Geometric data analysis, beyond convolutions

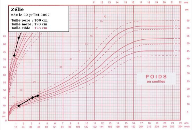
Jean Feydy,
under the supervision of Alain Trouvé.

Online PhD defense — July 2, 2020.

ENS Paris, ENS Paris-Saclay, Imperial College London.

Joint work with B. Charlier, J. Glaunès (numerical foundations),
T. Séjourné, F.-X. Vialard, G. Peyré (optimal transport theory),
P. Roussillon, P. Gori (applications to neuroanatomy).

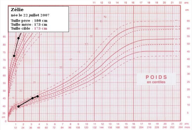
The medical imaging pipeline [Ptr19, EPW+11]



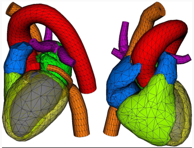
Valuable information

Sensor data

The medical imaging pipeline [Ptr19, EPW+11]



Valuable information

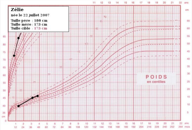


High-level description

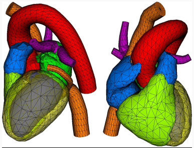
Raw image

Sensor data

The medical imaging pipeline [Ptr19, EPW+11]



Valuable information



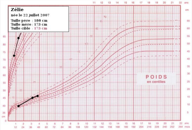
High-level description

Raw image

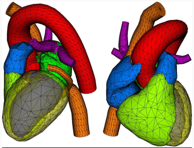
Signal processing

Sensor data

The medical imaging pipeline [Ptr19, EPW+11]



Valuable information



High-level description

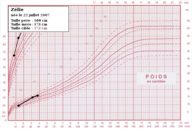
Computational anatomy

Raw image

Signal processing

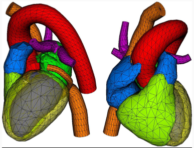
Sensor data

The medical imaging pipeline [Ptr19, EPW+11]



Valuable information

Statistics



High-level description

Computational anatomy

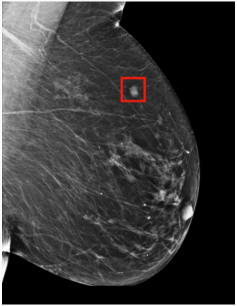
Raw image

Signal processing

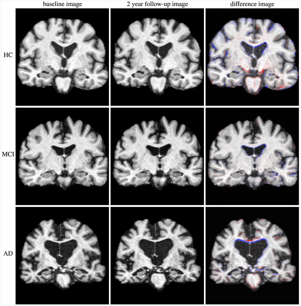
Sensor data

Computational anatomy [CSG19, LSG+18, CMN14]

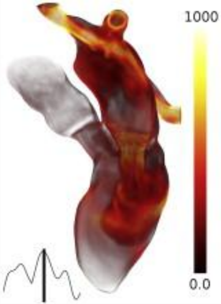
Three main problems:



Spot patterns



Analyze variations



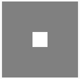
Fit models

The key operation for imaging: filtering, aka. “convolution product”

Convolution = weighted average of the neighboring pixels :

Cheap generalization of the **product** “$a \cdot x$”,

parameterized by the coefficients of a **small filter** φ.



$\star$

$=$

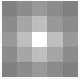
φ x $\varphi \star x$

The key operation for imaging: filtering, aka. "convolution product"

Convolution = weighted average of the neighboring pixels :

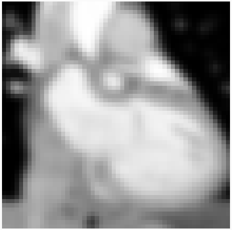
Cheap generalization of the **product** "$a \cdot x$",

parameterized by the coefficients of a **small filter** φ.



$\star$

$=$



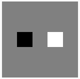
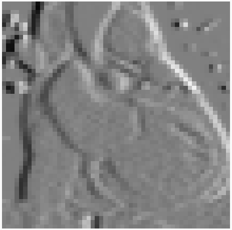
φ x $\varphi \star x$

The key operation for imaging: filtering, aka. “convolution product”

Convolution = weighted average of the neighboring pixels :

Cheap generalization of the **product** “$a \cdot x$”,

parameterized by the coefficients of a **small filter** φ.

 $\star$  $=$ 

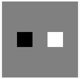
φ $\quad$ x $\quad$ $\varphi \star x$

The key operation for imaging: filtering, aka. “convolution product”

Convolution = weighted average of the neighboring pixels :

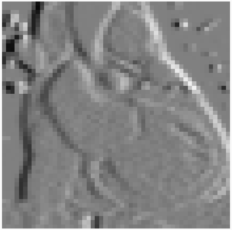
Cheap generalization of the **product** “$a \cdot x$”,

parameterized by the coefficients of a **small filter** φ.



$\star$

$=$



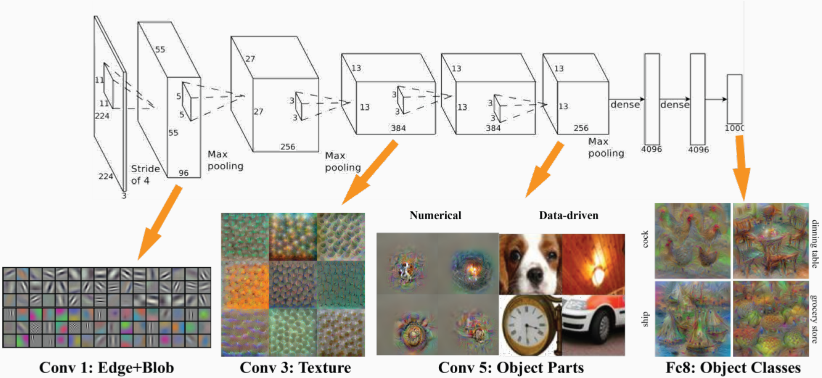
φ $\quad$ x $\quad$ $\varphi \star x$

Combine convolutions + pointwise operations + zooms/unzooms.

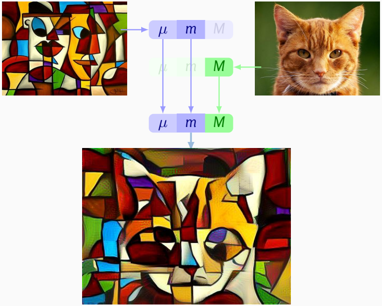
How do we pick the convolution weights?

2010-2020: the deep learning revolution [Mal16, PMC11, WZTF17]

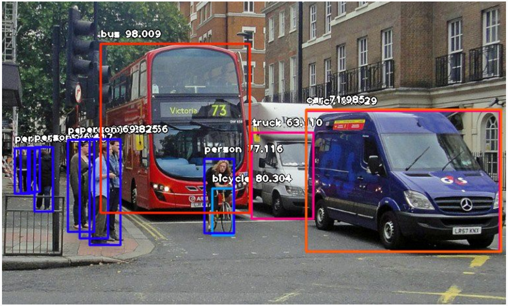
Explicit wavelets $\longrightarrow$ **Data-driven** Convolutional Neural Networks



(Wavelets → CNNs) = improvement for… [NN16, Ola18]

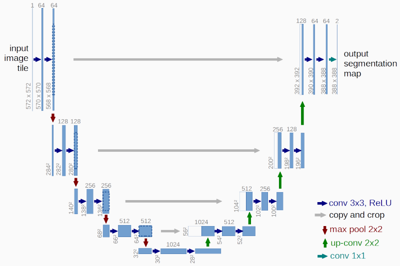


Texture processing

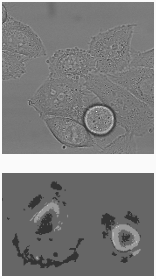


Object detection

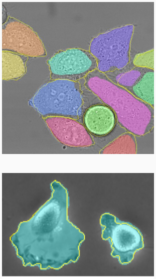
Segmentation with U-nets [RFB15]



Architecture



Input



Output

Shape analysis is still a very open problem

Geometric questions on segmented shapes:

Shape analysis is still a very open problem

Geometric questions on segmented shapes:

- Is this **heart** beating all right?
- How should we reconstruct this **mandible**?
- Has this **brain** grown or shrunk since last year?
- Can we link these anatomical changes to **other signals**?

Shape analysis is still a very open problem

Geometric questions on segmented shapes:

- Is this **heart** beating all right?
- How should we reconstruct this **mandible**?
- Has this **brain** grown or shrunk since last year?
- Can we link these anatomical changes to **other signals**?

Over the last 30 years, **robust methods** have been designed to answer these questions.

Today, we want to improve them with **data-driven** insights. This is challenging.

To replicate the "wavelets $\rightarrow$ CNNs" revolution in our field, we need to **revamp our numerical toolbox**.

Motivations

Geometric data analysis, beyond convolutions:

Motivations

Geometric data analysis, beyond convolutions:

- Focus on **geometric data**:
  segmentation maps, point clouds, surface meshes, etc.
- Focus on **geometric methods**:
  K-nearest neighbors, kernel methods, optimal transport, etc.
- Provide new **computational routines**:
  expand the toolbox for data sciences.

