Implicit surface reconstruction with a curl-free radial basis function partition of unity method

Grady B. Wright
Boise State University

Kathryn P. Drake
Boise State University

Edward J. Fuselier
High Point University

*This work is supported by NSF grant CCF 1717556

Motivation

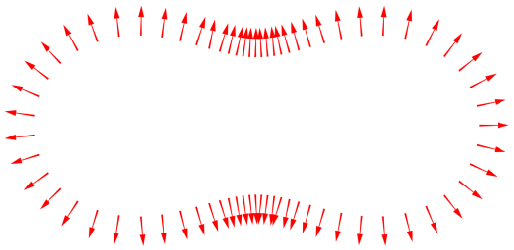
Reconstructing a surface from an (oriented) point cloud has many applications

computer graphics, computer-aided design, medical imaging, manufacturing, remote sensing

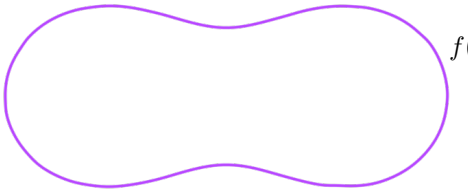
Common approach: implicit surface (zero-level set) methods

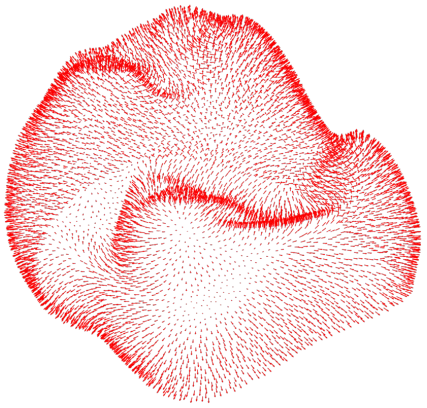
Problem: Given an oriented point cloud $\{(\mathbf{x}_j, \mathbf{n}_j)\}_{j=1}^{N}$, find a function f such that the level-set $S = \{\mathbf{x} \in \mathbb{R}^d \big| f(\mathbf{x}) = 0\}$ fits the point cloud. S is called the zero-level implicit surface.

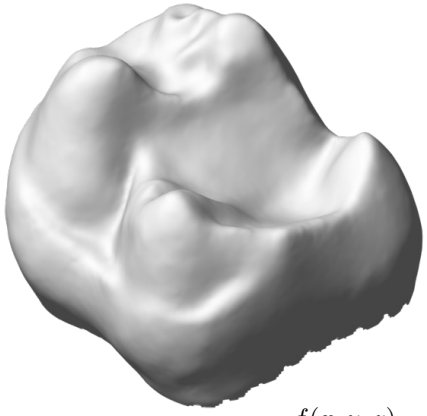
Oriented point cloud



Implicit surface







Overview of our solution

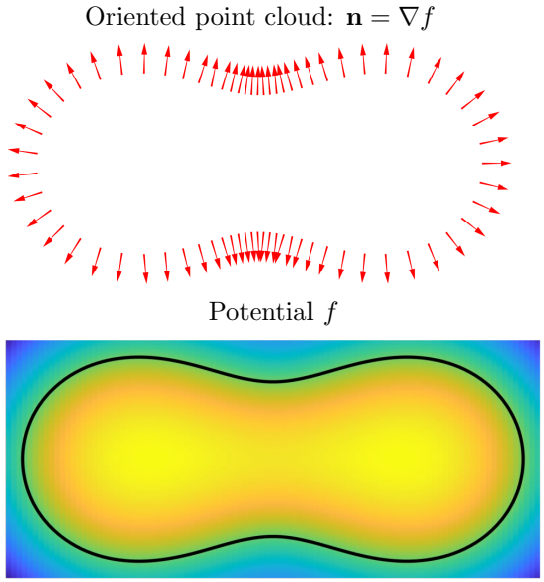
We use the following fundamental result from vector calculus

Proposition 1 *A vector field* $\mathbf{n}$ *is curl-free in* $\mathbb{R}^d$ *if and only if locally there exists a scalar potential* $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ *such that* $\mathbf{n} = \nabla f$. *Furthermore,* f *is unique up to a constant.*

Overview of our approach:

- Fit a curl-free RBF interpolant to the normal vectors
- Extract a scalar potential $\tilde{f}$ for the interpolant
- Shift the potential to be zero at the point cloud
- Include regularization if the data is noisy
- Combine the above in a partition of unity (PU) framework
- Use isosurface extractor to obtain the zero-level set

$$\Longrightarrow \text{CFPU Method}$$



Relationship to previous work

Narrow band: Carr et. al. (1997, 2001), Turk & O'Brien (1999), Morse et. al. (2001), Fasshauer (2007)
Indicator function: Kazhdan et. al. (2006, 2013, 2019), Calakli & Taubin (2011),
Hermite: Walder et. al. (2006), Süßmuth et. al. (2010), Macêdo et. al. (2011), Liu et. al. (2016),

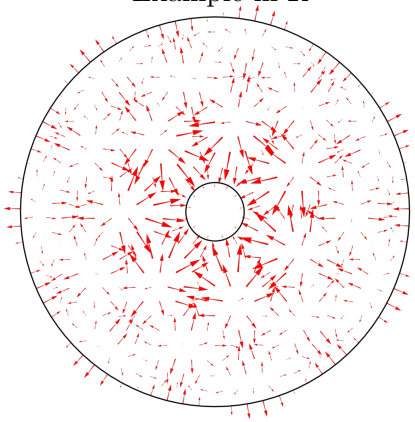
Outline

- Brief review of curl-free RBFs approximation
  - Polyharmonic splines
- Global curl-free RBF method for implicit surfaces
- Curl-free RBF PU method (CFPU)
- Numerical experiments
- Concluding remarks

Curl-free RBFs

Problem: Given scattered points $X = \{\mathbf{x}_j\}_{j=1}^N \subset \mathbb{R}^d$ and corresponding samples of a curl-free verctor field $\{\mathbf{u}\}_{j=1}^N \subset \mathbb{R}^d$, find a vector interpolant $\mathbf{s} : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ for the field such that $\nabla \times \mathbf{s} = 0$ for all $\mathbf{x} \in \mathbb{R}^d$, i.e. $\mathbf{s}$ is curl-free.

Example in $\mathbb{R}^2$



RBF-type solution

Amodei & Benbourhim (1991), Narcowich & Ward (1994), Dodut & Rabut (2004), Fuselier (2006)

Consider the following *matrix-valued kernel* $\Phi : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}^{d \times d}$

$$\Phi(\mathbf{x}, \mathbf{y}) = -\nabla\nabla^T \phi\left(\|\mathbf{x} - \mathbf{y}\|\right)$$

$\phi : [0, \infty) \to \mathbb{R}$ is a radial kernel

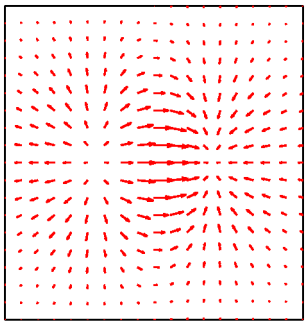
$\mathbf{y}$ is the "shift"

Claim: The columns of Φ are curl-free

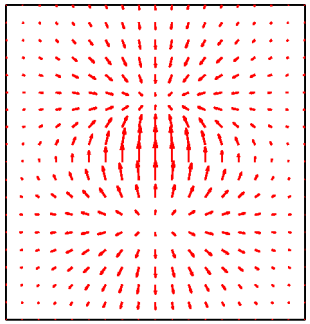
$$\implies \Phi(\mathbf{x}, \mathbf{y})\mathbf{e}_k = \nabla(\underbrace{-\nabla^T \phi(\|\mathbf{x} - \mathbf{y}\|)\mathbf{e}_k}_{g(\mathbf{x})}) = \nabla(g(\mathbf{x}))$$

Visualization d=2

$\Phi \begin{bmatrix} 1 \\ 0 \end{bmatrix}$



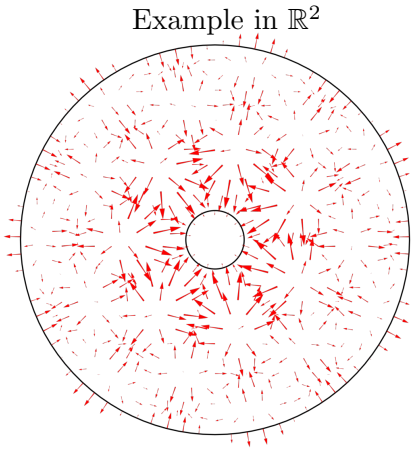
$\Phi \begin{bmatrix} 0 \\ 1 \end{bmatrix}$



We call Φ a curl-free RBF

Curl-free RBFs

Problem: Given scattered points $X = \{\mathbf{x}_j\}_{j=1}^N \subset \mathbb{R}^d$ and corresponding samples of a curl-free verctor field $\{\mathbf{u}\}_{j=1}^N \subset \mathbb{R}^d$, find a vector interpolant $\mathbf{s} : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ for the field such that $\nabla \times \mathbf{s} = 0$ for all $\mathbf{x} \in \mathbb{R}^d$, i.e. $\mathbf{s}$ is curl-free.



