Geometric data analysis, beyond convolutions

Jean Feydy

King's College London – December 2019

Imperial College London
Collaboration with B. Charlier, J. Glaunès (KeOps library);
F.-X. Vialard, G. Peyré, T. Séjourné, A. Trouvé (OT theory).

Who am I?

Quick CV:

- Typical French curriculum (ENS Maths + MVA + Siemens)

Who am I?

Quick CV:

- Typical French curriculum (ENS Maths + MVA + Siemens)
- 2016-2019: PhD with Alain Trouvé (ENS Paris-Saclay)

Who am I?

Quick CV:

- Typical French curriculum (ENS Maths + MVA + Siemens)
- 2016-2019: PhD with Alain Trouvé (ENS Paris-Saclay)
- 2019-2022: PostDoc at Imperial, Dept. of Computing

Who am I?

Quick CV:

- Typical French curriculum (ENS Maths + MVA + Siemens)
- 2016-2019: PhD with Alain Trouvé (ENS Paris-Saclay)
- 2019-2022: PostDoc at Imperial, Dept. of Computing

Who am I?

Quick CV:

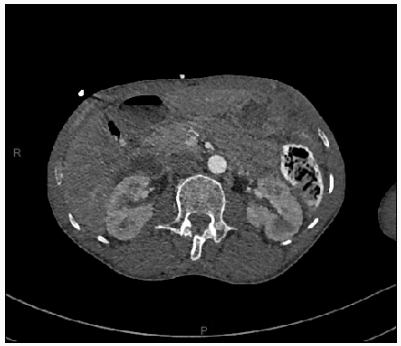
- Typical French curriculum (ENS Maths + MVA + Siemens)
- 2016-2019: PhD with Alain Trouvé (ENS Paris-Saclay)
- 2019-2022: PostDoc at Imperial, Dept. of Computing

$\Longrightarrow$ Looking for contacts in London!

The medical imaging pipeline [Ptr19, EPW+11]

Sensor data

The medical imaging pipeline [Ptr19, EPW+11]

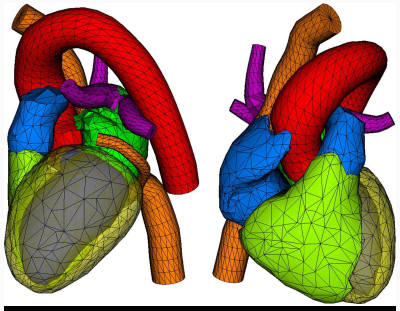


Raw image

Sensor data

Signal processing

The medical imaging pipeline [Ptr19, EPW+11]



High-level description

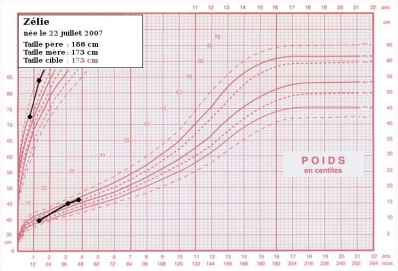
Raw image

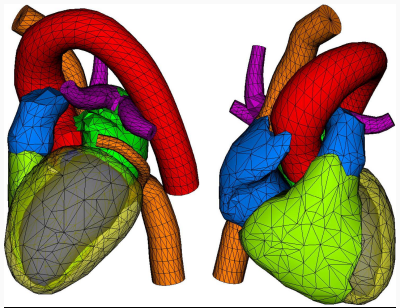
Sensor data

Computational anatomy

Signal processing

The medical imaging pipeline [Ptr19, EPW+11]





Valuable information

Statistics

High-level description

Computational anatomy

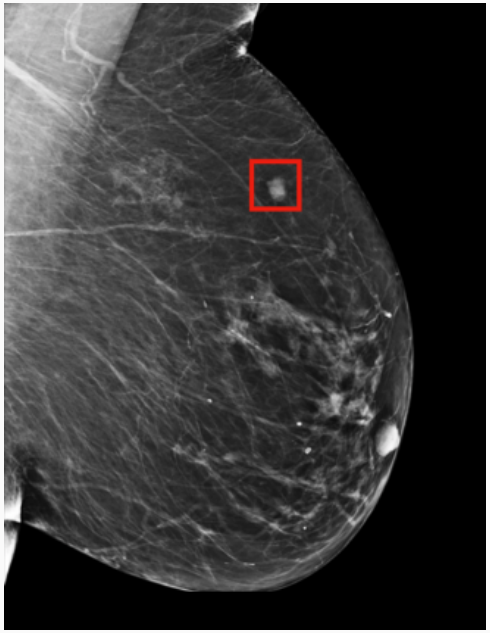
Raw image

Signal processing

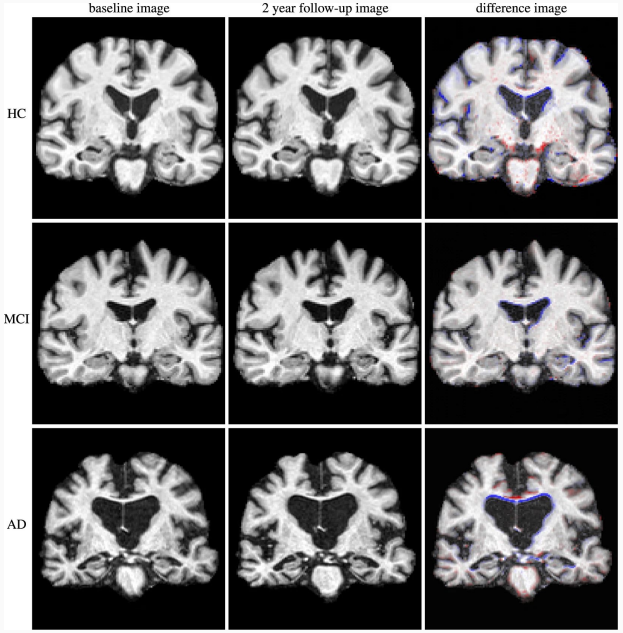
Sensor data

Computational anatomy [CSG19, LSG+18, CMN14]

Three main problems:



Spot patterns

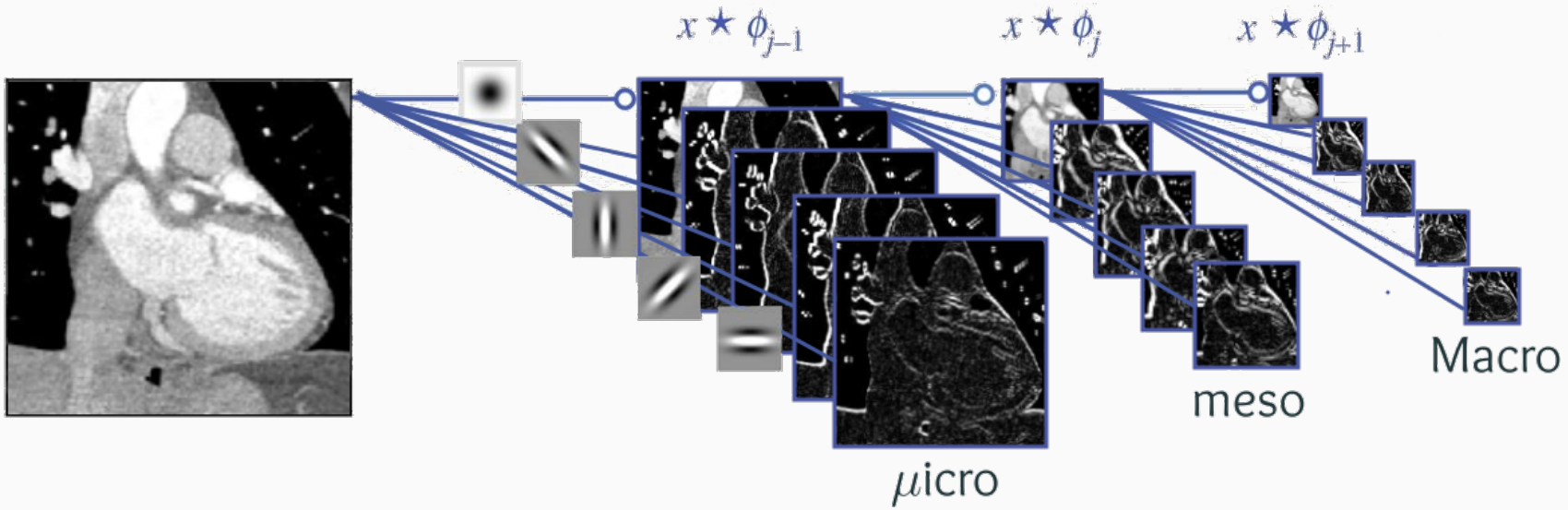


Analyze variations

Fit models

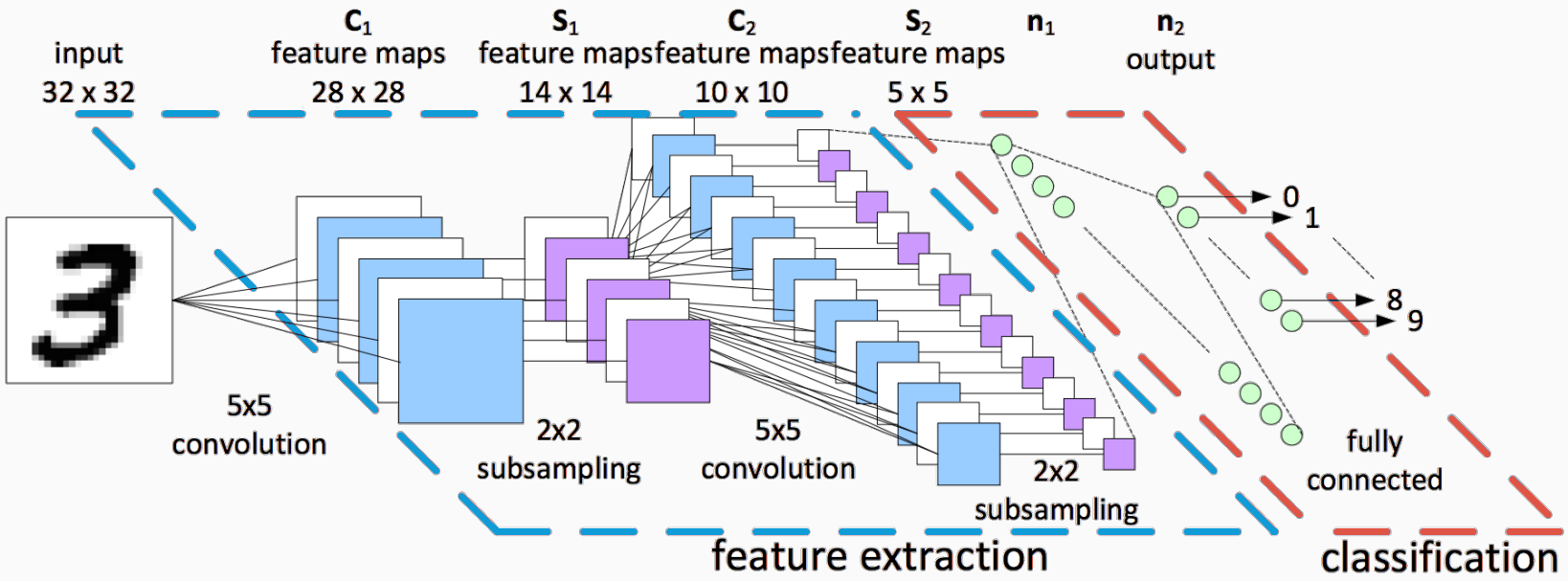
2010-2020: the deep learning revolution [Mal16, PMC11, WZTF17]

Explicit wavelet transform



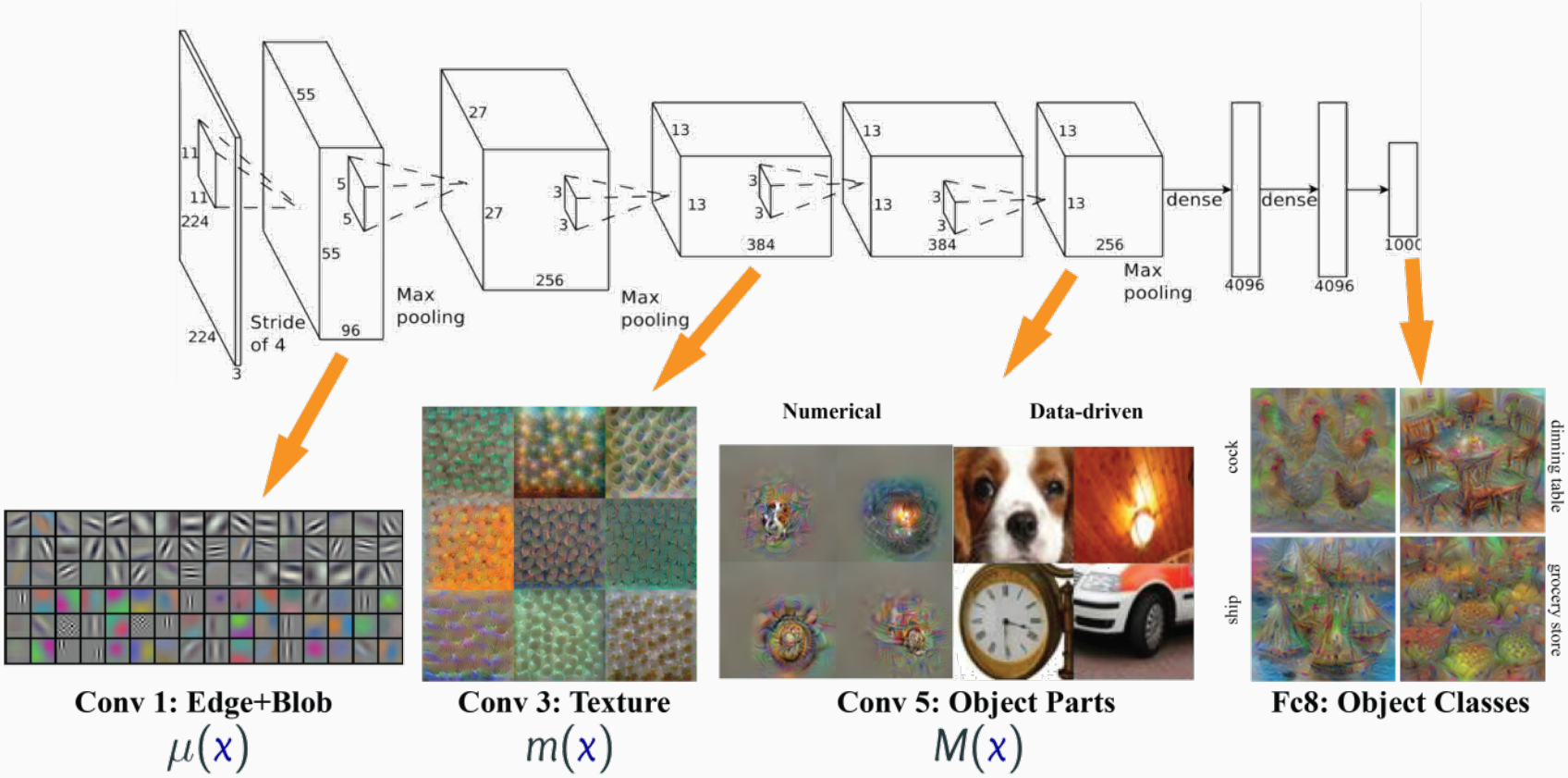
2010-2020: the deep learning revolution [Mal16, PMC11, WZTF17]

Explicit wavelet transform $\longrightarrow$ **data-driven** CNN



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

Explicit wavelet transform $\longrightarrow$ **data-driven** CNN $\longrightarrow$ **deep** CNN



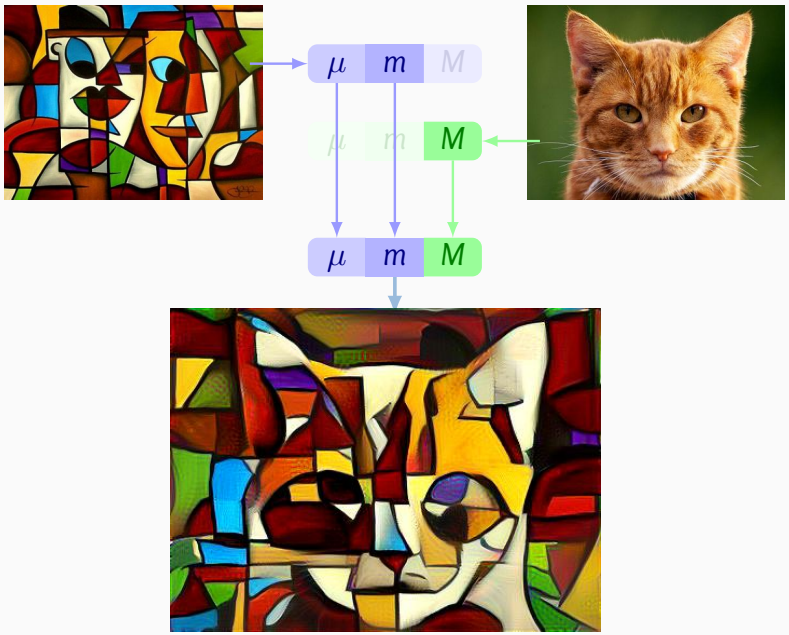
Conv 1: Edge+Blob $\mu(x)$

Conv 3: Texture $m(x)$

Conv 5: Object Parts $M(x)$

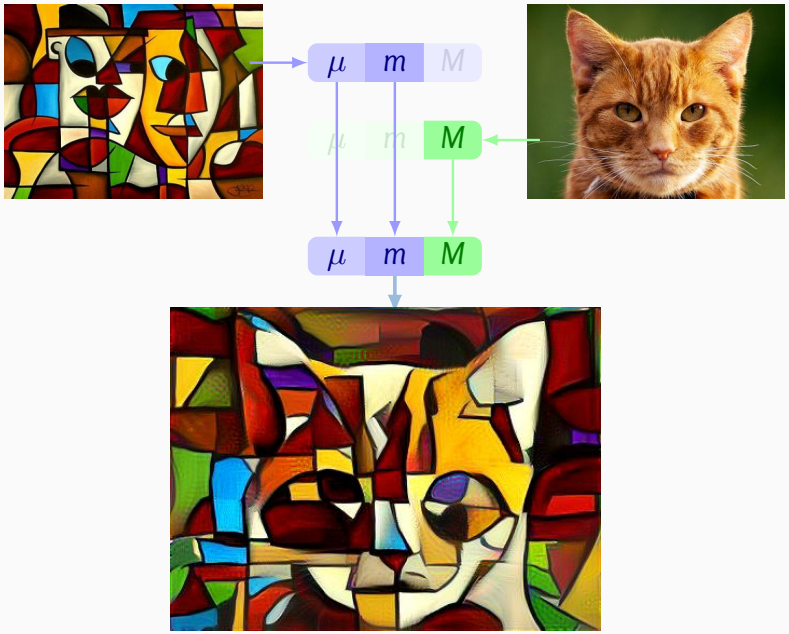
Fc8: Object Classes

(Wavelets $\rightarrow$ CNNs) = improvement for... [NN16, Ola18]

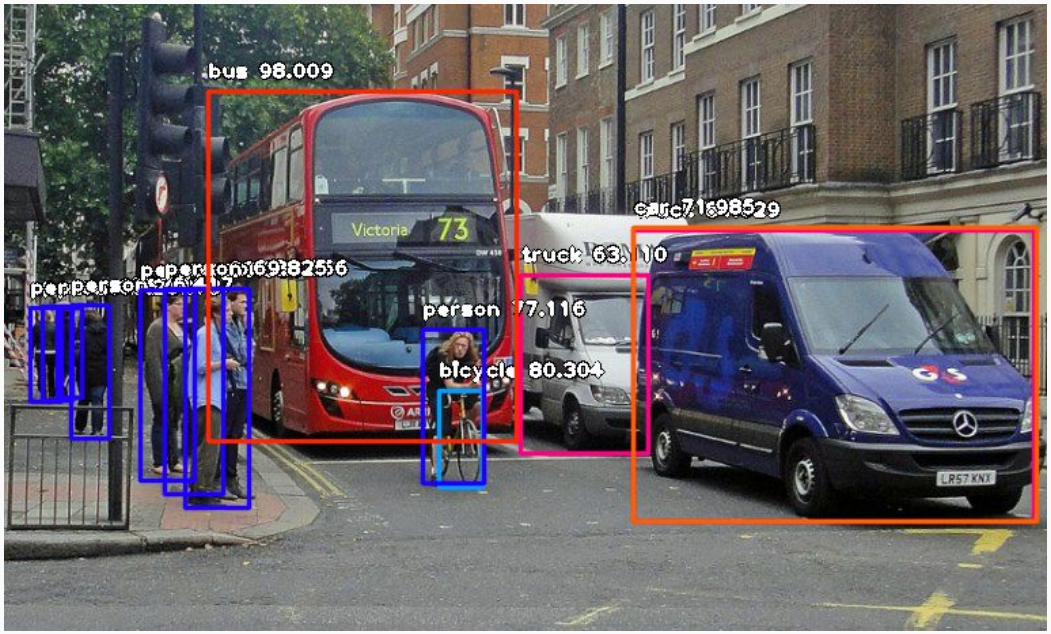


Texture processing

(Wavelets $\rightarrow$ CNNs) = improvement for... [NN16, Ola18]

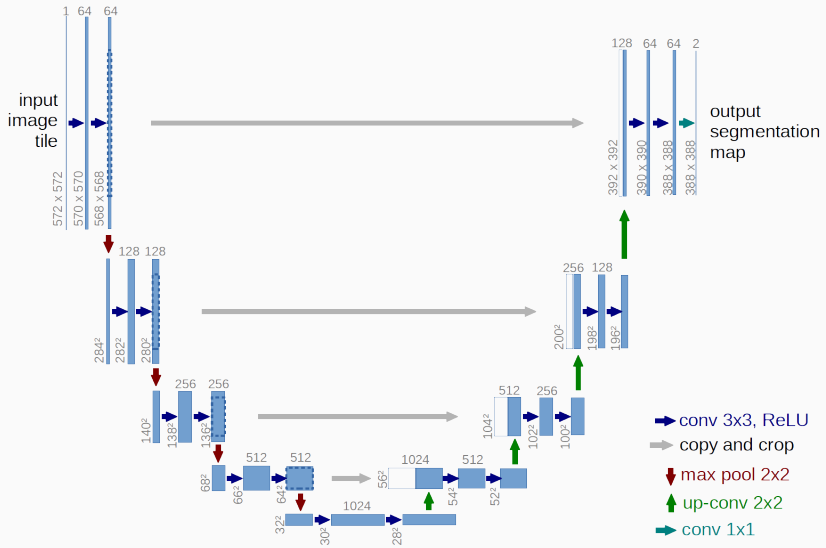


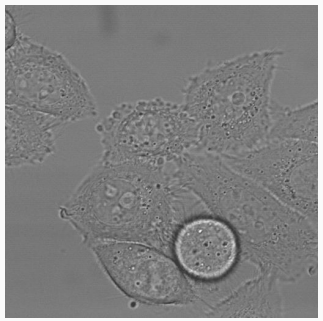
Texture processing

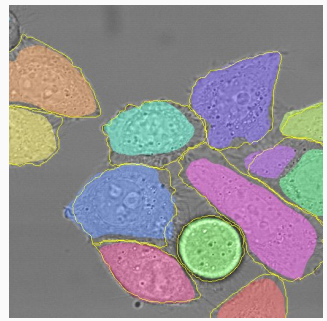


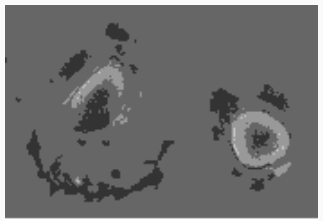
Feature detection

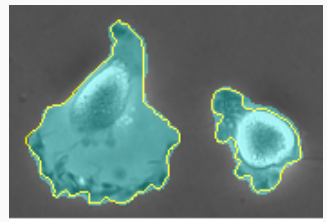
Segmentation with U-nets [RFB15]









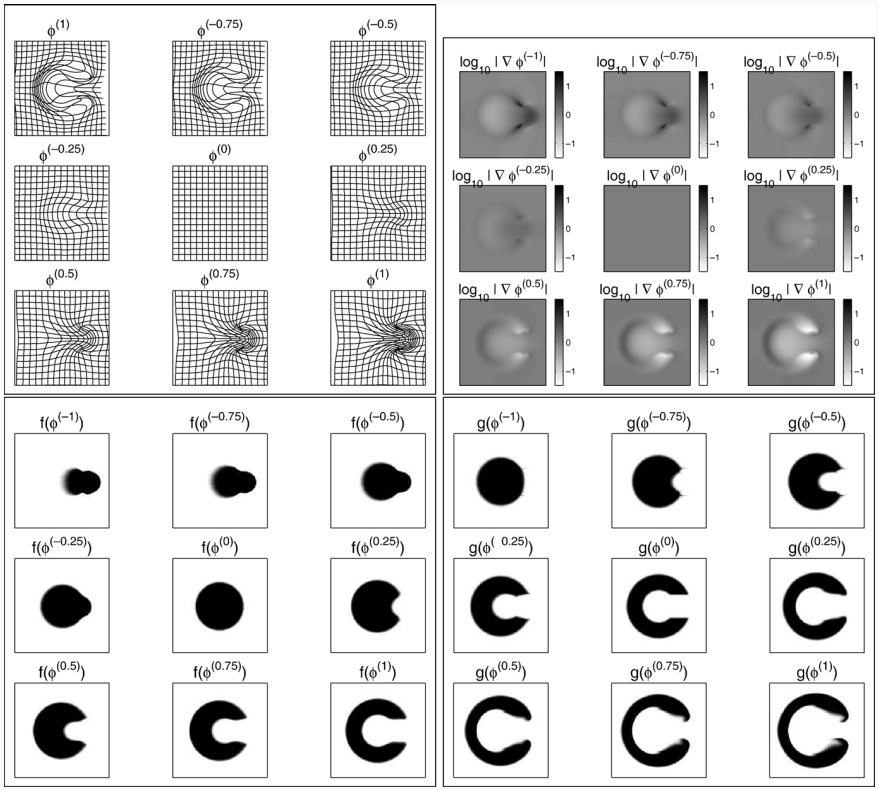


Architecture

Input

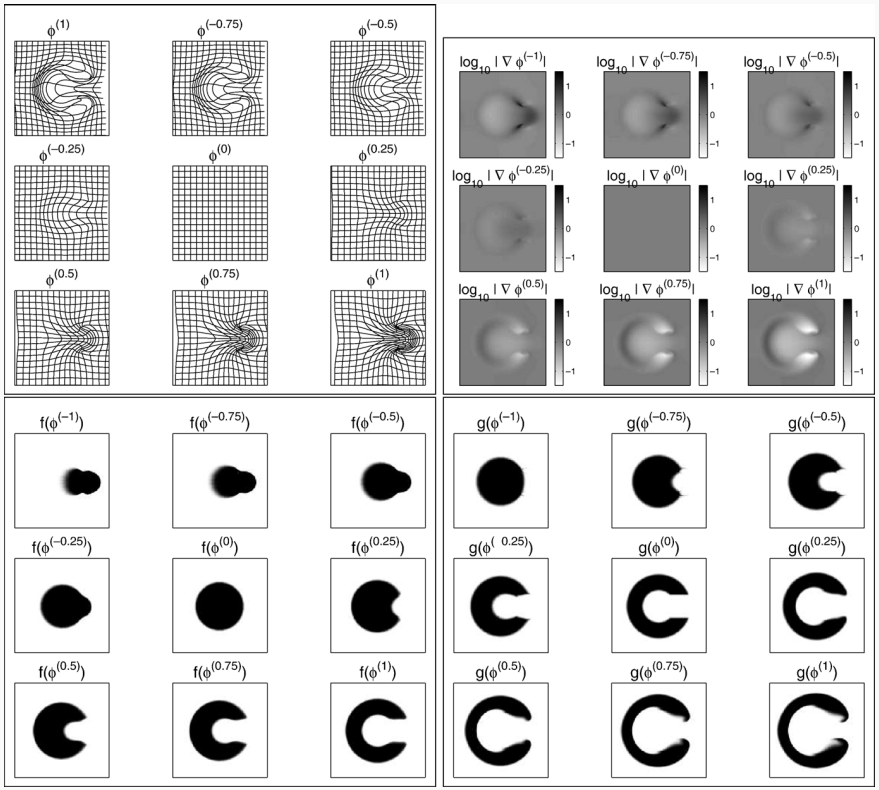
Output

Limitations of CNNs [Ash07, Gla05]



Advection $\neq$ Convolution

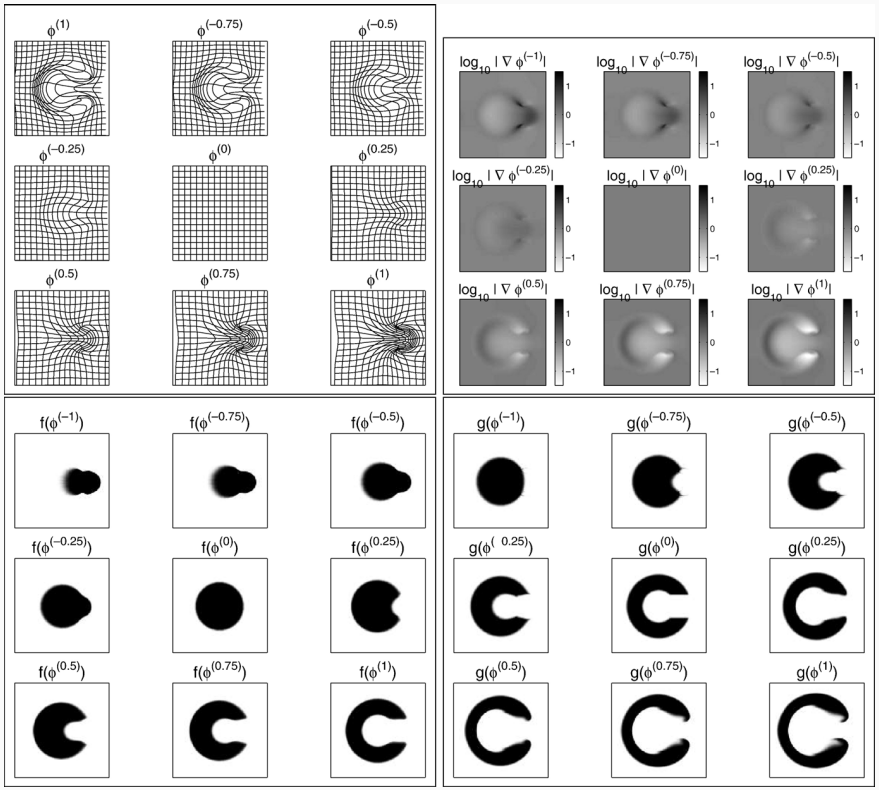
Limitations of CNNs [Ash07, Gla05]



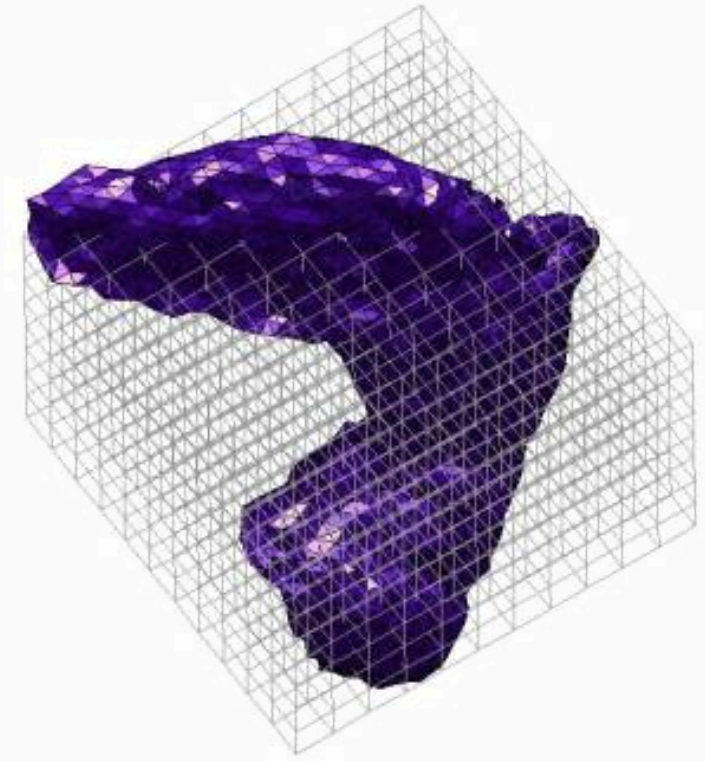
Advection ≠ Convolution

Mesh deformation

Limitations of CNNs [Ash07, Gla05]

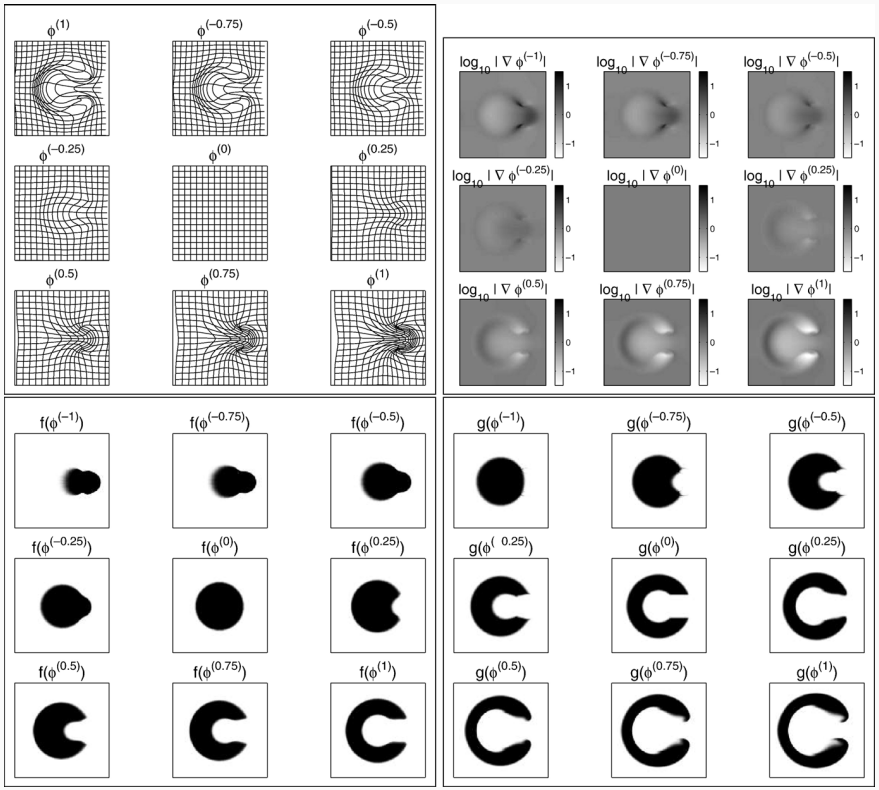


Advection ≠ Convolution

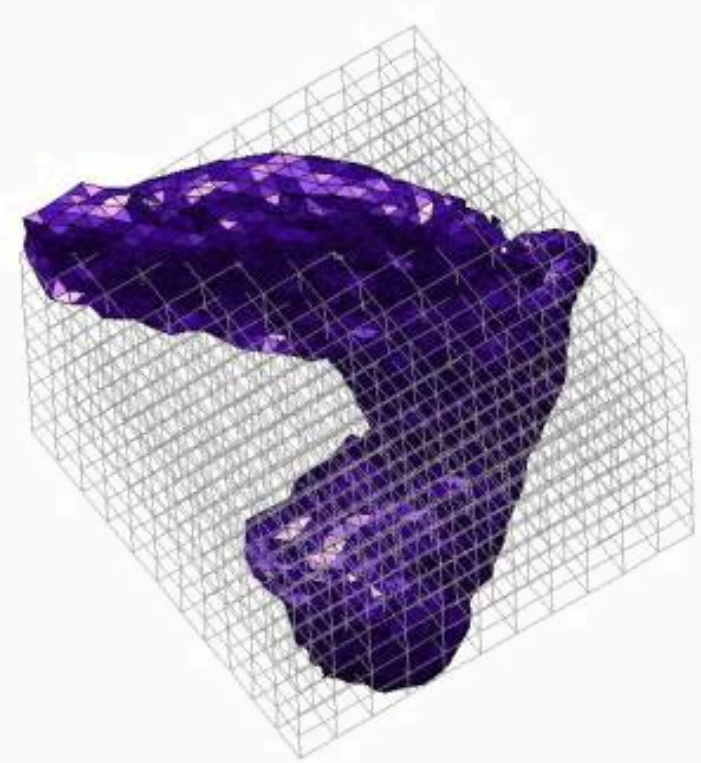


Mesh deformation

Limitations of CNNs [Ash07, Gla05]

