

# Geometric optimal transport libraries

---

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

6th of May, 2026  
Journée Bibliothèques pour le Transport Optimal, IHP, Paris

# 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).

# Now working in PariSanté Campus, Porte de Versailles



# HeKA : a translational research team for public health

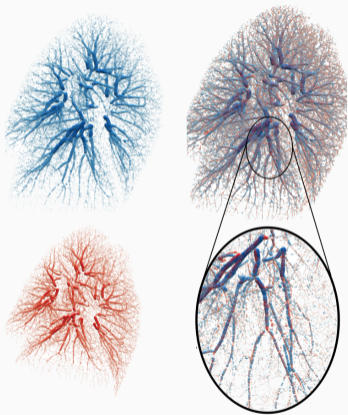
Hôpitaux

Inria      Inserm

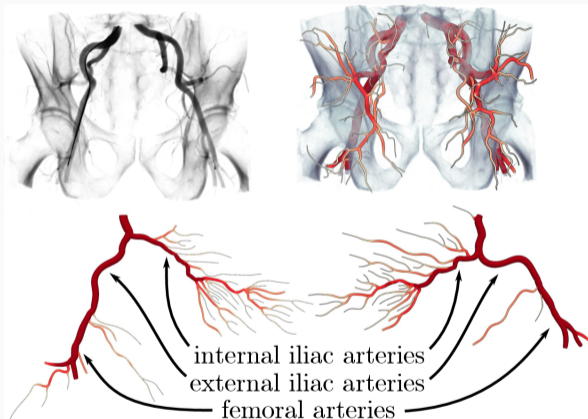
Universités



## Recent works

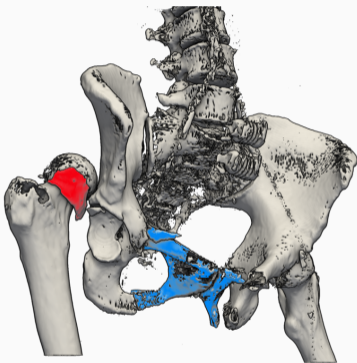


Lung **registration**.

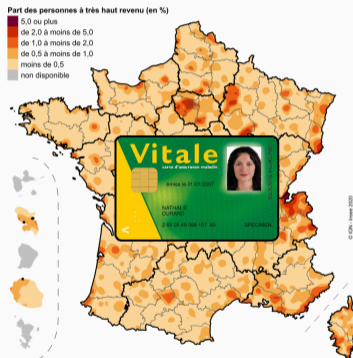


Interventional **radiology**.

## Recent works

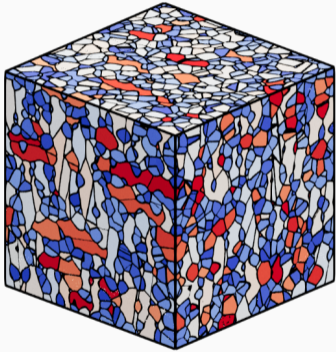


Orthopedic **surgery**.

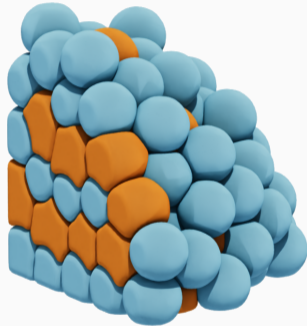


**Public** health.

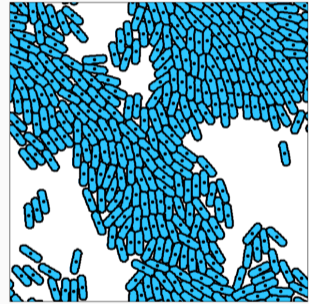
## Recent works



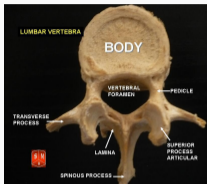
**Metallurgy.**



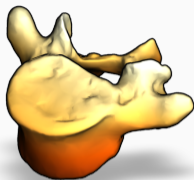
Swarms of incompressible **cells**.



# Speed and consistency with diverse data structures



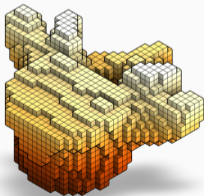
Genuine vertebra.



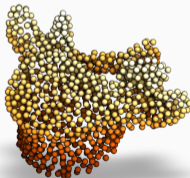
Surface mesh.



Volumetric mesh.



Segmentation mask.



Point cloud.



Gaussian mixture.

## My main motivation

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

1. Speed up **geometric machine learning** on GPUs:  
⇒ **pyKeOps** library for distance and kernel matrices, 1M+ downloads.
2. Scale up **pharmacovigilance** to the full French population:  
⇒ **survivalGPU**, a fast re-implementation of the R survival package.
3. Ease access to modern statistical **shape analysis**:  
⇒ **GeomLoss**, scalable optimal transport in Python, 900k+ downloads.  
⇒ **scikit-shapes**, available soon.

## Today's talk – assuming that you would enjoy some pretty pictures

1. A quick heads up on **fast geometric methods**.
2. Efficient discrete optimal transport **solvers**.
3. Applications to systems of **incompressible particles**.
4. Discussion about **future developments**.

## Differences between existing OT libraries

To put my work into perspective:

- **Combinatorial** solvers are accurate and handle arbitrary cost matrices, but **do not scale** to 3D data.
- **Semi-discrete** solvers provide **very accurate solutions** to a **specific problem** in dimensions 2 and 3.
- **Machine learning** libraries usually target **high-dimensional** applications and a **precision of about 10%** on the transport map.
- I solve problems with **explicit transportation costs** in **dimension 1 to 50**, with **10k to 1M** points and target a **precision of 1%** on the transport map.

## **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:

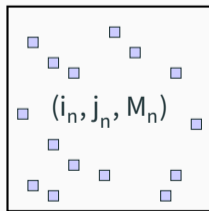
- **Long access times** to the registers penalize the use of large **dense** arrays.
- Hard-wired **contiguous** memory accesses penalize the use of **sparse** matrices.

**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, KPCConv...): **several months of work.**



**Dense array**



**Sparse matrix**

# The KeOps library: efficient support for symbolic matrices

**Solution.** KeOps – [www.kernel-operations.io](http://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.

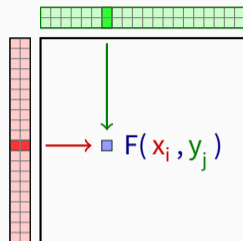
If the formula “F” is simple ( $\leq 100$  arithmetic operations):

“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!**

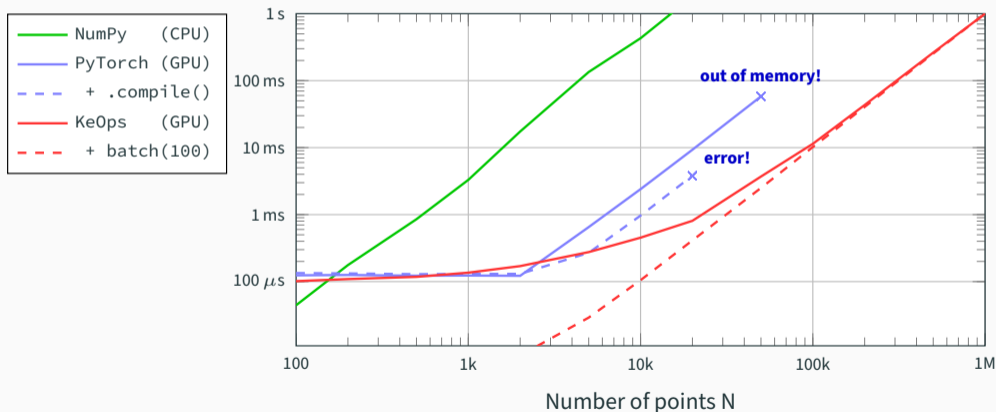
```
D_ij = ((x_i - x_j) ** 2).sum(dim=2)      # Euclidean
M_ij = (x_i - x_j).abs().sum(dim=2)     # Manhattan
C_ij = 1 - (x_i | x_j)                  # Cosine
H_ij = D_ij / (x_i[...,0] * x_j[...,0]) # Hyperbolic
```

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

- **Reductions:** sum, log-sum-exp, K-min, matrix-vector product, etc.
- **Operations:** +, ×, 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 in dimension  $< 50$ .
- **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:

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

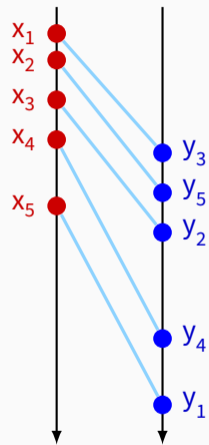
Generalizes **sorting** to metric spaces.

**Linear problem** on the permutation matrix  $\pi$ :

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

s.t.  $\pi_{i,j} \geq 0$        $\underbrace{\sum_j \pi_{i,j}} = 1$        $\underbrace{\sum_i \pi_{i,j}} = 1$ .

Each source point...      is transported onto the target.