Motivations

Geometric data analysis, beyond convolutions:

- Focus on **geometric data**:
  segmentation maps, point clouds, surface meshes, etc.
- Focus on **geometric methods**:
  K-nearest neighbors, kernel methods, optimal transport, etc.
- Provide new **computational routines**:
  expand the toolbox for data sciences.

We work with 10^3-10^6 points in dimension 2 to 10.
We focus on geometry and speed.

Outline of the thesis

Today, we will talk about:

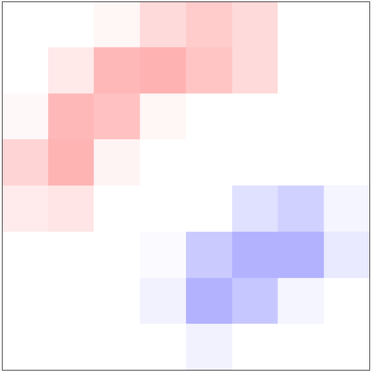
1. **Fast geometry** with symbolic matrices.
2. Scalable **optimal transport**.
3. New directions for **computational anatomy**.

Fast geometry with symbolic matrices.

Benjamin Charlier

Joan Glaunès

Working with images – implicit coordinates



$N_x = 8, N_y = 8, N_z = 1$
$C = 3$ (RGB channels)

Bitmap images and volumes:

- (N_x, N_y, N_z, C) pixels.
- `.bmp`, `.png`, `.jpg`
- Eulerian.

\+ **Standard** for radiology.

\+ Easy to find neighbors.

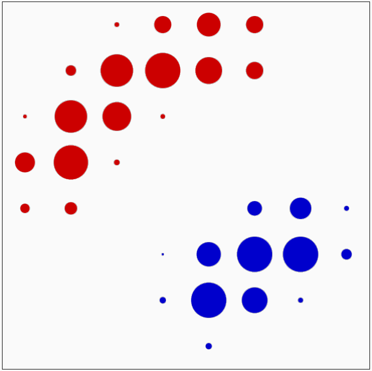
\+ Fast **convolutions.**

\+ Fast Fourier transforms.

– Precision vs. Memory.

– Cumbersome **deformations.**

Working with point clouds – explicit coordinates



$N = 31, D = 2,$
$C = 3$ (RGB channels)

Point clouds, sampled data:

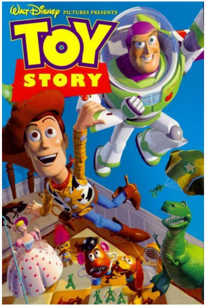
- (N, D) coordinates.
- (N, C) signals.
- `.svg`
- Lagrangian.

+ Compact representation.
+ High precision for geometry.
+ **Easy to deform.**

– Cumbersome convolutions and Fourier transforms.

We can now get the best of both worlds

Video games: millions of **textured triangles**, processed in real-time.



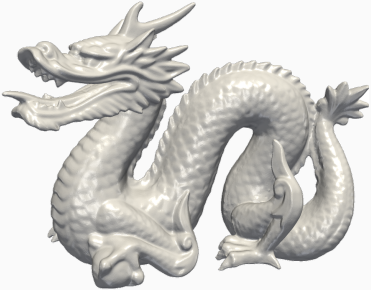
→

1995

Today

In research, 1,000,000 is the new 10,000 [CL96]

$\rightarrow$



11k triangles

871k triangles

Thank God for the gamers!

Nvidia RTX 2080 Ti, **~1,500$** = **4,352 cores**, 11Gb RAM.

Incredible performance: $\sim 10^{12}$ operations $(+, \times, \ldots)$ per second.

One catch: complex **memory management**, with 6 types of buffers.

GPU programming is a full-time job.

Deep learning frameworks: unlocking GPUs for research

TensorFlow and PyTorch combine:

+ Array-centric **Python interface**.
+ CPU *and* **GPU** backends.
+ **Automatic differentiation** engine.
+ Excellent support for imaging (convolutions) and linear algebra.

Deep learning frameworks: unlocking GPUs for research

TensorFlow and PyTorch combine:

+ Array-centric **Python interface**.
+ CPU *and* **GPU** backends.
+ **Automatic differentiation** engine.
+ Excellent support for imaging (convolutions) and linear algebra.

$\Longrightarrow$ Ideally suited for research.

Efficient algorithms still rely on C++ foundations

Explicit **C++/CUDA implementations** with a **Python interface** for:

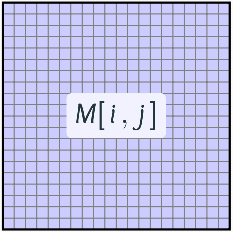
- Linear algebra (cuBLAS).
- Convolutions (cuDNN).
- Fourier (cuFFT) and wavelet transforms (Kymatio).

Geometric algorithms do not benefit from the same level of integration. Researchers can either:

- Work directly in C++/CUDA – cumbersome for data sciences.
- Rely on **explicit distance matrices.**

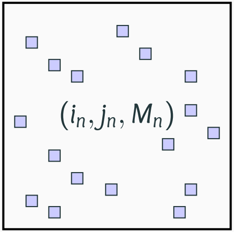
```
RuntimeError: cuda runtime error (2) : out of memory at
/opt/conda/.../THCStorage.cu:66
```

We provide efficient support for distance-like matrices



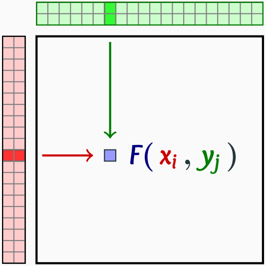
Dense matrix

Coefficients only



Sparse matrix

Coordinates + coeffs



Symbolic matrix

Formula + data

$\Longrightarrow$ `pip install pykeops` $\Longleftarrow$

KeOps works with PyTorch, NumPy, Matlab and R

```
# Large point cloud in ℝ^50:
import torch
N, D = 10**6, 50
x = torch.rand(N, D).cuda()  # (1M, 50) array

# Compute the nearest neighbor of every point:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x.view(N, 1, D))   # x_i is a "column"
x_j = LazyTensor(x.view(1, N, D))   # x_j is a "line"
D_ij = ((x_i - x_j)**2).sum(dim=2)  # (N, M) symbolic
indices_i = D_ij.argmin(dim=1)      # -> (N,) dense
```

On par with reference C++/CUDA libraries (FAISS-GPU).

Combining performance and flexibility

We can work with arbitrary formulas:

```
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
```

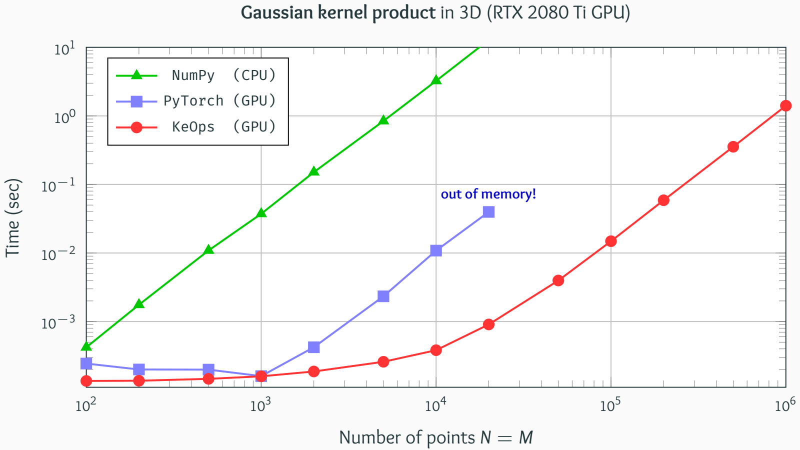
$\Longrightarrow \times$200 acceleration for UMAP on hyperbolic spaces.

KeOps supports:

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

Scaling up to large datasets

$$a_i \leftarrow \sum_{j=1}^{M} \underbrace{\exp(-\|x_i - y_j\|^2/2\sigma^2)}_{k(x_i,y_j)} \; b_j, \quad \forall i \in [\![1, N]\!]$$



+ **Cross-platform**: C++, R, Matlab, NumPy *and* PyTorch.
+ **Versatile**: many operations, variables, reductions.
+ **Efficient**: $O(N)$ memory, competitive runtimes.
+ **Powerful**: automatic differentiation, block-sparsity, etc.
+ **Transparent**: interface with **SciPy**, GPytorch, etc.
+ **Fully documented**:

  `www.kernel-operations.io`

– Requires a C++/CUDA environment (nvcc).

– Slow-down when $D > 100$.

Applications to Kriging, spline, Gaussian process, kernel regression

Solve a **kernel linear system:**

$$(\lambda\,\mathrm{Id} + K_{xx})\,a \;=\; b \qquad \text{i.e.} \qquad a \;\leftarrow\; (\lambda\,\mathrm{Id} + K_{xx})^{-1} b$$

where $\lambda \geqslant 0$ and $(K_{xx})_{i,j} = k(x_i, x_j)$ is a positive definite matrix.

