PyVista for CAE Datasets: Streamlining 3D Visualization and Analysis

Alex Kaszynski
Bane Sullivan
Tetsuo Koyama
And 180+ GitHub Contributors

Advanced Modeling & Simulation (AMS) Seminar Series
NASA Ames Research Center

March 24, 2025

Table of Contents

Speaker: Alex Kaszynski

Staff Simulation Engineer, Pasteur ISI & AFRL/USAF

- Co-creator of **PyVista**, an open-source Python library for 3D visualization and mesh analysis.
- Former Distinguished Engineer at **Ansys** and creator of PyAnsys, a suite of Python libraries to interface with Ansys's products.
- Advocate for open-source tools in research, engineering, and scientific computing.
- Extensive experience in computational engineering, scientific visualization, and CAE workflows.

PyVista - Introduction

The Backbone of Scientific Visualization

Key Statistics and Adoption:

- **Established in 1993**, VTK has over 5 million lines of code, reflecting its extensive development.
- **2.5+ million downloads annually** from Kitware's servers.
- **Global research adoption**: Over 100 papers per year in China since 2007 in medical imaging, geological exploration, and aerospace.
- **Foundation for major applications**: Powers ParaView, VisIt, and many other industry-standard tools.
- **Designed for modern computing**: Supports GPU acceleration and fine-grained parallelism through VTK-m.

Sources: Kitware, Exascale Project

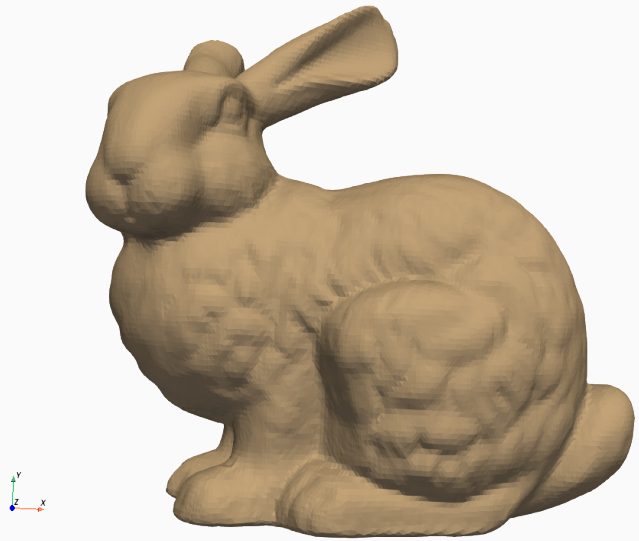
Comparison - VTK vs. PyVista

Raw VTK Implementation

```
import vtk
reader = vtk.vtkSTLReader()
reader.SetFileName("bunny.stl")
mapper = vtk.vtkPolyDataMapper()
output_port = reader.GetOutputPort()
mapper.SetInputConnection(output_port)
actor = vtk.vtkActor()
actor.SetMapper(mapper)
ren = vtk.vtkRenderer()
renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
ren.AddActor(actor)
iren.Initialize()
renWin.Render()
iren.Start()
```

Simplified PyVista Approach

```
from pyvista import examples
mesh = examples.download_bunny()
mesh.plot(cpos='xy')
```



Follow along with our online docs at Plotting a Mesh using PyVista.

The Philosophy Behind PyVista

Guiding Principles

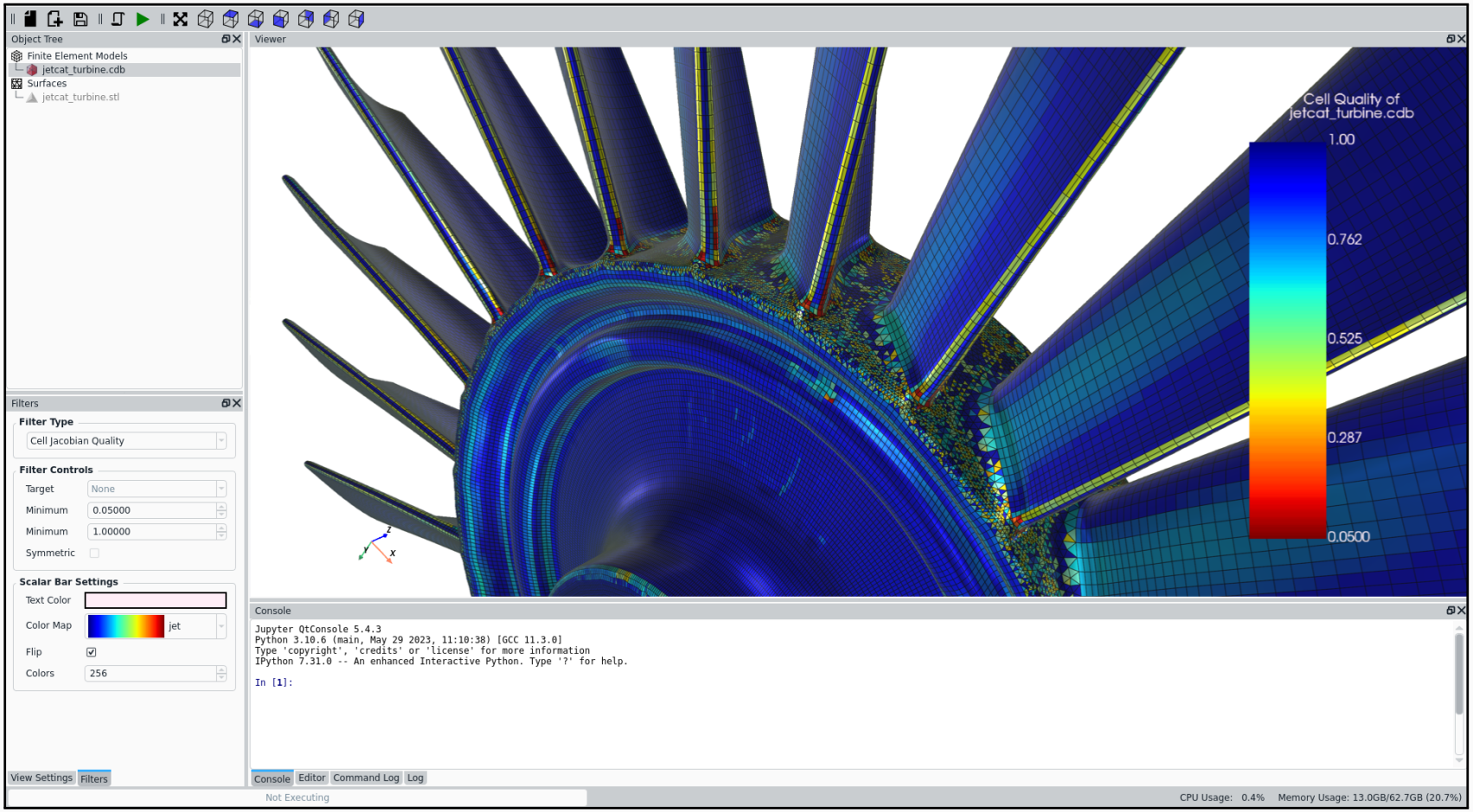
- ✓ Enable Python developers to work naturally within their ecosystem.
- ✓ Reduce boilerplate without sacrificing power or flexibility.
- ✓ Provide immediate, intuitive access to visualization and mesh operations.

- ➤ **Minimalism**: Less code. PyVista wraps complex VTK functionality into concise, readable syntax.
- ➤ **Documentation and Examples**: Classes and methods are meticulously documented with interactive examples.
- ➤ **Integration with Python Ecosystem**: Native integration with NumPy and other scientific libraries for intuitive data inspection and manipulation.
- ➤ **Automatic Resource Management**: Handles object lifecycle, memory, and cleanups internally.

