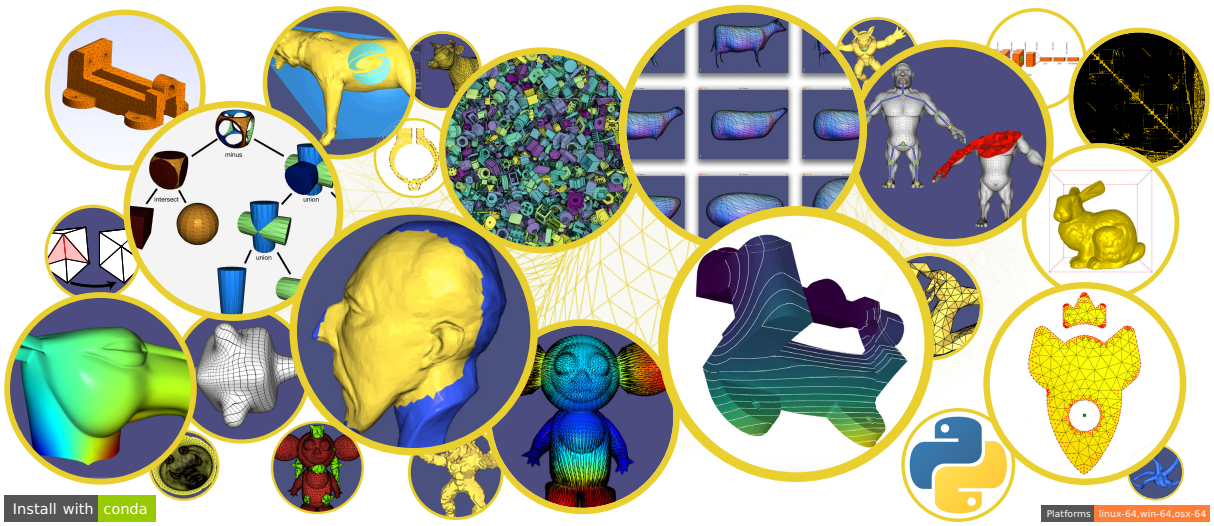
Black Box Geometric Computing with Python

From Theory to Practice

https://geometryprocessing.github.io/blackbox-computing-python

S. Koch[1], T. Schneider[2], C. Li[2], and D. Panozzo[2]

[1]TU Berlin, Germany
[2]NYU Courant Institute, USA



Abstract

The first part of the course is theoretical, and introduces the finite element method trough interactive Jupyter notebooks. It also covers recent advancements toward an integrated pipeline, considering meshing and element design as a single challenge, leading to a black box pipeline that can solve simulations on ten thousand in the wild meshes, without any parameter tuning.

In the second part we will move to practice, introducing a set of easy-to-use Python packages for applications in geometric computing. The presentation will have the form of live coding in a Jupyter notebook. We have designed the presented libraries to have a shallow learning curve, while also enabling programmers to easily accomplish a wide variety of complex tasks. Furthermore, these libraries utilize NumPy arrays as a common interface, making them highly composable with each-other as well as existing scientific computing packages. Finally, our libraries are blazing fast, doing most of the heavy computations in C++ with a minimal constant-overhead interface to Python.

In the course, we will present a set of real-world examples from geometry processing, physical simulation, and geometric deep learning. Each example is prototypical of a common task in research or industry and is implemented in a few lines of code. By the end of the course, attendees will have exposure to a swiss-army-knife of simple, composable, and high-performance tools for geometric computing.

CCS Concepts

• ***Mathematics of computing*** → ***Numerical analysis;*** • ***Theory of computation*** → ***Computational geometry;*** • ***Computing methodologies*** → ***Machine learning; Scientific visualization; Mesh models; Shape analysis;*** • ***Applied computing*** → ***Computer-aided design;***

DOI: 10.2312/egt.20201000

Full day tutorial, 4×90 minutes

1. Presenter Details

Sebastian Koch s.koch@tu-berlin.de.
Sebastian Koch is a Computer Science PhD student at Technische Universitaet Berlin, Germany. His research interests are in the fields of Geometric Machine Learning, Geometry Processing and Meshing as well as Digital Fabrication. Sebastian is one of the maintainers of the libigl python bindings (https://libigl.github.io). He maintains the ABC Dataset for Geometric Deep Learning (https://deep-geometry.github.io/abc-dataset) and is the developer of Meshplot (https://skoch9.github.io/meshplot), a library for the visualization of point clouds and meshes in Jupyter Notebooks.