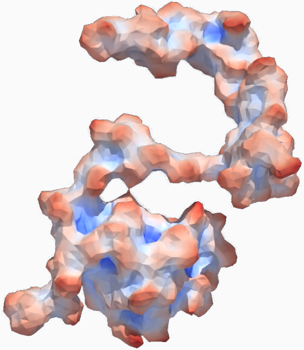
KeOps symbolic tensors:

- Can be fed to **standard solvers**: SciPy, GPytorch, etc.
- On the 3DRoad dataset ($N = 278k$, $D = 3$):

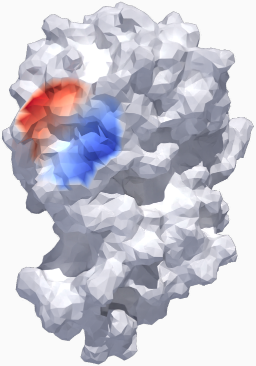
  7h with 8 GPUs $\rightarrow$ **15mn** with 1 GPU.

- Provide a **fast backend for research codes**: see e.g. *Kernel methods through the roof: handling **billions of points** efficiently*, by G. Meanti, L. Carratino, L. Rosasco, A. Rudi (2020).

Fast prototyping of geometry processing algorithms:



Mean curvature



Mesh convolution

Conclusion

The KeOps library provides:

- **Good performance** on geometric problems,
  with all the **convenient features** of a deep learning library.

- A first **stable release** last year; 23k downloads so far.

- The computational **foundations** of this thesis.

Computational optimal transport


Thibault Séjourné


F.-X. Vialard


Gabriel Peyré

We need robust loss functions for shape analysis

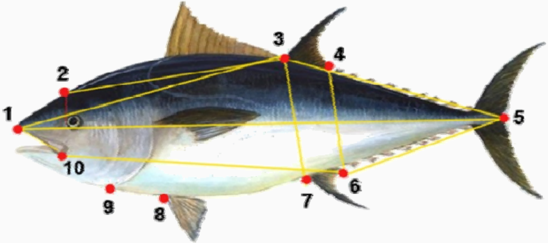
Working with point clouds is now **easier than ever**.
We can protoype new geometric algorithms in minutes.

But how should we **measure success** and **errors**?

$\Longrightarrow$ We must develop **geometric loss functions**
to compute distances between shapes.

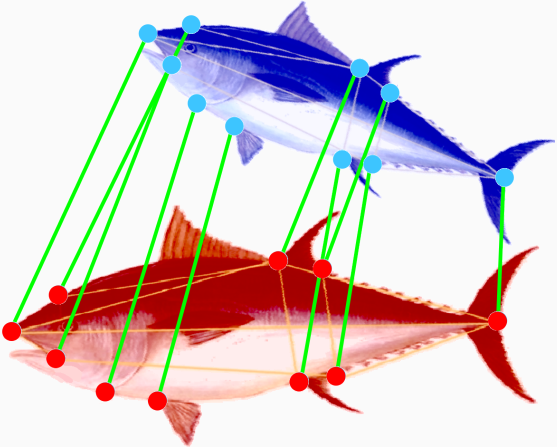
High-quality gradients will improve the **robustness**
of registration or training algorithms
and allow us to **focus on our models**.

Life is easy when you have landmarks...



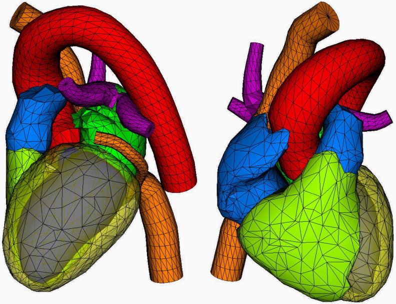
Anatomical landmarks from *A morphometric approach for the analysis of body shape in bluefin tuna*, Addis et al., 2009.

Life is easy when you have landmarks...

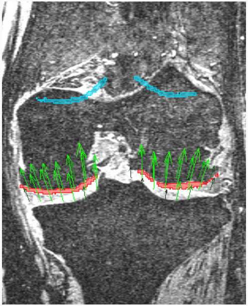


Anatomical landmarks from *A morphometric approach for the analysis of body shape in bluefin tuna*, Addis et al., 2009.

Unfortunately, medical data is often weakly labeled [EPW+11]



Surface meshes

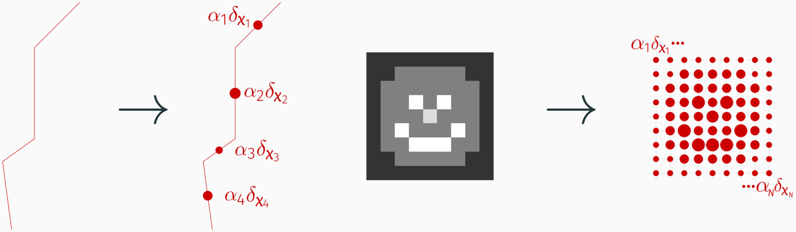


Segmentation masks

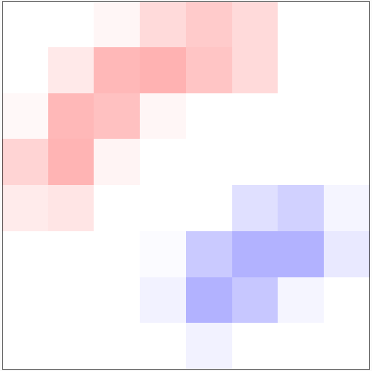
Encoding unlabeled shapes as measures

Let's enforce sampling invariance:

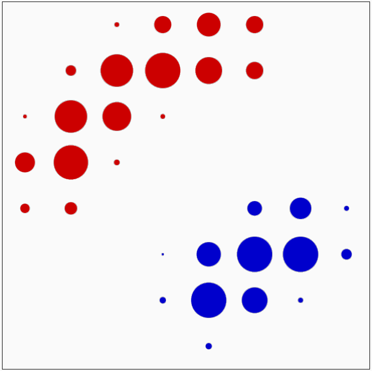
$$A \longrightarrow \alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}, \qquad B \longrightarrow \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}.$$



A baseline setting: density registration

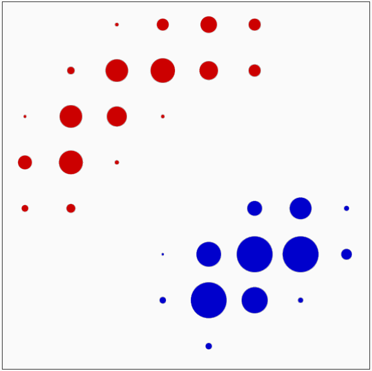


A baseline setting: density registration



$$\alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}\,, \quad \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}\,.$$

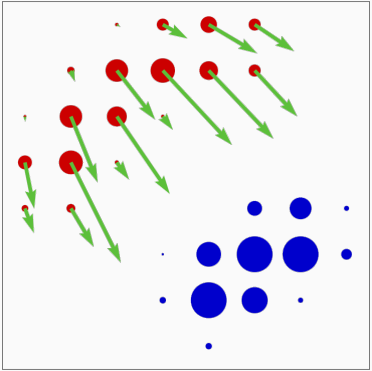
A baseline setting: density registration



$$\alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}\,, \quad \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}\,.$$

$$\sum_{i=1}^{N} \alpha_i = 1 = \sum_{j=1}^{M} \beta_j$$

A baseline setting: density registration

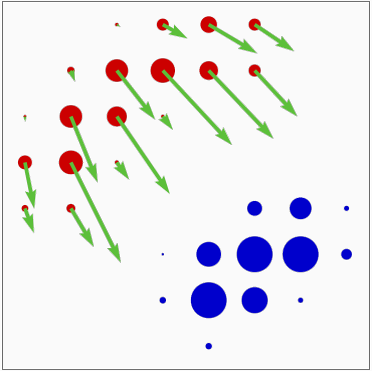


$$\alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}\,, \quad \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}\,.$$

$$\sum_{i=1}^{N} \alpha_i \;=\; 1 \;=\; \sum_{j=1}^{M} \beta_j$$

Display $v_i \;=\; -\frac{1}{\alpha_i} \nabla_{x_i} \text{Loss}(\alpha, \beta).$

A baseline setting: density registration



$$\alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i} , \quad \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j} .$$

$$\sum_{i=1}^{N} \alpha_i = 1 = \sum_{j=1}^{M} \beta_j$$

Display $v_i = -\frac{1}{\alpha_i} \nabla_{x_i} \text{Loss}(\alpha, \beta)$.

Seamless extensions to:

- $\sum_i \alpha_i \neq \sum_j \beta_j$, outliers [CPSV18],
- curves and surfaces [KCC17],
- variable weights α_i.

The Wasserstein distance

We need **clean gradients**, without artifacts.

The Wasserstein distance

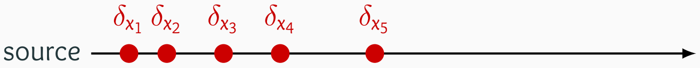
We need **clean gradients**, without artifacts.

Simple toy example in 1D :

The Wasserstein distance

We need **clean gradients**, without artifacts.

