Geometry in medical imaging

Part 1/2 – Diverse data structures

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

IPAIM, Bath – Thursday, 26 June 2025

Who am I?

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

2012–2016 ENS Paris, mathematics.

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

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

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

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

HeKA: a translational research team for public health

Hospitals

Inria Inserm

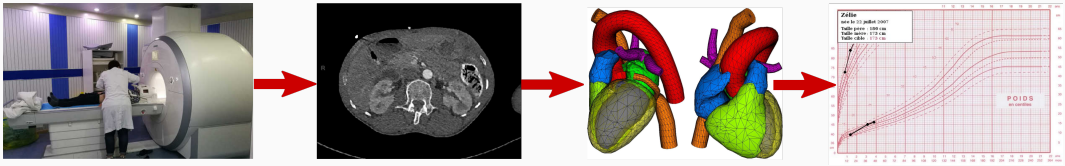
Universities

My main motivation

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

1. Speed up **geometric machine learning** on GPUs:
   $\Longrightarrow$ **pyKeOps** library for distance and kernel matrices, 800k+ downloads.
2. Scale up **pharmacovigilance** to the full French population:
   $\Longrightarrow$ **survivalGPU**, a fast re-implementation of the R survival package.
3. Ease access to modern statistical **shape analysis**:
   $\Longrightarrow$ **GeomLoss**, truly scalable optimal transport in Python.
   $\Longrightarrow$ **scikit-shapes**, beta release in September.

Today’s focus – medical imaging



Three main **characteristics**:

- **Heterogeneous data**: patient history, images, etc.
- Small stratified samples: 10 – 1 000 patients per group.
- Dealing with **outliers** and the **heavy tails** of our distributions is a priority.

Today's talk – assuming that you would enjoy some nice figures

1. What is a **medical image**?

2. **Diverse data structures** create a complex **software environment**.

3. **Mathematical** frameworks to make sense of this complexity:

- **Graph** theory
- Discrete **differential** geometry
- Geometric **measure** theory

What is a medical image?

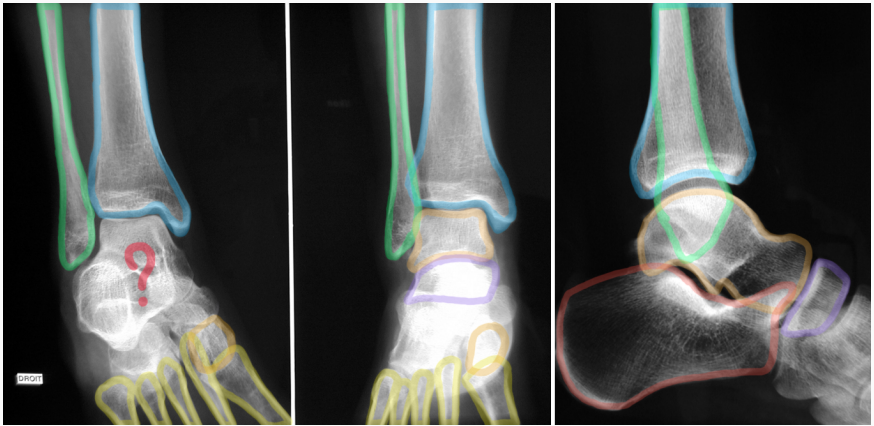
What do you see on a medical image? [Zyg]

What do you see on a medical image? [Zyg]

1. Pixels

What do you see on a medical image? [Zyg]

1. Pixels
2. Anatomy

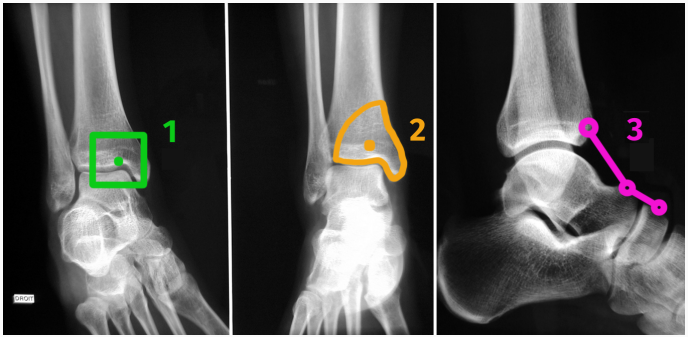


What do you see on a medical image? [Zyg]

1. Pixels
2. Anatomy
3. Function

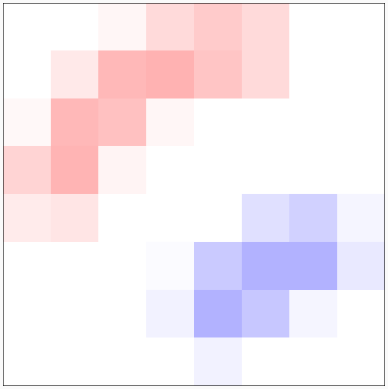
What do you see on a medical image? [Zyg]

1. Pixels
2. Anatomy
3. Function



Simplifying a bit, each level of analysis corresponds to a way of **grouping pixels** with their neighbors.

1st level: a pixel grid



$N_x \times N_y \times N_z$ array of pixels.

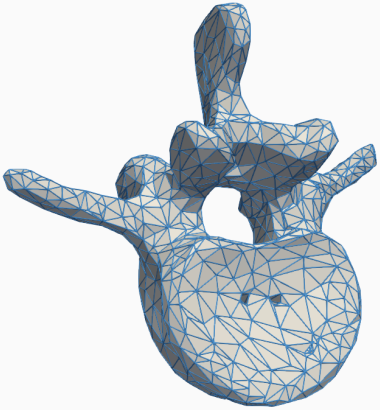
Bitmap images and volumes:

- `.bmp`, `.png`, `.jpg`
- Standard in **radiology**.

$+$ Ordered memory structure.
$+$ Explicit neighborhoods.
$+$ Fast **local** filters.

$\rightarrow$ **Texture** analysis.
$\rightarrow$ Organ **segmentation**.
$\rightarrow$ Pattern **detection**.

2nd level: point clouds and 3D surfaces



$N_{points} \times 3$ array of (x, y, z) coordinates.

Clouds of points ($\pm$ triangles):

- `.svg`
- Standard for **video games**.

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

$\rightarrow$ **3D visualization**.
$\rightarrow$ Anatomical **atlas**.
$\rightarrow$ **Shape** analysis.

3rd level: biomechanical and/or physiological model [Zyg]

Volumetric mesh,
graph of interactions.

Mechanical/biological model:

- Finite elements, networks.
- Standard for **CAD**.

\+ Prior **knowledge**.
\+ **Robust** to noise.
\+ **Realistic** behaviour.

→ **Physiological** interpretation.
→ **Infer** what cannot be seen (stress).
→ **Simulate** a surgery.

Strengths and weaknesses of these image formats

Looking for the **neighbors** of a point in 3D space?

- On a **grid** : read adjacent memory cells.
- With N **points** (x, y, z) : computation of N distances.

Want to **rotate** a bone by 10°?

- On a **grid** : artifacts, loss of details, transfers between memory cells.
- With N **points** (x, y, z) : simple arithmetics on the coordinates.

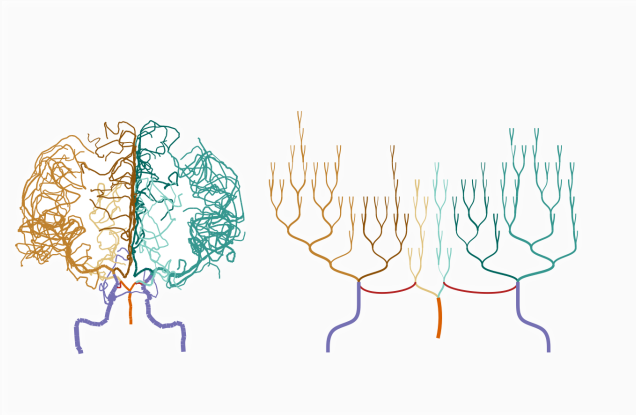
Computational **speed** $\iff$ Training on **large datasets**.

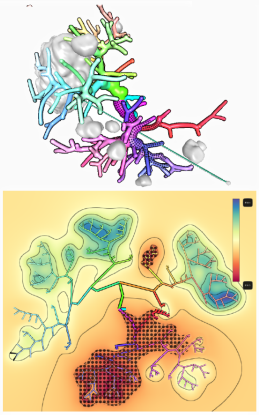
An example in interventional radiology [MKM+02]

The **team**.

The **operating table**.

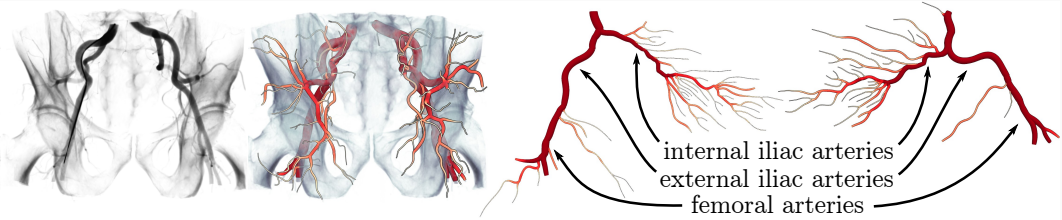
Can we help physicians to navigate vessel networks? [EMML22]





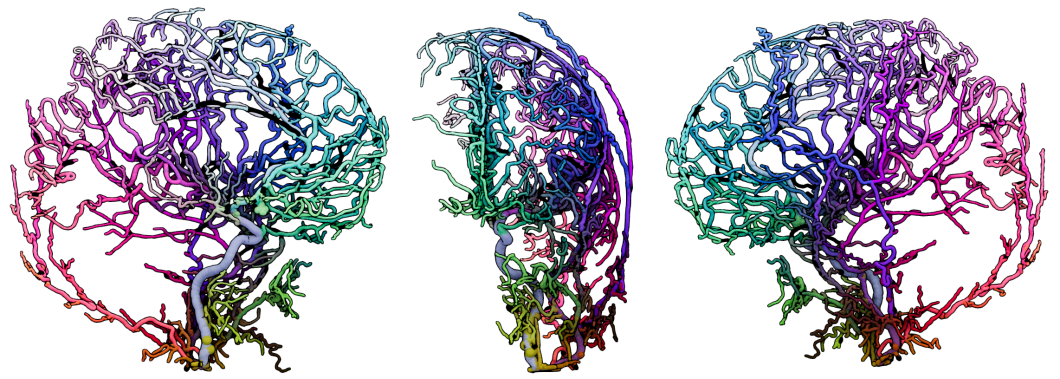
Some examples of **“vessel maps”** that are currently available.
They are optimized for different purposes.

