: "Dual Marching Cubes: Primal Contouring of Dual Grids" by Schaeffer et al.

Marching Cubes – Disadvantages



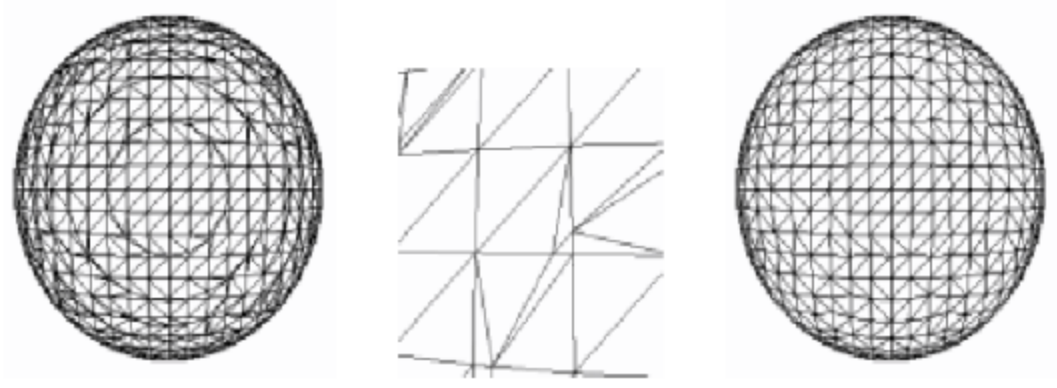
Marching Cubes – Problems

- Grid sampling can cause bad triangle edges.
  - Surface intersects the cube close to a corner ➔ small triangles.
  - Intersection is close to an edge of the cube ➔ „skinny“ triangles (bad aspect ratio).
- Triangles with short edges waste resources and do not contribute much.



Grid Snapping

- Solution: threshold the distances between the created vertices and the cube corners.
- Distance is smaller than d_{snap} ➔ snap vertex to the cube corner.
- More than one vertex of a triangle is snapped to the same point ➔ discard triangle.



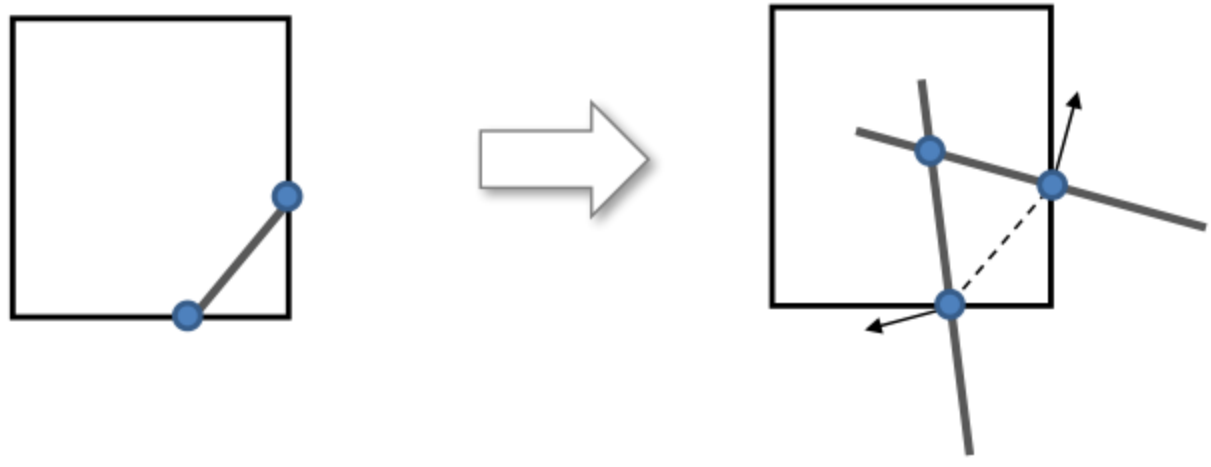
Grid Snapping

- Obtaining significant reduction of space consumption.

Parameter	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. 2001):
  - Evaluate the normals (use gradient of F).
  - Normals differ significantly ➔ create additional vertex.



Global RBF vs. Local MLS

- RBF:
  - Uses entire data set.
    - Possibly very smooth surfaces.
  - Global (dense) system to solve – expensive.
- MLS:
  - Sees only a small part of the dataset.
    - Outliers/noise can have a bigger effect.
  - Local linear solves – cheap.