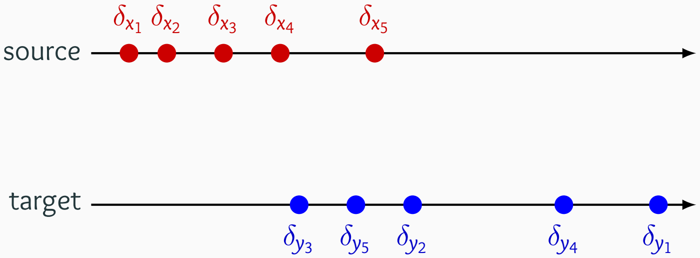
Simple toy example in 1D :



The Wasserstein distance

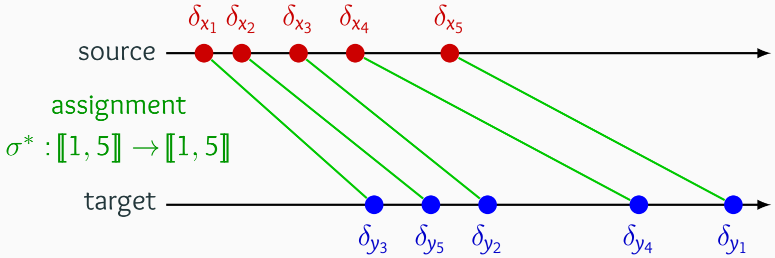
We need **clean gradients**, without artifacts.

Simple toy example in 1D :



We need **clean gradients**, without artifacts.

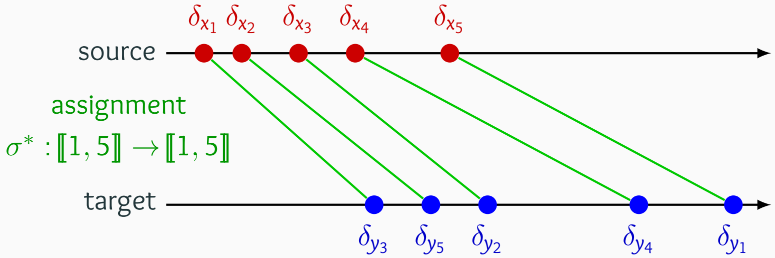
Simple toy example in 1D :



The Wasserstein distance

We need **clean gradients**, without artifacts.

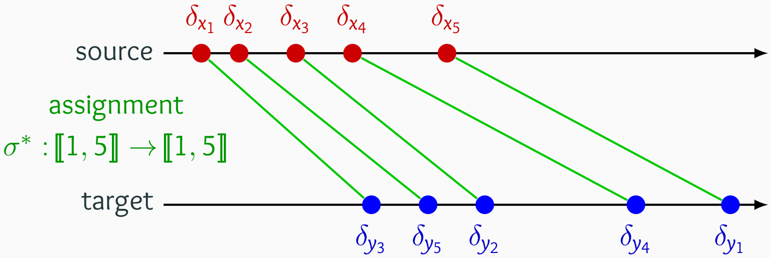
Simple toy example in 1D :



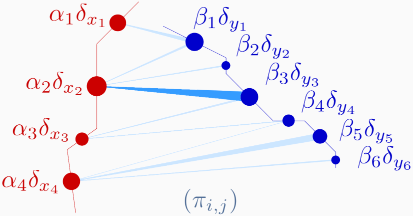
$$\mathrm{OT}(\alpha, \beta) = \frac{1}{2N} \sum_{i=1}^{N} |x_i - y_{\sigma^*(i)}|^2$$

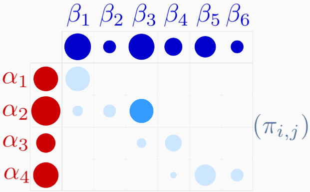
We need **clean gradients**, without artifacts.

Simple toy example in 1D :



$$\text{OT}(\alpha, \beta) \;=\; \frac{1}{2N}\sum_{i=1}^{N} \left|x_i - y_{\sigma^*(i)}\right|^2 \;=\; \min_{\sigma \in \mathcal{S}_N} \frac{1}{2N}\sum_{i=1}^{N} \left|x_i - y_{\sigma(i)}\right|^2$$





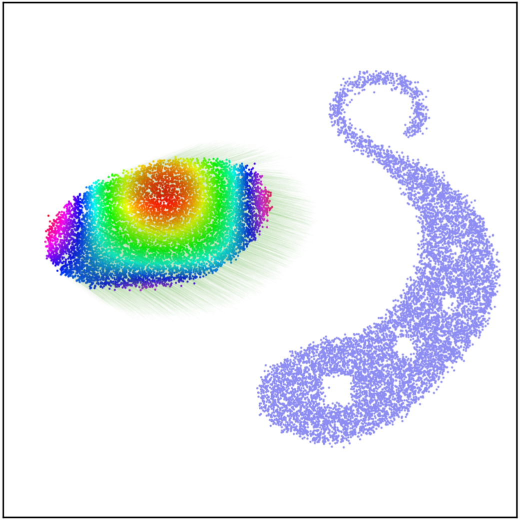
Minimize over N-by-M matrices (transport plans) π :

$$\text{OT}(\alpha, \beta) = \min_{\pi} \underbrace{\sum_{i,j} \pi_{i,j} \cdot \tfrac{1}{2}|x_i - y_j|^2}_{\text{transport cost}}$$

subject to $\pi_{i,j} \geqslant 0$,

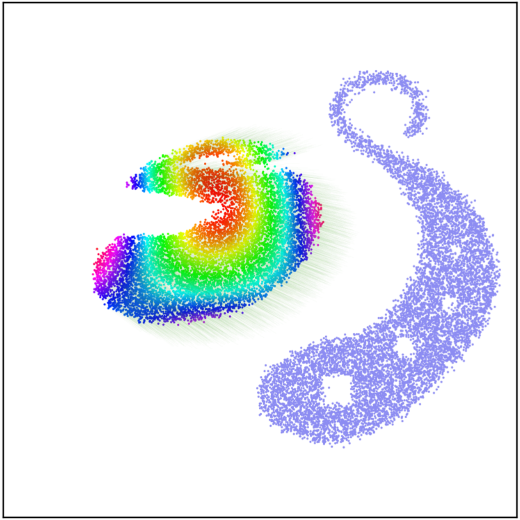
$$\sum_j \pi_{i,j} = \alpha_i, \quad \sum_i \pi_{i,j} = \beta_j.$$

Gradient flow as a toy registration: $x_i \leftarrow x_i - \delta t \frac{1}{\alpha_i} \nabla_{x_i} \mathrm{OT}(\alpha, \beta)$



$t = .00$

Gradient flow as a toy registration: $x_i \leftarrow x_i - \delta t \frac{1}{\alpha_i} \nabla_{x_i} \text{OT}(\alpha, \beta)$

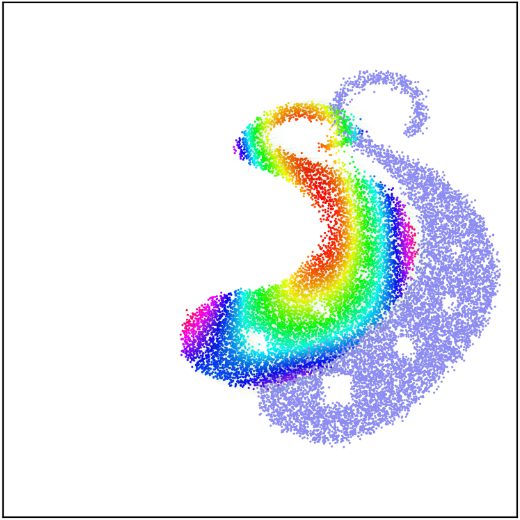


$t = .25$

Gradient flow as a toy registration: $x_i \leftarrow x_i - \delta t \frac{1}{\alpha_i} \nabla_{x_i} \mathrm{OT}(\alpha, \beta)$

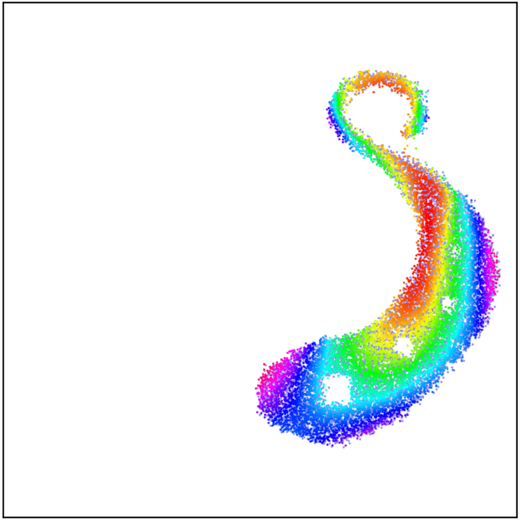
$t = .50$

Gradient flow as a toy registration: $x_i \leftarrow x_i - \delta t \frac{1}{\alpha_i} \nabla_{x_i} \text{OT}(\alpha, \beta)$



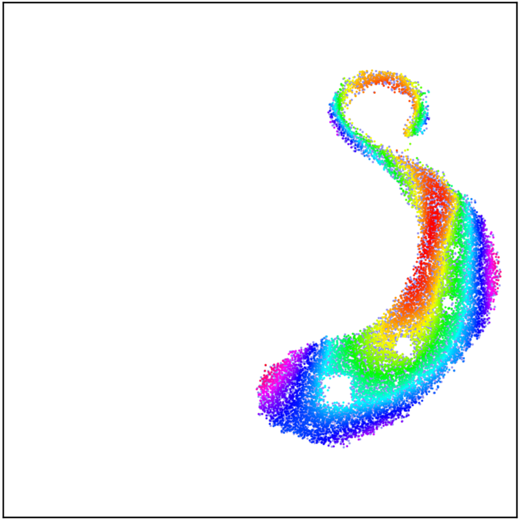
$t = 1.00$

Gradient flow as a toy registration: $x_i \leftarrow x_i - \delta t \frac{1}{\alpha_i} \nabla_{x_i} \text{OT}(\alpha, \beta)$