A vessel map that preserves vessel lengths and curvatures [HBAF25]



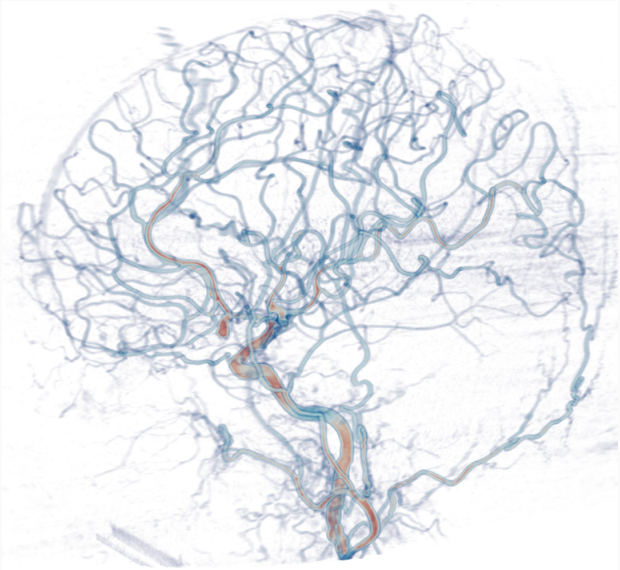
Our new method, tailored to **endovascular** interventions.

A brain arterial network, from three different perspectives

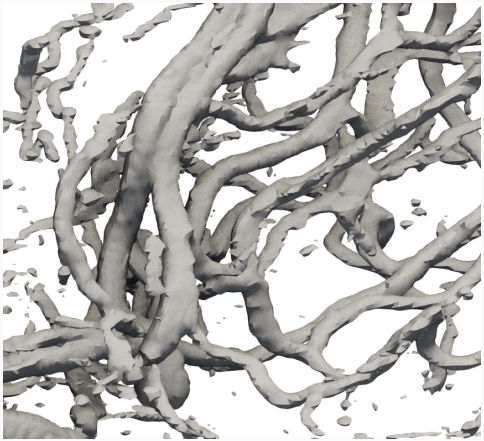


The contrast agent highlighted the left hemisphere.

We take as input a CBCT scan: a volumetric X-ray



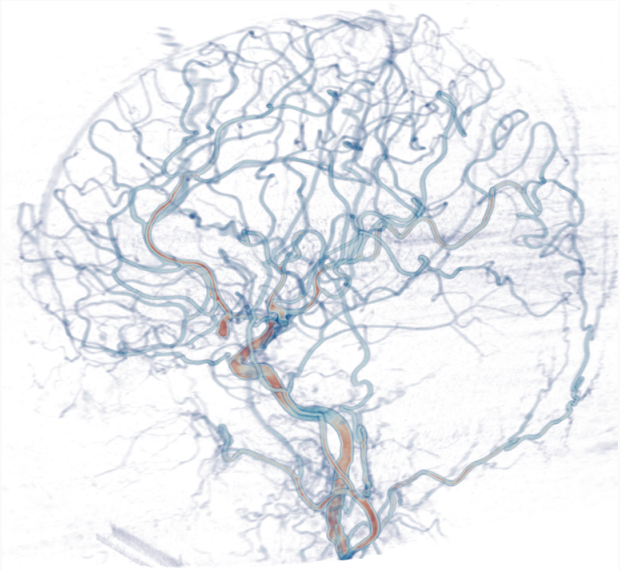
Naive thresholding is not going to work here



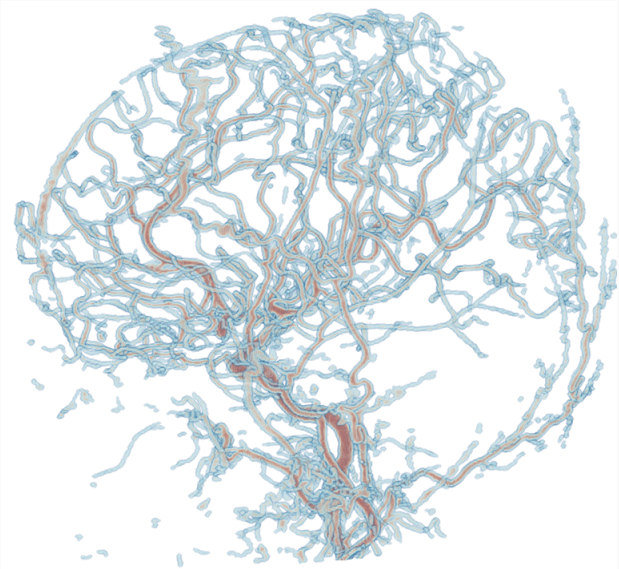
Useless segmentation.

Satisfying segmentation.

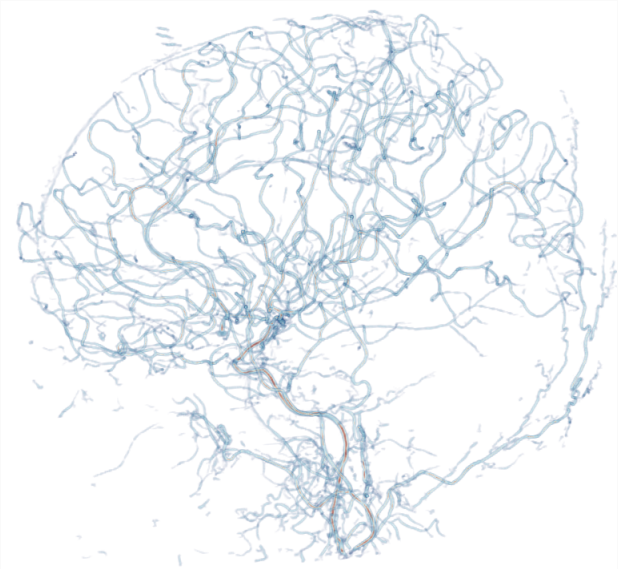
Step 1: our volumetric X-Ray



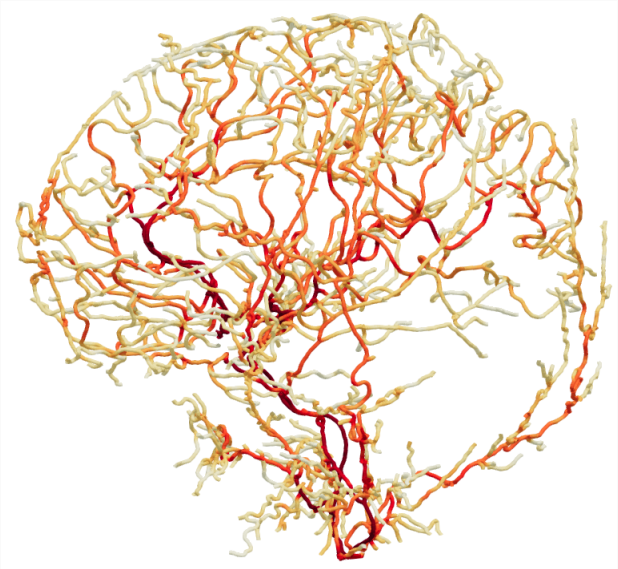
Step 2: compute a signed distance function to the surface



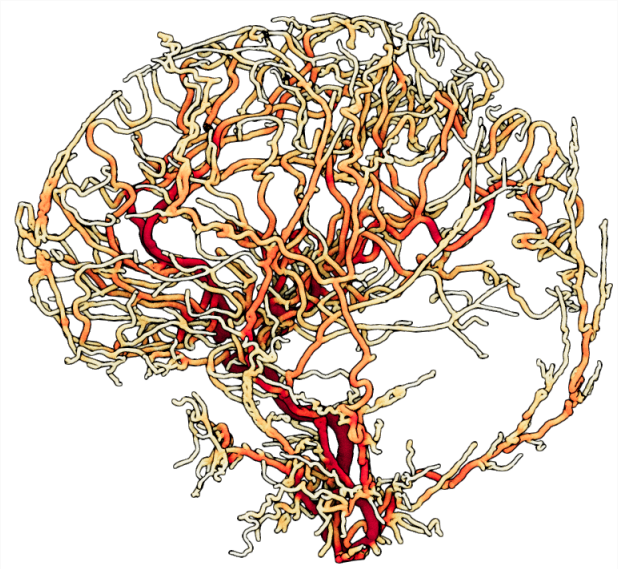
Step 3: Frangi filtering via the second derivatives