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

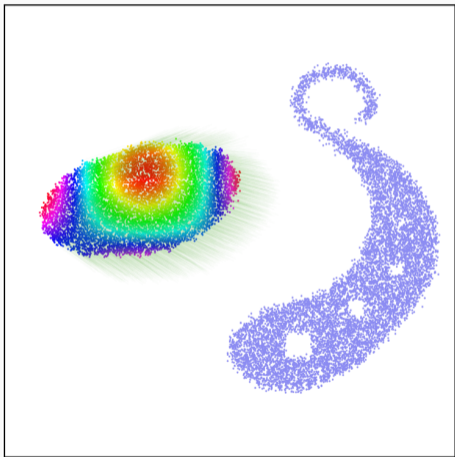
Alternatively, we understand OT as:

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

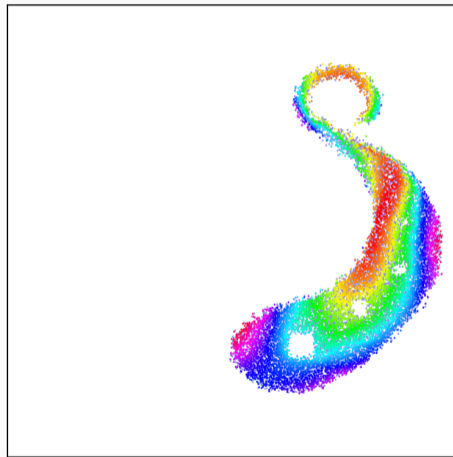
This theory induces two main quantities:

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

# The optimal transport plan

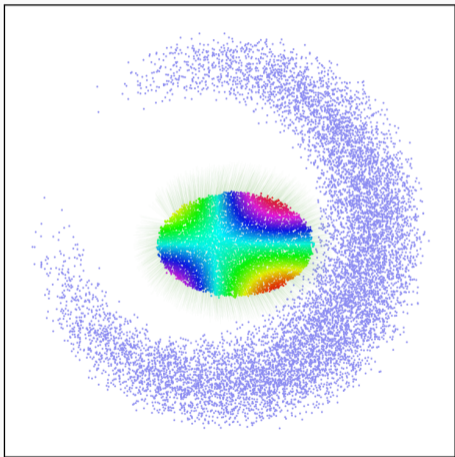


Before

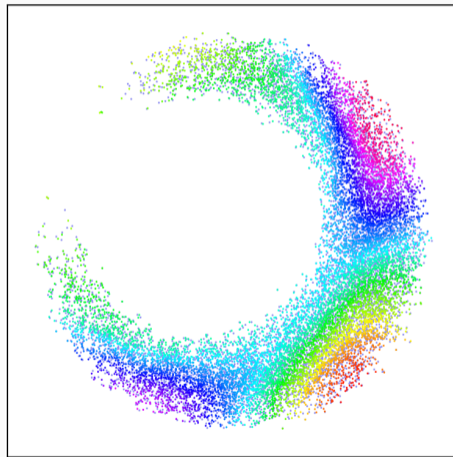


After

# The optimal transport plan

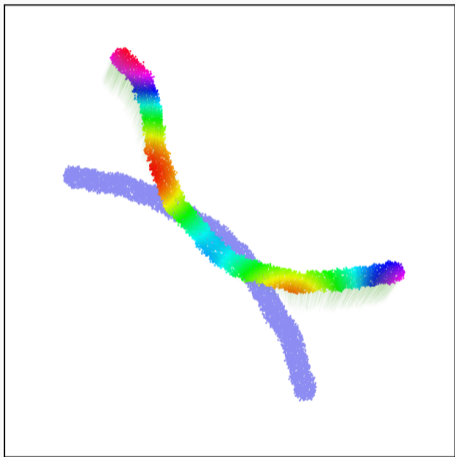


Before

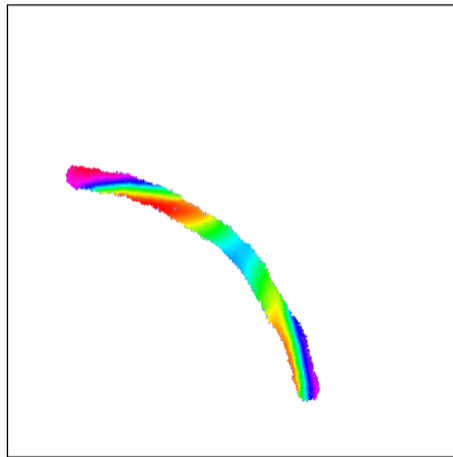


After

# The optimal transport plan

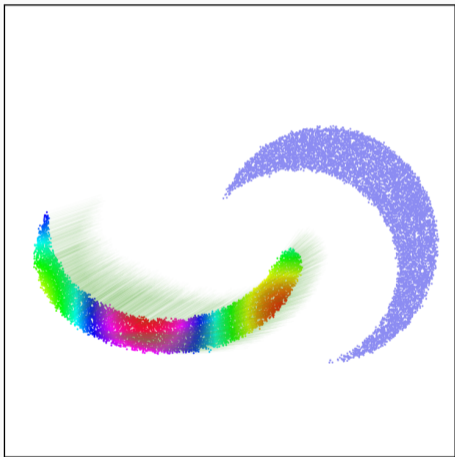


Before

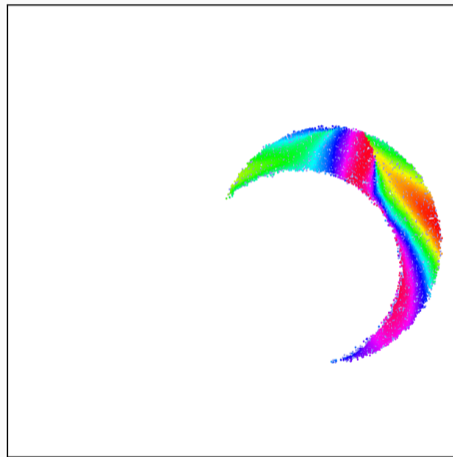


After

# The optimal transport plan



Before

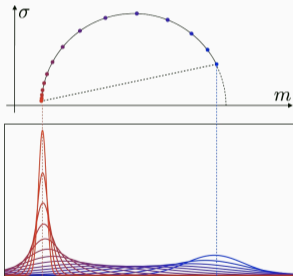


After

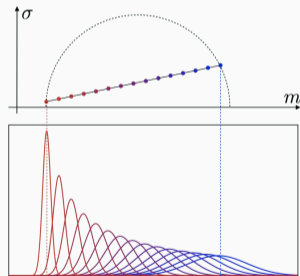
# OT induces a geometry-aware distance between probability distributions [PC18]

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

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**  $(m, \sigma)$ ?



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



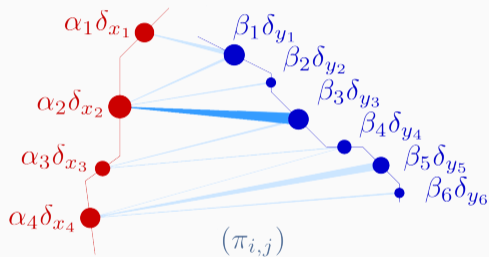
OT on  $\mathcal{N}(m, \sigma)$   
→ Flat **Euclidean** metric on  $(m, \sigma)$ .

## **How to solve the OT problem?**

---

# Duality: central planning with NM variables $\simeq$ outsourcing with $N + M$ variables

$$\text{OT}(\mathbf{A}, \mathbf{B}) = \min_{\pi} \langle \pi, \mathbf{C} \rangle, \quad \text{with } \mathbf{C}(x_i, y_j) = \frac{1}{p} \|x_i - y_j\|^p \quad \rightarrow \text{Assignment}$$
$$\text{s.t. } \pi \geq 0, \quad \pi \mathbf{1} = \alpha, \quad \pi^T \mathbf{1} = \beta$$

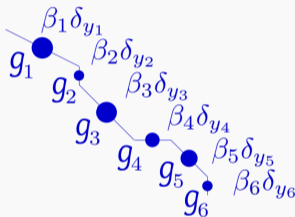
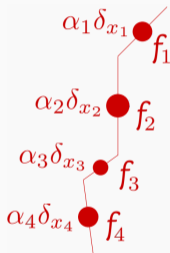


$$\sum_{i,j} \pi_{i,j} \mathbf{C}(x_i, y_j)$$

# Duality: central planning with NM variables $\simeq$ outsourcing with $N + M$ variables

$$\text{OT}(\mathbf{A}, \mathbf{B}) = \min_{\pi} \langle \pi, \mathbf{C} \rangle, \text{ with } \mathbf{C}(x_i, y_j) = \frac{1}{p} \|x_i - y_j\|^p \quad \rightarrow \text{Assignment}$$

$$\text{s.t. } \pi \geq 0, \quad \pi \mathbf{1} = \alpha, \quad \pi^T \mathbf{1} = \beta$$



$$\sum_{i,j} \pi_{i,j} \mathbf{C}(x_i, y_j)$$

$$\sum_i \alpha_i f_i + \sum_j \beta_j g_j$$

$$\max_{f, g} \quad \langle \alpha, f \rangle + \langle \beta, g \rangle$$

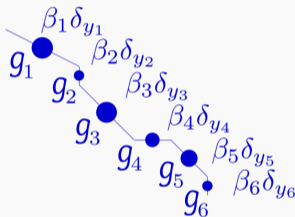
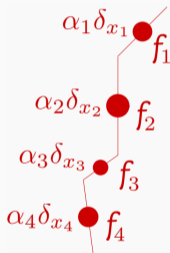
$\rightarrow$  FedEx

$$\text{s.t.} \quad f(x_i) + g(y_j) \leq \mathbf{C}(x_i, y_j),$$

# Duality: central planning with NM variables $\simeq$ outsourcing with $N + M$ variables

$$\text{OT}(\mathbf{A}, \mathbf{B}) = \min_{\pi} \langle \pi, \mathbf{C} \rangle, \text{ with } \mathbf{C}(x_i, y_j) = \frac{1}{p} \|x_i - y_j\|^p \quad \rightarrow \text{Assignment}$$

$$\text{s.t. } \pi \geq 0, \quad \pi \mathbf{1} = \alpha, \quad \pi^T \mathbf{1} = \beta$$



$$\sum_{i,j} \pi_{i,j} \mathbf{C}(x_i, y_j)$$

$$\sum_i \alpha_i f_i + \sum_j \beta_j g_j$$

$$= \max_{f, g} \langle \alpha, f \rangle + \langle \beta, g \rangle$$

$\rightarrow$  FedEx

$$\text{s.t. } f(x_i) + g(y_j) \leq \mathbf{C}(x_i, y_j),$$

# Being too greedy... doesn't work!

$$\text{OT}(\alpha, \beta) = \max_{\substack{(f_i) \in \mathbb{R}^N \\ (g_j) \in \mathbb{R}^M}} \sum_{i=1}^N \alpha_i f_i + \sum_{j=1}^M \beta_j g_j$$

