Scikit-Shapes, a transparent package for statistical shape analysis

Jean Feydy, Louis Pujol

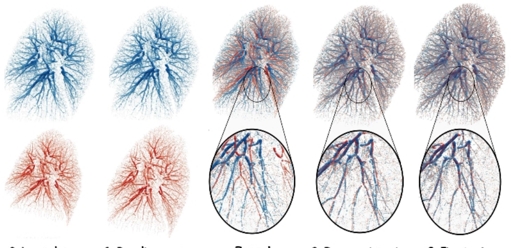
HeKA team, Inria, Inserm, Université Paris-Cité

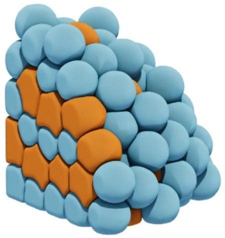
February 29, 2024

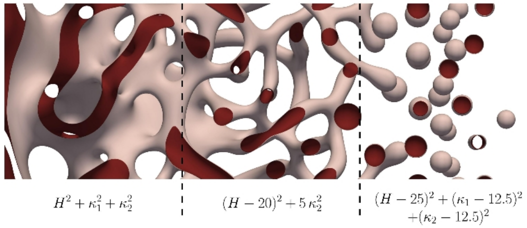
Outline

Table of Contents

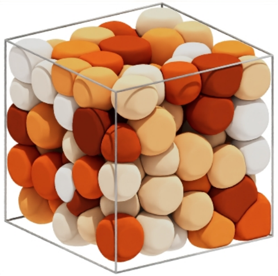
The barrier of entry to shape analysis is too high







$H^2 + \kappa_1^2 + \kappa_2^2$
$(H - 20)^2 + 5\,\kappa_2^2$
$(H - 25)^2 + (\kappa_1 - 12.5)^2 + (\kappa_2 - 12.5)^2$



Nice pictures... But none of these models is accessible to a non-specialist.

Getting credible performance is easier than ever

The C++ era (2005-15):

- Writing C++ by hand was necessary to get decent run times.
- Monolithic code-bases with a lot of **inertia**, cryptic to scientists.

The Python era (2015-25):

- **Modularity** via the Numpy arrays and Torch tensors.
- **Permissive** open source licences.

The Web era (2025-?):

- **Portable**, real-time 3D performance on modern web browsers.
- The Python stack is being ported: check out **JupyterLite**!

The scikit-shapes project

The necessary low-level bricks are now stable: PyVista, KeOps, etc.
The ecosystem feels mature for a modern **statistical shape analysis** toolbox, implemented in Python:

- **scikit-image** as the reference target.
- Short-term: fill the Deformetrica niche.
- Medium-term: a modular and **maintainable** platform.
- Long-term: foster cross-pollination with other communities.

Funding comes from Prairie and Inria – long-term support is realistic.

Today: some feedback after one year working on the project full-time.

Table of Contents

Update on Scikit-Shapes

Where we are:

- First features have been written for data manipulation (limited to points clouds, meshes), multiscaling and registration
- Code architecture have been set up.
- The development version of scikit-shapes can be installed, instructions and examples are available here

What will be done in the next few weeks:

- Documentation publication
- Package publication to PyPI

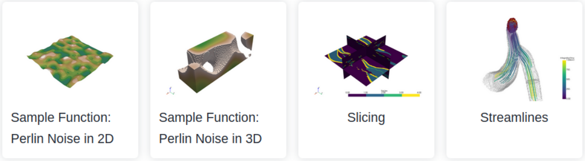
Table of Contents

Table of Contents

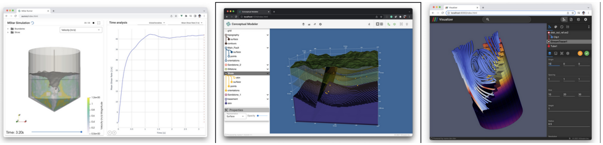
Recent efforts to export Pyvista/VTK viewers to webpages

Thanks to recent advances in the VTK ecosystem for web 3D visualization, some features are becoming more accessible for python developers:

- Generate galleries of examples like in the Pyvista documentation

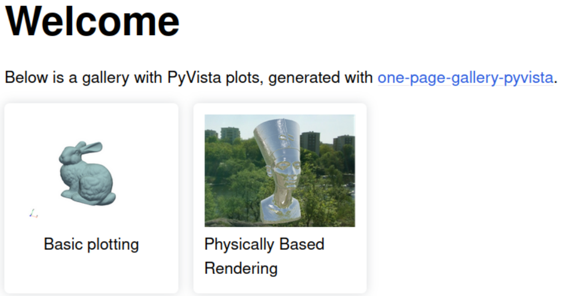

Sample Function: Perlin Noise in 2D
Sample Function: Perlin Noise in 3D
Slicing
Streamlines

- Generate interactive applications with Trame



One-page-gallery-pyvista

Tool to easily generate a webpage with a gallery including PyVista plots:



Welcome

Below is a gallery with PyVista plots, generated with one-page-gallery-pyvista.