PyVista is built for Python developers who expect their data visualization tools to look and operate just like the rest of their data science libraries.

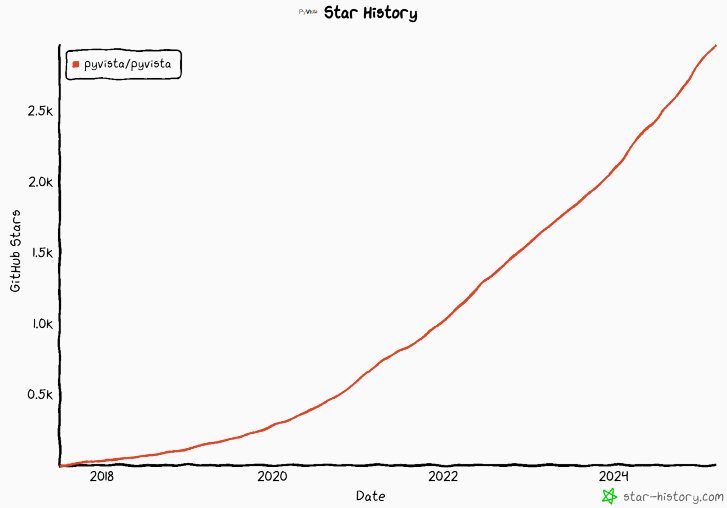
PyVista - Library Origin

Originally designed to replace repetitive boiler plate code within the mesh metamorphosis application FEMORPH. Open sourced in 2016 and developed over 8 years of community involvement and 180+ contributors.



PyVista - Popularity and Growth

- Already the most popular 3D visualization library on PyPI.
- Designed not just for visualization, but for scientific visualization focused on data post-processing, file IO, and interoperability with other libraries.
- Supports a growing ecosystem with strong community contributions and integrates with tools like `Jupyter`, `VTK`, and `meshio`.

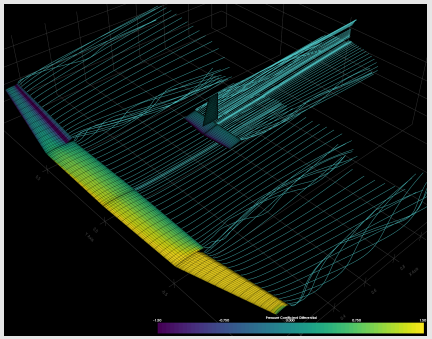


Name	Stars	Contributors	Downloads	License	Docs	PyPI	Conda	Sponsors	Built on
VTK	2.8k	259	pypi 757k/month	conda 79k/month	BSD	up	v9.4.1	conda invalid	Kitware	VTK
pyvista	3k	168	pypi 451k/month	conda 16k/month	MIT	up	v0.44.2	conda-forge v0.44.2	-	VTK
vispy	3.4k	165	pypi 142k/month	conda 12k/month	BSD-3-Clause	up	v0.14.3	conda-forge v0.14.3	-	OpenGL
ipyvolume	2k	44	pypi 53k/month	conda 2k/month	MIT	-	v0.6.3	conda-forge v0.6.3	-	OpenGL WebGL
mayavi	1.3k	58	pypi 13k/month	conda 7k/month	BSD	up	v4.8.2	anaconda v4.7.2	ENTHOUGHT	VTK OpenGL
vedo	2.1k	36	pypi 25k/month	conda 349/month	MIT	up	v2025.5.3	conda-forge v2025.5.3	EMBL	VTK
itkwidgets	594	7	pypi 13k/month	conda 1k/month	Apache-2.0	-	v0.32.6	conda-forge v0.32.6	-	WebGL
polyscope	1.9k	27	pypi 16k/month	-	MIT	up	v2.3.0	-	-	OpenGL
glumpy	1.3k	51	pypi 961/month	-	BSD License	docs passing	v1.2.1	-	-	OpenGL
datoviz	642	2	-	conda 0/month	-	up	-	mark.harfouche v0.1.0.alpha.1.29.ge26dafba	-	Vulkan

