Microstructure analysis with GPUs

Jean Feydy
HeKA team, Inria Paris
Inserm, Université Paris-Cité

3rd of December, 2024
Multiscale Mechanics and Artificial Microstructures, Edinburgh

Who am I?

Background in **mathematics** and **data sciences**:

2012–2016 ENS Paris, mathematics.

2014–2015 M2 mathematics, vision, learning at ENS Cachan.

2016–2019 PhD thesis in **medical imaging** with Alain Trouvé at ENS Cachan.

2019–2021 **Geometric deep learning** with Michael Bronstein at Imperial College.

2021+ **Medical data analysis** in the HeKA INRIA team (Paris).

HeKA : a translational research team for public health

Hôpitaux

Inria Inserm

Universités

My main motivation

Develop **robust and efficient** software that **stimulates other researchers**:

1. Speed up **geometric machine learning** on GPUs:

   $\Longrightarrow$ **pyKeOps** library for distance and kernel matrices, 700k+ downloads.

2. Scale up **pharmacovigilance** to the full French population:

   $\Longrightarrow$ **survivalGPU**, a fast re-implementation of the R survival package.

3. Ease access to modern statistical **shape analysis**:

   $\Longrightarrow$ **GeomLoss**, truly scalable optimal transport in Python.

   $\Longrightarrow$ **scikit-shapes**, alpha release now available.

Today’s talk – assuming that you would enjoy some nice simulations

1. A quick heads up on **fast geometric methods**.
2. Efficient discrete optimal transport **solvers**.
3. Applications to systems of **incompressible particles**.
4. Applications to the **synthesis of biological microstructures**.

How to code a N-body simulation?

Do you want to play with galaxies and (modified) gravity? [Pri11]

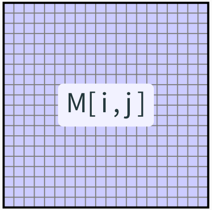
Scientific computing libraries represent most objects as tensors

Context. Constrained **memory accesses** on the GPU:

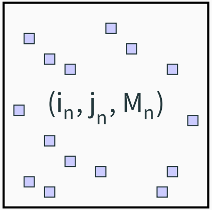
- **Long access times** to the registers penalize the use of large **dense** arrays.
- Hard-wired **contiguous** memory accesses penalize the use of **sparse** matrices.

Challenge. In order to reach optimal run times:

- **Restrict** ourselves to operations that are supported by the constructor: convolutions, FFT, etc.
- Develop new routines from scratch in C++/CUDA (FAISS, KPConv…): **several months of work**.



Dense array



Sparse matrix

The KeOps library: efficient support for symbolic matrices

Solution. KeOps – `www.kernel-operations.io`:

- For PyTorch, NumPy, Matlab and R, on **CPU and GPU**.
- **Automatic differentiation**.
- Just-in-time **compilation** of **optimized** C++ schemes, triggered for every new **reduction**: sum, min, etc.

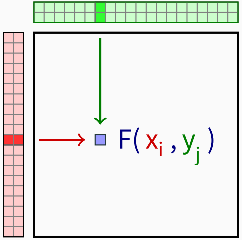
If the formula "F" is simple ($\leqslant$ 100 arithmetic operations):

"100k $\times$ 100k" computation $\rightarrow$ 10ms – 100ms,

"1M $\times$ 1M" computation $\rightarrow$ 1s – 10s.

Hardware ceiling of 10^{12} operations/s.

$\times$**10 to** $\times$**100 speed-up** vs standard GPU implementations for a wide range of problems.



Symbolic matrix

Formula + data

- Distances $d(x_i, y_j)$.
- Kernel $k(x_i, y_j)$.
- Numerous transforms.

A first example: efficient nearest neighbor search in dimension 50

Create large point clouds using **standard PyTorch syntax**:

```
import torch
N, M, D = 10**6, 10**6, 50
x = torch.rand(N, 1, D).cuda()  # (1M,  1, 50) array
y = torch.rand(1, M, D).cuda()  # ( 1, 1M, 50) array
```

Turn **dense** arrays into **symbolic** matrices:

```
from pykeops.torch import LazyTensor
x_i, y_j = LazyTensor(x), LazyTensor(y)
```

Create a large **symbolic matrix** of squared distances:

```
D_ij = ((x_i - y_j) ** 2).sum(dim=2)  # (1M, 1M) symbolic
```

Use an `.argmin()` **reduction** to perform a nearest neighbor query:

```
indices_i = D_ij.argmin(dim=1)  # -> standard torch tensor
```

The KeOps library combines performance with flexibility

Script of the previous slide = efficient nearest neighbor query,
on par with the bruteforce CUDA scheme of the **FAISS** library…
And can be used with **any metric**!

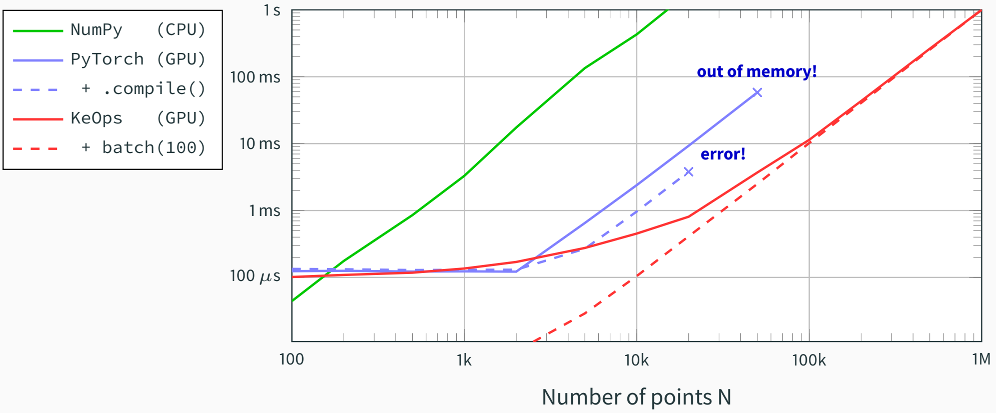
```
D_ij = ((x_i - x_j) ** 2).sum(dim=2)      # Euclidean
M_ij =  (x_i - x_j).abs().sum(dim=2)      # Manhattan
C_ij = 1 - (x_i | x_j)                    # Cosine
H_ij = D_ij / (x_i[...,0] * x_j[...,0])   # Hyperbolic
```

KeOps supports arbitrary **formulas** and **variables** with:

- **Reductions:** sum, log-sum-exp, K-min, matrix-vector product, etc.
- **Operations:** $+$, $\times$, sqrt, exp, neural networks, etc.
- **Advanced schemes:** batch processing, block sparsity, etc.
- **Automatic differentiation:** seamless integration with PyTorch.

KeOps lets users work with millions of points at a time

Benchmark of a Gaussian **convolution** $a_i \leftarrow \sum_{j=1}^{N} \exp(-\|x_i - y_j\|_{\mathbb{R}^3}^2)\, b_j$
between **clouds of N 3D points** on a A100 GPU.



Yet another ML compiler?

Many impressive tools out there (Taichi, Numba, Triton, Halide…):

- Focus on **generality** (software + hardware).
- Increasingly easy to use via e.g. PyTorch 2.0.

KeOps fills a different niche (a bit like cuFFT, FFTW…):

- Focus on a **single major bottleneck**: geometric interactions.
- **Agnostic** with respect to Euclidean / non-Euclidean formulas.
- Fully compatible with PyTorch, NumPy, R.
- Can actually be **used by mathematicians**.

KeOps is a **bridge** between geometers (with a maths background)
and compiler experts (with a CS background).

Optimal transport?

Optimal transport (OT) generalizes sorting to spaces of dimension $D > 1$

If $A = (x_1, \dots, x_N)$ and $B = (y_1, \dots, y_N)$ are two clouds of N points in $\mathbb{R}^D$, we define:

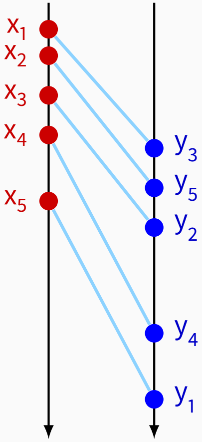
$$\text{OT}(A, B) = \min_{\sigma \in \mathcal{S}_N} \frac{1}{2N} \sum_{i=1}^{N} \| x_i - y_{\sigma(i)} \|^2$$

Generalizes **sorting** to metric spaces.

Linear problem on the permutation matrix P:

$$\text{OT}(A, B) = \min_{P \in \mathbb{R}^{N \times N}} \frac{1}{2N} \sum_{i,j=1}^{N} P_{i,j} \cdot \| x_i - y_j \|^2 ,$$

$$\text{s.t.} \quad P_{i,j} \geqslant 0 \qquad \underbrace{\textstyle\sum_j P_{i,j} = 1}_{\text{Each source point...}} \qquad \underbrace{\textstyle\sum_i P_{i,j} = 1}_{\text{is transported onto the target.}} .$$



assignment

$\sigma : [\![1, 5]\!] \rightarrow [\![1, 5]\!]$

Practical use

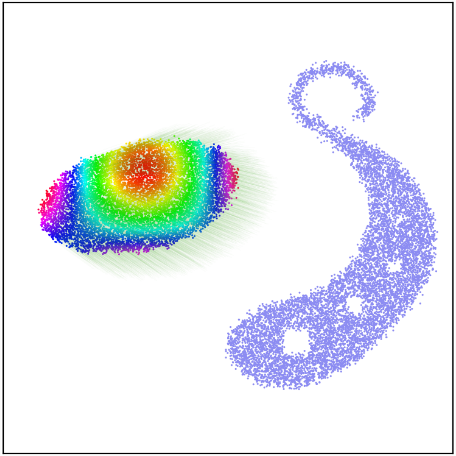
Alternatively, we understand OT as:

- Nearest neighbor **projection** + **incompressibility** constraint.
- Fundamental example of **linear optimization** over the transport plan $\text{P}_{i,j}$.