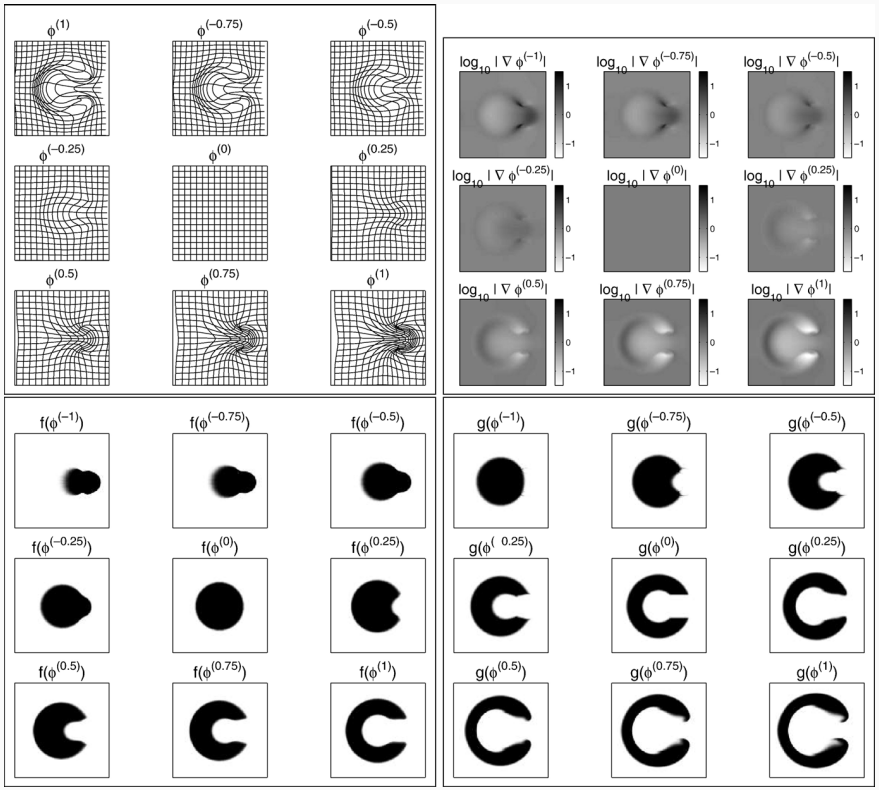


Advection $\neq$ Convolution

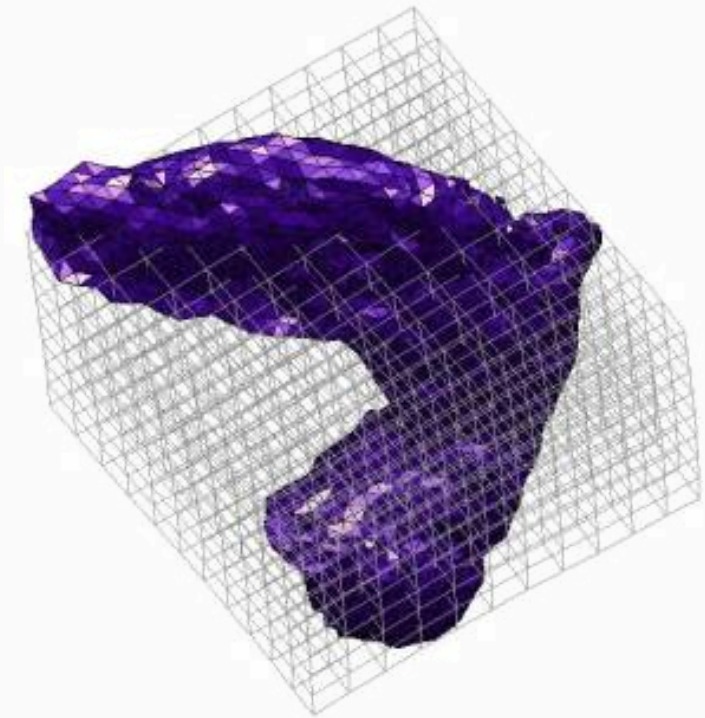


Mesh deformation

Limitations of CNNs [Ash07, Gla05]

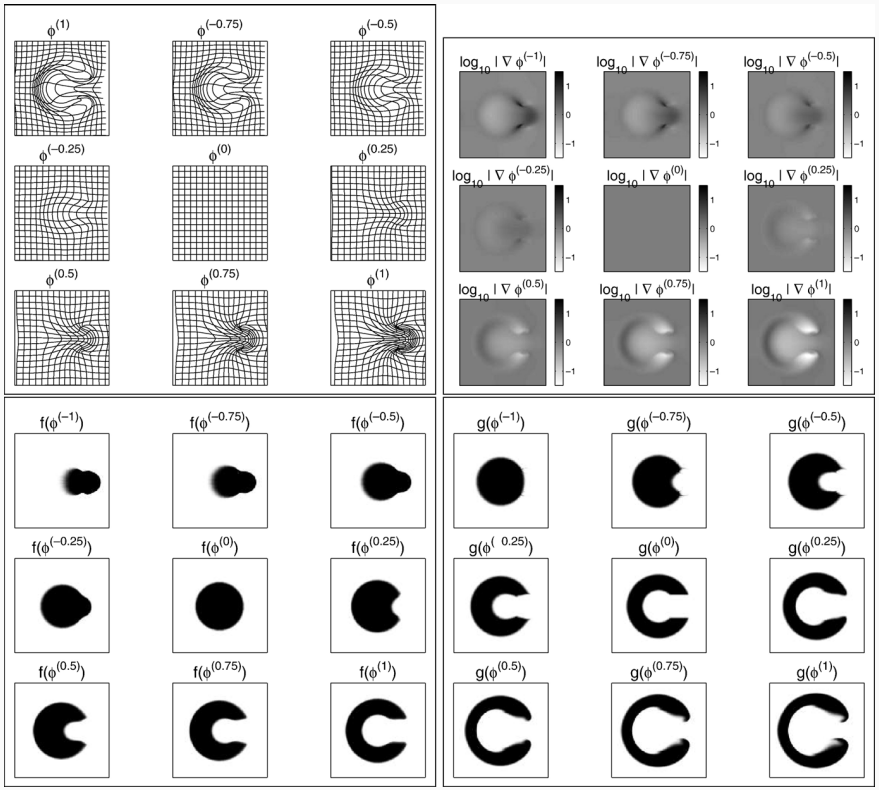


Advection ≠ Convolution

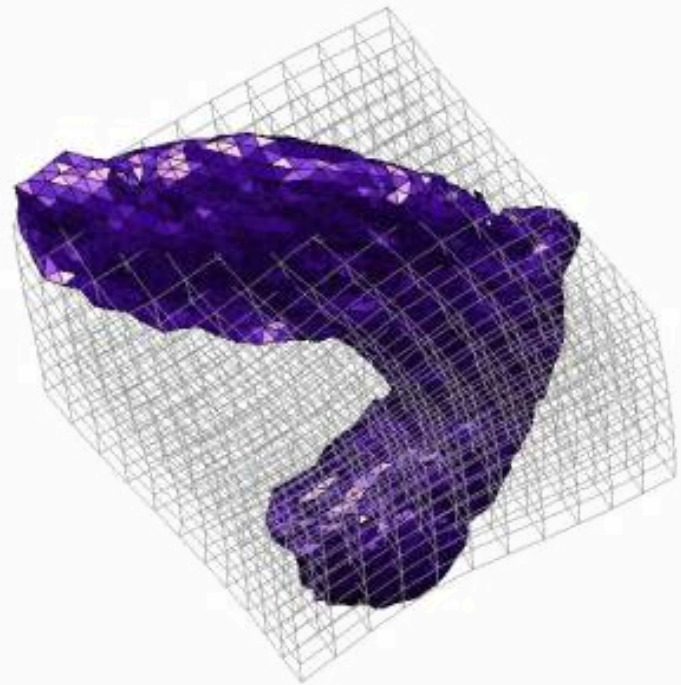


Mesh deformation

Limitations of CNNs [Ash07, Gla05]

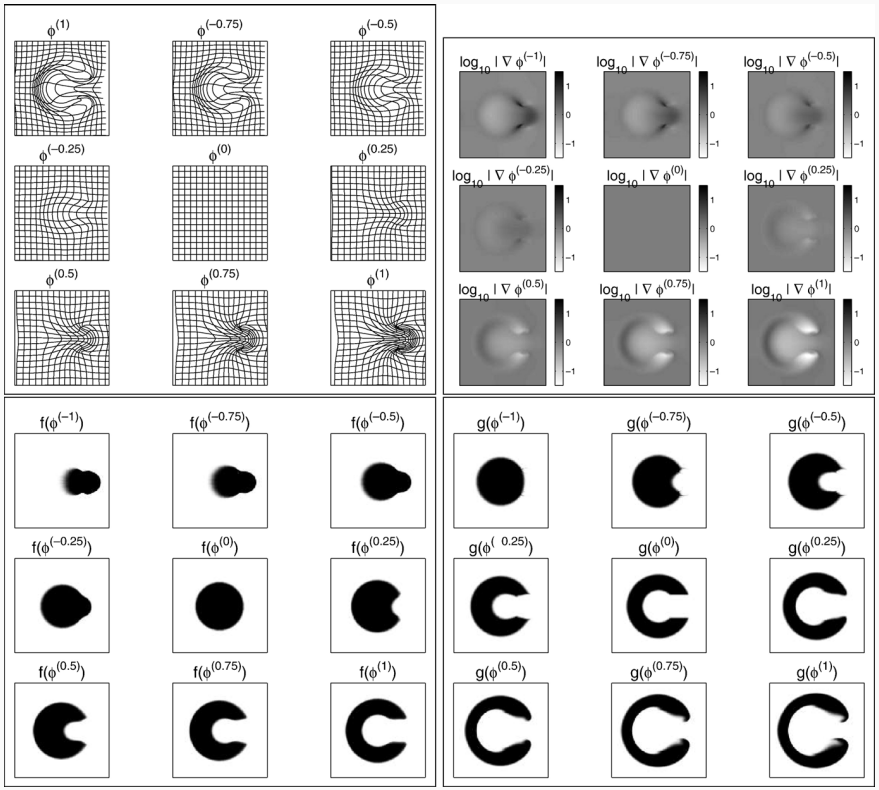


Advection ≠ Convolution

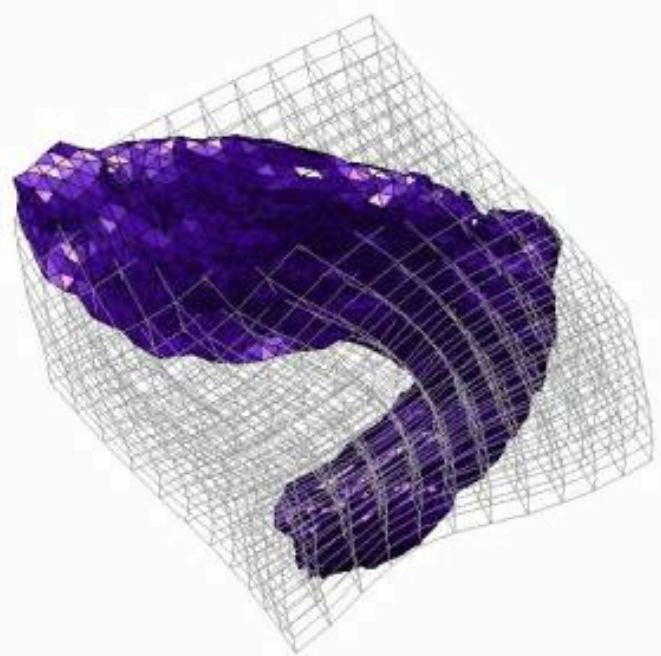


Mesh deformation

Limitations of CNNs [Ash07, Gla05]

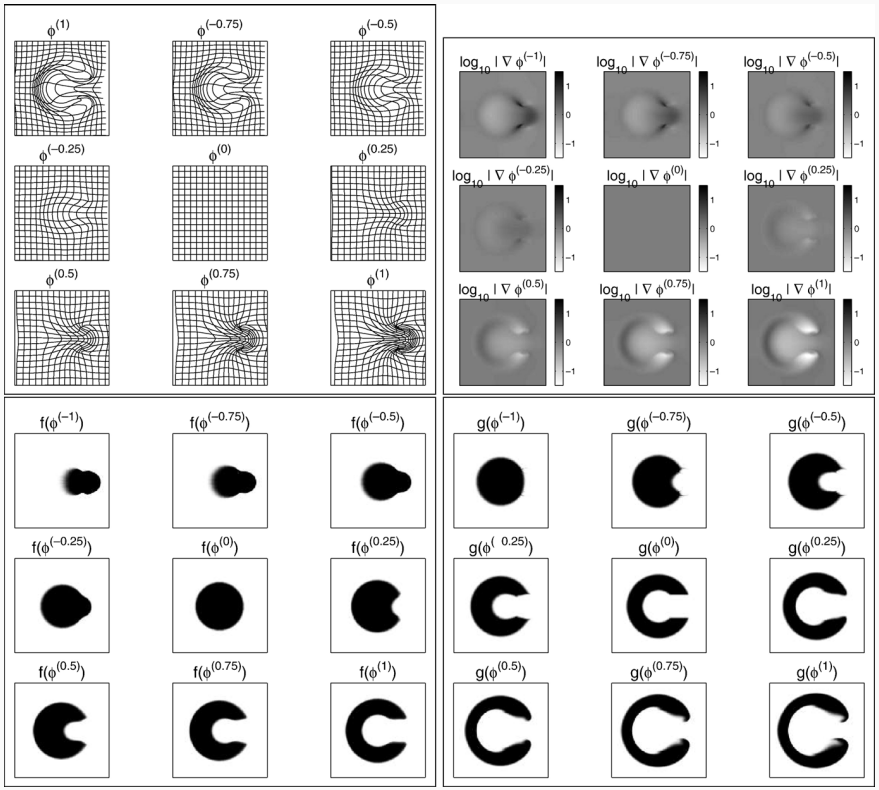


Advection ≠ Convolution

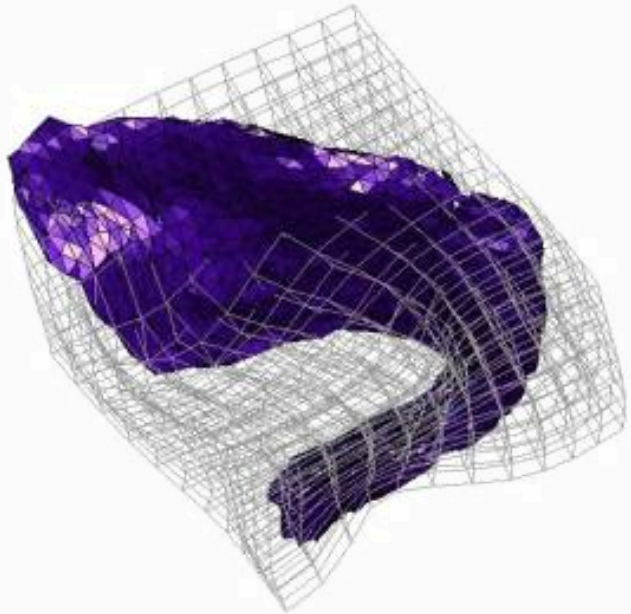


Mesh deformation

Limitations of CNNs [Ash07, Gla05]

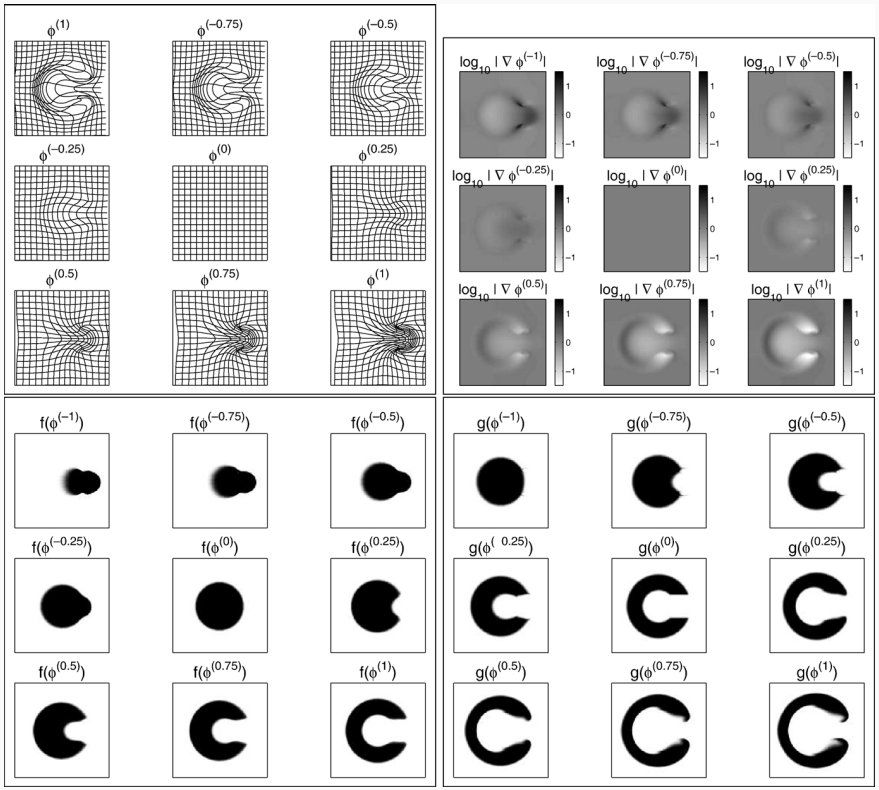


Advection ≠ Convolution

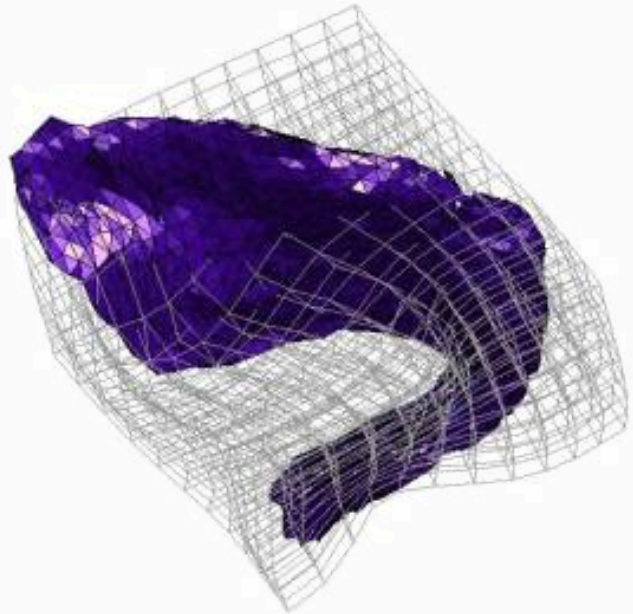


Mesh deformation

Limitations of CNNs [Ash07, Gla05]



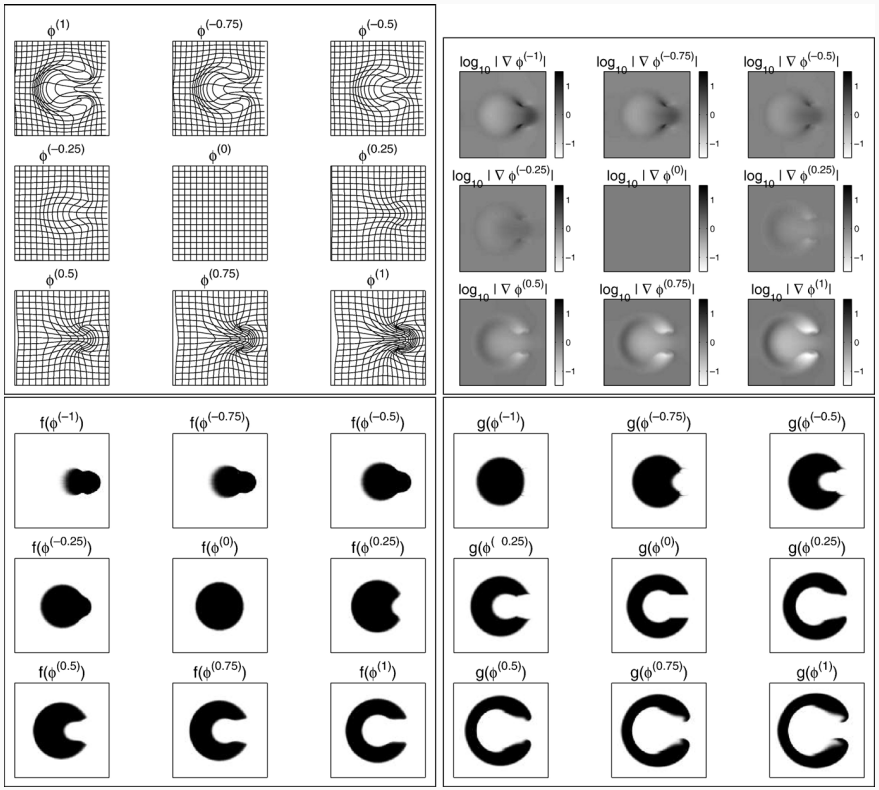
Advection $\neq$ Convolution



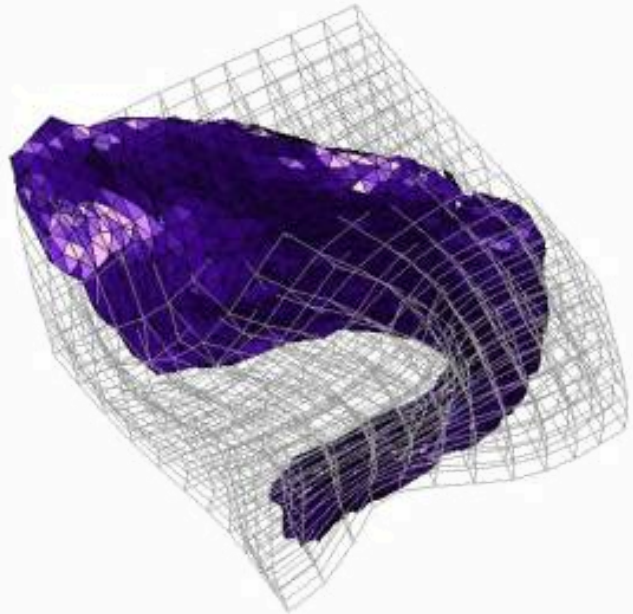
Mesh deformation

$\Longrightarrow$ Geometry is easier with **coordinates:** point clouds and meshes.

Limitations of CNNs [Ash07, Gla05]



Advection $\neq$ Convolution



Mesh deformation

$\Longrightarrow$ Geometry is easier with **coordinates:** point clouds and meshes.

Problem: not supported well by TensorFlow and PyTorch.

Geometry's back on the menu!

Today, we'll discuss:

Geometry's back on the menu!

Today, we'll discuss:

- **Efficient** GPU implementations
  $\longrightarrow$ KeOps extension for PyTorch, NumPy, Matlab, R.

Geometry's back on the menu!

Today, we'll discuss:

- **Efficient** GPU implementations
  $\longrightarrow$ KeOps extension for PyTorch, NumPy, Matlab, R.

- **Geometric** loss functions
  $\longrightarrow$ Optimal transport = Wasserstein distance.

Geometry's back on the menu!

Today, we'll discuss:

- **Efficient** GPU implementations
  $\longrightarrow$ KeOps extension for PyTorch, NumPy, Matlab, R.

- **Geometric** loss functions
  $\longrightarrow$ Optimal transport = Wasserstein distance.

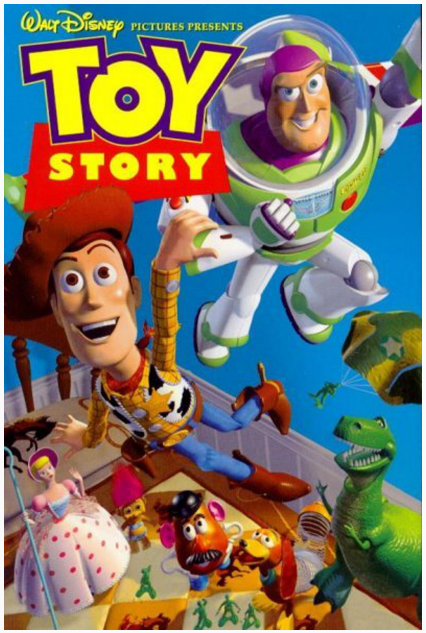
- **Robust** deformation architectures
  $\longrightarrow$ After the break :–)

Combining graphics and deep learning

What is a GPU?

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

Designed for graphics: 1 GFlop $\simeq$ 1 $



1995

Today

Deep learning frameworks: unlocking GPUs for research

Theano (RIP), TensorFlow and PyTorch combine:

- Matlab-like **Python interface**.

Deep learning frameworks: unlocking GPUs for research

Theano (RIP), TensorFlow and PyTorch combine:

- Matlab-like **Python interface**.
- CPU *and* **GPU** backends.

Deep learning frameworks: unlocking GPUs for research

Theano (RIP), TensorFlow and PyTorch combine:

- Matlab-like **Python interface**.
- CPU *and* **GPU** backends.
- **Automatic differentiation** engine.

Deep learning frameworks: unlocking GPUs for research

Theano (RIP), TensorFlow and PyTorch combine:

- Matlab-like **Python interface**.
- CPU *and* **GPU** backends.
- **Automatic differentiation** engine.
- Support for imaging (convolutions) and linear algebra.

Deep learning frameworks: unlocking GPUs for research

Theano (RIP), TensorFlow and PyTorch combine:

- Matlab-like **Python interface**.
- CPU *and* **GPU** backends.
- **Automatic differentiation** engine.
- Support for imaging (convolutions) and linear algebra.

Deep learning frameworks: unlocking GPUs for research

Theano (RIP), TensorFlow and PyTorch combine:

- Matlab-like **Python interface**.
- CPU *and* **GPU** backends.
- **Automatic differentiation** engine.
- Support for imaging (convolutions) and linear algebra.

$\Longrightarrow$ Ideally suited for research!

PyTorch, in practice

```
import torch              # GPU + autodiff library
# With PyTorch, using the GPU is that simple:
torch.set_default_tensor_type(torch.cuda.FloatTensor)
```

PyTorch, in practice

```
import torch            # GPU + autodiff library
# With PyTorch, using the GPU is that simple:
torch.set_default_tensor_type(torch.cuda.FloatTensor)

# Store arbitrary arrays on the GPU:
N, D = 10000, 3
x = torch.rand(N, D)  # Point clouds in [0,1]^3
y = torch.rand(N, D)  # encoded as N-by-D arrays
```

PyTorch, in practice

```
import torch                # GPU + autodiff library
# With PyTorch, using the GPU is that simple:
torch.set_default_tensor_type(torch.cuda.FloatTensor)

# Store arbitrary arrays on the GPU:
N, D = 10000, 3
x = torch.rand(N, D)  # Point clouds in [0,1]^3
y = torch.rand(N, D)  # encoded as N-by-D arrays

# Re-indexing:
x_i  = x[:,None,:]  # shape (N,D) -> (N,1,D)
y_j  = y[None,:,:]  # shape (N,D) -> (1,N,D)
```

PyTorch, in practice

```
import torch                # GPU + autodiff library
# With PyTorch, using the GPU is that simple:
torch.set_default_tensor_type(torch.cuda.FloatTensor)

# Store arbitrary arrays on the GPU:
N, D = 10000, 3
x = torch.rand(N, D)  # Point clouds in [0,1]^3
y = torch.rand(N, D)  # encoded as N-by-D arrays

# Re-indexing:
x_i  = x[:,None,:]  # shape (N,D) -> (N,1,D)
y_j  = y[None,:,:]  # shape (N,D) -> (1,N,D)
# Matrix of squared distances:
D_ij  = ((x_i - y_j)**2).sum(dim=2)  # (N,N,D) -> (N,N)
```

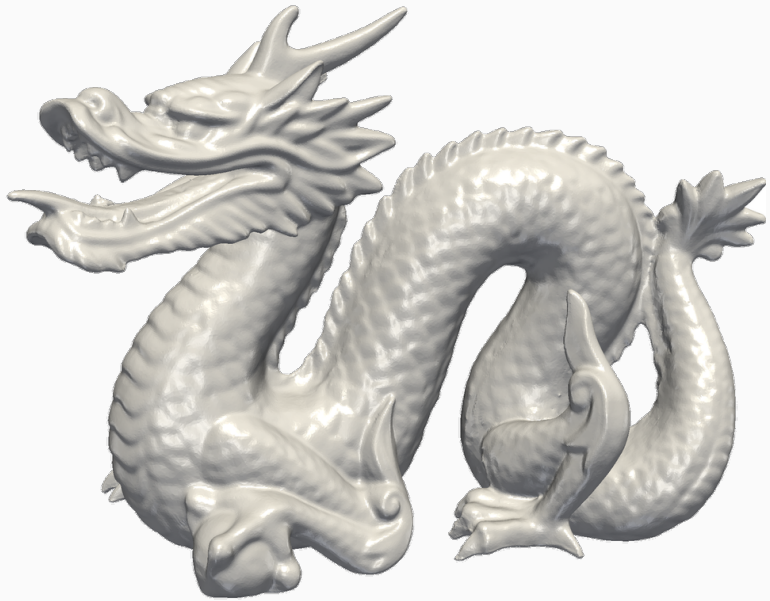
PyTorch, in practice

```
import torch                # GPU + autodiff library
# With PyTorch, using the GPU is that simple:
torch.set_default_tensor_type(torch.cuda.FloatTensor)

# Store arbitrary arrays on the GPU:
N, D = 10000, 3
x = torch.rand(N, D)  # Point clouds in [0,1]^3
y = torch.rand(N, D)  # encoded as N-by-D arrays

# Re-indexing:
x_i  = x[:,None,:]  # shape (N,D) -> (N,1,D)
y_j  = y[None,:,:]  # shape (N,D) -> (1,N,D)
# Matrix of squared distances:
D_ij  = ((x_i - y_j)**2).sum(dim=2)  # (N,N,D) -> (N,N)
```

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

Memory overflow with 10k points? We should do better. [CL96]

11k triangles



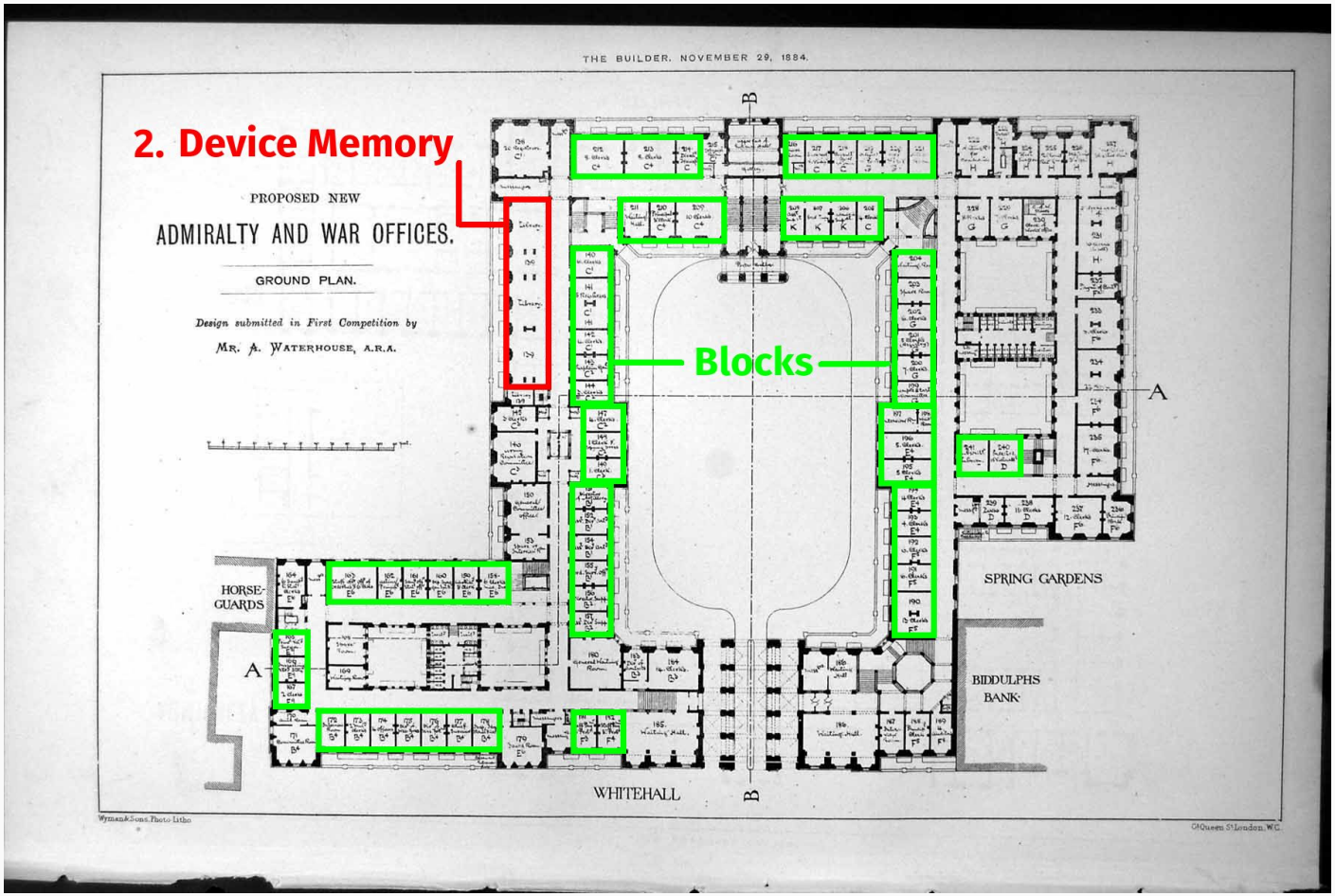
871k triangles

What a GPU really looks like

Turing architecture.

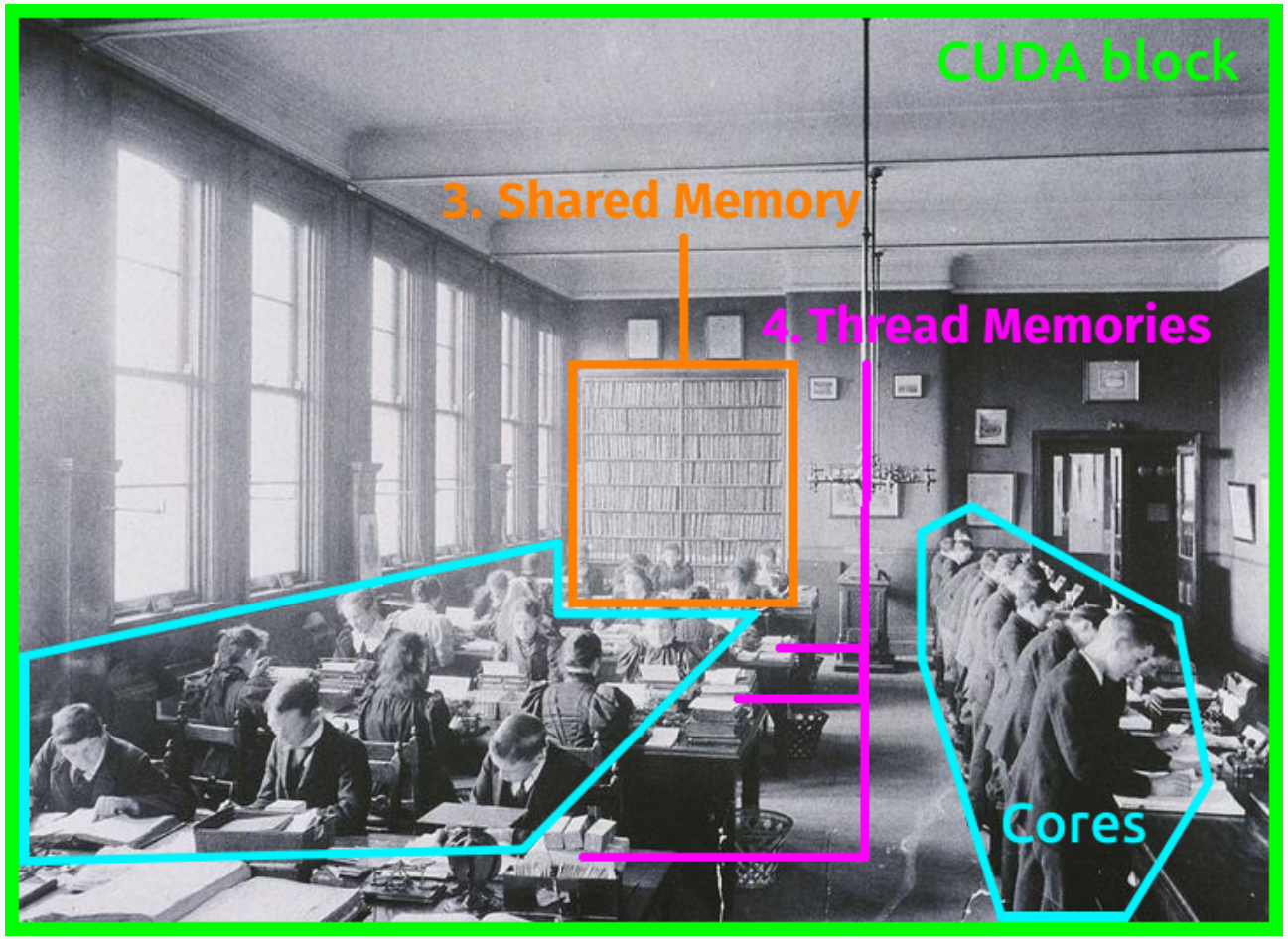
What a GPU really looks like

The Admiralty and War Offices, built in 1884.



What a GPU really looks like

Inside view of a CUDA block.



Memory management is key

Rules of CUDA programming:

- **Blocks** of ~200 threads.

Memory management is key

Rules of CUDA programming:

- **Blocks** of ~200 threads.
- 4 layers of **memory** (Host, Device, Block, Thread).

Memory management is key

Rules of CUDA programming:

- **Blocks** of ~200 threads.
- 4 layers of **memory** (Host, Device, Block, Thread).
- **Shared Block buffer** (~128 Kb) is 50x faster than Device mem.

Memory management is key

Rules of CUDA programming:

- **Blocks** of ~200 threads.
- 4 layers of **memory** (Host, Device, Block, Thread).
- **Shared Block buffer** (~128 Kb) is 50x faster than Device mem.

Memory management is key

Rules of CUDA programming:

- **Blocks** of ~200 threads.
- 4 layers of **memory** (Host, Device, Block, Thread).
- **Shared Block buffer** (~128 Kb) is 50x faster than Device mem.