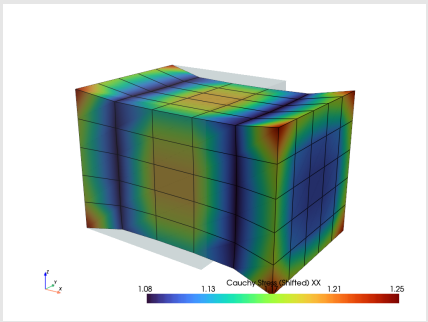
PyVista - Libraries Showcase

Several open-source CAE projects that leverage PyVista

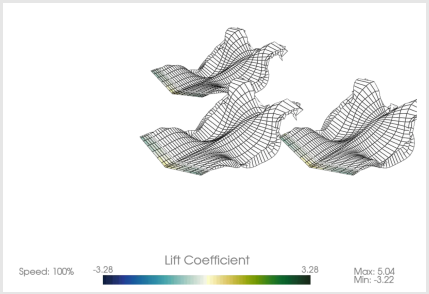
AeroSandbox



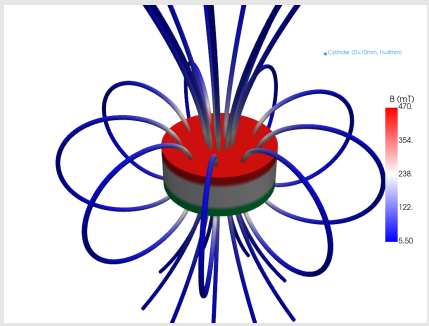
Felupe



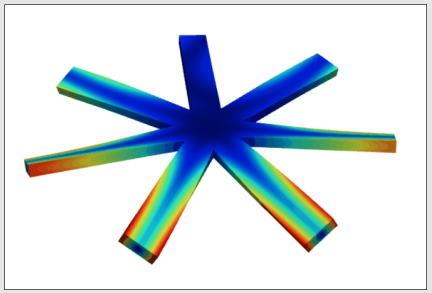
PteraSoftware



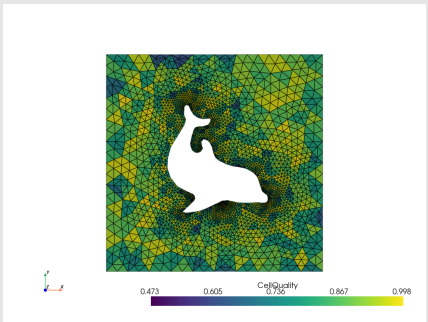
Magpylib



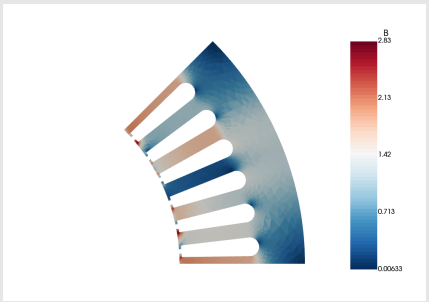
PyMAPDL



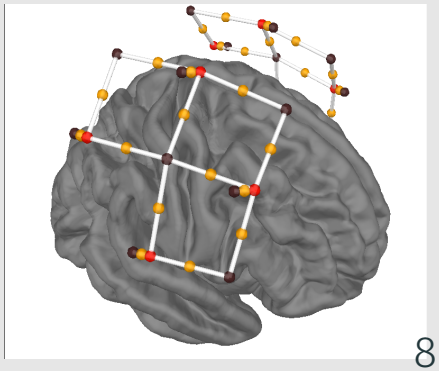
dolfinx



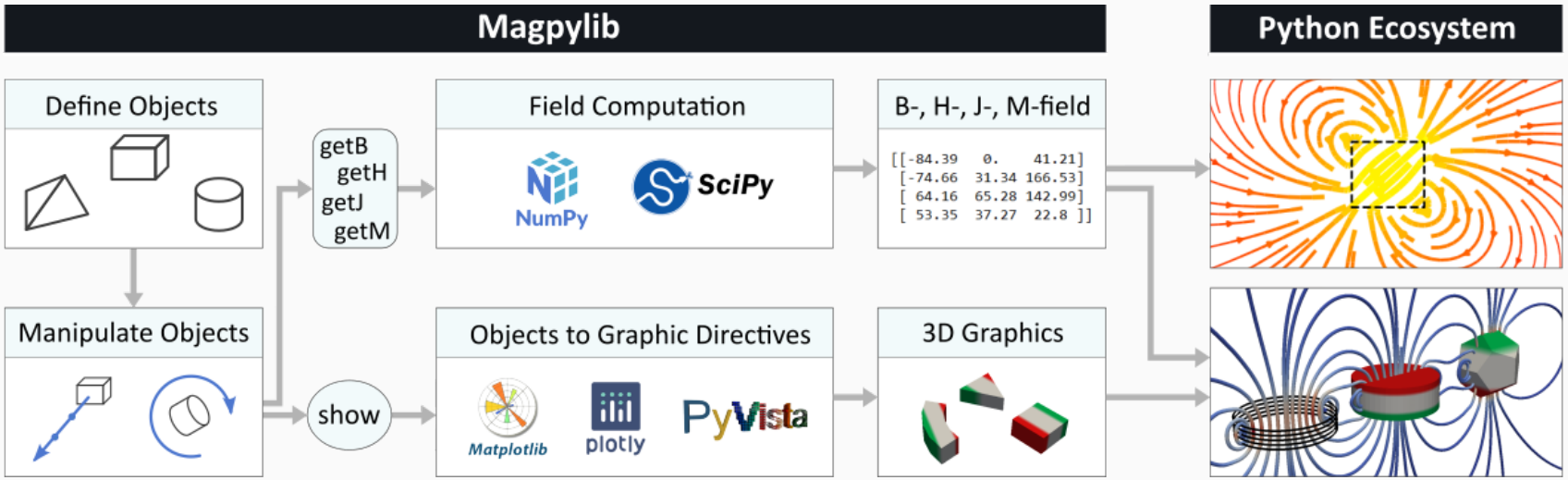
Pyleecan



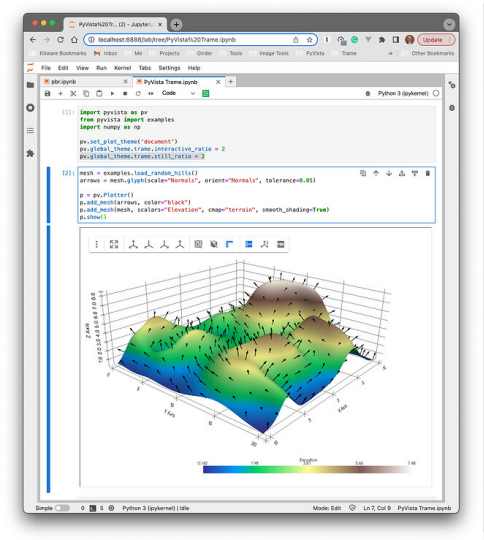
MNE



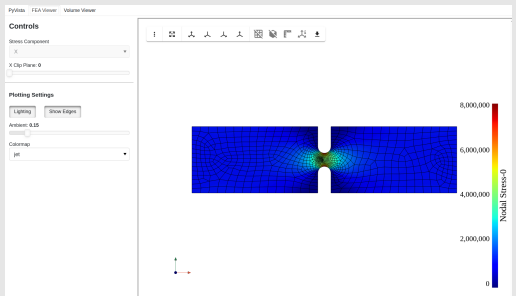
Incorporating PyVista within a CAE Workflow or Application



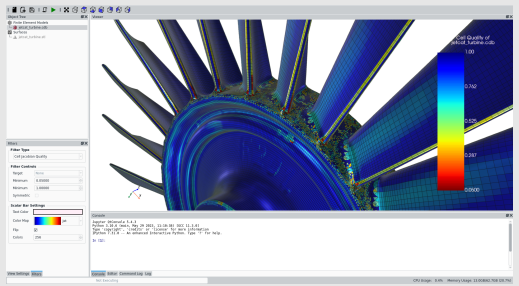
Viewer and Data science



Library or Standalone Tool

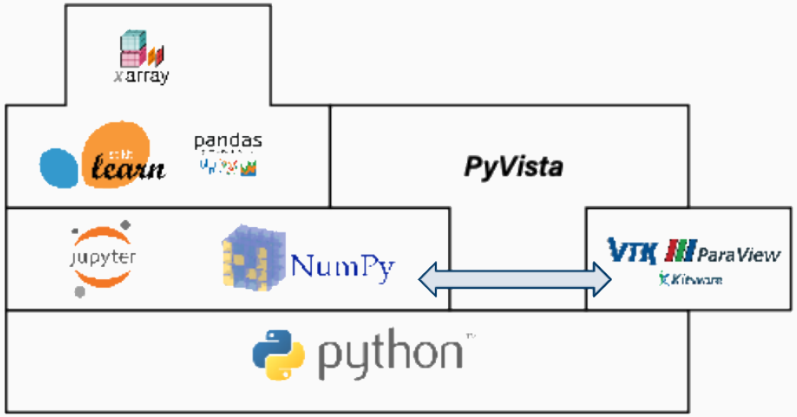


Complex Application



PyVista is built on a foundation of powerful open-source tools

- **NumPy** – Fundamental package for scientific computing with Python
- **VTK** – Advanced 3D visualization and data processing
- **trame** – Client and Server-side rendering of VTK objects
- **Matplotlib** – Colormaps and 2D plotting
- **Jupyter** – Interactive computing and in-browser computing



PyVista - General Usage

General Examples

PyVista - Introduction

- Comparison - VTK vs. PyVista

PyVista - General Usage

- Surface Plotting
- Basic Volumetric Plotting
- Filters

Extending PyVista: Deployment, Packaging, and Documentation

- PyInstaller and PyQT

PyVista - Ecosystem

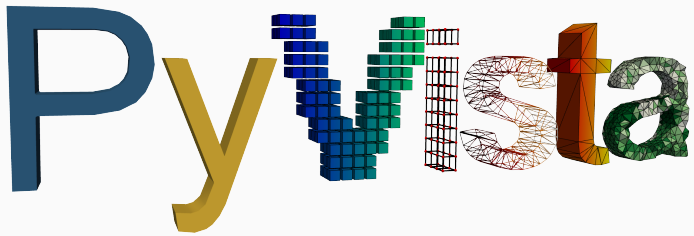
- Tutorial

PyVista Applied to CAE Datasets

- Introduction
- Jupyterlab Demos
- Surface Scan Analysis - HECC

PyVista General Usage - Introduction

```
>>> from pyvista import demos
>>> demos.plot_logo(
...     window_size=(1920, 1080),
        off_screen=False,
... )
```



PyVista allows you to rapidly load meshes and handles much of the “grunt work” of setting up plots, connecting classes and pipelines, and cleaning up plotting windows.