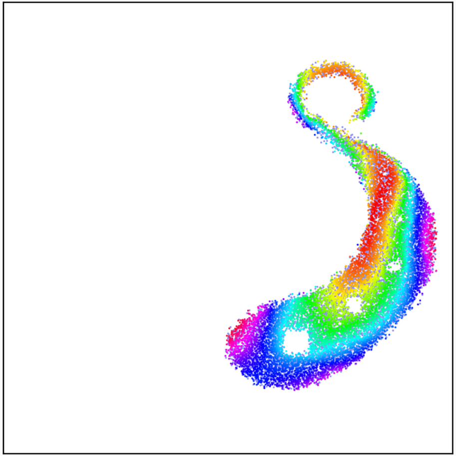
This theory induces two main quantities:

- The transport plan $\text{P}_{i,j} \simeq$ the optimal mapping $x_i \mapsto y_{\sigma(i)}$.
- The "Wasserstein" distance $\sqrt{\text{OT}(\text{A}, \text{B})}$.

The optimal transport plan

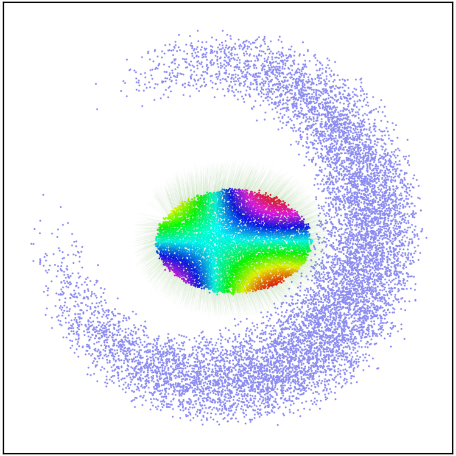


Before

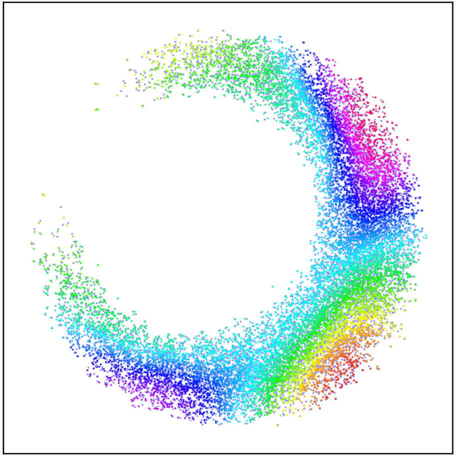


After

The optimal transport plan

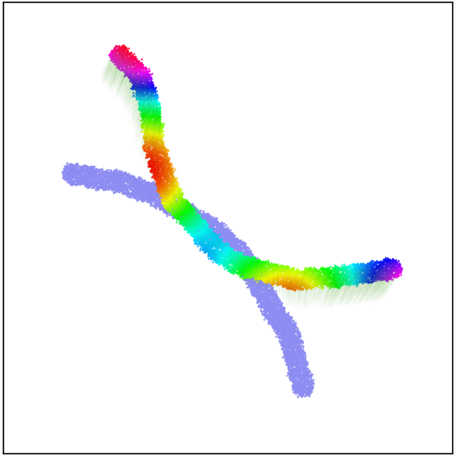


Before



After

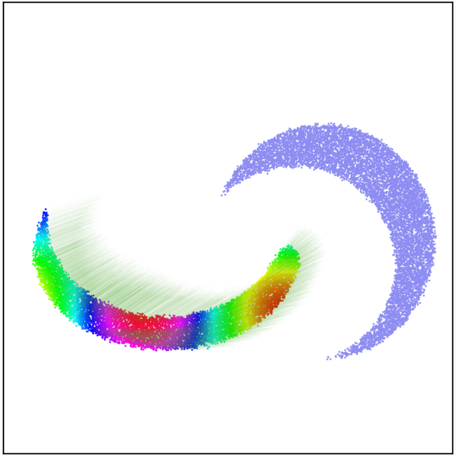
The optimal transport plan



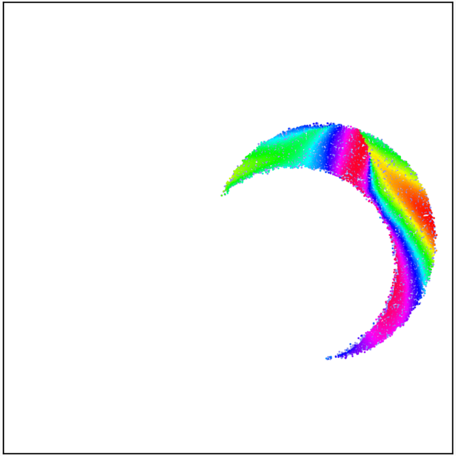
Before

After

The optimal transport plan



Before

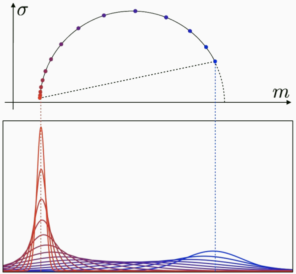


After

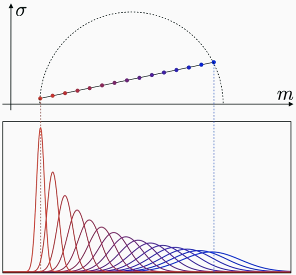
OT induces a geometry-aware distance between probability distributions [PC18]

Gauss map $\mathcal{N} : (m, \sigma) \in \mathbb{R} \times \mathbb{R}_{\geqslant 0} \quad \mapsto \quad \mathcal{N}(m, \sigma) \in \mathbb{P}(\mathbb{R})$.

If the space of **probability distributions** $\mathbb{P}(\mathbb{R})$ is endowed with a given metric, what is the "pull-back" geometry on the space of **parameters** (m, σ)?



Fisher-Rao ($\simeq$ relative entropy) on $\mathcal{N}(m, \sigma)$
$\rightarrow$ Hyperbolic **Poincaré** metric on (m, σ).



OT on $\mathcal{N}(m, \sigma)$
$\rightarrow$ Flat **Euclidean** metric on (m, σ).

How to solve the OT problem?

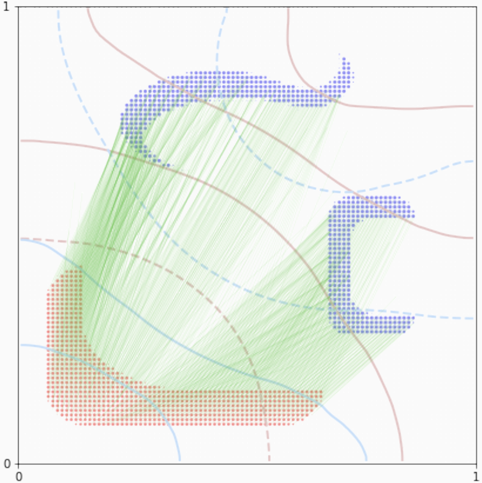
Key dates for discrete optimal transport with N points:

- [Kan42]: **Dual** problem of Kantorovitch.
- [Kuh55]: **Hungarian** methods in $O(\mathrm{N}^3)$.
- [Ber79]: **Auction** algorithm in $O(\mathrm{N}^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + simulated annealing, in $O(\mathrm{N}^2)$.
- [GRL$^+$98, CR00]: **Robust Point Matching** = Sinkhorn as a loss.
- [Cut13]: Start of the **GPU era.**
- [Mér11, Lév15, Sch19]: **multi-scale** solvers in $O(\mathrm{N}\log\mathrm{N})$.

- **Solution,** today: **Multiscale Sinkhorn algorithm, on the GPU**.

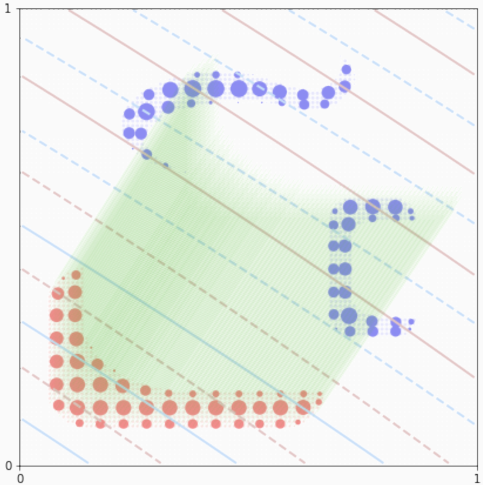
$\Longrightarrow$ Generalized **QuickSort** algorithm.

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



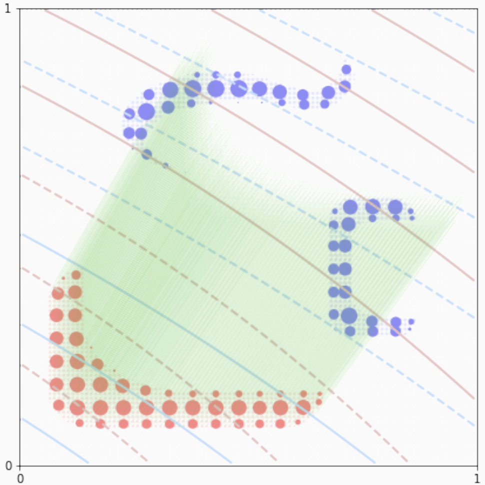
OT plan in 2D.