$t = 5.00$

Gradient flow as a toy registration: $x_i \leftarrow x_i - \delta t \frac{1}{\alpha_i} \nabla_{x_i} \mathrm{OT}(\alpha, \beta)$



$t = 10.00$

Key properties [Bre91]

The Wasserstein loss $\text{OT}(\alpha, \beta)$ is:

- **Symmetric:** $\text{OT}(\alpha, \beta) = \text{OT}(\beta, \alpha)$.
- **Positive:** $\text{OT}(\alpha, \beta) \geqslant 0$.
- **Definite:** $\text{OT}(\alpha, \beta) = 0 \iff \alpha = \beta$.
- **Translation-aware:** $\text{OT}(\alpha, \text{Translate}_{\vec{v}}(\alpha)) = \frac{1}{2}\|\vec{v}\|^2$.
- More generally, OT retrieves the unique **gradient of a convex function** $T = \nabla\varphi$ that maps α onto β :

$$\begin{aligned} &\text{In dimension 1,} && (x_i - x_j) \cdot (y_{\sigma(i)} - y_{\sigma(j)}) && \geqslant 0 \\ &\text{In dimension D,} && \langle\, x_i - x_j \,,\, T(x_i) - T(x_j) \,\rangle_{\mathbb{R}^D} && \geqslant 0\,. \end{aligned}$$

$\Longrightarrow$ Appealing generalization of an **increasing mapping.**

How should we solve the OT problem?

Key dates for discrete optimal transport with N points:

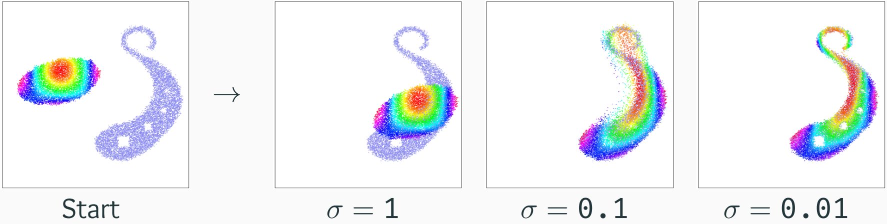
- [Kan42]: **Dual** problem.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + annealing, in $O(N^2)$.
- [GRL+98, CR00]: **Robust Point Matching** = Sinkhorn as a loss.
- [Cut13]: Start of the **GPU era**.
- [Mér11, Lév15,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

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

Key ingredient: the entropic blur

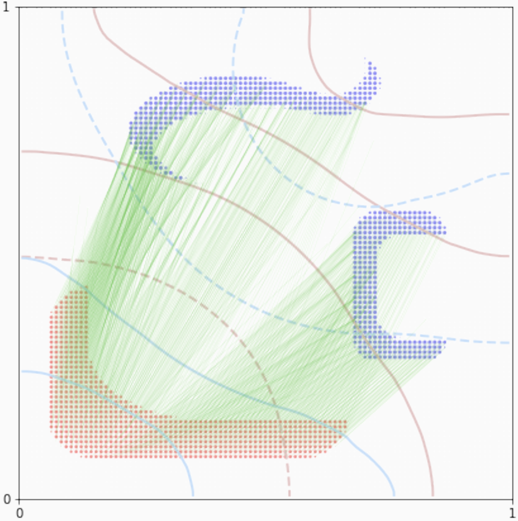
Sinkhorn divergence: with k_σ a Gaussian kernel of deviation σ,

$$\mathrm{S}_\sigma(\alpha, \beta) \;\simeq\; \mathrm{OT}(\, k_\sigma \star \alpha \,,\, k_\sigma \star \beta \,)\,.$$



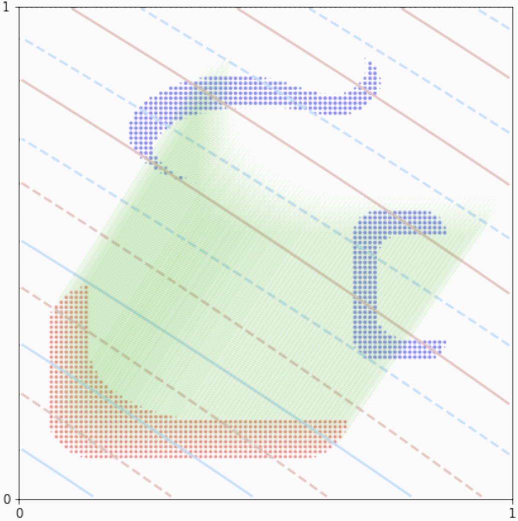
Theorem: If α and β have bounded support, then S_σ is suitable for gradient descent. It is symmetric, **positive**, definite, **convex** and metrizes the convergence in law.

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



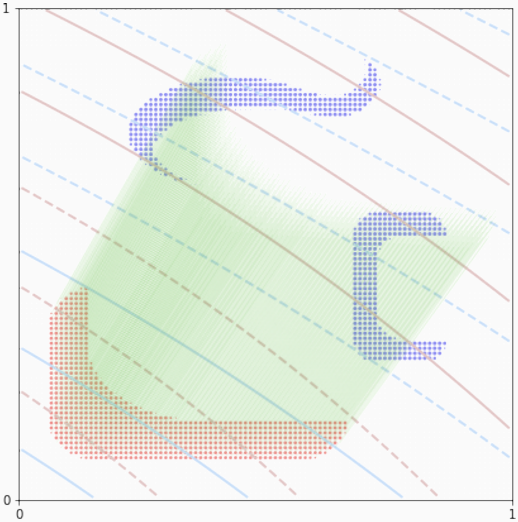
OT plan in 2D.

