

PYVISTA: MANAGING & VISUALIZING GEOSPATIAL DATA
USING AN OPEN-SOURCE FRAMEWORK

by
C. Bane Sullivan

© Copyright by C. Bane Sullivan, 2020

All Rights Reserved

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Hydrology).

Golden, Colorado

Date _____

Signed: _____

C. Bane Sullivan

Signed: _____

Dr. Whitney J. Trainor-Guitton
Thesis Advisor

Golden, Colorado

Date _____

Signed: _____

Dr. Josh Sharp
Professor and Director
Hydrologic Science & Engineering Program

ABSTRACT

There is a wide range of data types present in typical hydrogeophysical studies; being able to gather all data types for a given project into a single framework is challenging and often unachievable at an affordable cost for hydrological researchers. Steep licensing fees for commercial software and complex user interfaces in existing open software have limited the accessibility of tools to build, integrate, and make decisions with diverse types of 3D geospatial data and models. In earth science research, particularly in hydrology, restricted budgets exacerbate these limitations, creating barriers for using software to manage, visualize, and exchange 3D data in reproducible workflows. In response to these challenges, I have created the PyVista software as an open-source framework for 3D geospatial data management, fusion, and visualization. The PyVista Python package provides an accessible and intuitive interface back to a robust and established visualization library, the Visualization Toolkit (VTK), to facilitate rapid analysis and visual integration of spatially referenced datasets. This interface implements spatial data structures encompassing a majority of subsurface applications to make creating, managing, and analyzing spatial data more streamlined for domain scientists. PyVista's data management approach to 3D visualization furthers researchers' ability to work across existing and emerging Python-based software while offering streamlined routines for creating publication-quality, integrated 3D visualizations. Example code to produce visualizations of 3D subsurface models, surface topography, sparse data, and other geospatial information are provided. These examples demonstrate how PyVista enables researchers to gather all of their spatial data into a single framework to rapidly visualize, explore, and gain insight from spatial data. The software presented in this thesis is breaking down barriers for novice programmers hoping to use powerful, open-source, 3D software in reproducible geocomputing workflows.

TABLE OF CONTENTS

ABSTRACT	iii
LIST OF FIGURES	vi
LIST OF TABLES	viii
LIST OF ABBREVIATIONS	ix
ACKNOWLEDGMENTS	x
CHAPTER 1 CURRENT STATE OF 3D GEOSCIENCE	1
1.1 Data Management & Integration	3
1.2 3D Visualization and Fusion	4
1.3 The Rise of Python-based Research Software	5
1.4 Thesis Outline	7
1.5 Statement of Contribution	8
CHAPTER 2 STREAMLINING 3D GEOSCIENCE WORKFLOWS THROUGH THE PYVISTA SOFTWARE FRAMEWORK	9
2.1 Introduction to PyVista	10
2.2 PyVista: Data Management & Visualization	12
2.2.1 Data Structures	13
2.2.2 Visual Data Fusion	18
2.3 Data Exploration & Reproducible Workflows	20
2.3.1 Example: Geostatistical Kriging	22
2.3.2 Example: Visual Fusion	24

CHAPTER 3 CONCLUSION	28
REFERENCES CITED	31
APPENDIX A PVGEO: AN OPEN-SOURCE PYTHON PACKAGE FOR GEOSCIENTIFIC VISUALIZATION IN VTK AND PARAVIEW	33
A.1 Background	34
A.2 Mentions	35
A.3 Acknowledgements	36
APPENDIX B COMPARISON OF VISUALIZATION PROGRAMMING WITH THE VISUALIZATION TOOLKIT (VTK) AND PYVISTA	37
APPENDIX C SUPPLEMENTAL ELECTRONIC FILES	40

LIST OF FIGURES

Figure 1.1	An example of a typical subsurface study where data and models are co-located, including contextual information (DEM and satellite imagery), borehole measurements, a 3D structural geology model, and a 3D physical property model.	2
Figure 1.2	Typical data types found in subsurface research.	3
Figure 1.3	Code contribution graph for the PyVista software.	8
Figure 2.1	A visually integrated scene of geospatial data (FORGE Geothermal Site ⁹): left) view from upper south-west and right) view from lower north-west.	11
Figure 2.2	Common forms of geoscientific data that can be represented as 1D and 2D geometries in 3D space via the <code>PolyData</code> class.	14
Figure 2.3	Three commonly used mesh/volume types used in subsurface modeling applications.	15
Figure 2.4	An example output of PyVista data containers in a Jupyter Python environment.	16
Figure 2.5	Several of the datasets from the data structure in Figure 2.4 shown individually.	16
Figure 2.6	A threshold of the the <i>Kriged Temperature Model</i> produced from Listing 2.1 at 150° C.	17
Figure 2.7	Examples of visual fusion between two datasets: a 3D resource model and borehole measurements.	19
Figure 2.8	The same geospatial data from Figure 2.1 (FORGE Geothermal Site ⁹) with a threshold temperature model as produced from Listing 2.1 and Listing 2.2.	20
Figure 2.9	A typical geoscientific computing workflow (left to right) where PyVista can manage and visualize the data for exploration and insight at each stage of the workflow.	22

Figure 2.10	The grid created for kriging in relation to other datasets for the FORGE project.	23
Figure 2.11	The Mount St. Helens reseach area.	26
Figure 2.12	A location of interest where the water table is shown in the seismic and GPR images: top view.	26
Figure 2.13	A location of interest where the water table is shown in the seismic and GPR images: bottom view.	27
Figure 2.14	GPR images near the section of the debris blockage where there is thought to be an emergency pumping pipe.	27
Figure 3.1	PyVista as an underlying glue in the open-source geocomputing stack where data can be integrated and exchanged.	29
Figure A.1	PVGeo providing a link for geoscience to the VTK and ParaView realm of data visualization.	34
Figure B.1	An example 3D mesh of a salt body.	37

LIST OF TABLES

Table 1.1	List of free 3D visualization software capable of handling complex geoscientific data or models.	5
Table 2.1	PyVista Data Structures	14
Table C.1	List of supplemental electronic files.	40

LIST OF ABBREVIATIONS

Application Programming Interface	API
Digital Elevation Model	DEM
Geographic Information System	GIS
Ground Penetrating Radar	GPR
Open Mining Format	OMF
Three-dimensional	3D
Visualization Toolkit	VTK

ACKNOWLEDGMENTS

First and foremost, my thesis and my contributions to the PyVista project, as well as other open-source geoscientific software, were possible because of the support of my advisor Whitney J. Trainor-Guitton. Thank you to Whitney for the advice, knowledge, and continual challenge to pursue making a lasting impact in the geosciences. This thesis and my contributions to open-source software are my attempts at leaving my mark on earth science research. In a similar role, my committee members Dr. Yaoguo Li and Dr. John Bradford were a huge help in the realization of this project's impacts.

My deepest appreciation goes to the countless people who have given encouragement, insight, and help throughout the inception and continuing growth of the PyVista project. As an open initiative, the software has taken on a community of developers and users, for whom the project owes its success. I am deeply honored and humbled by that community's growth, support, and willingness to take on PyVista, and I am so excited to see where next the community takes the software. Further, I would like to thank Alex Kaszynski for all of his work in the inception and continuing development of the PyVista software; without Alex, PyVista would not exist today.

Last, but certainly not least, thank you to my friends and family. Your endless encouragement and frequent breaks from seriousness were much needed in this journey, keeping me sane.

CHAPTER 1

CURRENT STATE OF 3D GEOSCIENCE

Building three-dimensional (3D) subsurface models from field data and imaging techniques is a ubiquitous task in geological, hydrological, and resource estimation studies [1, 2]. These created models and results often hold high significance to stakeholders, yet the availability of open software to build, integrate, and make decisions with these data and models remains limited. Geoscientists need an underlying framework that can handle a variety of 3D data structures and visualize complex geospatial data types in addition to the limited data types that current free software handle. Such a framework would need to act as a *glue* across existing geo-software, streamlining data management, visualization, and development of reproducible research workflows.

Available free and open-source software to integrate subsurface information as well as to visualize those data in a 3D rendering space is limited. The limitations have arisen from disconnects between commercial software used across industry and a lack of open-source tooling within the research community. With little to no open software available, steep licensing fees for commercial software have created a barrier to entry for working in complex 3D environments. These high costs are arguably most relevant in academic and consulting settings, where research budgets are limited. Lack of software availability often leaves researchers without a toolset for integration across their different data types, such as pairing well locations, resource models, and geophysical images as shown in Figure 1.1, which can hinder their ability to interpret the spatial relationships of varying data types. This thesis proposes a new framework, PyVista, for managing and visualizing 3D geoscientific data to enhance how researchers explore, process, and communicate their data and spatial findings.

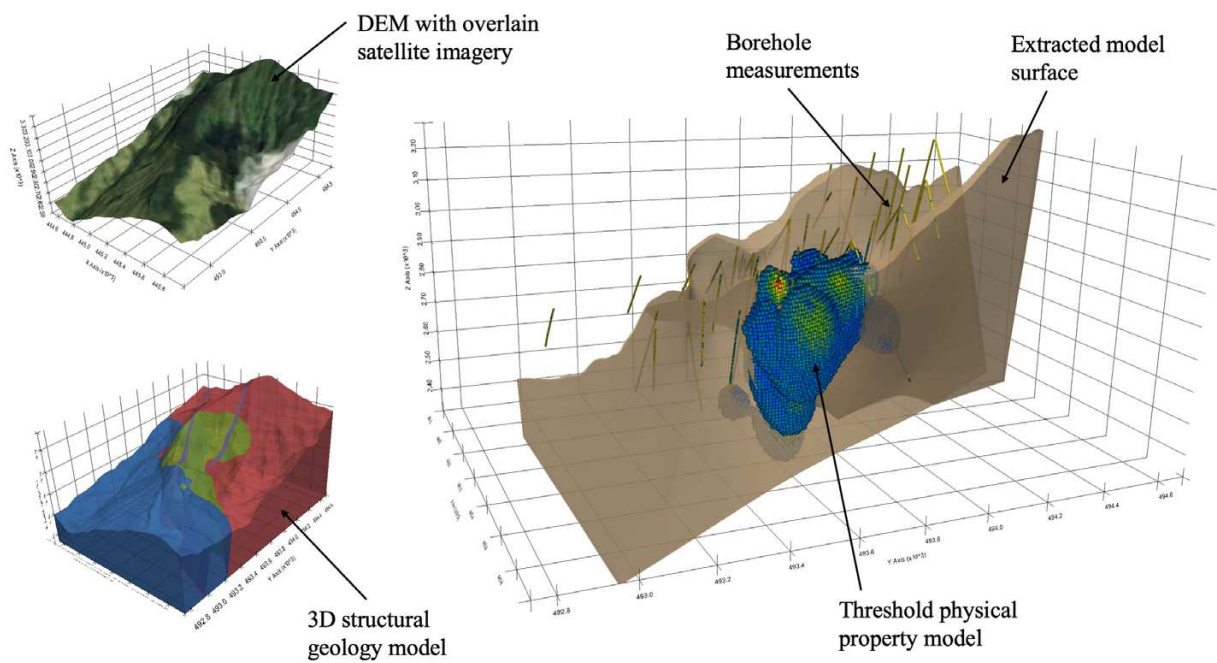


Figure 1.1: An example of a typical subsurface study where data and models are co-located, including contextual information (DEM and satellite imagery), borehole measurements, a 3D structural geology model, and a 3D physical property model.