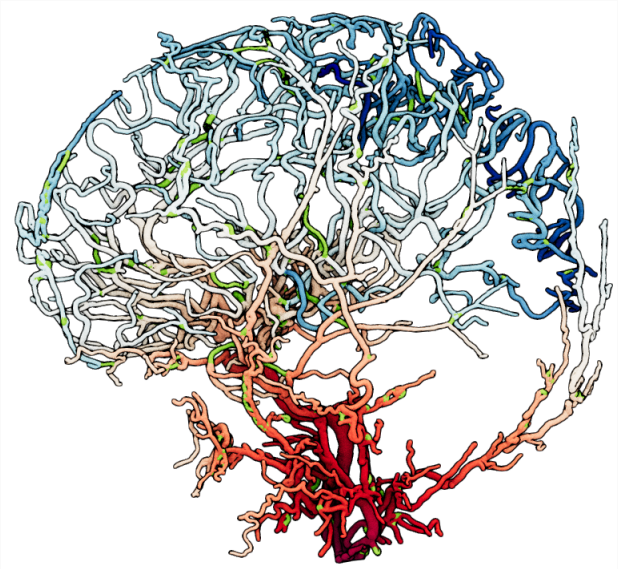
Step 4: centerline extraction



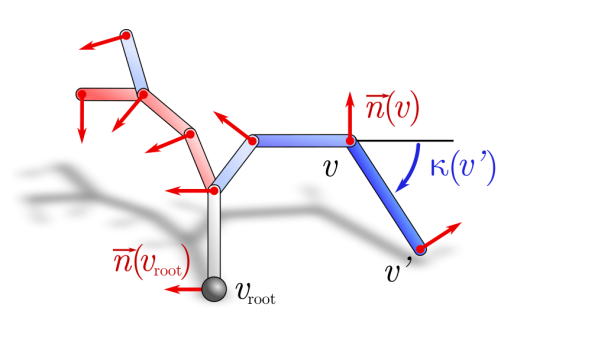
Step 5: with local radii

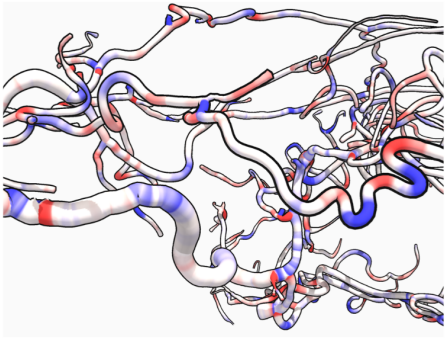


Step 6: topological pruning

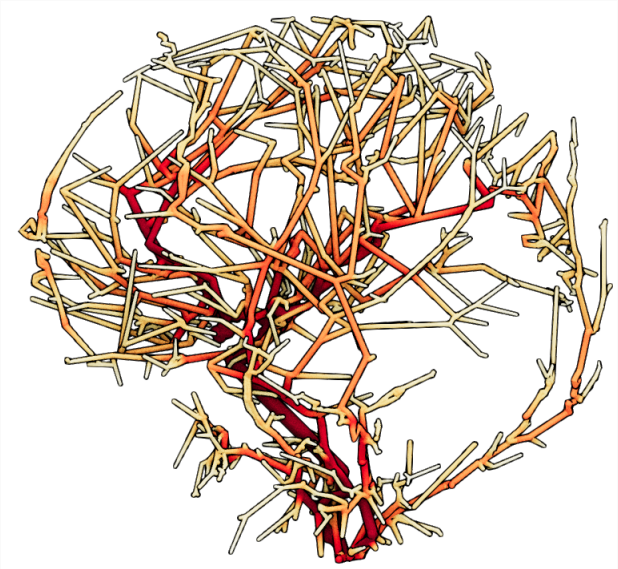


Step 7: curvature estimation

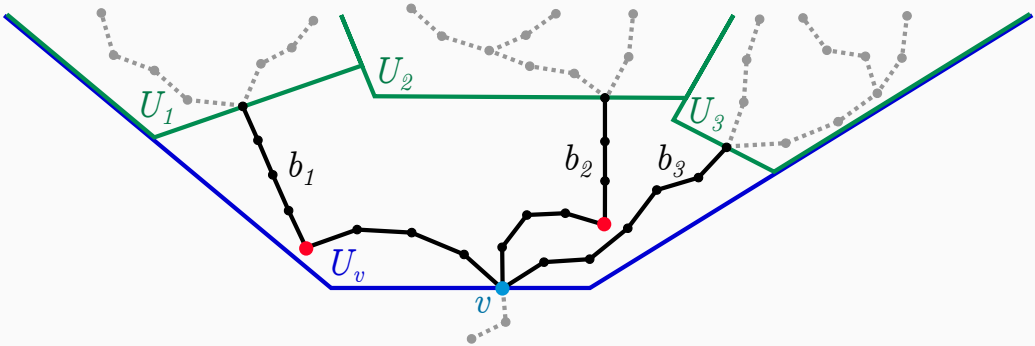




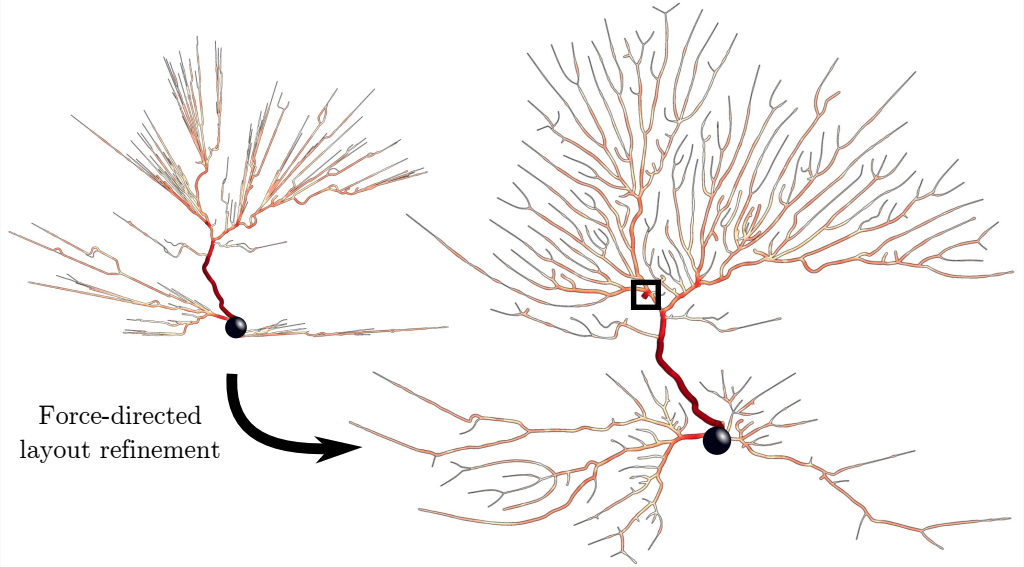
Step 8: coarse graph structure



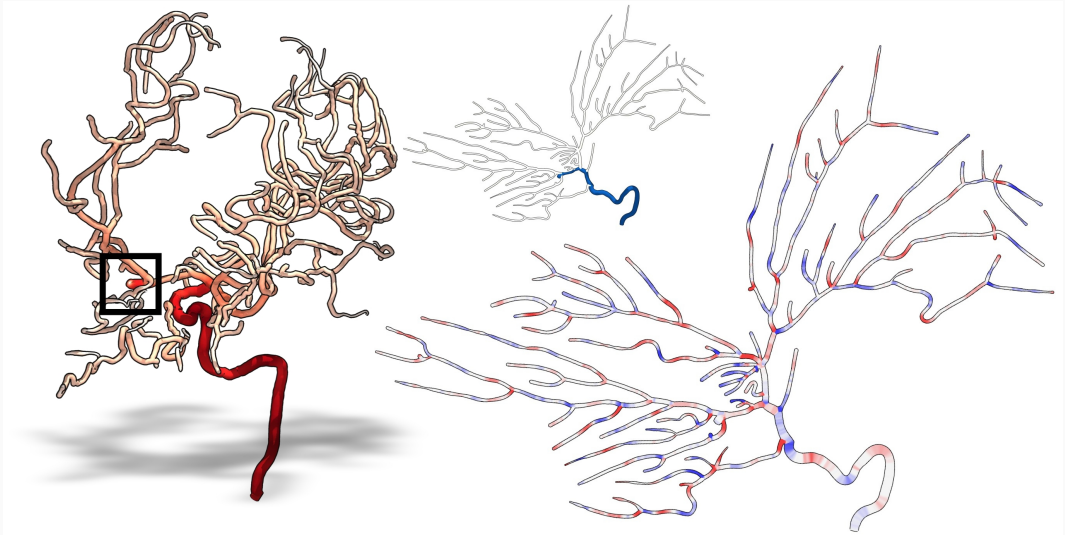
Step 9: recursive planar embedding



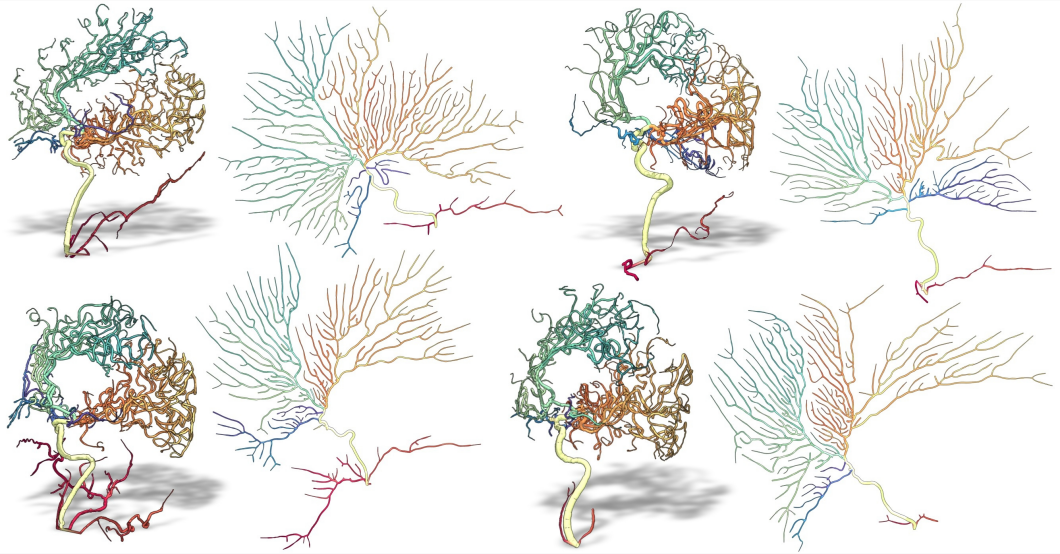
Step 10: iterative refinement



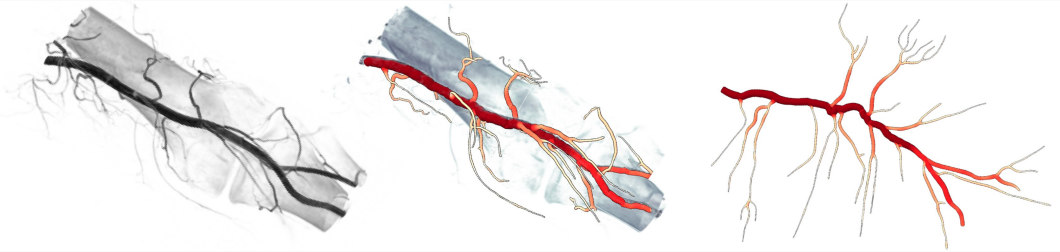
A typical result: planning an intervention to reach an aneurysm



Unlocking population studies



Portability: unfolding the popliteal artery across the knee



To summarize

AI = **statistical regression** method + relevant **computational model**.

In medical imaging, we represent patient data as:

1. A 2D or 3D **pixel grid**.
2. An array of (x, y, z) **coordinates**.
3. A **web** of complex interactions.
4. All three at once!

In most cases, we define a large **structured formula**:

$$\text{image} \xrightarrow{F} F(\text{image}) \simeq \text{diagnostic}$$

F is a parametric computing **architecture**
$\simeq$ **model** to fit $\simeq$ **network** to train.

Software bottlenecks for AI research

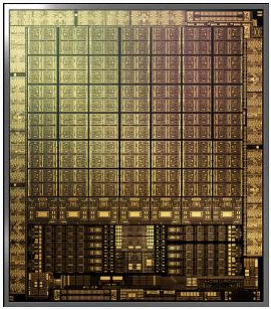
The AI revolution is driven by gaming computers

Digital images and machine **learning** have been studied for **decades**.
Breakthrough in 2010-15 : using **PlayStations** to do **science** became **easy**.