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



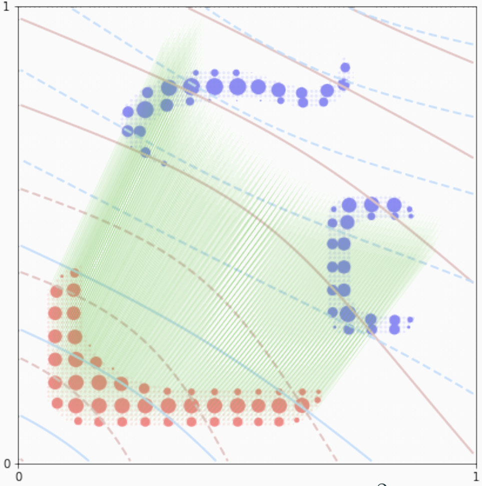
Iteration 0, blur $\sigma = 2^0$

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



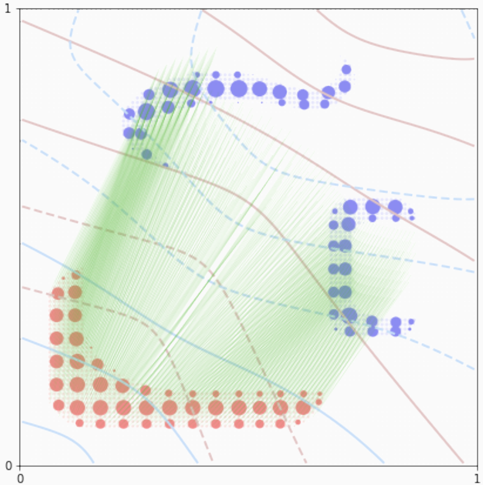
Iteration 1, blur $\sigma = 2^{-1}$

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



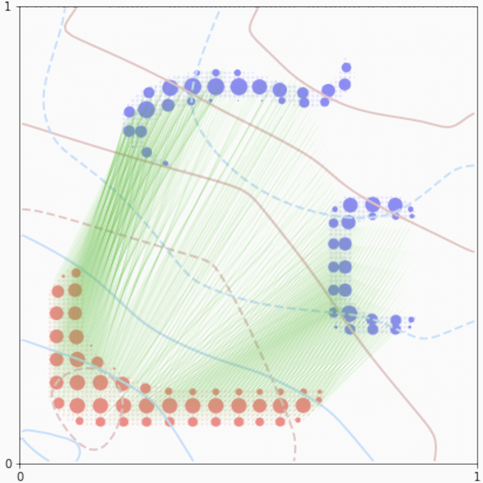
Iteration 2, blur $\sigma = 2^{-2}$

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



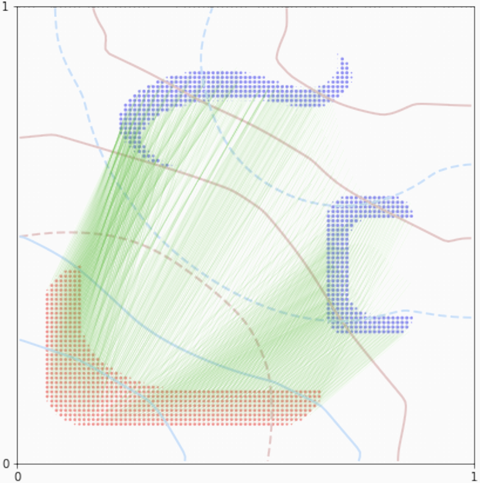
Iteration 3, blur $\sigma = 2^{-3}$

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha,\beta)$



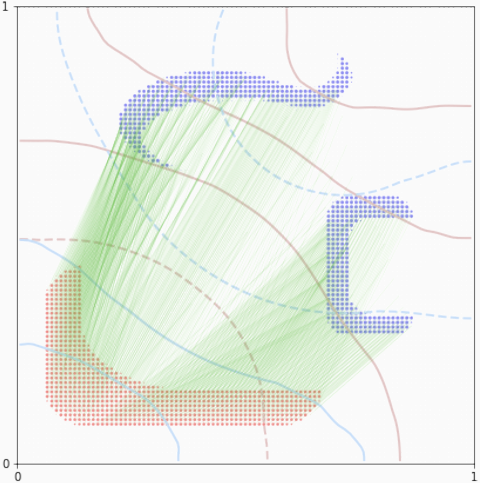
Iteration 4, blur $\sigma = 2^{-4}$

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\mathrm{OT}(\alpha, \beta)$



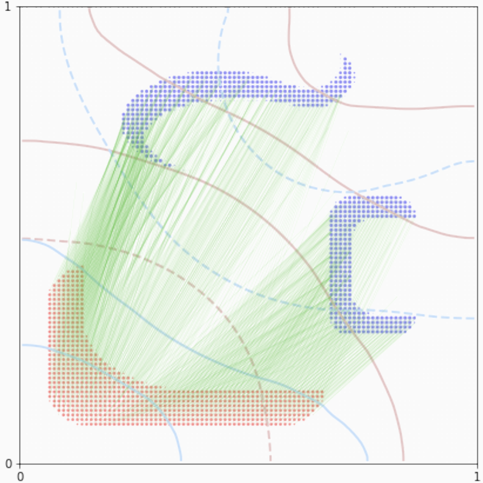
Iteration 5, blur $\sigma = 2^{-5}$

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\text{OT}(\alpha, \beta)$



Iteration 6, blur $\sigma = 2^{-6}$

Visualizing F, G and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i}\partial_{x_i}\text{OT}(\alpha, \beta)$



Iteration 7, blur σ = `.01`

Scaling up optimal transport to anatomical data

Progresses of the last decade add up to a $\times$**100** - $\times$**1000** acceleration:

$$\text{Sinkhorn GPU} \xrightarrow{\times 10} + \text{KeOps} \xrightarrow{\times 10} + \text{Annealing} \xrightarrow{\times 10} + \text{Multi-scale}$$

With a precision of 1%, on a modern gaming GPU:

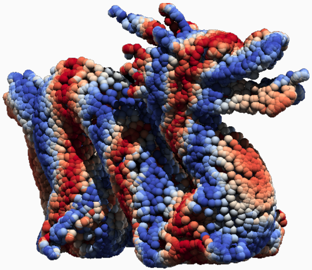
`pip install geomloss`

+

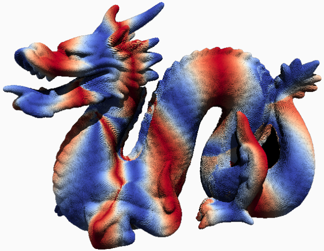
modern GPU

(1 000 €)

$\Longrightarrow$

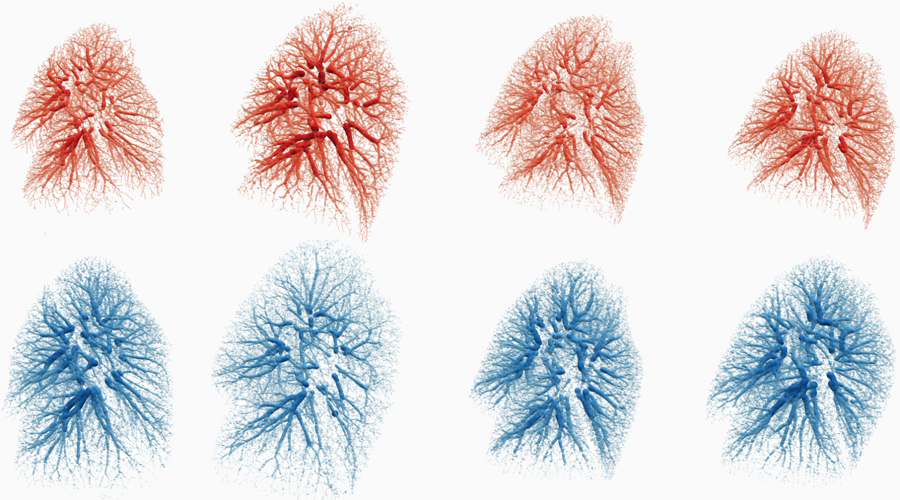


10k points in 30-50ms



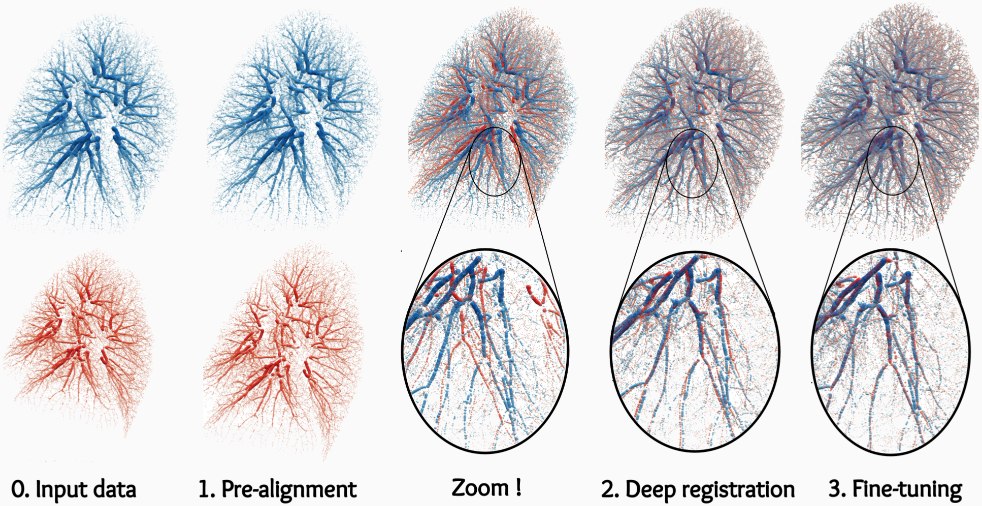
100k points in 100-200ms

A typical example in anatomy: lung registration “Exhale – Inhale”



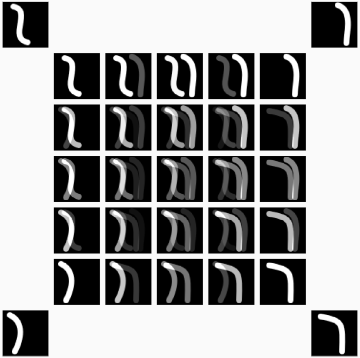
Complex deformations, high **resolution** (50k–300k points), high **accuracy** (< 1mm).