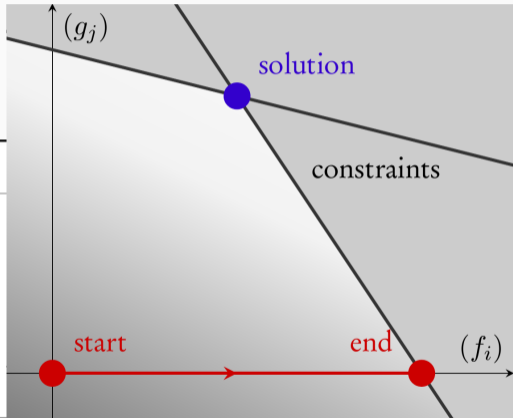
s.t.  $\forall i, j, f_i + g_j \leq \mathbf{C}(x_i, y_j)$

---

## Algorithm 3.1: Naive greedy algorithm

---

- 1:  $f_i, g_j \leftarrow \mathbf{0}_{\mathbb{R}^N}, \mathbf{0}_{\mathbb{R}^M}$
  - 2: **repeat**
  - 3:    $f_i \leftarrow \min_{j=1}^M [\mathbf{C}(x_i, y_j) - g_j]$
  - 4:    $g_j \leftarrow \min_{i=1}^N [\mathbf{C}(x_i, y_j) - f_i]$
  - 5: **until** convergence.
  - 6: **return**  $f_i, g_j$
- 



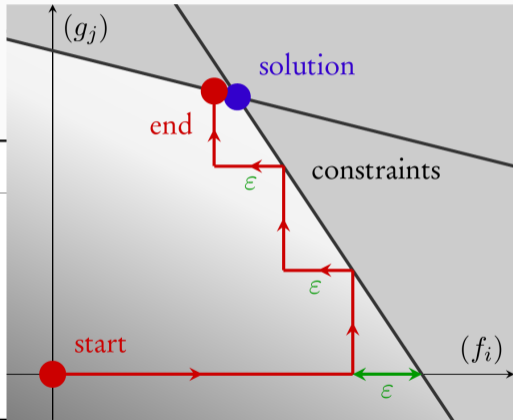
# The auction algorithm: take it easy with a slackness $\varepsilon > 0$

$$\text{OT}(\alpha, \beta) = \max_{\substack{(f_i) \in \mathbb{R}^N \\ (g_j) \in \mathbb{R}^M}} \sum_{i=1}^N \alpha_i f_i + \sum_{j=1}^M \beta_j g_j$$

s.t.  $\forall i, j, f_i + g_j \leq \mathbf{C}(x_i, y_j)$

## Algorithm 3.2: Pseudo-auction algorithm

- 1:  $f_i, g_j \leftarrow \mathbf{0}_{\mathbb{R}^N}, \mathbf{0}_{\mathbb{R}^M}$
- 2: **repeat**
- 3:  $f_i \leftarrow \min_{j=1}^M [\mathbf{C}(x_i, y_j) - g_j] - \varepsilon$
- 4:  $g_j \leftarrow \min_{i=1}^N [\mathbf{C}(x_i, y_j) - f_i]$
- 5: **until**  $\forall i, \exists j, f_i + g_j \geq \mathbf{C}(x_i, y_j) - \varepsilon$ .
- 6: **return**  $f_i, g_j$

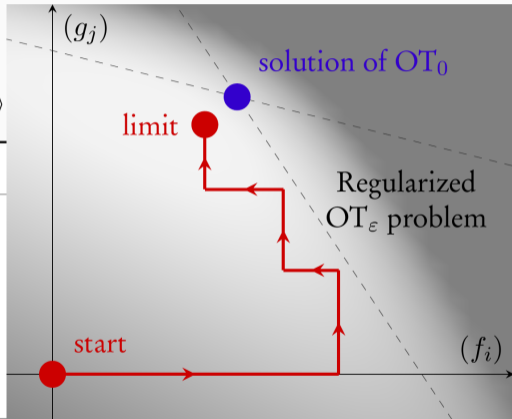


# The Sinkhorn algorithm: use a softmin, get a well-defined optimum

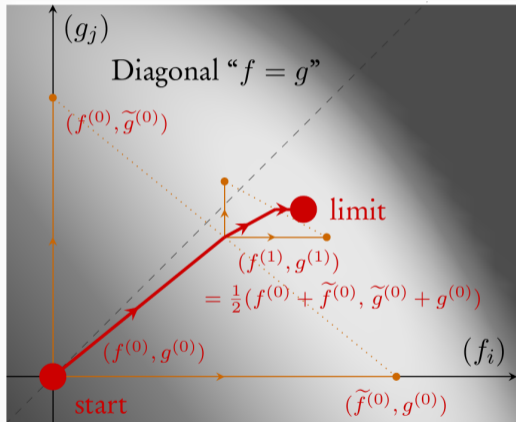
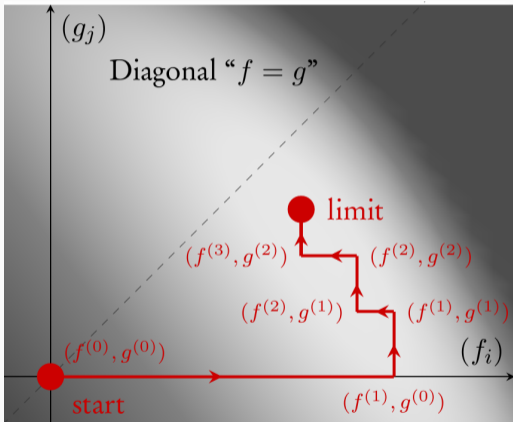
$$\text{OT}(\alpha, \beta) = \max_{\substack{(f_i) \in \mathbb{R}^N \\ (g_j) \in \mathbb{R}^M}} \sum_{i=1}^N \alpha_i f_i + \sum_{j=1}^M \beta_j g_j - \varepsilon \log \langle \alpha_i \otimes \beta_j, \exp \frac{1}{\varepsilon} [f_i \oplus g_j - \mathbf{C}_{ij}] \rangle$$

**Algorithm 3.3:** Sinkhorn or “soft-auction” algorithm

- 1:  $f_i, g_j \leftarrow \mathbf{0}_{\mathbb{R}^N}, \mathbf{0}_{\mathbb{R}^M}$
- 2: **repeat**
- 3:  $f_i \leftarrow -\varepsilon \log \sum_{j=1}^M \beta_j \exp \frac{1}{\varepsilon} [g_j - \mathbf{C}(x_i, y_j)]$
- 4:  $g_j \leftarrow -\varepsilon \log \sum_{i=1}^N \alpha_i \exp \frac{1}{\varepsilon} [f_i - \mathbf{C}(x_i, y_j)]$
- 5: **until** convergence up to a set tolerance.
- 6: **return**  $f_i, g_j$



# The symmetric Sinkhorn algorithm: stay close to the diagonal if $A \simeq B$



## Remark 1: a streamlined algorithm

One key operation – the soft, **weighted distance transform**:

$$\forall i \in [1, N], f(x_i) \leftarrow \min_{y \sim \beta} [\mathbf{C}(x_i, y) - g(y)] = -\varepsilon \log \sum_{j=1}^M \beta_j \exp \frac{1}{\varepsilon} [g_j - \mathbf{C}(x_i, y_j)].$$

Similar to the chamfer distance transform, convolution with a Gaussian kernel...

Fast implementations with **pyKeOps**:

- If  $\mathbf{C}(x_i, y_j)$  is a closed formula: **bruteforce** scales to  $N, M \simeq 100k$  in 10ms on a GPU.
- If **A** and **B** have a low-dimensional support:  
use a clustering and **truncation** strategy to get a x10 speed-up.
- If **A** and **B** are supported on a 2D or 3D grid and  $\mathbf{C}(x_i, y_j) = \frac{1}{2} \|x_i - y_j\|^2$ :  
use a **separable** distance transform to get a second x10 speed-up.  
(N.B.: FFTs run into numerical accuracy issues.)

## Remark 2: annealing works!

The **Auction/Sinkhorn** algorithms:

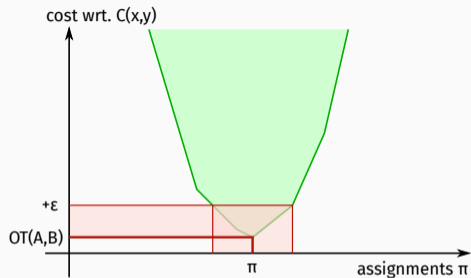
- Improve the dual cost by at least  $\varepsilon$  at each (early) step.
- Reach an  $\varepsilon$ -optimal solution with  $(\max C) / \varepsilon$  steps.

Simple heuristic: run the optimization with **decreasing values** of  $\varepsilon$ .

$\varepsilon$ -scaling

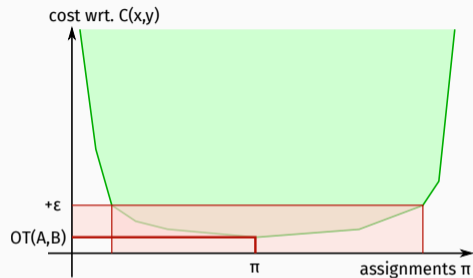
- = **simulated annealing**
- = **multiscale** strategy
- = **divide and conquer**

## Remark 3: the curse of dimensionality



### In **low dimension**:

- $\|x - y\|$  takes large and small values.
- The OT objective is **peaky** wrt.  $\pi$ .
- $\epsilon$ -optimal solutions are **useful**.
- $OT(\text{discrete samples}) \simeq OT(\text{underlying distributions})$



### With **high dimensional** distributions:

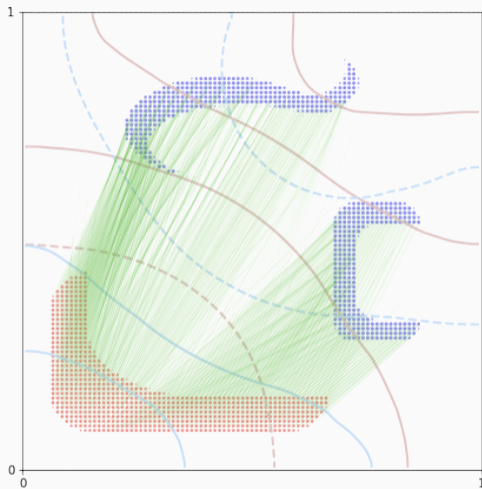
- $\|x - y\|$  gets closer to a constant.
- The OT objective is **flat** wrt.  $\pi$ .
- $\epsilon$ -optimal solutions are **random**.
- $OT(\text{discrete samples}) \neq OT(\text{underlying distributions})$

