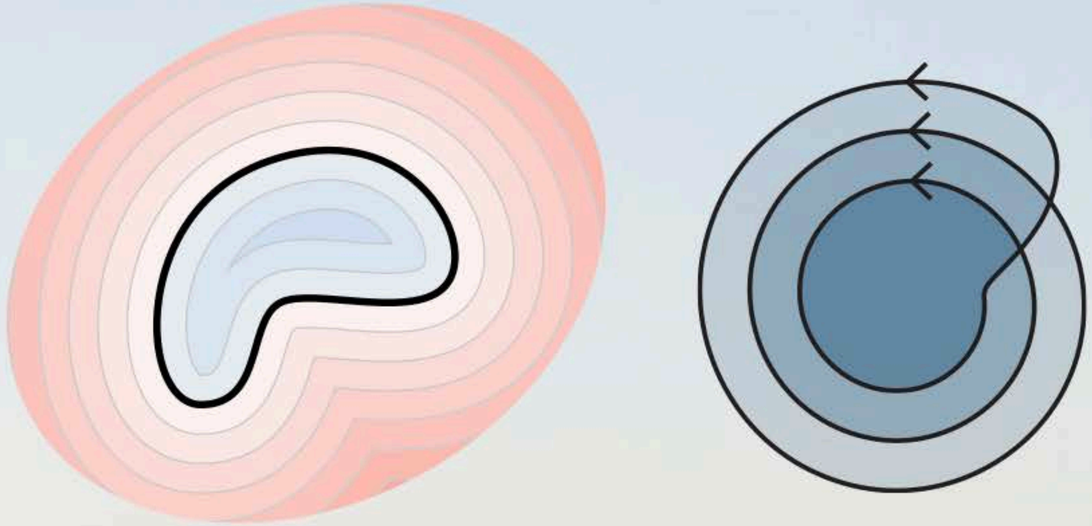
# Algorithms for Generalized Signed Distance and Winding Numbers

Nicole Feng

Carnegie Mellon University



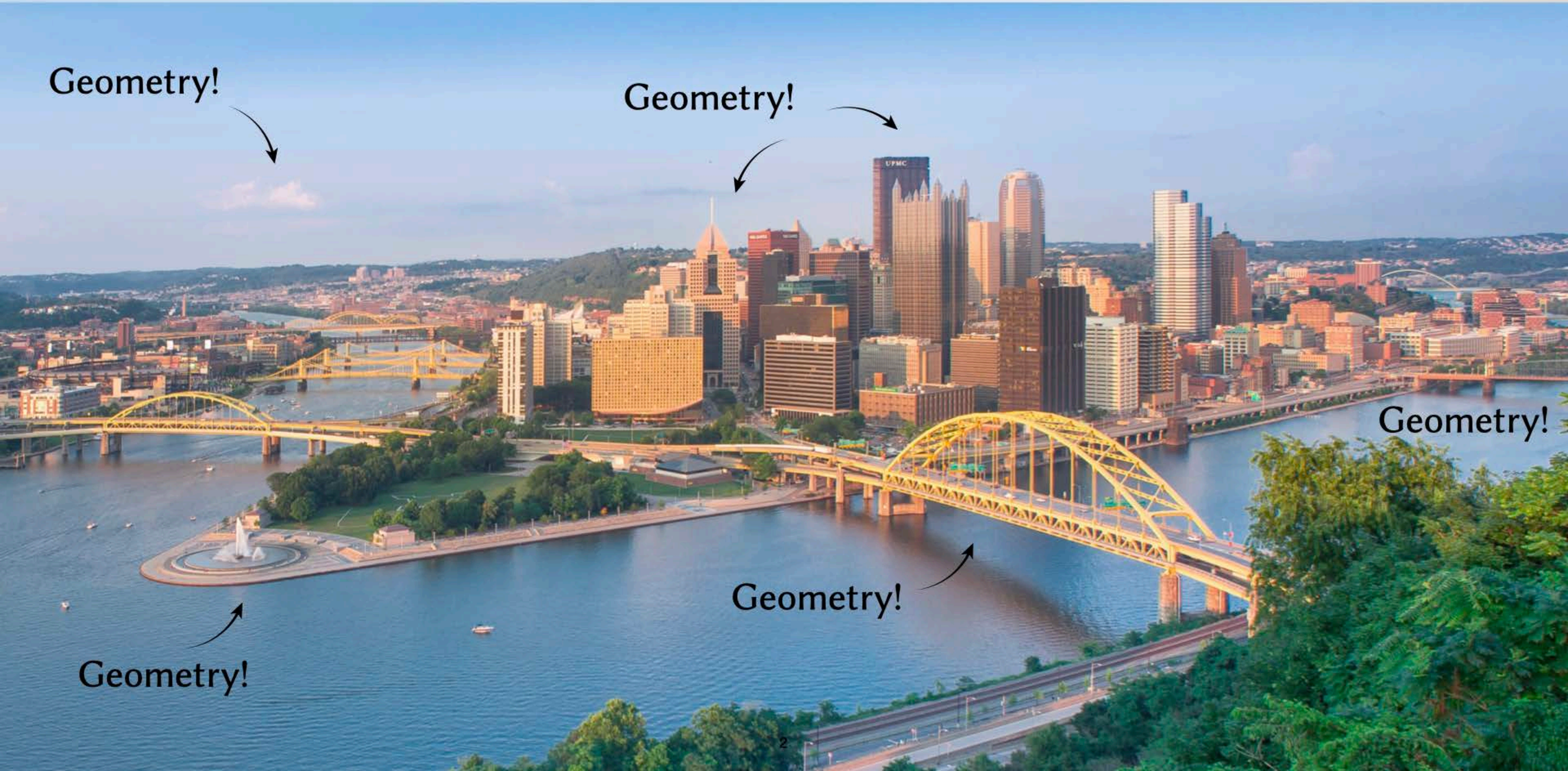
# Our lives are defined by geometry



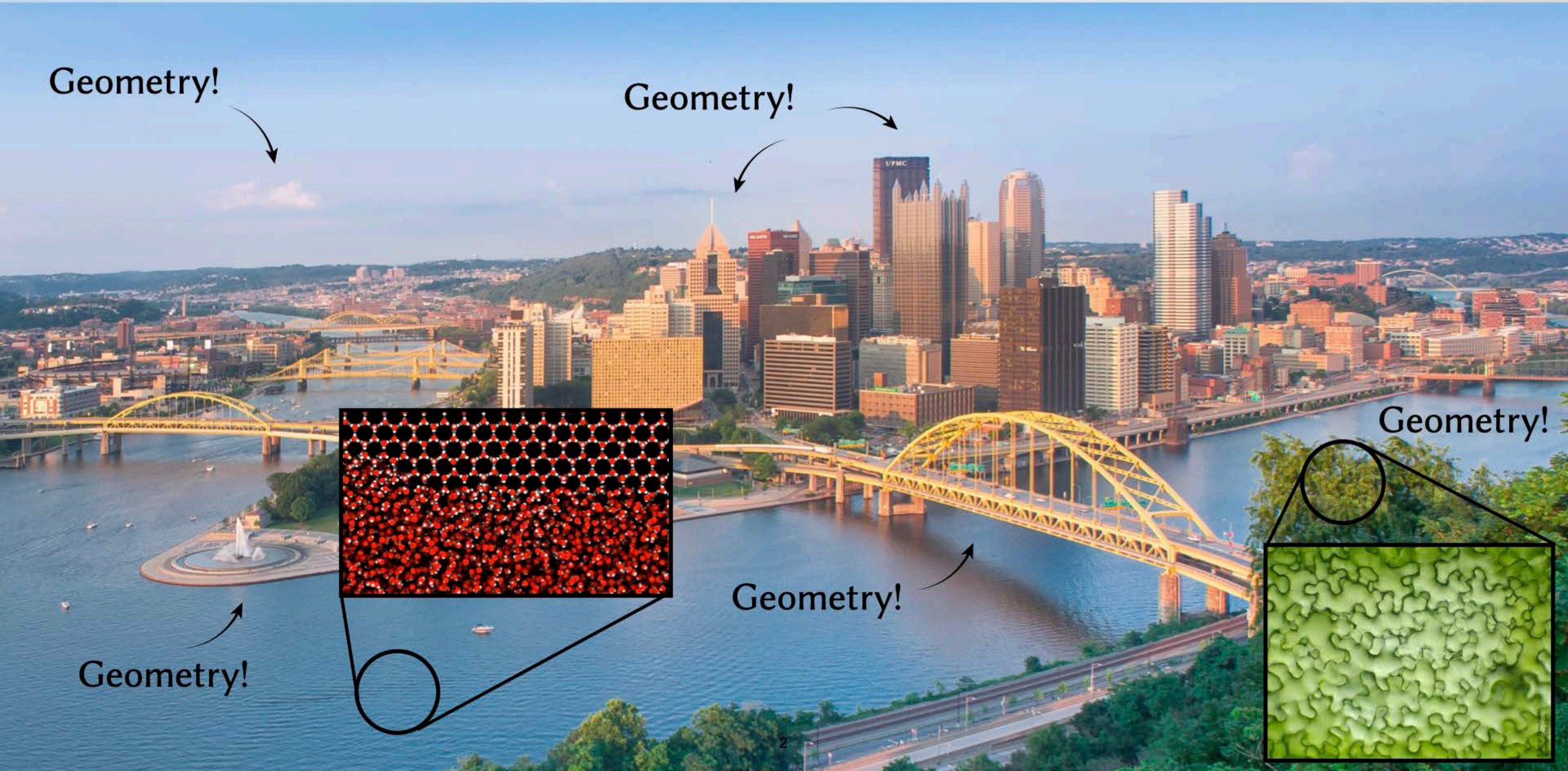
# Our lives are defined by geometry



# Our lives are defined by geometry

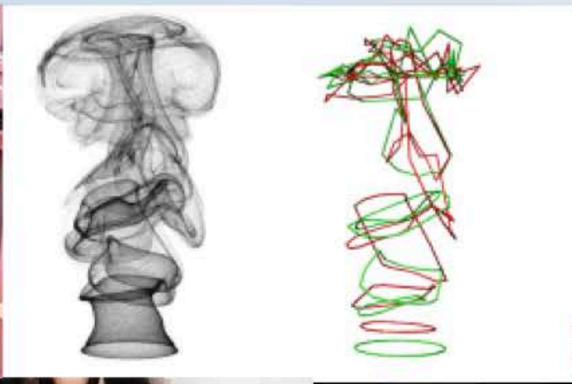


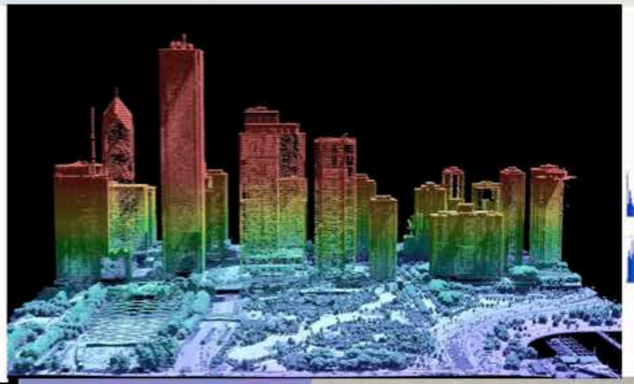
# Our lives are defined by geometry

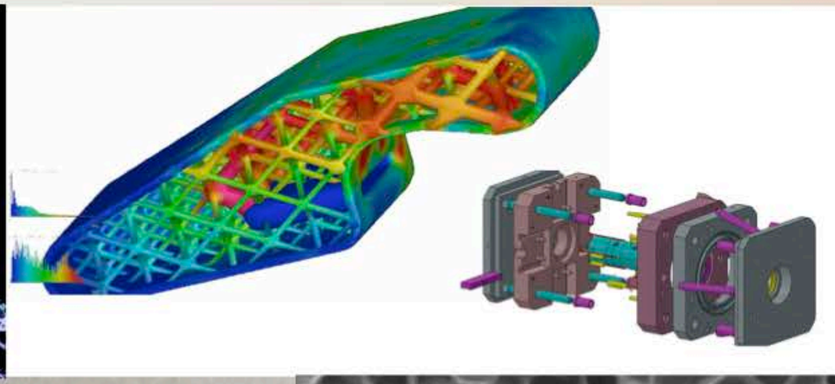


# Solving geometric problems drives progress













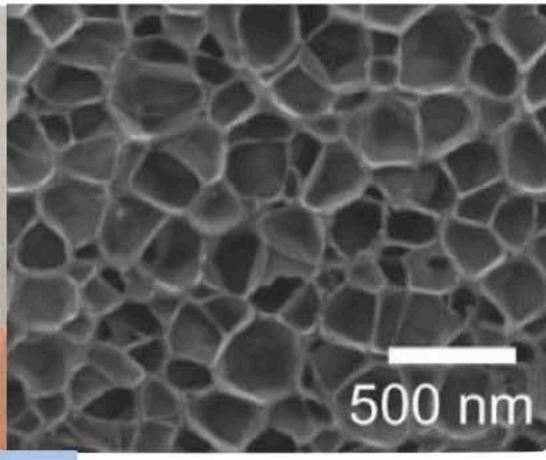








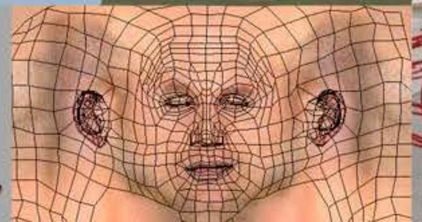






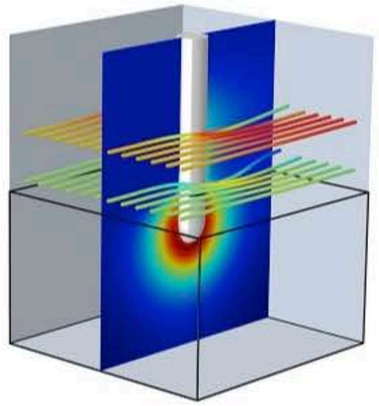






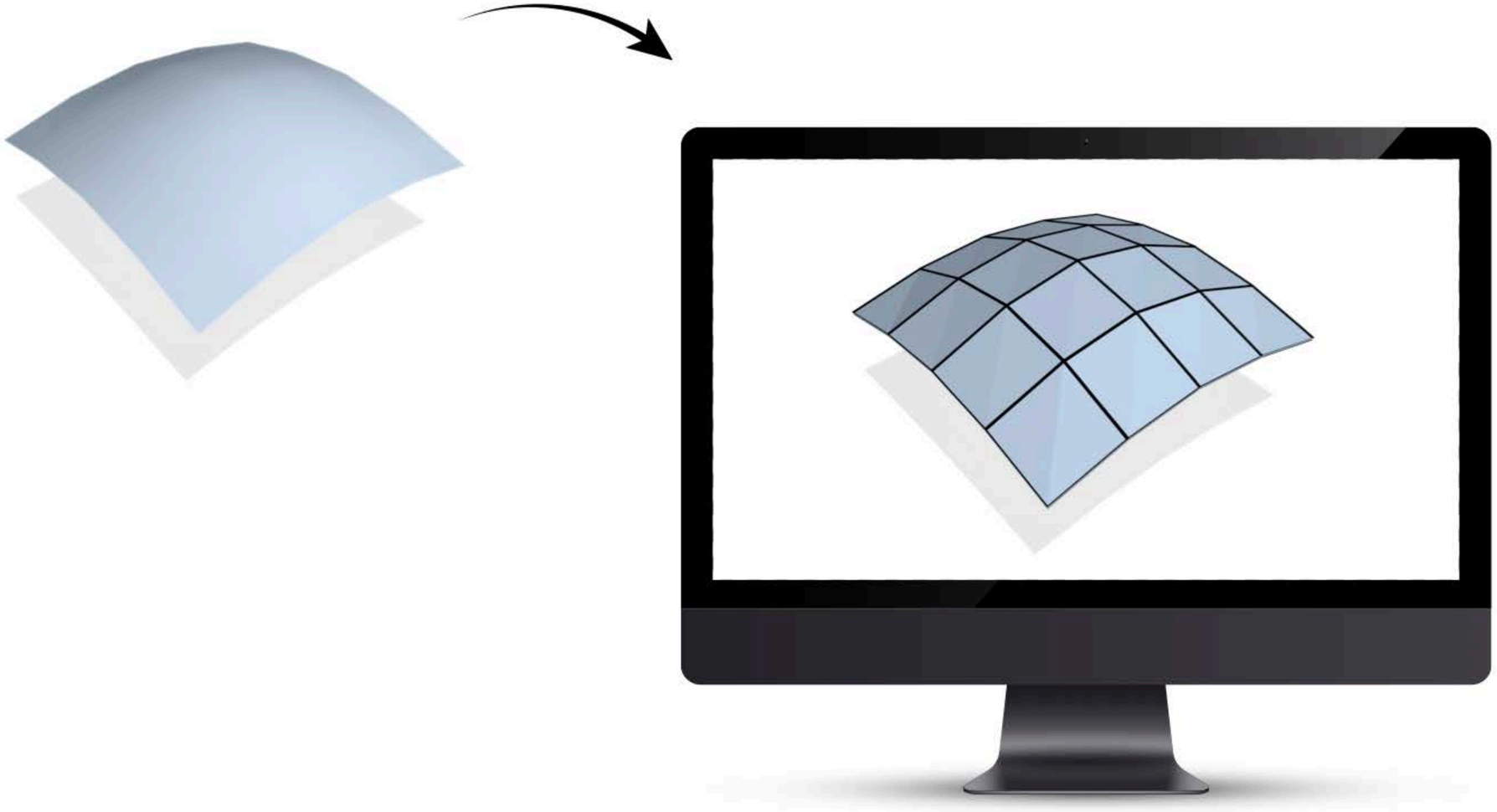




# Geometric problems are hard?

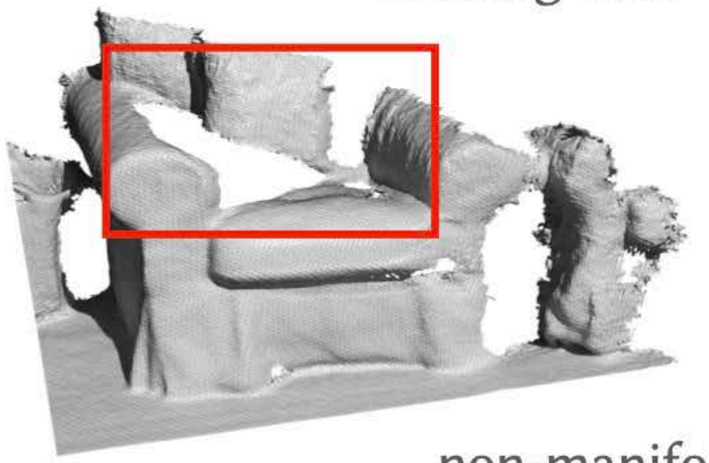
# Geometric ~~problems~~ computing is hard

# Geometric ~~problems~~ computing is hard

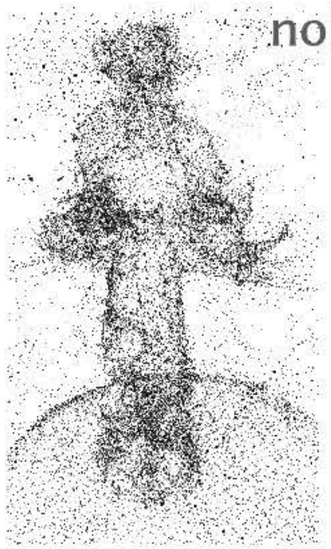


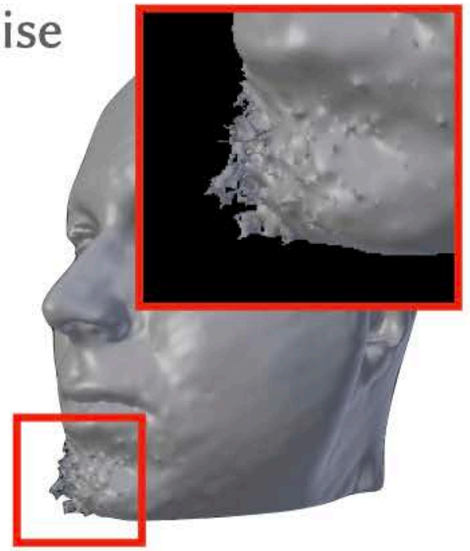
# There are many ways geometry can be messy...

missing data

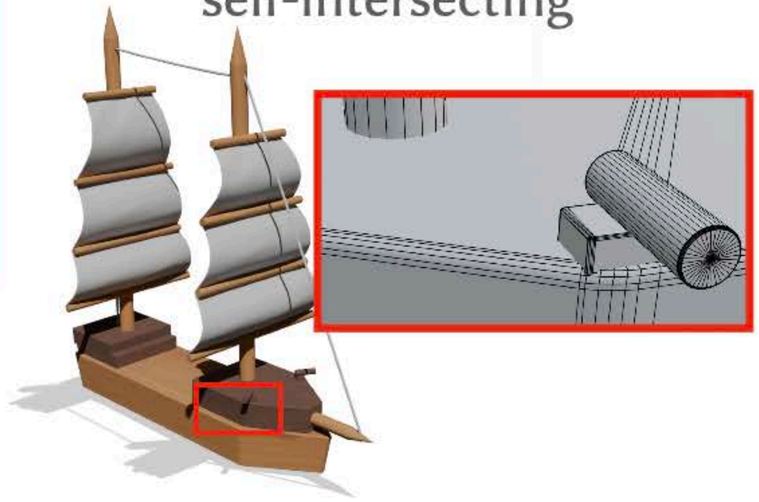


noise

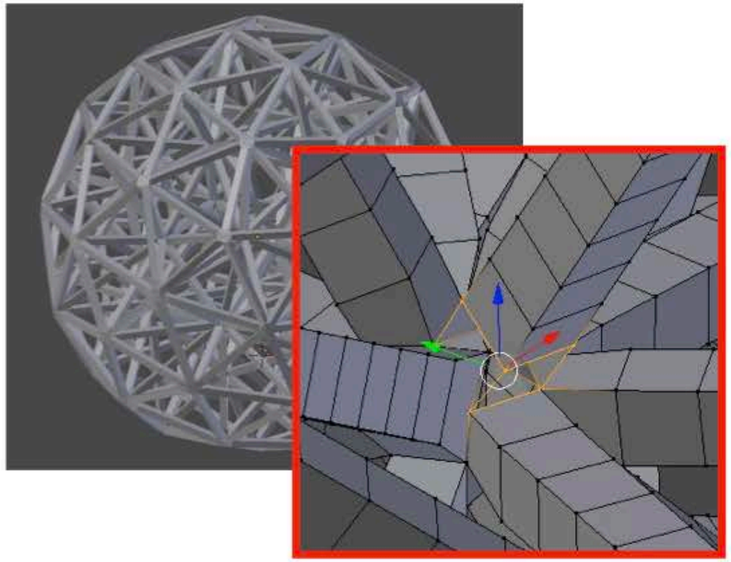


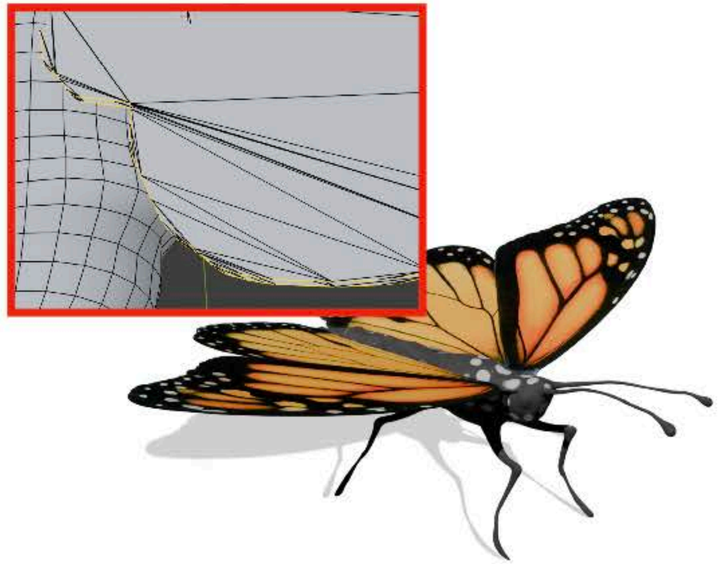


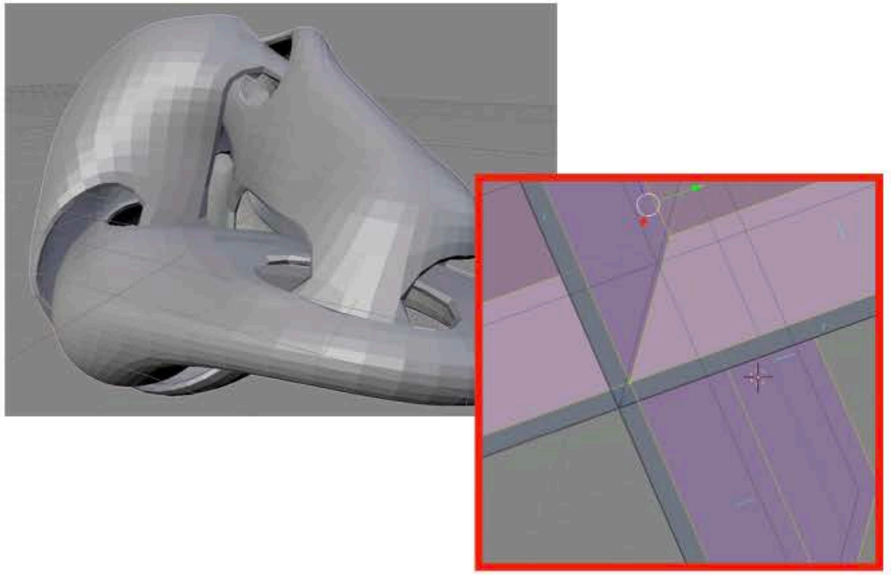
self-intersecting



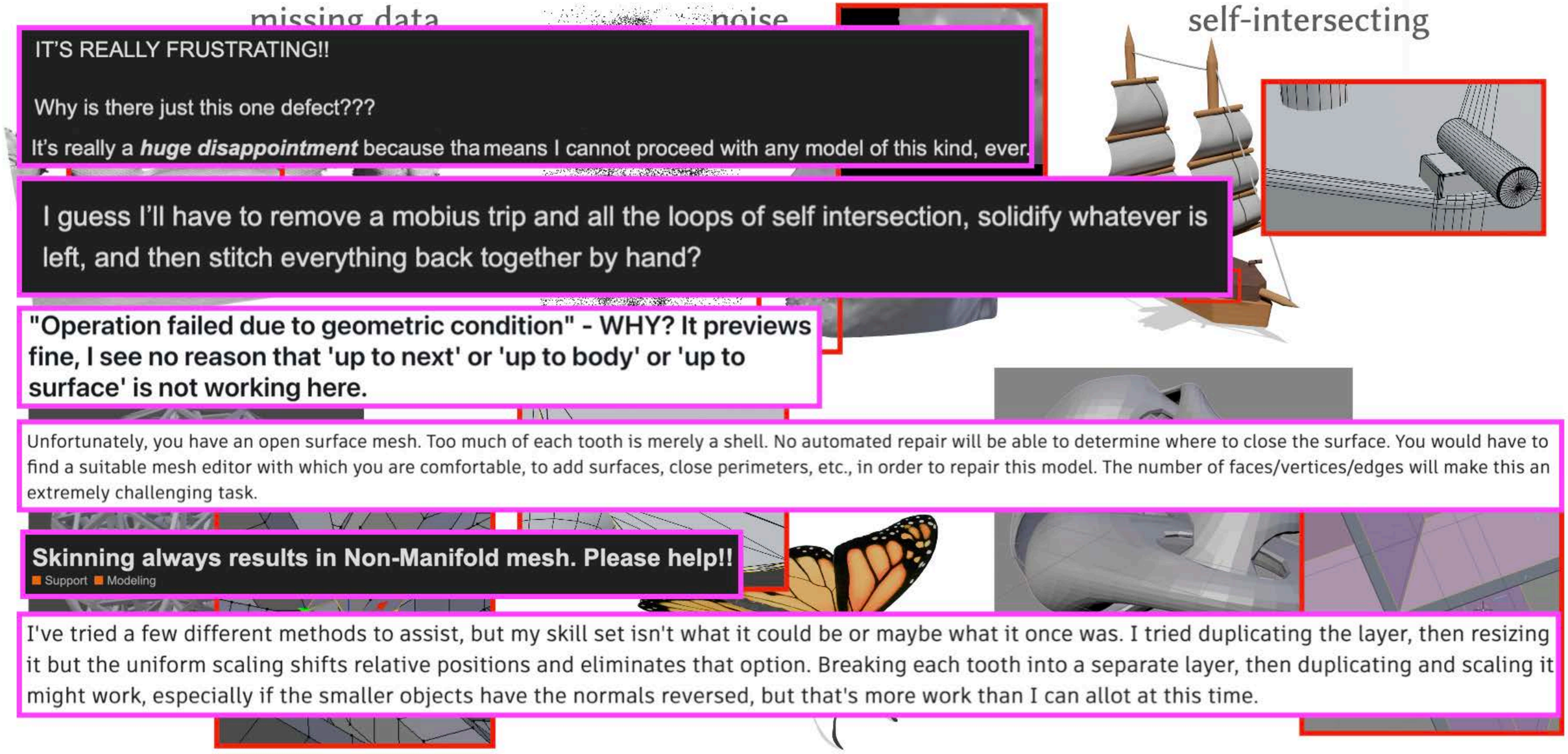
non-manifold







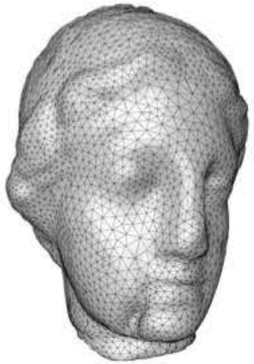
# There are many ways geometry can be messy...



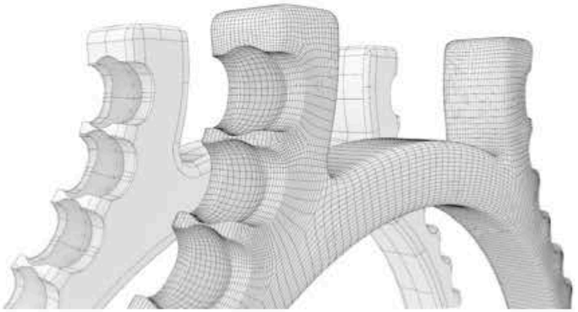
# … and many representations of geometry

# … and many representations of geometry

*triangle meshes*



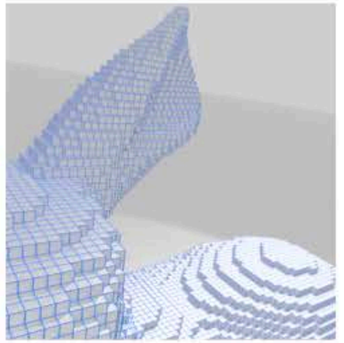
*polygon meshes*



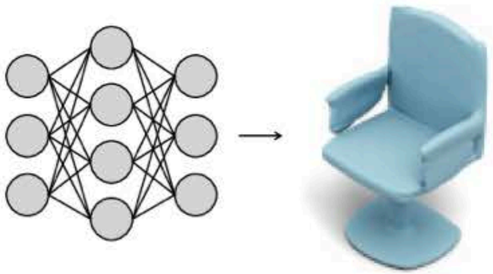
*point clouds*



*voxelizations*



*neural implicits*



*Gaussian splats*

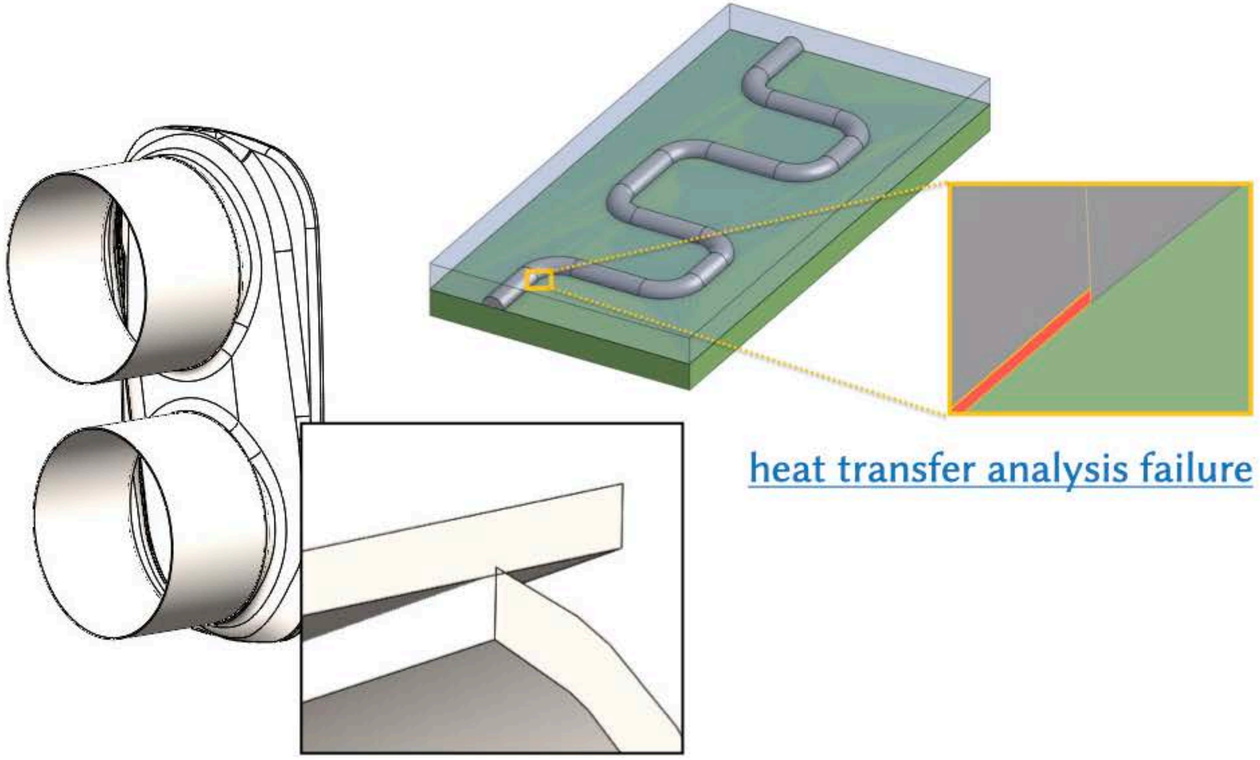


*… and there's always going to be more!*

# Without good geometric tools, nothing works

# Without good geometric tools, nothing works

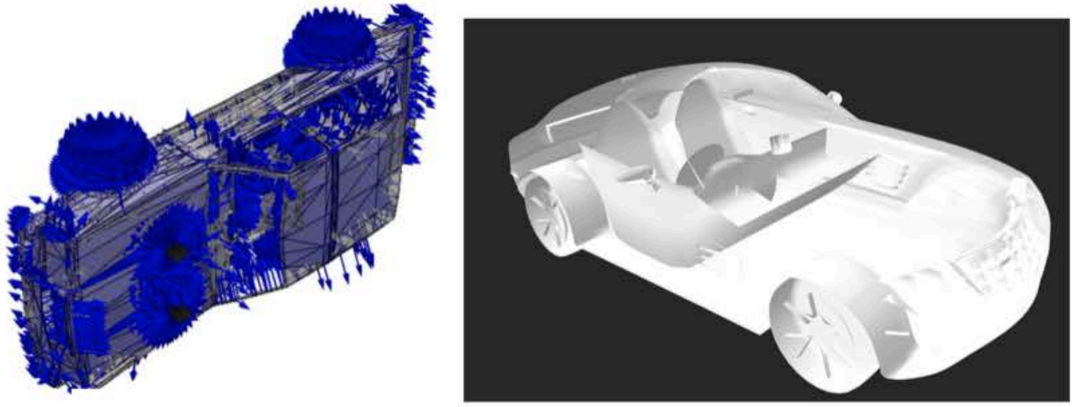
*physics analysis/simulation*

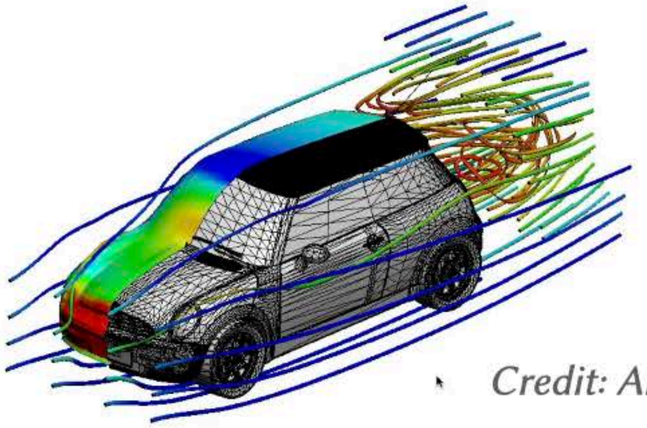


heat transfer analysis failure

SolidWorks Flow Simulation failure

*machine learning pipelines*

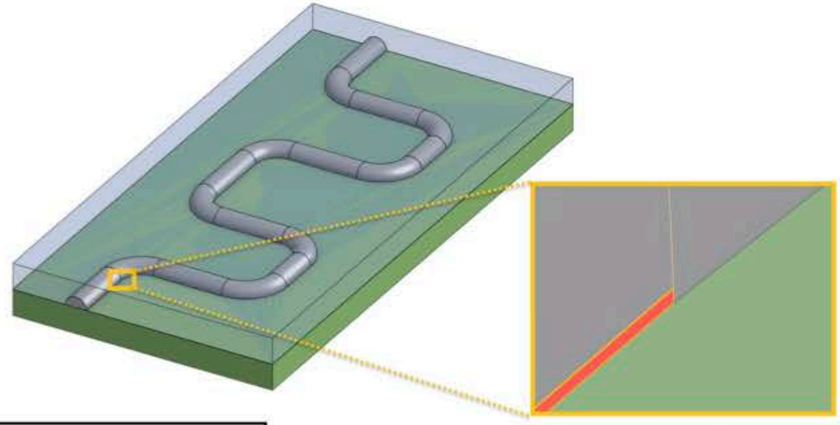




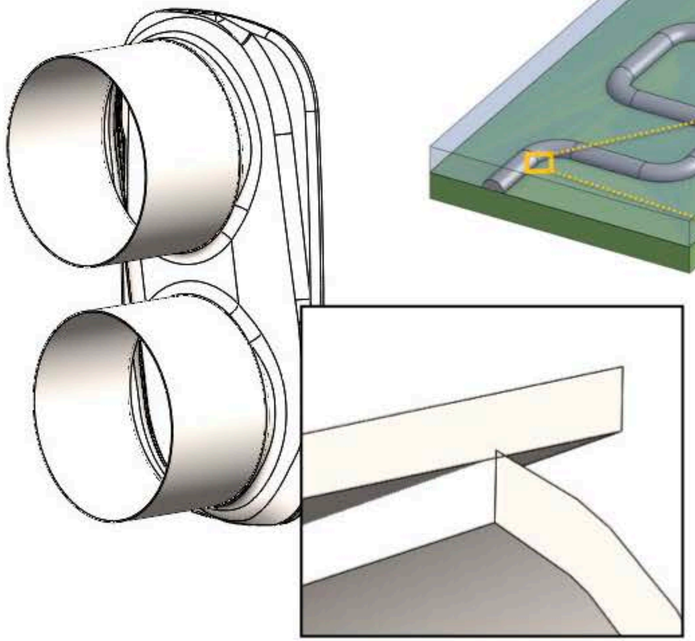
*Credit: Anima Lab*

# Without good geometric tools, nothing works

*physics analysis/simulation*

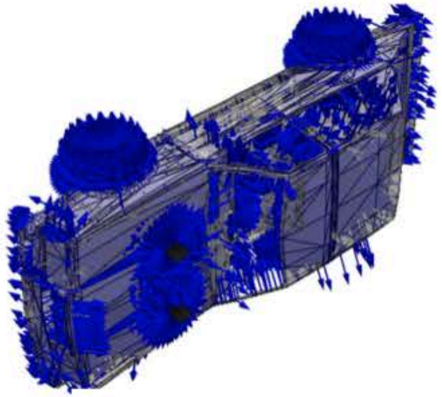


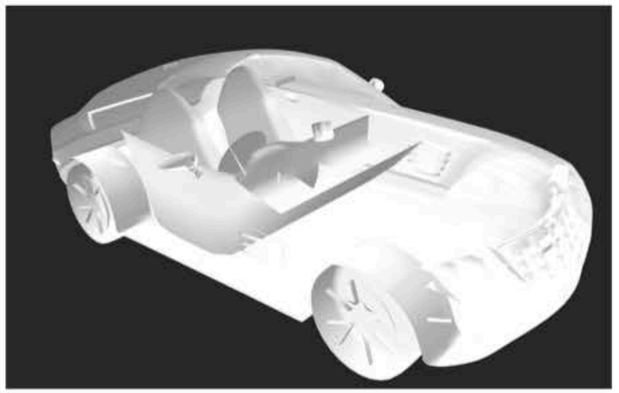
heat transfer analysis failure

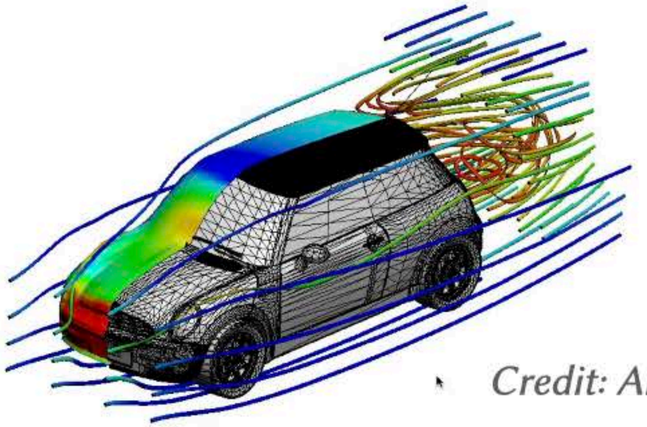


SolidWorks Flow Simulation failure

*machine learning pipelines*







*Credit: Anima Lab*

Need to answer fundamental questions about **geometry**.

# We need robust geometry processing

# We need robust geometry processing

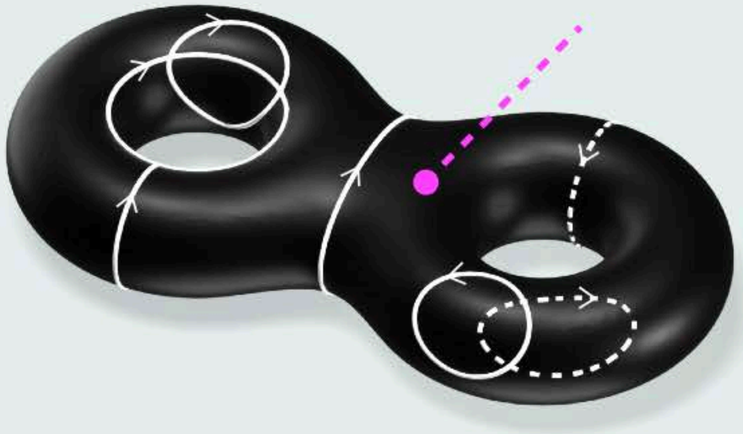
Robust geometry processing =

methods that work reliably across varying degrees of quality of their input.

# We need robust geometry processing

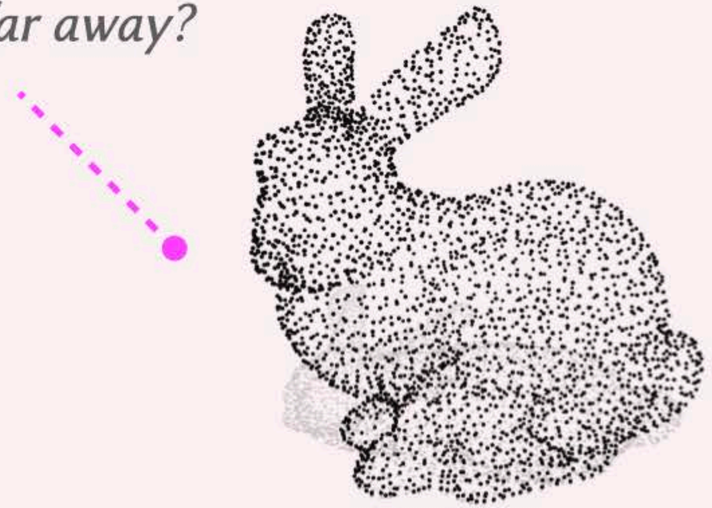
Robust geometry processing =
methods that work reliably across varying degrees of quality of their input.

*Inside or outside?*



GENERALIZED INSIDE/OUTSIDE

*How far away?*

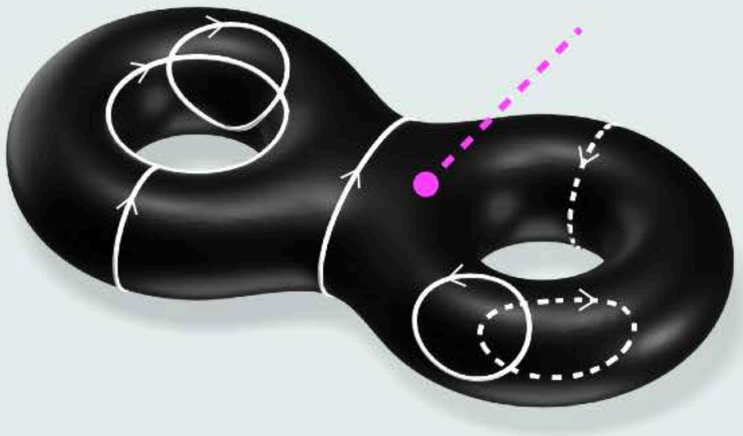


GENERALIZED SIGNED DISTANCE

# We need robust geometry processing

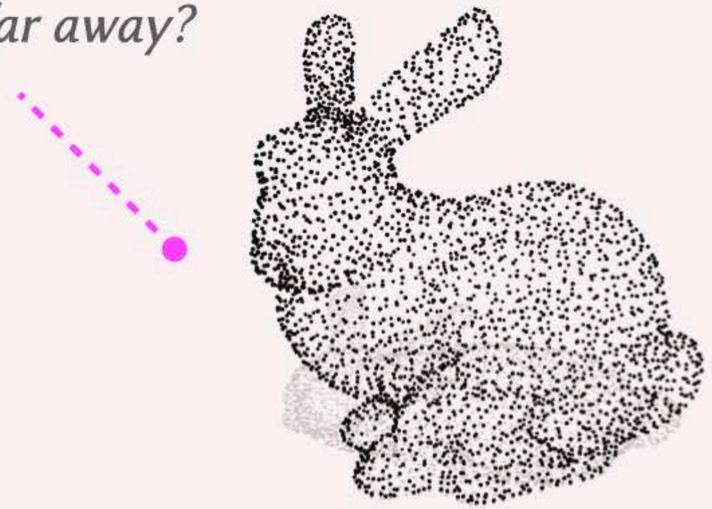
Robust geometry processing =
methods that work reliably across varying degrees of quality of their input.

*Inside or outside?*



GENERALIZED INSIDE/OUTSIDE

*How far away?*

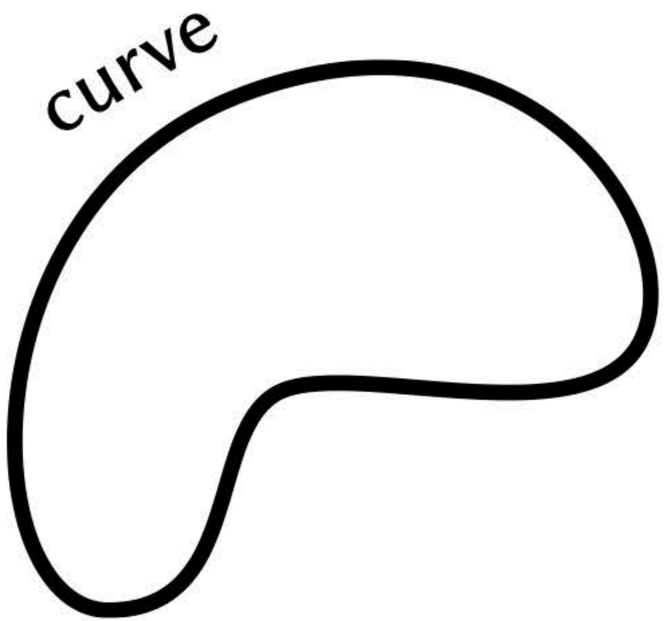


GENERALIZED SIGNED DISTANCE

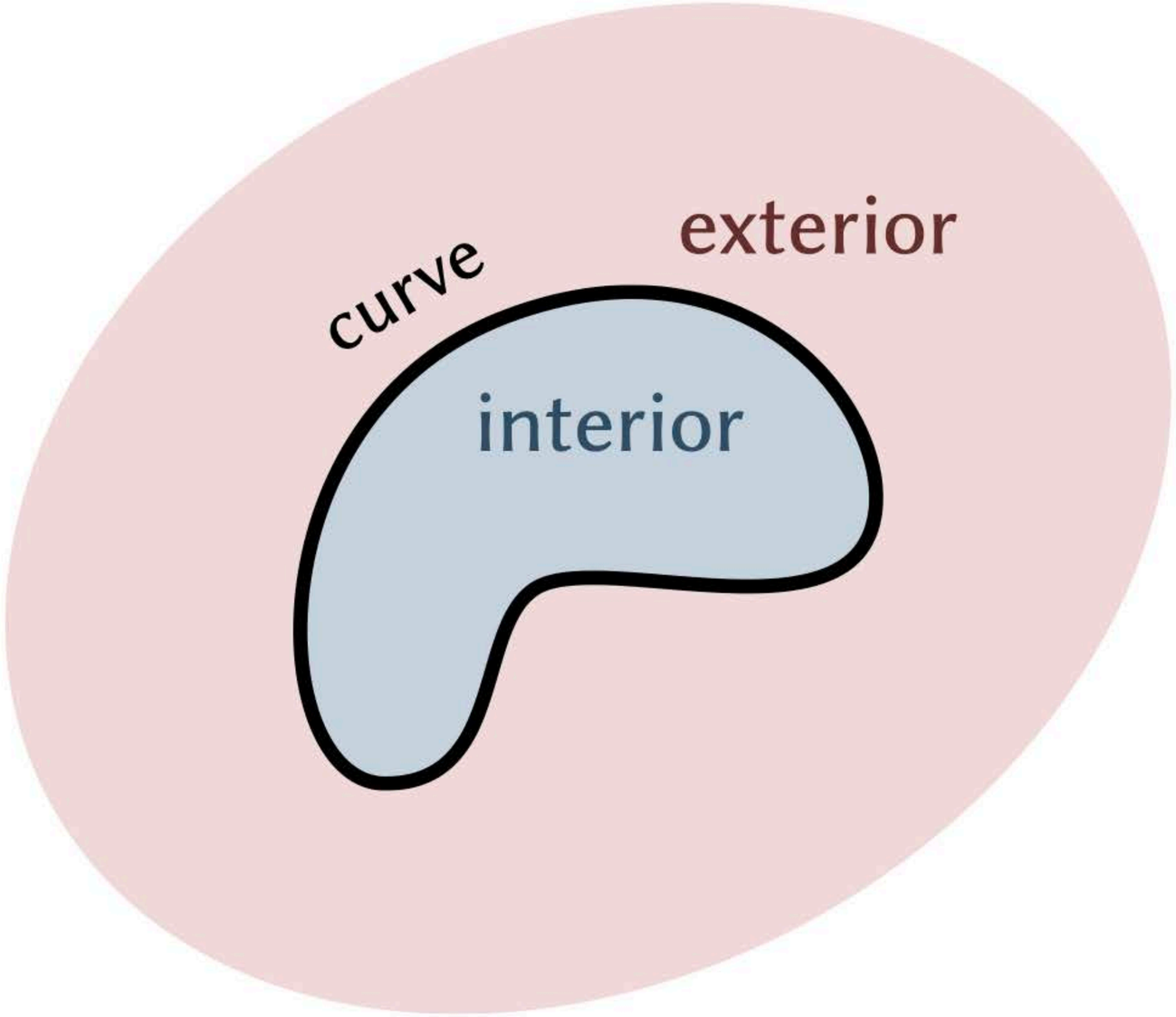
"generalized" = generalize from imperfect data to
answer queries about the underlying curve or surface

# Why inside/outside?

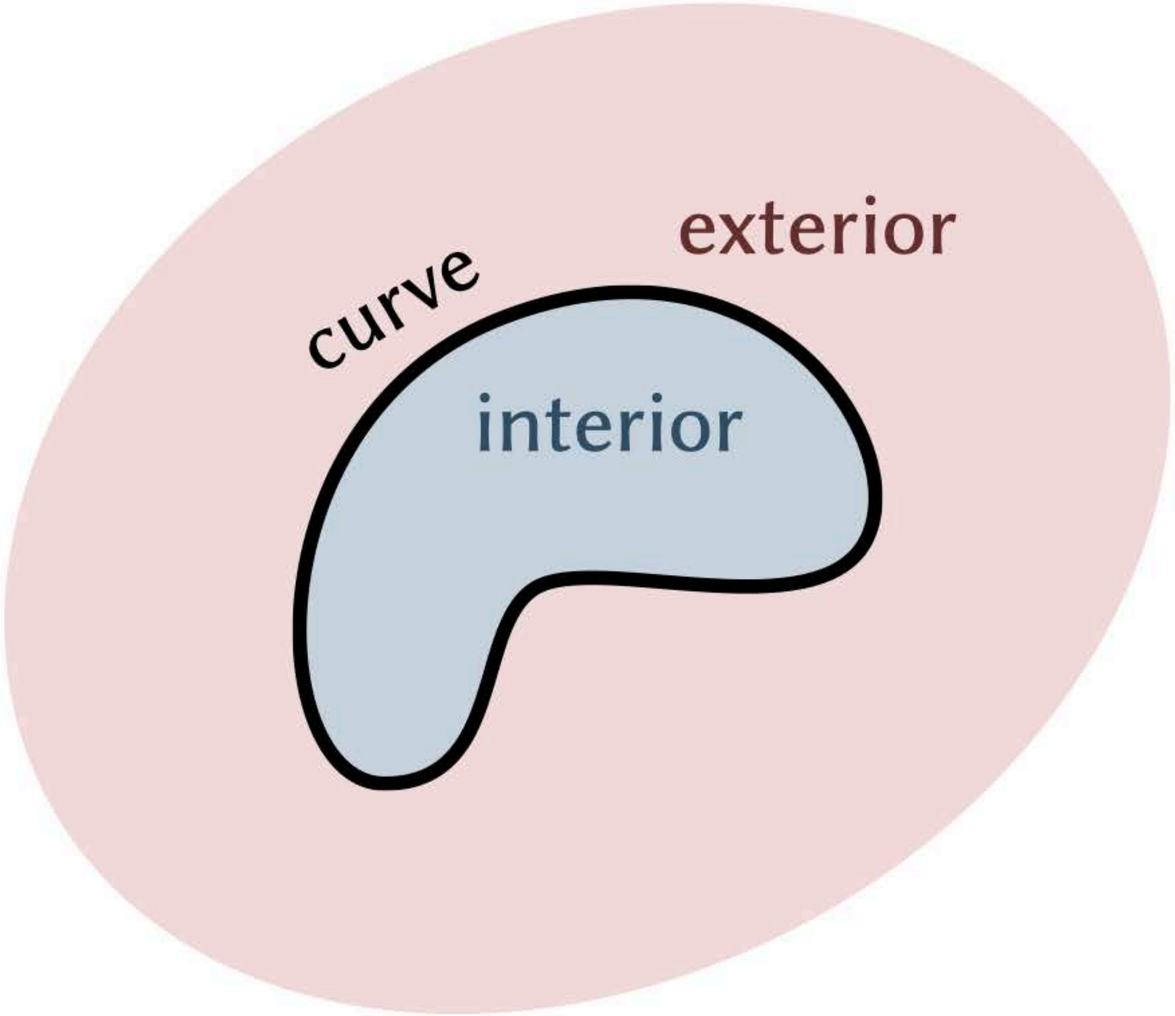
# Why inside/outside?



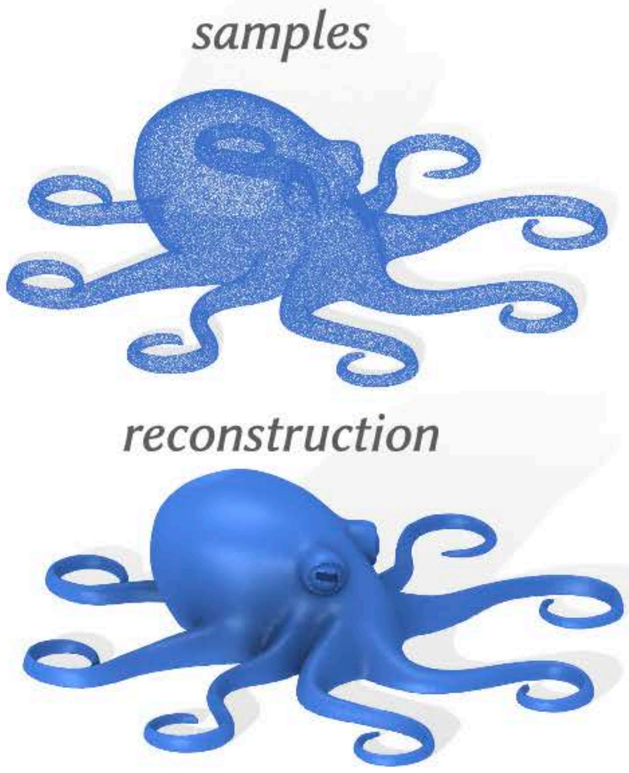
# Why inside/outside?



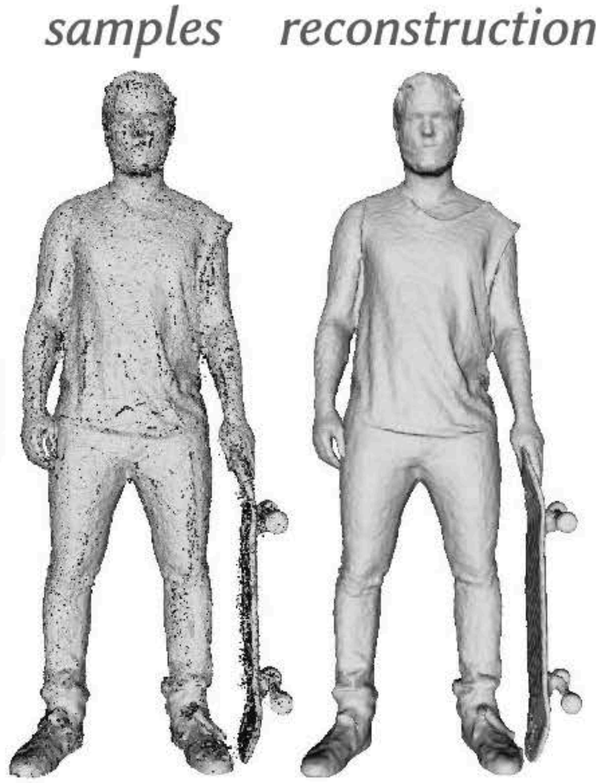
# Inside/outside is fundamental




"surface reconstruction"



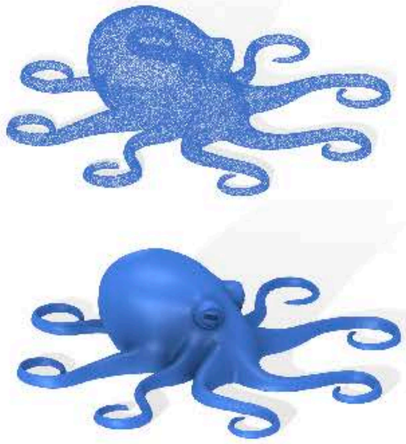

[Barill et al. 2018]



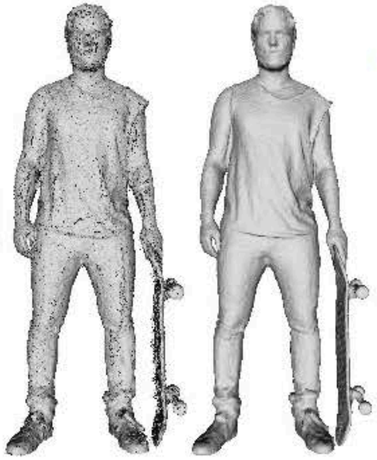

[Collet et al. 2015]

# Inside/outside is fundamental

## surface reconstruction

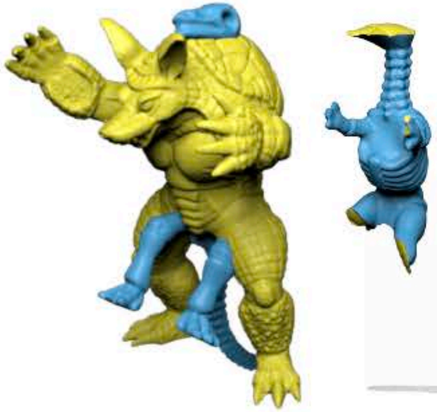


[Barill et al. 2018]



[Collet et al. 2015]

## mesh booleans



[Zhou et al. 2016]



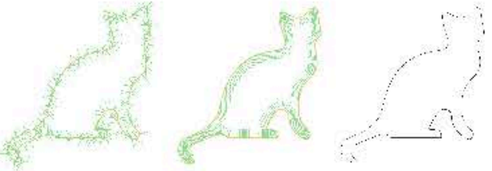
[Barill et al. 2018]

## orientation



[Xu et al. 2023]

[Hou et al. 2022]

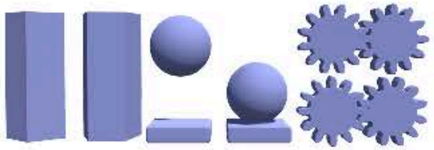


## geometric preprocessing

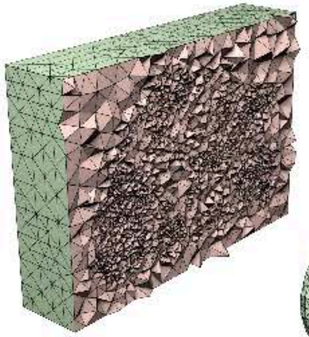


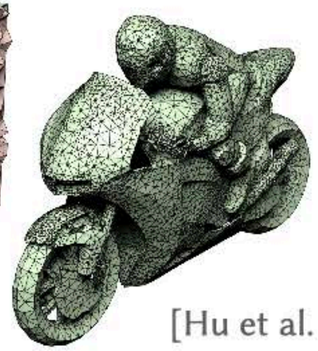
[Chi, Song 2021]

[Dvořák et al. 2021]



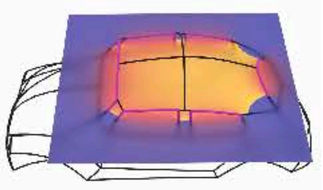
## meshing





[Hu et al. 2020]

## containment queries



[Martens & Bessmeltsev 2025]

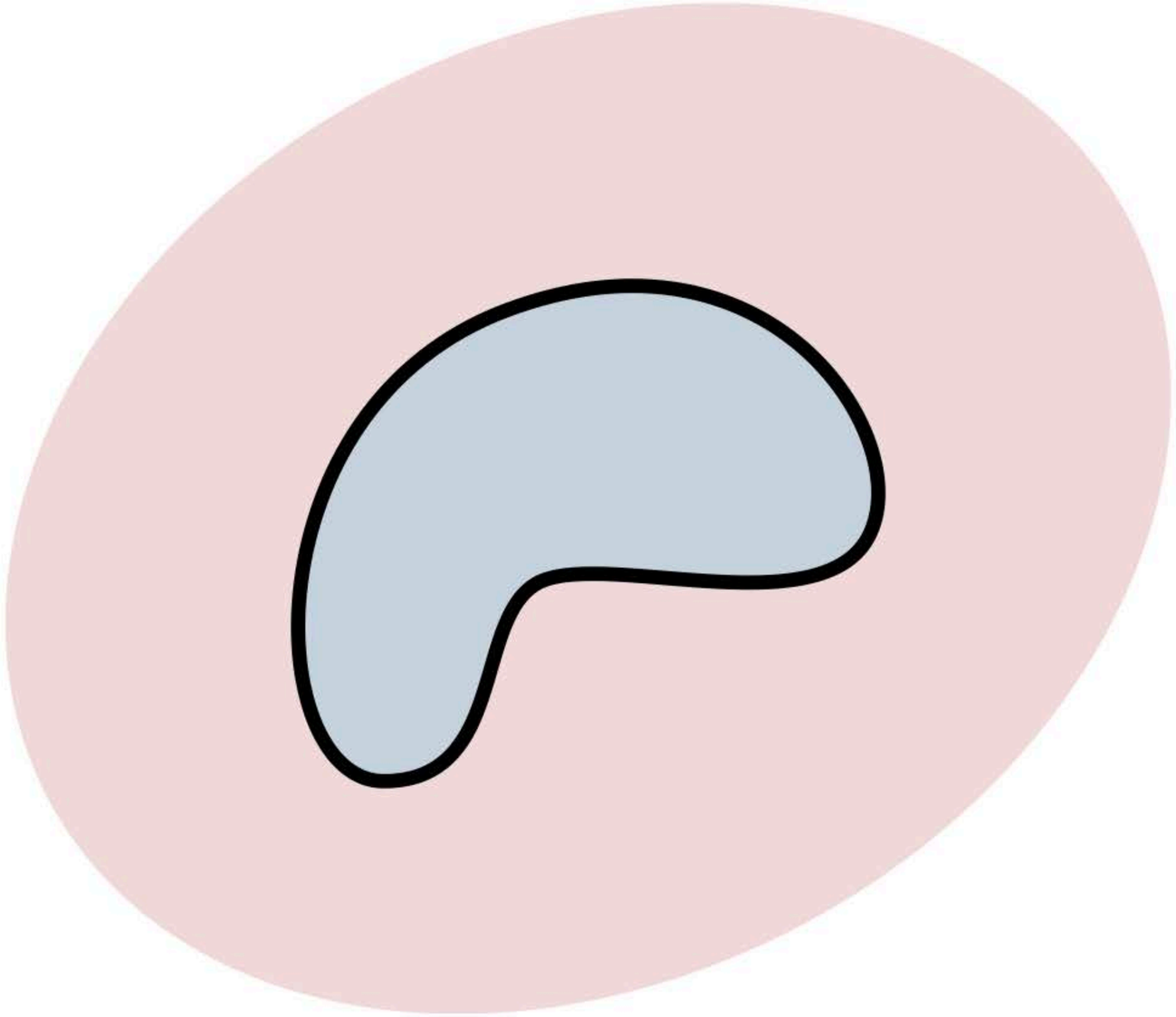


[Liu et al. 2025]

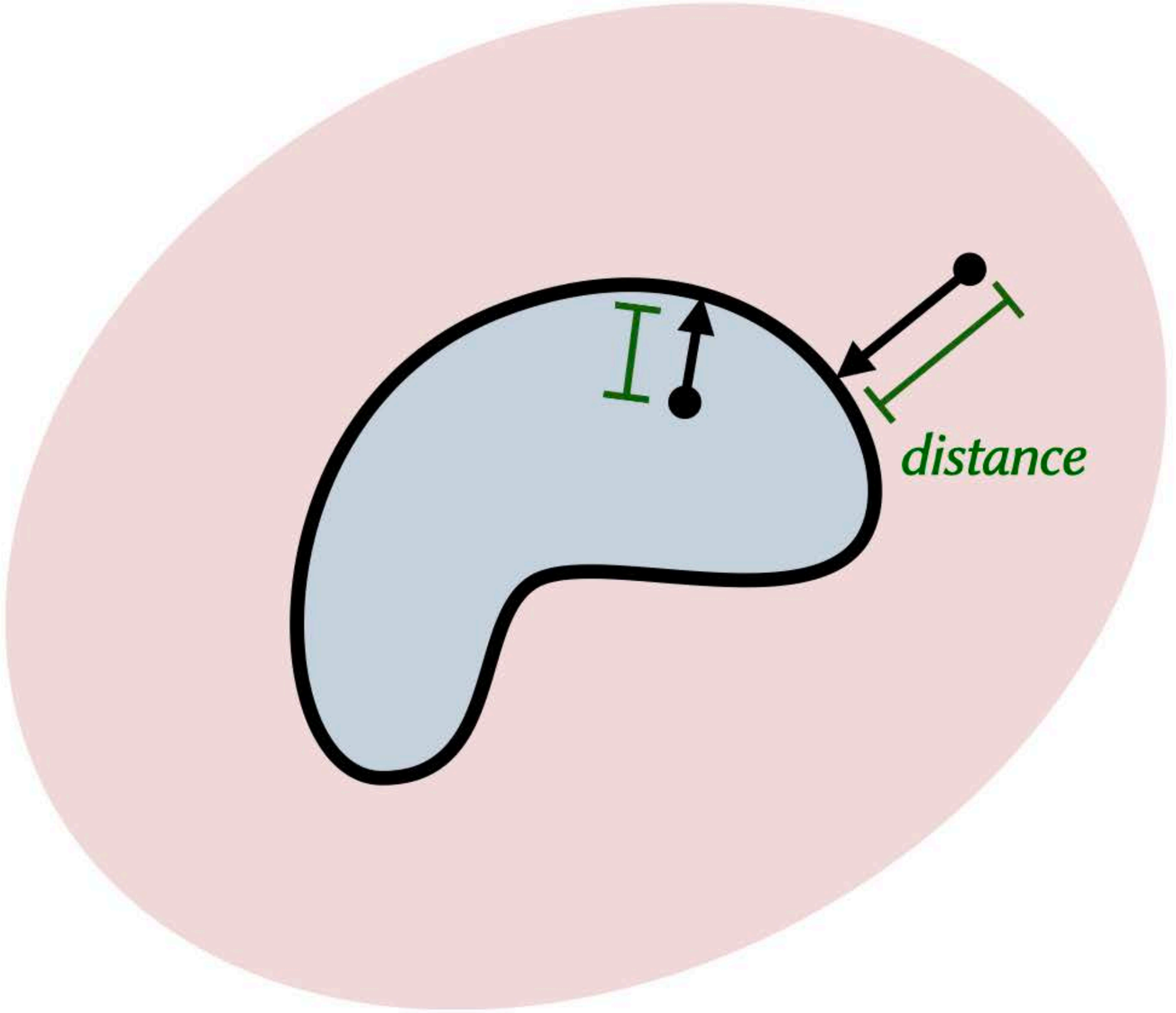


[Spainhour et al. 2024]

# Why distance?



# Why distance?



# Why distance?

*modeling*

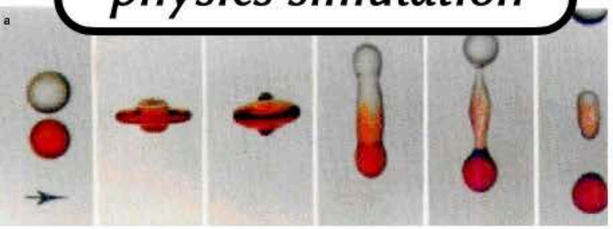


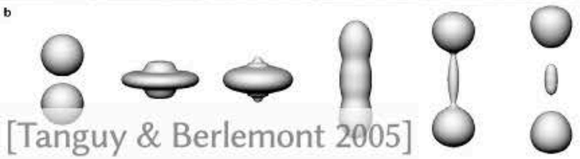
[Frisken & Perry 2006]



[Chou et al. 2023]

*physics simulation*



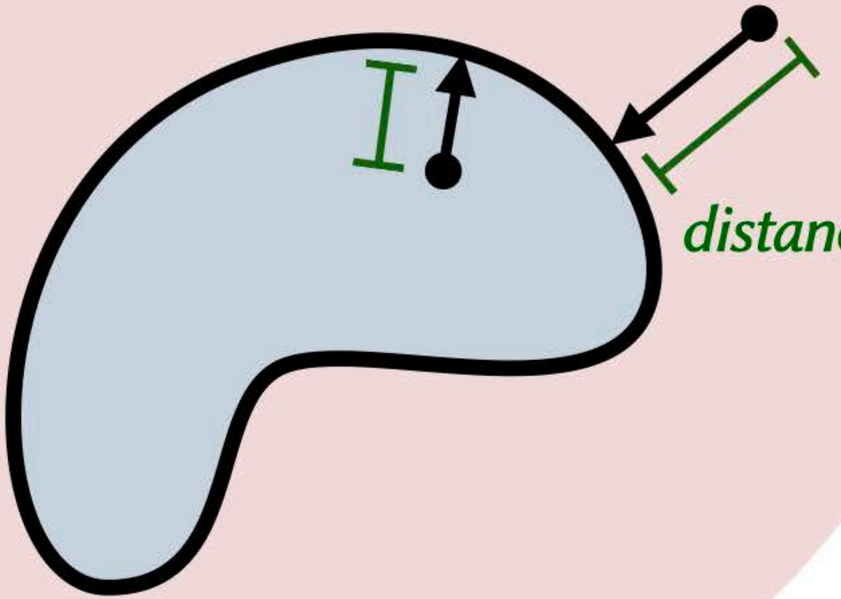


[Tanguy & Berlemont 2005]

*shape optimization*



[Vicini et al. 2022]

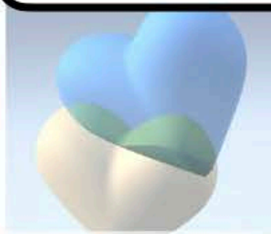



*rendering*



[Quilez 2008]

*collision detection*

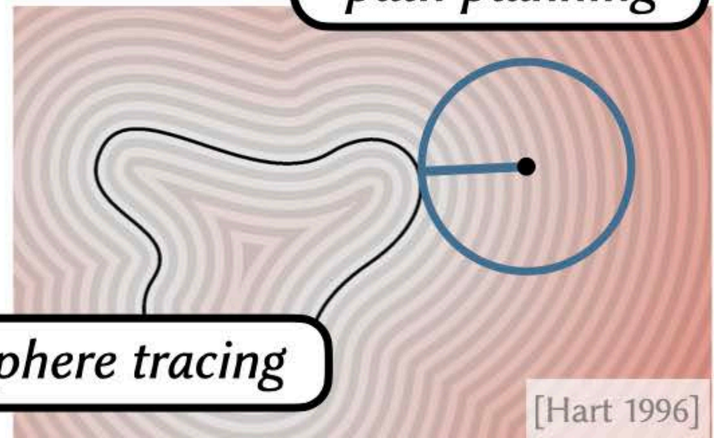


[Liu et al. 2024]



[Oleynikova et al. 2016]

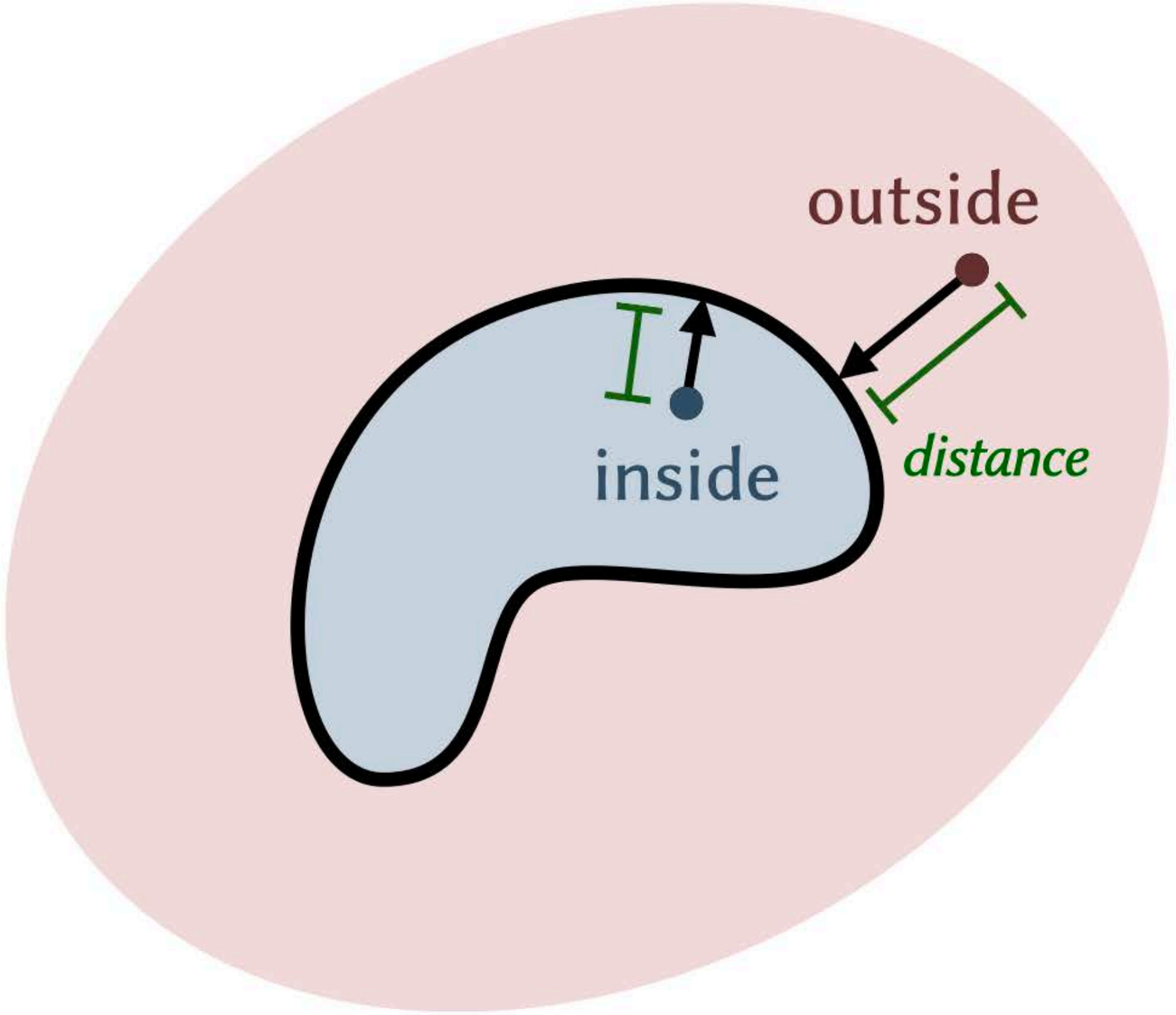
*path planning*



*sphere tracing*

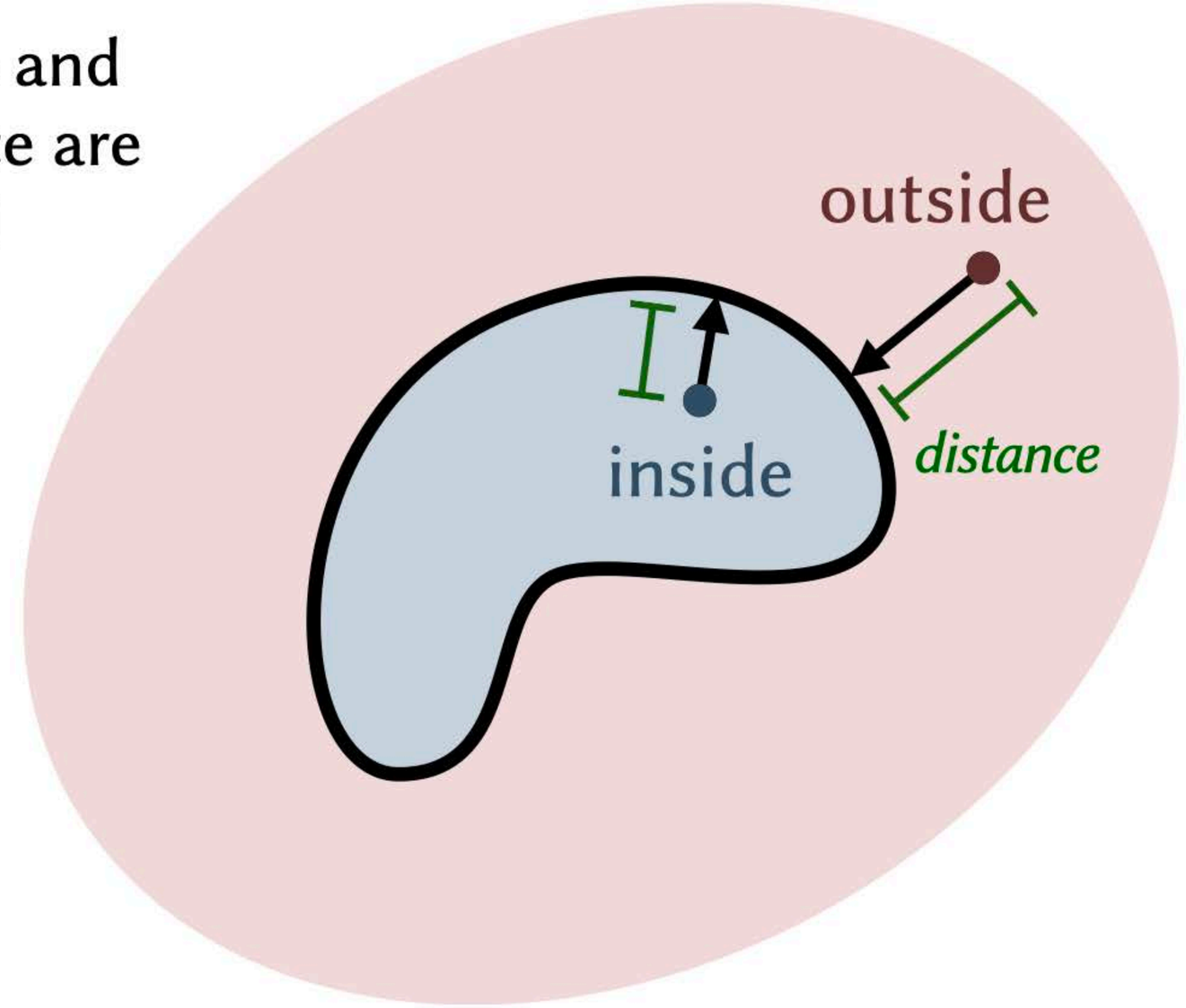
[Hart 1996]

# Why signed distance?



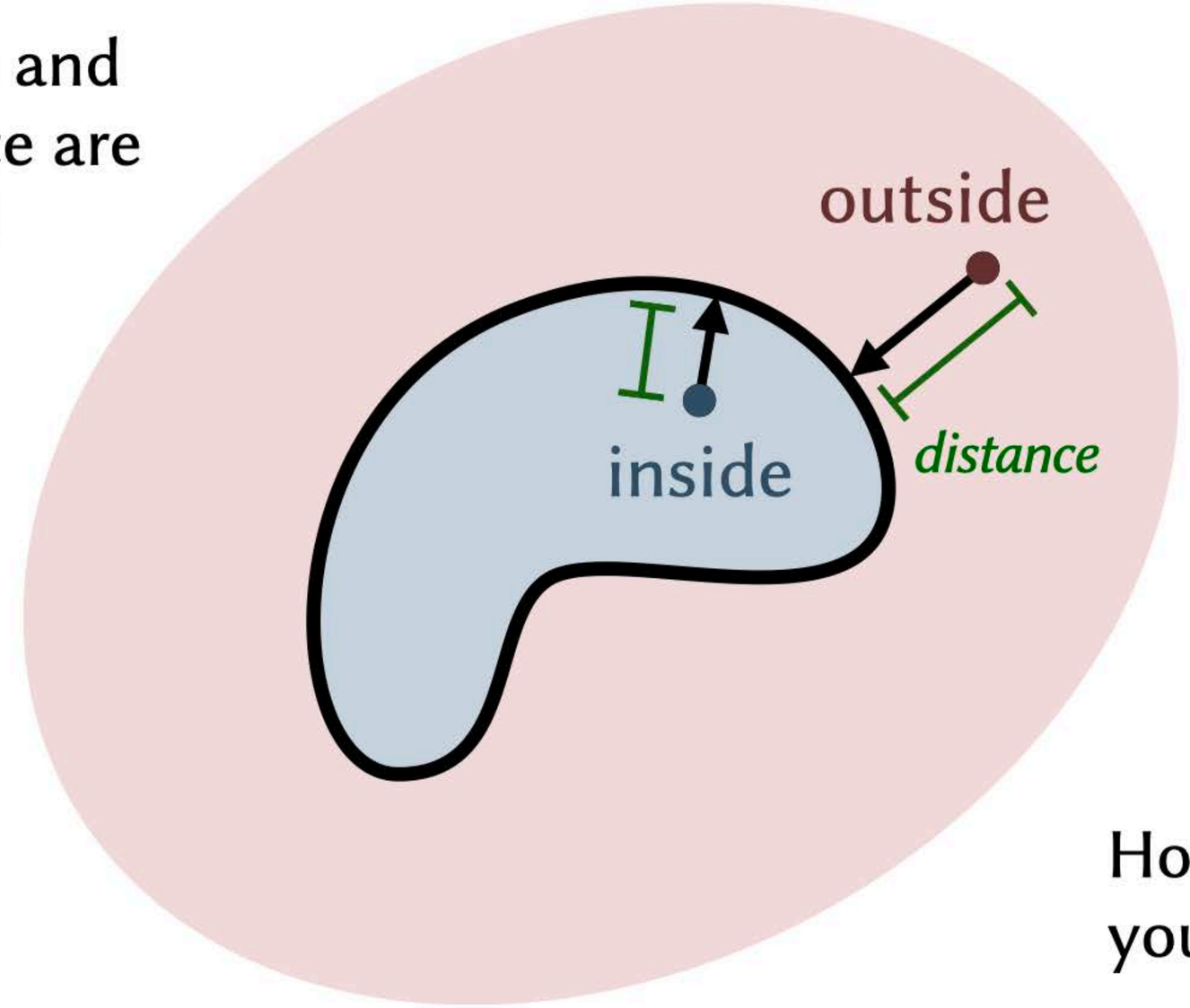
# Why signed distance?

Inside/outside and signed distance are closely related



# Why signed distance?

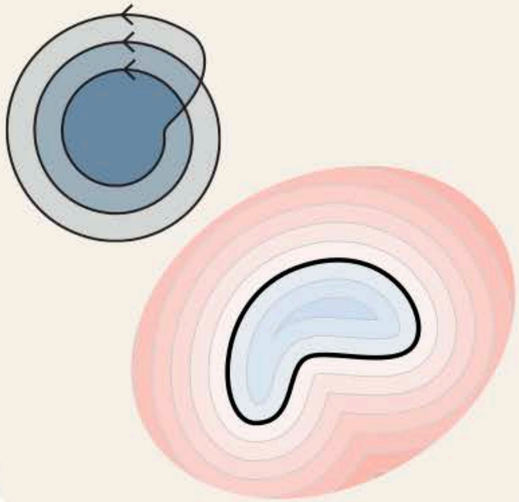
Inside/outside and signed distance are closely related



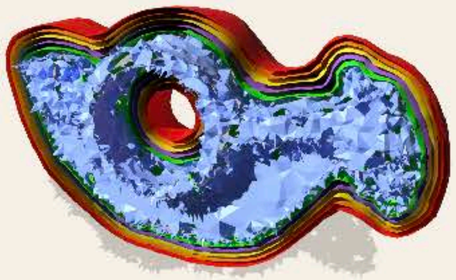

How can/should you do both?

# Outline

**Background:**
winding numbers &
signed distance

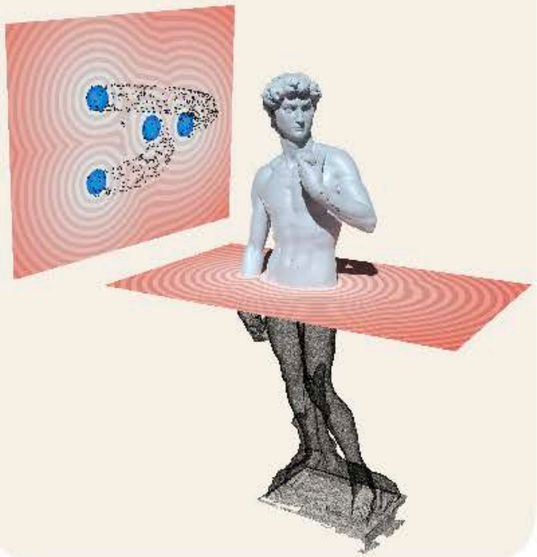


**Generalized signed distance**

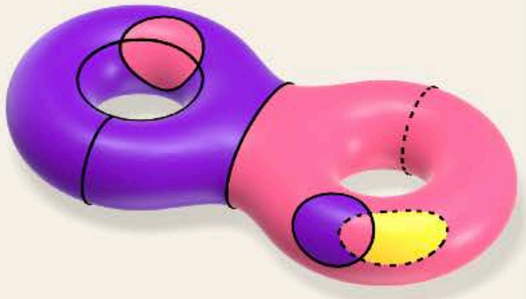


[**Feng** & Crane 2024. *A Heat Method for Generalized Signed Distance*]

**Generalized signed distance, but faster**



**Generalized inside/outside**

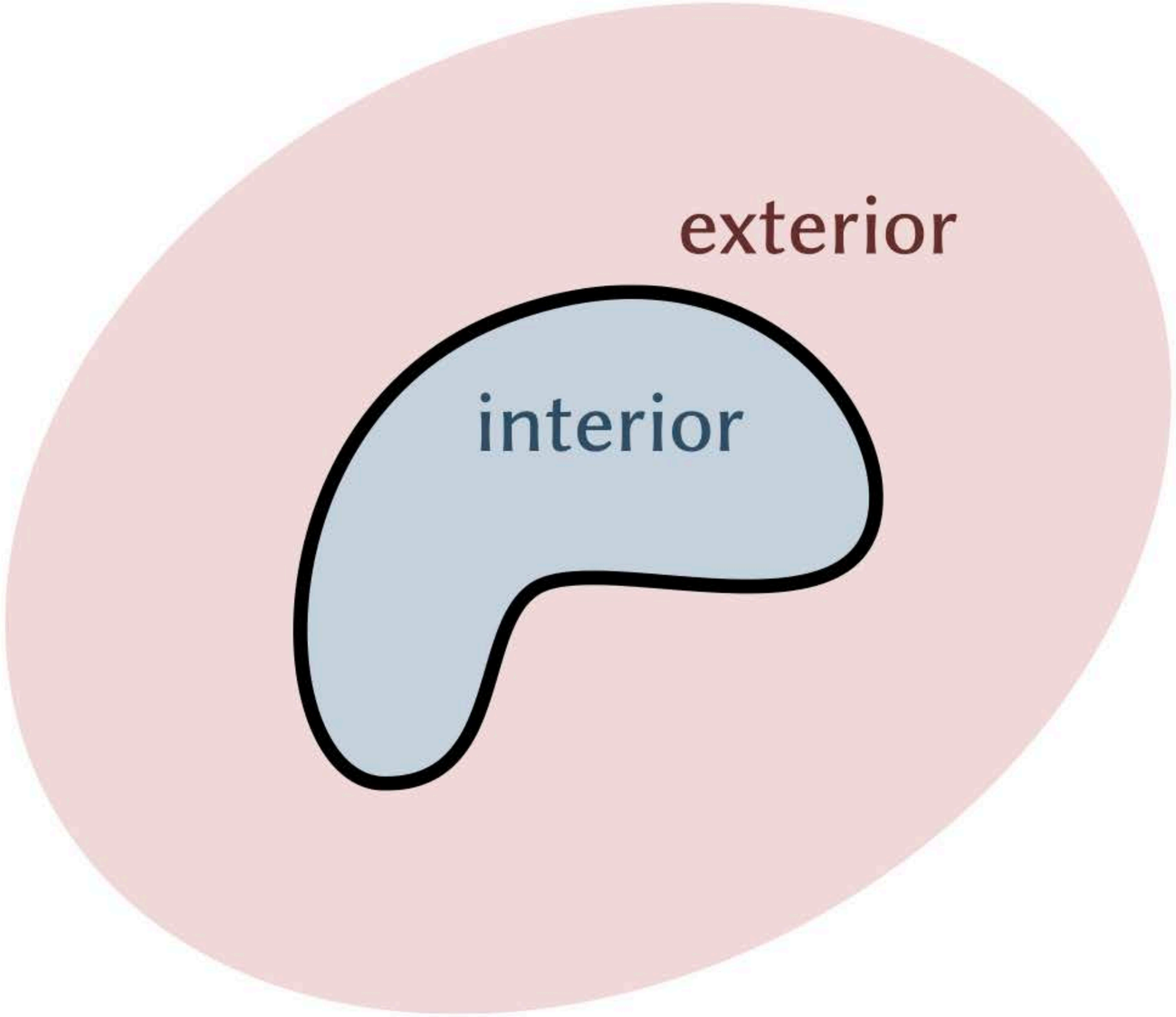


[**Feng**, Gillespie, Crane 2023. *Winding Numbers on Discrete Surfaces*]

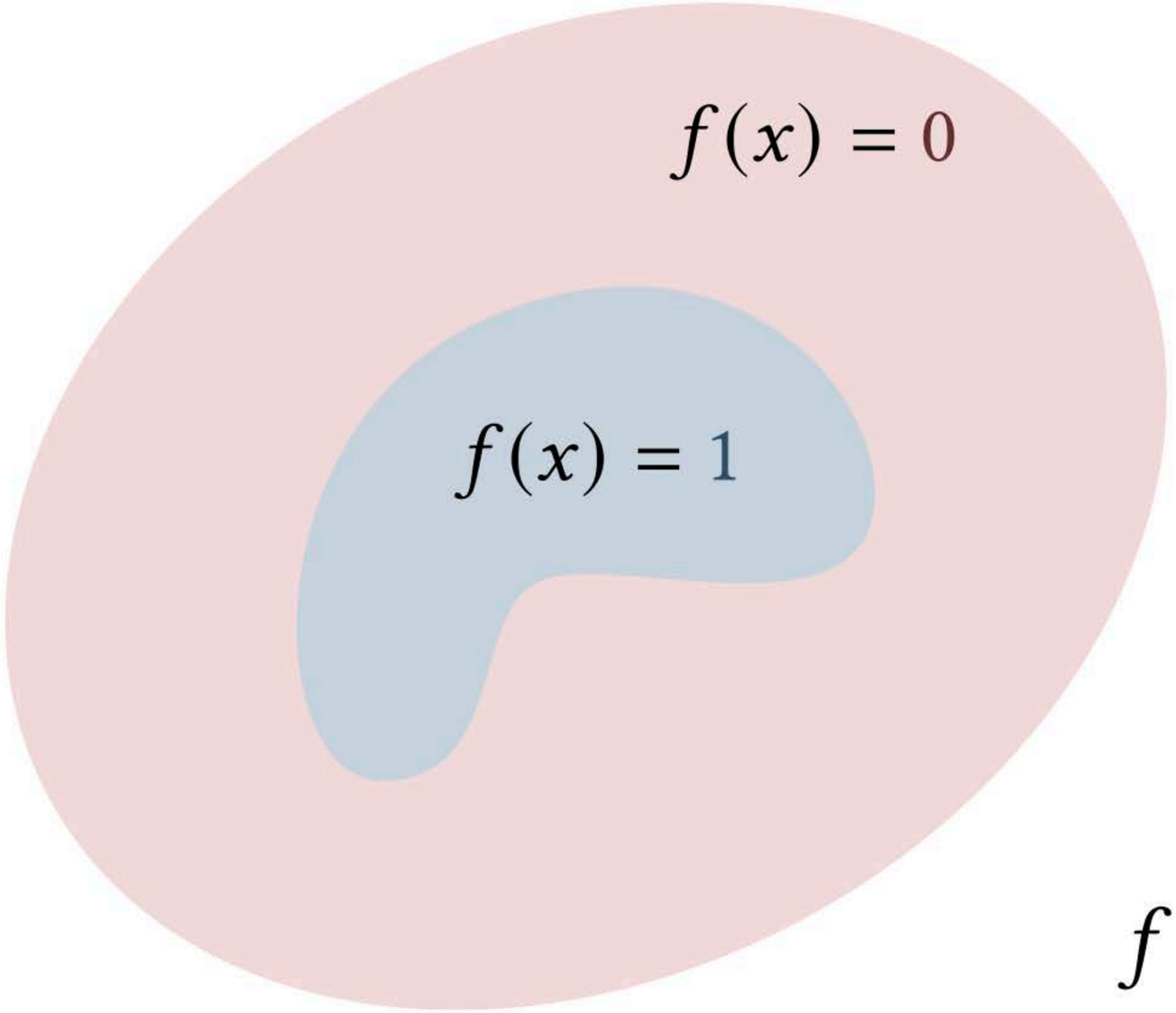
[**Feng**, Gkioulekas, Crane 2026. *Points as Tori: Fast Pointwise Signed Distance for Point Clouds*]

# WINDING NUMBERS & SIGNED DISTANCE

# Inside/outside as a function

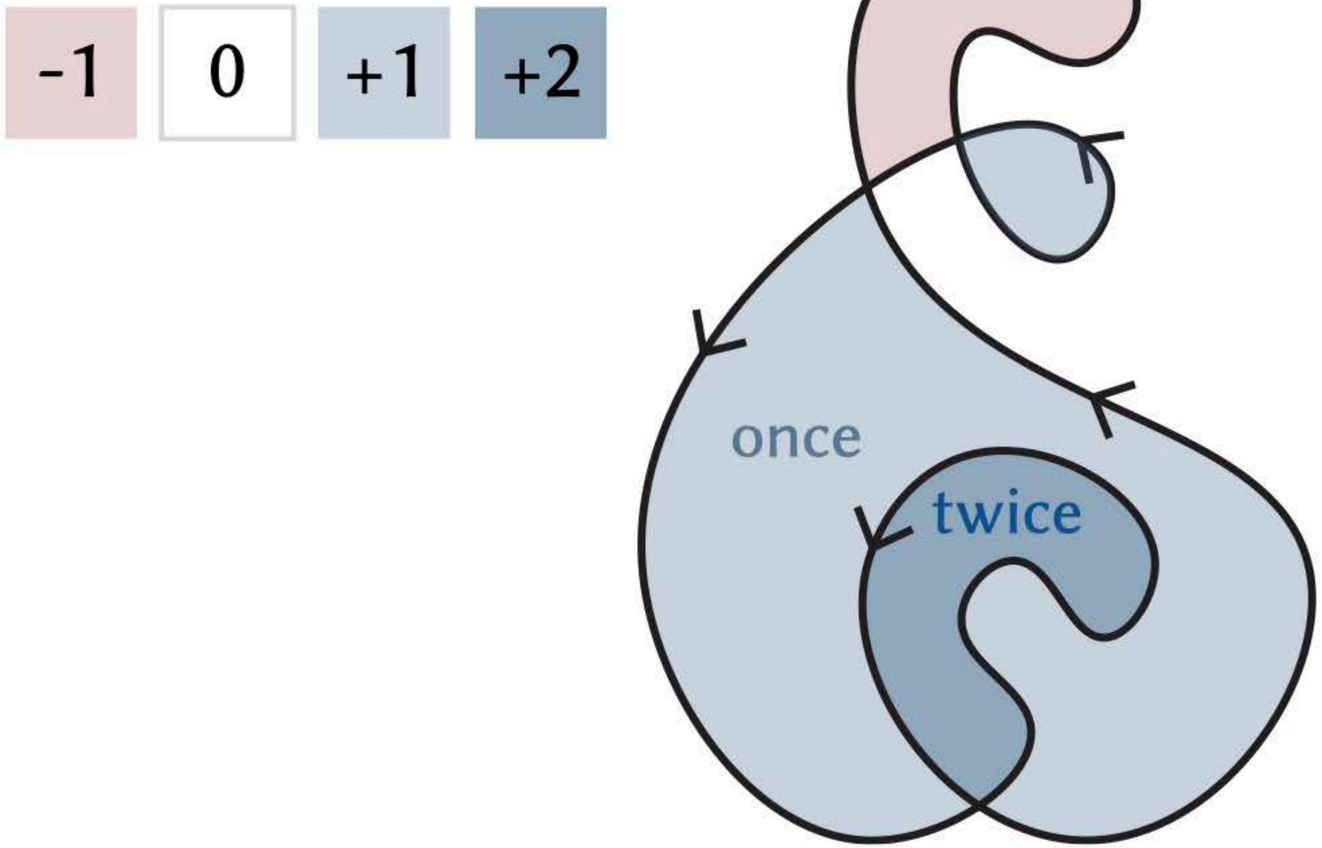


# Inside/outside as a function




$$f : \mathbb{R}^n \to \mathbb{R}$$

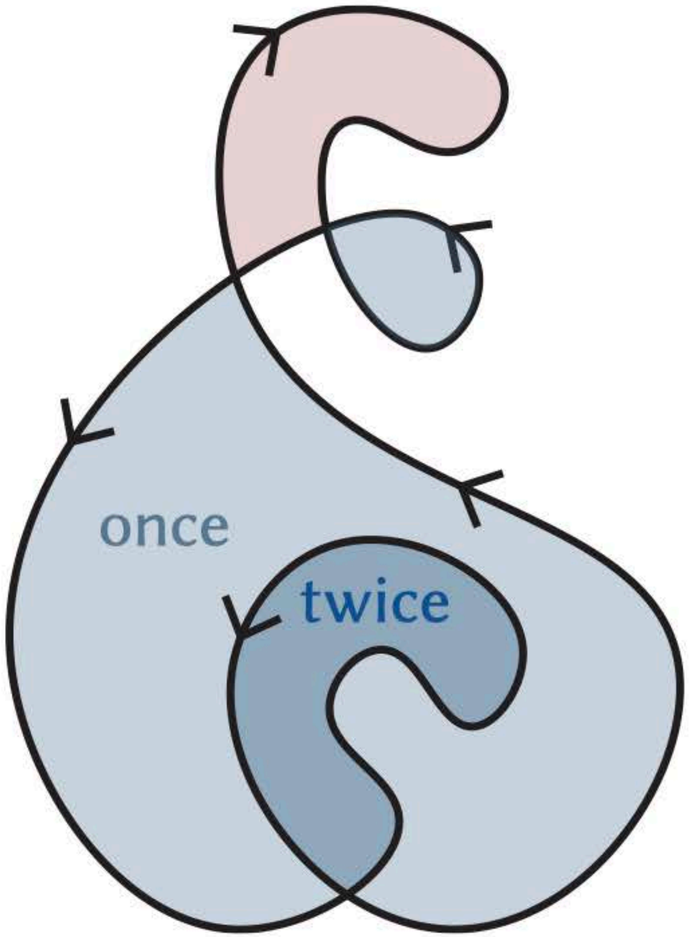
# Curves can enclose points multiple times