Research effort at all levels towards:

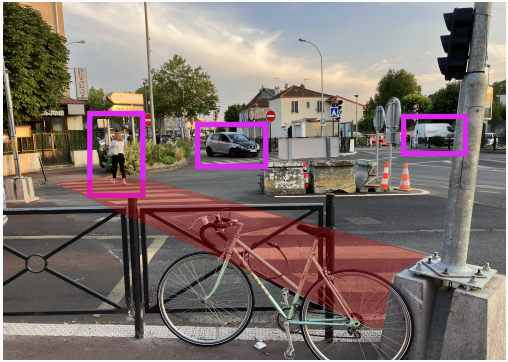
- Increasingly powerful **computers**.
- Increasingly convenient **software toolkits**.
- Increasingly relevant **models**.

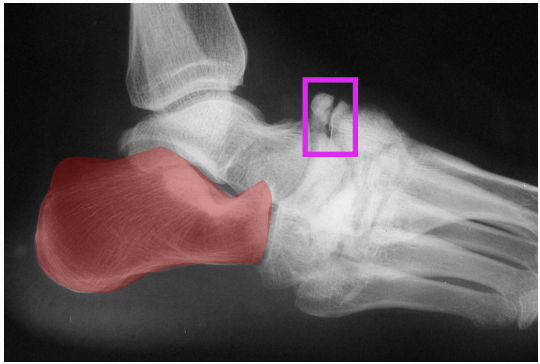
Spectacular results in a few applications
$\Longrightarrow$ massive **investments**, industry + governments.



10,000 cores on a GPU.

For grid images: a mature ecosystem





Main motivation for AI in 2012-2022: **self-driving cars**.
Key challenges: **segment** the environment, **detect** other actors.

Two full software suites to manipulate **images as grids of pixels**:
TensorFlow/JAX (Google) and PyTorch (Facebook-Meta).

To go beyond prototypes, engineers need a full software suite

Graphics: Printer + Driver + **Photoshop** $\Longrightarrow$ Illustrations

Tabular data: GPU + **cuBLAS** + **PyTorch** / TensorFlow $\Longrightarrow$ **"Classical"** neural networks

Pixel grids: GPU + **cuDNN** + PyTorch / TensorFlow $\Longrightarrow$ **Convolutional** neural networks

Point clouds and graphs : GPU + **CUDA** + **??** $\Longrightarrow$ **Geometric** neural networks

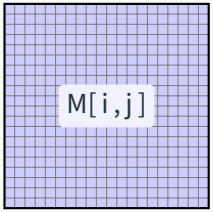
Scientific computing libraries represent most objects as tensors

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

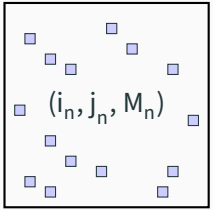
- **Long access times** to the registers penalize the use of large **dense** arrays.
- Hard-wired **contiguous** memory accesses penalize the use of **sparse** matrices.

Challenge. In order to reach optimal run times:

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



Dense array



Sparse matrix

The KeOps library: efficient support for symbolic matrices

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

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

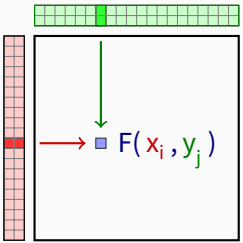
If the formula "F" is simple ($\leqslant$ 100 arithmetic operations):

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

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

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

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



Symbolic matrix

Formula + data

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

A first example: efficient nearest neighbor search in dimension 50

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

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

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

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

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

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

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

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

The KeOps library combines performance with flexibility

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

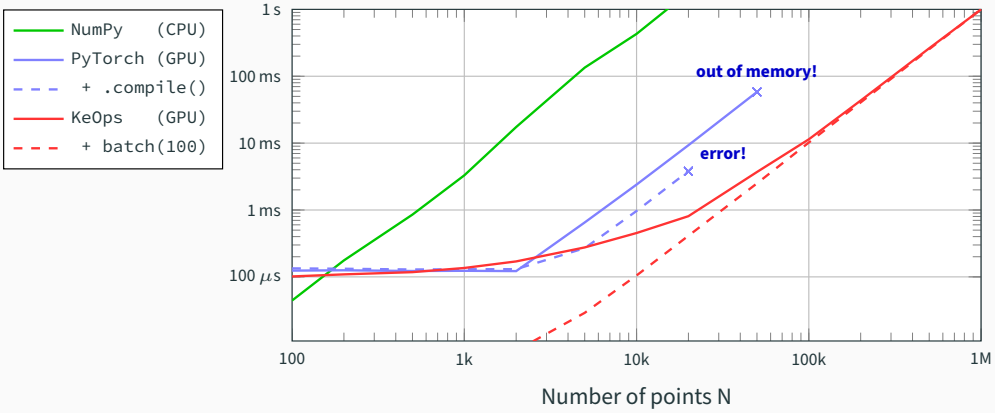
```
D_ij = ((x_i - x_j) ** 2).sum(dim=2)      # Euclidean
M_ij =  (x_i - x_j).abs().sum(dim=2)      # Manhattan
C_ij = 1 - (x_i | x_j)                    # Cosine
H_ij = D_ij / (x_i[...,0] * x_j[...,0])   # Hyperbolic
```

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

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

KeOps lets users work with millions of points at a time

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



Yet another ML compiler?

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

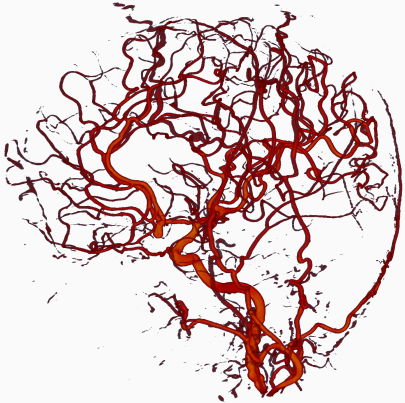
- Focus on **generality** (software + hardware).
- Increasingly easy to use via e.g. PyTorch 2.0.

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

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

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

For point clouds and graphs: work in progress



Brain arterial network.

How do we **process this object**?

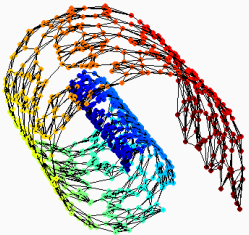
An ecosystem under construction:

- **KeOps** : since 2017
  - Fast learning with **point clouds**.
- **PyG** : since 2018
  - Fast learning with **graphs**.
- **Warp**, **FEniCSx** and **Taichi** : since 2018
  - Fast learning with **physics**.
- **PyVista** and **Vedo** : since 2019
  - **3D visualization.**
- **scikit-shapes**: in 2025
  - Easy **morphometrics.**

Can we find a unifying perspective?

Graph theory

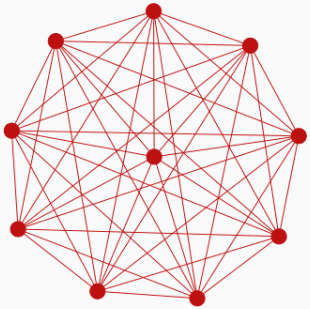
Graphs = local neighborhood structures



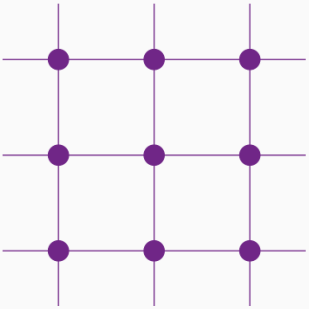
The K-NN graph describes
the **local** structure of a dataset.

Untangling a soup of edges to produce
a **global** understanding is hard.

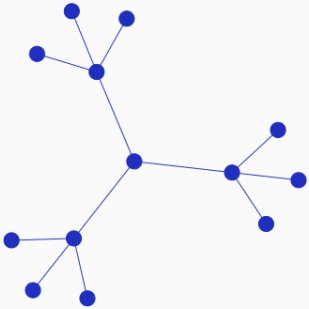
Simple archetypes [TDGC+21]



Cliques are like balls with positive curvature.

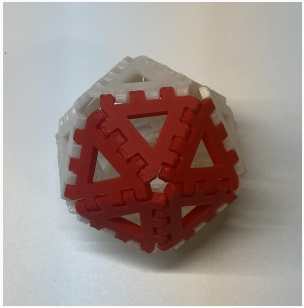


Grids are like planes with flat curvature.

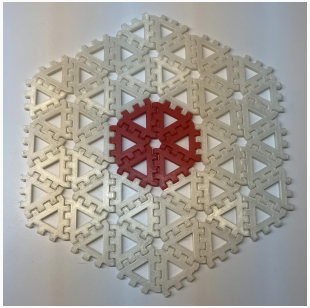


Trees are like saddles with negative curvature.

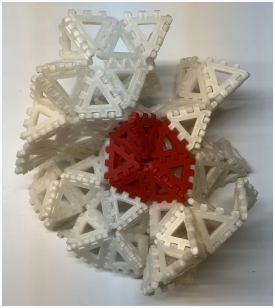
Simple archetypes



Cliques are like balls with positive curvature.

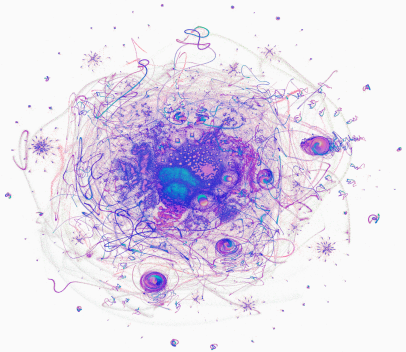


Grids are like planes with flat curvature.



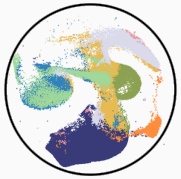
Trees are like saddles with negative curvature.

Embedding methods such as UMAP are excellent diagnostic tools [Wil]



Visualizing the set of **integer numbers** 1, 2, 3, …, 8,000,000.

