PVGeo: an open-source Python package for geoscientific visualization in VTK and ParaView

Christopher Sullivan[1] and Whitney Trainor Guitton[2]

[1]Colorado School of Mines
[2]Lawrence Livermore National Lab

November 23, 2022

Abstract

PVGeo is an open-source Python package for geoscientific visualization and analysis harnessing an already powerful software platform: the Visualization Toolkit (VTK) and its front-end application, ParaView. The VTK software platform is well-maintained, contains an expansive set of native functionality, and provides a robust foundation for scientific visualization, yet the development of tools compatible for geoscience data and models has been very limited. As a software extension package to VTK and ParaView, PVGeo addresses the lack of geoscientific compatibility by creating a framework for geo-visualization. This framework is a set of tools for visually integrating geoscience data and models directly within ParaView's graphical user interface, simplifying the required routines to make compelling visualizations of geoscientific datasets. The PVGeo package is available for download on PyPI (pip install PVGeo), documented online (http://pvgeo.org), and open-source on GitHub (https://github.com/OpenGeoVis/PVGeo) for community-driven developments.

PVGeo: *an open-source Python package for geoscientific visualization in VTK and ParaView*

Bane Sullivan, Whitney J. Trainor-Guitton
Colorado School of Mines, Golden, CO

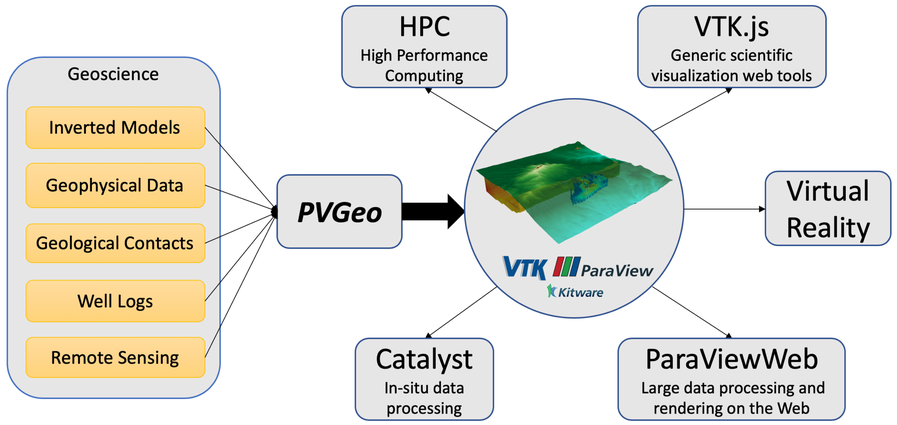
Learn More: *PVGeo.org*
Contact: *info@pvgeo.org*

About

- Constructing geoscience focused plugins for the open-source, multi-platform visualization application ParaView by Kitware (paraview.org).
- Developing tools to perform post processing analysis of geoscientific data and models.
- Building a framework users to easily build their own plugins
- All plugins, code documentation, and tutorials are published on PVGeo.org and all code is openly available on GitHub
- We encourage end-user feature requests and other community contributions!

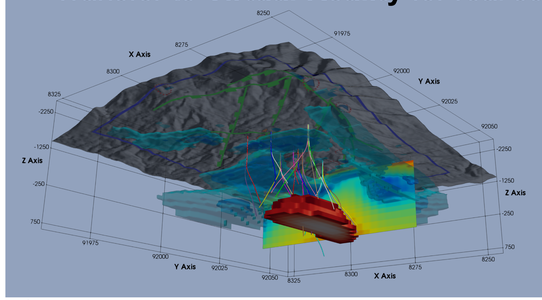
```
> pip install PVGeo
```

Bridging geoscience to VTK and ParaView



Applications

- Visual **data integration** (image below)
- Easily **share** 3D scenes with VTKjs
- Reference data in **relation to intuitive features** like topography, well locations, and other spatial data
- Transfer to **Virtual Reality** for real scale interaction

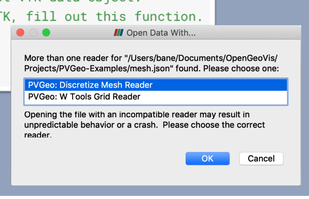


Interact with this data!