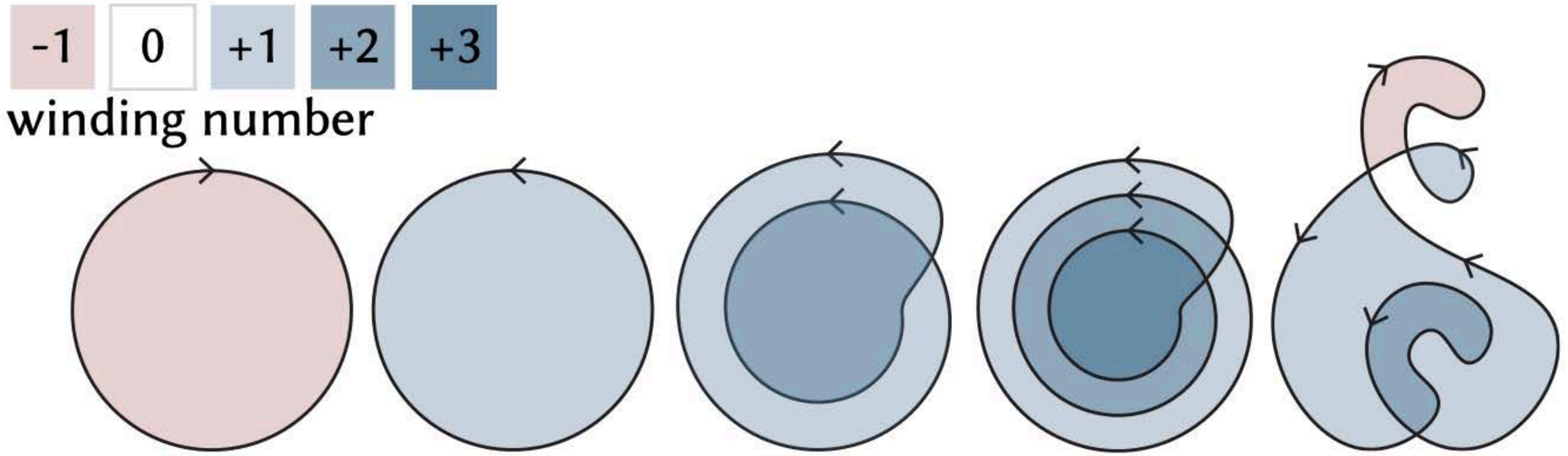
# Curves can enclose points multiple times

| -1 | 0 | +1 | +2 |
|---|---|---|---|

winding number

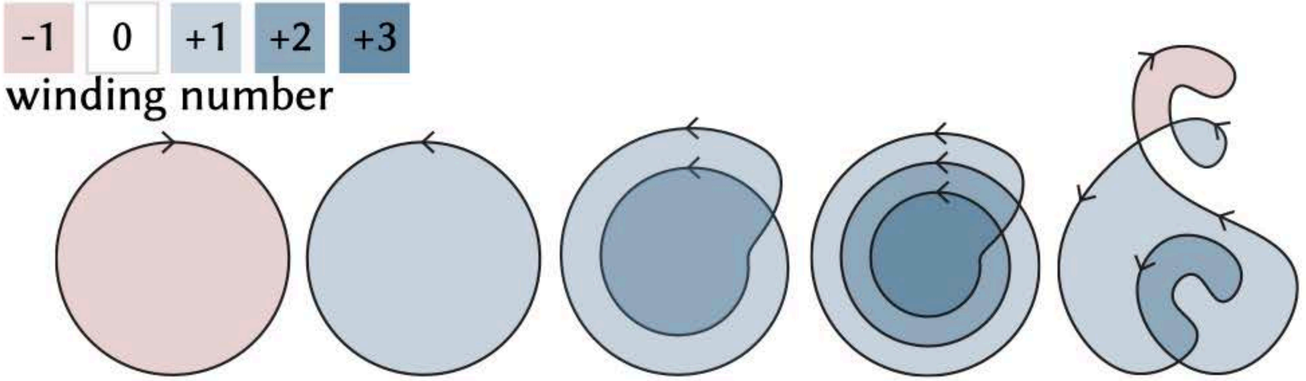


# Winding numbers across history




Complex analysis, differential geometry, topology, electromagnetism…

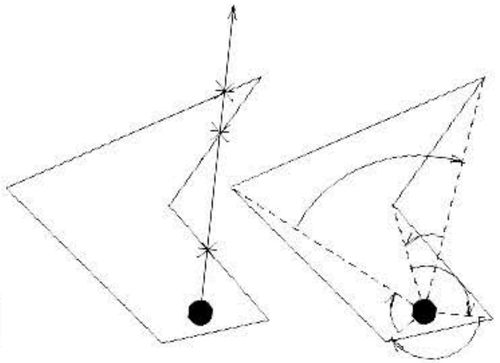
# Winding numbers across history




Complex analysis, differential geometry, topology, electromagnetism…

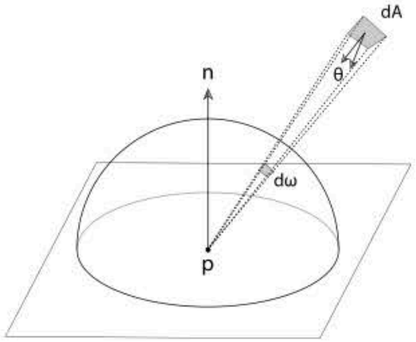
## Winding number in graphics

[Shimrat 1962; Haines 1994; Goral et al. 1984; Veach & Guibas 1995…]



[Haines 1994]

*point-in-polygon tests*

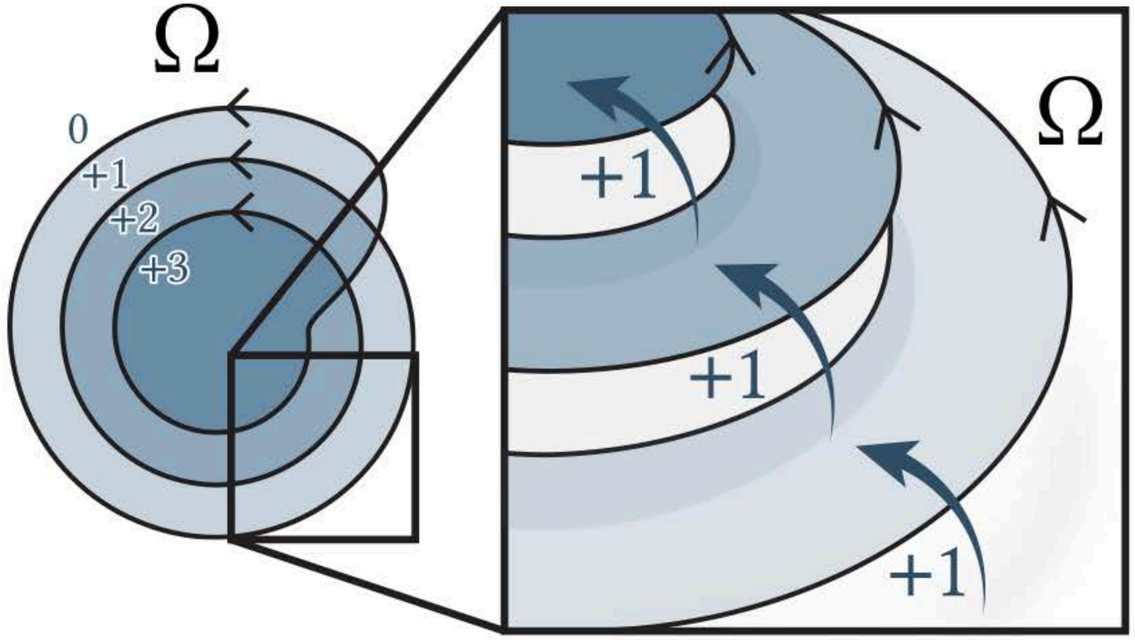



[Pharr et al. 2018]

*solid angle*

# Winding number is a function with “jumps”

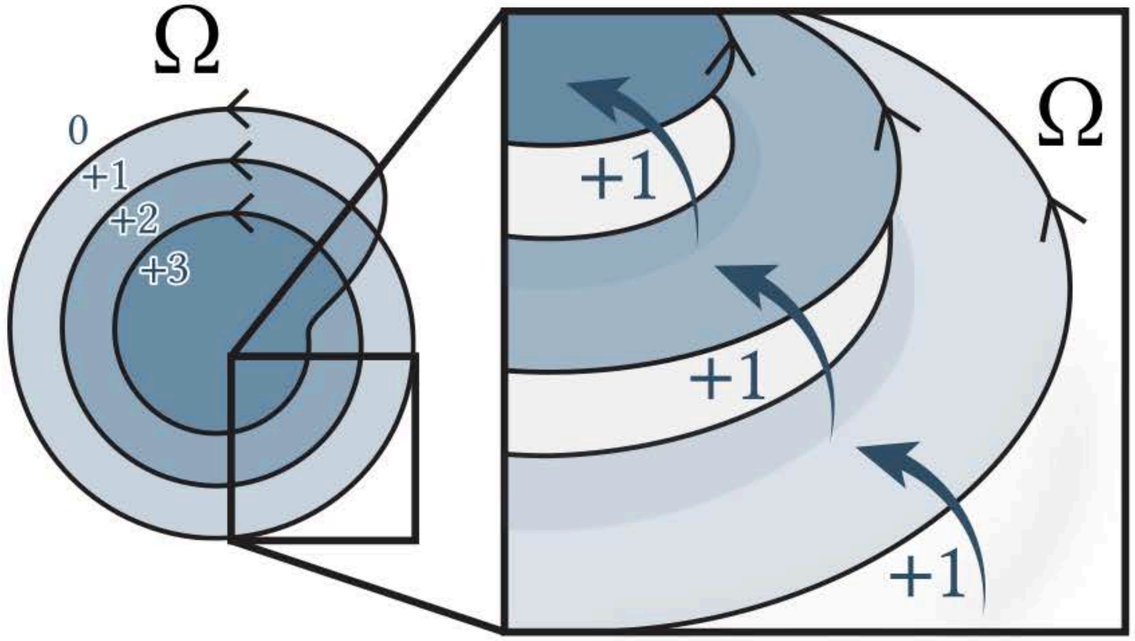
$$u^+ - u^- = 1, \qquad \text{on } \Omega,$$




*The jump problem for the Laplace equation.* Krutitskii (2001)

# Winding number is a function with "jumps"

jump $$u^+ - u^- = 1, \quad \text{on } \Omega,$$

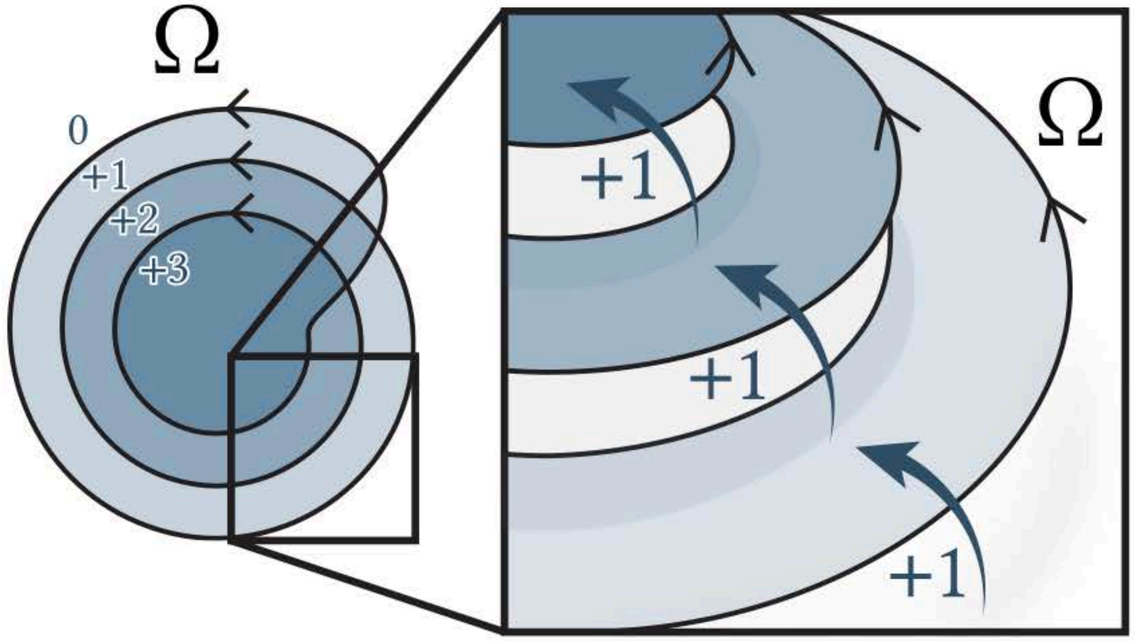



*The jump problem for the Laplace equation.* Krutitskii (2001)

# Winding number is a function with “jumps”

$$\begin{aligned} \Delta u &= 0, && \text{on } M \setminus \Omega, \\ u^+ - u^- &= 1, && \text{on } \Omega, \\ \partial u^+ / \partial n &= \partial u^- / \partial n, && \text{on } \Omega. \end{aligned}$$

jump



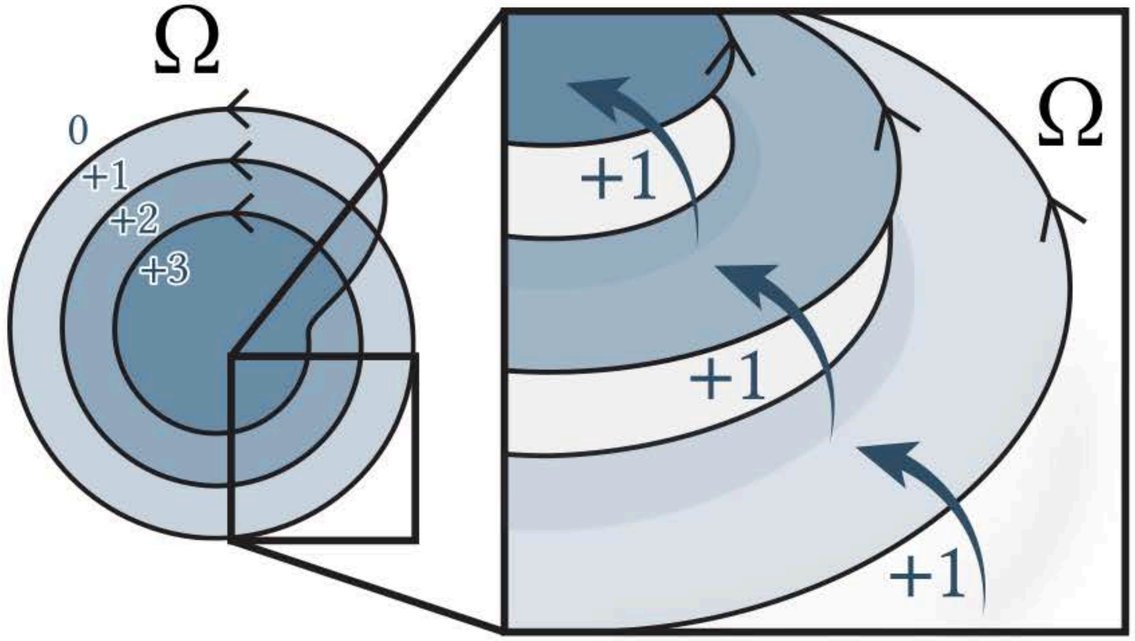

*The jump problem for the Laplace equation.* Krutitskii (2001)

# Winding number is a function with “jumps”

harmonic

jump

$$\begin{aligned} \Delta u &= 0, && \text{on } M \setminus \Omega, \\ u^+ - u^- &= 1, && \text{on } \Omega, \\ \partial u^+ / \partial n &= \partial u^- / \partial n, && \text{on } \Omega. \end{aligned}$$




*The jump problem for the Laplace equation.* Krutitskii (2001)

# Solving for winding number

$$
\begin{aligned}
\Delta u &= 0, && \text{on } M \setminus \Omega, \\
u^+ - u^- &= 1, && \text{on } \Omega, \\
\partial u^+ / \partial n &= \partial u^- / \partial n, && \text{on } \Omega.
\end{aligned}
$$

jump harmonic function



Feng, Gillespie, Crane 2023.
*Winding Numbers on Discrete Surfaces*

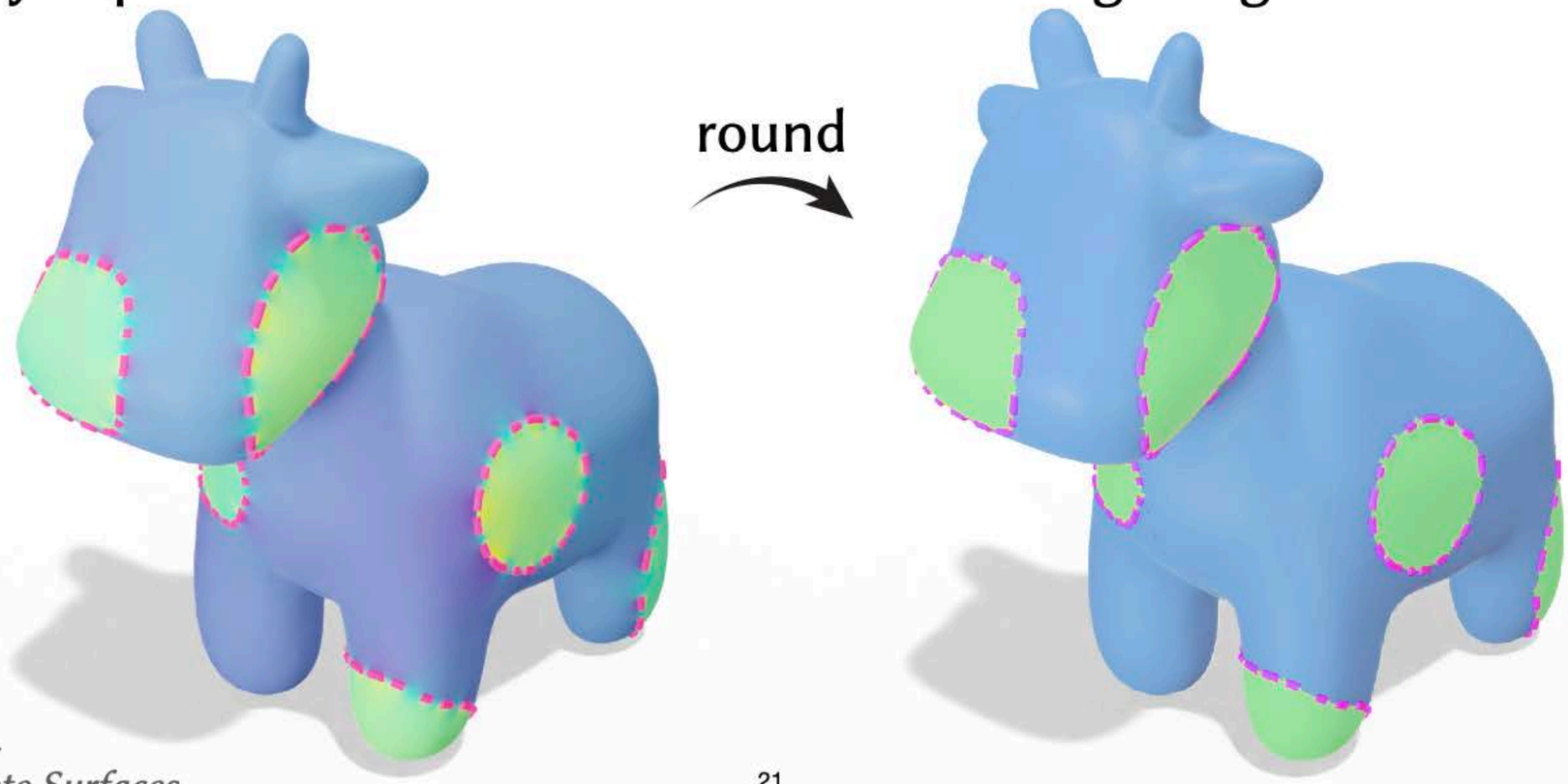
# Solving for winding number

$$\begin{aligned} \Delta u &= 0, && \text{on } M \setminus \Omega, \\ u^+ - u^- &= 1, && \text{on } \Omega, \\ \partial u^+ / \partial n &= \partial u^- / \partial n, && \text{on } \Omega. \end{aligned}$$

jump harmonic function

round

integer region labels



Feng, Gillespie, Crane 2023.
*Winding Numbers on Discrete Surfaces*

# Solving for winding number

jump harmonic equation

$$\begin{aligned} \Delta u &= 0, && \text{on } M \setminus \Omega, \\ u^+ - u^- &= 1, && \text{on } \Omega, \\ \partial u^+ / \partial n &= \partial u^- / \partial n, && \text{on } \Omega. \end{aligned}$$



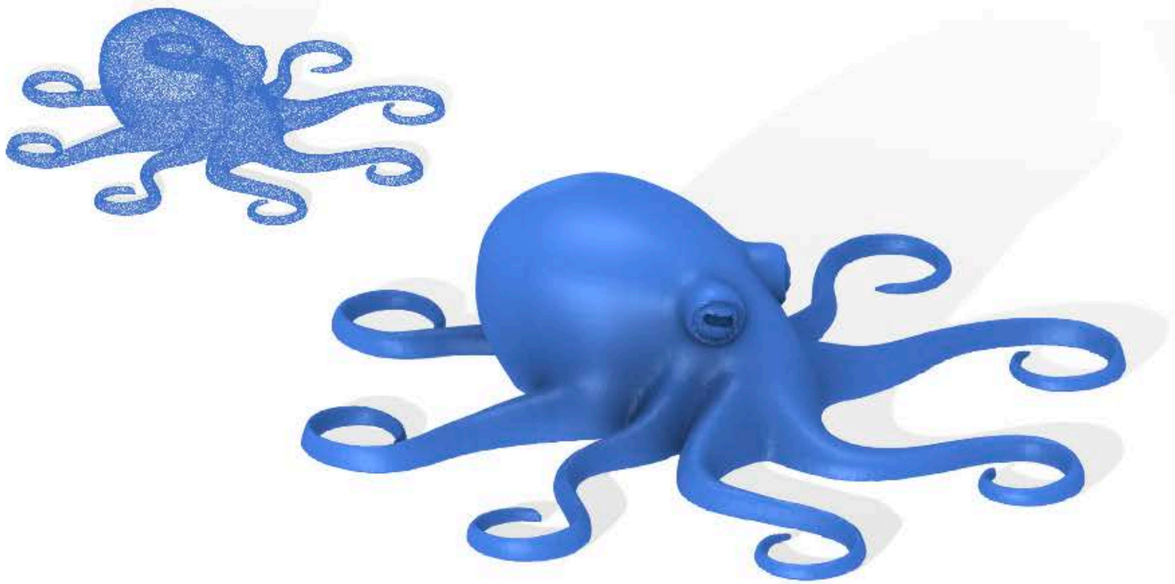
boundary integral

$$u(x) = \int_\Omega \frac{\partial G(x,z)}{\partial n(x)} \, \mathrm{d}z$$

# Solving for winding number

jump harmonic equation

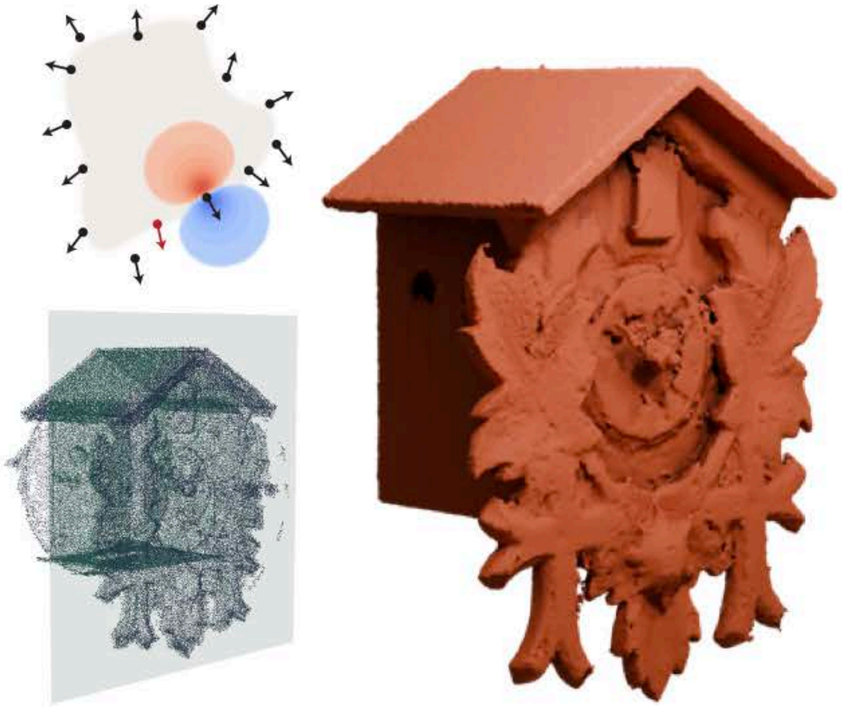
$$\begin{aligned} \Delta u &= 0, && \text{on } M \setminus \Omega, \\ u^+ - u^- &= 1, && \text{on } \Omega, \\ \partial u^+ / \partial n &= \partial u^- / \partial n, && \text{on } \Omega. \end{aligned}$$



*Fast Winding Numbers for Soups and Clouds*
[Barill et al. 2018]

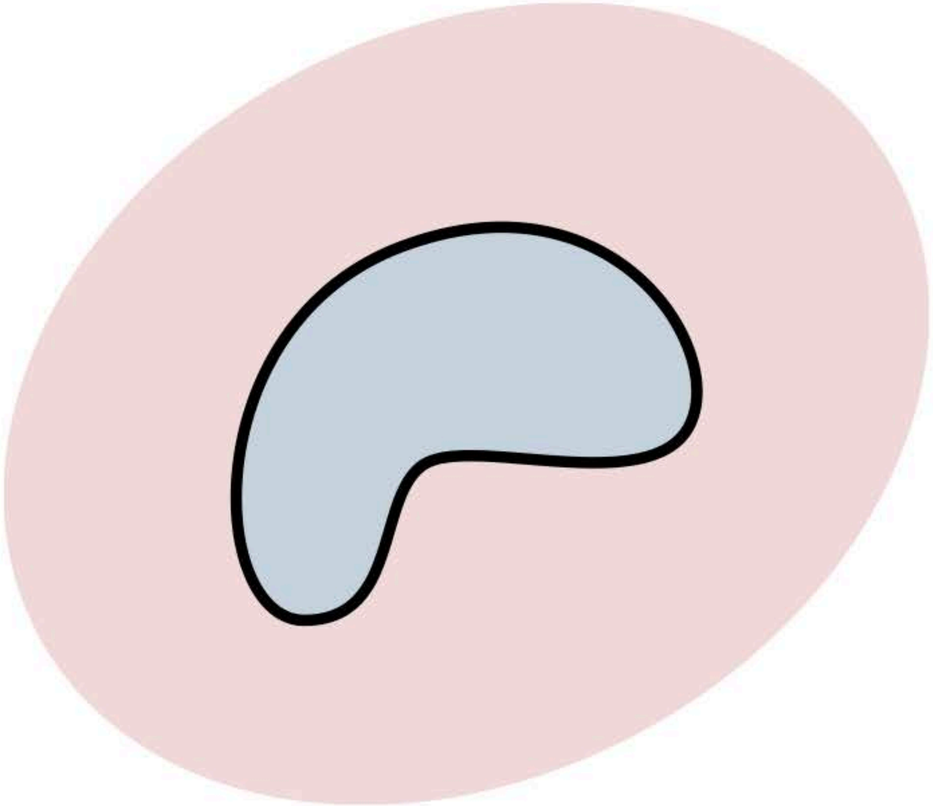
boundary integral

$$u(x) = \int_\Omega \frac{\partial G(x,z)}{\partial n(x)} \, \mathrm{d}z$$

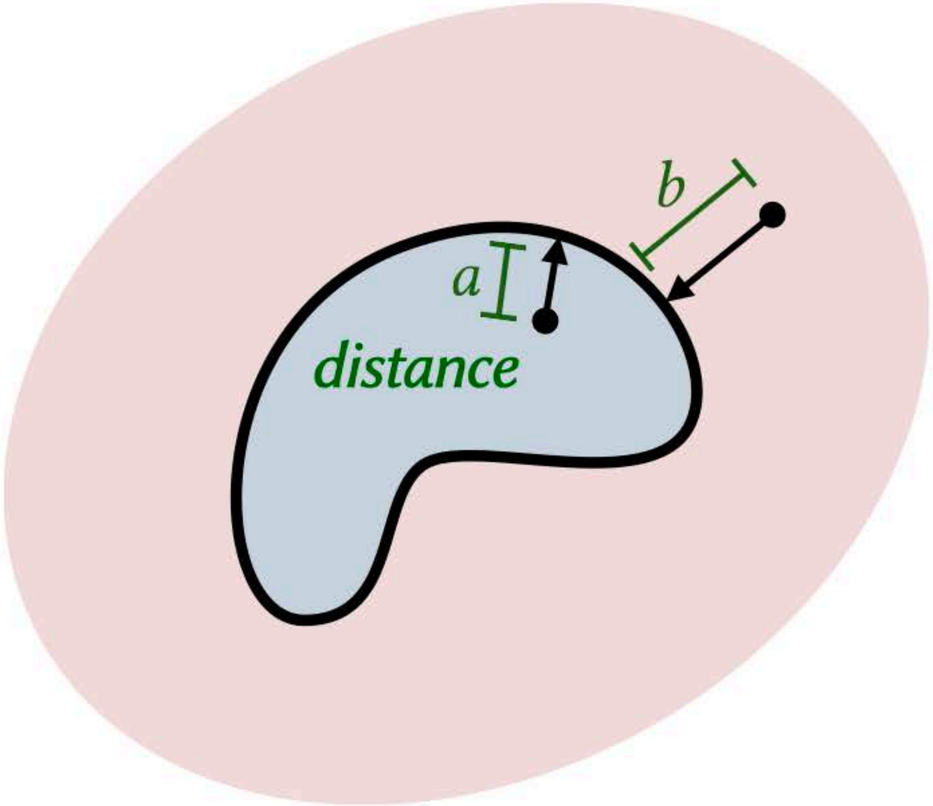


*3D Reconstruction with Fast Dipole Sums*
[Chen et al. 2024]

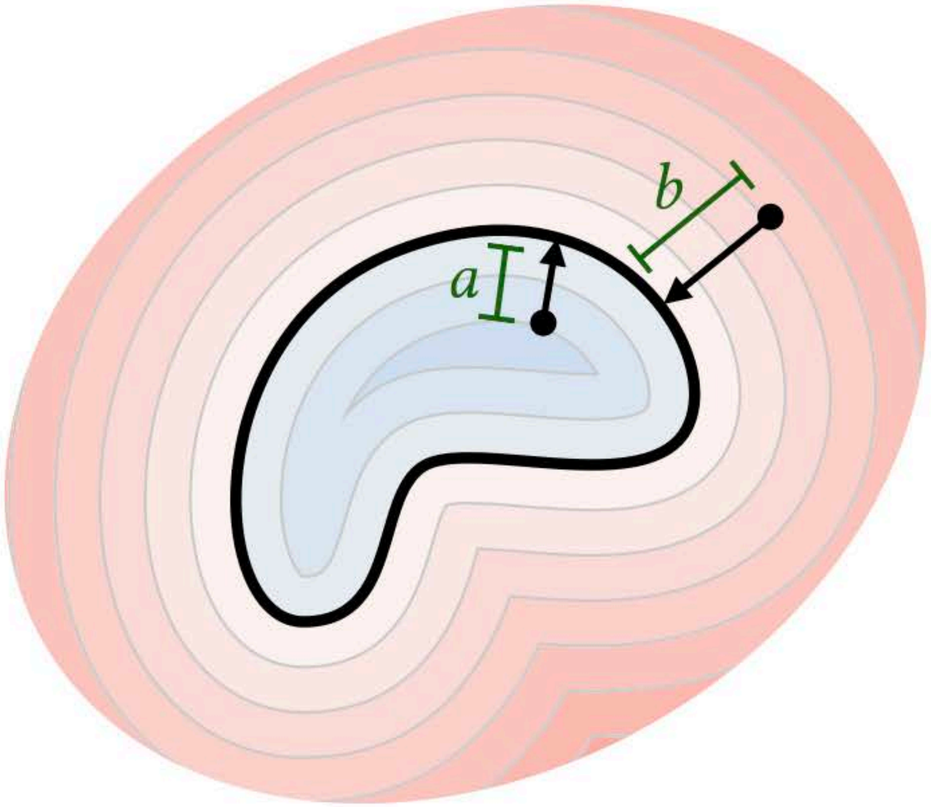
# Distance



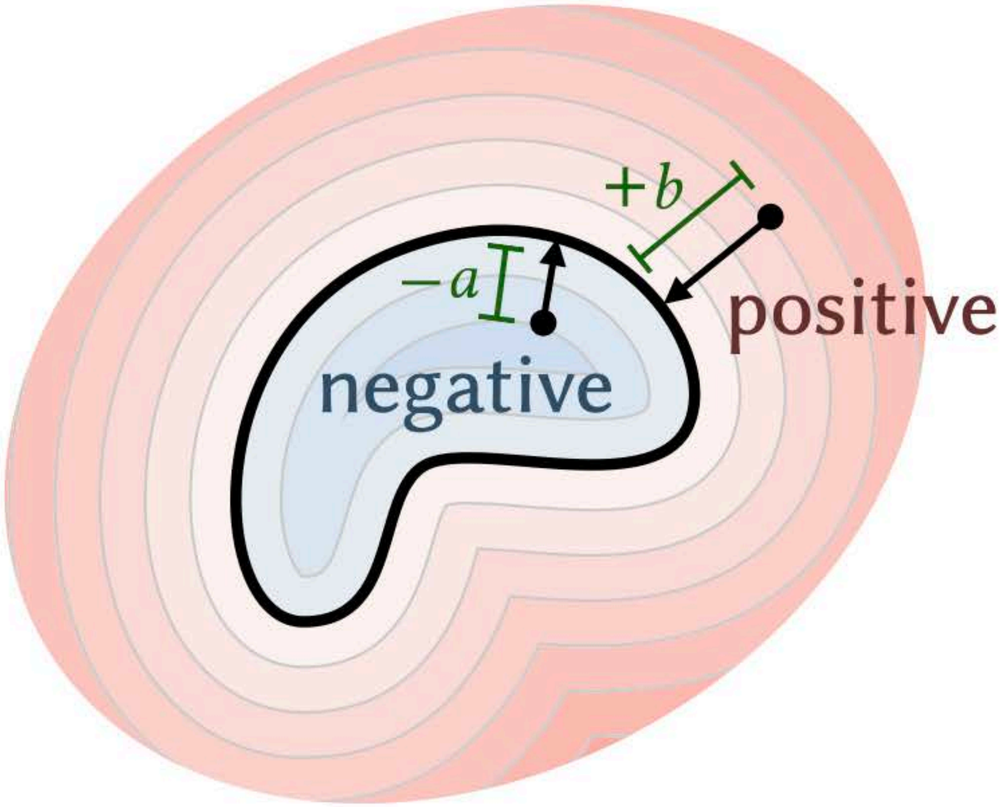
# Distance



# Signed distance



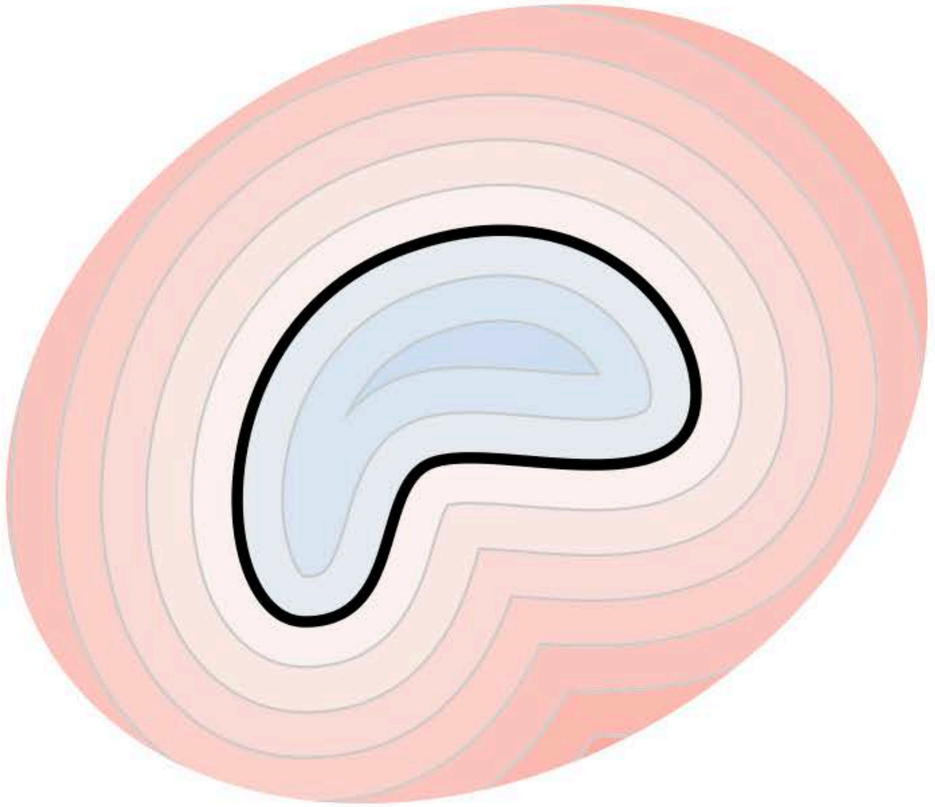
# Signed distance



# Signed distance

eikonal equation

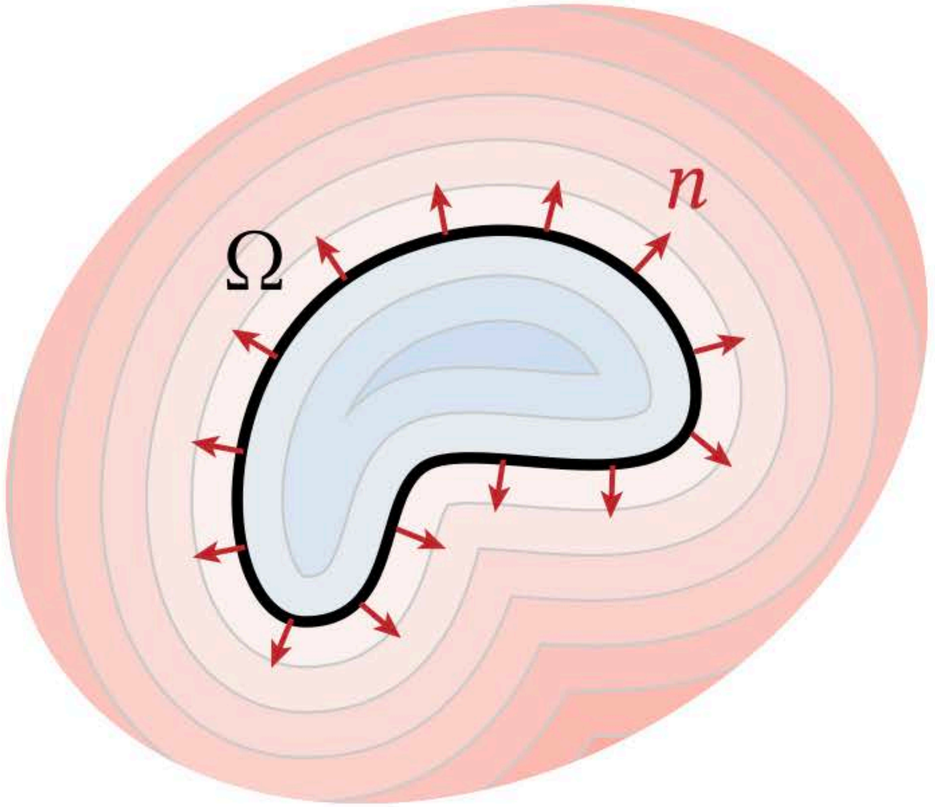
$$
\begin{aligned}
\|\nabla u(x)\|^2 &= 1, \quad x \notin \Omega \\
u(x) &= 0 \quad x \in \Omega \\
\frac{\partial u}{\partial n}(x) &= 1 \quad x \in \Omega
\end{aligned}
$$



# Signed distance

eikonal equation

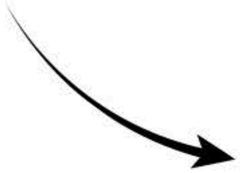
$$
\begin{aligned}
\|\nabla u(x)\|^2 &= 1, \quad x \notin \Omega \\
u(x) &= 0 \quad x \in \Omega \\
\frac{\partial u}{\partial n}(x) &= 1 \quad x \in \Omega
\end{aligned}
$$



# One small change to the eikonal equation...

eikonal equation

$$\begin{aligned} \|\nabla u(x)\|^2 &= 1, & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \frac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$



"viscous" signed eikonal equation:

$$\begin{aligned} \mathrm{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \frac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

# One small change to the eikonal equation...

eikonal equation

$$\begin{aligned} \|\nabla u(x)\|^2 &= 1, & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \frac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

"viscous" signed eikonal equation:

$$\begin{aligned} \mathrm{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 \quad \textit{viscosity} & x \in \Omega \\ \frac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

# … leads to winding numbers!

"viscous" signed eikonal equation:

$$\begin{aligned} \operatorname{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \tfrac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

# … leads to winding numbers!

*signed Hopf-Cole transformation*

$$w(x) = \text{sign}_w(x) \exp(-\lambda \text{sign}_w(x) u(x))$$

Lipman (2021), *Phase Transitions, Distance Functions, and Implicit Neural Representations*



"viscous" signed eikonal equation:

$$\begin{aligned} \text{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \tfrac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \tfrac{\partial w^+}{\partial n}(x) - \tfrac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

# … leads to winding numbers!

*signed Hopf-Cole transformation*

$$w(x) = \text{sign}_w(x) \exp(-\lambda \text{sign}_w(x) u(x))$$

Lipman (2021), *Phase Transitions, Distance Functions, and Implicit Neural Representations*



"viscous" signed eikonal equation:

$$\begin{aligned} \text{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \tfrac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \tfrac{\partial w^+}{\partial n}(x) - \tfrac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

# … leads to winding numbers!

*signed Hopf-Cole transformation*

$$w(x) = \text{sign}_w(x) \exp(-\lambda \text{sign}_w(x) u(x))$$

Lipman (2021), *Phase Transitions, Distance Functions, and Implicit Neural Representations*



"viscous" signed eikonal equation:

$$\begin{aligned} \text{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \tfrac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ \textit{jump conditions} \quad w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \tfrac{\partial w^+}{\partial n}(x) - \tfrac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

# … leads to winding numbers!

*signed Hopf-Cole transformation*

$$w(x) = \text{sign}_w(x)\exp(-\lambda \text{sign}_w(x) u(x))$$

Lipman (2021), *Phase Transitions, Distance Functions, and Implicit Neural Representations*

"viscous" signed eikonal equation:

$$\begin{aligned} \text{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \frac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

jump screened Laplace equation

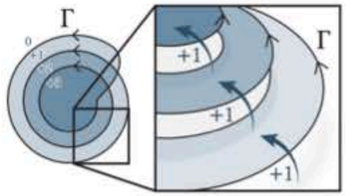
$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ \textit{jump conditions} \quad w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \frac{\partial w^+}{\partial n}(x) - \frac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

**Winding number is a function with "jumps"**

harmonic $\quad \Delta u = 0, \quad \text{on } M \setminus \Gamma,$

jump $\quad u^+ - u^- = 1, \quad \text{on } \Gamma,$

$\partial u^+/\partial n = \partial u^-/\partial n, \quad \text{on } \Gamma.$




The jump problem for the Laplace equation, Kratitzki (2003)

# … leads to winding numbers!

*signed Hopf-Cole transformation*

$$w(x) = \text{sign}_w(x) \exp(-\lambda \text{sign}_w(x) u(x))$$

Lipman (2021), *Phase Transitions, Distance Functions, and Implicit Neural Representations*

"viscous" signed eikonal equation:

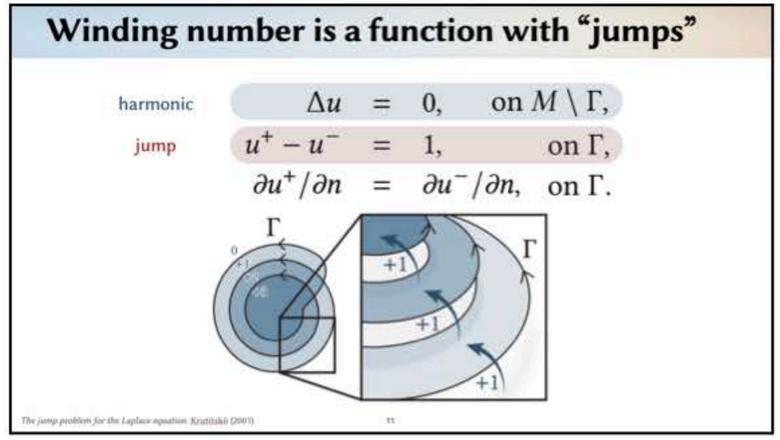
$$\begin{aligned} \text{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \frac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ \textit{jump conditions} \quad w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \frac{\partial w^+}{\partial n}(x) - \frac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

**unsigned:** Crandall & Lions (1983), Hopf (1950), Cole (1951)

Varadhan (1967), Kreisselmeier & Steinhauser (1980), Gurumoorthy & Rangajaran (2009), Karam et al. (2019), Abgrall (2022), Madan & Levin (2022)



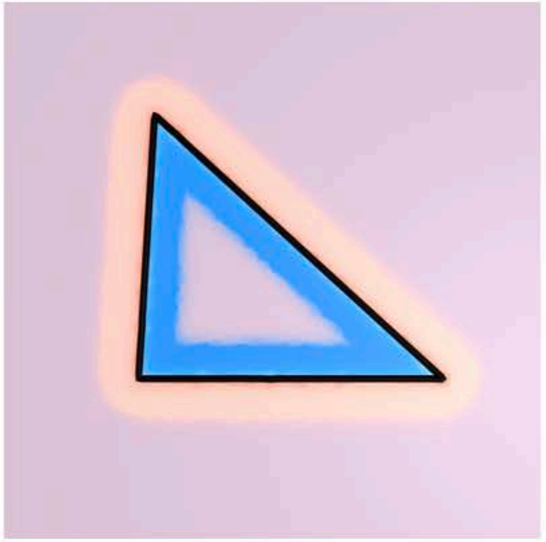
# Viscosity solutions of the *signed* eikonal equation

## Viscous signed eikonal equation:

$$\begin{aligned} \operatorname{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \tfrac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

## jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \tfrac{\partial w^+}{\partial n}(x) - \tfrac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$



# Viscosity solutions of the *signed* eikonal equation

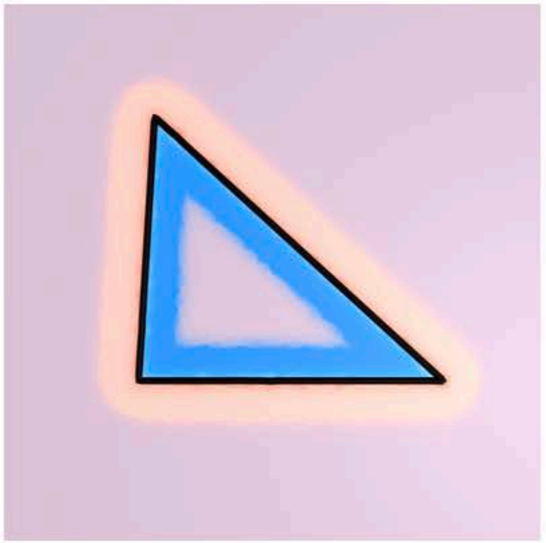
## Viscous signed eikonal equation:

$$\begin{aligned} \operatorname{sign}_\Omega(x)(\|\nabla u(x)\|^2 - 1) &= \tfrac{1}{\lambda}\Delta u(x), & x \notin \Omega \\ u(x) &= 0 & x \in \Omega \\ \tfrac{\partial u}{\partial n}(x) &= 1 & x \in \Omega \end{aligned}$$

## jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \tfrac{\partial w^+}{\partial n}(x) - \tfrac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$





inverse transformation

$$u(x) = -\lambda^{-1}\operatorname{sign}_w(x)\log|w(x)|$$

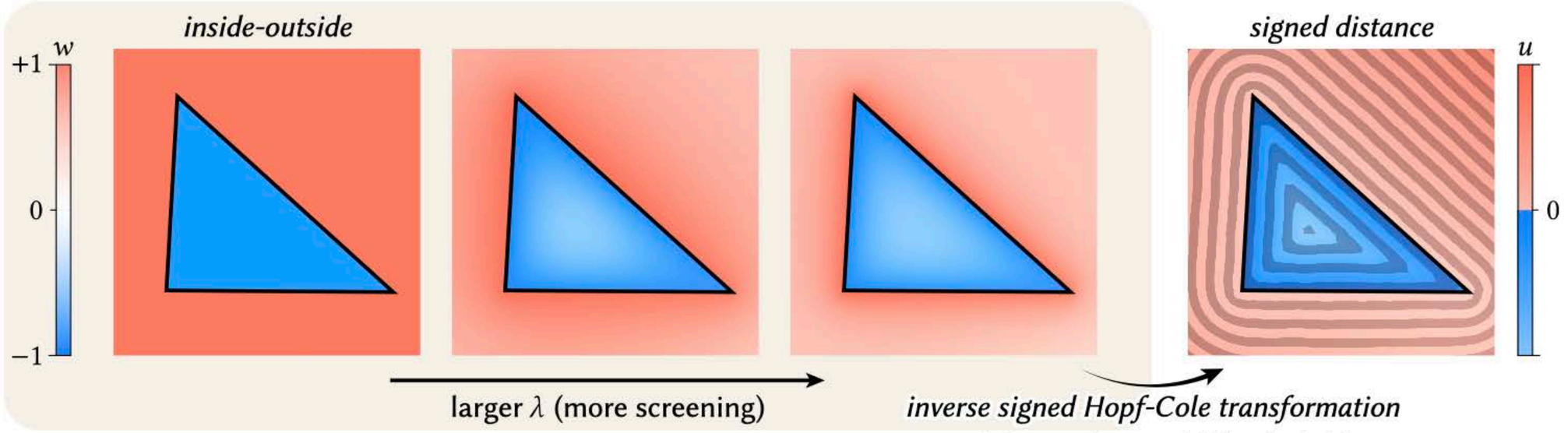
# Viscosity solutions of the *signed* eikonal equation

## jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \frac{\partial w^+}{\partial n}(x) - \frac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

## screened Poisson equation

$$\Delta w(x) - \lambda^2 w(x) = -2(\nabla \cdot n(x))\mu_\Omega(x)$$



# Viscosity solutions of the *signed* eikonal equation

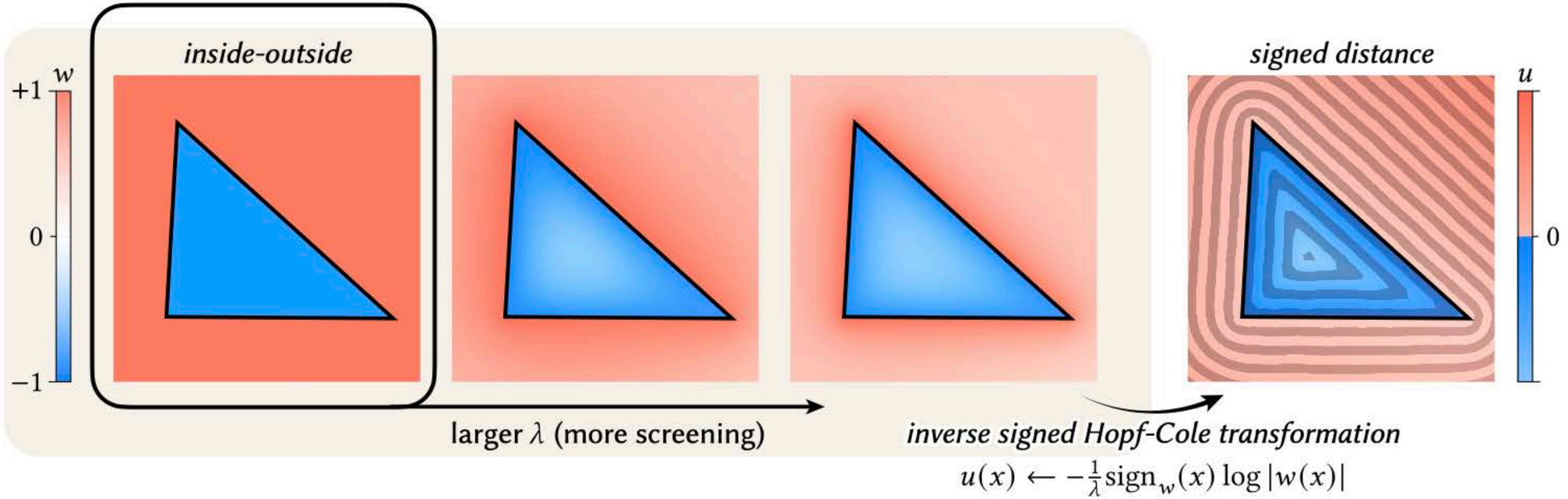
## jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \frac{\partial w^+}{\partial n}(x) - \frac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

## screened Poisson equation

$$\Delta w(x) - \lambda^2 w(x) = -2(\nabla \cdot n(x))\mu_\Omega(x)$$

$\lambda \to 0$: winding number (jump harmonic functions)



# Viscosity solutions of the *signed* eikonal equation

## jump screened Laplace equation

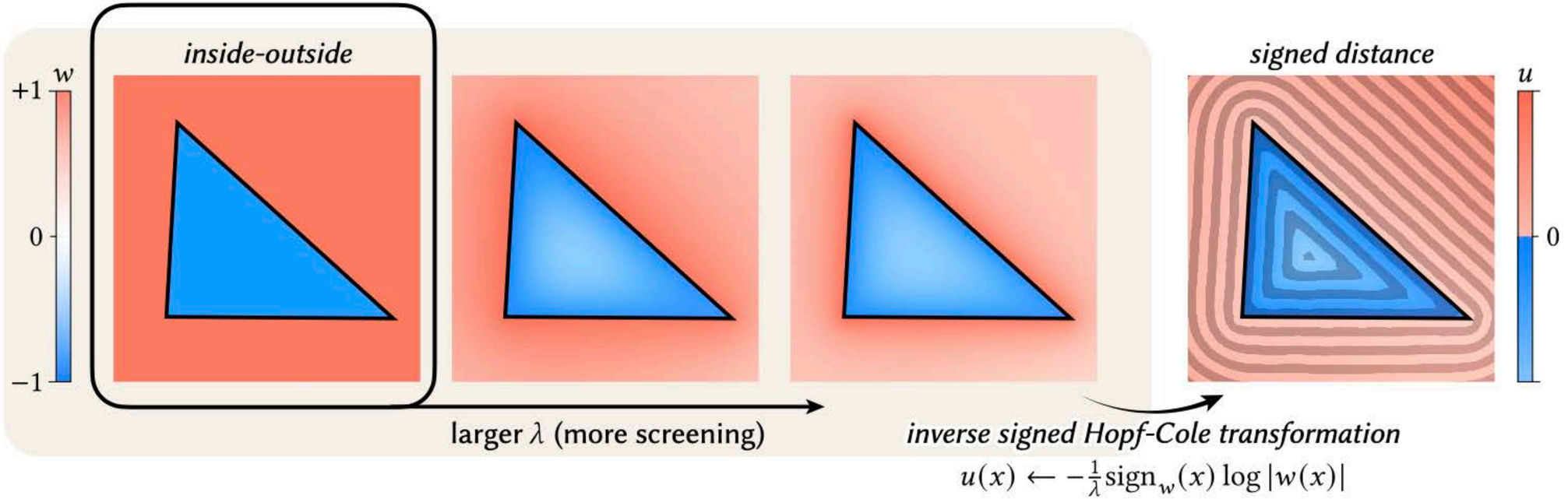
$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \frac{\partial w^+}{\partial n}(x) - \frac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

## screened Poisson equation

$$\Delta w(x) - \lambda^2 w(x) = -2(\nabla \cdot n(x))\mu_\Omega(x)$$

*Poisson Surface Reconstruction* [Kazhdan et al. 2006]

$\lambda \to 0$: winding number (jump harmonic functions)



# Viscosity solutions of the *signed* eikonal equation

## jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \frac{\partial w^+}{\partial n}(x) - \frac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

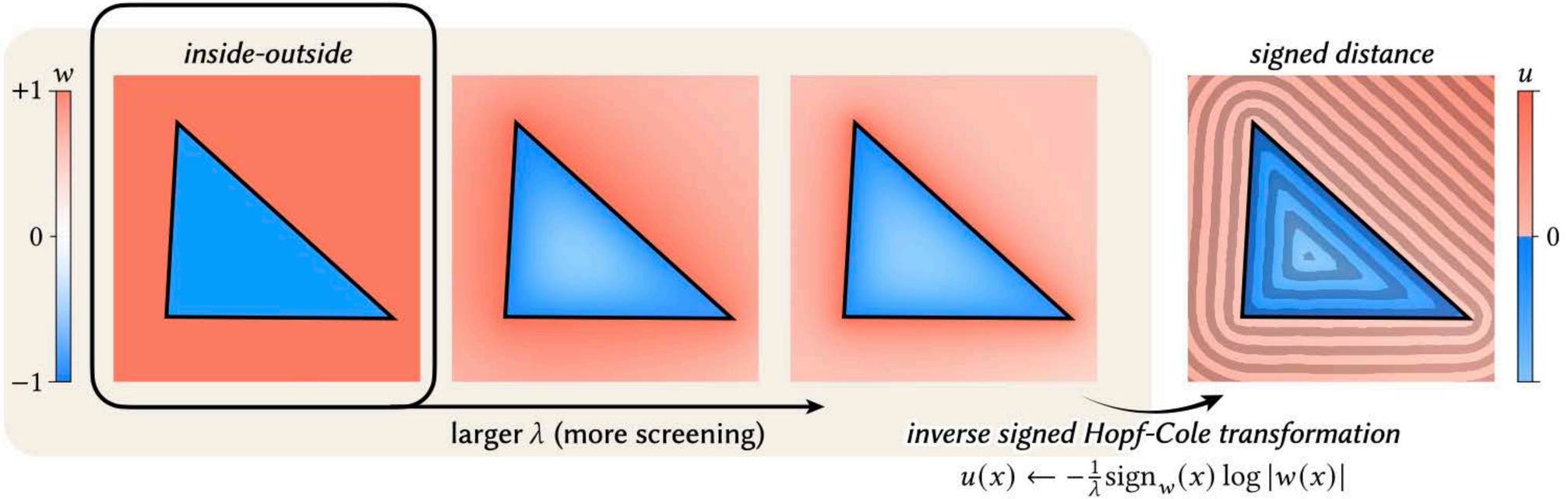
## screened Poisson equation

$$\Delta w(x) - \lambda^2 w(x) = -2(\nabla \cdot n(x))\mu_\Omega(x)$$

*Poisson Surface Reconstruction* [Kazhdan et al. 2006]

*regularized winding number* [Chen et al. 2024]

$\lambda \to 0$: winding number (jump harmonic functions)



# We can formulate a boundary integral

jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \frac{\partial w^+}{\partial n}(x) - \frac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$



boundary integral:

$$w(x) = 2 \int_{\Omega} \frac{\partial G^{\lambda}(x,z)}{\partial n(z)} \, \mathrm{d}z$$

# We can formulate a boundary integral

jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \frac{\partial w^{+}}{\partial n}(x) - \frac{\partial w^{-}}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

boundary integral:

$$w(x) = 2 \int_{\Omega} \frac{\partial G^{\lambda}(x, z)}{\partial n(z)} \, \mathrm{d}z$$

$$\frac{\partial G^{\lambda}(x, x')}{\partial n(x')} = \frac{(\lambda \|x - x'\| + 1)\langle x - x', n(x')\rangle}{4\pi \|x - x'\|^3} \exp(-\lambda \|x - x'\|)$$

# We can formulate a boundary integral

jump screened Laplace equation

$$\begin{aligned} \Delta w(x) - \lambda^2 w(x) &= 0, & x \notin \Omega \\ w^{\pm}(x) &= \pm 1 & x \in \Omega \\ \frac{\partial w^+}{\partial n}(x) - \frac{\partial w^-}{\partial n}(x) &= 0 & x \in \Omega \end{aligned}$$

boundary integral:

$$w(x) = 2 \int_\Omega \frac{\partial G^\lambda(x,z)}{\partial n(z)} \, \mathrm{d}z$$

$$\frac{\partial G^\lambda(x,x')}{\partial n(x')} = \frac{(\lambda \|x - x'\| + 1)\langle x - x', n(x')\rangle}{4\pi \|x - x'\|^3} \exp(-\lambda \|x - x'\|)$$

*exponential decay*

# Asymptotic analysis

$$\int_{\Omega} h(z) \exp(-\lambda \varphi(z)) \, \mathrm{d}z$$

# Asymptotic analysis

$$\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z \quad \underset{\lambda\to\infty}{\sim} \quad (2\pi/\lambda)^{d/2}\det\left(\nabla^2\varphi(x^*)\right)^{-1/2} h(x^*)\exp\left(-\lambda\varphi(x^*)\right)$$

*Laplace's method says…* *… only the contribution at* $x^* = \underset{z\in\Omega}{\operatorname{argmin}}\,\varphi(z)$ *matters.*

# Asymptotic analysis

$$\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z \quad \underset{\lambda\to\infty}{\sim} \quad (2\pi/\lambda)^{d/2}\det\left(\nabla^2\varphi(x^*)\right)^{-1/2} h(x^*)\exp\left(-\lambda\varphi(x^*)\right)$$

*Laplace's method says...* *... only the contribution at* $x^* = \underset{z\in\Omega}{\operatorname{argmin}}\,\varphi(z)$ *matters.*

*minimizer of* $\varphi$

# Asymptotic analysis

$$\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z \quad \underset{\lambda\to\infty}{\sim} \quad (2\pi/\lambda)^{d/2}\det\left(\nabla^2\varphi(x^*)\right)^{-1/2} h(x^*)\exp\left(-\lambda\varphi(x^*)\right)$$

*Laplace's method says...* *... only the contribution at* $x^* = \underset{z\in\Omega}{\operatorname{argmin}}\,\varphi(z)$ *matters.*

*minimizer of φ*

(All other contributions become exponentially small w.r.t. the peak contribution.)

# Asymptotic analysis

$$\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z \quad \underset{\lambda\to\infty}{\sim} \quad (2\pi/\lambda)^{d/2}\det\left(\nabla^2\varphi(x^*)\right)^{-1/2} h(x^*)\exp\left(-\lambda\varphi(x^*)\right)$$

*Laplace's method says...* *... only the contribution at* $x^* = \underset{z\in\Omega}{\operatorname{argmin}}\,\varphi(z)$ *matters.*

(All other contributions become exponentially small w.r.t. the peak contribution.)

Applying $-\frac{1}{\lambda}\log(\cdot)$ to both sides,

# Asymptotic analysis

$$\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z \underset{\lambda\to\infty}{\sim} (2\pi/\lambda)^{d/2}\det\left(\nabla^2\varphi(x^*)\right)^{-1/2} h(x^*)\exp\left(-\lambda\varphi(x^*)\right)$$

*Laplace's method says...* *... only the contribution at* $x^* = \underset{z\in\Omega}{\operatorname{argmin}}\,\varphi(z)$ *matters.*

(All other contributions become exponentially small w.r.t. the peak contribution.)

Applying $-\frac{1}{\lambda}\log(\cdot)$ to both sides,

$$-\frac{1}{\lambda}\log\left[\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z\right] \sim \varphi(x^*) + O(\lambda^{-1}) \quad \text{as } \lambda\to\infty$$

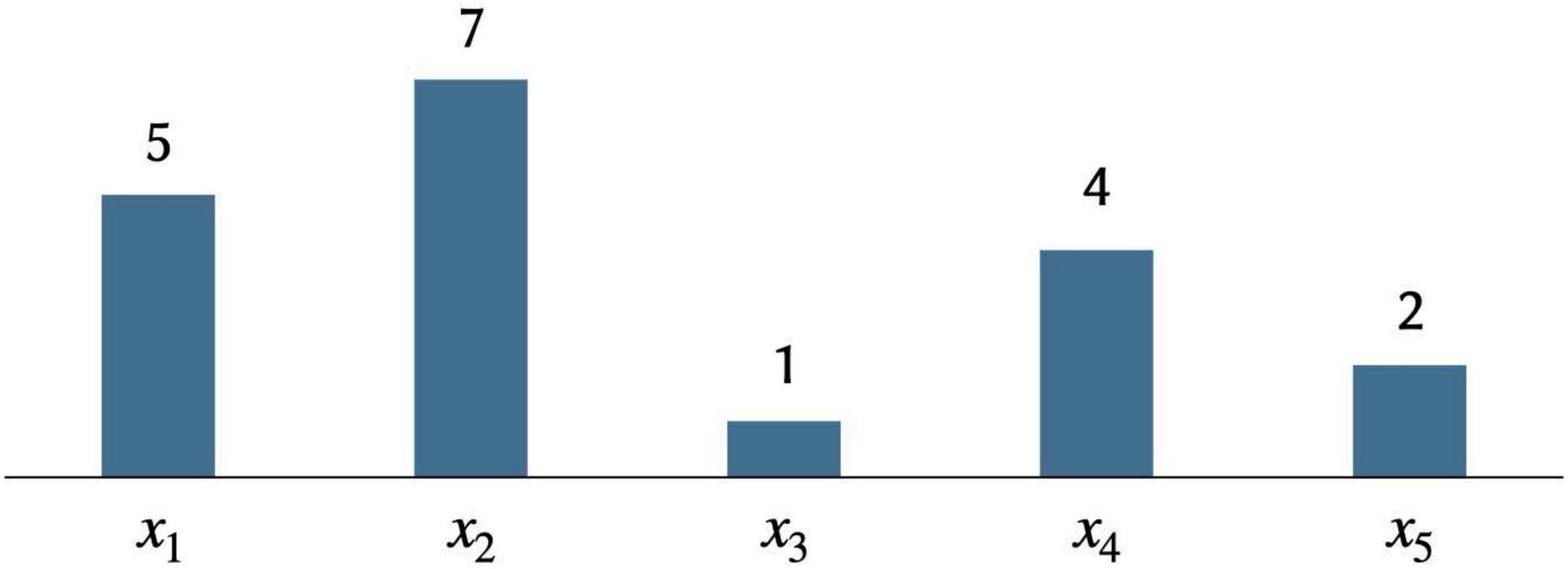
# Intuition in the discrete setting

# Intuition in the discrete setting

Input: discrete vector of values $(x_1, \ldots, x_n)$

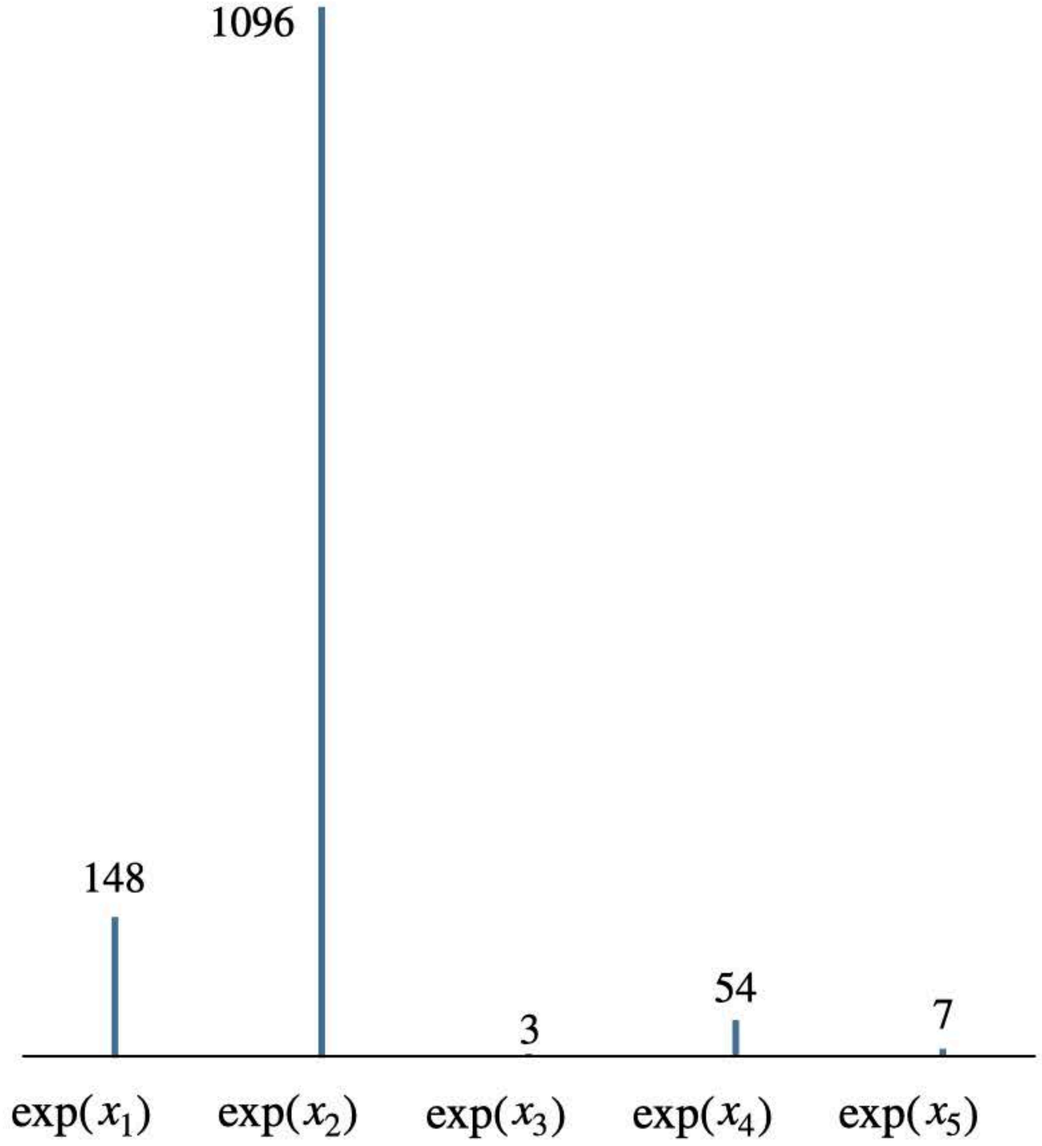
# Intuition in the discrete setting

Input: discrete vector of values $(x_1, \ldots, x_n)$



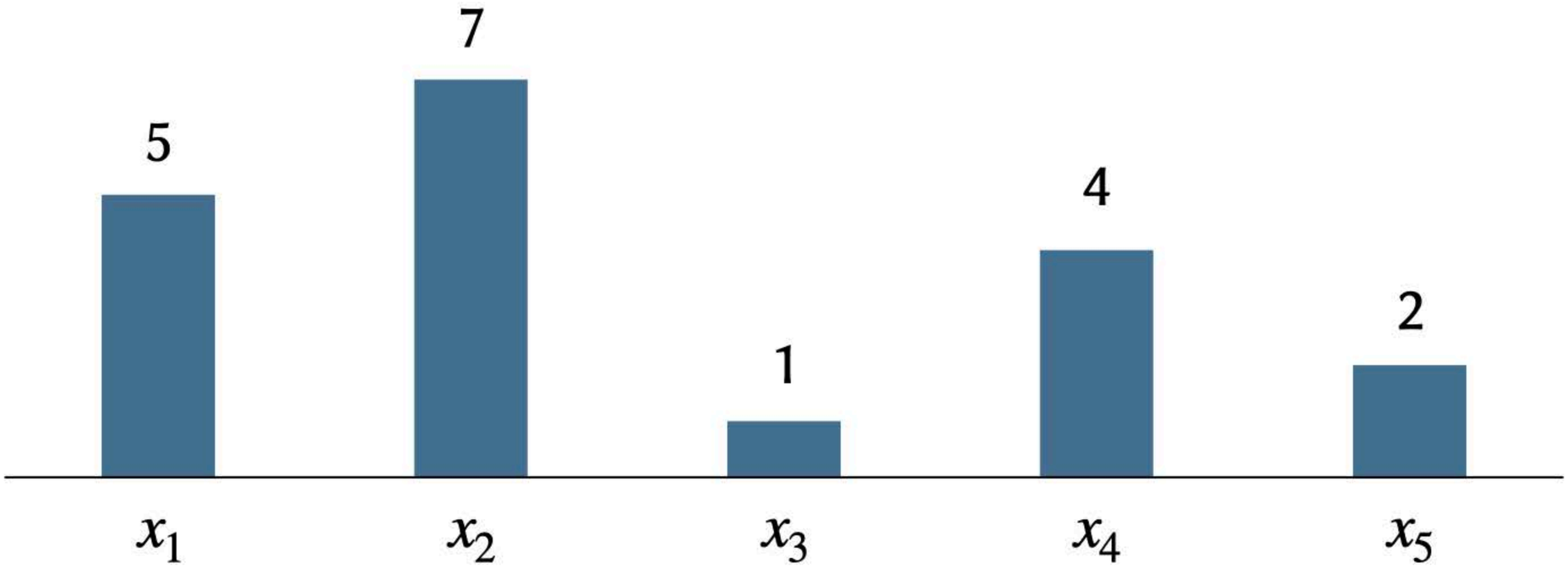
# Intuition in the discrete setting

$x_i \mapsto \exp(x_i)$



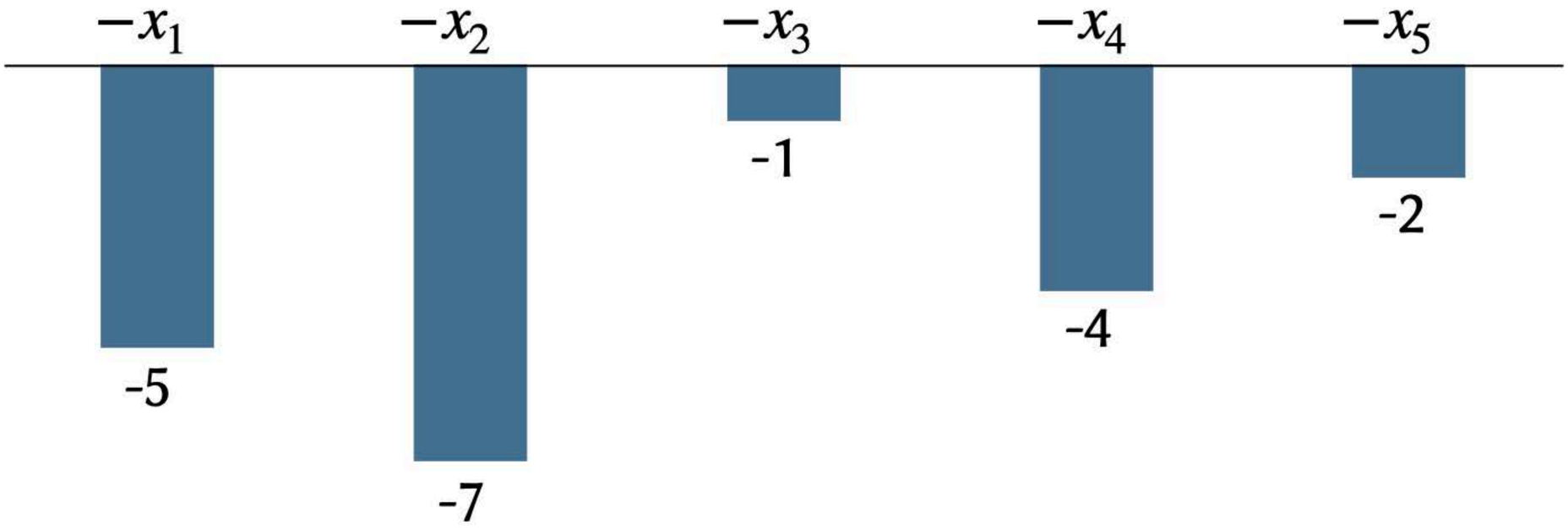
# Intuition in the discrete setting

Input: discrete vector of values $(x_1, \ldots, x_n)$



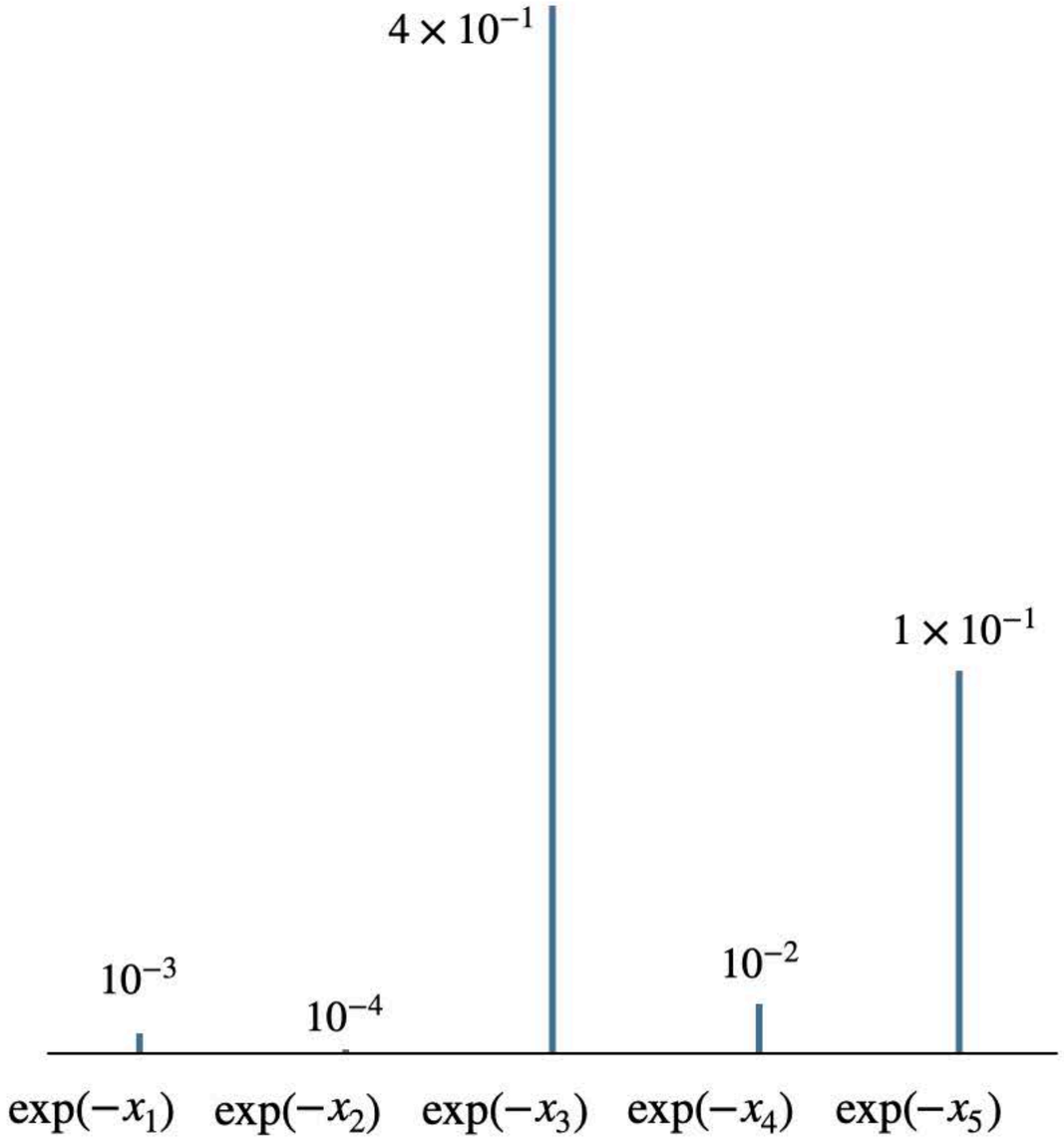
# Intuition in the discrete setting

Input: discrete vector of values $(x_1, \ldots, x_n)$



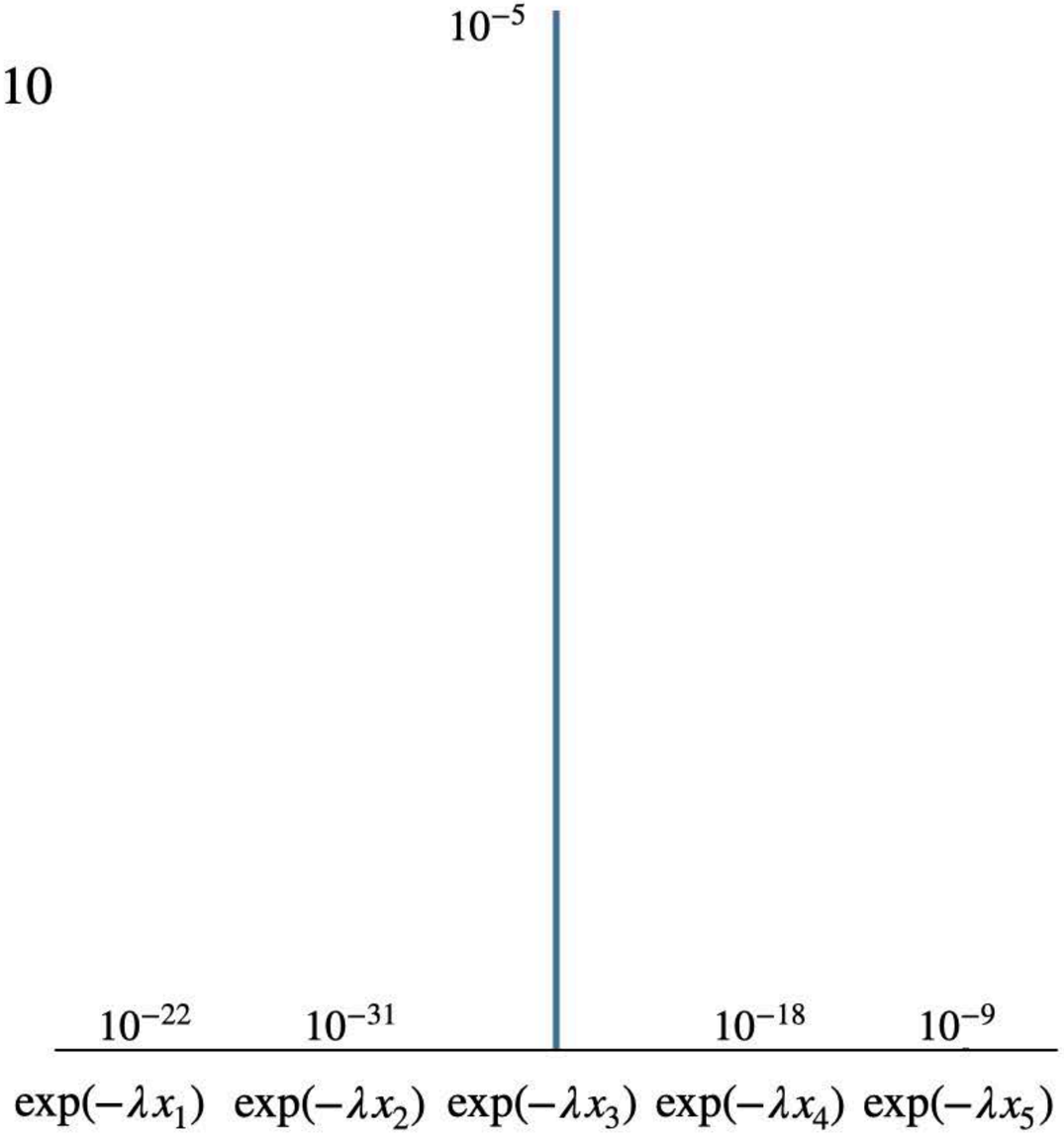
# Intuition in the discrete setting

$x_i \mapsto \exp(-x_i)$

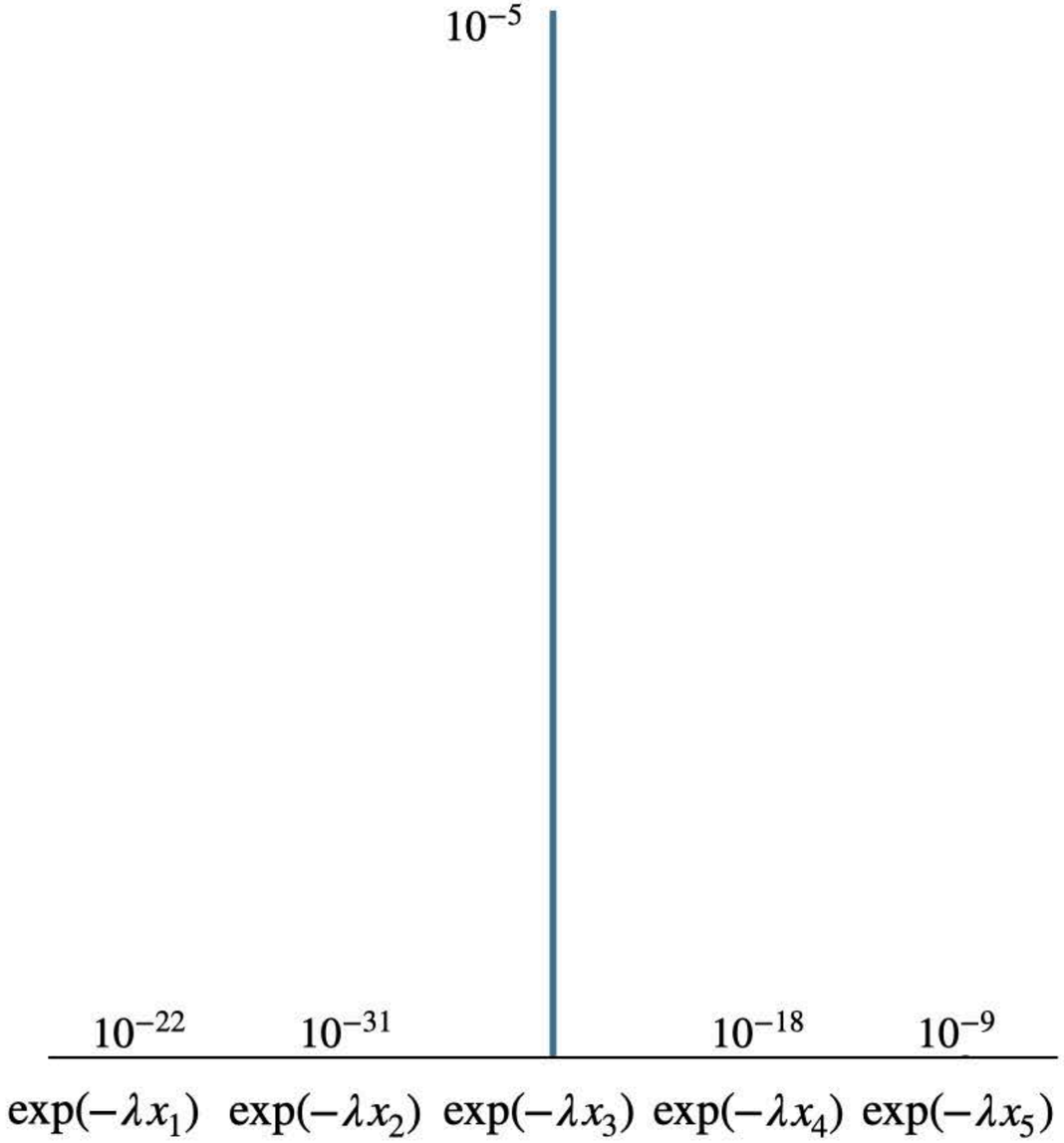


# Intuition in the discrete setting

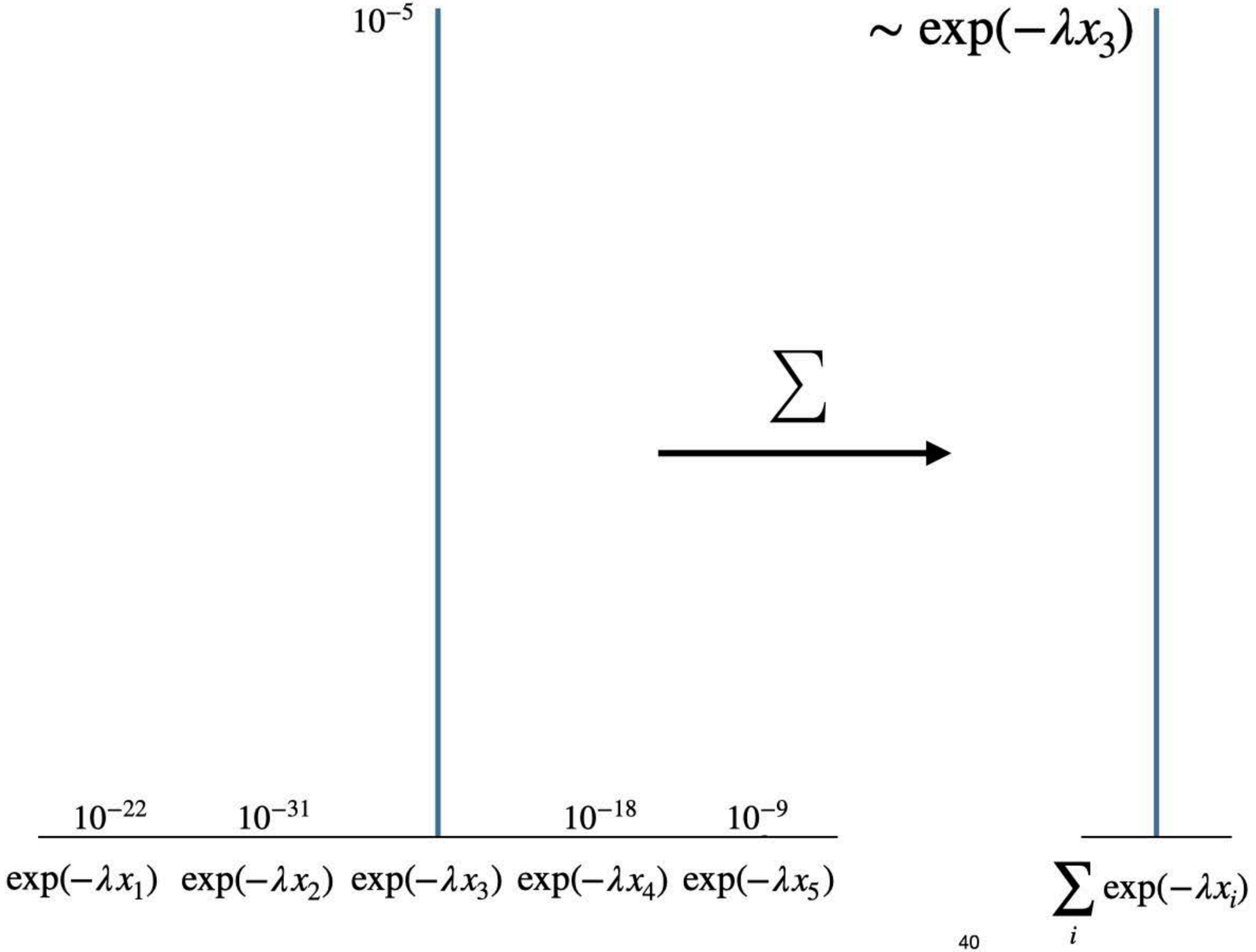
$x_i \mapsto \exp(-\lambda x_i), \ \lambda = 10$



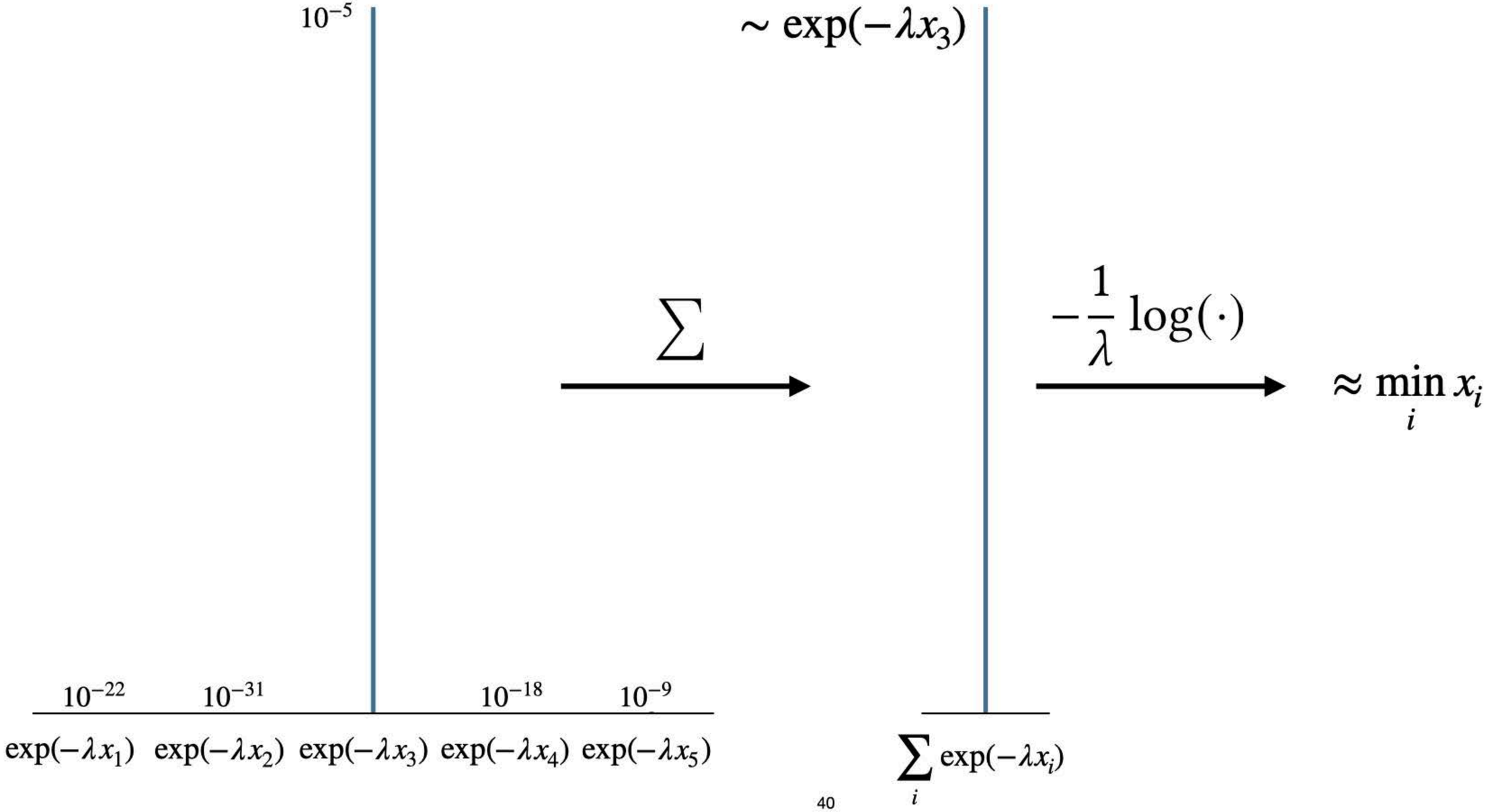
# Intuition in the discrete setting



# Intuition in the discrete setting



# Intuition in the discrete setting



# Deriving convolutional distance

*Back to our analysis…*

$$-\frac{1}{\lambda}\log\left[\int_{\Omega} h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z\right] \sim \varphi(x^*) + O(\lambda^{-1}) \qquad \text{as } \lambda \to \infty$$

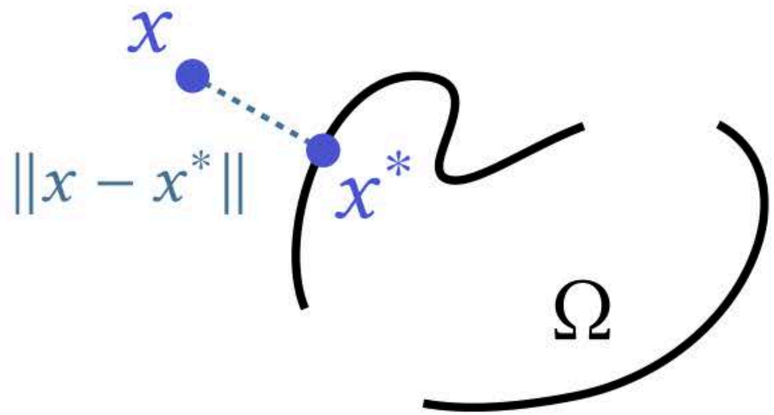
$$x^* = \operatorname*{argmin}_{z\in\Omega} \varphi(z)$$

# Deriving convolutional distance

$$-\frac{1}{\lambda}\log\left[\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z\right] \sim \varphi(x^*) + O(\lambda^{-1}) \quad \text{as} \quad \lambda \to \infty$$

$$x^* = \operatorname*{argmin}_{z\in\Omega} \varphi(z)$$

Use $\varphi_x(z) = \|x - z\|$ to get $\|x - x^*\| = \min_{z\in\Omega} \|x - z\|$ as $\lambda \to \infty$ .

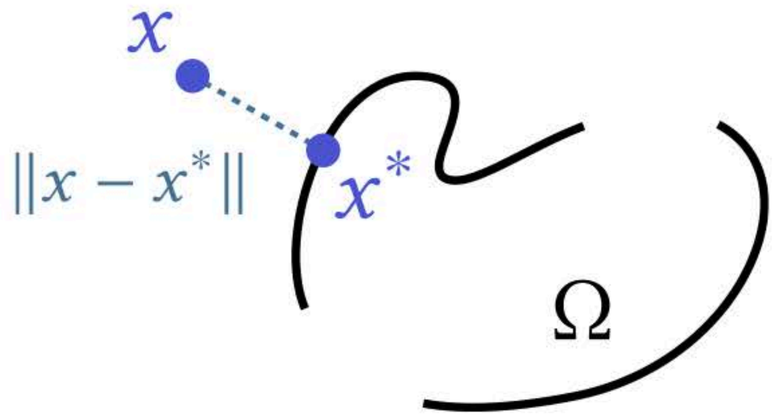


# Deriving convolutional distance

$$-\frac{1}{\lambda}\log\left[\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z\right] \sim \varphi(x^*) + O(\lambda^{-1}) \quad \text{as} \quad \lambda \to \infty$$

$$x^* = \operatorname*{argmin}_{z\in\Omega} \varphi(z)$$

Use $\varphi_x(z) = \|x - z\|$ to get $\|x - x^*\| = \min_{z\in\Omega} \|x - z\|$ as $\lambda \to \infty$ .



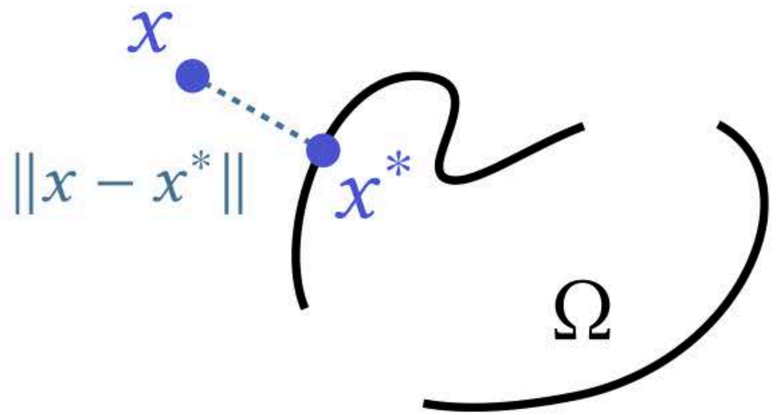
# Deriving convolutional distance

$$-\frac{1}{\lambda}\log\left[\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z\right] \sim \varphi(x^*) + O(\lambda^{-1}) \quad \text{as} \quad \lambda \to \infty$$

$$x^* = \operatorname*{argmin}_{z\in\Omega} \varphi(z)$$

Use $\varphi_x(z) = \|x - z\|$ to get $\|x - x^*\| = \min_{z\in\Omega} \|x - z\|$ as $\lambda \to \infty$ .

*distance from x to closest point in* $\Omega$

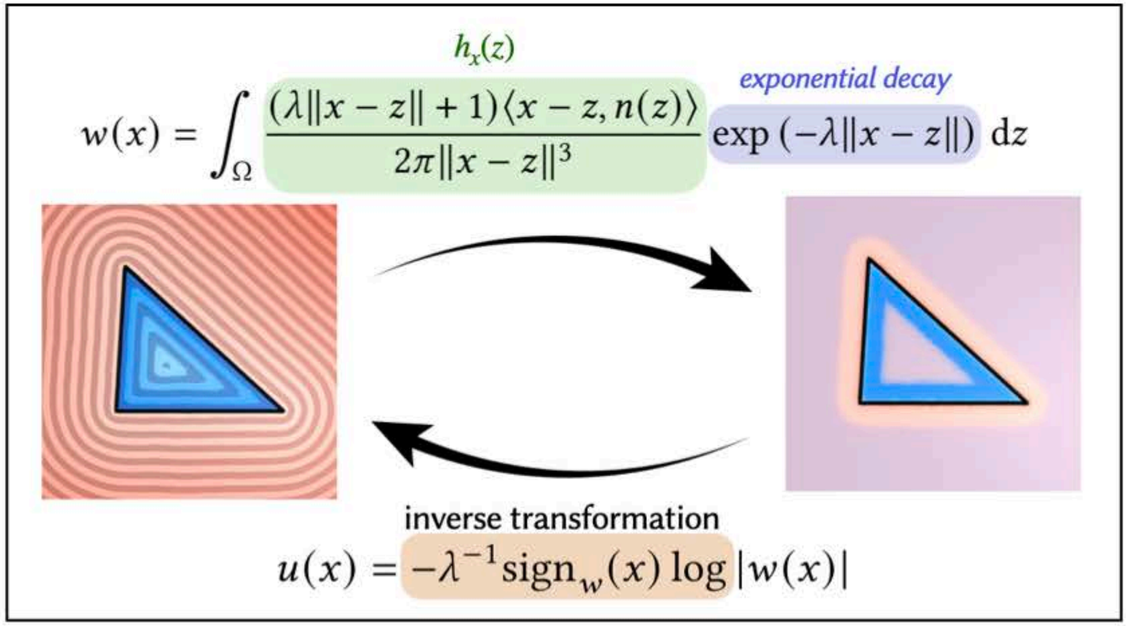


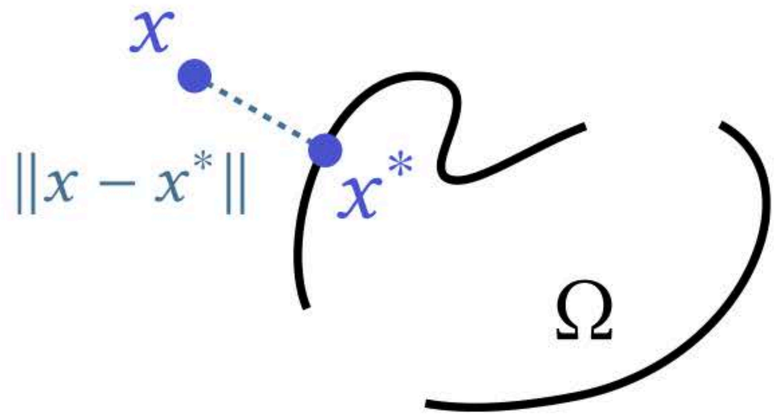
# Deriving convolutional distance

$$-\frac{1}{\lambda}\log\left[\int_\Omega h(z)\exp(-\lambda\varphi(z))\,\mathrm{d}z\right] \sim \varphi(x^*) + O(\lambda^{-1}) \quad \text{as} \quad \lambda \to \infty$$

$$x^* = \operatorname*{argmin}_{z\in\Omega} \varphi(z)$$

Use $\varphi_x(z) = \|x - z\|$ to get $\|x - x^*\| = \min_{z\in\Omega} \|x - z\|$ as $\lambda \to \infty$ .



