XLVI CILAMCE
VITÓRIA - ES - BRAZIL - 2025
XLVI IBERO-LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING

A Python interface for DOOLINES: a framework for nonlinear static and dynamic analysis of offshore lines

Erasmo A. B. Silva[1], Fábio M. G. Ferreira[1], Eduardo N. Lages[1]

[1]*Laboratory of Scientific Computing and Visualization, Federal University of Alagoas*
Avenida Lourival Melo Mota, S/N, Tabuleiro dos Martins, 57072-970, Maceió/Alagoas, Brazil
erasmo.bezerra@lccv.ufal.br, fabio.ferreira@lccv.ufal.br, enl@lccv.ufal.br

Abstract. DOOLINES is a finite element-based framework for nonlinear static and dynamic analysis of mooring lines and risers. By modeling structural configurations and environmental conditions, the framework computes force and displacement histories using time integration methods. Developed in C++ with object-oriented programming, DOOLINES can be used as a standalone system or integrated into other platforms, such as DYNASIM (developed by PETROBRAS), where it has supported the design of offshore mooring systems. To enhance accessibility and broaden its applicability, a Python interface was developed through a wrapper, preserving the high performance and usability of the underlying framework. This interface simplifies access to the framework's functionalities, enabling the analysis of complex systems with just a few lines of code and facilitating integration with visualization libraries and other analysis tools. The presented case studies demonstrate the effectiveness of the developed interface in modeling and analyzing offshore systems, highlighting its computational efficiency and user-friendliness for offshore engineering applications.

Keywords: offshore systems, line dynamics, Python wrapper.

1 Introduction

Offshore structures support the exploration and production of oil, gas and renewable energy in marine environments exposed to harsh conditions, including ocean currents, waves, and winds. Several of these components are modeled as cable-like structures to assess their mechanical behavior. Among these, mooring lines play a crucial role in maintaining the position of floating platforms and ensuring structural integrity during drilling, production, offloading, and offshore wind power generation. Other relevant slender structures include risers, pipes transporting fluids between the reservoir and the surface, as well as umbilicals–cables that provide essential services to subsea facilities, including control signals, electric power, and hydraulic fluids [1].

Structural analysis of these elements typically incorporates dynamic effects and nonlinearities in geometry and material. Finite element method (FEM)-based tools are commonly employed, with options for coupled or decoupled analysis relative to the floater. Examples include OrcaFlex [2], DeepLines [3], Flexcom [4] and DYNASIM [5], developed by PETROBRAS. The framework DOOLINES (dynamics of offshore lines) [6] has been integrated into the DYNASIM to enable FEM-based modeling of lines in coupled analysis of moored or dynamically positioned (DP) vessels, single point mooring (SPM) systems, turret mooring, DICAS (differentiated complacent anchoring system), and offloading operations [7].

In addition to its use as a software component, DOOLINES can also be easily used as a standalone tool. Although it provides high performance and flexibility for developing complex systems, its accessibility remains limited for researchers, engineers, and developers using programming languages other than C++. In this context, Python stands out as one of the most widely adopted languages in the scientific community due to its simplicity, extensive library ecosystem, and strong interoperability. In this paper, we present a Python interface for DOOLINES, developed through a Python wrapper. This solution enables access to the framework's capabilities from Python scripts, preserving the high performance and expanding the reach and usability of DOOLINES.

2 DOOLINES

The framework DOOLINES, developed in C++, is designed to analyze the nonlinear static and dynamic behavior of slender marine structures in the time domain. These structures include, among others, flexible risers, steel catenary risers, umbilicals, floating hoses, and mooring lines [6]. The process involves numerical modeling of the structure and its environment, including self-weight, hydrodynamic forces caused by currents and waves, soil-structure interaction, prescribed forces and displacements. The dynamic problem, defined by the equation of motion along with appropriate boundary and initial conditions, is solved using FEM and temporal integration methods [8]. After the simulation, time-series responses are provided, namely position, tension, bending, local and global forces.

Table 2 summarizes the models and capabilities available in DOOLINES.

Table 2. Overview of DOOLINES modeling approaches and numerical methods.