## To recap 80+ years of work...

Key dates for discrete optimal transport with  $N$  points:

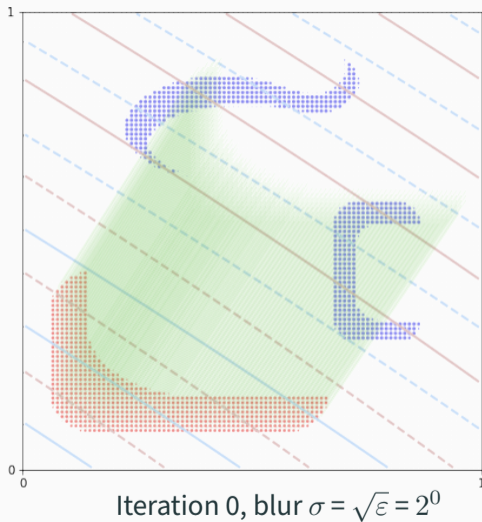
- [Kan42]: **Dual** problem of Kantorovitch.
- [Kuh55]: **Hungarian** methods in  $O(N^3)$ .
- [Ber79]: **Auction** algorithm in  $O(N^2)$ .
- [KY94]: **SoftAssign** = Sinkhorn + simulated annealing, in  $O(N^2)$ .
- [GRL<sup>+</sup>98, CR00]: **Robust Point Matching** = Sinkhorn as a loss.
- [Cut13]: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

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

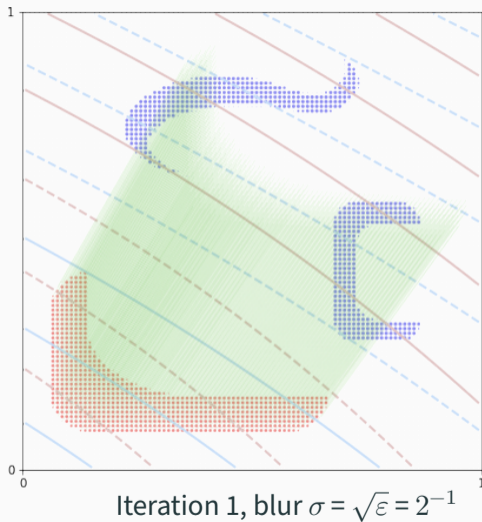


OT plan in 2D.

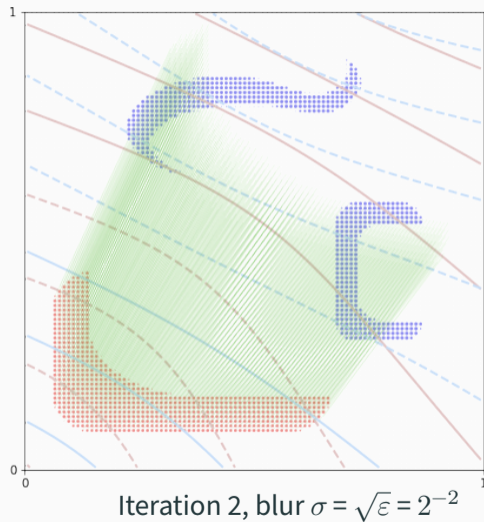
# Visualizing $F$ , $G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



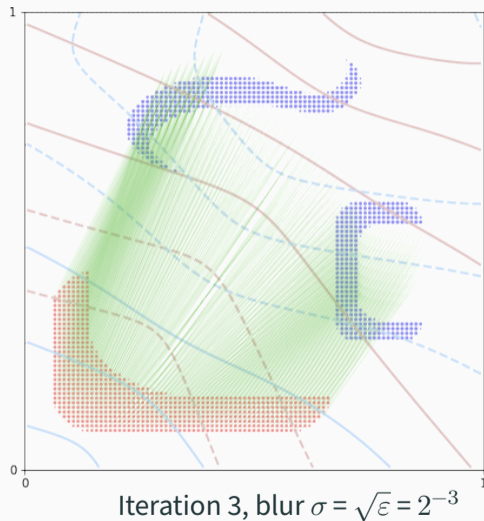
# Visualizing $F$ , $G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



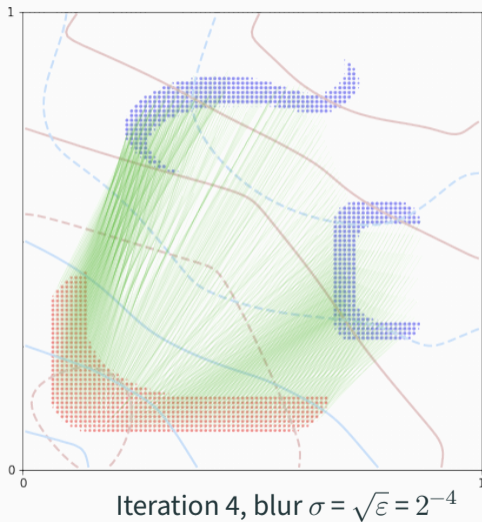
# Visualizing $F, G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



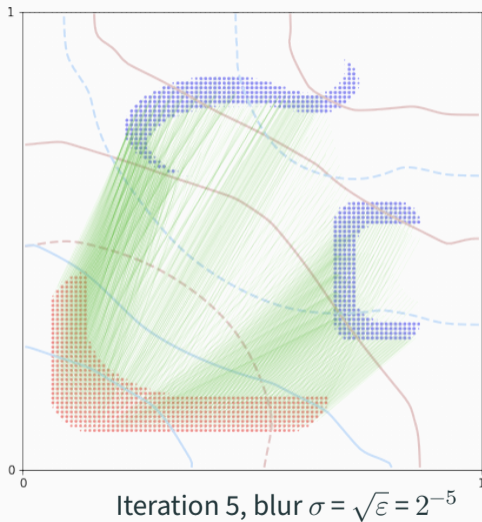
# Visualizing $F, G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



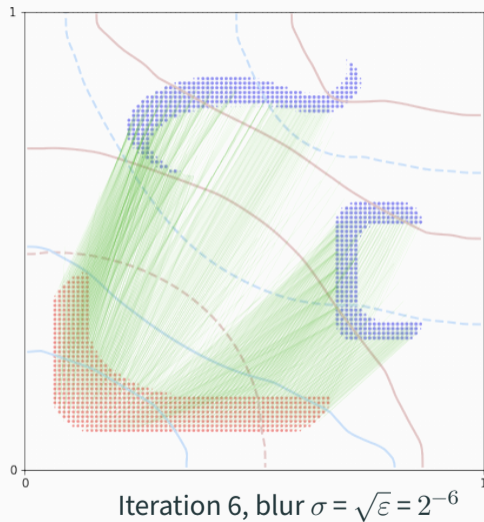
# Visualizing $F, G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



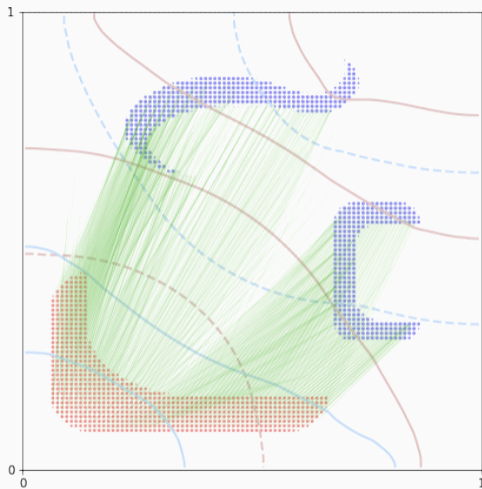
# Visualizing $F, G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



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

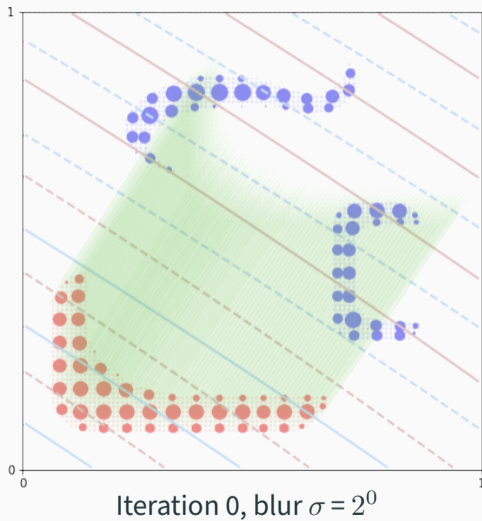


# Visualizing $F, G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$

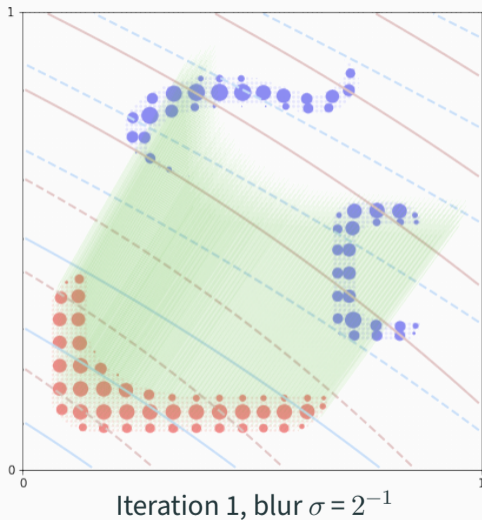


Iteration 7, blur  $\sigma = \sqrt{\varepsilon} = .01$

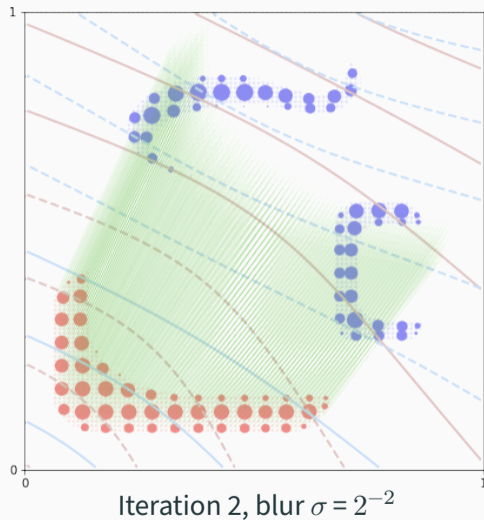
# Visualizing $F$ , $G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