# A general convolutional formula for distance

General convolutional distance formula:

$$\widetilde{d}(x) := -\frac{1}{\lambda} \log \left[ \int_{\Omega} h_x(z) \exp\left( -\lambda \varphi_x(z) \right) \mathrm{d}z \right]$$

where $\varphi_x(z) = \|x - z\|$.

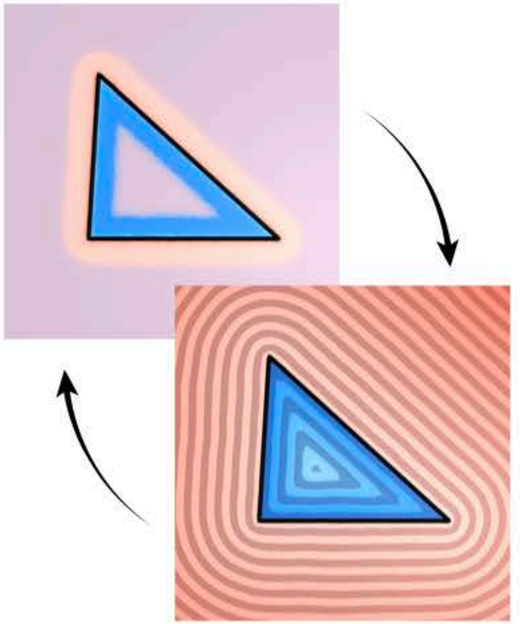
Examples:

- *Varadhan's formula* [Varadhan 1967]
- *LogSumExp* distance [Madan & Levin 2022]
- *Kreisselmeier-Steinhauser* functions [Kreisselmeier & Steinhauser 1980]
- *Schrödinger distance transform* [Gurumoorthy and Rangarajan 2009]
- Other instances [Karam et al. 2019, Abgrall 2022]
- Discrete LogSumExp

# A summary of our insights so far

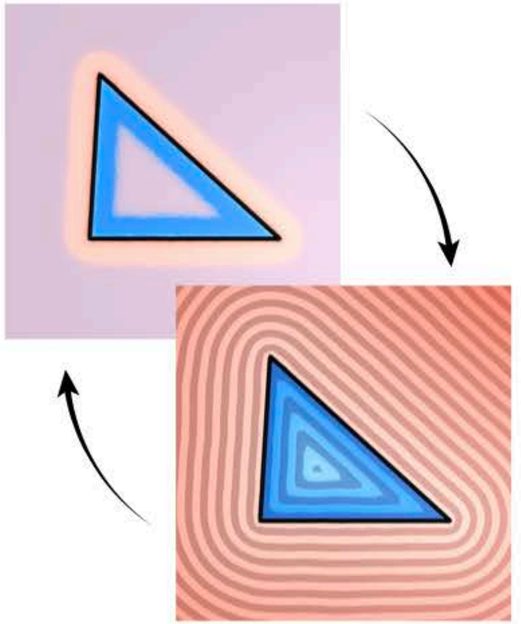
# A summary of our insights so far

- Signed distance and inside/outside are related through a simple change of variables



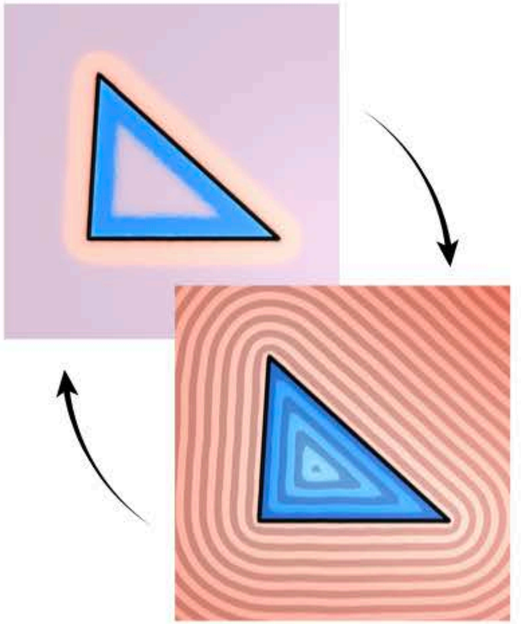
# A summary of our insights so far

- Signed distance and inside/outside are related through a simple change of variables
- We can generalize this change of variables to more general exponential transforms



# A summary of our insights so far

- Signed distance and inside/outside are related through a simple change of variables
- We can generalize this change of variables to more general exponential transforms
- We derive a general *convolutional distance formula* that unifies past methods for reconstruction and distance



# Convolutional formulas break down on real data

boundary integral:

$$w(x) = 2\int_{\Omega} \frac{\partial G^{\lambda}(x,z)}{\partial n(z)}\,\mathrm{d}z \longrightarrow 2\sum_{p\in P} \frac{\partial G^{\lambda}(x,p)}{\partial n_p}$$

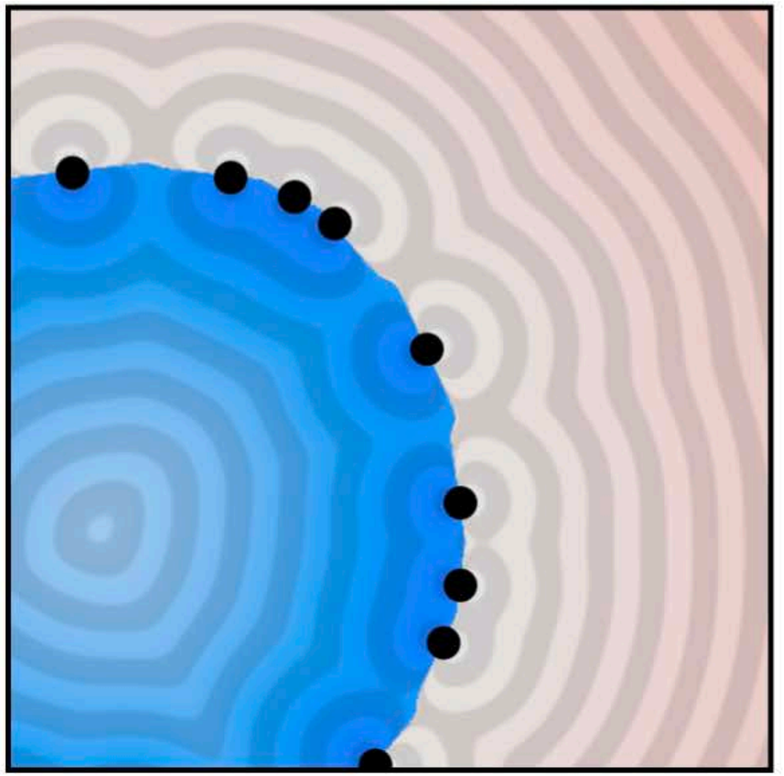
We overfit distance to the *discretization* of **Ω!**



*input point cloud P*

*signed distance approximation*

$$u(x) = -\lambda^{-1}\mathrm{sign}\big(w(x)\big)\log|w(x)|$$



# Convolutional formulas break down on real data

boundary integral:

$$w(x) = 2\int_{\Omega} \frac{\partial G^{\lambda}(x,z)}{\partial n(z)}\,\mathrm{d}z \longrightarrow 2\sum_{p\in P} \frac{\partial G^{\lambda}(x,p)}{\partial n_p}$$



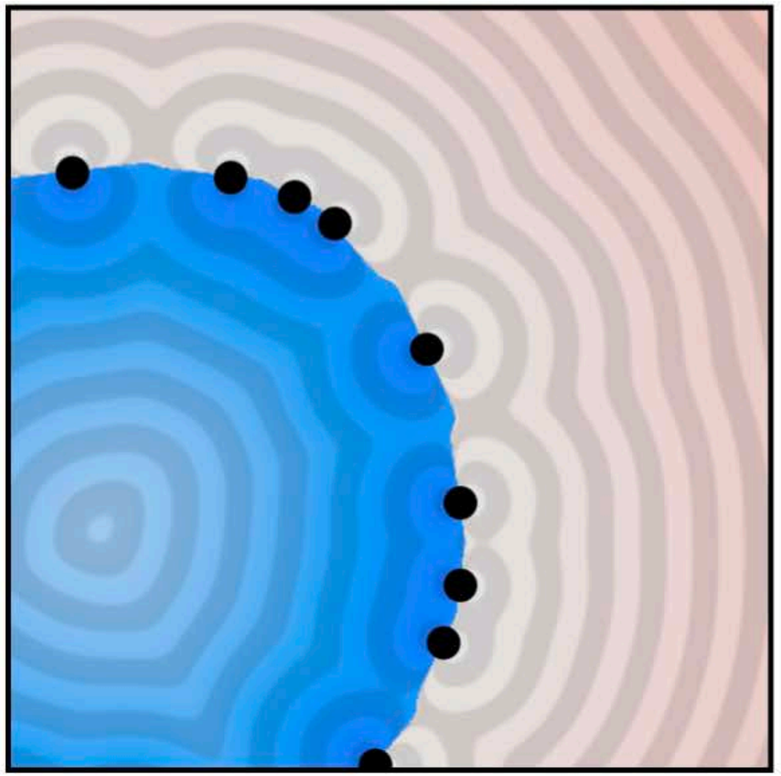
*input point cloud P*

We overfit distance to the *discretization* of **Ω**!

We want distance to the *surface*!

*signed distance approximation*

$$u(x) = -\lambda^{-1}\mathrm{sign}\big(w(x)\big)\log|w(x)|$$



# The exponential is (too) strong

General convolutional distance formula:

$$\widetilde{d}(x) := -\frac{1}{\lambda} \log \left[ \int_{\Omega} h_x(z) \exp\left( -\lambda \varphi_x(z) \right) \mathrm{d}z \right]$$

where $\varphi_x(z) = \|x - z\|$.

# The exponential is (too) strong

General convolutional distance formula:

$$\widetilde{d}(x) := -\frac{1}{\lambda} \log \left[ \int_{\Omega} h_x(z) \exp\left( -\lambda \varphi_x(z) \right) \mathrm{d}z \right]$$

where $\varphi_x(z) = \|x - z\|$.

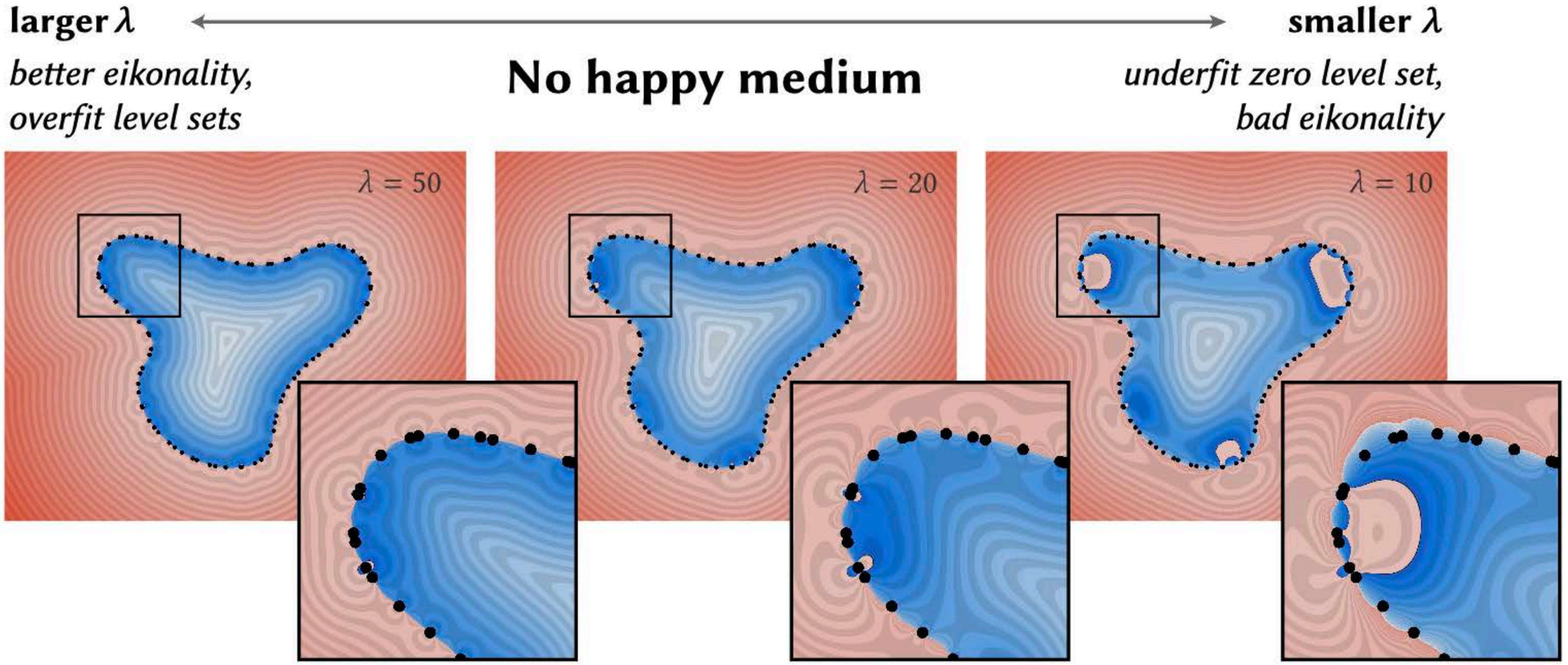
$$-\frac{1}{\lambda} \log \left[ \int_{\Omega} h(z) \exp(-\lambda \varphi(z)) \, \mathrm{d}z \right] \sim \varphi(x^*) + O(\lambda^{-1}) \quad \text{as} \quad \lambda \to \infty$$

*minimizer of $\varphi$*

$$x^* = \operatorname*{argmin}_{z \in \Omega} \varphi(z)$$

# The exponential is (too) strong

General convolutional distance formula:



# The exponential is (too) strong

General convolutional distance formula:

**larger $\lambda$**

*better eikond*

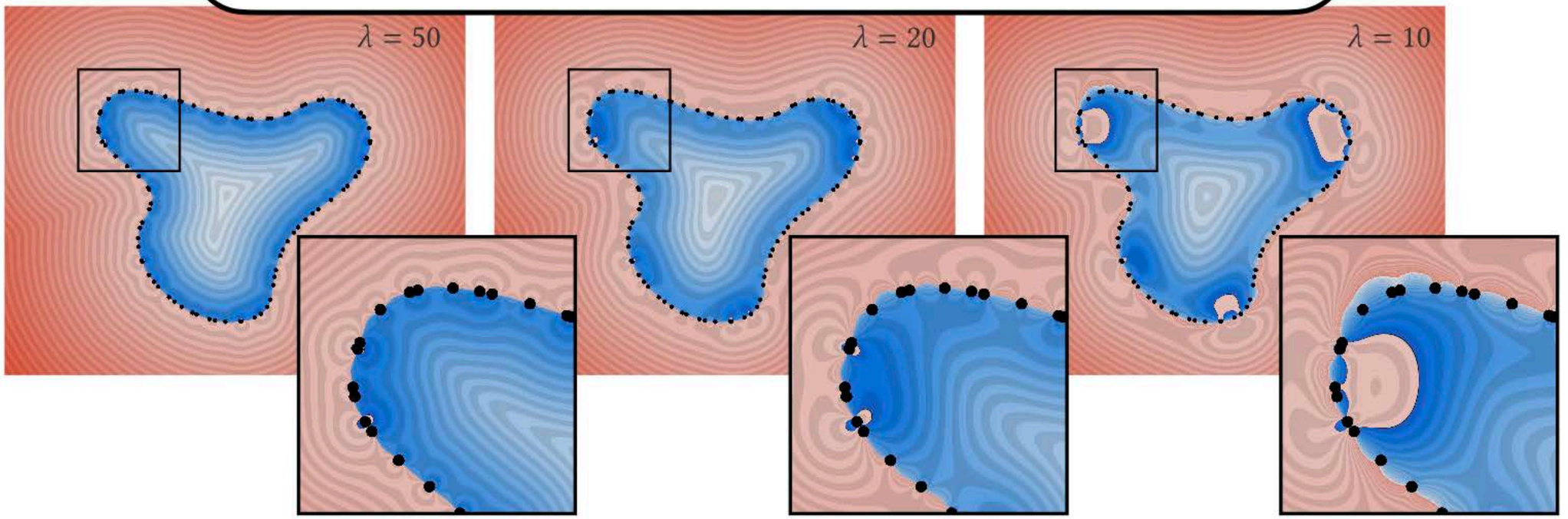
*overfit level s*

**These asymptotics are based on <u>scalar</u> diffusion.**

**aller $\lambda$**

*vel set,*

*onality*

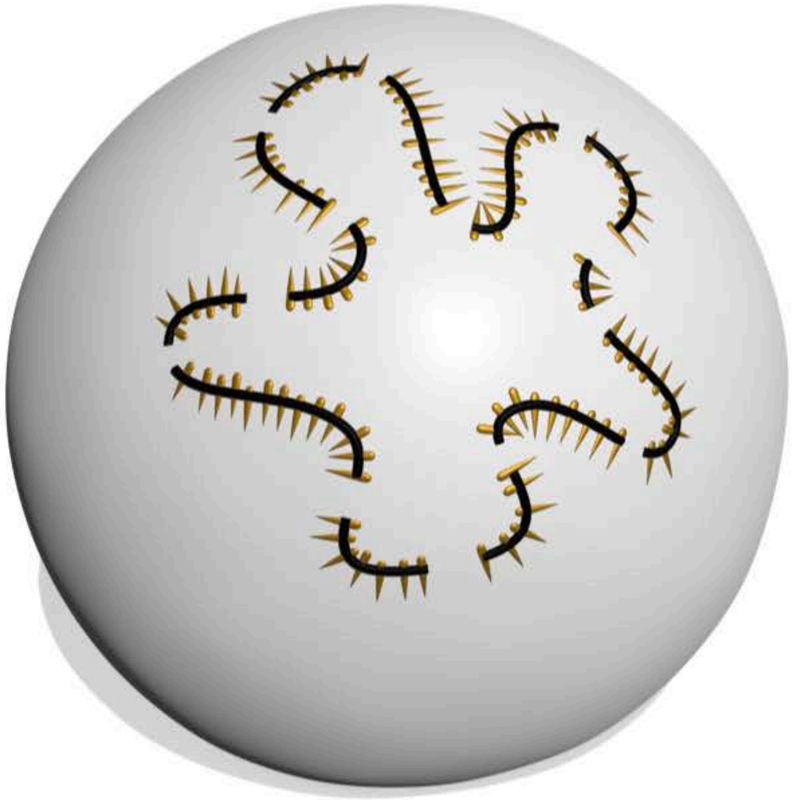


# GENERALIZED SIGNED DISTANCE

# Basic idea

# Basic idea

input

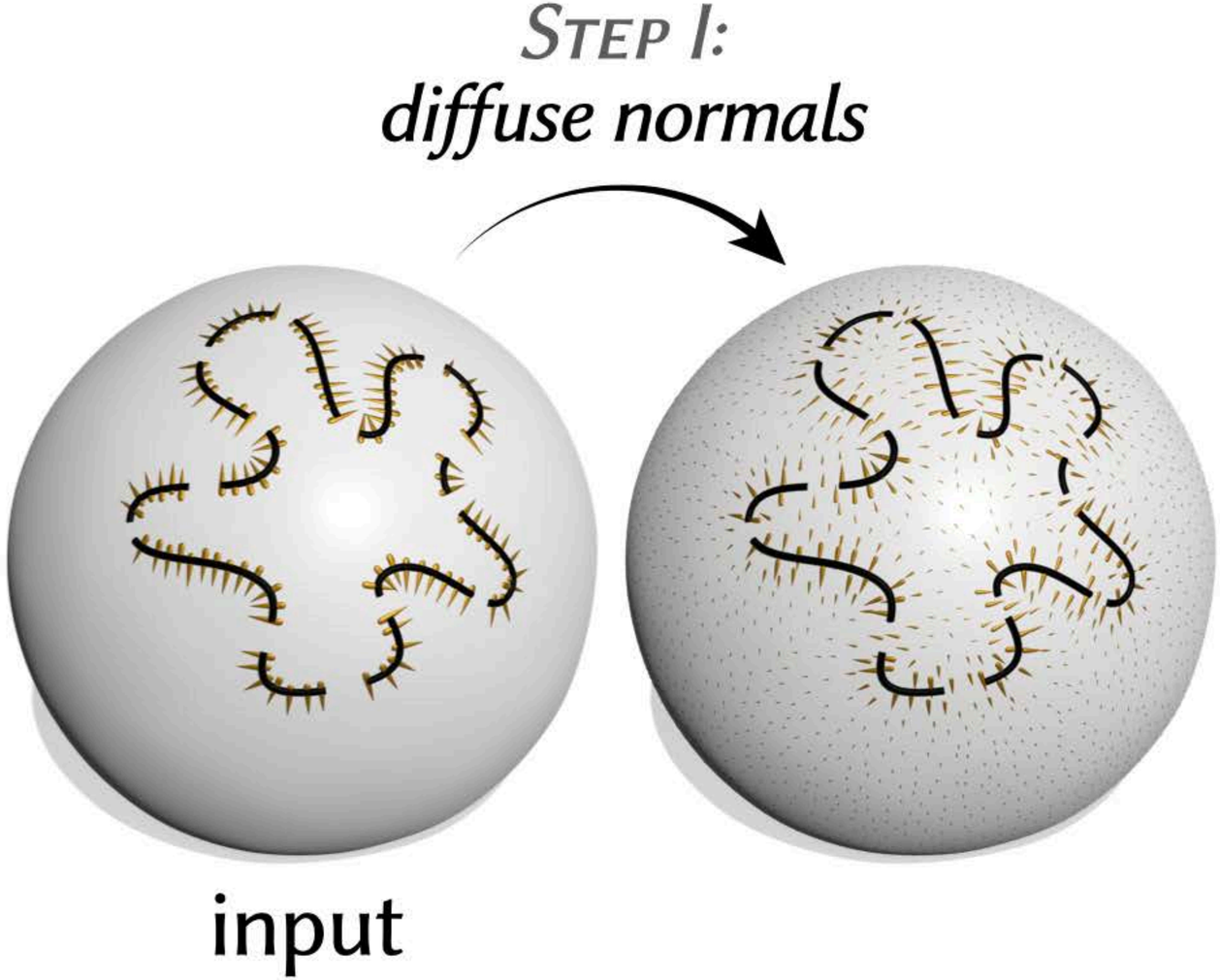


# Basic idea

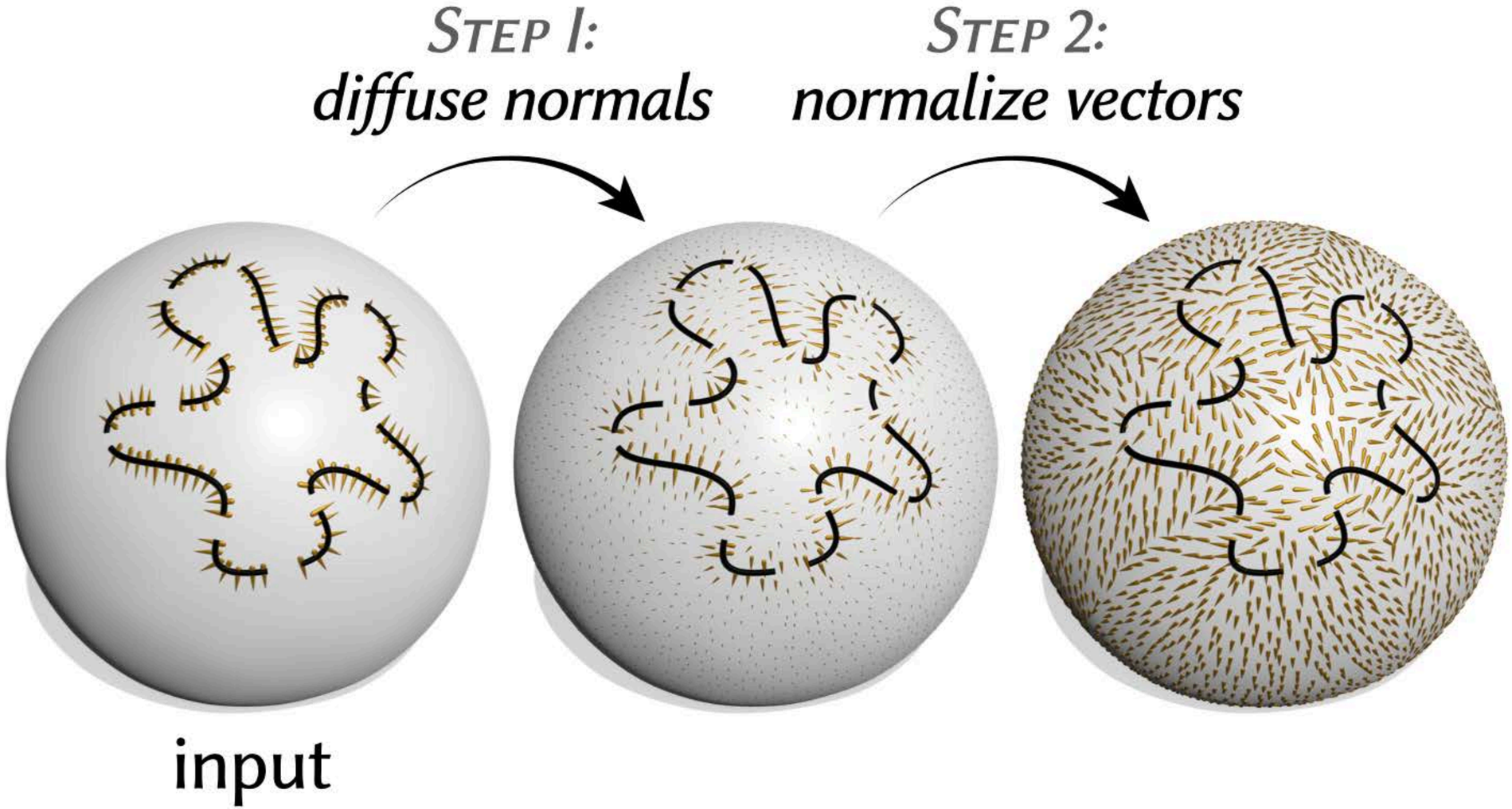


input

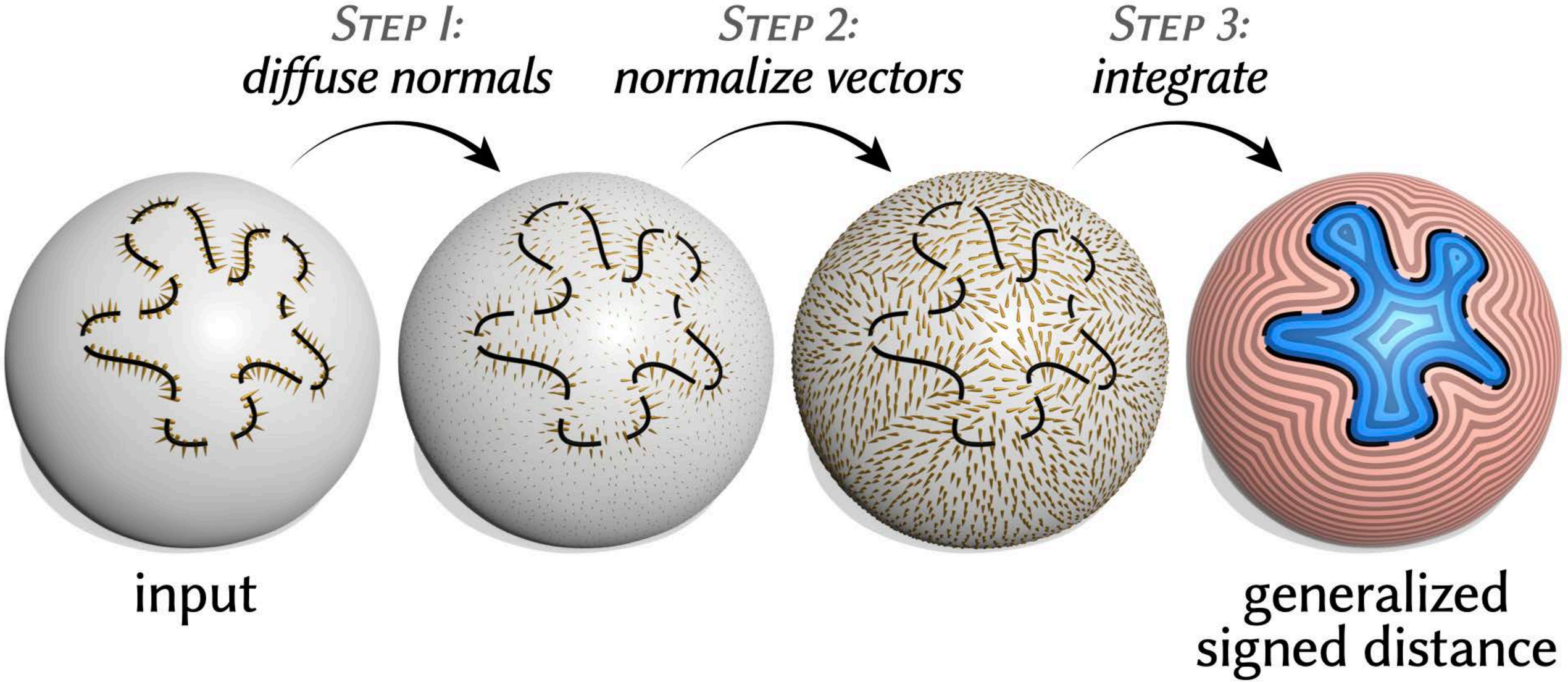
# Basic idea



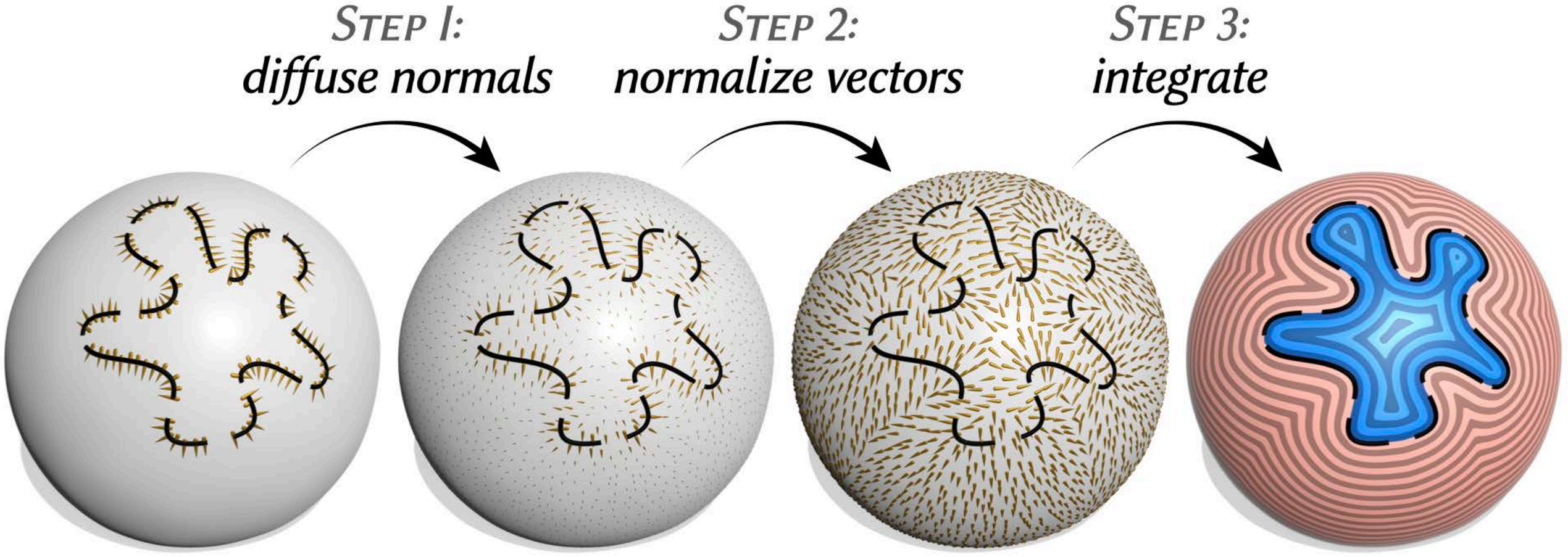
# Basic idea



# Basic idea


STEP 1:
diffuse normals
STEP 2:
normalize vectors
STEP 3:
integrate
input
generalized
signed distance

# Basic idea

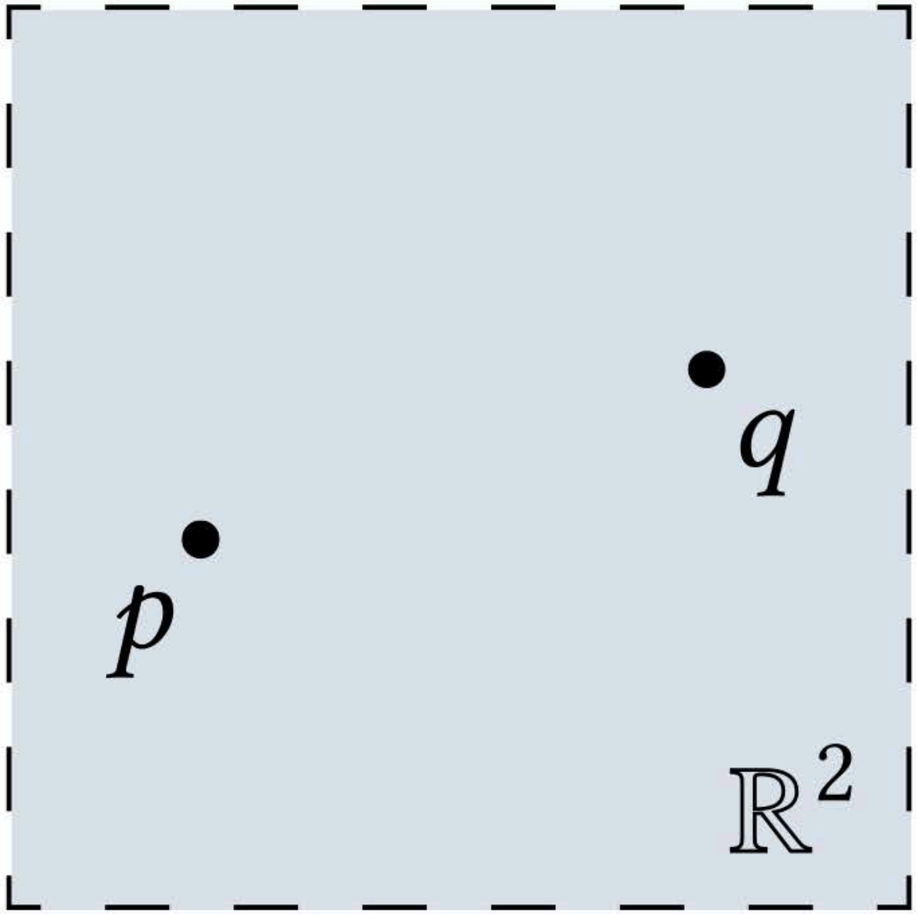

Step 1:
diffuse normals
Step 2:
normalize vectors
Step 3:
integrate
input
signed heat method (SHM)
generalized
signed distance

# Parallel transport can be computed using diffusion

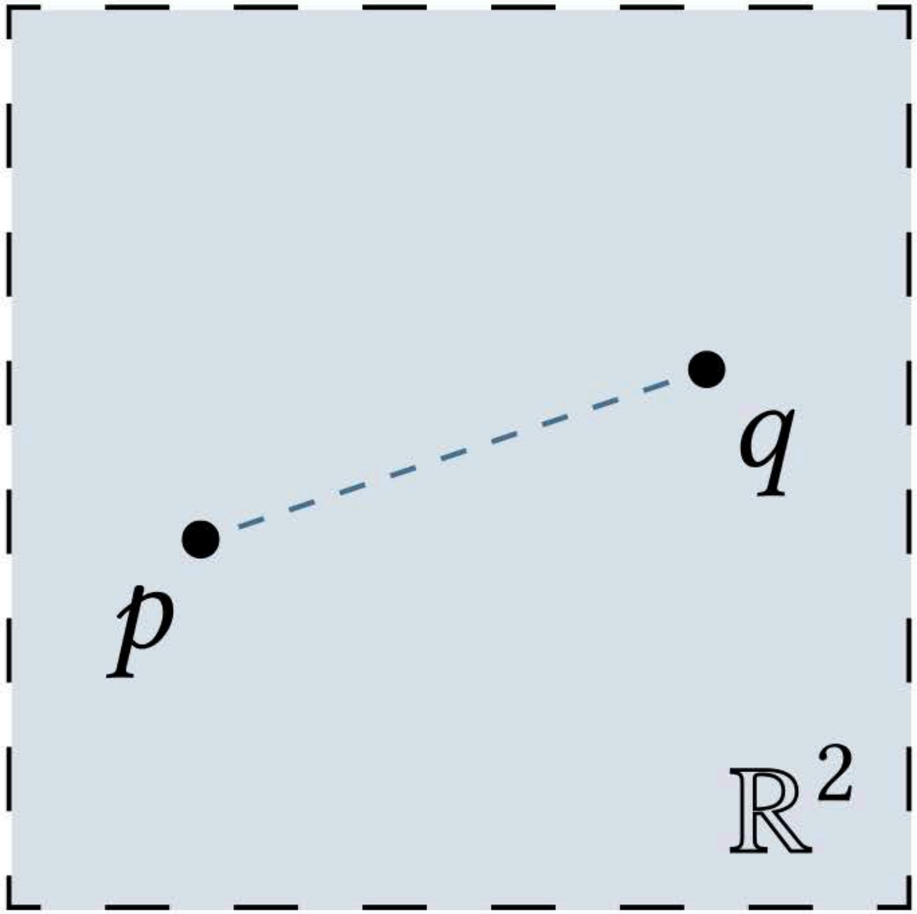
**Theorem.** Vector heat diffusion yields parallel transport along shortest geodesics, as diffusion time $\to 0$.

N. Berline, E. Getzler, M. Vergne, *Heat Kernels and Dirac Operators* (1992)

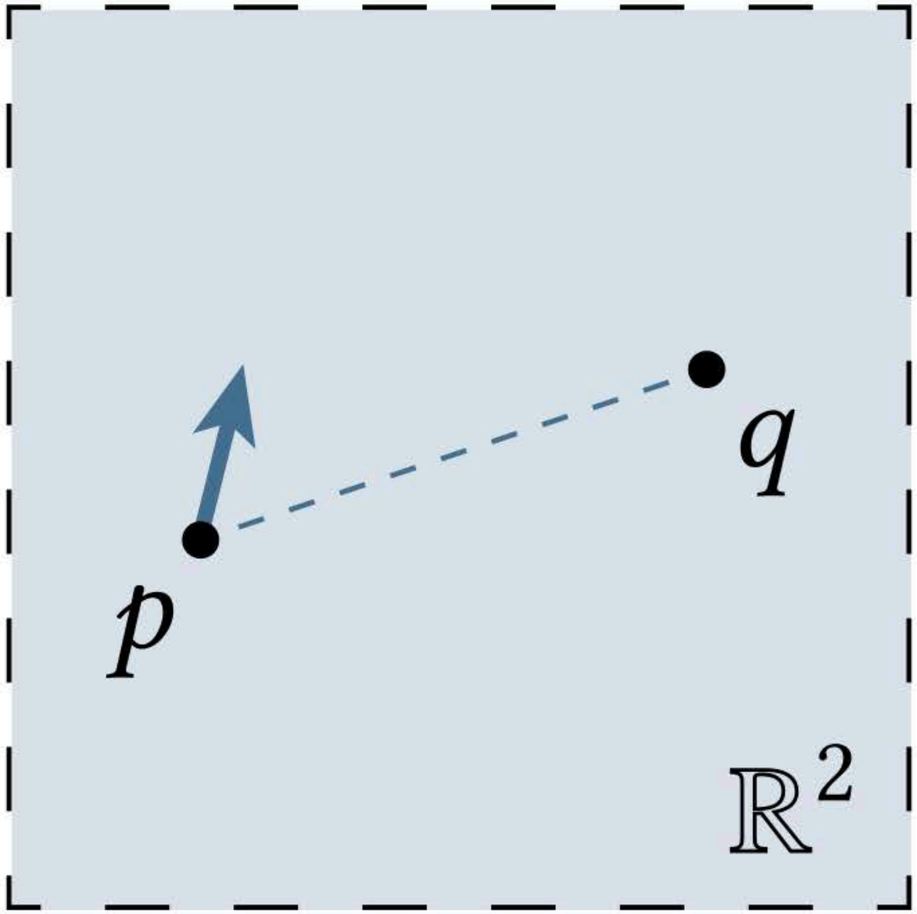
# Parallel transport along shortest geodesics



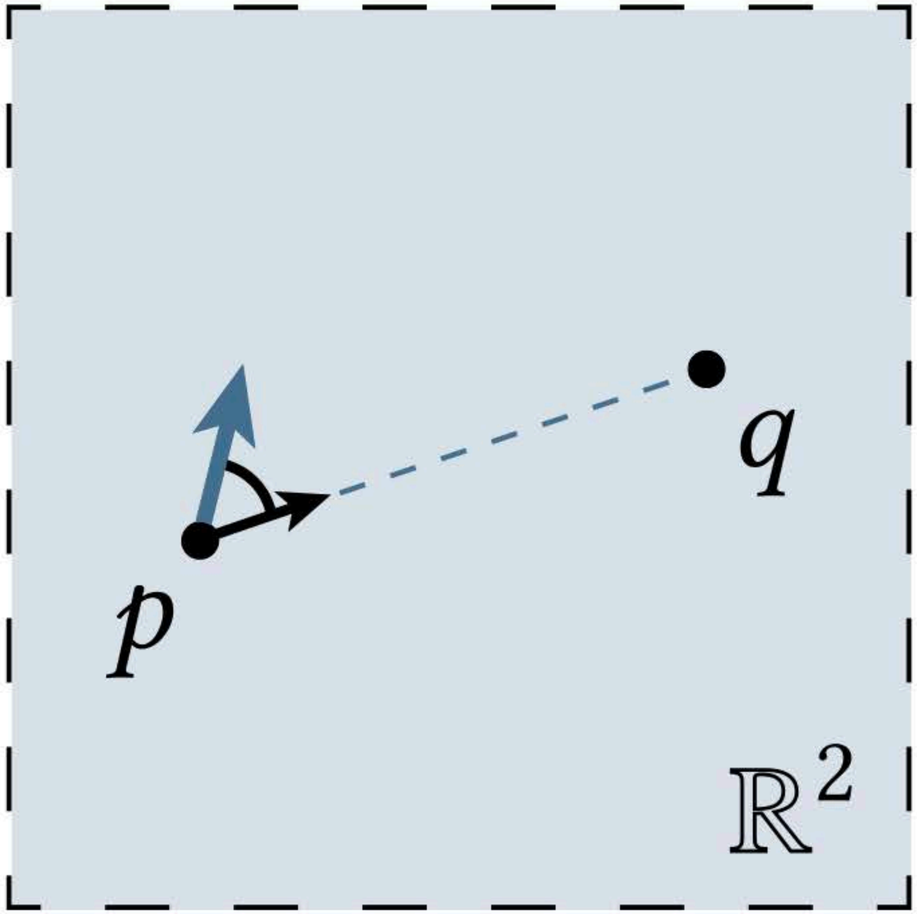
# Parallel transport along shortest geodesics



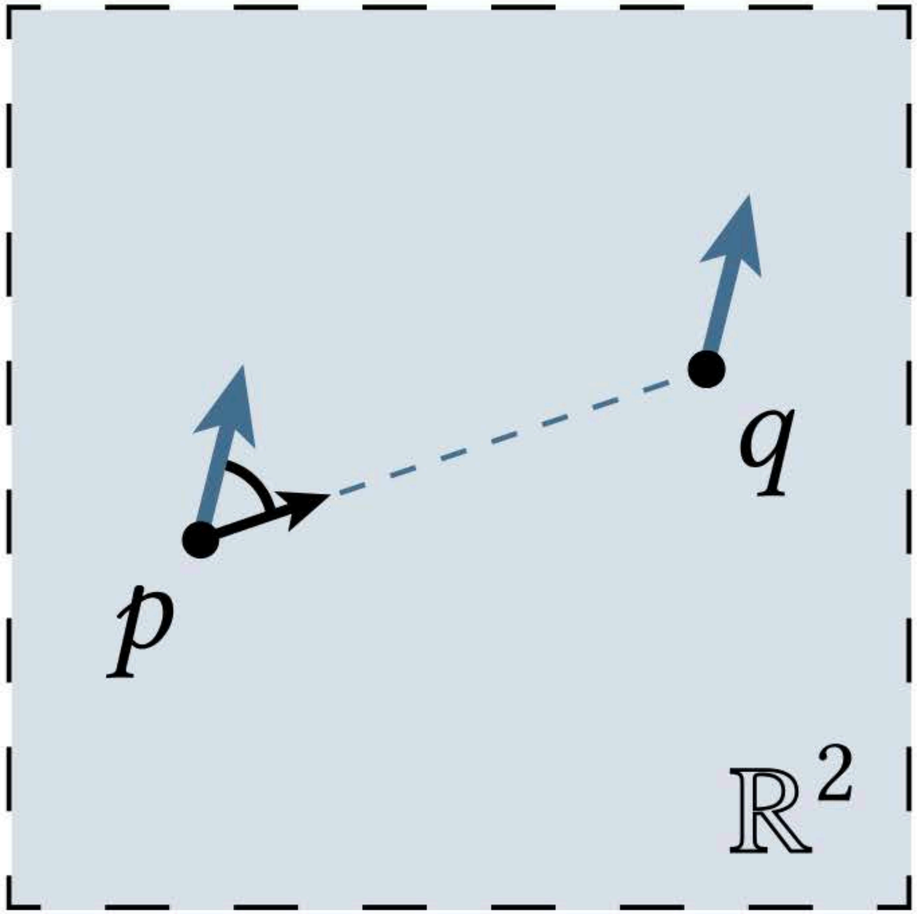
# Parallel transport along shortest geodesics



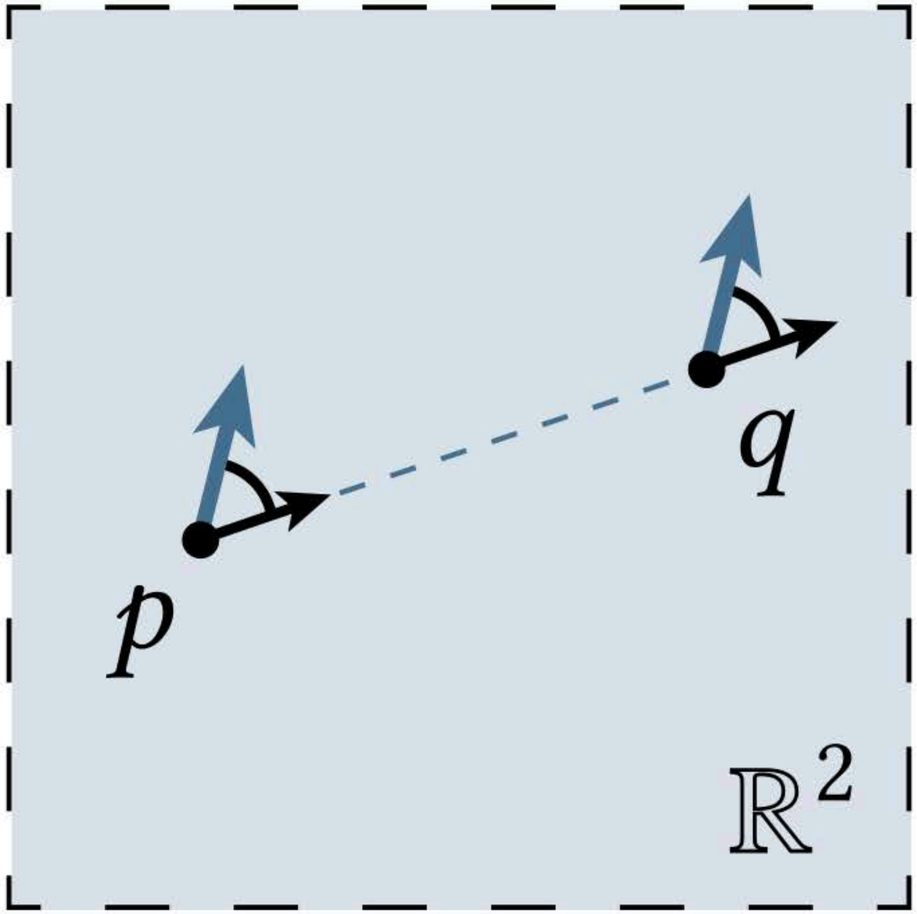
# Parallel transport along shortest geodesics



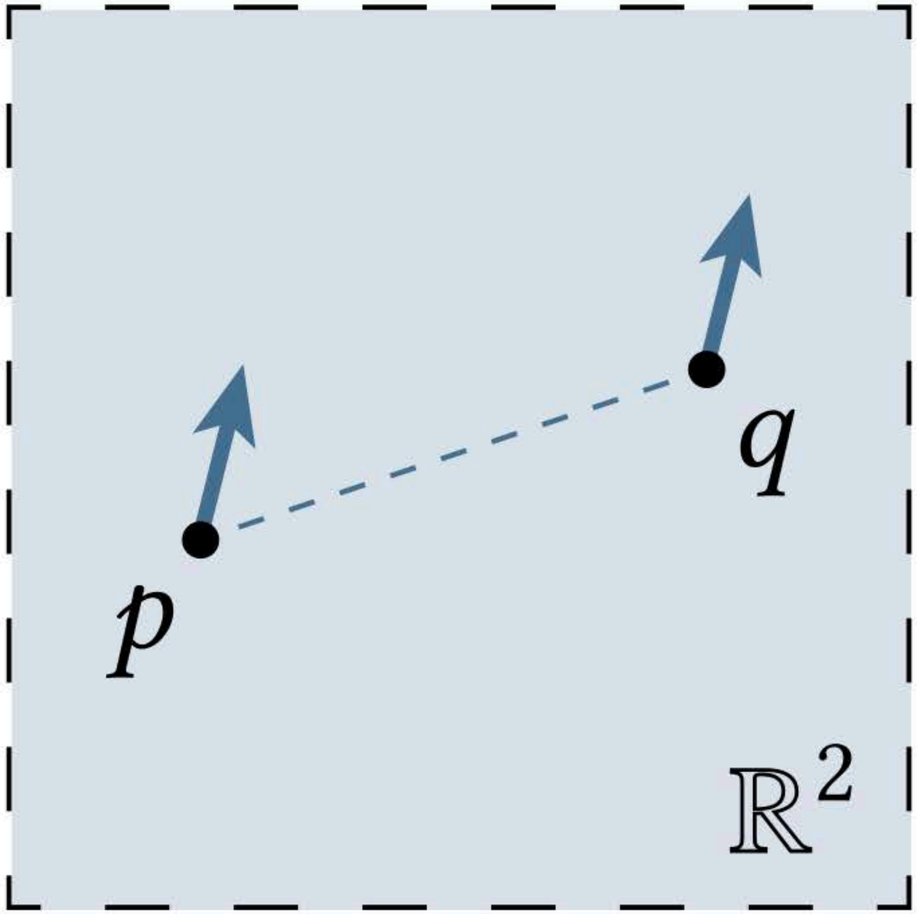
# Parallel transport along shortest geodesics



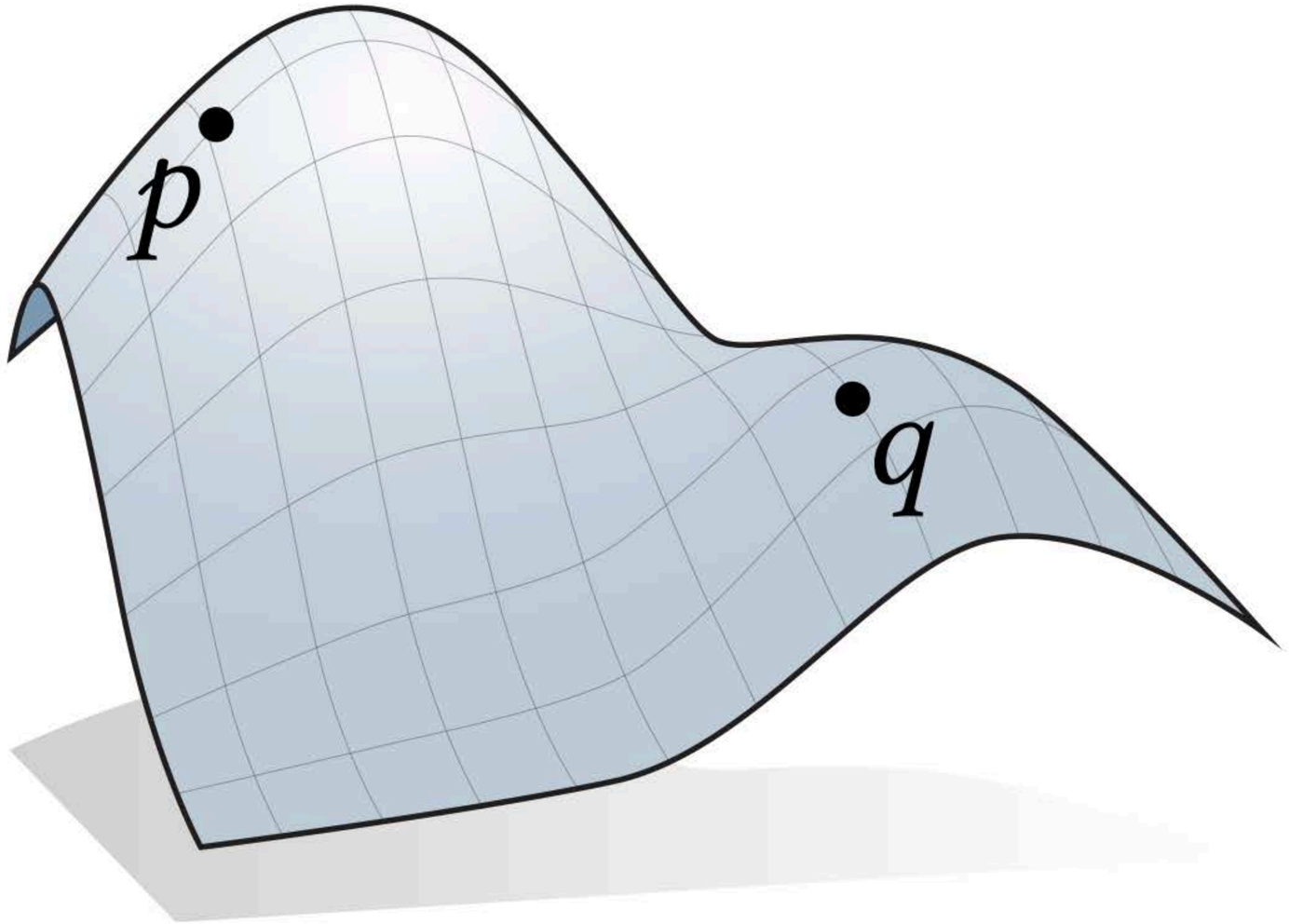
# Parallel transport along shortest geodesics



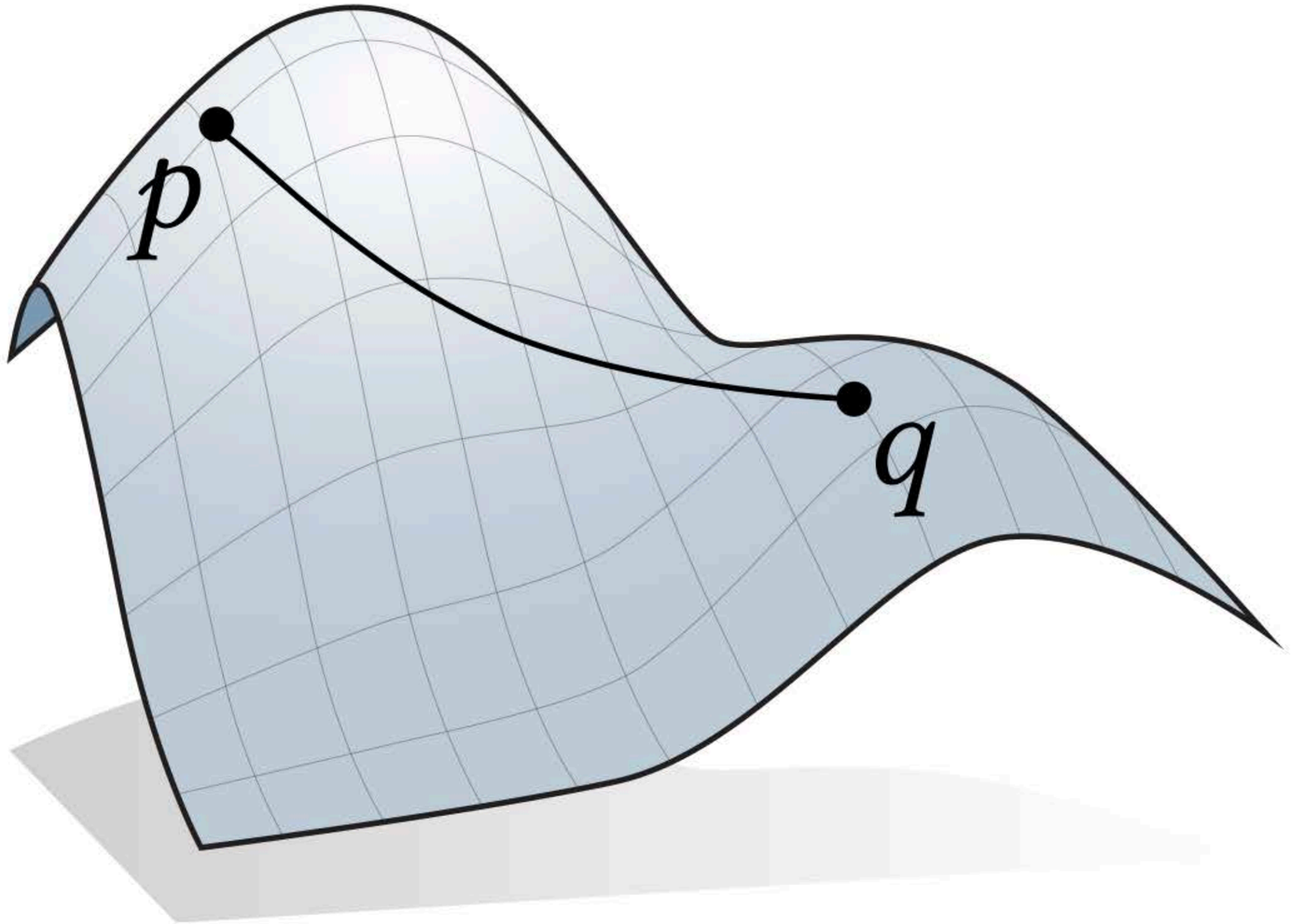
# Parallel transport along shortest geodesics



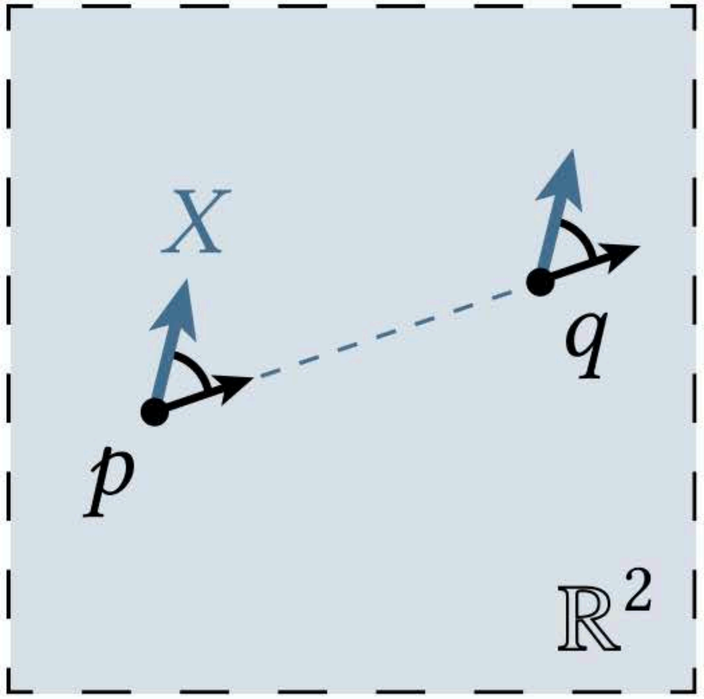
# Parallel transport on surfaces

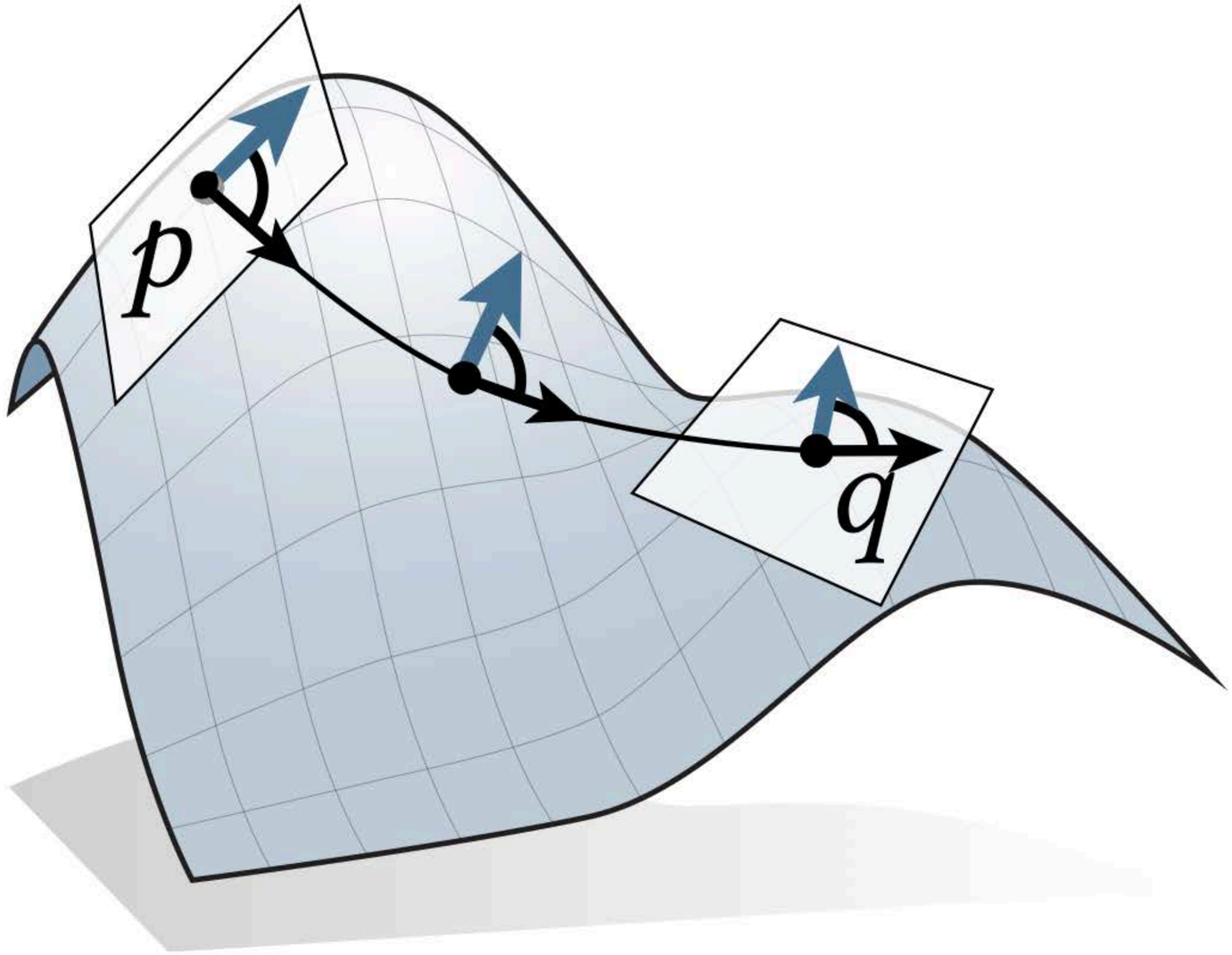


# Parallel transport on surfaces

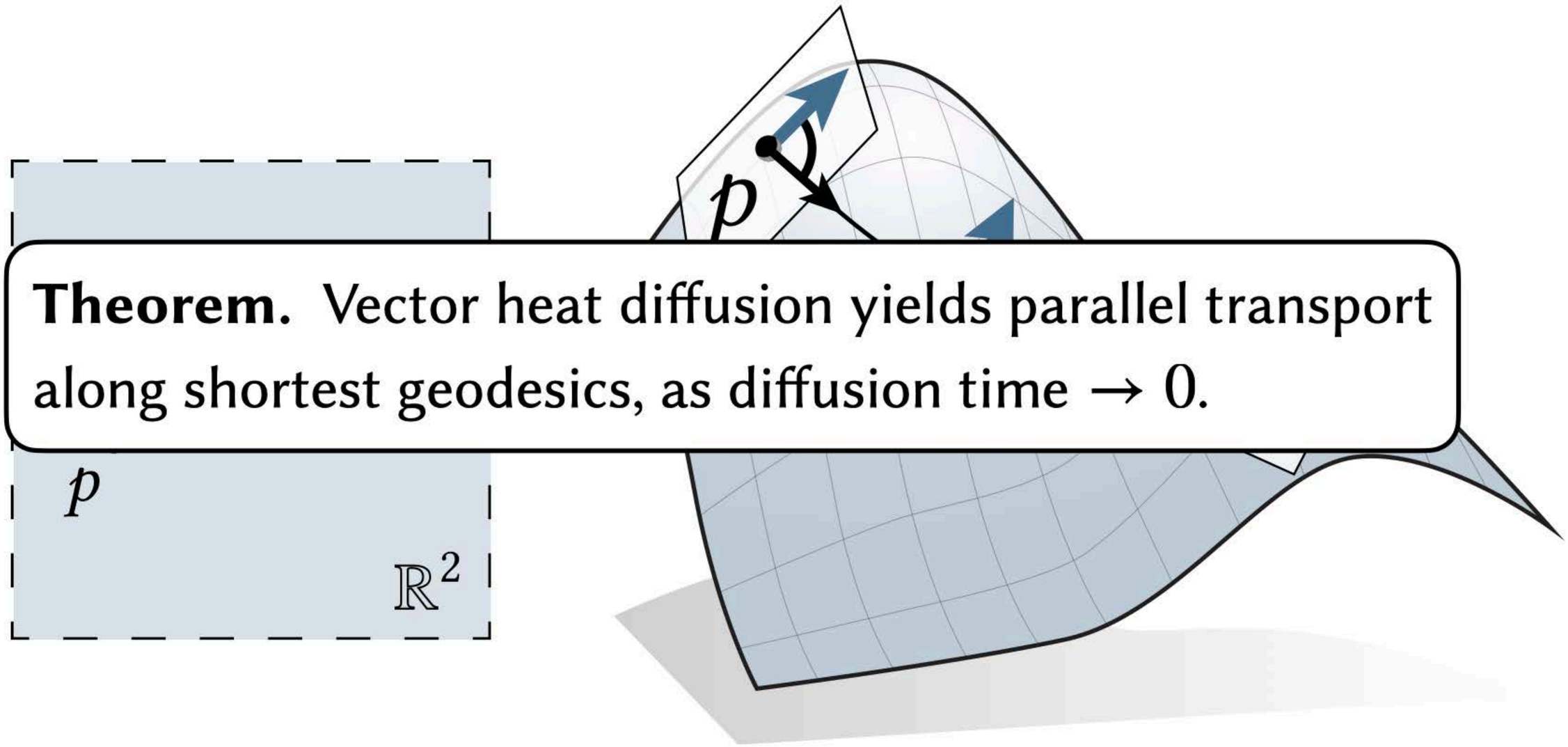


# Parallel transport on surfaces



# Parallel transport on surfaces




**Theorem.** Vector heat diffusion yields parallel transport along shortest geodesics, as diffusion time $\to 0$.

# Building intuition: the scalar heat kernel

# Building intuition: the scalar heat kernel

scalar heat equation

$$\frac{\partial q}{\partial t} = \Delta q$$

# Building intuition: the scalar heat kernel

scalar heat equation

change in time

$$\frac{\partial q}{\partial t} = \Delta q$$

# Building intuition: the scalar heat kernel

scalar heat equation

change in time

$$\frac{\partial q}{\partial t} = \Delta q$$

average deviation from neighbors
convexity/concavity

# Building intuition: the scalar heat kernel

scalar heat equation $\frac{\partial q}{\partial t} = \Delta q$

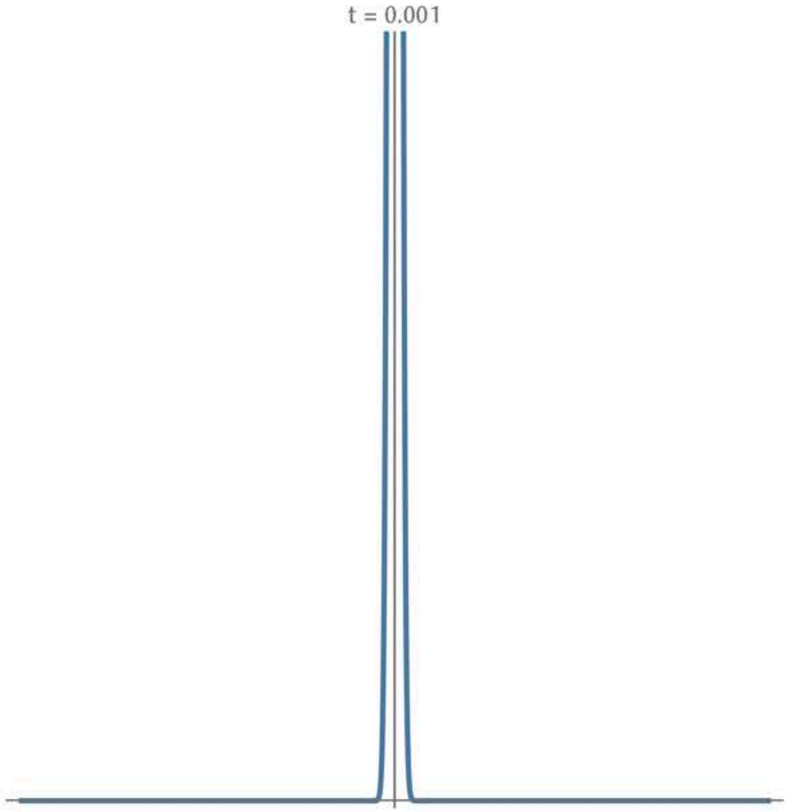
*fundamental solution*

scalar heat kernel

$$k_t(x, y) = (4\pi t)^{-d/2} \exp\left(-\|x - y\|^2 / 4t\right)$$



*2D*




*1D*

# Building intuition: the scalar heat kernel

scalar heat equation $\frac{\partial q}{\partial t} = \Delta q$

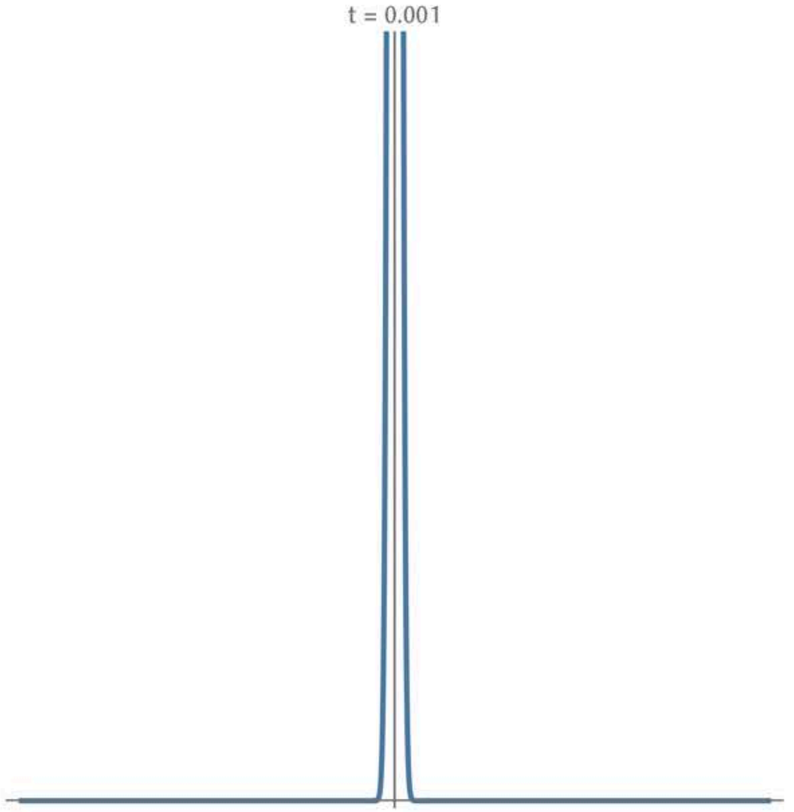
*fundamental solution*

scalar heat kernel

$$k_t(x, y) = (4\pi t)^{-d/2} \exp\left(-\|x - y\|^2 / 4t\right)$$



*2D*




*1D*

# Building intuition: the vector heat kernel

vector heat equation

$$\frac{\partial X}{\partial t} = \Delta^{\nabla} X$$

# Building intuition: the vector heat kernel

vector heat equation

$$\frac{\partial X}{\partial t} = \Delta^{\nabla} X$$

vector-valued data

# Building intuition: the vector heat kernel

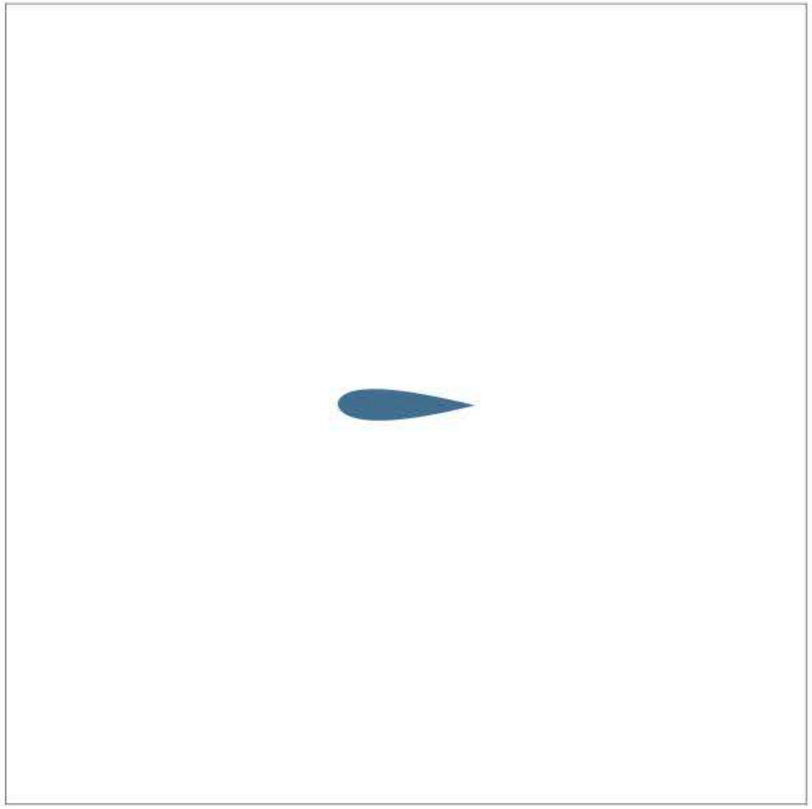
vector heat equation

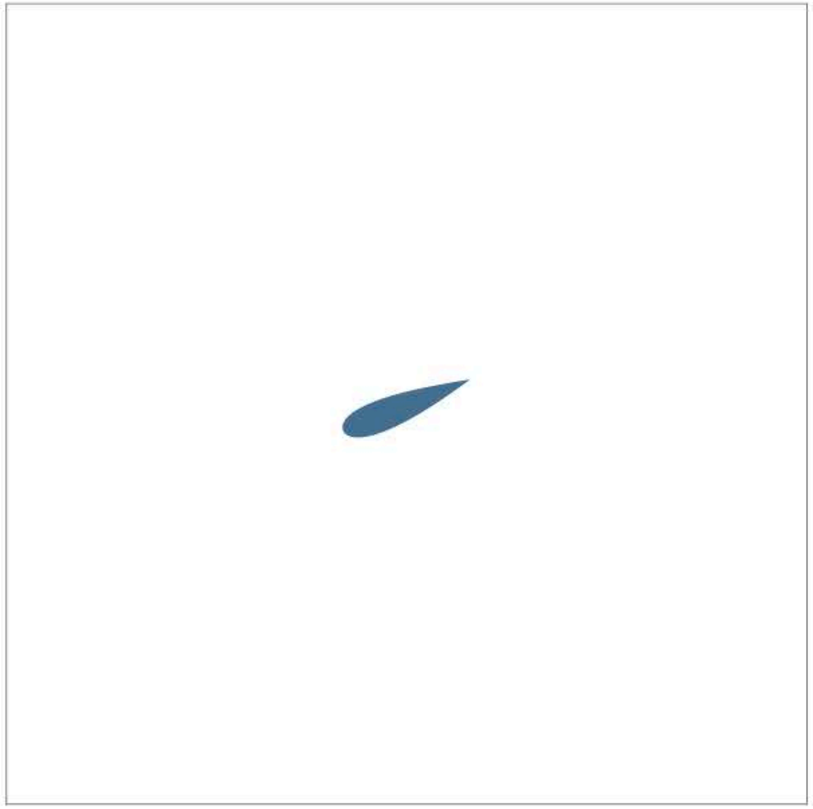
change in time $\frac{\partial X}{\partial t} = \Delta^{\nabla} X$ average deviation from neighbors

# Building intuition: the vector heat kernel

vector heat equation $\frac{\partial X}{\partial t} = \Delta^{\nabla} X$

In $\mathbb{R}^d$:

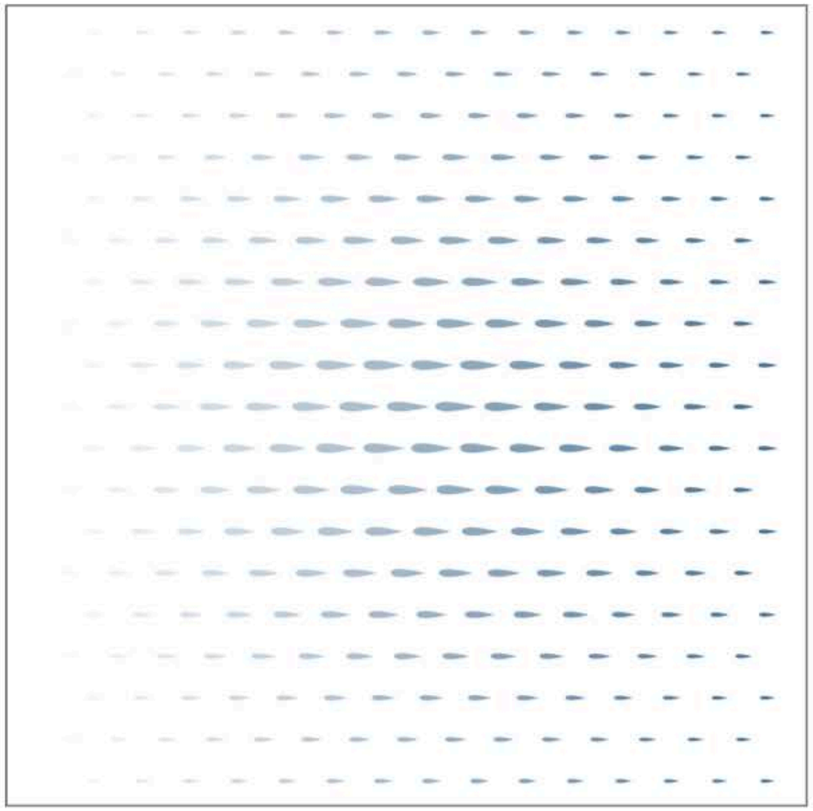


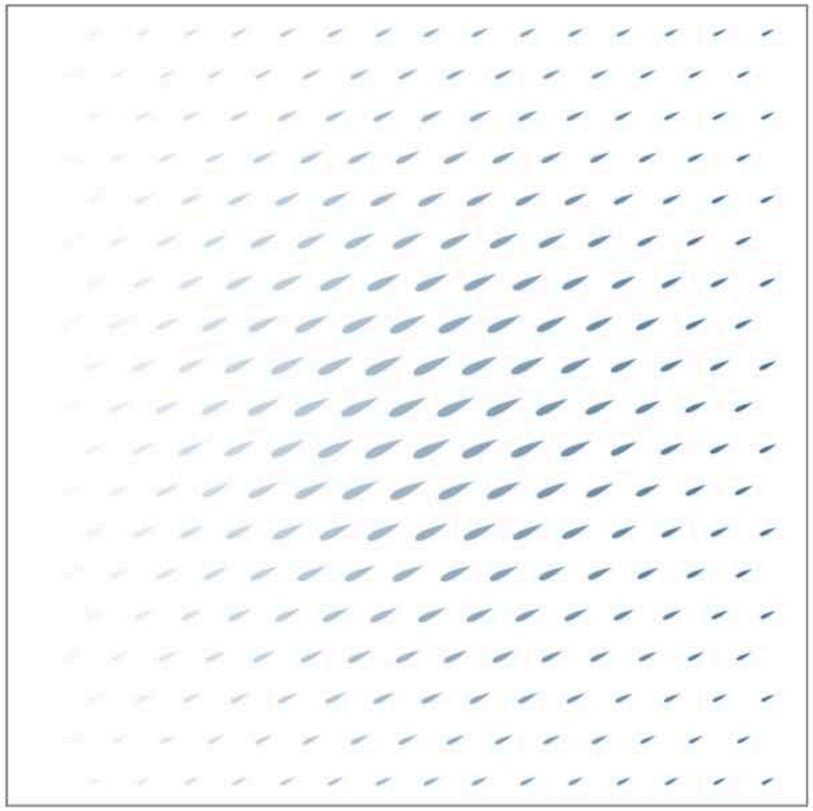


# Building intuition: the vector heat kernel

vector heat equation $\frac{\partial X}{\partial t} = \Delta^{\nabla} X$

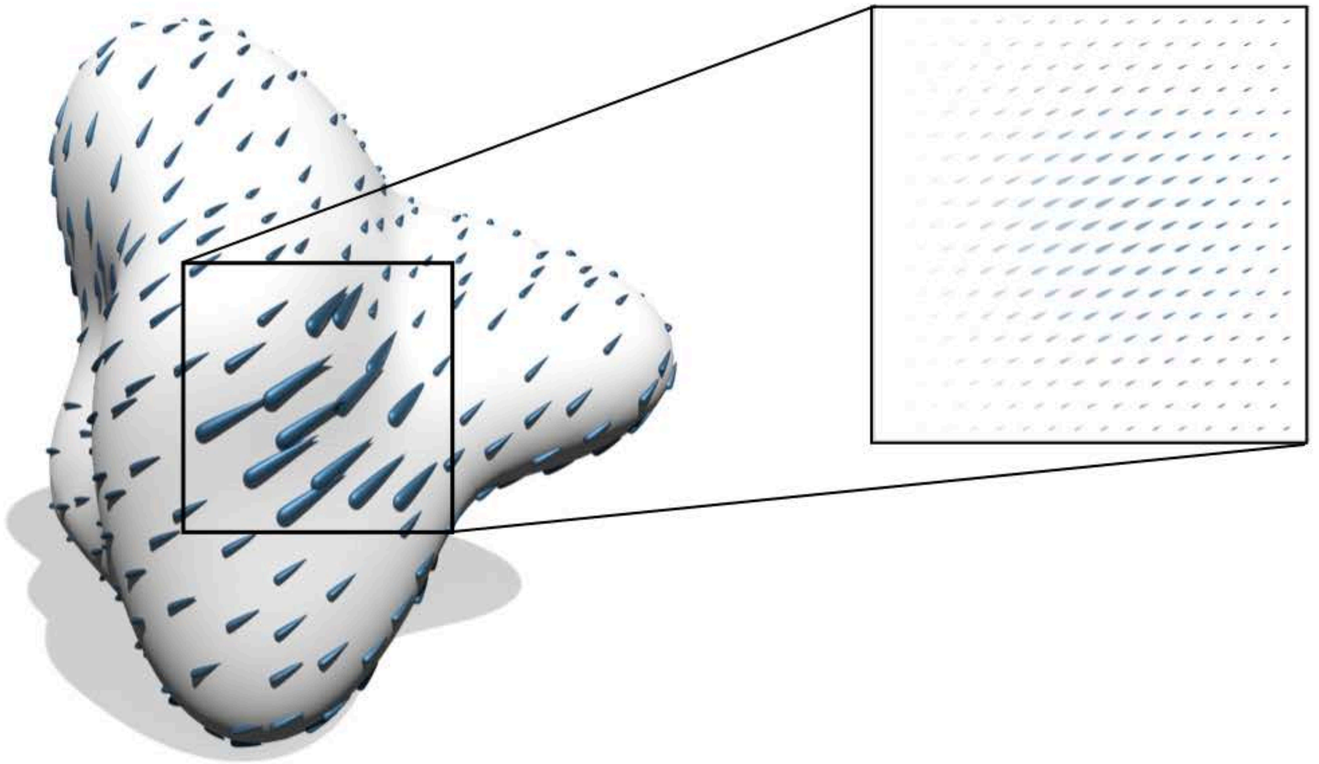
In $\mathbb{R}^d$:





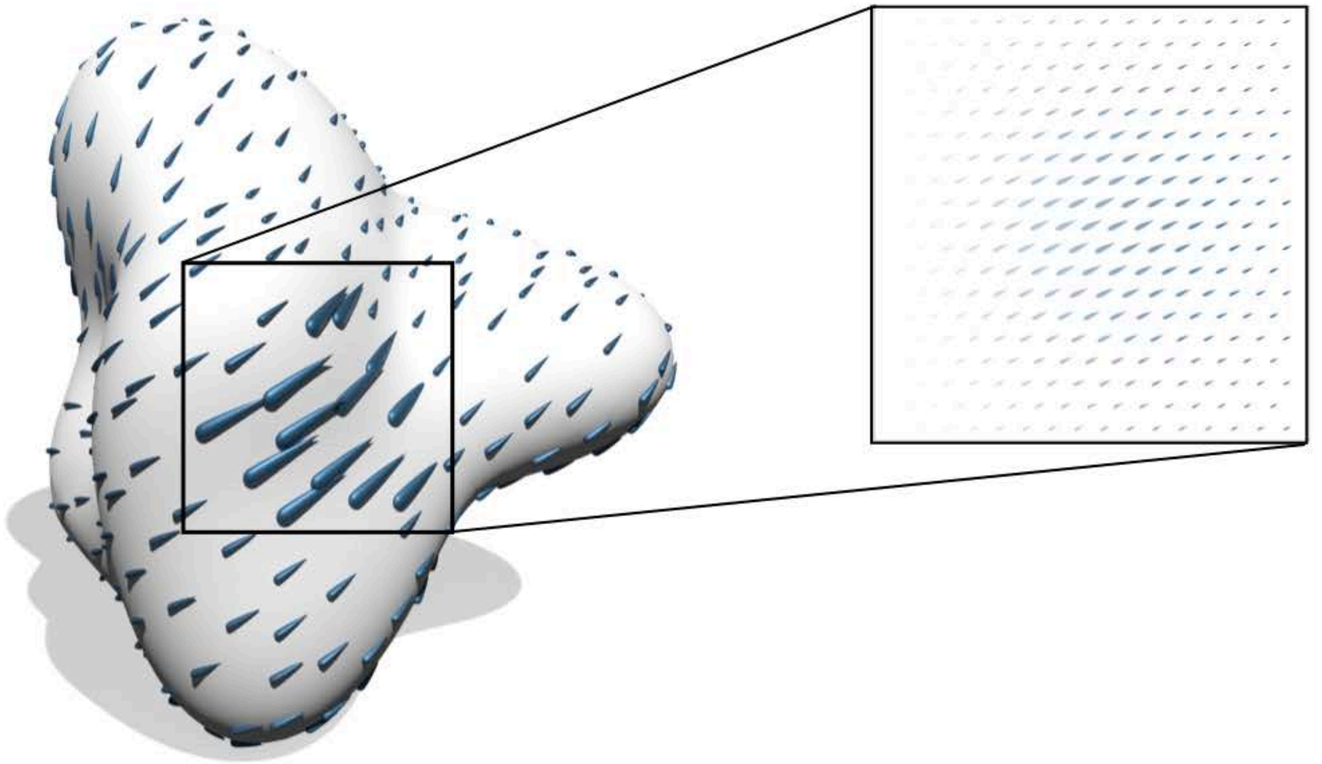
# Building intuition: the vector heat kernel

vector heat equation $\frac{\partial X}{\partial t} = \Delta^{\nabla} X$



# Building intuition: the vector heat kernel

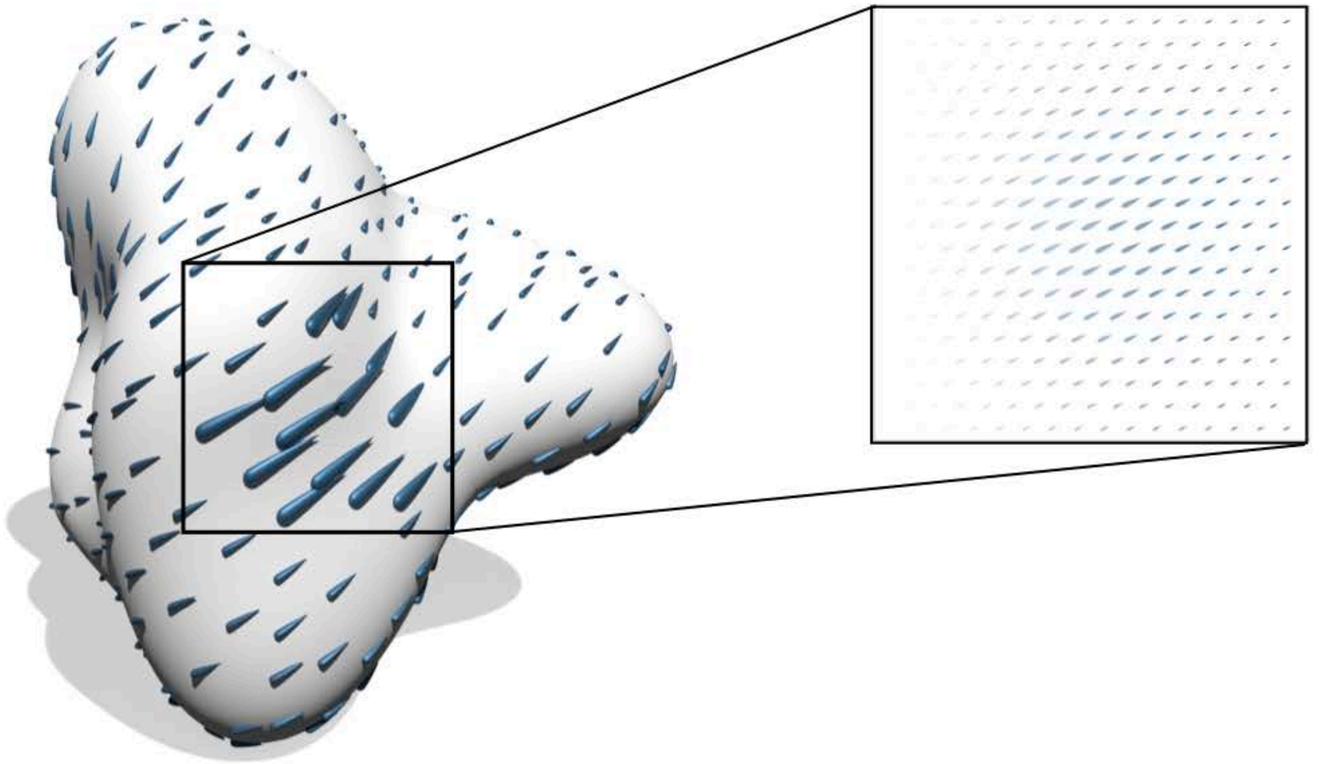
vector heat equation $\dfrac{\partial X}{\partial t} = \Delta^{\nabla} X$



vector heat kernel $k_t^{\nabla} \sim (4\pi t)^{-d/2} \exp\left(-\mathrm{dist}(x,y)^2/4t\right) c(x,y) \sum_{i=0}^{\infty} t^i \Phi_i^{\Delta}(x,y)$

# Building intuition: the vector heat kernel

vector heat equation $\dfrac{\partial X}{\partial t} = \Delta^{\nabla} X$

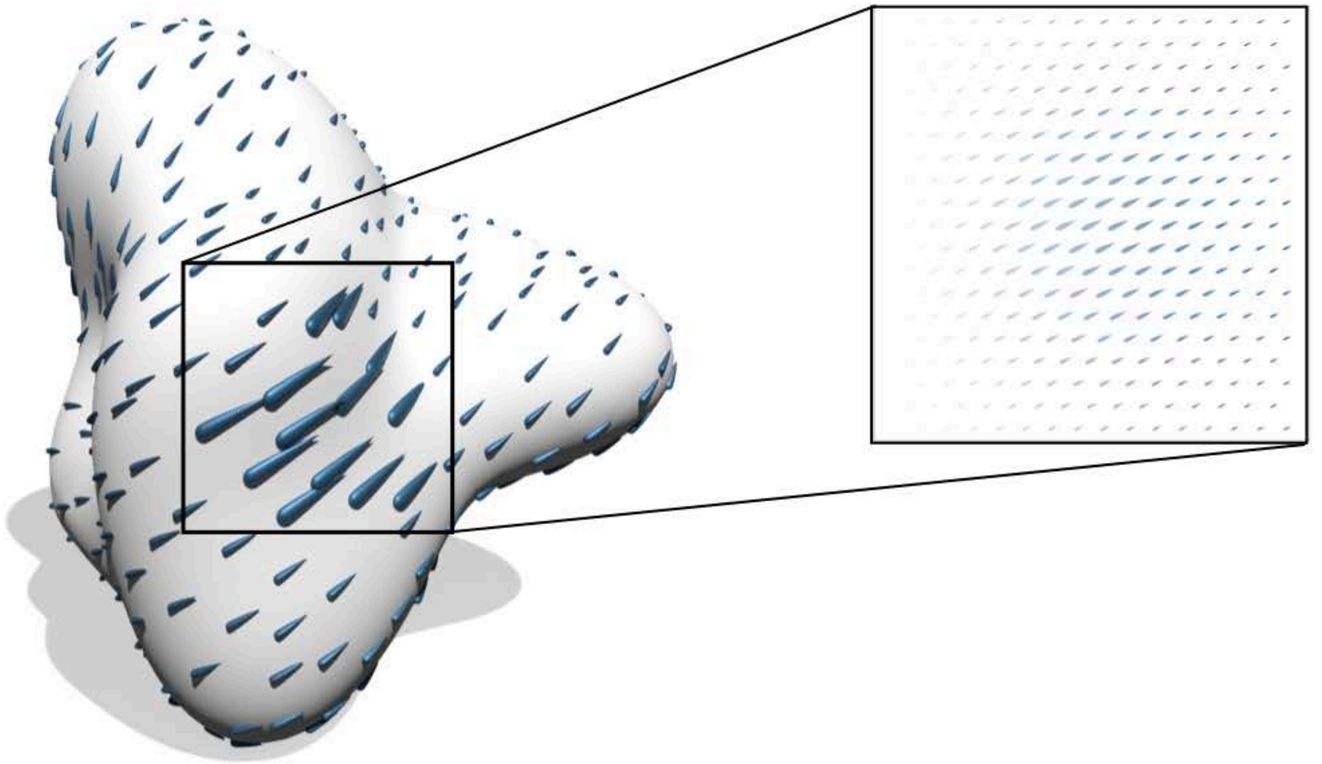


vector heat kernel $k_t^{\nabla} \sim (4\pi t)^{-d/2} \exp\left(-\mathrm{dist}(x,y)^2/4t\right) c(x,y) \sum_{i=0}^{\infty} t^i \Phi_i^{\Delta}(x,y)$

~ scalar heat decay

# Building intuition: the vector heat kernel

vector heat equation $\frac{\partial X}{\partial t} = \Delta^{\nabla} X$

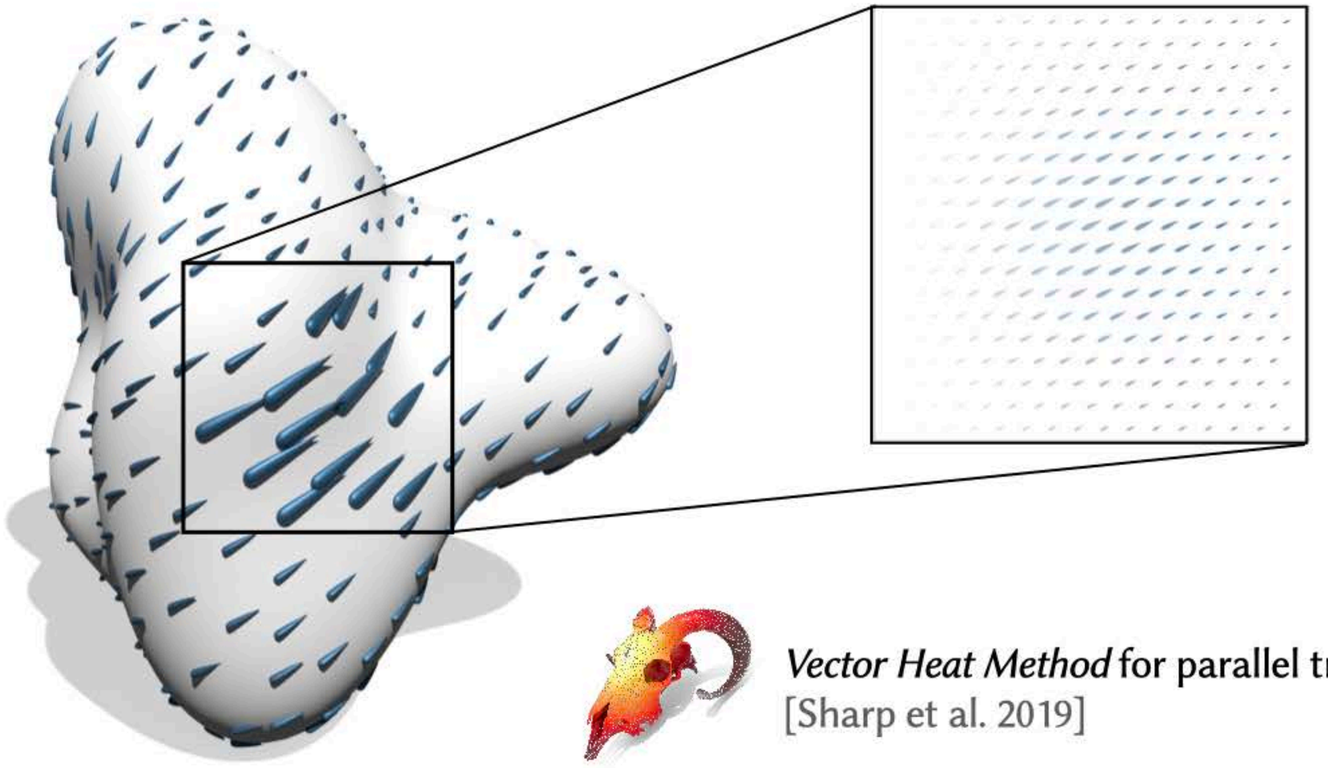


vector heat kernel $k_t^{\nabla} \sim (4\pi t)^{-d/2} \exp\left(-\mathrm{dist}(x,y)^2/4t\right) c(x,y) \sum_{i=0}^{\infty} t^i \Phi_i^{\Delta}(x,y)$ $\quad \Phi_0^{\Delta}(x,y) = P_{\gamma_{x\to y}}$

~ scalar heat decay

# Building intuition: the vector heat kernel

vector heat equation $\frac{\partial X}{\partial t} = \Delta^{\nabla} X$



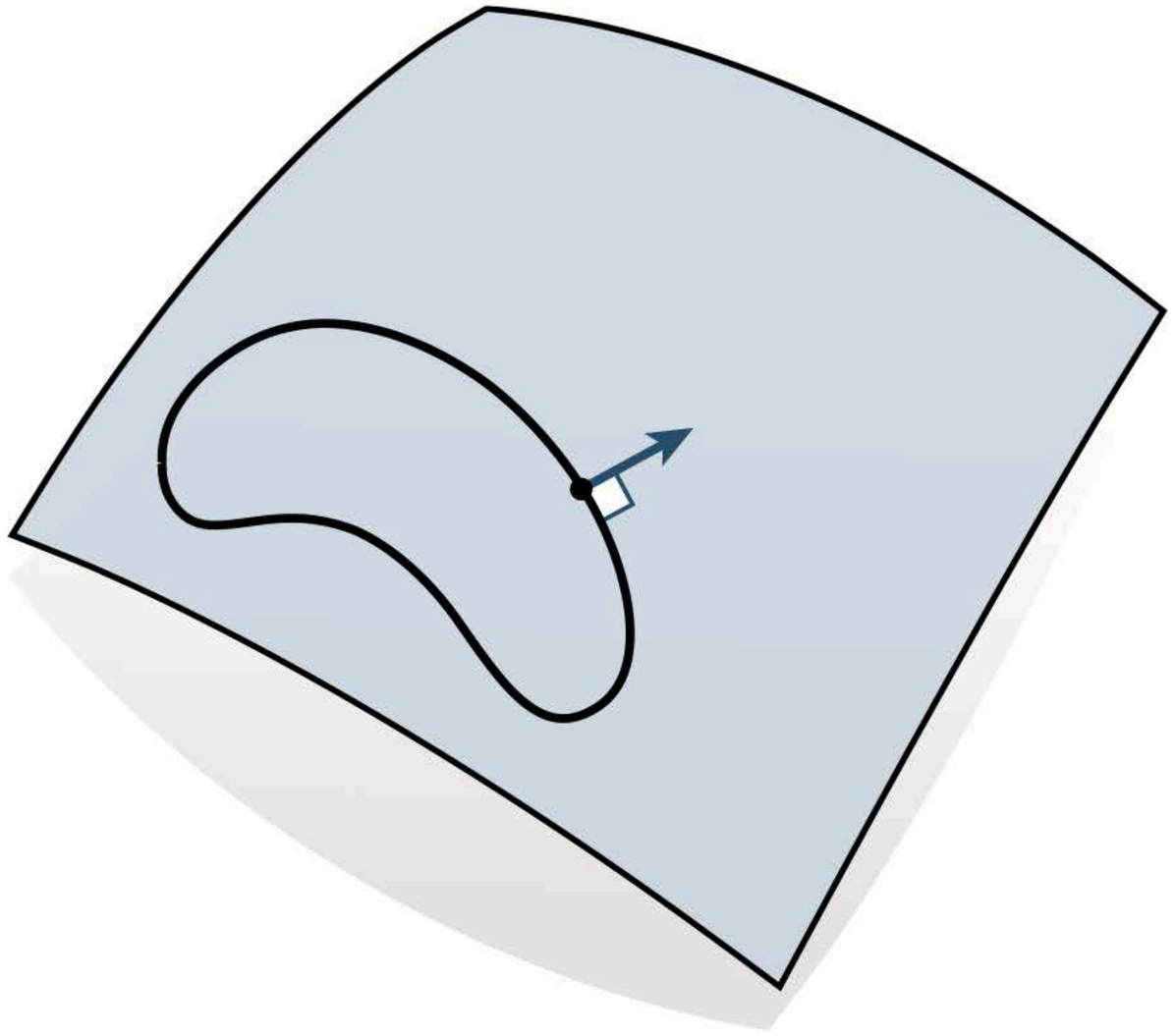


*Vector Heat Method* for parallel transport
[Sharp et al. 2019]

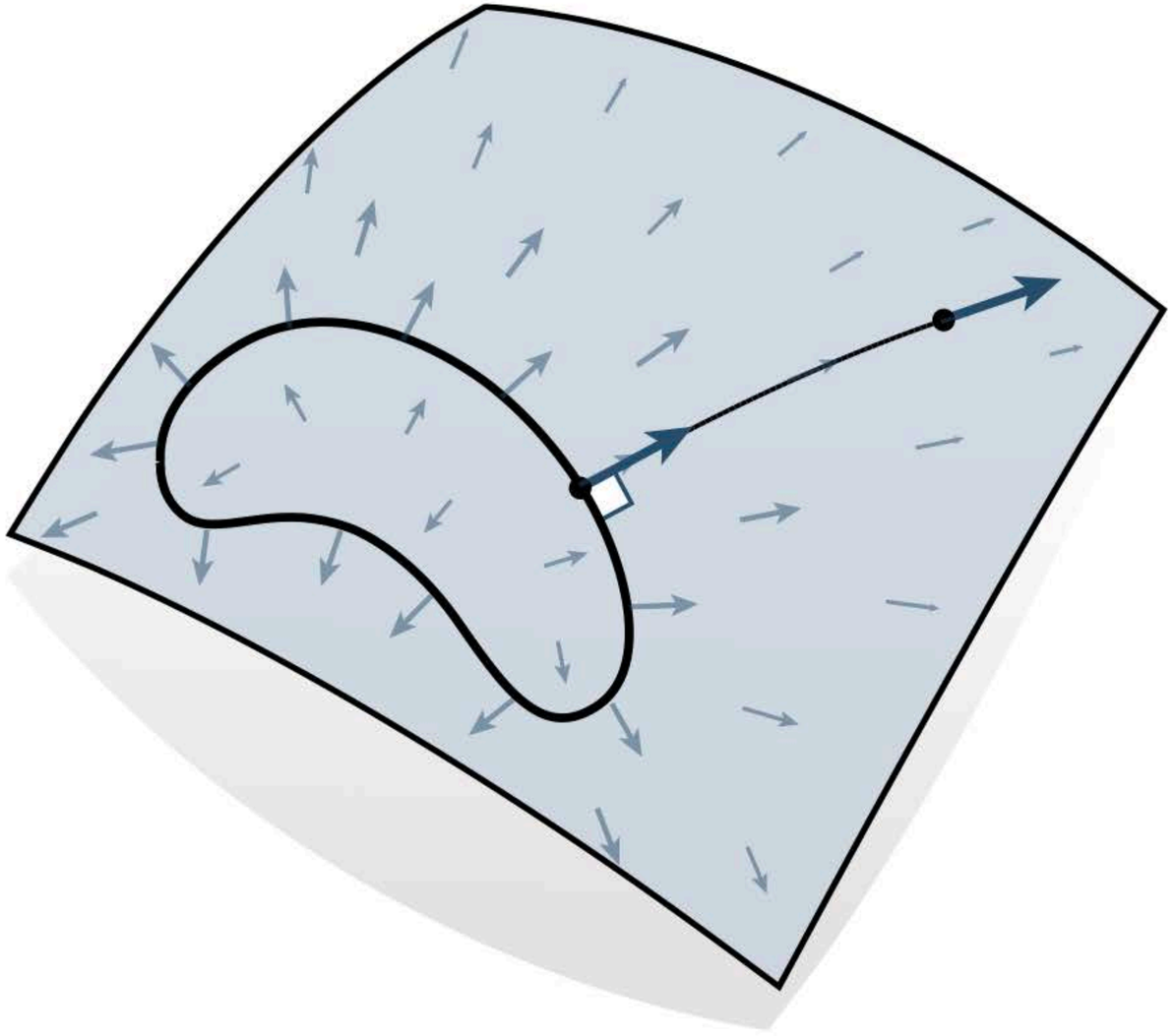
vector heat kernel $k_t^{\nabla} \sim (4\pi t)^{-d/2} \exp\left(-\mathrm{dist}(x,y)^2/4t\right) c(x,y) \sum_{i=0}^{\infty} t^i \Phi_i^{\Delta}(x,y)$ $\quad \Phi_0^{\Delta}(x,y) = P_{\gamma_{x\to y}}$

~ scalar heat decay

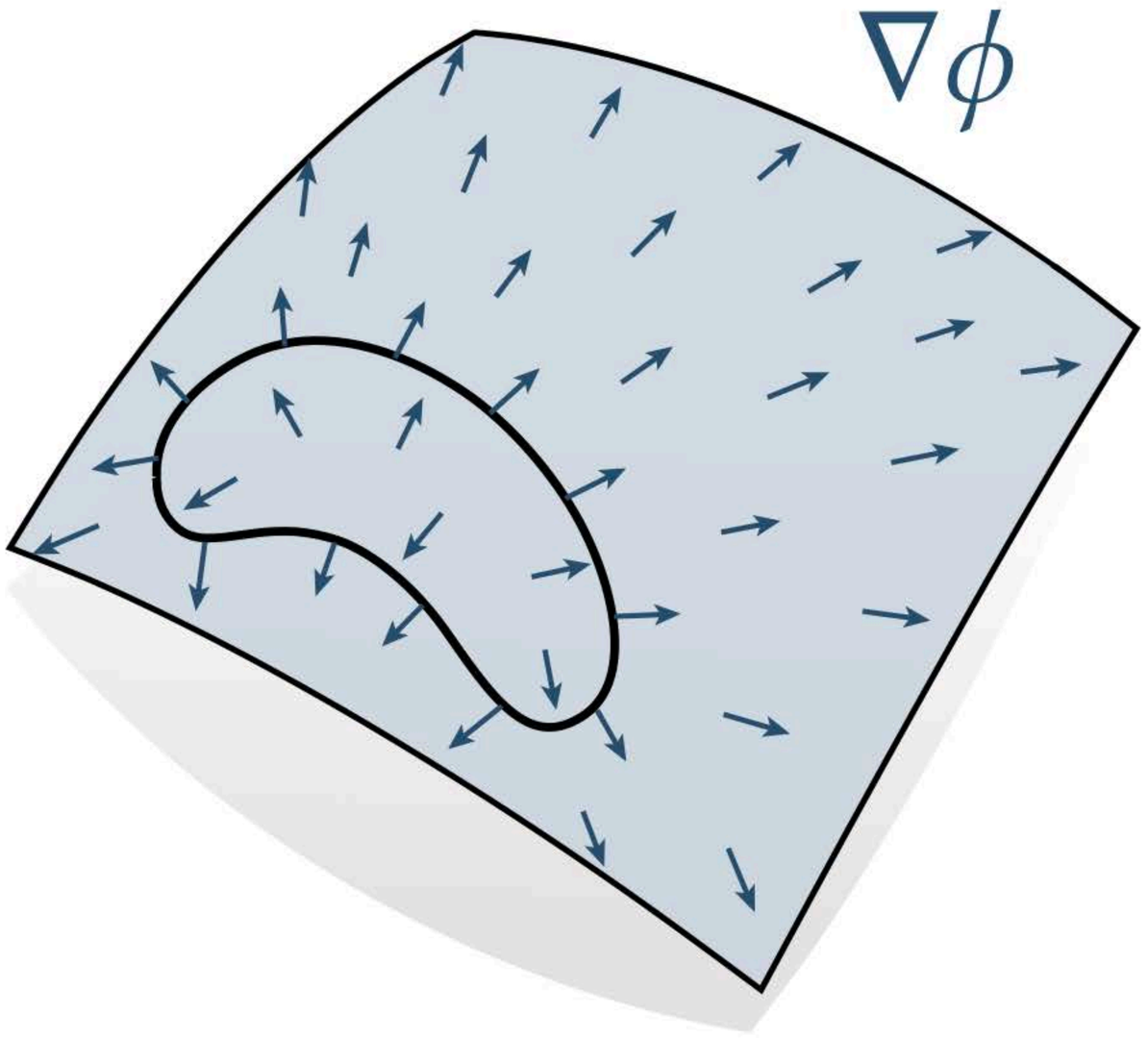
# Diffused normal vectors are parallel to the signed distance gradient



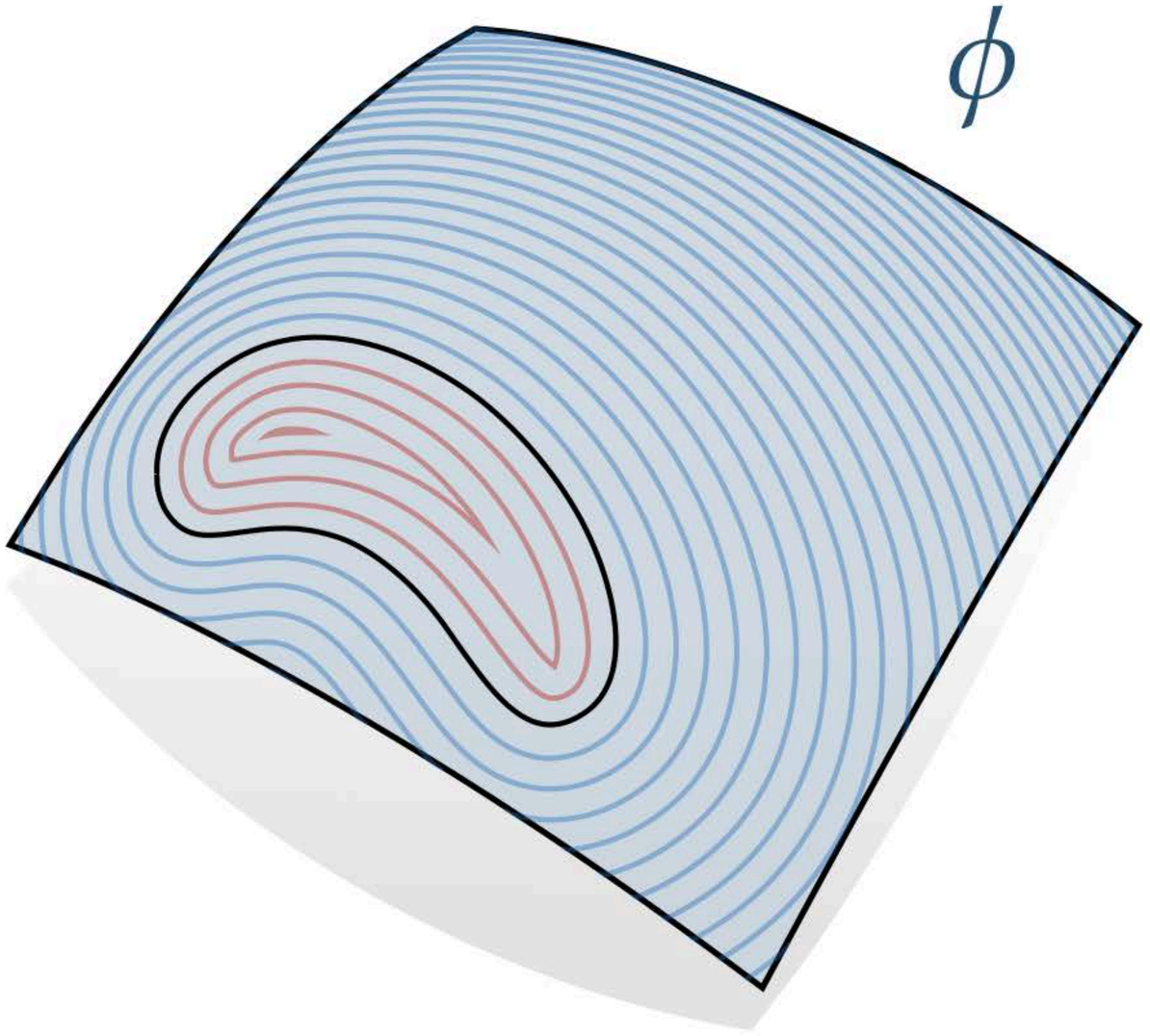
# Diffused normal vectors are parallel to the signed distance gradient



# Normalize and integrate into a distance function



# Normalize and integrate into a distance function



# We decouple reconstruction from distance quality

*input point cloud*



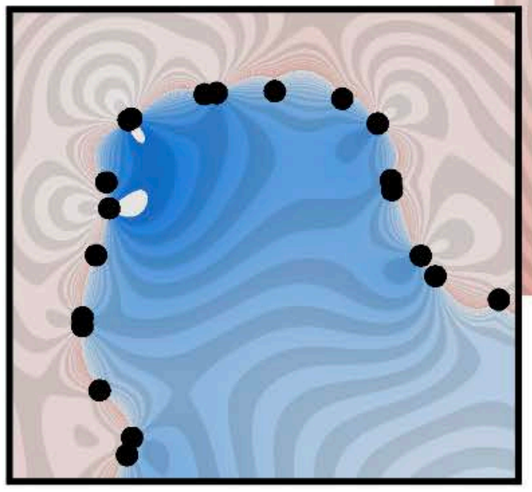
# We decouple reconstruction from distance quality

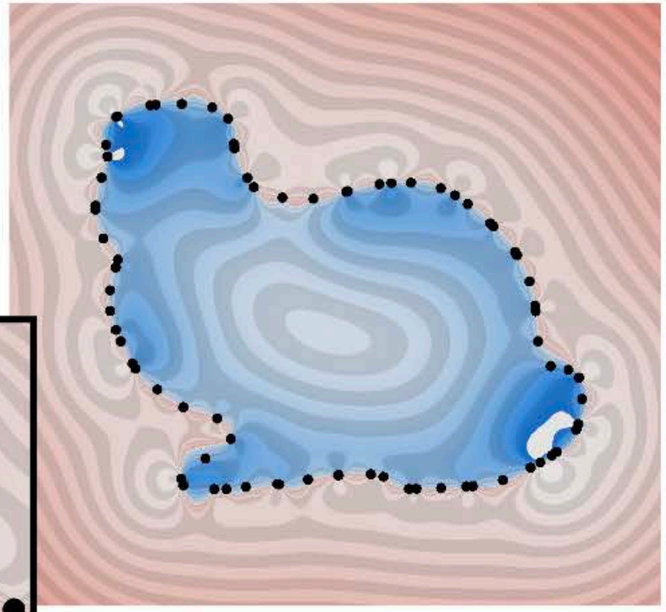
Before, screening/diffusion $\lambda$ was coupled to both regression quality and distance accuracy.