Feature	Properties	Description
FEM	*BTruss* element	Special truss element with only translational degrees of freedom that indirectly accounts for bending stiffness. It is suitable for discretizing mooring lines and risers when torsional stiffness can be neglected and full clamping is not required [9].
	BeamCR element	Beam element with six degrees of freedom per node, based on the corotational theory, suitable for simulations involving significant torsional effects and for analyzing clamped configurations, such as riser connections to floating units [9].
Constitutive models	Hooke	Used for materials exhibiting purely elastic behavior.
	Kelvin	Represents a simplified viscoelastic behavior by combining an elastic spring and a viscous damper in parallel.
	Zener	Incorporates two springs and a damper to provide a more accurate representation of viscoelastic behavior. It is particularly suitable for simulating polyester line segments.
Hydrodynamic forces	Morison formulation	Allows the modeling of complex fluid-structure interaction, accounting for fluid inertia and drag forces due to relative motion.
Wave theories	Linear Airy	Provides the fluid kinematics for calculating fluid-structure interaction due to waves. Assumes the water surface undergoes simple harmonic motion and is applicable for small-amplitude waves relative to the wavelength [10].
	Stokes nonlinear	Adds a second harmonic to the linear Airy formulation, allowing a more accurate representation of real waves, particularly for larger wave heights and shallow water [10].
	Irregular	Represents the sea state as the sum of multiple wave fronts with distinct characteristics, whose properties are derived from spectra adjusted to environmental parameters and capturing the observed local frequencies. Spectrum models include JONSWAP, Pierson-Moskowitz, and modified Pierson-Moskowitz [10].
Internal fluid flow	Centrifugal and Corolis forces	Accounts for forces generated by internal fluid flow in structures such as risers and jumpers.
Soil models	Elastic	Represents the soil-structure interaction through a distribution of forces in the transverse and longitudinal directions, corresponding to the restoring forces exerted by the soil.
	Elasto-visco-plastic	Allows the representation of soil-structure interaction forces for sections of line buried in the soil.
Boundary conditions	Buoy and clump	Accounts for forces generated by buoys and clumps, based on their buoyancy and weight characteristics.
	Prescribed forces and displacements	Allows the application of forces and displacements to represent, for example, the effect of the floating structure based on a displacement history, using time-dependent functions.
Time integration methods	Central difference	Explicit second-order accurate time integration algorithm [11].
	Chung Lee	Explicit second-order accurate time integration algorithm that uses numerical dissipation techniques [12].

Dynamic relaxation	Explicit first-order accurate time integration algorithm that solves static or quasi-static problems by introducing artificial inertia and damping. It is suitable for obtaining the initial configuration of the system [13].
Euler	Explicit first-order accurate time integration algorithm [11].
Generalized explicit - α	Explicit second-order accurate time integration algorithm that employs numerical dissipation and velocity predictors, enhancing reliability [6, 14].

3 Methodology

Wrapping C++ with SWIG

SWIG (Simplified Wrapper and Interface Generator) is a specialized and extensible development tool, developed in C++, that automates the task of integrating compiled code with scripting language interpreters. It was created to address a common challenge in the scientific community, which is integrating compiled code (written in C, C++, and Fortran) with scripting languages such as Python and Ruby, allowing scientists to more easily develop software that integrates capabilities such as data analysis, visualization, database management, and networking [11]. Examples reported in the literature include OrcaFlex [15], Pybel [16], PyKaldi [17], and PyMAGMA [18].

To improve the usability of the proposed Python interface, new functions and classes were specifically developed in C++ using DOOLINES and made available to Python via SWIG, rather than simply binding the original framework. This approach ensures a more intuitive and efficient experience for Python users.

The starting point for generating the wrapper is the interface file (.i). In it, the module is declared, the necessary headers are included to expose the framework's classes and functions, and special SWIG directives–such as *typemaps*–are applied to customize type conversion between C++ and Python. One example is a *typemap* specification that defines how the composite object *std::pair<Model*, Simulation*>* should be returned to Python, encapsulated as a typed pointer. From the interface file, SWIG automatically generates the C++ wrapper code (.cxx) that bridges the Python interpreter with functions and classes of DOOLINES. This generated code is then compiled and linked with the framework's core to produce a Python dynamic-link library (.pyd) module, which is then packaged into a Wheel (.whl) file for easy distribution and installation.