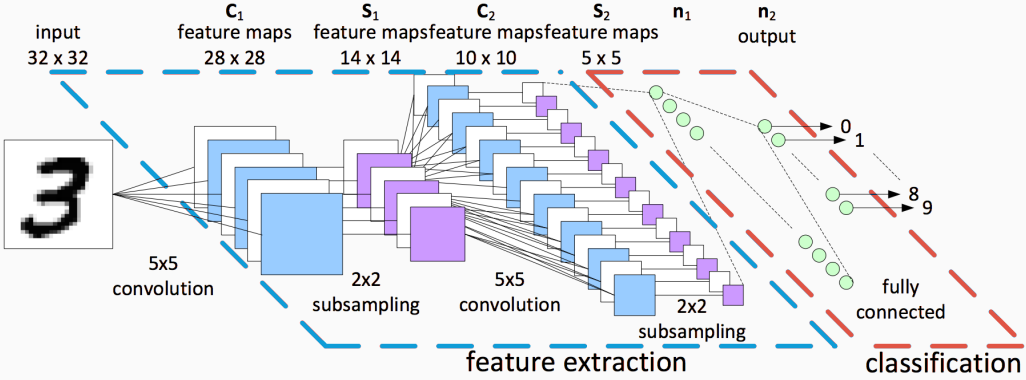
Liver
Week 7



Bone marrow
Week 20

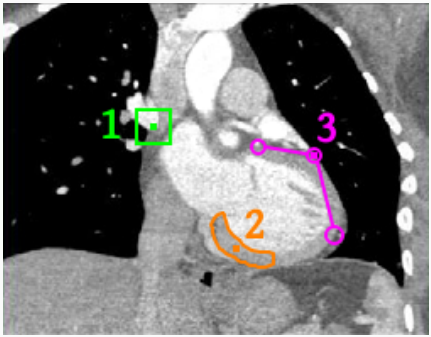
Visualizing the differentiation of Hematopoietic **stem cells**.

Since 2012, we have grown used to convolutional neural networks [PMC11]



Can we apply this methodology to higher-level descriptions? [EPW11, Man11]

1. Pixels
2. Anatomy
3. Function



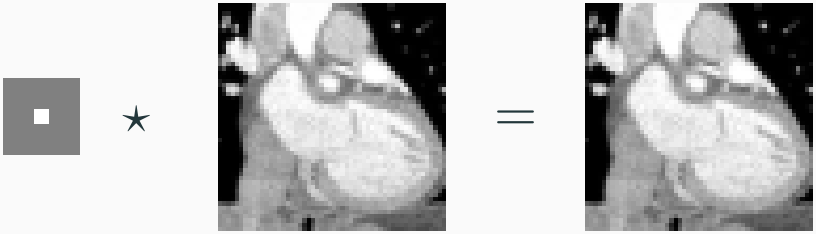
Simplifying a bit, each level of analysis corresponds to a way of **grouping pixels** with their neighbors.

What is a convolution?

Convolution (i.e. weighted average of the neighboring pixels) :

Cheap generalization of the **product** "a · x",

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



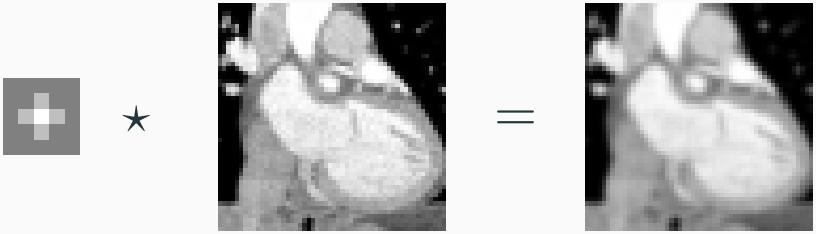
φ x $\varphi \star x$

What is a convolution?

Convolution (i.e. weighted average of the neighboring pixels) :

Cheap generalization of the **product** "a · x",

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



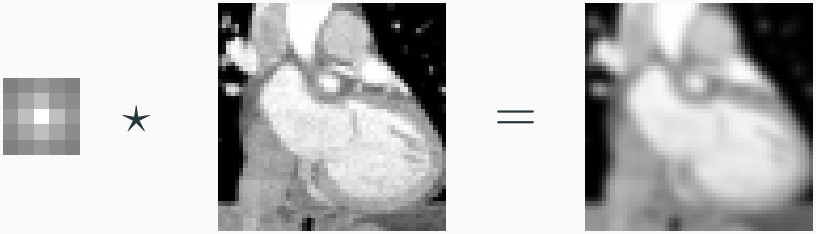
φ x $\varphi \star x$

What is a convolution?

Convolution (i.e. weighted average of the neighboring pixels) :

Cheap generalization of the **product** "a · x",

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



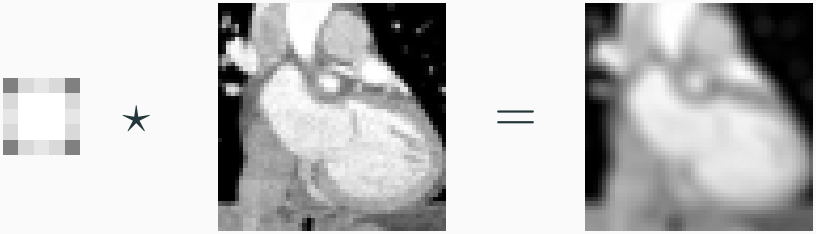
φ x $\varphi \star x$

What is a convolution?

Convolution (i.e. weighted average of the neighboring pixels) :

Cheap generalization of the **product** "a · x",

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



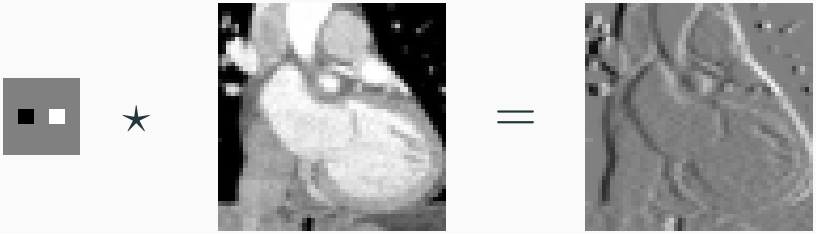
φ x $\varphi \star x$

What is a convolution?

Convolution (i.e. weighted average of the neighboring pixels) :

Cheap generalization of the **product** "a · x",

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



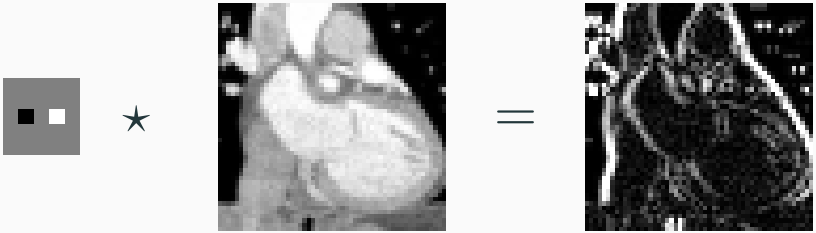
φ x $\varphi \star x$

What is a convolution?

Convolution (i.e. weighted average of the neighboring pixels) :

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

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



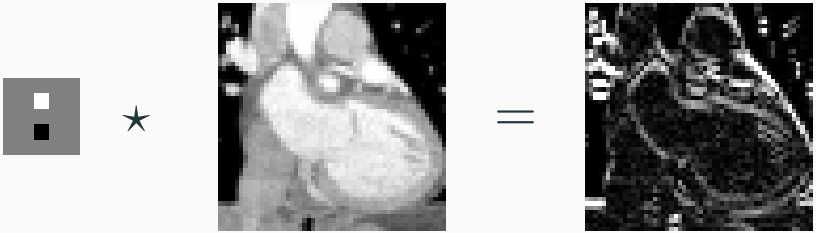
φ $\quad$ x $\quad$ $|\varphi \star x|$

What is a convolution?

Convolution (i.e. weighted average of the neighboring pixels) :

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

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



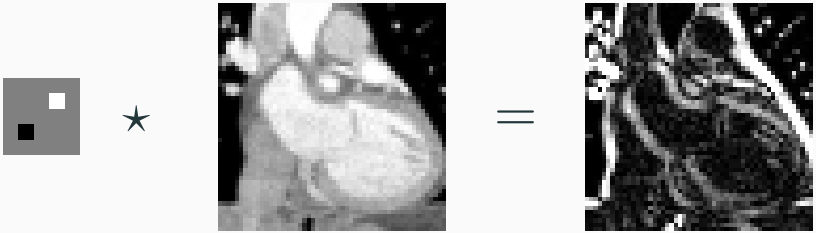
φ $\quad$ x $\quad$ $|\varphi \star x|$

What is a convolution?

Convolution (i.e. weighted average of the neighboring pixels) :

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

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



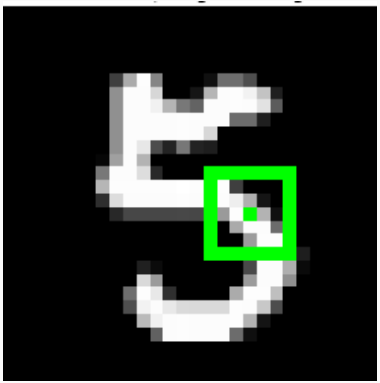
φ $\quad$ x $\quad$ $|\varphi \star x|$

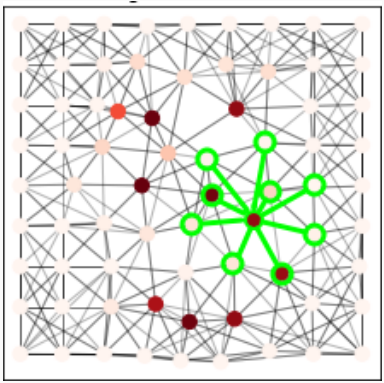
What is a convolution?

Convolutions on **grids**:

- Are **cheap**.
- Enable **pattern** detection and **texture** analysis.
- **Proven track record** since the 1960's:
  Gaussian blur, edge detectors, Laplacian pyramids,
  wavelets, JPEG2000, convolutional neural networks…

Message-passing on graphs [DJL+20]

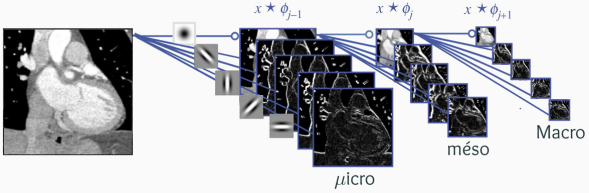




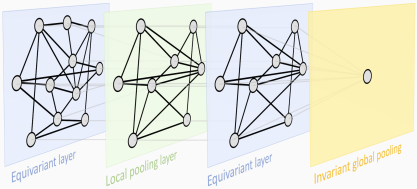
$$x[i,j] \leftarrow \sum_{k,l} \varphi[k,l] \cdot x[i-k, j-l]$$

$$x[i] \leftarrow \sum_{i \leftrightarrow k} \varphi(x[i], x[k], \text{edge}[i,k])$$

Multiscale architectures on graphs [Mal16, BBCV21]



Grid convolutions
+ downsamplings.