1.1 Data Management & Integration

The subsurface models and data in typical hydrological or geophysical studies come in many forms, typically depending on available data, the expertise of the researcher, and the tooling to create and interpret such models. The value of these models is most significant when they can be co-located with other models, data, and site information in a unified environment for processing, modeling, and 3D visualization [2]. A typical hydrogeophysical investigation, for instance, might include the data types shown in Figure 1.2: a) geospatial and site contextual information, b) digital elevation models (DEMs) and LiDAR scans, c) 2D geophysical image sections (produced by ground-penetrating radar or shallow seismic methods), d) sparse observational data (geochemical sampling, surface magnetic responses, etc.), e) 3D structural geology models, or even f) 3D geophysical models or images. Additional data types include well logs, hydrological pumping data, water table monitoring data, and more. There is a wide range of data types present in typical hydrogeophysical studies; being able to gather all of the data types for a given project into a single framework for co-interpretation and analysis is challenging and often unachievable at an affordable cost.

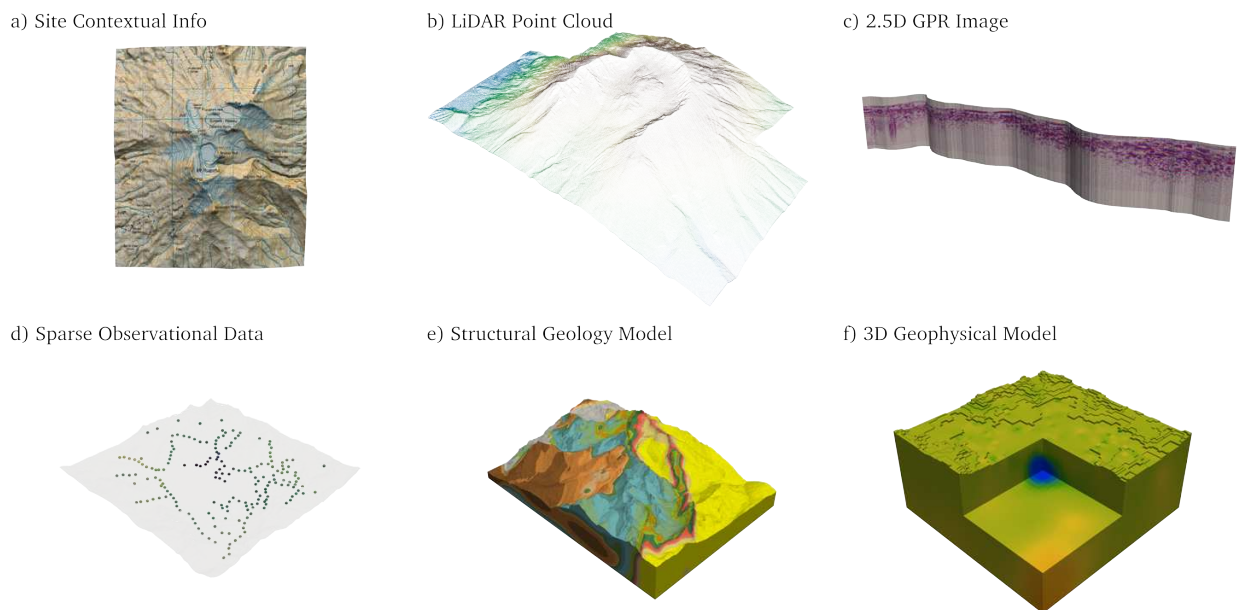


Figure 1.2: Typical data types found in subsurface research.

There are packages available for geoscientific data integration and visualization, however, these software often handle a few proprietary data formats and are closed-source with licensing fees. Having the ability to fuse datasets, construct 3D models, or generate horizons within the visualizations can be what separates closed-source software from open-source software. To date, there is no underlying framework for managing data types with a spatial reference that is open and extensible for the diversity of data types in subsurface science. Notable efforts have succeeded in creating robust frameworks such as Xarray [3] for regularly gridded data, GeoPandas [4] for geospatial feature data sets, or PyHSPF for hydrologic and water quality data integration [5]. Many of the existing tools are limited in the types of 3D data and geometries they can store and lack a clear connection to other software for data exchange and visualization. These limitations present a need for a robust and general toolset for working with spatial data that brings interoperability across existing scientific research software.

1.2 3D Visualization and Fusion

The term visual fusion refers to the synthesis of spatial data from multiple sources to provide a single view of all of the points and attributes of those data sources. In the geosciences, researchers often need 3D and 4D (time-varying) visualizations to understand complex spatial and temporal relationships in data, relationships that are challenging to capture in 2D visualizations [2]. Better perceptions or new understandings may arise from data when referenced in relation to intuitive features like topography, well locations, survey points, or other known information. Visualizing the spatial organization of the subsurface is necessary for geoscientists and stakeholders to be able to directly engage with their data and gain insight [1, 2, 6]. Current practices require skilled personnel to simultaneously evaluate data and models on various 2D planes: a qualitative and partial representation of complex subsurface data [1]. These qualitative investigation techniques can lead to incorrect findings, but this can be mitigated through visual fusion in an integrated 3D environment [1, 2].

While the availability of software for data integration and management is limited, there is a wide availability of 3D visualization software suitable for geoscientific data. However, it is important to note that many of the open-source packages are not capable of handling spatially-referenced datasets with complex geometries, and some are powerful yet have inherently complex application programming interfaces (APIs). Table 1.1 provides a list of many free visualization software capable of integrating geoscience datasets with interpretation and model building in mind. Table 1.1 compares: *API Complexity* (the software’s relative ease of use in a programming environment), *Data Structures* (describes whether the software has explicit spatial data structures for managing spatial data), and *Scripting* (describes whether the software can be incorporated into reproducible workflows).

Table 1.1: List of free 3D visualization software capable of handling complex geoscientific data or models.

Software	API Complexity	Data Structures	Scripting
PyVista	Low	Yes	Yes
LavaVu	Low	Yes	Yes
Mayavi	High	Yes	Yes
ParaView	High	Yes	Yes
SGeMS	High	Yes	Yes
plotly	Low	No	Yes
SketchUp Free	High	No	No
Blender	High	No	No
Geoscience ANALYST	N/A	Yes	No

1.3 The Rise of Python-based Research Software

Development for the next generation of geoscience research software focuses on being widely available and approachable, achieved through the open-source model, and enabling anyone to contribute code and anyone to use those tools. Examples of emerging open software in the geosciences include implicit geological modeling software like GemPy [7] and geophysical simulation and inversion software like SimPEG [8] and PyGIMLi [9] as well as

larger efforts including the Pangeo Project¹: all of these are built atop the Python data science ecosystem. Python's popularity is rising because it is both relatively easy to learn and flexible, making consistent, well-tested tools that can be extended, adapted, and combined readily accessible to researchers [8]. As an interpreted language, Python facilitates interactive development and visualization of results throughout research workflows. The interactive nature of Python has led to its rise as the central tool around emerging research topics where researchers can rapidly explore their data, gain insight, and make decisions in-situ.

With the rapid growth of Python-based software, differences between data types have made the integration of data and workflows complicated and non-reproducible. This is especially relevant in the case of closed source software where scripting interfaces exist but are often sandboxed into a framework that makes it challenging to work with external software. Furthermore, geoscientists often use specific visualization software for different data processing routines, which can lead to using several different visualization and analysis environments for a single project: further fragmenting their workflows and decreasing reproducibility. The availability of software for interactive analysis of complex geospatial data is limited due in part to inconsistencies around data integration and exchange which make it challenging for researchers to work across many of the open software tools. The inconsistencies around data integration and exchange result from the wide variety of data types used in geoscientific research as well as differences in physical scale and properties that are represented by those data. The Python software ecosystem for geoscientific research needs an open and extensible framework for data management and visualization to mitigate these challenges.

¹The Pangeo Project: <https://pangeo.io/>

1.4 Thesis Outline

The software presented in the remainder of this thesis addresses the challenges around 3D geospatial data exploration and reproducibility in geoscience computing workflows through a framework that enables scientists to work laterally across existing and emerging Python-based research software. This software provides data management for the diverse data types, like those in Figure 1.2, found in earth science research as well as streamlined routines for creating integrated visualizations like that shown in Figure 1.1.

Chapter 2 introduces the PyVista software with a modified inclusion of PyVista’s Journal of Open-Source Software publication in Section 2.1 to outline the basis of the software and its goals. Chapter 2 goes on to explore how PyVista’s data structures provide a data management framework that is useful for earth science research and interfaces that are exposed to researchers for filtering and visualizing 3D data. The concepts of visual data fusion are explored later in Chapter 2 and example code with PyVista is shown to create integrated 3D visualizations of complex geospatial data. With the PyVista software as a data management and visualization toolset, I explore how PyVista can be leveraged to improve data exchange and software interoperability in reproducible workflows. Workflows for visually fusing data and leveraging an external library for geostatistical modeling is outlined at the end of Chapter 2 and included as a Python Jupyter notebooks the Appendix. These Jupyter notebooks are hosted permanently online².

Chapter 3 provides an overview of how PyVista expands interoperability in the Python geocomputing software scene and the future directions of open software for geospatial data management, 3D visualization, and reproducible science.

Appendix A includes the Journal of Open-Source Software publication for the PVGeo software library, a part of this work to further expand the PyVista software’s links to earth science applications. Appendix C includes supplemental electronic files that can reproduce the workflows given in Chapter 2.

²Repository of Jupyter notebooks: <https://github.com/banesullivan/thesis-notebooks>

1.5 Statement of Contribution

It is essential to acknowledge that the PyVista software has been created in a community effort, and the success of the software is a result of that community's growth and embrace of PyVista. As the primary author of the PyVista software, I have made substantial contributions to the conception and design of the software. My efforts during this thesis focused on expanding the 3D data types used in the software and architecting the user-facing API for streamlined data filtering and 3D plotting, as well as writing the documentation and furthering efforts to make the software accessible to geoscientists. At the time of writing this thesis, the authorship breakdown of PyVista is as shown in Figure 1.3; these statistics were generated using the open-source package `gitinspector`³. The top section of the statistical information outline how many commits each author has during the history of the project, and the bottom section details the number of lines of code written by each author.

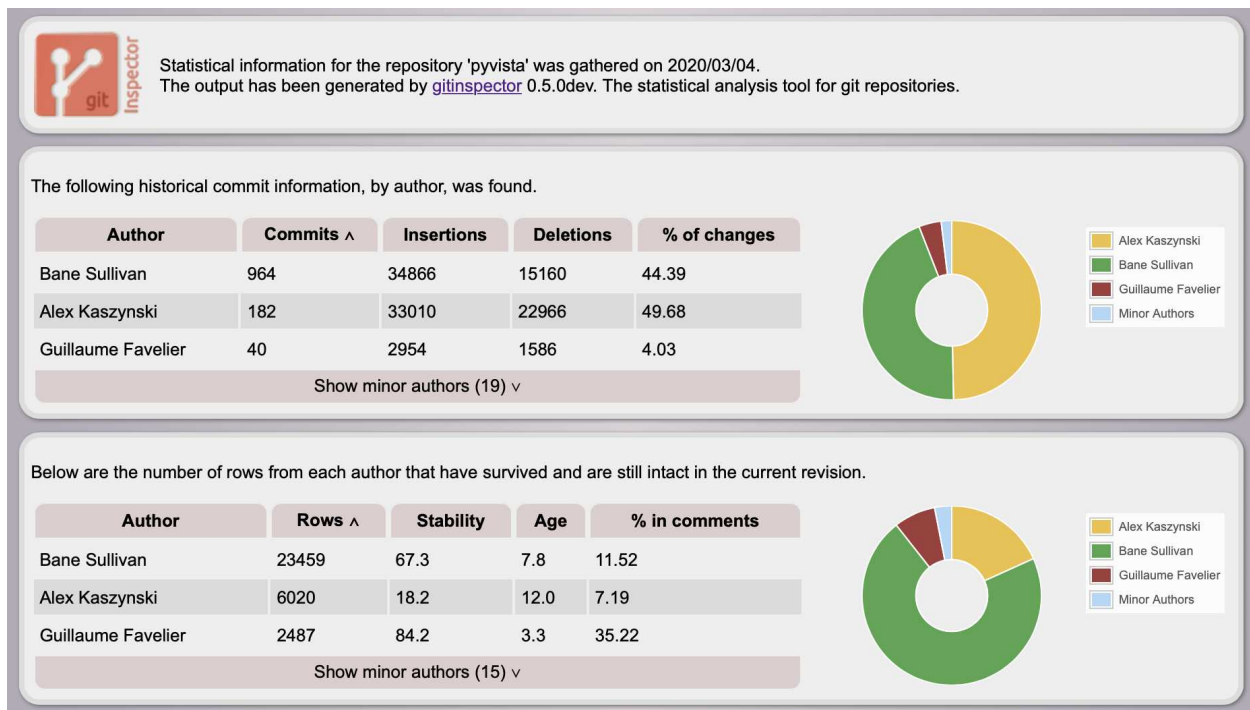


Figure 1.3: Code contribution graph for the PyVista software.