Teseo Schneider teseo.schneider@nyu.edu, https://cs.nyu.edu/~teseo/.
Teseo Schneider is an assistant professor/faculty fellow in Computer Science at the Courant Institute of Mathematical Sciences in New York University. Teseo earned his PhD in Computer Science from the Università della Svizzera italiana (2017) with the thesis entitled "Theory and Applications of Bijective Barycentric Mappings". He earned a Postdoc.Mobility fellowship by Swiss National Science Foundation (SNSF) to pursue his research aiming to bridge physical simulations and geometry. His research interests are in finite element simulations, mathematics, discrete differential geometry, and geometry processing. Teseo is the main developer of Polyfem (https://polyfem.github.io/) a flexible and easy to use Finite Element Library, he is one of the maintainers of the libigl python bindings (https://github.com/libigl/libigl), and contributor to wild meshing (https://github.com/wildmeshing), a 2D and 3D robust meshing library

Chengchen Li cl3940@nyu.edu.
Chengchen Li is an undergraduate student at New York University majoring in computer science and math. His research interests are computer graphics and virtual/augmented/mixed reality. He has experience in system and computer graphics and finished projects like GPU memory swapping and geometric computing with python.

Daniele Panozzo panozzo@nyu.edu, https://cims.nyu.edu/gcl/daniele.html.
Daniele Panozzo is an Assistant Professor of Computer Science at the Courant Institute of Mathematical Sciences in New York University. Prior to joining NYU he was a postdoctoral researcher at ETH Zurich (2012-2015). Daniele earned his PhD in Computer Science from the University of Genova (2012) and his doctoral thesis received the EUROGRAPHICS Award for Best PhD Thesis (2013). His research was awarded the EUROGRAPHICS Young Researcher Award in 2015, the NSF CAREER Award in 2017, and a Sloan Research Fellowship in 2020. Daniele's research group is leading the development of libigl (https://github.com/libigl/libigl), an award-winning (EUROGRAPHICS Symposium of Geometry Processing Software Award, 2015) open-source geometry processing library, polyfem (https://polyfem.github.io), a simple C++ and Python finite element library, and wild meshing (https://github.com/wildmeshing), a 2D and 3D robust meshing library. Daniele initiated the Graphics Replicability Stamp (http://www.replicabilitystamp.org), which is an initiative to promote reproducibility of research results and to allow scientists and practitioners to immediately benefit from state-of-the-art research results. His research interests are in digital fabrication, geometry processing, geometric deep learning, and discrete differential geometry.

2. Outline

The numerical solution of partial differential equations (PDEs) is ubiquitous in engineering applications, for the simulation of elastic deformations, fluids, and other physical phenomena. The finite element method (FEM) is the most commonly used discretization of PDEs due to its generality and rich selection of off-the-shelf commercial implementations. Ideally, a PDE solver should be a "black box": the user provides as input the domain boundary, boundary conditions, and the governing equations, and the code returns an evaluator that can compute the value of the solution at any point of the input domain. This is surprisingly far from being the case for all existing open-source or commercial software, despite the research efforts in this direction and the large academic and industrial interest.

To a large extent, this is due to treating meshing and FEM basis construction as two disjoint problems. The FEM basis construction may make a seemingly innocuous assumption (e.g., on the geometry of elements), that lead to exceedingly difficult requirements for meshing software.

This state of matters presents a fundamental problem for applications that require fully automatic, robust processing of large collections of meshes of varying sizes, an increasingly common situation as large collections of geometric data become available. Most importantly, this situation arises in the context of machine learning on geometric and physical data, when one can run large numbers of simulations to learn from, as well as problems of shape optimization, which require solving PDEs in the inner optimization loop on a constantly changing domain.

In other fields, such as machine learning, the availability of easy-to-use libraries dramatically pushed the state of the art, allowing researchers and practitioners to fast prototype algorithms in a few lines of code. This is unfortunately not yet the case for geometric computing and FEM, and our goal in this course is to bridge this gap, introducing our fully automatic, easy-to-use framework to prototype applications at the intersection between graphics, machine learning, FEM, and scientific computing.

The first part of the course is theoretical, and introduces the finite element method trough interactive Jupyter notebooks (Figure 1). It also covers recent advancements toward an integrated pipeline, considering meshing and element design as a single challenge, leading to a black box pipeline that can solve simulations on ten thousand in the wild meshes, without any parameter tuning.

In the second part we will move to practice, introducing a set of easy-to-use Python packages for applications in geometric computing. The presentation will have the form of live coding in a Jupyter notebook (Figures 2 and 3). We have designed the presented li-