PyTorch and TF variables are always stored in the **Device** mem.

Memory management is key

Rules of CUDA programming:

- **Blocks** of ~200 threads.
- 4 layers of **memory** (Host, Device, Block, Thread).
- **Shared Block buffer** (~128 Kb) is 50x faster than Device mem.

PyTorch and TF variables are always stored in the **Device** mem.
Explicit C++ implementations for:

- Convolutions.
- Fourier, wavelet transforms.

Memory management is key

Rules of CUDA programming:

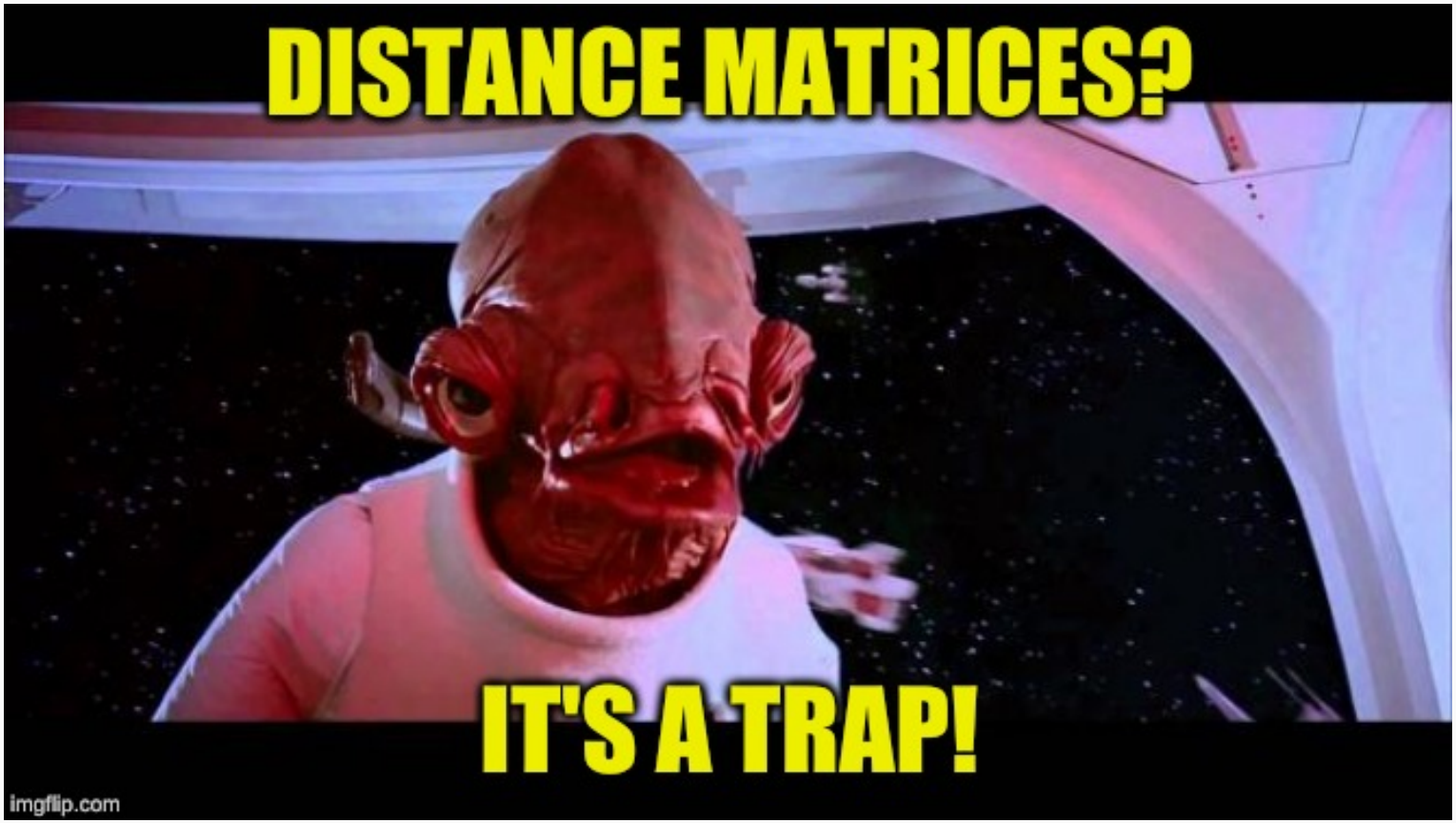
- **Blocks** of ~200 threads.
- 4 layers of **memory** (Host, Device, Block, Thread).
- **Shared Block buffer** (~128 Kb) is 50x faster than Device mem.

PyTorch and TF variables are always stored in the **Device** mem.
Explicit C++ implementations for:

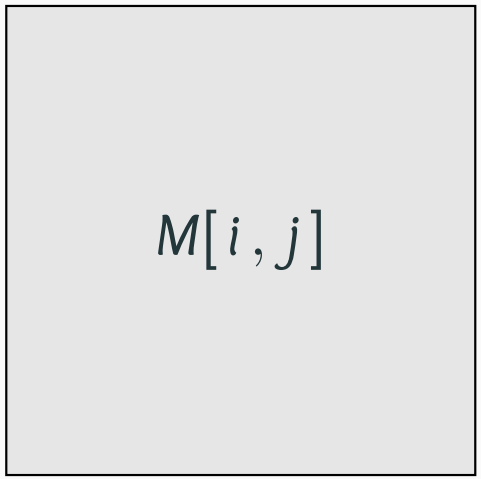
- Convolutions.
- Fourier, wavelet transforms.

$\Longrightarrow$ We need to do the same!

Keep in mind...



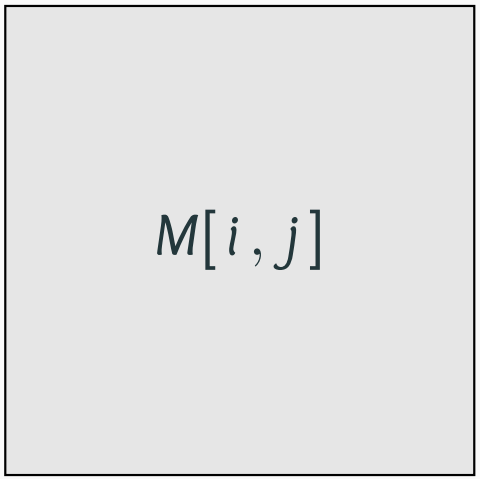
Introducing KeOps LazyTensors



Array

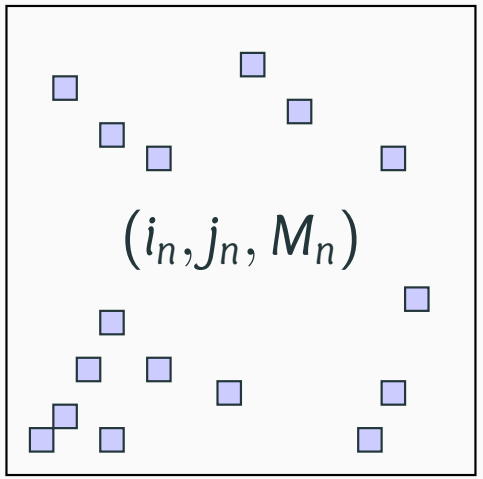
Coefficients only

Introducing KeOps LazyTensors



Array

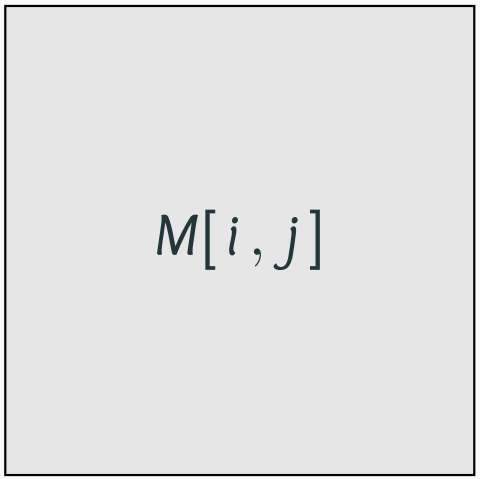
Coefficients only



Sparse matrix

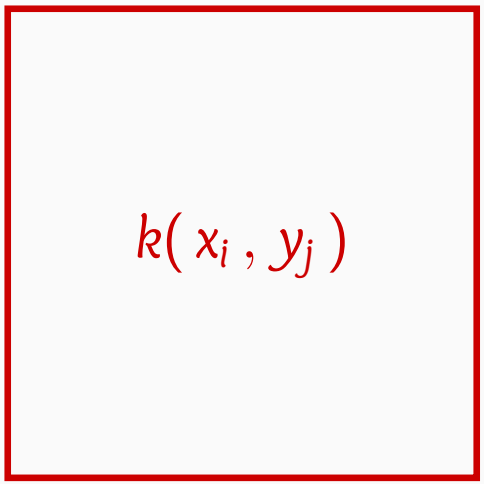
Coordinates + coeffs

Introducing KeOps LazyTensors



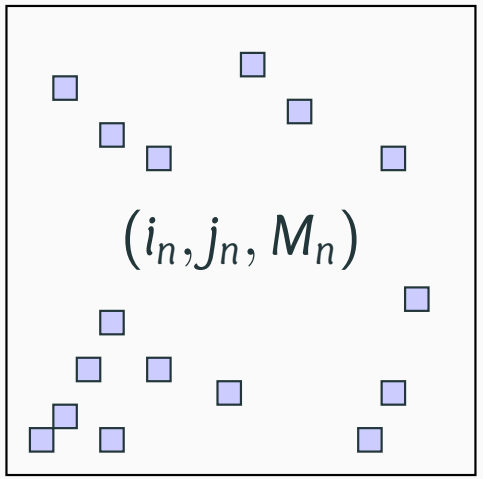
Array

Coefficients only



Symbolic LazyTensor

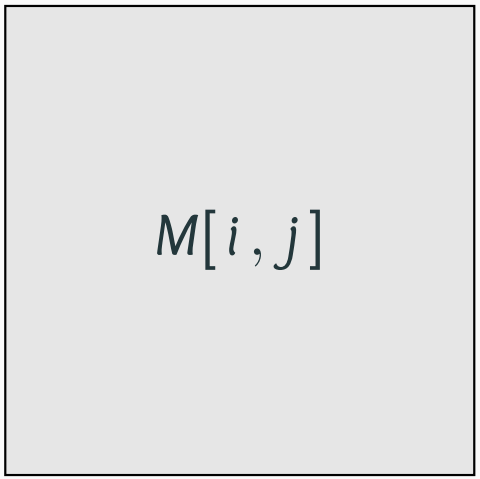
Formula + point clouds



Sparse matrix

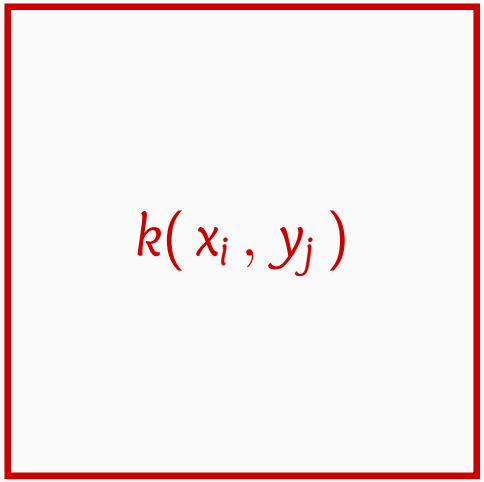
Coordinates + coeffs

Introducing KeOps LazyTensors



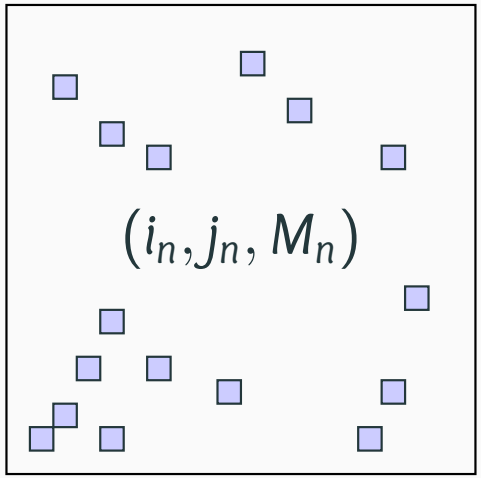
Array

Coefficients only



Symbolic LazyTensor

Formula + point clouds



Sparse matrix

Coordinates + coeffs

```
pip install pykeops
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)

# Perform large-scale computations:
D_ij = ((x_i - y_j)**2).sum(dim=2)  # (1e6,2e6,1)
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)

# Perform large-scale computations:
D_ij = ((x_i - y_j)**2).sum(dim=2)  # (1e6,2e6,1)
K_ij = (- D_ij).exp()               # (1e6,2e6,1)
```

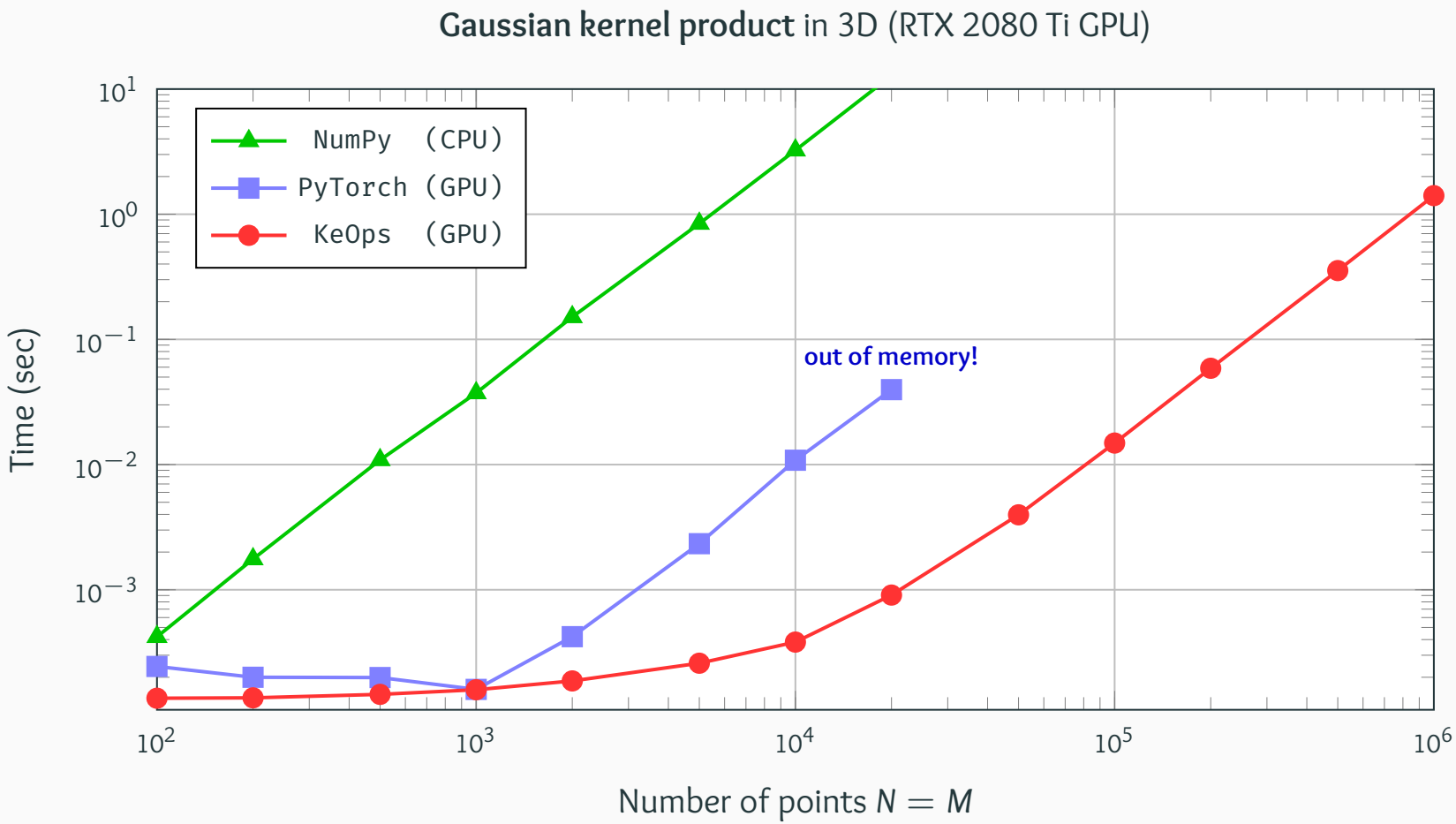
It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)

# Perform large-scale computations:
D_ij = ((x_i - y_j)**2).sum(dim=2)  # (1e6,2e6,1)
K_ij = (- D_ij).exp()               # (1e6,2e6,1)

# Reduction: symbolic LazyTensor -> genuine torch Tensor
a_i = K_ij.sum(dim=1)  # (1e6, 1) torch.cuda.FloatTensor
```

It works!

```
# Large point clouds in [0,1]^3
import torch
x = torch.rand(1000000, 3, requires_grad=True).cuda()
y = torch.rand(2000000, 3).cuda()

# Turn our Tensors into KeOps symbolic variables:
from pykeops.torch import LazyTensor
x_i = LazyTensor(x[:,None,:])  # x_i.shape = (1e6, 1, 3)
y_j = LazyTensor(y[None,:,:])  # y_j.shape = ( 1, 2e6,3)

# Perform large-scale computations:
D_ij = ((x_i - y_j)**2).sum(dim=2)  # (1e6,2e6,1)
K_ij = (- D_ij).exp()               # (1e6,2e6,1)

