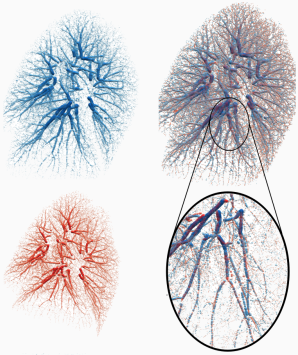
The geometric software stack: past, present, future

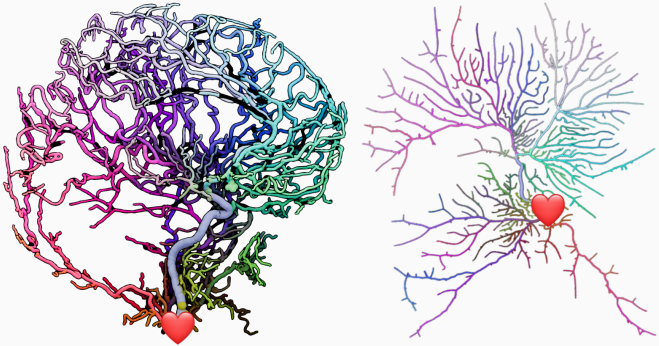
Jean Feydy
HeKA team, Inria Paris
Inserm, Université Paris-Cité

28th of October, 2024
SIGMA 2024
CIRM, Marseille

Recent works



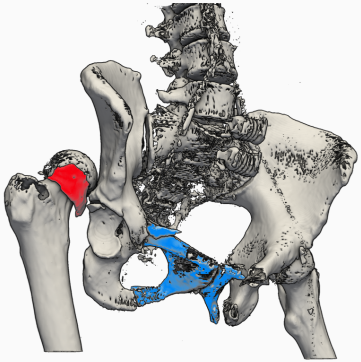
Lung **registration**.



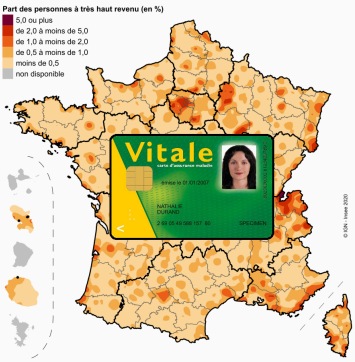
Interventional **radiology**.

$\Longrightarrow$ **Accessible** to you guys, but **barely anyone else**.

Recent works



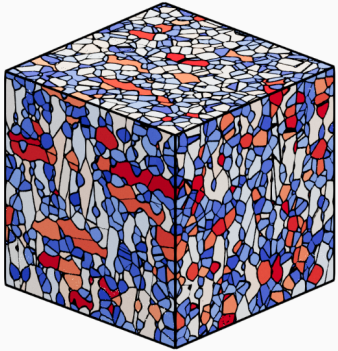
Orthopedic **surgery**.



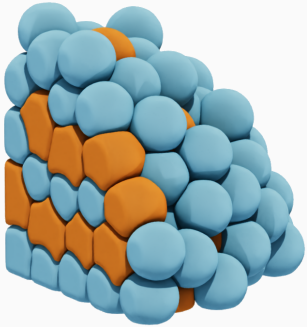
Public health.

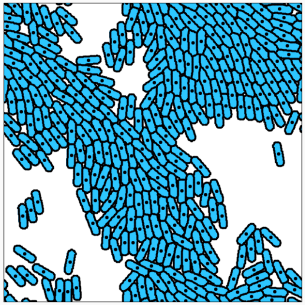
$\Longrightarrow$ **Accessible** to you guys, but **barely anyone else**.

Recent works



Metallurgy.





Swarms of incompressible **cells**.

$\Longrightarrow$ **Accessible** to you guys, but **barely anyone else**.

HeKA : a translational research team for public health

Hospitals

Inria Inserm

Universities

Geometric data analysts are in a delicate position

Our constraints:

1. Differential geometry is **not** part of the **mainstream curriculum**
   $\Longrightarrow$ High **entry cost** for students and users.

2. **Credibility** $\Longleftrightarrow$ **Performance** and high-resolution figures
   $\Longrightarrow$ Constant work to **keep up** with new technology.

3. We are already **very busy**
   $\Longrightarrow$ Our **career incentives** do **not** reward long-term software **maintenance**.

Today's talk – some practical answers to three pressing questions

1. Which **language** and **libraries** should I use?
2. Is my code still going to run in **2030**?
3. How do I get **rewarded** for all of that extra work?

Which language should I use?