The Python interface for DOOLINES

The workflow starts with the structural modeling. The *Model* class manages this process by creating all objects required to represent the system.

Figure 1 shows a simplified UML (unified modeling language) diagram.

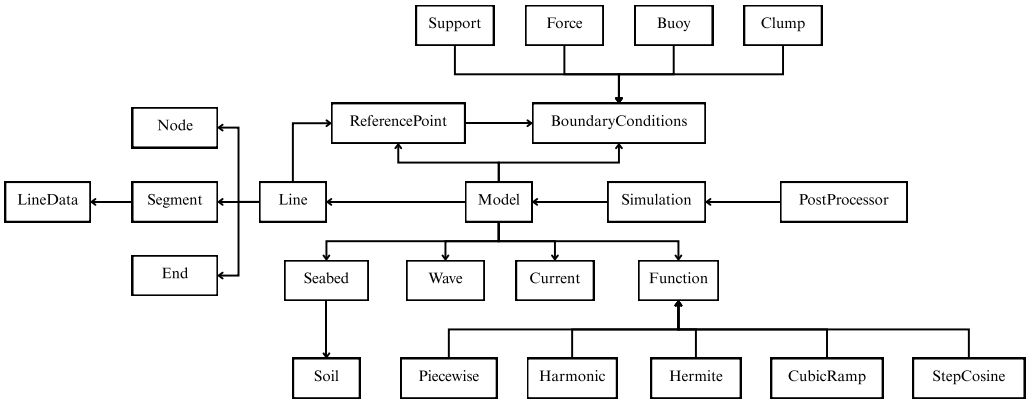


Figure 1. Class relationships in the Python interface for DOOLINES.

These include geometry (*Line*, *Segment* and *Node* class), where the node mesh can either be user-provided or automatically generated from vertex coordinates according to Pasquetti [19]. The *Model* class also handles physical properties (*LineData* class), boundary conditions (represented through the abstract *BoundaryConditions* class and applied via the *ReferencePoint* class), and environmental conditions (via the *Current* and *Wave* classes). The *Seabed* class defines the seabed plane, and the *Soil* class describes the soil properties. The abstract *Functions* class manages time-varying loads and displacements through interpolation. The *Simulation* class handles solving the model using time integration methods and allows restarting simulations from the final conditions (displacements, velocities, and accelerations) of previous analyses, which is particularly useful when the initial configuration is obtained through dynamic relaxation. Results are processed and exported by the *PostProcessor* class in JSON, MsgPack, and XMF formats, with the latter used for graphical visualization in ParaView [20].

4 Case studies

This section presents two case studies. The first case study, originally described by Silveira et al. [6], considers a simple mooring line, with the corresponding script provided. The second case study, representing a more realistic structure, includes multiple line segments and applies a prescribed top-line motion generated by DYNASIM.

Case 1

The first case study involves a mooring line with a harmonic prescribed displacement at the top, a buoy and the effect of ocean current. The model is schematically illustrated in Fig. 2. The physical and geometrical properties of the 100 *BTruss* elements are summarized in Tab. 2.

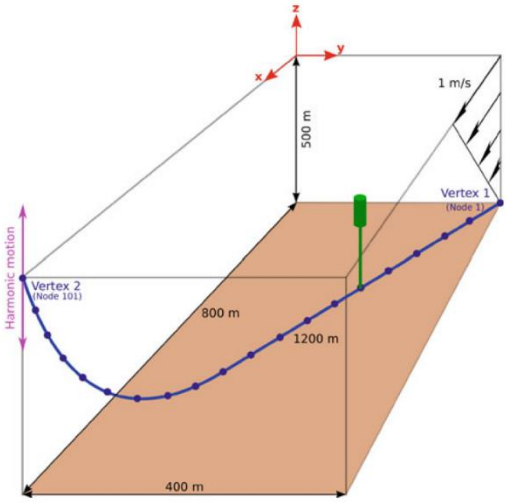


Figure 2. Simple case scheme [6].

Table 2. Physical and geometrical properties [6].