Three-steps registration



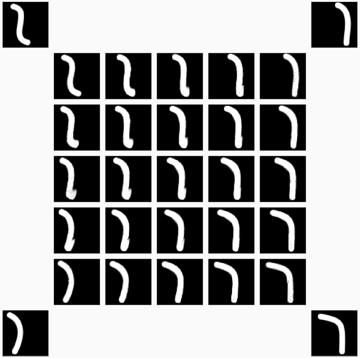
Wasserstein barycenters [AC11]

$$\text{Barycenter } A^* = \arg\min_{A} \sum_{i=1}^{4} \lambda_i \, \text{Loss}(A, B_i) \,.$$



Euclidean barycenters.

$$\text{Loss}(A, B) = \|A - B\|_{L^2}^2$$



Wasserstein barycenters.

$$\text{Loss}(A, B) = \text{OT}(A, B)$$

Incompressible particles

Two very talented colleagues

Maciej Buze

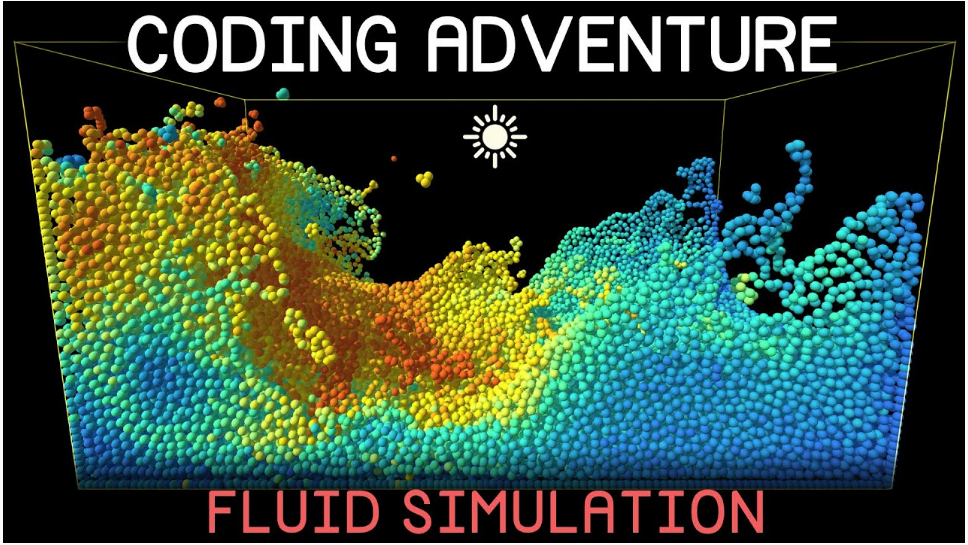
Heriot-Watt University

Antoine Diez

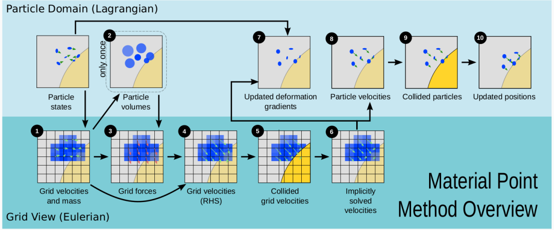
Kyoto University

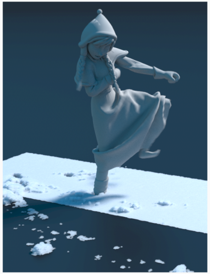
Original motivation: the N-body problem [Pri11]

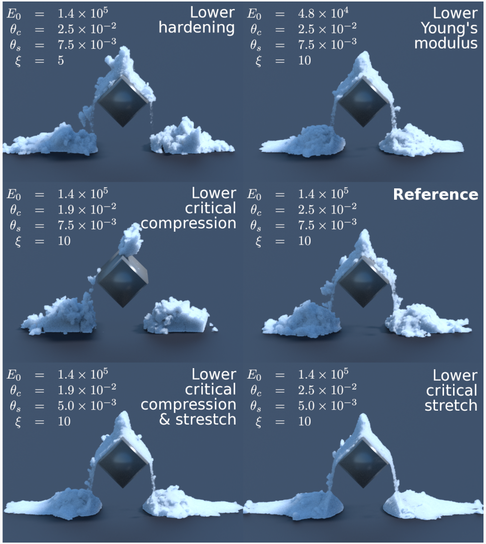
Coding a simple fluid simulation is now a matter of hours [Lag23]



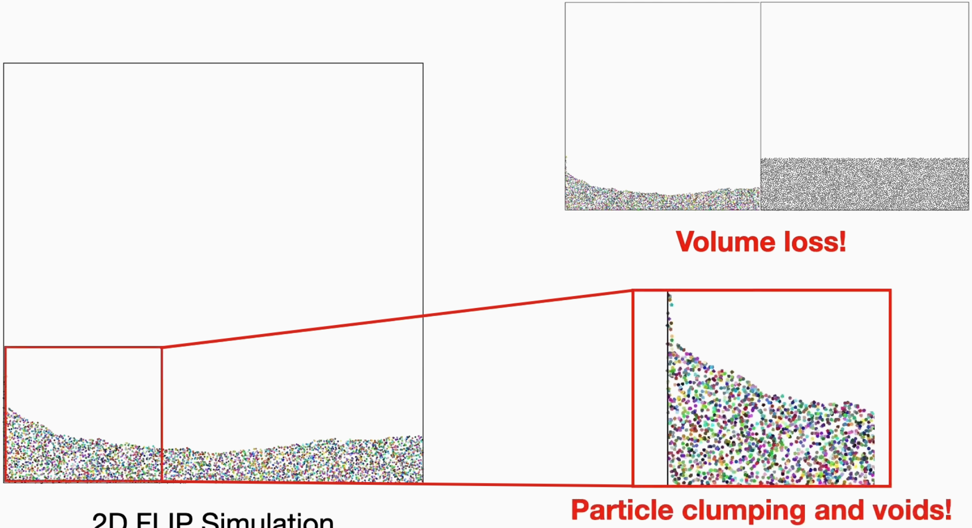
The material point method: Disney's Frozen [SSC+13]





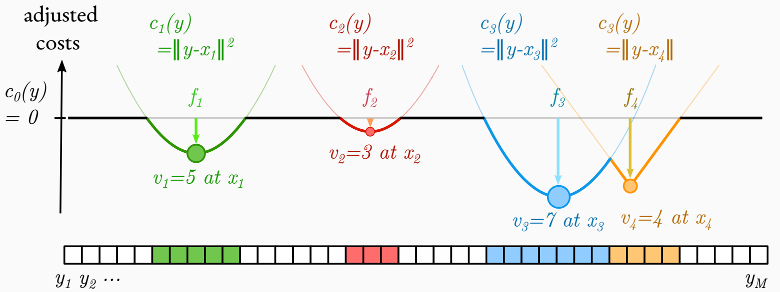


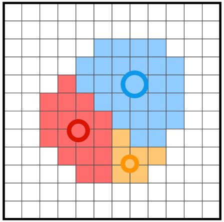
How can we enforce a volume preservation constraint? [QLDGJ22]



2D FLIP Simulation

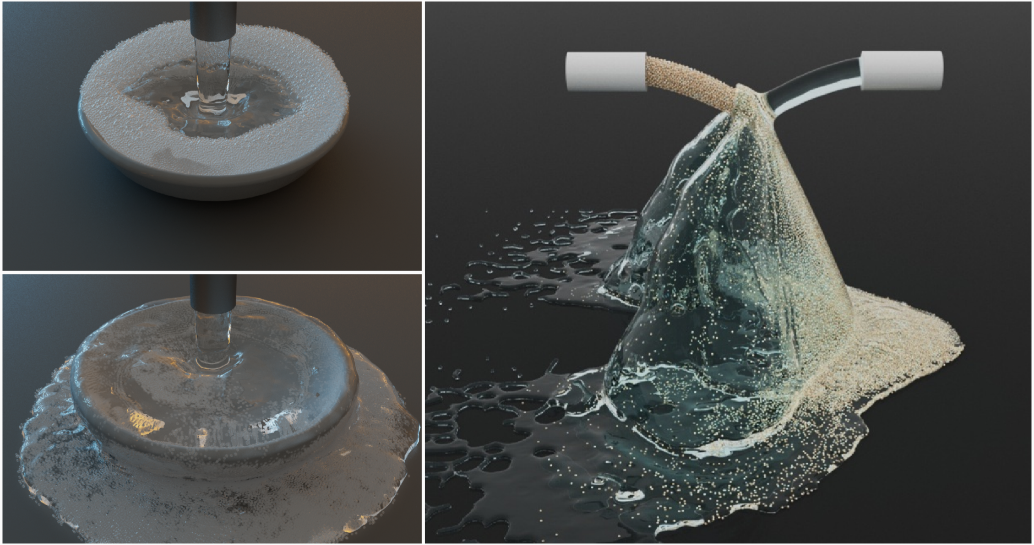
Use power diagrams i.e. semi-discrete optimal transport



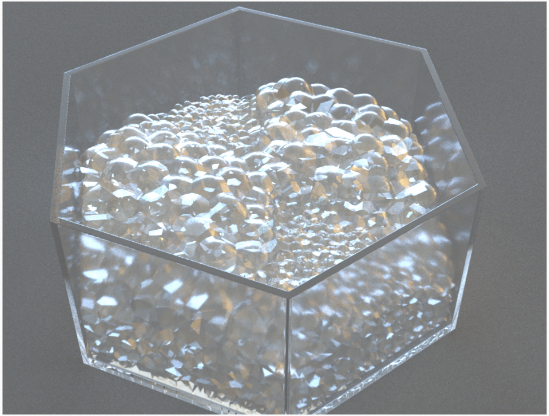


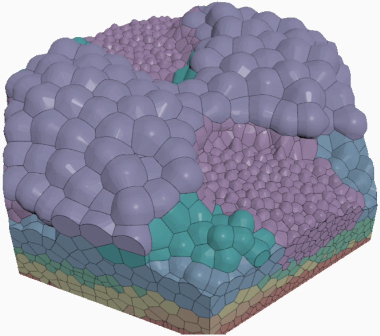
- The f_i's maximize the dual objective $\sum_{i=1}^{N} v_i f_i + \int_{y \in \Omega} \min_{i=0}^{N} [\, c_i(y) - f_i \,] \, \mathrm{d}y$.
- **Optimality** conditions $\iff$ $\mathrm{Vol}(\mathrm{Cell}_i) = v_i$.
- To **compute the cells**, the objective and its gradient:
  - If $c_i(y) = \|y - x_i\|^2$ for all cells, use a clever **grid-free** algorithm.
  - Otherwise, just use **KeOps**.

Power plastics [QLY+23]



Power plastics [QLY+23] – without the eye candy





Main numerical ingredients

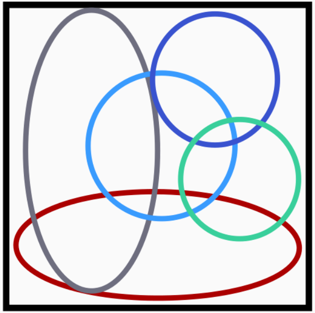
These simulations alternate between:

1. **Moving the particles** according to your favorite N-body model.
2. Computing Laguerre **cells** with the **correct volumes**:
   - (Multiscale) Sinkhorn for tolerance $> 5\%$.
   - (Quasi-)Newton for tolerance $< 1\%$.
3. **Correcting** the particle positions to enforce the volume-preservation constraint:
   - Jump to the centroid of the cell.
   - Or add a spring for smoother trajectories.