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



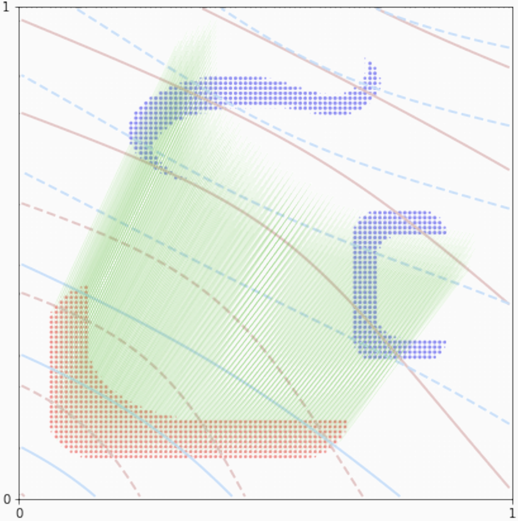
Iteration 0, blur $\sigma = 2^0$

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



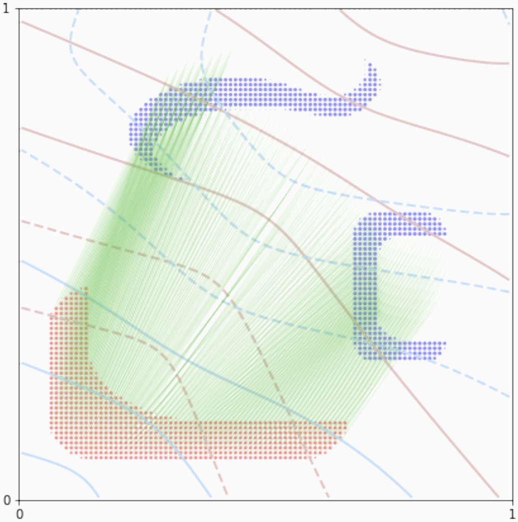
Iteration 1, blur $\sigma = 2^{-1}$

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



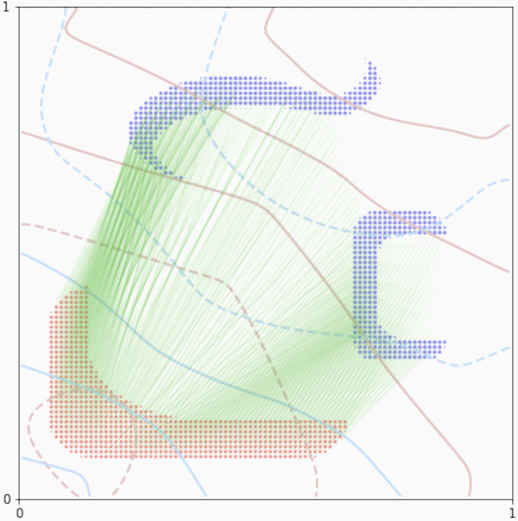
Iteration 2, blur $\sigma = 2^{-2}$

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



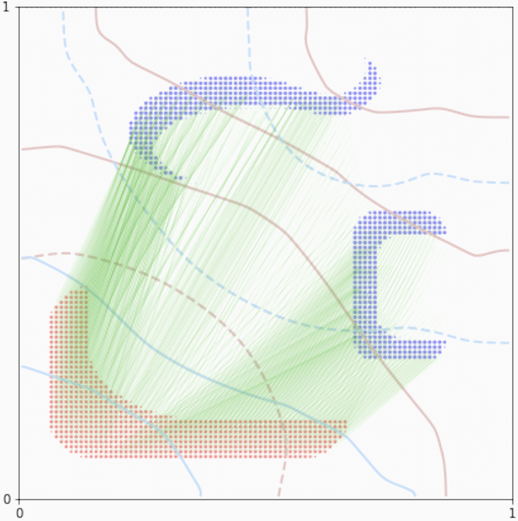
Iteration 3, blur $\sigma = 2^{-3}$

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



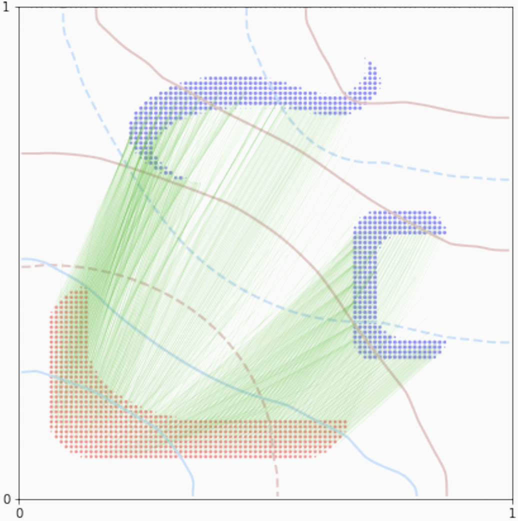
Iteration 4, blur $\sigma = 2^{-4}$

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



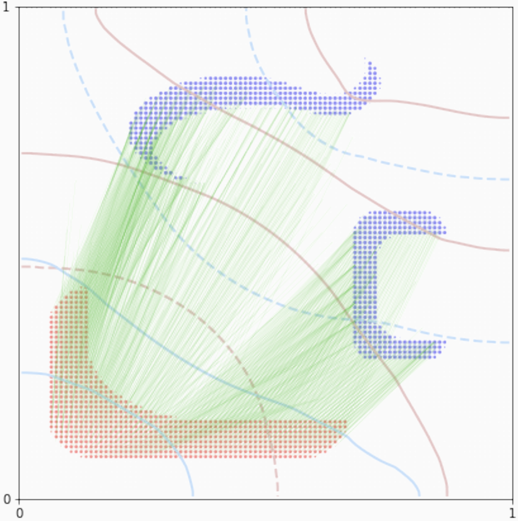
Iteration 5, blur $\sigma = 2^{-5}$

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



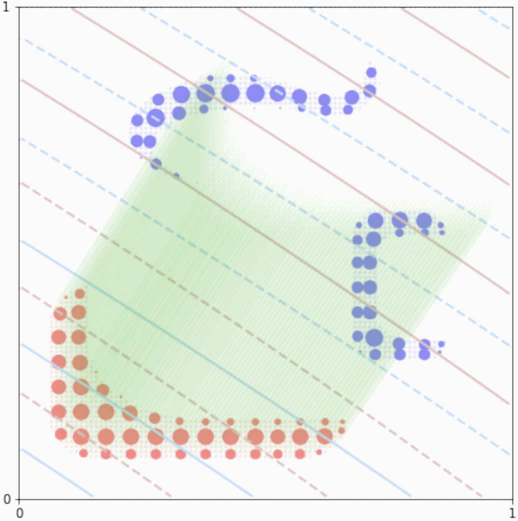
Iteration 6, blur $\sigma = 2^{-6}$

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



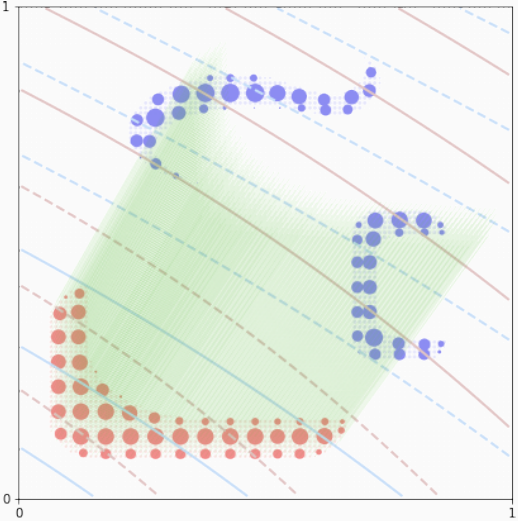
Iteration 7, blur σ = `.01`

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



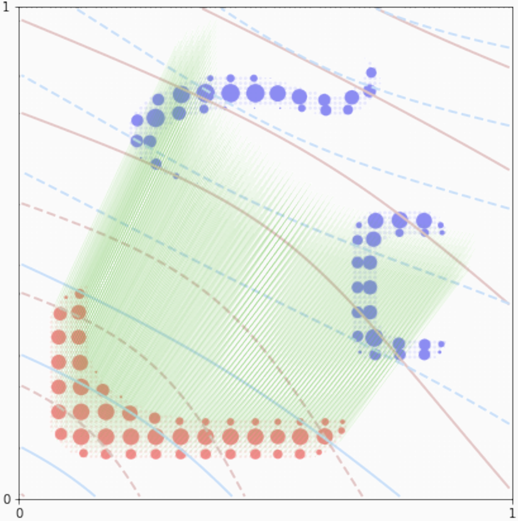
Iteration 0, blur $\sigma = 2^0$

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



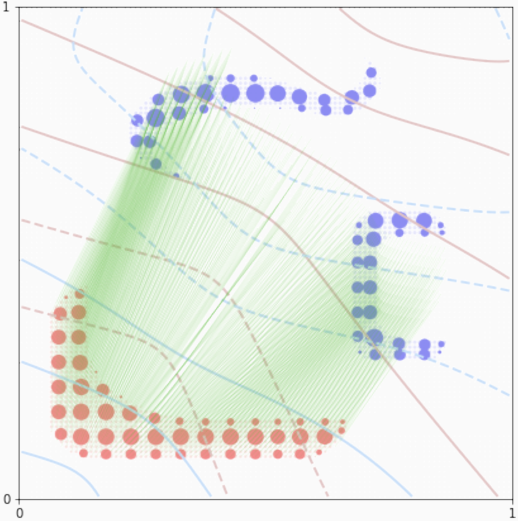
Iteration 1, blur $\sigma = 2^{-1}$

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



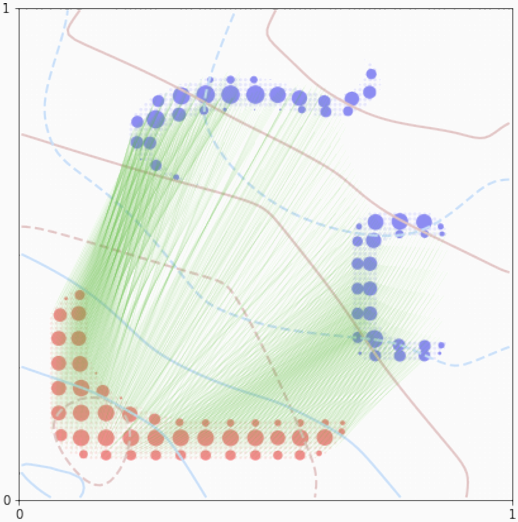
Iteration 2, blur $\sigma = 2^{-2}$

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



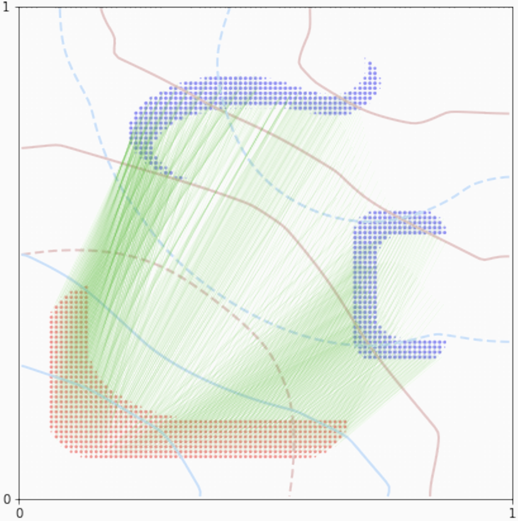
Iteration 3, blur $\sigma = 2^{-3}$

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



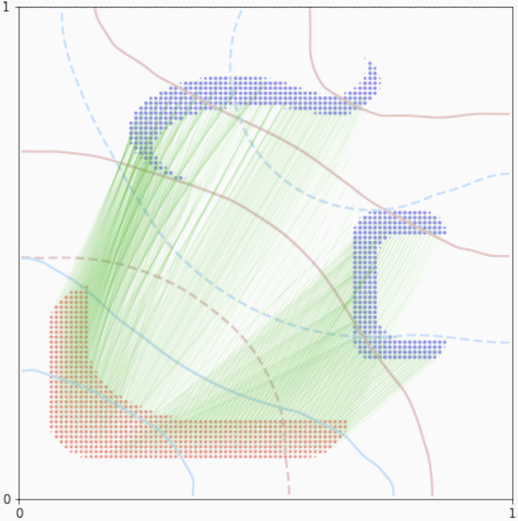
Iteration 4, blur $\sigma = 2^{-4}$

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



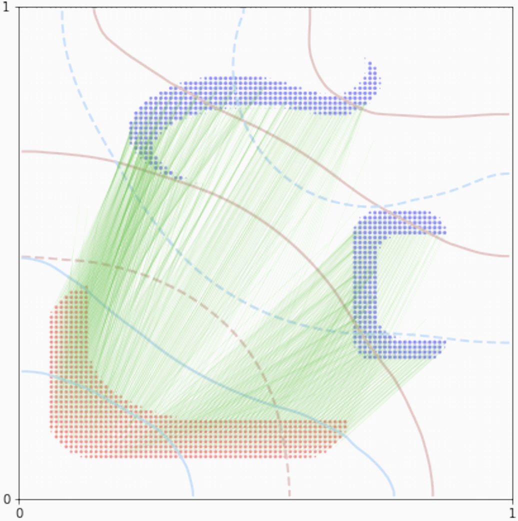
Iteration 5, blur $\sigma = 2^{-5}$

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



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

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



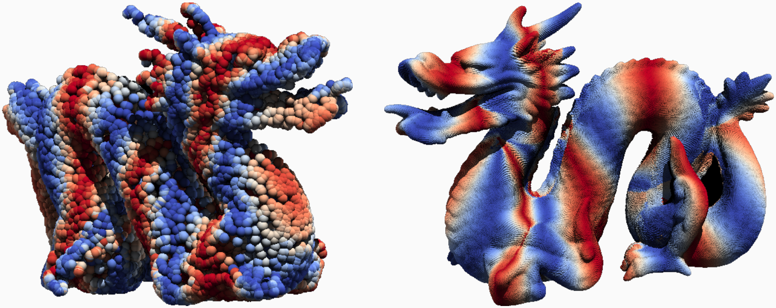
Iteration 7, blur σ = `.01`

Scaling up optimal transport to anatomical data

These progresses add up to a $\times 100$ - $\times 1000$ **acceleration:**

$$\text{Sinkhorn GPU} \xrightarrow{\times 10} + \text{KeOps} \xrightarrow{\times 10} + \text{Annealing} \xrightarrow{\times 10} + \text{Multiscale}$$