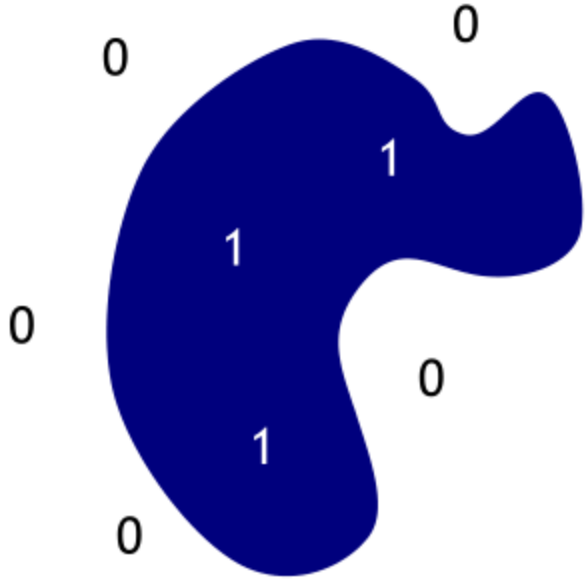
Poisson Surface Reconstruction

- [M. Kazhdan, M. Bolitho and H. Hoppe, SGP 2006] http://www.cs.jhu.edu/~misha/Code/PoissonRecon/
- Popular modern method.
- Code available!
- Global fitting of an *indicator function* using PDE
  - Robust to noise, sparse, computationally tractable.

- Practical comparison with MLS in the exercise.