*input point cloud*



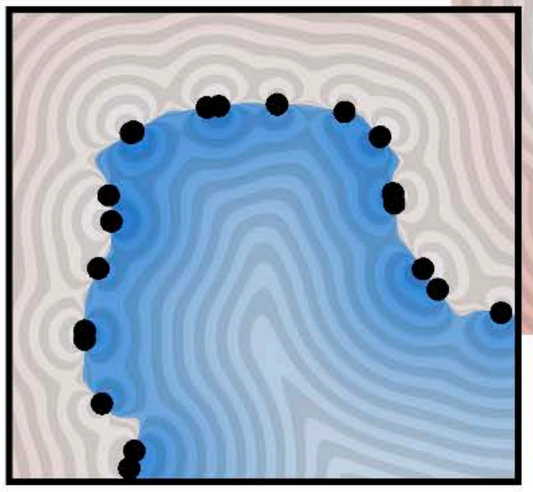
*small* $\lambda$ ⟷ *large* $\lambda$

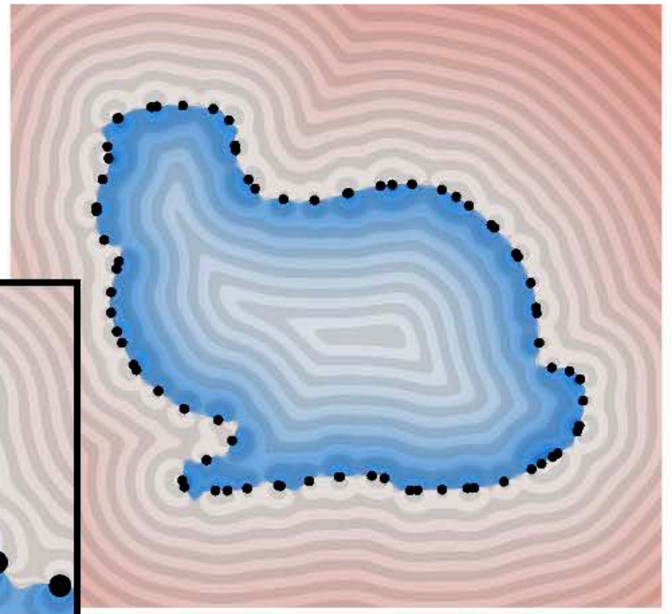




smoother reconstruction

bad eikonality





better eikonality

bad level sets

# We provide accurate distance to a good level set

*input point cloud*

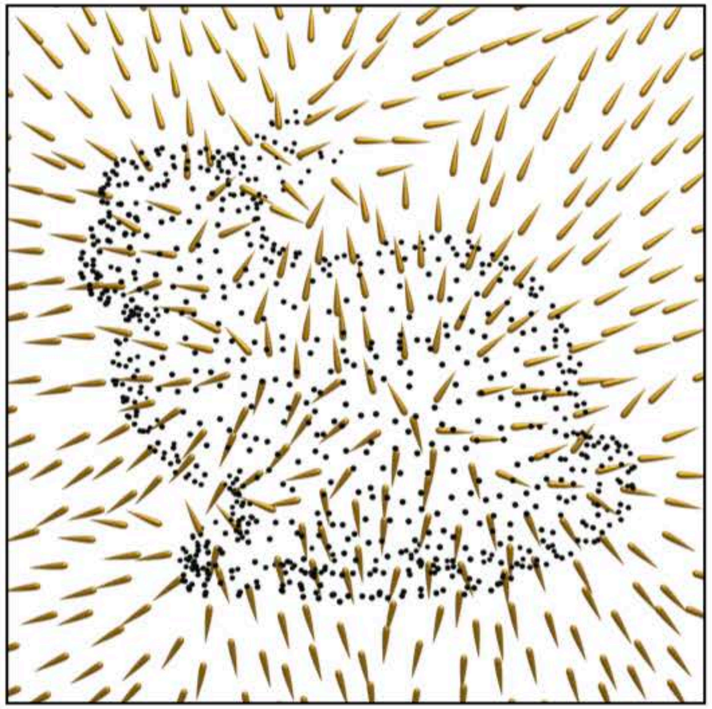


# We provide accurate distance to a good level set

*input point cloud*



*normalize diffused gradient vectors*

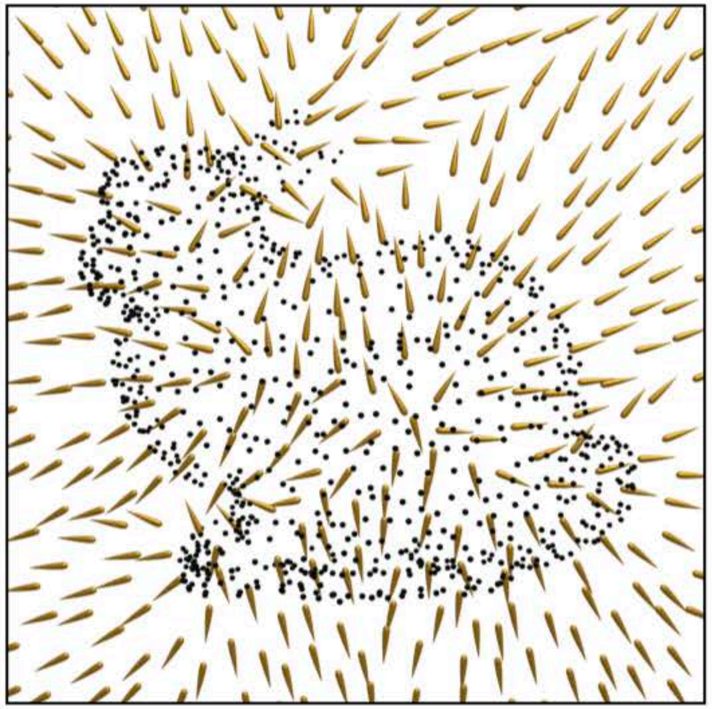


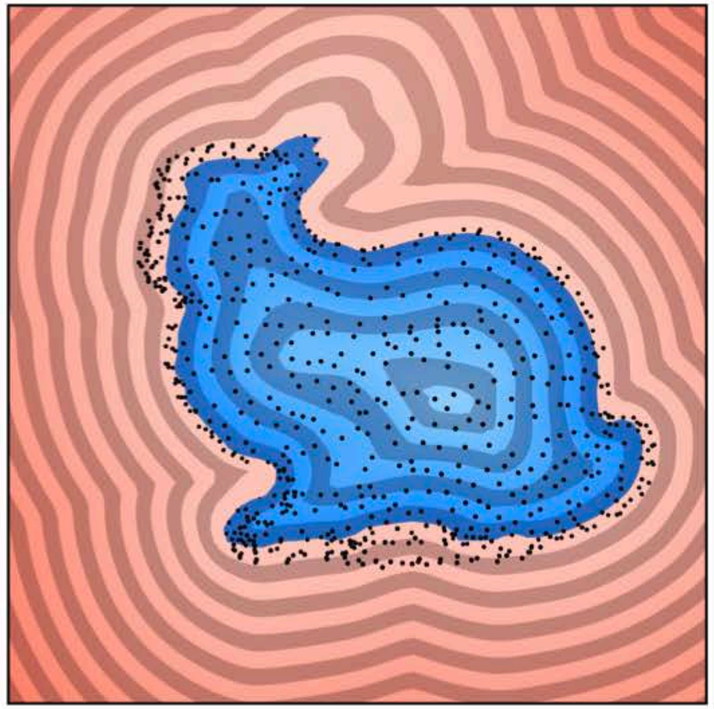
# We provide accurate distance to a good level set

*input point cloud*



*normalize diffused gradient vectors*

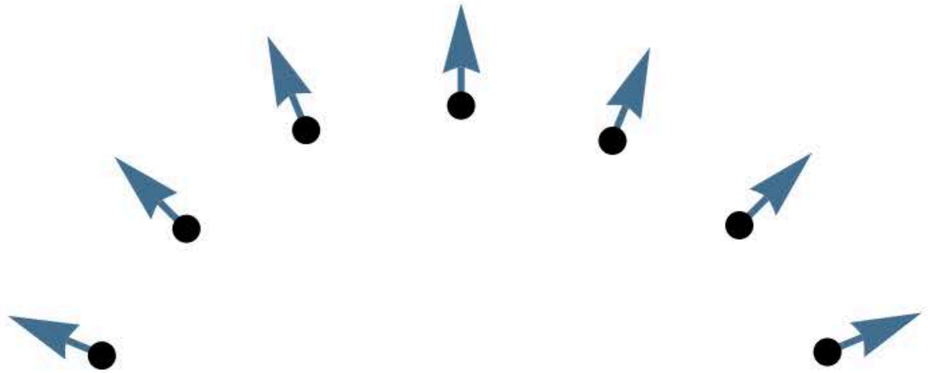




*(isosurfaces may look warped as cross-sections)*

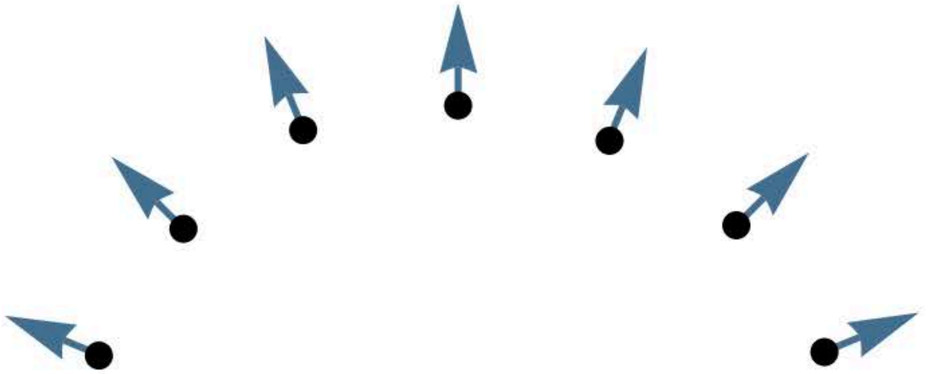
# Only magnitudes are concentrated

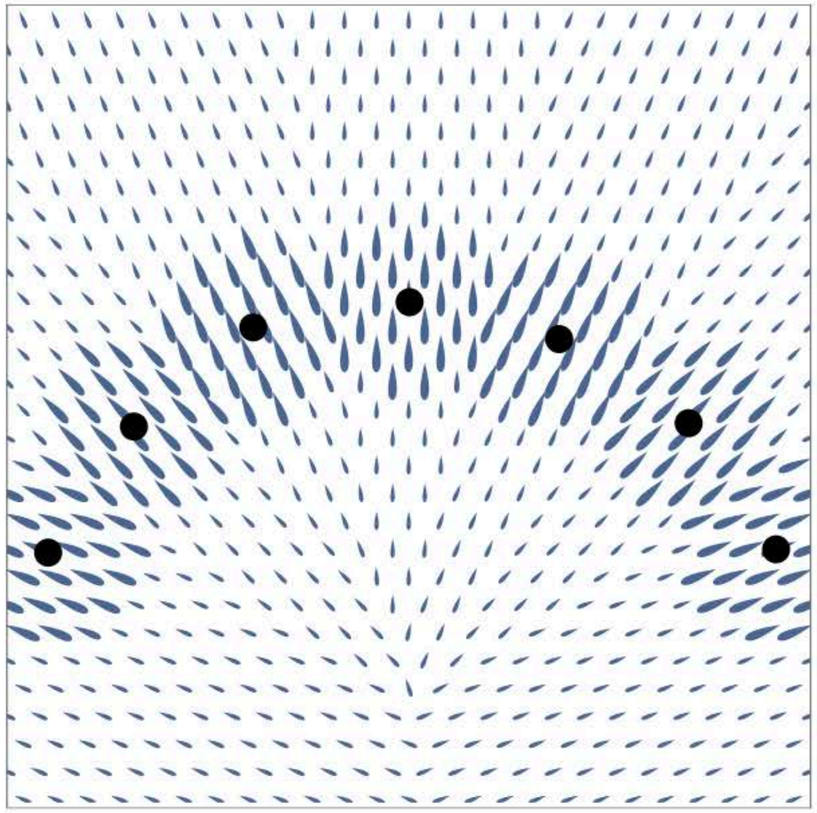
*source points and vectors*



# Only magnitudes are concentrated

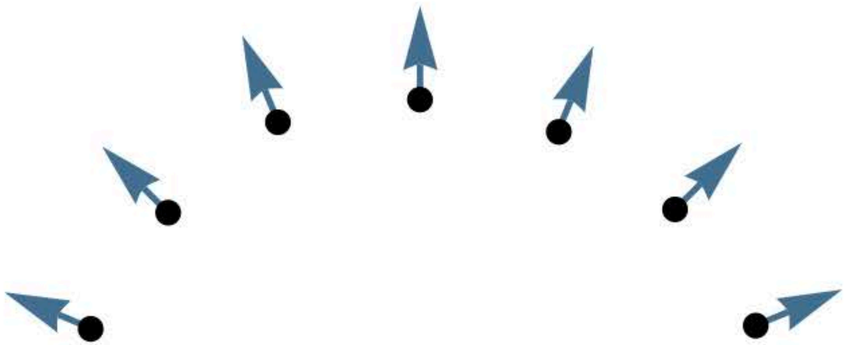
*source points and vectors*

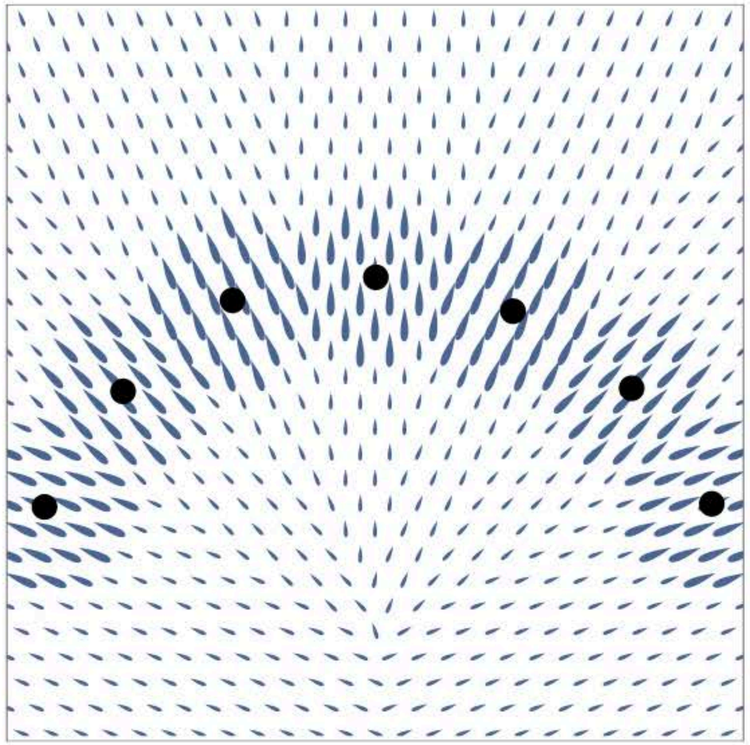


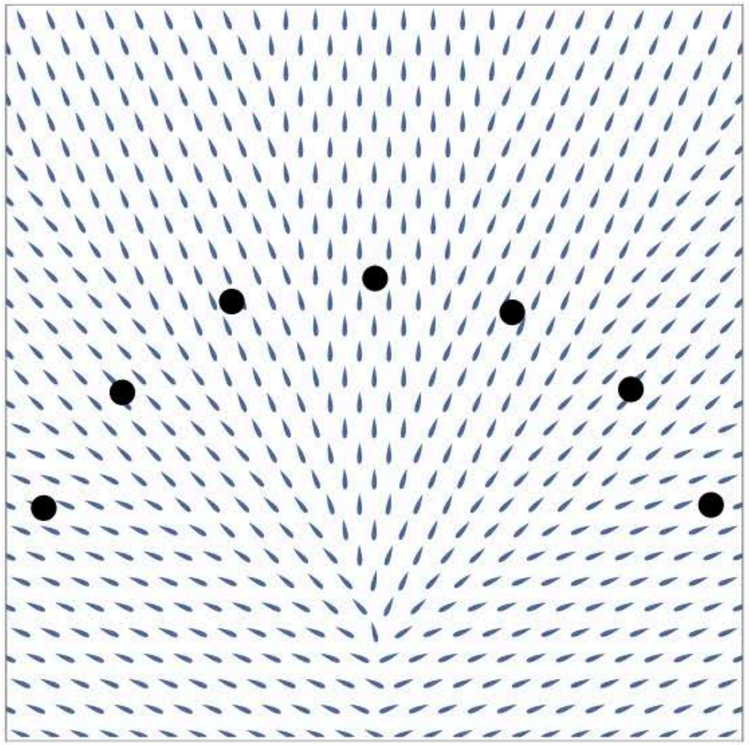


*diffused vectors*

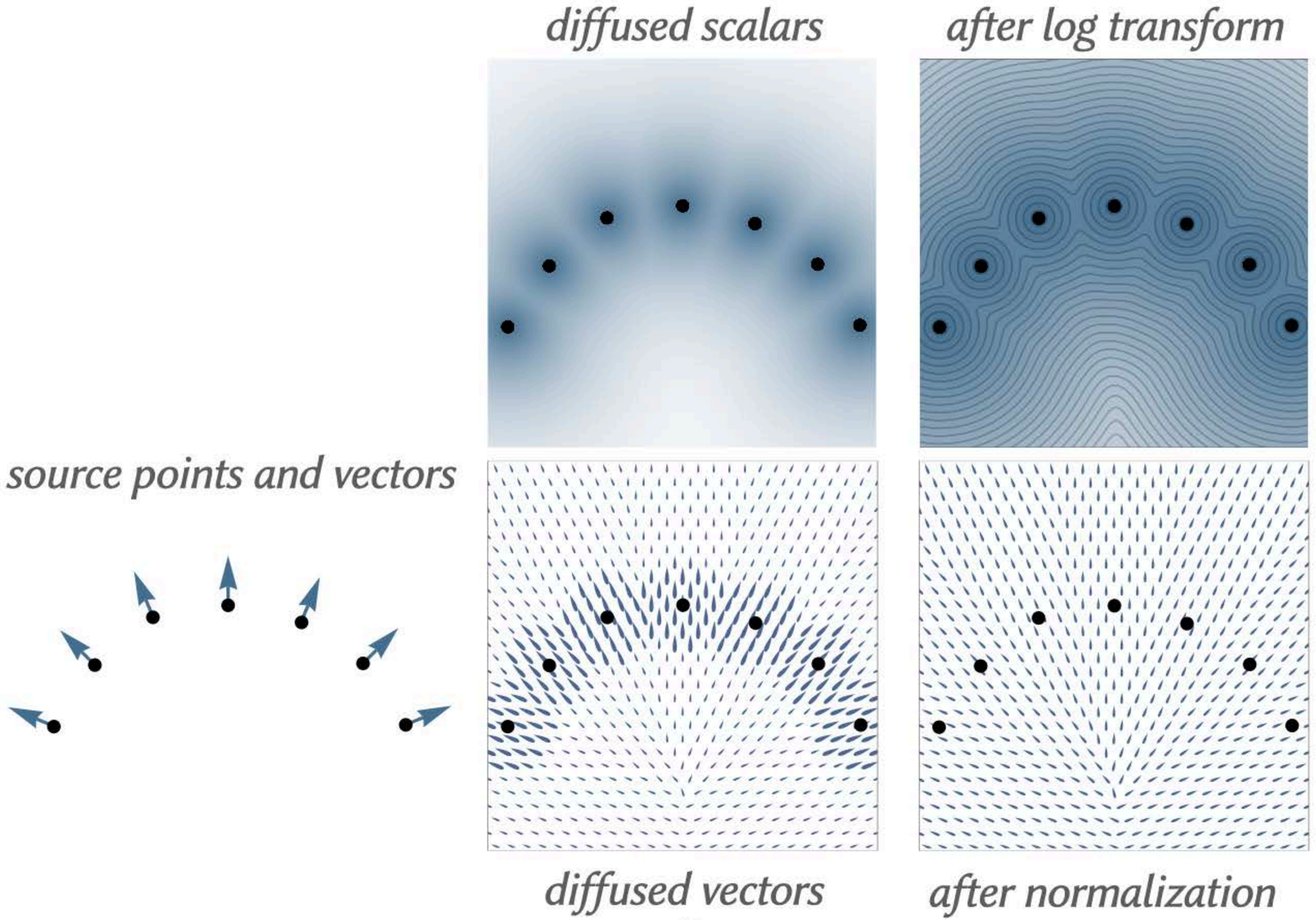
# Only magnitudes are concentrated



# Only magnitudes are concentrated



# Signed heat algorithm

## *Step 1:* vector diffusion

*vector heat equation*

$$\frac{\mathrm{d}}{\mathrm{d}t} X_t = \Delta^{\nabla} X_t$$

$$X_0 = N \mu_{\Omega}$$

## *Step 2:* normalization

$$Y_t \leftarrow X_t / \|X_t\|$$

## *Step 3:* integration

*Poisson equation*

$$\Delta \phi = \nabla \cdot Y_t \quad \text{on } M$$

$$\frac{\partial \phi}{\partial n} = n \cdot Y_t \quad \text{on } \partial M$$

# Discretization

## *Step 1:* vector diffusion

*vector heat equation*

$$\frac{\mathrm{d}}{\mathrm{d}t} X_t = \Delta^{\nabla} X_t$$

$$X_0 = N\mu_{\Omega}$$

↓

$$(\mathrm{id} - t\Delta^{\nabla}) X_t = X_0$$

*flow for* $t > 0$

↓

*mass matrix* — *connection Laplacian* — *initial normal vectors*

$$(\mathbf{M} + t\mathbf{L}^{\nabla})\mathbf{X} = \mathbf{X}_0$$

## *Step 2:* normalization

Normalize the diffused vector field **X**.

$$Y_t \leftarrow X_t / \|X_t\|$$

## *Step 3:* integration

*Poisson equation*

$$\Delta\phi = \nabla \cdot Y_t \quad \text{on } M$$

$$\frac{\partial\phi}{\partial n} = n \cdot Y_t \quad \text{on } \partial M$$

↓

*cotan Laplacian* — $\mathbf{C}\phi = \mathbf{b}$ — discrete divergence

# Discretization

## *Step 1:* vector diffusion

*vector heat equation*

$$\frac{\mathrm{d}}{\mathrm{d}t} X_t = \Delta^{\nabla} X_t$$

$$X_0 = N\mu_{\Omega}$$

$$(\mathrm{id} - t\Delta^{\nabla}) X_t = X_0$$

*flow for* $t > 0$

*mass matrix* — *connection Laplacian* — *initial normal vectors*

$$(\mathbf{M} + t\mathbf{L}^{\nabla})\mathbf{X} = \mathbf{X}_0$$

sparse linear system

## *Step 2:* normalization

Normalize the diffused vector field **X**.

$$Y_t \leftarrow X_t / \|X_t\|$$

## *Step 3:* integration

*Poisson equation*

$$\Delta\phi = \nabla \cdot Y_t \quad \text{on } M$$

$$\frac{\partial\phi}{\partial n} = n \cdot Y_t \quad \text{on } \partial M$$

*cotan Laplacian* — *discrete divergence*

$$\mathbf{C}\phi = \mathbf{b}$$

sparse linear system

# Discretization

## *Step 1:* vector diffusion

*vector heat equation*

$$\frac{\mathrm{d}}{\mathrm{d}t} X_t = \Delta^{\nabla} X_t$$

$$X_0 = N\mu_{\Omega}$$

$$(\mathrm{id} - t\Delta^{\nabla}) X_t = X_0$$

*flow for* $t > 0$

*mass matrix* — *connection Laplacian* — *initial normal vectors*

$$(\mathbf{M} + t\mathbf{L}^{\nabla})\mathbf{X} = \mathbf{X}_0$$

sparse linear system

## *Step 2:* normalization

Normalize the diffused vector field **X**.

$$Y_t \leftarrow X_t / \|X_t\|$$

## *Step 3:* integration

*Poisson equation*

$$\Delta\phi = \nabla \cdot Y_t \quad \text{on } M$$

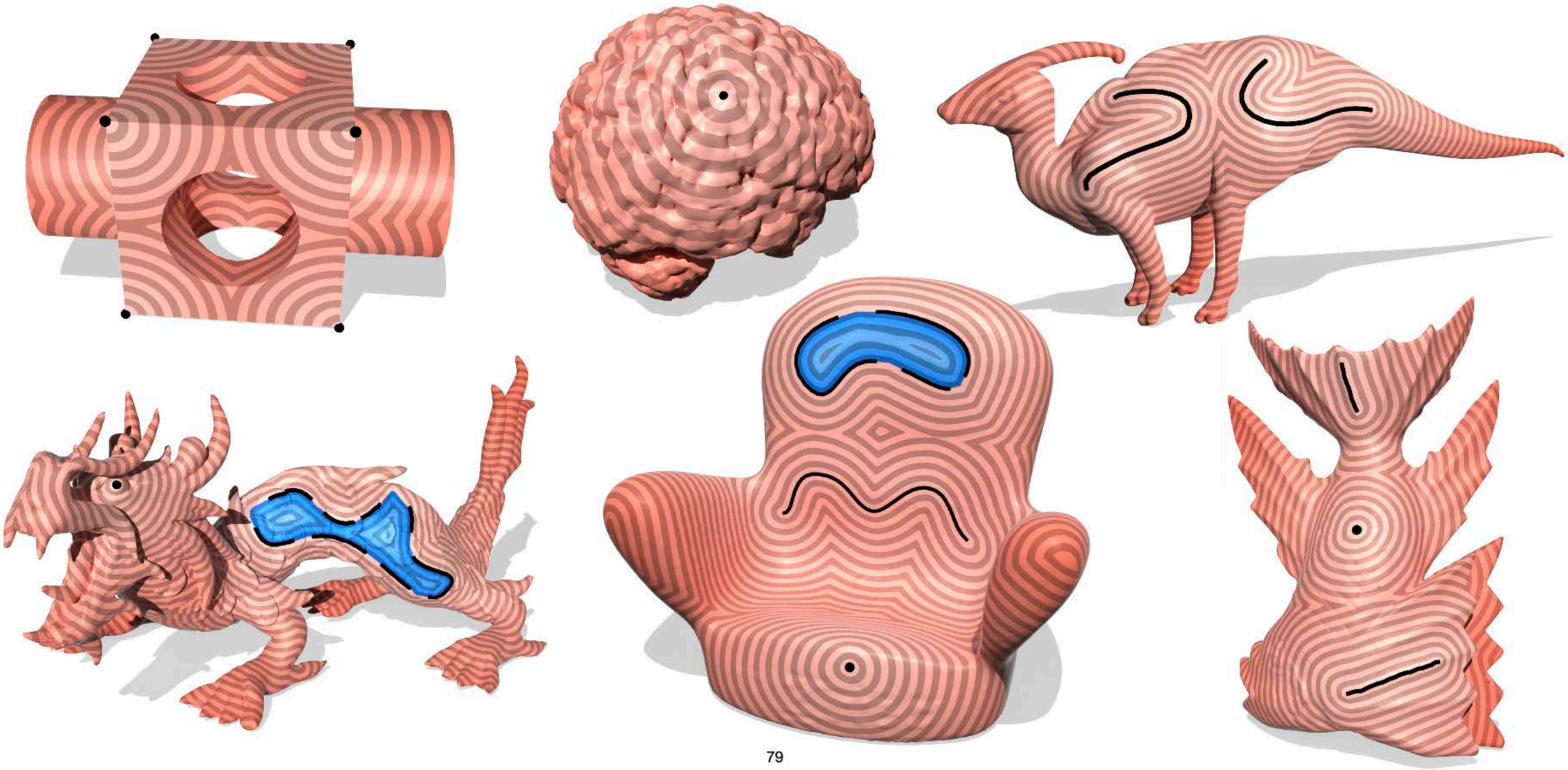
$$\frac{\partial\phi}{\partial n} = n \cdot Y_t \quad \text{on } \partial M$$

*cotan Laplacian* — *discrete divergence*

$$\mathbf{C}\phi = \mathbf{b}$$

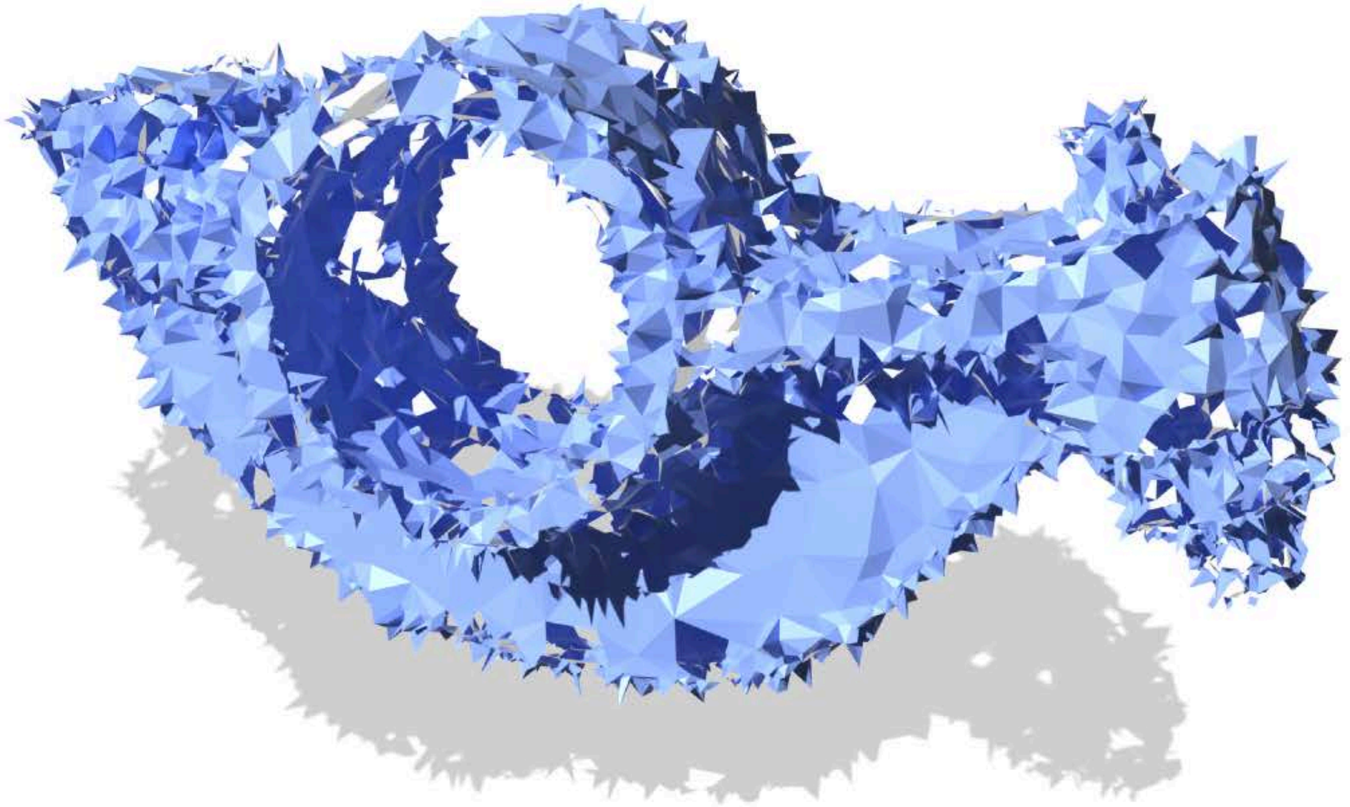
sparse linear system

# Can mix signed & unsigned distance

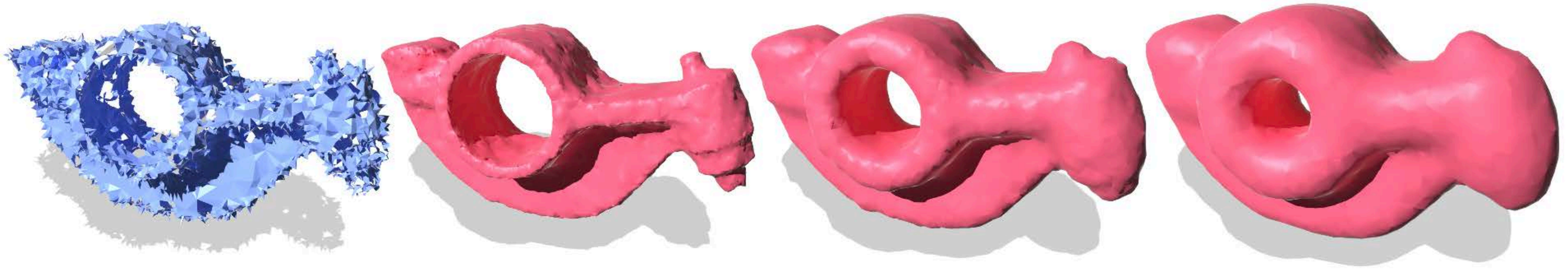


# Offset surfaces in 3D

input

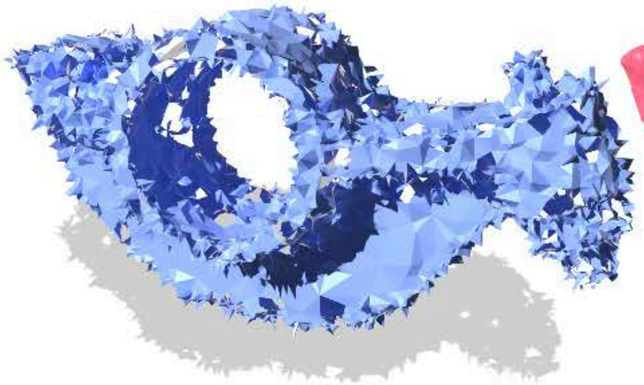


# Offset surfaces in 3D

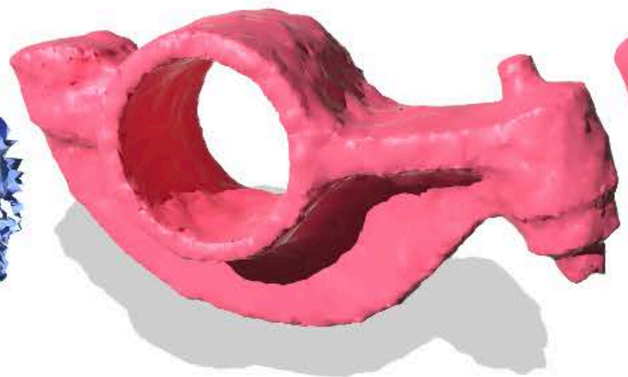


# Offset surfaces in 3D

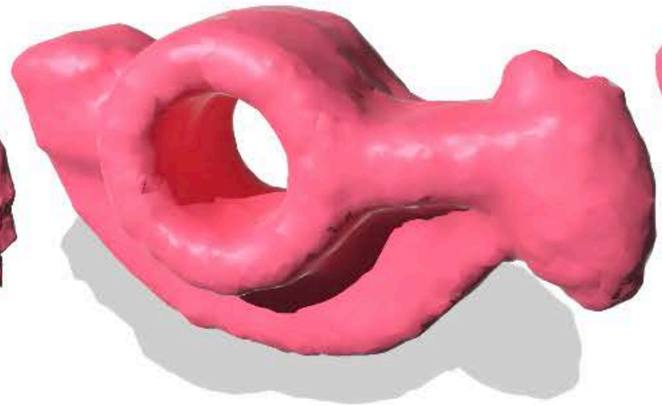
input



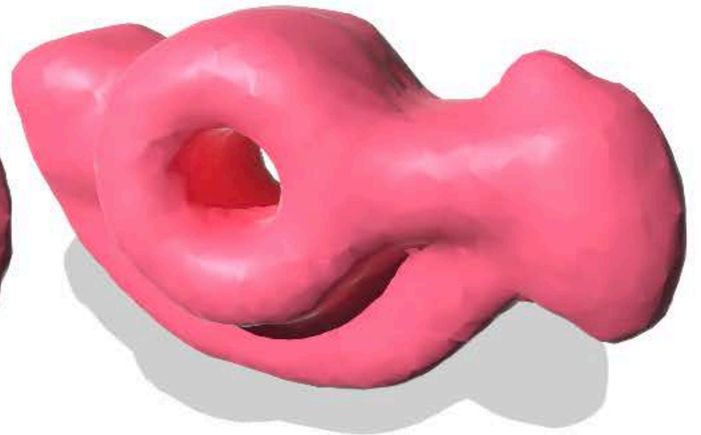
0-offset



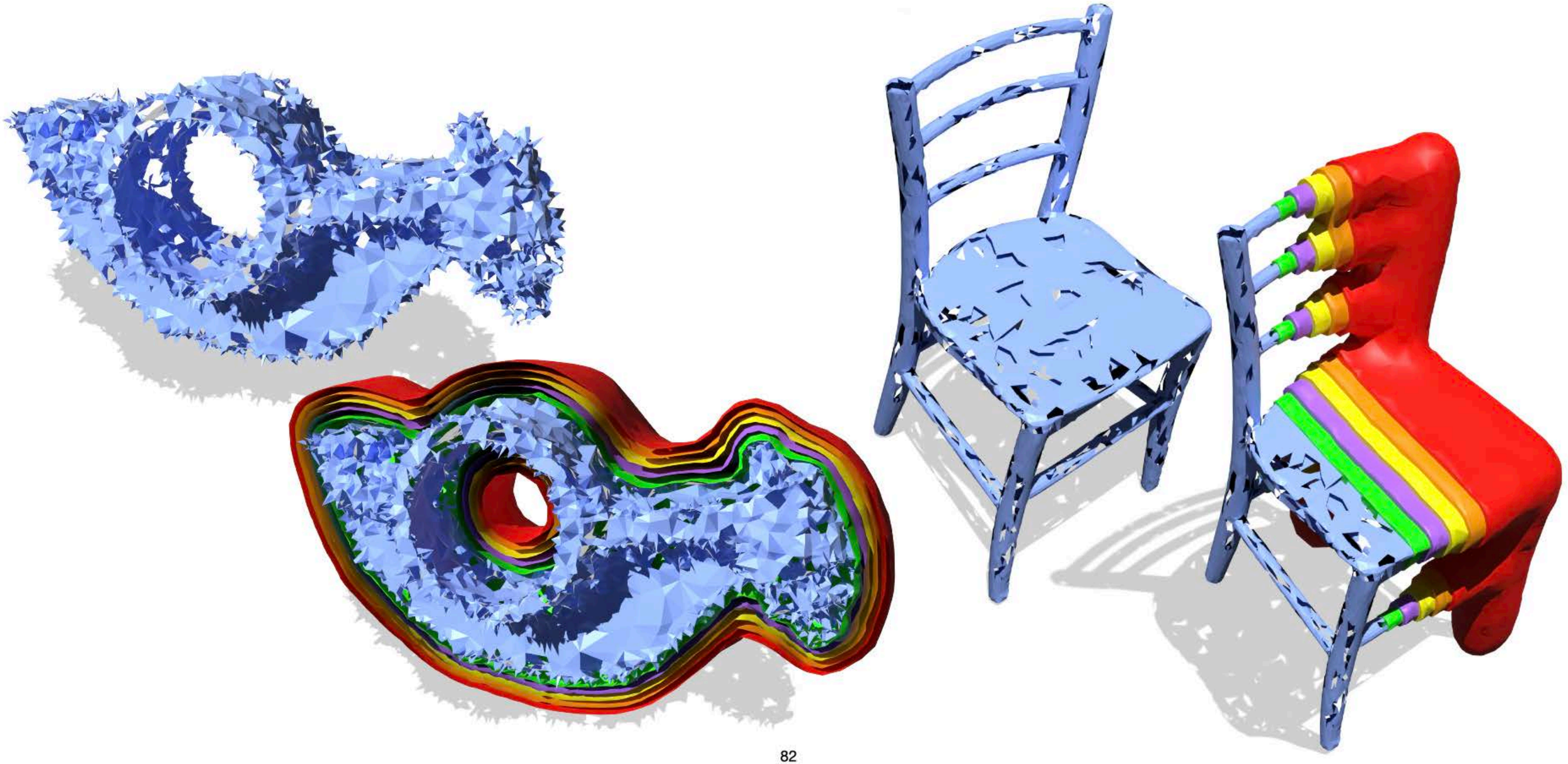
+0.2-offset



+0.4-offset

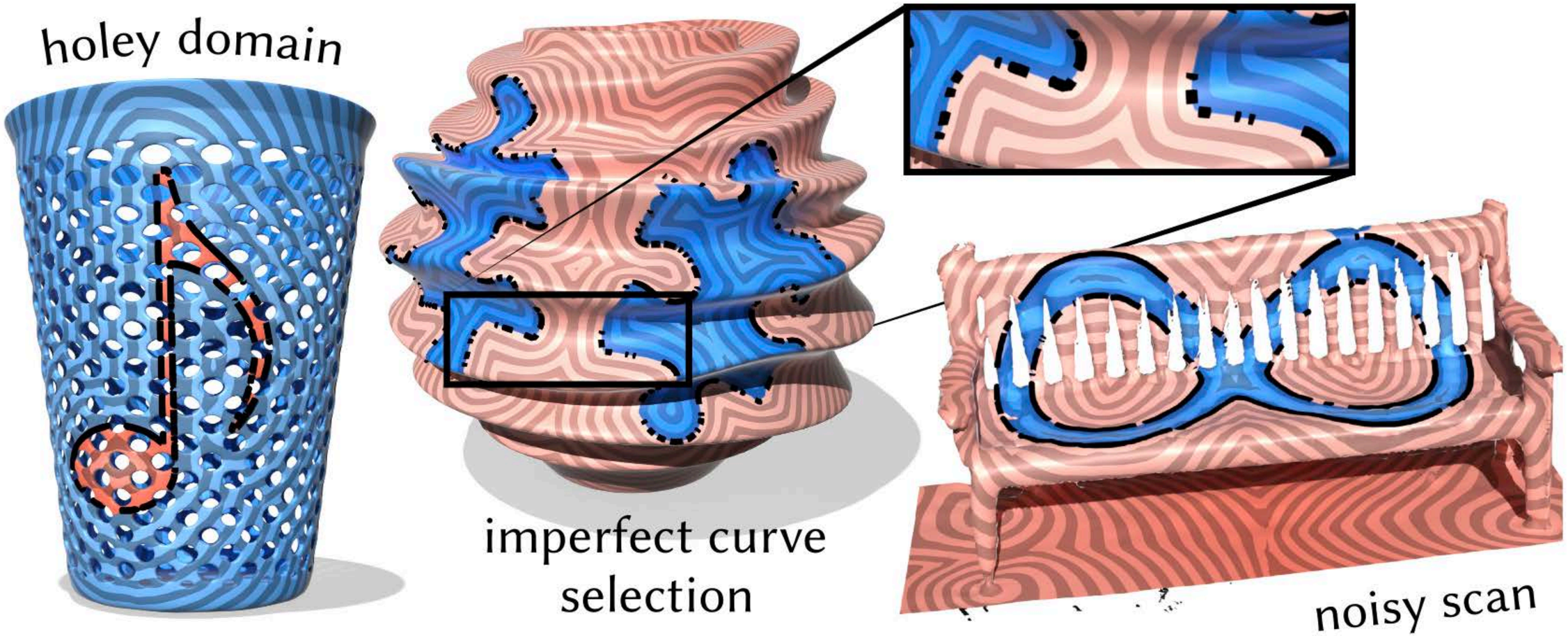


# Offset surfaces in 3D



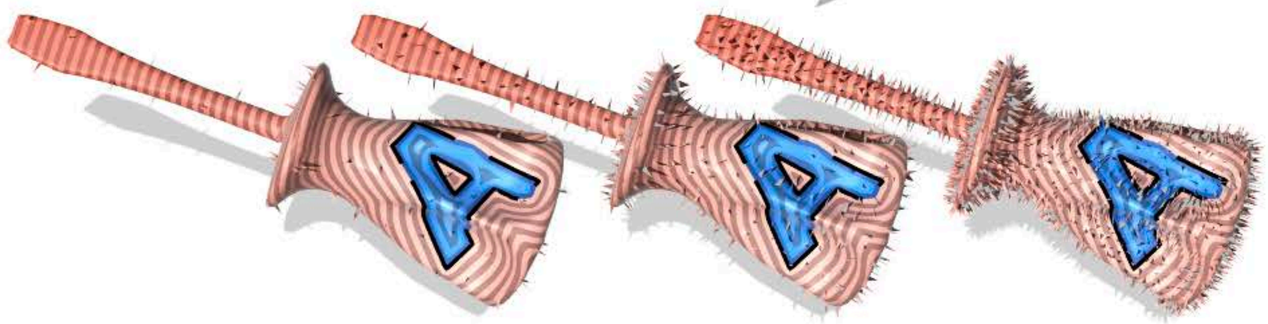
# **Robustness** to defects in the source geometry (curves)

# **Robustness** to defects in the source geometry (curves)

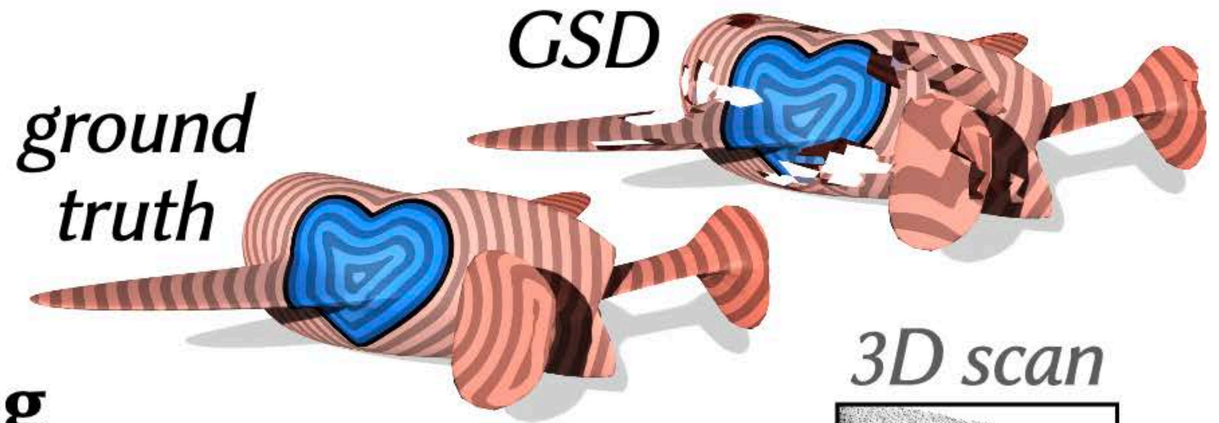


# **Robustness** to errors in the domain geometry (surface)

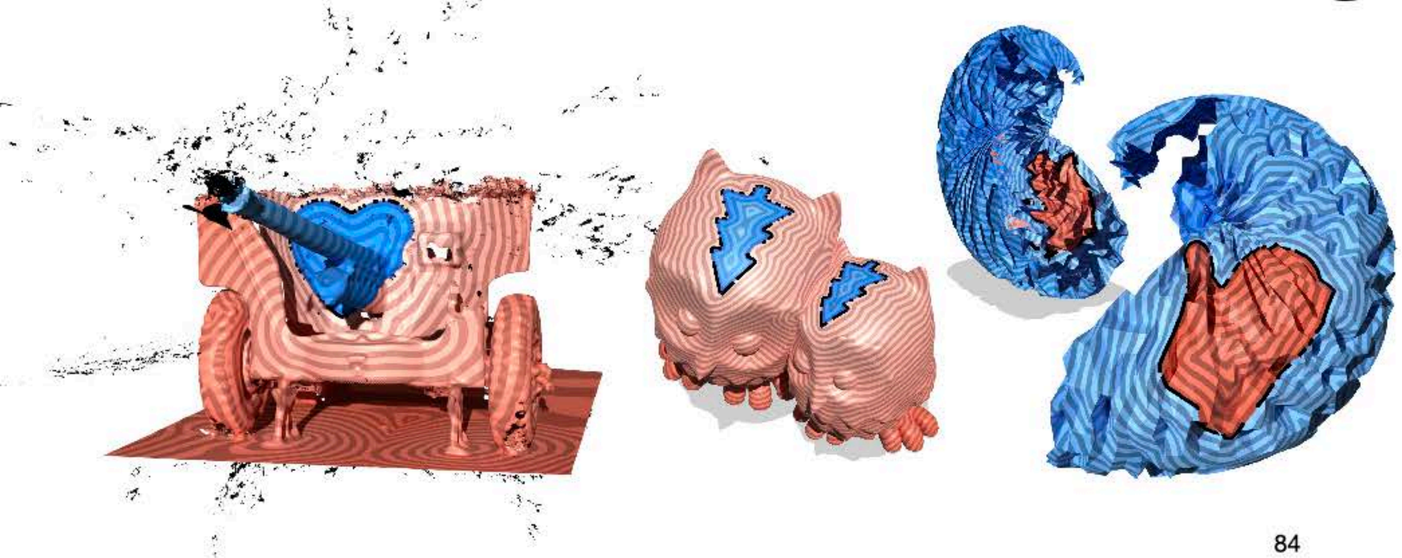
**more non-manifold**

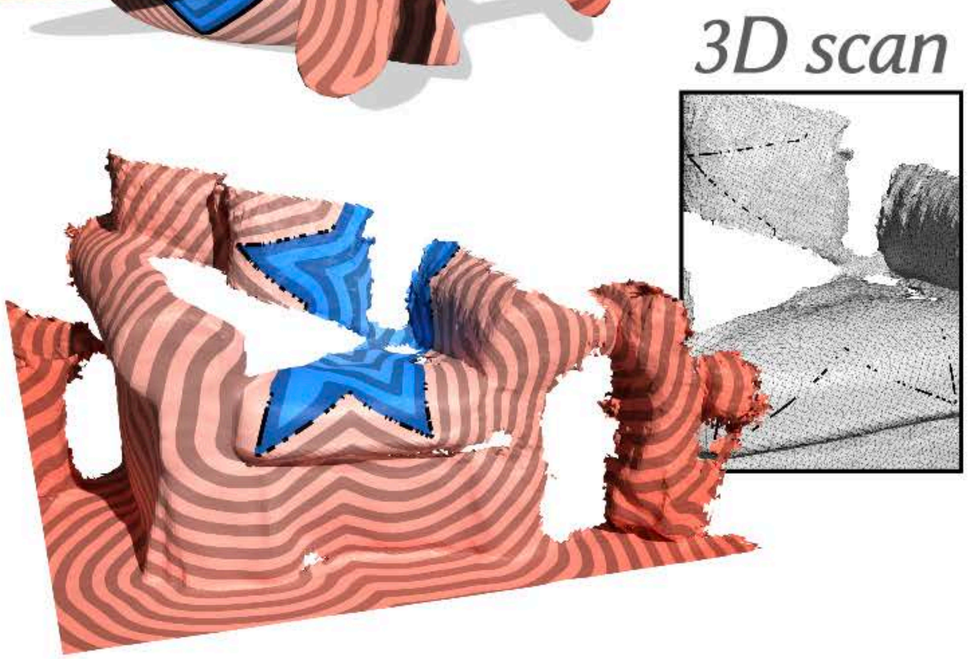


**large holes**




**non-manifold & self-intersecting**

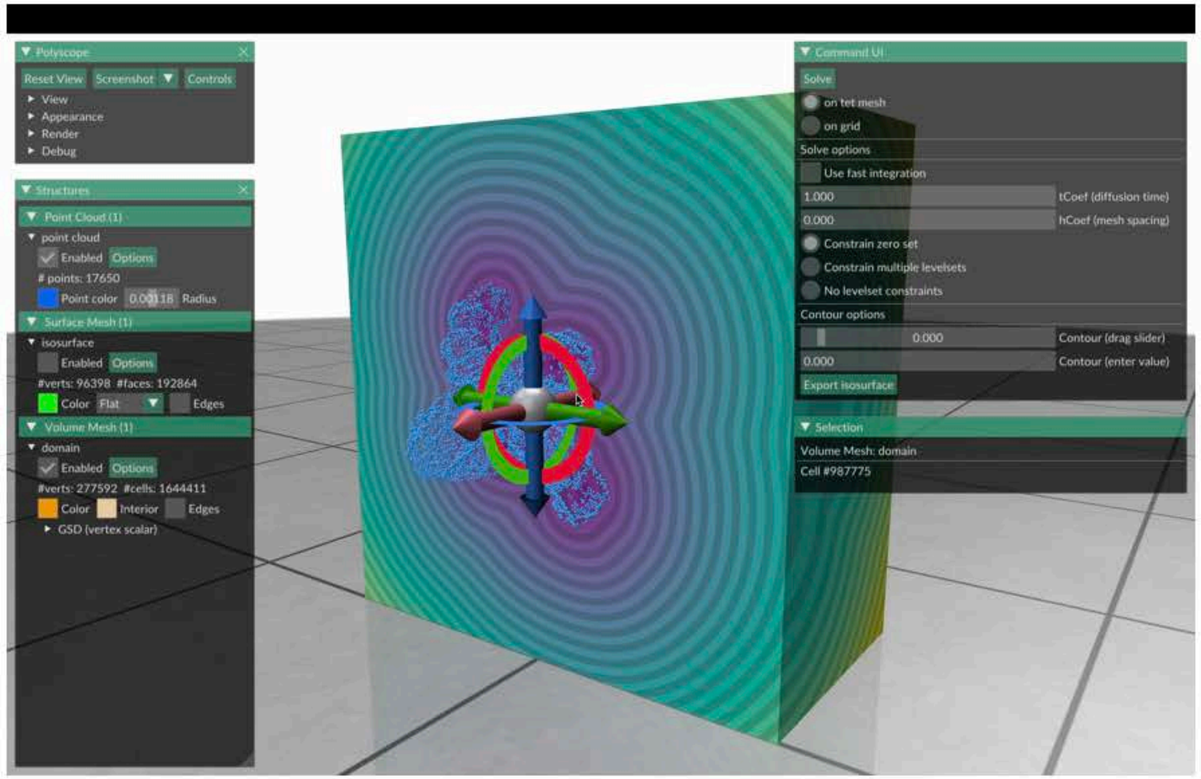




# *GENERALIZED SIGNED DISTANCE, BUT EVEN FASTER*

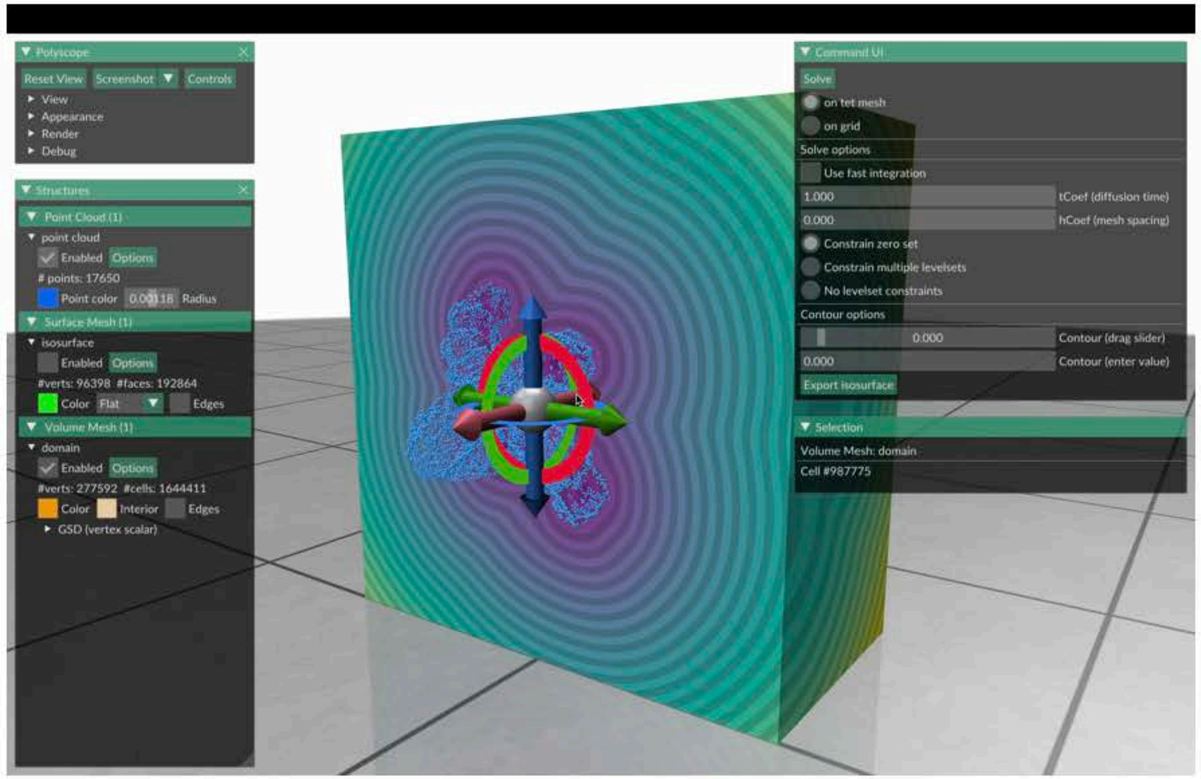
# Drawbacks

# Drawbacks


Polyscope
Reset View
Screenshot
Controls
View
Appearance
Render
Debug
Structures
Point Cloud (1)
point cloud
Enabled
Options
# points: 17650
Point color
0.0118
Radius
Surface Mesh (1)
isosurface
Enabled
Options
#verts: 96398 #faces: 192864
Color
Flat
Edges
Volume Mesh (1)
domain
Enabled
Options
#verts: 277592 #cells: 1644411
Color
Interior
Edges
GSD (vertex scalar)
Command UI
Solve
on tet mesh
on grid
Solve options
Use fast integration
1.000
tCoef (diffusion time)
0.000
hCoef (mesh spacing)
Constrain zero set
Constrain multiple levelsets
No levelset constraints
Contour options
0.000
Contour (drag slider)
0.000
Contour (enter value)
Export isosurface
Selection
Volume Mesh: domain
Cell #987775

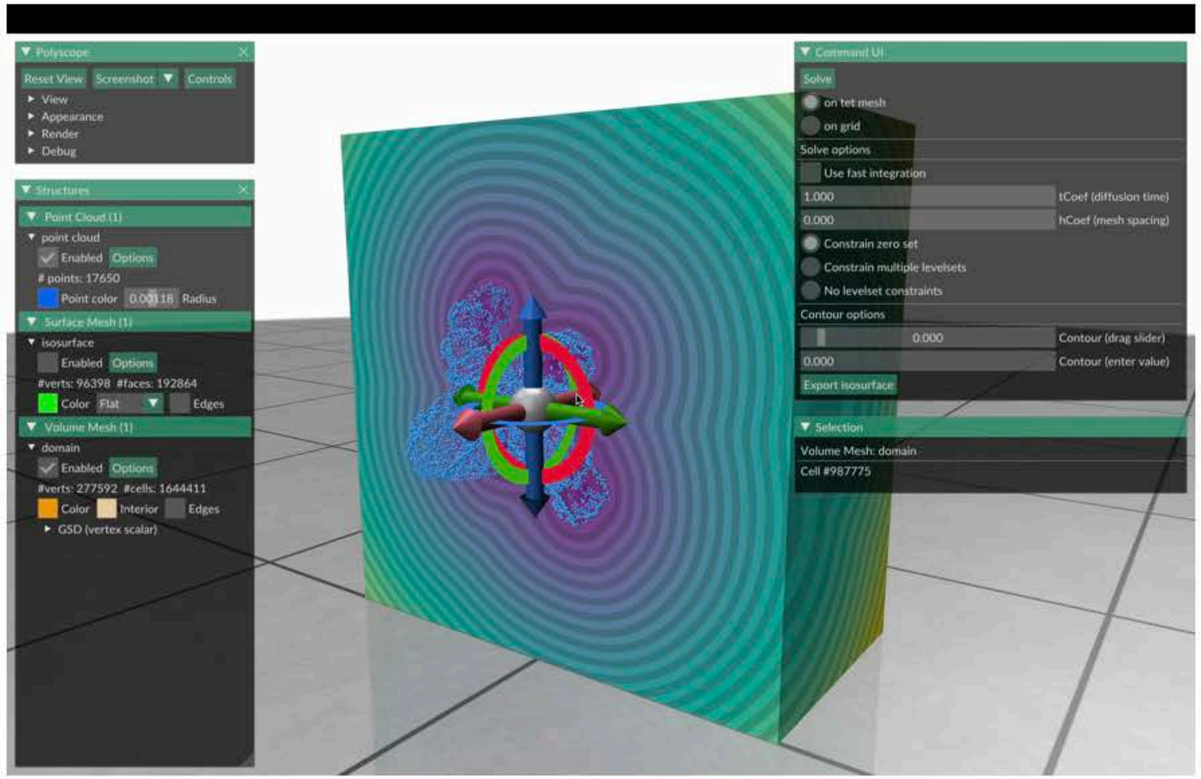
# Drawbacks

- Must discretize space → at the mercy of mesher, linear solver



# Drawbacks

- Must discretize space → at the mercy of mesher, linear solver
- Must solve global system → can't easily query at particular points



# In $\mathbb{R}^n$, we might try an integral formulation...

# In $\mathbb{R}^n$, we might try an integral formulation...

## PDE formulation

**Step 1:**

*Solve for* $X_t$*:*

$$(\mathrm{id} - t\Delta)X_t = X_0$$

**Step 2:**

*Normalize* $X_t$*:*

$$Y_t := X_t / \|X_t\|$$

**Step 3:**

*Solve for* $\phi$*:*

$$\Delta\phi = \nabla \cdot Y_t \quad \text{on } \mathbb{R}^n$$

# In $\mathbb{R}^n$, we might try an integral formulation...

| | PDE formulation | integral formulation |
|---|---|---|
| **Step 1:** | *Solve for* $X_t$*:* $(\mathrm{id} - t\Delta)X_t = X_0$ | *Solve for* $X_t$*:* $X_t(x) = \int_\Omega G^t(x,z)X_0(z)\,\mathrm{d}z$ |
| **Step 2:** | *Normalize* $X_t$*:* $Y_t := X_t / \|\|X_t\|\|$ | *Normalize* $X_t$*:* $Y_t := X_t / \|\|X_t\|\|$ |
| **Step 3:** | *Solve for* $\phi$*:* $\Delta\phi = \nabla \cdot Y_t \quad \text{on } \mathbb{R}^n$ | *Solve for* $\phi$*:* $\phi(x) = \int_{\mathbb{R}^n} G(x,y)(\nabla \cdot Y_t)(y)\,\mathrm{d}y$ |

# … but the integral formulation isn't well-posed.

*Solve for $\phi$:*

$$\phi(x) = \int_{\mathbb{R}^n} G(x,y)(\nabla \cdot Y_t)(y)\,\mathrm{d}y$$

*source*

*Green's function*

# … but the integral formulation isn't well-posed.

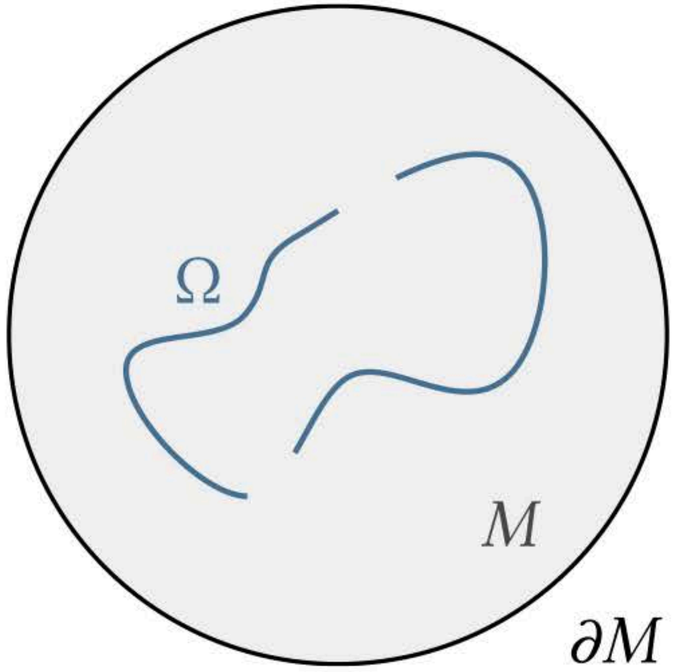
*Solve for $\phi$:*

$$\phi(x) = \int_{\mathbb{R}^n} \underbrace{G(x,y)}_{\text{Green's function}} \overbrace{(\nabla \cdot Y_t)(y)}^{\text{source}} \, \mathrm{d}y$$

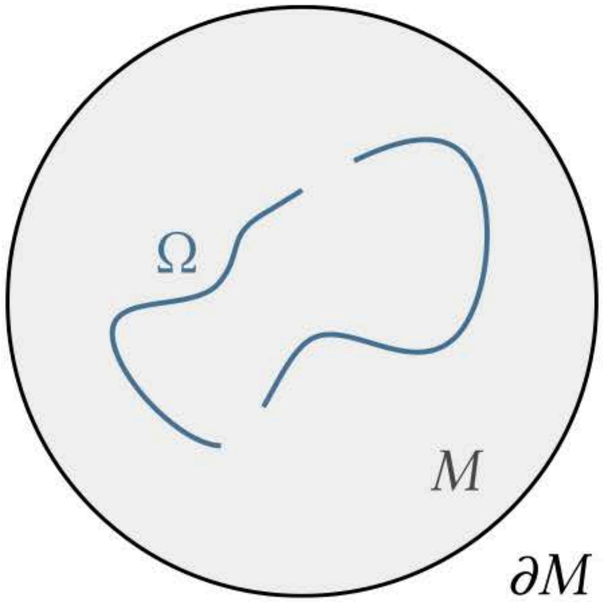
Not well-posed because $|\phi(x)| \nrightarrow 0$ as $\|x\| \longrightarrow \infty$.

# We could try a finite domain...

# We could try a finite domain...

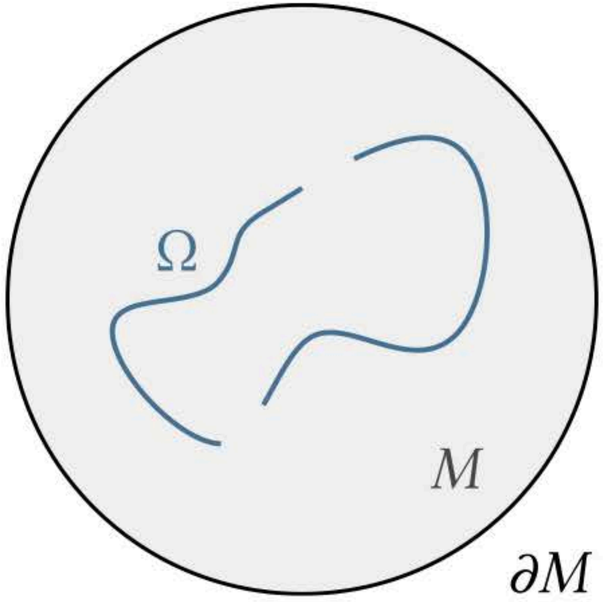


# … but the integral formulation is also not tractable.




$$\phi(x) = \int_{\partial M} \left[ \frac{\partial G(x,z)}{\partial n_z} \phi(z) - G(x,z) \frac{\partial \phi(z)}{\partial n_z} \right] \mathrm{d}z + \int_M G(x,y) \left( \nabla \cdot Y_t \right)(y) \, \mathrm{d}y$$

# … but the integral formulation is also not tractable.




$$\phi(x) = \int_{\partial M} \left[ \frac{\partial G(x,z)}{\partial n_z} \phi(z) - G(x,z) \frac{\partial \phi(z)}{\partial n_z} \right] \mathrm{d}z + \int_M G(x,y) \left( \nabla \cdot Y_t \right)(y) \, \mathrm{d}y$$

# Revisiting convolutional distance formulas

General convolutional distance formula:

$$\widetilde{d}(x) := -\frac{1}{\lambda} \log \left[ \int_{\Omega} h_x(z) \exp\left( -\lambda \varphi_x(z) \right) \mathrm{d}z \right]$$

where $\varphi_x(z) = \|x - z\|$.

# Revisiting convolutional distance formulas

General convolutional distance formula:

$$\widetilde{d}(x) := -\frac{1}{\lambda} \log \left[ \int_{\Omega} h_x(z) \exp\left( -\lambda \varphi_x(z) \right) \mathrm{d}z \right]$$

No good for generalized signed distance

where $\varphi_x(z) = \|x - z\|$.

# Revisiting convolutional distance formulas

General convolutional distance formula:

$$\widetilde{d}(x) := -\frac{1}{\lambda} \log \left[ \int_{\Omega} h_x(z) \exp\left( -\lambda \varphi_x \right. \right.$$

where $\varphi_x(z) = \|x - z\|$.

No good for generalized signed distance

But they're fast...

# Revisiting convolutional distance formulas

General convolutional distance formula:

$$\widetilde{d}(x) := -\frac{1}{\lambda} \log \left[ \int_\Omega h_x(z) \exp\left( -\lambda \varphi_x \right. \right.$$

where $\varphi_x(z) = \|x - z\|$.

No good for generalized signed distance

But they're fast...

*apply Laplace's method or take gradient*

"Self-normalized" convolutional distance:

*a.k.a. kernel density estimator*

$$\widehat{d}(x) = \frac{\int_\Omega g_x(z) \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}{\int_\Omega \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}$$

# Revisiting convolutional distance formulas

General convolutional distance formula:

$$\widetilde{d}(x) := -\frac{1}{\lambda} \log \left[ \int_\Omega h_x(z) \exp\left( -\lambda \varphi_x \right. \right.$$

where $\varphi_x(z) = \|x - z\|$.

No good for generalized signed distance

But they're fast...

*apply Laplace's method or take gradient*

"Self-normalized" convolutional distance:

*a.k.a. kernel density estimator*

*closest-point interpolation*

$$\widehat{d}(x) = \frac{\int_\Omega g_x(z) \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}{\int_\Omega \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}$$

# We've already seen self-normalized convolutional distance

"Self-normalized" convolutional distance:

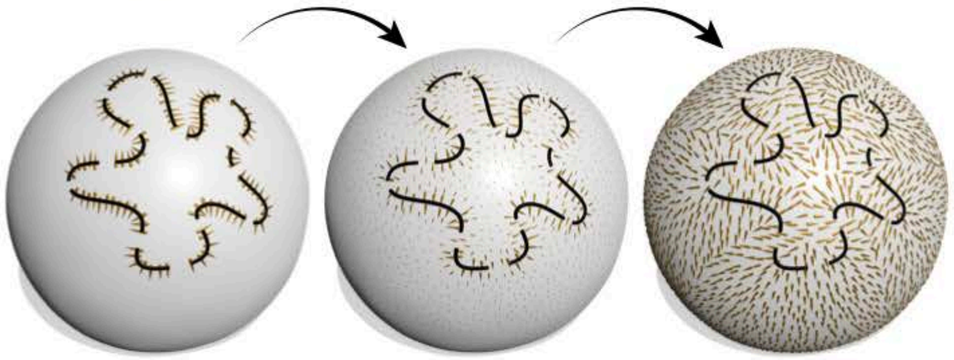
*a.k.a. kernel density estimator*

*closest-point interpolation*

$$\widehat{d}(x) = \frac{\int_\Omega g_x(z) \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}{\int_\Omega \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}$$



*Signed Heat Method* for signed distance



# We've already seen self-normalized convolutional distance

"Self-normalized" convolutional distance:

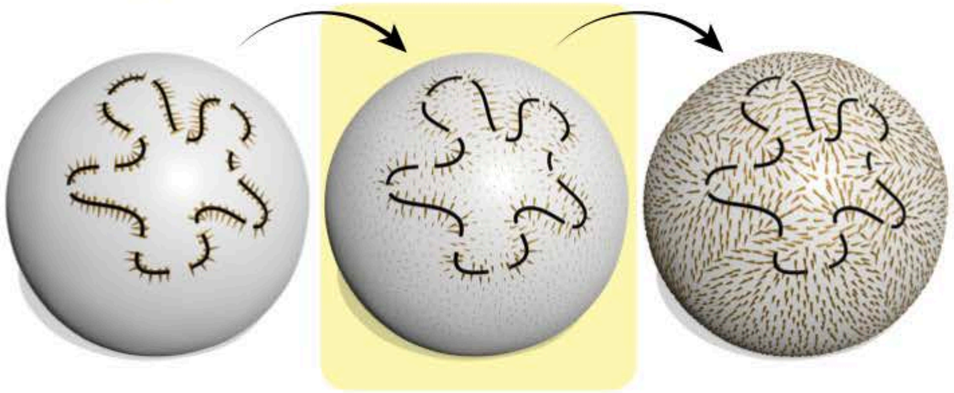
*a.k.a. kernel density estimator*

*closest-point interpolation*

$$\widehat{d}(x) = \frac{\int_\Omega g_x(z) \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}{\int_\Omega \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}$$



*Signed Heat Method* for signed distance



# We've already seen self-normalized convolutional distance

"Self-normalized" convolutional distance:

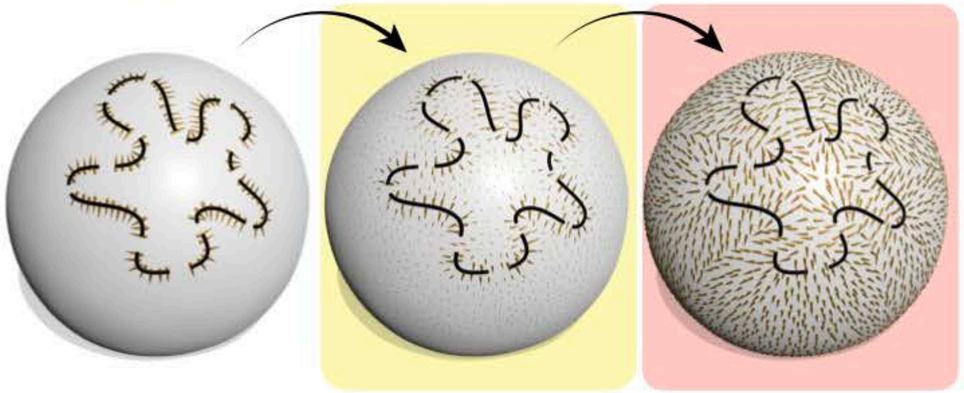
*a.k.a. kernel density estimator*

*closest-point interpolation*

$$\widehat{d}(x) = \frac{\int_\Omega g_x(z) \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}{\int_\Omega \exp\left(-\lambda \|x - z\|\right) \, \mathrm{d}z}$$



*Signed Heat Method* for signed distance



# We've already seen self-normalized convolutional distance

"Self-normalized" convolutional distance:

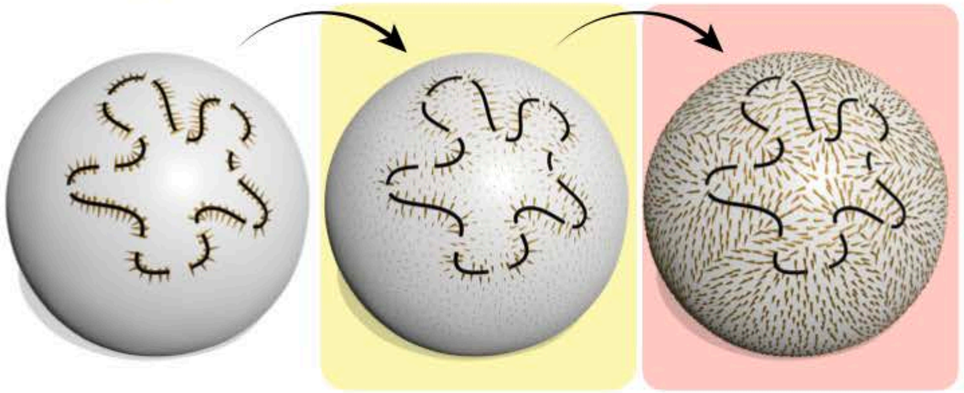
*a.k.a. kernel density estimator*

*closest-point interpolation*

$$\widehat{d}(x) = \frac{\int_\Omega g_x(z) \exp\left(-\lambda \|x - z\|\right)\, \mathrm{d}z}{\int_\Omega \exp\left(-\lambda \|x - z\|\right)\, \mathrm{d}z}$$



*Signed Heat Method* for signed distance





*Vector Heat Method* for parallel transport
[Sharp et al. 2019]

**ALGORITHM 1:** Vector Heat Method

**Input:** A vector field $X$ supported on a subset $\Omega \subset M$ of the domain $M$.

**Output:** A vector field $\overline{X}$ on all of $M$.

I. Integrate the vector heat flow $\frac{d}{dt} Y_t = \Delta^\nabla Y_t$ for time $t$, with $Y_0 = X$.

II. Integrate the scalar heat flow $\frac{d}{dt} u_t = \Delta u_t$ for time $t$, with $u_0 = |X|$.

III. Integrate the scalar heat flow $\frac{d}{dt} \phi_t = \Delta \phi_t$ for time $t$, with $\phi_0 = \mathbb{1}_\Omega$.

IV. Evaluate the vector field $\overline{X}_t = u_t Y_t / \phi_t |Y_t|$.

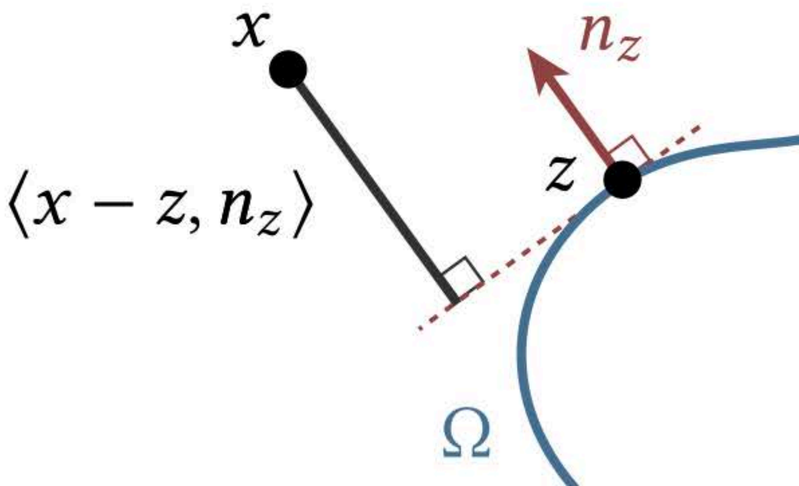
# Linear approximations are undesirable...

"Self-normalized" convolutional distance:

*a.k.a. kernel density estimator*

*"normal" distance*

$$\widehat{d}(x) = \frac{\int_\Omega \langle x - z, n_z \rangle \exp(-\lambda \|x - z\|)\, \mathrm{d}z}{\int_\Omega \exp(-\lambda \|x - z\|)\, \mathrm{d}z}$$



# Linear approximations are undesirable...

"Self-normalized" convolutional distance:

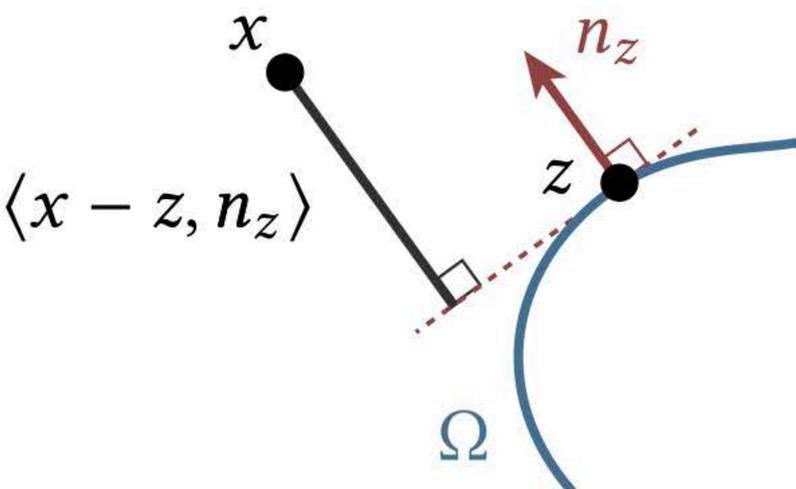
*a.k.a. kernel density estimator*

*"normal" distance*

$$\widehat{d}(x) = \frac{\int_\Omega \langle x - z, n_z \rangle \exp(-\lambda \|x - z\|)\, \mathrm{d}z}{\int_\Omega \exp(-\lambda \|x - z\|)\, \mathrm{d}z}$$

"pseudonormal test"
"tangent plane approximation"
"plane test"

[Alexa et al. 2001; Boissonnat & Cazals 2002; Bærentzen & Aanæs 2005, Kolluri 2008; Öztireli et al. 2009; Yang et al. 2025]



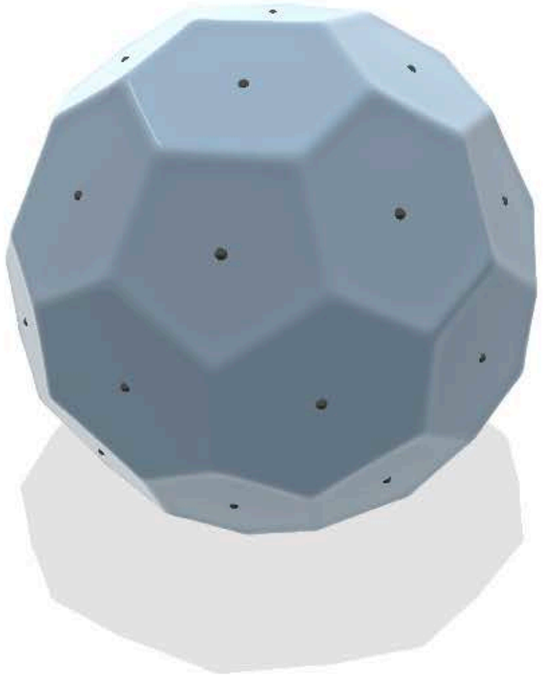
# Linear approximations are undesirable...

"Self-normalized" convolutional distance:

*a.k.a. kernel density estimator*

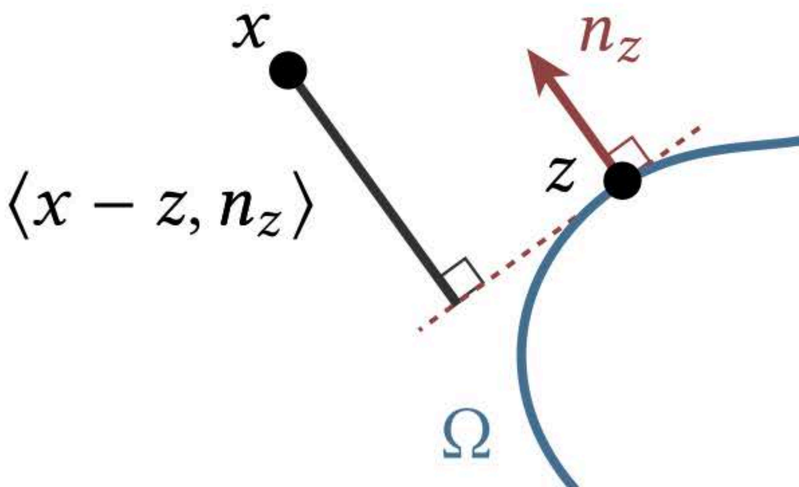
*"normal" distance*

$$\widehat{d}(x) = \frac{\int_\Omega \langle x - z, n_z \rangle \exp(-\lambda \|x - z\|)\, \mathrm{d}z}{\int_\Omega \exp(-\lambda \|x - z\|)\, \mathrm{d}z}$$



"pseudonormal test"
"tangent plane approximation"
"plane test"

[Alexa et al. 2001; Boissonnat & Cazals 2002; Bærentzen & Aanæs 2005, Kolluri 2008; Öztireli et al. 2009; Yang et al. 2025]



# … so let's use second-order surfaces!

"Self-normalized" convolutional distance:

*a.k.a. kernel density estimator*

$$\phi(x) = \frac{\sum_{i=1}^{|P|} g_i(x) \exp\left(-\lambda \|x - p_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|x - p_i\|\right)}$$

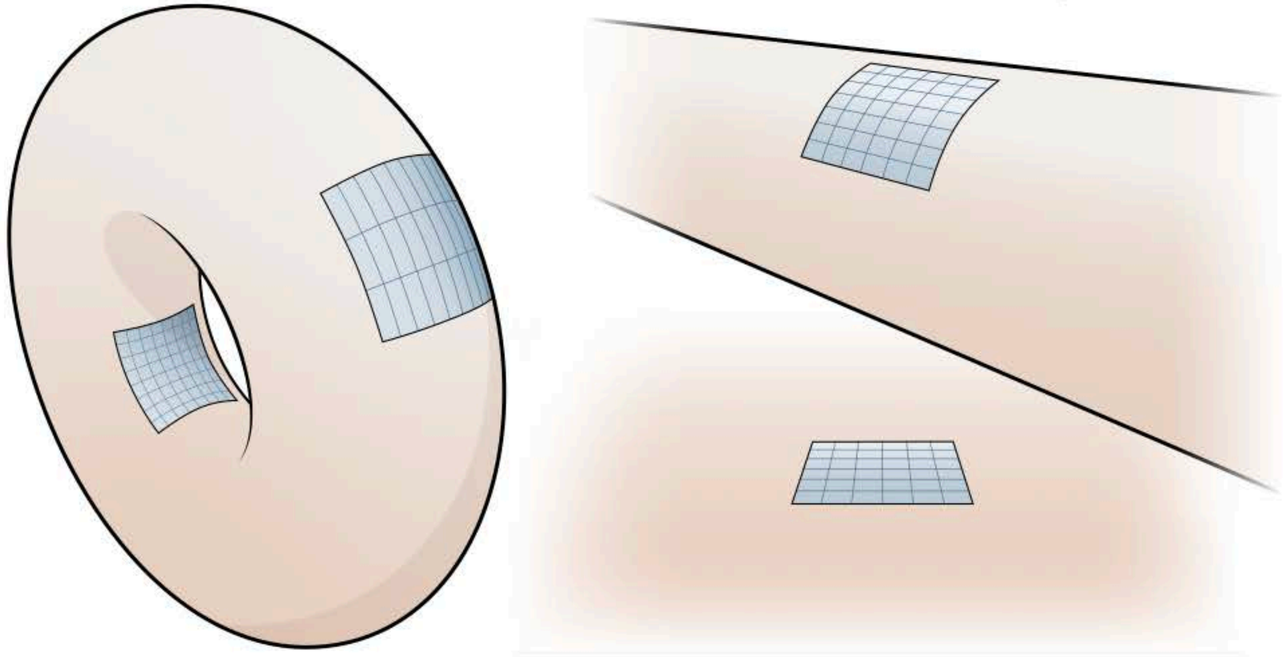
# … so let's use second-order surfaces!