# Reduction: symbolic LazyTensor -> genuine torch Tensor
a_i = K_ij.sum(dim=1)  # (1e6, 1) torch.cuda.FloatTensor
g_x = torch.autograd.grad((a_i ** 2).sum(), [x])
```

Scaling up to real data



The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier.**

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.
- **Efficient**: $O(N)$ memory, competitive runtimes.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.
- **Efficient**: $O(N)$ memory, competitive runtimes.
- **Powerful**: autodiff, block-sparsity, etc.

The KeOps library

- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.
- **Efficient**: $O(N)$ memory, competitive runtimes.
- **Powerful**: autodiff, block-sparsity, etc.
- **Transparent**: interface with **SciPy**, GPytorch, etc.

The KeOps library

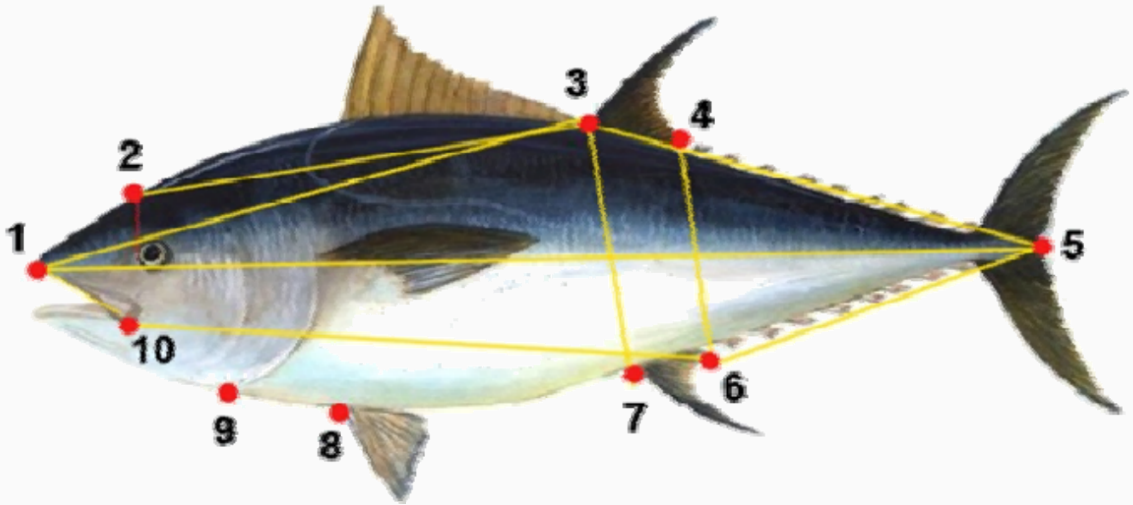
- 2 years of work with **Joan Glaunès** and **Benjamin Charlier**.
- **Cross-platform**: R, Matlab, NumPy *and* PyTorch.
- **Versatile**: many operations, variables, reductions.
- **Efficient**: $O(N)$ memory, competitive runtimes.
- **Powerful**: autodiff, block-sparsity, etc.
- **Transparent**: interface with **SciPy**, GPytorch, etc.

- Scikit-learn-like **documentation**:

`www.kernel-operations.io`

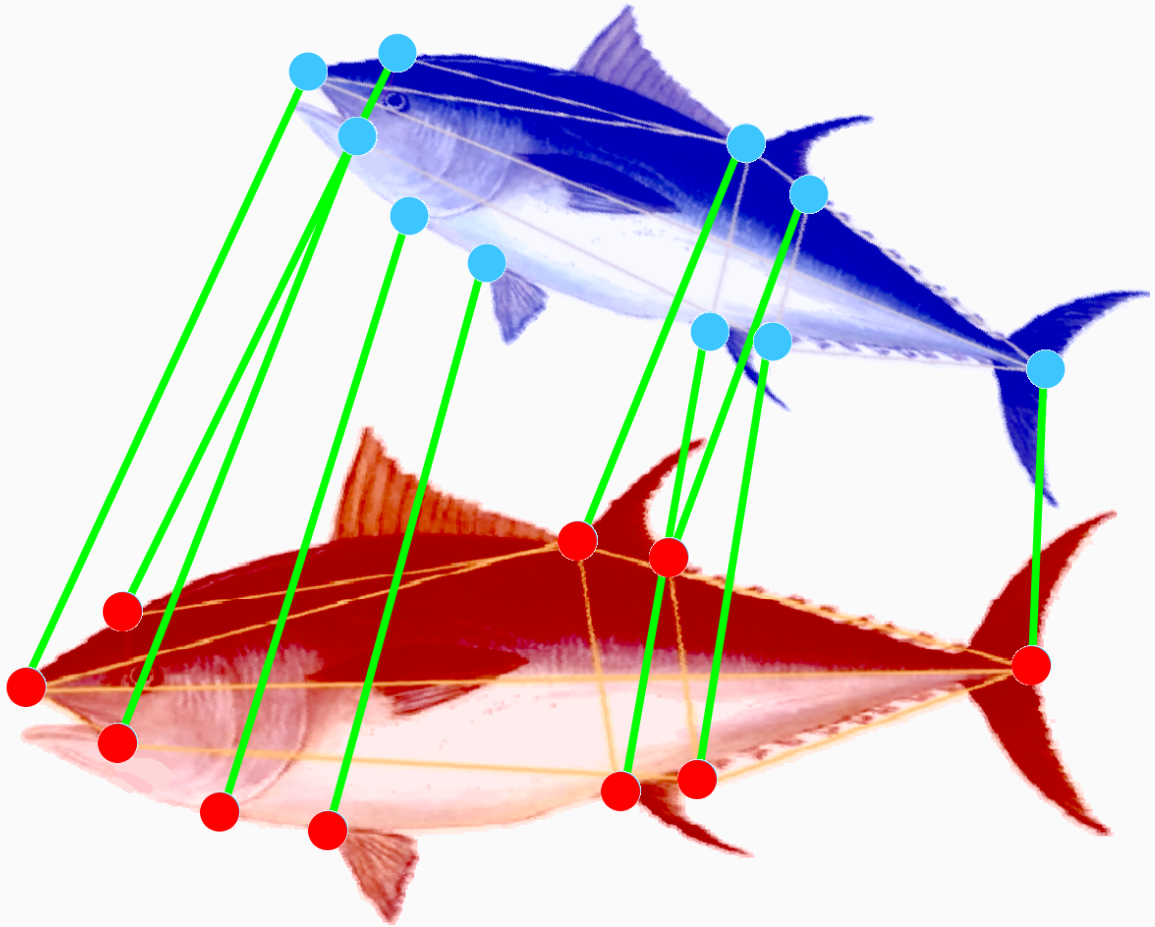
Geometric loss functions

Life is easy when you have landmarks...



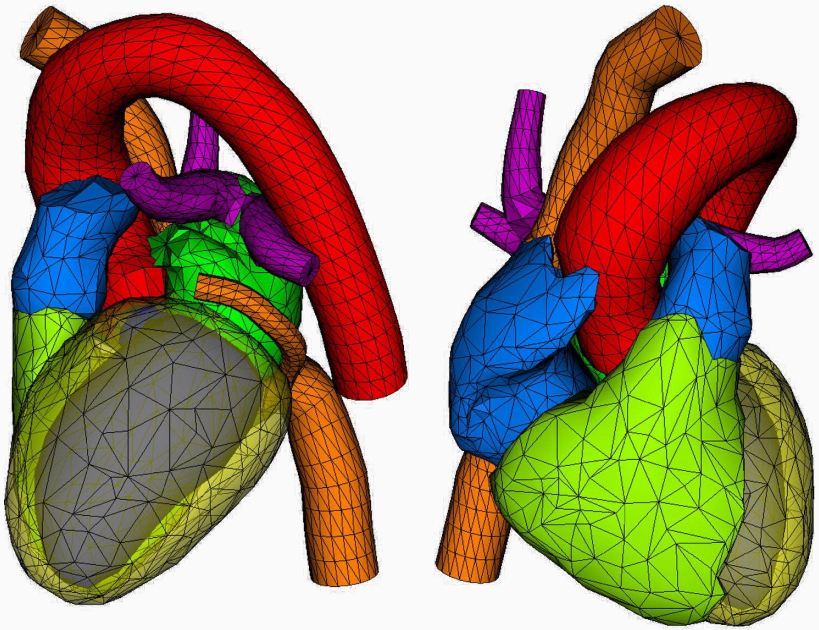
Anatomical landmarks from *A morphometric approach for the analysis of body shape in bluefin tuna*, Addis et al., 2009.

Life is easy when you have landmarks...

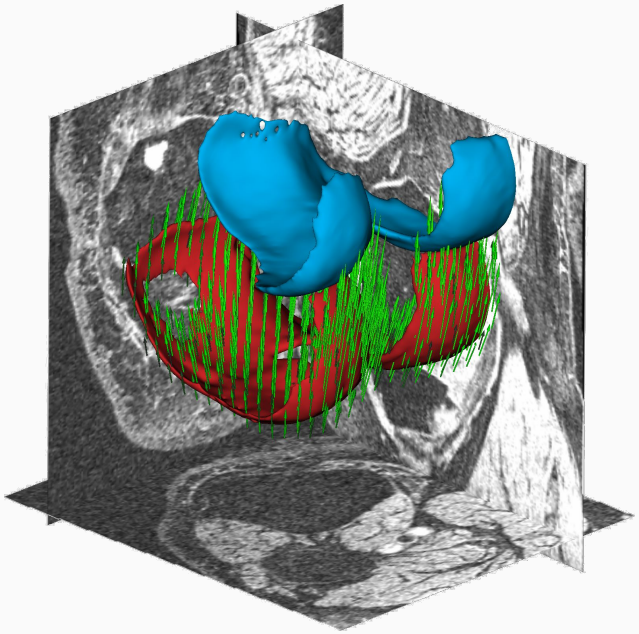


Anatomical landmarks from *A morphometric approach for the analysis of body shape in bluefin tuna*, Addis et al., 2009.

Unfortunately, medical data is often unlabeled [EPW+11]



Surface meshes



Segmentation masks

Encoding unlabeled shapes as measures

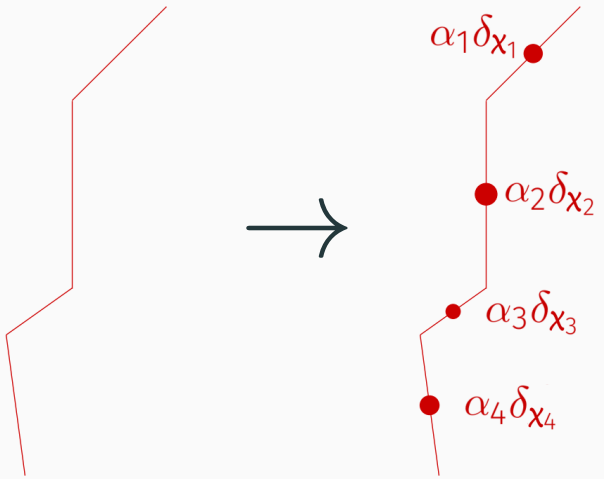
Let's enforce sampling invariance:

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

Encoding unlabeled shapes as measures

Let's enforce sampling invariance:

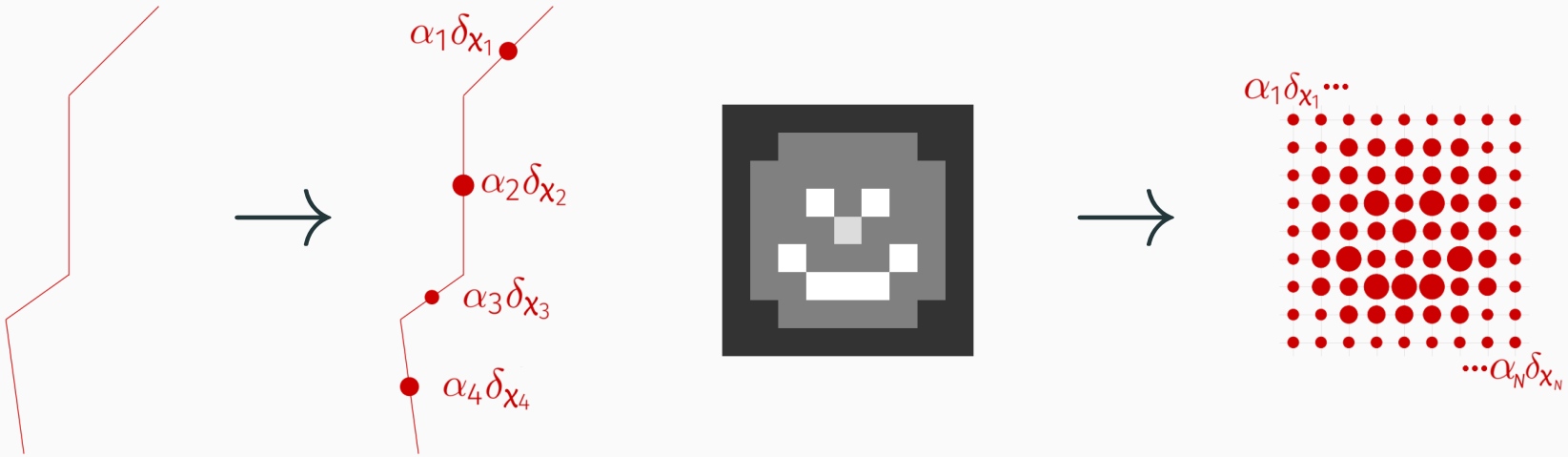
$$A \longrightarrow \alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i} , \qquad B \longrightarrow \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j} .$$



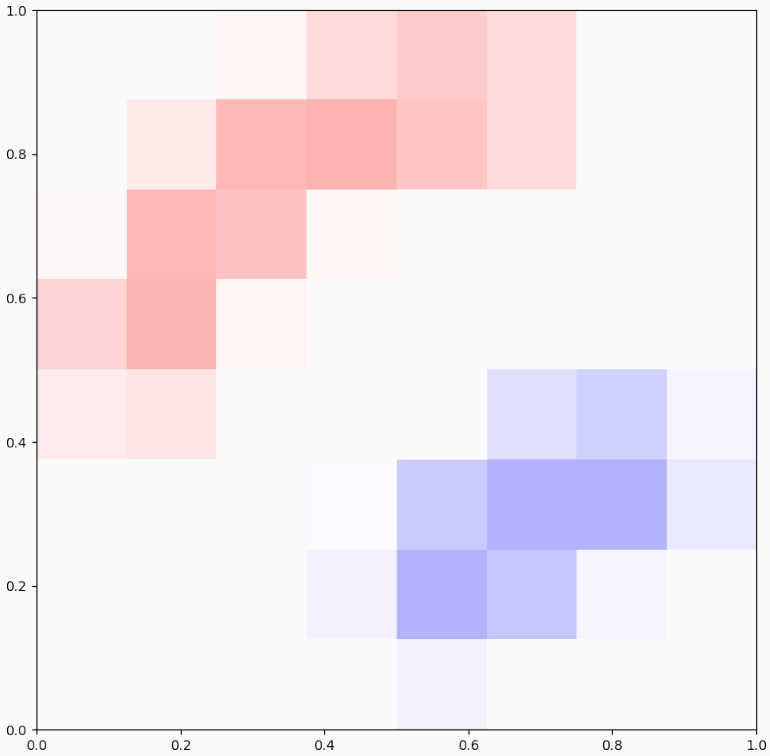
Encoding unlabeled shapes as measures

Let's enforce sampling invariance:

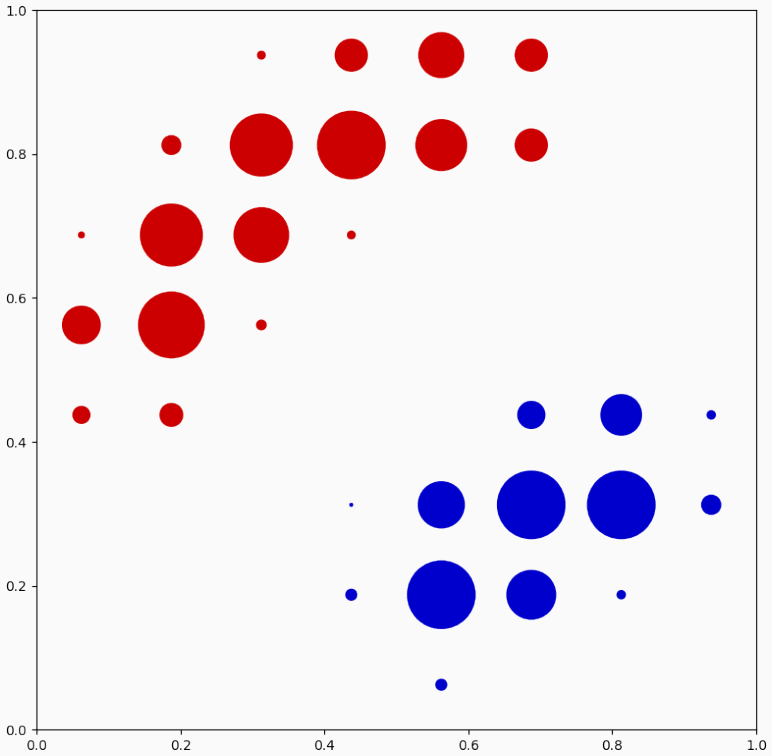
$$A \longrightarrow \alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i} , \qquad B \longrightarrow \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j} .$$



A baseline setting: density registration

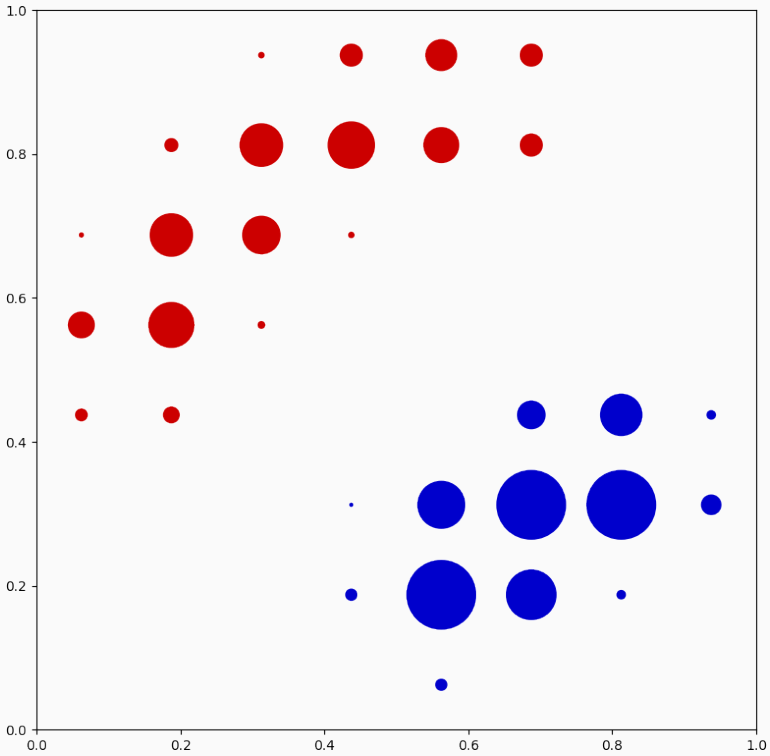


A baseline setting: density registration



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

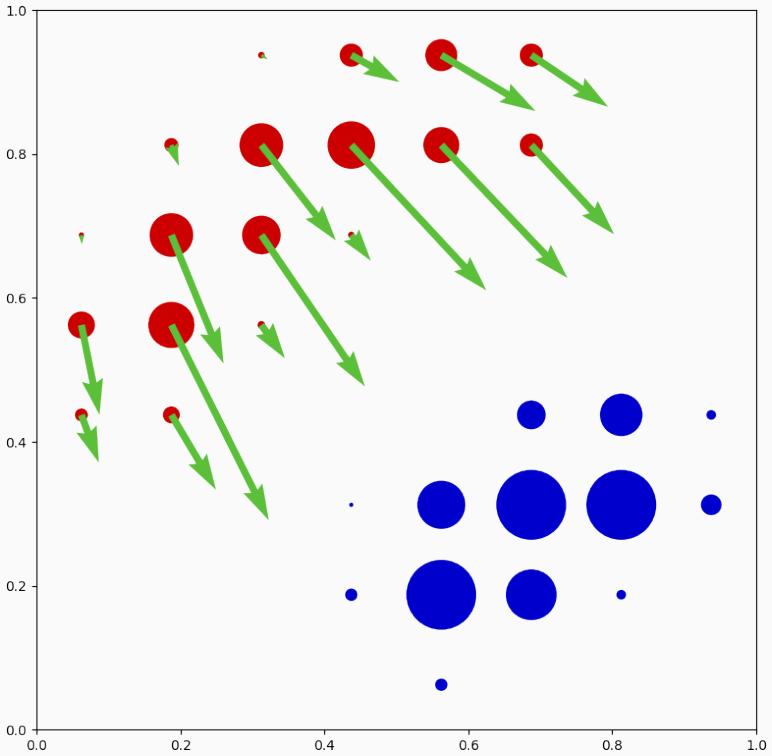
A baseline setting: density registration



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

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

A baseline setting: density registration

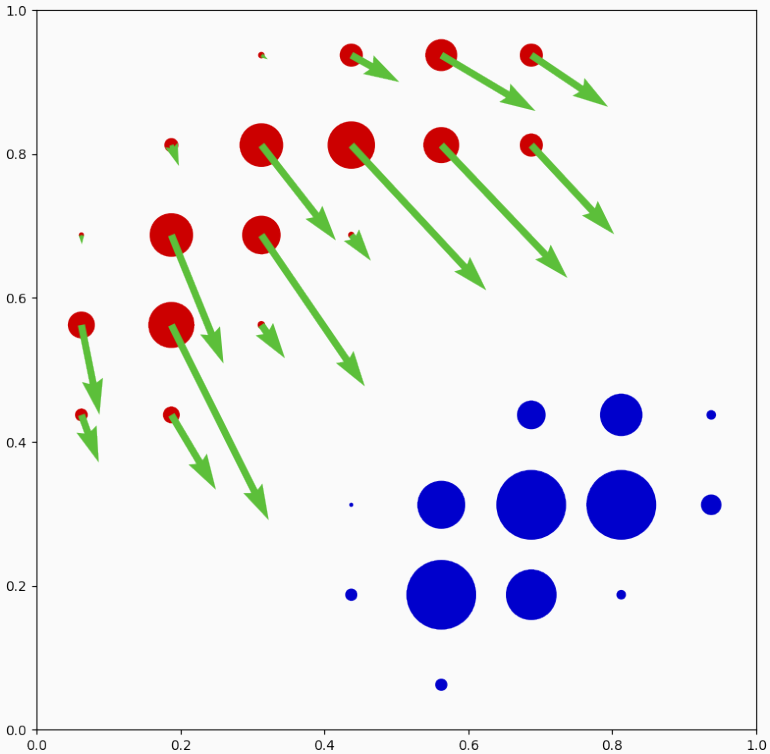


$$\alpha = \sum_{i=1}^{N} \alpha_i \delta_{x_i}, \quad \beta = \sum_{j=1}^{M} \beta_j \delta_{y_j}.$$

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

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

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

Seamless extensions to:

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

Simple loss functions

- **Chamfer** distance $\simeq$ soft-**Hausdorff**:
  Projection-based $\longrightarrow$ Degenerate gradients.

Simple loss functions

- **Chamfer** distance $\simeq$ soft-**Hausdorff**:
  Projection-based $\longrightarrow$ Degenerate gradients.

- Kernel distance $\simeq$ Blurred SSD, convolution-based:

  $$\text{Loss}(\alpha, \beta) \;=\; \tfrac{1}{2}\|g \star (\alpha - \beta)\|^2_{L^2(\mathbb{R}^D)} \;=\; \tfrac{1}{2}\langle\, \alpha - \beta \,,\, k \star (\alpha - \beta)\,\rangle$$

  where $k \;=\; (g \circ (x \mapsto -x)) \star g\,.$

Simple loss functions

- **Chamfer** distance $\simeq$ soft-**Hausdorff**:
  Projection-based $\longrightarrow$ Degenerate gradients.

- Kernel distance $\simeq$ Blurred SSD, convolution-based:

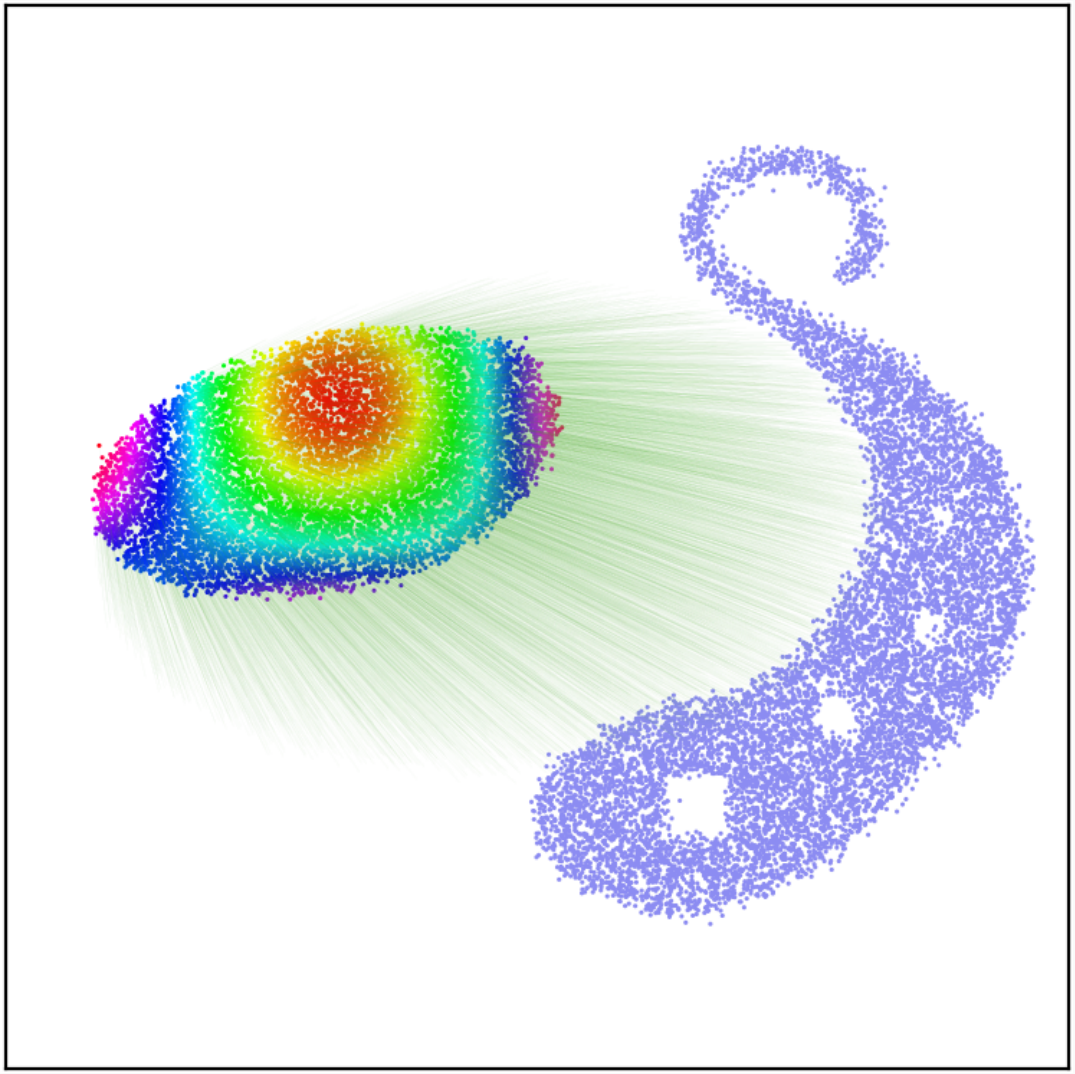
$$\text{Loss}(\alpha, \beta) \;=\; \tfrac{1}{2}\|g \star (\alpha - \beta)\|^2_{L^2(\mathbb{R}^D)} \;=\; \tfrac{1}{2}\langle\, \alpha - \beta \,,\, k \star (\alpha - \beta)\,\rangle$$

  where $k \;=\; (g \circ (x \mapsto -x)) \star g\,.$

- Example: the Energy Distance, $k(x,y) = -\|x-y\|$:

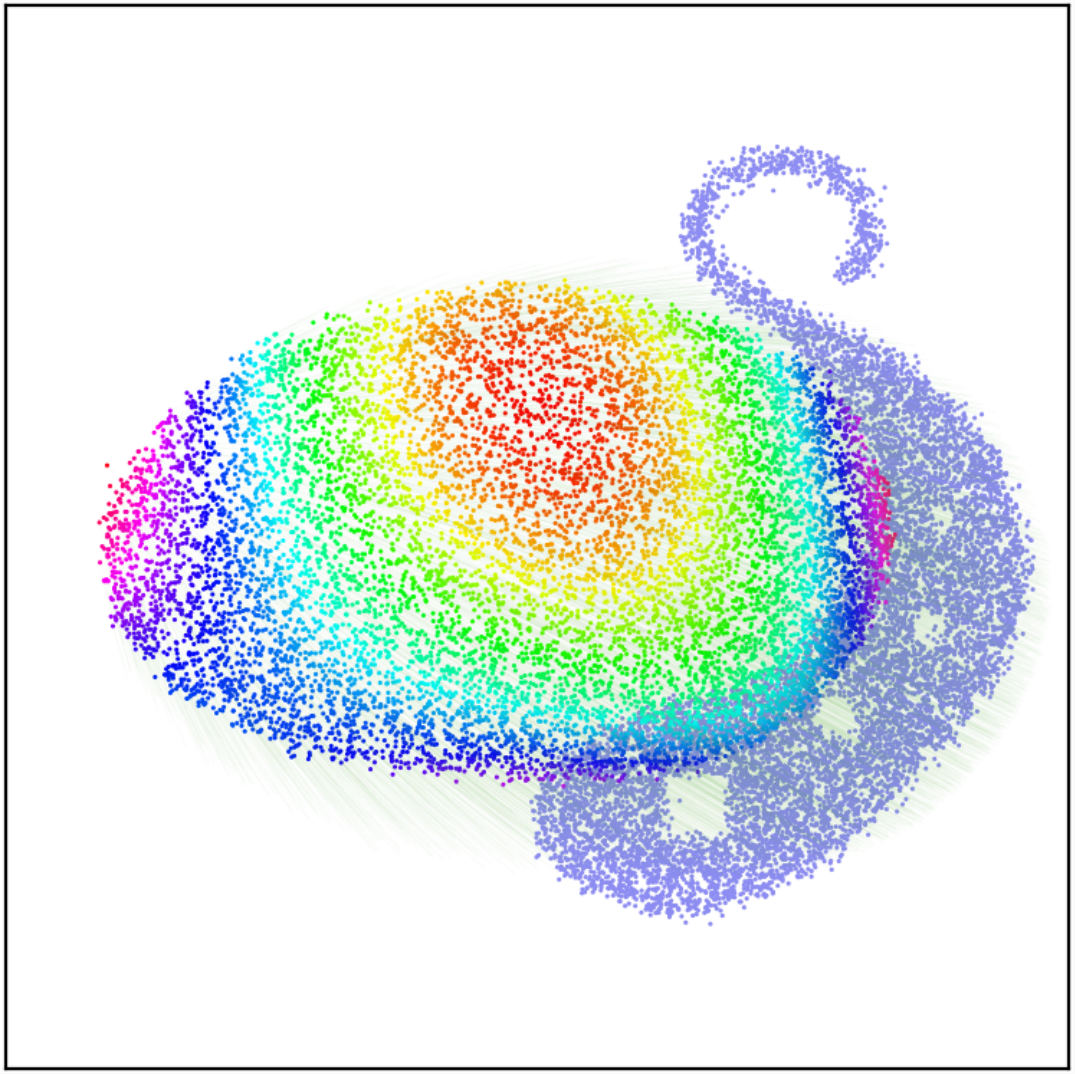
$$\begin{aligned}\text{Loss}(\alpha, \beta) \;=\; & \sum_i \sum_j \alpha_i \beta_j \, \|x_i - y_j\| \\ & - \tfrac{1}{2} \sum_i \sum_j \alpha_i \alpha_j \, \|x_i - x_j\| \;-\; \tfrac{1}{2} \sum_i \sum_j \beta_i \beta_j \, \|y_i - y_j\| \,.\end{aligned}$$

Gradient flow as a toy registration problem



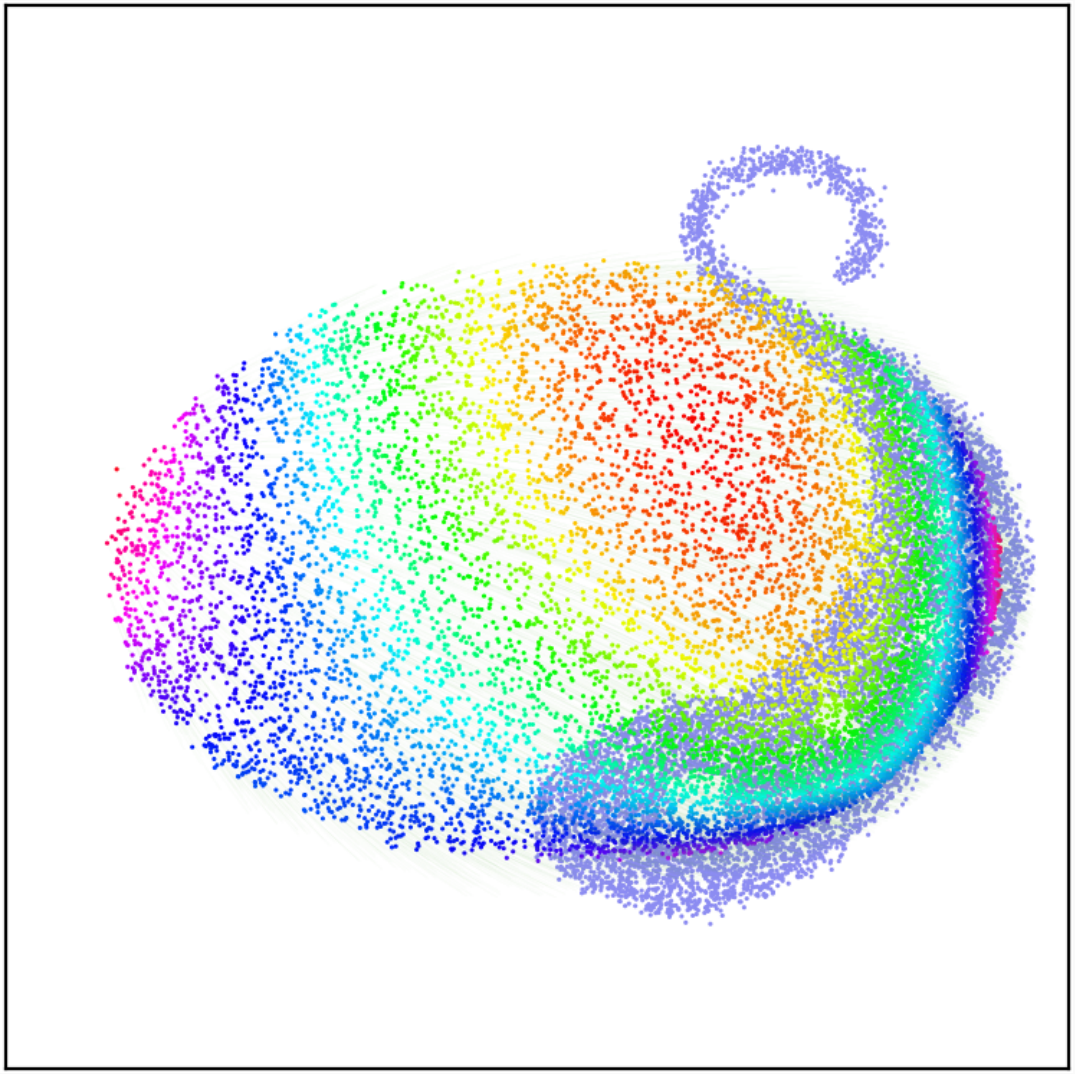
$t = .00$

Gradient flow as a toy registration problem



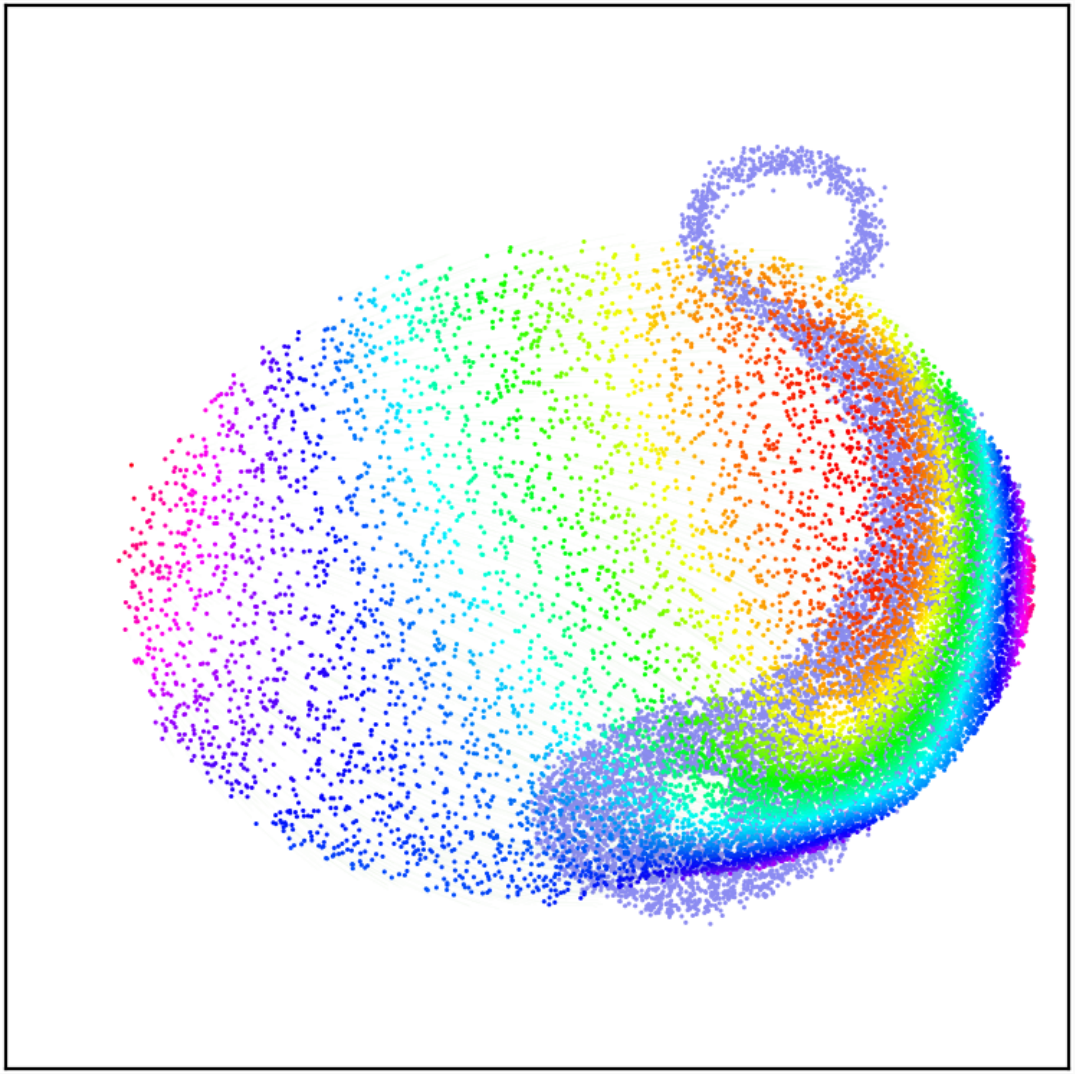
$t = .25$

Gradient flow as a toy registration problem



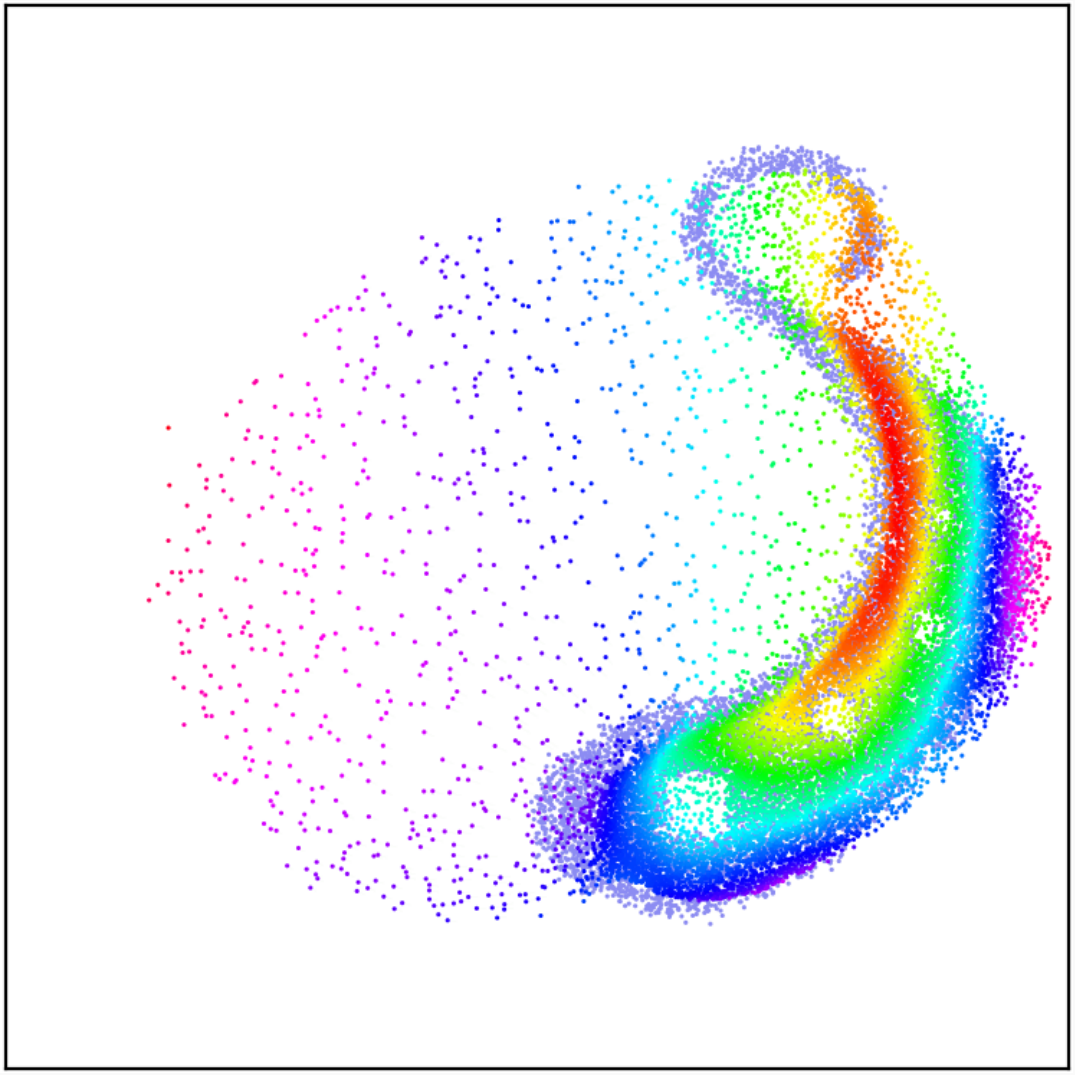
$t = .50$

Gradient flow as a toy registration problem



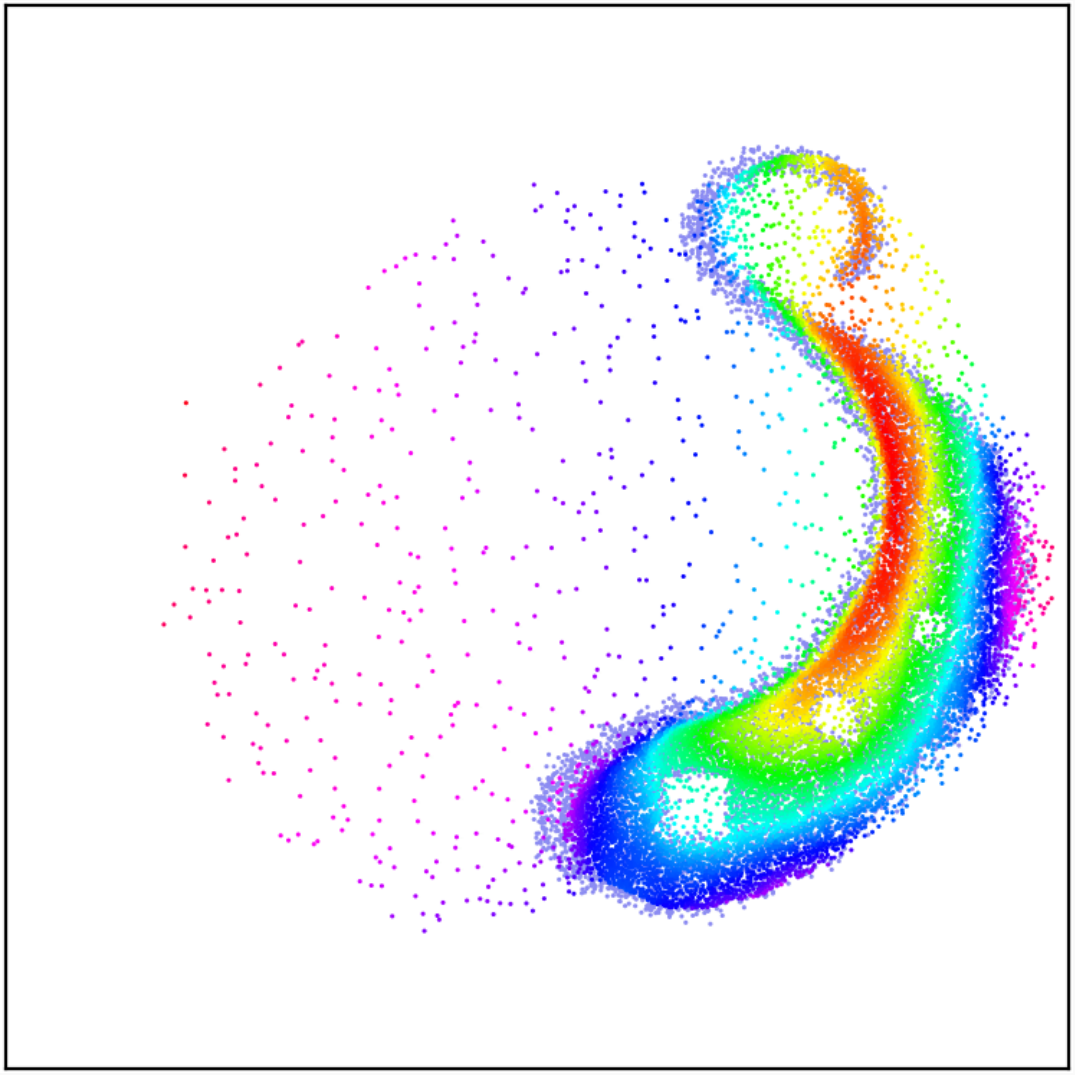
$t = 1.00$

Gradient flow as a toy registration problem



$t = 5.00$

Gradient flow as a toy registration problem



$t = 10.00$

The Wasserstein distance

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

The Wasserstein distance

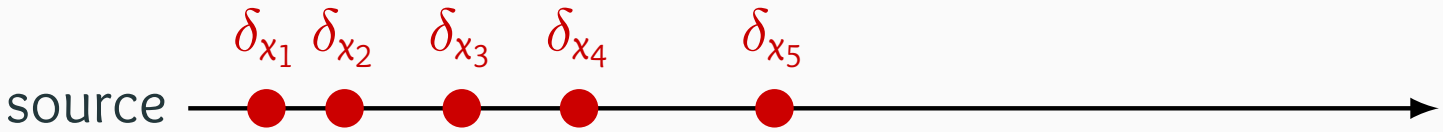
We need **clean gradients**, without artifacts.

Simple toy example in 1D :

The Wasserstein distance

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

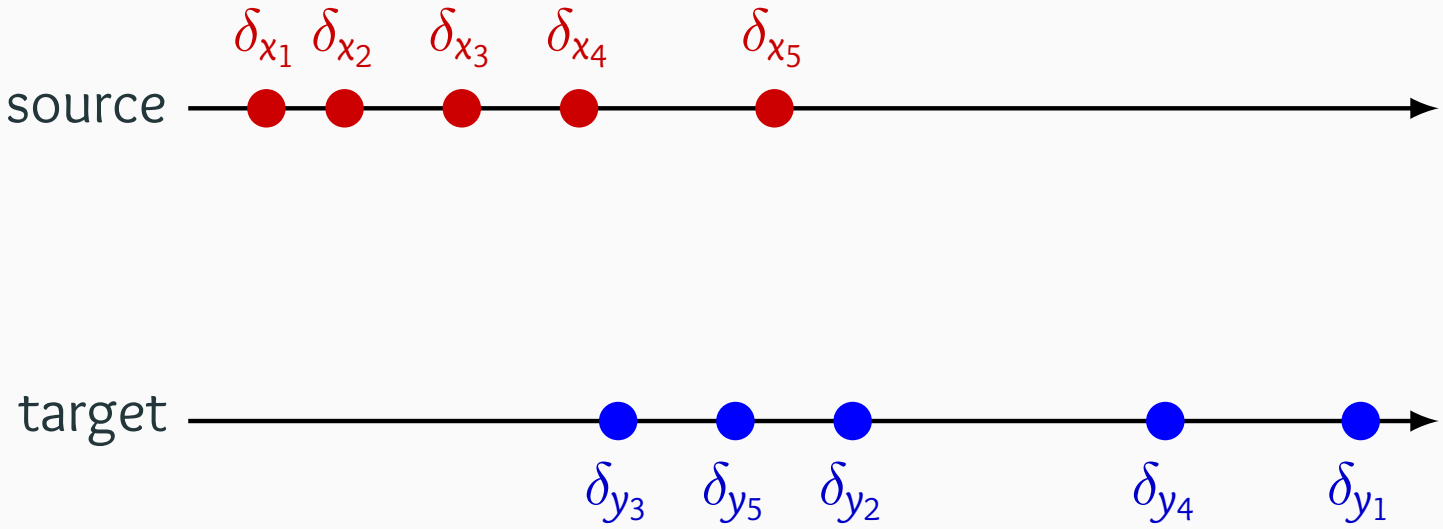
Simple toy example in 1D :



The Wasserstein distance

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

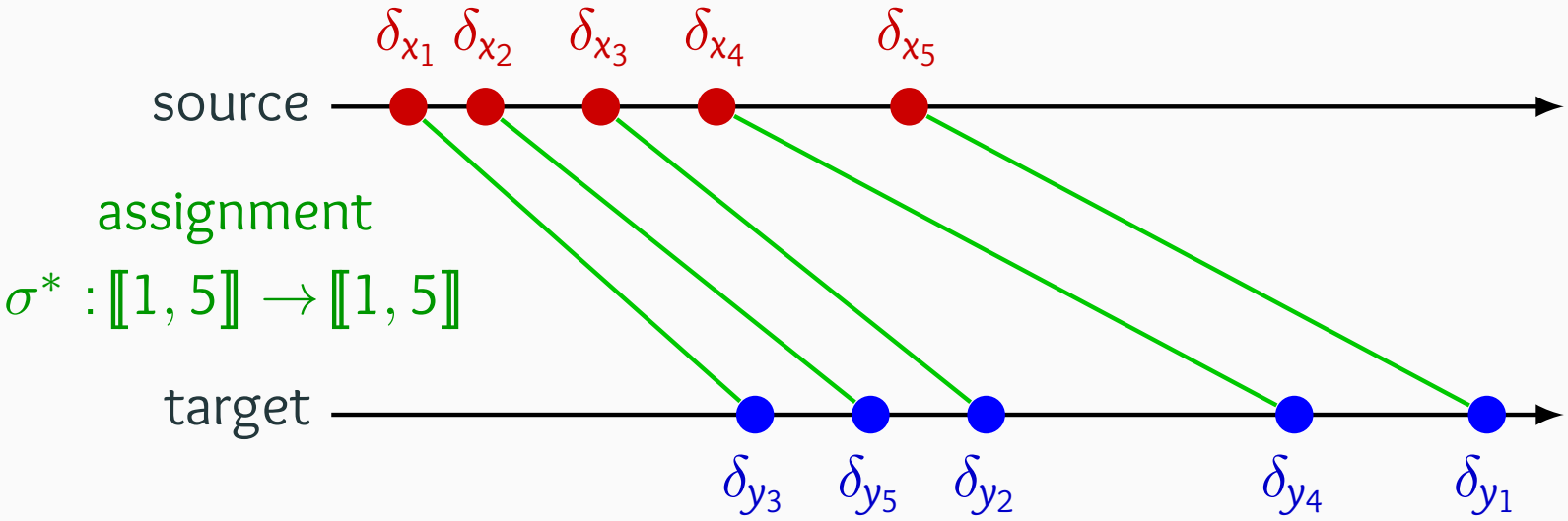
Simple toy example in 1D :



The Wasserstein distance

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

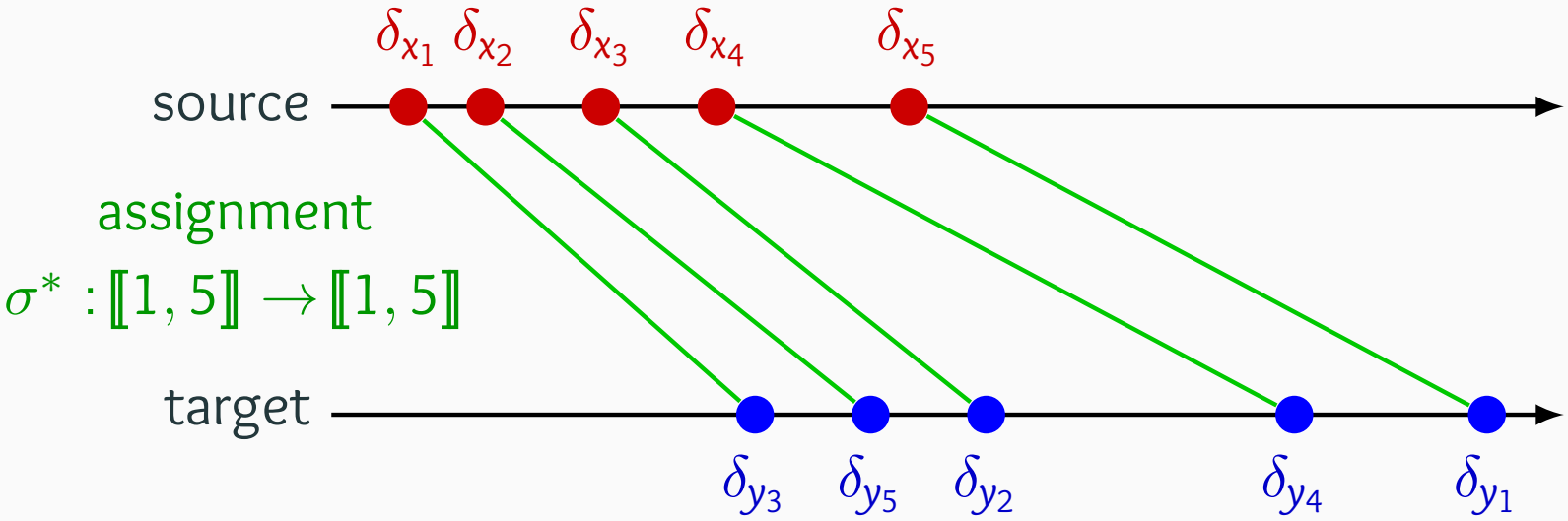
Simple toy example in 1D :



The Wasserstein distance

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

Simple toy example in 1D :

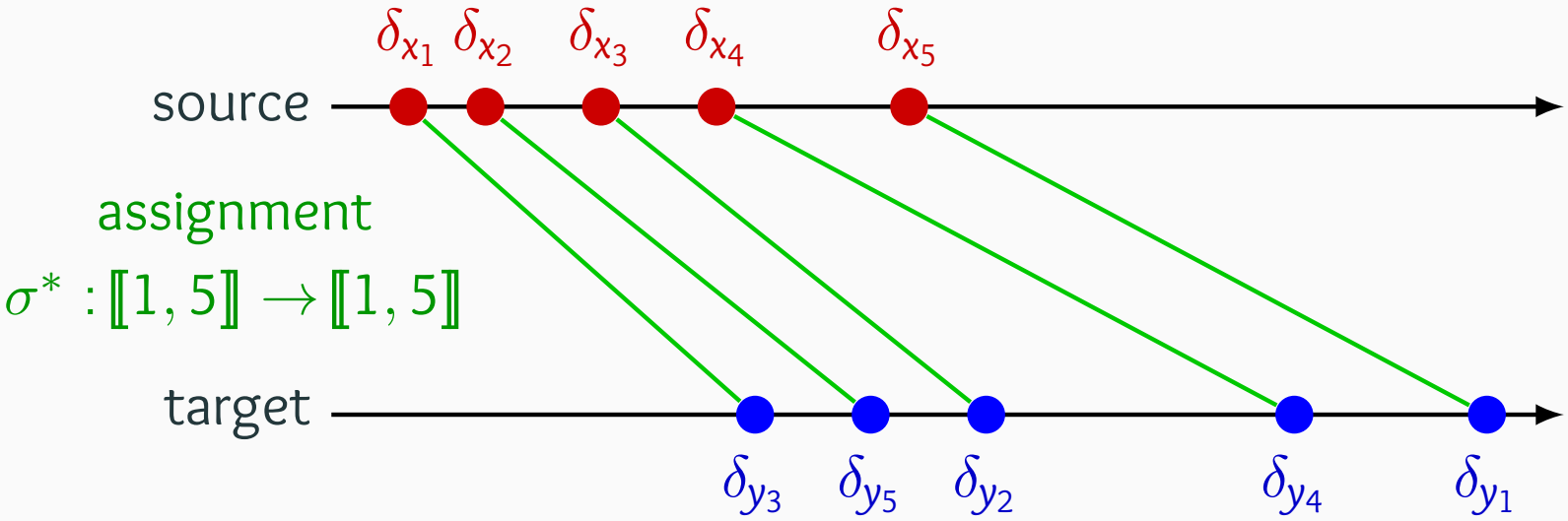


$$\mathrm{OT}(\alpha, \beta) = \frac{1}{2N} \sum_{i=1}^{N} |x_i - y_{\sigma^*(i)}|^2$$

The Wasserstein distance

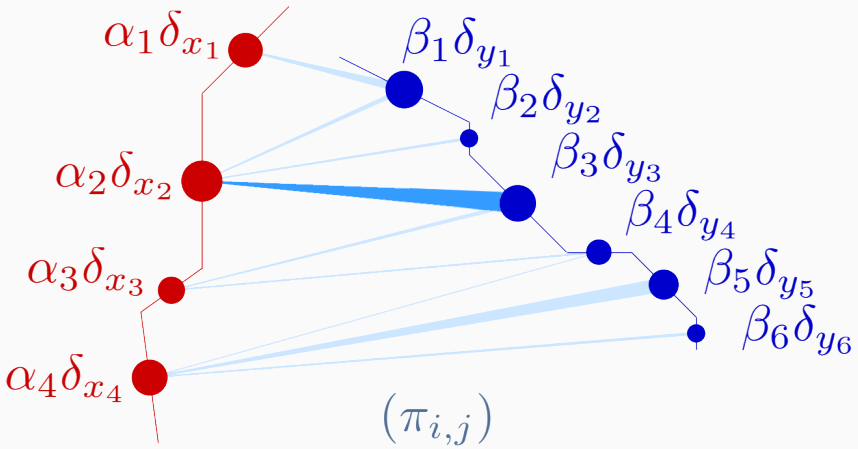
We need **clean gradients**, without artifacts.

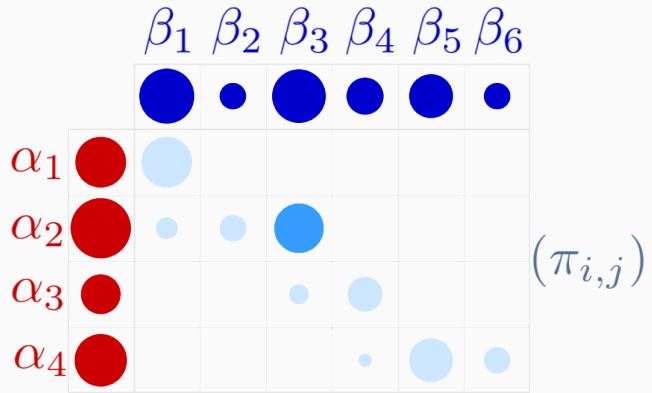
Simple toy example in 1D :



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

Optimal transport generalizes sorting to D $>$ 1





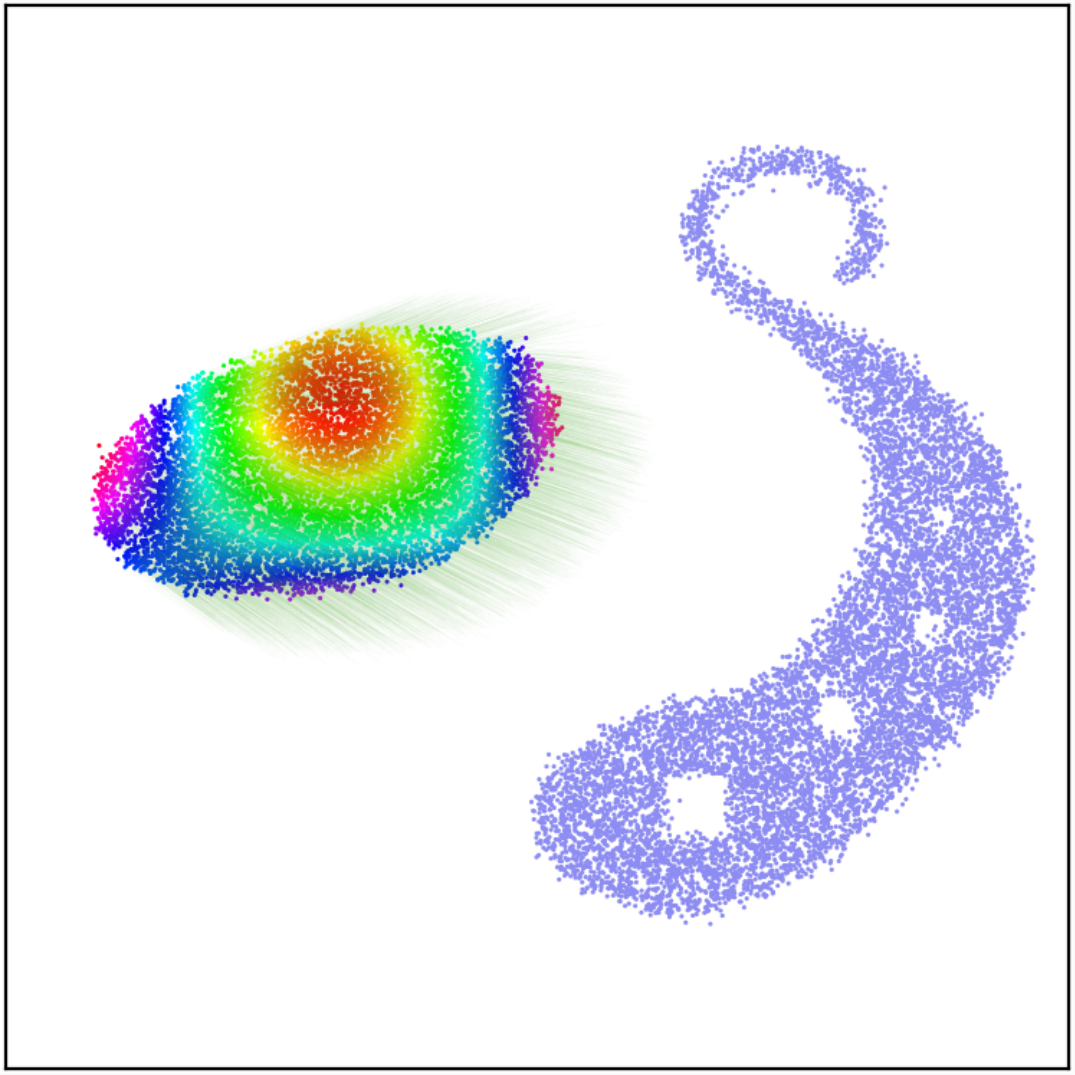
Minimize over N-by-M matrices (transport plans) π :

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

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