RBF-type solution

Amodei & Benbourhim (1991), Narcowich & Ward (1994), Dodut & Rabut (2004), Fuselier (2006)

Consider the following *matrix-valued kernel* $\Phi : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}^{d \times d}$

$$\Phi(\mathbf{x}, \mathbf{y}) = -\nabla \nabla^T \phi(\|\mathbf{x} - \mathbf{y}\|)$$

$\phi : [0, \infty) \to \mathbb{R}$ is a radial kernel

$\mathbf{y}$ is the "shift"

Curl-free RBF interpolant:

$$\mathbf{s}(\mathbf{x}) = \sum_{j=1}^{N} \Phi(\mathbf{x}, \mathbf{x}_j)\mathbf{c}_j, \quad \mathbf{s}(\mathbf{x}_k) = \mathbf{u}_k, \ k = 1, \ldots, N, \quad \mathbf{c}_j \in \mathbb{R}^d$$

dN-by-dN interpolation matrix

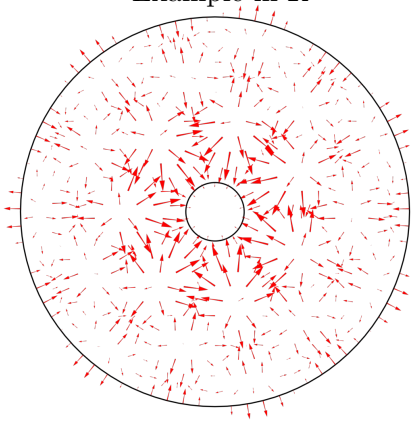
$$\underbrace{\begin{bmatrix} \Phi(\mathbf{x}_1, \mathbf{x}_1) & \Phi(\mathbf{x}_1, \mathbf{x}_2) & \cdots & \Phi(\mathbf{x}_1, \mathbf{x}_N) \\ \Phi(\mathbf{x}_2, \mathbf{x}_1) & \Phi(\mathbf{x}_2, \mathbf{x}_2) & \cdots & \Phi(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi(\mathbf{x}_N, \mathbf{x}_1) & \Phi(\mathbf{x}_N, \mathbf{x}_2) & \cdots & \Phi(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}}_{A_\Phi} \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_N \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N \end{bmatrix}$$

A_Φ is (conditionally) positive definite (depending on ϕ)

Curl-free RBFs

Problem: Given scattered points $X = \{\mathbf{x}_j\}_{j=1}^N \subset \mathbb{R}^d$ and corresponding samples of a curl-free verctor field $\{\mathbf{u}\}_{j=1}^N \subset \mathbb{R}^d$, find a vector interpolant $\mathbf{s} : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ for the field such that $\nabla \times \mathbf{s} = 0$ for all $\mathbf{x} \in \mathbb{R}^d$, i.e. $\mathbf{s}$ is curl-free.

Example in $\mathbb{R}^2$



RBF-type solution

Amodei & Benbourhim (1991), Narcowich & Ward (1994), Dodut & Rabut (2004), Fuselier (2006)

Consider the following *matrix-valued kernel* $\Phi : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}^{d \times d}$

$$\Phi(\mathbf{x}, \mathbf{y}) = -\nabla \nabla^T \phi\left(\|\mathbf{x} - \mathbf{y}\|\right)$$

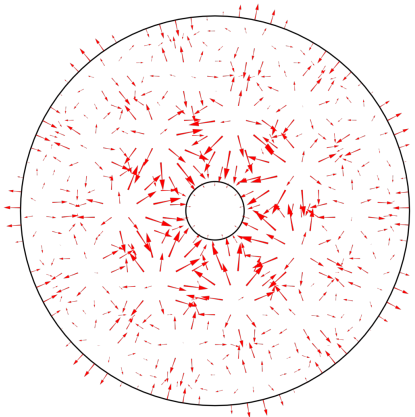
$\phi : [0, \infty) \to \mathbb{R}$ is a radial kernel

$\mathbf{y}$ is the "shift"

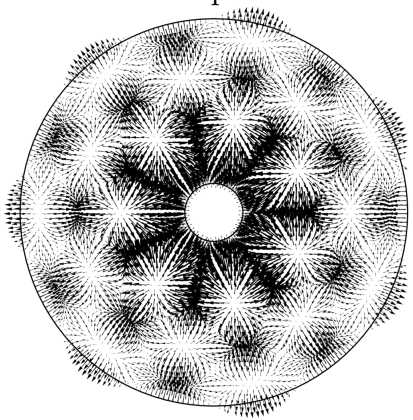
Bonus: a scalar potential for the field can also be obtained for free: Fuselier & W (2017)

$$\mathbf{s}(\mathbf{x}) = \sum_{j=1}^N \Phi(\mathbf{x}, \mathbf{x}_j)\mathbf{c}_j = \nabla\left(-\sum_{j=1}^N \nabla^T \phi\left(\|\mathbf{x} - \mathbf{x}_j\|\right)\mathbf{c}_j\right) = \nabla(s(\mathbf{x}))$$

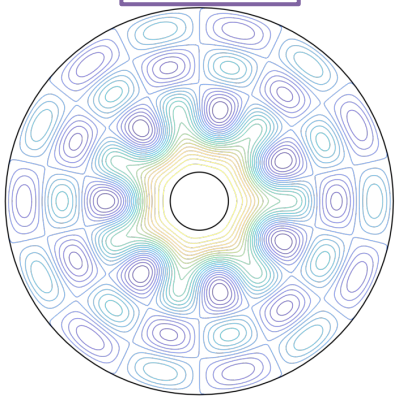
Data



Interpolant



Potential



Curl-free polyharmonic splines (PHS)

In this work, we use curl-free polyharmonic splines (PHS):

Amodei & Benbourhim (1991), Dodut & Rabut (2002), Benbourhim & Bouhamidi (2010)

$$\Phi_\ell(\mathbf{x},\mathbf{y}) = -\nabla\nabla^T \phi_\ell\left(\|\mathbf{x}-\mathbf{y}\|\right) \qquad \text{where} \qquad \phi_\ell(r) = (-1)^{\ell+1} \begin{cases} r^{2\ell}\log r, & d \text{ even},\ \ell \in \mathbb{Z}^{\geq 2}, \\ r^{2\ell+1}, & d \text{ odd},\ \ell \in \mathbb{Z}^{\geq 1} \end{cases}$$

Interpolant is augmented with *curl-free polynomials* to ensure well-posedness

$$\mathbf{s}(\mathbf{x}) = \sum_{j=1}^{N} \Phi_\ell(\mathbf{x},\mathbf{x}_j)\mathbf{c}_j + \sum_{k=1}^{L} b_k \mathbf{p}_k\left(\mathbf{x}\right)$$

Here $\{\mathbf{p}_1,\ldots,\mathbf{p}_L\}$ is a basis for curl-free polynomials of degree $\ell-1$

Example $\ell = 2$

$$\text{Basis in } \mathbb{R}^2: \quad \left\{ \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}y\\x\end{bmatrix}, \begin{bmatrix}x\\0\end{bmatrix}, \begin{bmatrix}0\\y\end{bmatrix} \right\}$$

$$\text{Basis in } \mathbb{R}^3: \quad \left\{ \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}y\\x\\0\end{bmatrix}, \begin{bmatrix}z\\0\\x\end{bmatrix}, \begin{bmatrix}0\\z\\y\end{bmatrix}, \begin{bmatrix}x\\0\\0\end{bmatrix}, \begin{bmatrix}0\\y\\0\end{bmatrix}, \begin{bmatrix}0\\0\\z\end{bmatrix} \right\} \qquad (\mathbf{p}_k = \nabla p_k)$$

Curl-free polyharmonic splines (PHS)

In this work, we use curl-free polyharmonic splines (PHS):

Amodei & Benbourhim (1991), Dodut & Rabut (2002), Benbourhim & Bouhamidi (2010)

$$\Phi_\ell(\mathbf{x},\mathbf{y}) = -\nabla\nabla^T \phi_\ell\left(\|\mathbf{x}-\mathbf{y}\|\right) \qquad \text{where} \qquad \phi_\ell(r) = (-1)^{\ell+1} \begin{cases} r^{2\ell}\log r, & d \text{ even},\ \ell \in \mathbb{Z}^{\geq 2}, \\ r^{2\ell+1}, & d \text{ odd},\ \ell \in \mathbb{Z}^{\geq 1} \end{cases}$$

Interpolant is augmented with *curl-free polynomials* to ensure well-posedness

$$\mathbf{s}(\mathbf{x}) = \sum_{j=1}^{N} \Phi_\ell(\mathbf{x},\mathbf{x}_j)\mathbf{c}_j + \sum_{k=1}^{L} b_k \mathbf{p}_k\left(\mathbf{x}\right)$$