"Self-normalized" convolutional distance:

*a.k.a. kernel density estimator*

*torus distance!*

$$\phi(x) = \frac{\sum_{i=1}^{|P|} g_i(x) \exp\left(-\lambda \|x - p_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|x - p_i\|\right)}$$



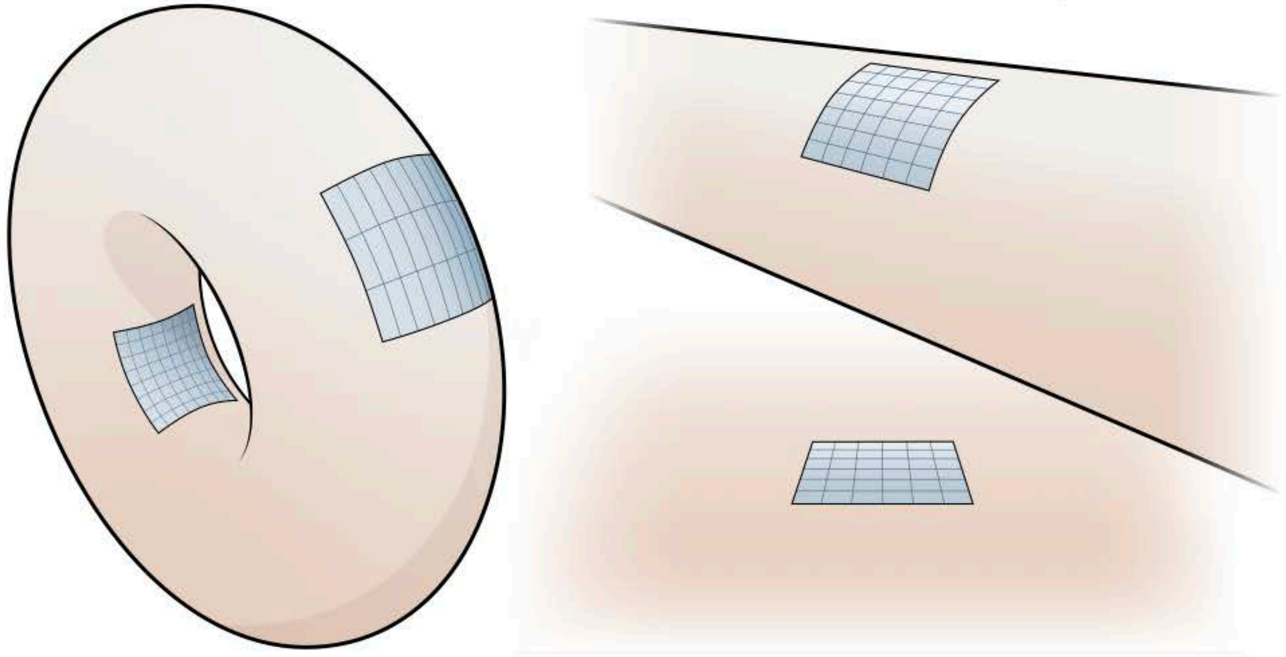
# … so let's use second-order surfaces!

"Self-normalized" convolutional distance:

*a.k.a. kernel density estimator*

*torus distance!*

$$\phi(x) = \frac{\sum_{i=1}^{|P|} g_i(x) \exp\left(-\lambda \|x - p_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|x - p_i\|\right)}$$



Fitting geometric proxies to point clouds (planes, cylinders, spheres, quadrics, splines, polynomial patches…)

[Faugeras 1983; Besl & Jain 1988; Cao et al. 1994; Kaveti et al. 1996; Yang & Lee 1999; Cohen-Steiner et al. 2004; Cazals & Pouget 2005; Wu & Kobbelt 2005; Simari & Singh 2005; Yan et al. 2006; Attene et al. 2006; Xia & Ju 2025…]

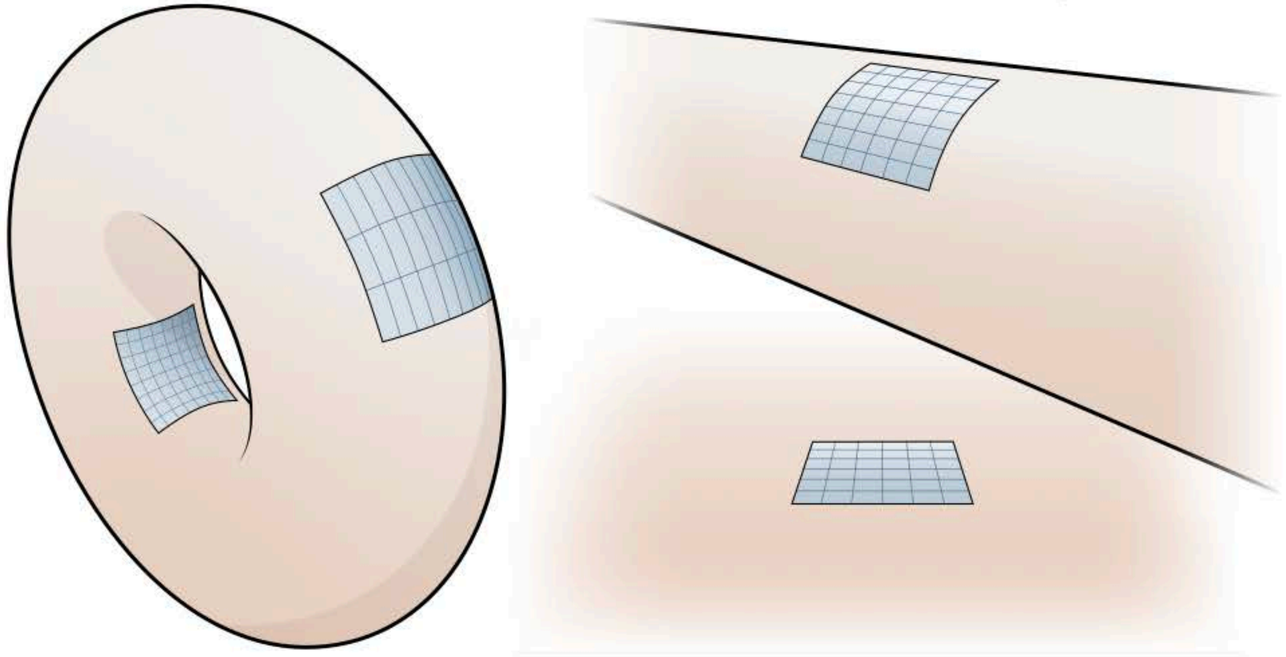
# … so let's use second-order surfaces!

"Self-normalized" convolutional distance:

*a.k.a. kernel density estimator*

*torus distance!*

$$\phi(x) = \frac{\sum_{i=1}^{|P|} g_i(x) \exp\left(-\lambda \|x - p_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|x - p_i\|\right)}$$



Fitting geometric proxies to point clouds (planes, cylinders, spheres, quadrics, splines, polynomial patches…)

[Faugeras 1983; Besl & Jain 1988; Cao et al. 1994; Kaveti et al. 1996; Yang & Lee 1999; Cohen-Steiner et al. 2004; Cazals & Pouget 2005; Wu & Kobbelt 2005; Simari & Singh 2005; Yan et al. 2006; Attene et al. 2006; Xia & Ju 2025…]

Tori have closed-form SDFs!

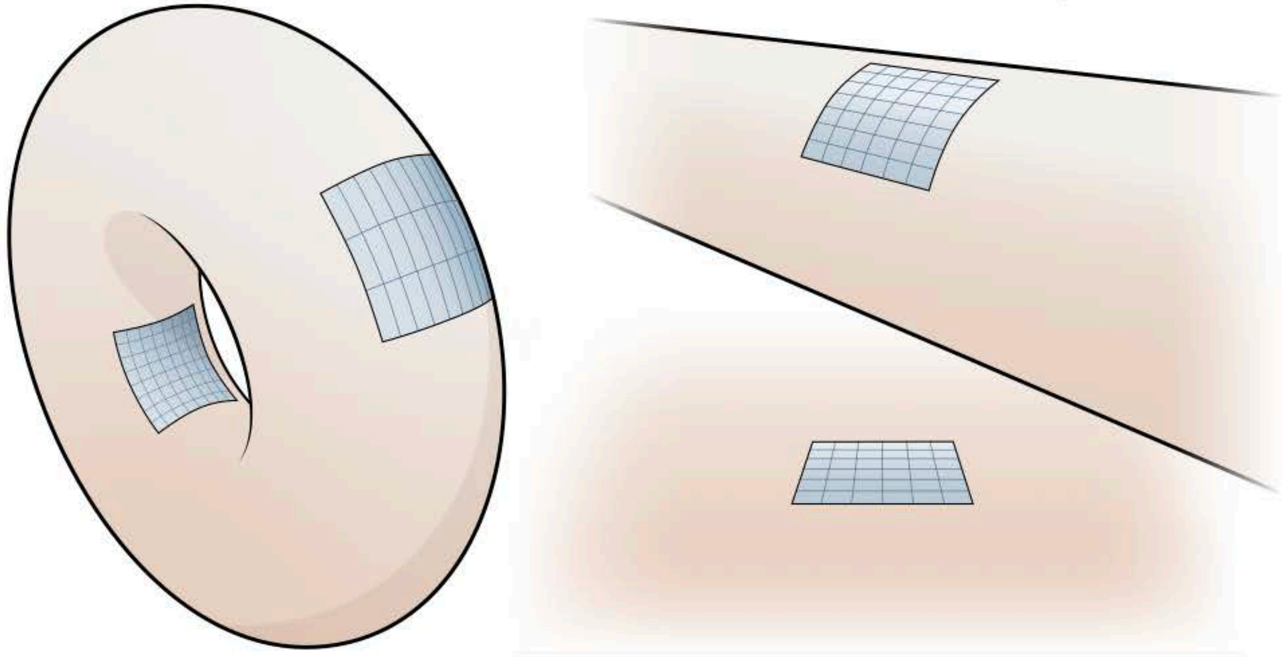
# … so let's use second-order surfaces!

"Self-normalized" convolutional distance:

*a.k.a. kernel density estimator*

*torus distance!*

$$\phi(x) = \frac{\sum_{i=1}^{|P|} g_i(x) \exp\left(-\lambda \|x - p_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|x - p_i\|\right)}$$



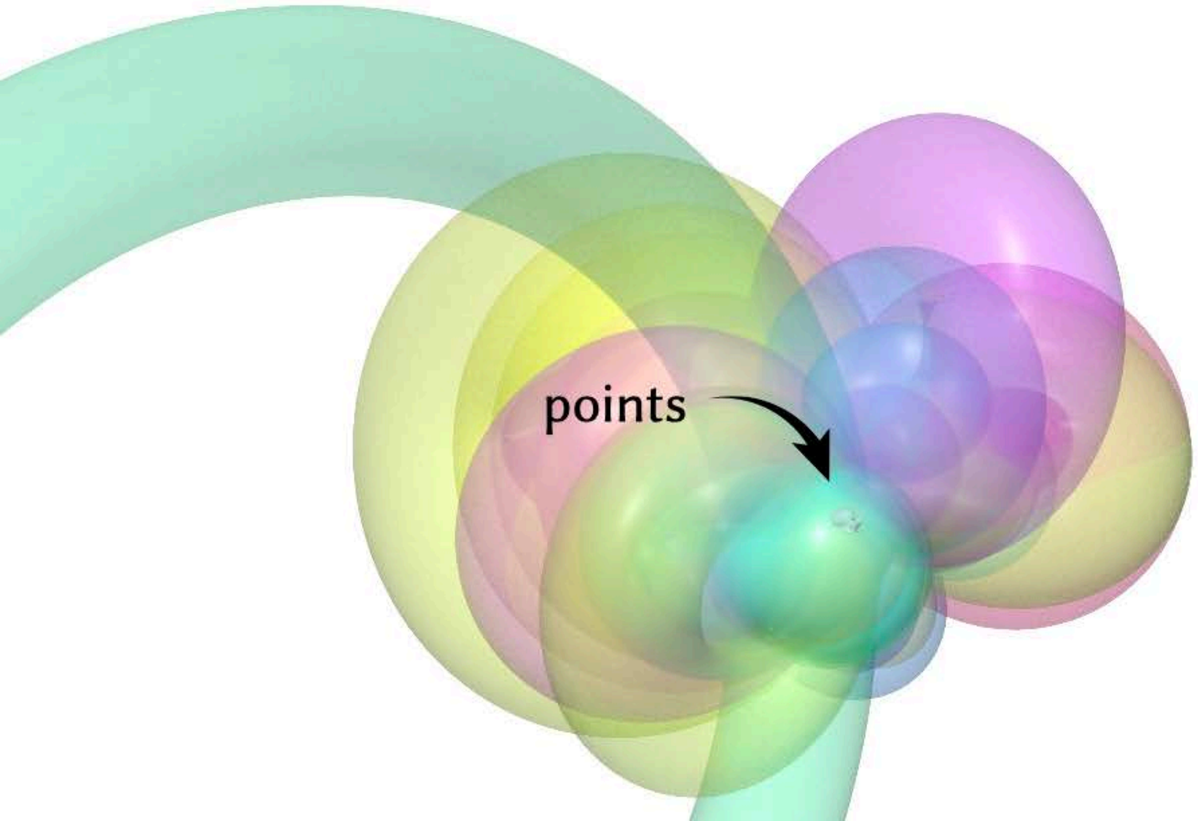
Fitting geometric proxies to point clouds (planes, cylinders, spheres, quadrics, splines, polynomial patches…)

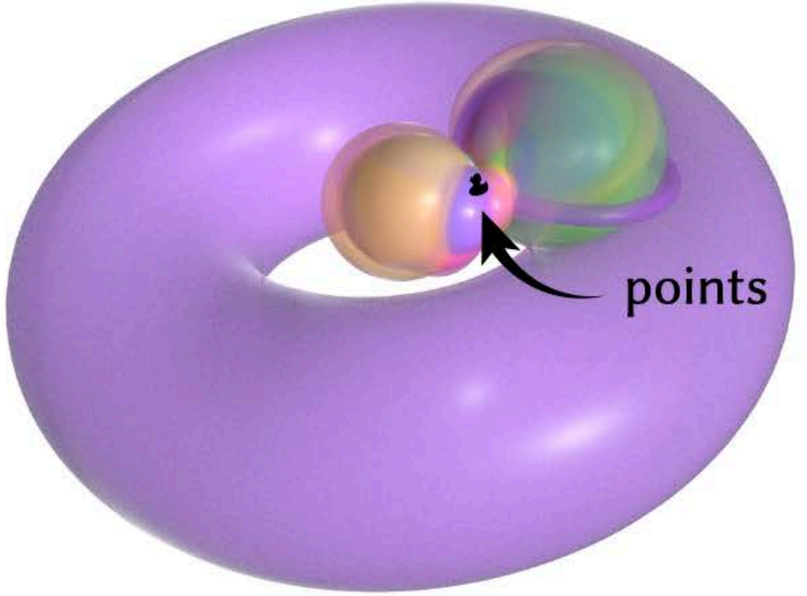
[Faugeras 1983; Besl & Jain 1988; Cao et al. 1994; Kaveti et al. 1996; Yang & Lee 1999; Cohen-Steiner et al. 2004; Cazals & Pouget 2005; Wu & Kobbelt 2005; Simari & Singh 2005; Yan et al. 2006; Attene et al. 2006; Xia & Ju 2025…]

Tori: [Lukács et al. 1998; Liu et al. 2009; Eberly 2020]

Tori have closed-form SDFs!

# The tori look crazy... but they work together

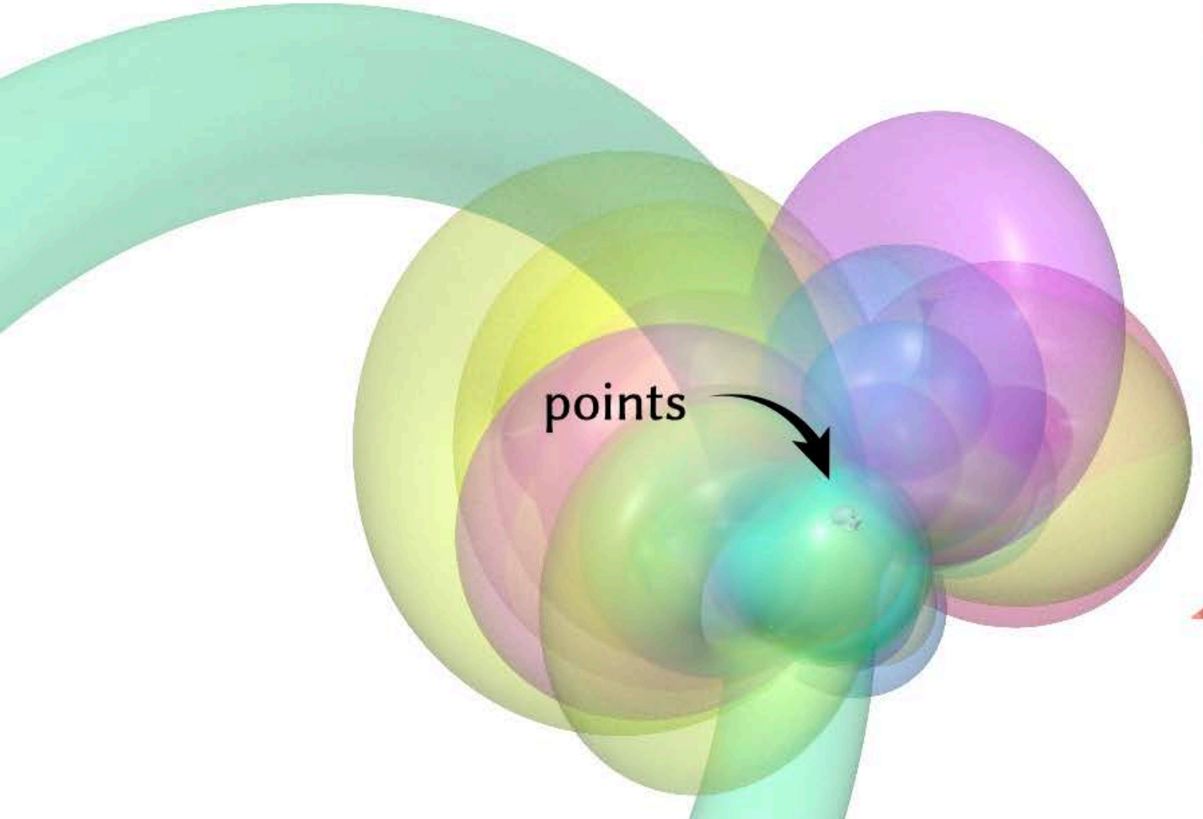


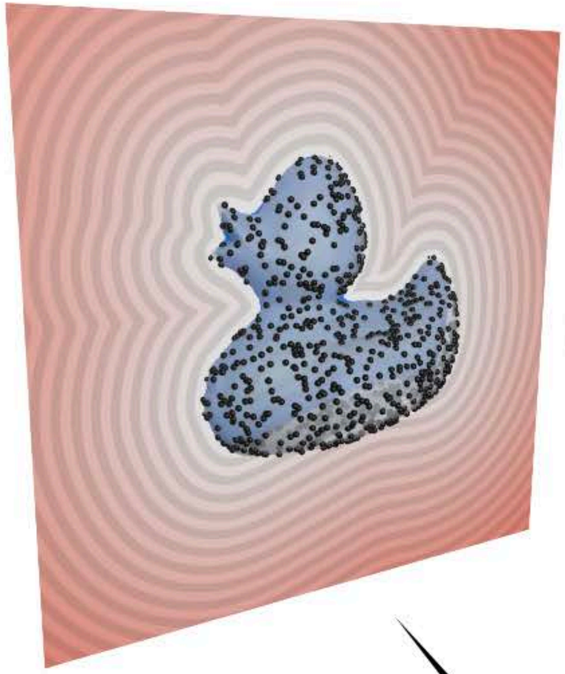



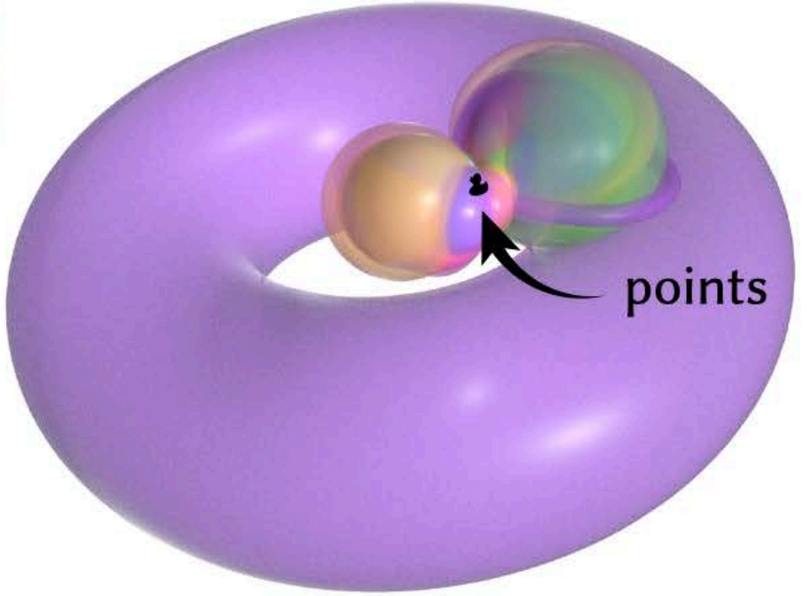
# The tori look crazy... but they work together

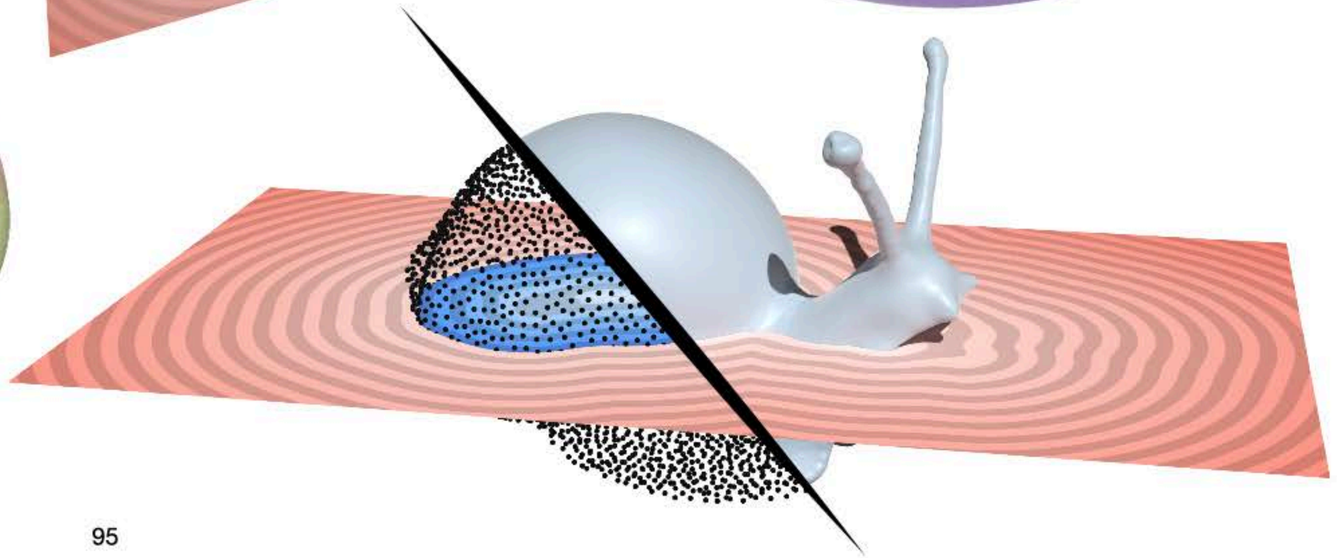
$$\phi(x) = \frac{\sum_{i=1}^{|P|} g_i(x) \exp\left(-\lambda \|x - p_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|x - p_i\|\right)}$$

*i-th torus SDF*





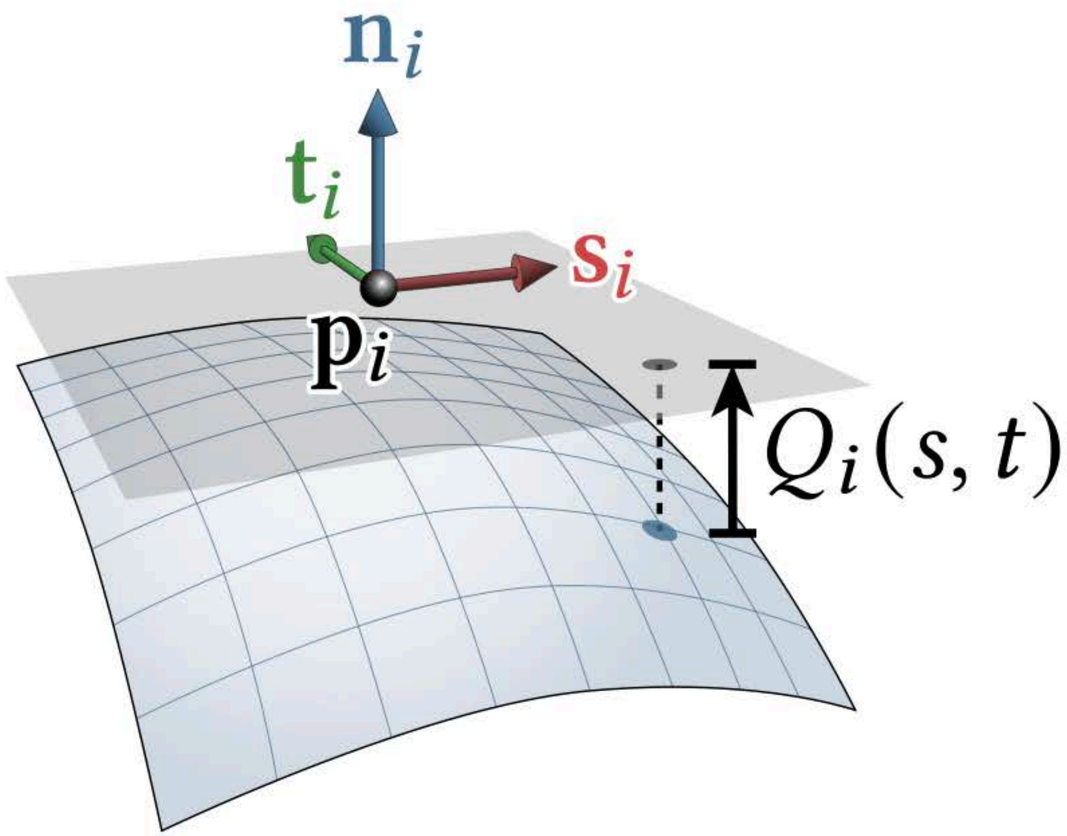



# The surface, one polynomial patch at a time

# The surface, one polynomial patch at a time

Quadratic polynomial around each point:

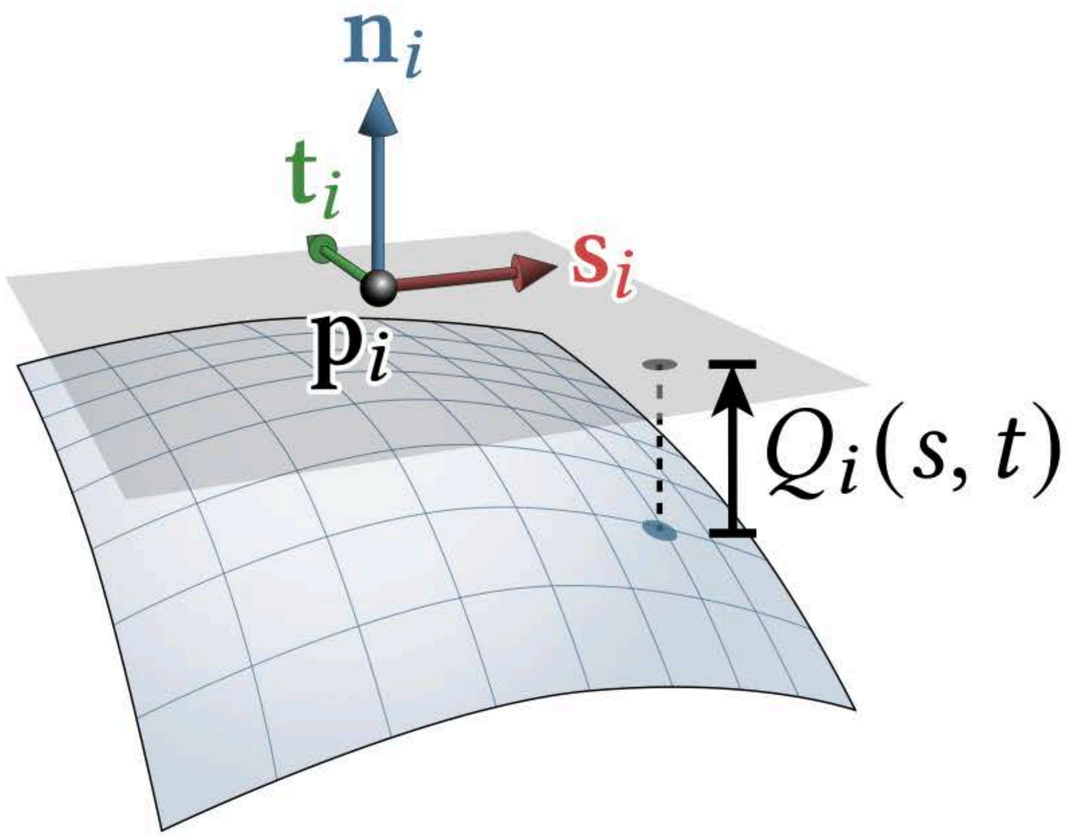
$$Q_i(s,t) = \sum_{n=0}^{2} \sum_{m=0}^{2} a_{n,m} s^n t^m$$



# The surface, one polynomial patch at a time

Quadratic polynomial around each point:

$$Q_i(s,t) = \sum_{n=0}^{2} \sum_{m=0}^{2} a_{n,m} s^n t^m$$



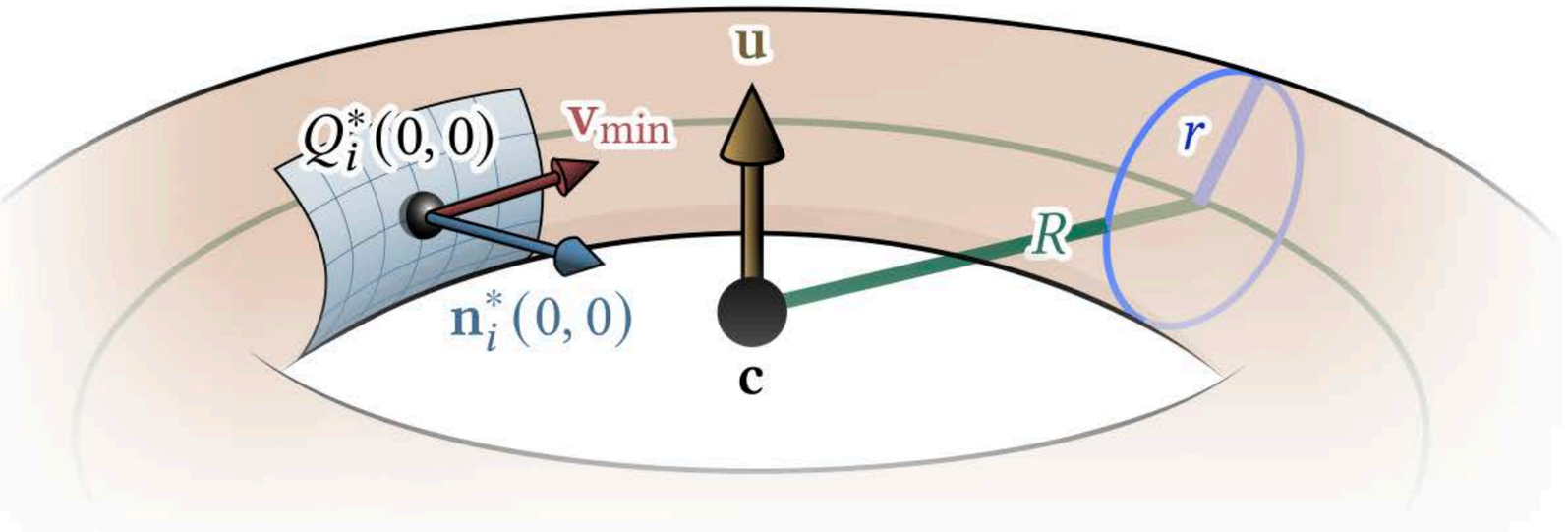

This completely determines a torus!

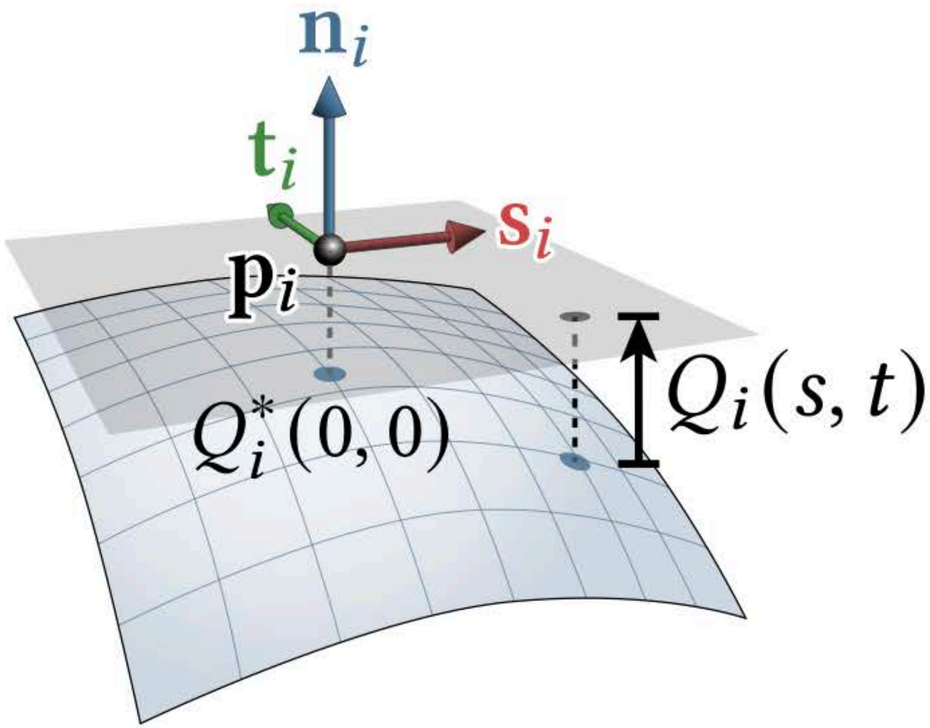
A torus is determined by only six coefficients: $a_{0,0}, a_{0,1}, a_{1,0}, a_{1,1}, a_{0,2}, a_{2,0}$

# Each smooth neighborhood has an osculating torus

Simple, closed-form expressions
for all torus parameters:

- center $\mathbf{c}$
- axis of revolution $\mathbf{u}$
- major radius $R$
- minor radius $r$



# Each smooth neighborhood has an osculating torus

Simple, closed-form expressions for all torus parameters:

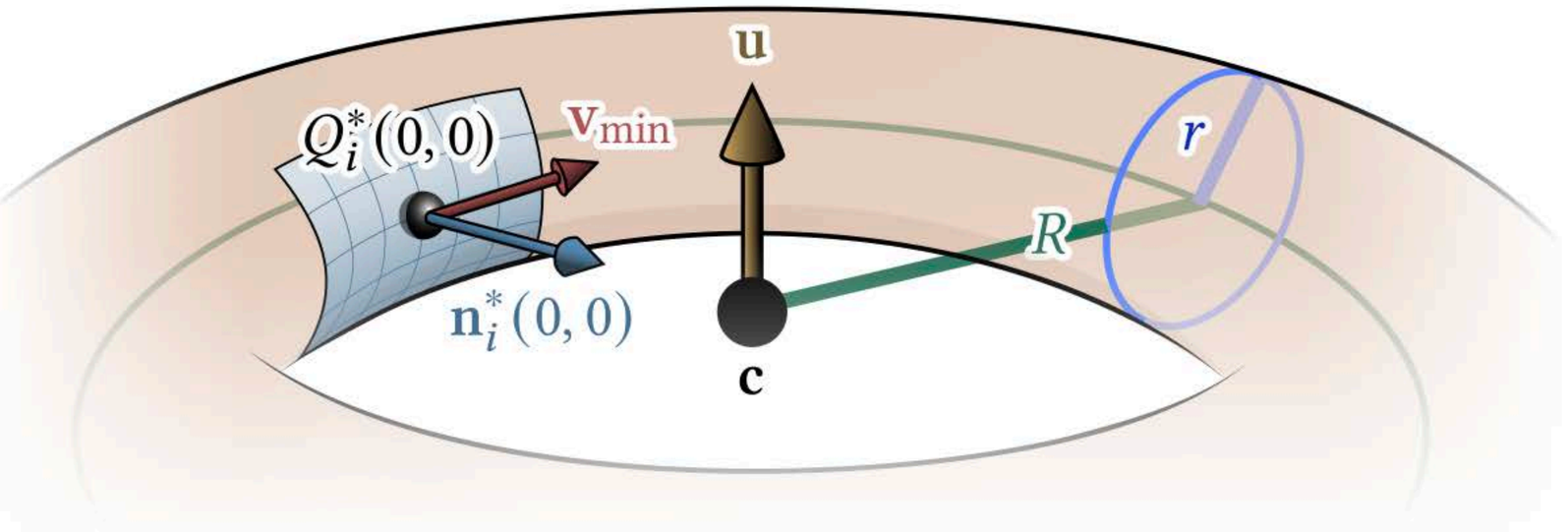
- center $\mathbf{c}$
- axis of revolution $\mathbf{u}$
- major radius $R$
- minor radius $r$

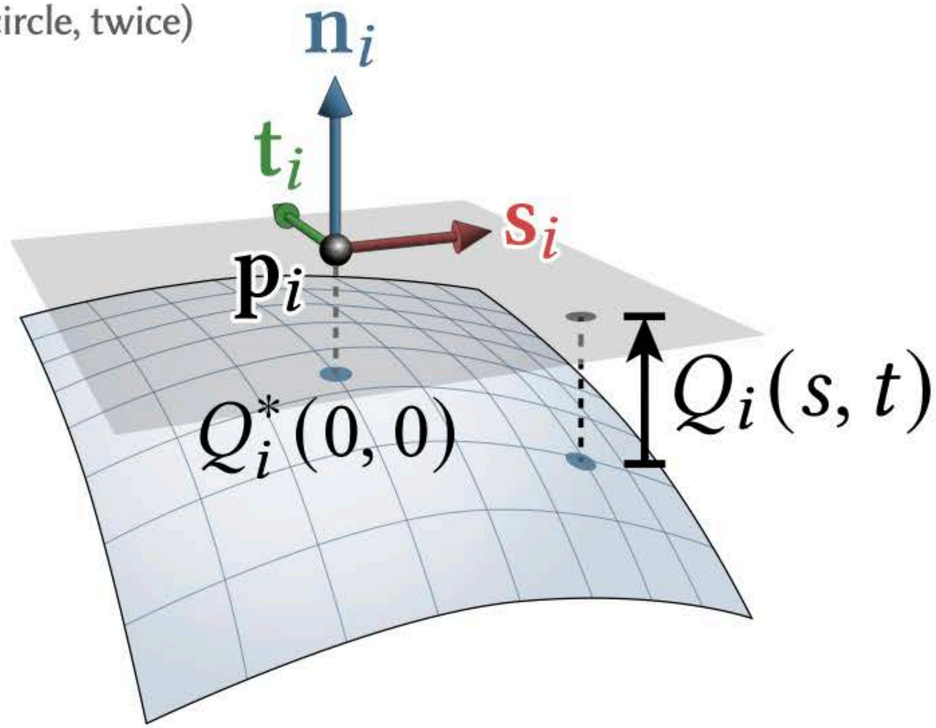
SDF of a torus:

$$\phi_{\mathbb{T}} = \|\mathbf{d}\| - r,$$

*where* $\mathbf{d} := (\|(\mathbf{x} - \mathbf{c}) \times \mathbf{u}\| - R, \langle \mathbf{x} - \mathbf{c}, \mathbf{u} \rangle)$

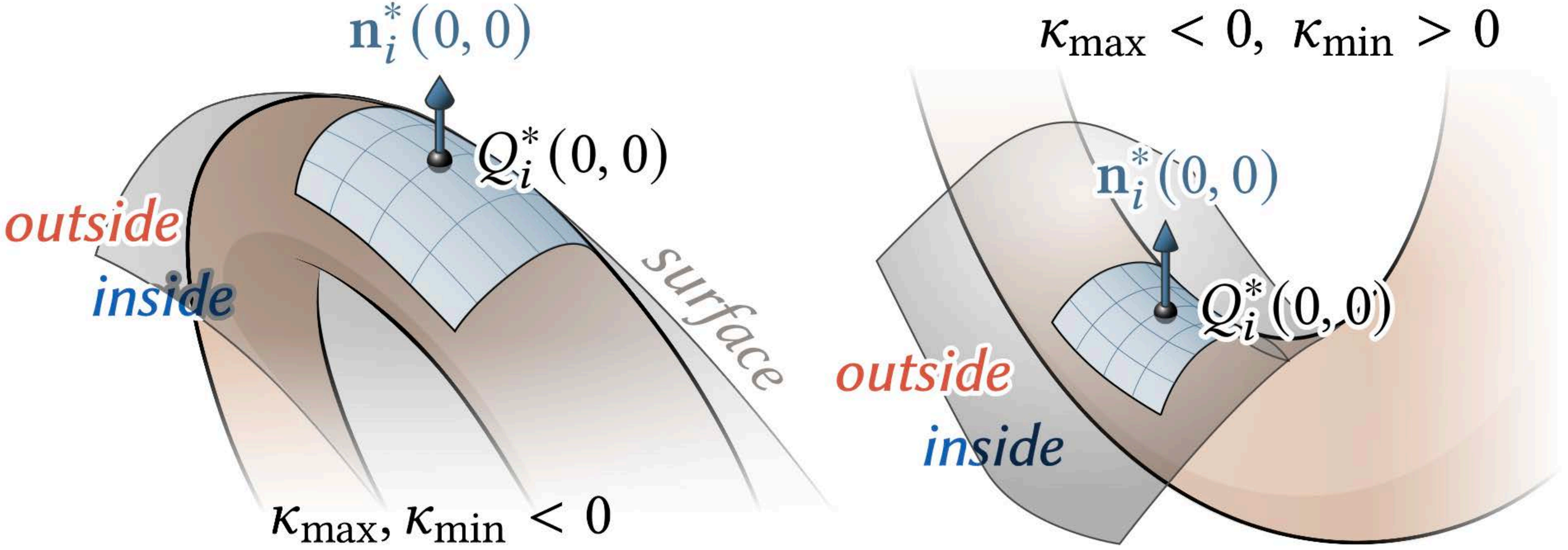
(apply SDF to a circle, twice)



# The torus's sign

# The torus's sign



# Inferring the surface, even locally, is hard

# Inferring the surface, even locally, is hard

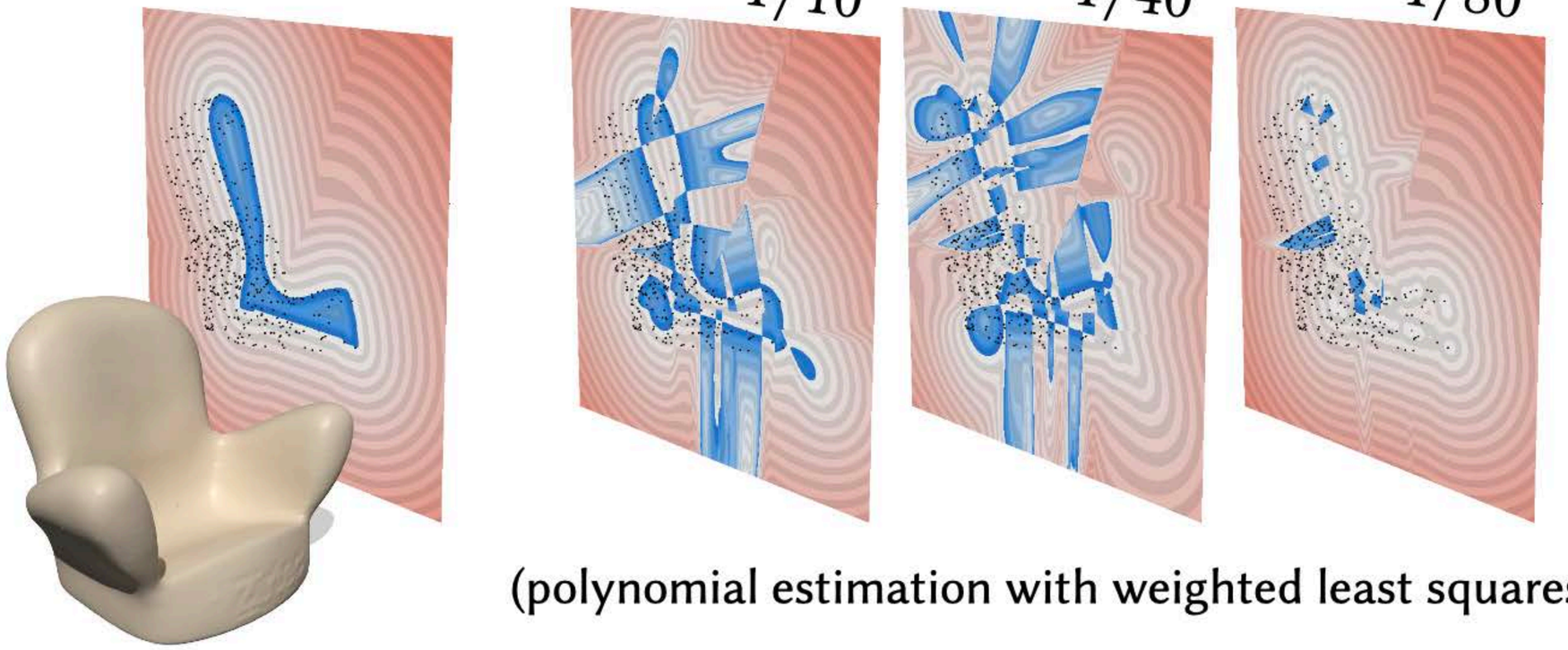
Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

ground truth

neighborhood estimation with Gaussians




(polynomial estimation with weighted least squares)

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

A **huge** bag of possible approaches:

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

A **huge** bag of possible approaches:

- Endless kernel-based reconstruction methods (*MLS, partition-of-unity, meshless interpolation…*)

[Lancaster & Salkauskas 1981; Levin 1998; Turk 7 O'Brien 1999; Carr et al. 2001; Boissonnat & Cazals 2002; Amenta & Kil 2004; Shen et al. 2004; Dey & Sun 2005; Ohtake et al. 2005; Fleishman et al. 2005; Adamson & Alexa 2006; Kolluri 2008; Wang et al. 2008; Öztireli et al. 2009; Zagorchev & Goshtasby 2012; Fuhrmann & Goesele 2014; Wei et al. 2023…]

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

A **huge** bag of possible approaches:

- Endless kernel-based reconstruction methods (*MLS, partition-of-unity, meshless interpolation…*)
- Anisotropic kernels *(PCA, quadric error metric, etc.)* ⟶ *brittle*

[Lancaster & Salkauskas 1981; Levin 1998; Turk 7 O'Brien 1999; Carr et al. 2001; Boissonnat & Cazals 2002; Amenta & Kil 2004; Shen et al. 2004; Dey & Sun 2005; Ohtake et al. 2005; Fleishman et al. 2005; Adamson & Alexa 2006; Kolluri 2008; Wang et al. 2008; Öztireli et al. 2009; Zagorchev & Goshtasby 2012; Fuhrmann & Goesele 2014; Wei et al. 2023…]

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

A **huge** bag of possible approaches:

- Endless kernel-based reconstruction methods (*MLS, partition-of-unity, meshless interpolation…*)
- Anisotropic kernels *(PCA, quadric error metric, etc.)* ⟶ *brittle*
- Hierarchy ⟶ *brittle*

[Lancaster & Salkauskas 1981; Levin 1998; Turk 7 O'Brien 1999; Carr et al. 2001; Boissonnat & Cazals 2002; Amenta & Kil 2004; Shen et al. 2004; Dey & Sun 2005; Ohtake et al. 2005; Fleishman et al. 2005; Adamson & Alexa 2006; Kolluri 2008; Wang et al. 2008; Öztireli et al. 2009; Zagorchev & Goshtasby 2012; Fuhrmann & Goesele 2014; Wei et al. 2023…]

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

A **huge** bag of possible approaches:

- Endless kernel-based reconstruction methods (*MLS, partition-of-unity, meshless interpolation…*)
- Anisotropic kernels *(PCA, quadric error metric, etc.)* ⟶ *brittle*
- Hierarchy ⟶ *brittle*
- Iterative/nonlinear solvers ⟶ *slow*

[Lancaster & Salkauskas 1981; Levin 1998; Turk 7 O'Brien 1999; Carr et al. 2001; Boissonnat & Cazals 2002; Amenta & Kil 2004; Shen et al. 2004; Dey & Sun 2005; Ohtake et al. 2005; Fleishman et al. 2005; Adamson & Alexa 2006; Kolluri 2008; Wang et al. 2008; Öztireli et al. 2009; Zagorchev & Goshtasby 2012; Fuhrmann & Goesele 2014; Wei et al. 2023…]

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

A **huge** bag of possible approaches:

- Endless kernel-based reconstruction methods (*MLS, partition-of-unity, meshless interpolation…*)
- Anisotropic kernels *(PCA, quadric error metric, etc.)* ⟶ *brittle*
- Hierarchy ⟶ *brittle*
- Iterative/nonlinear solvers ⟶ *slow*
- Robust statistics ⟶ *slow*

[Lancaster & Salkauskas 1981; Levin 1998; Turk 7 O'Brien 1999; Carr et al. 2001; Boissonnat & Cazals 2002; Amenta & Kil 2004; Shen et al. 2004; Dey & Sun 2005; Ohtake et al. 2005; Fleishman et al. 2005; Adamson & Alexa 2006; Kolluri 2008; Wang et al. 2008; Öztireli et al. 2009; Zagorchev & Goshtasby 2012; Fuhrmann & Goesele 2014; Wei et al. 2023…]

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

A **huge** bag of possible approaches:

- Endless kernel-based reconstruction methods (*MLS, partition-of-unity, meshless interpolation…*)
- Anisotropic kernels *(PCA, quadric error metric, etc.)* ⟶ *brittle*
- Hierarchy ⟶ *brittle*
- Iterative/nonlinear solvers ⟶ *slow*
- Robust statistics ⟶ *slow*
- *… any one method is unlikely to generalize!*

[Lancaster & Salkauskas 1981; Levin 1998; Turk 7 O'Brien 1999; Carr et al. 2001; Boissonnat & Cazals 2002; Amenta & Kil 2004; Shen et al. 2004; Dey & Sun 2005; Ohtake et al. 2005; Fleishman et al. 2005; Adamson & Alexa 2006; Kolluri 2008; Wang et al. 2008; Öztireli et al. 2009; Zagorchev & Goshtasby 2012; Fuhrmann & Goesele 2014; Wei et al. 2023…]

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting, moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

A **huge** bag of possible approaches:

- Endless kernel-based reconstruction methods *(MLS, partition of unity, meshless finite elements)*
- Anisotropic kernels *(PCA, quadric error metric, etc.)* ⟶ *brittle*
- Hierarchy ⟶ *brittle*
- Iterative/nonlinear solvers ⟶ *slow*
- Robust statistics ⟶ *slow*
- …

Just use a *fixed-size* neighborhood with a *data-driven* approach.

# Inferring the surface, even locally, is hard

Classic methods like *jet-fitting*, *moving least squares* are extremely dependent on brittle, domain-specific heuristics for neighborhood size and shape.

A **huge** bag of possible approaches:

- Endless kernel-based reconstruction methods (*MLS, partition-of-unity, wavelets, sparse, ...*)
- Anisotropic kernels (*PCA, quadric approximation*) → *brittle*
- Hierarchy → *brittle*
- Iterative, implicit, ... → *slow*
- Robust statistics → *slow*
- ...

Just use a *fixed-size* neighborhood with a *data-driven* approach.

$$\phi(x) = \frac{\sum_{i=1}^{|P|} \underbrace{g_i(x)}_{\textit{local function}} \exp\left(-\lambda \|x - p_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|x - p_i\|\right)}$$

- Dominated by local surface behavior

# Inferring the surface, even locally, is hard

Just use a *fixed-size* neighborhood with a *data-driven* approach.

*local function*

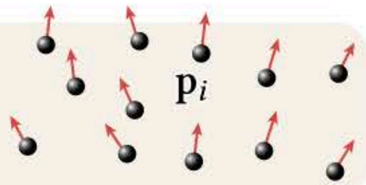
$$\phi(x) = \frac{\sum_{i=1}^{|P|} g_i(x) \exp\left(-\lambda \|x - p_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|x - p_i\|\right)}$$

- Dominated by local surface behavior
- Easy to generate data for small surface patches

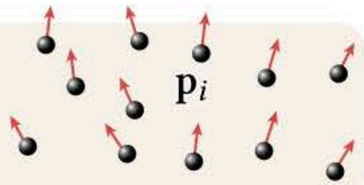
# Network architecture

# Network architecture

**Input:** size-$k$ neighborhood



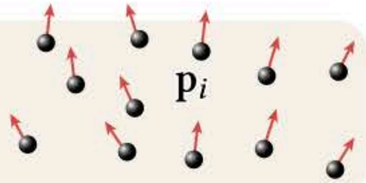
# Network architecture

**Input:** size-$k$ neighborhood 


**Output:** six polynomial coefficients

# Network architecture

**Input:** size-$k$ neighborhood



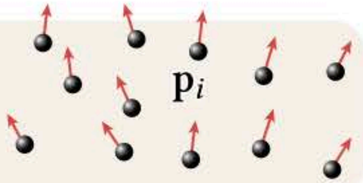

coordinate features

normal features

**Output:** six polynomial coefficients

# Network architecture

**Input:** size-$k$ neighborhood




per-point MLP

coordinate features

normal features

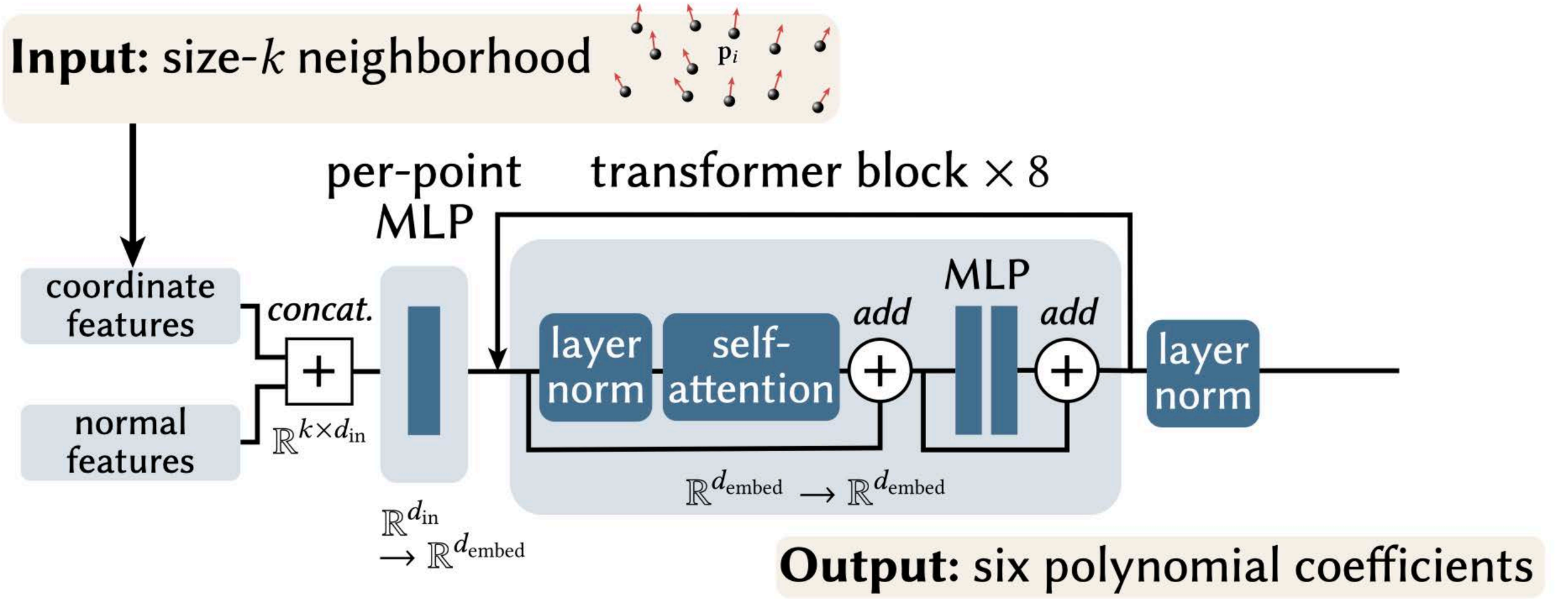
*concat.*

$\mathbb{R}^{k \times d_{\text{in}}}$

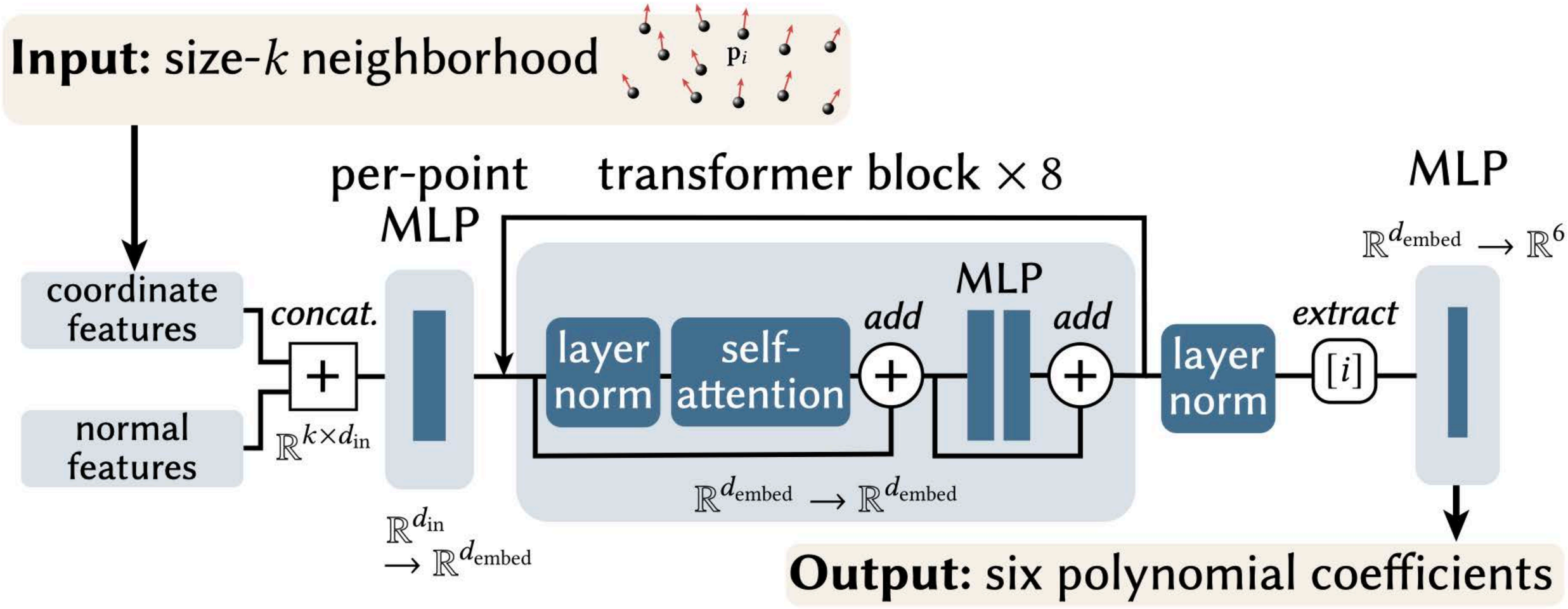
$\mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{embed}}}$

**Output:** six polynomial coefficients

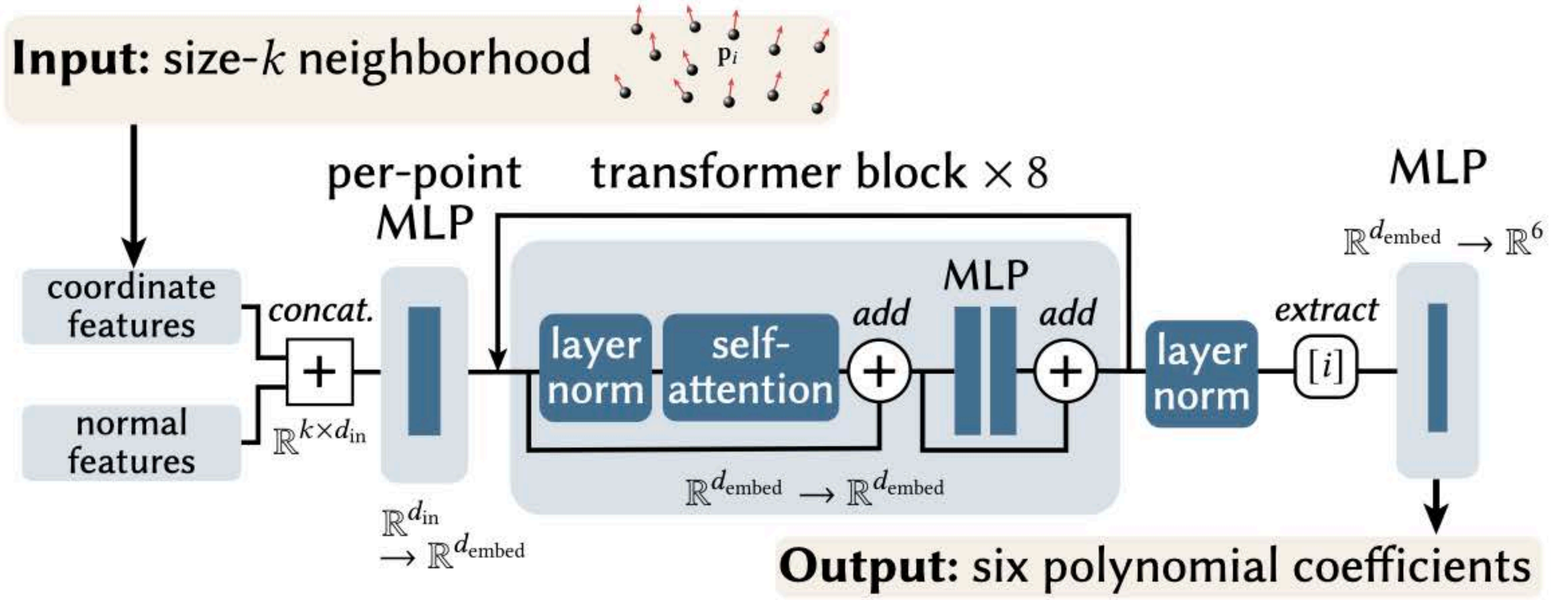
# Network architecture



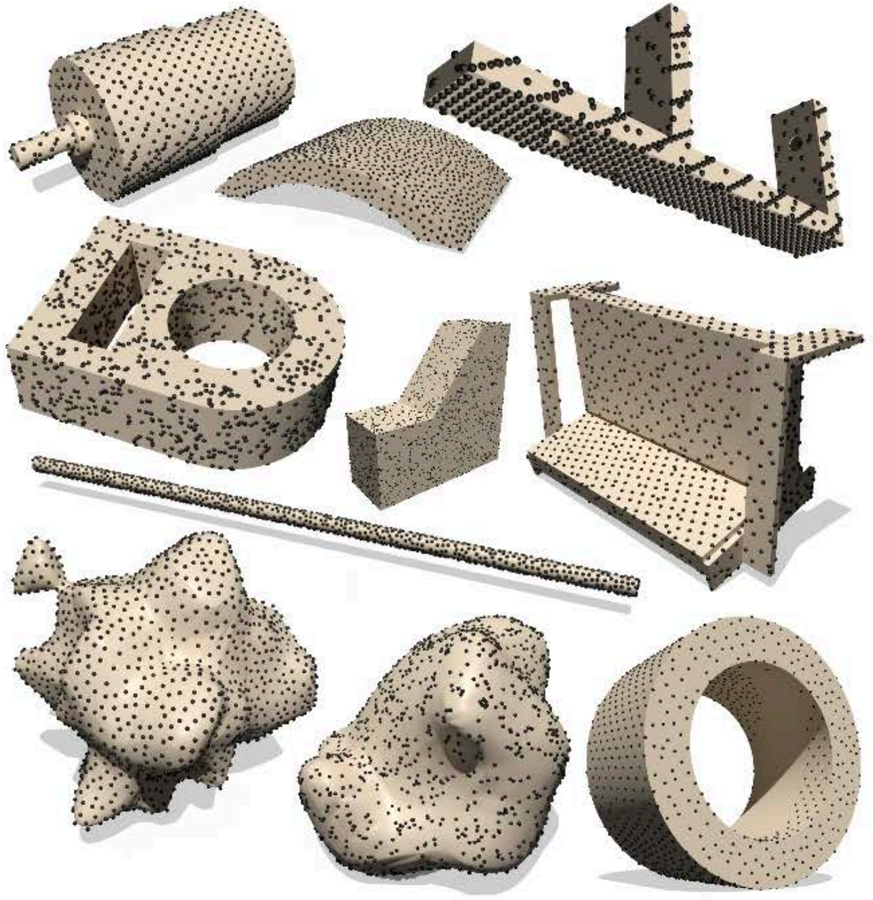
# Network architecture



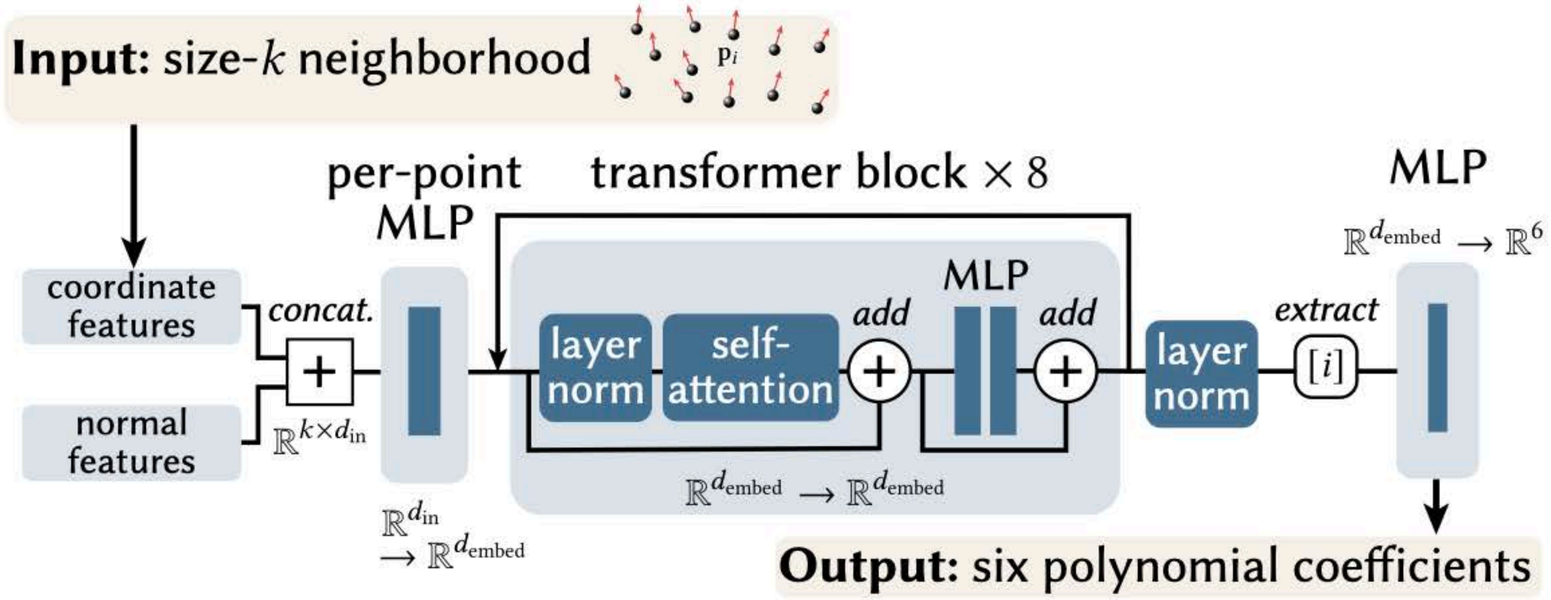
# Network architecture




## Training data

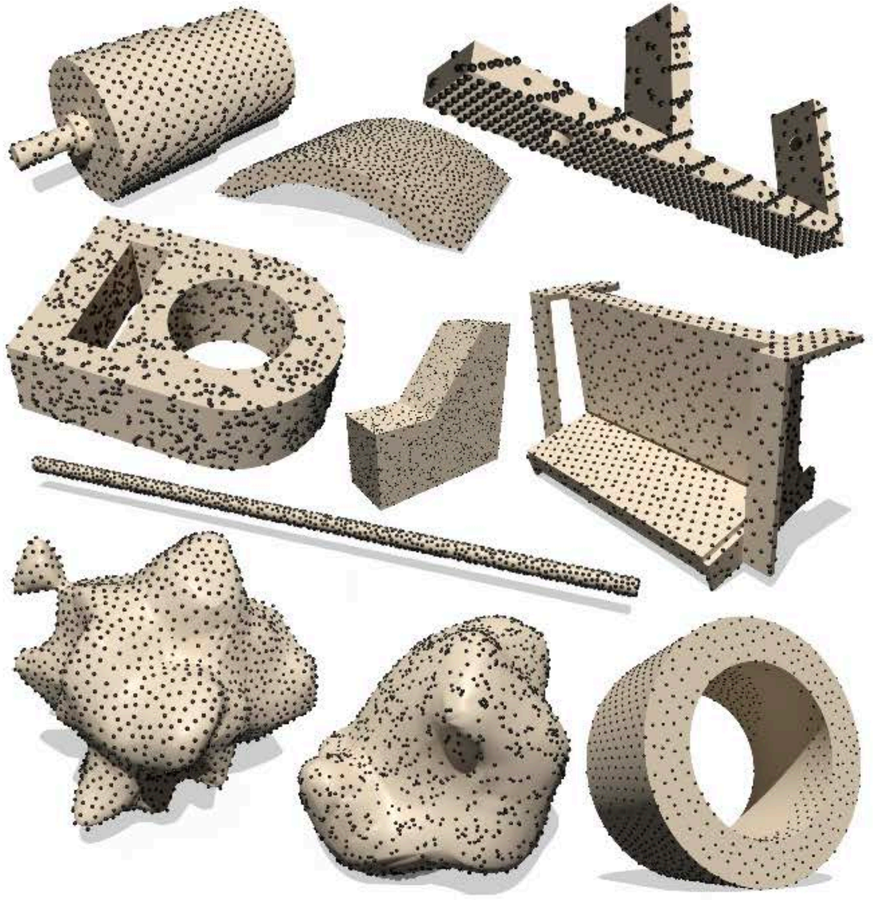


# Network architecture

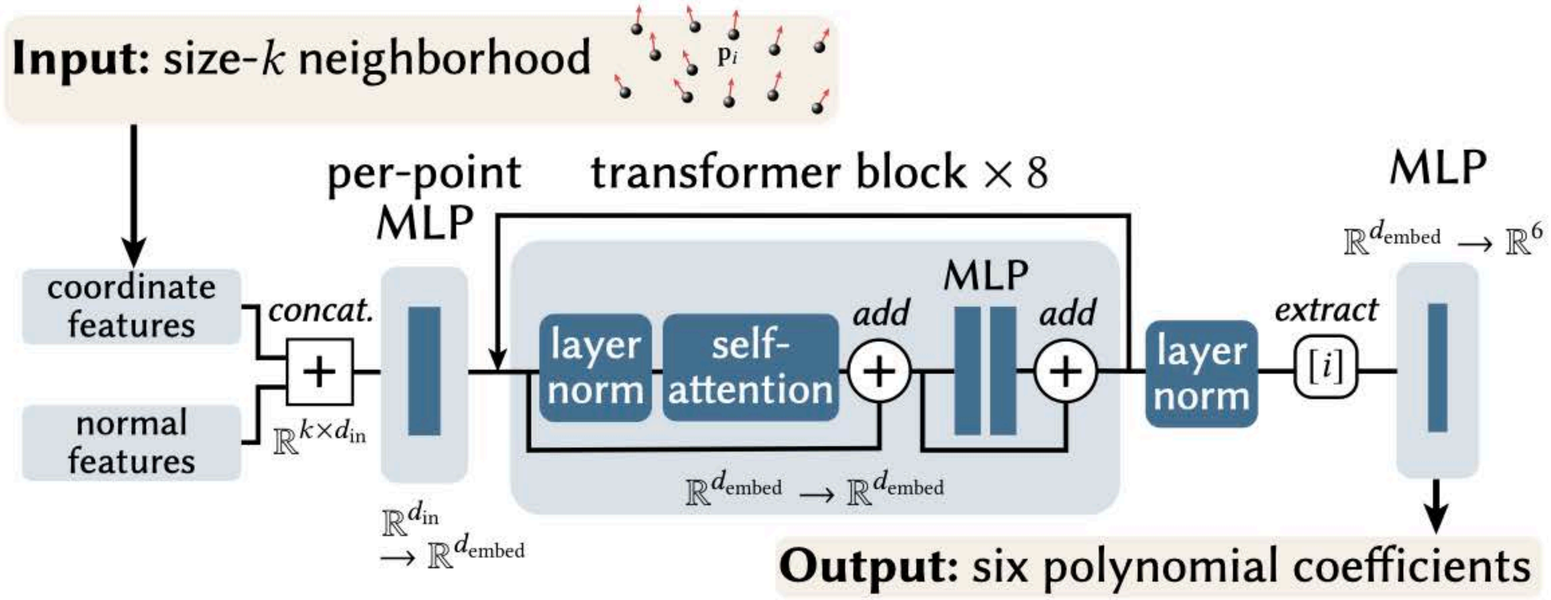



- 10,000 models from ABC dataset [Koch et al. 2019], 2000 procedural

## Training data

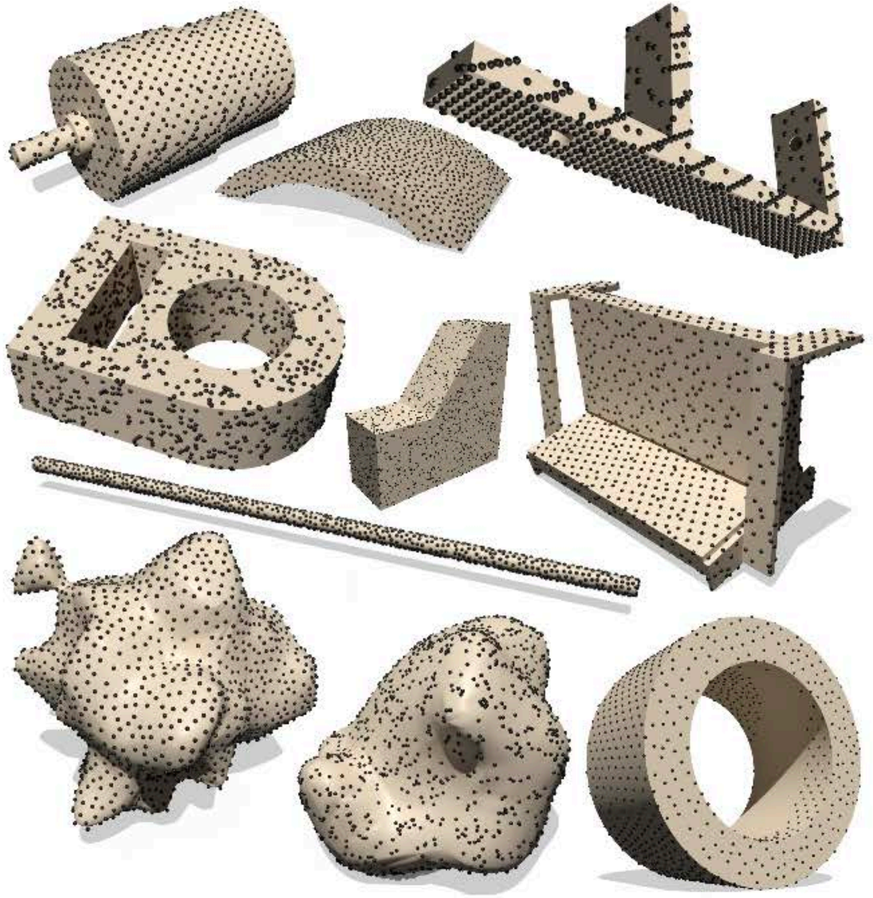


# Network architecture

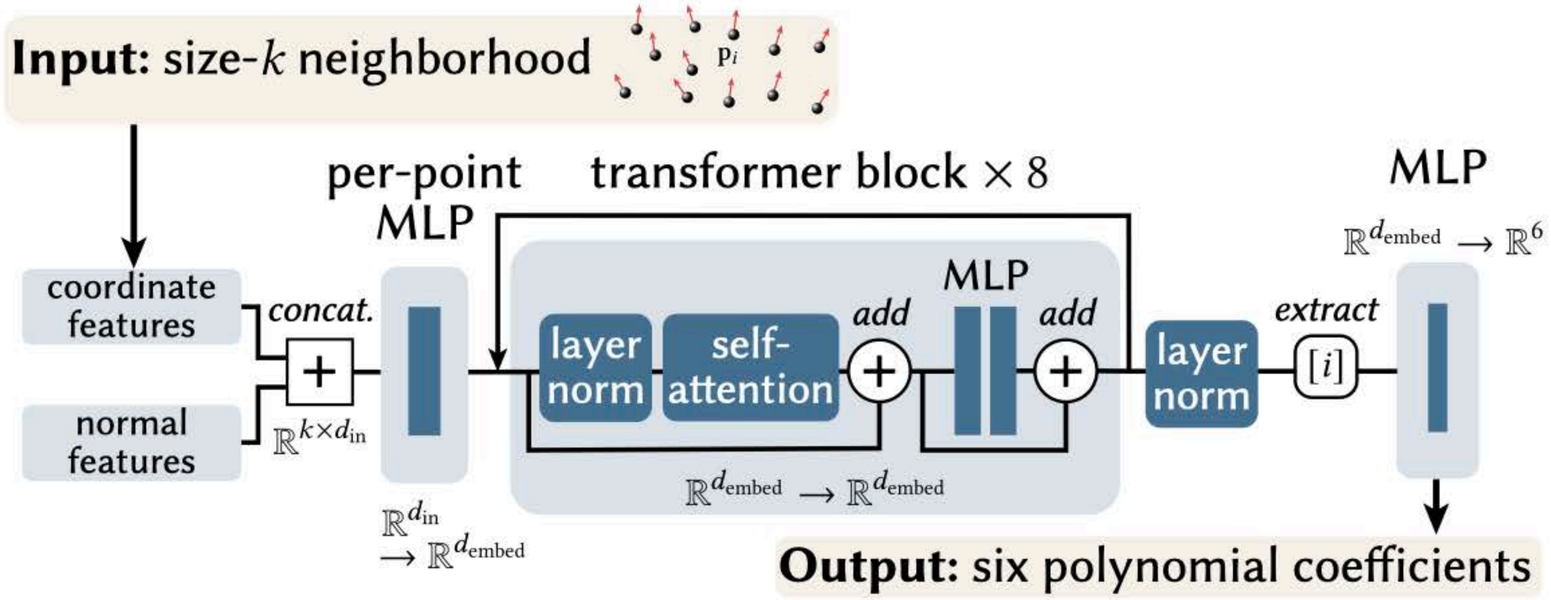



- 10,000 models from ABC dataset [Koch et al. 2019], 2000 procedural
- Normalize and sample into size-2048 point clouds:
  - Uniform sampling
  - Farthest-point sampling
  - Simulated scanning

## Training data

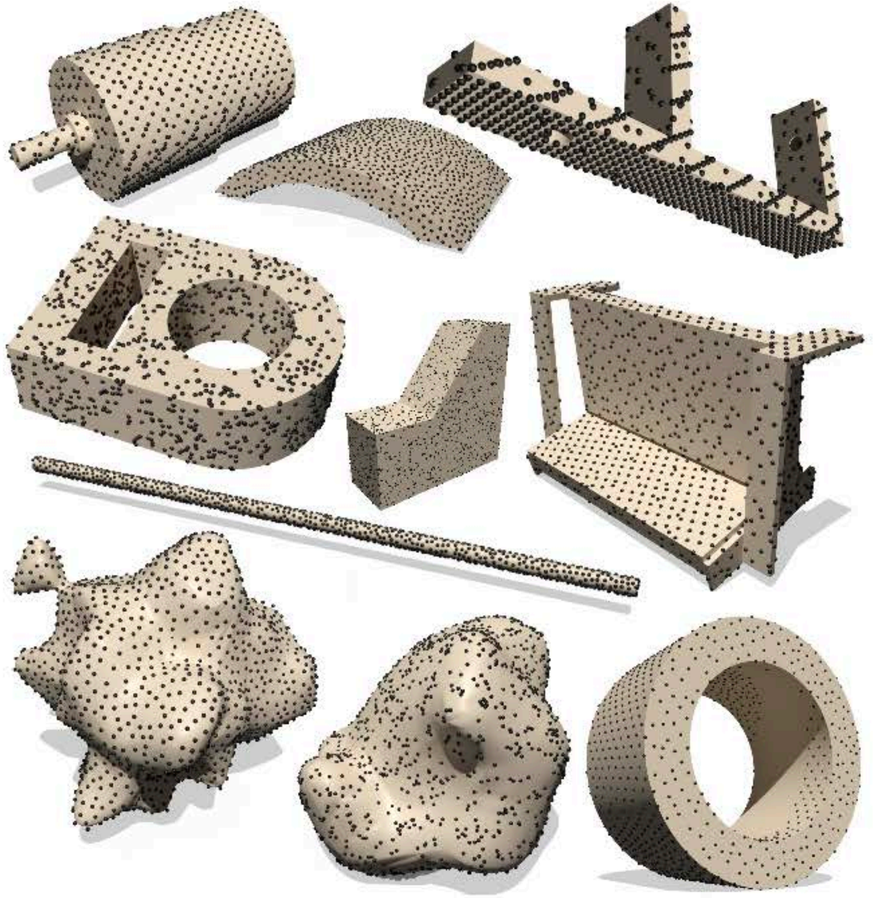


# Network architecture

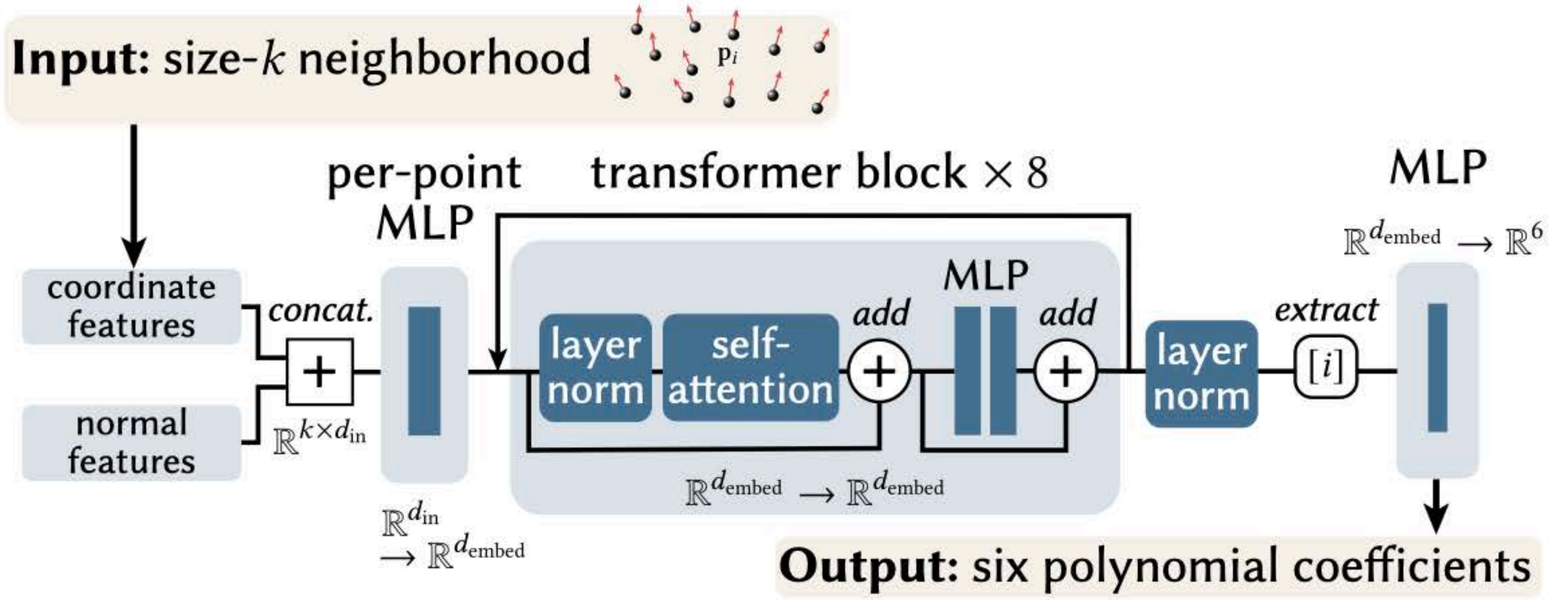



- 10,000 models from ABC dataset [Koch et al. 2019], 2000 procedural
- Normalize and sample into size-2048 point clouds:
  - Uniform sampling
  - Farthest-point sampling
  - Simulated scanning
- Input neighborhood of size $k = 64$

## Training data

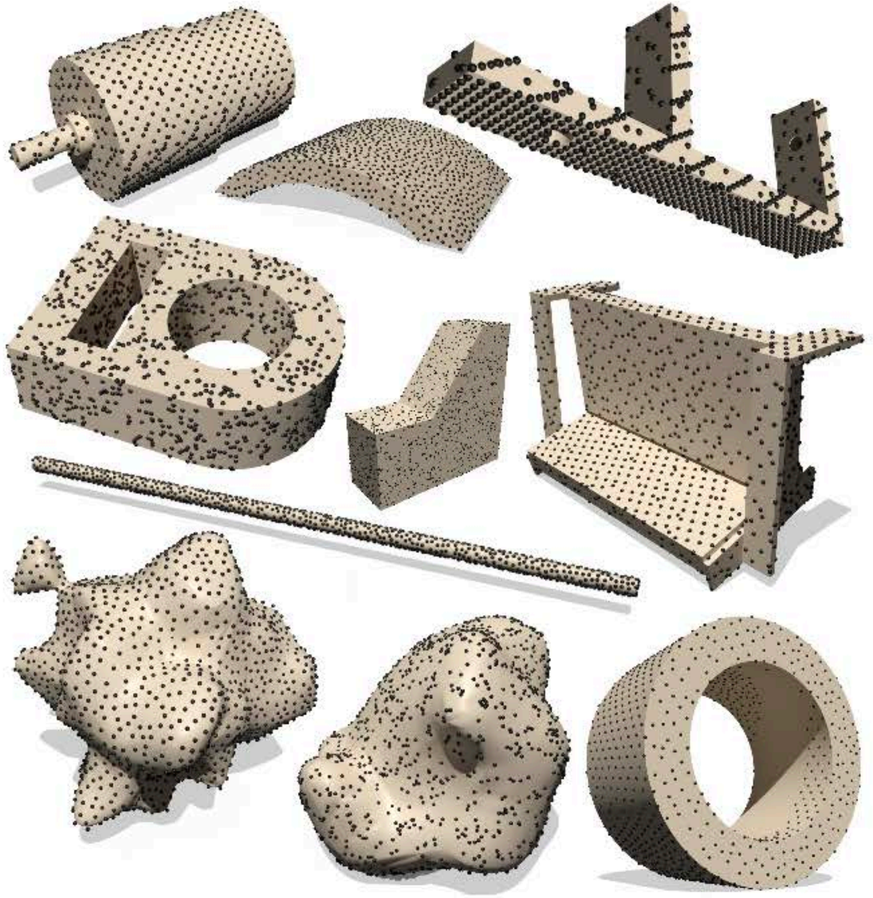


# Network architecture

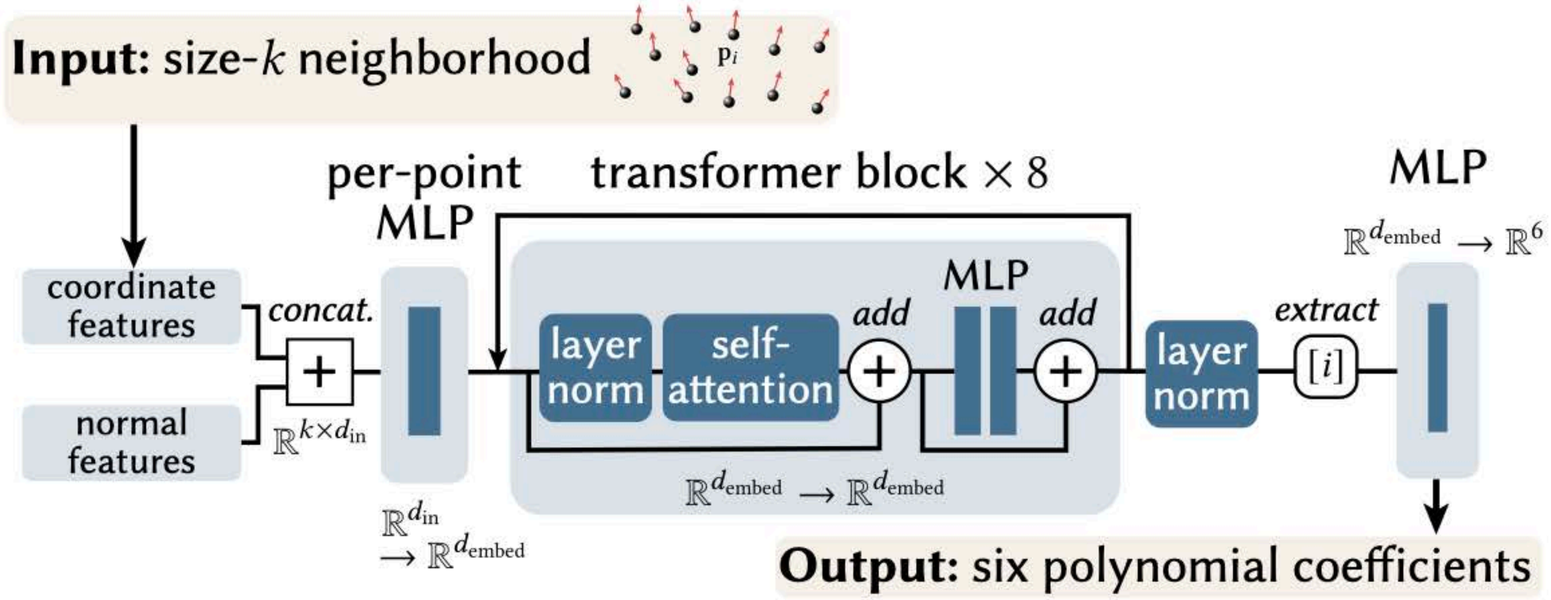



- 10,000 models from ABC dataset [Koch et al. 2019], 2000 procedural
- Normalize and sample into size-2048 point clouds:
  - Uniform sampling
  - Farthest-point sampling
  - Simulated scanning
- Input neighborhood of size $k = 64$
- >40 million training examples

## Training data

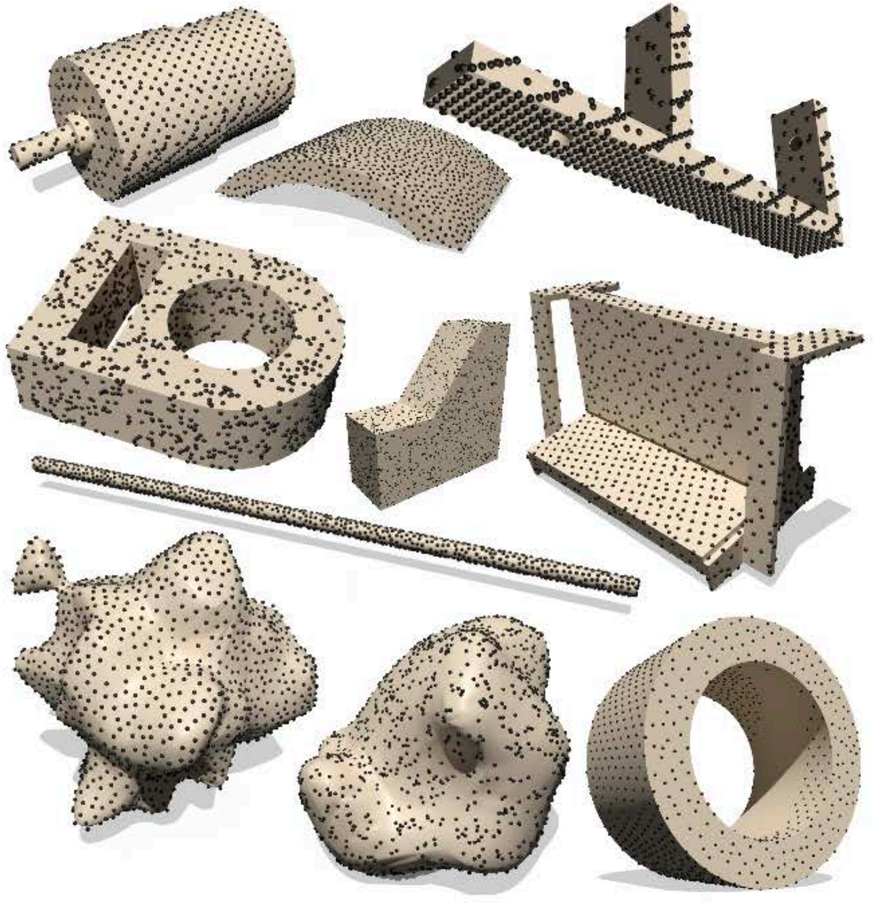


# Network architecture

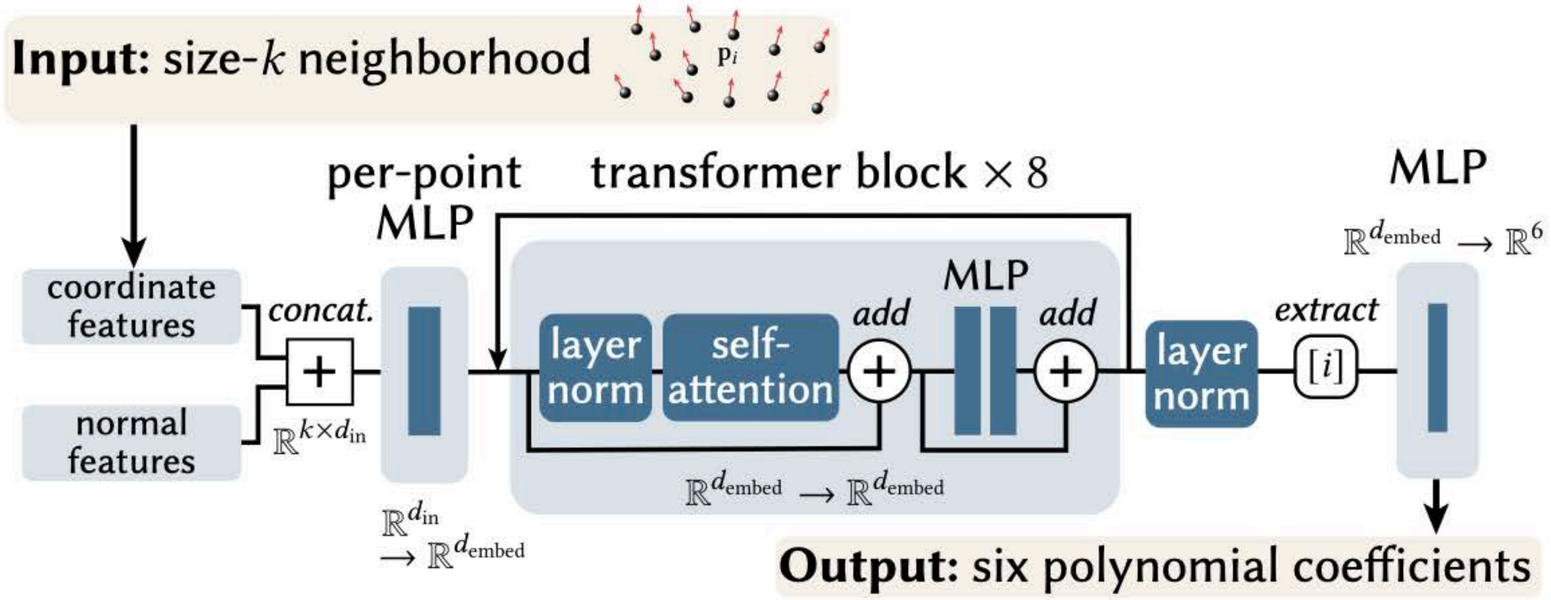



## Loss function

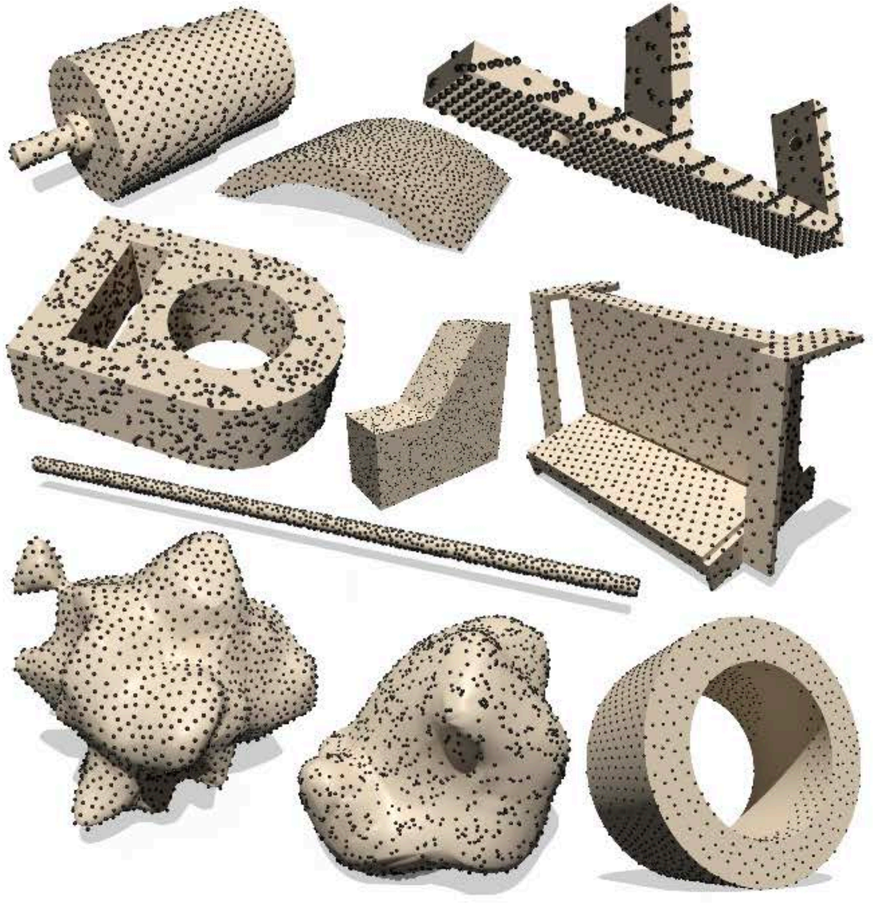
## Training data



# Network architecture




## Training data

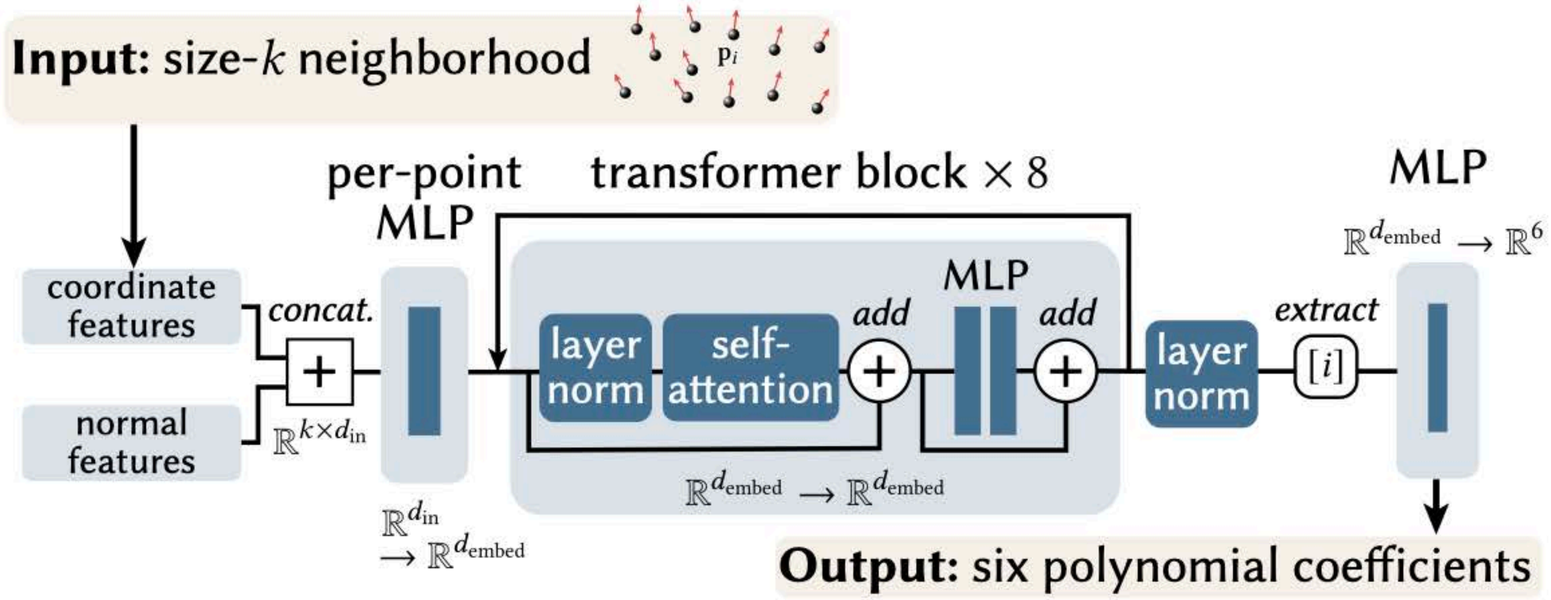


## Loss function

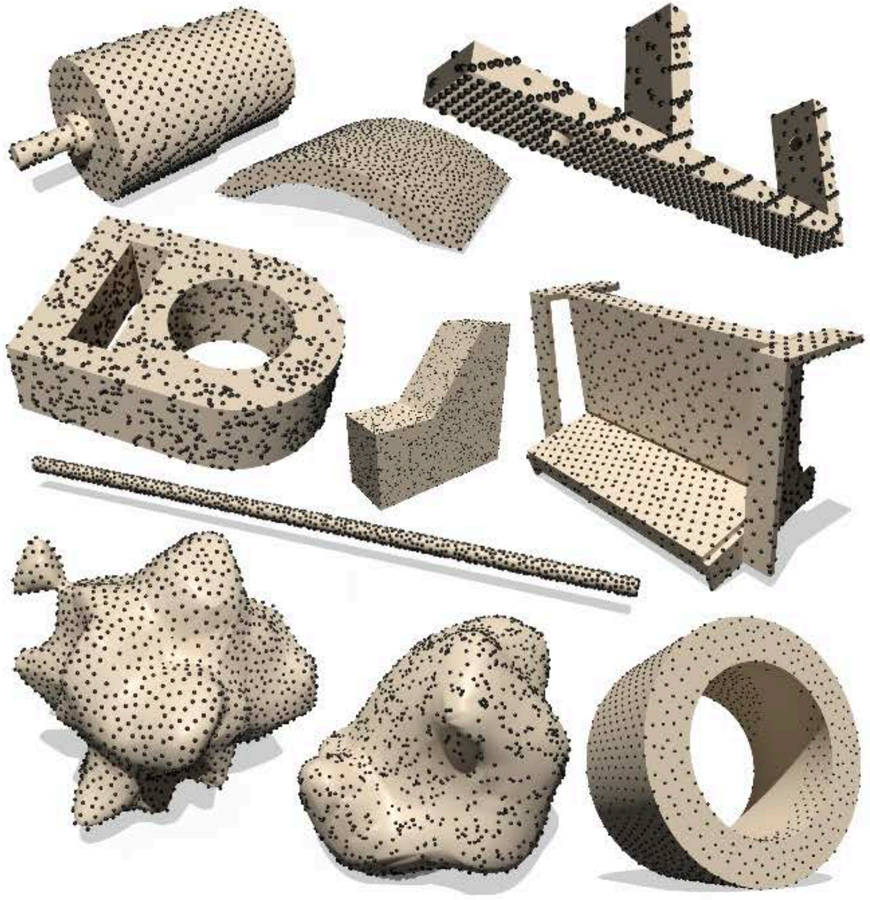
$$\mathcal{L} \coloneqq \mathcal{L}_{\text{distance}}$$

$$\text{where } \mathcal{L}_{\text{distance}} \coloneqq \frac{1}{Q} \sum_{j=1}^{Q} \left| \phi(\mathbf{q}_j) - \phi_{\text{true}}(\mathbf{q}_j) \right|,$$

# Network architecture




## Training data



## Loss function

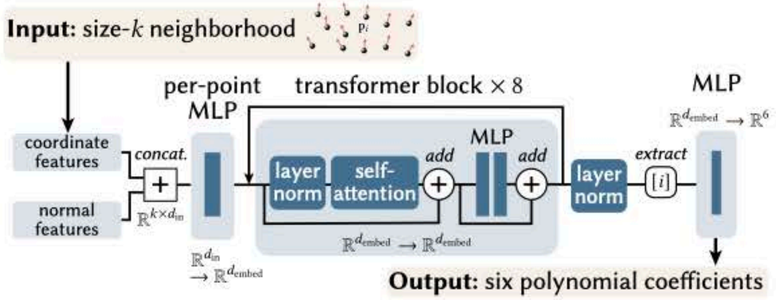
$$\mathcal{L} \coloneqq \mathcal{L}_{\text{distance}} + \mathcal{L}_{\text{eikonal}}$$

$$\text{where } \mathcal{L}_{\text{distance}} \coloneqq \frac{1}{Q} \sum_{j=1}^{Q} \left| \phi(\mathbf{q}_j) - \phi_{\text{true}}(\mathbf{q}_j) \right|,$$

$$\mathcal{L}_{\text{eikonal}} \coloneqq \frac{1}{Q} \sum_{j=1}^{Q} \left| 1 - \|\nabla \phi(\mathbf{q}_j)\| \right|$$

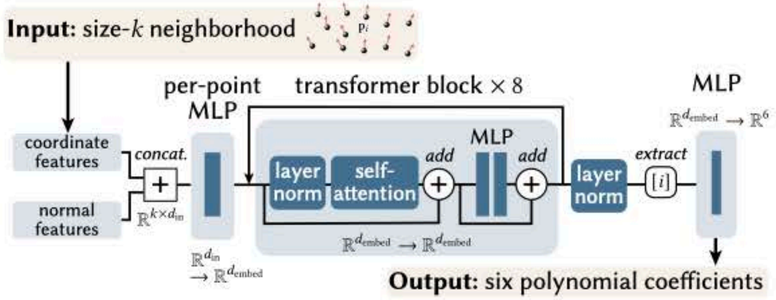
# The *points as tori (PAT)* algorithm

## Pre-training



# The *points as tori (PAT)* algorithm

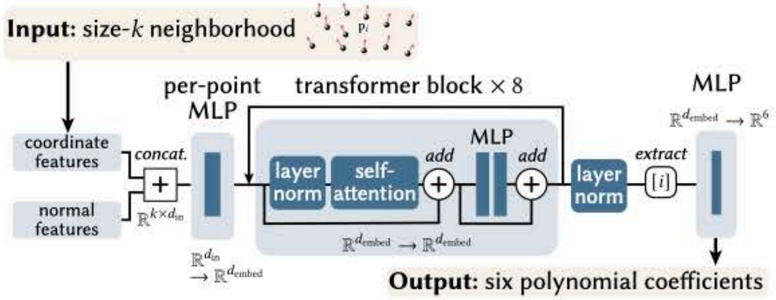
**Pre-training**




*~45 hours on a single NVIDIA RTX 3090*

# The *points as tori (PAT)* algorithm

**Pre-training**

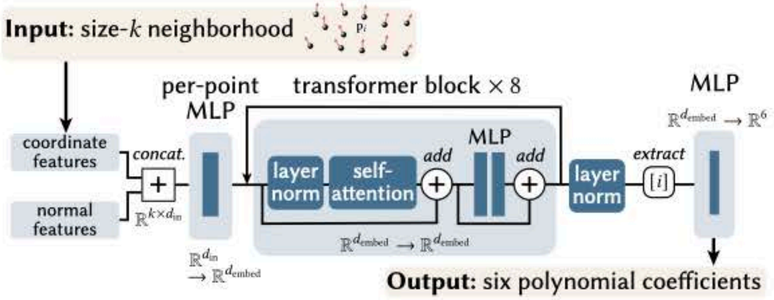



*~45 hours on a single NVIDIA RTX 3090*

*Given an input point cloud* $P = \{\mathbf{p}_i, \mathbf{n}_i\}_{i=1}^{|P|}$ ...