Poisson Surface Reconstruction

Oriented points

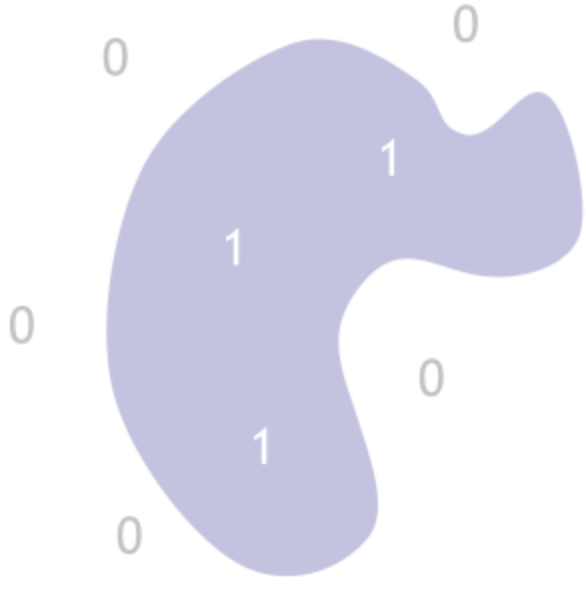


Indicator function

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

Poisson Surface Reconstruction

Oriented points



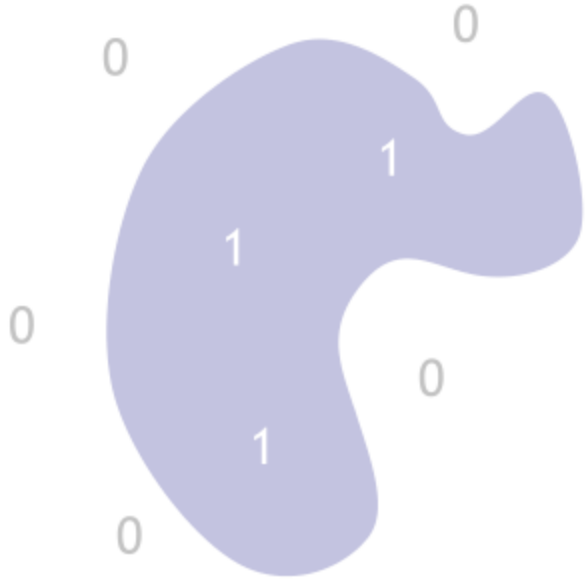
Indicator function

$\chi_{\mathcal{M}}$

We don't know the indicator function ☹

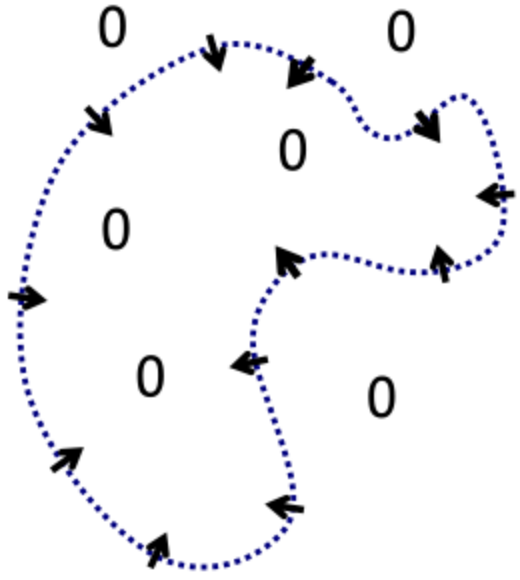
Poisson Surface Reconstruction

Oriented points



Indicator function

$\chi_{\mathcal{M}}$



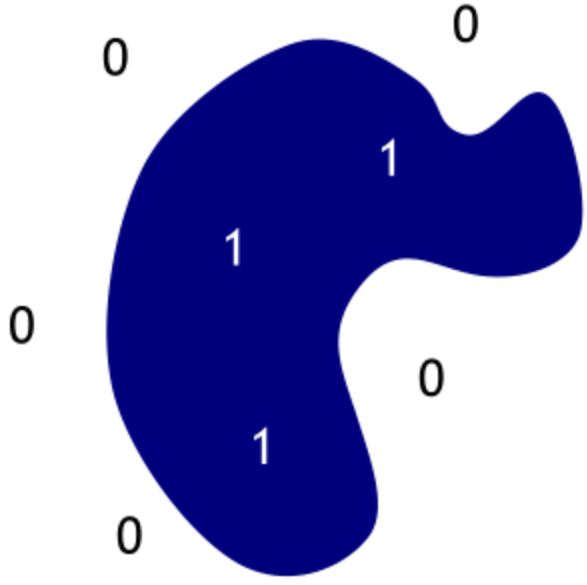
Indicator gradient

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

But we can estimate its gradient! ☺

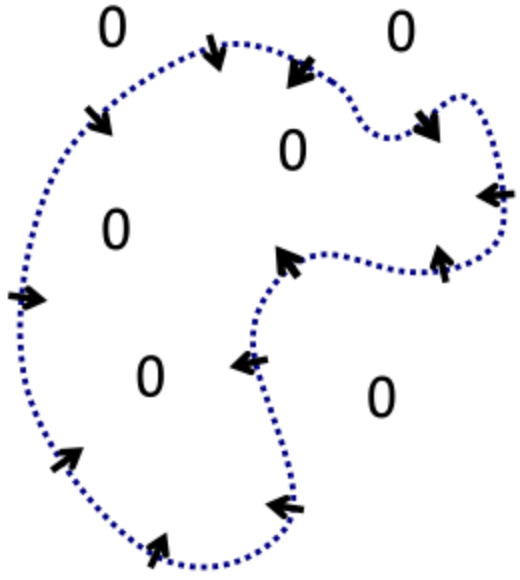
Poisson Surface Reconstruction

Oriented points



Indicator function

$\chi_{\mathcal{M}}$



Indicator gradient

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

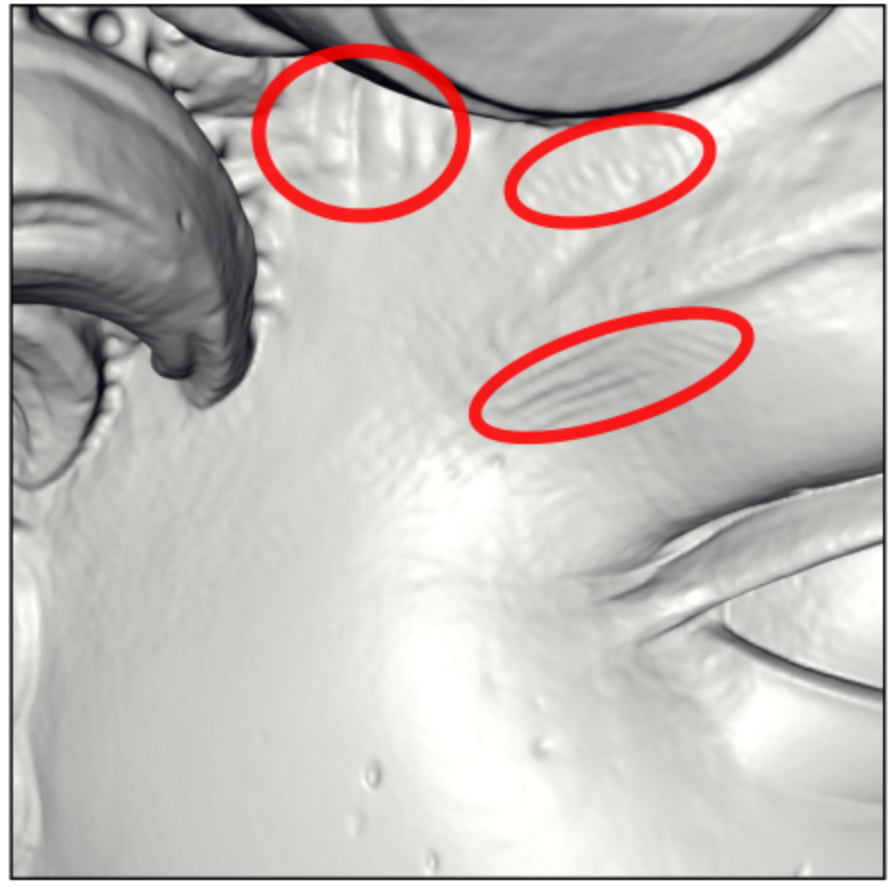
Reconstruct χ by solving the Poisson equation

$$\Delta \mathrm{X}_M = \nabla \cdot (\nabla \mathrm{X}_M)$$

Michelangelo's David

- 215 million data points from 1000 scans.
- 22 million triangle reconstruction.
- Compute Time: 2.1 hours (this was in year 2006).
- Peak Memory: 6600MB.

David – Chisel marks



David – Drill Marks