Graph convolutions
+ downsamplings.

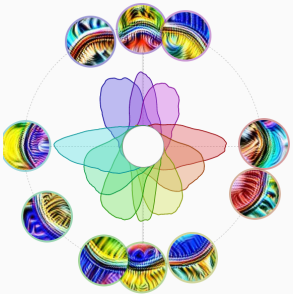
The promises of geometric deep learning

We intend to leverage the **intrinsic structure** of:

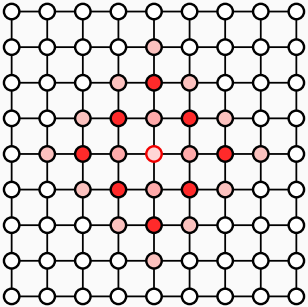
- **Point** clouds.
- Surface and volume **meshes**.
- **Molecular** graphs, proteins.
- Social and communication **networks**.

Unfortunately, things are **not that simple**.

Problem 1: How do we deal with the lack of orientation? [CGC+20, NoJ07]



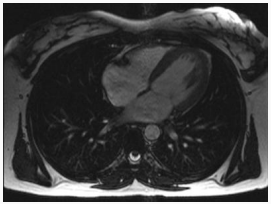
CNNs learn **oriented** curve detectors.



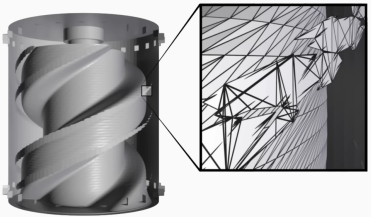
Vanilla graph convolutions define **isotropic** filters.

Hairy ball theorem: no globally consistent 2D coordinates on a sphere.

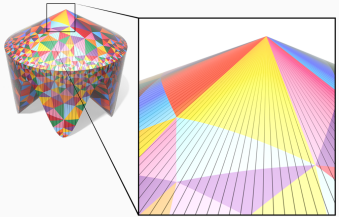
Problem 2: How do we deal with varying sampling densities? [SSC19]



MRI slice:
voxel size = 1 mm^3.

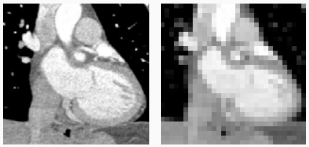


"**Hell** is other people's meshes"
– Jean-Paul Sartre

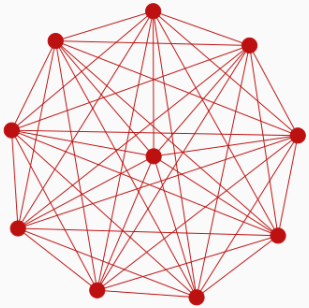


Intrinsic triangulations.
Can we use them for ML?

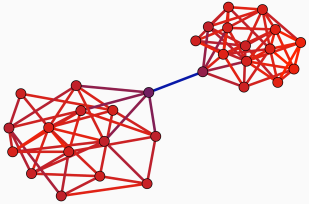
Problem 3: How do we deal with highly non-Euclidean graphs? [TDGC+21]



Downsampling a **grid** is easy.

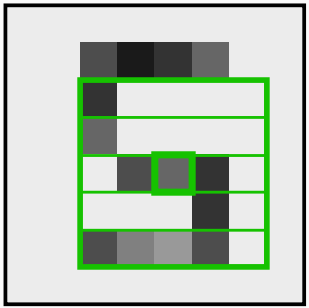


How do we downsample a **clique**?

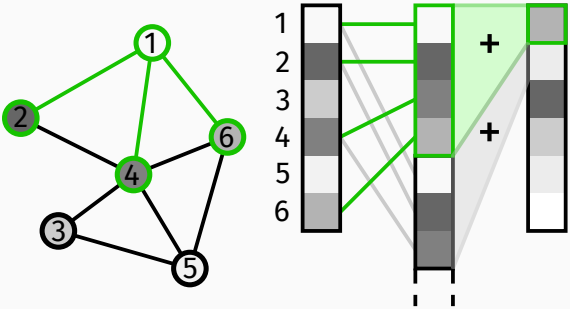


How do we handle **bottlenecks**?

Problem 4: GPUs are not optimized for graphs

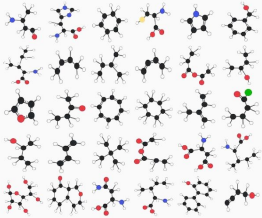


Fixed + **contiguous** neighborhoods
$\Longrightarrow$ Optimal compilation.

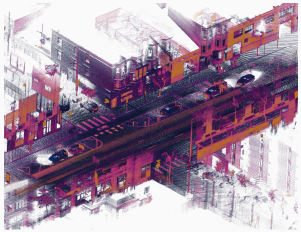


Varying sizes + **random** memory accesses
$\Longrightarrow$ x100 slow-down.

Problem 5: The range of target applications is too wide. [Dil15, Lu19, Gra19]



Molecules.



Lidar scans.

Social networks.

A fast-growing community

In 2017, geometric deep learning was **a solution in search of a problem**.

With an appealing pitch, it attracted:

- **Computer scientists**, looking for new ways of combining features.
- **Mathematicians**, looking for new applications of their insights.
- **Domain experts**, looking for a breakthrough on their data.

A blooming corpus of methods

The literature since 2017:

- Dozens of different **convolution** operators.
- Renewed interest in **theoretical graph ML**.
- Useful **extensions** for PyTorch: PyG, DGL, KeOps…
- **Cross-pollination** between different fields: computer graphics, signal processing, chemistry…

A field without consensus

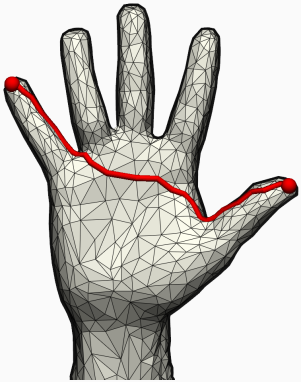
Unfortunately, it is hard to recommend some methods over the others:

- Applications are **wildly different** from each other.
- Most theoretical and experimental works are **proof-of-concepts**.
- Benchmarks are **highly unreliable**.

Realistically, in **medical imaging**, we must go beyond the simple "data as graph" model and use a **stronger structure**.

The limitations of graph theory

Home turf for graph theory.



Is this good enough?

Group theory is everywhere: see e.g. the set of complex numbers $(\mathbb{C}, +, 0)$.

But if you forget that it is also a **field** of **dimension 2**,

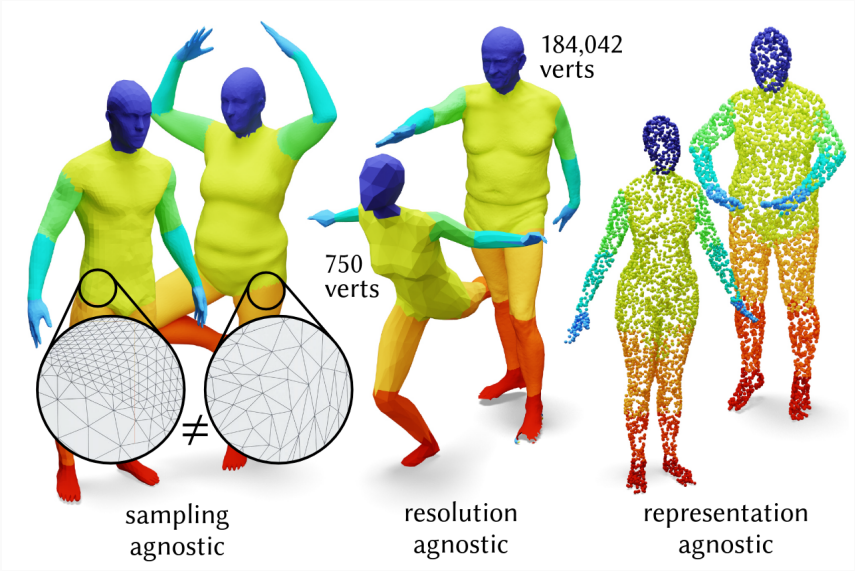
you'll miss out on the **most interesting results**.

Discrete differential geometry

On the blackboard

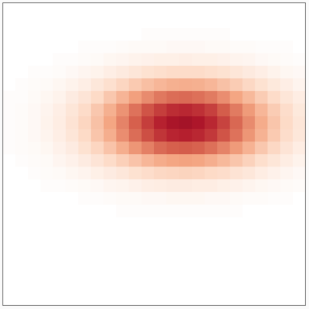
- Geometric structure $\iff$ Function smoothness
- The cotan Laplacian
- The heat method for geodesic distances

DiffusionNet [SACO22]



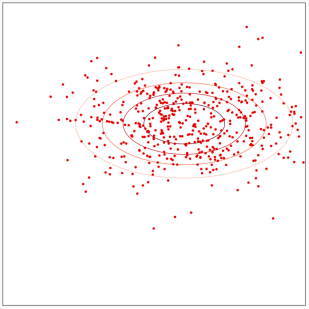
Geometric measure theory

Probability distribution $\alpha =$ weights a_i at locations x_i



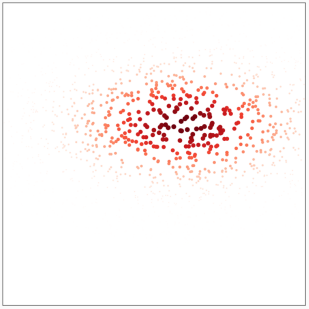
Histogram:

variable weights a_i,
fixed locations x_i.



Sample:

fixed weights $1/\mathrm{N}$,
variable locations x_i.



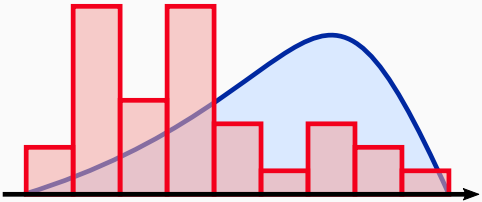
Weighted point cloud:

variable weights a_i,
variable locations x_i.

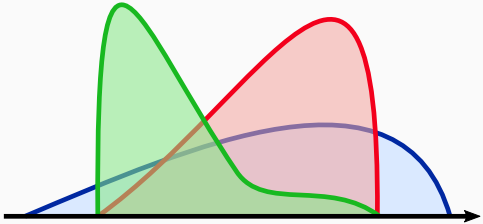
Discrete sum $\alpha = \sum_{i=1}^{\mathrm{N}} a_i \delta_{x_i}$ $\implies$ **Continuous** density $\alpha = \int_x a(x)\,\mathrm{d}x$.