³`gitinspector`: <https://github.com/ejwa/gitinspector>

CHAPTER 2

STREAMLINING 3D GEOSCIENCE WORKFLOWS THROUGH THE PYVISTA SOFTWARE FRAMEWORK

PyVista⁴, a 3D data management and visualization software, has been developed along with a suite of companion software, including PVGeo⁵ and `omfvista`⁶, to manage, analyze, and visualize 3D geoscientific data and models in a common geo-referenced framework. This chapter provides a look into the PyVista software framework and how it is leveraged to manage and visualize geospatial data providing insight throughout Python geocomputing workflows.

Geoscientists rely on calibrating and integrating data with all types of subsurface information to further illuminate the value of modeling results and to gain insight into the research questions at hand. This software, PyVista, extends the functionality of the open-source visualization platform the Visualization Toolkit (VTK) [10] by Kitware Inc.⁷ and makes it possible to integrate geoscientific information in a multidimensional rendering space much like Figure 1.1, where viewers can relate geophysical and hydrological data/models to intuitive features like topography and well locations. The PyVista software is published in the Journal of Open Source Software⁸. This chapter provides an introduction to PyVista and an overview of applications in the geosciences. Similarly, the PVGeo companion software is published in the Journal of Open Source Software and is included in Appendix A.

⁴PyVista: <https://docs.pyvista.org>

⁵PVGeo: <https://pvgeo.org>

⁶`omfvista`: <https://opengeovis.github.io/omfvista/>

⁷Kitware Inc.: <https://kitware.com>

⁸Journal of Open Source Software: <https://joss.theoj.org>

2.1 Introduction to PyVista

C. Bane Sullivan and Alexander Kaszynski. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, May 2019. doi: 10.21105/joss.01450. URL <https://doi.org/10.21105/joss.01450>

There are several options for 3D visualization in Python, a few notable projects include Matplotlib [12], Mayavi [13], the yt Project [14], and VTK [10] (more software included in Table 1.1). However, few open-source packages are capable of managing and visualizing spatially-referenced 3D datasets and some are powerful yet have inherently complex APIs, which might create a barrier to entry for new users. Notably, VTK is a powerful scientific visualization software library, and with Python bindings, it combines the speed of C++ with the rapid prototyping of Python. Despite this, VTK code programmed in Python using the base VTK Python package is unnecessarily complicated as its API binds existing C++ calls. The PyVista Python package provides a concise, well-documented interface exposing VTK’s powerful visualization backend: enabling researchers to rapidly explore complex datasets, communicate their spatial findings, and facilitate reproducibility. PyVista further seeks to simplify standard mesh creation and plotting routines without compromising on the speed of the C++ VTK backend.

Plotting VTK datasets using only the VTK Python package or a similar visualization library is often an ambitious programming endeavor. Reading a supported file and plotting it requires a user to write a complicated sequence of routines to render the data object while having to remember which VTK classes to use for file reading and dataset mapping. PyVista includes plotting routines that are intended to be intuitive and highly controllable with syntax and keyword arguments similar to Matplotlib [12]. These plotting routines are defined to make the process of visualizing spatially referenced data straightforward and easily implemented by novice programmers: streamlining the process for creating integrated

scenes like that of the FORGE Geothermal Site ⁹ shown in Figure 2.1. A comparison of the necessary programming routines to visualize a 3D data file using both VTK and PyVista is included in Appendix B.

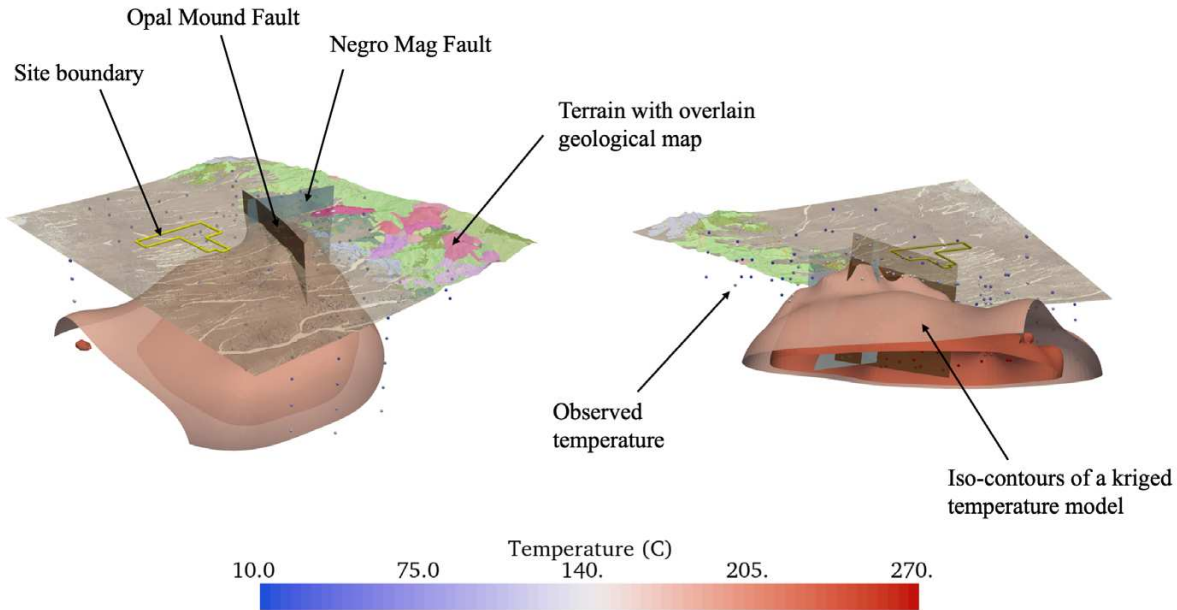


Figure 2.1: A visually integrated scene of geospatial data (FORGE Geothermal Site⁹): left) view from upper south-west and right) view from lower north-west.

VTK implements an object-oriented approach to 3D visualization [10], and PyVista adheres to that underlying structure to provide an API that expands on VTK’s data types. These expanded, wrapped types hold methods and attributes for quickly accessing scalar arrays, inspecting properties of the dataset, or using filtering algorithms to transform datasets. PyVista wrapped objects have a suite of common filters ready for immediate use directly on the objects. These filters are commonly used algorithms in the VTK library that have been made more accessible by binding a method to control that algorithm directly onto all PyVista datasets, providing a shared set of functionality. Through the use of these bound

⁹Data shown in Figure 2.1 is from Bane Sullivan and Adam Kinard’s second place entry in the 2019 Geothermal Design Challenge for the Utah FORGE underground field laboratory sponsored by U.S. Department of Energy (<https://utahforge.com>). Competition details can be found at <https://www.energy.gov/eere/articles/and-winners-2019-geothermal-student-competition-are>

filtering methods, powerful VTK algorithms can be leveraged and controlled via keyword arguments designed to be intuitive for novice users. This chapter will provide examples of PyVista’s wrapped data types and filtering methods.

At its core, PyVista is a pure Python helper module for VTK that interfaces back to VTK data objects through NumPy¹⁰ and direct array access adhering to VTK’s object-oriented approach to 3D visualization [10]. The PyVista Python package provides an accessible and intuitive interface back to the VTK library to facilitate rapid prototyping, analysis, and visual integration of spatially referenced datasets. Figure 2.1 demonstrates an integrated scene of geospatial data generated by PyVista; the code to create this 3D scene is given later in this chapter (found in Listing 2.2). Figure 2.1 includes a digital land surface with overlain satellite imagery and geologic map, a subsurface temperature model as two isosurfaces, scattered points of the sampled temperature values in the subsurface, Geographic Information System (GIS) site boundary, and interpreted faulting surfaces. PyVista’s streamlined interface to VTK includes simplified routines to texture map the satellite imagery and geologic units over the digital elevation model, threshold value ranges of interest in the subsurface temperature model, and visually integrate all the data into a single 3D view in Figure 2.1.

2.2 PyVista: Data Management & Visualization

The PyVista software package has two primary modules: 1) the `core` module, the fundamental data structures for managing spatially-referenced information, and 2) the `plotting` module, a comprehensive toolset for creating integrated 3D visualizations of the `core` data structures. The following sections outline how PyVista’s generalized data structures provide an extensible framework for managing geoscientific data and how PyVista’s data management approach to 3D visualization can be leveraged for insight throughout geocomputing workflows.

¹⁰NumPy is the fundamental package for scientific computing in Python containing powerful n-dimensional numerical array objects and linear algebra functions [15].

2.2.1 Data Structures

Geoscientific data and models are discrete, a fact arising from how we acquire and represent these data in digital form as geospatial data represents distinct points or regions in 3D space. The nature of spatial data requires that any computational framework for managing and analyzing those data have discretized structures for containing those data. VTK, and PyVista by extension, is a framework with discrete structures that can handle the majority of 3D data types found in subsurface science. As such, PyVista creates a platform for managing spatially-referenced data and visualizing those data in a robust 3D rendering platform aiming to make analysis and visualization routines more accessible in Python. PyVista implements the spatial data structures listed in Table 2.1 to make creating, managing, and analyzing VTK data types more streamlined for domain scientists. These spatial data structures, commonly referred to as datasets, consist of *geometry*, *topology*, and *attributes* to which PyVista provides direct access over a NumPy¹⁰ interface.

- *Geometry* is the collection of points and cells in 2D or 3D space.
- *Topology* defines the structure of the dataset, or how the points are connected to form cells defining a surface or volume.
- *Attributes* are any data values that are associated with either the points (the nodes) or cells (regions between the nodes) of the dataset.

As mentioned in Section 1.1, there is a wide variety of data types used in geoscientific research settings; all of the examples listed have a corresponding class (data structure) in PyVista. For instance, Figure 2.2 shows how `PolyData` can represent a) point clouds, b) DEMs with overlain imagery, c) sparse observational data, d) geospatial line sets, and e) bounding surfaces within structural geology models. Furthermore, gridded volumes can be represented by either the `RectilinearGrid` or `UniformGrid` classes, meshed volumes and surfaces with a gridded structure can be represented by the `StructuredGrid`

Table 2.1: PyVista Data Structures

PyVista Class	Description	Example
<code>PolyData</code>	Consists of any 1D or 2D geometries to construct vertices, lines, polygons, and triangles. Generally used for scattered points and closed/open surfaces (non-volumetric datasets).	See Figure 2.2
<code>RectilinearGrid</code>	Implicit geometries along the axes directions that are rectangular and regular.	See Figure 2.3-a
<code>UniformGrid</code>	Implicit geometries where cell sizes are uniformly assigned along each axis and the spatial reference is built out from an origin point.	See Figure 2.3-b
<code>StructuredGrid</code>	A regular lattice of points aligned with an internal coordinate axes such that the connectivity can be defined by a grid ordering. Contains 2D Quads or 3D Hexahedrons.	See Figure 2.3-c
<code>UnstructuredGrid</code>	The most general dataset type that can hold any 1D, 2D, or 3D cell geometries. Typically generated by filtering routines and contain subsets or combinations of the above data types.	See Figure 2.6

class, and meshed volumes and surfaces of arbitrary geometry can be represented by the `UnstructuredGrid` class.