Here $\{\mathbf{p}_1,\ldots,\mathbf{p}_L\}$ is a basis for curl-free polynomials of degree $\ell-1$ $\quad(\mathbf{p}_k = \nabla p_k)$

Conditions to determine the coefficients $\mathbf{c}_j$ and b_k

$$\begin{aligned} \mathbf{s}(\mathbf{x}_k) &= \mathbf{u}_k,\ k=1,\ldots,N, \\ \sum_{j=1}^{N} \mathbf{c}_j^T \mathbf{p}_k(\mathbf{x}_j) &= 0,\ k=1,2,\ldots,L \end{aligned} \quad \Longrightarrow \quad \begin{bmatrix} A_\Phi & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{u} \\ \mathbf{0} \end{bmatrix}$$

Bonus: a scalar potential for the field can also be obtained for free

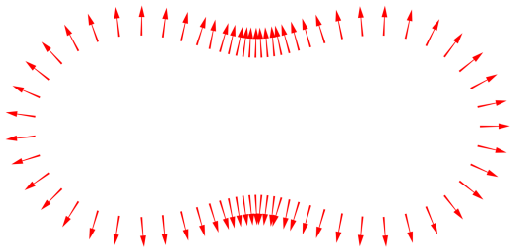
$$\mathbf{s}(\mathbf{x}) = \nabla\left(-\sum_{j=1}^{N} \nabla^T \phi_\ell\left(\|\mathbf{x}-\mathbf{x}_j\|\right)\mathbf{c}_j + \sum_{k=1}^{L} b_k p_k(\mathbf{x}) \right) = \nabla(s(\mathbf{x}))$$

Basic algorithm: global method CF method

Step 1: solve for curl-free (CF) interpolation coefficients

$$\begin{bmatrix} A_\Phi & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{n} \\ \mathbf{0} \end{bmatrix}$$

Input: oriented point cloud



$$X = \{\mathbf{x}_j\}_{j=1}^N \quad \{\mathbf{n}_j\}_{j=1}^N$$

Step 2: extract the potential for the field

$$s(\mathbf{x}) = -\sum_{j=1}^{N} \nabla^T \phi_\ell \left(\|\mathbf{x} - \mathbf{x}_j\| \right) \mathbf{c}_j + \sum_{k=1}^{L} b_k p_k(\mathbf{x})$$

Step 3: shift the potential so it's zero-level surface fits the point cloud

Idea 1: shift by the mean

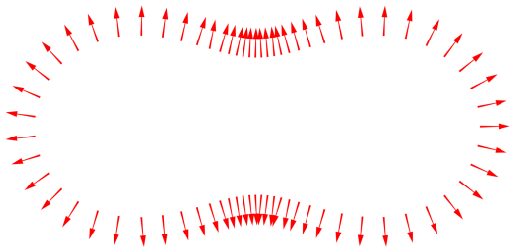
$$\overline{s}(\mathbf{x}) := s(\mathbf{x}) - \mu, \quad \text{where } \mu = \frac{1}{N} \sum_{j=1}^{N} s(\mathbf{x}_j)$$

Basic algorithm: global method CF method

Step 1: solve for curl-free (CF) interpolation coefficients

$$\begin{bmatrix} A_\Phi & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{n} \\ \mathbf{0} \end{bmatrix}$$

Input: oriented point cloud



$$X = \{\mathbf{x}_j\}_{j=1}^N \quad \{\mathbf{n}_j\}_{j=1}^N$$

Step 2: extract the potential for the field

$$s(\mathbf{x}) = -\sum_{j=1}^{N} \nabla^T \phi_\ell \left(\|\mathbf{x} - \mathbf{x}_j\| \right) \mathbf{c}_j + \sum_{k=1}^{L} b_k p_k(\mathbf{x})$$

Step 3: shift the potential so it's zero-level surface fits the point cloud

Idea 2: shift by the residual

$\widetilde{s}(\mathbf{x}) := s(\mathbf{x}) - \sigma(\mathbf{x})$, where $\sigma(\mathbf{x})$ is a scalar PHS interpolant with $\sigma(\mathbf{x}_j) = s(\mathbf{x}_j)$

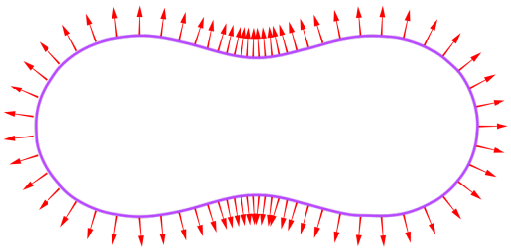
$\Longrightarrow$ The zero-level surface of $\tilde{s}$ interpolates the point cloud

Basic algorithm: global method CF method

Step 1: solve for curl-free (CF) interpolation coefficients

$$\begin{bmatrix} A_\Phi & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{n} \\ \mathbf{0} \end{bmatrix}$$

Output: implicit surface



$$X = \{\mathbf{x}_j\}_{j=1}^N \quad \{\mathbf{n}_j\}_{j=1}^N$$

Step 2: extract the potential for the field

$$s(\mathbf{x}) = -\sum_{j=1}^{N} \nabla^T \phi_\ell \left(\|\mathbf{x} - \mathbf{x}_j\| \right) \mathbf{c}_j + \sum_{k=1}^{L} b_k p_k(\mathbf{x})$$

Step 3: shift the potential so it's zero-level surface fits the point cloud

Idea 2: shift by the residual

$\widetilde{s}(\mathbf{x}) := s(\mathbf{x}) - \sigma(\mathbf{x})$, where $\sigma(\mathbf{x})$ is a scalar PHS interpolant with $\sigma(\mathbf{x}_j) = s(\mathbf{x}_j)$

$\Longrightarrow$ The zero-level surface of $\tilde{s}$ interpolates the point cloud

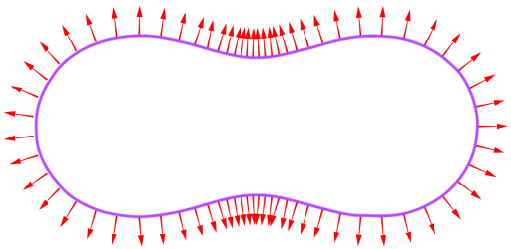
Step 4: Use isosurface extractor (e.g. marching cubes) to obtain the zero-level implicit surface

Basic algorithm: global method CF method

Step 1: solve for curl-free (CF) interpolation coefficients

$$\begin{bmatrix} A_\Phi & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{n} \\ \mathbf{0} \end{bmatrix}$$

Output: implicit surface



$$X = \{\mathbf{x}_j\}_{j=1}^N \quad \{\mathbf{n}_j\}_{j=1}^N$$

Step 2: extract the potential for the field

$$s(\mathbf{x}) = -\sum_{j=1}^{N} \nabla^T \phi_\ell \left(\|\mathbf{x} - \mathbf{x}_j\| \right) \mathbf{c}_j + \sum_{k=1}^{L} b_k p_k(\mathbf{x})$$

Step 3: shift the potential so it's zero-level surface fits the point cloud

Idea 2: shift by the residual

$\widetilde{s}(\mathbf{x}) := s(\mathbf{x}) - \sigma(\mathbf{x})$, where $\sigma(\mathbf{x})$ is a scalar PHS interpolant with $\sigma(\mathbf{x}_j) = s(\mathbf{x}_j)$

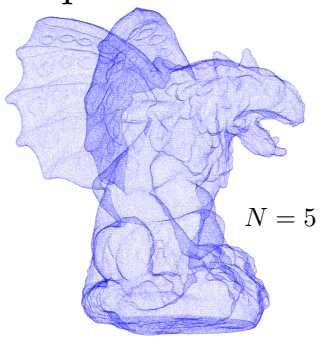
$\Longrightarrow$ The zero-level surface of $\tilde{s}$ interpolates the point cloud

Step 4: Use isosurface extractor (e.g. marching cubes) to obtain the zero-level implicit surface

Issues

Requires solving a $(dN + L)$-by-$(dN + L)$ system to find $\mathbf{c}_j$ and b_k

Each evaluation of $\tilde{s}$ requires $\mathcal{O}(dN)$ operations.



$N = 5 \times 10^5$

RBF partition of unity method (RBF-PUM)

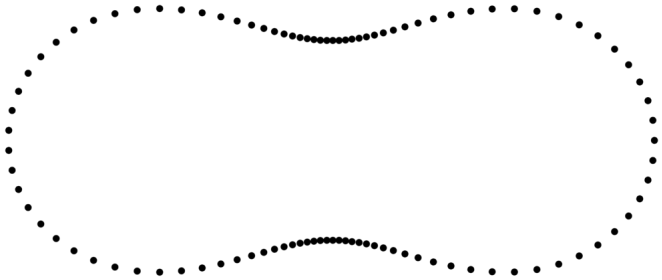
Babŭska & Melenk (1997), Wendland (2002), Lazzaro & Montefusco (2002), Tobor et. al. (2004), Ohtake et. al. (2005)

Many others: Cavoretto, De Rossi, De Marchi, Heryudono, Larsson, Perrachione, Safdari-Vaighani, Shcherbakov, Shankar, etc.