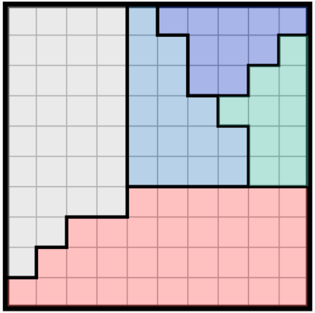
See e.g. Thomas Gallouët for a rigorous analysis with Mérigot, Lévy, etc.

But today: new applications with **custom cost functions** (thanks KeOps).

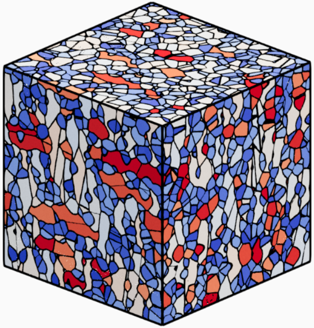
Anisotropic power diagrams let us model polycrystalline metals [BFR+24]



Ellipsoids.

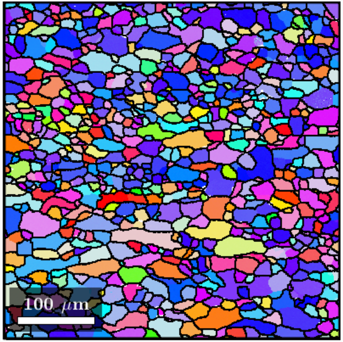


Pixel cells.

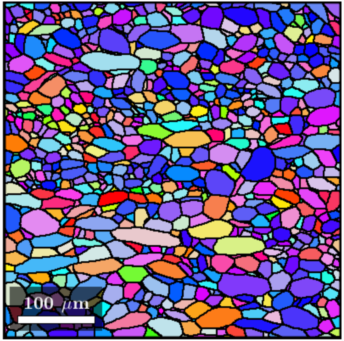


5,000 crystals in 3D.

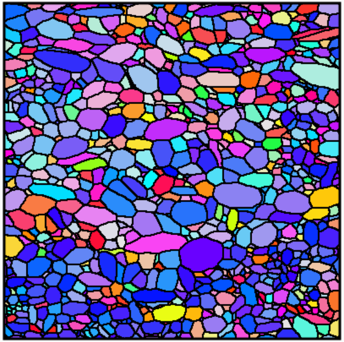
Fit to real EBSD scan of low-carbon steel [BFR+24]



Data from Tata steel.



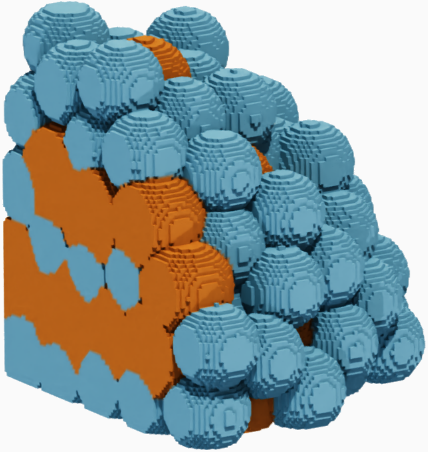
Our APD model.



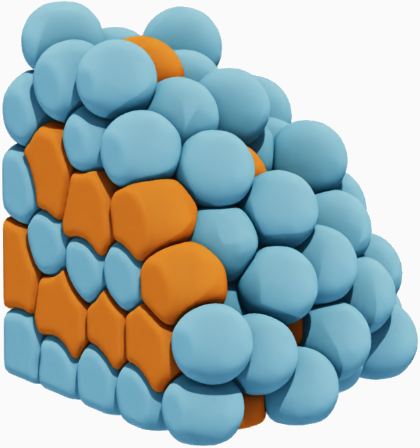
New synthetic image.

We can generate new, realistic 3D images with **prescribed properties** in seconds.

Change the cost function to simulate hard (blue) and soft (orange) cells [DF24]



The **raw** 100x100x100 pixel grid…

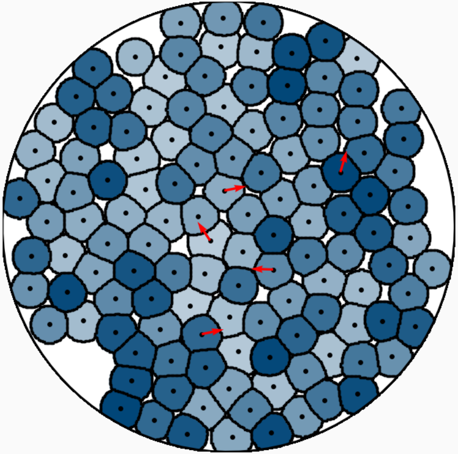


with some Hollywood **makeup**.

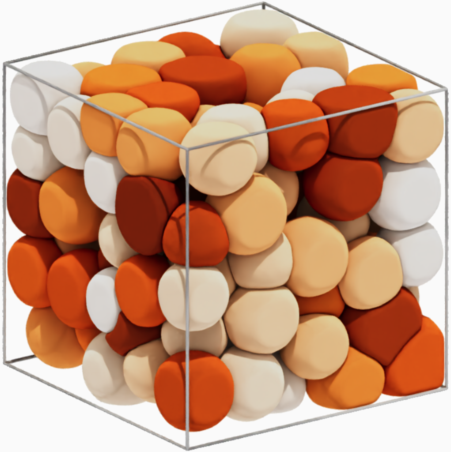
Let’s visit Antoine’s website

$\Longrightarrow$ https://iceshot.readthedocs.io $\Longleftarrow$

Run-and-tumble motion [DF24]

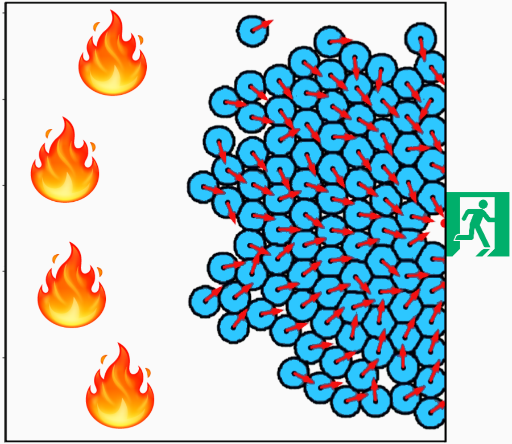


2D disk.

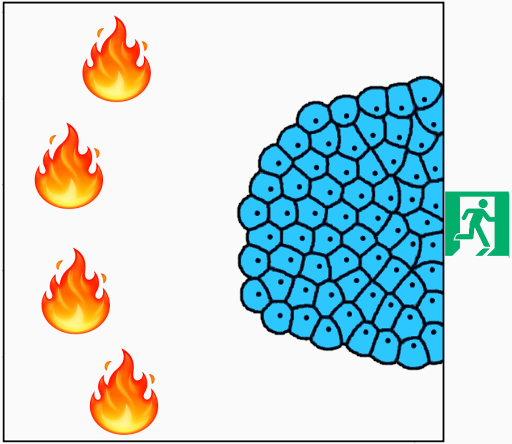


3D cube.

Fire alarm! [DF24]

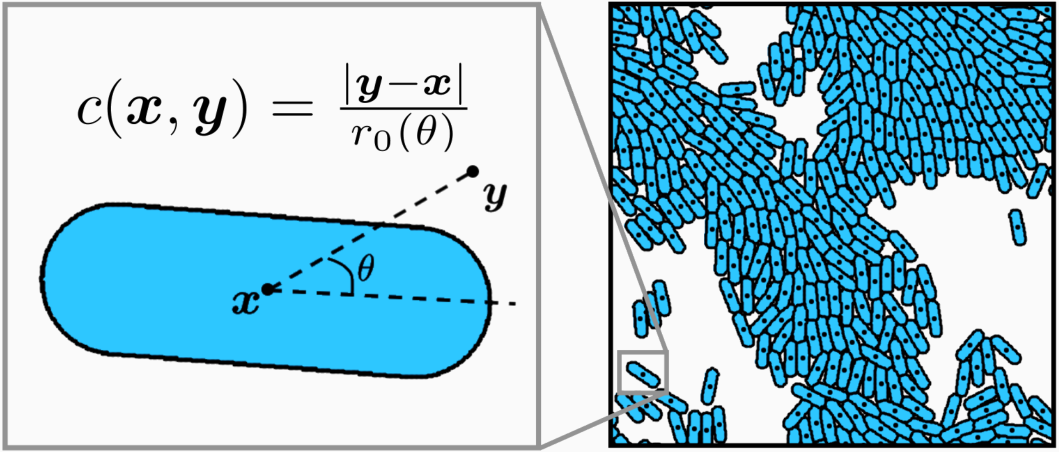


Hard particles **burn**.

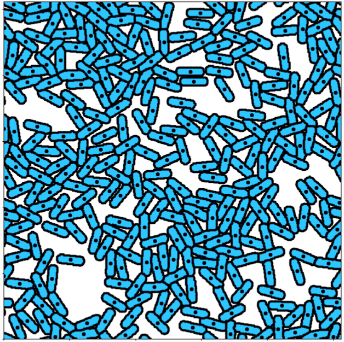


Soft particles **escape**.