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



10k points in 30-50ms

100k points in 100-200ms

Geometric Loss functions for PyTorch

Our website: www.kernel-operations.io/geomloss

$\Longrightarrow$ pip install geomloss $\Longleftarrow$

```
# Large point clouds in [0, 1]^3
import torch
x = torch.rand(100000, 3, requires_grad=True).cuda()
y = torch.rand(200000, 3).cuda()

# Define a Wasserstein loss between sampled measures
from geomloss import SamplesLoss
loss = SamplesLoss(loss="sinkhorn", p=2, blur=.05)

L = loss(x, y)  # By default, use constant weights
# GeomLoss supports autograd, batch processing, etc.
g_x, = torch.autograd.grad(L, [x])
```

Overview of the last two sections

Geometry processing:

+ KeOps provides support for **distance-like matrices**.
+ It **relieves us** from C++/CUDA programming.

Overview of the last two sections

Geometry processing:

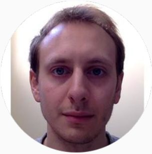
+ KeOps provides support for **distance-like matrices.**
+ It **relieves us** from C++/CUDA programming.

Computational optimal transport:

+ Significant **progress** over the last decade.
+ Efficient solvers are being **packaged** for the global community:
  GeomLoss, SD-OT, Geogram, etc.
− Some **challenging settings** remain wide open:
  high-dimensional spaces, graphs, etc.

+ The problem is essentially **solved** in three “simple” settings:
  imaging, **3D geometry**, fluid mechanics.

New paths for computational anatomy

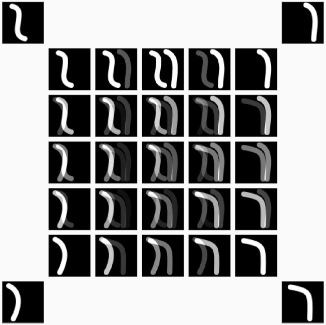


Pierre Roussillon

Pietro Gori

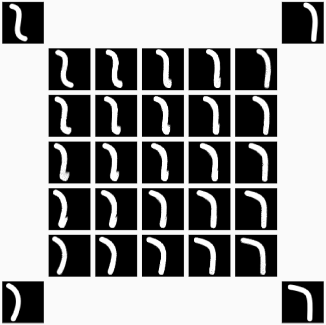
Affordable geometric interpolation [AC11]

$$\text{Barycenter } \alpha^* = \arg\min_{\alpha} \sum_{i=1}^{N} \lambda_i \, \text{Loss}(\alpha, \beta_i) .$$



Linear barycenters

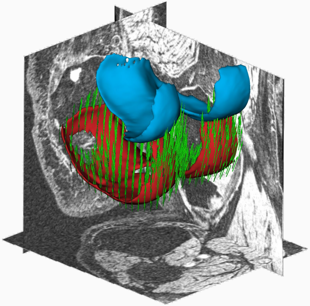
$\text{Loss}(\alpha, \beta) = \|\alpha - \beta\|_{L^2}^2$



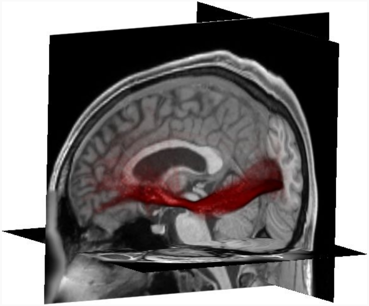
Wasserstein barycenters

$\text{Loss}(\alpha, \beta) = \text{OT}(\alpha, \beta)$

Applications to medical imaging

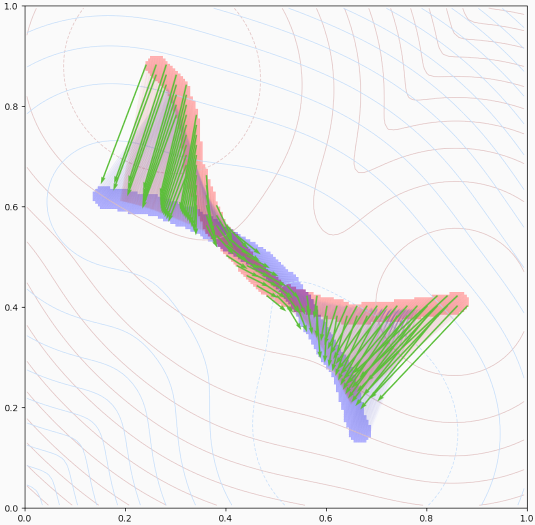


Knee caps



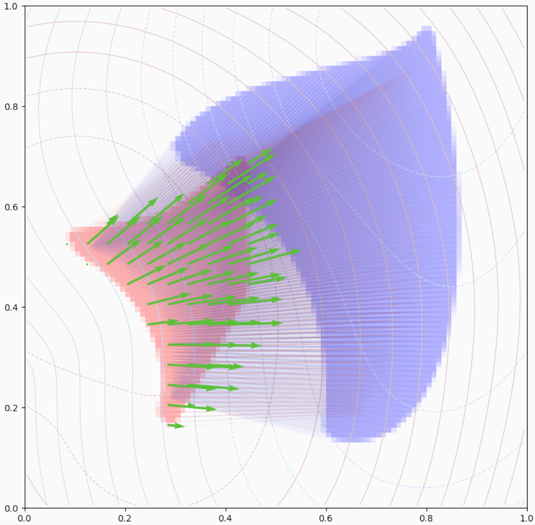
White matter bundles

A global and geometric loss function



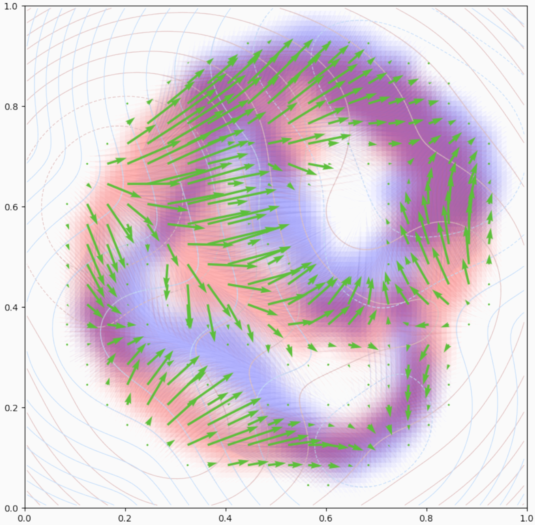
A high-quality gradient...

A global and geometric loss function



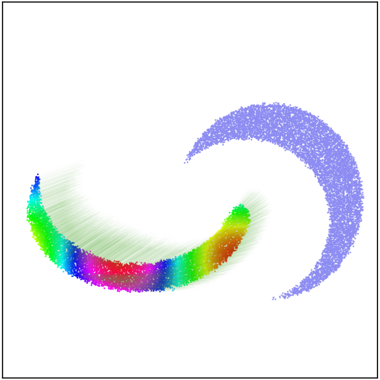
A high-quality gradient...