The **C++ era** (2000-2015):

- High-performance C++ was **necessary** to handle **3D data**.
- **Monolithic** code-bases with a lot of **inertia**, cryptic to scientists.
- The Visualization ToolKit, the Computational Geometry Algorithms Library…

The **Python era** (since 2015):

- **Modular** and **inter-operable** tools via dictionaries and NumPy arrays.
- **Permissive** open source licences create trust.
- Scikit-learn, Scikit-image, PyVista, Vedo…

Which language should I use?

Domain-specific languages are fine too:

- **R** is data-centric: native idiom for biologists and medical doctors.
- **Julia** is convenient for numerical analysis.

But **Python** is the **lingua franca** for **gluing** pipelines together:

1. Identify the key **building blocks** in your method.
2. Implement them in the language that suits you best.
3. Write a **Python interface** – now super easy.

$\Longrightarrow$ Speak **French**, **German** or **Hindi** at home… but publish in **English**.

The KeOps library: efficient support for symbolic matrices, with Joan and Benjamin

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.

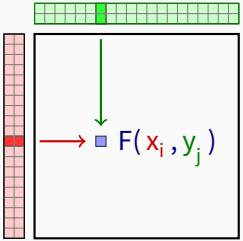
If the formula "F" is simple ($\leqslant$ 100 arithmetic operations):

"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.

Yet another python compiler?

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

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

KeOps fills a **scientific niche** (like FFT libraries):

- 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** (**700k+** downloads).

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

Which libraries should I use?

Exciting libraries get **killed** all the time **:-(**

Theano (2008-2017):

- **Pioneering** deep learning library: Python + Autodiff + GPU.
- Created and maintained in Montreal (MILA).
- Development stopped when **PyTorch** became available.

Taichi (2017-2023?):

- **Awesome** Python dialect for 3D shape processing and graphics, 25k GitHub stars.
- **PhD thesis** of Yuanming Hu at MIT, now **CEO** of Meshy.
- Active development stopped in the summer of 2023.

Tip #1: look at the developers' long term incentives

PyTorch (Meta) – sending all the **right signals**:

- Business strategy on AI is to make it an **open source commodity**.
- **Transparent governance structure**, PyTorch foundation.
- **Extensive** internal documentation.

JAX and TensorFlow (Google) – several **red flags**:

- Business strategy on AI is to protect the **Google search monopoly** and **GCP**.
- **Opaque** governance structure, `killedbygoogle.com`.
- **Minimal** internal documentation.

Tip #2: Implement a future-proof interface

Insulate users from deprecations:

- Numpy arrays.
- Human-readable files.

User-centric design:

- Principle of **least surprise**.
- Write **tutorials** – a feature that is not documented **does not exist**.
- Plain, descriptive names:

Kernel $\longrightarrow$ `covariance`

Splines $\longrightarrow$ `deformation(covariance="thin plate spline")`

LDDMM $\longrightarrow$ `deformation(covariance="gaussian", scale=2, n_steps=10)`

Problem: software rots

Some personal nightmares:

- CMake, **Boost**…
- Nvidia **actively deprecates** “old” GPUs.
- `torch.solve(A, B)` $= B^{-1}A \to A^{-1}B$.

Without **constant gardening**,
software breaks after 3-5 years.

Some of my old GitHub repositories.

Research you're proud of should be in a library

#1 – Include your model in a **pre-existing** library:

- **Outsource maintenance**, gain visibility.
- Permissive licenses are key: MIT, BSD…

#2 – Develop and maintain **your own** library:

- Be realistic: focus on your **core expertise**.
- Bet on **interoperability** with other packages.
- Freedom for you, minimize risk for users.

$\Longrightarrow$ Agree on a **consistent interface** with the community and **keep your word**.

*A **professional** storage facility.*

GeomLoss: scaling up optimal transport to anatomical data

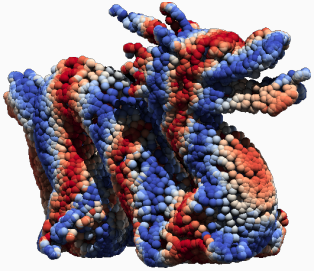
Progresses of the last decade add up to a $\times$**100** - $\times$**1000** acceleration:

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

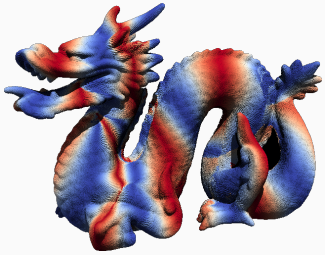
With a precision of 1%, on a modern gaming GPU:

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

$\Longrightarrow$



10k points in 30-50ms



100k points in 100-200ms

GeomLoss: going forward

Current landscape in computational optimal transport:

- **Python Optimal Transport (POT)**: tons of tutorials, but slow solvers from 2015.
- Mérigot, Lévy, De Goes: super-fast OT solvers for **physics**.
- Schmitzer, GeomLoss: super-fast OT solvers for **geometric data**.
- Massive **waste of time for newcomers** in the field.