Drake, Fuselier, & Wright. A Partition of Unity Method for Divergence-free or Curl-free Radial Basis Function Approximation. *SIAM J. Sci. Comput.* 43(3), A1950-A1974 (2021)

- Cover Ω with overlapping patches Ω_i: $\Omega \subset \cup_{i=1}^{M} \Omega_i$

Domain Ω and node set X



RBF partition of unity method (RBF-PUM)

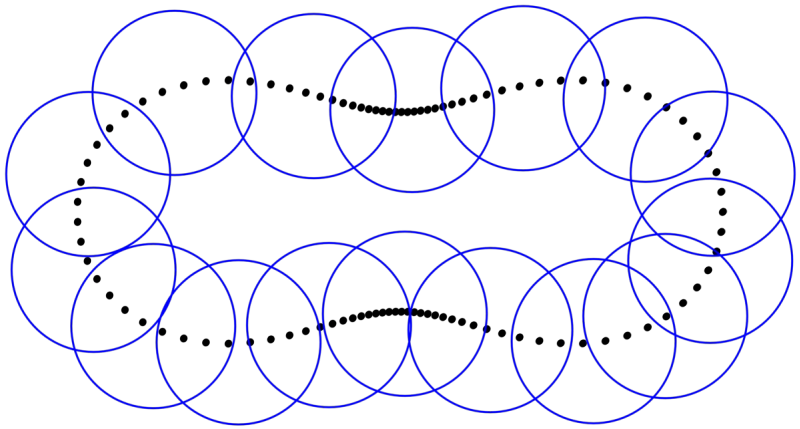
Babŭska & Melenk (1997), Wendland (2002), Lazzaro & Montefusco (2002), Tobor et. al. (2004), Ohtake et. al. (2005)

Many others: Cavoretto, De Rossi, De Marchi, Heryudono, Larsson, Perrachione, Safdari-Vaighani, Shcherbakov, Shankar, etc.

Drake, Fuselier, & Wright. A Partition of Unity Method for Divergence-free or Curl-free Radial Basis Function Approximation. *SIAM J. Sci. Comput.* 43(3), A1950-A1974 (2021)

- Cover Ω with overlapping patches Ω_i: $\Omega \subset \cup_{i=1}^{M} \Omega_i$

Patches covering Ω



RBF partition of unity method (RBF-PUM)

Babŭska & Melenk (1997), Wendland (2002), Lazzaro & Montefusco (2002), Tobor et. al. (2004), Ohtake et. al. (2005)

Many others: Cavoretto, De Rossi, De Marchi, Heryudono, Larsson, Perrachione, Safdari-Vaighani, Shcherbakov, Shankar, etc.

Drake, Fuselier, & Wright. A Partition of Unity Method for Divergence-free or Curl-free Radial Basis Function Approximation. *SIAM J. Sci. Comput.* 43(3), A1950-A1974 (2021)

- Cover Ω with overlapping patches Ω_i: $\Omega \subset \cup_{i=1}^{M} \Omega_i$

- Define PU weight functions:
  $w_i : \mathbb{R}^2 \to [0,1]$, $\text{supp}(w_i) \subset \Omega_i$, $\sum_{i=1}^{M} w_i(\mathbf{x}) \equiv 1$

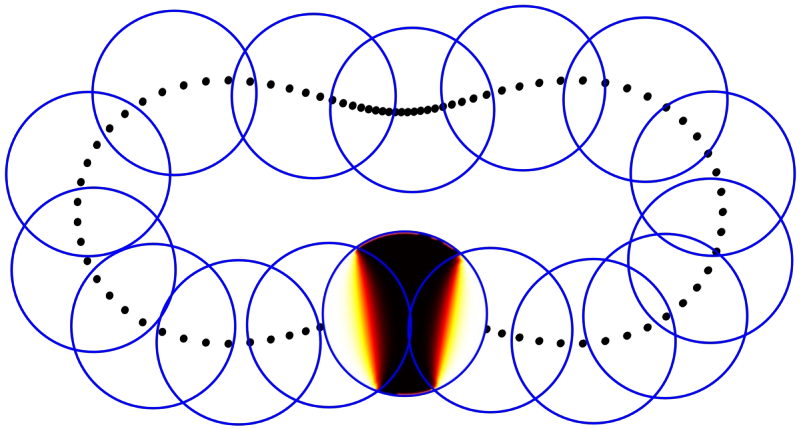
  Example:

$$w_i(\mathbf{x}) = \kappa_i(\mathbf{x}) / \sum_{j=1}^{M} \kappa_j(\mathbf{x}), \ i = 1, \ldots, M$$

$$\kappa_i(\mathbf{x}) := \kappa\left(\|\mathbf{x} - \boldsymbol{\xi}_i\| / \rho_i\right)$$

ρ_i = radius Ω_i $\quad \kappa$=quadratic B-spline

Illustration of a w_ℓ



RBF partition of unity method (RBF-PUM)

Babŭska & Melenk (1997), Wendland (2002), Lazzaro & Montefusco (2002), Tobor et. al. (2004), Ohtake et. al. (2005)

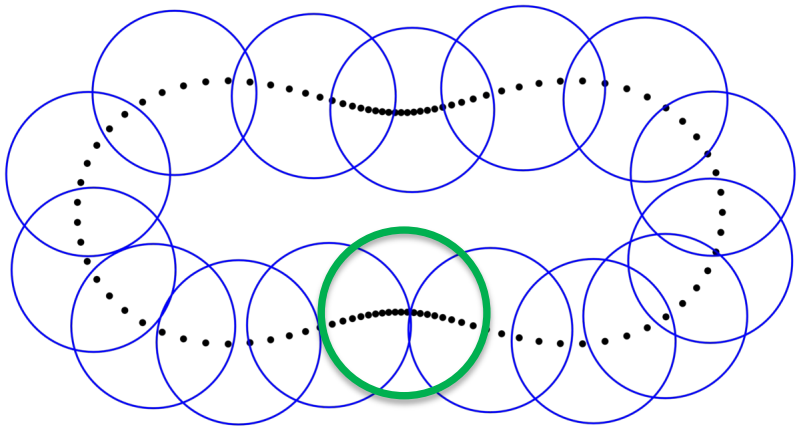
Many others: Cavoretto, De Rossi, De Marchi, Heryudono, Larsson, Perrachione, Safdari-Vaighani, Shcherbakov, Shankar, etc.

Drake, Fuselier, & Wright. A Partition of Unity Method for Divergence-free or Curl-free Radial Basis Function Approximation. *SIAM J. Sci. Comput.* 43(3), A1950-A1974 (2021)

- Cover Ω with overlapping patches Ω_i: $\Omega \subset \cup_{i=1}^{M} \Omega_i$
- Define PU weight functions:
  $w_i : \mathbb{R}^2 \to [0,1]$, $\text{supp}(w_i) \subset \Omega_i$, $\sum_{i=1}^{M} w_i(\mathbf{x}) \equiv 1$
- Let X_i contain the n_i nodes in Ω_i. Compute approximate potential for $\mathbf{n}_j$ over X_i

$$\begin{bmatrix} A_{\Phi}^{(i)} & P^{(i)} \\ (P^{(i)})^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c}^{(i)} \\ \mathbf{b}^{(i)} \end{bmatrix} = \begin{bmatrix} \mathbf{n}^{(i)} \\ \mathbf{0} \end{bmatrix}$$

Patches covering Ω



$$s_i(\mathbf{x}) = -\sum_{j=1}^{n_i} \nabla^T \phi_\ell\left(\|\mathbf{x}-\mathbf{x}_j\|\right) \mathbf{c}_j^{(i)} + \sum_{k=1}^{L} b_k^{(i)} p_k(\mathbf{x}) \quad \Longrightarrow \quad \boxed{\widetilde{s}_i(\mathbf{x}) := s_i(\mathbf{x}) - \sigma_i(\mathbf{x})}$$

RBF partition of unity method (RBF-PUM)

Babŭska & Melenk (1997), Wendland (2002), Lazzaro & Montefusco (2002), Tobor et. al. (2004), Ohtake et. al. (2005)

Many others: Cavoretto, De Rossi, De Marchi, Heryudono, Larsson, Perrachione, Safdari-Vaighani, Shcherbakov, Shankar, etc.