Goals:

- Easily expose notebooks in a webpage
- Share the code: expose the necessary packages to run the examples, make the code downloadable

Repository: https://github.com/Louis-Pujol/one-page-gallery-pyvista

Table of Contents

Scientific-python

Scientific-python is a federation of open-source python developers that maintains tutorials about good practices for python scientific programming. You can find on their website:

- A development guide
- A repo-review plugin.
- The sp-repo-review plugin can be test direclty here

Install the sp-repo-review pre-commit hook

- Create a .pre-commit-config.yaml file at the root of your folder with the content :

```
repos:
  - repo: https://github.com/scientific-python/cookie
    rev: 2024.01.24
    hooks:
      - id: sp-repo-review
```

- install pre-commit (pip install pre-commit) and run:

```
pre-commit install
pre-commit run --all-files
```

Table of Contents

Type annotations

Since python 3.6, type annotations can be added to the functions definitions (and more, see PEP526):

```
def sum(a: int, b: int) -> int:
    return a + b
```

Type annotations have no impact on the behavior of the code. They can be interpreted by third-party tools:

- Documentation generators
- Static type checkers: read the codebase without running it and detect types errors
- Runtime type checkers: check arguments types at each call of a function

In Scikit-shapes we do runtime typecheckings. We use beartype as a type checker and jaxtyping to generate types for tensors.

Beartype/Jaxtyping example - 1

```python
from beartype import beartype
from jaxtyping import Float32, jaxtyped
import torch

Points = Float32[torch.Tensor, "_ d"]
TranslationVector = Float32[torch.Tensor, "d"]

@jaxtyped(typechecker=beartype)
def translate(
    points: Points,
    t: TranslationVector
    ) -> Points:
    return points + t
```

Beartype/Jaxtyping example - 2

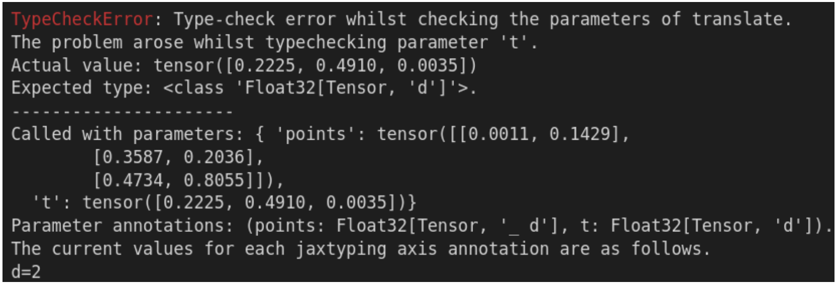
This will work:

```
p, t = torch.rand(3, 2), torch.rand(2)
translate(points=p, t=t)
```

But this will fail:

```
p, t = torch.rand(3, 2), torch.rand(3)
translate(points=p, t=t)
```

Output:



```
TypeCheckError: Type-check error whilst checking the parameters of translate.
The problem arose whilst typechecking parameter 't'.
Actual value: tensor([0.2225, 0.4910, 0.0035])
Expected type: <class 'Float32[Tensor, 'd']'>.
----------------------
Called with parameters: { 'points': tensor([[0.0011, 0.1429],
        [0.3587, 0.2036],
        [0.4734, 0.8055]]),
  't': tensor([0.2225, 0.4910, 0.0035])}
Parameter annotations: (points: Float32[Tensor, '_ d'], t: Float32[Tensor, 'd']).
The current values for each jaxtyping axis annotation are as follows.
d=2
```

Interest of runtime type checking

- **For users**: prevent bad usage and provide clear error messages
- **For developers**: write cleaner functions with less assertions at the beginning, ensure the coherence of the data flow through different parts of the codebase

End of the presentation

Thank you for your attention. Do you have any question ?