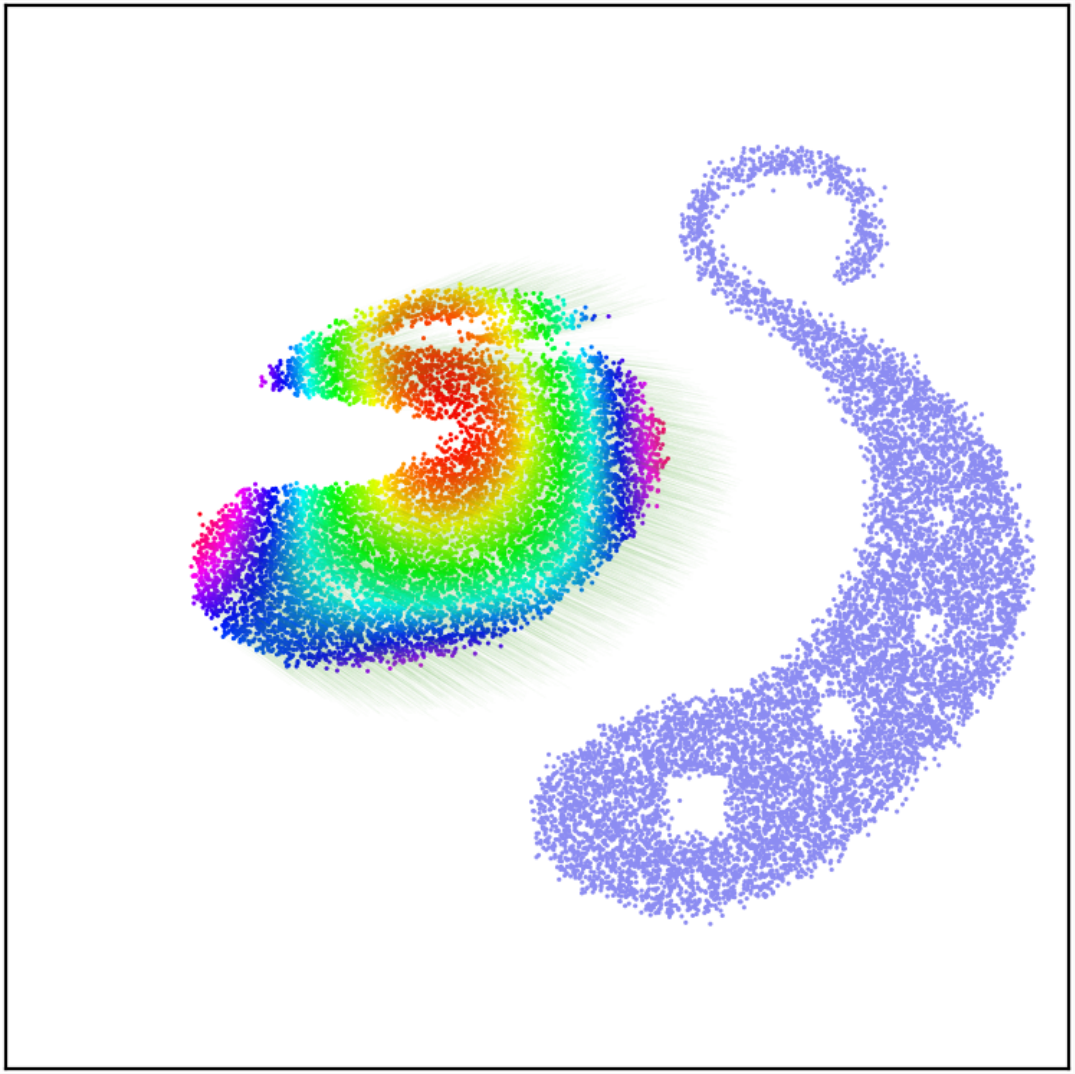
$$\sum_j \pi_{i,j} = \alpha_i, \quad \sum_i \pi_{i,j} = \beta_j.$$

Wasserstein gradients are homogeneous



$t = .00$

Wasserstein gradients are homogeneous

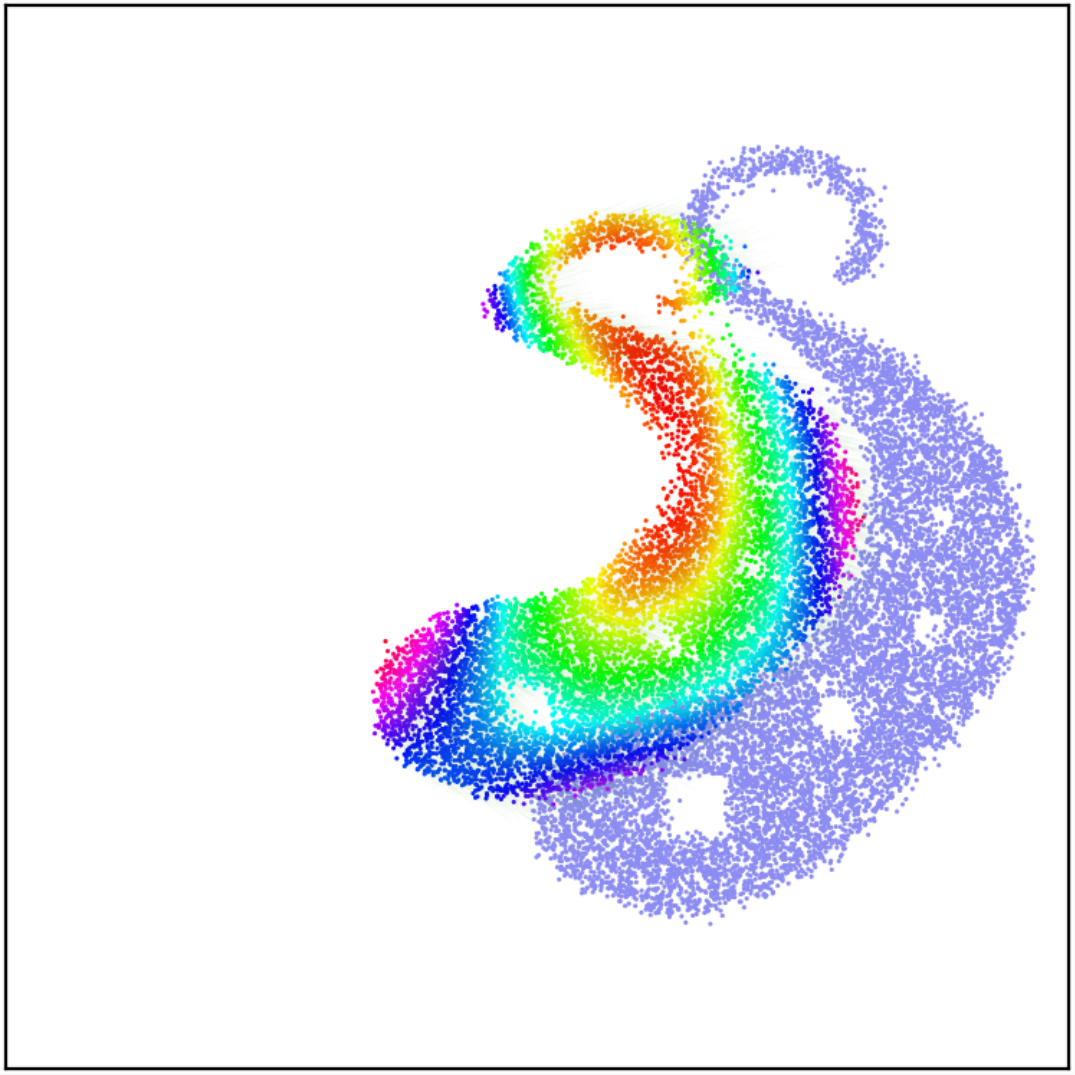


$t = .25$

Wasserstein gradients are homogeneous

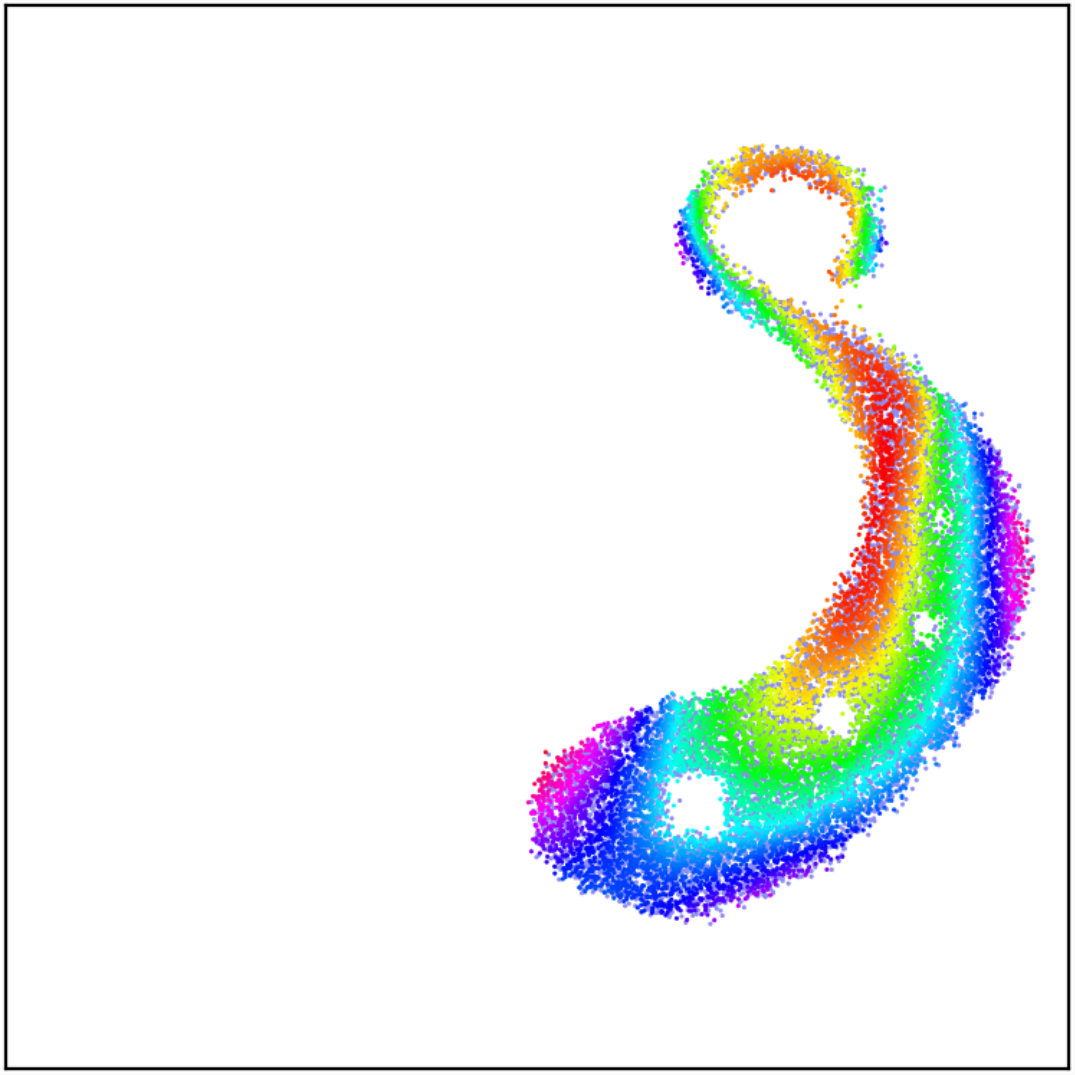
$t = .50$

Wasserstein gradients are homogeneous



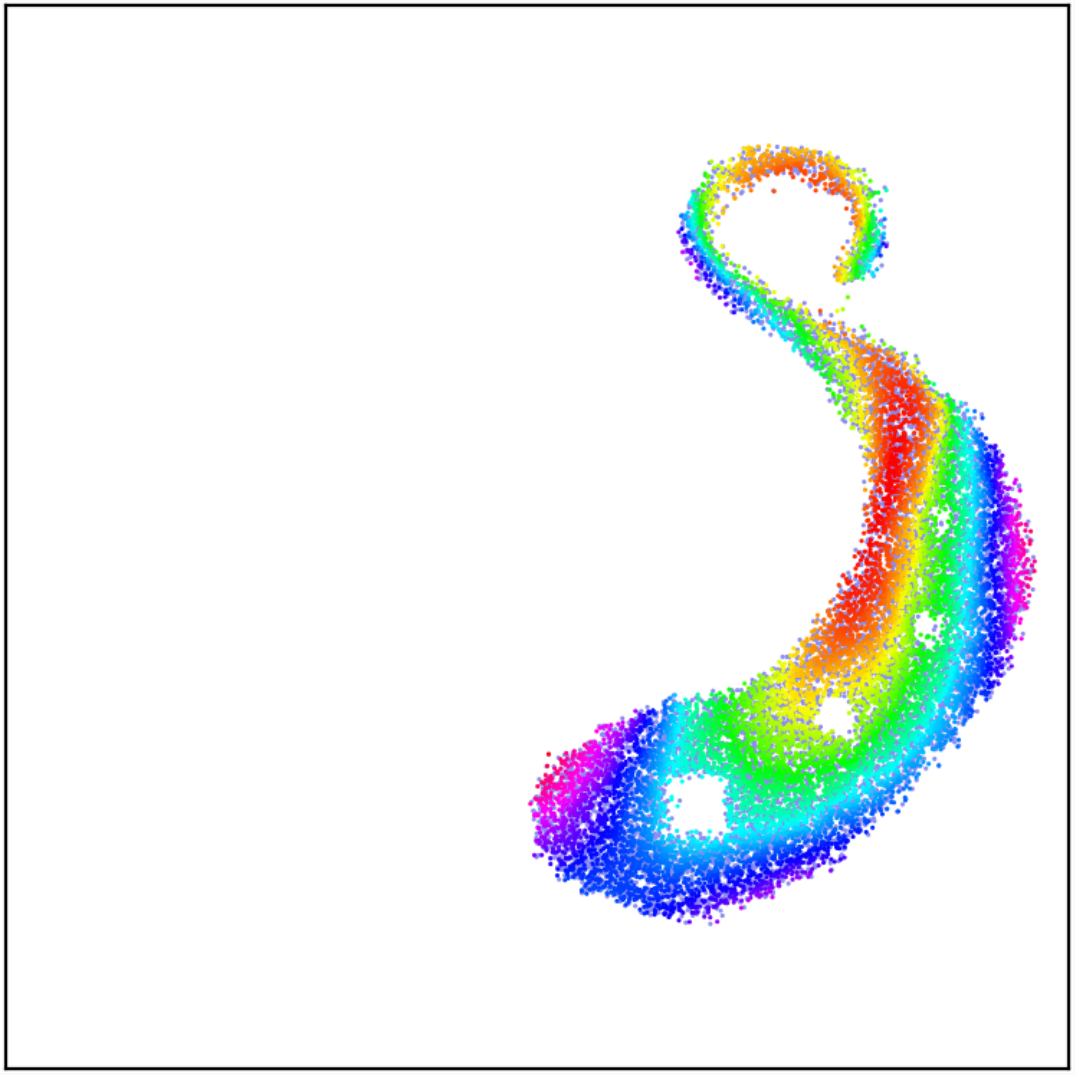
$t = 1.00$

Wasserstein gradients are homogeneous



$t = 5.00$

Wasserstein gradients are homogeneous



$t = 10.00$

Key properties [Bre91]

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

- **Symmetric:** $\mathrm{OT}(\alpha, \beta) = \mathrm{OT}(\beta, \alpha)\,.$

Key properties [Bre91]

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

- **Symmetric:** $\mathrm{OT}(\alpha, \beta) = \mathrm{OT}(\beta, \alpha)\,.$
- **Positive:** $\mathrm{OT}(\alpha, \beta) \geqslant 0\,.$

Key properties [Bre91]

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

- **Symmetric:** $\mathrm{OT}(\alpha, \beta) = \mathrm{OT}(\beta, \alpha)\,.$
- **Positive:** $\mathrm{OT}(\alpha, \beta) \geqslant 0\,.$
- **Definite:** $\mathrm{OT}(\alpha, \beta) = 0 \Longleftrightarrow \alpha = \beta\,.$

Key properties [Bre91]

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

- **Symmetric:** $\mathrm{OT}(\alpha, \beta) = \mathrm{OT}(\beta, \alpha)\,.$
- **Positive:** $\mathrm{OT}(\alpha, \beta) \geqslant 0\,.$
- **Definite:** $\mathrm{OT}(\alpha, \beta) = 0 \iff \alpha = \beta\,.$
- **Translation-aware:** $\mathrm{OT}(\alpha, \mathrm{Translate}_{\vec{v}}(\alpha)\,) = \frac{1}{2}\|\,\vec{v}\,\|^2\,.$

Key properties [Bre91]

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

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

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

Key properties [Bre91]

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

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

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

Key properties [Bre91]

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

- **Symmetric:** $\mathrm{OT}(\alpha, \beta) = \mathrm{OT}(\beta, \alpha)\,.$
- **Positive:** $\mathrm{OT}(\alpha, \beta) \geqslant 0\,.$
- **Definite:** $\mathrm{OT}(\alpha, \beta) = 0 \iff \alpha = \beta\,.$
- **Translation-aware:** $\mathrm{OT}(\alpha, \mathrm{Translate}_{\vec{v}}(\alpha)\,) = \frac{1}{2}\|\,\vec{v}\,\|^2\,.$
- 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:

- [Kan42]: **Dual** problem.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem.
- [Kuh55]: **Hungarian** method in $O(N^3)$.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.

How should we solve the OT problem?

Key dates:

- [Kan42]: **Dual** problem.
- [Kuh55]: **Hungarian** method in $O(N^3)$.
- [Ber79]: **Auction** algorithm in $O(N^2)$.
- [KY94]: **SoftAssign** = Sinkhorn + annealing, in $O(N^2)$.

How should we solve the OT problem?

Key dates:

- [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.

How should we solve the OT problem?

Key dates:

- [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**.

How should we solve the OT problem?

Key dates:

- [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** CPU solvers in $O(N \log N)$.

How should we solve the OT problem?

Key dates:

- [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** CPU solvers in $O(N \log N)$.
- Today: **Multiscale Sinkhorn algorithm, on the GPU**.

How should we solve the OT problem?

Key dates:

- [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** CPU solvers in $O(N \log N)$.
- Today: **Multiscale Sinkhorn algorithm, on the GPU**.

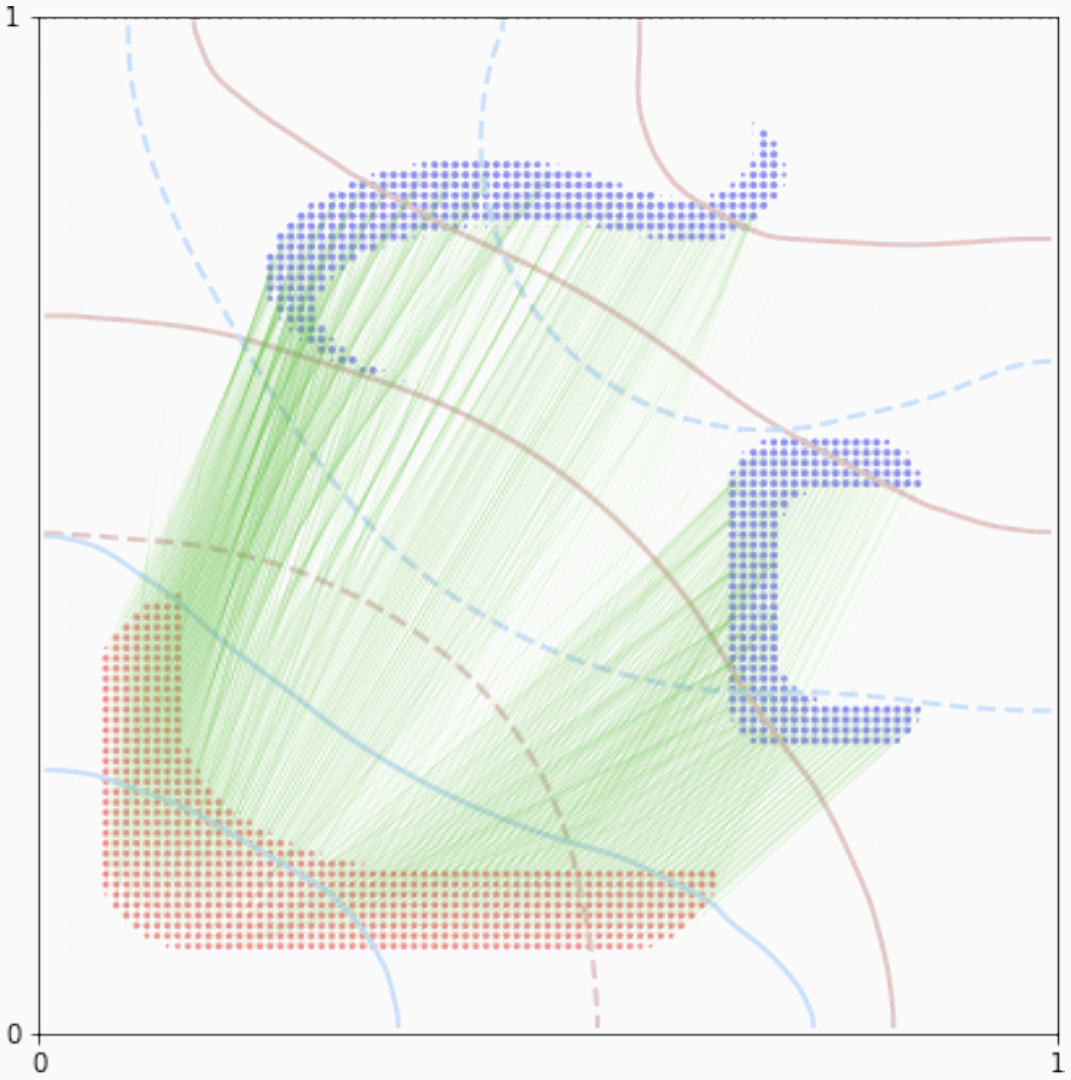
How should we solve the OT problem?

Key dates:

- [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** CPU solvers in $O(N \log N)$.
- Today: **Multiscale Sinkhorn algorithm, on the GPU**.

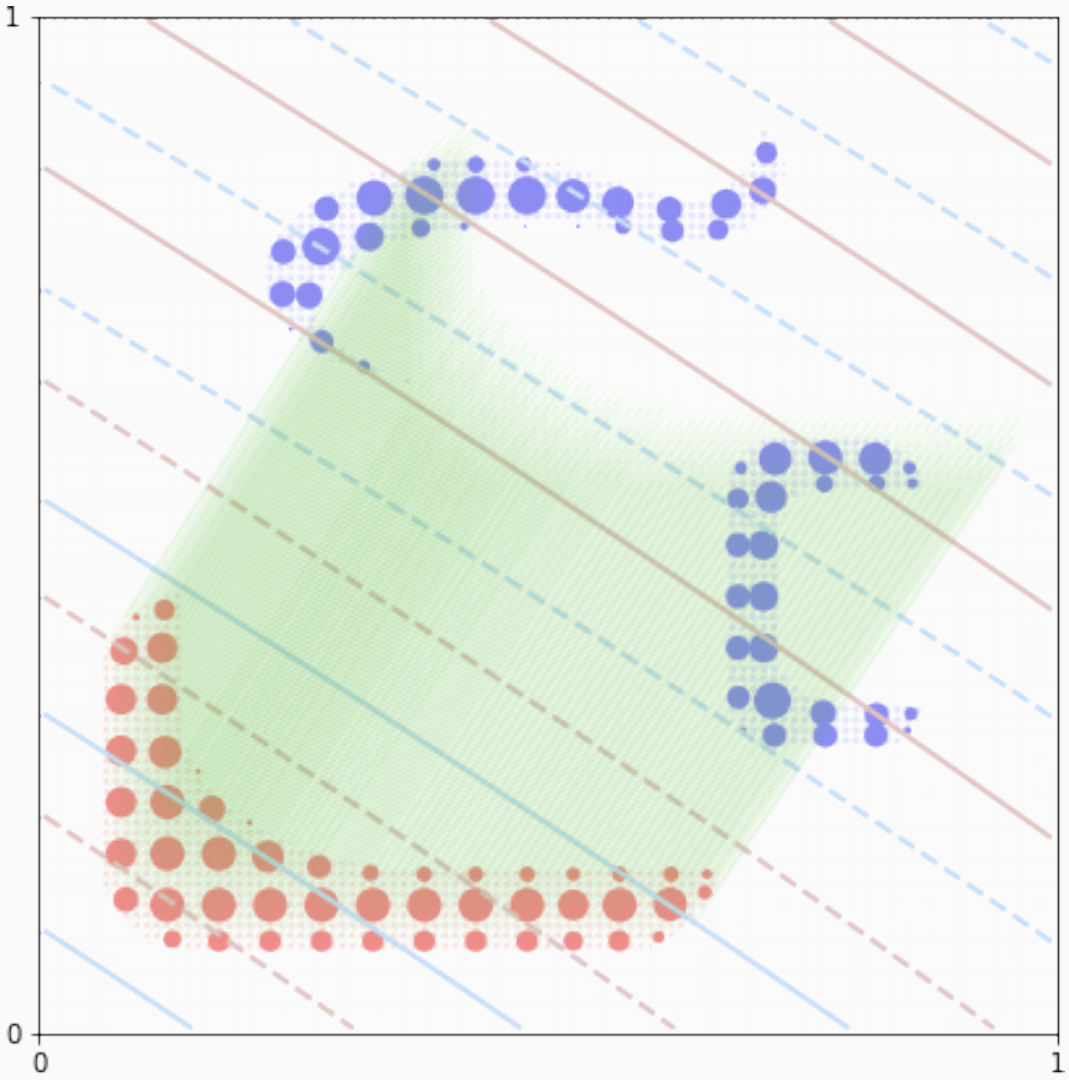
$\Longrightarrow$ Generalized **QuickSort** algorithm.

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



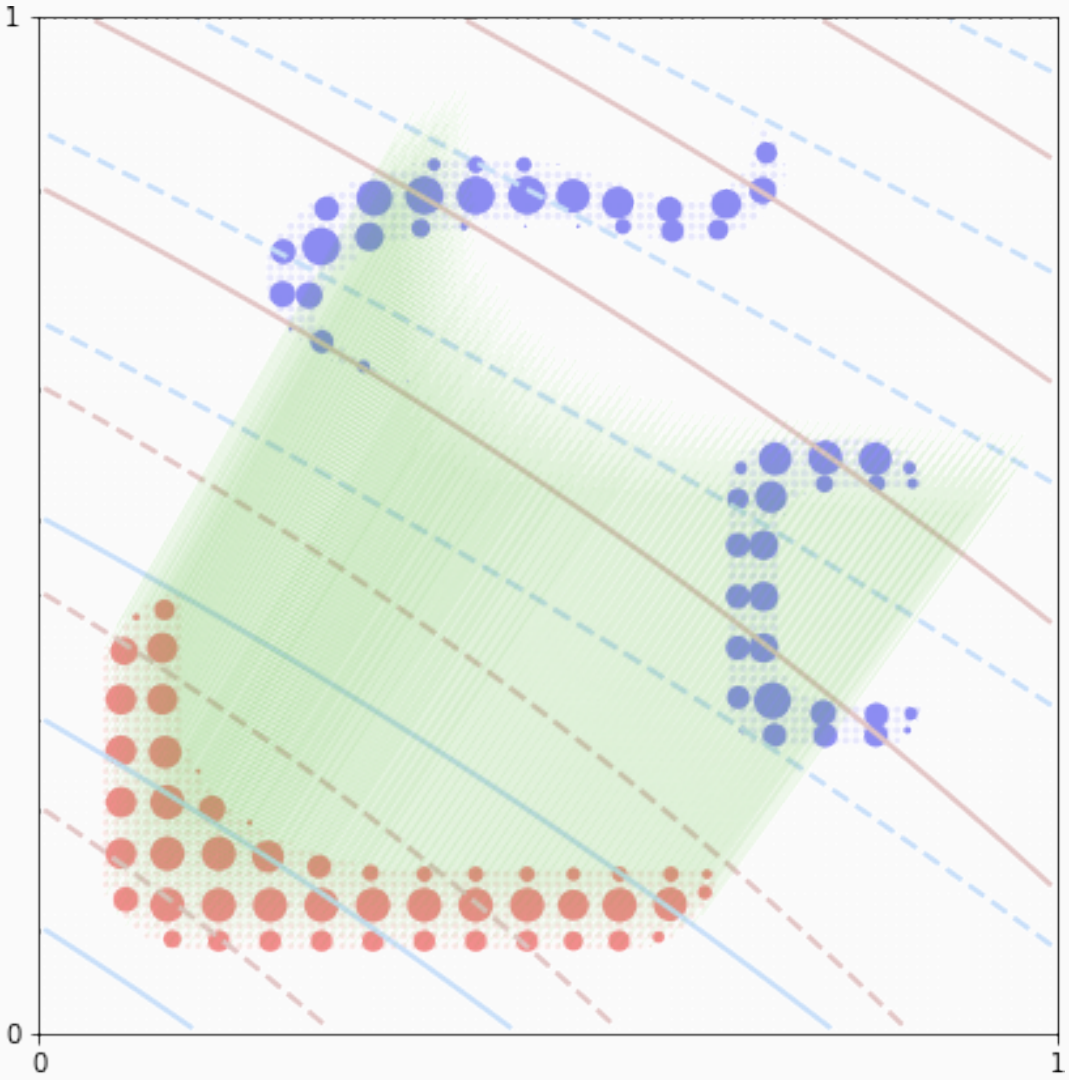
OT plan in 2D.

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



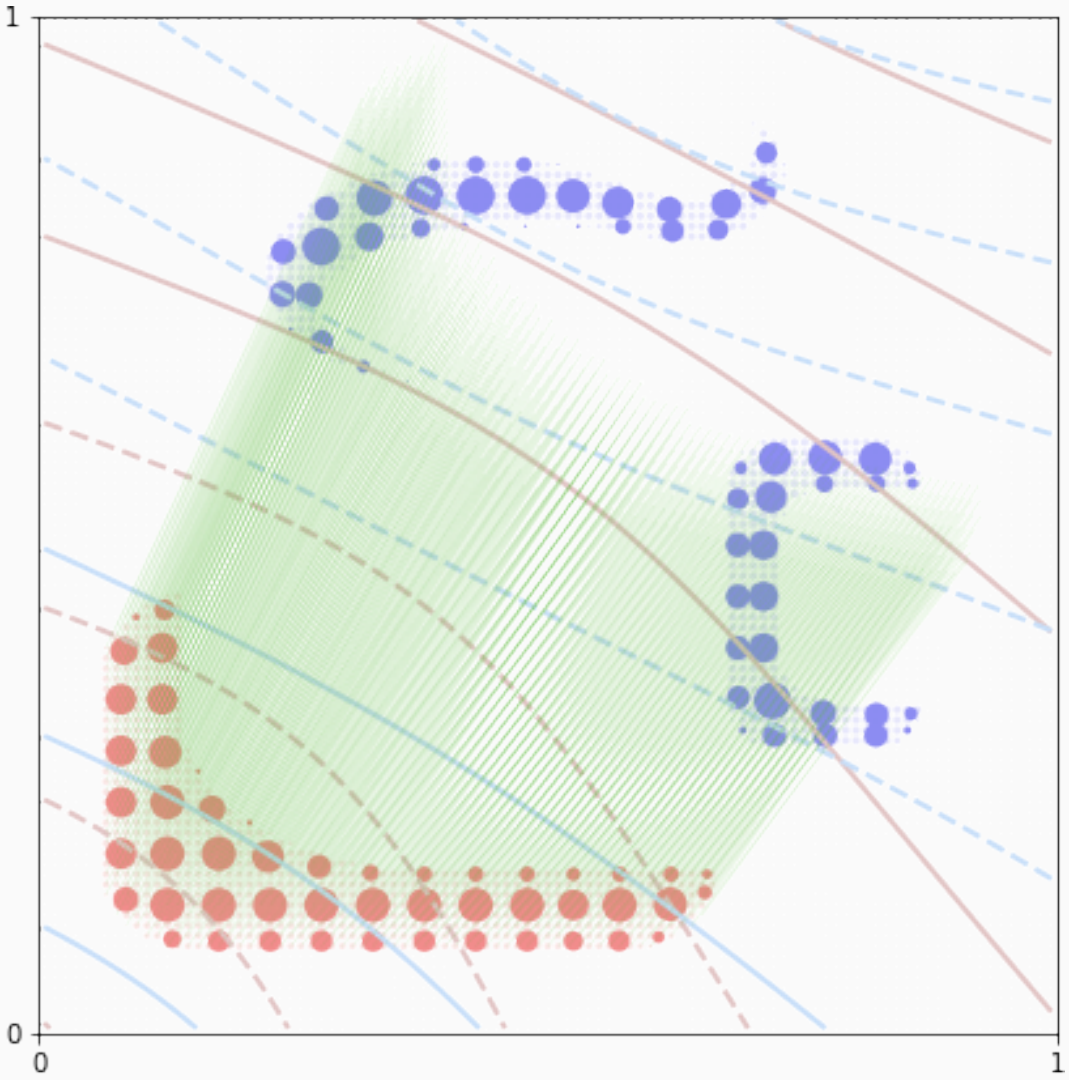
Iteration 0, blur = 2^0

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



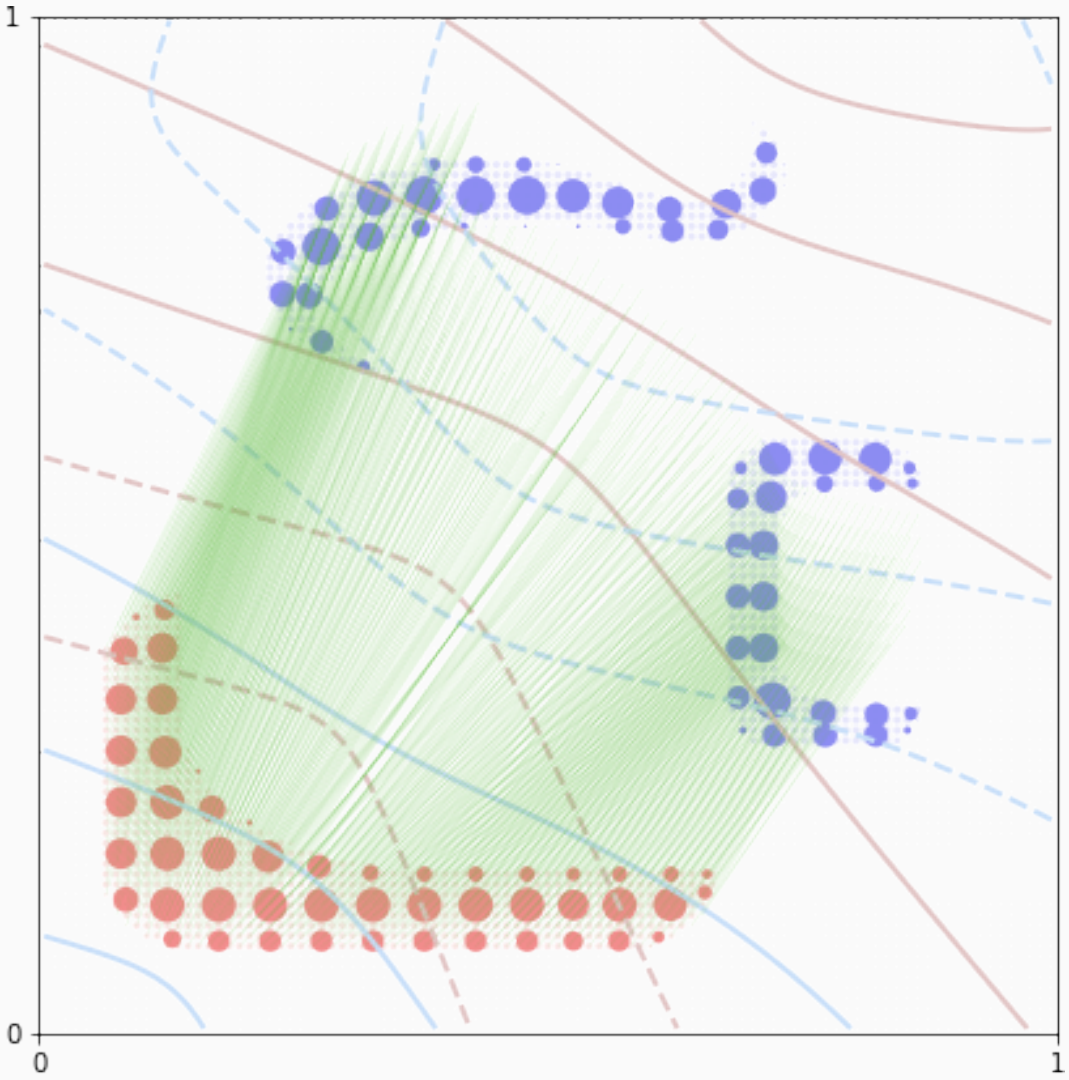
Iteration 1, blur = 2^{-1}

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



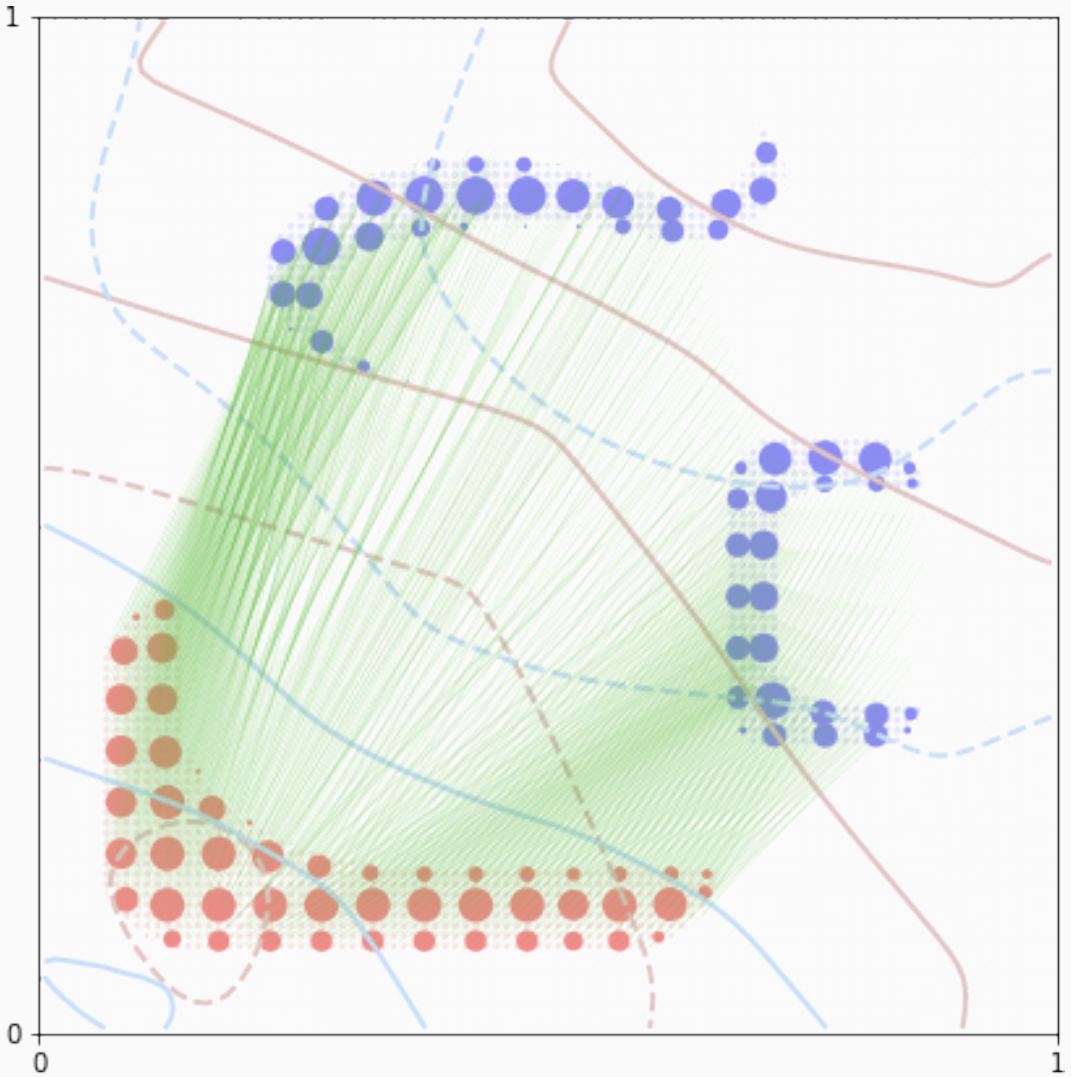
Iteration 2, blur = 2^{-2}

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



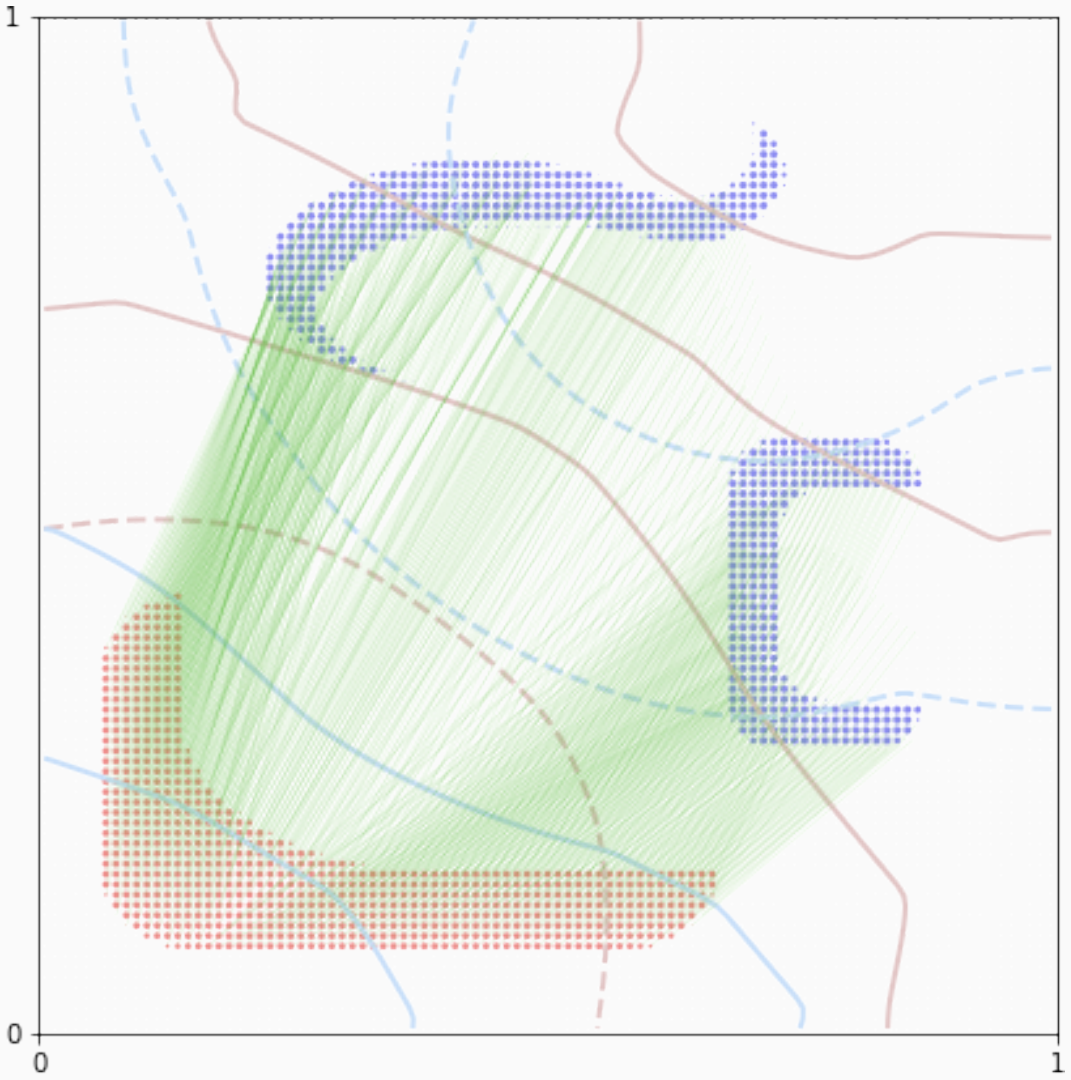
Iteration 3, blur = 2^{-3}

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



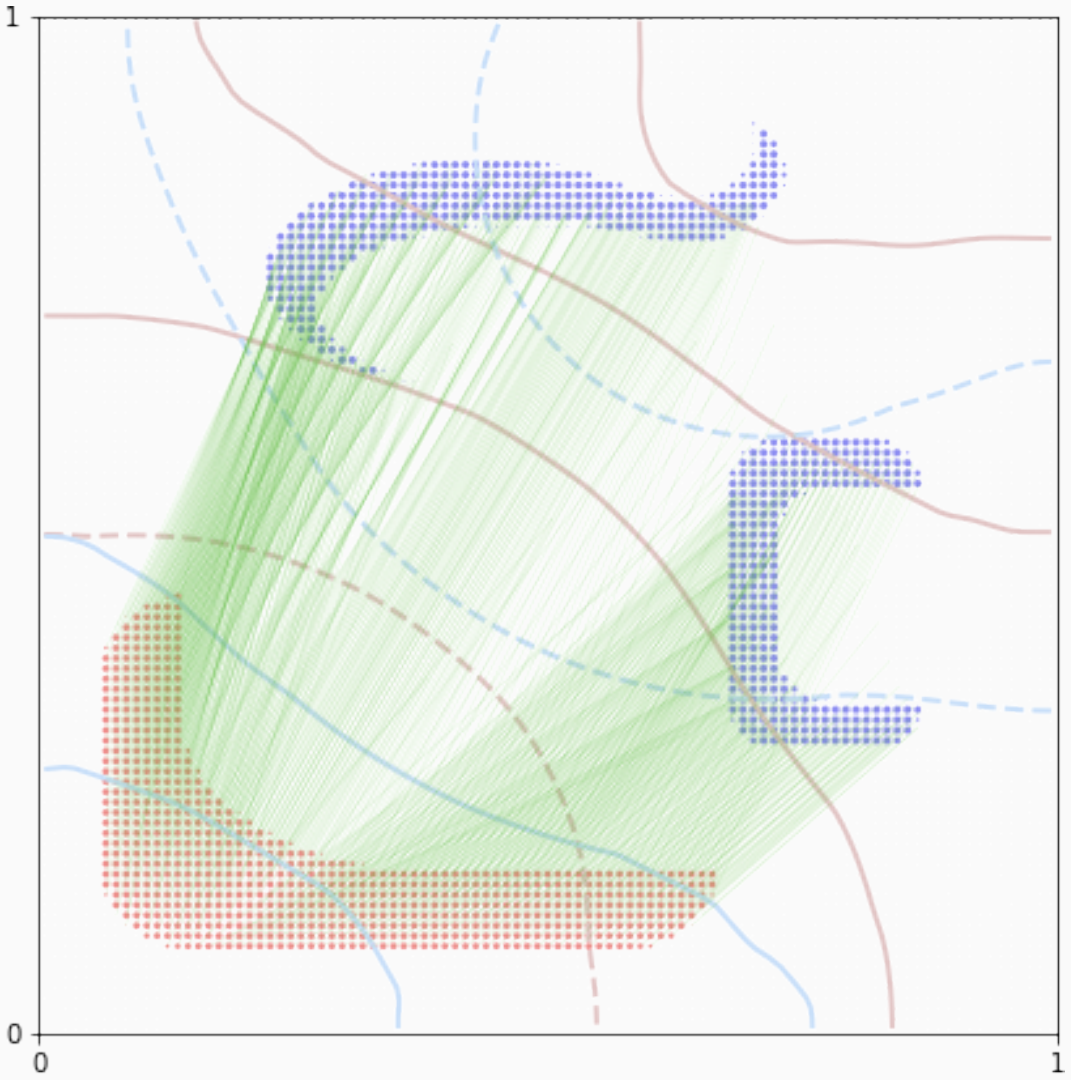
Iteration 4, blur = 2^{-4}

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



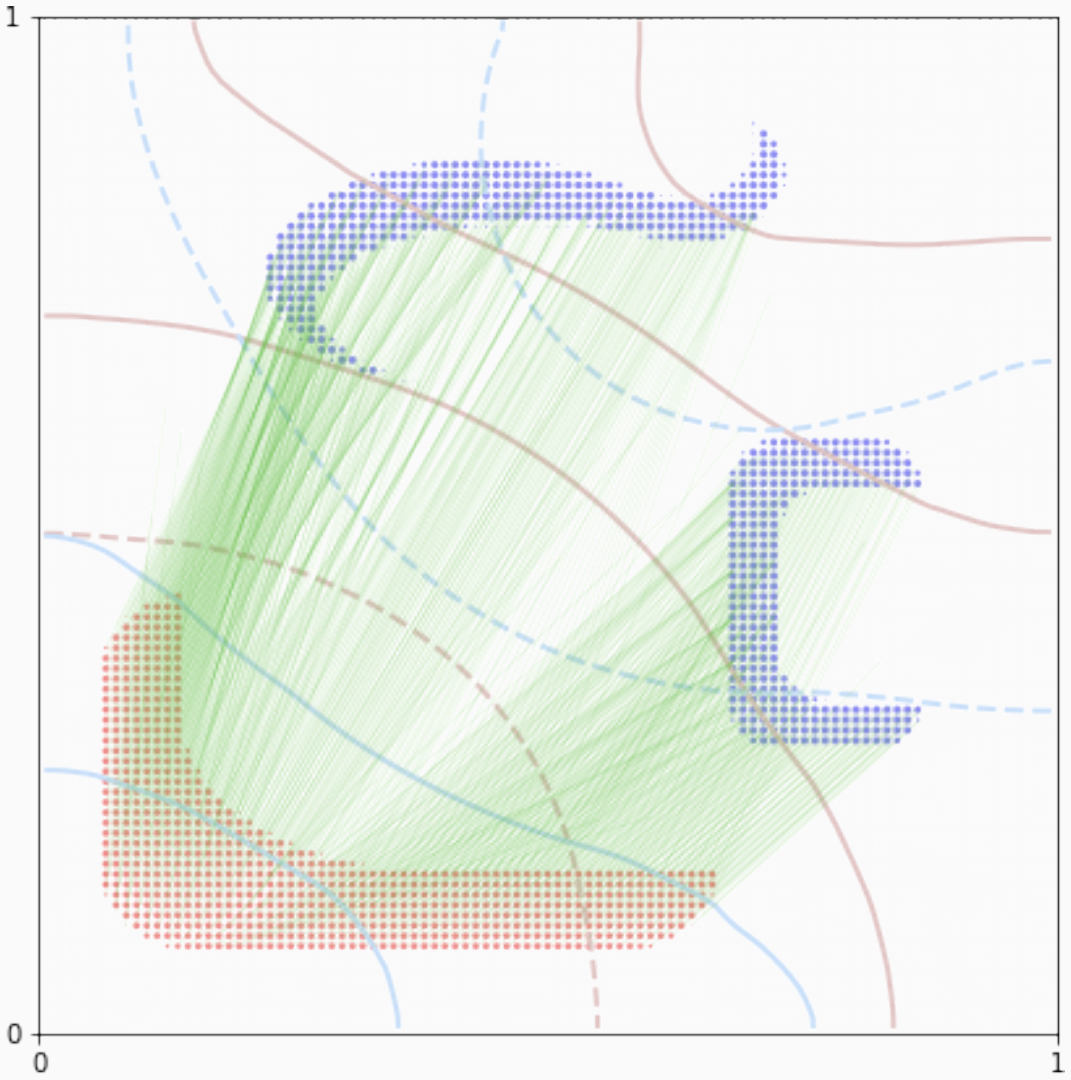
Iteration 5, blur = 2^{-5}

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



Iteration 6, blur = 2^{-6}

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 = `.01`

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.

Overview of the last decade

Progresses since [KY94, Cut13]:

- $+$ $O(N \log N)$ instead of $O(N^2)$.
- $+$ Converge in $\leqslant 10$ iterations.

Overview of the last decade

Progresses since [KY94, Cut13]:

- $+$ $O(N \log N)$ instead of $O(N^2)$.
- $+$ Converge in $\leqslant 10$ iterations.
- $+$ Fully symmetric.

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.
+ Converge in $\leqslant 10$ iterations.
+ Fully symmetric.
+ **Positive and definite** [FSV+18].

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.
+ Converge in $\leqslant 10$ iterations.
+ Fully symmetric.
+ **Positive and definite** [FSV+18].

− Much harder to implement.

Overview of the last decade

Progresses since [KY94, Cut13]:

+ $O(N \log N)$ instead of $O(N^2)$.
+ Converge in $\leqslant 10$ iterations.
+ Fully symmetric.
+ **Positive and definite** [FSV+18].

− Much harder to implement.

Overview of the last decade

Progresses since [KY94, Cut13]:

- $+$ $O(N \log N)$ instead of $O(N^2)$.
- $+$ Converge in $\leqslant 10$ iterations.
- $+$ Fully symmetric.
- $+$ **Positive and definite** [FSV+18].

- $-$ Much harder to implement.

$\Longrightarrow$ Use the **GeomLoss** library!

Geometric Loss functions for PyTorch

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

$\Longrightarrow$ `pip install geomloss` $\Longleftarrow$

Geometric Loss functions for PyTorch

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

$$\Longrightarrow \texttt{pip install geomloss} \Longleftarrow$$

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

Geometric Loss functions for PyTorch

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

$$\Longrightarrow \texttt{pip install geomloss} \Longleftarrow$$