Drake, Fuselier, & Wright. A Partition of Unity Method for Divergence-free or Curl-free Radial Basis Function Approximation. *SIAM J. Sci. Comput.* 43(3), A1950-A1974 (2021)

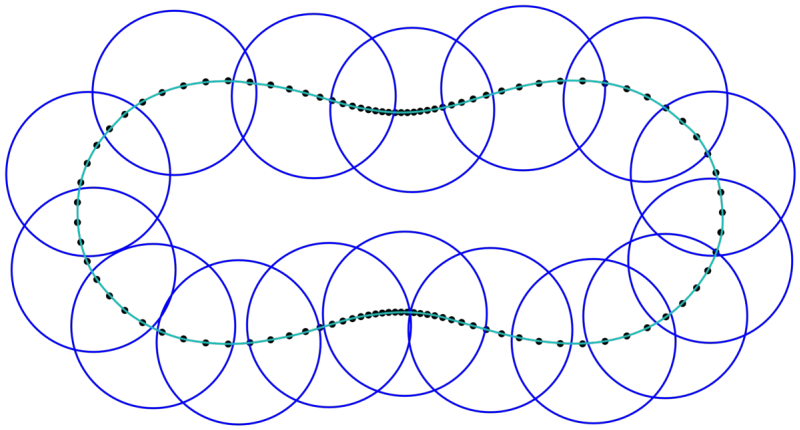
- Cover Ω with overlapping patches Ω_i: $\Omega \subset \cup_{i=1}^{M} \Omega_i$

- Define PU weight functions:
  $w_i : \mathbb{R}^2 \to [0,1]$, $\text{supp}(w_i) \subset \Omega_i$, $\sum_{i=1}^{M} w_i(\mathbf{x}) \equiv 1$

- Let X_i contain the n_i nodes in Ω_i. Compute approximate potential for $\mathbf{n}_j$ over X_i

$$\begin{bmatrix} A_\Phi^{(i)} & P^{(i)} \\ (P^{(i)})^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c}^{(i)} \\ \mathbf{b}^{(i)} \end{bmatrix} = \begin{bmatrix} \mathbf{n}^{(i)} \\ \mathbf{0} \end{bmatrix}$$

Patches covering Ω



$$s_i(\mathbf{x}) = -\sum_{j=1}^{n_i} \nabla^T \phi_\ell\left(\|\mathbf{x}-\mathbf{x}_j\|\right) \mathbf{c}_j^{(i)} + \sum_{k=1}^{L} b_k^{(i)} p_k(\mathbf{x}) \quad \Longrightarrow \quad \widetilde{s}_i(\mathbf{x}) := s_i(\mathbf{x}) - \sigma_i(\mathbf{x})$$

- Blend the $\widetilde{s}_i$ to obtain a global potential to extract the zero-level surface

$$\tilde{s}(\mathbf{x}) = \sum_{i=1}^{M} w_i(\mathbf{x}) \widetilde{s}_i(\mathbf{x})$$

$\Longrightarrow$ CFPU Method

Computational cost: $\sum_{i=1}^{M} \mathcal{O}\left(n_i^3\right)$

Refine patches with increasing N

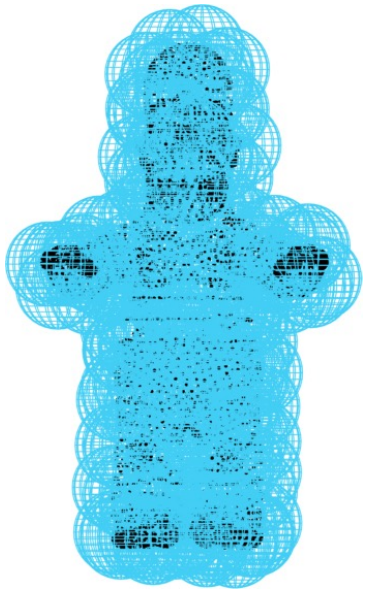
Highly parallelizable

Some details on CFPU

Choosing the partition of unity patches

1) Use MeshLab's "point cloud simplification" to pick a subset of the points $X = \{\mathbf{x}\}_{j=1}^{N}$

$$\Longrightarrow \quad \Xi = \{\boldsymbol{\xi}_i\}_{i=1}^{M}$$



2) Choose the initial patch radii as $\rho = (1+\delta)\tau/2$ (δ controls overlap)

$$\tau = \max_{1\le i\le M} \min_{\substack{1\le j\le M \\ j\neq i}} \|\boldsymbol{\xi}_i - \boldsymbol{\xi}_j\|$$

Adjust each patch radius so that $n_i \geq n_{\min}$

Noisy data

Two different sources of noise: 1) normals 2) point samples

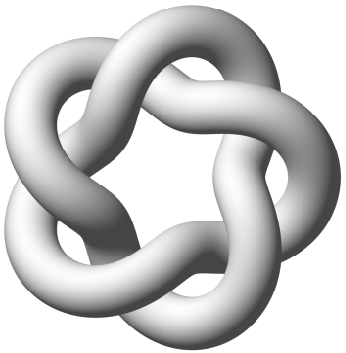
We use smoothing spline or ridge regression regularizations (Wahba, 1990) on each patch Ω_i

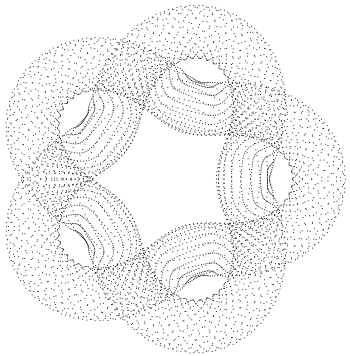
Normals: $$\min_{\mathbf{c}\in\mathbb{R}^{3n}} \left[\frac{1}{3n} \sum_{j=1}^{n} \|\mathbf{s}_i(\mathbf{x}_j) - \mathbf{n}_j^{(i)}\|^2 + \lambda (\mathbf{c}^{(i)})^T A_{\Phi}^{(i)} \mathbf{c}^{(i)} \right], \text{ subject to } (P^{(i)})^T \mathbf{c}^{(i)} = 0$$

Points: $$\min_{\mathbf{c}\in\mathbb{R}^{n}} \left[\frac{1}{n} \sum_{j=1}^{n} \|\sigma_i(\mathbf{x}_j) - s_i(\mathbf{x}_j)\|^2 + \alpha (\mathbf{c}^{(i)})^T A_{\phi}^{(i)} \mathbf{c}^{(i)} \right], \text{ subject to } (P^{(i)})^T \mathbf{c}^{(i)} = 0$$

Numerical experiments: Accuracy

Target surface and point cloud:





Kernel:

$$\phi_\ell(r) = (-1)^{\ell+1} r^{2\ell+1}$$

$$\Phi_\ell(\mathbf{x}, \mathbf{y}) = -\nabla\nabla^T \phi_\ell\left(\|\mathbf{x} - \mathbf{y}\|\right)$$

CFPU Convergence

	$\ell = 1$		$\ell = 2$	
N	RMS error	Max-norm error	RMS error	Max-norm error
6114	9.90×10^{-5}	7.09×10^{-4}	8.08×10^{-6}	7.69×10^{-5}
8664	4.12×10^{-5}	5.21×10^{-4}	3.45×10^{-6}	2.36×10^{-5}
11616	2.27×10^{-5}	1.57×10^{-4}	1.69×10^{-6}	1.49×10^{-5}
18816	7.46×10^{-6}	5.43×10^{-5}	4.53×10^{-7}	2.82×10^{-6}
23064	5.21×10^{-6}	3.69×10^{-5}	2.67×10^{-7}	1.71×10^{-6}
27744	4.10×10^{-6}	2.75×10^{-5}	1.68×10^{-7}	1.14×10^{-6}
32856	2.87×10^{-6}	2.56×10^{-5}	9.69×10^{-8}	8.13×10^{-7}