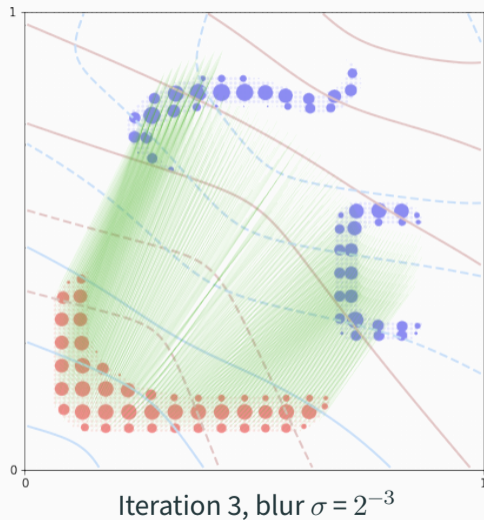
# Visualizing $F, G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



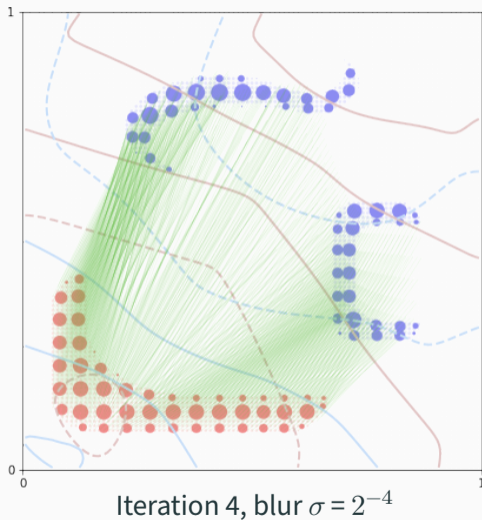
# Visualizing $F, G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



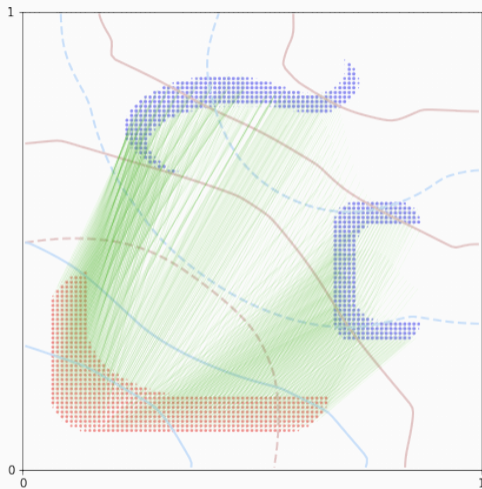
# Visualizing $F, G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$



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

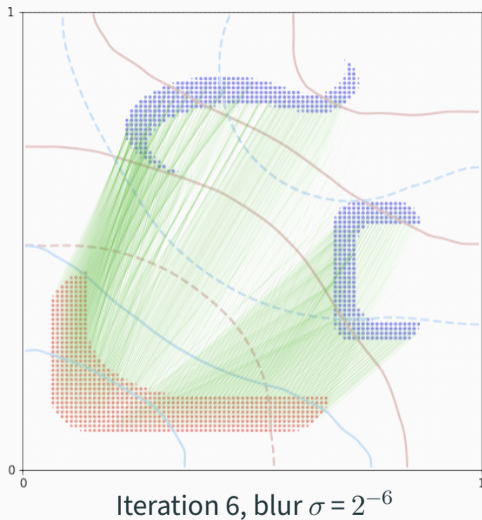


# Visualizing $F, G$ and the Brenier map $\nabla F(x_i) = -\frac{1}{\alpha_i} \partial_{x_i} \mathbf{OT}(\alpha, \beta)$

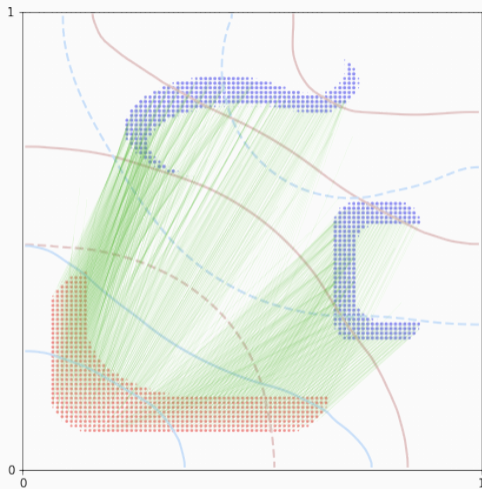


Iteration 5, blur  $\sigma = 2^{-5}$

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



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



Iteration 7, blur  $\sigma = .01$

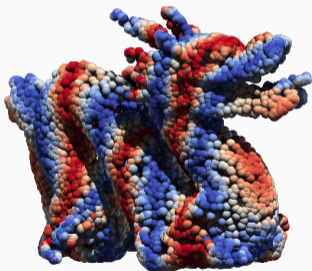
## Scaling up optimal transport to anatomical data

Progresses of the last 15 years add up to a  $\times 100$  -  $\times 1000$  acceleration:

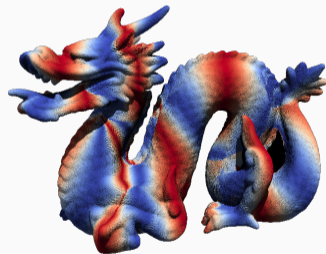
Sinkhorn GPU  $\xrightarrow{\times 10}$  + KeOps  $\xrightarrow{\times 10}$  + Annealing  $\xrightarrow{\times 10}$  + Multi-scale

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

```
pip install  
geomloss  
+  
modern GPU  
(1 000 €)
```

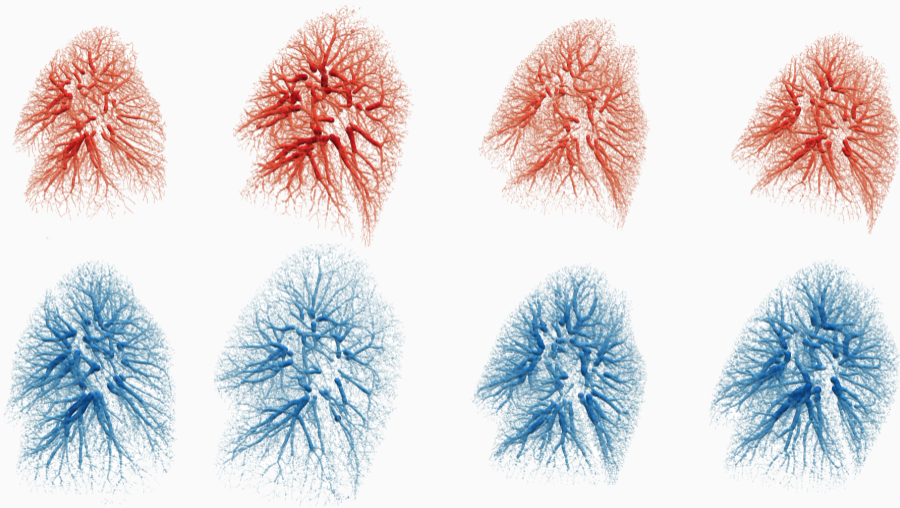


10k points in 30-50ms



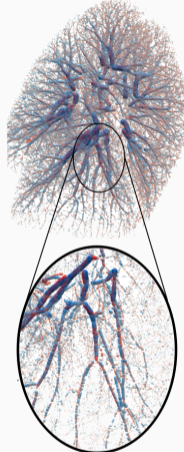
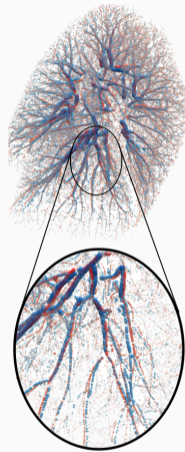
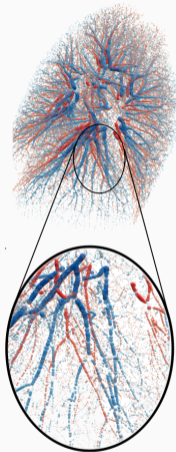
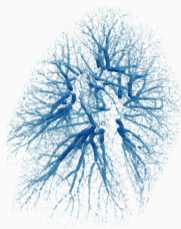
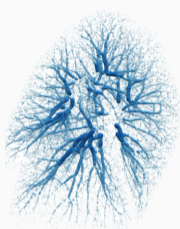
**100k points in 100-200ms**

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



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

# Three-steps registration



0. Input data

1. Pre-alignment

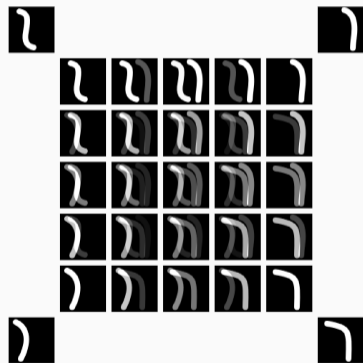
Zoom !

2. Deep registration

3. Fine-tuning

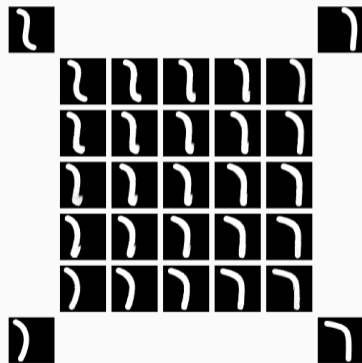
## Wasserstein barycenters [AC11]

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



**Euclidean** barycenters.

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



**Wasserstein** barycenters.

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

# Wasserstein barycenters

From a computational perspective:

- The problem is **convex** (easy) wrt. the weights.
- The support of the barycenter lies in the **convex hull** of the input distributions.