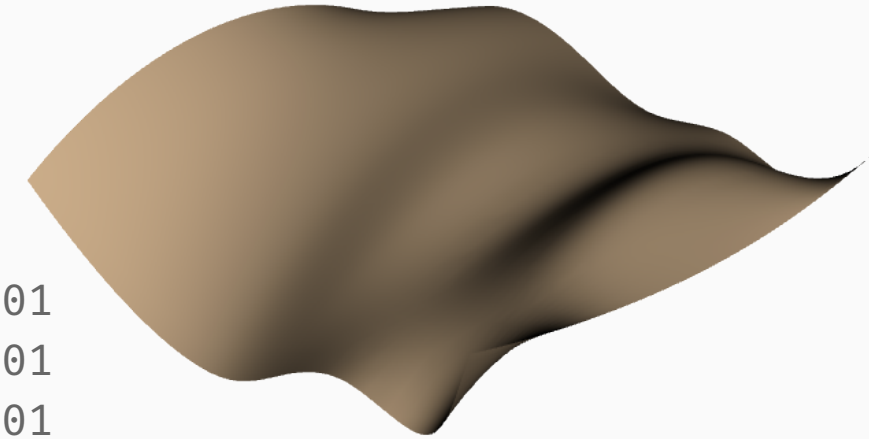
PyVista allows you to:

- Easily load a wide variety of datasets and file types.
- Leverage powerful VTK filters and perform complex data operations.
- Quickly set up simple or complex plots.
- Integrate with NumPy, SciPy, Pandas, and many other Python libraries.

Examples - Surface Plotting

Load a surface dataset from the built-in examples and show the `PolyData` object. Plot it.

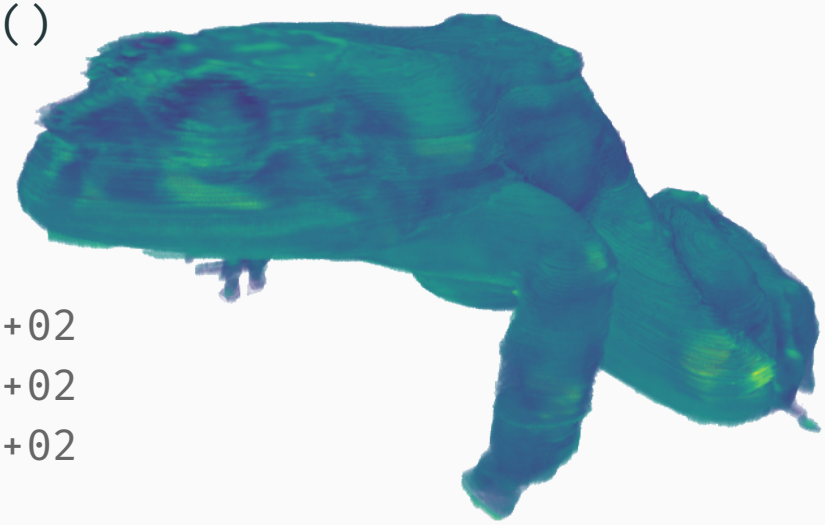
```
>>> from pyvista import examples
>>> dataset = examples.download_saddle_surface()
>>> dataset
PolyData (0x7f4d81806c40)
  N Cells:      5131
  N Points:     2669
  N Strips:     0
  X Bounds:     -2.001e+01, 2.000e+01
  Y Bounds:     -6.480e-01, 4.024e+01
  Z Bounds:     -6.093e-01, 1.513e+01
  N Arrays:     0
>>> dataset.plot(color='tan')
```



Examples - Basic Volumetric Plot

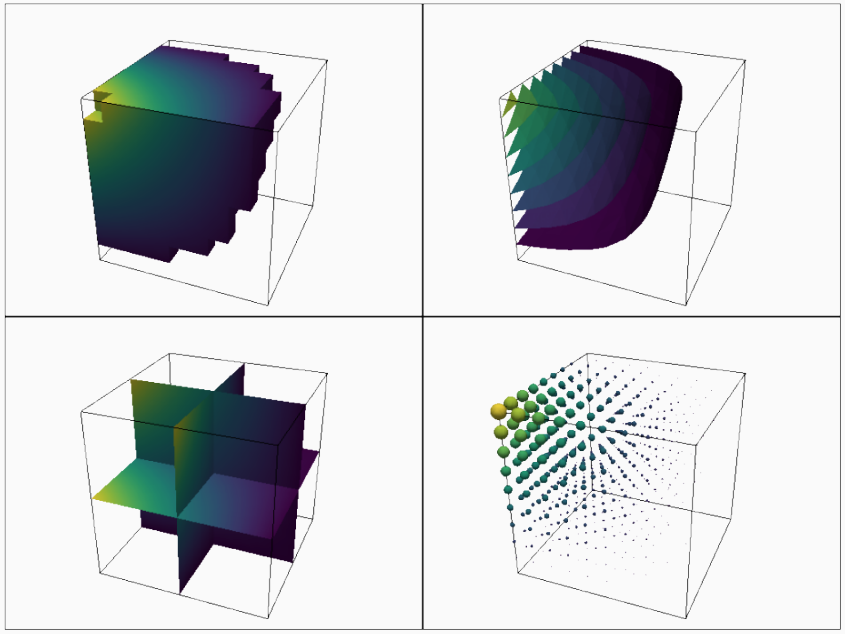
Load a volumetric dataset from the built-in examples and plot it.

```
>>> from pyvista import examples
>>> dataset = examples.download_frog()
>>> dataset
UniformGrid (0x7f4d81806700)
  N Cells:      31594185
  N Points:     31960000
  X Bounds:     0.000e+00, 4.990e+02
  Y Bounds:     0.000e+00, 4.690e+02
  Z Bounds:     0.000e+00, 2.025e+02
  Dimensions:   500, 470, 136
  Spacing:      1.000e+00, 1.000e+00, 1.500e+00
  N Arrays:     1
>>> dataset.plot(volume=True)
```



Examples - Filters

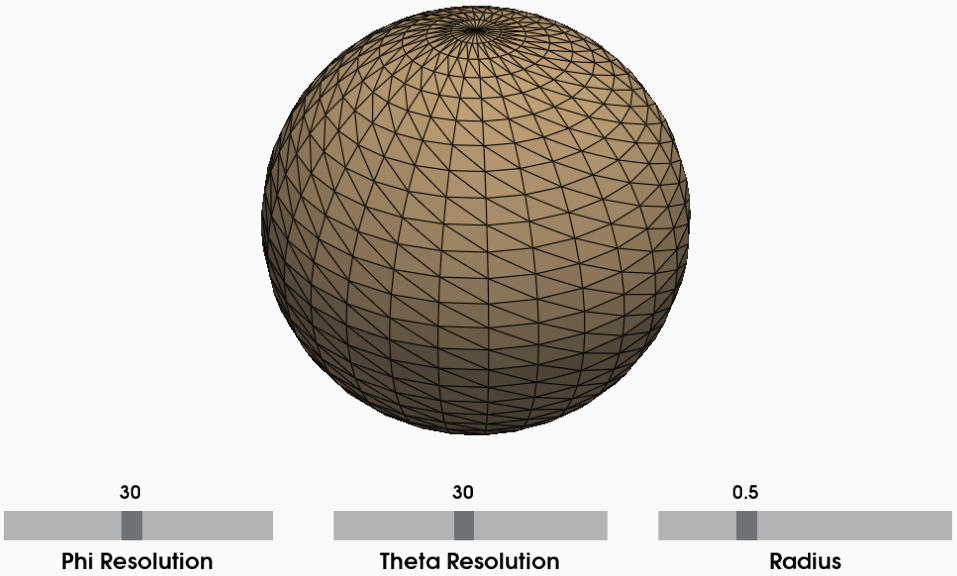
```
import pyvista as pv
from pyvista import examples

dataset = examples.load_uniform()
outline = dataset.outline()
threshed = dataset.threshold(
    [100, 500],
)
contours = dataset.contour()
slices = dataset.slice_orthogonal()
glyphs = dataset.glyph(
    factor=1e-3,
    geom=pv.Sphere(),
)

pl = pv.Plotter(shape=(2, 2))
pl.add_mesh(outline, color="k")
pl.add_mesh(threshed, show_scalar_bar=False)
```



Examples - Widgets

Demonstrate interactive filters by loading a mesh and modifying the resolution of a VTK pipeline.