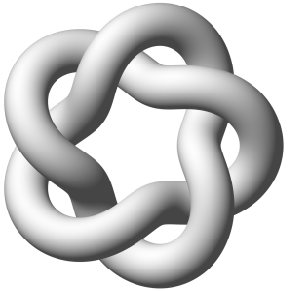
Fixed number of M=864 patches

Numerical experiments: noisy normals

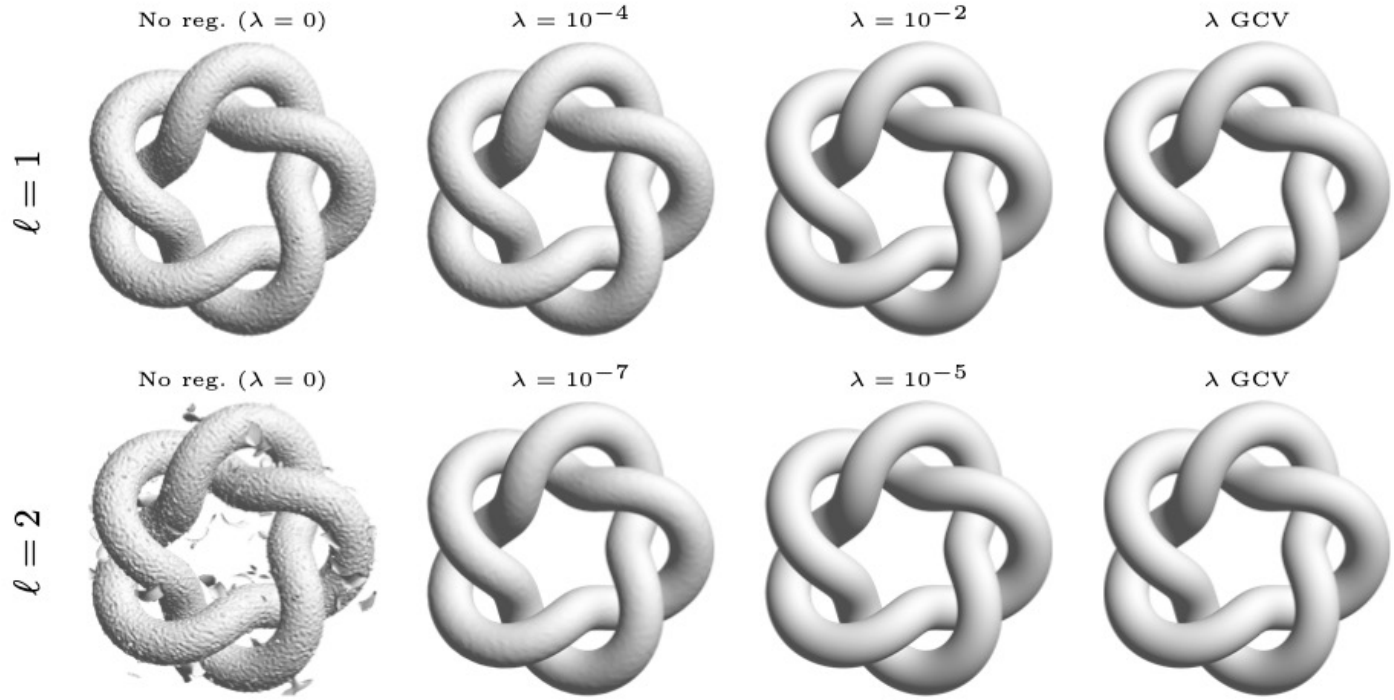
Normally distributed random noise added (mean zero, std=0.3):

$$\mathbf{n}_j^* = \mathbf{n}_j + \boldsymbol{\epsilon}_j$$

Target



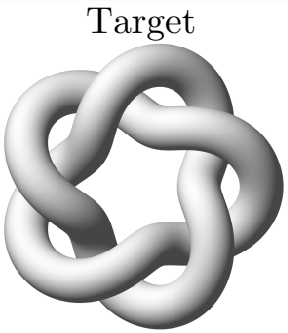
CFPU

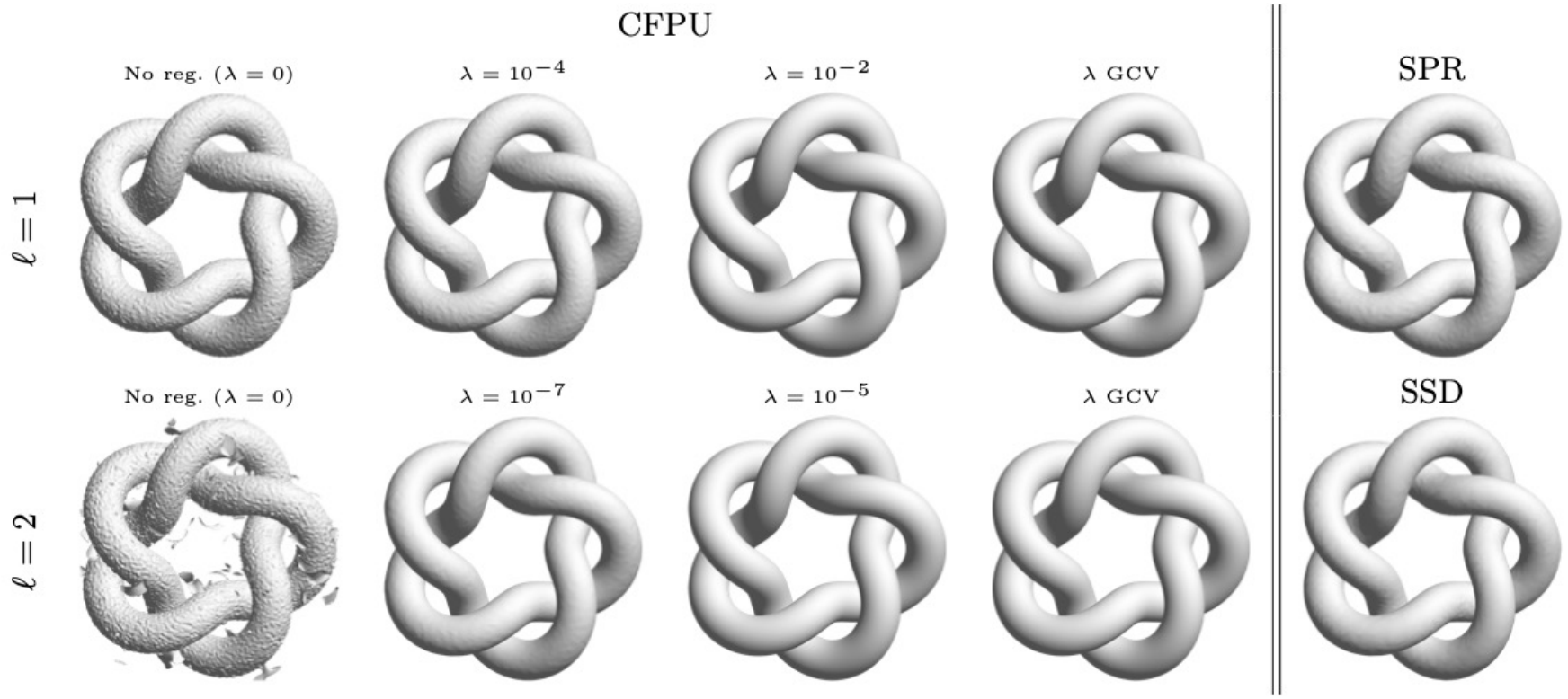


Numerical experiments: noisy normals

Normally distributed random noise added (mean zero, std=0.3):

$$\mathbf{n}_j^* = \mathbf{n}_j + \boldsymbol{\epsilon}_j$$

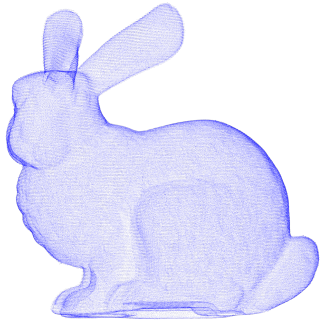




SPR = screened Poisson surface reconstruction method (Kazhdan, Bolitho, & Hoppe, 2006 & 2013)
SSD = smoothed signed distance method (Calakli & Taubin, 2006 & 2013)