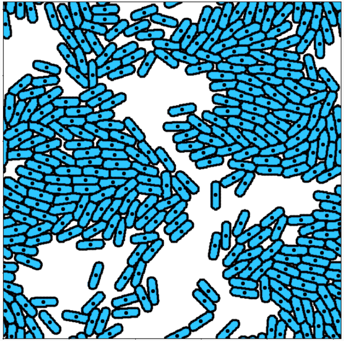
Self-organizing swarms of blind, incompressible swimmers [DF24]



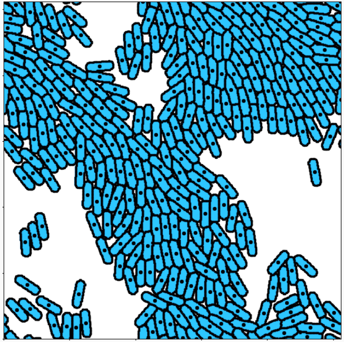
Self-organizing swarms of blind, incompressible swimmers [DF24]



$t = 0$



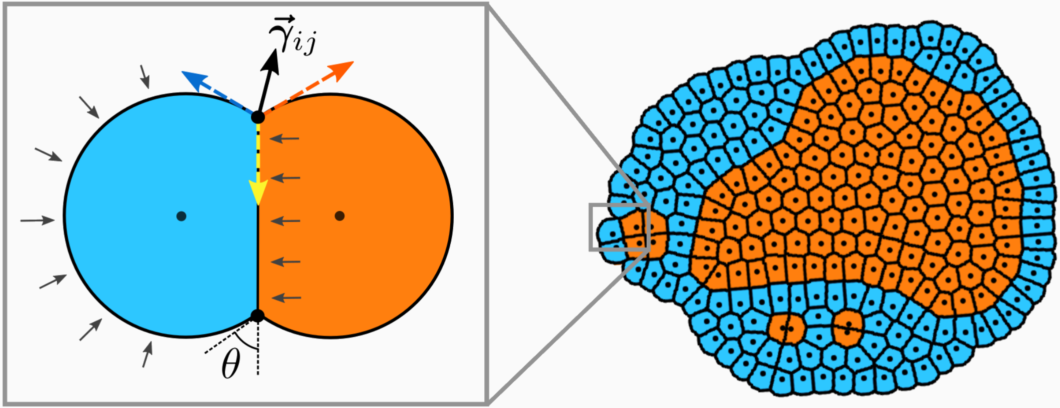
$t = 4$



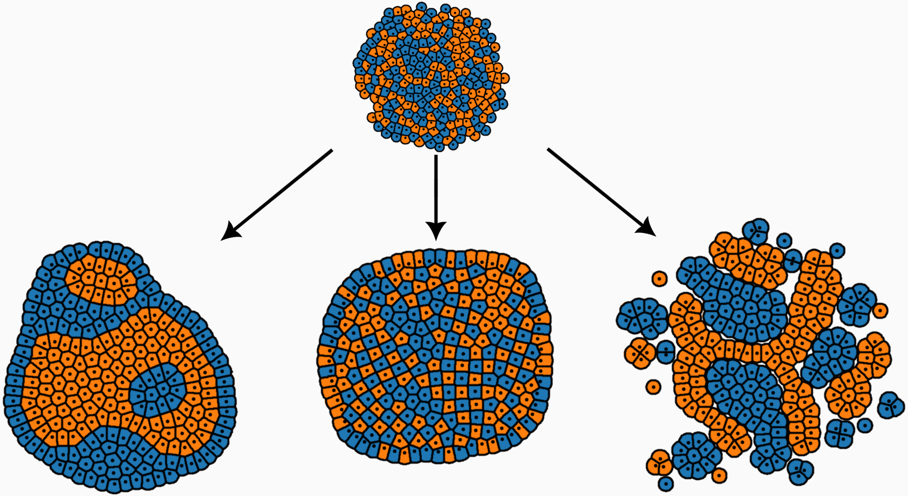
$t = 30$

Order emerges out of blind collisions and re-alignments.

Surface tension [DF24]



Surface tension [DF24] – playing with the energy parameters

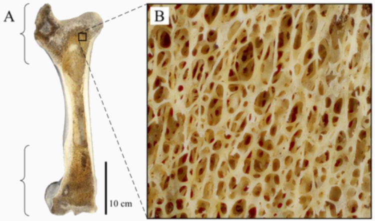


Biological microstructures

Memories from the Covid lockdown...

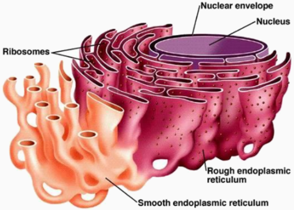
Anna Song
Francis Crick Institute,
now at Owkin.

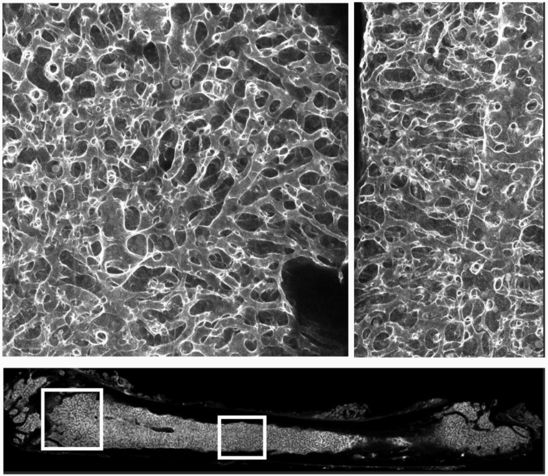
Biological motivation



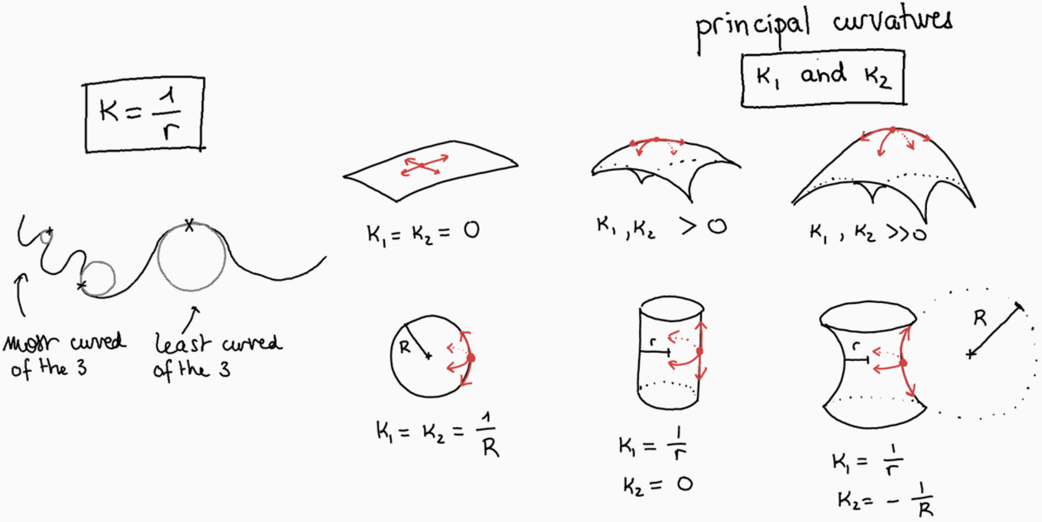
Trabecular bone has plates and rods, from Bishop et al., PeerJ (2018)

Endoplasmic reticulum has sheets and tubules

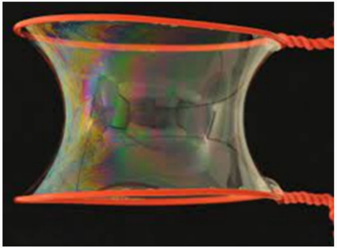
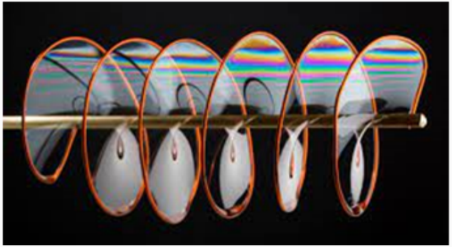




Curvature of a 2D surface



Minimal surfaces in physics

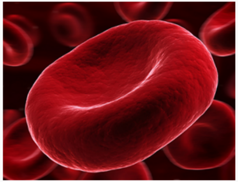
Soap bubbles minimize:

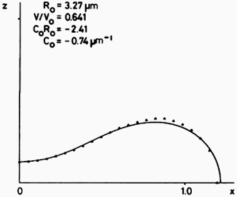
$$\text{area}(\mathcal{S}) = \int_{\mathcal{S}} 1 \, dA$$

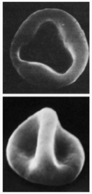
under constant volume, or with boundary conditions.

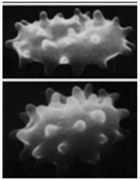
They correspond to minimal surfaces with $H = \kappa_1 + \kappa_2 = 0$ in cases 2 and 3.

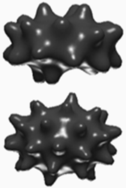
Minimal surfaces in biology









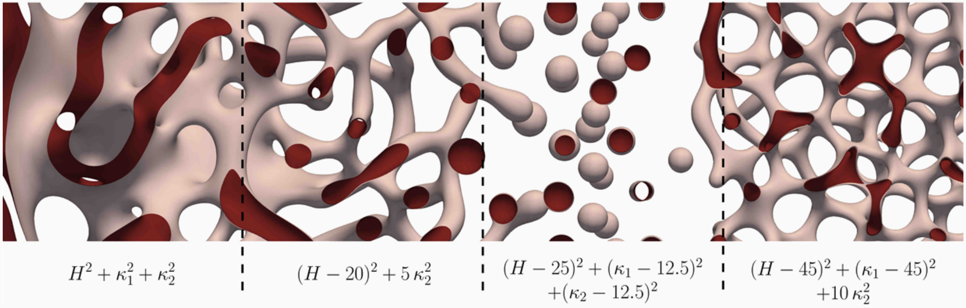


Red blood cells minimize:

$$\mathrm{Helfrich}(\mathcal{S}) = \int_{\mathcal{S}} (H - H_0)^2 \, dA$$

or a variant of this energy, under constant volume.