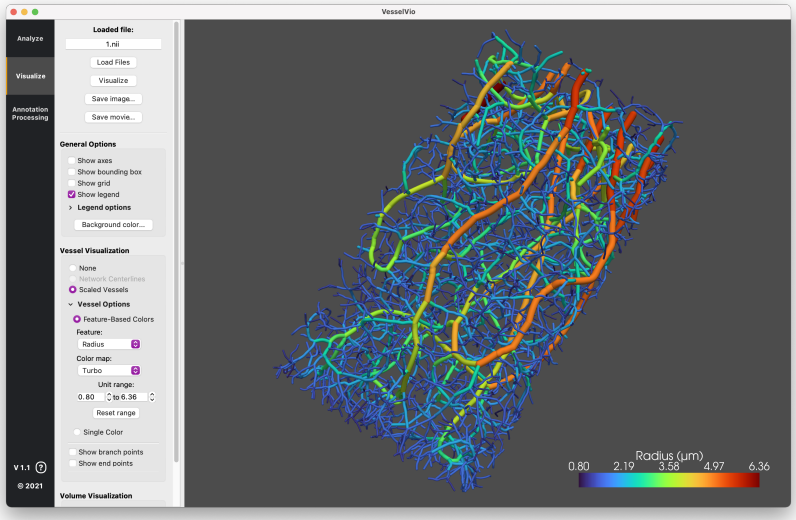
```
pl = pv.Plotter()
pl.add_mesh(
    starting_mesh,
    show_edges=True,
)
pl.add_slider_widget(
    callback=callback,
    rng=[3, 60],
    value=30,
    title="Phi Resolution",
    pointa=(0.025, 0.1),
    pointb=(0.31, 0.1),
    style="modern",
)
```



Extending PyVista: Deployment, Packaging, and Documentation

Examples - PyInstaller and PyQT

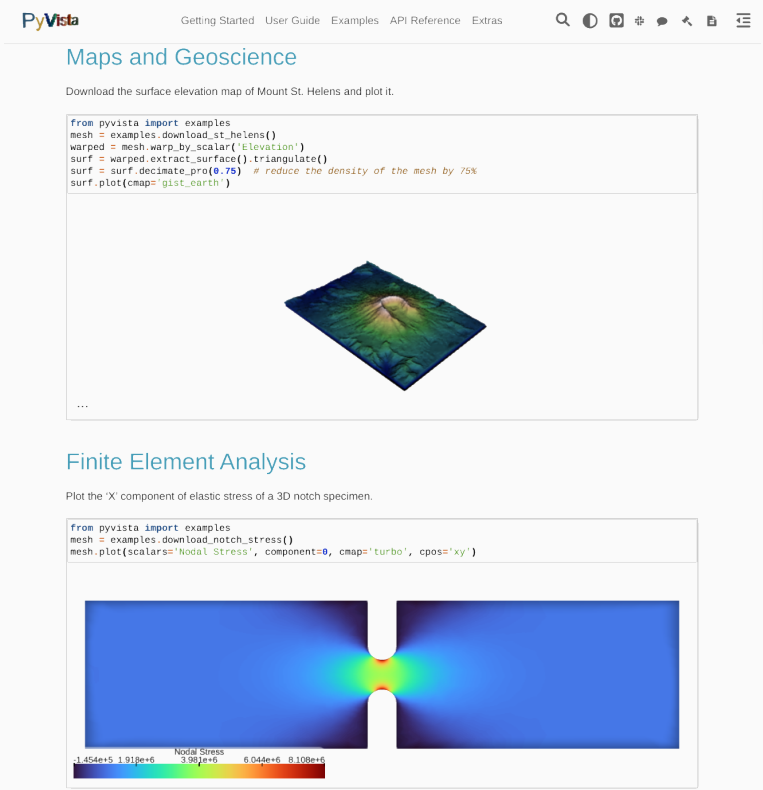
- Use PyInstaller and PyQT or PySide to create a standalone application.
- Multi-platform. Build on the OS you intend to deploy.
- Compatible with GitHub Actions and can be automated.
- Deploy as using an installer like NSIS.




```
pip install -r requirements_build.txt
pyinstaller \
    --add-data=Library;Library \
    --additional-hooks-dir=Hooks \
    --icon library\icons\icon.ico \
    --windowed VesselVio.py
```

Examples - Documentation with Sphinx

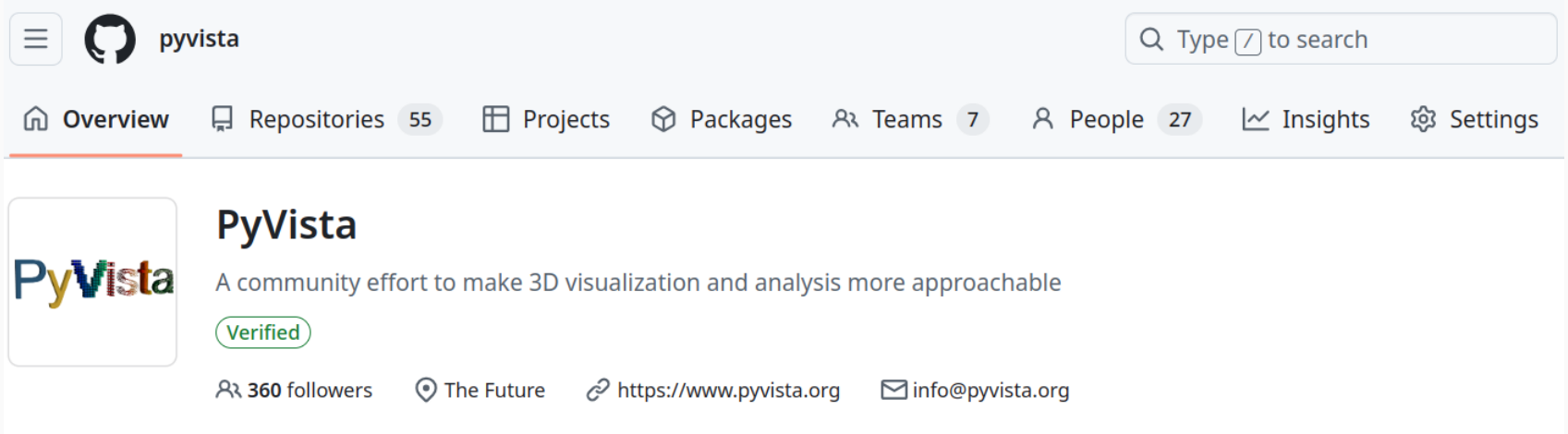
- PyVista supports the Sphinx documentation generator.
- Allows you to generate static and interactive documentation.
- Place code snippets directly in the documentation as examples.




```
.. jupyter-execute::

    from pyvista import examples
    mesh = examples.download_st_helens()
    warped = mesh.warp_by_scalar('Elevation')
    surf = warped.extract_surface().triangulate()
    surf = surf.decimate_pro(0.75)
    surf.plot(cmap='gist_earth')
```

PyVista - Ecosystem

PyVista - Ecosystem



- pymeshfix - Python Wrapper for MeshFix: Easily repair holes in surface meshes
- tetgen - Generate tetrahedral meshes of any 3D polyhedral domain
- PyACVD - Python implementation of surface mesh resampling algorithm ACVD
- fast-simplification - Fast Quadratic Mesh Simplification
- pyvista-qt - Qt support for PyVista
- pyvista-xarray - xarray DataArray accessors for PyVista

Tutorial

The PyVista Tutorial contains a variety of lessons to help you get started with PyVista. Includes:

- Introduction - Using PyVista for 3D Visualization within Python.
- Reading and plotting 3D data using the PyVista module and external files.
- Learn the basics of the PyVista data types and how to open common 3D file formats to visualize the data in 3D.
- Demonstrate many features of the PyVista plotting API to create compelling 3D visualizations and touch on animations.
- Demonstrate the PyVista filters API to perform mesh analysis and alteration.