Numerical experiments: raw range data

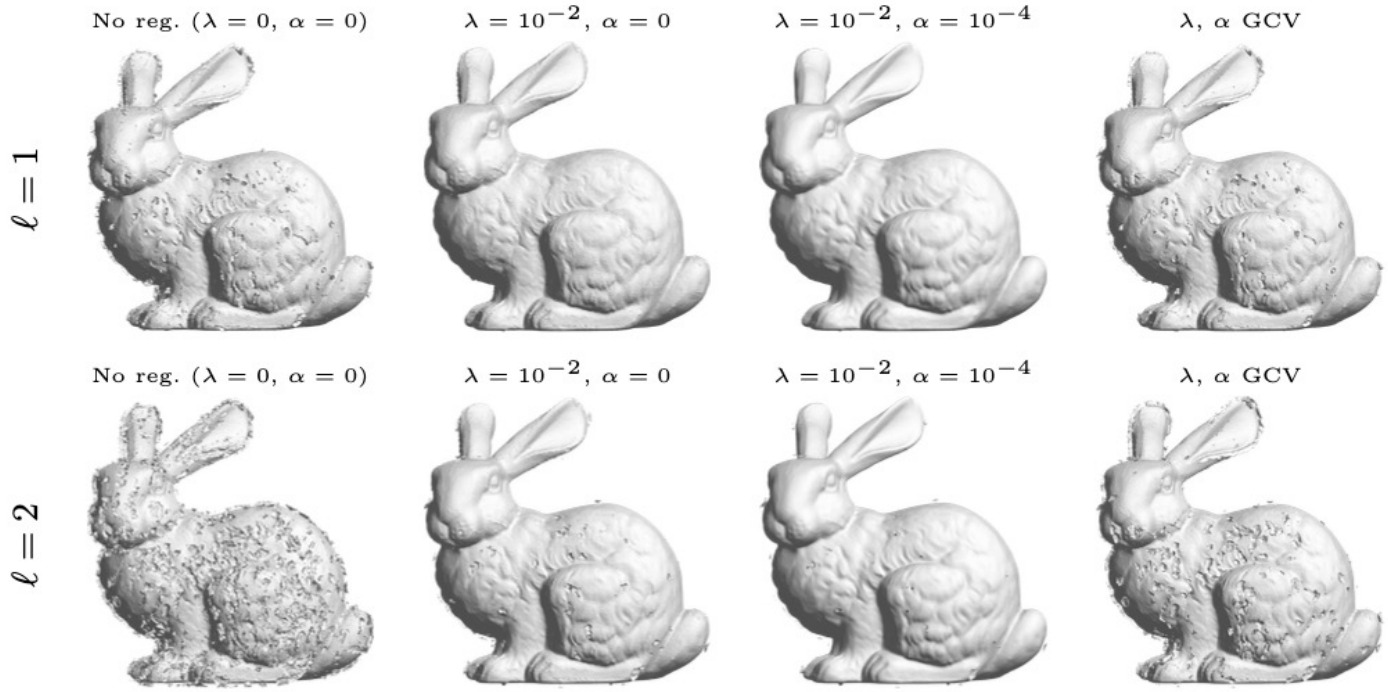
Stanford bunny:
N=362271 points



Data comes from an optical scan and contains noise in the normal and the points

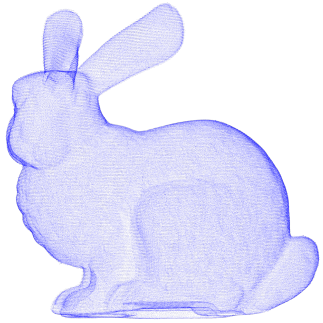
λ = normals regularization, α = residual regularization

CFPU



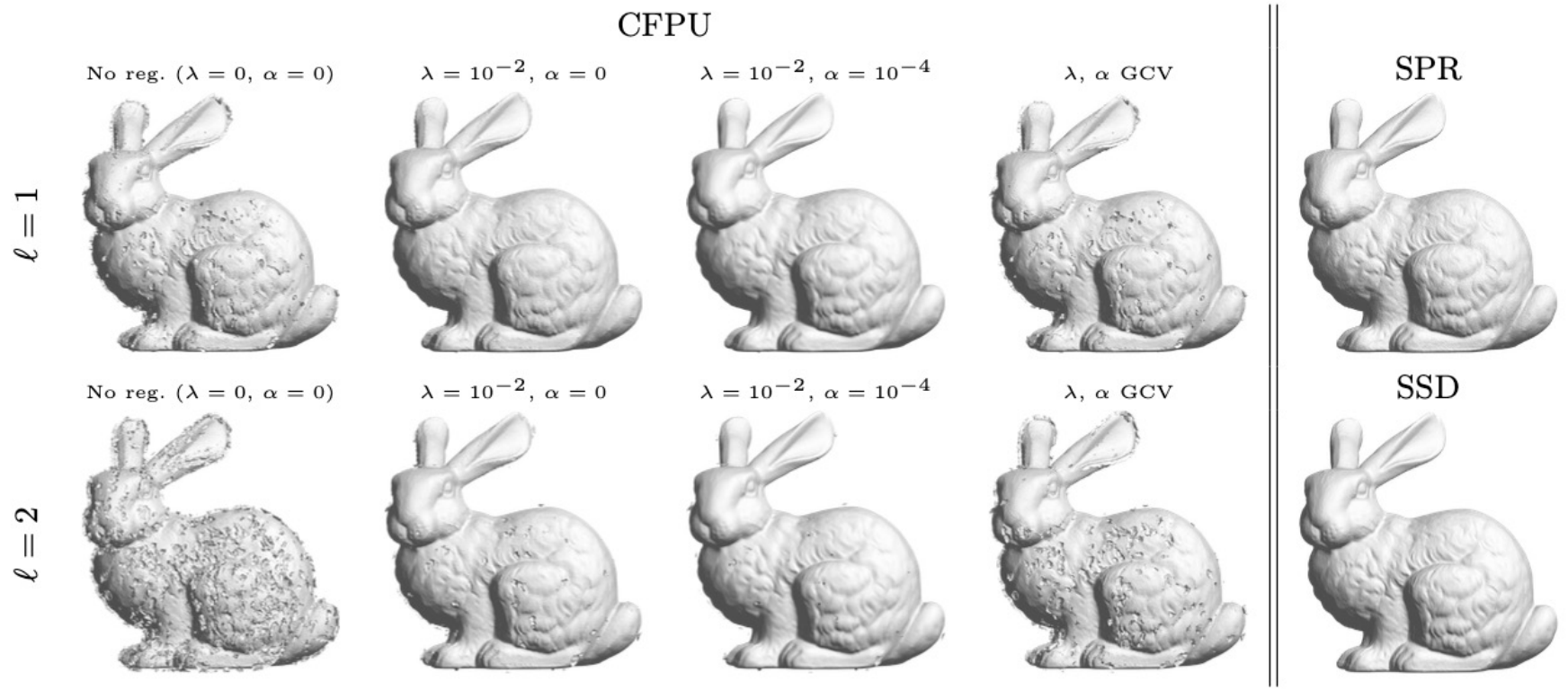
Numerical experiments: raw range data

Stanford bunny:
N=362271 points



Data comes from an optical scan and contains noise in the normal and the points

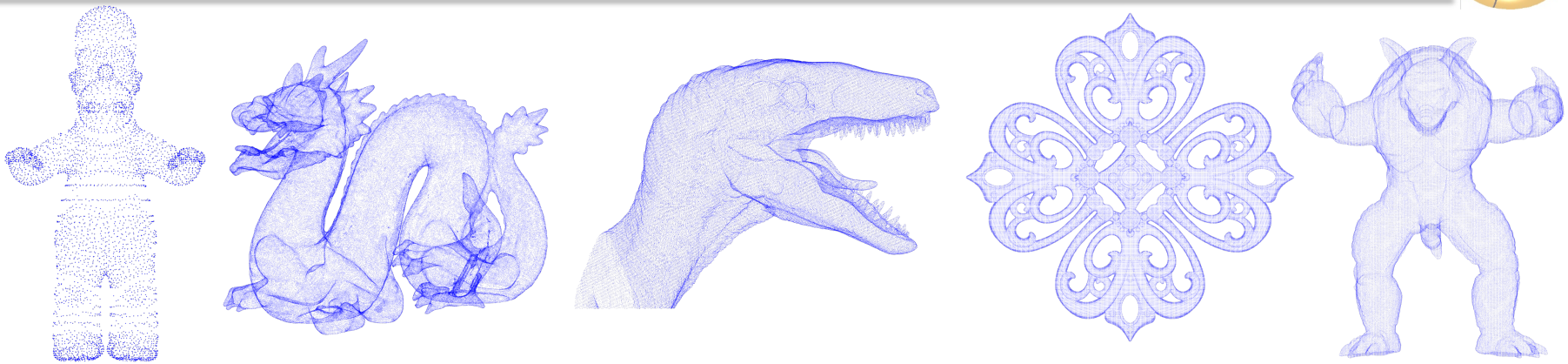
λ = normals regularization, α = residual regularization



SPR = screened Poisson surface reconstruction method (Kazhdan, Bolitho, & Hoppe, 2006 & 2013)
SSD = smoothed signed distance method (Calakli & Taubin, 2006 & 2013)