The **curse of dimensionality** hits hard:

- In high dimension, identifying the support can become **NP-hard**.
- In dimensions 2 and 3, we can just use a grid and recover **super fast** algorithms.  
Computing OT distances and barycenters between **density maps** is a solved problem.

⇒ We can now **easily** do manifold learning (= non-linear Model Order Reduction) in Wasserstein spaces of **2D and 3D** distributions.

## **Incompressible particles**

---

## Two very talented colleagues



**Maciej Buze**

Heriot-Watt University

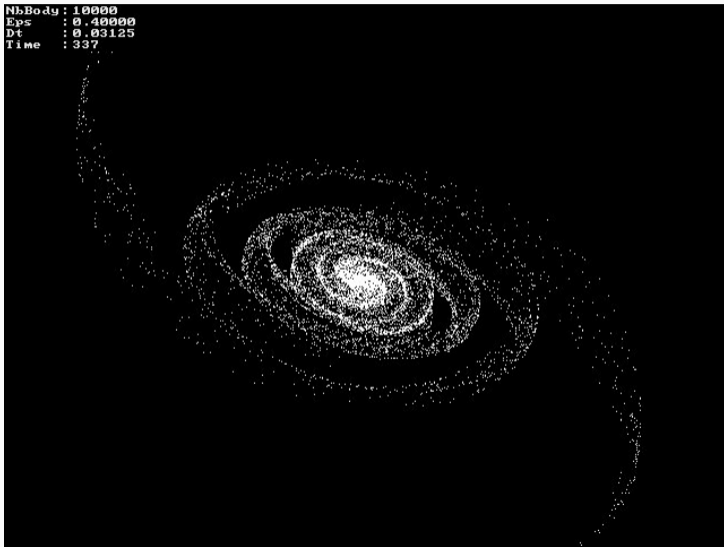


**Antoine Diez**

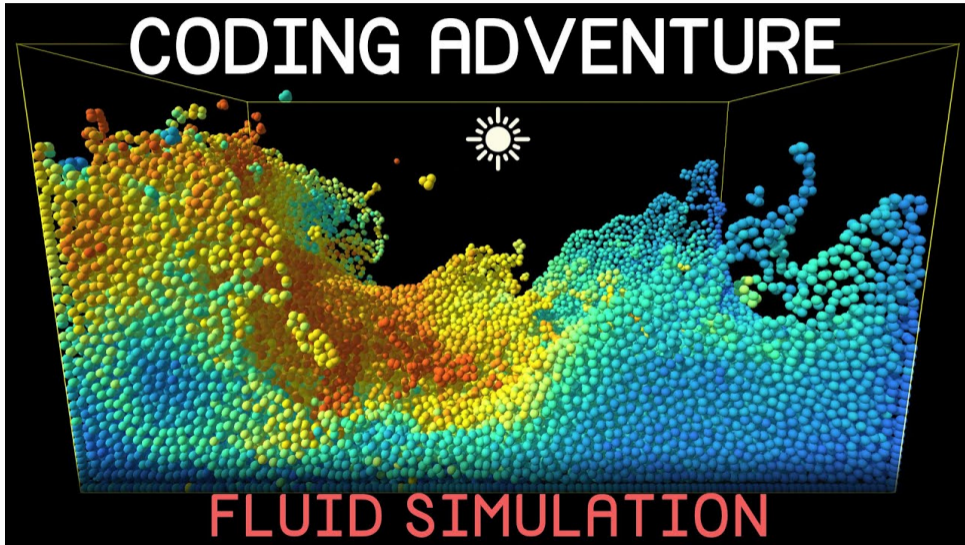
Kyoto University

## Original motivation: the N-body problem [Pri11]

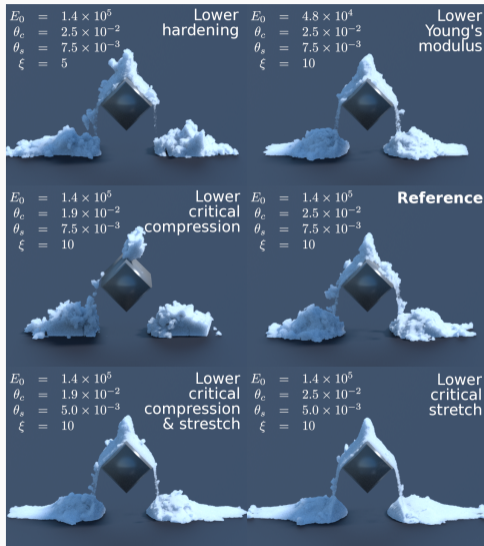
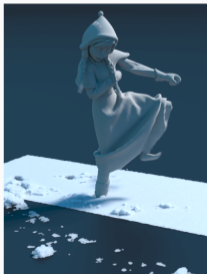
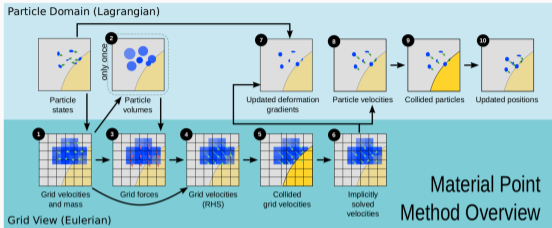
```
NbBody : 10000  
Eps     : 0.40000  
Dt      : 0.03125  
Time    : 337
```



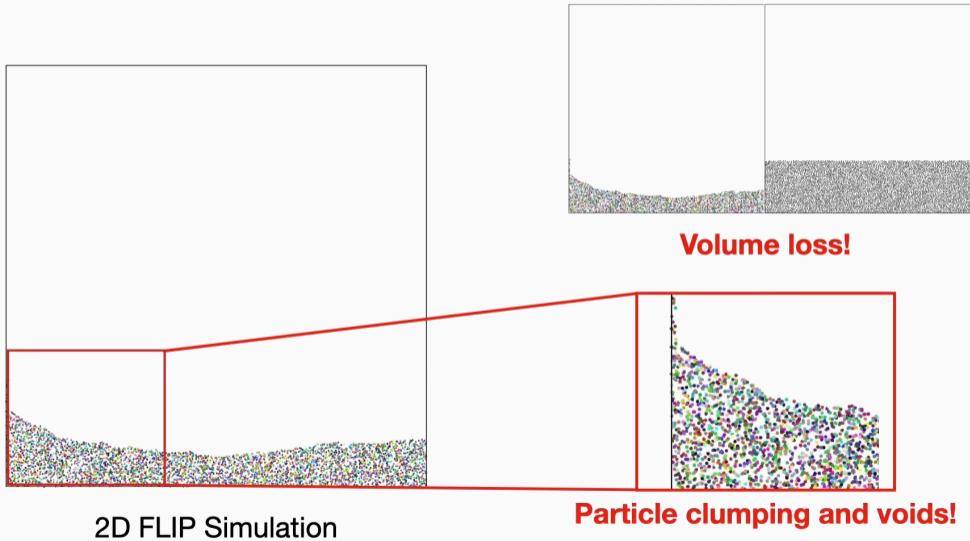
Coding a simple fluid simulation is now a matter of hours [Lag23]



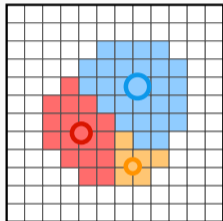
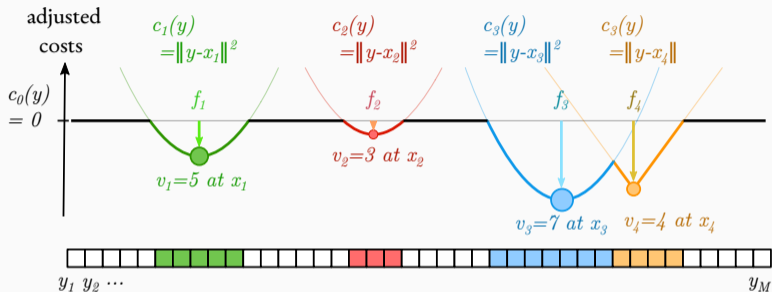
# The material point method: Disney's Frozen [SSC+13]



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

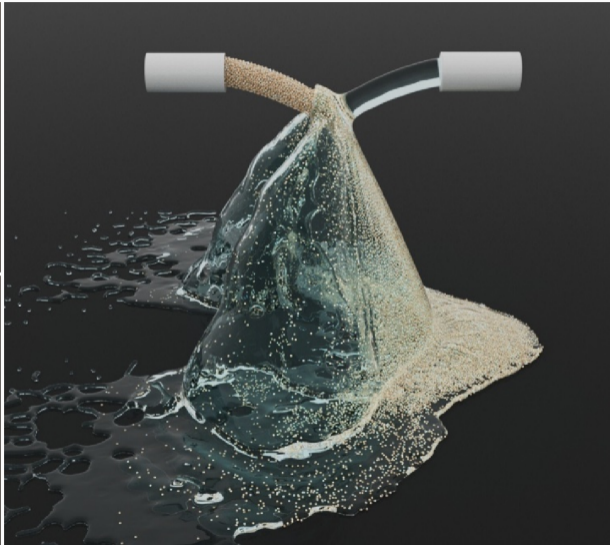
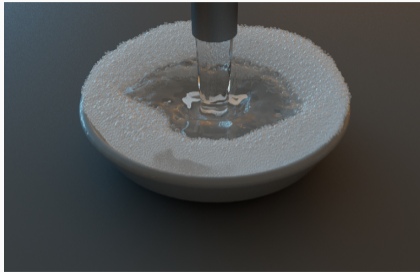


# Use power diagrams i.e. semi-discrete optimal transport

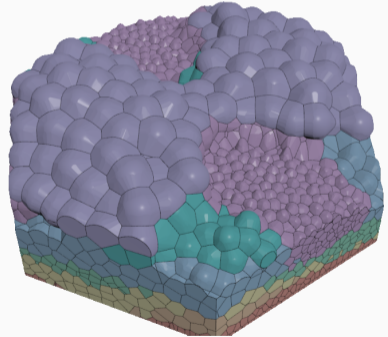
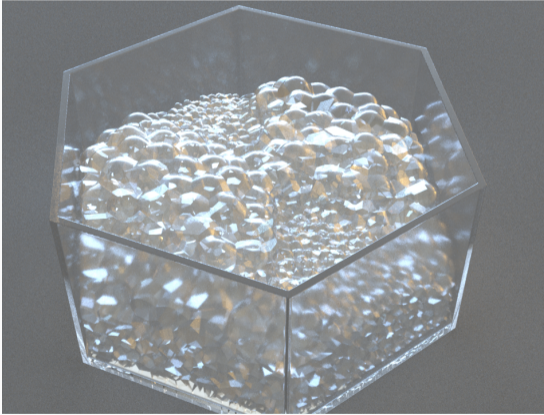


- The  $f_i$ 's maximize the dual objective  $\sum_{i=1}^N v_i f_i + \int_{y \in \Omega} \min_{i=0}^N [c_i(y) - f_i] dy$ .
- **Optimality** conditions  $\iff \text{Vol}(\text{Cell}_i) = v_i$ .
- To **compute the cells**, the objective and its gradient:
  - If  $c_i(y) = \|y - x_i\|^2$  for all cells,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

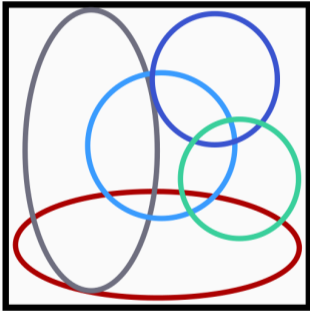
These simulations alternate between:

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.