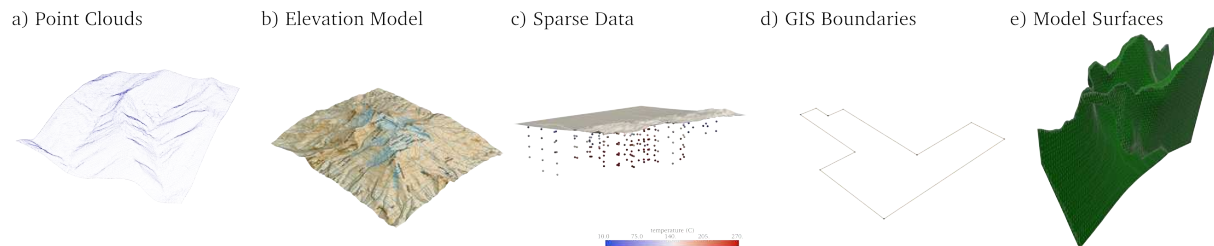


Figure 2.2: Common forms of geoscientific data that can be represented as 1D and 2D geometries in 3D space via the `PolyData` class.

Typically in subsurface geoscience, researchers leverage `RectilinearGrid`, `UniformGrid`, and `StructuredGrid` data structures for 3D modeling (shown in Figure 2.3). A `RectilinearGrid` is commonly used when the volume is aligned with the cartesian axes and varying cell size is desired along each of the axes. The `UniformGrid` is used likewise when the volume is aligned with the cartesian axes, but when the cell size along a given axis is uniformly de-

finer. The `StructuredGrid` class is also commonly used when dealing with meshed volumes of arbitrary rotation or curvilinear internal structures. The examples in Figure 2.3 show these three common mesh types in a simplified form, but these types are often used at a larger scale in extensive geophysical modeling frameworks like SimPEG [8] where there are equivalent classes for the `RectilinearGrid` and `StructuredGrid` types.

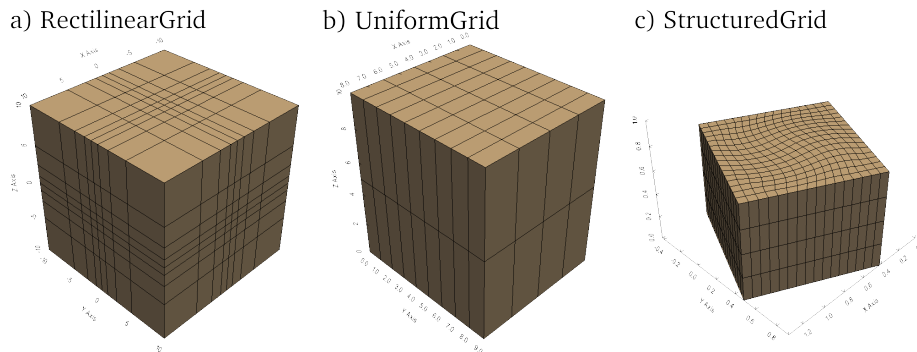


Figure 2.3: Three commonly used mesh/volume types used in subsurface modeling applications.

The data in Figure 2.1 primarily utilizes PyVista’s `PolyData` type to represent the faulting planes, geospatial site boundary, terrain surface, and subsurface temperature measurements but also leverages PyVista’s `UniformGrid` type for the 3D temperature model that is iso-contoured. These data can be collected into a single PyVista `MultiBlock` composite structure, as shown in output 2 of Figure 2.4, that is used to manage all of these spatial data in a computational environment. `MultiBlock` objects streamline accessing and managing each dataset as well as creating integrated visualizations. Any dataset for the project can be accessed by name from the container shown in Figure 2.4, such as the *Terrain* mesh by the code `project[‘Terrain’]` yielding the `PolyData` mesh in output 3. Further, each dataset in this data structure can be utilized and visualized individually as shown in Figure 2.5.

PyVista’s data structures make it possible to collect, manage, and track all of the spatial information for a given subsurface investigation in a consistent and accessible framework. These data structures are each co-located in a shared geo-referenced coordinate system making interpretation and analysis a fluid and interactive process for researchers as well as

```

type(project)
[1]: pyvista.core.composite.MultiBlock

[2]: project
[2]:
Information                                     Blocks
-----
Index      Name      Type
0          Terrain  PolyData
MultiBlock Values
N Blocks   8
X Bounds   329700.000, 344450.000
Y Bounds   4252600.000, 4271100.000
Z Bounds   -2700.000, 2906.000
3          1      Opal Mound Fault  PolyData
4          2      Negro Mag Fault  PolyData
5          3      Top of Basement  PolyData
6          4      Site Boundary    PolyData
7          5      Observed Temperature  PolyData
8          6      Observed Gravity  PolyData
9          7      Kriged Temperature Model  UniformGrid

[3]: project["Terrain"]
[3]:
Header                                     Data Arrays
-----
PolyData      Information
N Cells       824278
N Points       413250
X Bounds       3.299e+05, 3.442e+05
Y Bounds       4.253e+06, 4.271e+06
Z Bounds       1.494e+03, 2.723e+03
N Arrays       3
Name  Field  Type  N Comp  Min  Max
geo_aer  Points  float32  2  3.737e-01  8.576e-01
geo_no_aer  Points  float32  2  3.737e-01  8.576e-01
topo_map  Points  float32  2  2.254e-01  8.844e-01

```

Figure 2.4: An example output of PyVista data containers in a Jupyter Python environment.

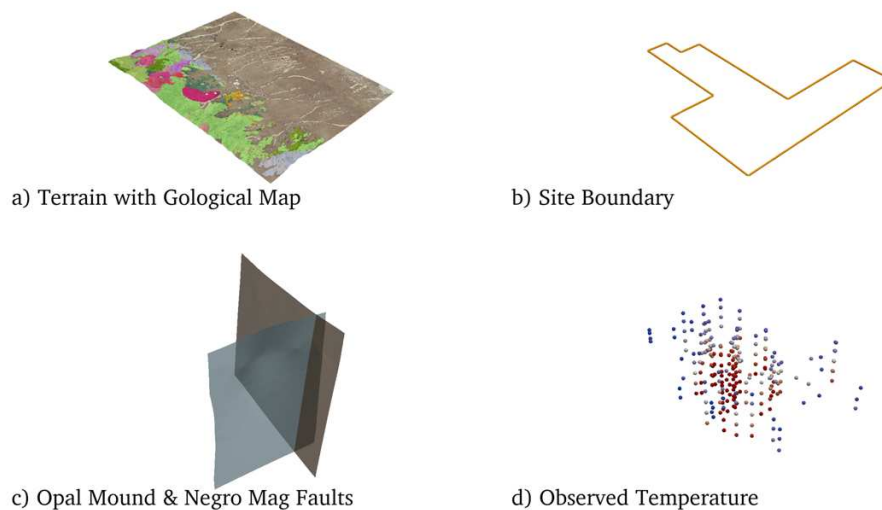


Figure 2.5: Several of the datasets from the data structure in Figure 2.4 shown individually.

providing straightforward routines to visually fuse datasets, combine attributes, and gain insight into the spatial relationships of the data before and during scientific processing.

Once the data for a workflow are collected into PyVista data structures, aspects of the data like their bounding boxes, coordinates, center locations, etc. are accessible via properties directly bound to those objects. Likewise, further analysis routines are accessible as bound filtering methods available on all PyVista objects. For example, a subset can be extracted from a volumetric dataset via the code in Listing 2.1 to produce the mesh shown in Figure 2.6. PyVista's immediate access to data information, attributes, and analysis routines provides a toolset for rapidly inspecting and gaining insight from spatial data in an interactive computing environment. Full documentation of the API for PyVista data structures and how users can interact with the bound properties and methods can be found within PyVista's online documentation¹¹.

```
model = project["Kriged Temperature Model"]  
threshold = model.threshold(150)
```

Listing 2.1: Example code to threshold a the *Kriged Temperature Model* creating an `UnstructuredGrid` mesh where the temperature is above 150°C.

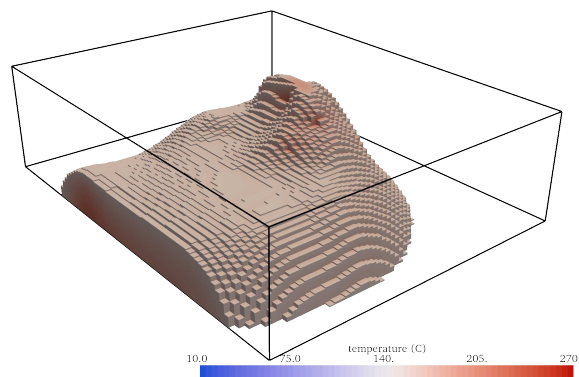


Figure 2.6: A threshold of the the *Kriged Temperature Model* produced from Listing 2.1 at 150° C.

¹¹PyVista's online documentation: <https://docs.pyvista.org>

2.2.2 Visual Data Fusion

Visual data fusion is often used to gain insight into the spatial trends and relationships between several datasets, insights that can be achieved through visualization and exploration with PyVista. To explore the spatial relationships of several datasets, a researcher would visualize those data sources in a common geo-referenced scene for exploration. For example, Figure 2.7 demonstrates visual data fusion between a resource model and borehole data in 3D (*a* and *b*) and in 2D (*c*). Figure 2.7 shows the 3D resource model threshold at a value of interest (*a*) and sliced along a plane (*b* and *c*) with co-located borehole measurements in 3D (*a* and *b*) and projected to the same plane (*c*). The 2D scene in Figure 2.7-*c* shows the borehole data projected to the same viewing plane as the cross-section leading to visual differences between the model and the borehole data; this 2D representation of the data only partially represents the 3D variability of the model. Through leveraging PyVista’s data management framework, researchers have a toolset for accessing, filtering, and visually fusing all the spatial data for their workflow through an intuitive, programmatic interface: enabling the creation of sophisticated 3D visualizations in a reproducible fashion.

Similarly, integrated 3D visualizations of the data listed in Figure 2.4 can be created in a few lines of code when managing the data via PyVista throughout a research workflow. PyVista’s plotting routines are built to directly handle the spatial data types such that these objects can be directly passed to a routine for rapid visual data fusion. In Listing 2.2, several of the datasets from the composite structure in Figure 2.4 are fetched and passed to a PyVista `Plotter` instance which creates a 3D rendering scene; each object is passed with a set of descriptive keyword arguments to define how that dataset is displayed (e.g., color, opacity, and point size). The code in Listing 2.2 produces the fully integrated scene shown in Figure 2.1. The contouring filter applied in Listing 2.2 as `.contour([175, 225])` could be changed to a thresholding filter as `.threshold(150)` from Listing 2.1 to extract the volumetric region where the scalar data is above the given value: this result is shown in Figure 2.8.