```
# 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  # See also ImagesLoss...
loss = SamplesLoss(loss="sinkhorn", p=2, blur=.05)
```

Geometric Loss functions for PyTorch

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

$$\Longrightarrow \texttt{pip install geomloss} \Longleftarrow$$

```
# 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  # See also ImagesLoss...
loss = SamplesLoss(loss="sinkhorn", p=2, blur=.05)

L = loss(x, y)  # By default, use constant weights
```

Geometric Loss functions for PyTorch

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

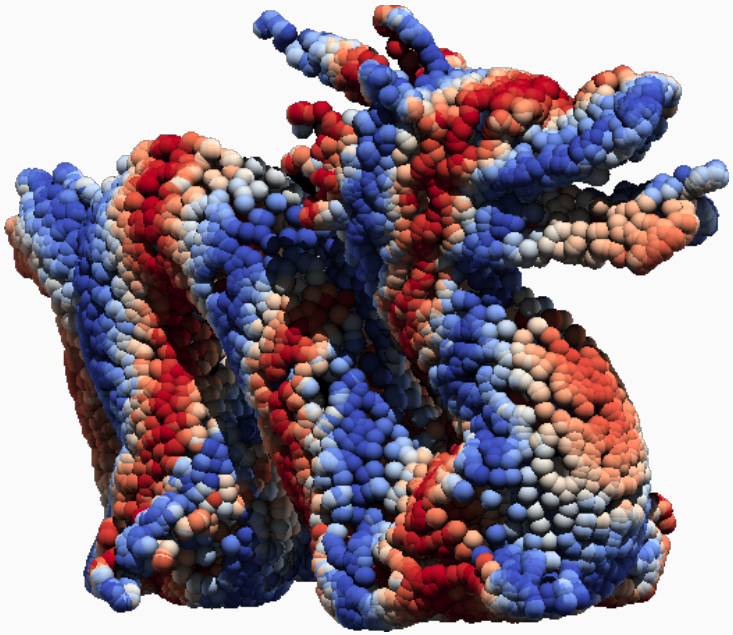
$$\Longrightarrow \texttt{pip install geomloss} \Longleftarrow$$