# The *points as tori (PAT)* algorithm

**Pre-training**



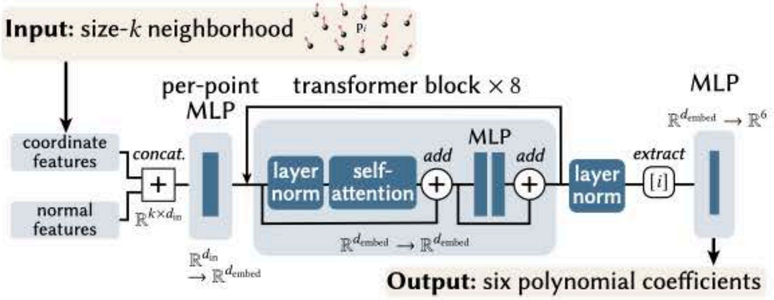

*~45 hours on a single NVIDIA RTX 3090*

*Given an input point cloud* $P = \{\mathbf{p}_i, \mathbf{n}_i\}_{i=1}^{|P|}$ ...

**One-time precompute:**
Apply pre-trained network to each point neighborhood (parallelizable)

# The *points as tori (PAT)* algorithm

**Pre-training**




*~45 hours on a single NVIDIA RTX 3090*

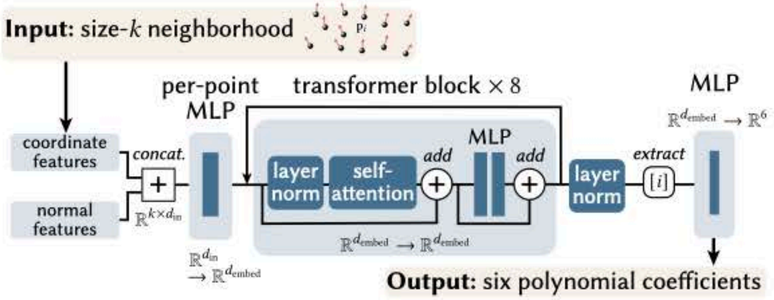
*Given an input point cloud* $P = \{\mathbf{p}_i, \mathbf{n}_i\}_{i=1}^{|P|}$ *...*

**One-time precompute:** *<10s for 100k points, <60s for 1M points*

Apply pre-trained network to each point neighborhood (parallelizable)

# The *points as tori (PAT)* algorithm

**Pre-training**




*~45 hours on a single NVIDIA RTX 3090*

*Given an input point cloud* $P = \{\mathbf{p}_i, \mathbf{n}_i\}_{i=1}^{|P|}$ ...

**One-time precompute:** *<10s for 100k points, <60s for 1M points*

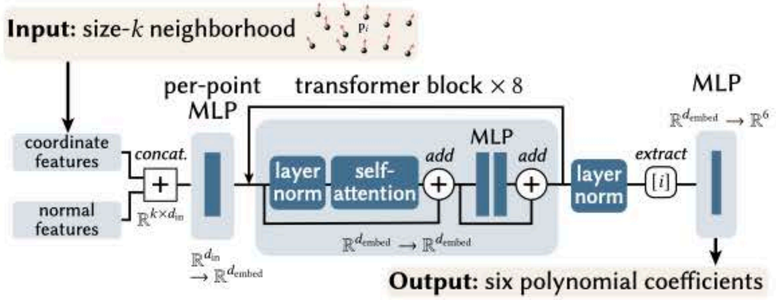
Apply pre-trained network to each point neighborhood (parallelizable)

**Inference time:**

Evaluate the SDF at $\mathbf{x} \in \mathbb{R}^3$ as $\phi(\mathbf{x}) = \dfrac{\sum_{i=1}^{|P|} g_i(\mathbf{x}) \exp\left(-\lambda \|\mathbf{x} - \mathbf{p}_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|\mathbf{x} - \mathbf{p}_i\|\right)}$

# The *points as tori (PAT)* algorithm

**Pre-training**




*~45 hours on a single NVIDIA RTX 3090*

*Given an input point cloud* $P = \{\mathbf{p}_i, \mathbf{n}_i\}_{i=1}^{|P|}$ ...

**One-time precompute:** *<10s for 100k points, <60s for 1M points*

Apply pre-trained network to each point neighborhood (parallelizable)

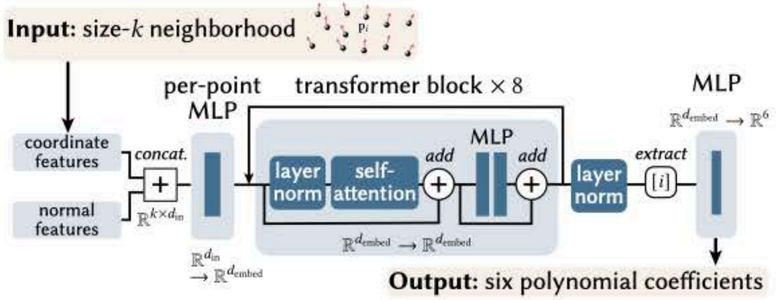
**Inference time:**

Evaluate the SDF at $\mathbf{x} \in \mathbb{R}^3$ as $\phi(\mathbf{x}) = \dfrac{\sum_{i=1}^{|P|} g_i(\mathbf{x}) \exp\left(-\lambda \|\mathbf{x} - \mathbf{p}_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|\mathbf{x} - \mathbf{p}_i\|\right)}$

*$10^{-4}$ —$10^{-3}$ seconds for a single query to 1M-10M+ points*

# The *points as tori (PAT)* algorithm

**Pre-training**




*~45 hours on a single NVIDIA RTX 3090*

*Given an input point cloud* $P = \{\mathbf{p}_i, \mathbf{n}_i\}_{i=1}^{|P|}$ ...

**One-time precompute:**
Apply pre-trained network to each point neighborhood (parallelizable)

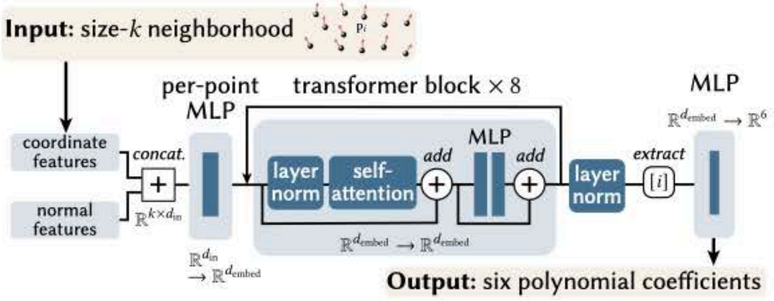
*<10s for 100k points, <60s for 1M points*

**Inference time:**
Evaluate the SDF at $\mathbf{x} \in \mathbb{R}^3$ as $\phi(\mathbf{x}) = \dfrac{\sum_{i=1}^{|P|} g_i(\mathbf{x}) \exp\left(-\lambda \|\mathbf{x} - \mathbf{p}_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|\mathbf{x} - \mathbf{p}_i\|\right)}$

*$10^{-4}$ —$10^{-3}$ seconds for a single query to 1M-10M+ points*
*multiple queries trivially parallelizable*

# The *points as tori (PAT)* algorithm

**Pre-training**




*~45 hours on a single NVIDIA RTX 3090*

*Given an input point cloud* $P = \{\mathbf{p}_i, \mathbf{n}_i\}_{i=1}^{|P|}$ ...

**One-time precompute:**
Apply pre-trained network to each point neighborhood (parallelizable)

*<10s for 100k points, <60s for 1M points*

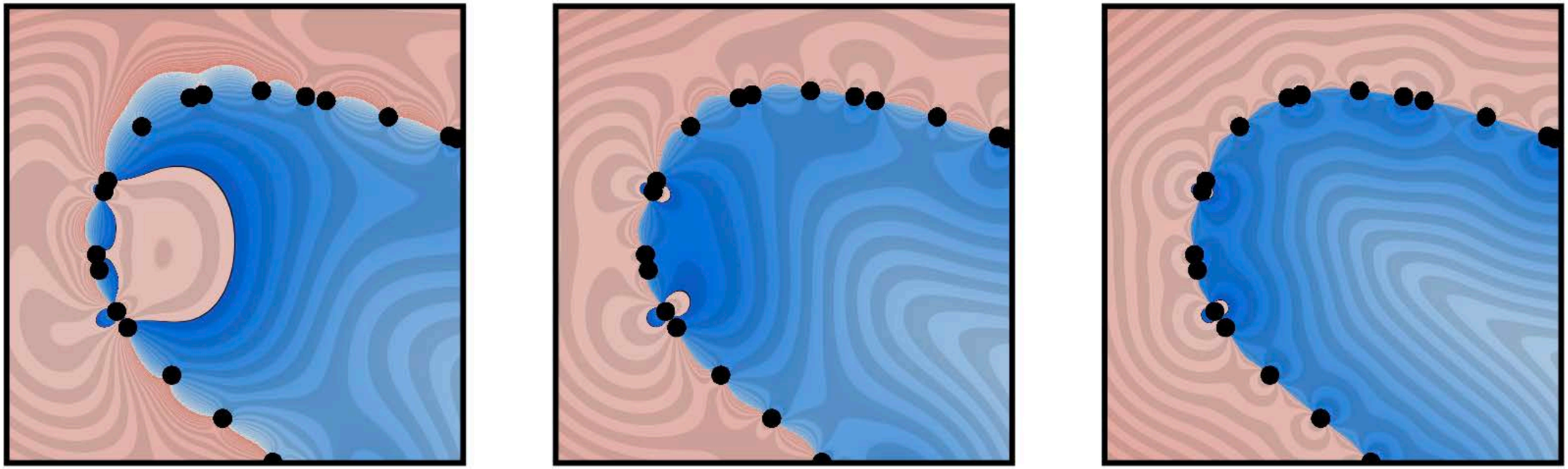
**Inference time:**
Evaluate the SDF at $\mathbf{x} \in \mathbb{R}^3$ as $\phi(\mathbf{x}) = \dfrac{\sum_{i=1}^{|P|} g_i(\mathbf{x}) \exp\left(-\lambda \|\mathbf{x} - \mathbf{p}_i\|\right)}{\sum_{i=1}^{|P|} \exp\left(-\lambda \|\mathbf{x} - \mathbf{p}_i\|\right)}$

simple formula for $\lambda$

*$10^{-4}$ —$10^{-3}$ seconds for a single query to 1M-10M+ points*
*multiple queries trivially parallelizable*

# Better than naive convolutional distance

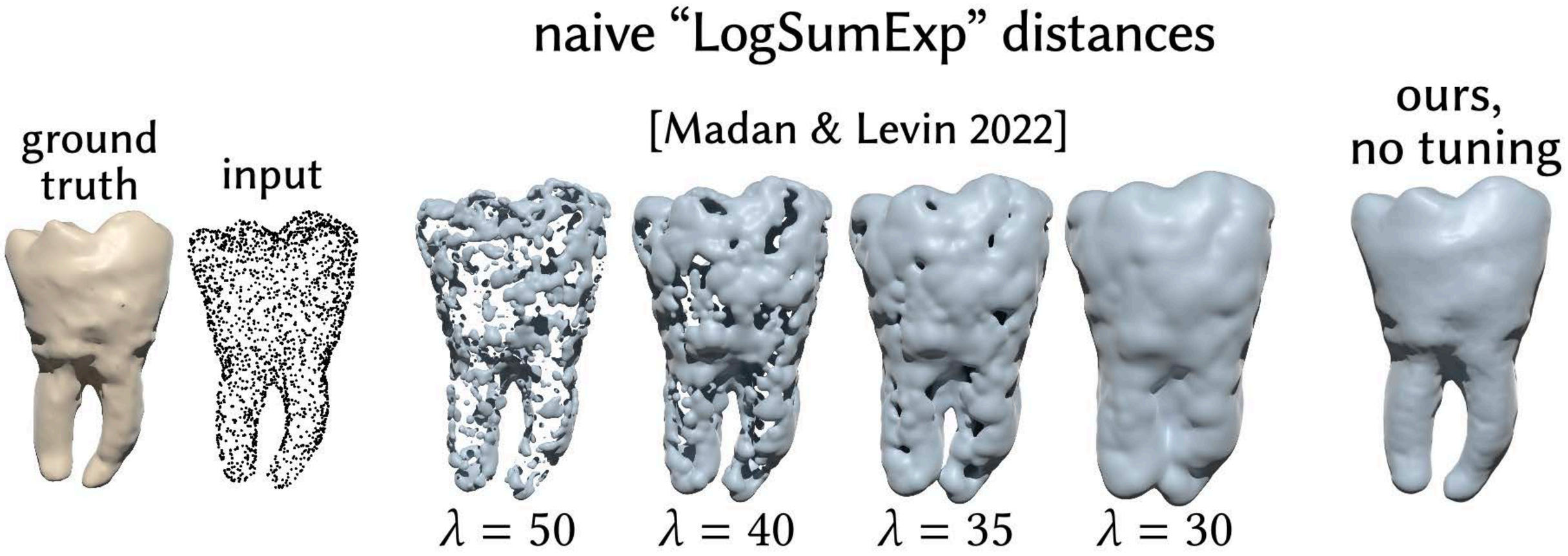
*smaller* $\lambda$ ⟵ ⟶ *larger* $\lambda$



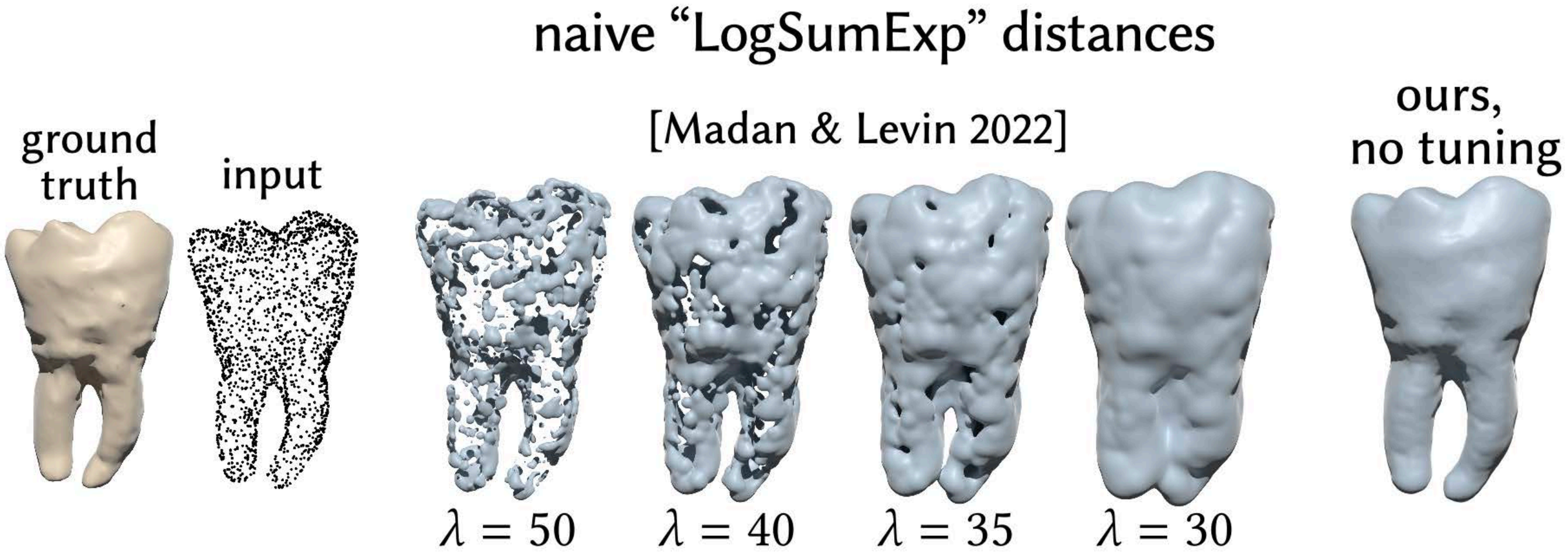
worse eikonality
underfit level sets

better eikonality
overfit level sets

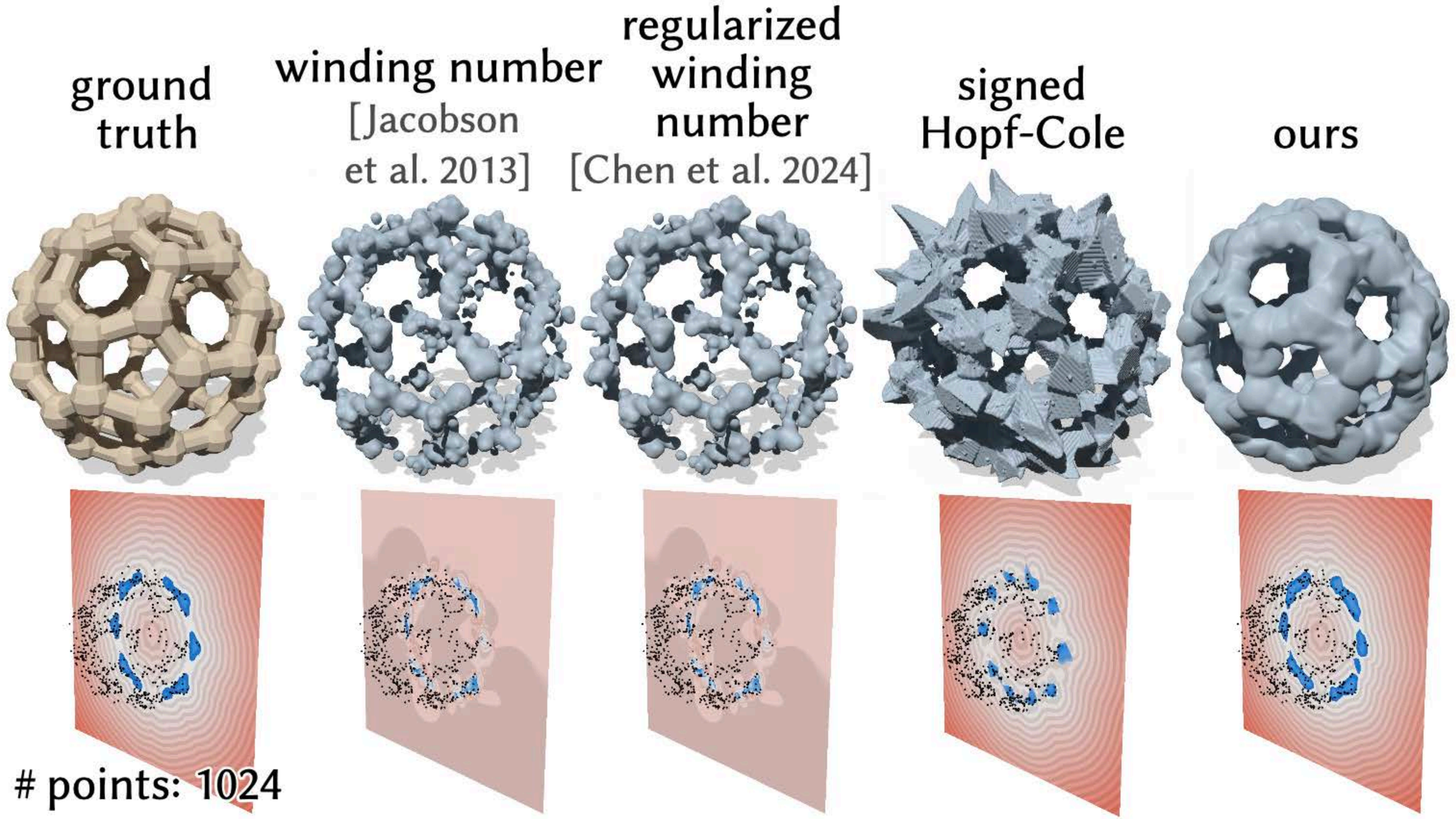
# Better than naive convolutional distance



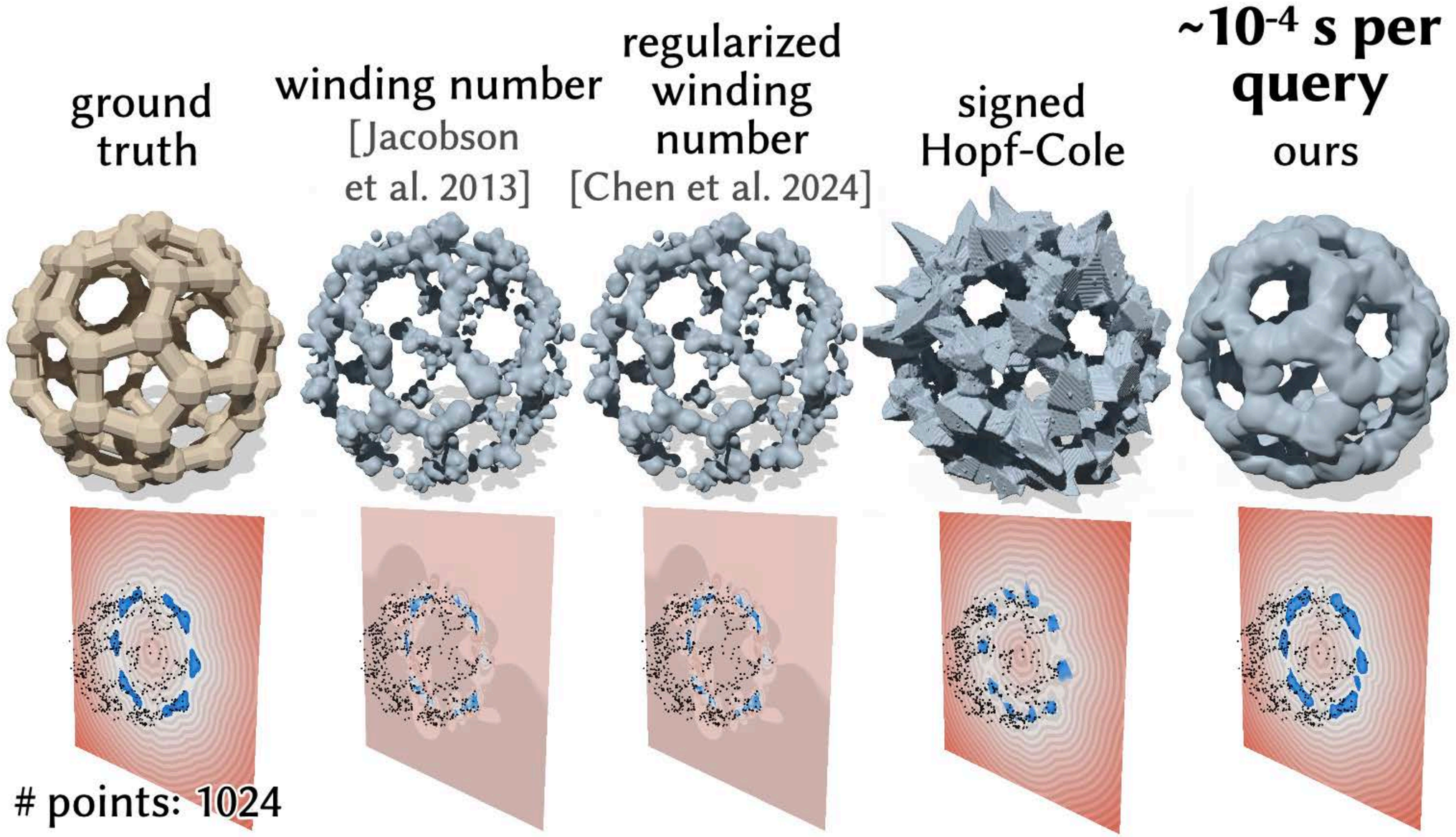
# Better than naive convolutional distance



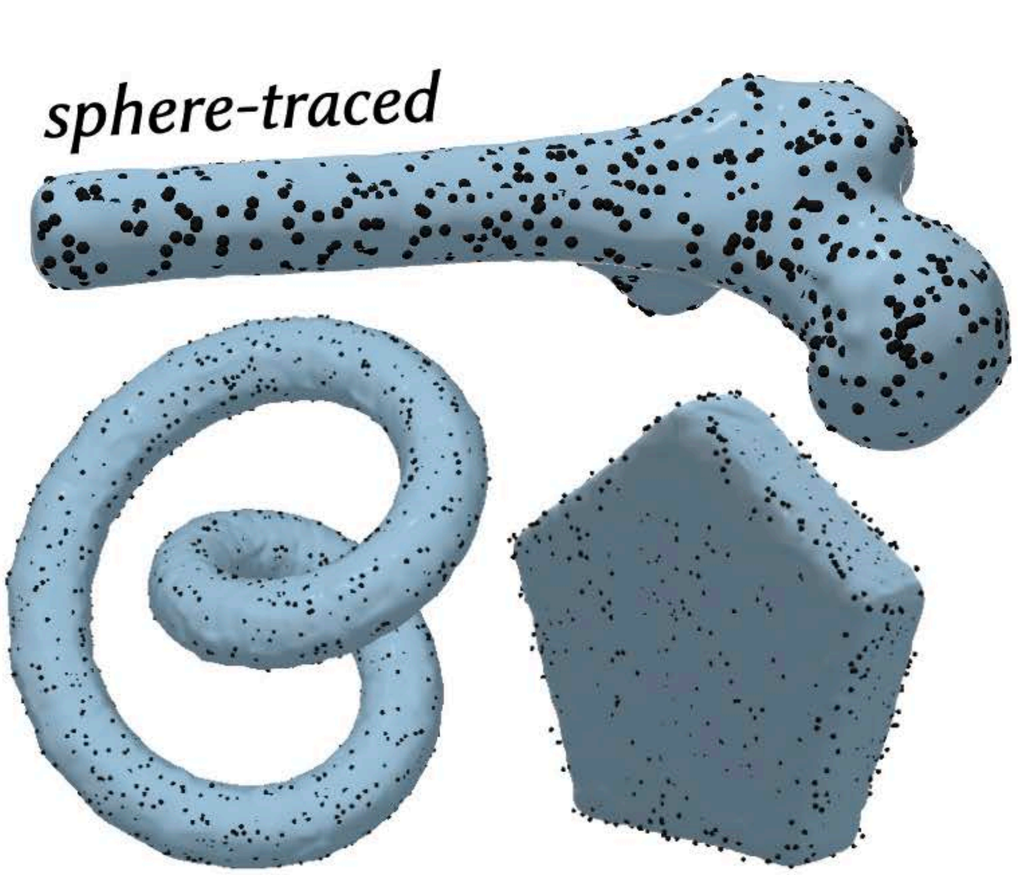
# Better than winding number, and almost as fast



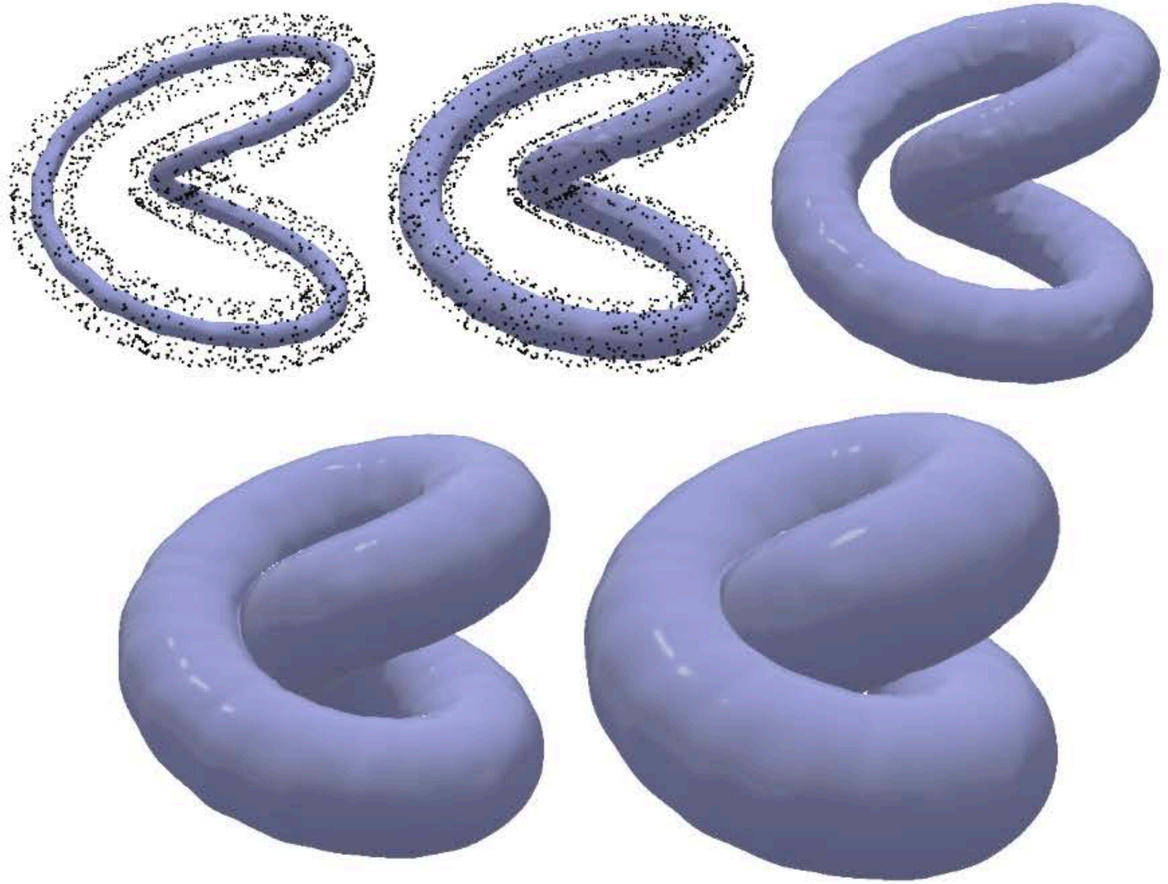
# Better than winding number, and almost as fast



# Real-time interaction

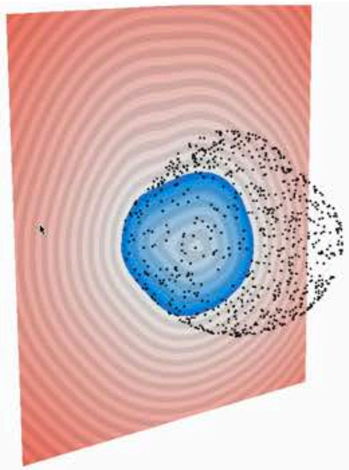


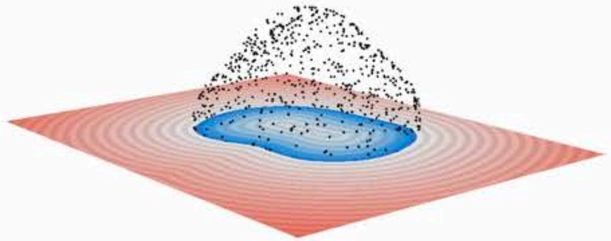

zero level set

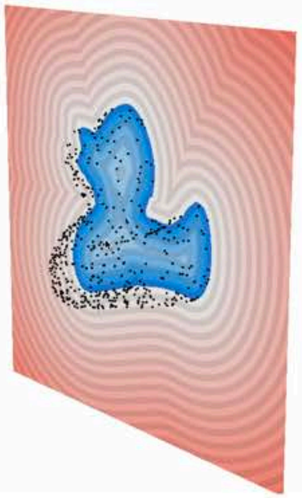


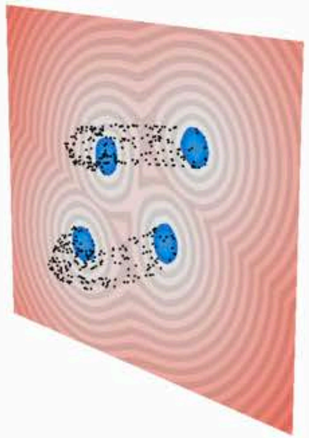
sphere-traced offset surfaces

# Real-time interaction

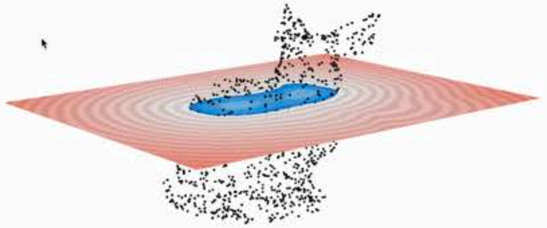




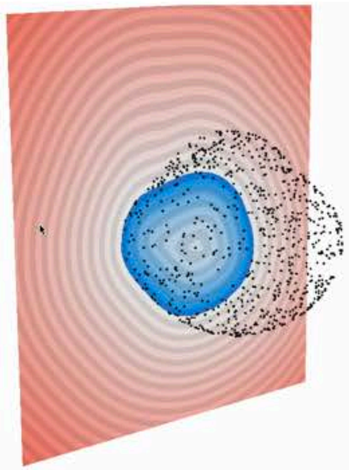


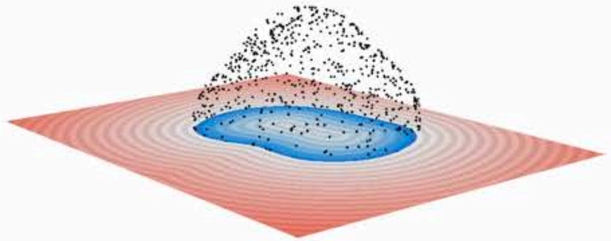


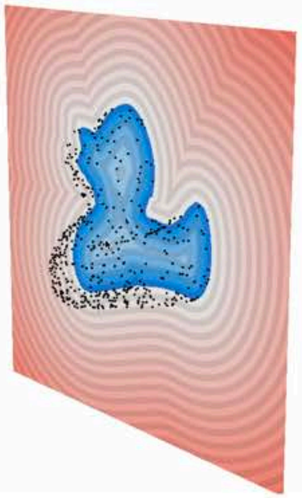


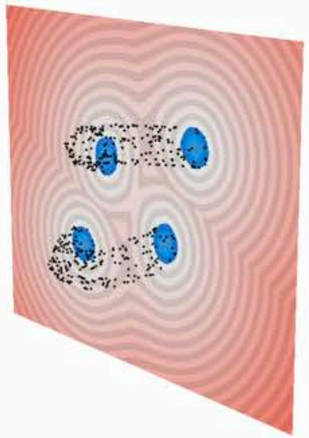


# Real-time interaction

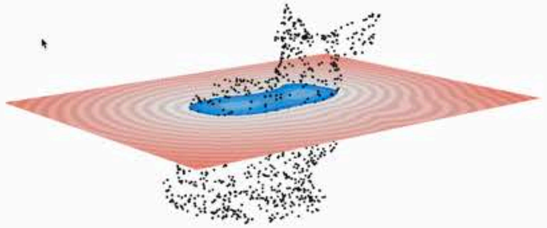




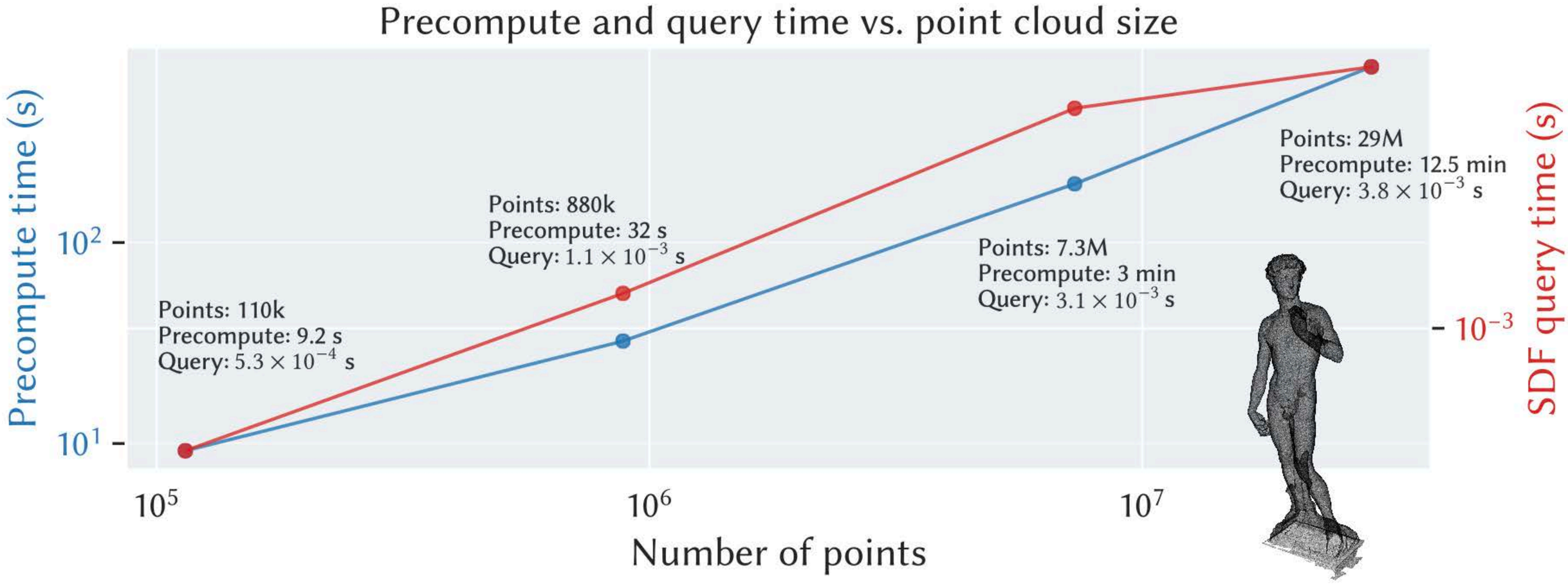




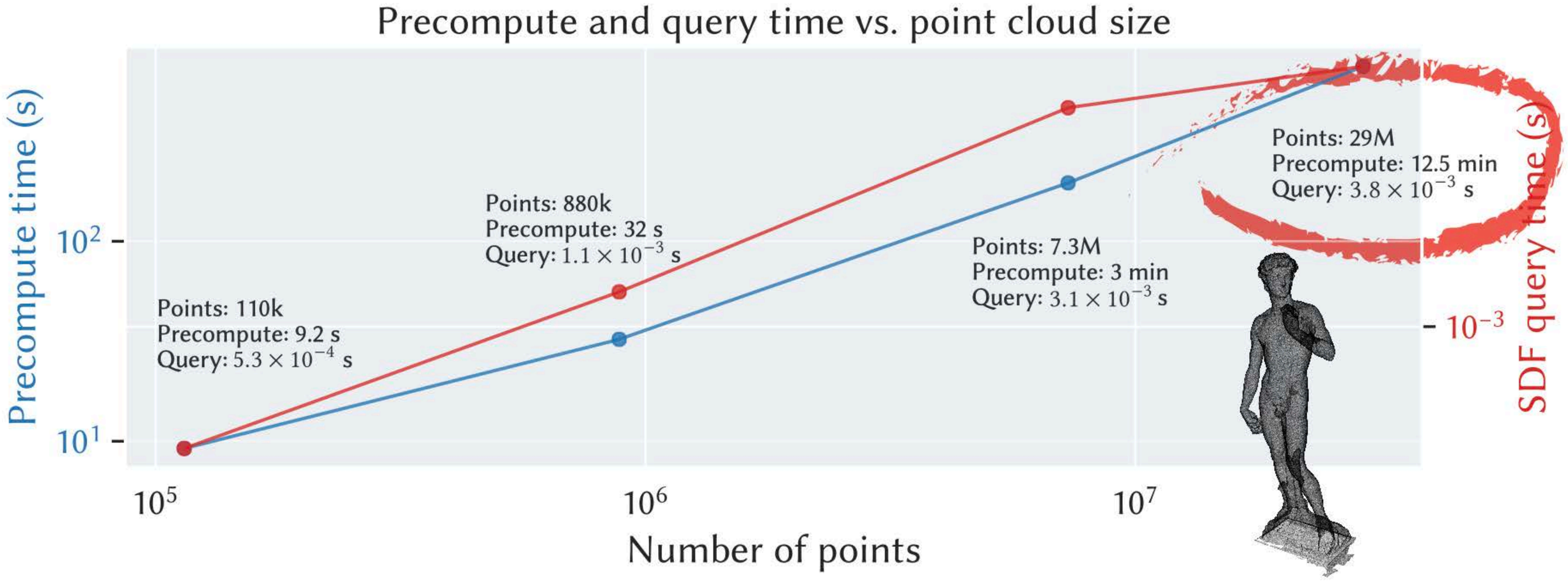




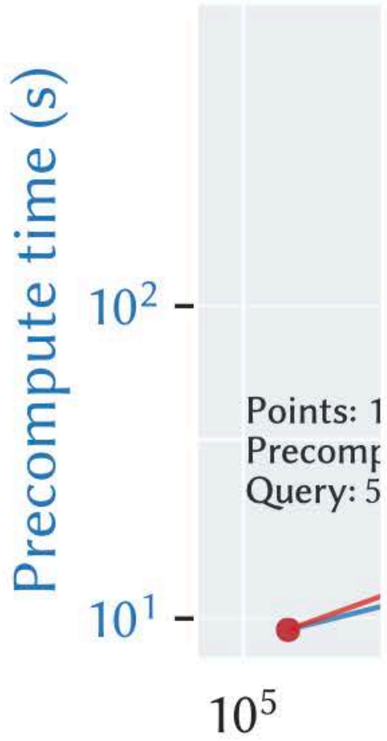
# Performance

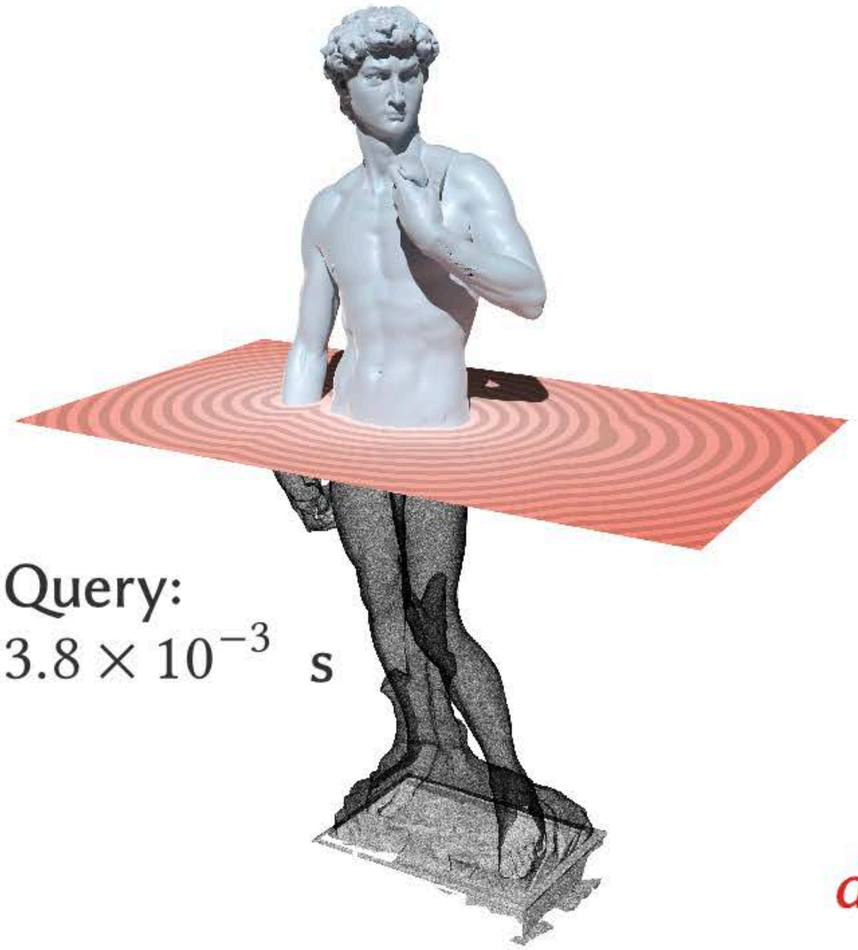


# Performance

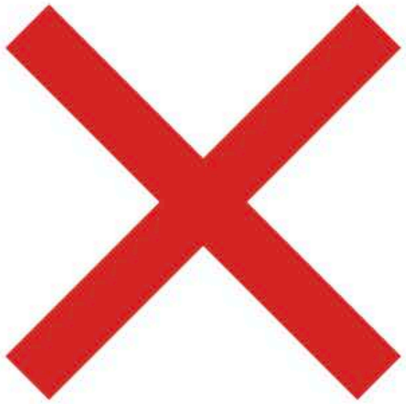


# Performance

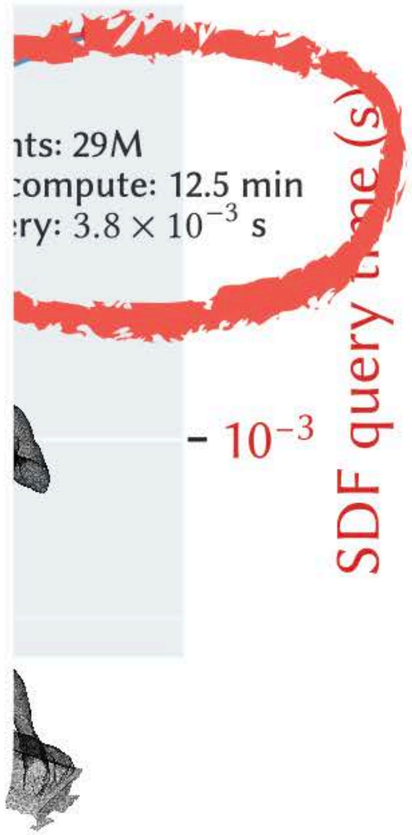



signed heat method



*failed after 2 hours*
*due to linear solver error*



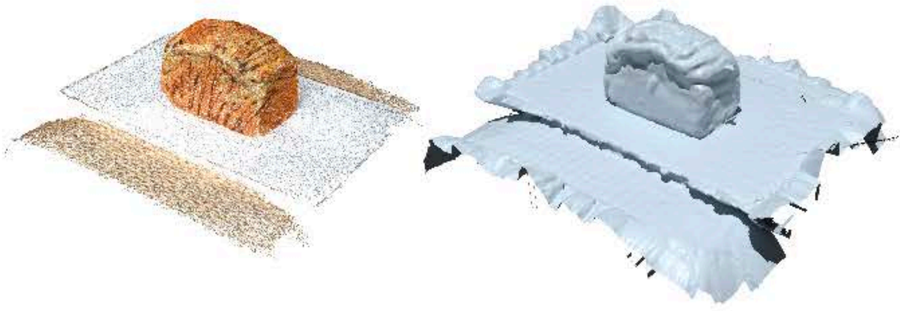
# Examples with noise and varying density



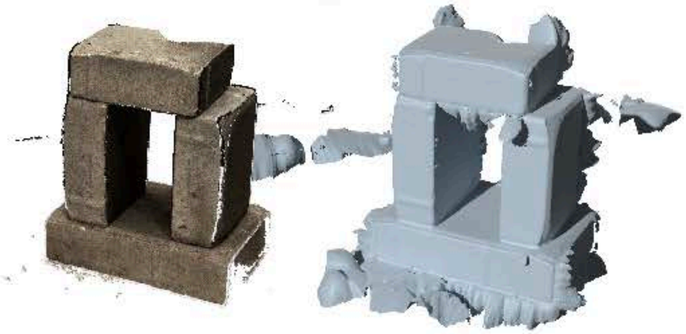

Points: 174k
Precompute: 14.7 s
Query: $5.9 \times 10^{-4}$ s



Points: 261k
Precompute: 16.4 s
Query: $3.0 \times 10^{-4}$ s



Points: 775k
Precompute: 34.9 s
Query: $6.4 \times 10^{-4}$ s



# Reconstruction quality

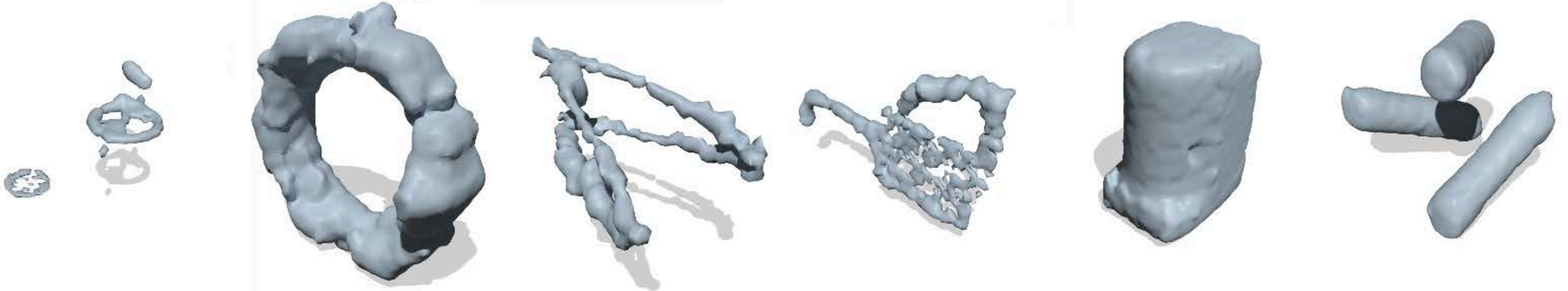
**true**
(512 points)



**signed heat**



**points as tori**



# Reconstruction quality

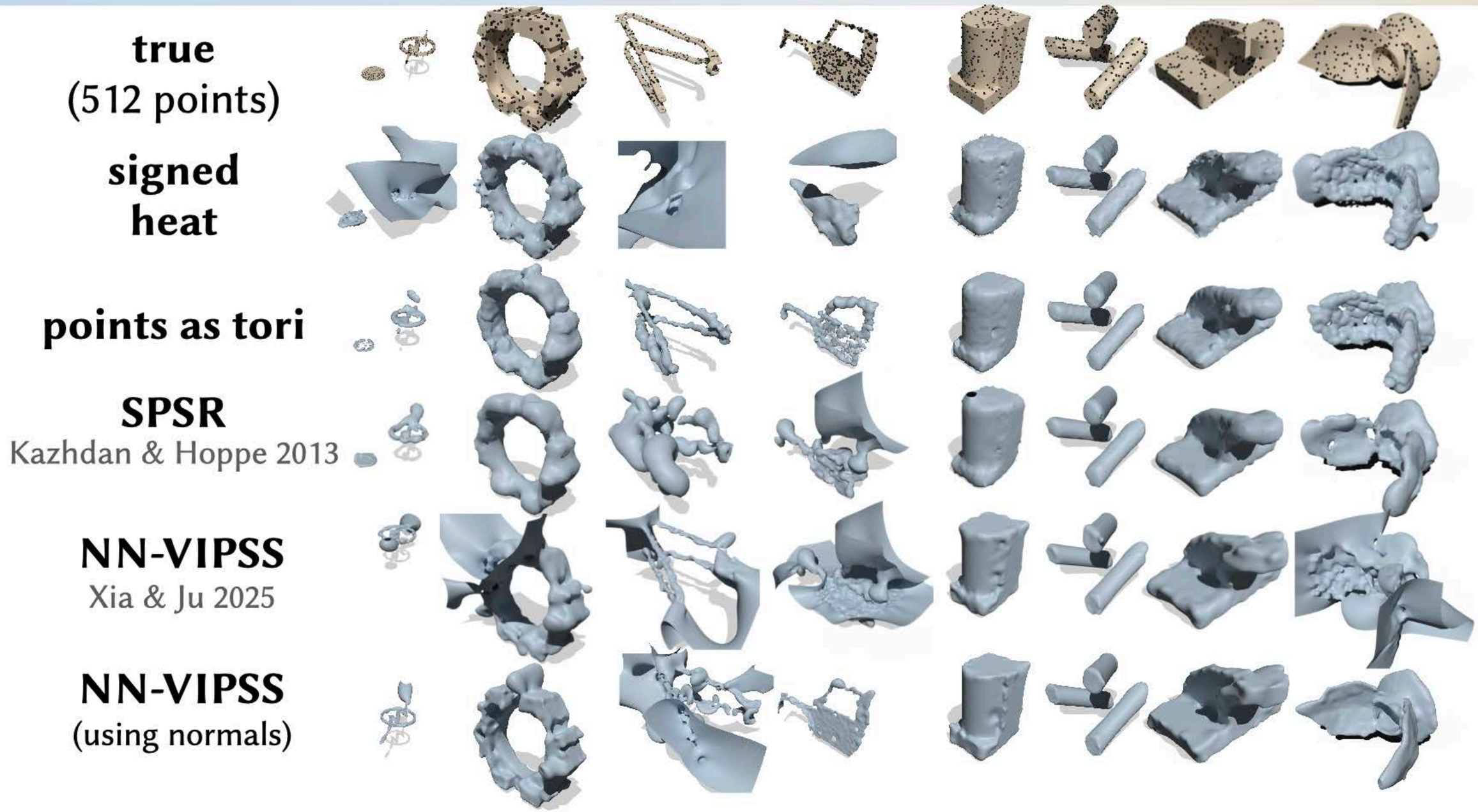
**true**
(512 points)

**signed heat**

**points as tori**



# Reconstruction quality



# Limitations

110k points



880k points



7.3M points



29M points



# Limitations

110k points



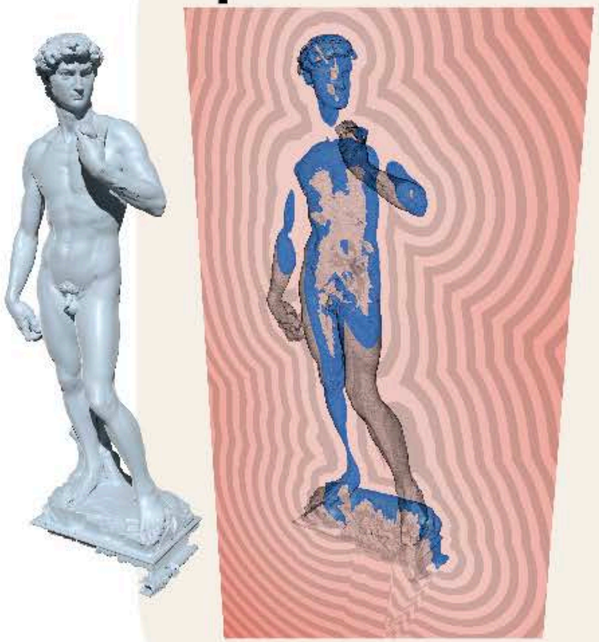
880k points



7.3M points



29M points



# Limitations

110k points



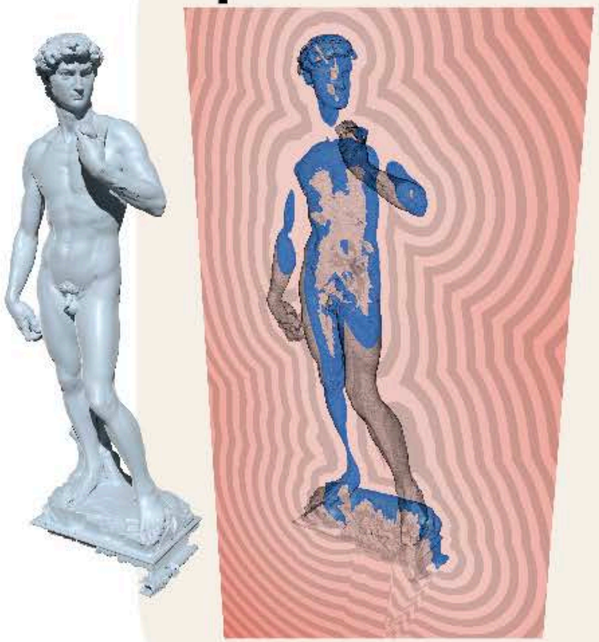
880k points



7.3M points



29M points



- Train on varying-density point clouds

# Limitations

110k points



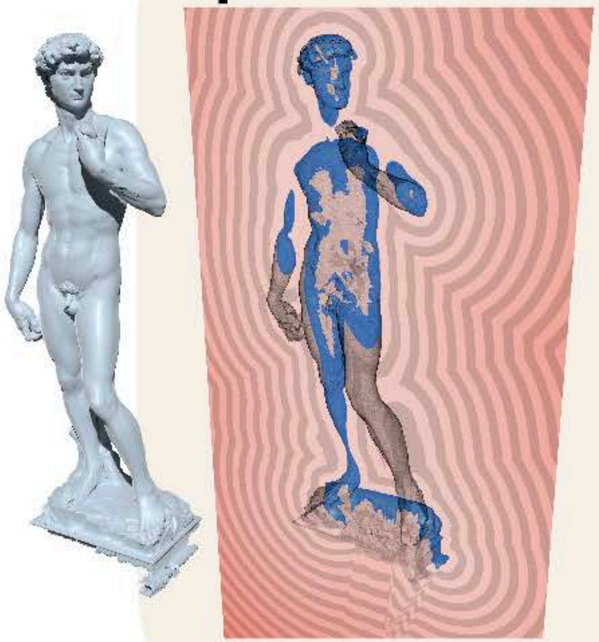
880k points



7.3M points



29M points



- Train on varying-density point clouds
- Neighborhood size estimation (learned or classical)

# Limitations

110k points



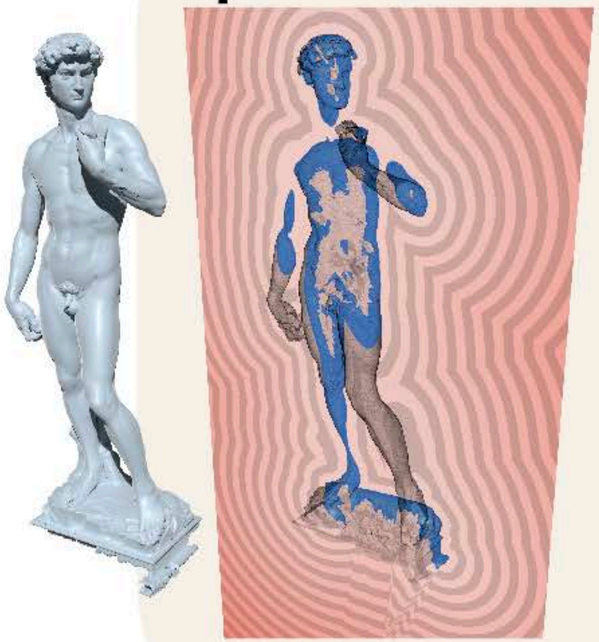
880k points



7.3M points



29M points



- Train on varying-density point clouds
- Neighborhood size estimation (learned or classical)
- Subsample point clouds

# An appeal for small, judicious use of learning

# An appeal for small, judicious use of learning

*"foundation models"* $\stackrel{!}{=}$ *general, re-usable, scalable*

# An appeal for small, judicious use of learning

*"foundation models"* $\stackrel{!}{=}$ *general, re-usable, scalable*

- We use a simple analytical formula for signed distance reconstruction.
  We blend tori SDFs, which are both expressive and efficient.

# An appeal for small, judicious use of learning

*"foundation models"* $\stackrel{!}{=}$ *general*, *re-usable*, *scalable*

- We use a simple analytical formula for signed distance reconstruction.
  We blend tori SDFs, which are both expressive and efficient.
- We pre-train a small neural network to learn local surface parameters.
  This network is trained **once** and can be applied to **any** shape.

# An appeal for small, judicious use of learning

*"foundation models"* $\stackrel{!}{=}$ *general*, *re-usable*, *scalable*

- We use a simple analytical formula for signed distance reconstruction.
  We blend tori SDFs, which are both expressive and efficient.
- We pre-train a small neural network to learn local surface parameters.
  This network is trained **once** and can be applied to **any** shape.
- Fast inference, embarrassingly parallel, output-sensitive, infinite resolution
  $10^{-4}$-$10^{-3}$ seconds for a single query to point clouds with millions of points.

# An appeal for small, judicious use of learning

*"foundation models"* $\stackrel{!}{=}$ *general*, *re-usable*, *scalable*

*Transferable foundation models for geometric tasks on point cloud representations: geometric neural operators*
[Quackenbush & Atzberger 2025]

*Defining Foundation Models for Computational Science: A Call for Clarity and Rigor*
[Choi et al. 2025]

...

- We use a simple analytical formula for signed distance reconstruction.
  We blend tori SDFs, which are both expressive and efficient.
- We pre-train a small neural network to learn local surface parameters.
  This network is trained **once** and can be applied to **any** shape.
- Fast inference, embarrassingly parallel, output-sensitive, infinite resolution
  $10^{-4}$-$10^{-3}$ seconds for a single query to point clouds with millions of points.

# An appeal for small, judicious use of learning

*"foundation models"* $\stackrel{!}{=}$ *general*, *re-usable*, *scalable*

*Transferable foundation models for geometric tasks on point cloud representations: geometric neural operators*
[Quackenbush & Atzberger 2025]

*Defining Foundation Models for Computational Science: A Call for Clarity and Rigor*
[Choi et al. 2025]

...

Use data to gain representational power; use classical methods get good solutions.

- We use a simple analytical formula for signed distance reconstruction.
  We blend tori SDFs, which are both expressive and efficient.
- We pre-train a small neural network to learn local surface parameters.
  This network is trained **once** and can be applied to **any** shape.
- Fast inference, embarrassingly parallel, output-sensitive, infinite resolution
  $10^{-4}$-$10^{-3}$ seconds for a single query to point clouds with millions of points.

# Connection to generative models

# Connection to generative models

*Goal:* Sample from a target distribution.

# Connection to generative models

*Goal:* Sample from a target distribution.

*One approach:* Sample from an easy-to-sample *source distribution*, then apply a transformation.

$\rho_0$ — *transform samples* → $\rho_1$

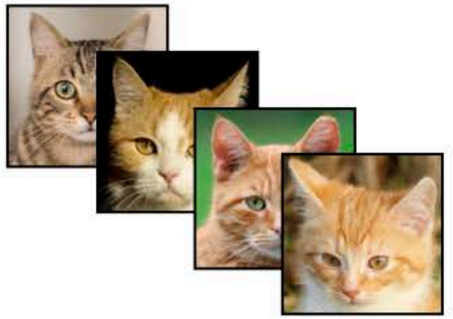
$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t}z + \left(\frac{1}{t} - 1\right)\nabla \log \rho(t, z) \quad t \in (0, 1]$$

source distribution

Song et al. 2021, Liu et al. 2022, etc.

target distribution

Ex: (“images of cats”)



# Connection to generative models

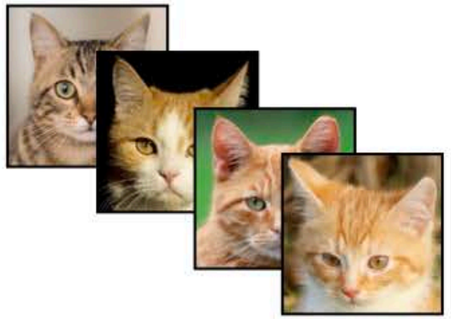
$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \quad t \in (0, 1]$$

source distribution

target distribution

Ex: (“images of cats”)



training data $\{x_i\}_{i=1}^N$

# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

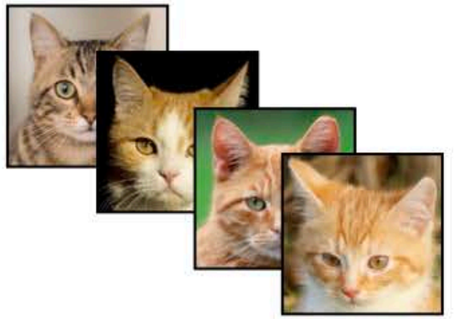
$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

source distribution

target distribution

Ex: (“images of cats”)

$$\nabla \log \rho(t, z) \propto \frac{\sum_{i=1}^{N} t x_i \exp\left(-\frac{1}{2} \frac{\|z - t x_i\|^2}{(1-t)^2}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{1}{2} \frac{\|z - t x_j\|^2}{(1-t)^2}\right)} - z$$



training data $\{x_i\}_{i=1}^{N}$

*Flow Straight and Fast*

[Liu et al. 2021]

*Closed-form Diffusion Models*

[Scarvelis et al. 2023]

# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

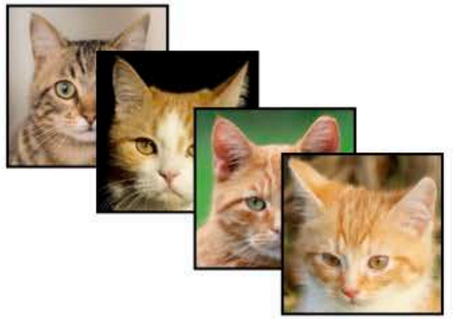
source distribution

target distribution

Ex: (“images of cats”)

$$\nabla \log \rho(t, z) \propto \frac{\sum_{i=1}^{N} t x_i \exp\left(-\frac{1}{2} \frac{\|z - t x_i\|^2}{(1-t)^2}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{1}{2} \frac{\|z - t x_j\|^2}{(1-t)^2}\right)} - z$$

*current sample*



training data $\{x_i\}_{i=1}^{N}$

*Flow Straight and Fast*

[Liu et al. 2021]

*Closed-form Diffusion Models*

[Scarvelis et al. 2023]

# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

source distribution

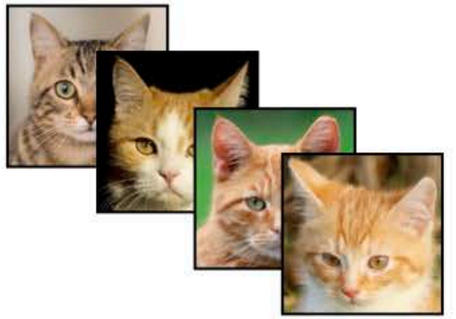
target distribution

Ex: (“images of cats”)

$$\nabla \log \rho(t, z) \propto \frac{\sum_{i=1}^{N} t x_i \exp\left(-\frac{1}{2} \frac{\|z - t x_i\|^2}{(1-t)^2}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{1}{2} \frac{\|z - t x_j\|^2}{(1-t)^2}\right)} - z$$

*current sample*

*approximation of* $\operatorname{argmin}_{t x_i} \|z - t x_i\|^2$



training data $\{x_i\}_{i=1}^{N}$

*Flow Straight and Fast*

[Liu et al. 2021]

*Closed-form Diffusion Models*

[Scarvelis et al. 2023]

# Connection to generative models

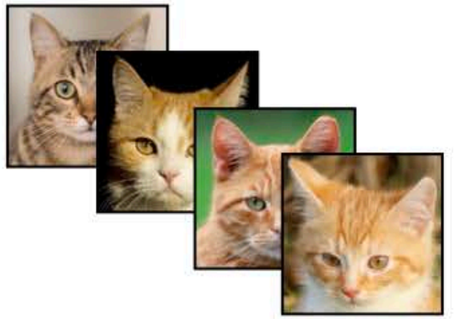
$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

source distribution

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

target distribution

Ex: (“images of cats”)



training data $\{x_i\}_{i=1}^N$

$$\nabla \log \rho(t, z) \propto \frac{\sum_{i=1}^N t x_i \exp\left(-\frac{1}{2} \frac{\|z - t x_i\|^2}{(1-t)^2}\right)}{\sum_{j=1}^N \exp\left(-\frac{1}{2} \frac{\|z - t x_j\|^2}{(1-t)^2}\right)} - z$$

*current sample*

*approximation of* $\operatorname{argmin}_{t x_i} \|z - t x_i\|^2$

*Flow Straight and Fast*

[Liu et al. 2021]

*Closed-form Diffusion Models*

[Scarvelis et al. 2023]

# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

source distribution

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

*pushes sample z towards closest training point*

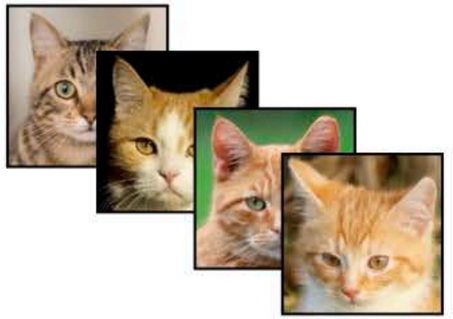
target distribution

Ex: ("images of cats")

$$\nabla \log \rho(t, z) \propto \frac{\sum_{i=1}^{N} t x_i \exp\left(-\frac{1}{2} \frac{\|z - t x_i\|^2}{(1-t)^2}\right)}{\sum_{j=1}^{N} \exp\left(-\frac{1}{2} \frac{\|z - t x_j\|^2}{(1-t)^2}\right)} - z$$

*current sample*

*approximation of* $\operatorname{argmin}_{t x_i} \|z - t x_i\|^2$



training data $\{x_i\}_{i=1}^{N}$

*Flow Straight and Fast*

[Liu et al. 2021]

*Closed-form Diffusion Models*

[Scarvelis et al. 2023]

# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

source distribution

target distribution

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

*pushes sample z towards closest training point*

Flow-based models can only "memorize" their training data!

[Liu et al. 2021, Scarvelis et al. 2023, Gu et al. 2023, Gao & Li 2024, Carlini et al. 2023, Jain et al. 2024, Biroli et al. 2024, Gu et al. 2025, Somepalli et al. 2022; Pidstrigach 2022, Yoon et al. 2023, Kamb & Ganguli 2025]

# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

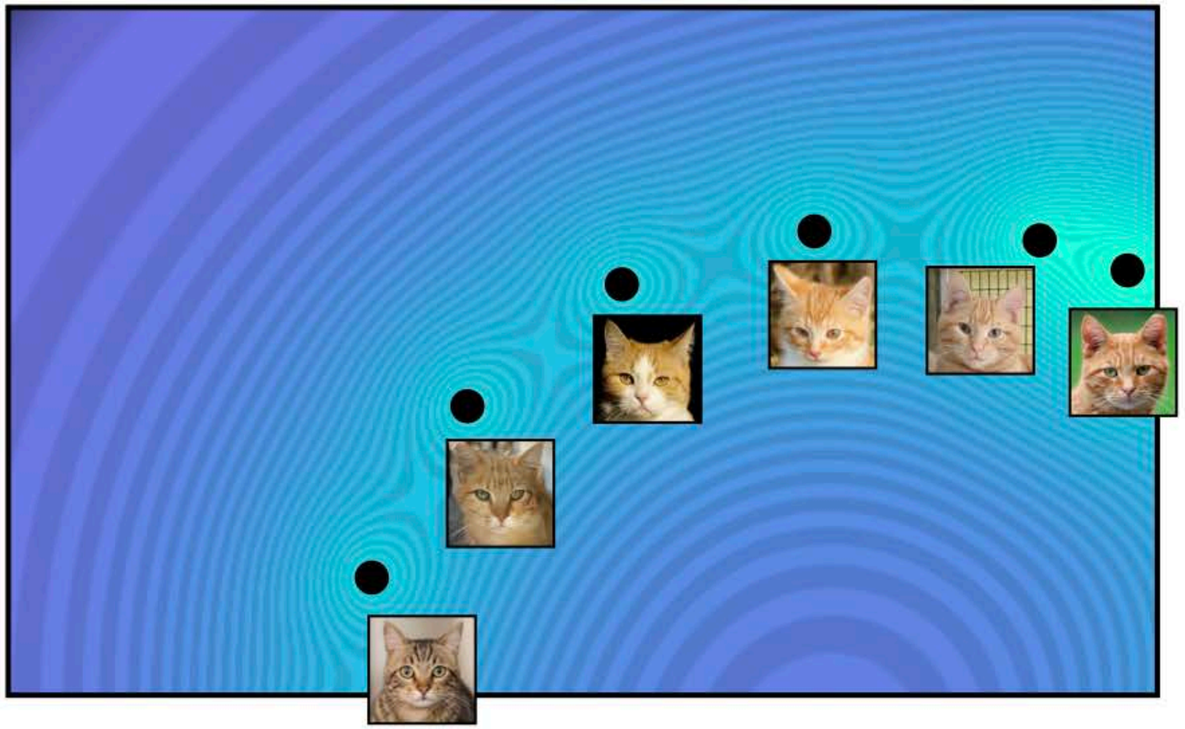
source distribution

target distribution

$$\frac{dz}{dt} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

*pushes sample z towards closest training point*

Flow-based models can only "memorize" their training data!



# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

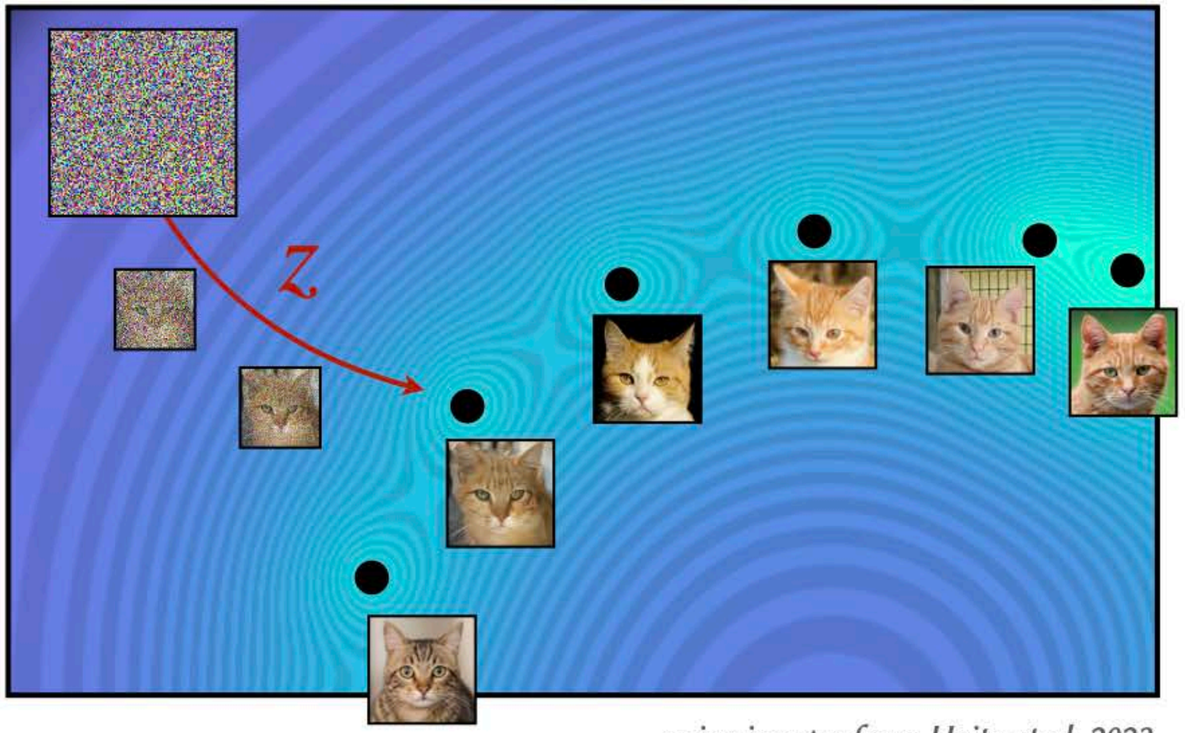
source distribution

target distribution

$$\frac{dz}{dt} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

*pushes sample z towards closest training point*

Flow-based models can only "memorize" their training data!




*noisy images from Heitz et al. 2023*

# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

source distribution

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

target distribution

*pushes sample z towards closest training point*

Flow-based models can only "memorize" their training data!






# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

source distribution

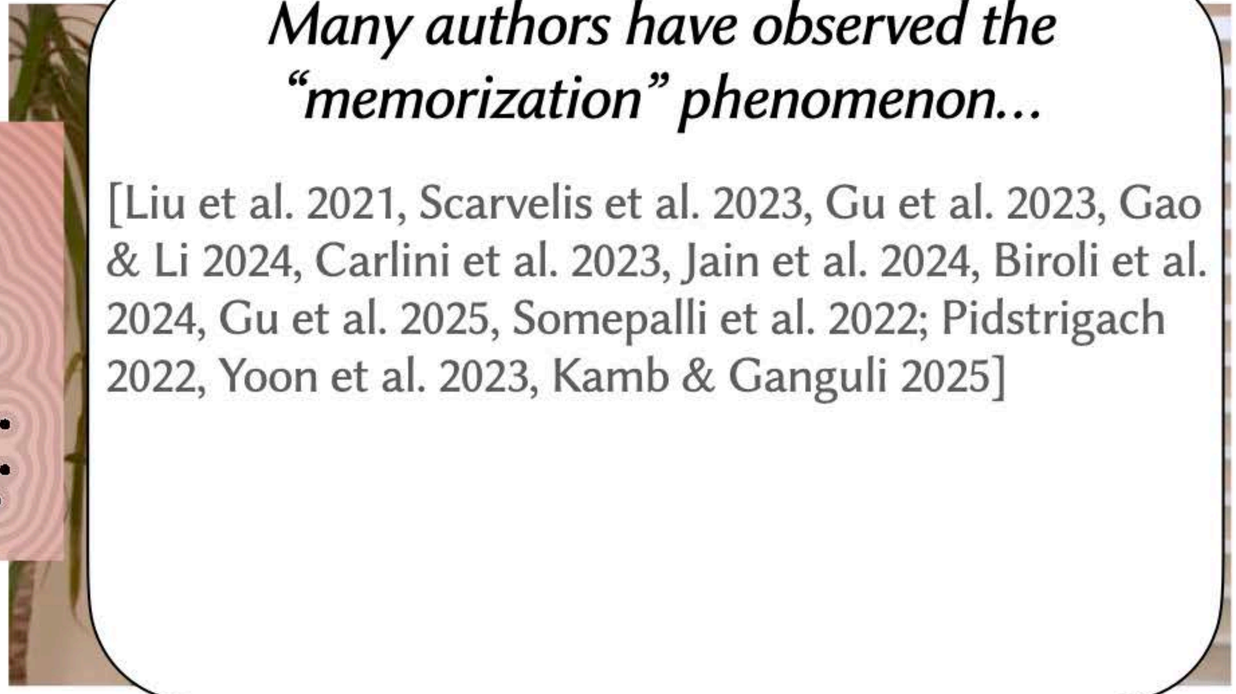
target distribution

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

*pushes sample z towards closest training point*

Diffusion models can only "memorize" their training data!



*Many authors have observed the "memorization" phenomenon…*

[Liu et al. 2021, Scarvelis et al. 2023, Gu et al. 2023, Gao & Li 2024, Carlini et al. 2023, Jain et al. 2024, Biroli et al. 2024, Gu et al. 2025, Somepalli et al. 2022; Pidstrigach 2022, Yoon et al. 2023, Kamb & Ganguli 2025]

# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

source distribution

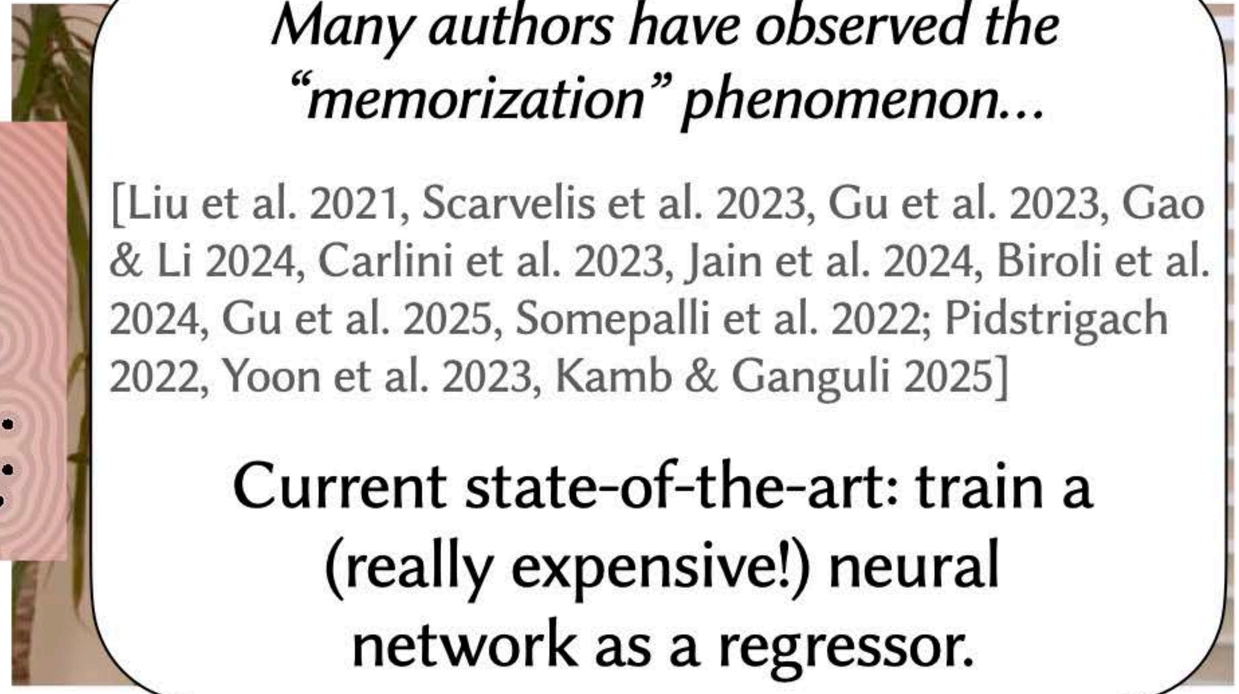
target distribution

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

*pushes sample z towards closest training point*

Diffusion models can only "memorize" their training data!



*Many authors have observed the "memorization" phenomenon…*

[Liu et al. 2021, Scarvelis et al. 2023, Gu et al. 2023, Gao & Li 2024, Carlini et al. 2023, Jain et al. 2024, Biroli et al. 2024, Gu et al. 2025, Somepalli et al. 2022; Pidstrigach 2022, Yoon et al. 2023, Kamb & Ganguli 2025]

Current state-of-the-art: train a (really expensive!) neural network as a regressor.

# Connection to generative models

$$\rho_0 = \mathcal{N}(0, I) \xrightarrow{\text{transform samples}} \rho_1$$

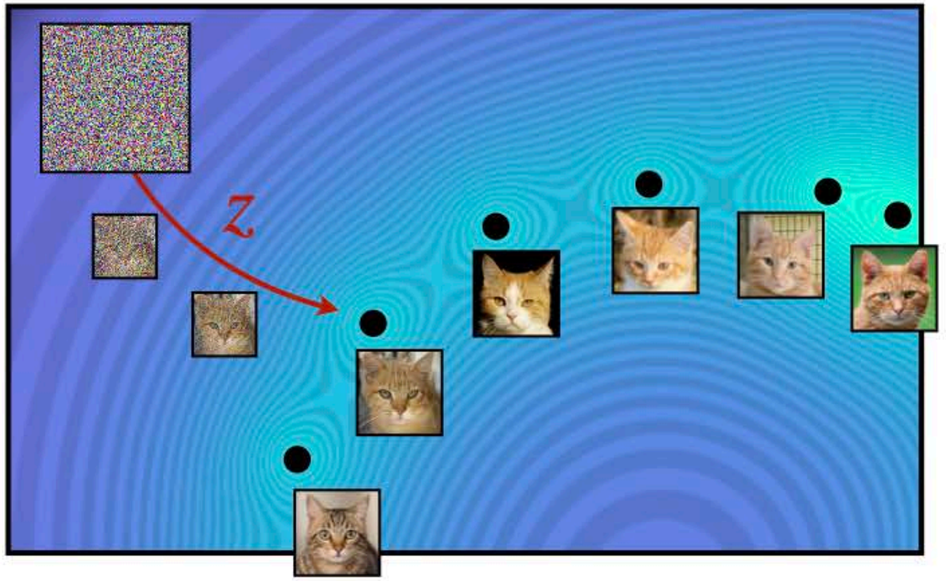
source distribution

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \frac{1}{t} z + \left(\frac{1}{t} - 1\right) \nabla \log \rho(t, z) \qquad t \in (0, 1]$$

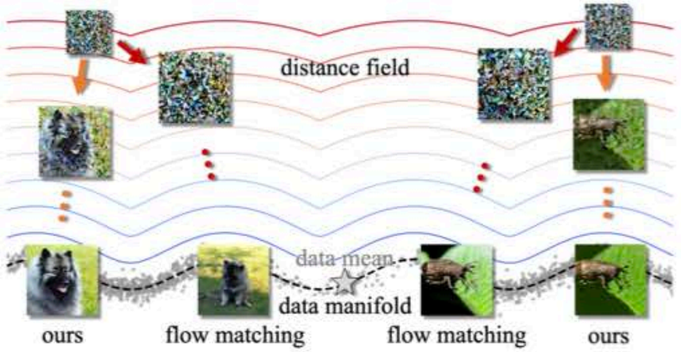
target distribution

*pushes sample z towards closest training point*

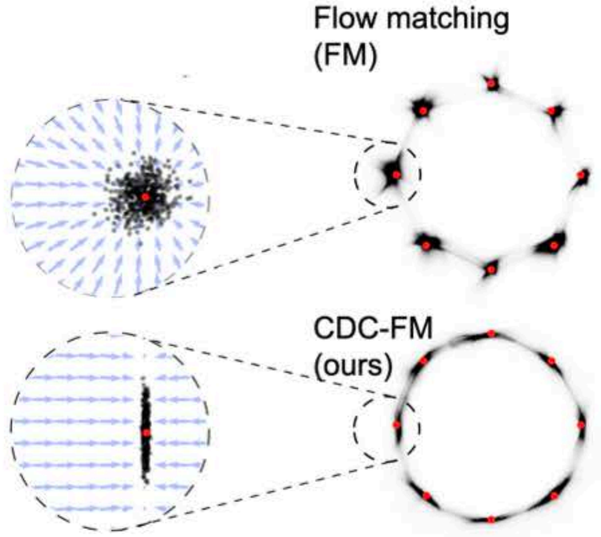
Diffusion models can only "memorize" their training data!




*Distance Marching for Generative Modeling*
[Wang et al. 2026]



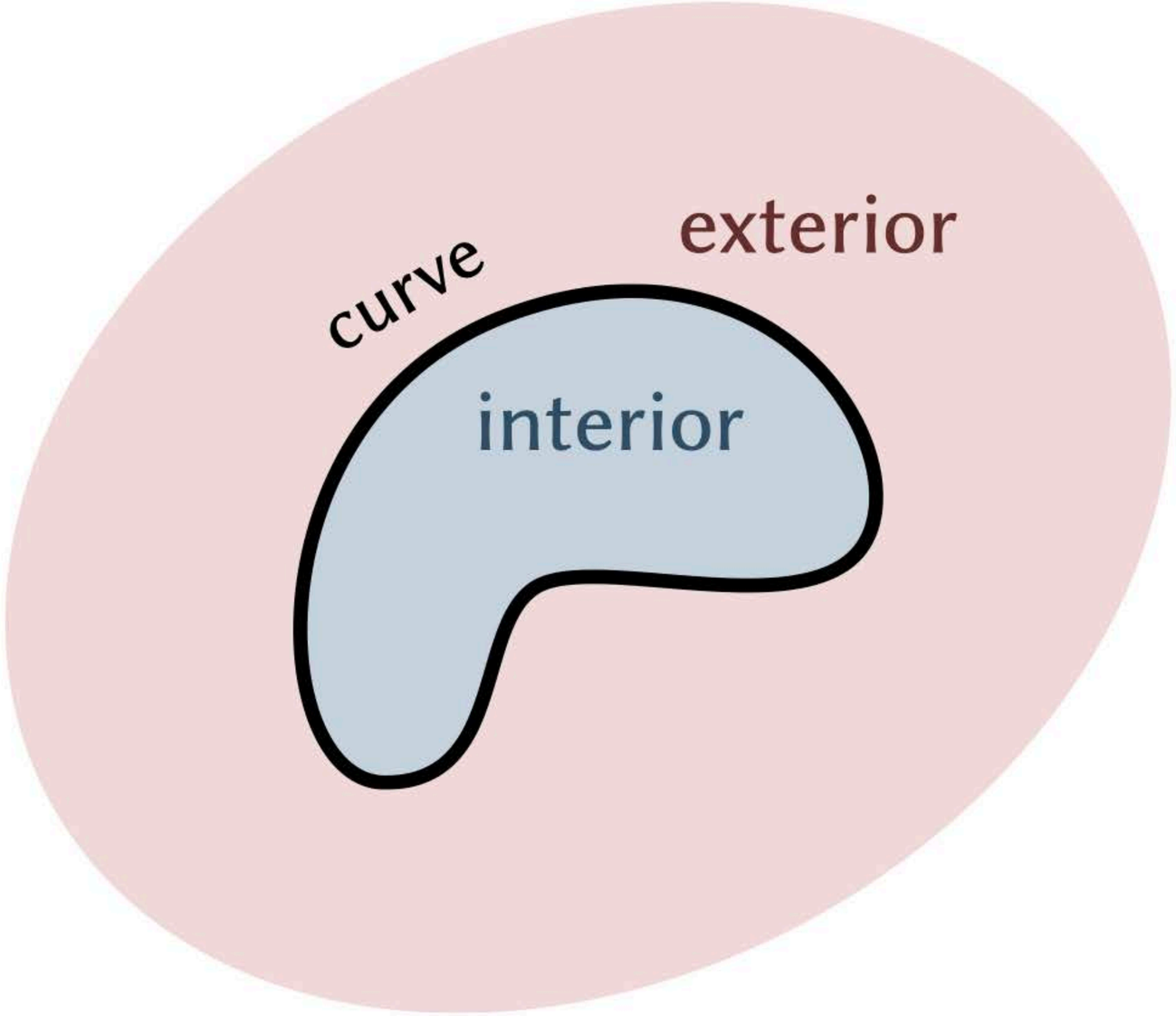

*Carré du champ Flow Matching: Better Quality-Generalisation Tradeoff in Generative Models*
[Bamberger et al. 2026]



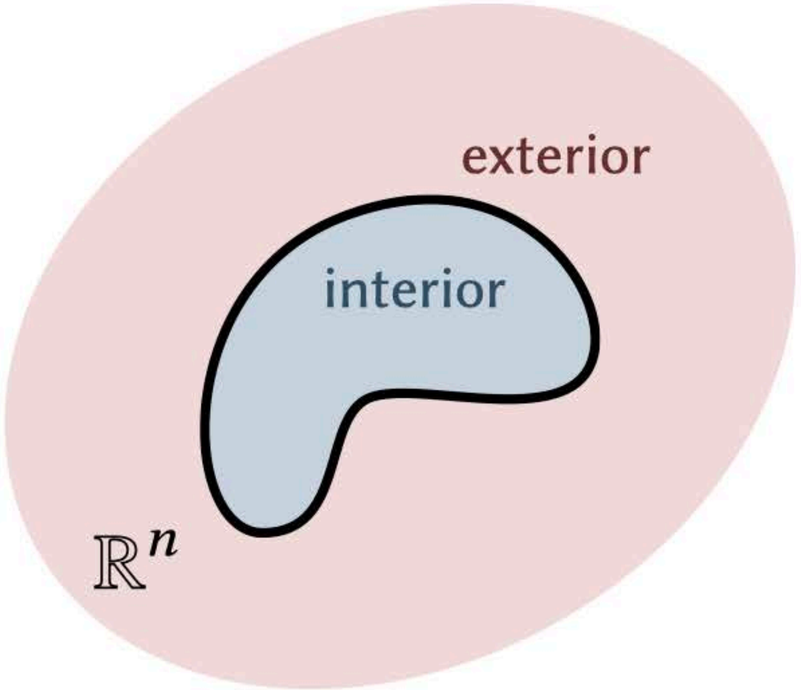
# GENERALIZED INSIDE/OUTSIDE

# The inside and outside of a shape, revisited

# The inside and outside of a shape, revisited



# Not all closed curves have an inside/outside




*bounding*



Fence bounds a region.

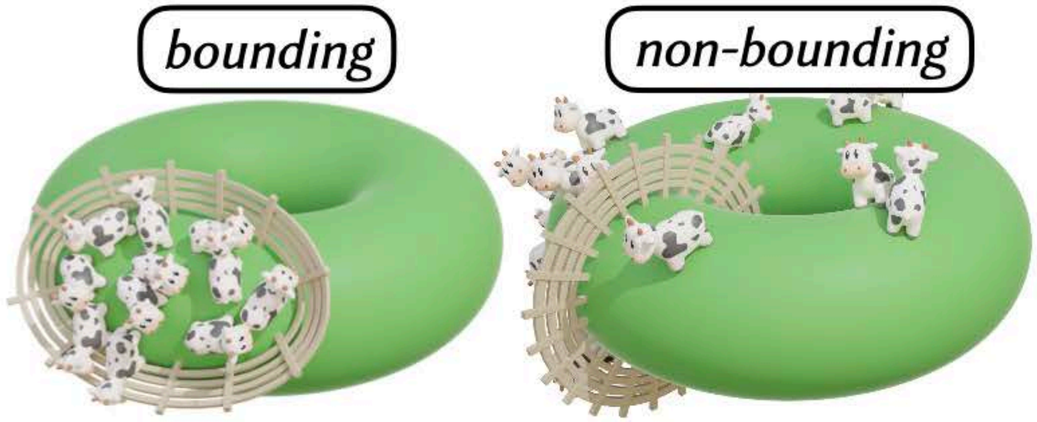
*non-bounding*



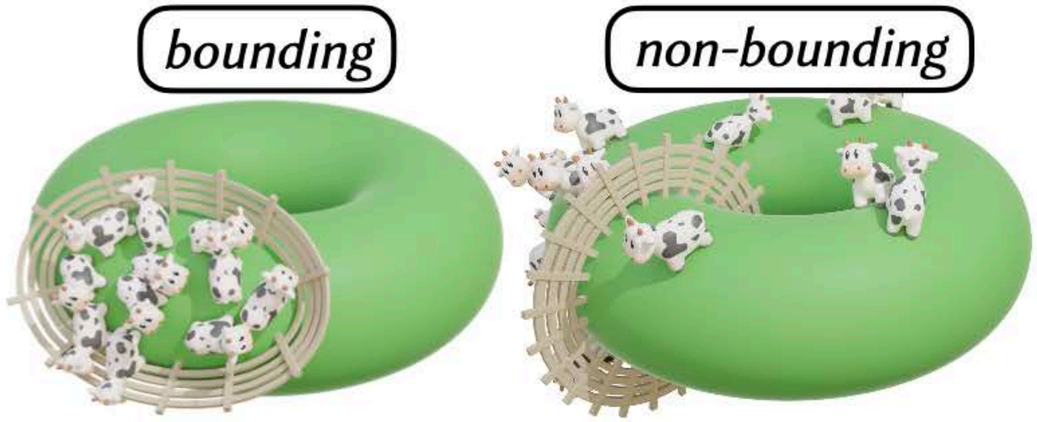
Fence doesn't bound a region.