Today, we assume that $a \geqslant 0$ and sums up to 1.

Quantifying distances between probability distributions

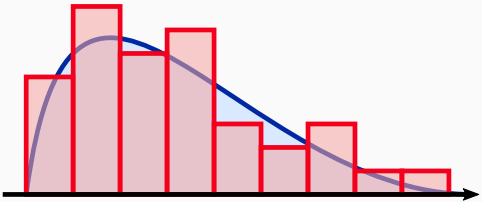


We must **handle** both **discrete** and **continuous** distributions.

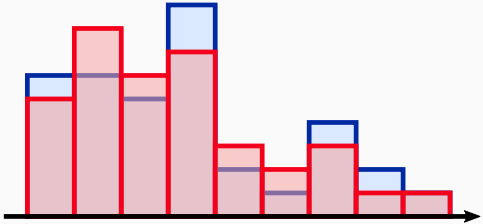


We must **choose** if α is closer to β (same mean value) or to γ (same support).

Application 1: One-sample and Two-sample testing



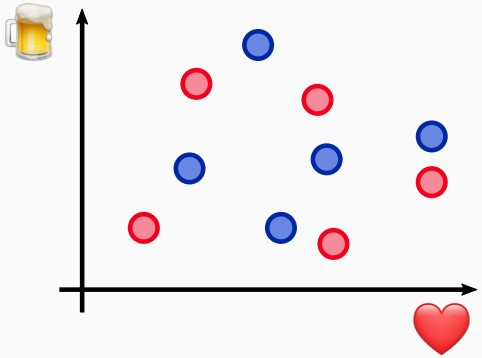
One-sample test:
discrete observation α,
continuous model β.



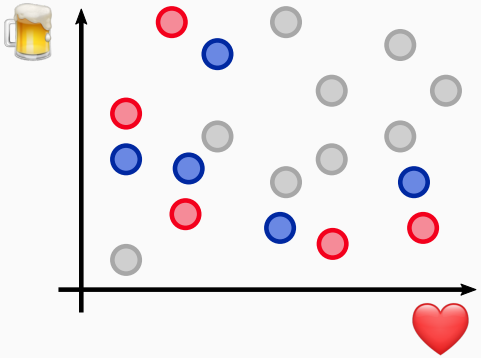
Two-sample test:
two discrete observations α and β.

Null hypothesis: α and β come from the **same distribution**.
Test: reject if $\mathrm{d}(\alpha, \beta)$ is too large.

Example: Splitting a population evenly for a clinical trial



Problem 1: ensure that the treatment and control groups have similar characteristics.



Problem 2: given a large population, pick a group of control patients that have similar characteristics to our treated patients.

Application 2: Classification = regression in a space of distributions

Linear regression:

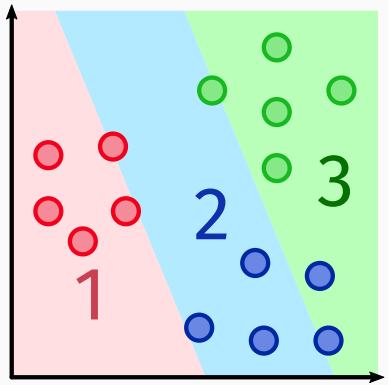
- Encode class labels as **integer numbers**

$$l(x) \in \{1, 2, 3\} .$$

- Predict a **score** $s(x)$ at every location x.
- Minimize the **least square error**:

$$\frac{1}{\mathrm{N}} \sum_{i=1}^{\mathrm{N}} \left| l(x) - s(x) \right|^2 .$$

Massive **bias** depending on the **ordering** of the labels.



2 input features, 3 classes.

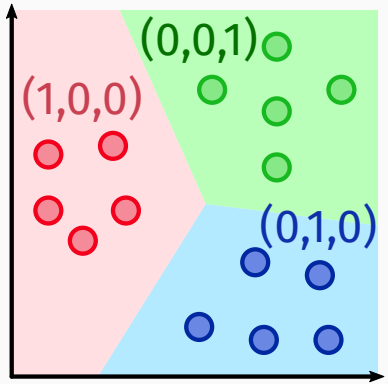
Application 2: Classification = regression in a space of distributions

Logistic regression:

- Encode class labels as **probability distributions** $\delta(x) \in \mathbb{P}(\{1, 2, 3\})$.
- Predict a vector of **scores** $s_i(x)$ at every location x and turn it into a probability distribution using the **SoftMax**: $\alpha(x) = (e^{s_1(x)}, e^{s_2(x)}, e^{s_3(x)}) \,/\, \sum e^{s_i(x)}$
- Minimize the **relative entropy**:

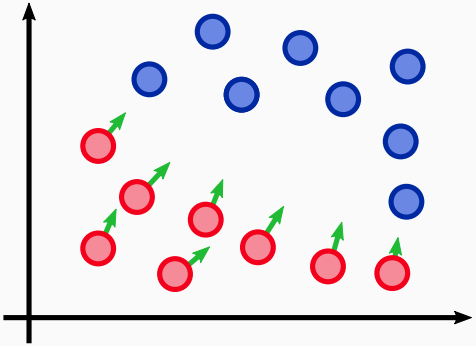
$$\frac{1}{\mathsf{N}} \sum_{i=1}^{\mathsf{N}} \mathrm{KL}(\delta(x), \alpha(x)) \,.$$

Invariant to the **ordering** of the labels.

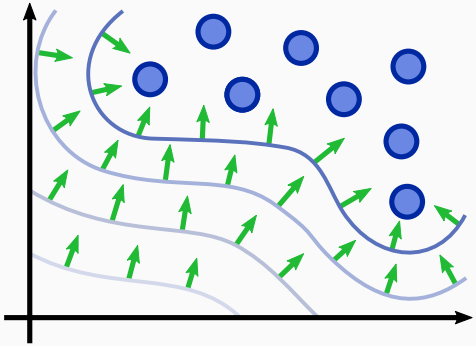


2 input features, 3 classes.

Application 3: Generative modelling

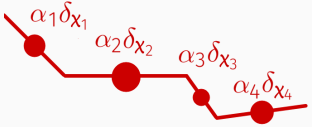


Generative Adversarial Networks and Variational Auto-Encoders **minimize a distance** between a synthetic sample and a reference data sample.

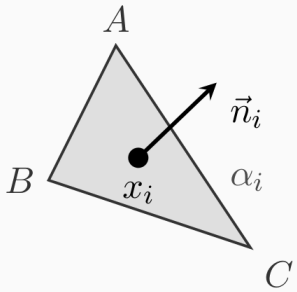


Diffusion and score-based models estimate a gradient of the **distance to the support** of a reference data sample.

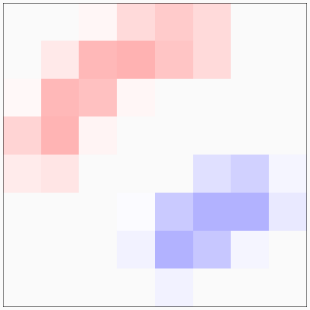
Application 4: Shape registration [KCC17]



Curve:
one weight per **segment**.



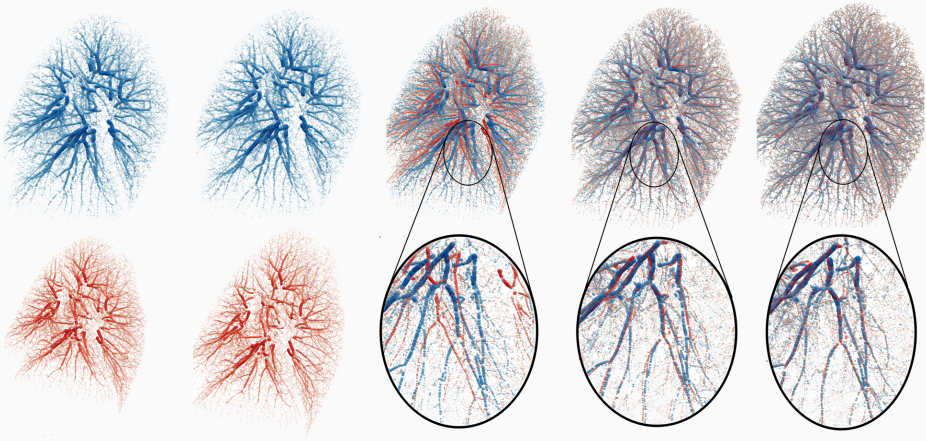
Surface:
one weight per **triangle**.



Segmentation mask:
one weight per **voxel**.

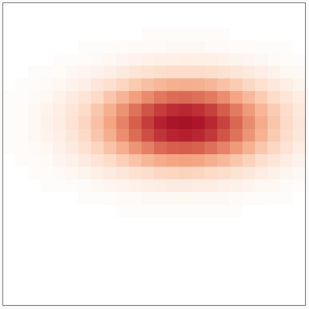
Encoding shapes as distributions guarantees an **invariance to resamplings**.
We may work with **basic** (x, y, z) coordinates or with **better features**.

Application 4: Shape registration [SFL+21]



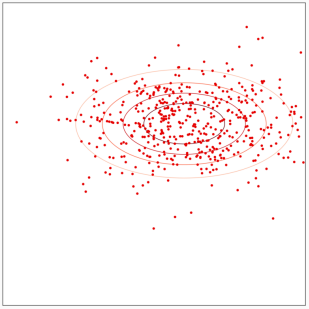
Registration algorithms **minimize a distance**
between a deformable model α and the fixed target β.

A point about implementations



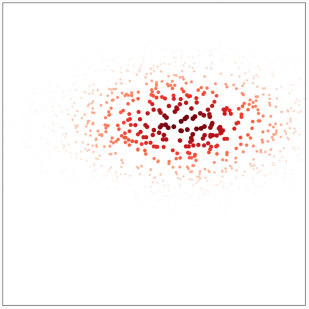
Histogram:

explicit weights a_i,
implicit locations x_i.



Sample:

implicit weights $1/\mathrm{N}$,
explicit locations x_i.