```
# 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  # See also ImagesLoss...
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])
```

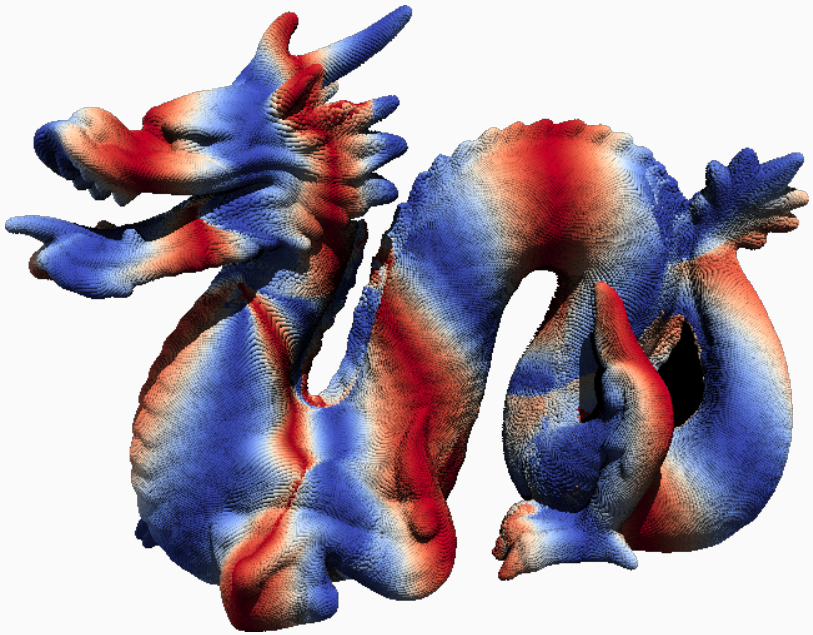
Scaling up optimal transport

Precision controlled by the ratio $\frac{\text{blur}}{\text{diameter}}$.

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



10k points in 30-50ms



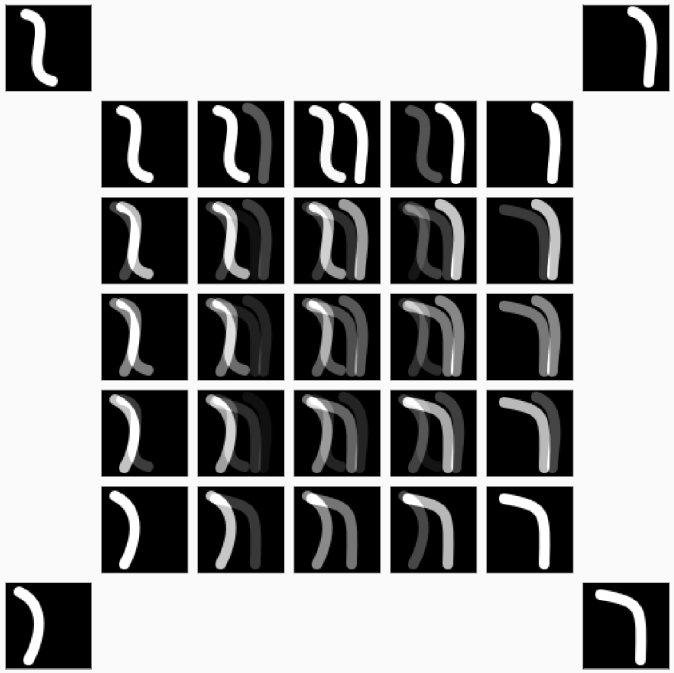
100k points in 100-200ms

Affordable geometric barycenters

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

Affordable geometric barycenters

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

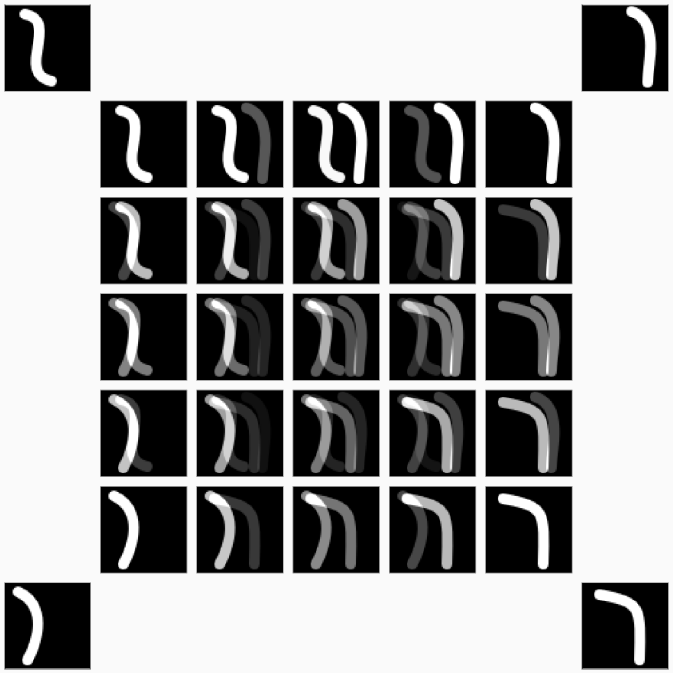


Linear barycenters

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

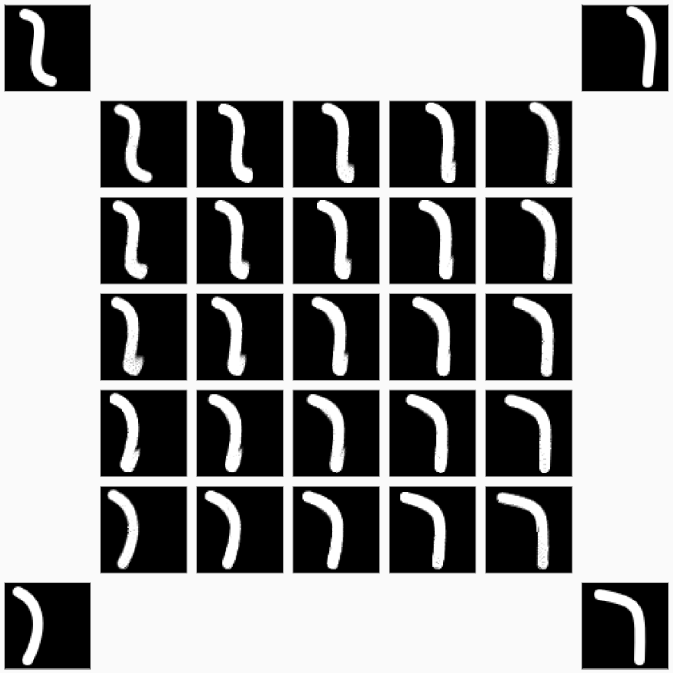
Affordable geometric barycenters

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



Linear barycenters

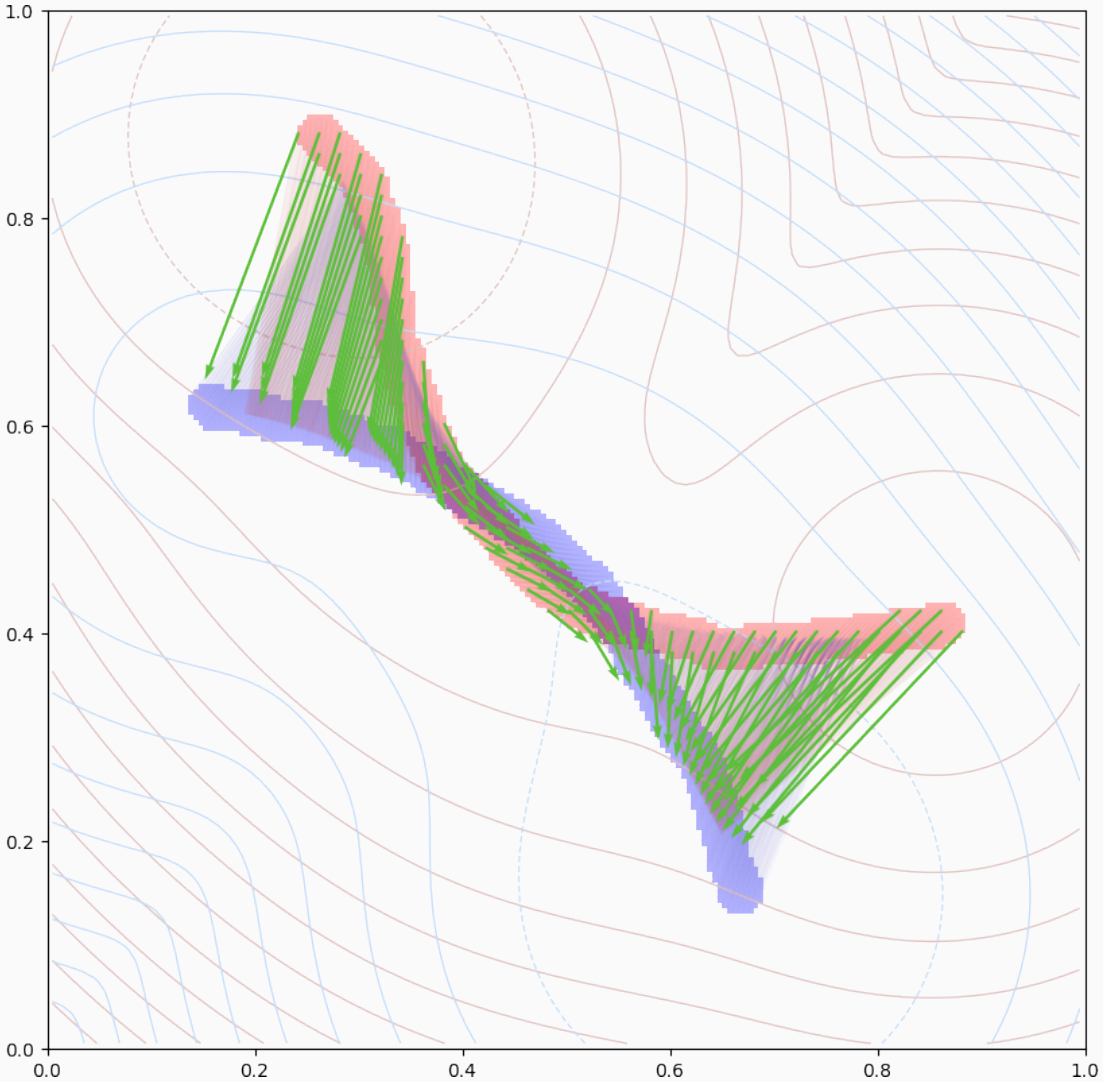
$\text{Loss}(\alpha, \beta) = \|\alpha - \beta\|_{L^2}^2$



Wasserstein barycenters

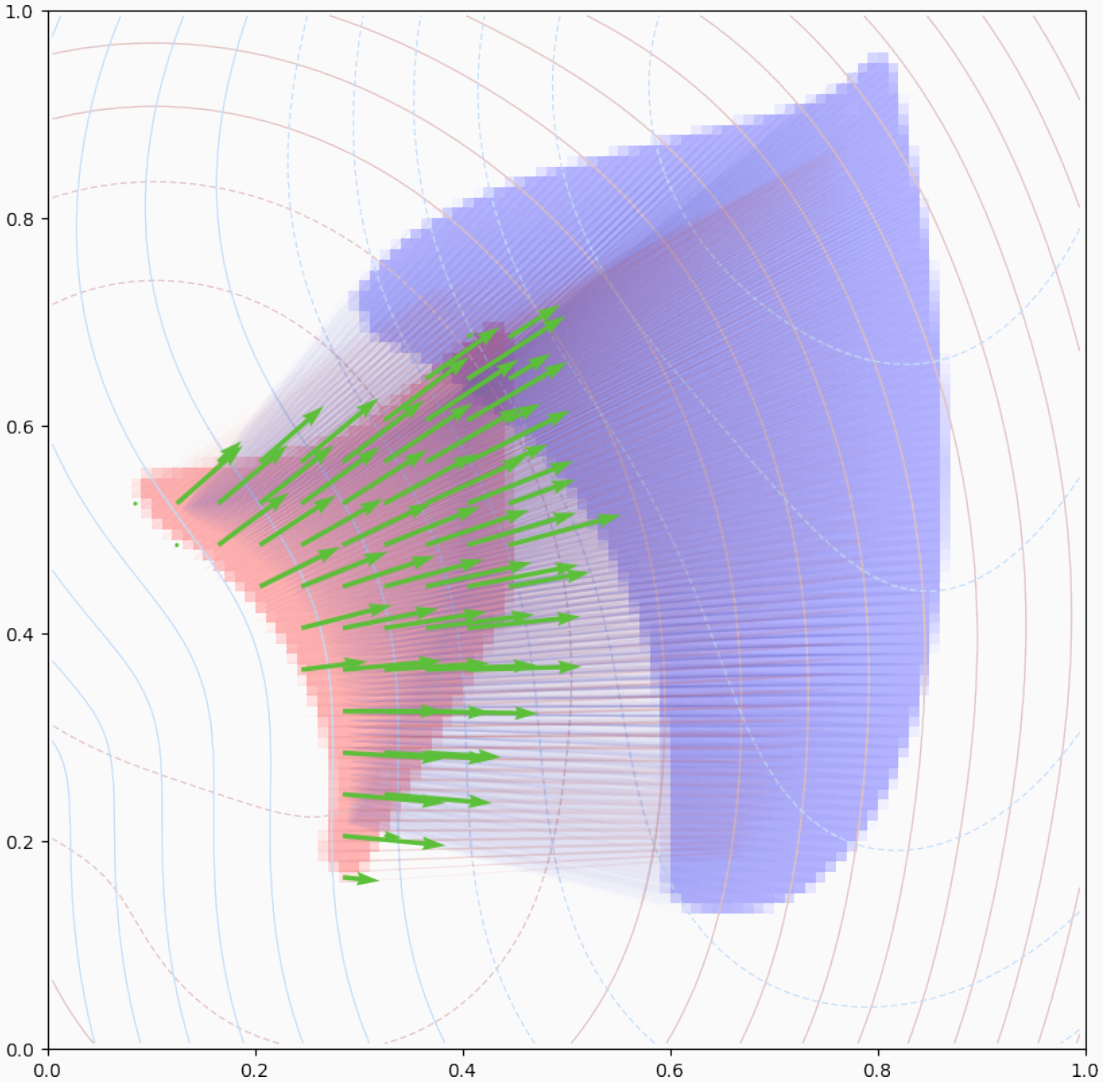
$\text{Loss}(\alpha, \beta) = \text{OT}(\alpha, \beta)$

A robust loss function



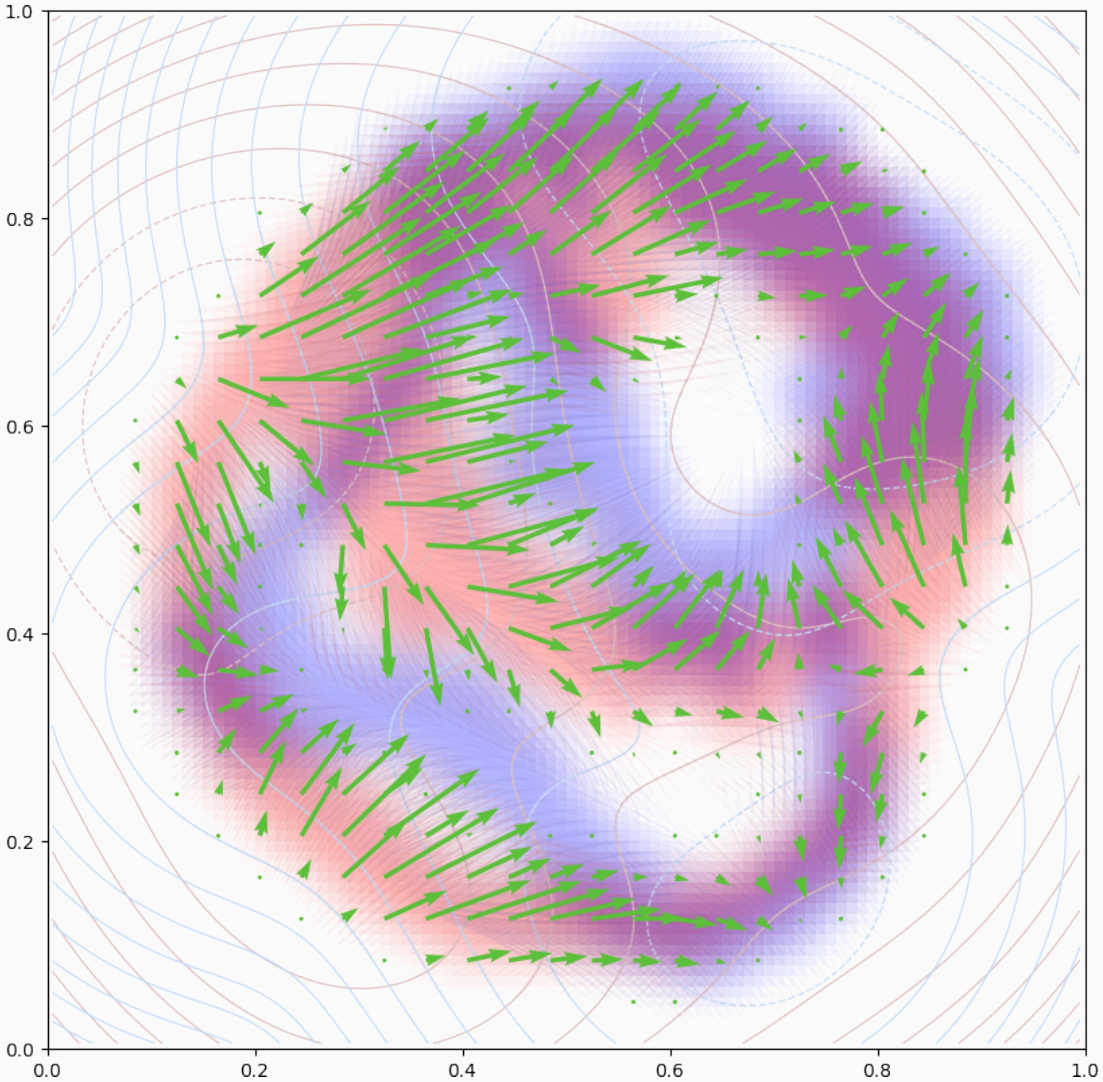
A high-quality gradient...

A robust loss function



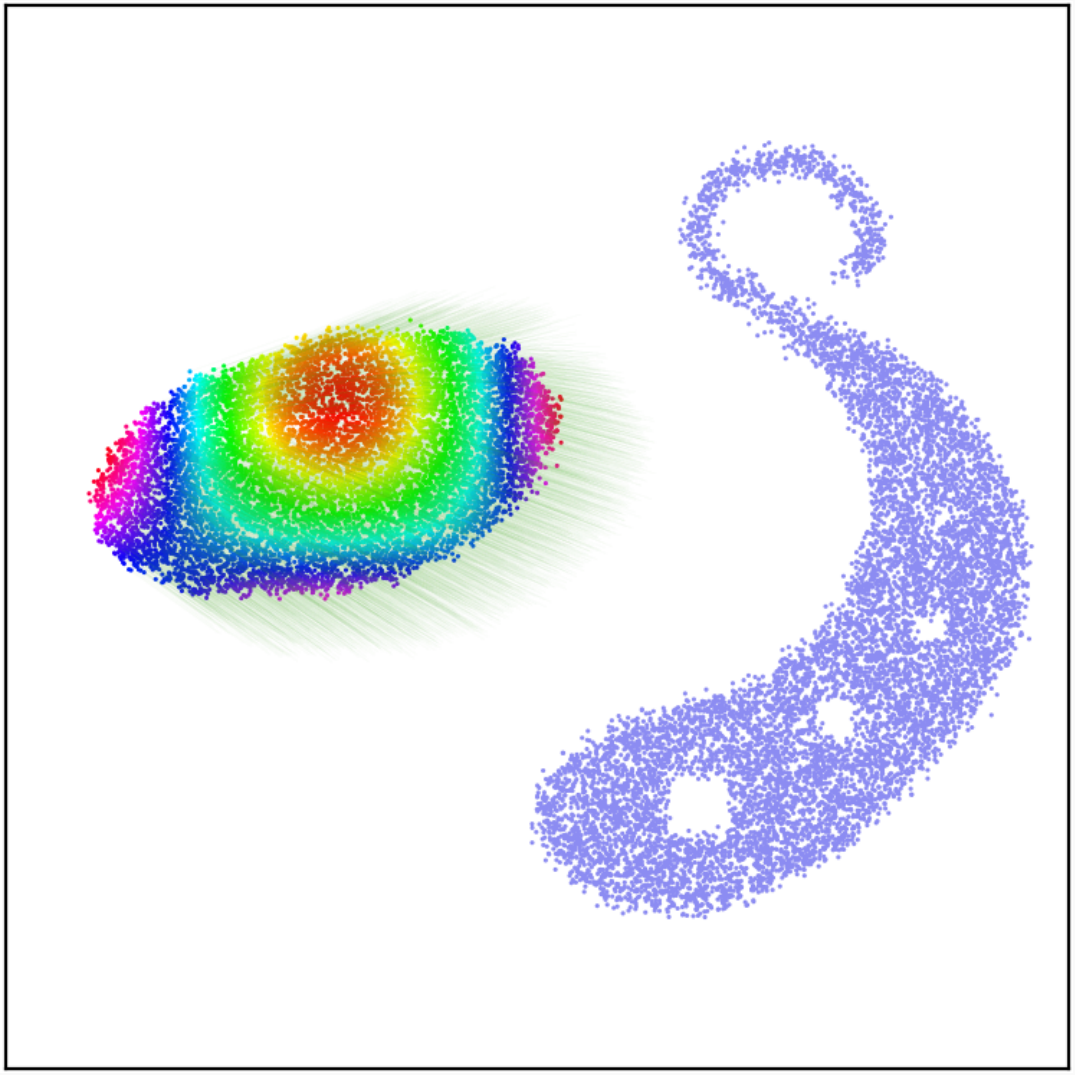
A high-quality gradient…

A robust loss function



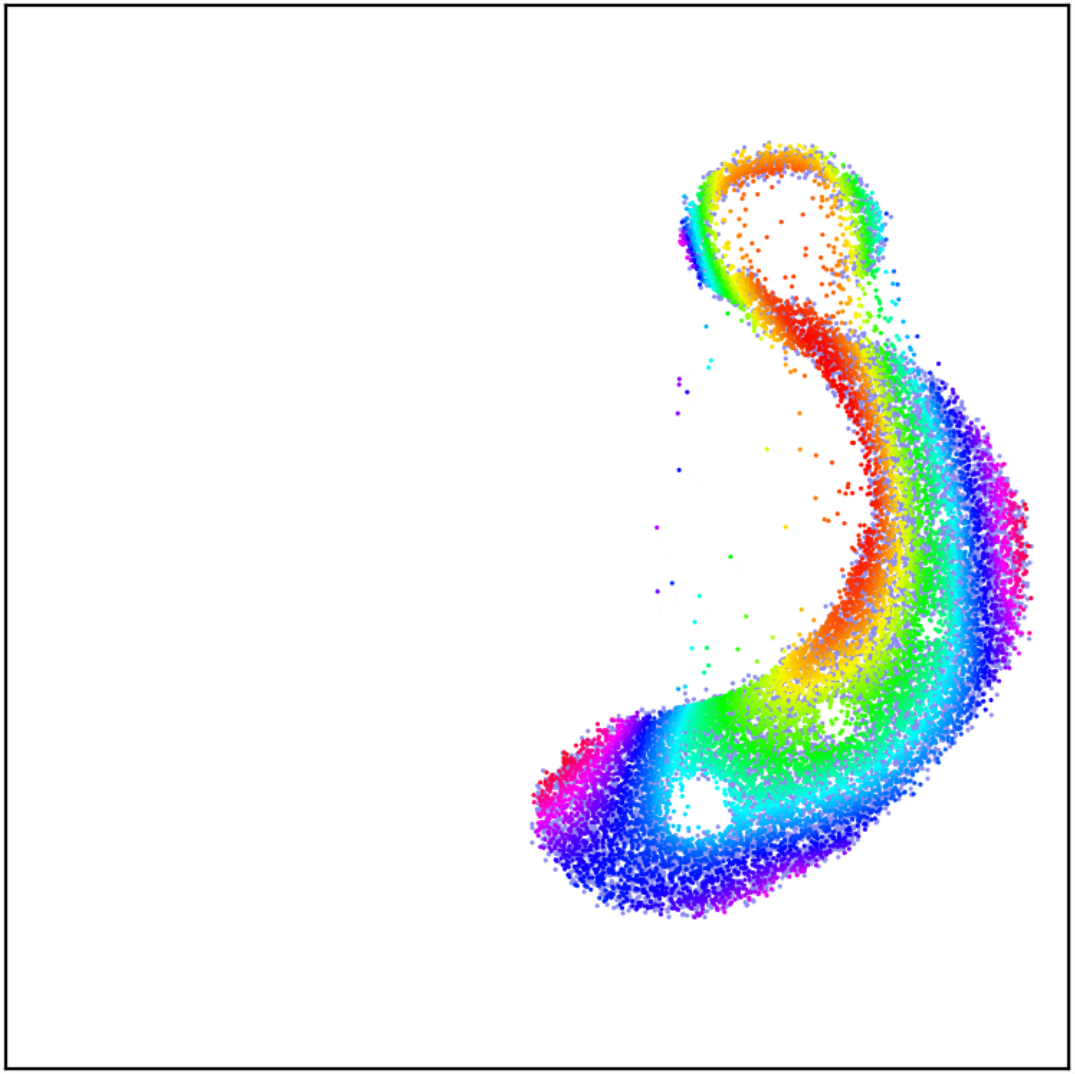
A high-quality gradient... But no preservation of topology!

Gradient descent with OT: cheap'n easy registration?



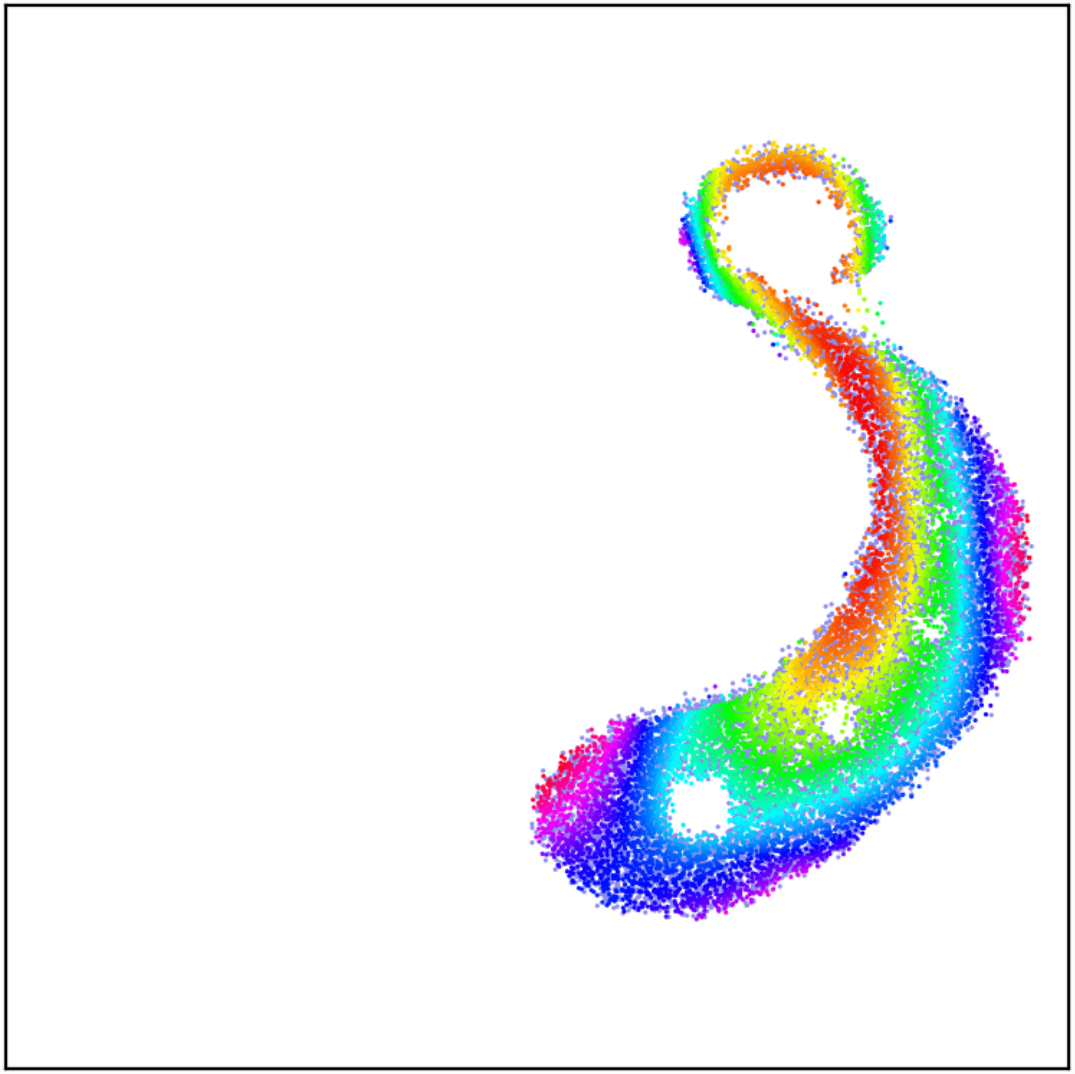
Iteration 0

Gradient descent with OT: cheap'n easy registration?



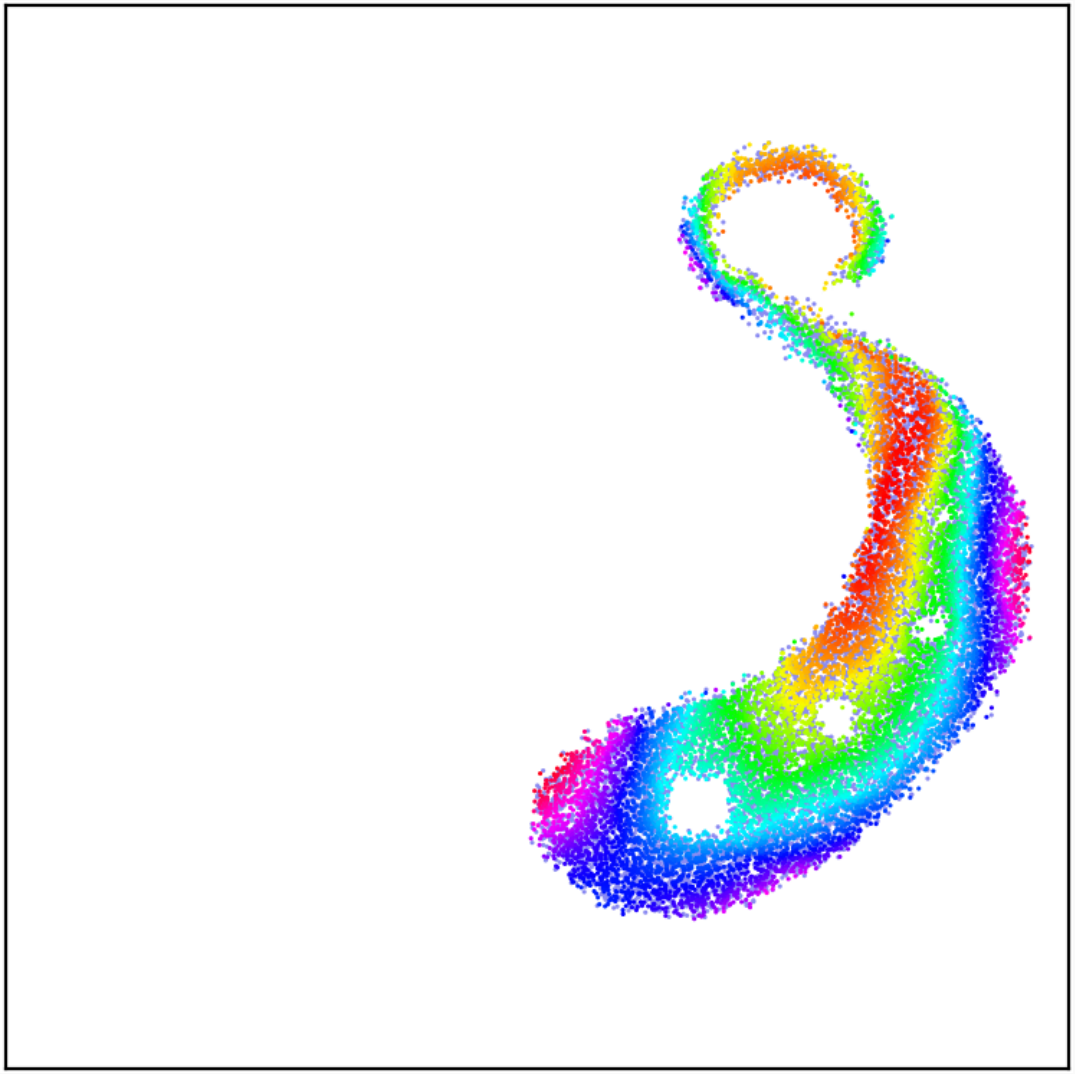
Iteration 1

Gradient descent with OT: cheap'n easy registration?



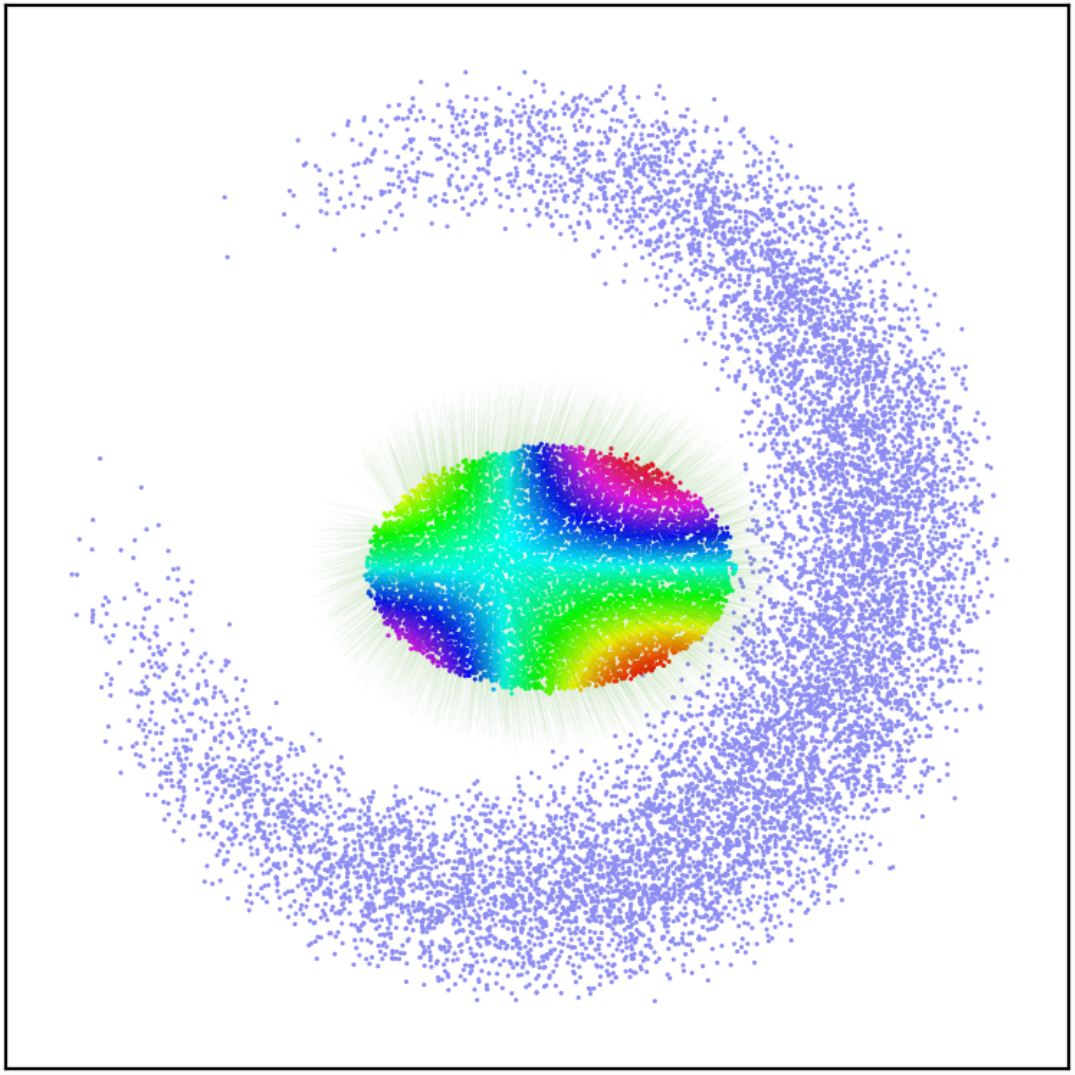
Iteration 2

Gradient descent with OT: cheap'n easy registration?



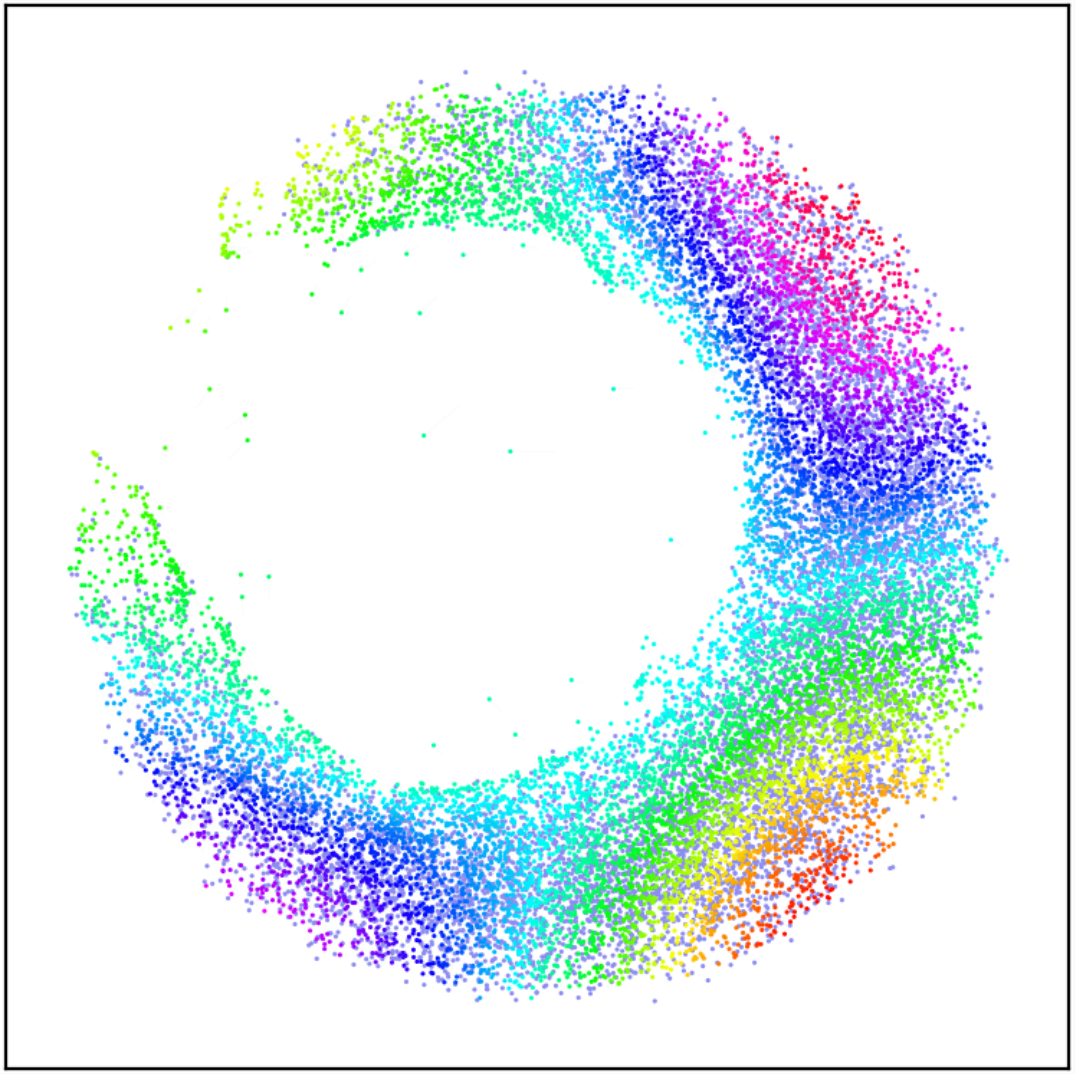
Iteration 10

Gradient descent with OT: cheap'n easy registration?



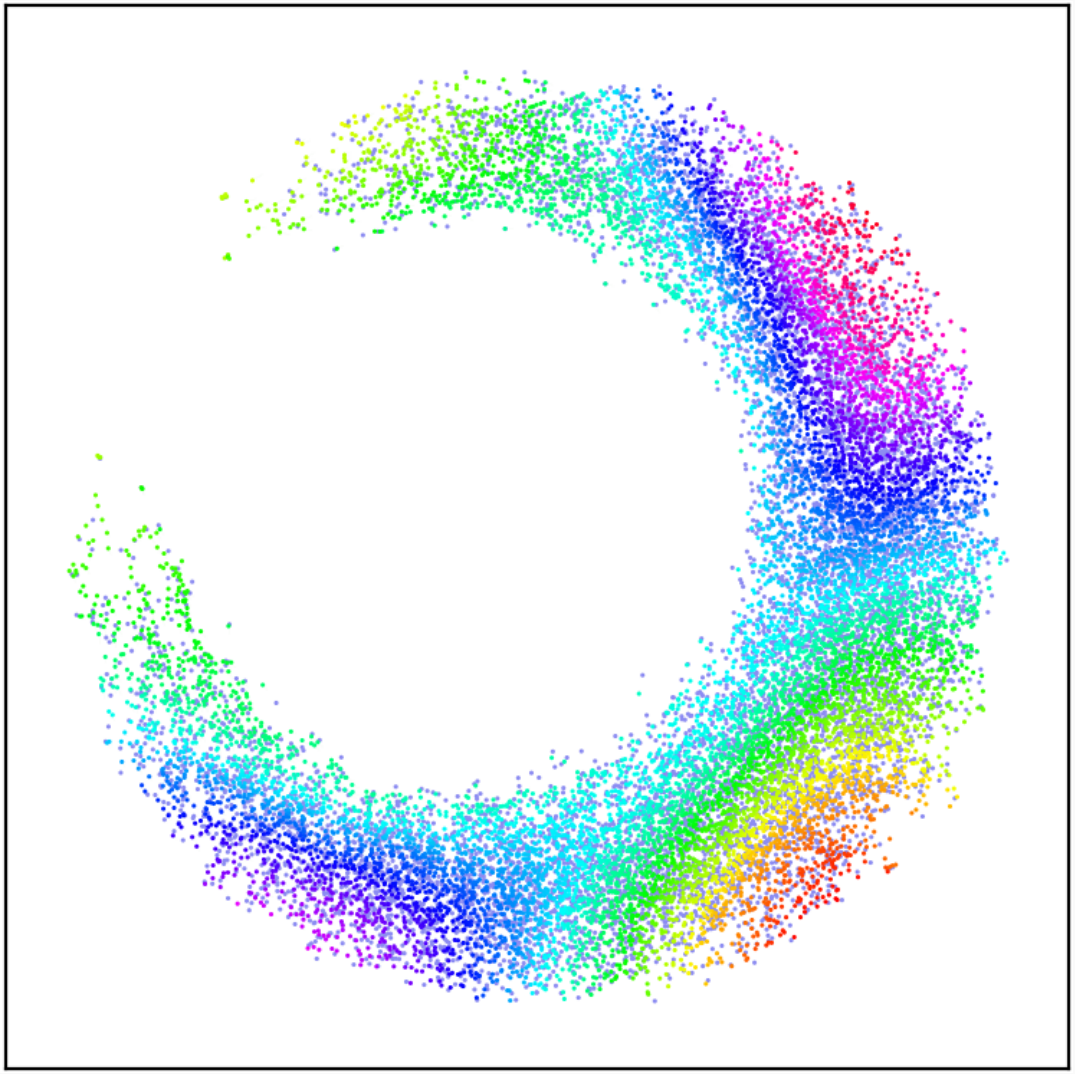
Iteration 0

Gradient descent with OT: cheap'n easy registration?



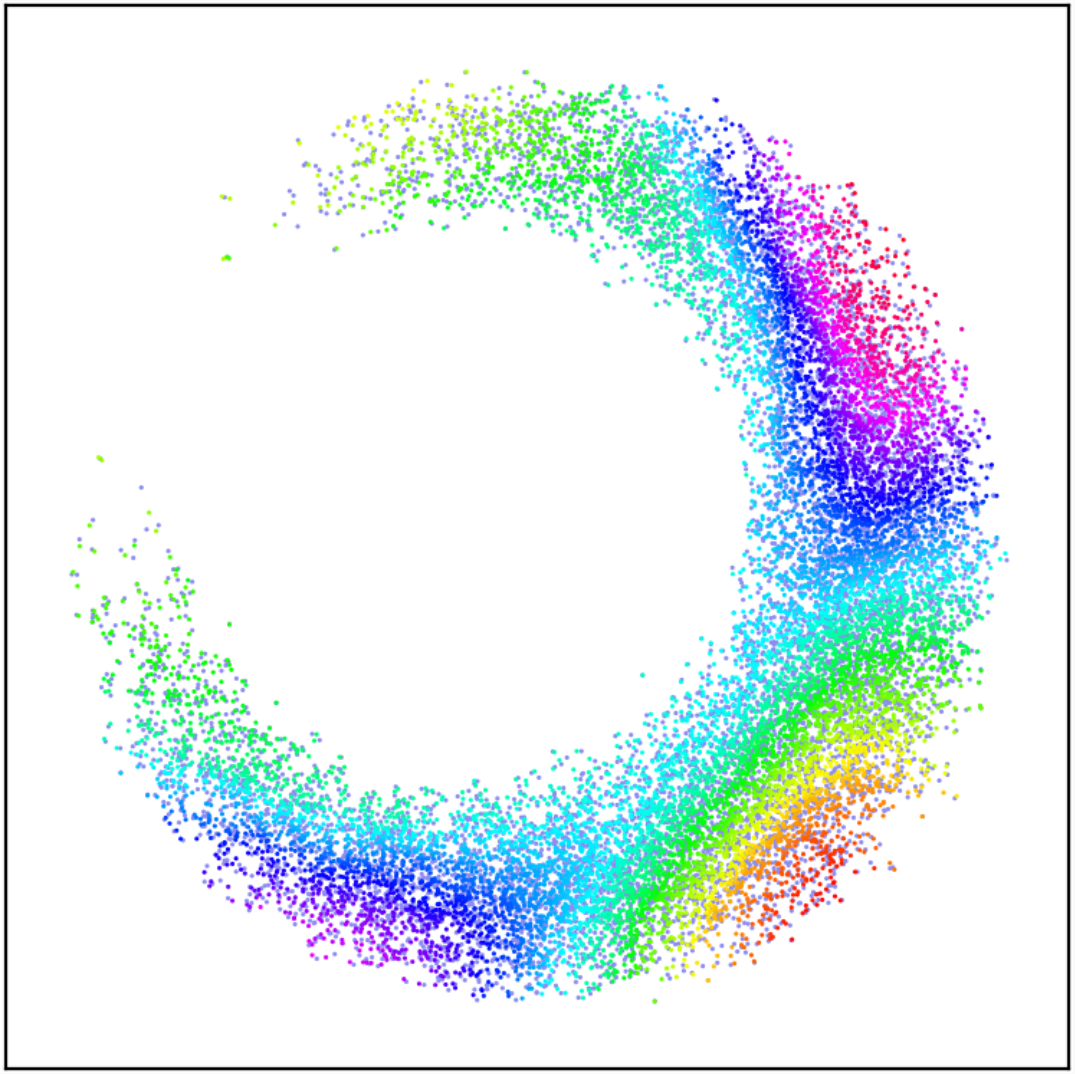
Iteration 1

Gradient descent with OT: cheap'n easy registration?



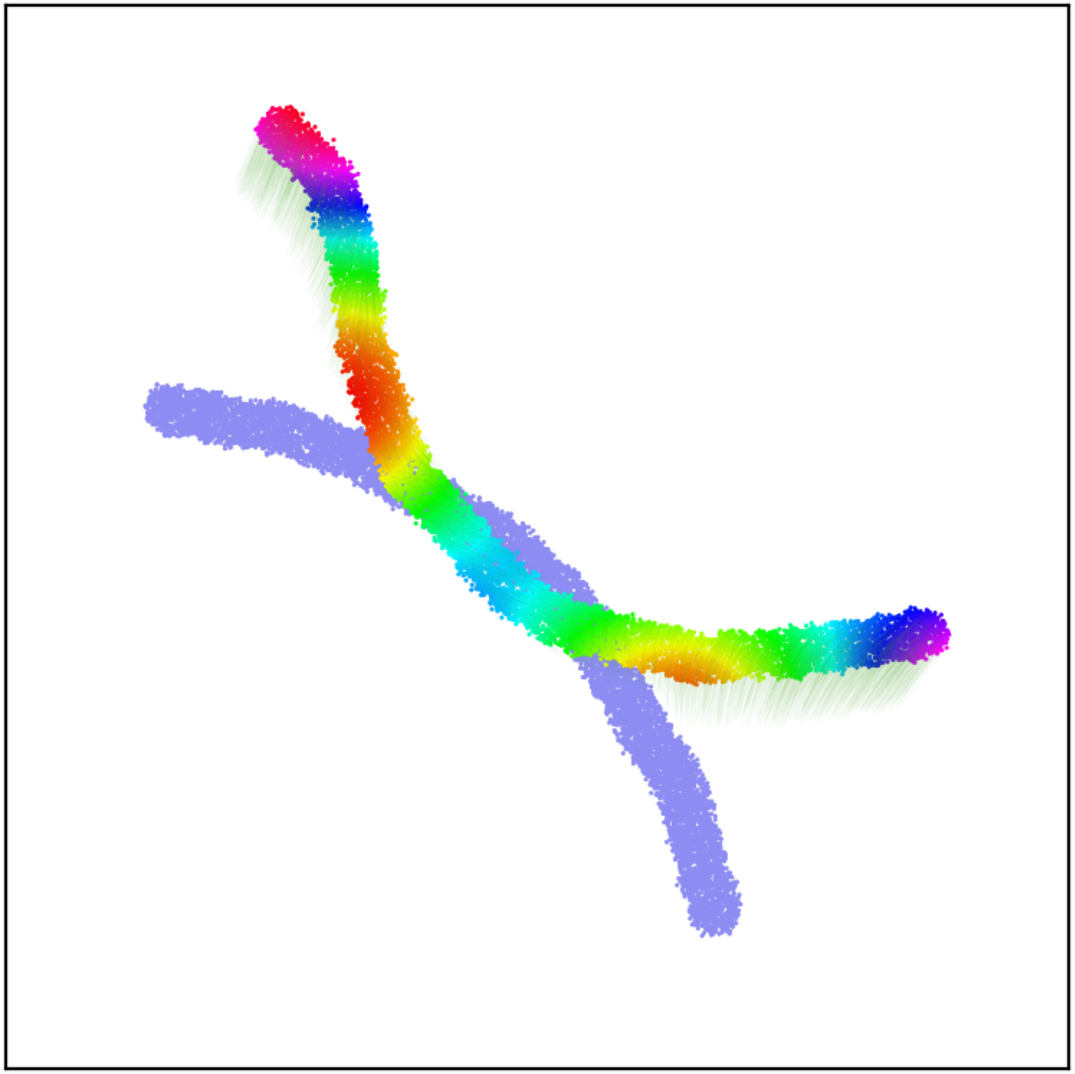
Iteration 2

Gradient descent with OT: cheap'n easy registration?



Iteration 10

Gradient descent with OT: cheap'n easy registration?



Iteration 0

Gradient descent with OT: cheap'n easy registration?

Iteration 1

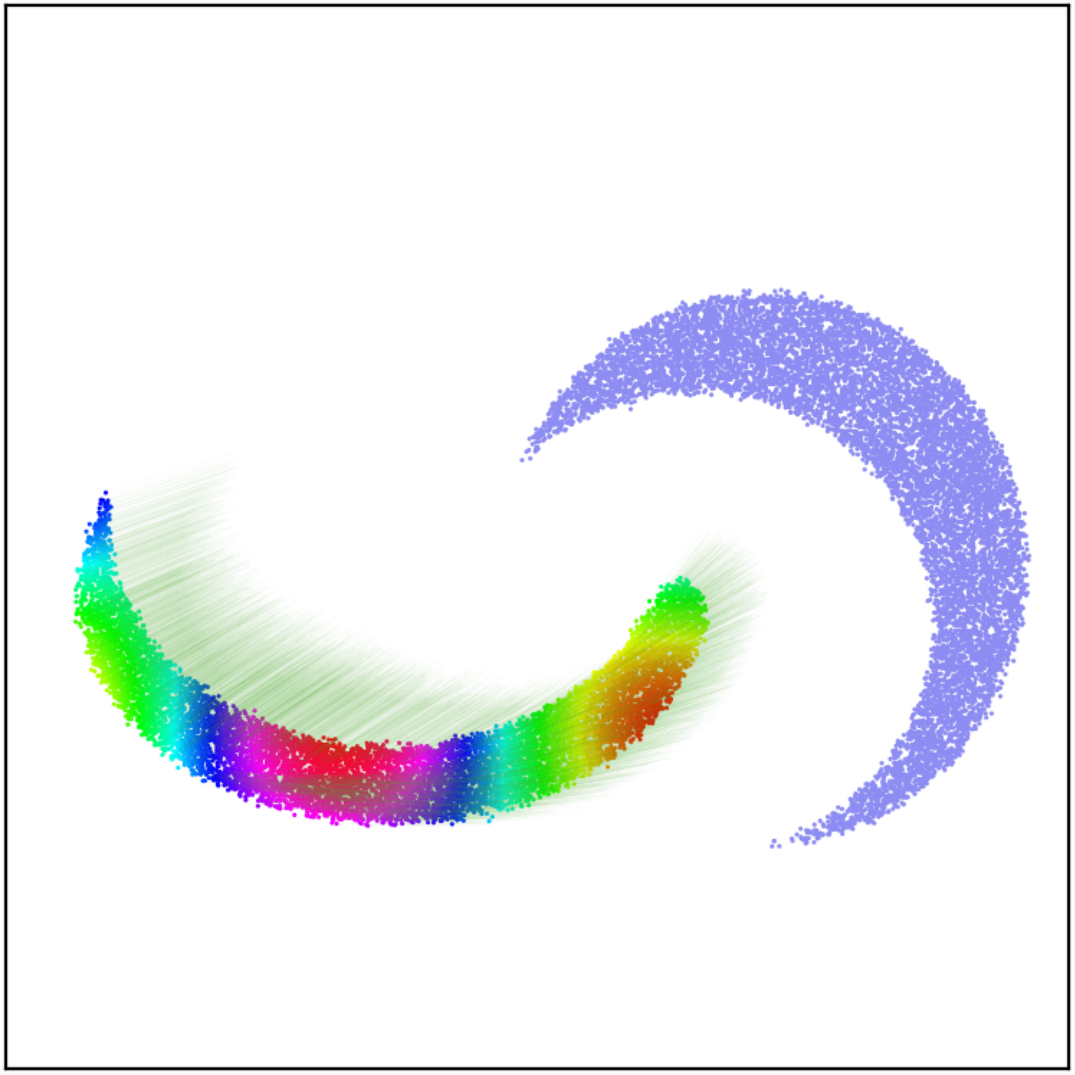
Gradient descent with OT: cheap'n easy registration?

Iteration 2

Gradient descent with OT: cheap'n easy registration?

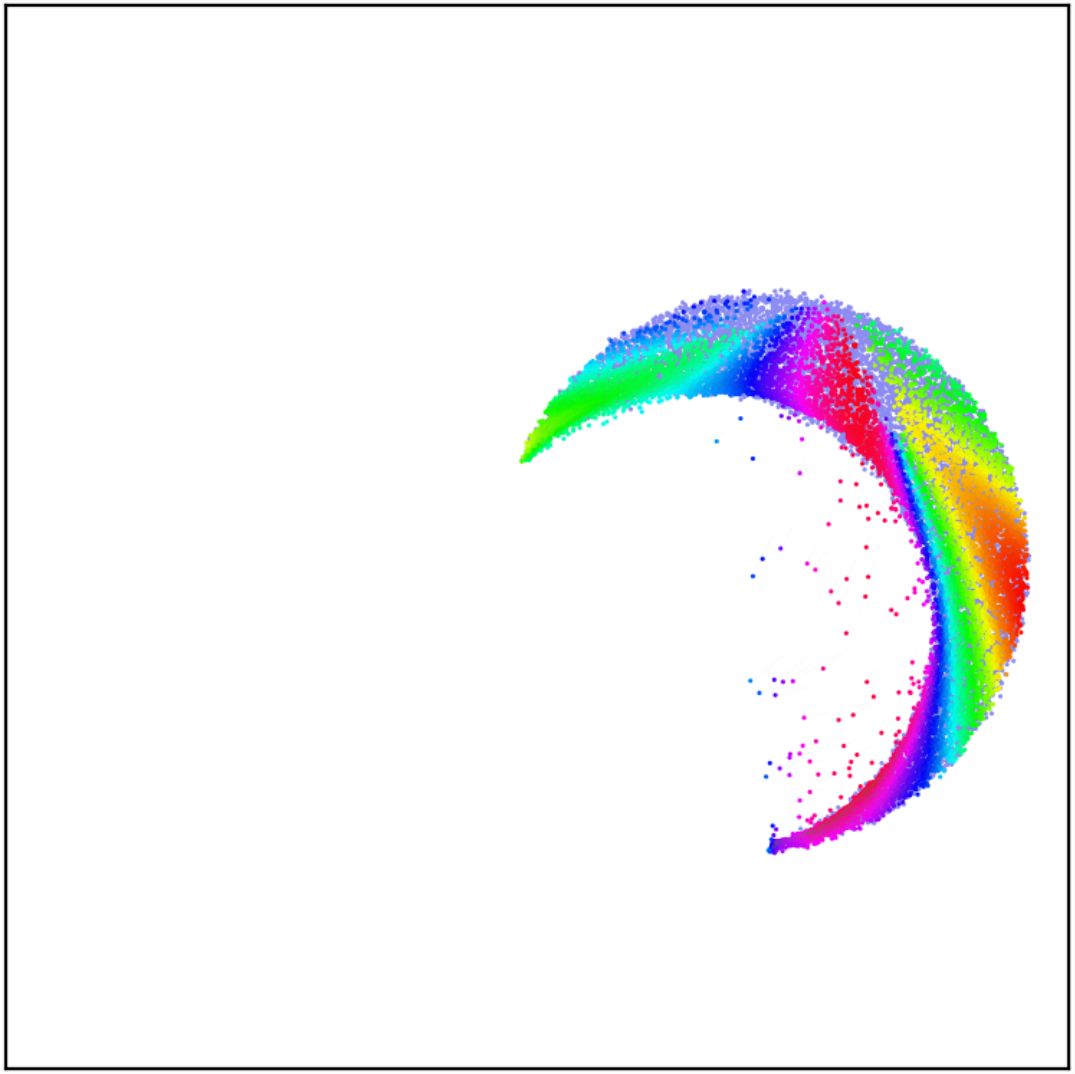
Iteration 10

Gradient descent with OT: cheap'n easy registration? Beware!



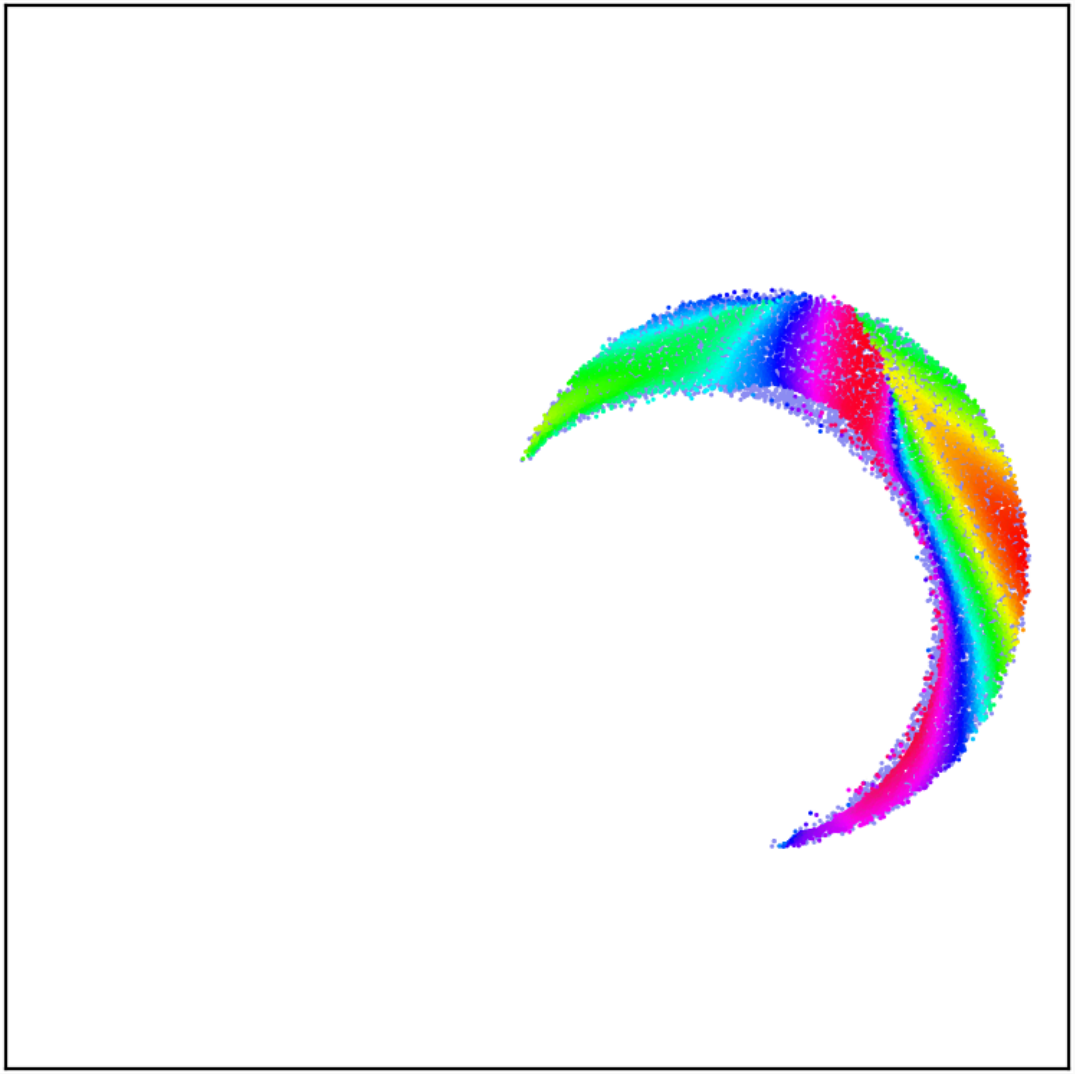
Iteration 0

Gradient descent with OT: cheap'n easy registration? Beware!



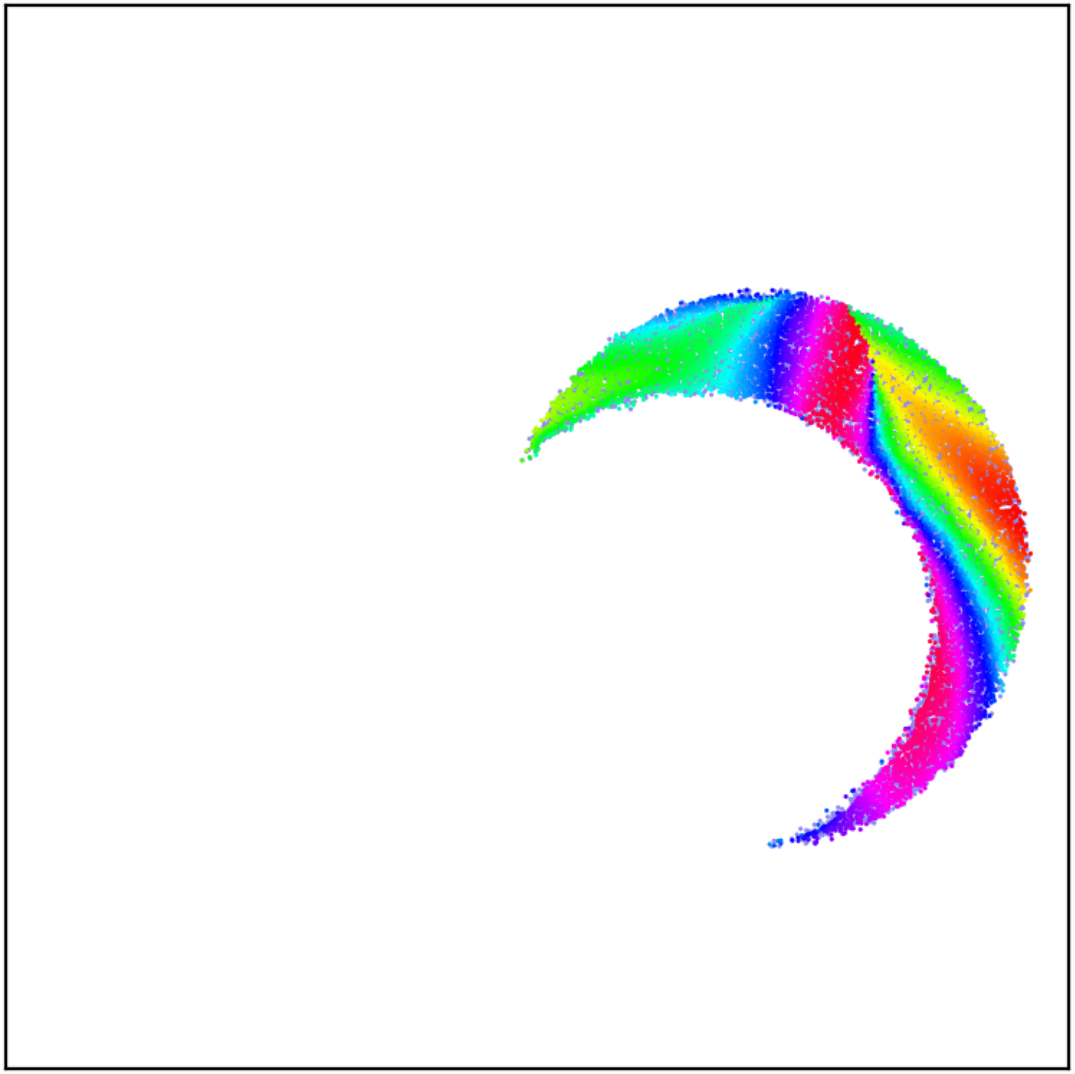
Iteration 1

Gradient descent with OT: cheap'n easy registration? Beware!



Iteration 2

Gradient descent with OT: cheap'n easy registration? Beware!



Iteration 10

Conclusion

Key points

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

Key points

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

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

Key points

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

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

Key points

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

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

$$\Longrightarrow \texttt{www.kernel-operations.io} \Longleftarrow$$

Key points

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

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

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

Main reference: my **PhD thesis**, available in January.

We should go further

Topology-aware registration is a hard, non-convex problem.

OT is good enough as a **loss**, but is *not* the panacea.

We should go further

Topology-aware registration is a hard, non-convex problem.

OT is good enough as a **loss**, but is *not* the panacea.

$\Longrightarrow$ We need robust, data-driven shape metrics.

We should go further

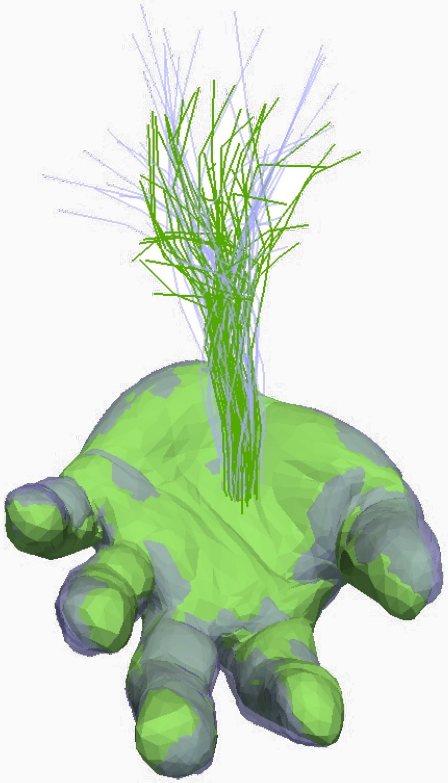
Topology-aware registration is a hard, non-convex problem.

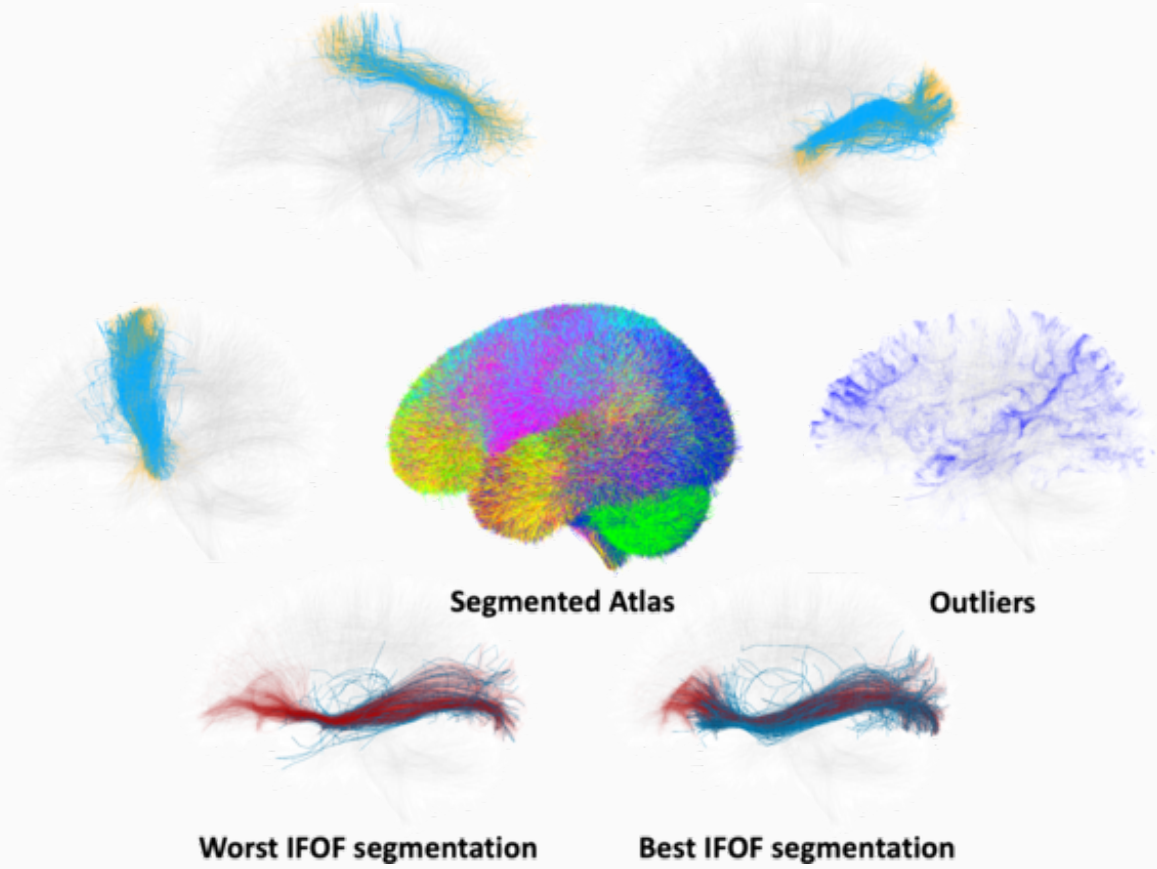
OT is good enough as a **loss**, but is *not* the panacea.

$\Longrightarrow$ We need robust, data-driven shape metrics.

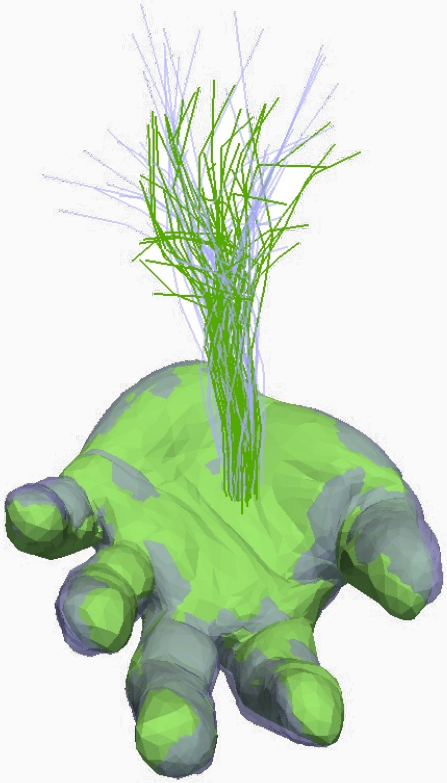
(3^{rd} half)

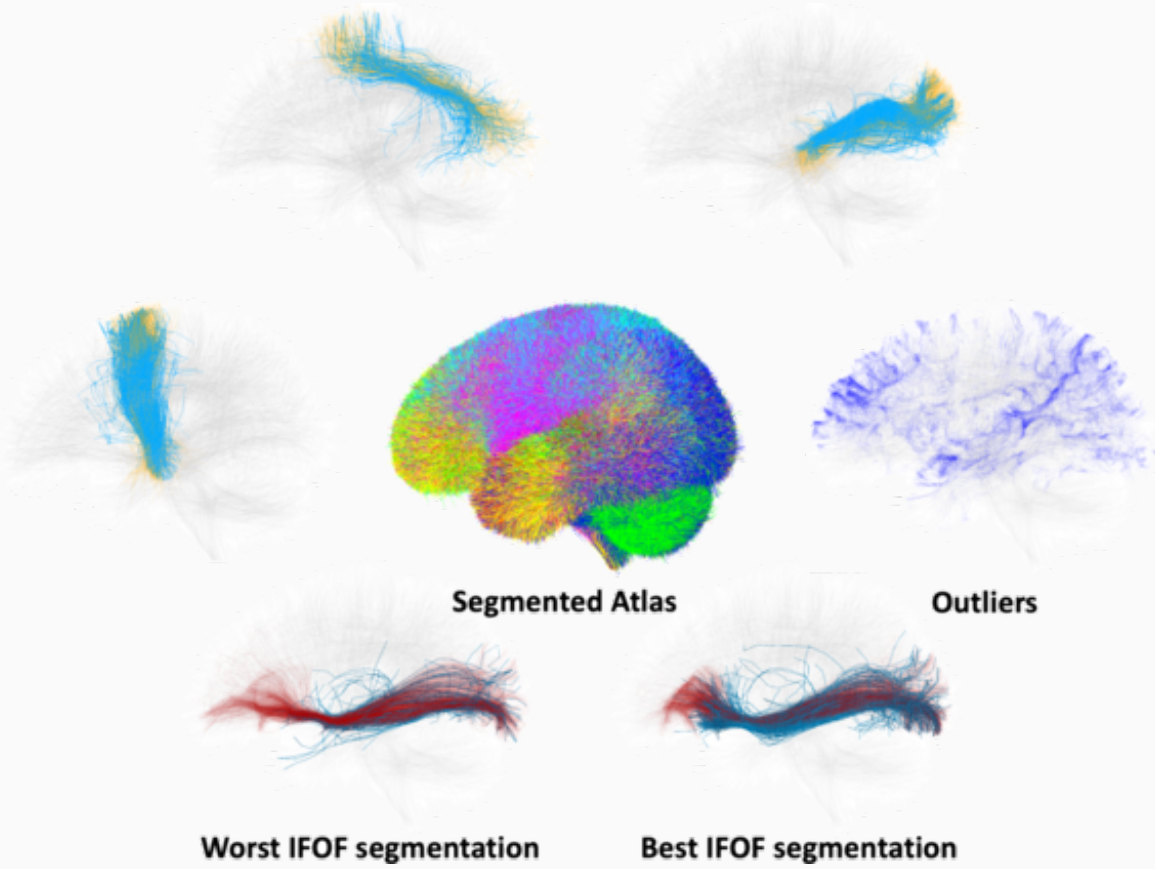
My work so far: theory, implementation and proofs of concept





My work so far: theory, implementation and proofs of concept



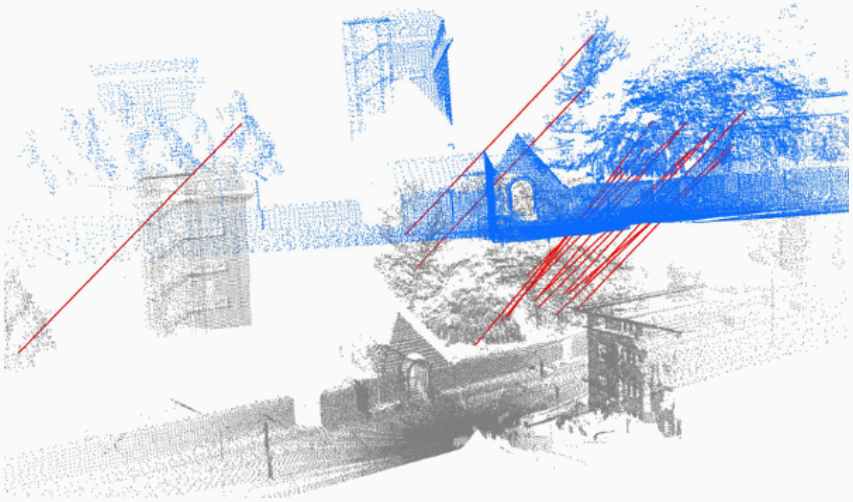


Theorems, libraries, documentation… But no real validation :–(

Thank you for your attention.

Let’s have a chat!

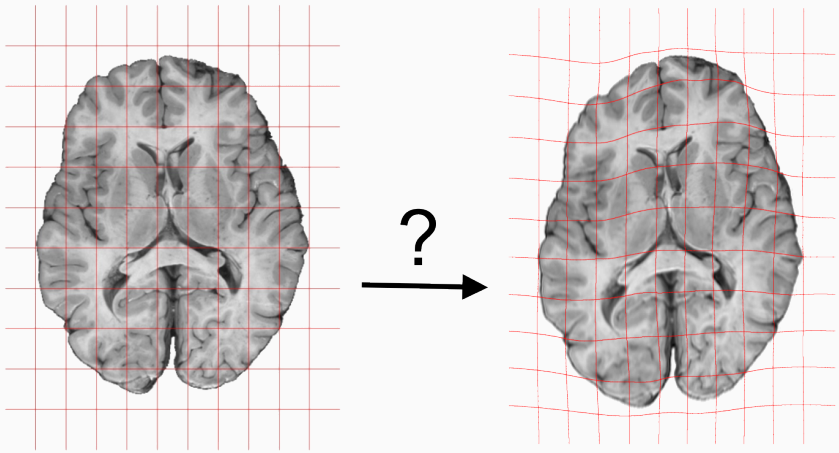
First setting: processing of point clouds



From the documentation of the Point Cloud Library.

- φ is **rigid** or affine
- Occlusions
- Outliers

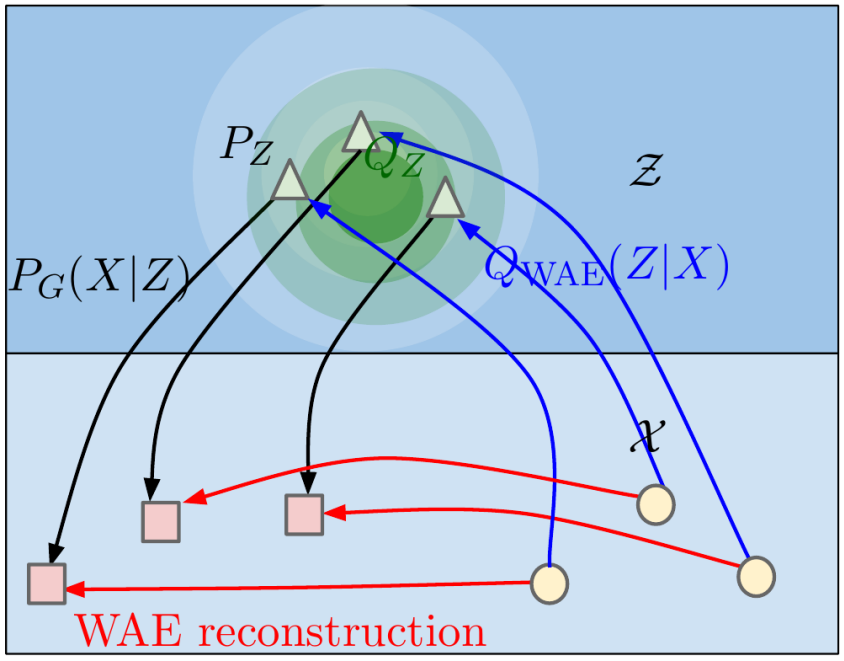
Second setting: medical imaging



From Marc Niethammer's Quicksilver slides.

- φ is a spline or a **diffeomorphism**
- Ill-posed problem
- Some occlusions