PyVista Applied to CAE Datasets

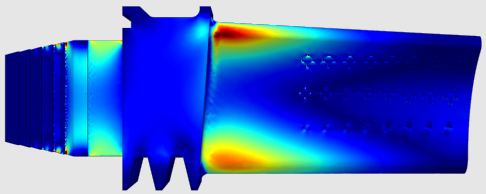
PyVista Applied to CAE Datasets - Introduction

- **Built on the Best:** Built on VTK, PyVista simplifies complex CAE data processing while maintaining full compatibility with industry-standard formats.
- **Efficient Data Handling:** Supports large-scale meshes, point clouds, volumetric data, and time-dependent simulations with optimized memory management.
- **Advanced Visualization:** Enables rapid 3D rendering of simulation results, including contouring, slicing, and isosurfacing.
- **Interactivity:** Provides interactive plotting tools and GUI integration for intuitive CAE analysis.
- **Extensive Filtering & Processing:** Leverages powerful mesh manipulation and computational tools, ideal for FEA, CFD, and structural simulations.
- **Automation & Scripting:** Python-based, making it easy to integrate with CAE workflows for automated post-processing and analysis.

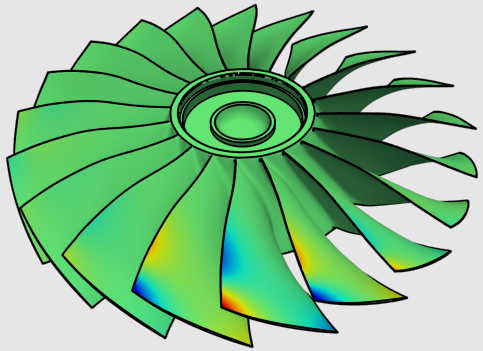
PyVista - CAE Usage

PyVista is widely used to alongside structural and CFD simulations, offering efficient visualization and analysis of simulation results. Its integration with VTK enables handling large datasets, automating workflows, and enhancing data interpretation.