Property	Value
Initial length	1200.0 m
External diameter	0.1 m
Internal diameter	0.0 m
Hydrodynamical diameter	0.1 m
Axial stiffness	450000.0 kN
Water weight	1.2 kN/m
Air weight	1.0 kN/m
Normal drag coefficient	1.7
Axial drag coefficient	0.085
Additional mass coefficient	1.0
Static friction coefficient	0.5
Dynamic friction coefficient	0.3

The script presented in List. 1 initializes a model and defines a single line segment with specified properties. The mesh of nodes is generated based on the coordinates of the initial and final vertices and the number of elements. Harmonic displacement is applied at the final end of the line (vertex 2) in the Z direction, while rotational degrees of freedom around the axes remain free at both ends. A buoyancy load is applied in a reference point at a relative position of 0.35 along the line. The seabed is modeled considering depth and inclination parameters, and a two-point current profile is defined. Soil properties, as well as simulation parameters, such as simulation time and critical time step percentage, are kept at their default values. After running the simulation, the results are exported in both XMF and JSON formats, along with the model data in JSON format. This example demonstrates how the proposed tool enables the evaluation of a mooring line structure through concise and readable code.

```python
# simple_case.py
import doolines

doolines.set_log_severity(doolines.VERBOSE_LOG)
doolines.init("simple_case", "./")

model = doolines.create_model()
line = model.create_line()
line.set_num_segments(1)

line.seg(1).num_elements = 100            # number of elements
line.seg(1).length = 1200.0               # length of segment
line.seg(1).line_data.EA = 450000.0       # axial stiffness
line.seg(1).line_data.de = 0.1            # external diameter
line.seg(1).line_data.di = 0.0            # internal diameter
line.seg(1).line_data.dc = 0.1            # contact diameter
line.seg(1).line_data.dan = 0.1           # normal drag diameter
line.seg(1).line_data.daa = 0.1           # axial drag diameter
line.seg(1).line_data.mass = 1/9.806      # mass in ton/m
line.seg(1).line_data.cdn = 1.7           # normal drag coefficient
line.seg(1).line_data.cda = 0.085         # axial drag coefficient
line.seg(1).line_data.can = 1.0           # normal additional mass coefficient
line.seg(1).line_data.caa = 1.0           # axial additional mass coefficient
line.seg(1).line_data.cfn = 0.5           # normal friction coefficient
line.seg(1).line_data.cfa = 0.5           # axial friction coefficient
line.seg(1).line_data.active_hooke_material()

harm_func = model.create_harmonic_func()
harm_func.set_data(2.0, 0.1, 90.0, 0.0, 100.0)
# function parameters: amplitude, frequency, phase, initial and final ramp

line.endA.coord(0.0, 400.0, -500.0)       # set vertex 1 coordinates
line.endA.set_free("xx")                  # set degree of freedom
line.endA.set_free("yy")
line.endA.set_free("zz")

line.endB.coord(800, 0.0, 0.0)            # set vertex 2 coordinates
line.endB.set_displacement("z", harm_func)# prescribe displacement
line.endB.set_free("xx")
line.endB.set_free("yy")
line.endB.set_free("zz")

buoy = model.create_load_buoy()
buoy.set_data(100.0, 2.0, 0.0, 5.0)
# buoy parameters: buoyancy, height, pendant weight and height

# create a reference point for the buoyancy load at relative position from initial vertex
buoy_point = model.create_refpoint("buoy_point", line, 0.35)
buoy_point.add_boundary_conditions("buoy", buoy)

seabed = model.seabed()
seabed.type = doolines.INCLINATION        # set seabed modeling type
seabed.depth = 500.0                      # seabed depth
seabed.direction = 0.0                    # seabed direction
seabed.slope = 0.0                        # seabed slope

current = model.current()
current.rhow = 1.025                      # water density
current.point(1).set_data(0.0, 1.0, 0.0)  # current parameters: depth, velocity and angle
current.point(2).set_data(500.0, 0.0, 0.0)

sim = doolines.create_simulation(model)
sim.run()                                 # run simulation

post = doolines.create_post_processor(sim)
post.save_xmf()
results = post.export_data()              # export results as JSON

doolines.export_data()                    # export model data
```

Listing 1. Python script.

Case 2

A mooring line composed of 13 line segments is analyzed, with their properties listed in Tab. 3 and 4. The *BTruss* element is used in all segments. Loads due to a buoy and a clump are applied at specified arc lengths in Tab. 5, along with the properties of the components. The coordinates in meters of the top and the anchor are (45, 45, 8) and (2268.25, 2268.25, -1800), respectively. A surface current velocity of 1.42 m/s is applied in the 135° direction. The simulation uses the generalized explicit–α algorithm, with 85% of the critical time step.