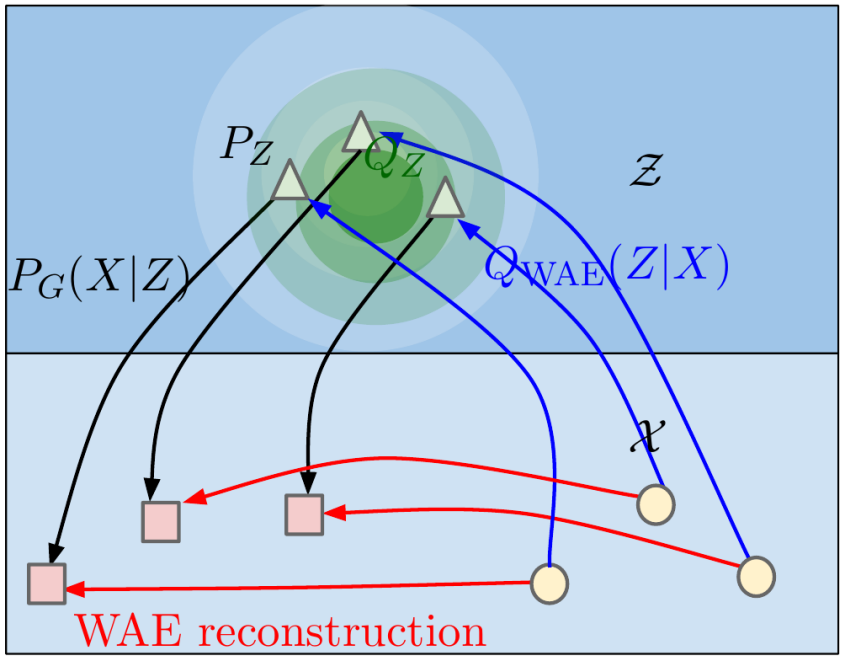
Third setting: training a generative model



Wasserstein Auto-Encoders,
Tolstikhin et al., 2018.

- φ is a **neural network**
- Very weak regularization
- High-dimensional space

Third setting: training a generative model



Wasserstein Auto-Encoders,
Tolstikhin et al., 2018.

- φ is a **neural network**
- Very weak regularization
- High-dimensional space

Which **Loss** function
should we use?

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) = \max_{f \in B} \langle \alpha - \beta , f \rangle,$$

$$\text{look for } \theta^* = \arg\min_{\theta} \max_{f \in B} \langle \alpha(\theta) - \beta , f \rangle$$

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, \|f\|_\infty \leqslant 1 \,\} \implies$ Loss $=$ TV norm:
  - zero geometry
  - **too many** test functions

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, \| f \|_\infty \leqslant 1 \,\} \implies$ Loss $=$ TV norm:
  - zero geometry
  - **too many** test functions

- $B = \{\, \| f \|_2^2 + \| \nabla f \|_2^2 + \cdots \leqslant 1 \,\} \implies$ Loss $=$ kernel norm:
  - may saturate at infinity
  - **screening** artifacts

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \Longrightarrow$ Loss $=$ Wasserstein-1 (OT_0):

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \Longrightarrow$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \Longrightarrow$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512\times 512}, \|\cdot\|_2)$: the ground cost makes no sense

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \Longrightarrow$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512\times 512}, \|\cdot\|_2)$: the ground cost makes no sense

- $B \simeq \{\, f \text{ is 1-Lipschitz} \,\} \bigcap \{\, f \text{ is a } \textbf{CNN} \,\}$
  $\Longrightarrow$ Loss $=$ Wasserstein GAN :

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \Longrightarrow$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512\times512}, \|\cdot\|_2)$: the ground cost makes no sense

- $B \simeq \{\, f \text{ is 1-Lipschitz} \,\} \bigcap \{\, f \text{ is a } \textbf{CNN} \,\}$
  $\Longrightarrow$ Loss $=$ Wasserstein GAN :
  - use **perceptually sensible** test functions

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \Longrightarrow$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512\times 512}, \|\cdot\|_2)$: the ground cost makes no sense

- $B \simeq \{\, f \text{ is 1-Lipschitz} \,\} \bigcap \{\, f \text{ is a } \textbf{CNN} \,\}$
  $\Longrightarrow$ Loss $=$ Wasserstein GAN :
  - use **perceptually sensible** test functions
  - no simple formula: use **gradient ascent**

Dual norms - link with the GANs literature

$$\text{Loss}(\alpha, \beta) \;=\; \max_{f \in B} \langle\, \alpha - \beta \,,\, f \,\rangle,$$

$$\text{look for } \theta^* \;=\; \arg\min_{\theta} \max_{f \in B} \langle\, \alpha(\theta) - \beta \,,\, f \,\rangle$$

- $B = \{\, f \text{ is 1-Lipschitz} \,\} \Longrightarrow$ Loss $=$ Wasserstein-1 (OT_0):
  - S_ε is nearly as efficient as a **closed formula**
  - relevant in **low dimensions**
  - **useless** in $(\mathbb{R}^{512\times512}, \|\cdot\|_2)$: the ground cost makes no sense

- $B \simeq \{\, f \text{ is 1-Lipschitz} \,\} \bigcap \{\, f \text{ is a } \textbf{CNN} \,\}$
  $\Longrightarrow$ Loss $=$ Wasserstein GAN :
  - use **perceptually sensible** test functions
  - no simple formula: use **gradient ascent**
  - can we provide relevant **insights** to the ML community?

References

Our papers:

- *Global divergences between measures: from Hausdorff distance to Optimal Transport*, F., Trouvé, 2018

References

Our papers:

- *Global divergences between measures: from Hausdorff distance to Optimal Transport*, F., Trouvé, 2018
- *Sinkhorn entropies and divergences*,
  F., Séjourné, Vialard, Amari, Trouvé, Peyré, 2018

References

Our papers:

- *Global divergences between measures: from Hausdorff distance to Optimal Transport*, F., Trouvé, 2018
- *Sinkhorn entropies and divergences*,
  F., Séjourné, Vialard, Amari, Trouvé, Peyré, 2018
- *Optimal Transport for diffeomorphic registration*,
  F., Charlier, Vialard, Peyré, 2017

References i

John Ashburner.
A fast diffeomorphic image registration algorithm.
Neuroimage, 38(1):95–113, 2007.

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.

References ii

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.

References iii

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.

References iv

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.

Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-Ichi Amari, Alain Trouvé, and Gabriel Peyré.
Interpolating between optimal transport and MMD using Sinkhorn divergences.
arXiv preprint arXiv:1810.08278, 2018.

Joan Glaunes.
Transport par difféomorphismes de points, de mesures et de courants pour la comparaison de formes et l'anatomie numérique.
These de sciences, Université Paris, 13, 2005.

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.

References viii


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.

References x

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.
lrm 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.