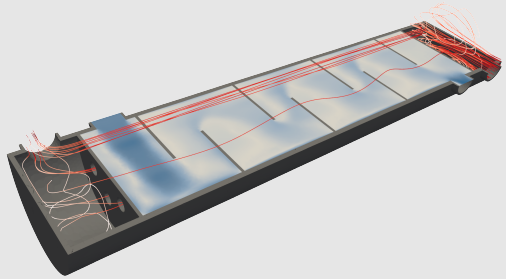
Structural - Static Results



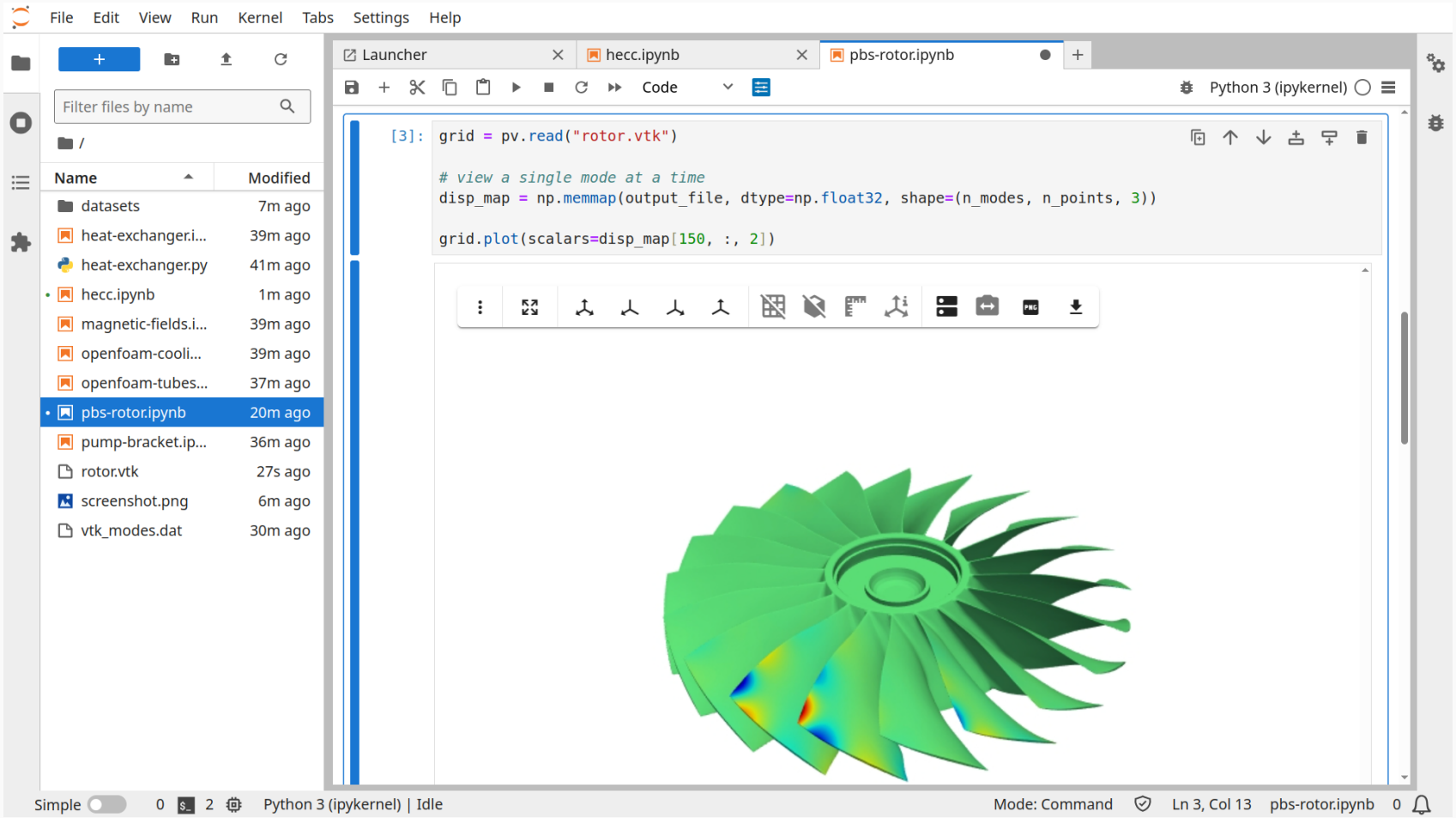
Structural - Modal Analysis



CFD - Steady and Turbulent



Jupyterlab Demos



Surface Scan Analysis - HECC

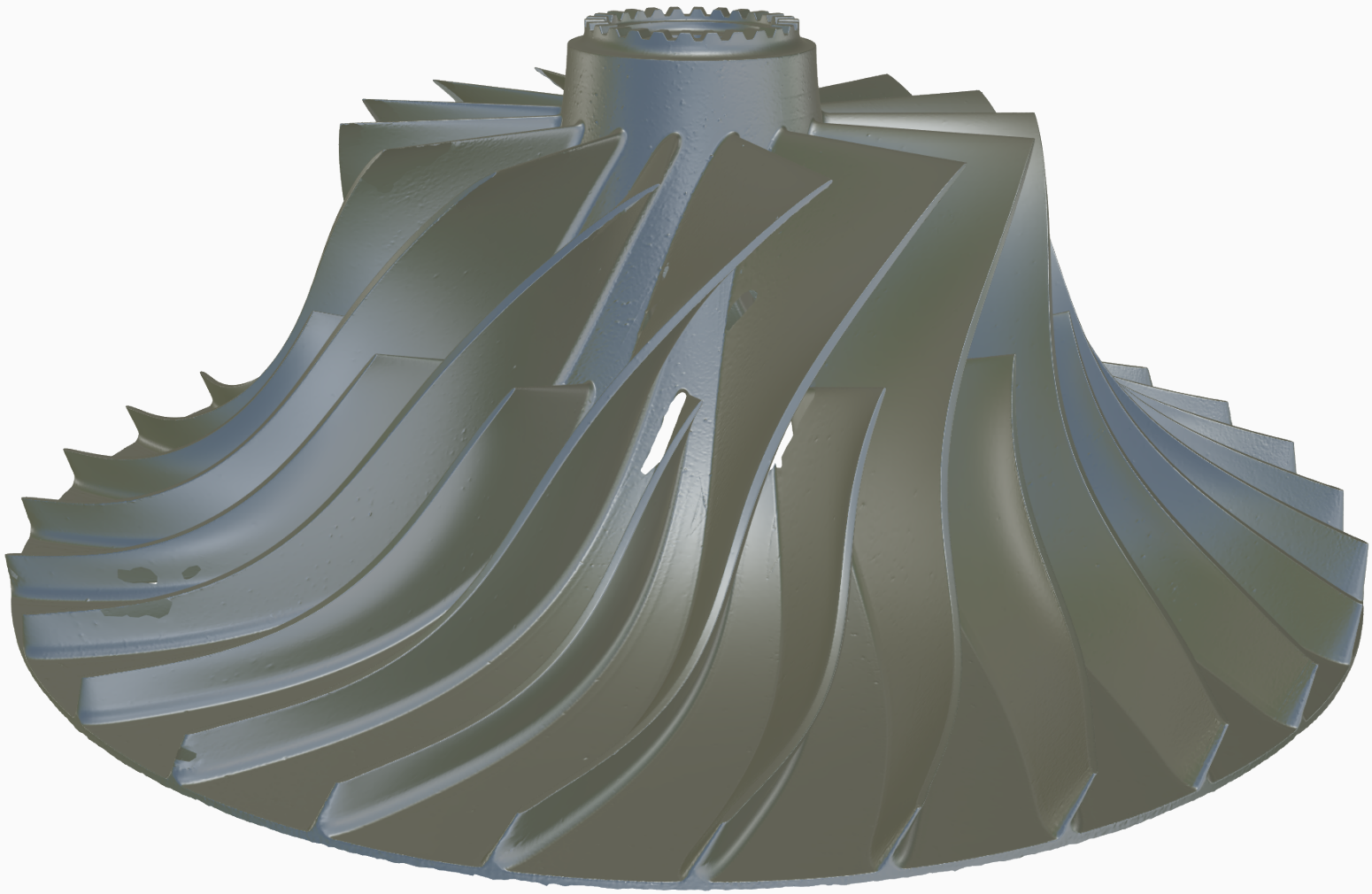
High Efficiency Centrifugal Compressor for Rotorcraft Applications

Plotting and demonstrating filters on the 4.5M point surface scan.

Surface Scan Analysis - HECC

```
import pyvista as pv
from pyvista import examples

mesh = pv.read("CE-18_Impeller.ply")

cubemap = examples.download_sky_box_cube_map()
pl = pv.Plotter()
pl.set_environment_texture(cubemap)
pl.add_mesh(
    mesh,
    color="w",
    pbr=True,
    metallic=1.0,
    roughness=0.5,
    diffuse=0.4,
)
pl.show()
```