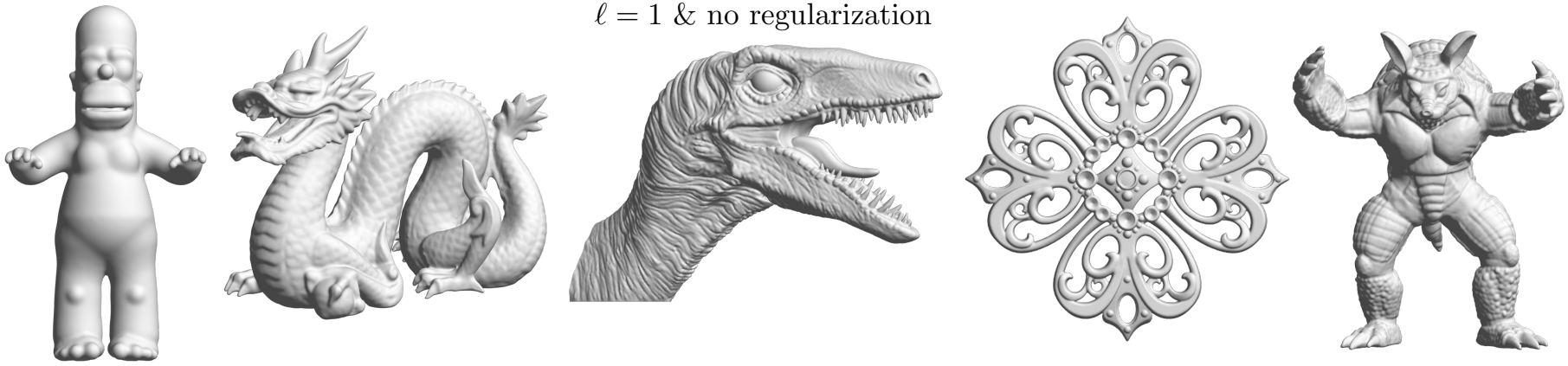
Numerical experiments: common 3D models

Oriented point cloud

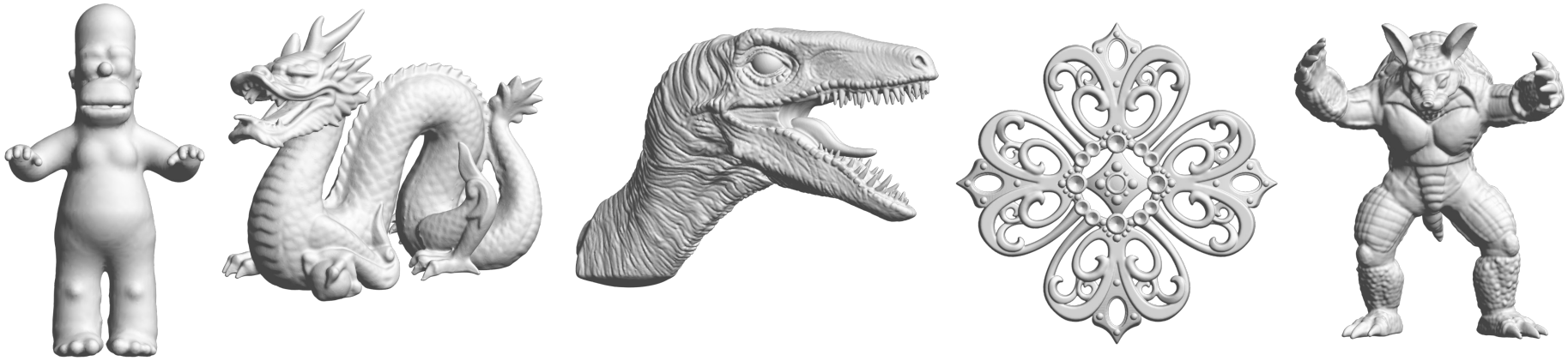


$\ell = 1$ & no regularization

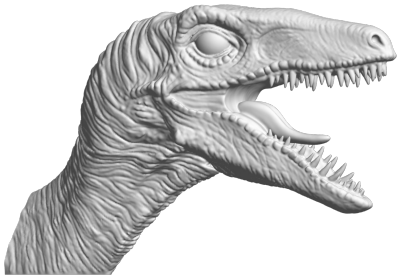
CFPU Method



SPR Method



Computational performance



Raptor

Filigree

Dragon

Buddha

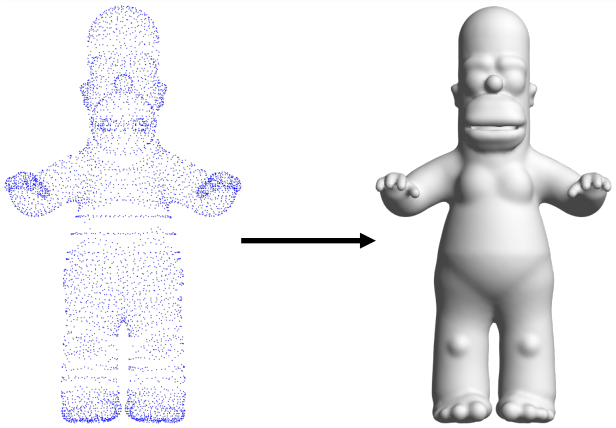
Armadillo

Surface	N	M	Grid size	Time in seconds Number of cores 1	2	4	8
Raptor	135740	10337	384×143×258	18.2 (16.5)	6.67 (8.84)	4.14 (7.22)	3.35 (5.69)
Filigree	514300	35130	383×63×384	30.4 (7.70)	10.9 (4.10)	6.63 (3.39)	4.99 (3.13)
Dragon	434856	14400	384×175×272	42.73 (11.83)	17.24 (6.46)	10.7 (5.46)	9.20 (4.82)
Buddha	583079	42861	161×162×384	37.34 (7.71)	14.0 (4.20)	8.32 (3.43)	5.99 (3.02)
Armadillo	172974	14349	323×294×384	9.42 (10.0)	3.62 (5.54)	2.25 (4.85)	1.61 (4.55)

fit time & eval. time

- All code implemented in MATLAB R2020b using the parallel computing toolbox
- Zero-level surfaces obtained using the `isosurface` function in MATLAB
- Machine: MacBook Pro with 2.4 GHz 8-Core Intel Core i9 processor and 32 GB RAM

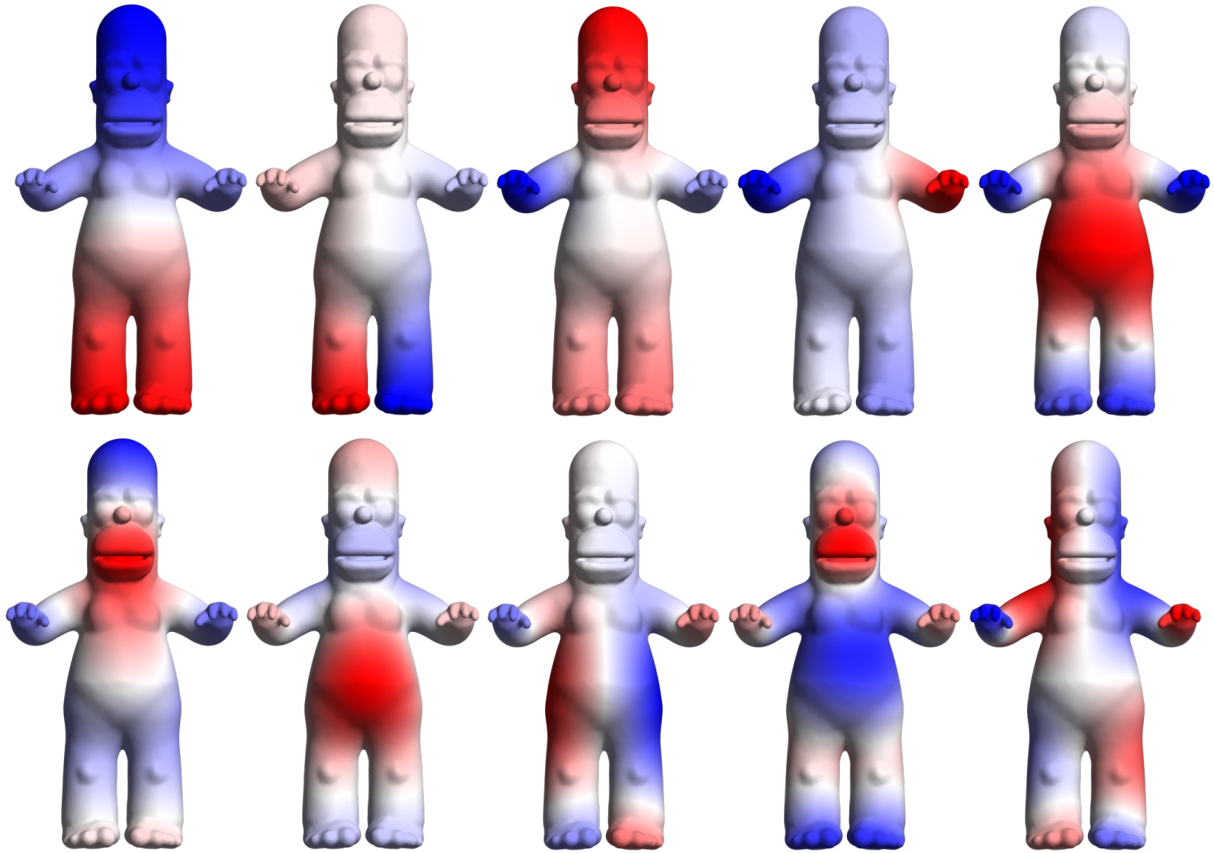
Solving surface PDEs



Compute the surface harmonics:

$$\Delta_S u = \lambda u$$

Meshfree method based on Shankar et. al. 2015



Concluding remarks

- CFPU is a computationally efficient method for implicit surface reconstruction
- Using PHS avoids having to choose shape or support parameters
- Exact interpolation or smoothing can be easily incorporated
- Pleasingly parallelizable: only requires solving small, decoupled linear systems
- Compares favorably to other popular methods

K. P. Drake, E. J. Fuselier, and G. B. Wright. Implicit Surface Reconstruction with a Curl-free Radial Basis Function Partition of Unity Method. arXiv:2101.05940

Future:

- Develop error estimates
- Automated technique for generating the PU patches for non-uniformly sampled point clouds
- Adapt the shapes of the PU patches from balls to ellipsoids that better conform to the underlying surface
- Automated approaches for selecting the regularization parameters
- Investigate least squares regularization (Larsson et. al. 2017)
- Implement a parallel version of the method in C/C++