https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

PyMeshIt: An open-source geological solid modeling and tetrahedral meshing engine integrated in PZero

Waqas Hussain[1], Mauro Cacace[2], Andrea Bistacchi[1], Riccardo Monti[1]
[1] Dipartimento di Scienze dell'Ambiente e della Terra, Università degli Studi di Milano-Bicocca, 20126, Italy
[2] Helmholtz Centre Potsdam (GFZ) German Research Centre for Geosciences, Telegrafenberg, 14473 Potsdam, Germany

Correspondence: Waqas Hussain (w.hussain7@campus.unimib.it)

Abstract

Volumetric mesh generation from surface-based geological models is a fundamental preprocessing step for numerical simulation of subsurface processes including fluid flow, heat transport, and deformation, with applications spanning geothermal energy, groundwater modeling and management, underground facility design, slope stability and hazard modeling, carbon sequestration and hydrocarbon reservoirs. To convert a surface-based geomodel into a volumetric mesh, build a watertight Piecewise Linear Complex (PLC) of the geological model as input to boundary-conforming meshing kernels to guarantee geometry-consistent tetrahedralization. Discretization remains challenging because geological models contain internal intersecting discontinuities, concave boundaries, folded or overturned surfaces, and geological attributes such as stratigraphic and structural identifiers and boundary condition tags that must be preserved during the meshing stage.

We present PyMeshIt, an open-source Python implementation and extension of the C++ MeshIt meshing engine (Cacace and Blöcher 2015), developed to generate watertight PLCs and tetrahedral meshes for complex geological models. PyMeshIt is developed to be used either as a standalone meshing software or as a modeling engine integrated into the open-source PZero geological modeling framework, a platform for 3D structural interpretation and modeling (Bistacchi et al. 2021). The internal architecture of PyMeshIt splits the meshing process into several sequential stages: geometric preprocessing, hull construction, primary surface generation and interpolation, intersection resolution into a consistent PLC, and a final constrained tetrahedralization.

A central technical contribution in PyMeshIt is the preservation and recovery of facet identity through the Python-TetGen interface, enabling exact extraction of fault and boundary triangle sets from the final volumetric mesh that can be input to the later numerical simulation stages (e.g., to be used to impose boundary conditions). To define boundary objects, PyMeshIt combines adaptive alpha-shape extraction for concave but projection-compatible surfaces, a 3D angular gap method for raw folded point clouds where projection-based hull algorithms fail, and a mesh-connectivity-based boundary extraction for pre-triangulated surfaces and already triangulated surfaces imported from PZero, with respect to MeshIt. We tested PyMeshIt on multiple synthetic benchmarks and five real case studies, reporting element counts and mesh quality metrics, and compared the outcomes against the original C++ implementation of MeshIt. Results demonstrate that PyMeshIt extends the original MeshIt workflow from faulted reservoir geometries to more complex folded geological models while maintaining controllable per-surface resolution, PLC consistency, and simulation-ready geological attribution. We additionally discuss the main limitations of the current implementation when dealing with very tight folds in the raw point-cloud pathway, where k-nearest-neighbour graphs used for Isomap may create cross-limb shortcuts; for such cases, pre-triangulated PZero surfaces with topology-derived boundaries currently provide a more robust alternative.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

1 Introduction

In some cases, the final goal of a 3D geological modelling project is the geological model per se, used to illustrate or validate geological concepts and processes (Arienti et al., 2024, 2025), but in other situations the geological model is just a step in a larger workflow aimed at developing geostatistical models for various applications (e.g. mining, reservoirs of geofluids, etc. (Journel and Ch.J, 1978; Deutsch, 2002) or to define input geometries and parameters for numerical simulations in geomechanical, hydraulic and/or geothermal problems (Houlding, 1994; Mallet, 2002; Farmer, 2005). However, to convert the output from a 3D structural model into a simulation-ready mesh remains challenging (Caumon et al., 2004, 2009; Pellerin et al., 2017; Legentil et al., 2022, 2023). The difficulty stems not only from the geometric complexity of the surfaces involved but also from their topology, i.e., the mutual relationships among faults, horizons, and discontinuities that intersect and terminate against one another (Caumon et al., 2009).

Numerical simulators discretize continuous physical processes, such as fluid flow, heat transport, or mechanical deformation, over a computational domain that must provide an unambiguous geometric/topological representation of the model, in which all material regions and bounding surfaces are consistently defined and closed under intersection (Mallet, 2002). This representation is referred to as "watertight" in the sense that the different regions of the model are sealed and do not "communicate" with each other (Mallet, 2002, p.34; Anquez et al., 2017). Watertightness is also a necessary condition to apply robust triangulation and tetrahedralization codes, such as Triangles and TetGen (Shewchuk, 1996; Si, 2015), to generate simulation meshes with properly traced material regions and boundary sets .