A global and geometric loss function

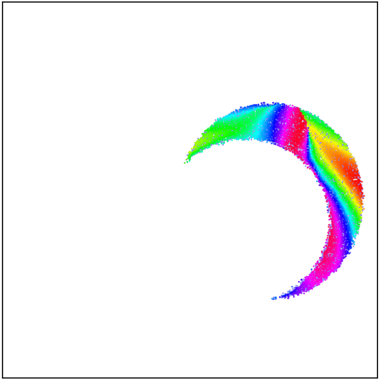


A high-quality gradient... But no preservation of topology!

Optimal transport = cheap'n easy registration? Beware!



Before



After

Topology-aware shape models

Optimal Transport = **independent** particles + mass preservation.
We need stronger metrics.

Topology-aware models are often related to physics:
elastic materials, **fluid** mechanics, etc.

We now have access to **large datasets**, **reliable segmentations**
and efficient **feature detectors**.

Can we plug them into our models?

We are reaching the **limits** of what can be done
with existing **Matlab/C++** codebases.

Since 2017, a new development paradigm

Toolboxes for computational anatomy are becoming increasingly:

+ **Efficient**, with GPU backends.
+ **Differentiable**, to fit in neural pipelines.
+ **Modular** and un-opinionated: freedom!
+ **Easy-to-use** by newcomers.

Since 2017, a new development paradigm

Toolboxes for computational anatomy are becoming increasingly:

+ **Efficient**, with GPU backends.
+ **Differentiable**, to fit in neural pipelines.
+ **Modular** and un-opinionated: freedom!
+ **Easy-to-use** by newcomers.

KeOps and **GeomLoss** fit within this ecosystem
and support e.g. the **Deformetrica** software.

We look forward to finally **using** them!
(See Chapter 5 for examples of shape models.)

Conclusion

Key points

- **Symbolic matrices** are key to performance:
  $\longrightarrow$ KeOps, x30 speed-up vs. PyTorch and TF.

- Optimal Transport = **generalized sorting:**
  $\longrightarrow$ Geometric gradients.
  $\longrightarrow$ Super-fast $O(N \log N)$ solvers.

- Going forward, we must develop
  data-driven, efficient yet robust shape models.

Genuine team work


Alain Trouvé


Thibault Séjourné


F.-X. Vialard


Gabriel Peyré

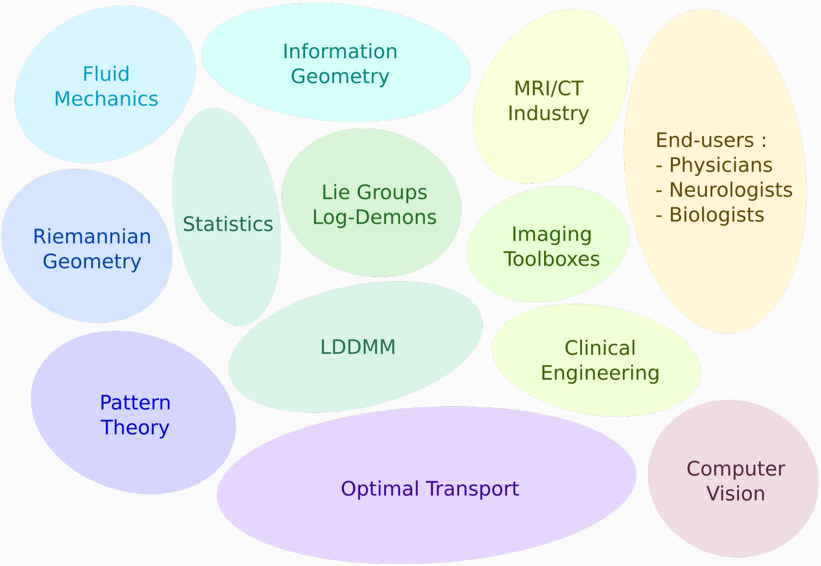

Benjamin Charlier


Joan Glaunès

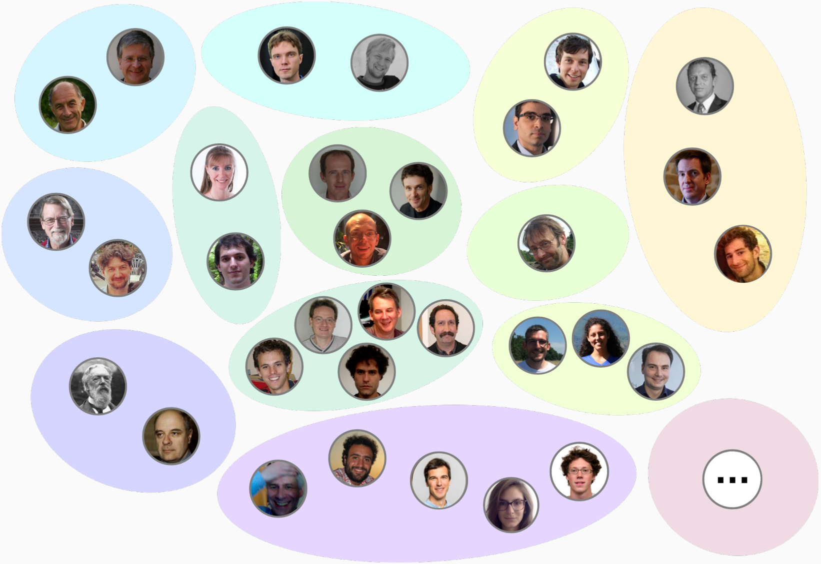

Pierre Roussillon


Pietro Gori

Promoting cross-field interactions



Promoting cross-field interactions



Reaching out to students and engineers is a priority

Online documentation:

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

PhD thesis, written as an introduction to the field:

`www.jeanfeydy.com/geometric_data_analysis.pdf`

Thank you for your attention.

Any questions?

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.

Y. Brenier.
Polar factorization and monotone rearrangement of vector-valued functions.
Comm. Pure Appl. Math., 44(4):375–417, 1991.

Brian Curless and Marc Levoy.
A volumetric method for building complex models from range images.
In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996.

Christophe Chnafa, Simon Mendez, and Franck Nicoud.
Image-based large-eddy simulation in a realistic left heart.
Computers & Fluids, 94:173–187, 2014.

Lénaïc Chizat, Gabriel Peyré, Bernhard Schmitzer, and François-Xavier Vialard.
Unbalanced optimal transport: Dynamic and kantorovich formulations.
Journal of Functional Analysis, 274(11):3090–3123, 2018.

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.

- Adam Conner-Simons and Rachel Gordon.
  Using ai to predict breast cancer and personalize care.
  http://news.mit.edu/2019/
  using-ai-predict-breast-cancer-and-personalize-c
  2019.
  MIT CSAIL.
- Marco Cuturi.
  Sinkhorn distances: Lightspeed computation of optimal transport.
  In *Advances in Neural Information Processing Systems*, pages 2292–2300, 2013.

References v

Olivier Ecabert, Jochen Peters, Matthew J Walker, Thomas Ivanc, Cristian Lorenz, Jens von Berg, Jonathan Lessick, Mani Vembar, and Jürgen Weese.
Segmentation of the heart and great vessels in CT images using a model-based adaptation framework.
Medical image analysis, 15(6):863–876, 2011.

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.

Irene Kaltenmark, Benjamin Charlier, and Nicolas Charon.
A general framework for curve and surface comparison and registration with oriented varifolds.
In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

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.
- Bruno Lévy.
  A numerical algorithm for l2 semi-discrete optimal transport in 3d.
  ESAIM: Mathematical Modelling and Numerical Analysis, 49(6):1693–1715, 2015.

Christian Ledig, Andreas Schuh, Ricardo Guerrero, Rolf A Heckemann, and Daniel Rueckert.
Structural brain imaging in Alzheimer's disease and mild cognitive impairment: biomarker analysis and shared morphometry database.
Scientific reports, 8(1):11258, 2018.

Stéphane Mallat.
Understanding deep convolutional networks.
Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 374(2065):20150203, 2016.

Quentin Mérigot.
A multiscale approach to optimal transport.
In *Computer Graphics Forum*, volume 30, pages 1583–1592. Wiley Online Library, 2011.

Yaroslav Nikulin and Roman Novak.
Exploring the neural algorithm of artistic style.
arXiv preprint arXiv:1602.07188, 2016.

Moses Olafenwa.
Object detection with 10 lines of code.
https://towardsdatascience.com/
object-detection-with-10-lines-of-code-d6cb4d86f
2018.

Towards Data Science.

Maurice Peemen, Bart Mesman, and Henk Corporaal.
Speed sign detection and recognition by convolutional neural networks.
In *Proceedings of the 8th international automotive congress*, pages 162–170. sn, 2011.

Ptrump16.
Irm picture.
https://commons.wikimedia.org/w/index.php?curid=64157788, 2019.
CC BY-SA 4.0.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox.
U-net: Convolutional networks for biomedical image segmentation.
In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Bernhard Schmitzer.
Stabilized sparse scaling algorithms for entropy regularized transport problems.
SIAM Journal on Scientific Computing, 41(3):A1443–A1481, 2019.

Donglai Wei, Bolei Zhou, Antonio Torralba, and William T Freeman.
mNeuron: A Matlab plugin to visualize neurons from deep models.
Massachusetts Institute of Technology, 2017.