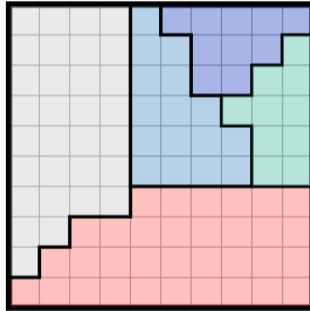
See e.g. Thomas Gallouët for a rigorous analysis with Mérigot, Lévy, etc.

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

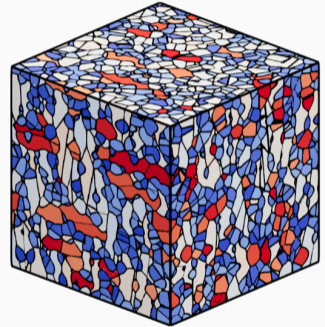
# Anisotropic power diagrams let us model polycrystalline metals [BFR<sup>+</sup>24]



Ellipsoids.

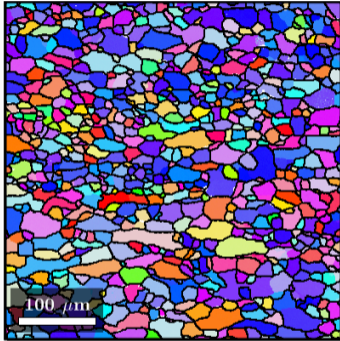


Pixel cells.

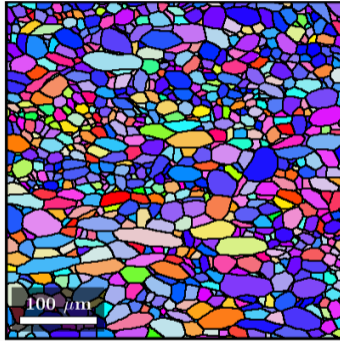


5,000 crystals in 3D.

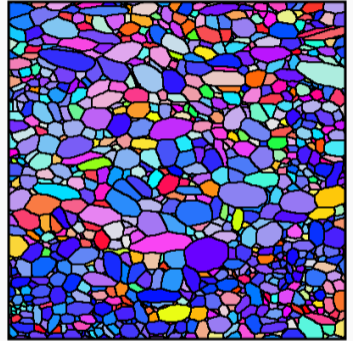
## Fit to real EBSD scan of low-carbon steel [BFR+24]



Data from Tata steel.



Our APD model.

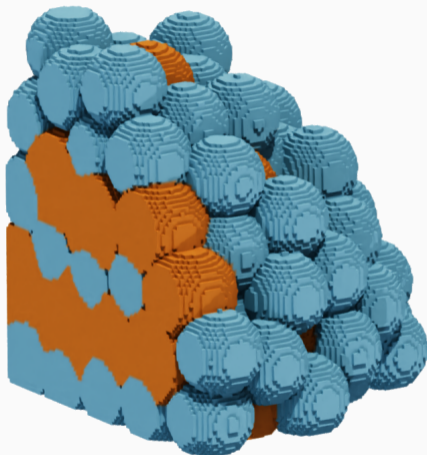


New synthetic image.

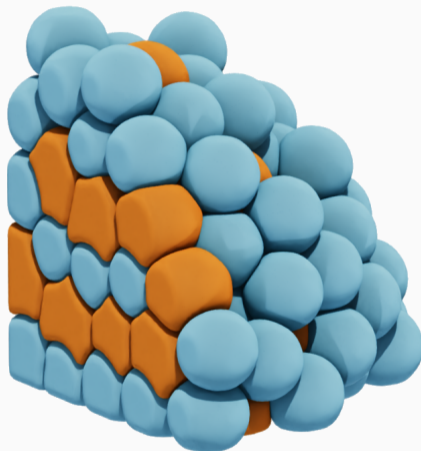
We can generate new, realistic 3D images with **prescribed properties** in seconds.

⇒ pip install PyAPD ⇐

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



The **raw** 100x100x100 pixel grid...

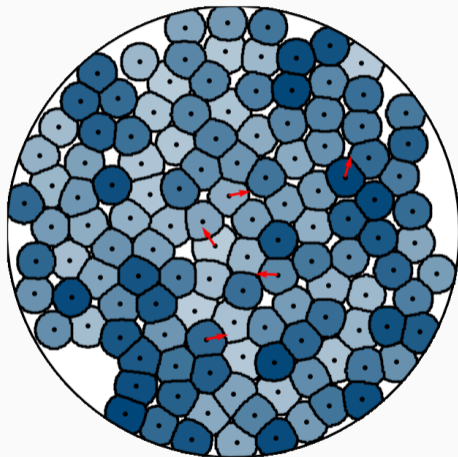


with some Hollywood **makeup**.

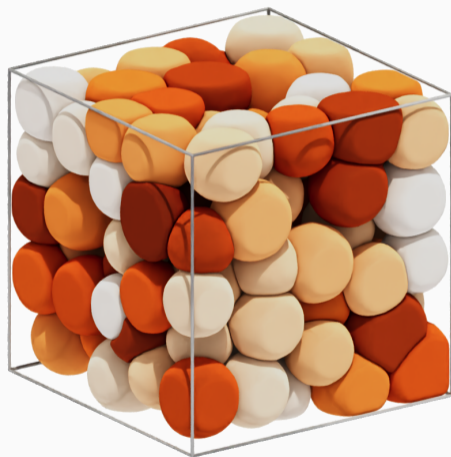
## Let's visit Antoine's website

⇒ <https://iceshot.readthedocs.io> ⇐

## Run-and-tumble motion [DF24]

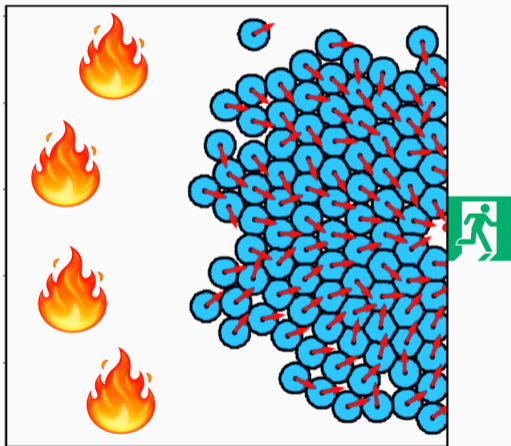


2D disk.

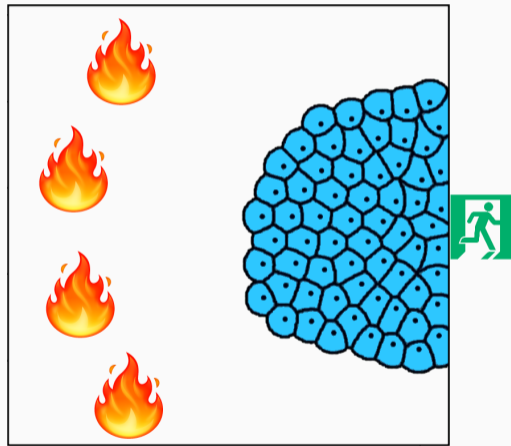


3D cube.

## Fire alarm! [DF24]

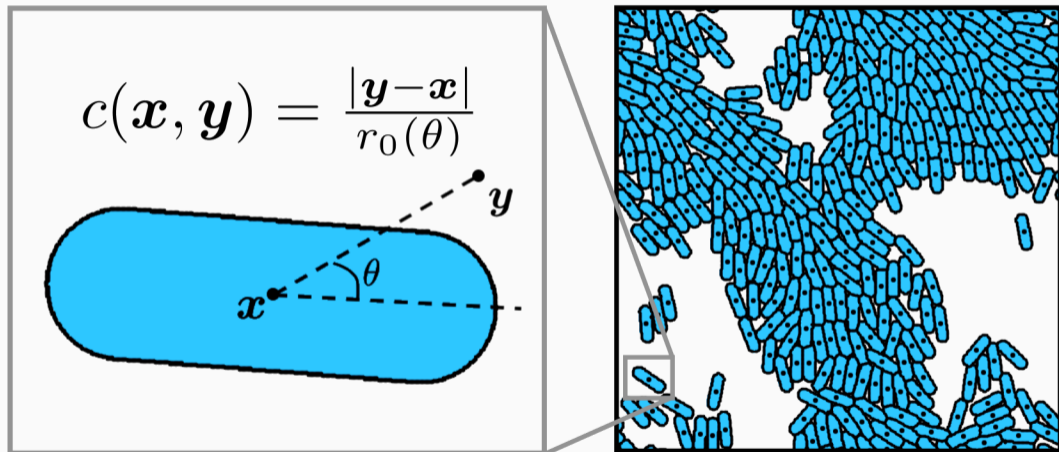


Hard particles **burn**.