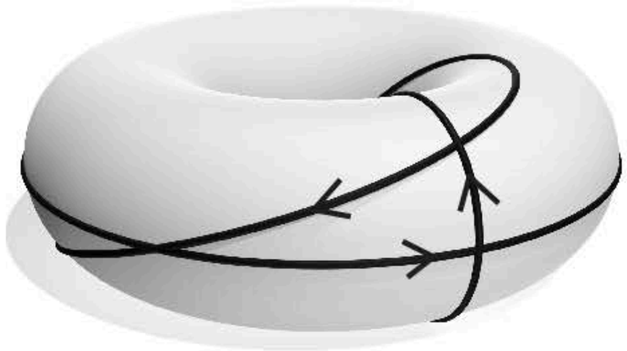
*Cow idea from Mark Gillespie*

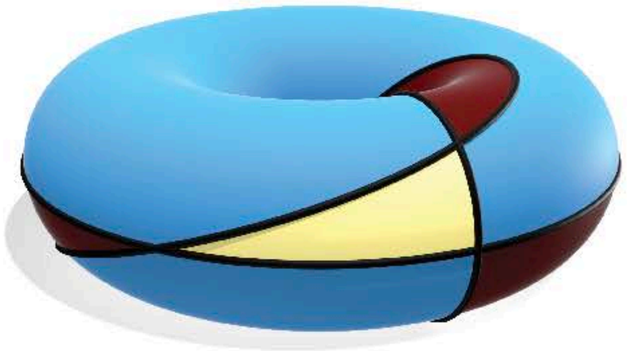
# Inside/outside can be complicated



# Inside/outside can be complicated





# Inside/outside can be complicated

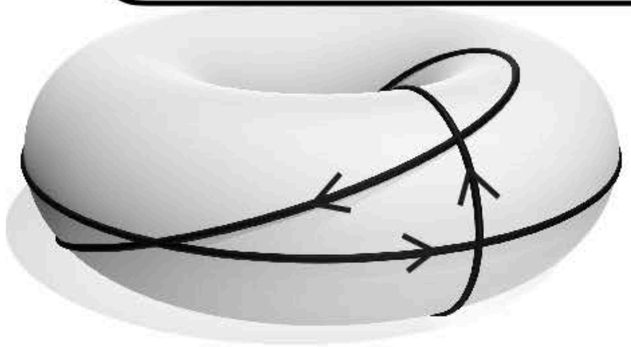
*bounding*

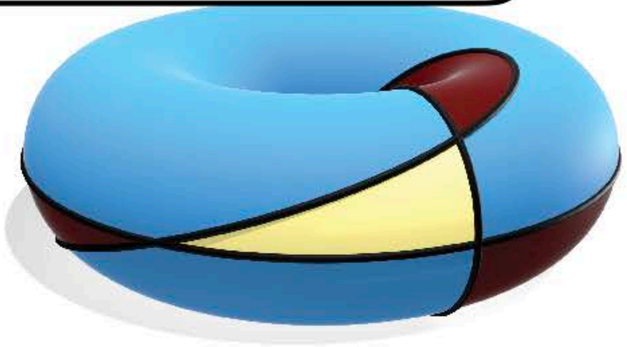


*non-bounding*



Can't just throw away non-bounding curves





# Inside/outside can be complicated

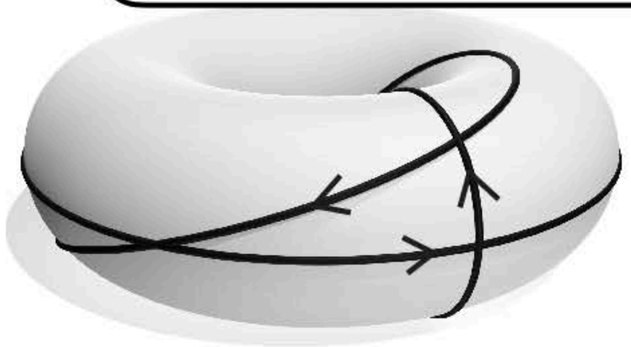
*bounding*

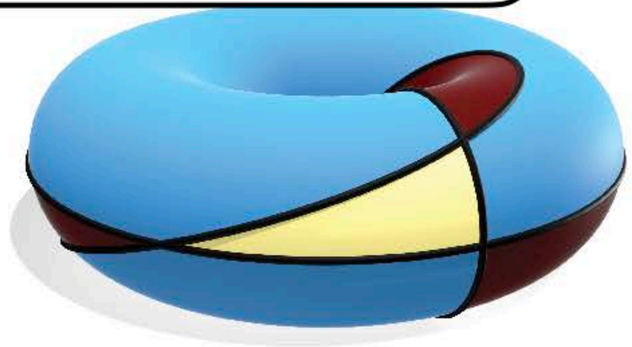


*non-bounding*

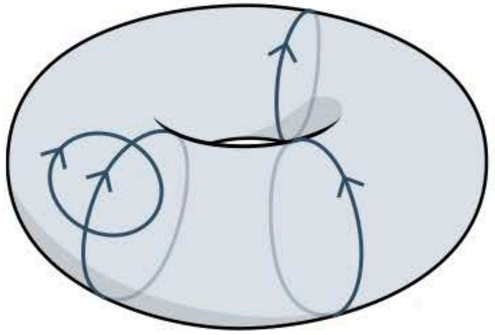


Can't just throw away non-bounding curves

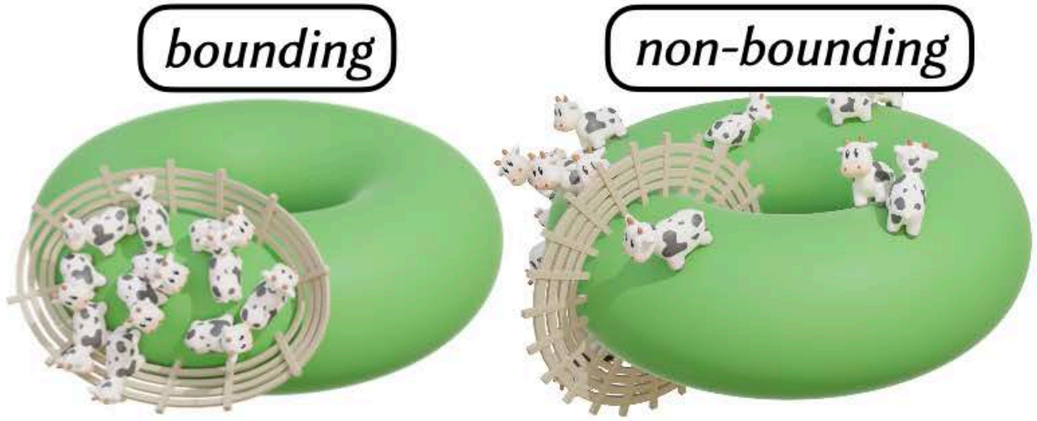




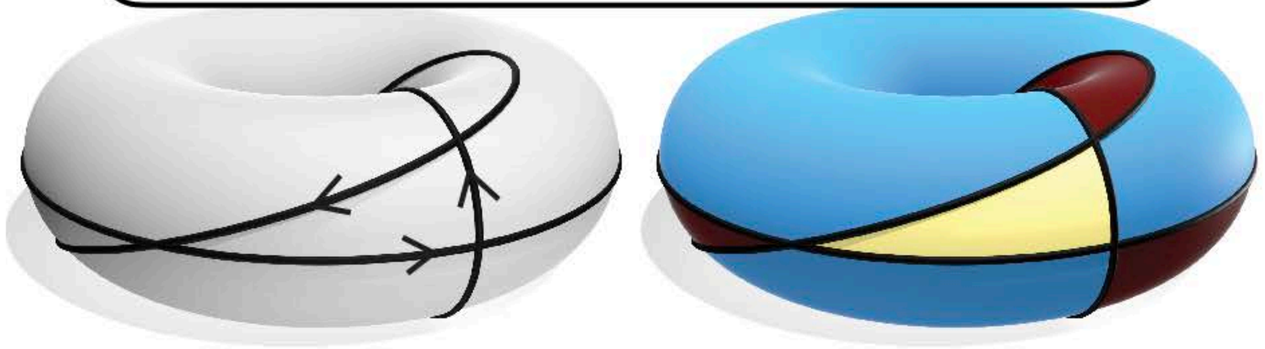
input



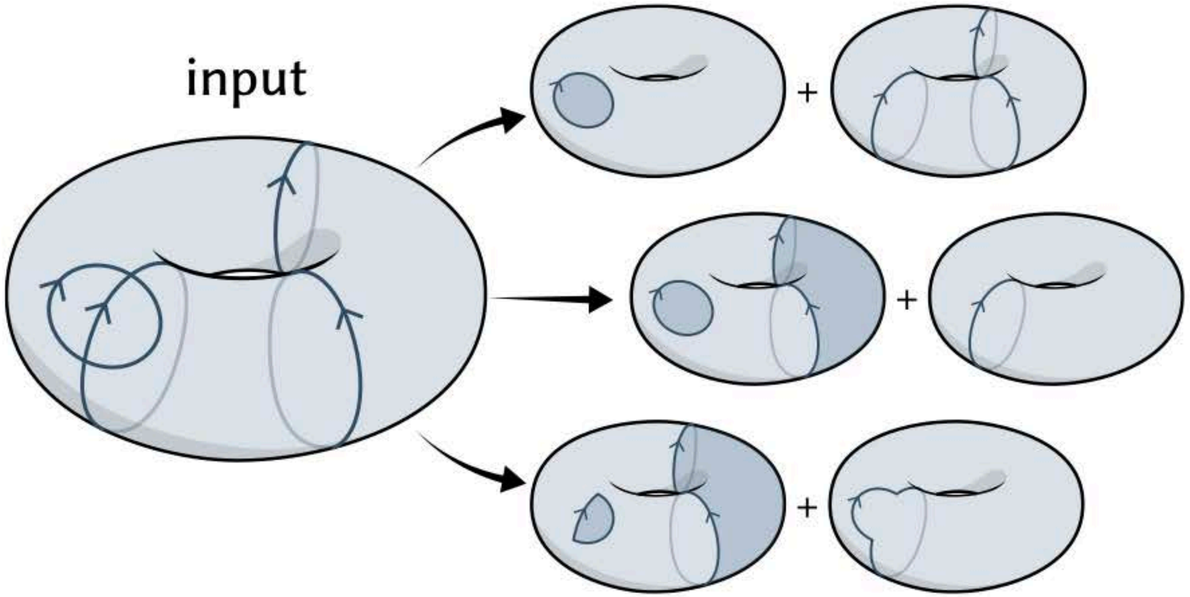
# Inside/outside can be complicated



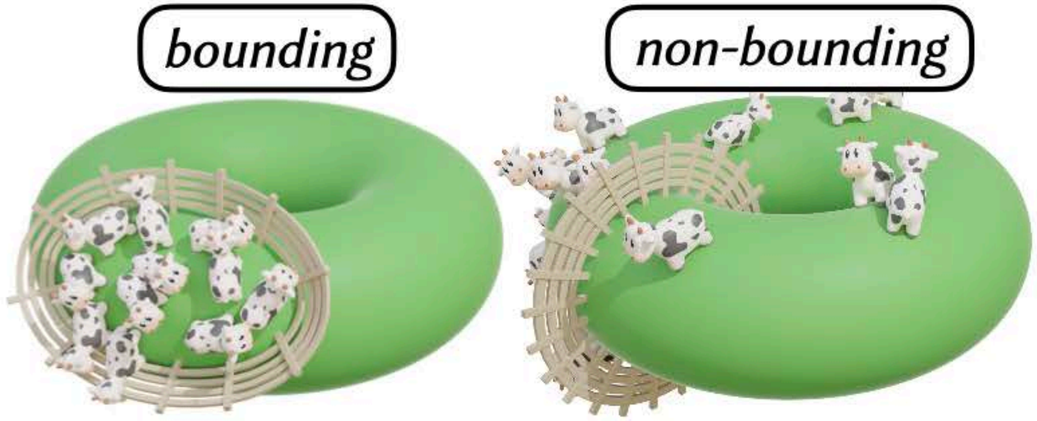

Can't just throw away non-bounding curves



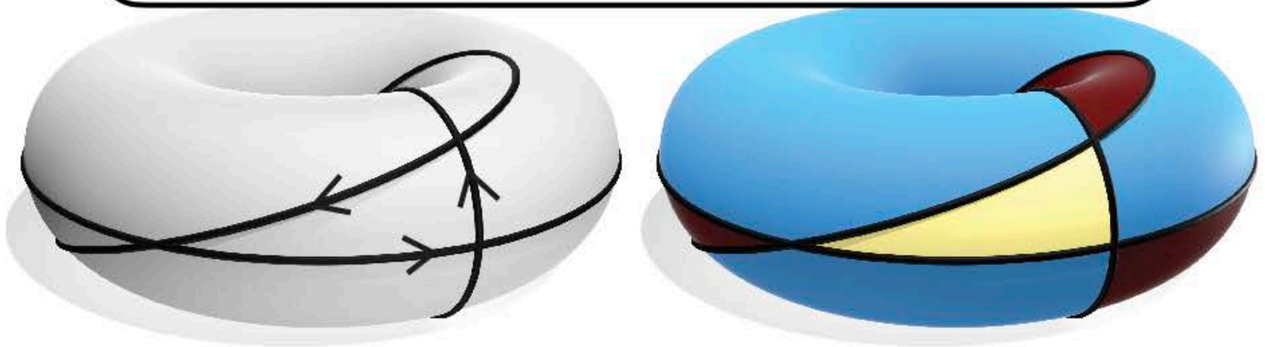
Many ways to partition input into regions



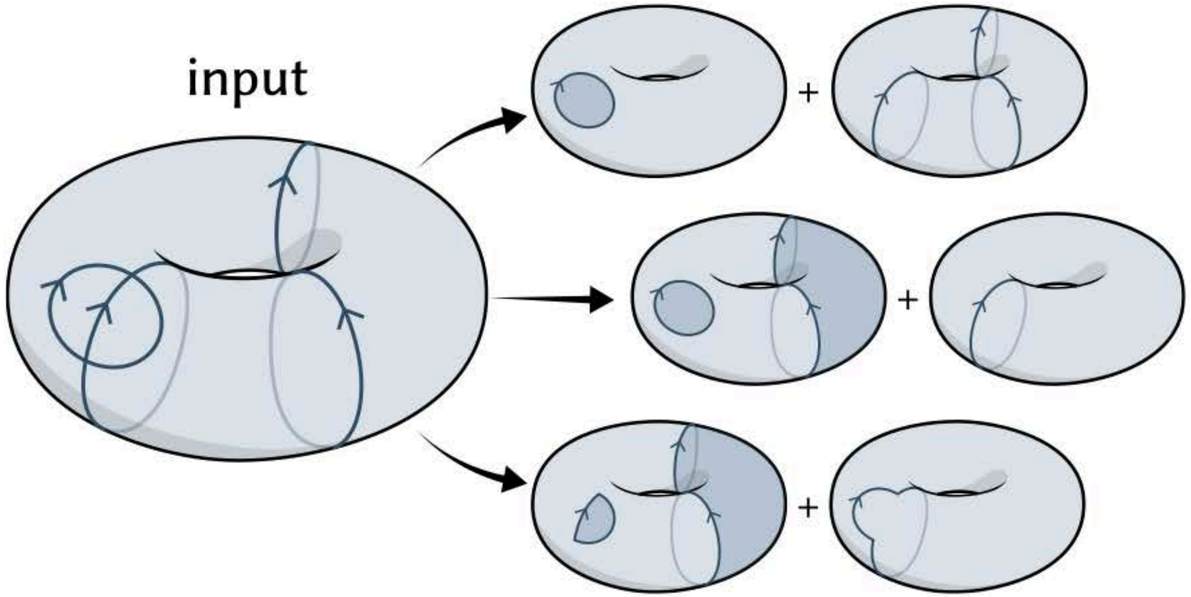
# Inside/outside can be complicated



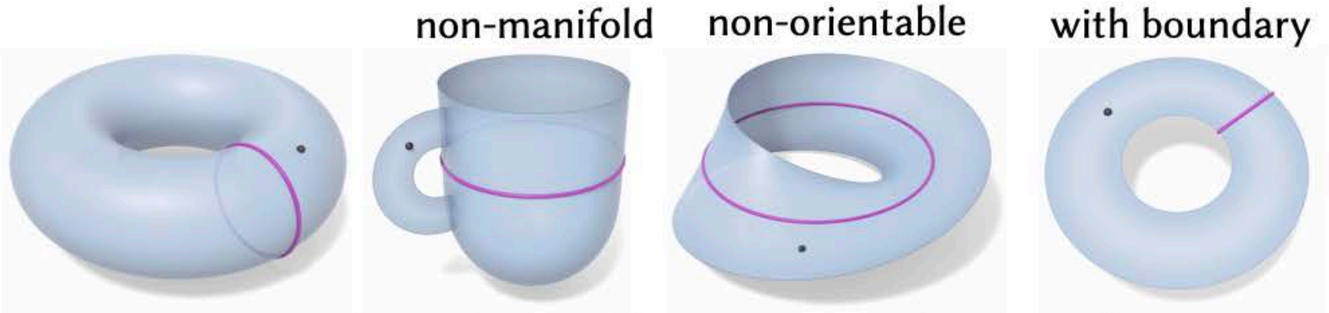

Can't just throw away non-bounding curves



Many ways to partition input into regions

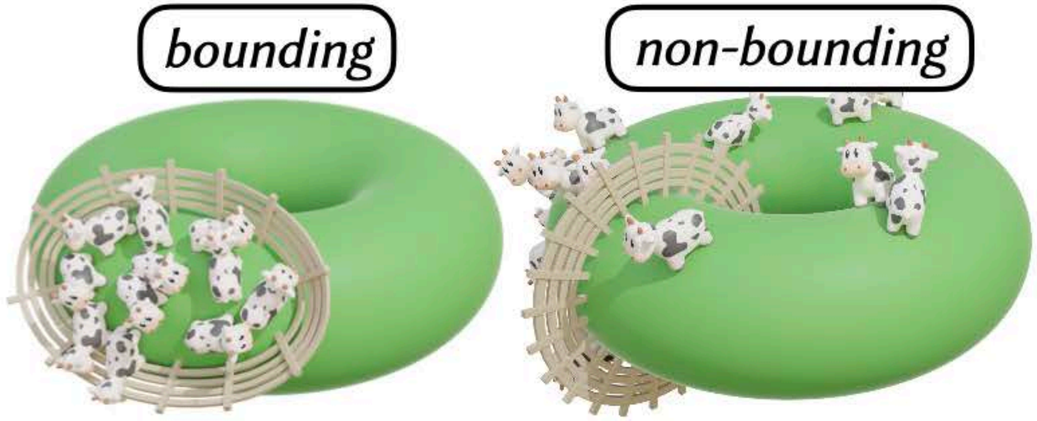



Other ambiguous cases...

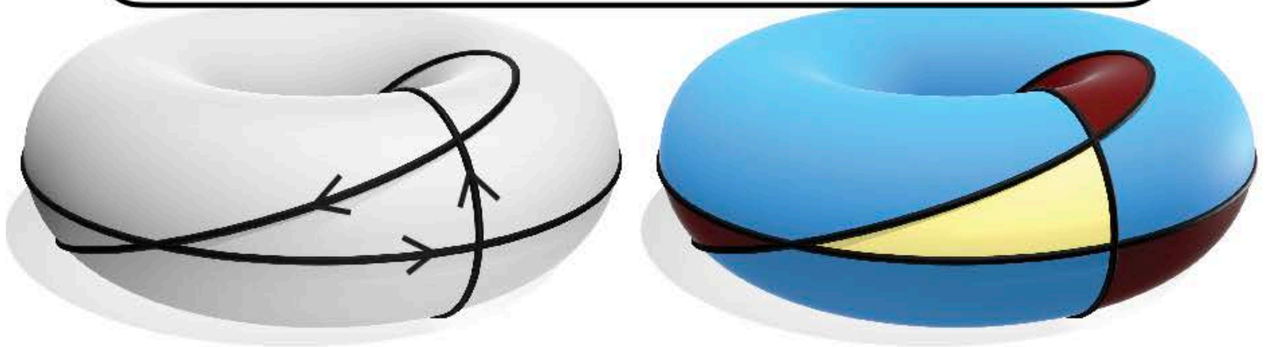



*Is the point inside or outside the curve?*

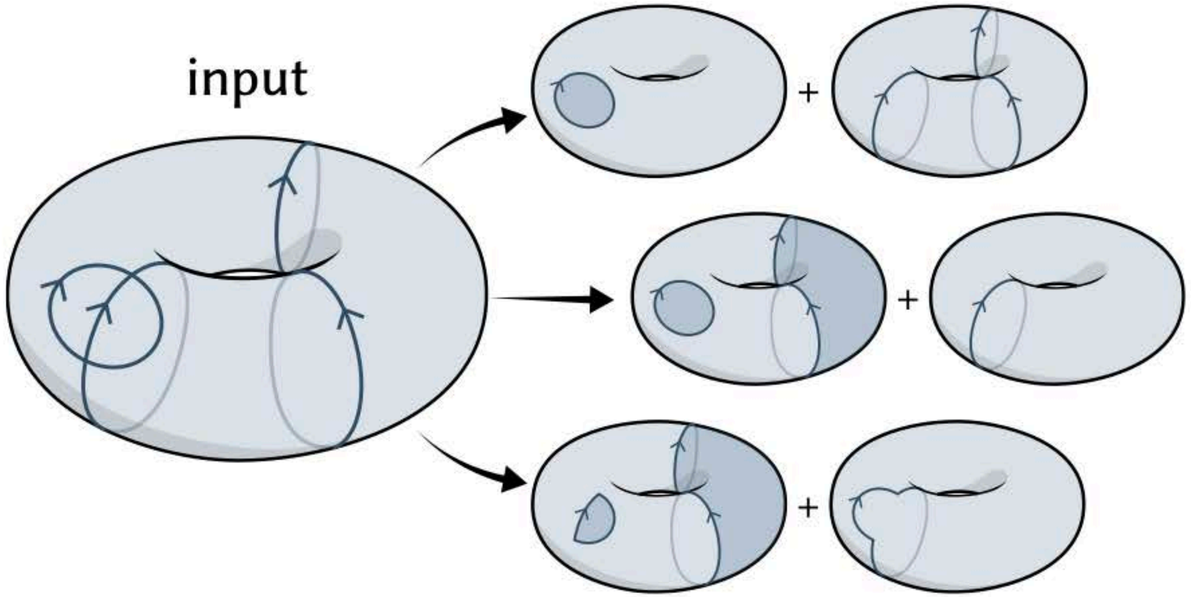
# Inside/outside can be complicated



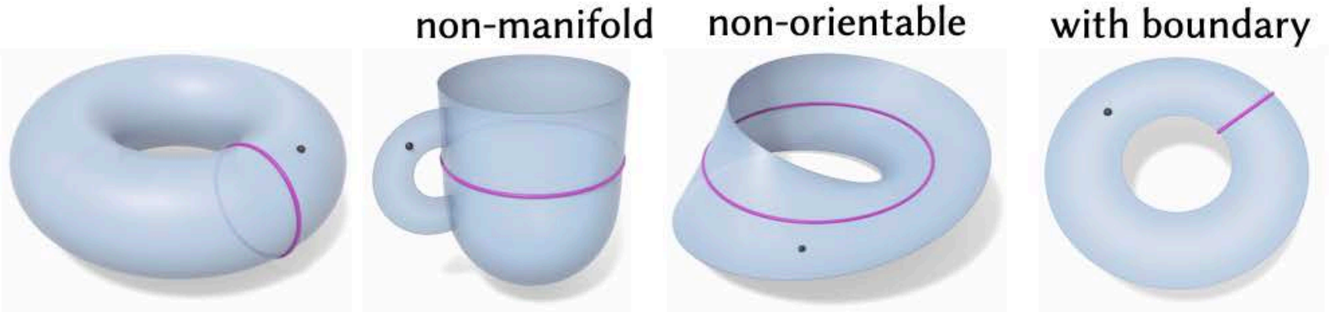

Can't just throw away non-bounding curves



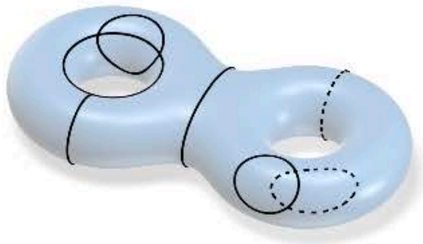
Many ways to partition input into regions



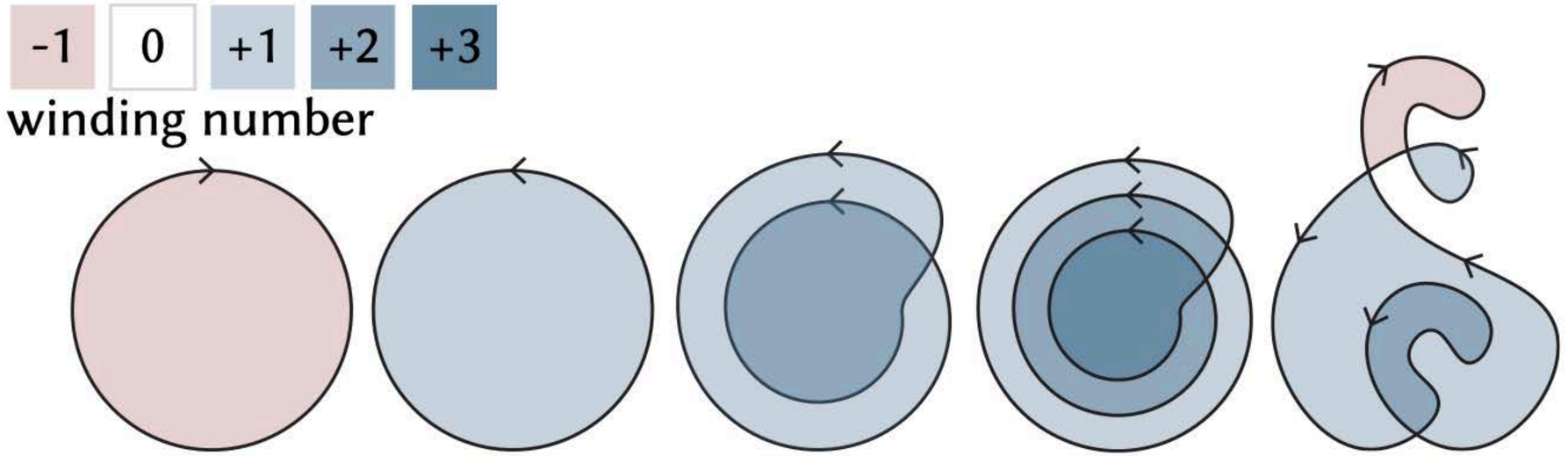

Other ambiguous cases...




*Is the point inside or outside the curve?*

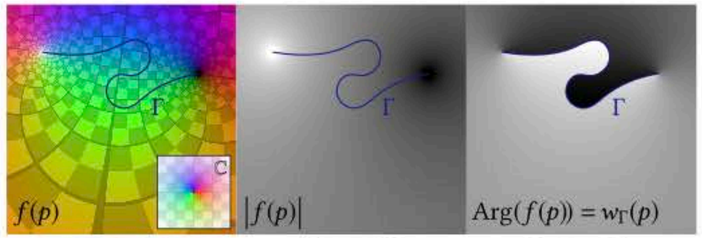


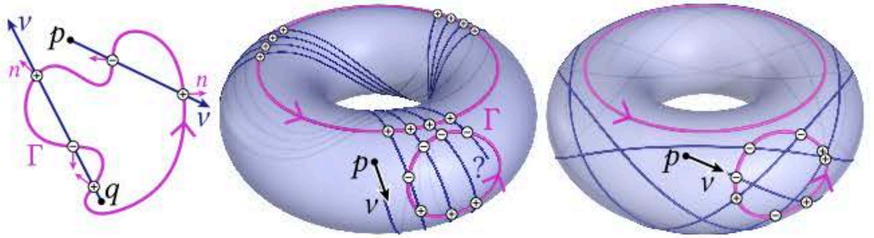
# Winding number, revisited

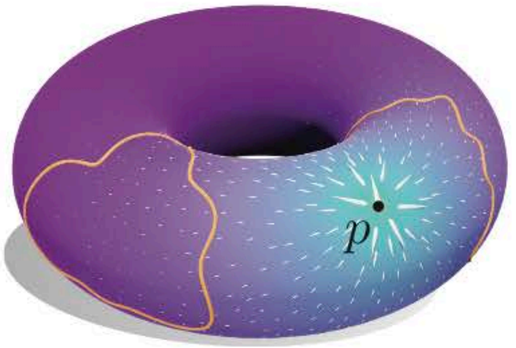


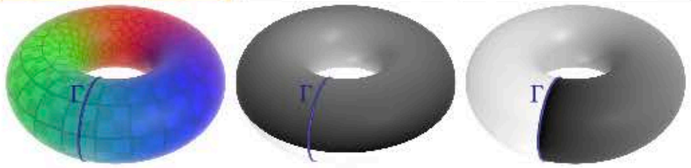

Complex analysis, differential geometry, topology, electromagnetism…

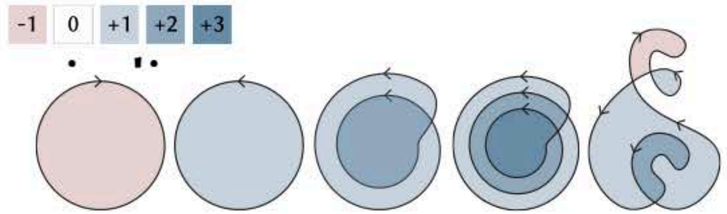
# Many formulas...

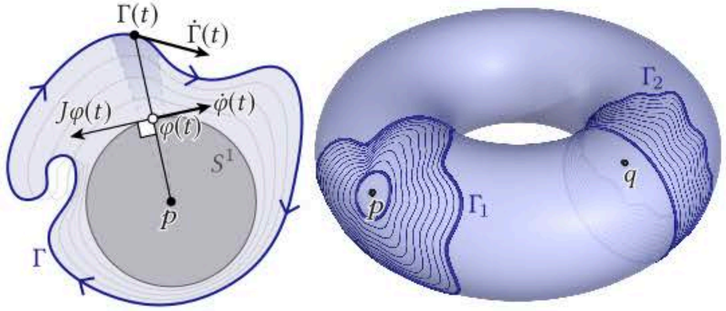


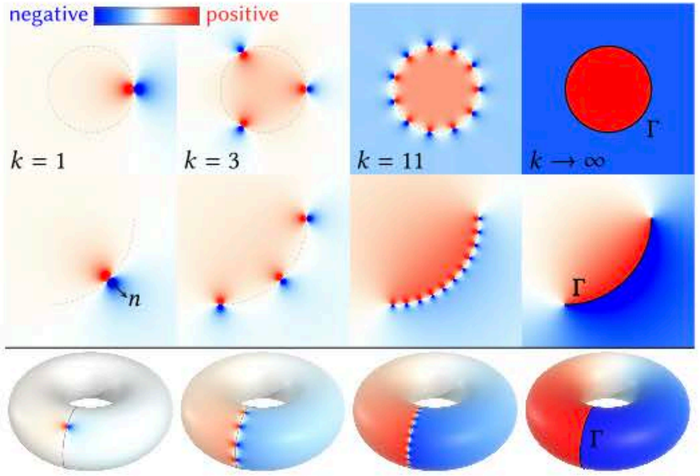



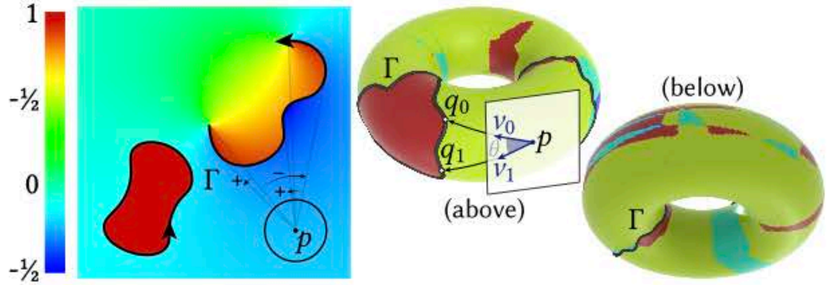



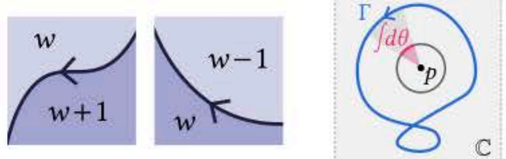



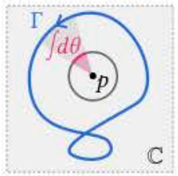



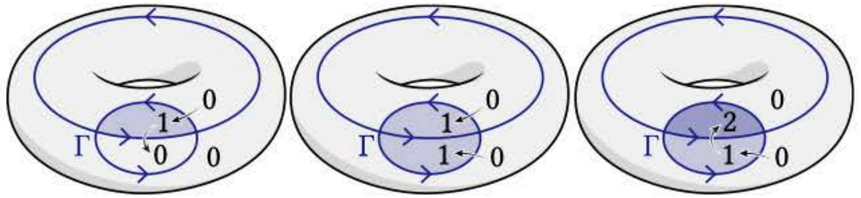



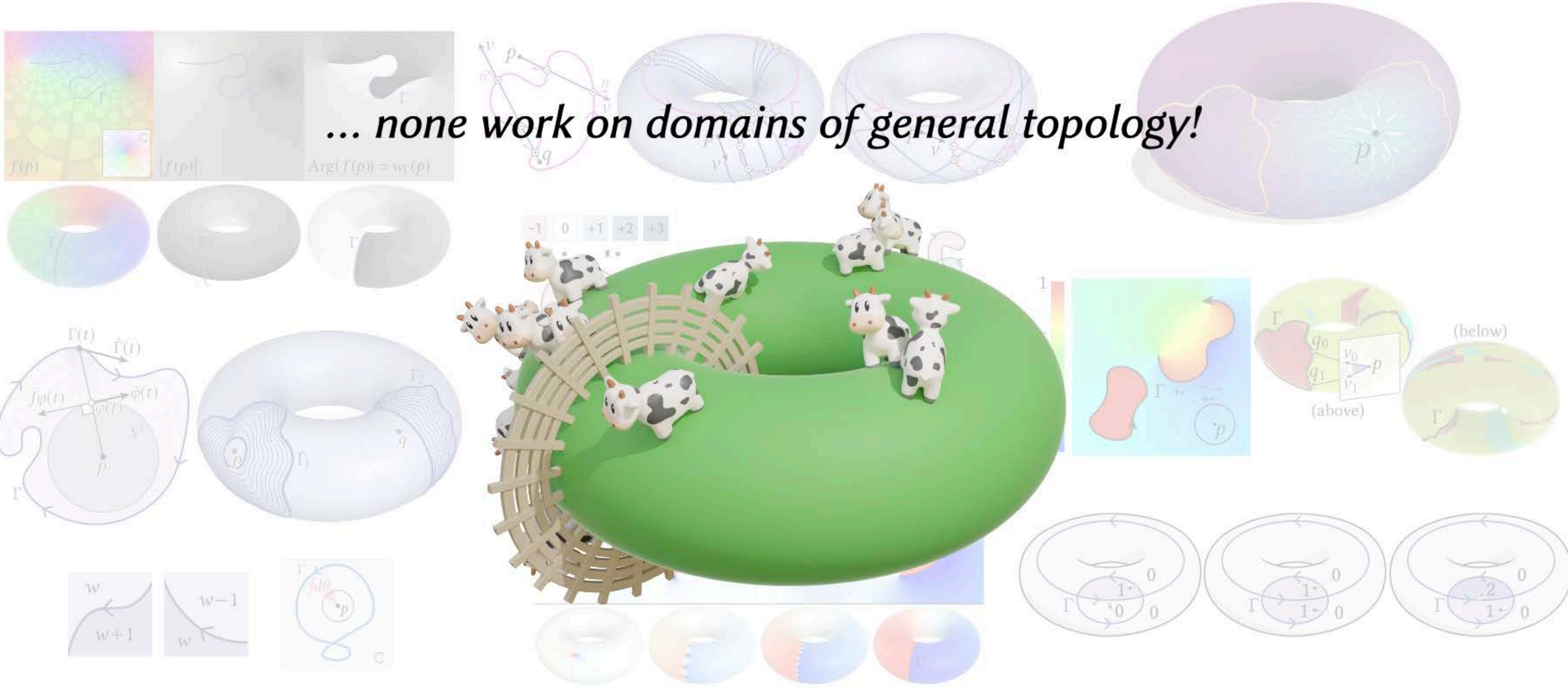



# Many formulas...

*... none work on domains of general topology!*

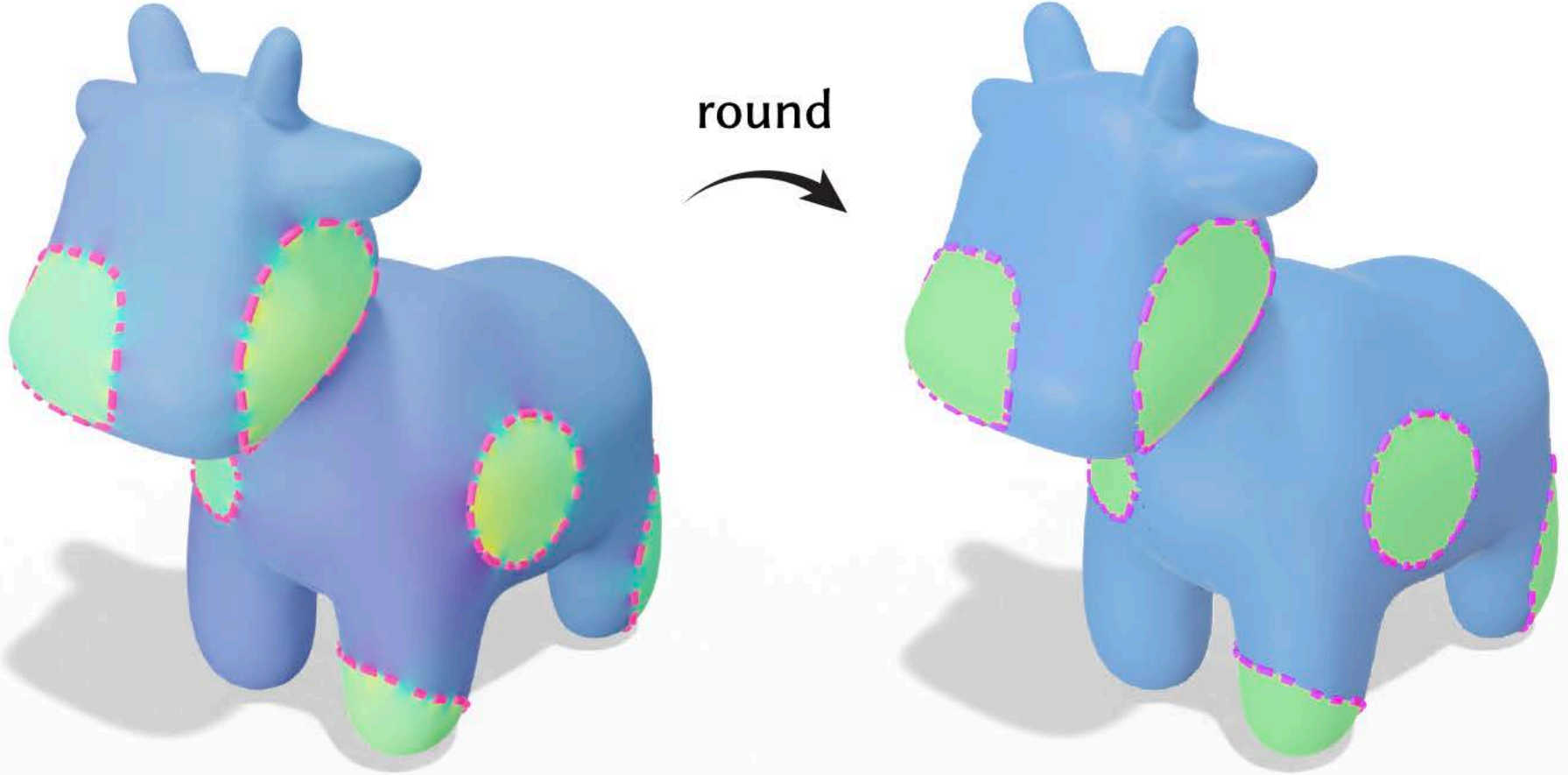


# If domain has trivial topology:

jump harmonic function

integer region labels

round



$$\begin{aligned} \Delta u &= 0, && \text{on } M \setminus \Omega, \\ u^+ - u^- &= 1, && \text{on } \Omega, \\ \partial u^+ / \partial n &= \partial u^- / \partial n, && \text{on } \Omega. \end{aligned}$$

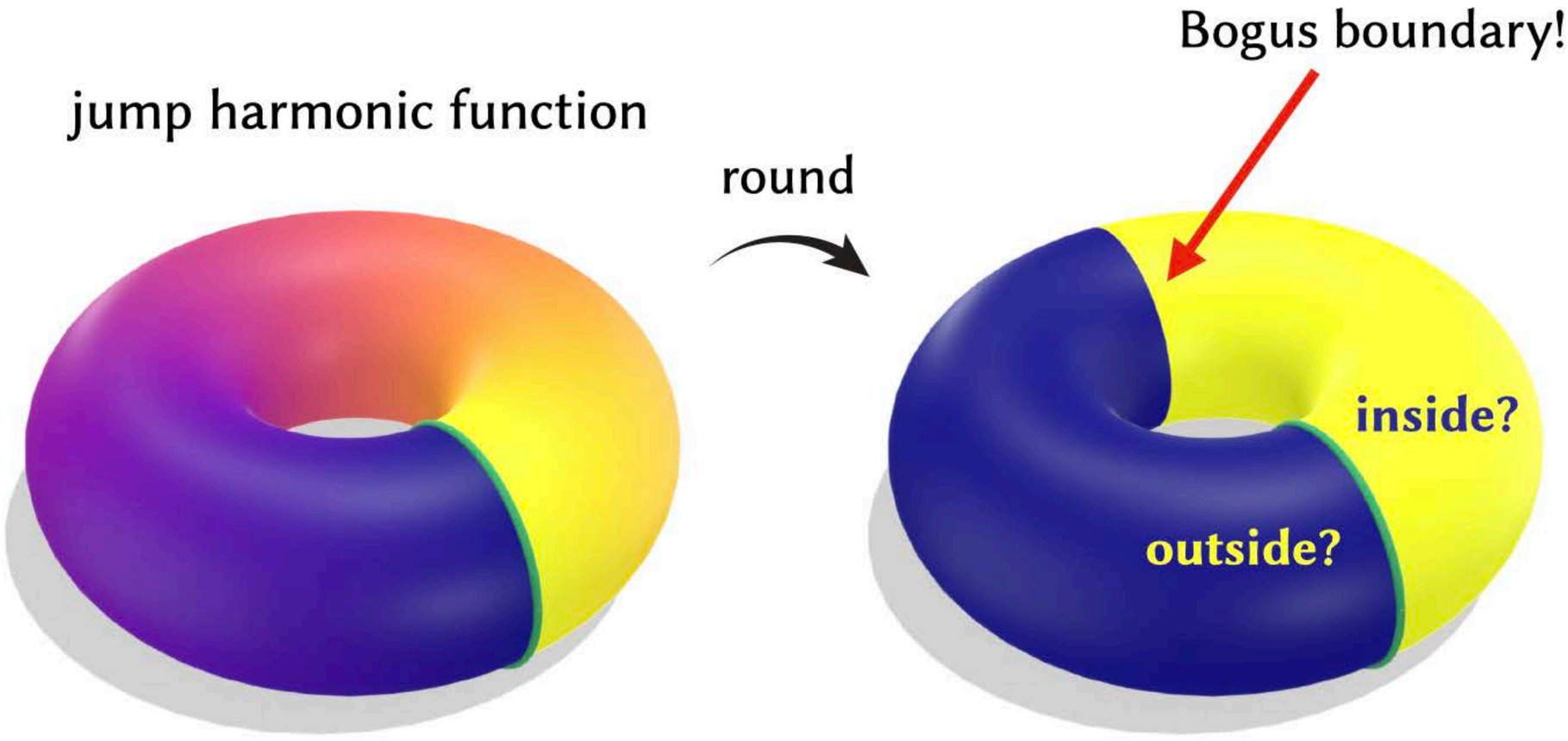
# If domain has <u>nontrivial</u> topology:

jump harmonic function

round



# If domain has <u>nontrivial</u> topology:



# We differentiate jump harmonic functions

jump harmonic function



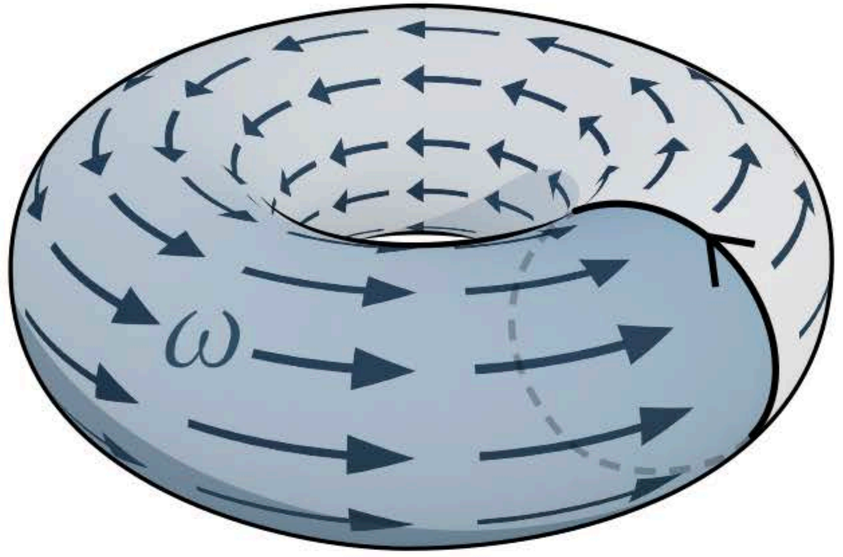
$u$

# We differentiate jump harmonic functions

jump harmonic function



$u$

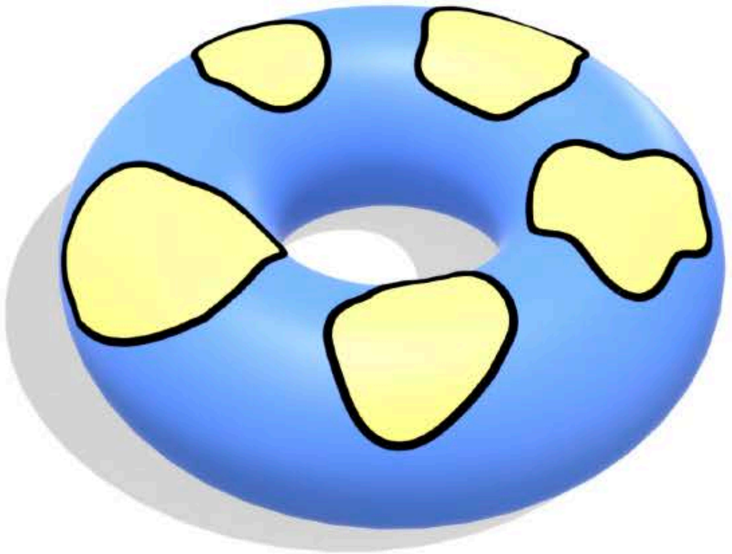



$\omega :=$ derivative of $u$

# The derivative tells us about non-bounding curves

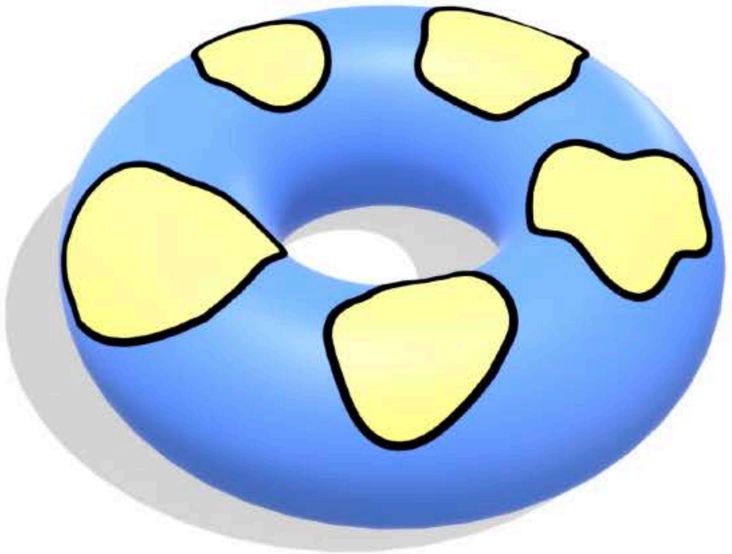
# The derivative tells us about non-bounding curves

If all curves are **bounding**,
the derivative is **zero**.

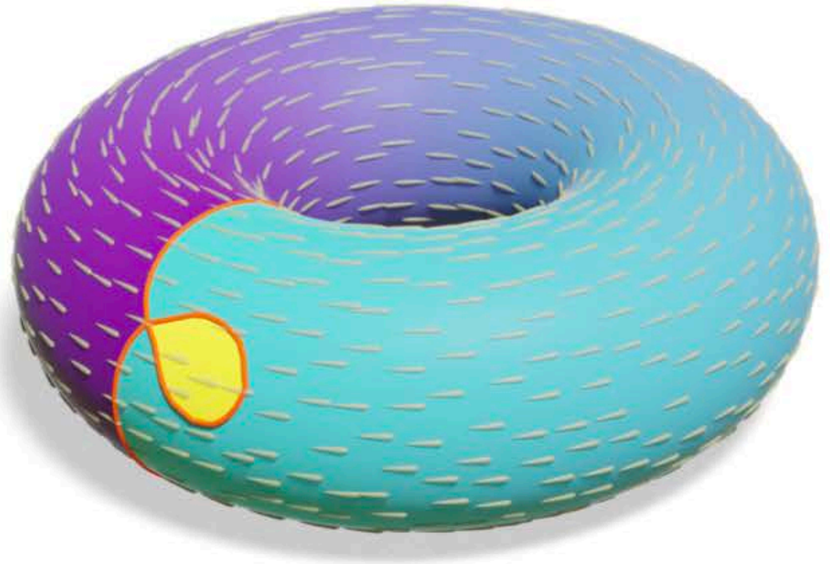


# The derivative tells us about non-bounding curves

If all curves are **bounding**, the derivative is **zero**.

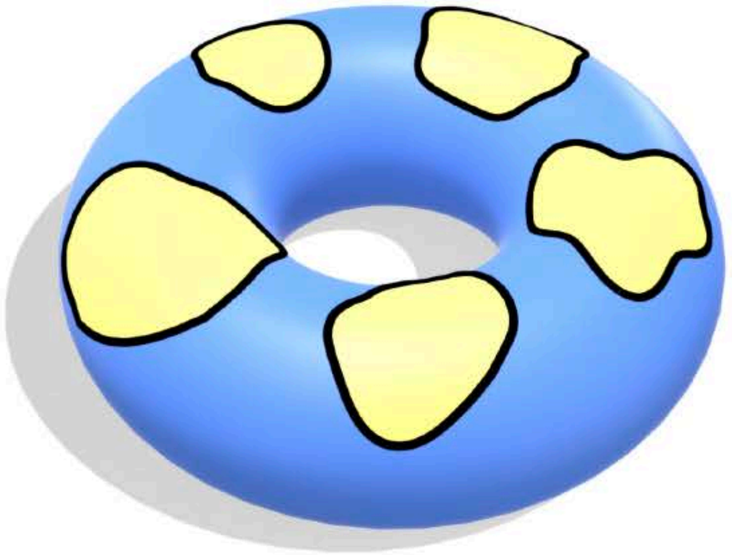


If some curves are **non-bounding**, the derivative is nonzero and **harmonic**.



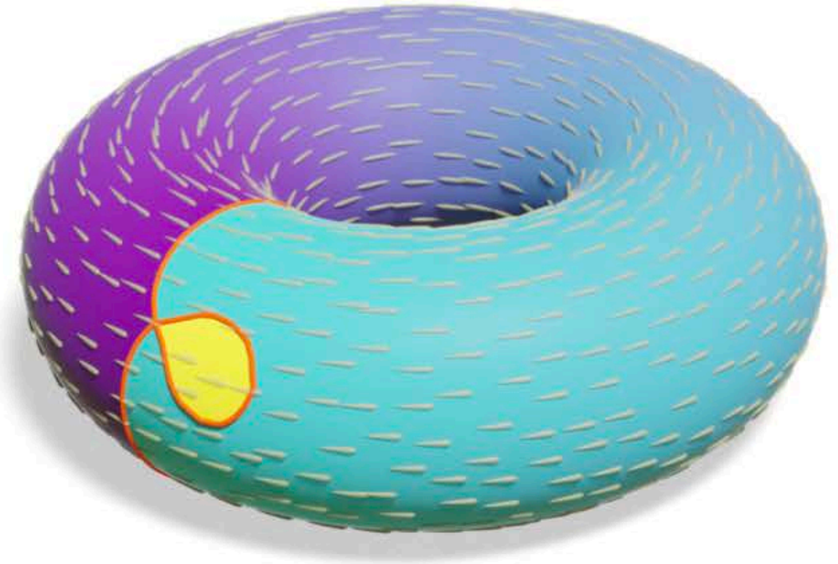
# The derivative tells us about non-bounding curves

If all curves are **bounding**,
the derivative is **zero**.

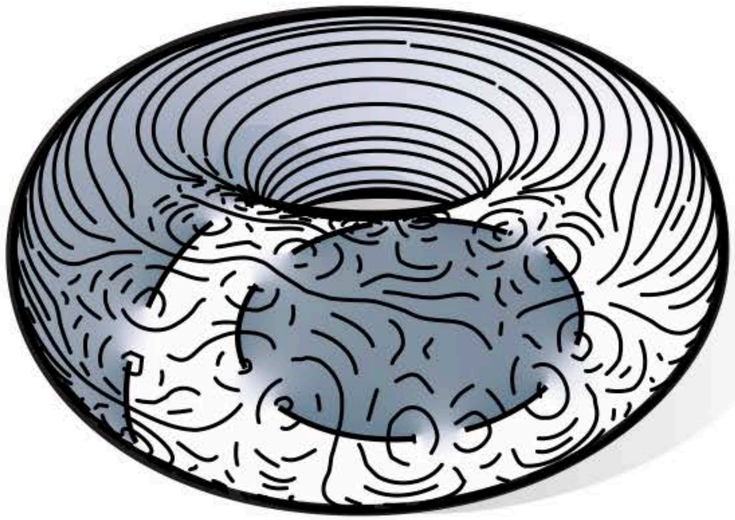


If some curves are **non-bounding**,
the derivative is nonzero and **harmonic**.

*Fact:* Harmonic 1-forms correspond to non-bounding curves!

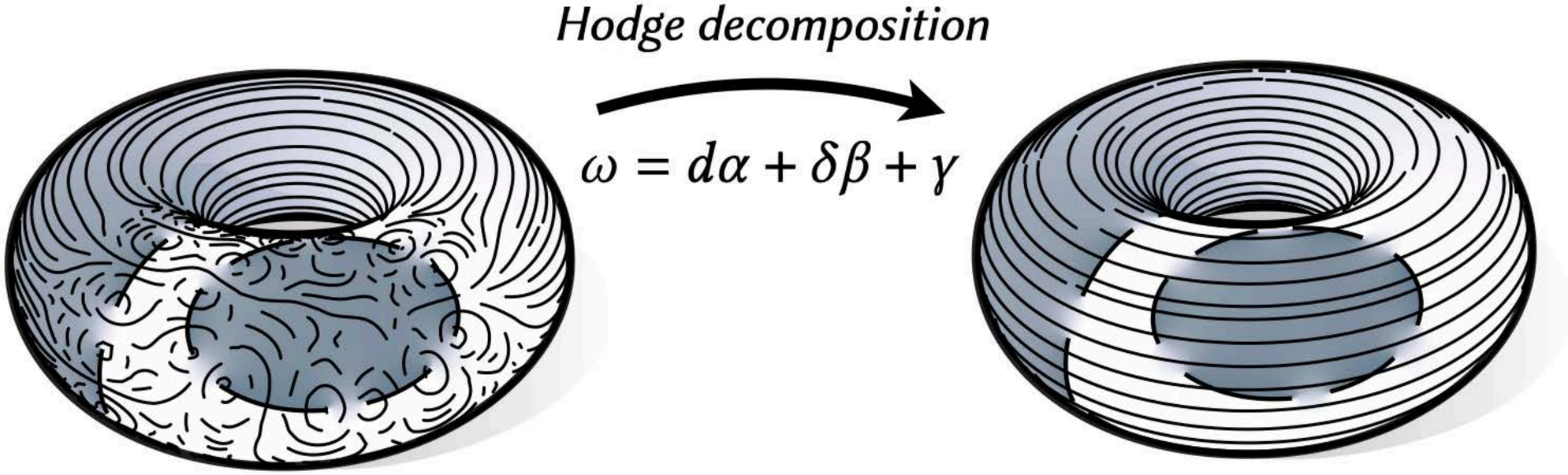


# If the curve is broken, find closest harmonic field

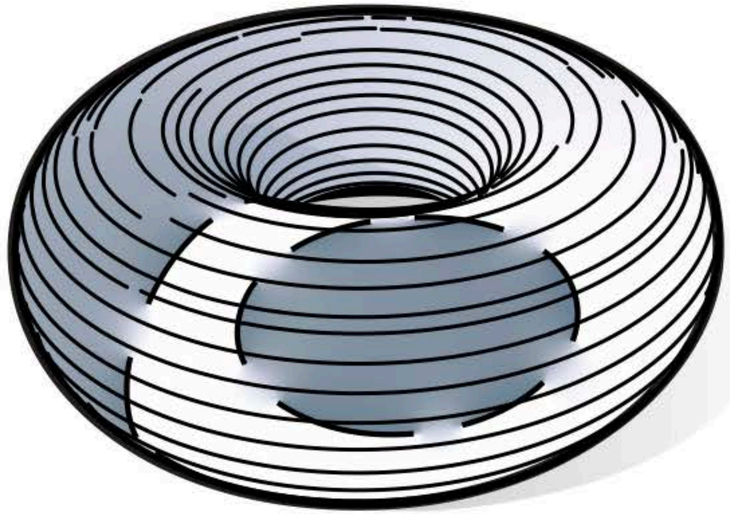


$\omega :=$ derivative of $u$

# If the curve is broken, find closest harmonic field

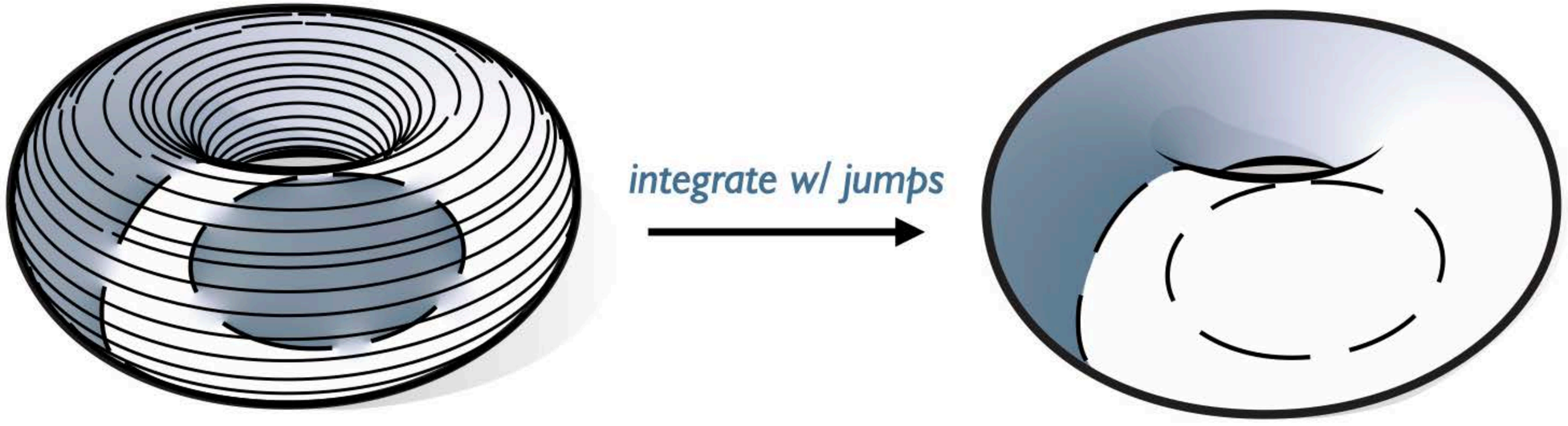


# Use harmonic vector field to decompose curve



harmonic 1-form

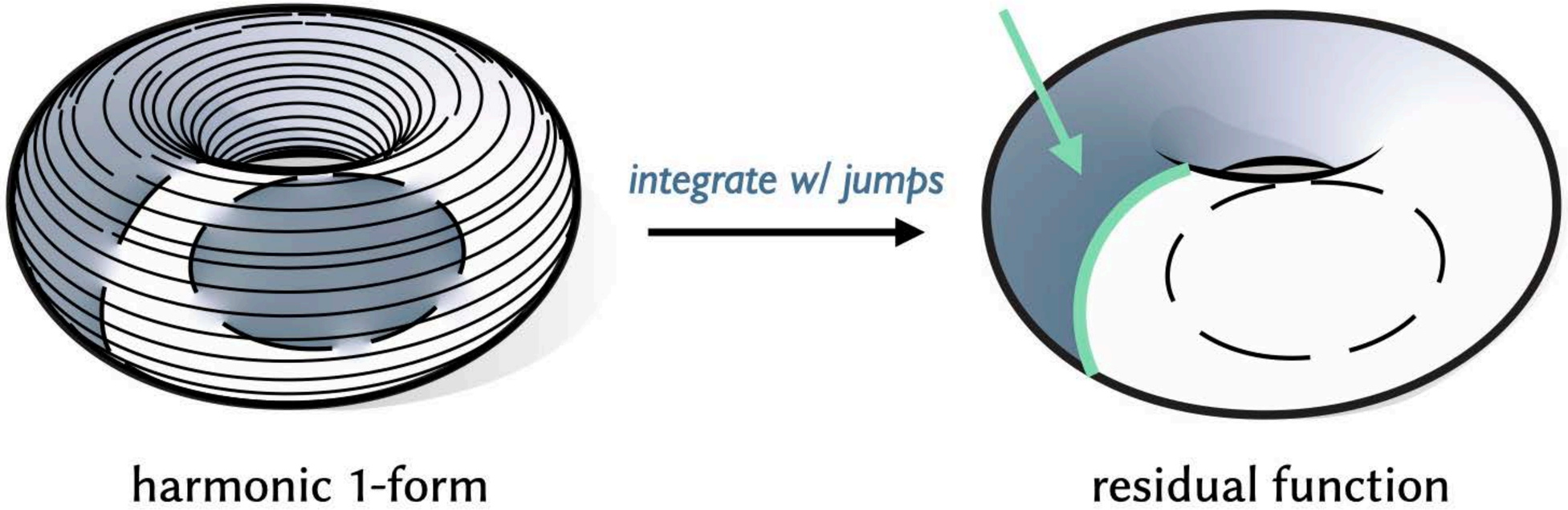
# Use harmonic vector field to decompose curve



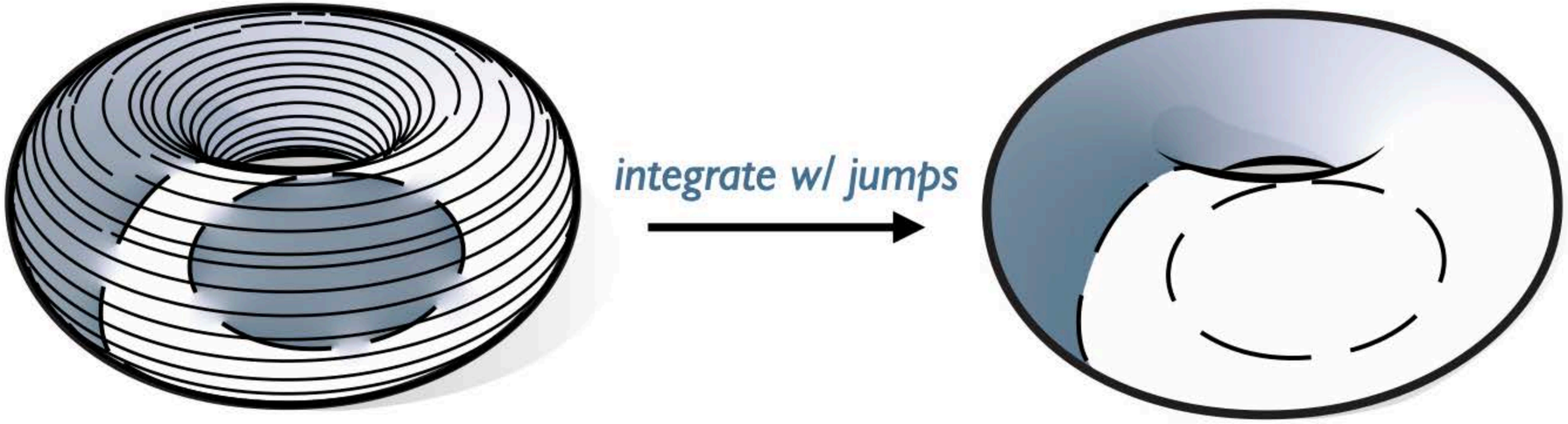

harmonic 1-form

residual function

# Use harmonic vector field to decompose curve



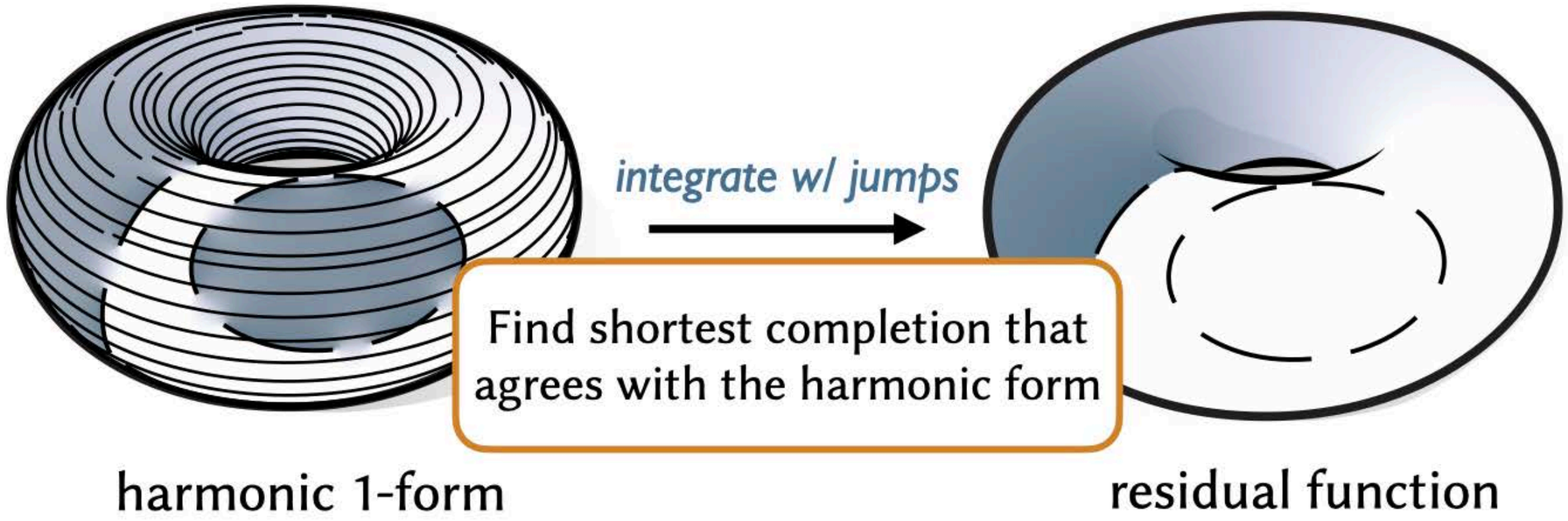
# Use harmonic vector field to decompose curve



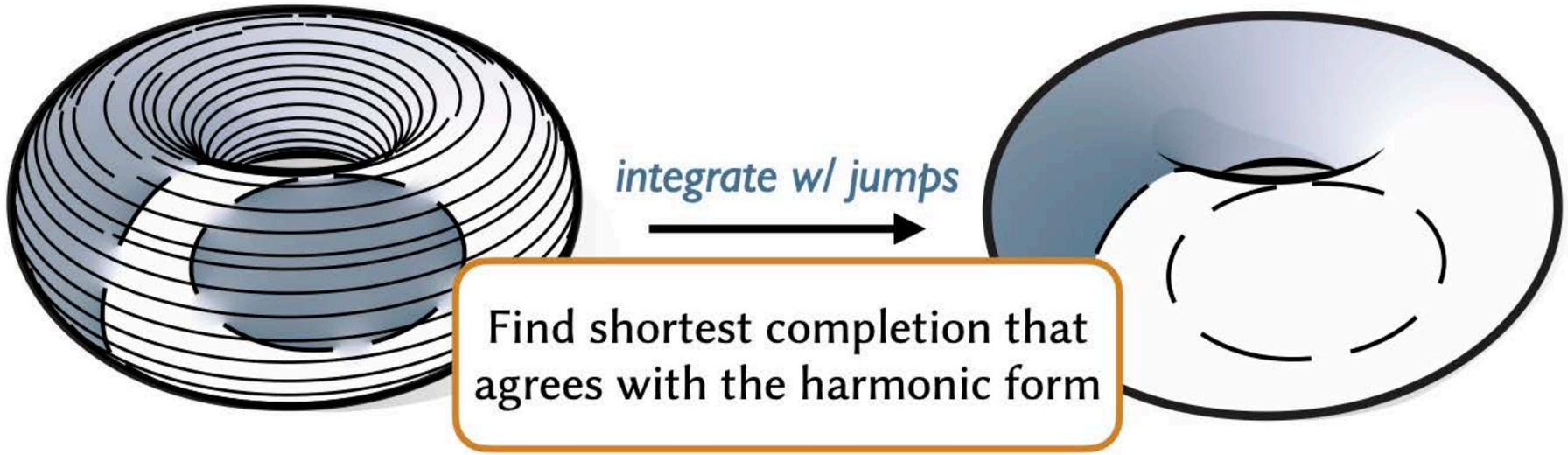

harmonic 1-form

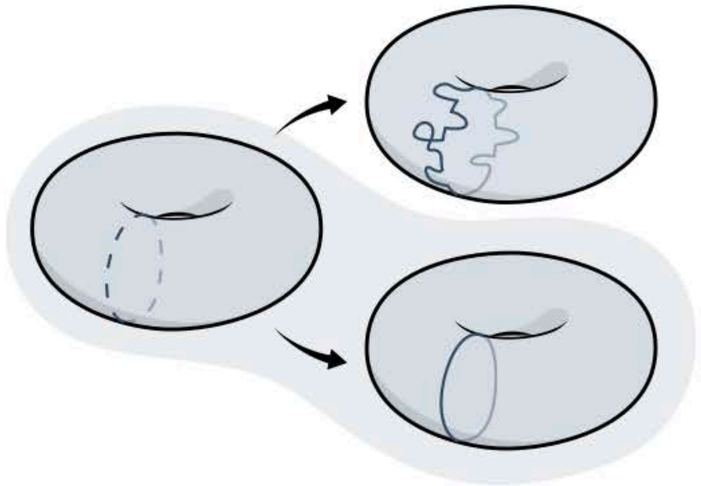
residual function

# Use harmonic vector field to decompose curve

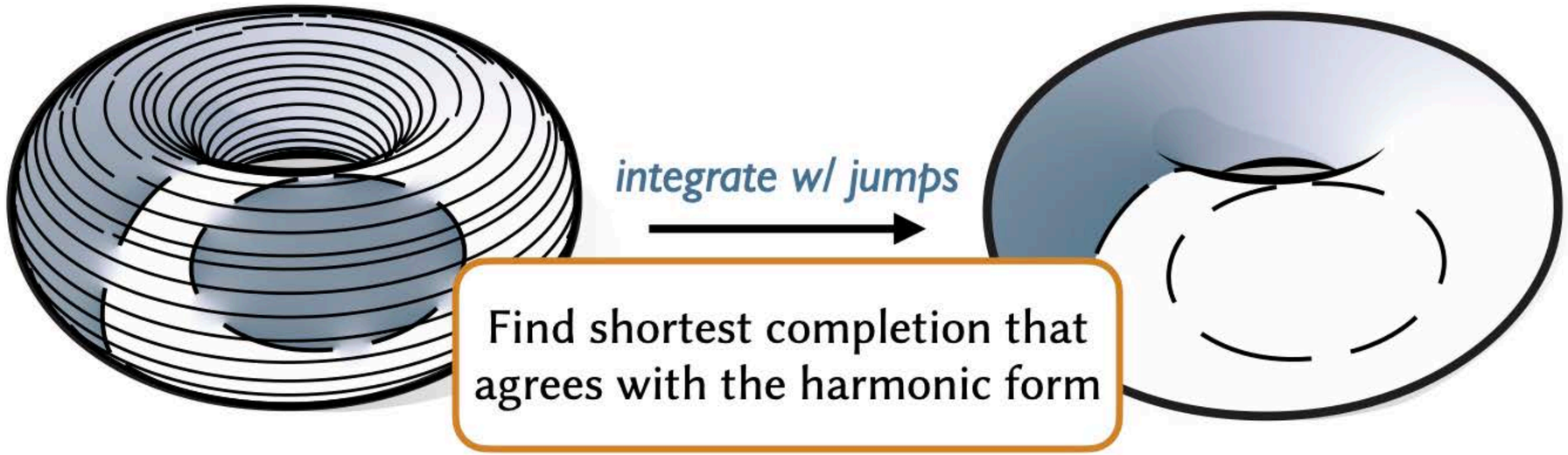


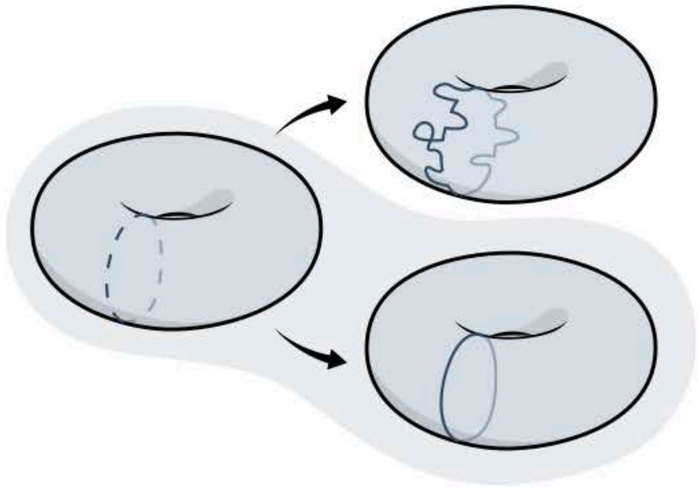
# Use harmonic vector field to decompose curve

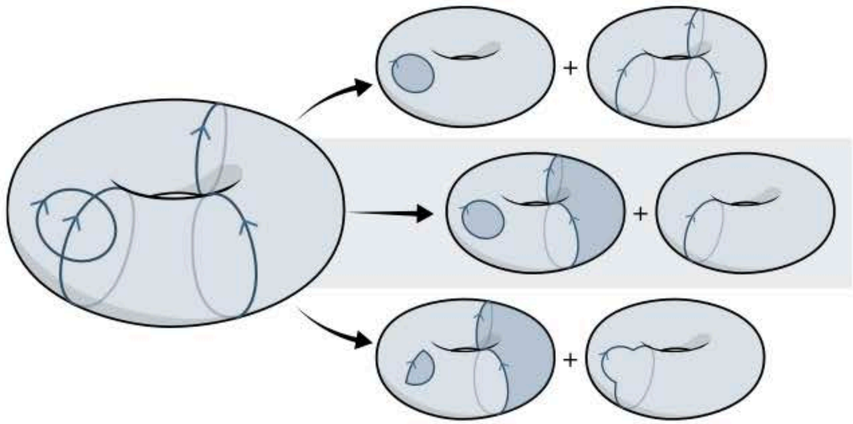


# Use harmonic vector field to decompose curve

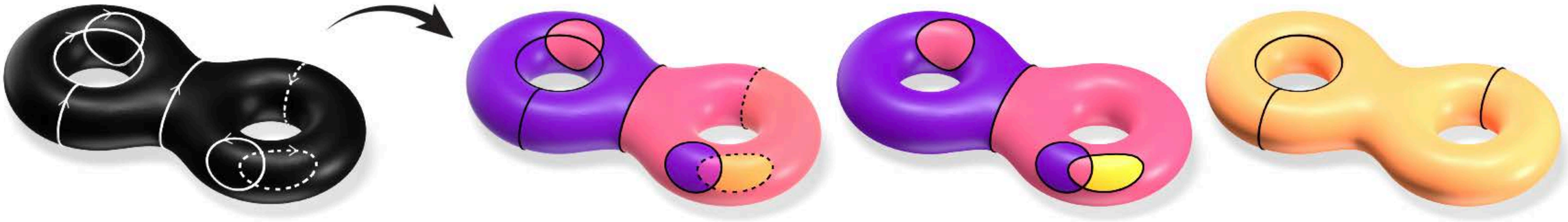




# *Surface winding numbers* (SWN)

**Input**: oriented curves on a surface mesh



**Outputs:**

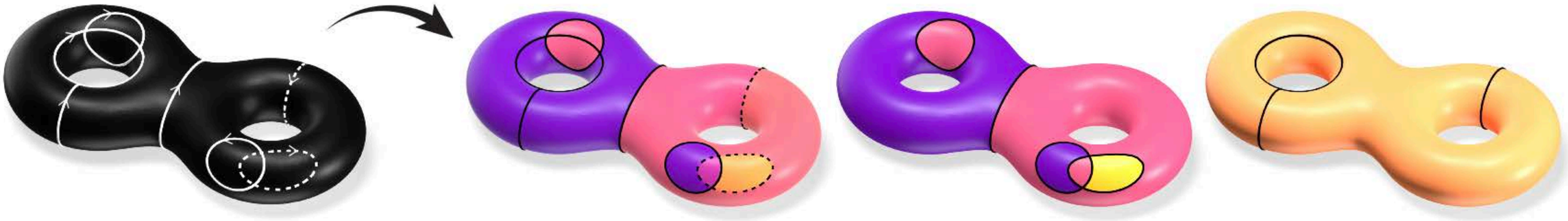
winding number function

bounding components

non-bounding components

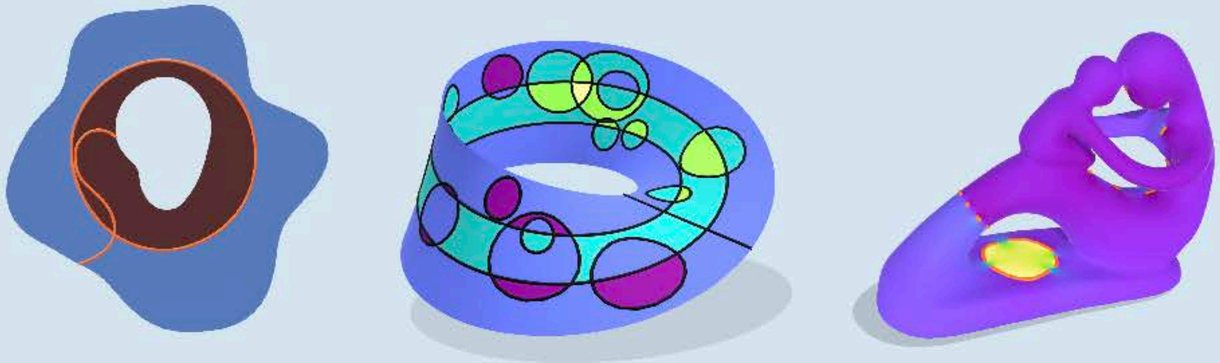
# *Surface winding numbers* (SWN)

**Input**: oriented curves on a surface mesh



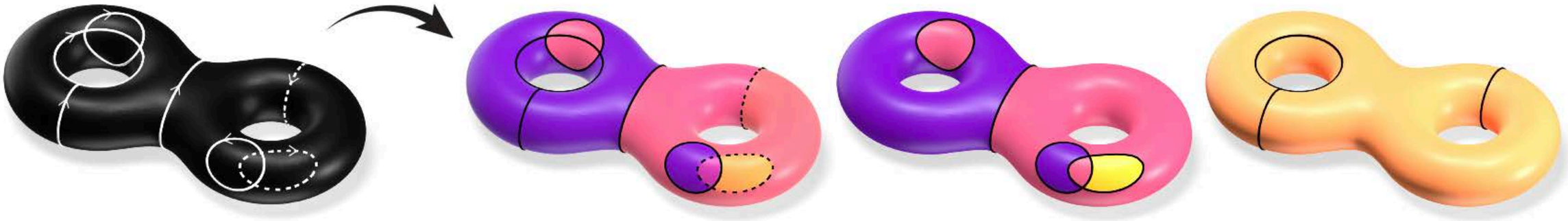
**Outputs:** winding number function | bounding components | non-bounding components

Handles general topology, broken curves



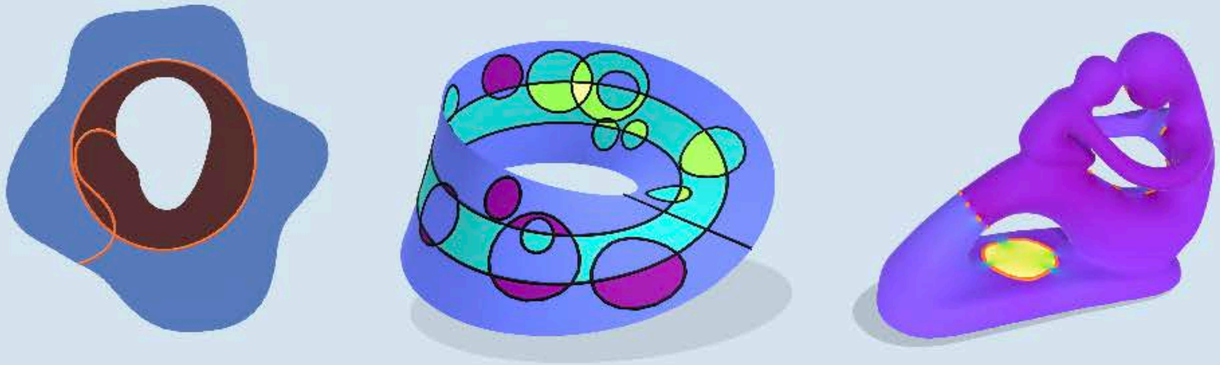
# *Surface winding numbers* (SWN)

**Input**: oriented curves on a surface mesh

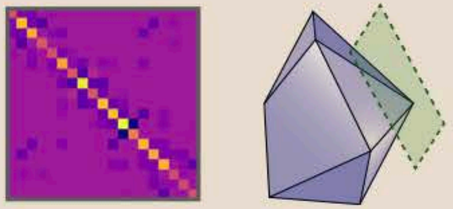


**Outputs:** winding number function | bounding components | non-bounding components

Handles general topology, broken curves

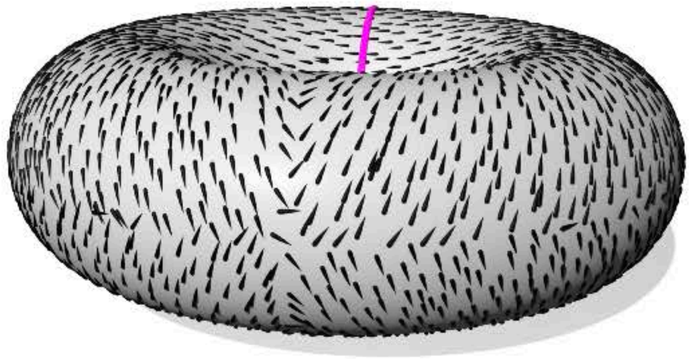


sparse Poisson problems + LP



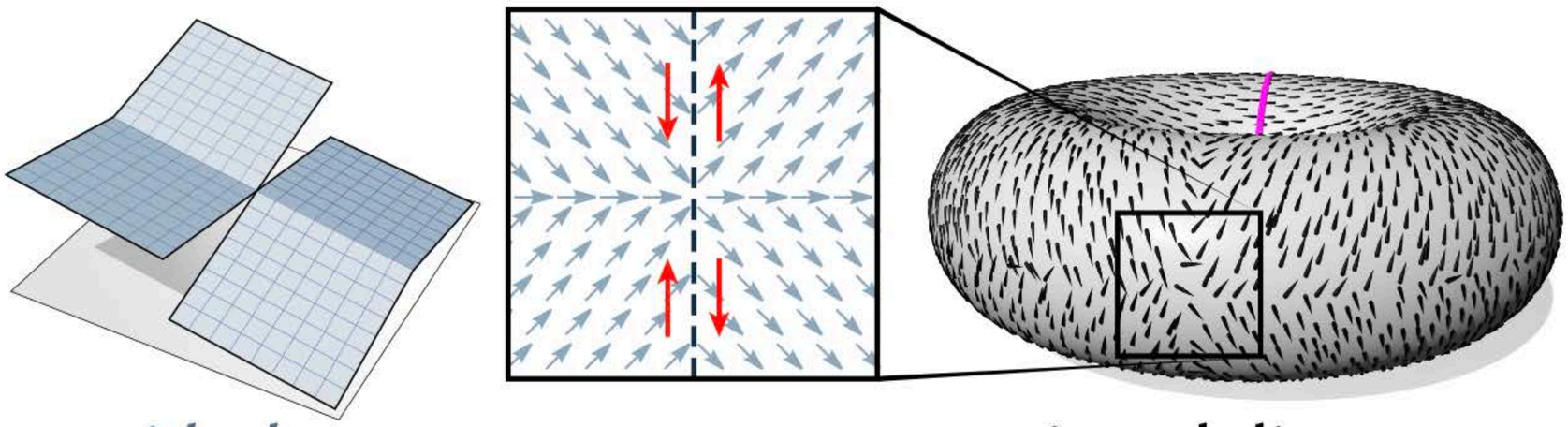
# Signed distance gradient for non-bounding curves

# Signed distance gradient for non-bounding curves



signed distance
gradient

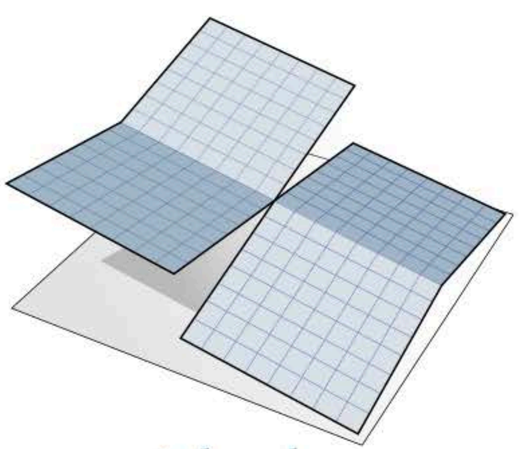
# Signed distance gradient for non-bounding curves



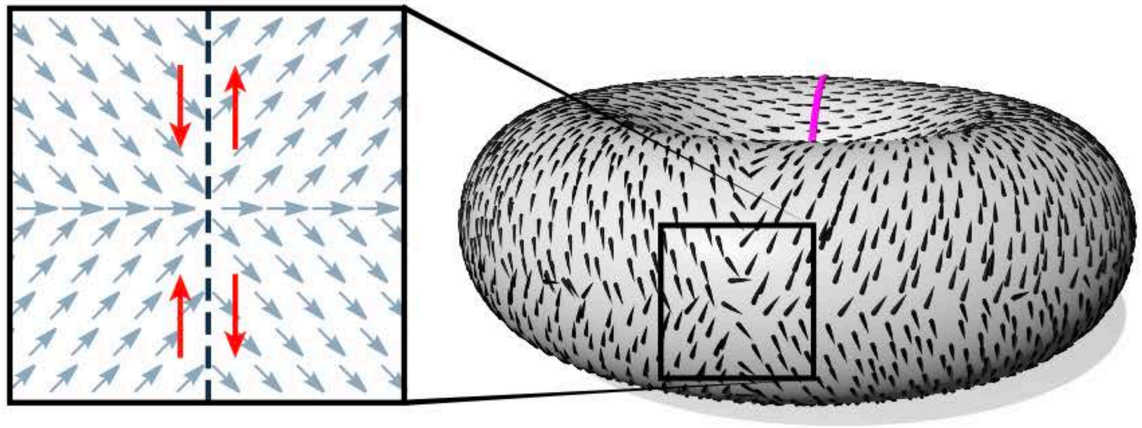
*ideal solution*

signed distance gradient

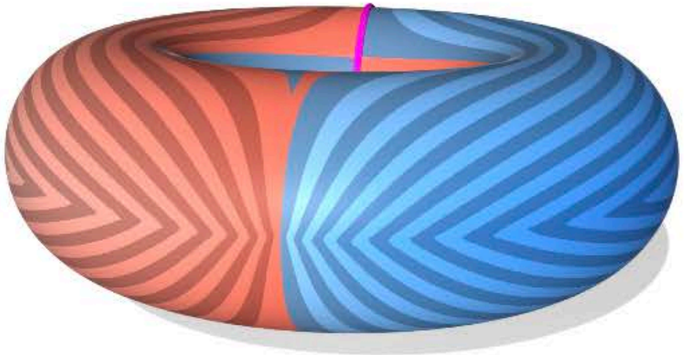
# Signed distance gradient for non-bounding curves



*ideal solution*

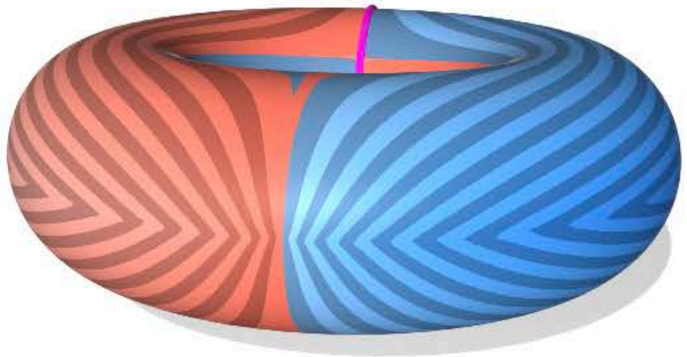


signed distance gradient



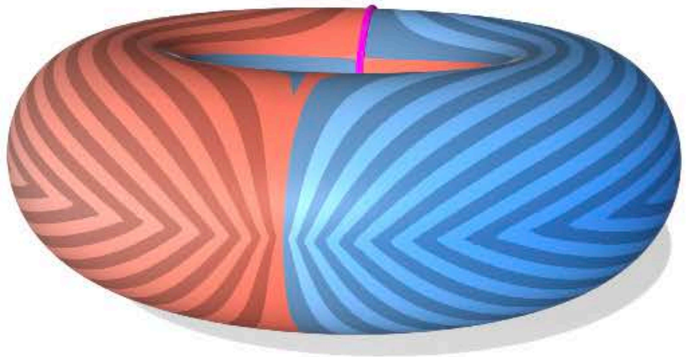
*obtained solution*

# Piecewise continuous distance for non-bounding curves

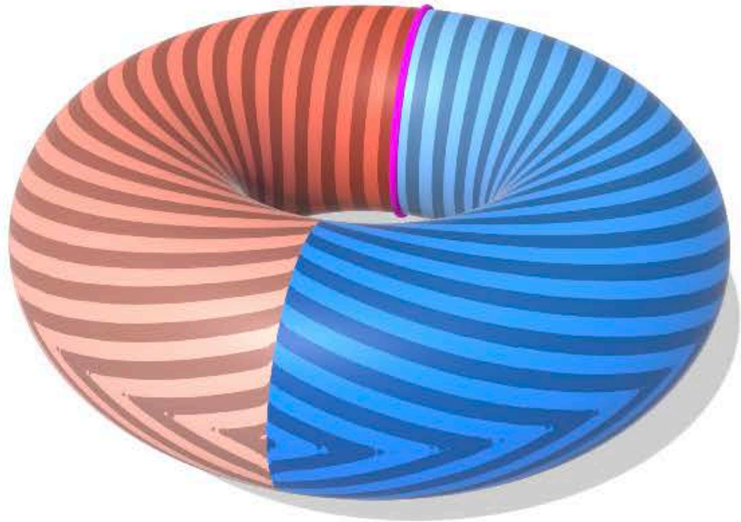


# Piecewise continuous distance for non-bounding curves

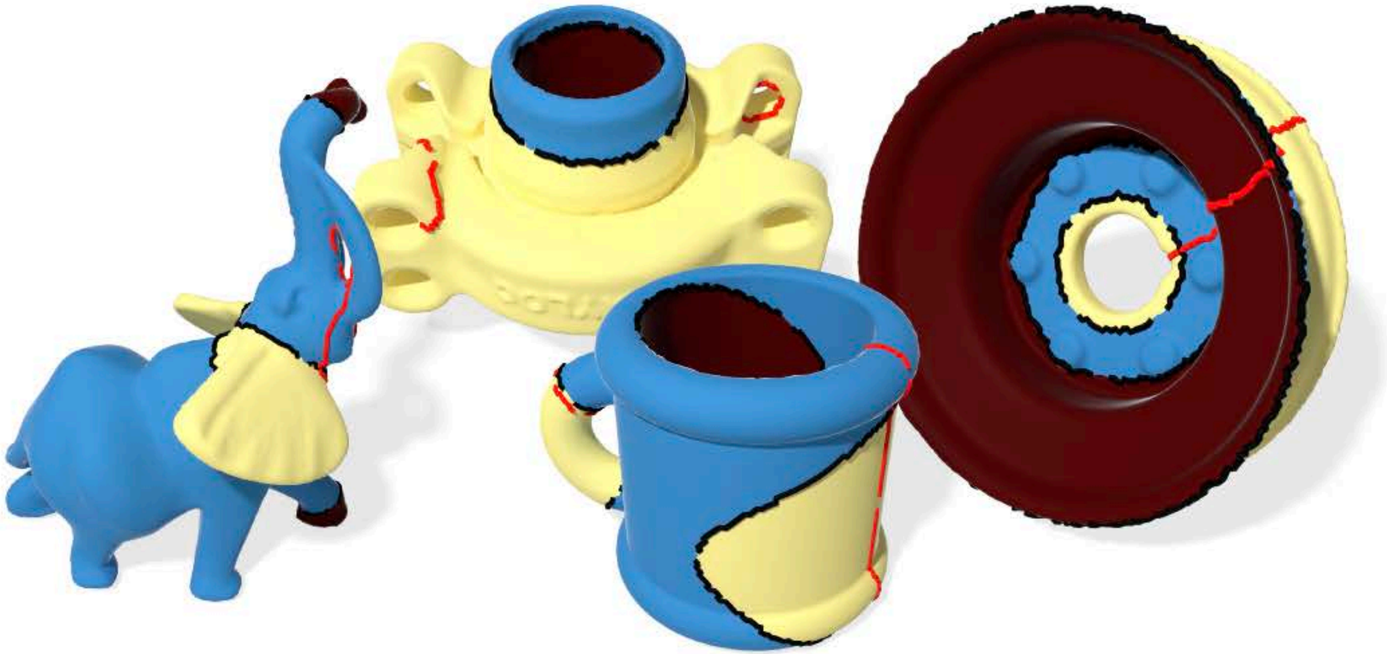
standard integration



piecewise continuous integration



# Evaluation of inside-outside classification



**Benchmark setup**

(1) Generate random ground-truth regions, take boundaries *[sub-levelsets of low-frequency Laplacian eigenfunctions]*

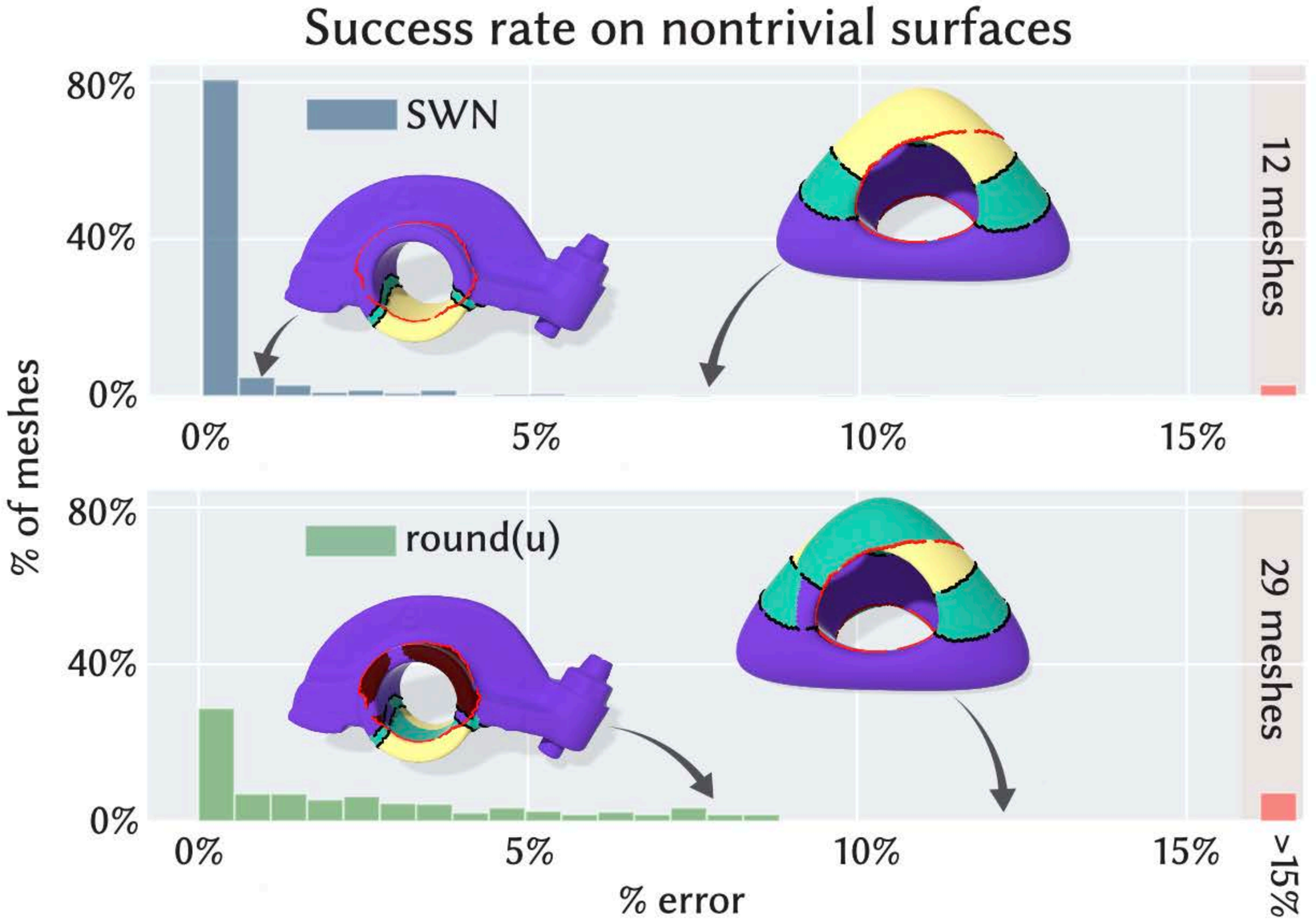
(2) Add non-bounding loops *[compute homology basis, select random subset]*

(3) Break up curves *[use random gap & dash size]*

(4) Run SWN; compute % of surface area correctly classified *[shift $w$ to match ground-truth value in an arbitrary face]*

*934 total test cases, 451 multiply-connected*

# Evaluation of inside-outside classification



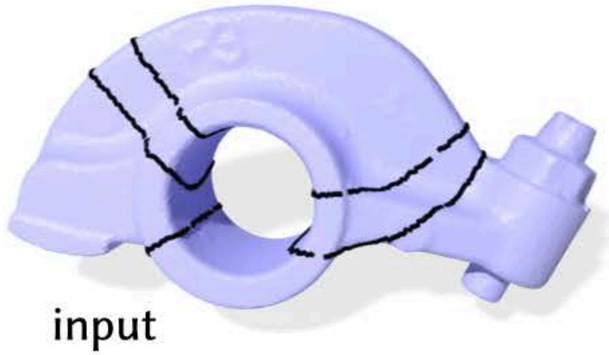

Ours

PSR/GWN

Much less accurate without processing nonbounding curves

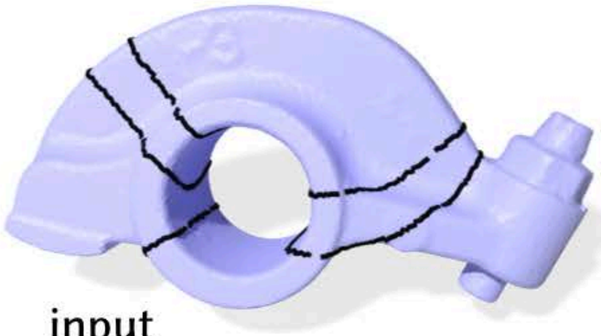
# “Failure” cases

# “Failure” cases

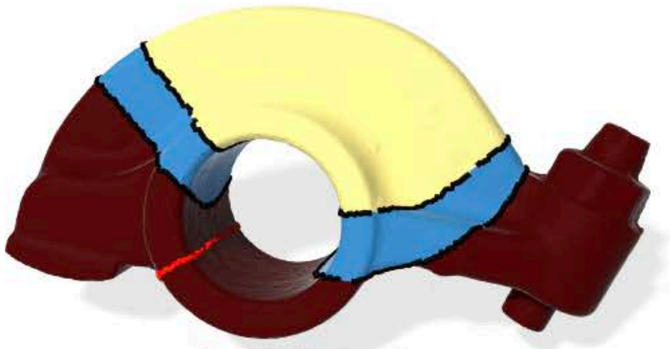


input

# “Failure” cases

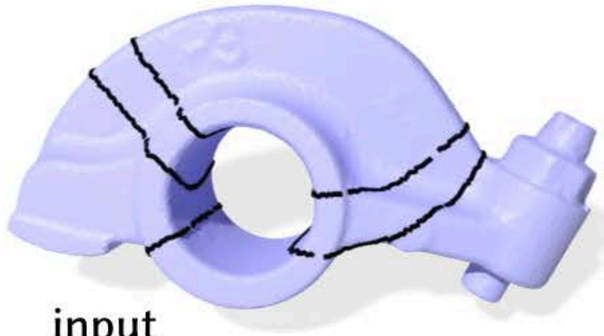


input

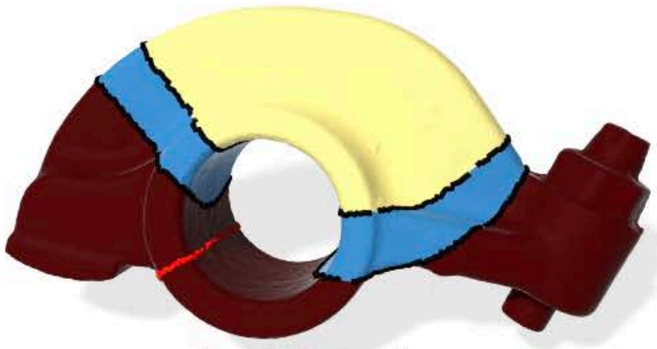


ground truth regions

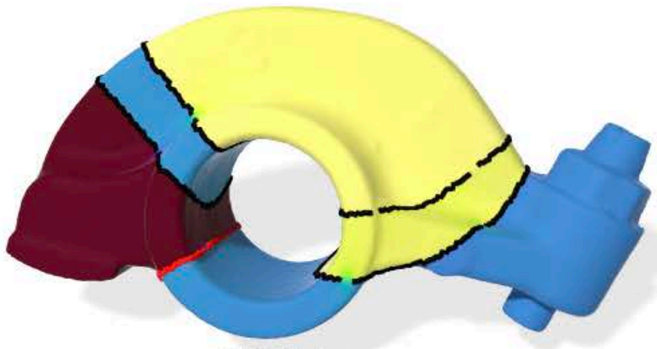
# “Failure” cases



input



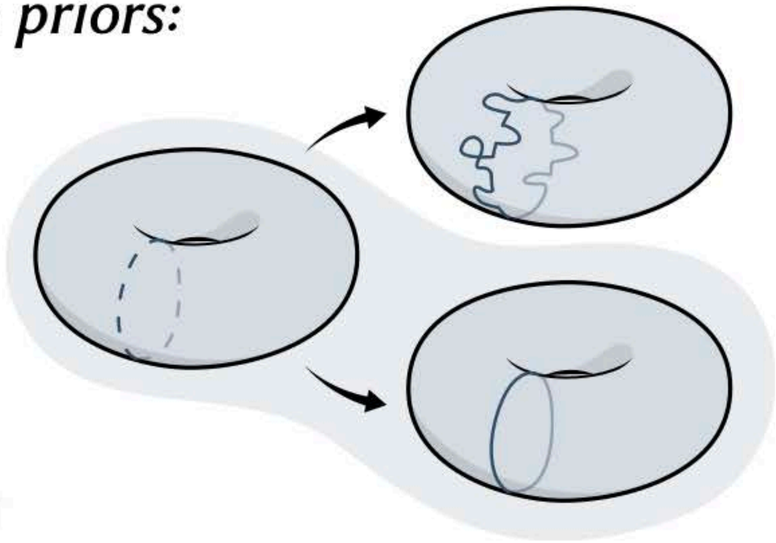
ground truth regions

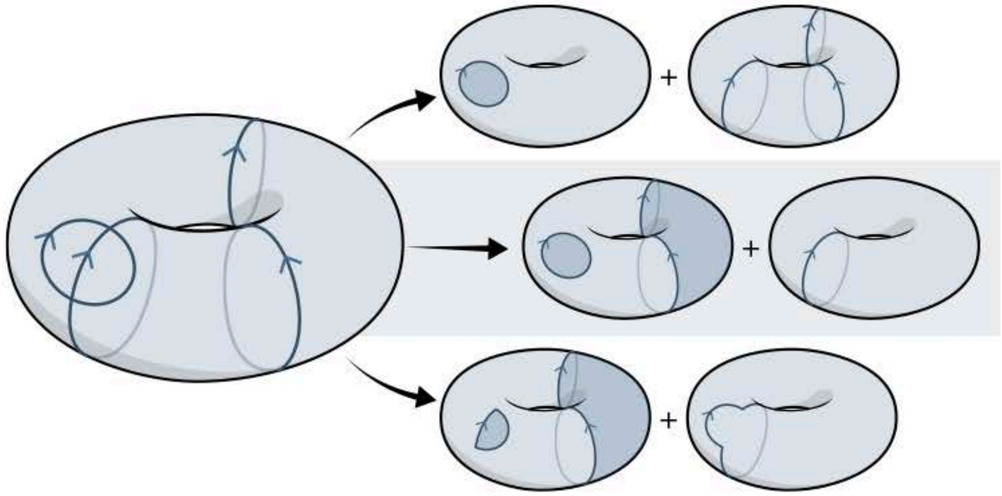


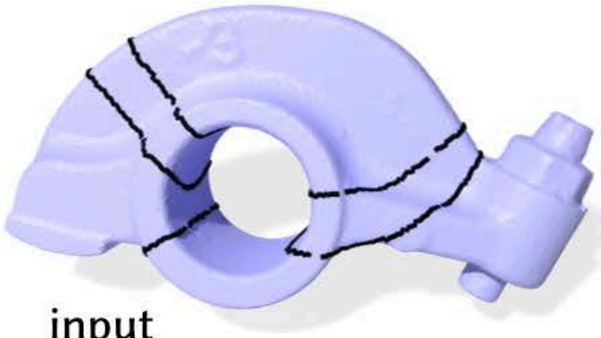
SWN

# “Failure” cases

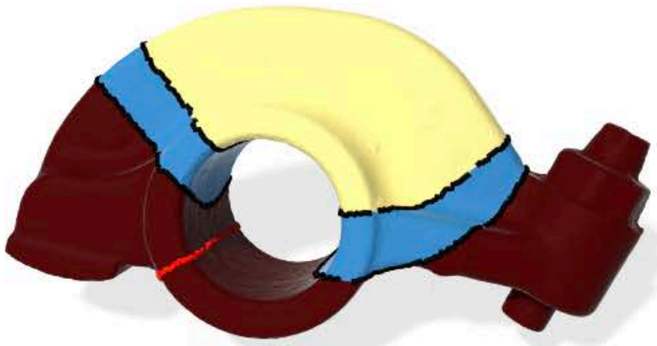
*Recall shortest-length priors:*



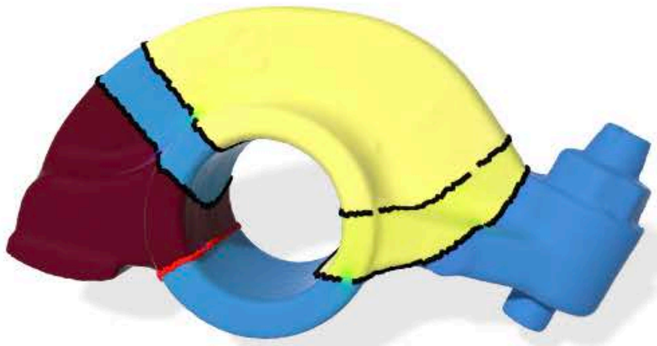




input



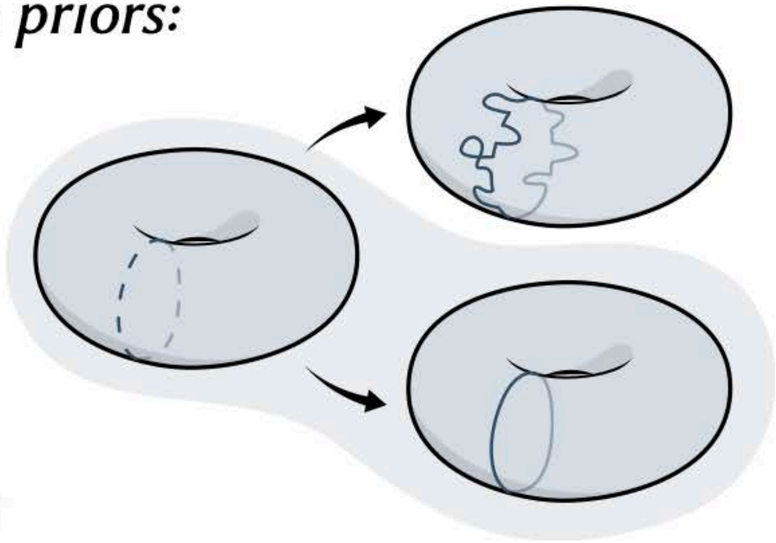
ground truth regions

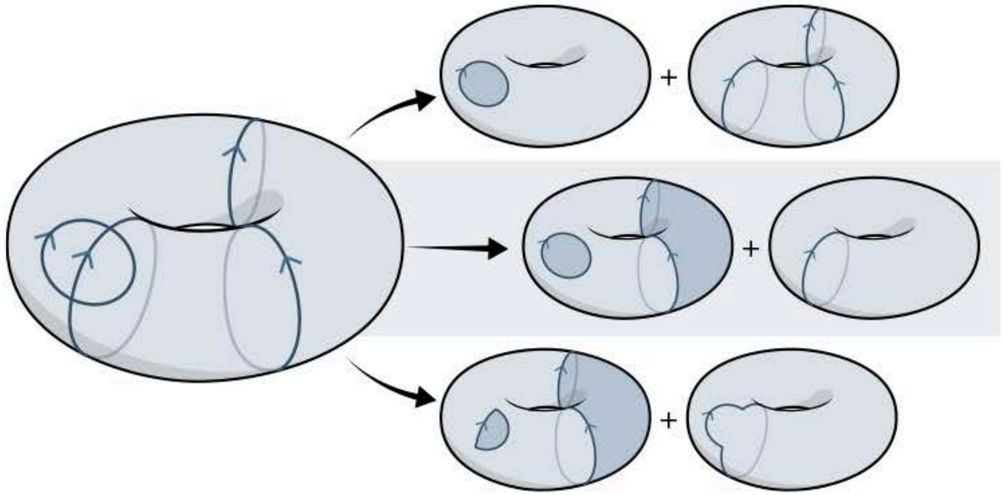


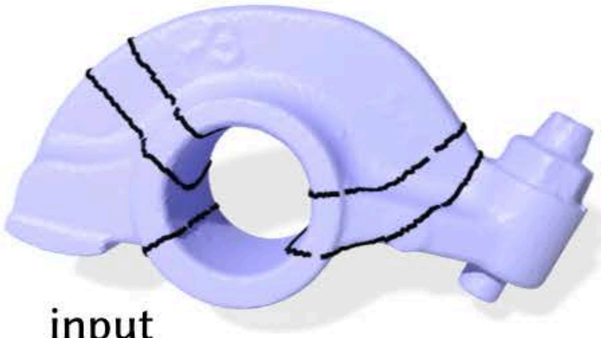
SWN

# “Failure” cases

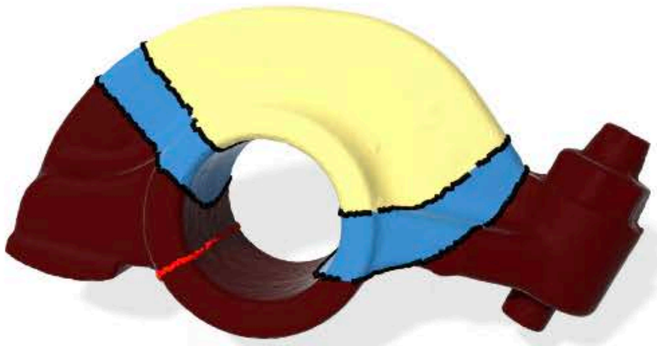
*Recall shortest-length priors:*



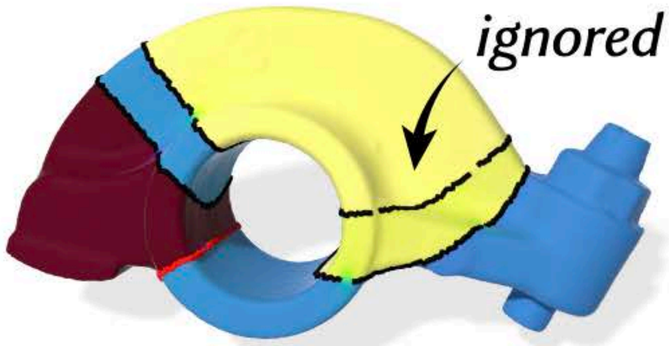




input



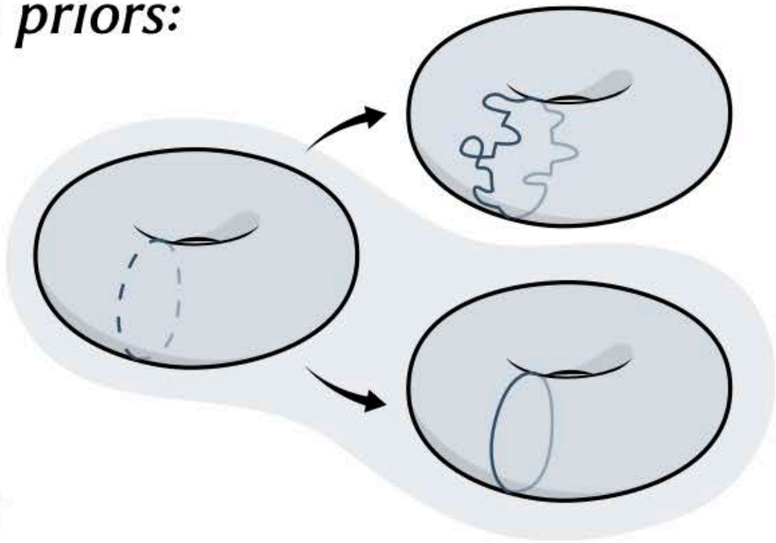
ground truth regions

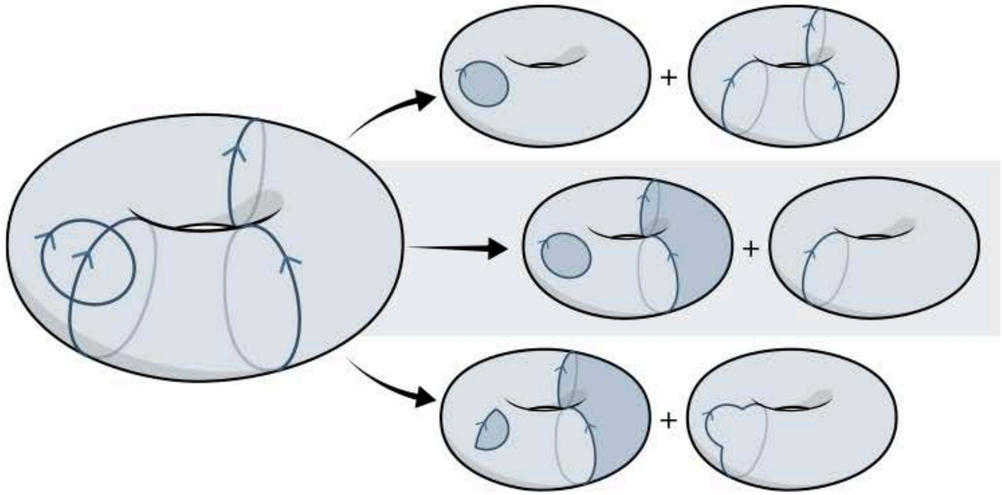


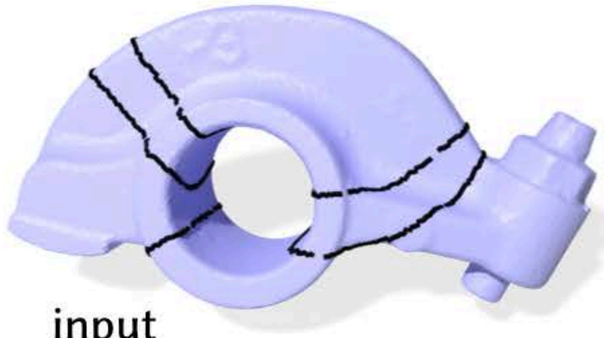

SWN

# “Failure” cases

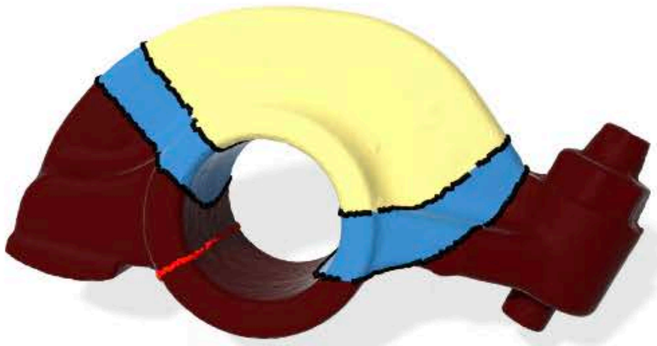
*Recall shortest-length priors:*



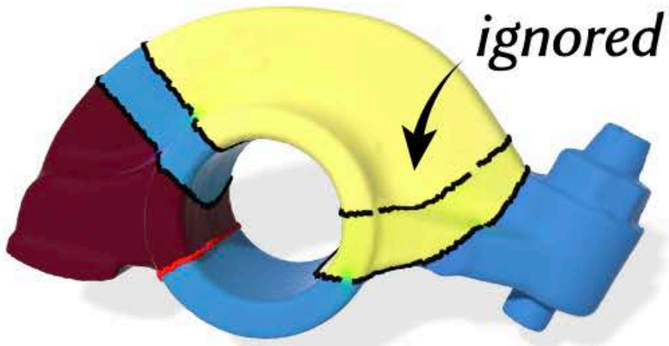




input



ground truth regions




SWN

SWN still gives a valid curve decomposition!
(no single “right” answer.)