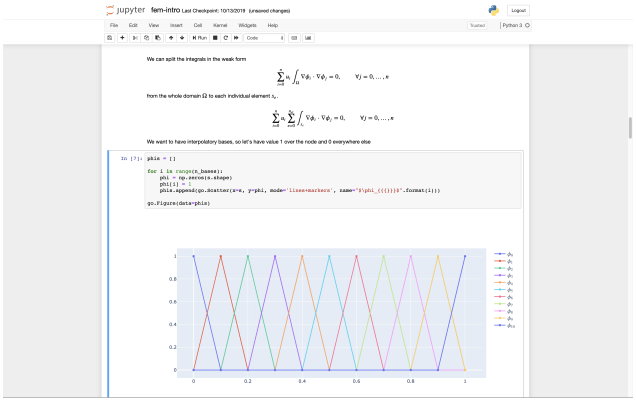


Figure 1: *Example of course material for the introduction to finite elements.*

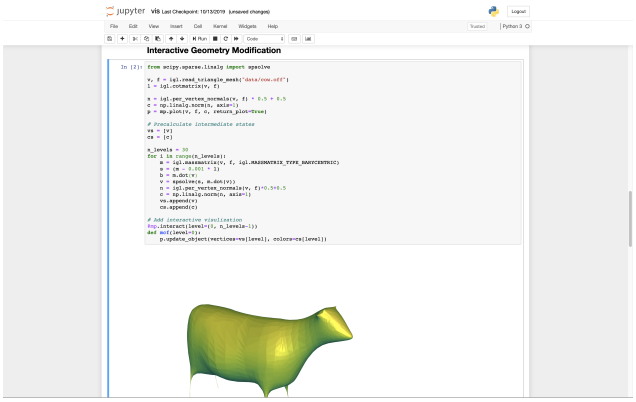


Figure 2: *Interactive geometry processing in Python.*

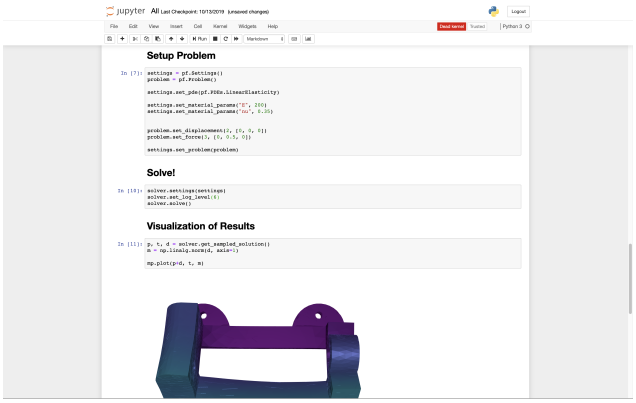


Figure 3: *Example of easy-to-use FEM code.*

braries to have a shallow learning curve, while also enabling programmers to easily accomplish a wide variety of complex tasks. Furthermore, these libraries utilize NumPy arrays as a common interface, making them highly composable with each-other as well as existing scientific computing packages. Finally, our libraries are blazing fast, doing most of the heavy computations in C++ with a minimal constant-overhead interface to Python.

In the course, we will present a set of real-world examples from geometry processing, physical simulation, and geometric deep learning. Each example is prototypical of a common task in research or industry and is implemented in a few lines of code. By the end of the course, attendees will have exposure to a swiss-army-knife of simple, composable, and high-performance tools for geometric computing.

The main course material is available at
https://geometryprocessing.github.io/blackbox-computing-python

3. Code Example

```python
import numpy as np
import igl
import meshplot as mp
import wildmeshing as wm
import polyfempy as pf

# Read an obj and plot, igl
V, F = igl.read_triangle_mesh("mesh.obj")
mp.plot(V, F, shading={"wireframe": True})

# Tetrahedralize with wildmeshing
wm.tetrahedralize("mesh.obj", "out.mesh",
    mute_log=True)


# Numerical simulation with Polyfem
solver = pf.Solver()
solver.load_mesh_from_path("out.mesh")
solver.set_boundary_side_set_from_bary(sideset)

settings = pf.Settings(
    pde=pf.PDEs.LinearElasticity)
settings.set_material_params("E", 200)
settings.set_material_params("nu", 0.35)

problem = pf.Problem()
problem.set_displacement(2, [0, 0, 0])
problem.set_force(3, [0, 0.5, 0])
settings.problem = problem

solver.settings(settings)
solver.solve()

# Plotting solution with meshplot
p, t, d = solver.get_sampled_solution()
m = np.linalg.norm(d, axis=1)

mp.plot(p+d, t, m)
```

4. Schedule

Full day tutorial, 4×90 minutes

- **Theory** 2×90 minutes
  - Introduction
  - FEM Basics
  - Black Box Geometric Computing
  - Intro to a set of libraries with bindings: igl, triwild, tetwild, and polyfem
  - Open Q&A
- **Practice** 2×90 minutes
  - IGL
  - Meshing and Analysis
  - CAD processing for Deep Learning Applications
  - Concluding Remarks and Open Q&A

5. Intended Audience

This course is intended for students and researchers interested in prototyping cutting-edge simulation and geometry processing algorithms. Whether just starting graduate school, supervising an academic lab, or building tools for industry, this course will introduce fast prototyping for advanced techniques with a few lines of python!

6. Prerequisites

Attendees should have a firm understanding of undergraduate linear algebra and calculus. Previous experience with Computer Graphics, Geometry Processing, and Partial Differential Equations is recommended, but not required. A basic knowledge of Python is necessary.

7. Previously Held Tutorials

Parts of this tutorial were presented at:

- SIGGRAPH 19, https://geometryprocessing.github.io/geometric-computing-python/, https://dl.acm.org/citation.cfm?id=3328067
- IMR 19, https://teseoch.github.io/blackbox-course-imr/