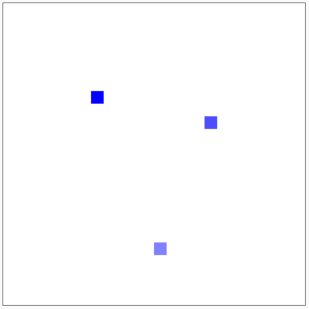


Weighted point cloud:

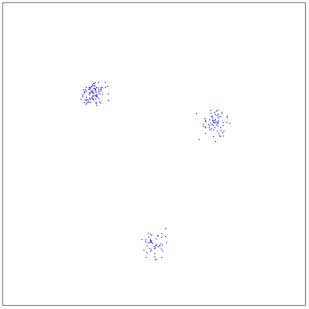
explicit weights a_i,
explicit locations x_i.

Depending on the application, we may choose
a **different encoding** for our distributions.

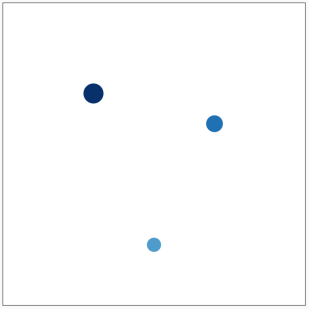
A point about implementations



Histogram:
explicit weights a_i,
implicit locations x_i.



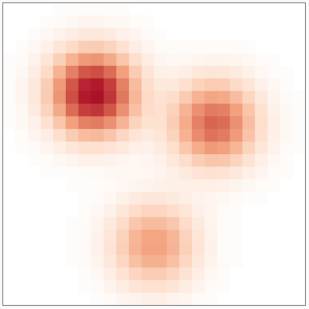
Sample:
implicit weights $1/\mathrm{N}$,
explicit locations x_i.



Weighted point cloud:
explicit weights a_i,
explicit locations x_i.

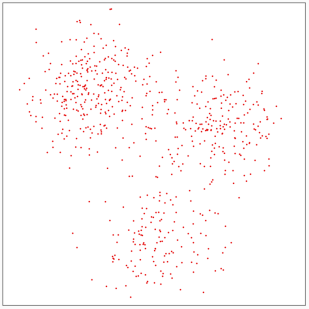
Understanding that **different implementations** correspond to
the same operation is key to insightful research in the field.

A point about implementations



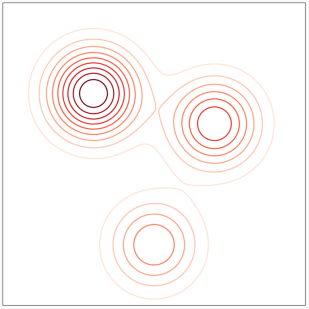
Convolution

of the **density map** $a[i, j]$
with a filter $g[i, j]$.



Additive noise:

$x_i \mapsto x_i + w_i$
where $w_i \sim \mathcal{N}(0, \sigma^2)$.



Soft distance:

log-likelihood $\ell(x) =$
$\log\left(\sum_i a_i e^{-\|x-x_i\|^2/2\sigma^2}\right)$.

Understanding that **different implementations** correspond to
the same operation is key to insightful research in the field.

A point about notations

If $\alpha = \sum_{i=1}^{\mathsf{N}} a_i \delta_{x_i}$ is a **probability distribution**
and $f : x \mapsto f(x) \in \mathbb{R}$ is a **continuous function**,

$$\underbrace{\sum_{i=1}^{\mathsf{N}} a_i f(x_i)}_{\text{Programming}} = \underbrace{\int_x f(x)\,\mathrm{d}\alpha(x)}_{\text{Integration}} = \underbrace{\langle\, \alpha\,,\, f\,\rangle}_{\text{Analysis}} = \underbrace{\mathbb{E}_{X\sim\alpha}[f(X)]}_{\text{Probability}}\,.$$

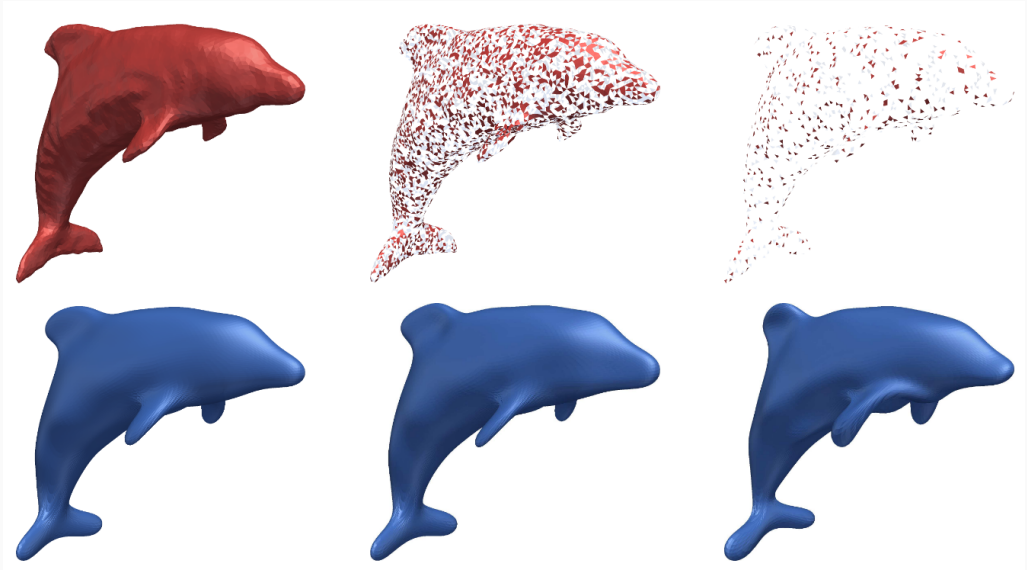
To study **spaces** of probability distributions,
the $\langle\, \alpha\,,\, f\,\rangle$ notation is **superior** as it highlights
the **linearity** with respect to **both** distributions and functions:

$$\begin{aligned}\langle\, \tfrac{1}{2}\alpha + \tfrac{1}{2}\beta\,,\, f\,\rangle &= \tfrac{1}{2}\langle\, \alpha\,,\, f\,\rangle + \tfrac{1}{2}\langle\, \beta\,,\, f\,\rangle\,,\\ \langle\, \alpha\,,\, f+g\,\rangle &= \langle\, \alpha\,,\, f\,\rangle + \langle\, \alpha\,,\, g\,\rangle\,.\end{aligned}$$

On the blackboard

- Kernel norms
- Optimal transport
- Diffeomorphic flows
- Varifolds and other extensions

Robustness to topological noise [KCC17]



Conclusion

- Medical imaging is **well-funded** and **diverse**.
  Much deeper than “simply” training U-Nets to segment 2D slices.
- Ongoing effort to create a convenient toolbox for all **data structures**.

Two major theoretical frameworks:

- **Discrete differential geometry**:
  CPU-friendly, geodesic distances, expects clean topology.
- **Geometric measure theory**:
  GPU-friendly, Euclidean distances, robust to dirty data.

$\Longrightarrow$ To **bridge the gap** between the two, see you on Monday! $\Longleftarrow$

References

References i

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković.

Geometric deep learning: Grids, groups, graphs, geodesics, and gauges.

arXiv preprint arXiv:2104.13478, 2021.

Nick Cammarata, Gabriel Goh, Shan Carter, Ludwig Schubert, Michael Petrov, and Chris Olah.

Curve detectors.

Distill, 2020.

https://distill.pub/2020/circuits/curve-detectors.

References ii

James Dillon.

Molecules.

https://github.com/chemplexity/molecules, 2015.

MIT License.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson.

Benchmarking graph neural networks.

arXiv preprint arXiv:2003.00982, 2020.

Pepe Eulzer, Monique Meuschke, Gabriel Mistelbauer, and Kai Lawonn.

Vessel maps: A survey of map-like visualizations of the cardiovascular system.

In *Computer Graphics Forum*, volume 41, pages 645–673. Wiley Online Library, 2022.

Olivier Ecabert, Jochen Peters, and Matthew Walker.

Segmentation of the heart and great vessels in ct images using a model-based adaptation framework.

Medical Image Analysis, (15):863–876, 2011.

References iv

Martin Grandjean.

Social network analysis visualization.

https://fr.wikipedia.org/wiki/Fichier:Social_Network_Analysis_Visualization.png, 2019.

CC BY-SA 3.0.

Guillaume Houry, Tom Boeken, Stéphanie Allassonnière, and Jean Feydy.

Untangling vascular trees for surgery and interventional radiology.

In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, page to appear. Springer, 2025.

Irene Kaltenmark, Benjamin Charlier, and Nicolas Charon.

A general framework for curve and surface comparison and registration with oriented varifolds.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3346–3355, 2017.

Daniel L. Lu.

Ouster os1-64 lidar point cloud of intersection of folsom and dore st, san francisco.

https://en.wikipedia.org/wiki/File:Ouster_OS1-64_lidar_point_cloud_of_intersection_of_Folsom_and_Dore_St,_San_Francisco.png, 2019.

CC BY 4.0.

Stéphane Mallat.

Understanding deep convolutional networks.

Phil. Trans. R. Soc. A, 374(2065):20150203, 2016.

Tomaso Mansi.

A statistical model for quantification and prediction of cardiac remodelling: Application to tetralogy of fallot.

IEEE transactions on medical imaging, 2011.

Michael M Maher, Susan Kealey, Ann McNamara, Risteard O'Laoide, Robert G Gibney, and Dermot E Malone.

Management of visceral interventional radiology catheters: a troubleshooting guide for interventional radiologists.

Radiographics, 22(2):305–322, 2002.

NoJhan.

Hairy head of a one year old girl.

https://commons.wikimedia.org/wiki/File:Baby_hairy_head_DSCN2483.jpg, 2007.

CC BY-SA 2.0.

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, 2011.

Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov.

Diffusionnet: Discretization agnostic learning on surfaces.

ACM Transactions on Graphics (TOG), 41(3):1–16, 2022.

Zhengyang Shen, Jean Feydy, Peirong Liu, Ariel H Curiale, Ruben San Jose Estepar, Raul San Jose Estepar, and Marc Niethammer.

Accurate point cloud registration with robust optimal transport.

Advances in Neural Information Processing Systems, 34:5373–5389, 2021.

Nicholas Sharp, Yousuf Soliman, and Keenan Crane.

Navigating intrinsic triangulations.

ACM Transactions on Graphics (TOG), 38(4):1–16, 2019.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein.

Understanding over-squashing and bottlenecks on graphs via curvature.

arXiv preprint arXiv:2111.14522, 2021.

John Williamson.

What do numbers look like?

https://johnhw.github.io/umap_primes/index.md.html.

Zygote.

Solid 3d human foot and ankle model.

https://www.zygote.com/cad-models/solid-3d-human-anatomy/cad-human-foot-ankle-model.