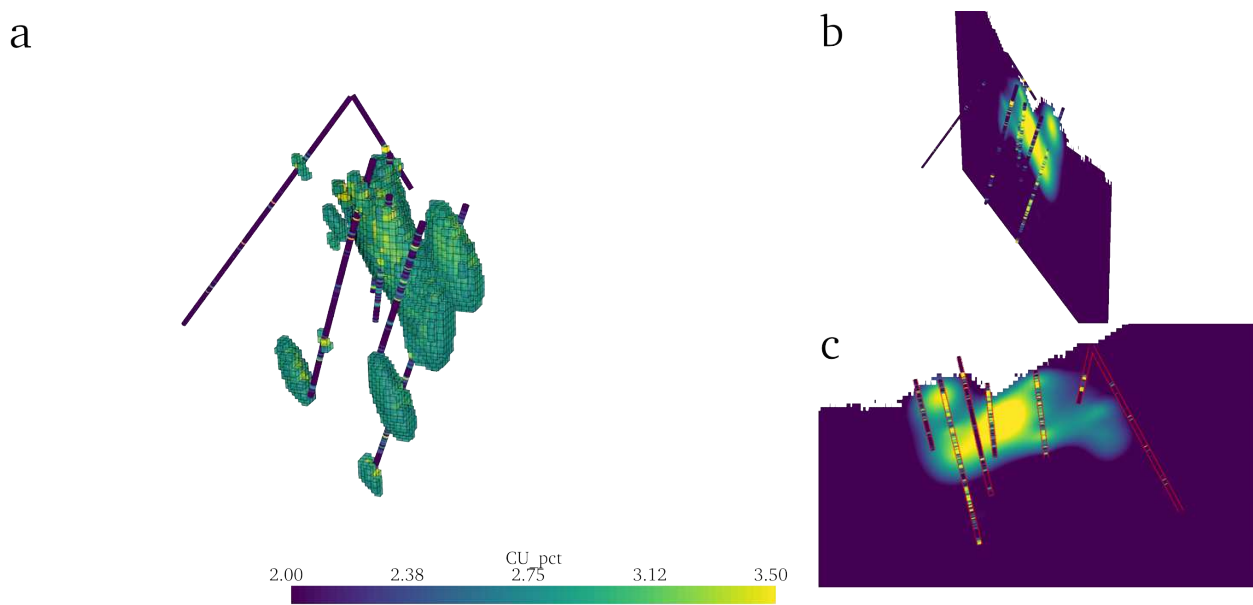


Figure 2.7: Examples of visual fusion between two datasets: a 3D resource model and borehole measurements.

```

p = pyvista.Plotter()
p.add_mesh(project["Site Boundary"],
           color="yellow", render_lines_as_tubes=True)
p.add_mesh(project["Terrain"],
           texture="geo_aer", opacity=0.7, lighting=False)
p.add_mesh(project["Opal Mound Fault"],
           color="brown", opacity=0.7)
p.add_mesh(project["Negro Mag Fault"],
           color="lightblue", opacity=0.7)
p.add_mesh(project["Kriged Temperature Model"].contour([175, 225]),
           cmap="coolwarm", clim=[10,270], opacity=0.9)
p.add_mesh(project["Observed Temperature"],
           cmap="coolwarm", clim=[10,270], point_size=10,
           render_points_as_spheres=True)
p.show()

```

Listing 2.2: The code to produce the 3D visualization shown in Figure 2.1 from the MultiBlock dataset in Figure 2.4.

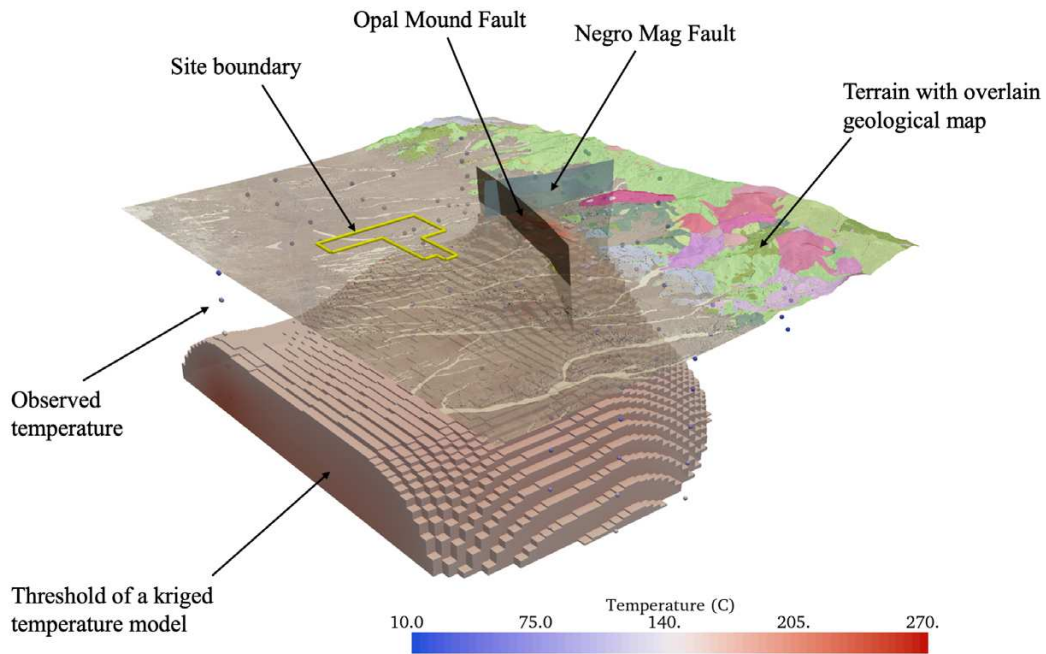


Figure 2.8: The same geospatial data from Figure 2.1 (FORGE Geothermal Site⁹) with a threshold temperature model as produced from Listing 2.1 and Listing 2.2.

2.3 Data Exploration & Reproducible Workflows

Data exploration is the process of quickly extracting insight from data, typically through visualization techniques, when the investigator is unaware of regions or attributes of interest at the onset [16]. In traditional scientific computing workflows, researchers use data exploration techniques to gain insight into the raw data when pre-processing to inform decisions during simulation/modeling as well as to gain insight from the modeling results. The ability to rapidly visualize and explore spatial datasets in geoscientific research would provide an invaluable tool to drive decision making and build insight throughout the scientific computing process.

The PyVista framework has created an ecosystem of software built atop PyVista’s robust wrappings of VTK, making it relatively straightforward to develop custom visualization and mesh analysis tools for subdomain problems. The PVGeo software package (see Appendix A) is one such effort that has routines tailored to data visualization and analysis in the

geosciences with a heavy focus on structured data sets like 2D or 3D time-varying grids. Other tools that are built atop PyVista include `omfvista`⁶ for 3D visualization of the Open Mining Format (OMF), `PyMeshFix`¹² for repairing holes in PyVista surface meshes, and `TetGen`¹³ for generating tetrahedral meshes of any 3D polyhedral domains. Furthermore, several geoscience-oriented Python software link to PyVista; an example geostatistical modeling with `GSTools` [17] will be demonstrated in Section 2.3 of this chapter.

The PyVista data management and visualization framework enables exploration at every stage of computing workflows. Since PyVista’s data structures are built with a direct access `NumPy`¹⁰ interface, the data can be passed to external Python processing and analysis routines then collected directly back into the PyVista data structures at any stage of the workflow. This would enable, for example, a geophysical inversion workflow where observational data is passed to an external modeling library, and the intermediate results of the inversion could be displayed in real-time at each realization: this workflow is demonstrated in Figure 2.9.

In this case, PyVista acts as an underlying data management platform that stores and visually integrates geospatial data. Figure 2.9 demonstrates PyVista’s role as a data management framework with 3D visualization and analysis built-in as a tool for sending data to external software and collecting their results for visual fusion. As an underlying layer managing all of the data for a given geoscientific investigation, PyVista leverages data standards common throughout the Python computing stack to make interoperability and data exchange simple. The following sections demonstrate geocomputing research workflows where PyVista manages the spatial data providing exploration tools while other scientific computing libraries are used for modeling.

¹²`PyMeshFix`: <https://github.com/pyvista/pymeshfix>

¹³`TetGen`: <https://github.com/pyvista/tetgen>

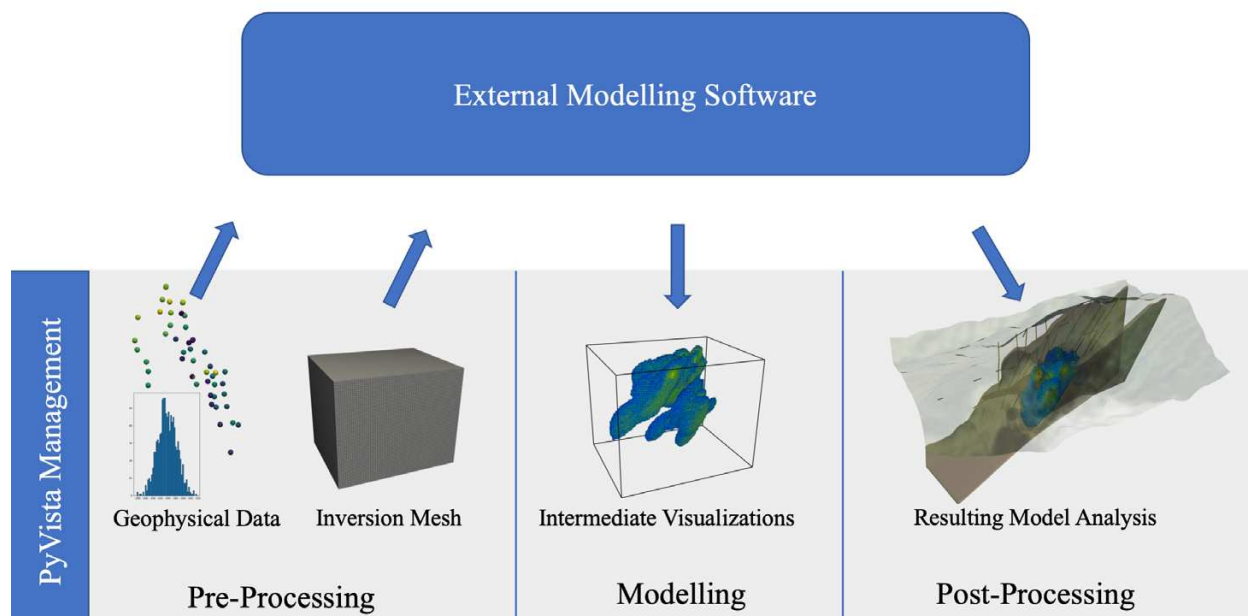


Figure 2.9: A typical geoscientific computing workflow (left to right) where PyVista can manage and visualize the data for exploration and insight at each stage of the workflow.

2.3.1 Example: Geostatistical Kriging

This example works from the data already shown in the previous sections for the FORGE⁹ geothermal research site (see data in Figure 2.1 and Figure 2.4) to create the 3D **Kriged Temperature Model** by kriging the *Observed Temperature* data (sparse observational data). The open-source, Python software GSTools [17] is used to perform variogram analysis and kriging of the temperature data onto a PyVista mesh. A complete, reproducible version of this workflow is included as a Python Jupyter Notebook in `Kriging.ipynb` of Appendix C with the data included in the file `FORGE.omf`.

The workflow starts by creating a 3D grid as a PyVista `UniformGrid` object that encompasses the region to Krige. Listing 2.3 details the creation of this grid from user define parameters and shows code to visualize that grid with the *Terrain* and *Observed Temperature* datasets in Figure 2.10 to ensure it is placed properly.

```

# Create the kriging grid
grid = pyvista.UniformGrid()
# Bottom south-west corner
grid.origin = (329700, 4252600, -2700)
# Cell sizes
grid.spacing = (250, 250, 50)
# Number of cells in each direction
grid.dimensions = (60, 75, 100)

# Visually inspect the kriging grid in relation to data
p = pyvista.Plotter()
p.add_mesh(grid, opacity=0.5, color=True)
p.add_mesh(project["Terrain"],
            texture="geo_aer", opacity=0.75)
p.add_mesh(project["Observed Temperature"],
            cmap="coolwarm")
p.show()

```

Listing 2.3: Creation of the grid for kriging.