Table 3. Segment properties.

Seg	Length (m)	Num. Elem.	Material
1	20.00	1	Studless chain
2	210.00	10	Studless chain
3	50.00	2	Studless chain
4	789.74	28	Polyester
5	10.00	1	Studless chain
6	789.74	28	Polyester
7	10.00	1	Studless chain
8	789.74	28	Polyester
9	10.00	1	Studless chain
10	789.74	28	Polyester
11	50.00	2	Studless chain
12	9.87	1	Polyester
13	85.39	4	Studless chain

Table 4. Material properties.

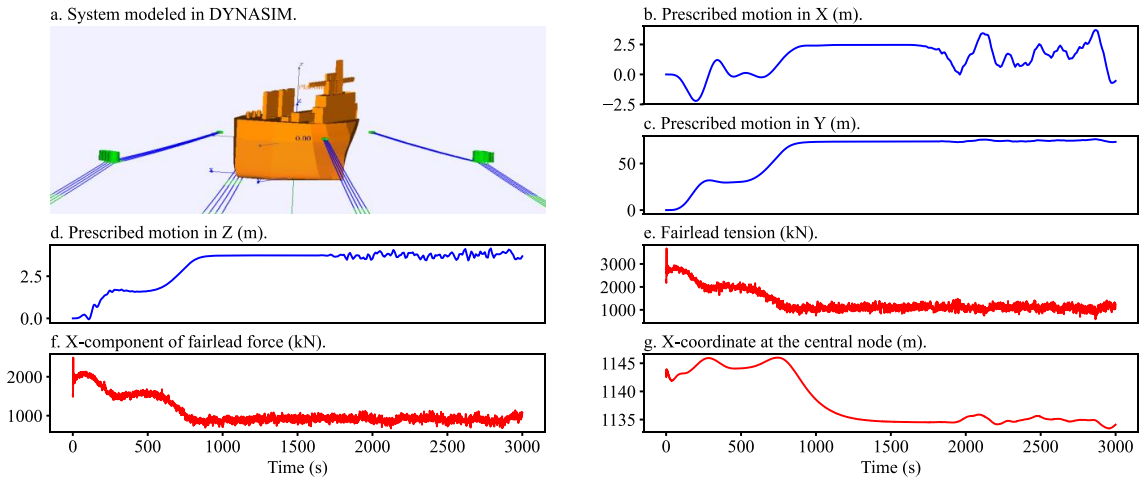
Property	Studless chain	Polyester
External diam. (mm)	114	222
Trans. Area (m²)	0.01	0.04
Weight in air (kN/m)	2.58	0.32
EA (kN)	990438.16	211513.00
EI (kN.m²)	61902.39	13219.56
Normal drag coeff.	1.7	1.2
Axial drag coeff.	1.15	0.01
Normal add. mass coeff.	2.0	2.0
Axial add. mass coeff.	1.5	1.0
Lateral friction coeff.	0.5	0.5
Axial friction coeffi.	0.7	0.7

Table 5. Clump and buoy load properties.

Load	Height (m)	Diameter (m)	Pendant length (m)	Buoyancy (kN)	Arc length position (relative to anchor)
Clump	7.00	7.00	2.00	-500.00	0.06
Buoy	9.00	9.00	2.00	1000.00	0.97

Prescribed displacements in the X, Y, and Z directions, resulting from the coupled analysis of the line and vessel using DYNASIM, are applied at the top of the line. The coupled system model is illustrated in Fig. 3a. The motions in all three directions are presented in Figs. 3b, 3c and 3d. Time series for nodal and element properties are generated from the simulation. Nodal properties include coordinates and curvature, while element data include tension, local and global forces at the start and end nodes, bending, and deformed length. Figs. 3e, 3f and 3g show,

Figure 3. Applied motion and resulting time series responses.



respectively, the tension and the X-component of the force at the top and the X-coordinate at the central node.