These open-source meshing libraries, however, do not solve the major bottleneck in the workflow leading from the geological model to simulation, that is, constructing a consistent piecewise linear complex (PLC): a valid geometric representation of the domain consisting of vertices, edges, and planar facets that describe all internal and external boundaries without gaps, reconcile intersections across multiple surfaces, preserve and propagate boundaries, and verify if each resulting surface is suitable for tetrahedralization (Miller, 1998; Si, 2015). In addition, most surface remeshing stages require a two-dimensional parameterization of each geological surface before constrained triangulation can be applied. This assumption is acceptable for quasi-planar faults and gently dipping horizons, but it breaks down for folded or overturned surfaces where different limbs overlap if based on planar-like projection algorithms (Mallet, 2002, p.109).

Unfortunately, most geological models, which are usually assembled from heterogeneous and sparse data, generally contain features that pose problems both in building a PLC and when meshing or remeshing, such as sharp fault discontinuities or terminating faults, thin layers, concave boundaries, irregular sampling densities, and folded or overturned surfaces, the geometry of which cannot be represented reliably by a single planar projection (Mallet, 2002, p.109; Caumon et al., 2004; Fernández et al., 2004; Dhont et al., 2005). Convex-hull-based boundary construction, for instance, handles non-concave surfaces well but artificially fills in concavities on wavy or irregular structures (Edelsbrunner et al., 1983; Edelsbrunner and Mücke, 1994); alpha-shape extraction addresses concavity but remains projection-dependent, and so cannot resolve the more severe cases of folded or overturned surfaces whose limbs overlap in any planar view (Mallet, 2002, p.109). In such scenarios, meshing failures arise from small topological inconsistencies rather than from the core tetrahedralization itself (Attene et al., 2013; Zehner et al., 2015; Wellmann and Caumon, 2018). As no widely adopted open-source tool automates this path end-to-end, the step is still usually done by hand, exporting surfaces, repairing them in a separate meshing package, and manually reassigning attributes lost in the conversion. This makes the workflow hard to reproduce and automate (Pellerin et al., 2017).

Previous geomodelling-specific workflows, such as the C++ Meshit software package, have already demonstrated how a practical end-to-end workflow can be built around Triangles and TetGen by combining geometric preprocessing, constraint building, quality control, and final constrained Delaunay tetrahedralization to generate high-quality volumetric watertight meshes for complex faulted reservoirs (Cacace and Blöcher, 2015). The Meshit design choice to rely on TetGen and Triangle reflects the well-established advantages of constrained Delaunay tetrahedralization for geological domains, including boundary-conforming properties and guaranteed mesh quality, a choice we adopt and extend in PyMeshIt, as detailed in this contribution.

However, the operational difficulty of deploying a stand-alone C++ meshing pipeline, including its installation, compilation, dependency management, platform support, and integration with modelling/interpretation software, undermines the usability of MeshIt by a broader community (Bangerth and Heister, 2013). For this reason, we have developed PyMeshIt, a Python reimplementation and extension of the MeshIt workflow. The reason for choosing

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Python's scientific ecosystem is that it provides mature, high-performance numerical, interpolation, and spatial analysis and a deployment model that removes the barriers described above, while remaining fully compatible with the already existing algorithms such as the Delaunay/PLC paradigm that were developed in MeshIt.

PyMeshIt is organized around a user-friendly, multi-staged workflow that guides the user from geometric preprocessing and boundary definition through surface reconstruction, intersection resolution, PLC construction, conforming surface remeshing, and final volumetric discretization with material assignment. An additional feature is that PyMeshIt is natively integrated within PZero, an open-source Python platform for 3D modelling and data management (Bistacchi et al., 2021). This integration aims to eliminate the handoff between geological interpretation (surfaces, polylines, geological entities, and attributes) and meshing (PLC constraints and tetrahedralization) by providing a unique, open workflow from data integration and model building to simulation-ready discretization (Si, 2015).

The remainder of this paper is organized as follows. The Methodology describes the open-source software architecture and the staged workflow used to generate watertight meshes, including PLC construction, facet-marker recovery, adaptive boundary extraction, PZero integration, and the folded-surface meshing extensions based on 3D angular-gap boundary detection and Isomap unfolding (Shewchuk, 1996; Si, 2015). The Results section evaluates the workflow on a suite of synthetic and real benchmark cases, including faulted reservoirs, intersecting geological surfaces, raw folded point clouds, pre-triangulated PZero folds, and real complex structural models containing multiple formations. The Discussion interprets these results in relation to the original C++ MeshIt workflow, clarifies the role of each boundary and triangulation pathway, and discusses remaining limitations, particularly those associated with very tight folds where nearest-neighbour geodesic graphs may create cross-limb shortcuts.