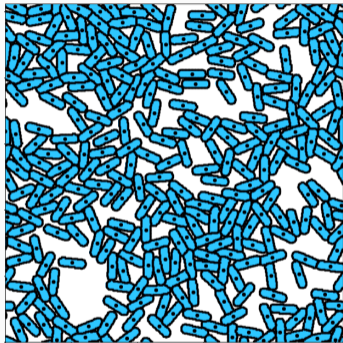


Soft particles **escape**.

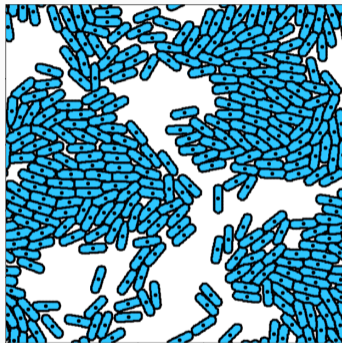
## Self-organizing swarms of blind, incompressible swimmers [DF24]



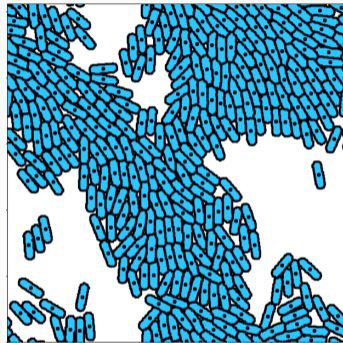
## Self-organizing swarms of blind, incompressible swimmers [DF24]



$t = 0$



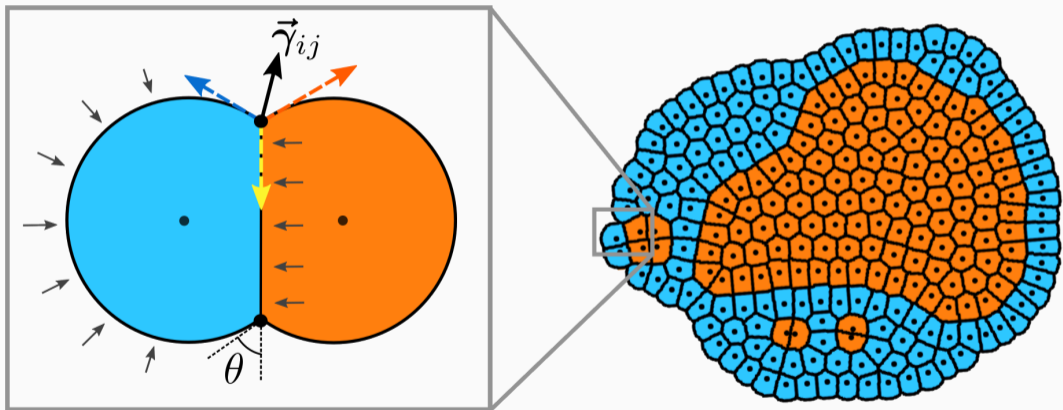
$t = 4$



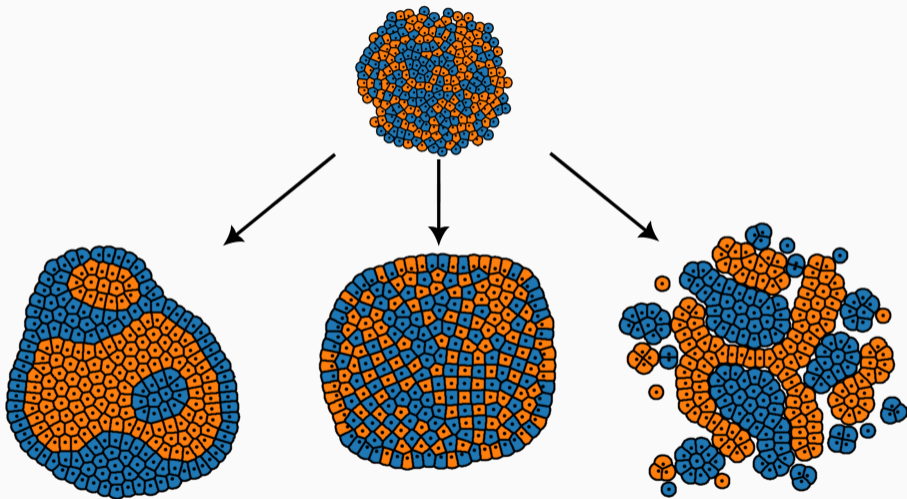
$t = 30$

**Order emerges** out of blind collisions and re-alignments.

## Surface tension [DF24]



## Surface tension [DF24] – playing with the energy parameters



## **Future developments**

---

# High-dimensional backend – funded by LLMs [YLY<sup>+</sup>26]

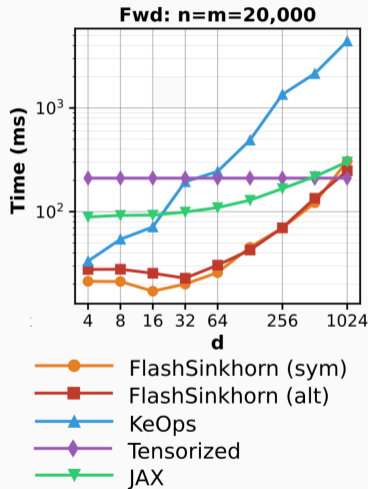
**Attention layers** are key building blocks of transformer networks. They correspond to normalized convolutions with a kernel

$$k(x, y) = e^{x^\top y}$$

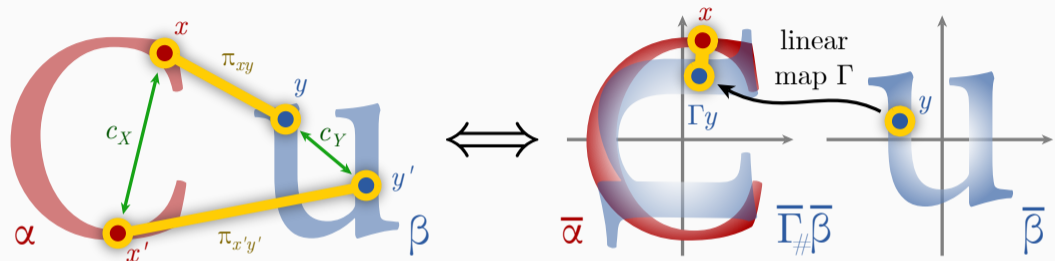
Optimized implementations are now supported by PyTorch: FlashAttention, FlexAttention...

They enable fast Sinkhorn implementations in **any dimension** when:

$$C(x_i, y_j) = \|x_i - y_j\|^2 = \|x_i\|^2 + \|y_j\|^2 - 2 \cdot x_i^\top y_j.$$



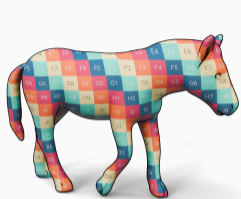
## Gromov-Wasserstein at scale, beyond squared norms [HFV26]



GW minimizes **distortions** instead of **displacements**.  
This is a much harder, non-convex optimization problem.

But for many distortion penalties, GW is equivalent to a **linear alignment** problem in a **high-dimensional feature space** with a **squared Euclidean OT** loss.

# Gromov-Wasserstein at scale, beyond squared norms [HFV26]



(a) Source



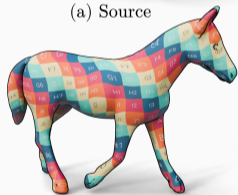
(b) E-GW



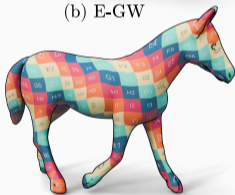
(c) K-GW



(d) QLR-GW



(e) CNT-GW



(f) MS-GW



(g) LR-GW, rank = 100



(h) LR-GW, rank = 500

This unlocks the computation of **sharp**,  
distortion-minimizing maps for **high-resolution 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** :
  - Super-fast solvers on **simple domains**, especially 2D/3D spaces.
  - **Fundamental tool** at the intersection of geometry and statistics.
- **“Video-game physics”** is great for modelling:
  - **Expressive**, real-time simulations that you can implement without being a Finite Elements guru: XPBD, DiffPD, Taichi...
- GPUs are more **versatile** than you think.
  - Ongoing work to provide **fast GPU backends** to researchers, going beyond what Google and Facebook are ready to pay for.

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

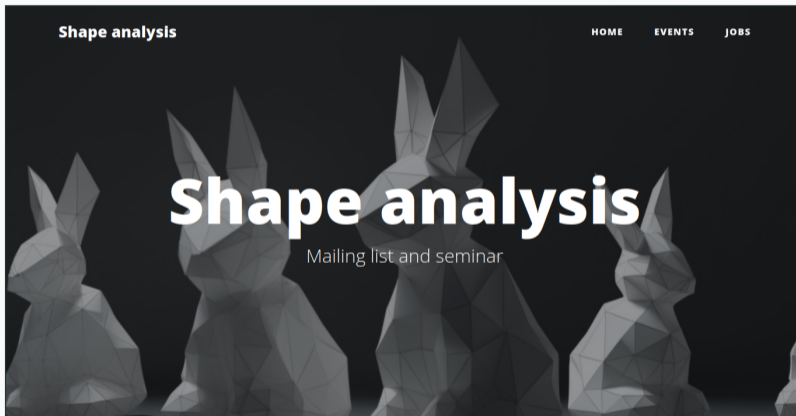
I have funding for a **2-year post-doc** around KeOps and structured tensors :-)



## Documentation and tutorials are available online




[shape-analysis.github.io](https://shape-analysis.github.io)



Monthly seminar, videos on YouTube.

## References

---

 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.*

 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.*

 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.

 Guillaume Houry, Jean Feydy, and François-Xavier Vialard.

**Gromov-Wasserstein at scale, beyond squared norms.**

*arXiv preprint arXiv:2602.06658*, 2026.

 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.
-  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.

 Felix X-F Ye, Xingjie Li, An Yu, Ming-Ching Chang, Linsong Chu, and Davis Wertheimer.

**FlashSinkhorn: IO-aware entropic optimal transport.**

*arXiv preprint arXiv:2602.03067, 2026.*