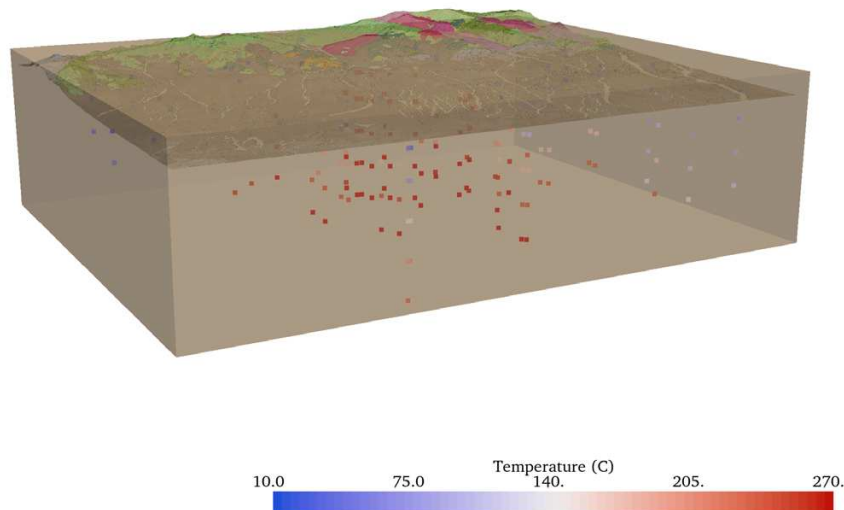


Figure 2.10: The grid created for kriging in relation to other datasets for the FORGE project.

Next, an exponential variogram model is fit to the sparse observational data and kriging is performed on a model domain contained by a PyVista `UniformGrid`.

```

import numpy as np
from gstools import Exponential, krig, vario_estimate_unstructured

# Estimate the variogram
bins = np.linspace(0, 12300, 1000)
bin_center, gamma = vario_estimate_unstructured(
    project["Observed Temperature"].points.T,
    project["Observed Temperature"]["temperature (C)"],
    bins)

# Fit an Exponential model to the variogram
fit_model = Exponential(dim=3)
fit_model.fit_variogram(bin_center, gamma, nugget=False)

# Create the kriging operator from the Exponential model
krig = krig.Ordinary(fit_model,
                    project["Observed Temperature"].points.T,
                    project["Observed Temperature"]["temperature (C)"])

# Run the kriging operator on the PyVista grid
krig.mesh(grid, name="temperature (C)")

# Place the grid into the project MultiBlock dataset
project["Kriged Temperature Model"] = grid

```

Listing 2.4: Performing the kriging and collecting the results back into PyVista data structures.

With the code in Listing 2.4, the *Kriged Temperature Model* is now a part of the project composite dataset that is shown in Figure 2.4 and the code in Listing 2.2 produces an integrated visualization with this kriged model in Figure 2.1. Likewise, the *Kriged Temperature Model* can have any number of filters applied directly to it, much like the thresholding example in Listing 2.1.

2.3.2 Example: Visual Fusion

The following figures were created with PyVista in collaboration with Kristen Marberry's thesis [18] exploring the hydrogeophysical conditions of the debris blockage at Spirit Lake near Mount St. Helens. Figure 2.11 shows the research site area adjacent to Mount St. Helens and the adjacent lake. This investigation's focus was to provide a geophysical interpretation

of the subsurface, including locating the water table and a buried 1.8 *m* diameter steel pipe installed for the emergency pumping of Spirit Lake in the early 1980s. Ground-penetrating radar (GPR) and shallow seismic images were collected and are shown in a common georeferenced environment to gain insight into subsurface conditions, identify regions where the water table is perched, and locate the buried pumping pipe. Figure 2.12 and Figure 2.13 show a section of the seismic and GPR images where a perched water table is possibly present beneath the debris blockage. The seismic refraction velocity analysis shows a region around 1500 *m/s* adjacent to where the GPR images show coherent horizontal reflectors, possibly indicating the presence of a perched water table. Figure 2.14 shows a section of the GPR images near where the emergency pumping pipe is proposed to be located, and two coherent reflections are visible in line with the proposed path.

The GPR and seismic processing workflows for this investigation were performed in several separate software packages, making it challenging to collect the results into a single environment for co-interpretation. The PyVista framework provides tooling to collect the processed data from those fragmented workflows and bring all of the data together. PyVista enabled us to gain insight into the 3D relationships and variability of all of the project data by showing the GPR and seismic images next to each other in 3D with a high-resolution surface DEM whereas only 2D analysis was previously available. The reproducible workflow given in `Mt_St_Helens.ipynb` of Appendix C with the `MSH.vtm.zip` data archive yields these figures and provides this demonstration of using PyVista for visual data fusion to gain insight into subsurface hydrological conditions. The data can be further explored in interactive 3D renderings with the attached `Mt_St_Helens.ipynb` resource of Appendix C.

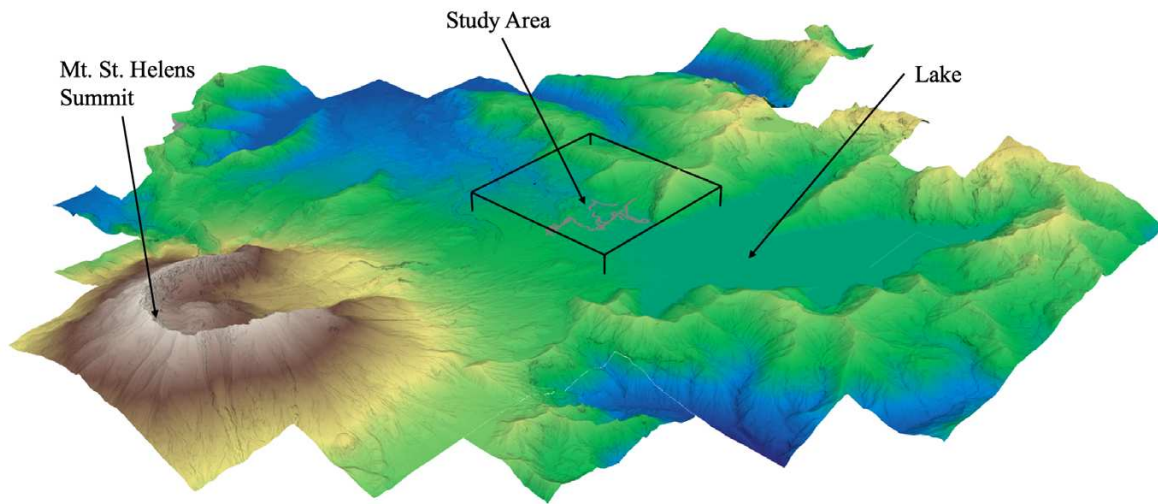


Figure 2.11: The Mount St. Helens research area.

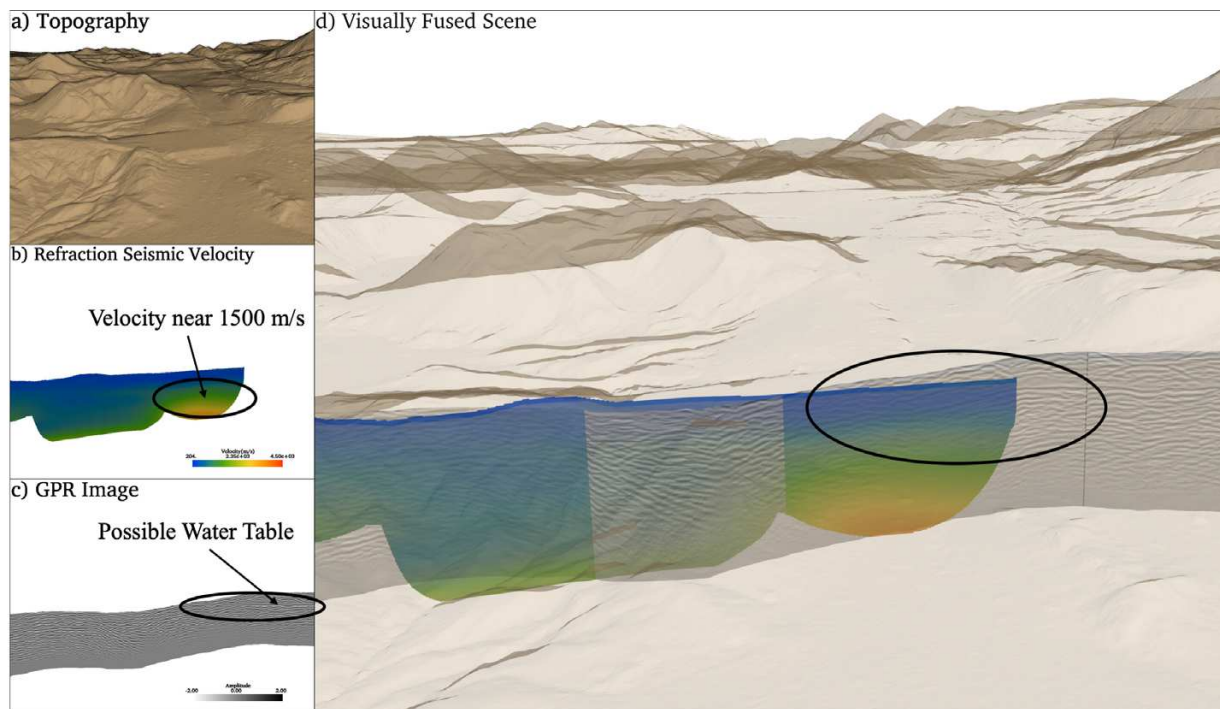


Figure 2.12: A location of interest where the water table is shown in the seismic and GPR images: top view.

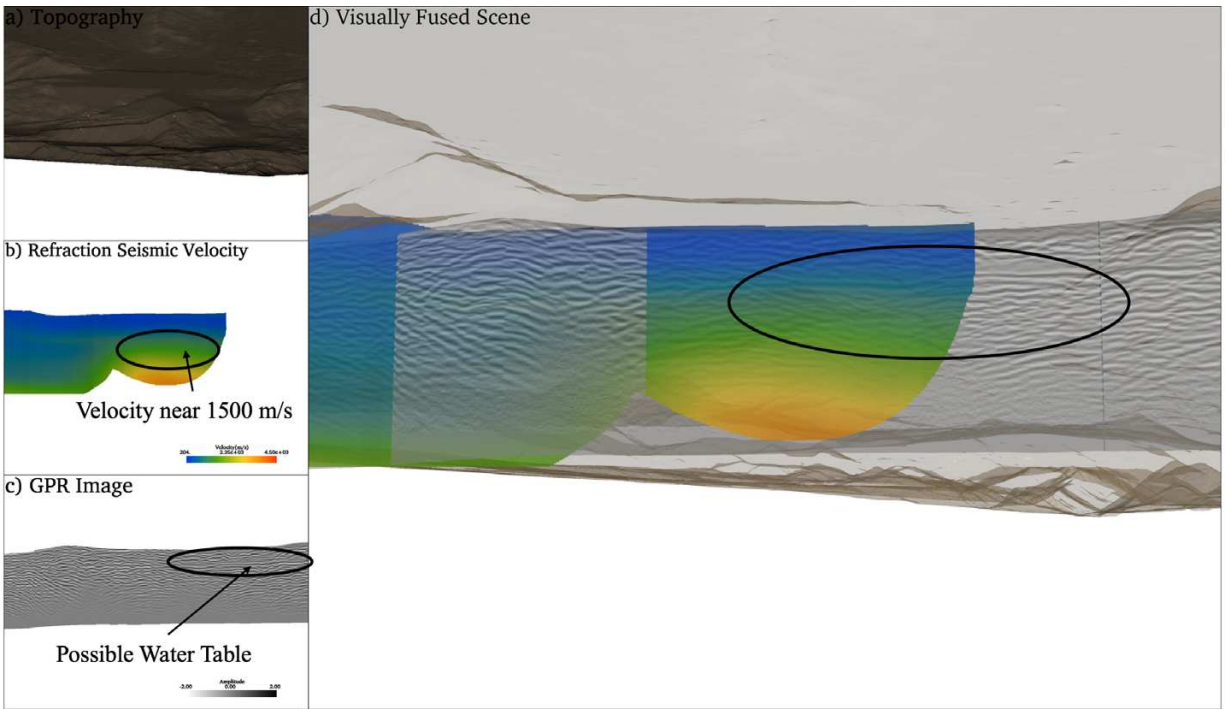


Figure 2.13: A location of interest where the water table is shown in the seismic and GPR images: bottom view.