5 Conclusion

This work presented the Python interface for DOOLINES, a framework that simplifies structural analysis using advanced numerical methods, making it accessible to users with varying levels of programming experience. The system enables efficient modeling of offshore lines with automatic mesh generation and the application of loads, prescribed displacements, and hydrodynamic effects in just a few lines of code.

Integration with libraries such as Pandas facilitates handling data imported from external software, particularly in cases involving complex configurations, such as systems comprising multiple segments with varying materials. Additionally, the use of visualization tools (Matplotlib, Plotly, Streamlit) allows detailed analysis of results, and its compatibility with other Python frameworks facilitates integration into automated workflows and web-based applications.

The developed DOOLINES interface provides a practical and accessible solution for offshore engineering simulations, suitable for both industrial and academic applications.

Acknowledgements. The authors gratefully acknowledge PETROBRAS and CNPq for their financial support.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] K. T. Ma, Y. Lou and Y. Wu, *Mooring System Engineering for Offshore Structures*. Elsevier, 2019.
[2] Orcina. *OrcaFlex*. Available: https://www.orcina.com/orcaflex/. Accessed: Aug. 4, 2025.
[3] Principia. *DeepLines.* Available: https://www.principia-group.com/blog/product/produit-deeplines/. Accessed: Aug. 4, 2025.
[4] Wood Group Kenny, *Flexcom*. Available: https://flexcom.fea.solutions/flexcom.html. Accessed: Aug 4, 2025.
[5] K. Nishimoto, C. H. Fucatu and I. Q. Masetti, "Dynasim–A Time Domain Simulator of Anchored FPS0". *Journal of Offshore Mechanics and Arctic Engineering*, vol. 124, n. 4, 2002.
[6] E. S. S. Silveira, E. N. Lages and F. M. G. Ferreira, "DOOLINES: an object-oriented framework for non-linear static and dynamic analyses of offshore lines". *Engineering with Computers*, vol. 28, pp. 149–159, 2012.
[7] Dynasim, *Manual Teórico*.
[8] F. M. G. Ferreira, *Desenvolvimento e Aplicações de um Framework Orientado a Objetos para Análise Dinâmica de Linhas de Ancoragem e de Risers.* MSc dissertation, Federal University of Alagoas, 2005.
[9] H. P. Bezerra Neto, *Desempenho de elementos finitos de viga com formulação completa e simplificada na análise dinâmica de risers*. MSc dissertation, Federal University of Alagoas, 2014.
[10] J. C. A. Silva Júnior, *Desenvolvimento da interface gráfica Wave Kinematics para análise cinemática de ondas regulares e irregulares.* Bachelor's thesis, Federal University of Alagoas, 2021.
[11] M. A. Gomes Ruggiero and V. L. R. Lopes, *Cálculo Numérico: Aspectos Teóricos e Computacionais*. Pearson, 1996.
[12] J. Chung and J. M. Lee, "A new family of explicit time integration methods for linear and non-linear structural dynamics". *International Journal for Numerical Methods in Engineering*, v. 37, p. 3961–3976, 1994.
[13] D. M. Guirardi, *O método da relaxação dinâmica aplicado à análise de estruturas de cabos e membranas*. Phd thesis, University of São Paulo, 2011.
[14] G. M. Hulbert and J. Chung, "Explicit time integration algorithms for structural dynamics with optimal numeric dissipation". *Computer Methods in Applied Mechanics and Engineering*, vol. 137, pp. 175–188, 1996.
[15] D. M. Beazley, "Automated scientific software scripting with SWIG". *Future Generation Computer Systems*, vol. 19, pp. 559–609, 2003.
[16] D. Hefferman, *An introduction to the Python interface to OrcaFlex*. 2025.
[17] N. M. O'Boyle, C. Morley and G. R. Hutchison, "Pybel: a Python wrapper for the OpenBabel cheminformatics toolkit". *Chemistry Central Journal*, vol. 2, n. 5, 2008.
[18] D. Can, C. R. Martinez, P. Papadopoulos and S. S. Narayanan, "PyKaldi: a Python wrapper for Kaldi". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5889-5893.
[19] E. Pasquetti, *Estabilidade Estática e Dinâmica de Torres Estaiadas*. MSc dissertation, Pontifical Catholic University of Rio de Janeiro, 2003.
[20] Kitware, *ParaView*. Available: https://www.paraview.org/. Accessed: Aug 4, 2025.