The curvatubes model [Son22]



Curvatubes minimize:

$$F(\mathcal{S}) = \int_{\mathcal{S}} p(\kappa_1, \kappa_2)\, dA$$

under constant volume, where:

$$p(\kappa_1, \kappa_2) = a_{2,0}\,\kappa_1^2 + a_{1,1}\,\kappa_1\kappa_2 + a_{0,2}\,\kappa_2^2 + a_{1,0}\,\kappa_1 + a_{0,1}\,\kappa_2 + a_{0,0}\,.$$

Under the hood: a phase-field formulation

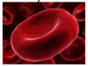
$$\mathbf{E}_A(\mathcal{S}) = \int_{\mathcal{S}} 1 \, dA$$

minimal surfaces (1750')

$$\mathbf{E}_{\mathrm{W}}(\mathcal{S}) = \int_{\mathcal{S}} H^2 \, dA$$

Willmore (1960')

$$\mathbf{E}_{\mathrm{H}}(\mathcal{S}) = \int_{\mathcal{S}} \left(\frac{\chi_b}{2} (H - H_0)^2 + \chi_G K \right) dA$$



Helfrich (1970')

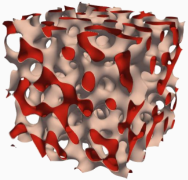
$$\mathbf{F}(\mathcal{S}) = \int_{\mathcal{S}} p(\kappa_1, \kappa_2) \, dA$$

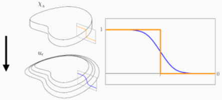
Curvatubes (2021)

$$\mathbf{F}(\mathcal{S}) = \int_{\mathcal{S}} p(\kappa_1, \kappa_2) \, dA$$

2D surface energy

hard to simulate

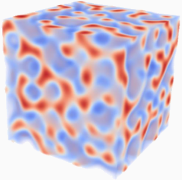




$$\mathcal{E}_\epsilon(u) = \int_\Omega p(\kappa^\epsilon_{1,u}, \kappa^\epsilon_{2,u}) \, \epsilon |\nabla u|^2 \, dx$$

3D phase-field energy

easy to simulate on GPUs



https://github.com/annasongmaths/curvatubes

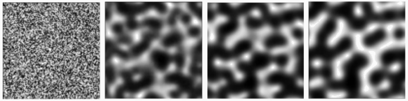
Texture generation via gradient descent on a convolutional energy

curvatubes

parameters inside energy

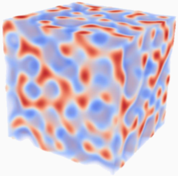
$$\dot{u} = \Delta \frac{\partial \mathcal{F}_\epsilon}{\partial u}$$

random initialization



=

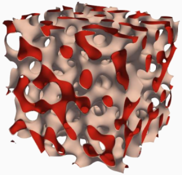
phase-field



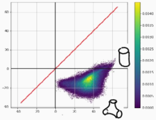
$$\mathcal{F}_\epsilon(u) = \int_\Omega \Big[\frac{a_{2,0} + a_{0,2} - a_{1,1}}{2\epsilon} \, \|\mathcal{M}_u^\epsilon\|^2 + \frac{a_{1,1}}{2\epsilon} \, (\mathrm{Tr}\mathcal{M}_u^\epsilon)^2 + \frac{a_{2,0} - a_{0,2}}{2\epsilon} \, \mathrm{Tr}\mathcal{M}_u^\epsilon \sqrt{(2\|\mathcal{M}_u^\epsilon\|^2 - (\mathrm{Tr}\mathcal{M}_u^\epsilon)^2)^+}$$
$$+ \frac{a_{1,0} + a_{0,1}}{2} \, |\nabla u| \mathrm{Tr}\mathcal{M}_u^\epsilon + \frac{a_{1,0} - a_{0,1}}{2} \, |\nabla u| \sqrt{(2\|\mathcal{M}_u^\epsilon\|^2 - (\mathrm{Tr}\mathcal{M}_u^\epsilon)^2)^+} + a_{0,0} \, \epsilon |\nabla u|^2 \Big] \, dx.$$

Pytorch
GPU
Adam or L-BFGS

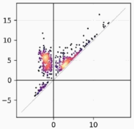
outputs



surface



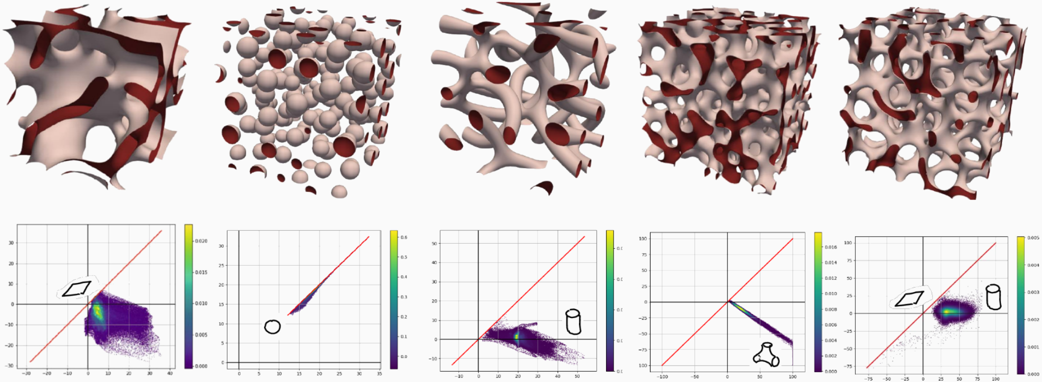
curvature diagram



persistence diagram

other features...

Some simple examples



$H^2 + \kappa_1^2 + \kappa_2^2$

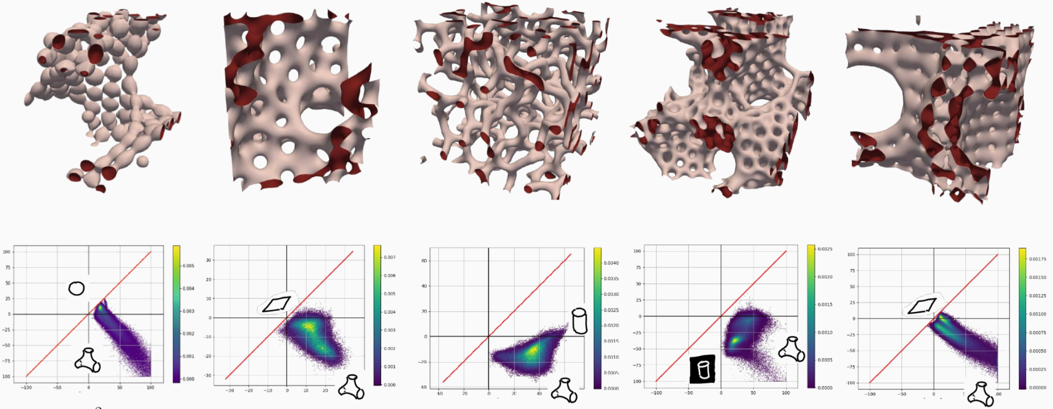
$(H - 25)^2 + (\kappa_1 - 12.5)^2 + (\kappa_2 - 12.5)^2$

$(H - 20)^2 + 5\kappa_2^2$

$(H - 5)^2 + 0.8K + \kappa_2^2$

(no natural form)

Some complex examples

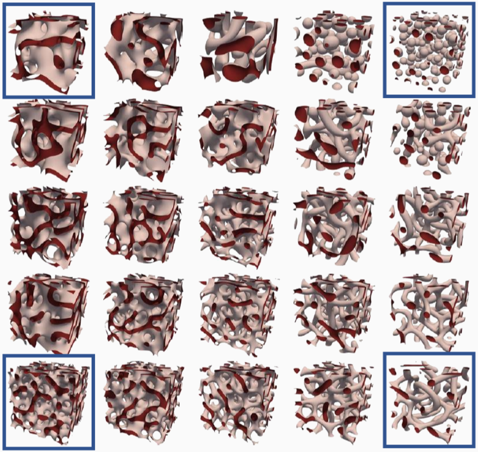


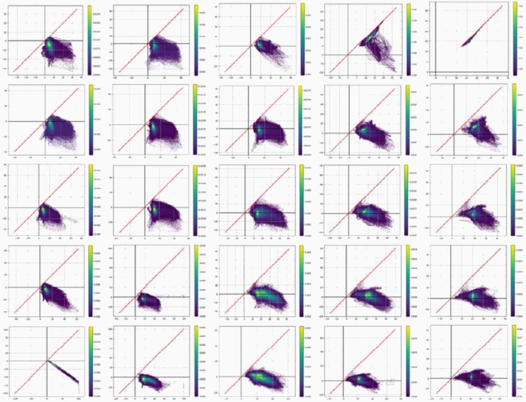
$(H - 28)^2 + 1.55K$ | (no natural form) | (no natural form) | (no natural form) | (no natural form)

Continuity of the model with respect to its parameters

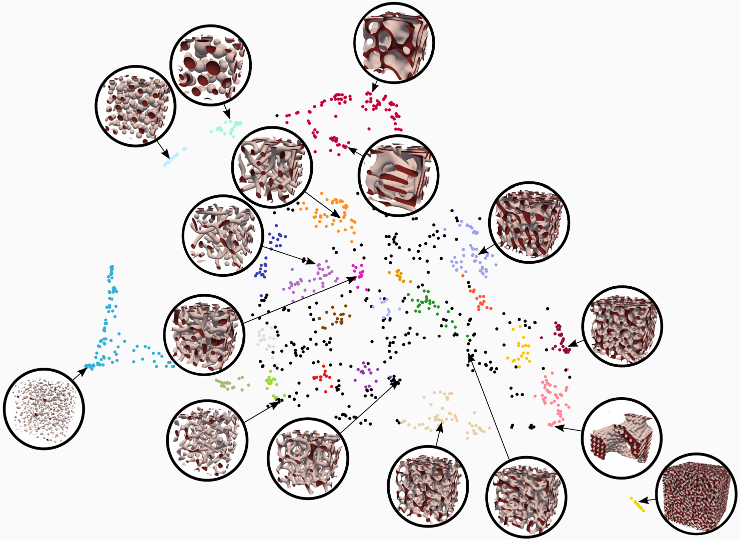
same initialization
different energies

bilinear interpolation between 4 shape parameters leads to continuum of morphologies

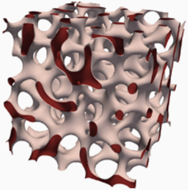


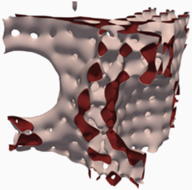


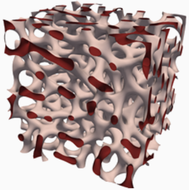
UMAP visualization of the model's output for 1,000 random polynomials