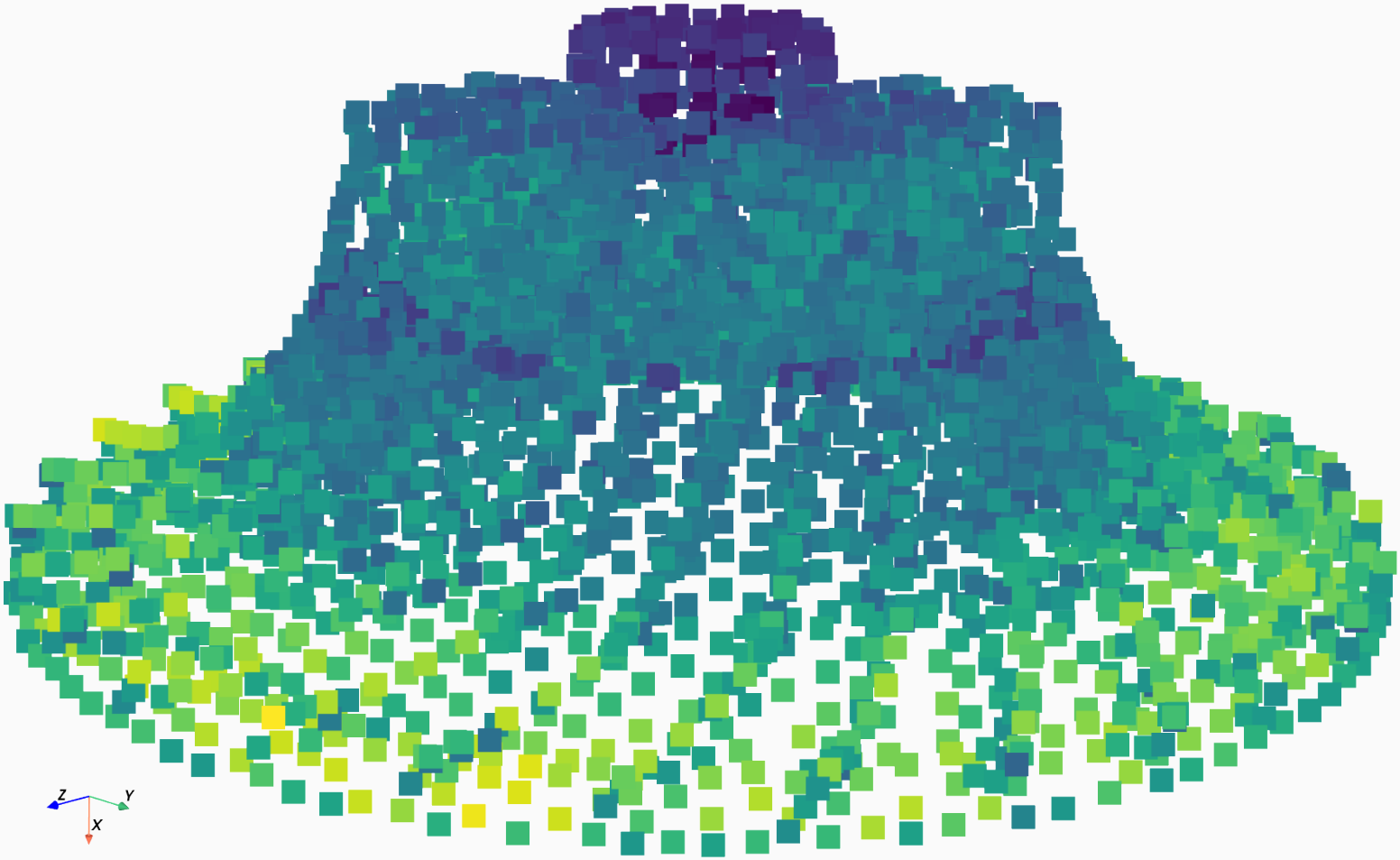
Surface Scan Analysis - HECC



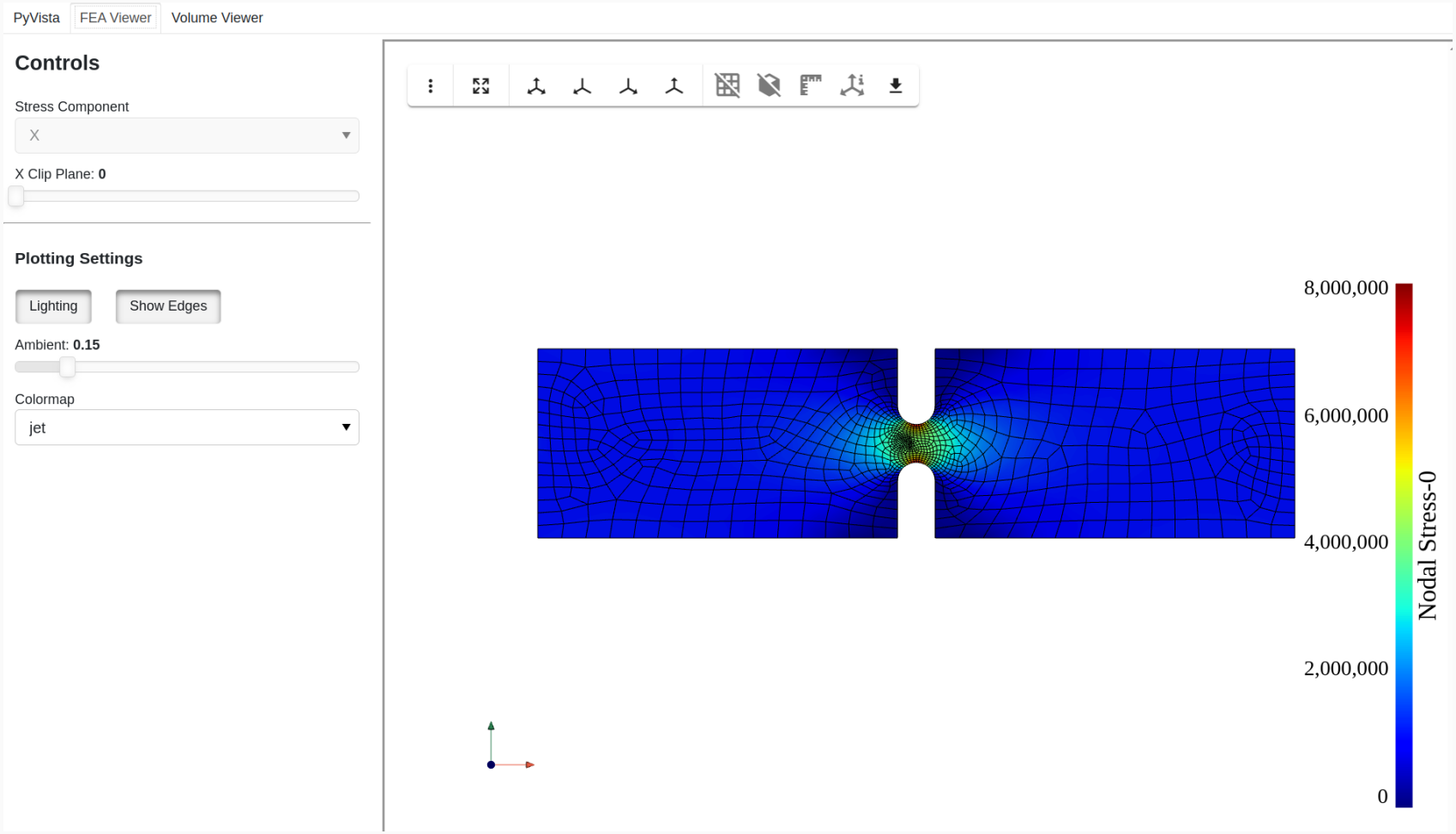
Surface Scan Analysis - HECC

```
from femorph import utilities
import numpy as np

# compute PFH
vox_length = mesh.length / 50
mesh.points = mesh.points.astype(np.float64)
pfh, pfh_points, _, _ = utilities.compute_pfh(mesh, vox_length)

# construct a pointset out of the PFH and plot it
pfh_ps = pv.PointSet(pfh_points)
pfh_ps["pfh"] = pfh
pfh_ps.plot(component=5, point_size=30, show_scalar_bar=False)
```

Surface Scan Analysis - HECC



Single Page Applications using Panel



Further Resources and Support

Tutorial:

tutorial.pyvista.org

Community Discussions (General Usage):

github.com/pyvista/pyvista/discussions

Issue Tracking and Feature Requests:

github.com/pyvista/pyvista/issues

Professional Support:

Kitware Support Services

Contact:

GitHub: @akaszynski

Email: Alex.Kaszynski@simulation.science

Slides Source

Slides Source (after 22 March 2025)

github.com/akaszynski/pyvista-nasa-presentation

```
import pyminiply

>>> timeit pv.read("CE-18_Impeller.ply")
1.23 s ± 6.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

>>> timeit mesh = pyminiply.read_as_mesh("CE-18_Impeller.ply")
358 ms ± 6.51 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

>>> mesh = pyminiply.read_as_mesh("CE-18_Impeller.ply")
PolyData (0x719e12b9a140)
  N Cells:    9065852
  N Points:   4535625
  N Strips:   0
  X Bounds:   -1.121e+00, 5.420e+00
  Y Bounds:   -8.481e+00, 8.485e+00
  Z Bounds:   -8.481e+00, 8.482e+00
  N Arrays:   1
```

Building a Python Library with PyVista

```toml
[build-system]
build-backend = "scikit_build_core.build"
requires = ["scikit-build-core >=0.4.3", "nanobind >=1.3.2"]

[project]
authors = [
  {name = "PyVista Developers", email = "info@pyvista.org"}
]
classifiers = [
  "Intended Audience :: Science/Research",
]
dependencies = [
  "numpy"
  "pyvista"
]
description = "Rapidly read in PLY files using a wrapper over miniply"
name = "pyminiply"
readme = "README.rst"
requires-python = ">=3.8"
version = "0.3.dev0"
```

Building a Python Library with PyVista

```cpp
#include "miniply/miniply.h"
#include <cstdio>

#include <nanobind/nanobind.h>
#include <nanobind/ndarray.h>
#include <nanobind/stl/string.h>

#include "array_support.h"

namespace nb = nanobind;
using namespace nb::literals;

/**
 * This function reads a 3D mesh from a .ply file using the miniply
 * and reutrns numpy arrays with vertices and triangle indices.
 */
nb::tuple LoadPLY(const std::string &filename,
                  bool read_normals = true, bool read_uv = true) {

  miniply::PLYReader reader(filename.c_str());
```

Building a Python Library with PyVista

```python
"""Python wrapper of the miniply library."""

import numpy as np
from pyvista import ID_TYPE
from pyvista.core.utilities import numpy_to_idarr
from pyvista.core.pointset import PolyData
from vtkmodules.vtkCommonDataModel import vtkCellArray
from pyminiply._wrapper import load_ply


def _load_ply(filename: str, read_normals: bool = True) -> PolyData:
    points, faces = load_ply(filename, read_normals)

    pdata = PolyData()
    pdata.points = points

    carr = vtkCellArray()
    offset = np.arange(0, faces.size + 1, faces.shape[1], dtype=ID_TYPE)
    carr.SetData(numpy_to_idarr(offset, deep=True), numpy_to_idarr(faces, deep=True))
    pdata.SetPolys(carr)
    return pdata
```