How to solve the issue:

- Agree on a **common interface**.
- Include GeomLoss and others as **optional backends** in POT.
- **Automated benchmark** website to highlight “solved” and “open” problems.

$\Longrightarrow$ Put egos aside, move forward as a community.

$\longrightarrow$ Only possible because we are not judged by our h-index.

But Jean… I'm just a mathematician!

Writing good code is easy now!
Use **professional tools**:

- Black and Ruff **beautify** your code.
- Pytest and Hypothesis **find bugs**.
- Copilot **writes documentation**.
- Sphinx creates a **clean website**.
- GitHub actions **deploy automagically**.

Check out `scientific-python.org`.

*Invest in **power** tools.*

But Jean… I’m just a mathematician!

Why should I bother?

- If you don’t code your method first, **no one will**.
- Get to meet a wide range of **exciting users**.
- Open up **career paths** for students.

Publish or perish?

- French open source software awards from the Ministry of research.
- At INRIA, **clear incentives** for software development.
- Career paths for **research engineers** in academia?

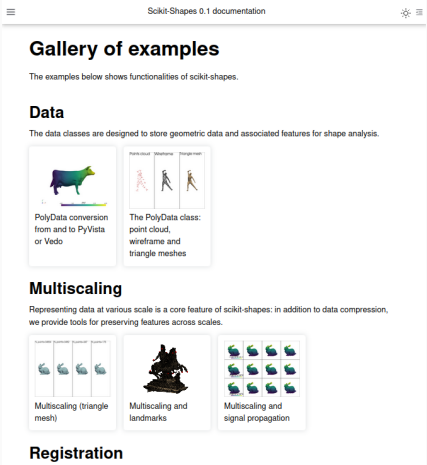
From Deformetrica to scikit-shapes (with an ‘s’)

scikit-shapes:

- Follows the tips above!
- Named after **scikit-image**: a reference library for classical image processing.
- Targets shape data analysis.
- **Abstracts** multiscaling and feature extraction.
- **Foundations** are now solid (Louis Pujol).
- Funded by INRIA and Prairie.

Next steps:

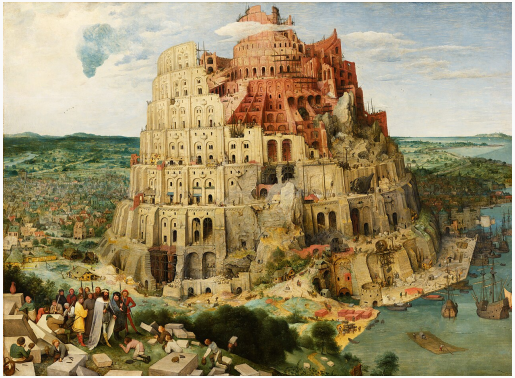
- GPMM, elastic metrics, functional maps.
- **Research** on robustness and modularity.
- Ready for JupyterLite + WebGPU ?



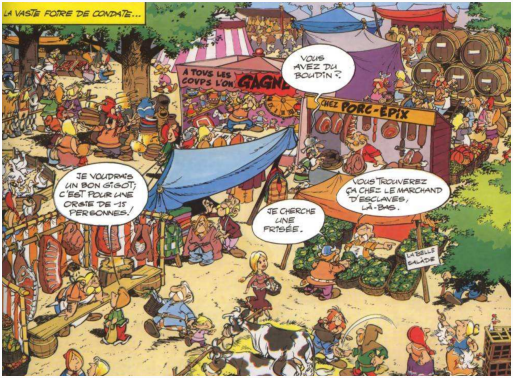
scikit-shapes.github.io

Check it out in 2025!

Our community is judged by its software output



The C++ tower of Babel.



The Python market of ideas.

Major challenge: beyond goodwill, create **sustainable open business models**.

Are **universities** hostile environments? Kitware (VTK), Tutte Institute (UMAP), INRIA…

Documentation and tutorials are available online

⟹ shape-analysis.github.io ⟸

Monthly seminar, videos on YouTube.

References

References i