Create a File Reader

In this code snippet we demonstrate how to interface PVGeo to an external library for file reading directly into ParaView via GUI menus. The super classes in PVGeo enable users to focus on filling out two methods:

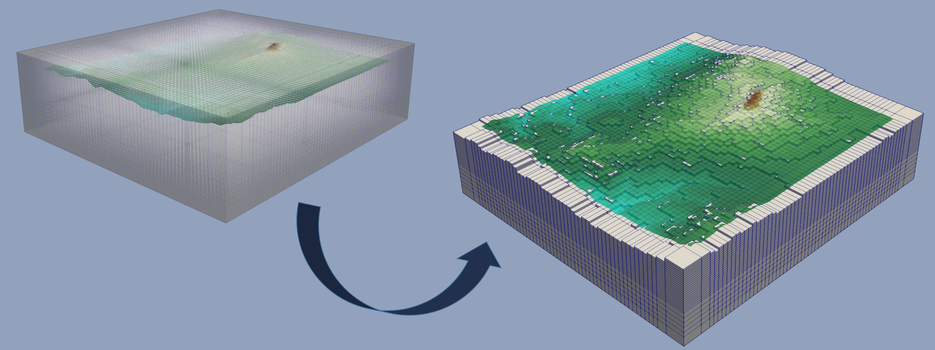
```python
import PVGeo
import discretize # or any external library with file IO methods

class DiscretizeMeshReader(PVGeo.InterfacedBaseReader):
    extensions = 'json'
    __displayname__ = 'Discretize Mesh Reader'
    description = 'Serialized Discretize Meshes'
    def __init__(self, **kwargs):
        InterfacedBaseReader.__init__(self, **kwargs)

    @staticmethod
    def _readFile(filename):
        """Uses an external library to perform file IO"""
        return discretize.MeshIO.load_mesh(filename)

    @staticmethod
    def _getVTKObject(obj):
        """Uses external library to get VTK data object.
        If external library lacking VTK, fill out this function.
        """
        return obj.toVTK()
```

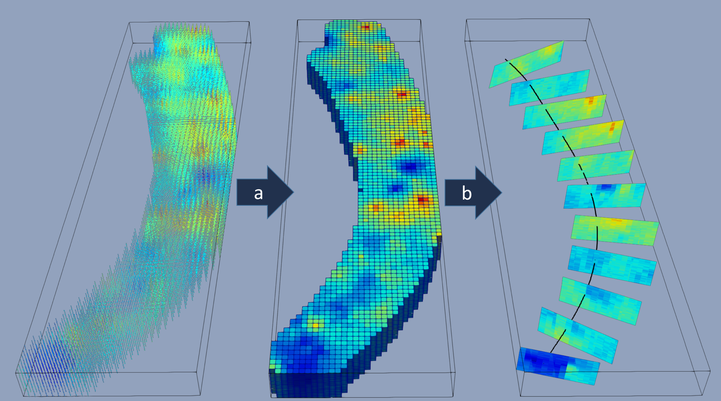


Transform Your Data

PVGeo's *ExtractTopography* filter splits volumetric datasets on a topography surface to extract a subsurface model.

```python
# Load a topography surface
topo = PVGeo.pointsToPolyData(np.loadtxt('topo.xyz'))
# Create a meshed volume
vol = PVGeo.model_build.CreateUniformGrid().Apply()
# Split the volumetric model on the topography surface
subvol = PVGeo.grid.ExtractTopography().Apply(vol, topo)
```





A progression from point-attribute data (left) to connected cell-attribute data using PVGeo's *Voxelize Points filter* (a), then sliced along a path using PVGeo's *Many Slices Along Points filter* (b).

```python
# Read in point data
pts = PVGeo.pointsToPolyData(np.loadtxt('point-cloud.xyz'))
# Create a volumetric dataset of these points
vol = PVGeo.filters.VoxelizePoints(estimate=True).Apply(pts)
# Load in a path through the dataset
path = PVGeo.pointsToPolyData(np.loadtxt('path.xyz'))
# Slice the volume along the path
slices = PVGeo.filters.ManySlicesAlongPoints(numSlices=10).Apply(path, vol)
```