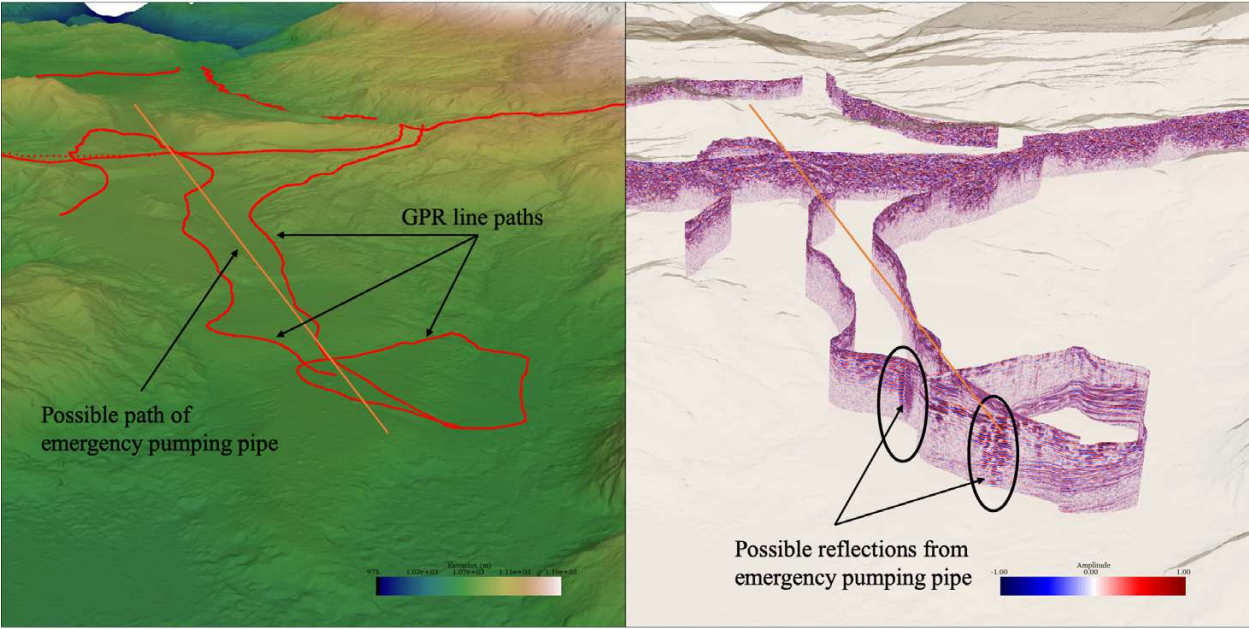


Figure 2.14: GPR images near the section of the debris blockage where there is thought to be an emergency pumping pipe.

CHAPTER 3

CONCLUSION

PyVista is a capable framework for geospatial data management and visual fusion, providing tools to integrate and make decisions with 3D data and models. The PyVista framework acts like a *glue* as an underlying layer connecting data management and visualization software with modeling and analysis libraries in the Python software ecosystem. Figure 3.1 shows how PyVista builds on the open-source Python data science stack, including libraries like NumPy¹⁰ and VTK to create an underlying layer for open-source geocomputing software where data can be integrated and exchanged between previously fragmented software. Furthermore, through connections to open data standards like OMF and PyVista’s companion package `omfvista`⁶, data can be exchanged with commercial software platforms that support those standards.

As an open-source framework designed to be used by novice programmers, PyVista breaks down the barrier to entry for 3D visualization and data management that steep licensing fees from commercial software and complicated APIs in open software have created. Researchers are able to leverage PyVista to gather all the available data for a given project into a single workflow for visualization and interpretation through PyVista’s generalized and accessible 3D data structures. This software enhances researchers’ ability to explore, process, and communicate their spatial data and findings in a reproducible fashion. Particularly, the PyVista software provides a toolset for earth science researchers to rapidly visualize, explore, and gain insight into their spatial data: improving insight and decision-making processes throughout research workflows. This toolset can be expanded to develop custom visualization and analysis tools for subdomain problems, enabling the developers of geoscientific research software to have a spatial data management and 3D visualization framework available to integrate into their own software.

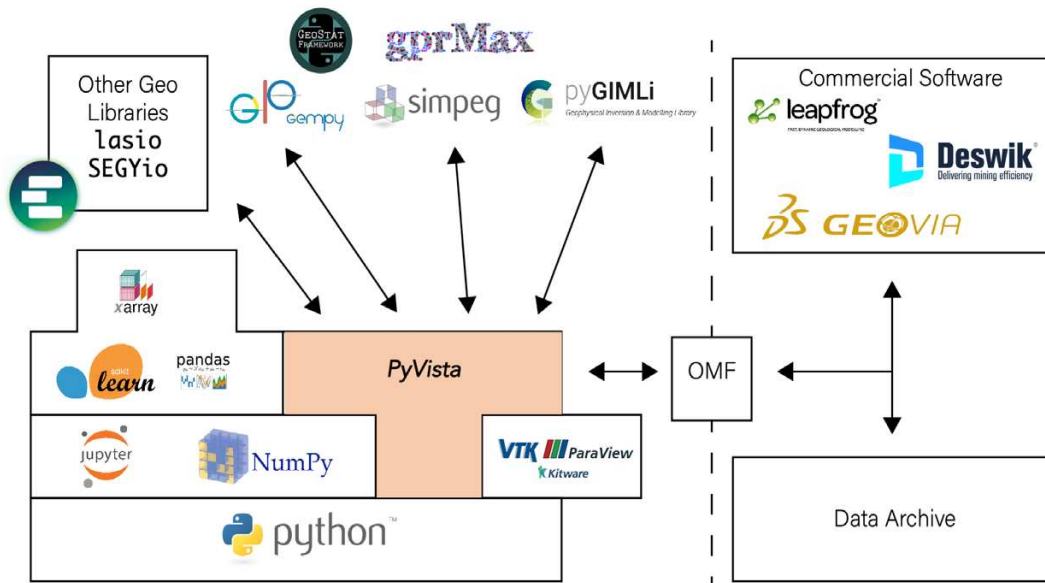


Figure 3.1: PyVista as an underlying glue in the open-source geocomputing stack where data can be integrated and exchanged.

Furthermore, the PyVista software is designed to be used by the typical hydrologist or geophysicist with limited programming experience. This ease-of-use is made possible by building PyVista on top of a modern programming language, leveraging best practices for maintainability and usability, and streamlining the installation process for PyVista and its dependencies. These efforts were undertaken to move hydrologists toward sustainable software and data practices while smoothing the modern geoscientists' transition into the era of open software.

Today the number of Python-based, open research software are growing rapidly, making powerful software immediately available to geoscientists: tooling that was previously only available from expensive, commercial software platforms. With this growth, new interfaces between new libraries and the ones shown in Figure 3.1 will need to be further developed. In its current state, PyVista provides an intuitive framework for transferring data between software, but direct data-sharing interfaces have yet to be developed for many of these

libraries. Since PyVista acts as a level playing field where geospatial data is managed and integrated, the scientific computing process becomes more dynamic across existing research tools. This further expands how available research software are able to exchange data and how researchers are able to create reproducible workflows: keeping all of their data within a single framework and leveraging the rapid growth of open-source software.

As new 3D geoscientific data types emerge and more massive datasets are collected, for example, the recent prevalence Distributed Acoustic Sensing (DAS) data, geospatial data integration and visualization will remain a dynamic challenge. Though PyVista provides an extensible framework for tackling current barriers around 3D geospatial data management and visualization, new challenges will emerge that will require further effort to build out data exchange and interoperability across geocomputing software. These efforts will need to see the rise of a centralized set of standards for working with spatial data in existing and emerging research software. Furthermore, scientific publishers will need to be adopted and enforce those standards to ensure that data processing and visualization routines for a publication are reproducible and verifiable. With the rapid growth of Python-based research software, these efforts will see rising momentum as new earth science software is released, and researchers continue to use the latest available tools. Thus expanding the need for a centralized framework for managing data between open software; PyVista is one such extensible framework well-positioned for that momentum.

REFERENCES CITED

- [1] G. Caumon, P. Collon-Drouaillet, C. Le Carlier de Veslud, S. Viseur, and J. Sausse. Surface-based 3d modeling of geological structures. *Mathematical Geosciences*, 41(8): 927–945, Nov 2009. ISSN 1874-8953. doi: 10.1007/s11004-009-9244-2. URL <https://doi.org/10.1007/s11004-009-9244-2>.
- [2] Jeffrey B Witter and Glenn Melosh. The Value and Limitations of 3D Models for Geothermal Exploration. *43rd Workshop on Geothermal Reservoir Engineering, Stanford Univeristy*, pages SGP-TR-213, 2018.
- [3] Stephan Hoyer and Joe Hamman. xarray: Nd labeled arrays and datasets in python. *Journal of Open Research Software*, 5(1), 2017.
- [4] K Jordahl. Geopandas: Python tools for geographic data. URL: <https://github.com/geopandas/geopandas>, 2014.
- [5] DJ Lampert and M Wu. Development of an open-source software package for watershed modeling with the hydrological simulation program in fortran. *Environmental Modelling & Software*, 68:166–174, 2015.
- [6] Timothy R. Carr, Rex C. Buchanan, Dana Adkins-Heljeson, Thomas D. Mettille, and Janice Sorensen. The future of scientific communication in the earth sciences: The impact of the internet. *Computers and Geosciences*, 23(5):503–512, 1997. ISSN 00983004. doi: 10.1016/S0098-3004(97)00032-0.
- [7] Miguel de la Varga, Alexander Schaaf, and Florian Wellmann. GemPy 1.0: open-source stochastic geological modeling and inversion. *Geoscientific Model Development*, 2019.
- [8] Rowan Cockett, Seogi Kang, Lindsey J Heagy, Adam Pidlisecky, and Douglas W Oldenburg. Simpeg: An open source framework for simulation and gradient based parameter estimation in geophysical applications. *Computers & Geosciences*, 85:142–154, 2015.
- [9] Carsten Rücker, Thomas Günther, and Florian M Wagner. pyGIMLi: An open-source library for modelling and inversion in geophysics. *Computers & Geosciences*, 109:106–123, 2017.
- [10] Will J Schroeder, Bill Lorensen, and Ken Martin. *The visualization toolkit: an object-oriented approach to 3D graphics*. Kitware, 2004.