# Conclusion

# Takeaways

# Takeaways

- We need algorithms for processing geometry robustly!

# Takeaways

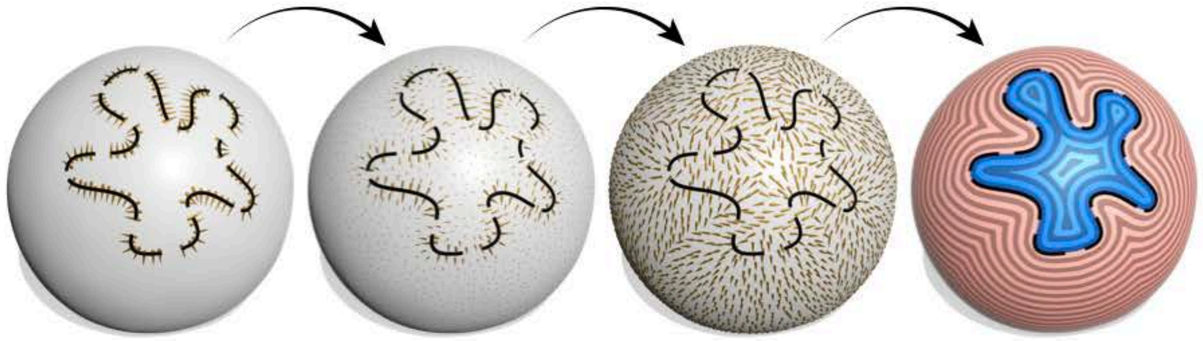
- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry

# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable

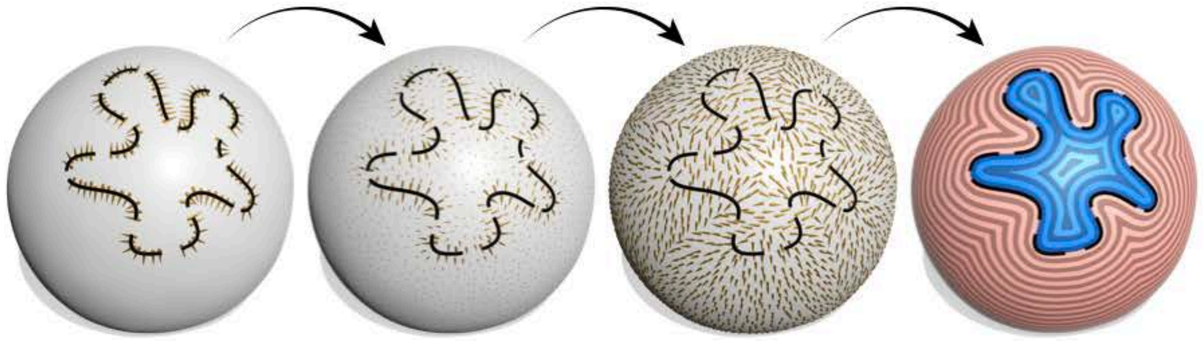
# Takeaways

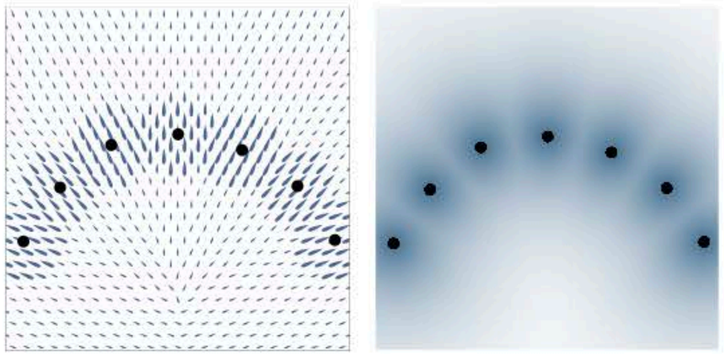
- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable



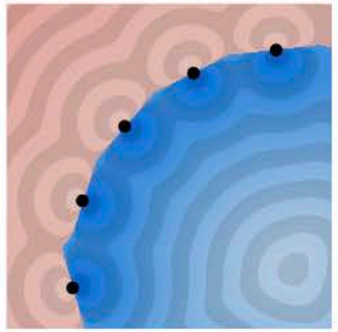
# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable



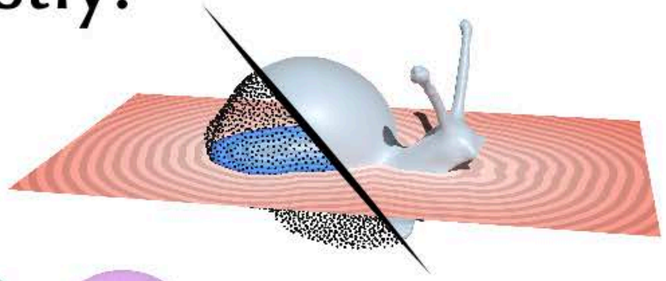



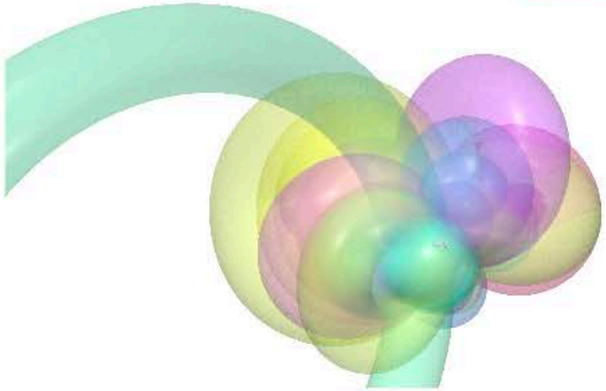
*naive Hopf-Cole*

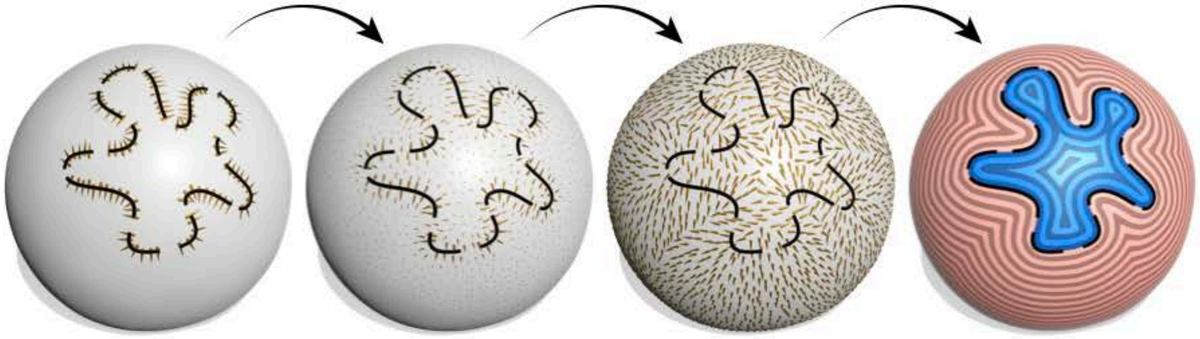


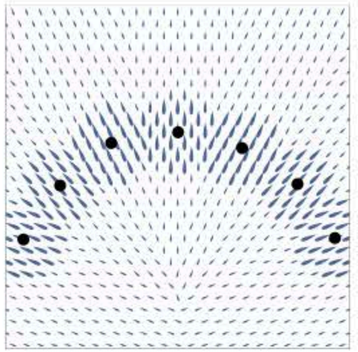
# Takeaways

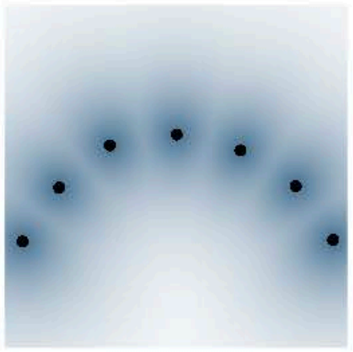
- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable



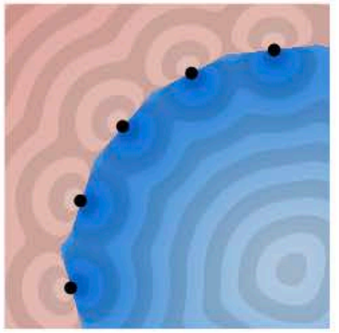






naive Hopf-Cole



# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful!

# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful!

*interpolating data*



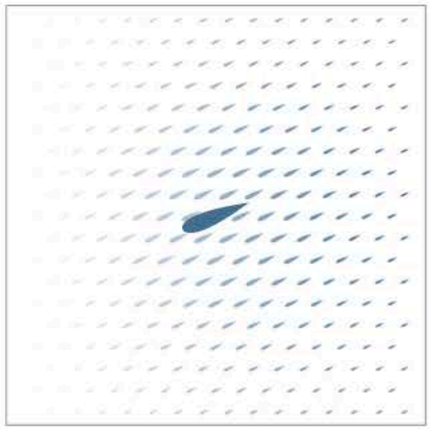
# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful!

*interpolating data*



*parallel transport*



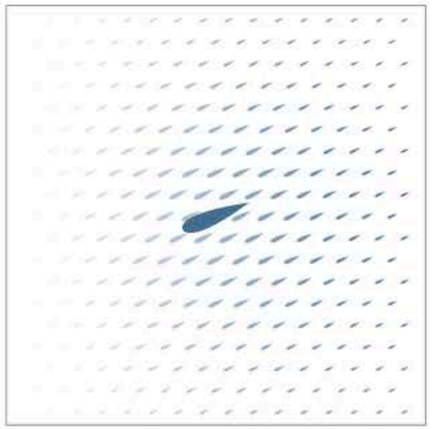
# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful!

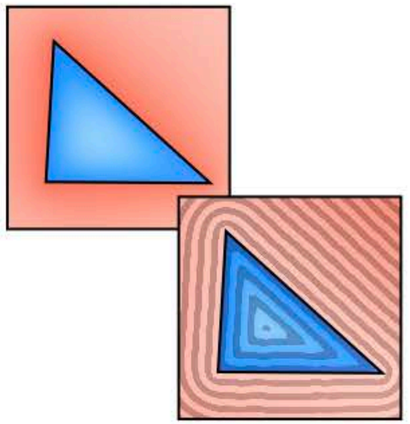
*interpolating data*



*parallel transport*



*geodesic distance*



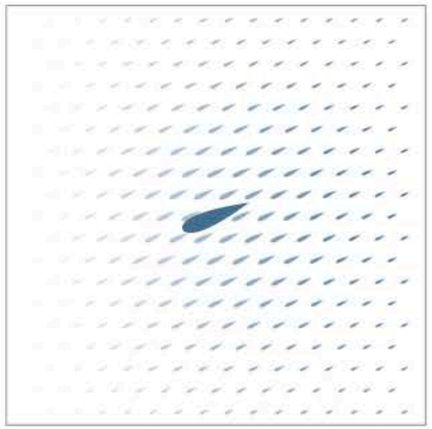
# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful!

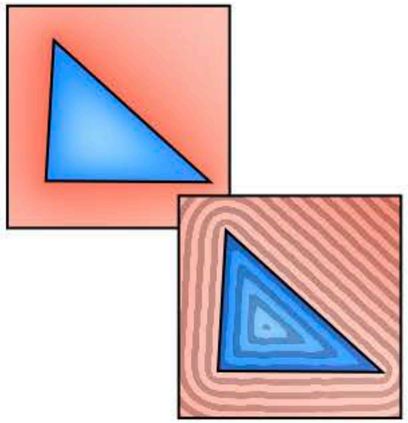
*interpolating data*



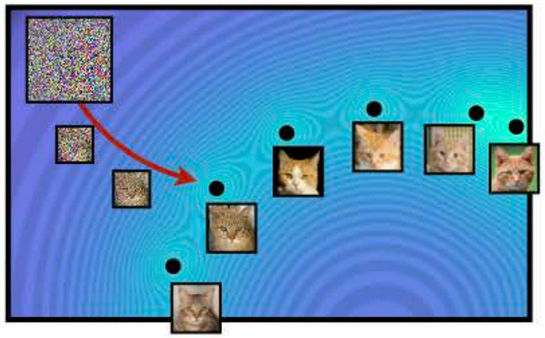
*parallel transport*



*geodesic distance*



*kernel methods at large*



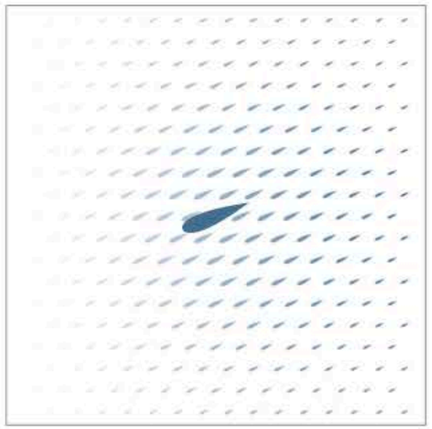
# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful!

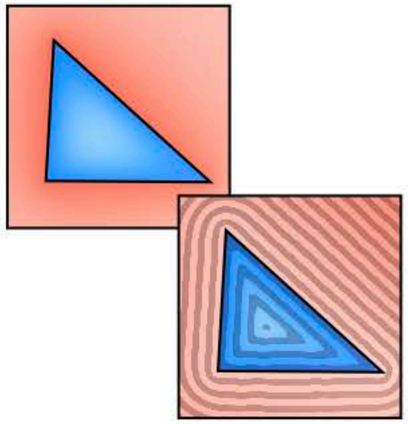
*interpolating data*



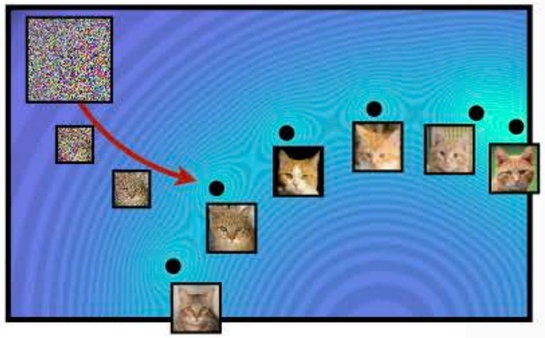
*parallel transport*



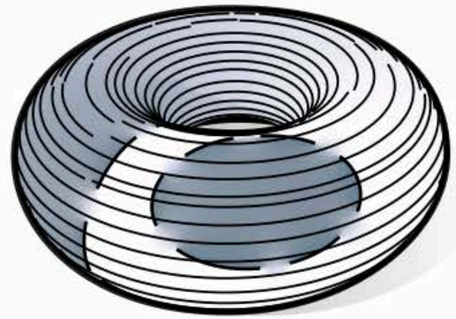
*geodesic distance*



*kernel methods at large*



*topology*



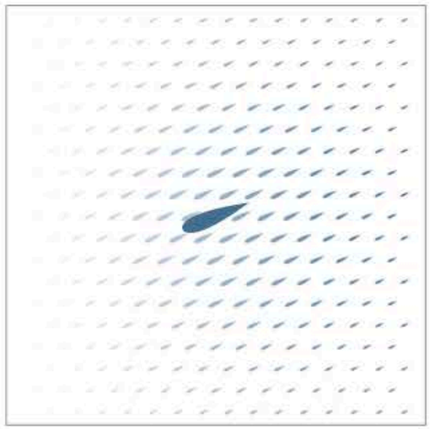
# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful! However, there are nuances.

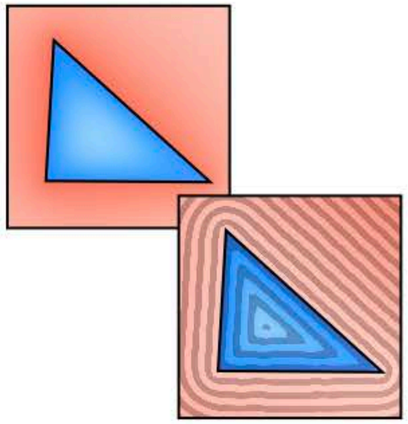
*interpolating data*



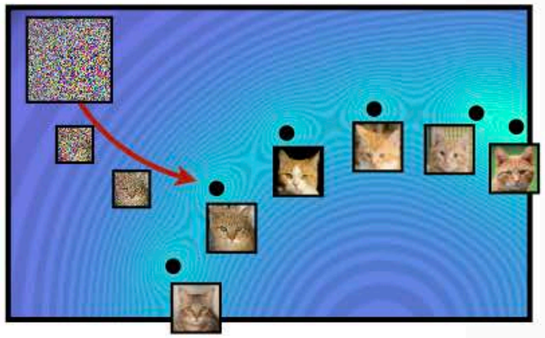
*parallel transport*



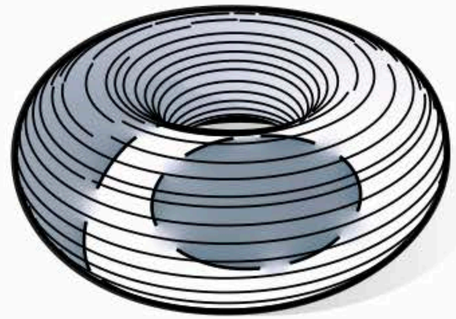
*geodesic distance*



*kernel methods at large*



*topology*



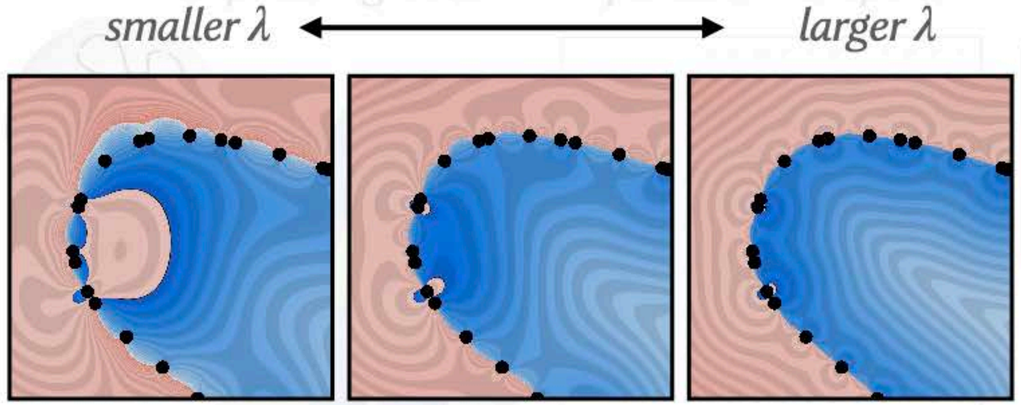
# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful! However, there are nuances.

*interpolating data* *parallel transport* *geodesic distance* *kernel methods at large* *topology*

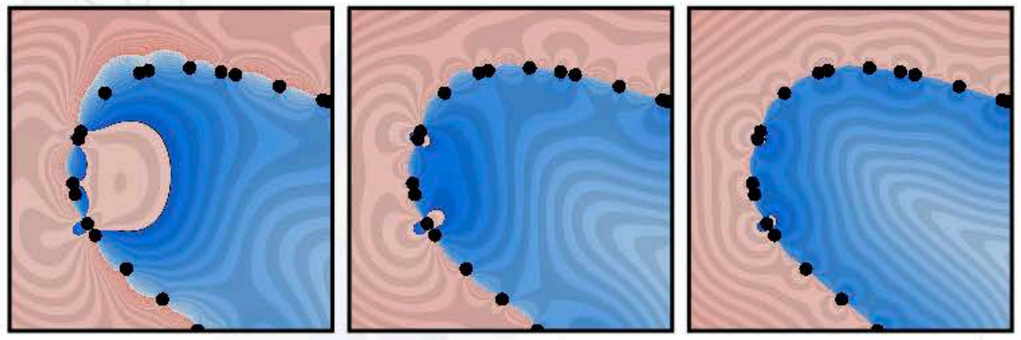
# Takeaways

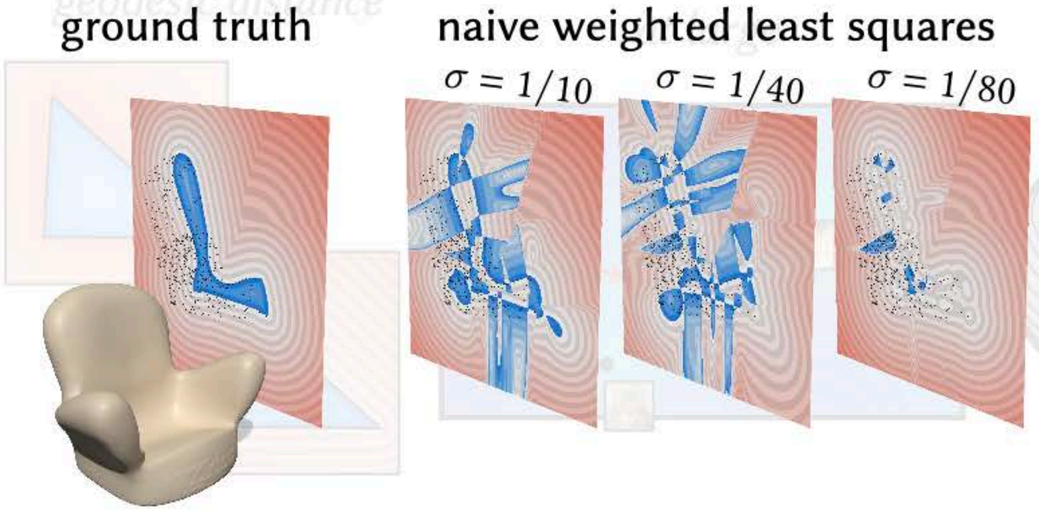
- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful! However, there are nuances.

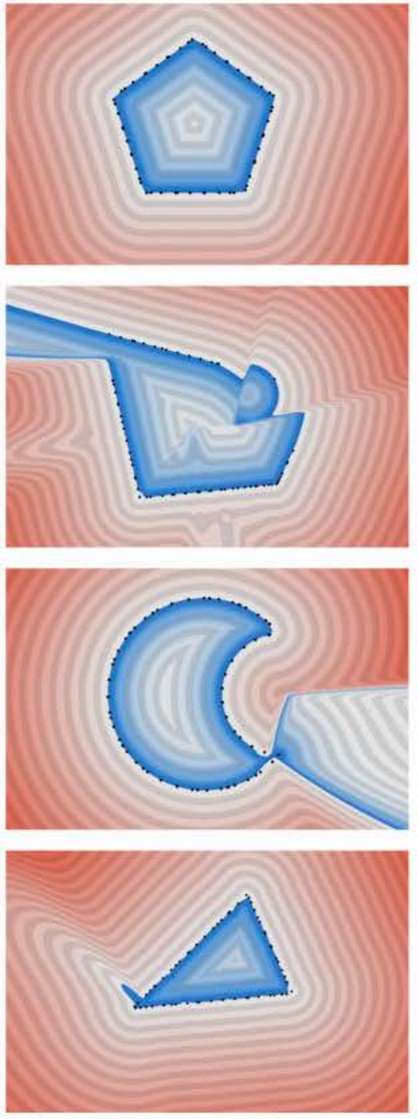


# Takeaways

- We need algorithms for processing geometry robustly!
- Functions are nicer than broken geometry
- Higher-order information is valuable
- Diffusion is powerful! However, there are nuances.



# Technology adoption

# Technology adoption

Static C++ library: signed-heat-3d *(141 GitHub stars)*

# Technology adoption

Static C++ library: signed-heat-3d *(141 GitHub stars)*

Python package: signedheat3d *(~2k downloads)*

# Technology adoption

Static C++ library: signed-heat-3d *(141 GitHub stars)*

Python package: signedheat3d *(~2k downloads)*

```
pip install signedheat3d
```

# Technology adoption

Static C++ library: signed-heat-3d *(141 GitHub stars)*

Python package: signedheat3d *(~2k downloads)*

```
pip install signedheat3d
```

C++ demos: *(61 GitHub stars)*

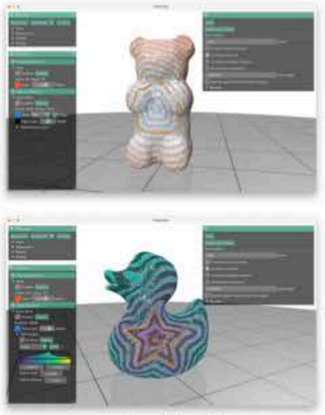
signed-heat-demo

signed-heat-demo-3d



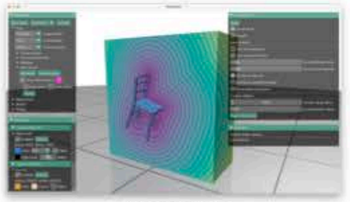
triangle meshes

*mix signed & unsigned distances*

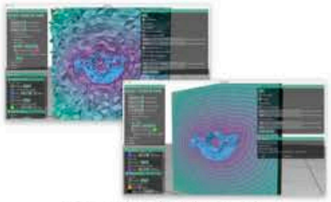


general polygon meshes

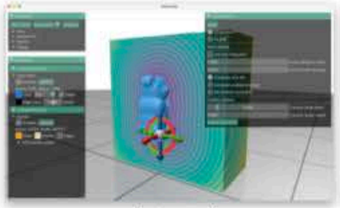
point clouds



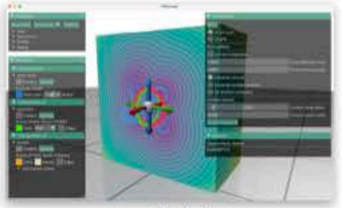
*triangle mesh*



*solve on a tet mesh or grid*



*polygon mesh*



*point cloud*

# Technology adoption

Static C++ library: signed-heat-3d *(141 GitHub stars)*

Python package: signedheat3d *(~2k downloads)*

```
pip install signedheat3d
```

C++ demos: *(61 GitHub stars)*

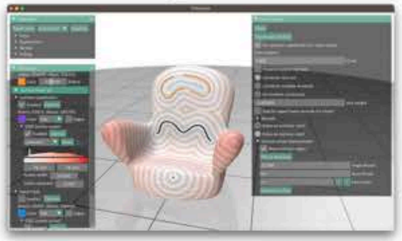
signed-heat-demo

signed-heat-demo-3d

Web-based documentation:

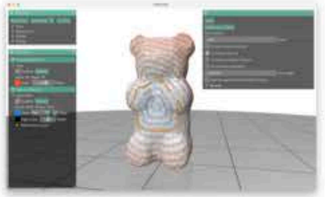
nzfeng.github.io/signed-heat-docs

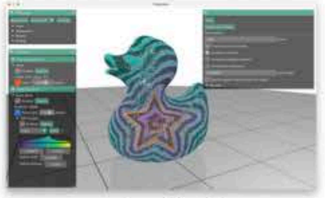
triangle meshes



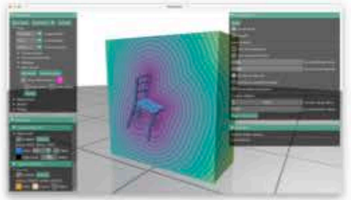
*mix signed & unsigned distances*

general polygon meshes

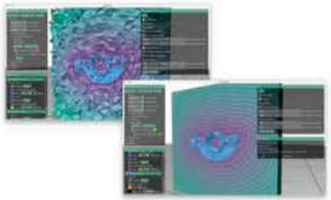




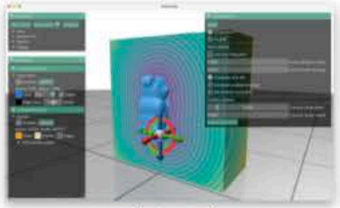
point clouds



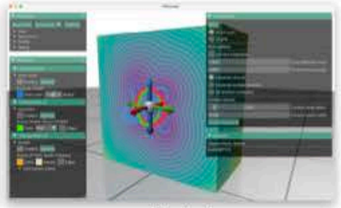
*triangle mesh*



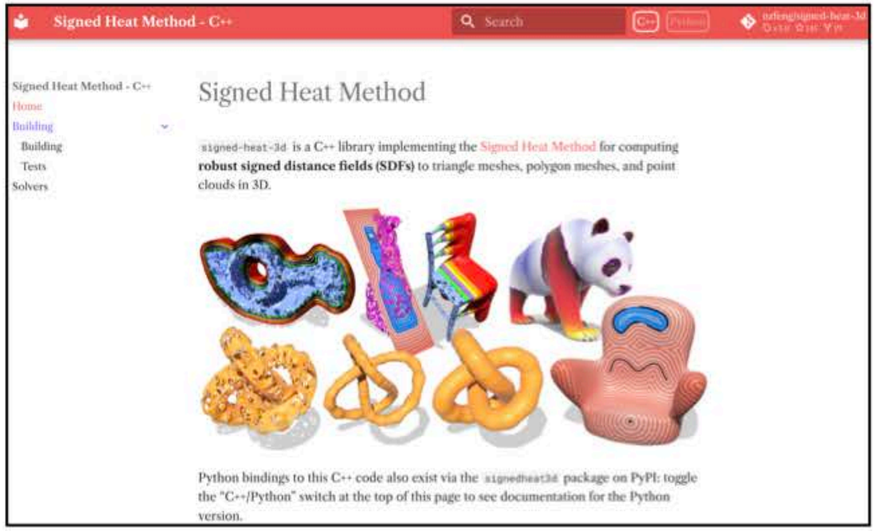
*solve on a tet mesh or grid*



*polygon mesh*



*point cloud*



# Technology adoption

Static C++ library: signed-heat-3d *(141 GitHub stars)*

Python package: signedheat3d *(~2k downloads)*

```
pip install signedheat3d
```

C++ demos: *(61 GitHub stars)*

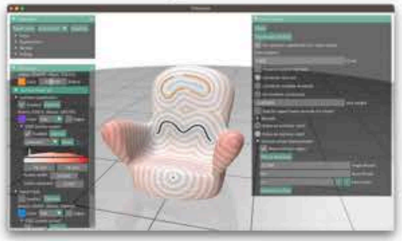
signed-heat-demo

signed-heat-demo-3d

Web-based documentation:

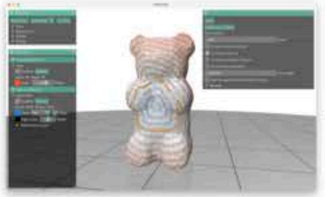
nzfeng.github.io/signed-heat-docs

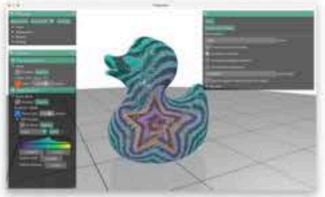
triangle meshes



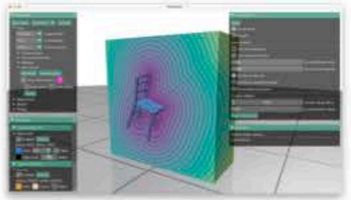
*mix signed & unsigned distances*

general polygon meshes

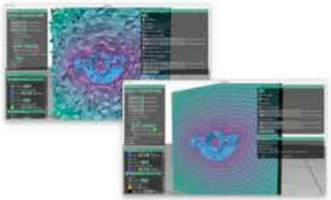




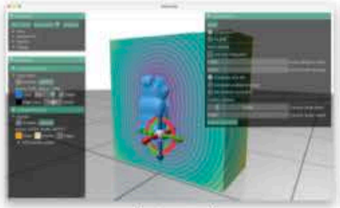
point clouds



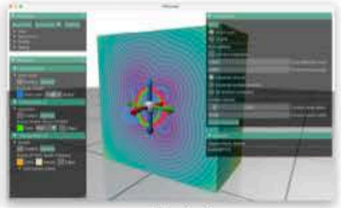
*triangle mesh*



*solve on a tet mesh or grid*



*polygon mesh*



*point cloud*



# Technology adoption

Static C++ library: signed-heat-3d *(141 GitHub stars)*

Python package: signedheat3d *(~2k downloads)*

```
pip install signedheat3d
```

C++ demos: *(61 GitHub stars)*

signed-heat-demo

signed-heat-demo-3d

Web-based documentation:

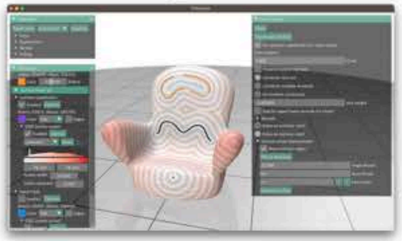
nzfeng.github.io/signed-heat-docs

Integration:

geometry-central *(1.3k GitHub stars)*

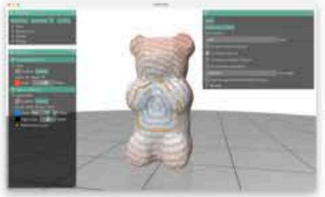
potpourri3d *(567 GitHub stars)*

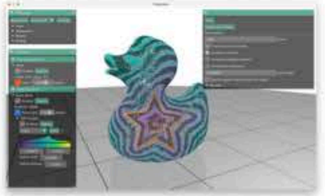
triangle meshes



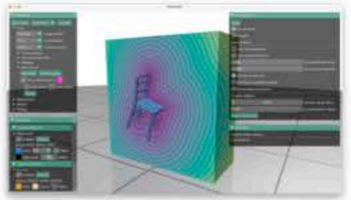
*mix signed & unsigned distances*

general polygon meshes

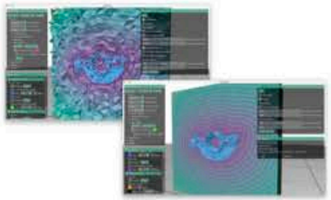




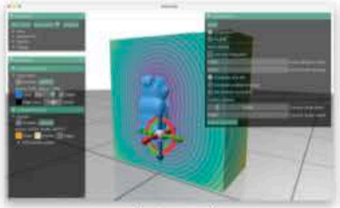
point clouds



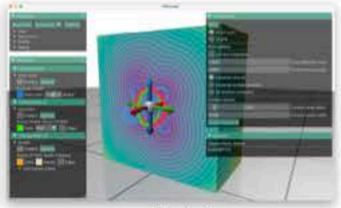
*triangle mesh*



*solve on a tet mesh or grid*



*polygon mesh*



*point cloud*



# Technology adoption

Static C++ library: signed-heat-3d *(141 GitHub stars)*

Python package: signedheat3d *(~2k downloads)*

```
pip install signedheat3d
```

C++ demos: *(61 GitHub stars)*

signed-heat-demo

signed-heat-demo-3d

Web-based documentation:

nzfeng.github.io/signed-heat-docs

Integration:

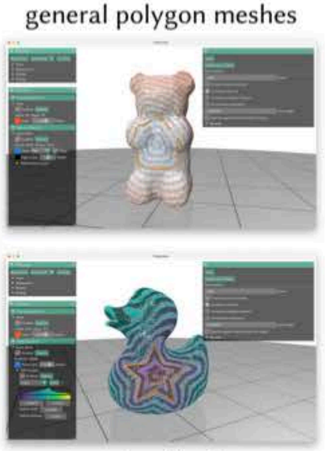
geometry-central *(1.3k GitHub stars)*

potpourri3d *(567 GitHub stars)*

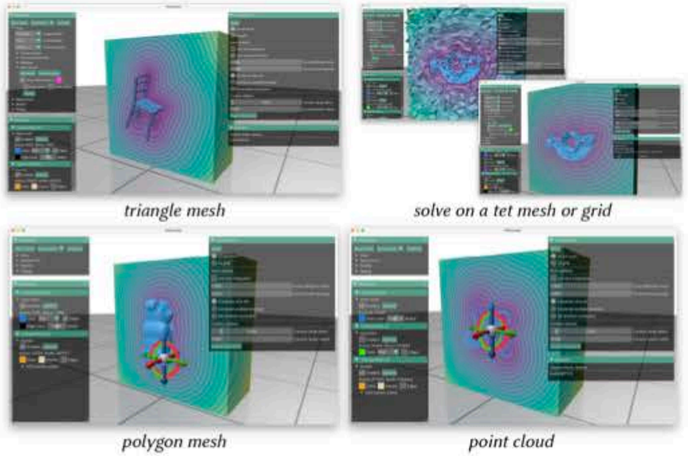
Everything cross-platform (MacOS, Linux, Windows)




*mix signed & unsigned distances*




point clouds






# Technology adoption

Static C++ library: signed-heat-3d *(141 GitHub stars)*

Python package: signedheat3d *(~2k downloads)*

```
pip install signedheat3d
```

C++ demos: *(61 GitHub stars)*

signed-heat-demo

signed-heat-demo-3d

Web-based documentation:

nzfeng.github.io/signed-heat-docs

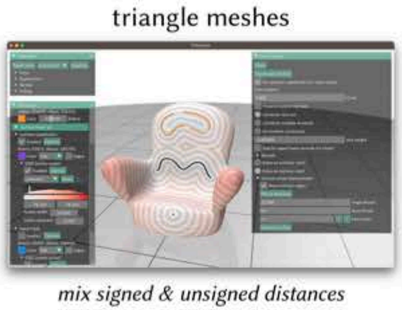
Integration:

geometry-central *(1.3k GitHub stars)*

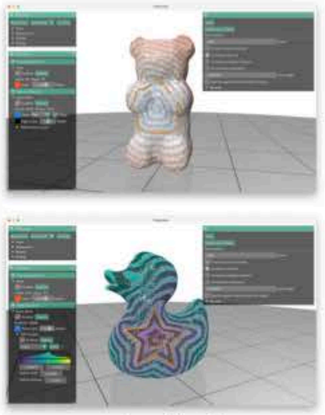
potpourri3d *(567 GitHub stars)*

Everything cross-platform (MacOS, Linux, Windows)

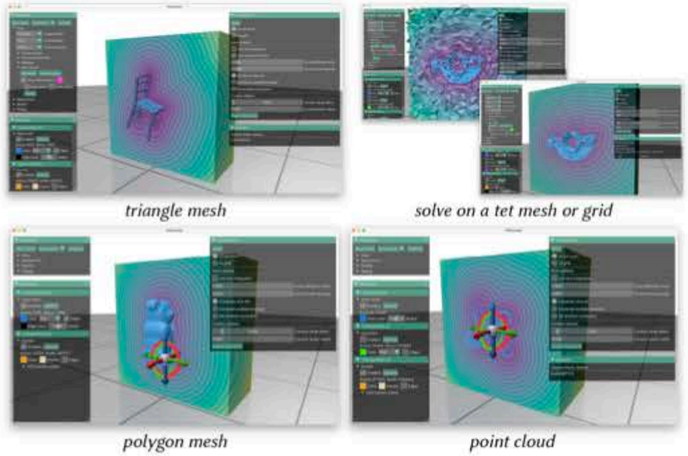
Everything open-source




*mix signed & unsigned distances*




point clouds






# Technology adoption

Static C++ library: signed-heat-3d (

Python package: signedheat3d (~2k

```
pip install signedheat3d
```

C++ demos: (61 GitHub stars)

signed-heat-demo

signed-heat-demo-3d

Web-based documentation:

nzfeng.github.io/signed-heat-docs

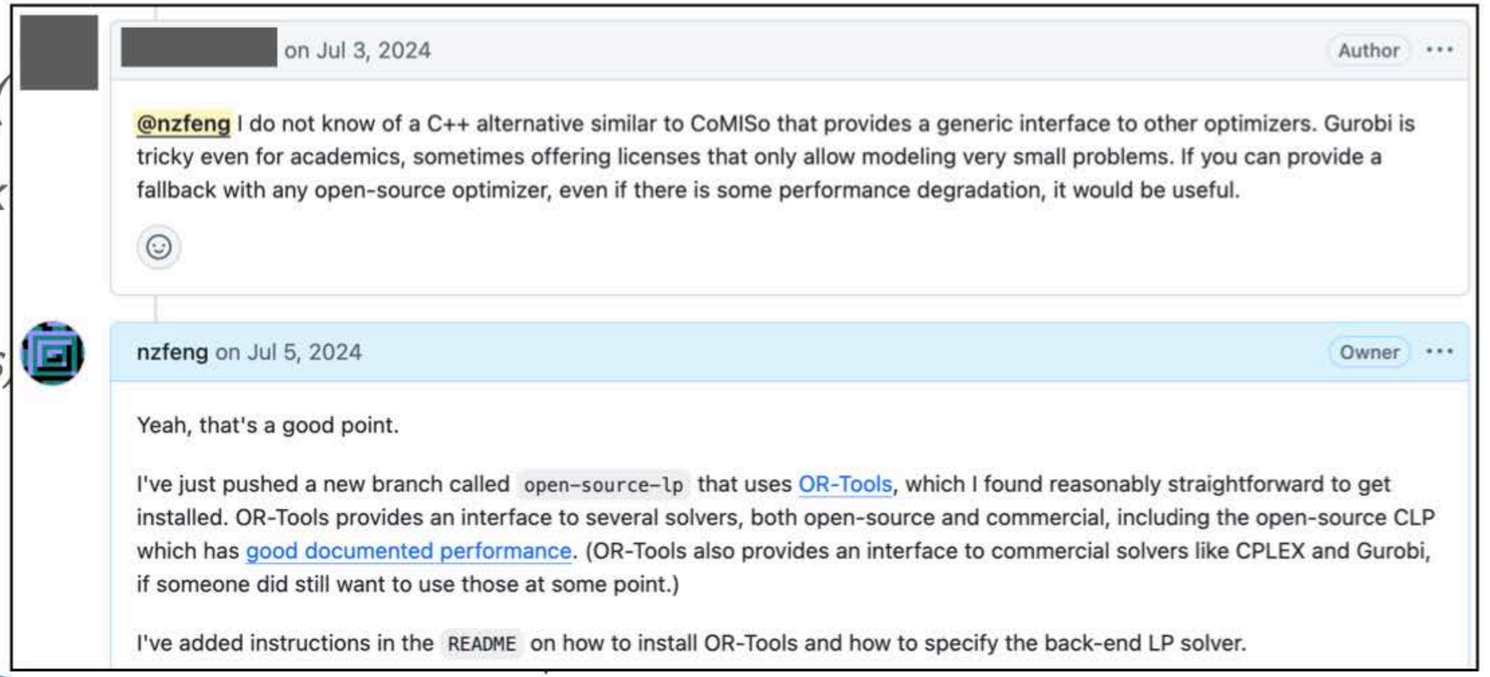
Integration:

geometry-central (1.3k GitHub stars)

potpourri3d (567 GitHub stars)

Everything cross-platform (MacOS, Linux, Windows)

Everything open-source






# Technology adoption

Static C

Python

pip

C++ der

signed

signed

Web-bas

nzfeng

Integrat

geome

potpou

Everythi

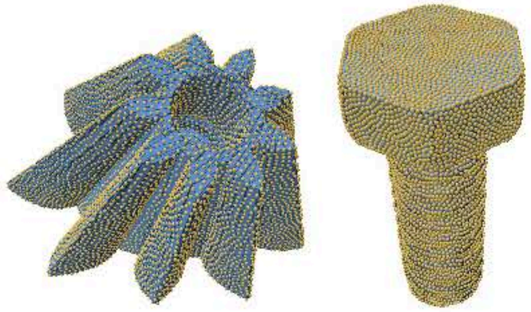
Everythi

improving manufacturability



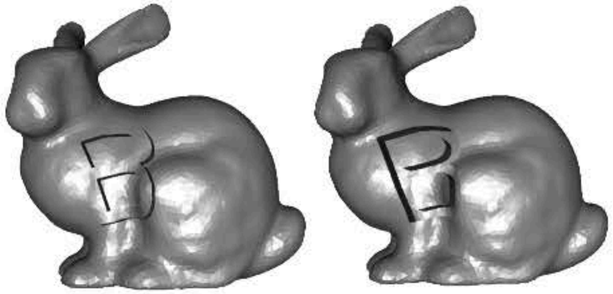
[Wang et al. 2025]

resampling



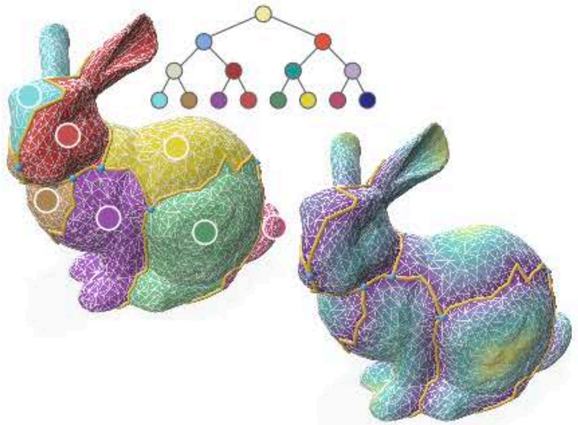
[Suriyababu et al. 2025]

mesh booleans



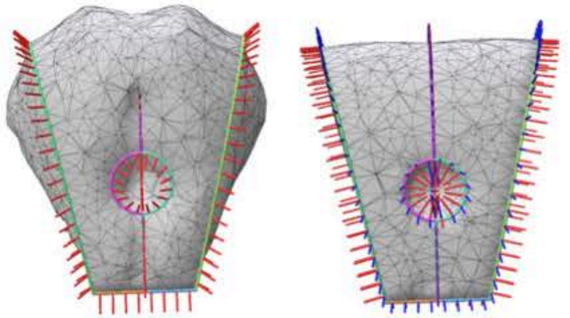
[Jung et al. 2025]

mesh LODs

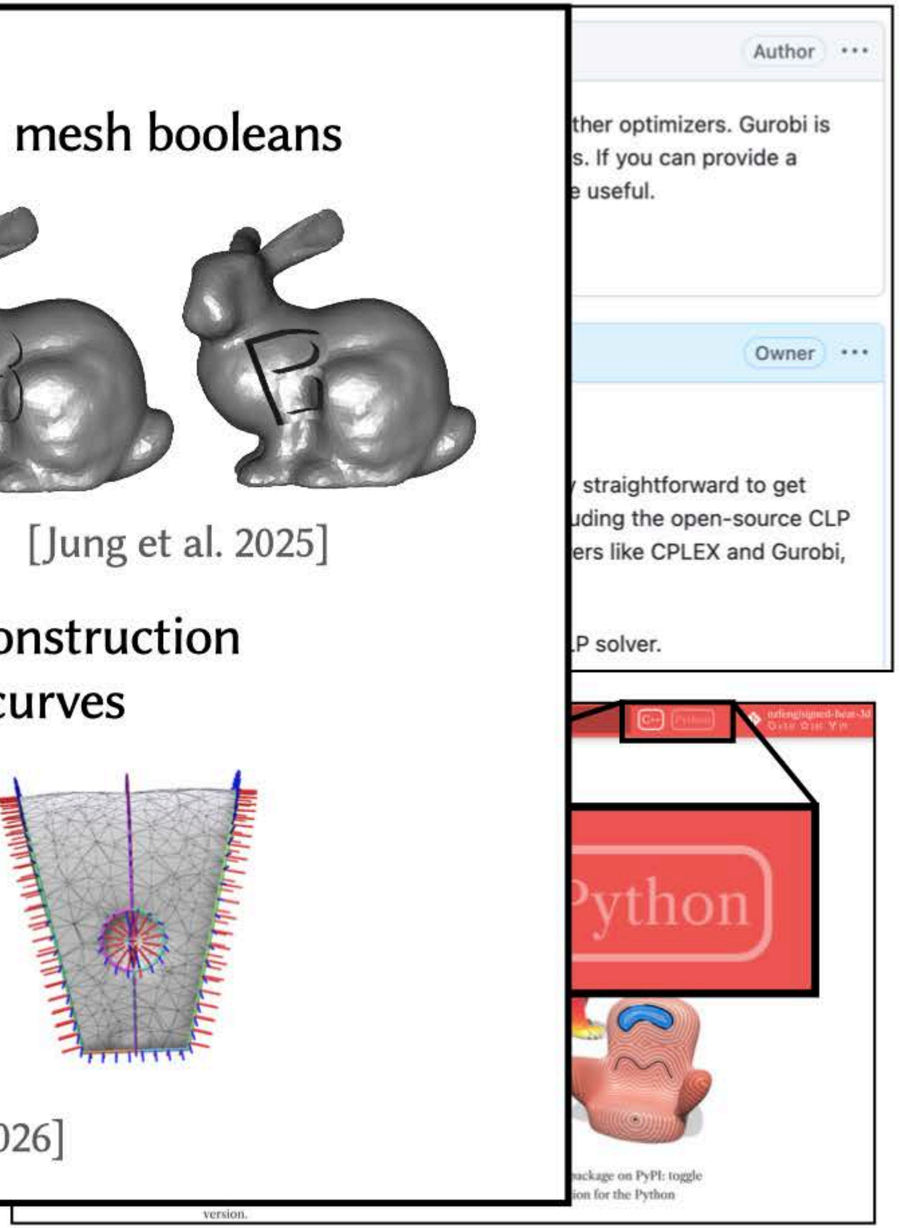


[Ladeuil et al. 2026]

surface reconstruction from curves



[Yu 2026]



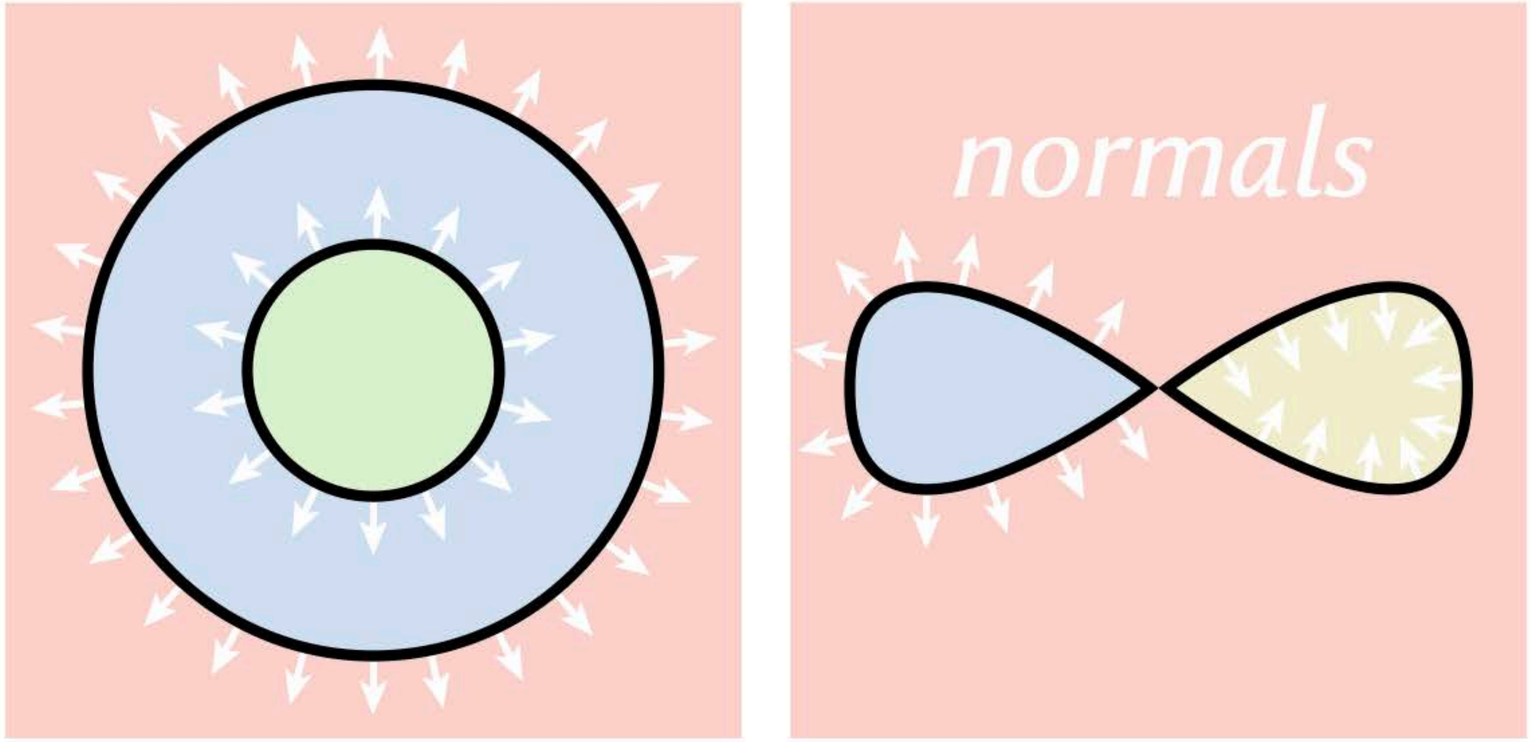
# Limitations and unanswered questions

# Limitations and unanswered questions

- *n*-ary distance

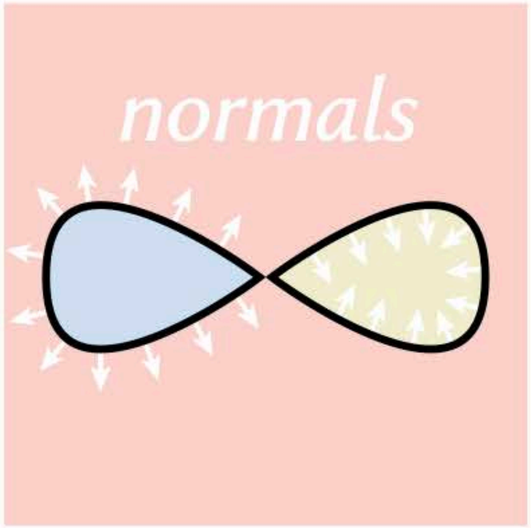
# Limitations and unanswered questions

- *n*-ary distance



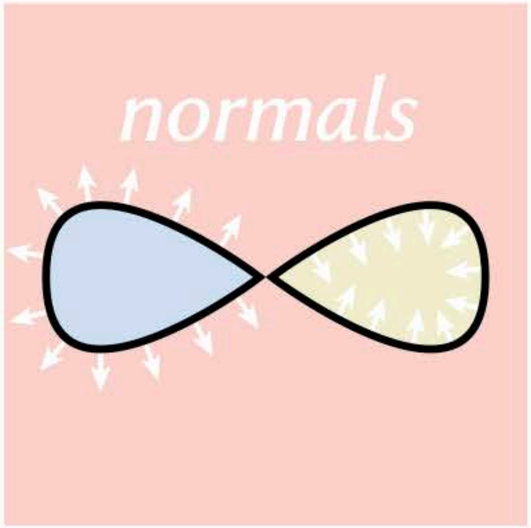
# Limitations and unanswered questions

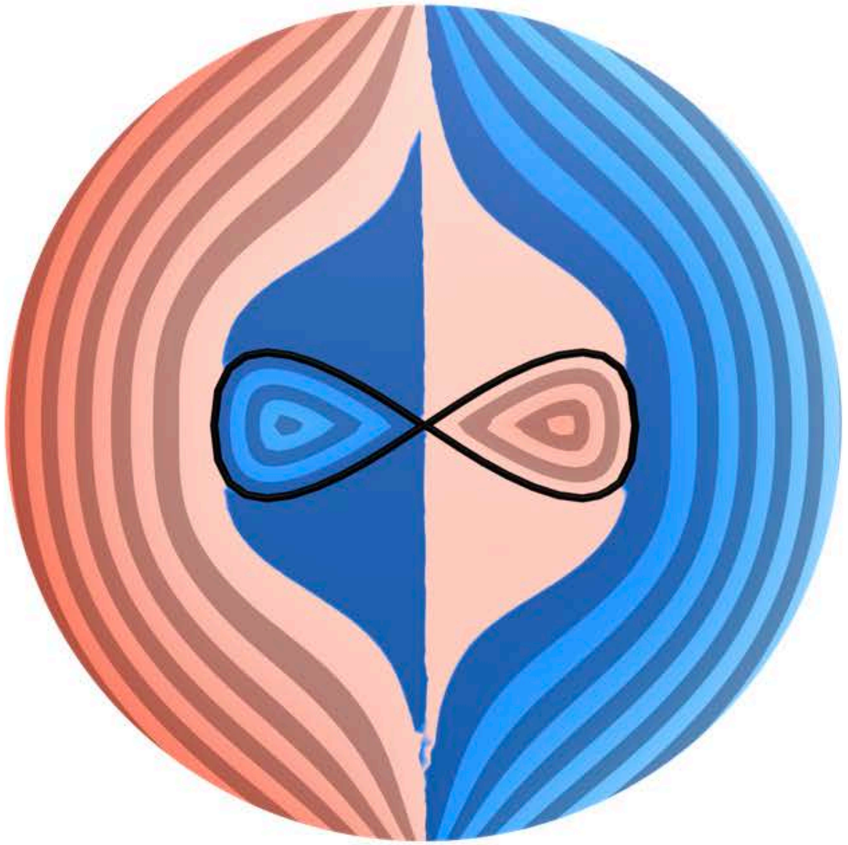
- *n*-ary distance



# Limitations and unanswered questions

- *n*-ary distance

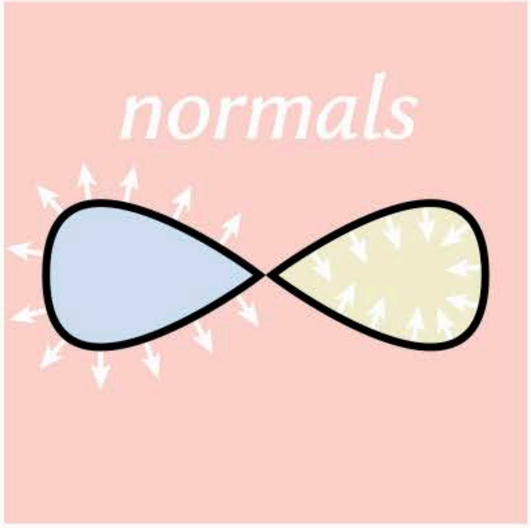


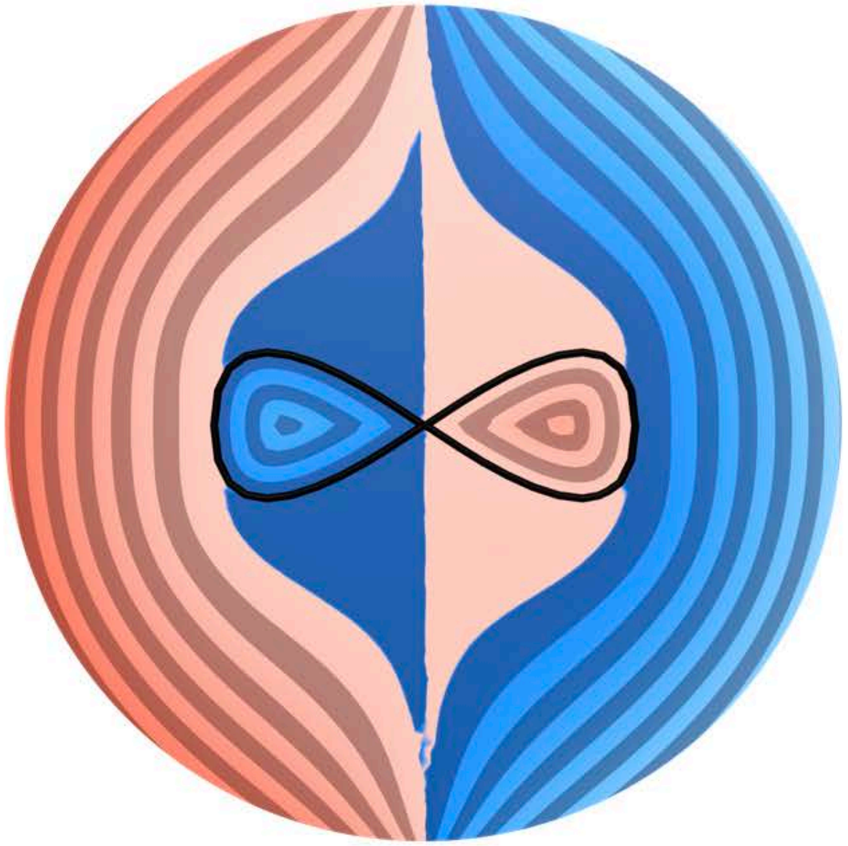



*signed heat method*

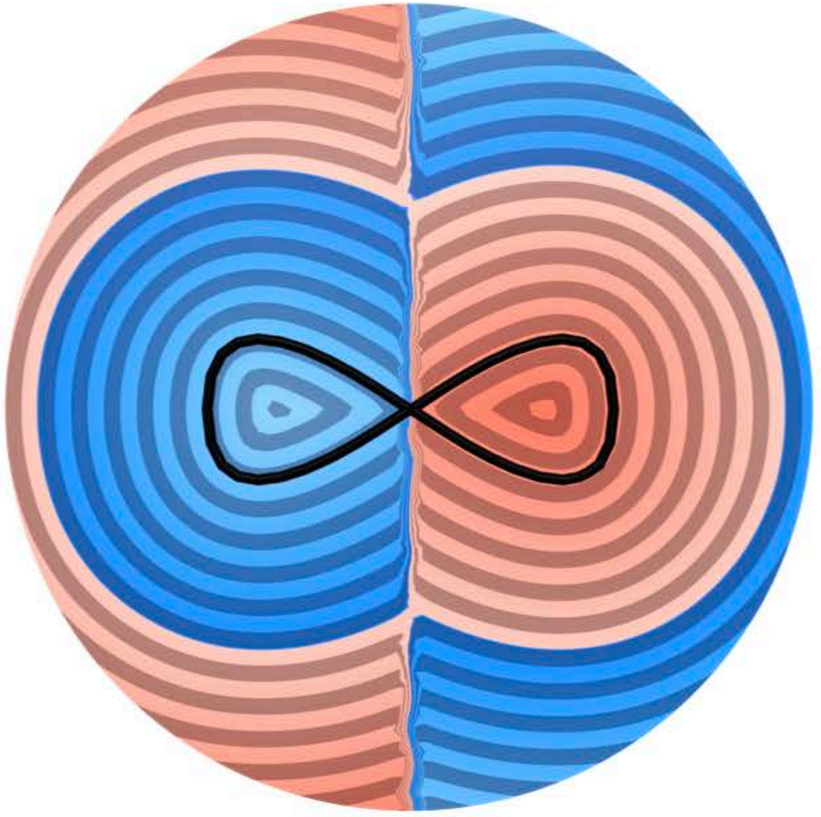
# Limitations and unanswered questions

- *n*-ary distance



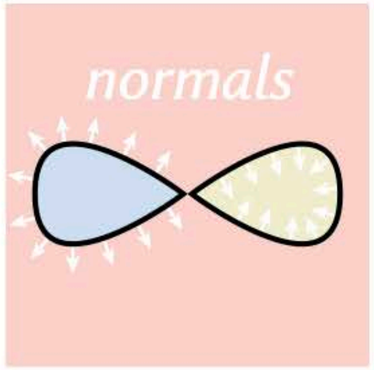
*signed heat method*

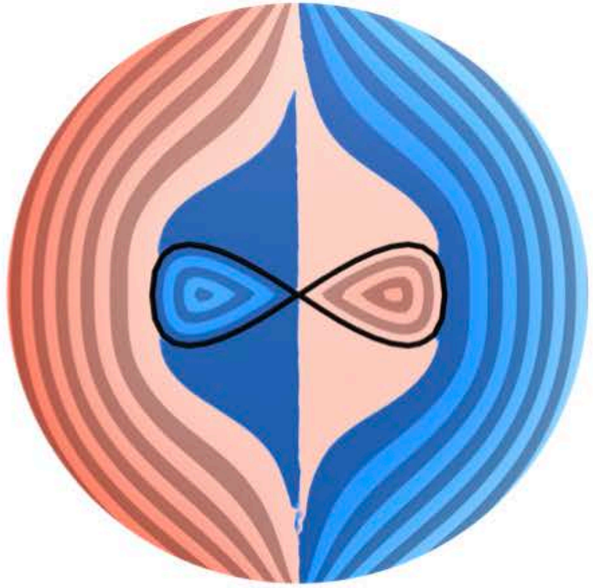


$L^1$ *integration*

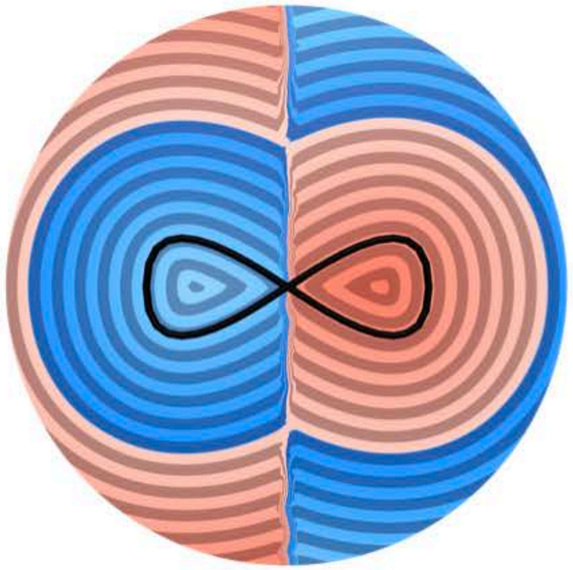
# Limitations and unanswered questions

- *n*-ary distance

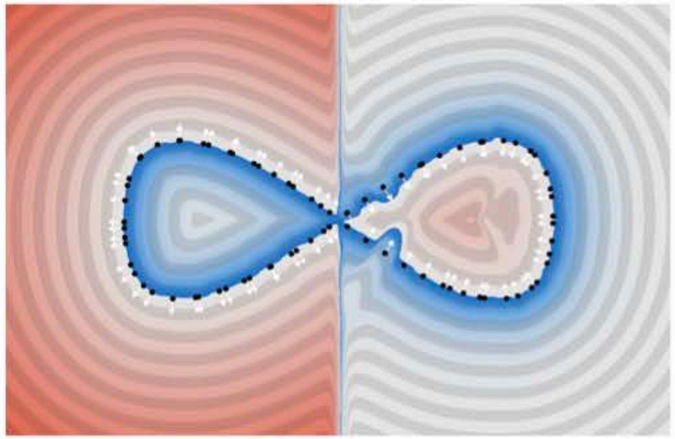


*signed heat method*



$L^1$ *integration*



*PAT**

**preliminary 2D version*

# Limitations and unanswered questions

- *n*-ary distance
- As with all ill-posed problems, we can't guarantee the true solution

# Limitations and unanswered questions

- *n*-ary distance
- As with all ill-posed problems, we can't guarantee the true solution

*Statistical geometry processing:*

2004 Pauly et al., *Uncertainty and variability in point cloud surface data*
2004 Hofsetz et al., *Image-based rendering of range data with estimated depth uncertainty*
2006 Jenke et al., *Bayesian point cloud reconstruction*
2011 Pöthkow et al., *Probabilistic marching cubes*
2021 Senin et al., *Statistical point cloud model to investigate measurement uncertainty in coordinate metrology*
2022 Sellán & Jacobson, *Stochastic Poisson surface reconstruction*
…

# Limitations and unanswered questions

- *n*-ary distance
- As with all ill-posed problems, we can't guarantee the true solution
- Pointwise evaluation can/should be accelerated

# Limitations and unanswered questions

- *n*-ary distance
- As with all ill-posed problems, we can't guarantee the true solution
- Pointwise evaluation can/should be accelerated

*Precompute times:*

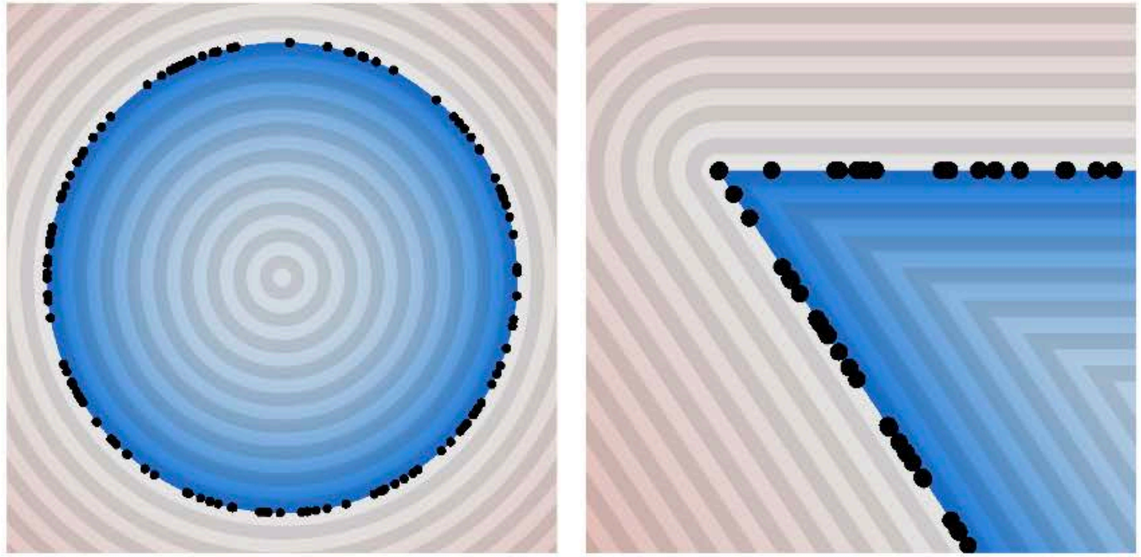
- <10s for 100k points
- <1 min for 1M points
- ~1-4 min for 5-10M points
- >10 min for 20M+ points

# Limitations and unanswered questions

- *n*-ary distance
- As with all ill-posed problems, we can't guarantee the true solution
- Pointwise evaluation can/should be accelerated

*Precompute times:*

- <10s for 100k points
- <1 min for 1M points
- ~1-4 min for 5-10M points
- >10 min for 20M+ points

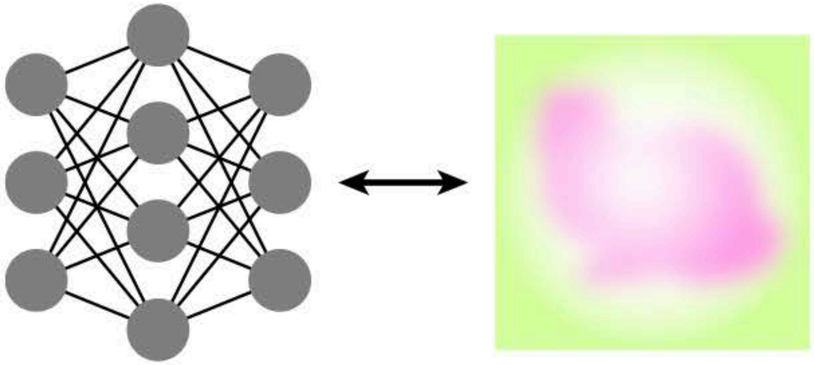


*Surely many shapes can be represented with fewer points and associated tori…*

# Broader future work: geometric optimization

# Broader future work: geometric optimization

*neural fields* $\stackrel{?}{=}$ *discretization-free*, infinite-resolution*, differentiable*, compressed**



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

Yang et al. 2021, *Geometry Processing with Neural Fields*

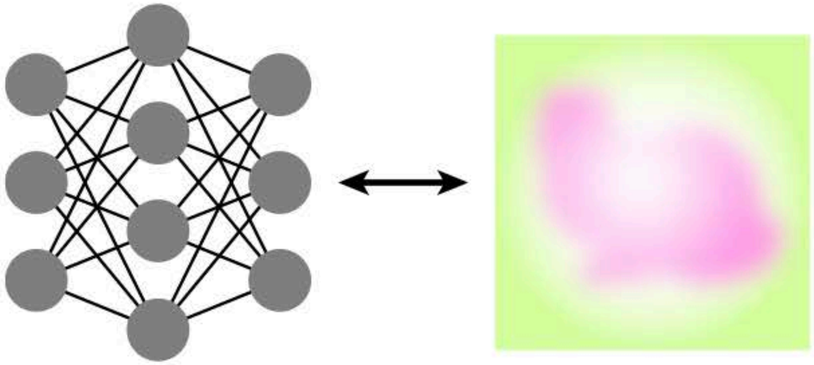
Mehta et al. 2022, *A Level Set Theory for Neural Implicit Evolution under Explicit Flows*

Guillard et al. 2024, *DeepMesh: Differentiable Iso-Surface Extraction*

...

# Broader future work: geometric optimization

*neural fields* $\stackrel{?}{=}$ *discretization-free*, infinite-resolution*, differentiable*, compressed**



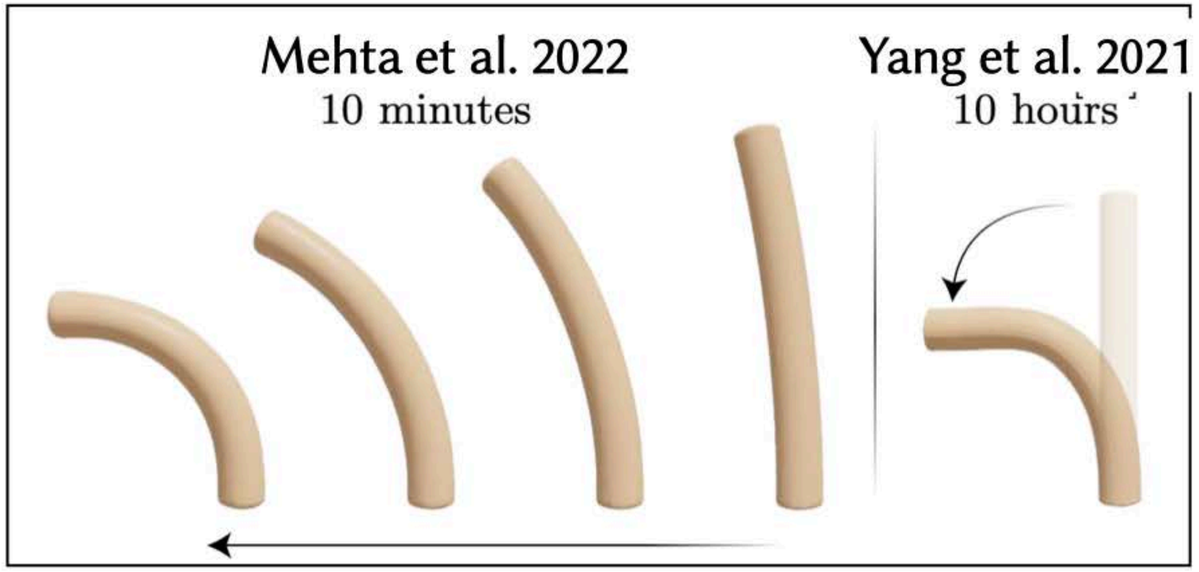
$$f : \mathbb{R}^3 \to \mathbb{R}$$

Yang et al. 2021, *Geometry Processing with Neural Fields*

Mehta et al. 2022, *A Level Set Theory for Neural Implicit Evolution under Explicit Flows*

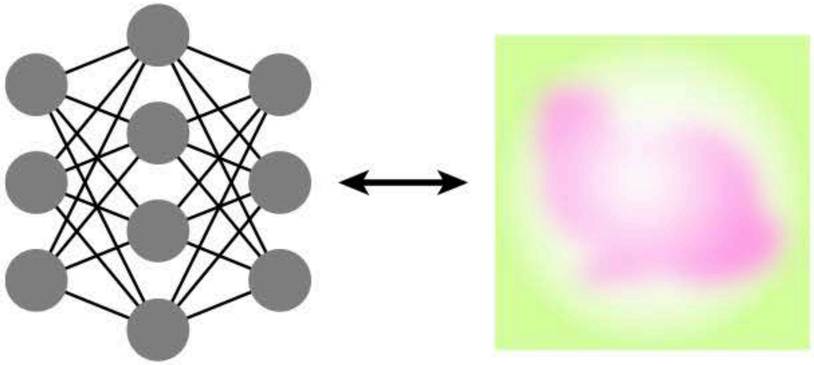
Guillard et al. 2024, *DeepMesh: Differentiable Iso-Surface Extraction*

...



# Broader future work: geometric optimization

*neural fields* $\stackrel{?}{=}$ *discretization-free*, infinite-resolution*, differentiable*, compressed**



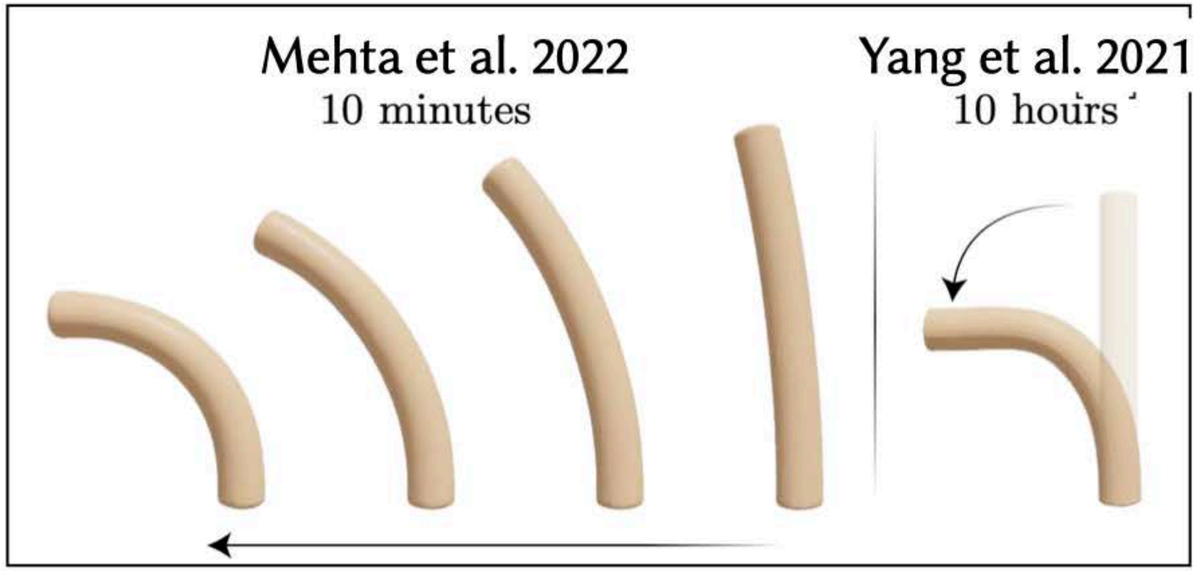
$$f : \mathbb{R}^3 \to \mathbb{R}$$

Yang et al. 2021, *Geometry Processing with Neural Fields*

Mehta et al. 2022, *A Level Set Theory for Neural Implicit Evolution under Explicit Flows*

Guillard et al. 2024, *DeepMesh: Differentiable Iso-Surface Extraction*

...



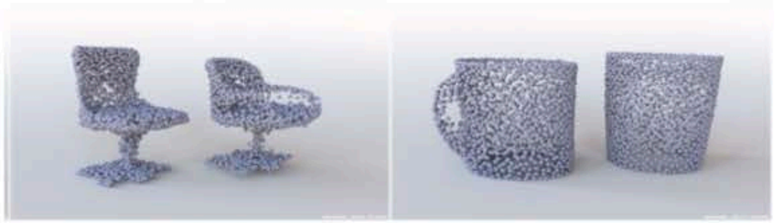
# Broader future work: geometric optimization

*Chen & Zhang* 2019, *Learning Implicit Fields for Generative Shape Modeling*



*implicit functions?*

Achlioptas et al. 2018, *Learning representations and generative models for 3d point clouds*



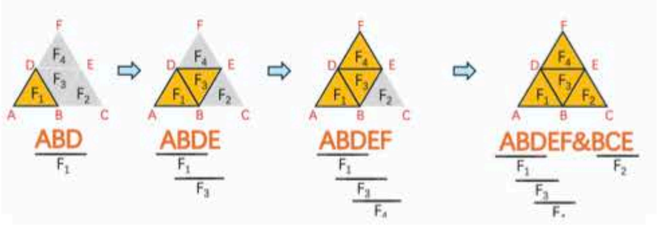
*point clouds?*

Li et al. 2025, *Sparc3D: Sparse Representation and Construction for High-Resolution 3D Shapes Modeling*



*voxel-based?*

*Chen et al.* 2025, *MeshAnything V2: Artist-Created Mesh*




*tokenize meshes?*

# Broader future work: geometric optimization

*Chen & Zhang* 2019, *Learning Implicit Fields for Generative Shape Modeling*



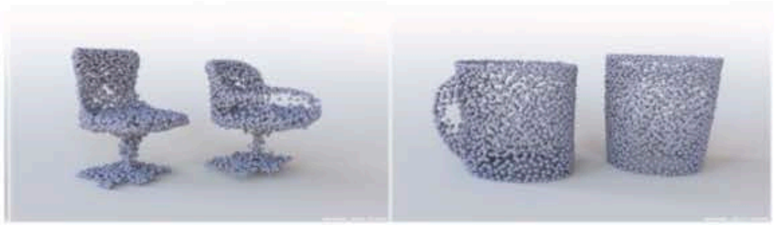
*implicit functions?*

Li et al. 2025, *Sparc3D: Sparse Representation and Construction for High-Resolution 3D Shapes Modeling*



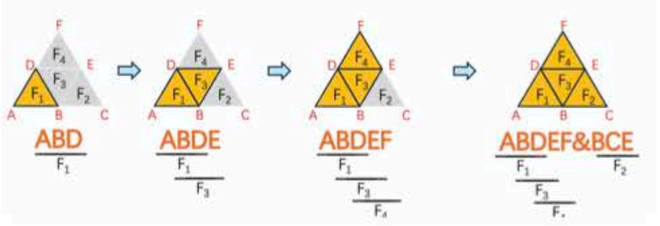
*voxel-based?*

Achlioptas et al. 2018, *Learning representations and generative models for 3d point clouds*



*point clouds?*

*Chen et al.* 2025, *MeshAnything V2: Artist-Created Mesh*

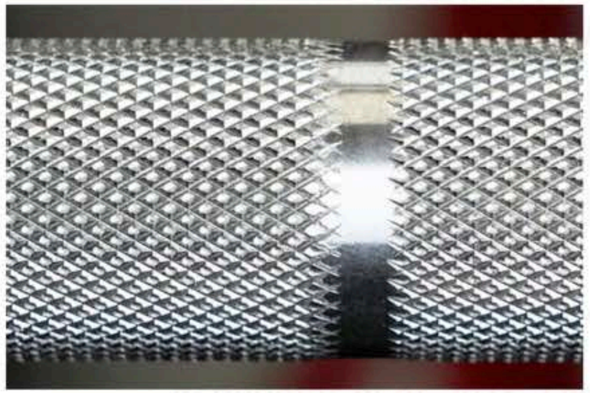


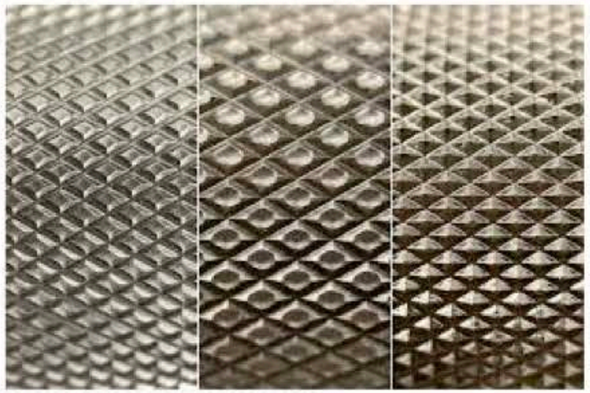

*tokenize meshes?*

A very active research area!

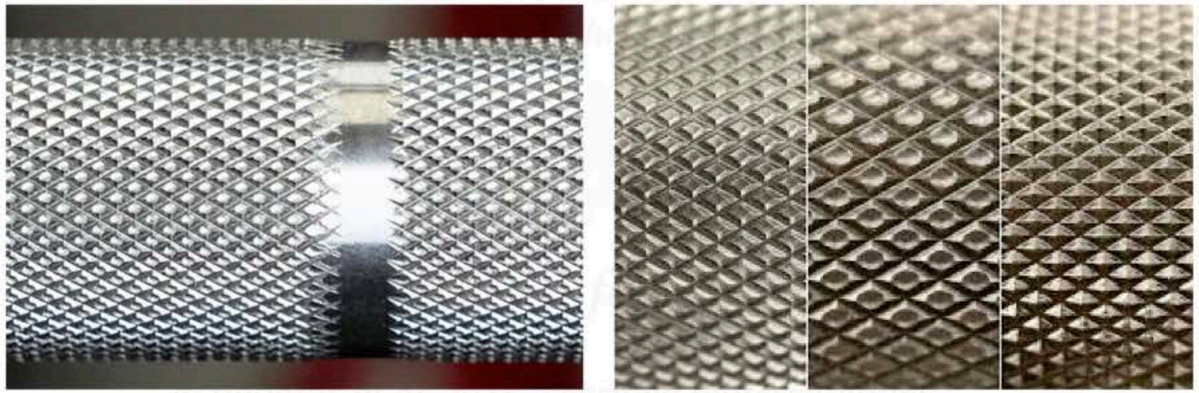
It's not clear yet what's the "best" approach for encoding 3D geometry.

# Broader future work: geometric optimization

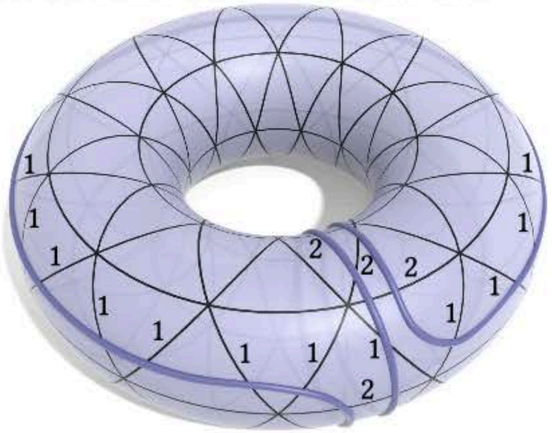




# Broader future work: geometric optimization



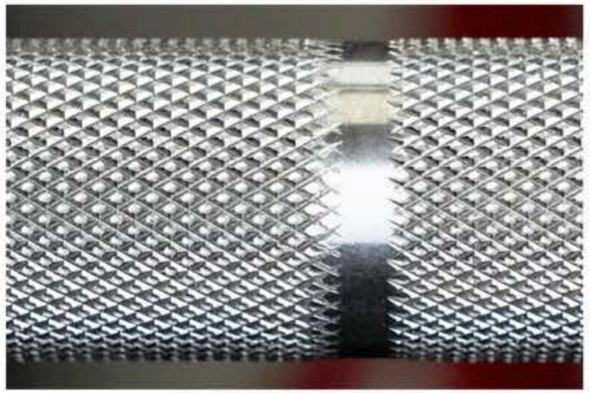
*normal coordinates*

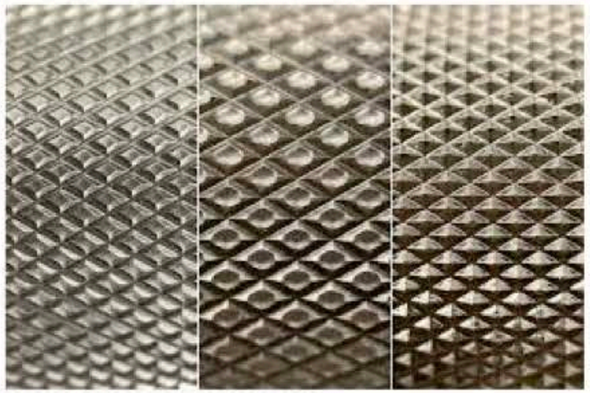



Gillespie et al. 2021, *Integer Coordinates for Intrinsic Geometry Processing*

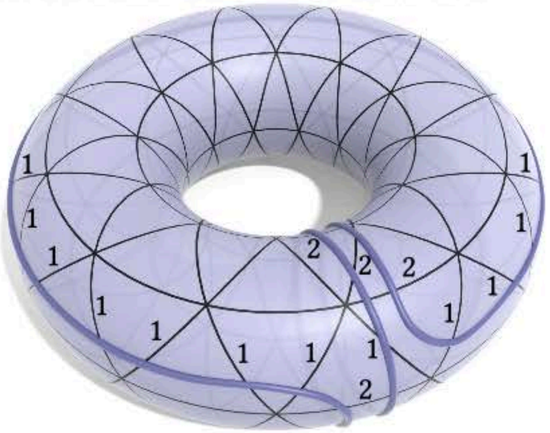
Baktash et al. 2026, *Subgrid Marching Tetrahedra*

# Broader future work: geometric optimization





## *normal coordinates*



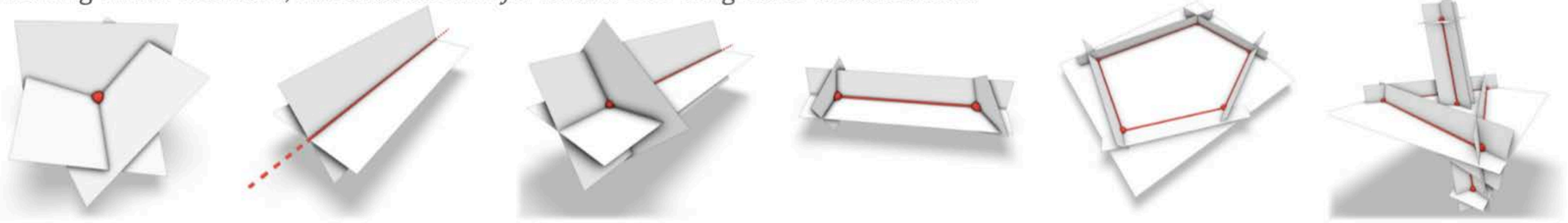

Gillespie et al. 2021, *Integer Coordinates for Intrinsic Geometry Processing*

Baktash et al. 2026, *Subgrid Marching Tetrahedra*

## *Nef polyhedra*

Trettner et al. 2022, *EMBER: exact mesh booleans via efficient & robust local arrangements*

Nehring-Wirxel et al. 2021, *Fast exact booleans for iterated CSG using octree-embedded BSPs*



(a) point (b) line (c) ray (d) segment (e) polygon (f) intersection

# Thanks!