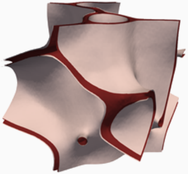


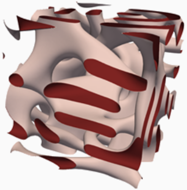
A surprisingly expressive model

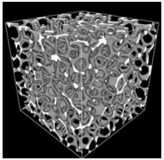






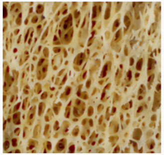




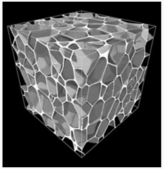


μCT image of open aluminium foam

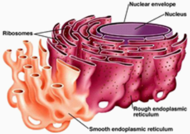
lamina cribrosa behind the eye



trabecular bone

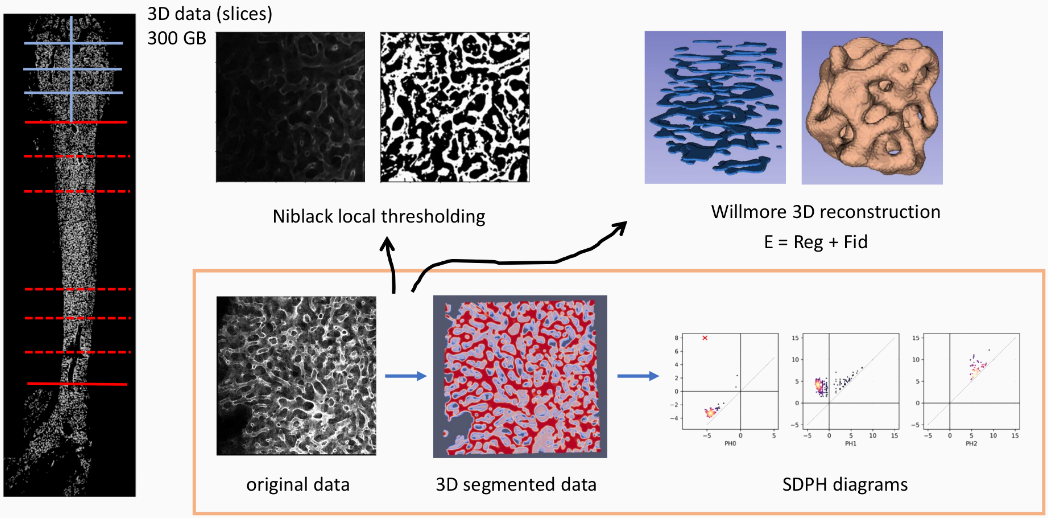


μCT image closed polymer foam

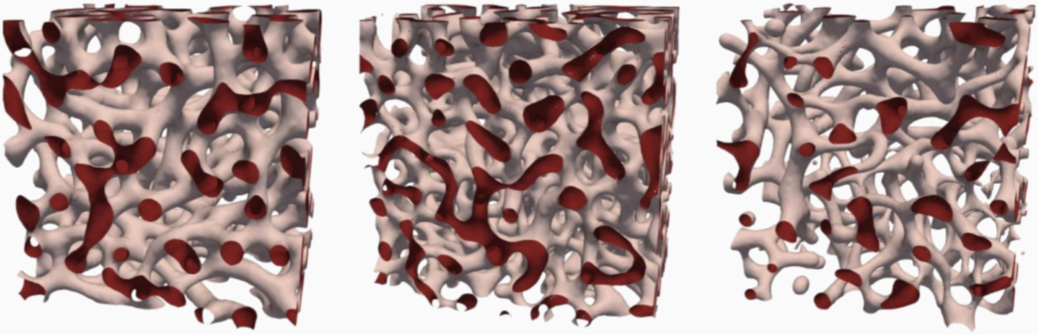


endoplasmic reticulum

Acute Myeloid Leukaemia in the bone marrow



Fitting the model to experimental data with Bayesian optimization

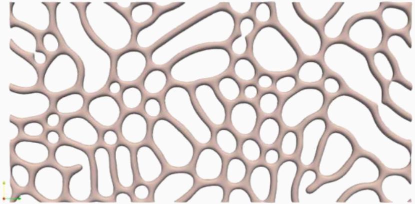


Emulated A

Emulated B

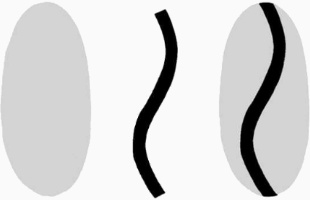
Emulated C

Designing plausible substrate for cell cultures



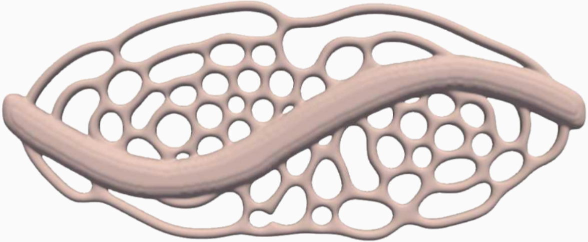
texture

+



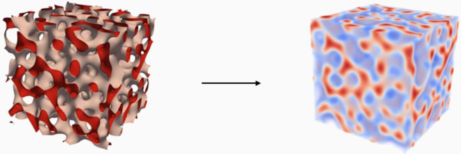
structure

=



These textures are surprisingly easy to print!

PyTorch and GPUs go way beyond deep learning research



$$\mathbf{F}(S) = \int_S p(\kappa_1, \kappa_2)\, dA \qquad \mathcal{E}_\epsilon(u) = \int_\Omega p(\kappa^\epsilon_{1,u}, \kappa^\epsilon_{2,u})\, \epsilon |\nabla u|^2\, dx$$

2D surface energy | **3D phase-field energy**

An **inspiring** model:

- Surface energy $\rightarrow$ convolutional volumetric loss function (phase-field).
- Start with white noise (texture generation) and minimize with gradient descent.
- Implemented on GPU with PyTorch.

$\Longrightarrow$ Combines maths + GPU computing + imaging data

Conclusion

Genuine team work

Benjamin Charlier

Joan Glaunès

Thibault Séjourné

F.-X. Vialard

Gabriel Peyré

Alain Trouvé

Marc Niethammer

Shen Zhengyang

Olga Mula

Hieu Do

Key points

- Optimal Transport = volume preservation = **generalized sorting** :

  $\longrightarrow$ Super-fast solvers on **simple domains**, especially 2D/3D spaces.

  $\longrightarrow$ **Fundamental tool** at the intersection of geometry and statistics.

- **“Video-game physics”** is great for modelling:

  $\longrightarrow$ **Expressive**, real-time simulations that you can implement without being a Finite Elements guru: XPBD, DiffPD, Taichi...

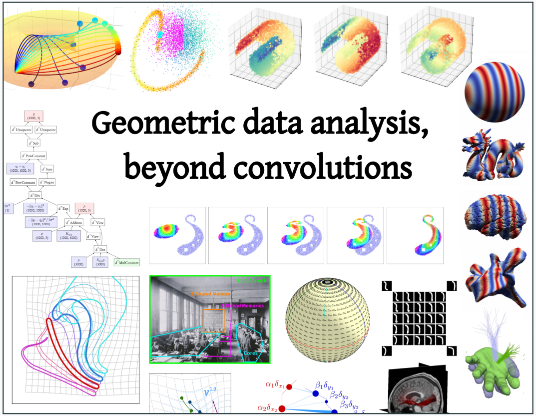
- GPUs are more **versatile** than you think.

  $\longrightarrow$ Ongoing work to provide **fast GPU backends** to researchers, going beyond what Google and Facebook are ready to pay for.

2026 target for scientific Python: **interactive, web-based** simulations à la ShaderToy.

Documentation and tutorials are available online

$\Longrightarrow$ www.kernel-operations.io $\Longleftarrow$



www.jeanfeydy.com/geometric_data_analysis.pdf

Documentation and tutorials are available online

⟹ shape-analysis.github.io ⟸

Monthly seminar, videos on YouTube.

References

References i

M. Agueh and G. Carlier.

Barycenters in the Wasserstein space.

SIAM Journal on Mathematical Analysis, 43(2):904–924, 2011.

Dimitri P Bertsekas.

A distributed algorithm for the assignment problem.

Lab. for Information and Decision Systems Working Paper, M.I.T., Cambridge, MA, 1979.

References ii

Maciej Buze, Jean Feydy, Steven Roper, Karo Sedighiani, and David P Bourne.

Anisotropic power diagrams for polycrystal modelling: efficient generation of curved grains via optimal transport.

arXiv submission 5452163, 2024.

Haili Chui and Anand Rangarajan.

A new algorithm for non-rigid point matching.

In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 44–51. IEEE, 2000.

References iii

Marco Cuturi.

Sinkhorn distances: Lightspeed computation of optimal transport.

In *Advances in Neural Information Processing Systems*, pages 2292–2300, 2013.

Antoine Diez and Jean Feydy.

An optimal transport model for dynamical shapes, collective motion and cellular aggregates, 2024.

Steven Gold, Anand Rangarajan, Chien-Ping Lu, Suguna Pappu, and Eric Mjolsness.

New algorithms for 2d and 3d point matching: Pose estimation and correspondence.

Pattern recognition, 31(8):1019–1031, 1998.

Leonid V Kantorovich.

On the translocation of masses.

In *Dokl. Akad. Nauk. USSR (NS)*, volume 37, pages 199–201, 1942.

Harold W Kuhn.

The Hungarian method for the assignment problem.

Naval research logistics quarterly, 2(1-2):83–97, 1955.

Jeffrey J Kosowsky and Alan L Yuille.

The invisible hand algorithm: Solving the assignment problem with statistical physics.

Neural networks, 7(3):477–490, 1994.

Sebastian Lague.

Coding adventure: Simulating fluids.

https://www.youtube.com/watch?v=rSKMYc1CQHE&t=1s, 2023.

Bruno Lévy.

A numerical algorithm for l2 semi-discrete optimal transport in 3d.

ESAIM: Mathematical Modelling and Numerical Analysis, 49(6):1693–1715, 2015.

Quentin Mérigot.

A multiscale approach to optimal transport.

In *Computer Graphics Forum*, volume 30, pages 1583–1592. Wiley Online Library, 2011.

Gabriel Peyré and Marco Cuturi.

Computational optimal transport.

arXiv preprint arXiv:1803.00567, 2018.

Anthony Prieur.

Simulation de la formation des structures de l'univers.

https://github.com/devpack/nbody-cosmos, 2011.

Ziyin Qu, Minchen Li, Fernando De Goes, and Chenfanfu Jiang.

The power particle-in-cell method.

ACM Transactions on Graphics, 41(4), 2022.

- Ziyin Qu, Minchen Li, Yin Yang, Chenfanfu Jiang, and Fernando De Goes.

  Power plastics: A hybrid Lagrangian/Eulerian solver for mesoscale inelastic flows.

  ACM Transactions on Graphics (TOG), 42(6):1–11, 2023.

- Bernhard Schmitzer.

  Stabilized sparse scaling algorithms for entropy regularized transport problems.

  SIAM Journal on Scientific Computing, 41(3):A1443–A1481, 2019.

Anna Song.

Generation of tubular and membranous shape textures with curvature functionals.

Journal of Mathematical Imaging and Vision, 64(1):17–40, 2022.

Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle.

A material point method for snow simulation.

ACM Transactions on Graphics (TOG), 32(4):1–10, 2013.