- [11] C. Bane Sullivan and Alexander Kaszynski. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, May 2019. doi: 10.21105/joss.01450. URL <https://doi.org/10.21105/joss.01450>.
- [12] Hunter, John D. Matplotlib: A 2D graphics environment. *Computing in science & engineering*, 9(3):90, 2007. doi: 10.1109/MCSE.2007.55.
- [13] Ramachandran, Prabhu and Varoquaux, Gaël. Mayavi: 3D visualization of scientific data. *Computing in Science & Engineering*, 13(2):40–51, 2011. doi: 10.1109/MCSE.2011.35.
- [14] Turk, Matthew J and Smith, Britton D and Oishi, Jeffrey S and Skory, Stephen and Skillman, Samuel W and Abel, Tom and Norman, Michael L. yt: A multi-code analysis toolkit for astrophysical simulation data. *The Astrophysical Journal Supplement Series*, 192(1):9, 2010. doi: 10.1088/0067-0049/192/1/9.
- [15] David Ascher, Paul F Dubois, Konrad Hinsen, Jim Hugunin, Travis Oliphant, et al. Numerical python, 2001.
- [16] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 277–281. ACM, 2015.
- [17] Sebastian Müller and Lennart Schüler. GeoStat-Framework/GSTools: Reverberating Red, October 2019. URL <https://doi.org/10.5281/zenodo.3468230>.
- [18] Kristen Marberry. Subsurface characterization and liquefaction potential assessment of the spirit lake debris blockage at mount st. helens, washington using multi-method geophysics, 2020.
- [19] C. Bane Sullivan and Whitney J. Trainor-Guitton. PVGeo: an open-source Python package for geoscientific visualization in VTK and ParaView. *Journal of Open Source Software*, 4(38):1451, Jun 2019. doi: 10.21105/joss.01451. URL <https://doi.org/10.21105/joss.01451>.
- [20] Utkarsh Ayachit. The paraview guide: a parallel visualization application. 2015.
- [21] Sebastien Jourdain, Utkarsh Ayachit, and Berk Geveci. Paraviewweb, a web framework for 3d visualization and data processing. In *IADIS international conference on web virtual reality and three-dimensional worlds*, volume 7, page 1, 2010.
- [22] Vtk.js documentation. URL <https://kitware.github.io/vtk-js/docs/>. visited on 2018-08-01.

APPENDIX A

PVGEO: AN OPEN-SOURCE PYTHON PACKAGE FOR GEOSCIENTIFIC VISUALIZATION IN VTK AND PARAVIEW

C. Bane Sullivan and Whitney J. Trainor-Guitton. PVGeo: an open-source Python package for geoscientific visualization in VTK and ParaView. *Journal of Open Source Software*, 4 (38):1451, Jun 2019. doi: 10.21105/joss.01451. URL <https://doi.org/10.21105/joss.01451>

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 with geoscience data and models has been limited. As a software extension package to VTK and ParaView, PVGeo addresses the lack of geoscientific compatibility by creating a framework for geovisualization. PVGeo aims to make the process of importing geoscience data into VTK-based software fluid and straightforward for users while providing a framework for new features that avoids the typical, ambitious programming endeavor of building VTK software plugins. We have developed this code library, PVGeo, to link geoscientific data and models with VTK-based 3D rendering environments like ParaView: an open-source platform built on top of VTK [20]. Since VTK is an established and robust visualization platform, it provides a rich toolbox of features common for visualization and spatial analysis across disciplines [10]. Examples of standard features include volume rendering, glyphing, subsetting, K-Means clustering, volume interpolation, iso-contouring, and Virtual Reality [10, 20]. By linking geoscience to VTK and ParaView, geoscientists can harness all of the native tools within ParaView, and other VTK powered libraries like ParaViewWeb [21], VTK.js [22], and PyVista [11] or extend

that data into new domains like Virtual Reality, as outlined in Figure A.1. PVGeo couples geoscientific information to software libraries at the forefront of scientific visualization, which enables scientists to cost-effectively and reproducibly communicate their findings.

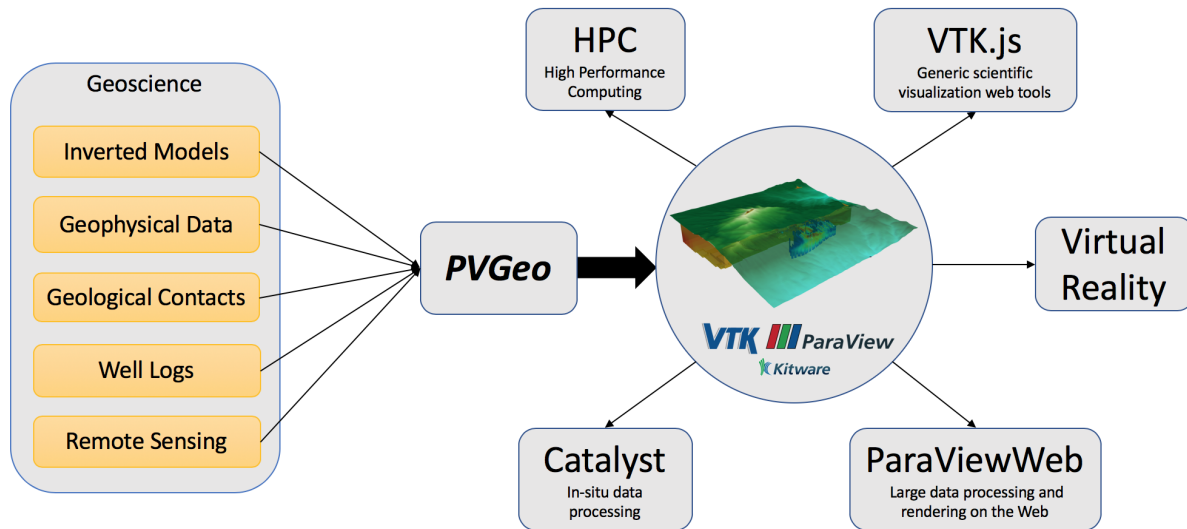


Figure A.1: PVGeo providing a link for geoscience to the VTK and ParaView realm of data visualization.

A.1 Background

The results of geophysical imaging techniques often hold high significance to stakeholders yet the effective perception of those results remains a dynamic challenge. In the geosciences and especially the field of geophysics, researchers often need 3D and 4D (time-varying) visualizations to understand complex spatial and temporal relationships in data which are challenging to capture in 2D visualizations [2]. Better perceptions or new understandings may arise from data when referenced in relation to intuitive features like topography, well locations, survey points, or other known information. Through these spatial relations, geoscientists and stakeholders can directly engage with their data to gain insight and begin to rapidly evaluate data and models either on various 2D planes simultaneously or in a complex

3D environment [2, 6].

Geoscientists often use specific visualization software for different data processing routines which can lead to using several different visualization environments for a single project. Unfortunately, geoscientists are often left without a toolset for visual integration across those different data types, like pairing well locations, resource models, and geological models, which can hinder their ability to interpret the spatial relationships of varying data types.

As authors of this software and geoscientists, we rely on calibrating and integrating our data with all types of subsurface information to further illuminate the value of geophysical imaging techniques. This fosters a need for a visualization package to work seamlessly across data types and formats that extends the functionality of an already robust visualization platform like ParaView [20] or PyVista [11]. This visualization library is the PVGeo Python package; a free and open-source library for integrating geoscientific datasets in a common rendering environment to address various visualization and spatial analysis needs in geoscience. The PVGeo package is powered by VTK [10] and provides plugins for ParaView. As a pure-Python package, PVGeo is interoperable with other Python and VTK-based software like the PyVista Python package [11].

There are various software available for geoscientific visualization; however, these software often handle a few proprietary data formats and are closed-source with licensing fees. Witter & Melosh (2018) provide a comprehensive list and discussion of the various software packages and finds that there are many platforms available for integrated visualizations for limited data types. Having the ability to visually fuse datasets, construct 3D models, or generate horizons within the visualizations can be what separates closed-source software from open-source software [2] - development of PVGeo aims to create an open-source alternative for researchers.

A.2 Mentions

Development for PVGeo is complemented by development for PyVista. PVGeo provides an extension package to PyVista linking data formats and filtering routines common in

geoscientific disciplines to PyVista’s generalized framework for 3D visualization. PVGeo leverages PyVista to make the inputs and outputs of PVGeo algorithms more accessible so that users can create compelling, integrated visualizations of their work in a reproducible workflow.

A.3 Acknowledgements

Funding: Newmont Mining Corporation supported this work, and we thank Marcelo Godoy and Richard Inglis for collaboration on software development for the ParaView platform.

We thank Jacob Grasmick (Colorado School of Mines) for his interest throughout the project’s inception and for providing sample datasets to demonstrate PVGeo’s filtering algorithms.

APPENDIX B

COMPARISON OF VISUALIZATION PROGRAMMING WITH THE VISUALIZATION TOOLKIT (VTK) AND PYVISTA

The Visualization Toolkit (VTK) is an excellent visualization framework, and with Python bindings it should be able to combine the speed of C++ with the rapid prototyping of Python. However, VTK code programmed in Python generally looks the same as its C++ counterpart. The PyVista software seeks to simplify mesh creation and plotting without losing the functionality of the underlying VTK software.

Figure B.1 shows an example 3D mesh of a salt body in the PLY¹⁴ file format that is visualized by VTK Python code in Listing B.5 and by PyVista Python code in Listing B.6.



Figure B.1: An example 3D mesh of a salt body.

¹⁴The PLY polygon file format: <http://paulbourke.net/dataformats/ply/>

```

import vtk

# create reader
reader = vtk.vtkPLYReader()
reader.SetFileName("salt_body.ply")

mapper = vtk.vtkPolyDataMapper()
if vtk.VTK_MAJOR_VERSION <= 5:
    mapper.SetInput(reader.GetOutput())
else:
    mapper.SetInputConnection(reader.GetOutputPort())

# create actor
actor = vtk.vtkActor()
actor.SetMapper(mapper)

# Create a rendering window and renderer
ren = vtk.vtkRenderer()
renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)

# Create a renderwindowinteractor
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)

# Assign actor to the renderer
ren.AddActor(actor)

# Enable user interface interactor
iren.Initialize()
renWin.Render()
iren.Start()

# clean up objects
del iren
del renWin

```

Listing B.5: Example VTK code to produce a 3D visualization from commonly use 3D file format.

```
import pyvista as pv

# read the 3D file
mesh = pv.read("salt_body.ply")

# visualize the mesh object
mesh.plot()
```

Listing B.6: Example PyVista code to produce the same 3D visualization.

APPENDIX C

SUPPLEMENTAL ELECTRONIC FILES

Table C.1 lists the included supplemental electronic files for recreating the figures in this thesis. The data files and Python Jupyter notebooks included here will reproduce the data figures for the FORGE geothermal project and the Mount St. Helens hydrogeophysical project using PyVista. These files are included with the thesis and will be permanently available and maintained in the GitHub repository: <https://github.com/banesullivan/thesis-notebooks>.

Table C.1: List of supplemental electronic files.

File Name	Description
FORGE.omf	Data from the FORGE geothermal research site in the Open Mining Format (OMF) version 1.0 specification.
Kriging.ipynb	Reproducible workflow demonstrating the use of external software for geostatistical model building with the PyVista framework.
MSH.zip	Data from Kristen Marberry's thesis investigating hydrological conditions near Mount St. Helens.
Mt_St_Helens.ipynb	Reproducible workflow demonstrating the synthesis of various geoscientific datasets under the PyVista framework for visual fusion.
environment.yml	The Anaconda Python virtual environment specification file for installing all dependencies needed to run the code included in this appendix.