2 Software implementation

2.1 Software architecture and dependencies

PyMeshIt is an open-source Python-based application capable of generating high-quality, watertight tetrahedral meshes for geological reservoir modeling. Unlike its predecessor, MeshIt (Cacace and Blöcher 2015), which was implemented in C++, PyMeshIt leverages the dynamic Python ecosystem to ease its integration with modern open-source geomodelling software (e.g., PZero). NumPy and SciPy libraries handle the core operations of PyMeshIt for efficiency (Harris et al., 2020; Virtanen et al., 2020). Triangle, a constrained 2D delaunay triangulator, and TetGen, its 3D tetrahedralization counterpart, are called through dedicated Python wrappers to perform surface triangulation and final volumetric discretization, respectively. PyMeshIt's underlying data model is built on VTK, which provides robust unstructured-mesh data structures, Spatial search and geometric processing algorithms, and a wide variety of file interchangeability; PyVista (Sullivan and Kaszynski, 2019) is used as a Pythonic middle layer over VTK, simplifying both scripting and visualization.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

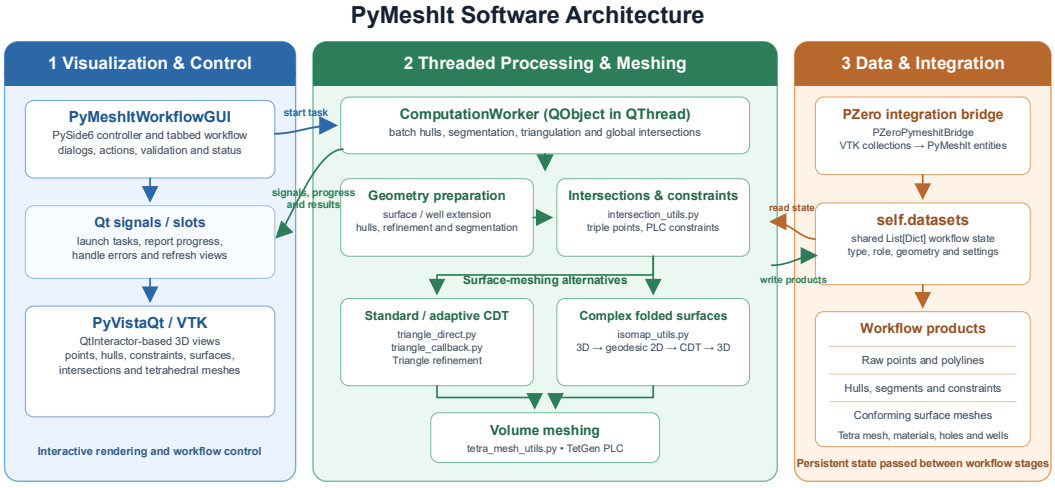


Figure 1. The software architecture of PyMeshIt comprises a Visualization layer, a Threaded Processing and Meshing layer, and a Data & Integration layer, each layer represented by its respective modules

PyMeshIt is organized as a modular Python application composed of four main modules: the graphical user interface and data-management layer (workflow_gui.py), intersection and constraint-processing utilities (intersection_utils.py), surface triangulation routines (triangle_direct.py, Isomap_utils.py), and tetrahedral mesh-generation utilities (tetra_mesh_utils.py) (Figure 1). This separation allows the meshing algorithms to be invoked either through the embedded GUI for interactive model building or programmatically via Python scripts for batch processing and automated workflows.

The GUI implements the workflow as sequential tabs, providing on-the-fly 3D visualization of each processing step through embedded PyVista viewports. This interactive feedback enables rapid identification of geometric issues, such as missing intersections or non-watertight surfaces.

A key implementation difference from the original C++ MeshIt concerns the underlying parallelization strategy. Where MeshIt employed Qt's threading classes, PyMeshIt distributes surface-surface intersection tests across available CPU cores, relying on Python's concurrent processing facilities. For a model with n surfaces, the $n(n-1)/2$ pairwise intersection tests are computed in parallel, reducing preprocessing time for structurally complex models.

PyMeshIt is distributed as either a Python package or a standalone GUI executable for Windows, macOS, and Linux, built with PyInstaller (Hussain, 2026) (https://github.com/waqashussain117/PyMeshit). This packaging avoids user-side compilation and simplifies deployment in teaching and applied modelling environments. The software exports mesh in VTK/VTU format for visualization, NetCDF/Exodus format for finite element solvers (OpenGeoSys, MOOSE), and standard computer graphics and CAD formats (STL, OBJ) for interoperability with other modeling tools.

2.2 Geometric pre-processing and hull generation

PyMeshIt takes as input unprocessed point clouds representing geological horizons and other structural features such as faults. Before triangulation, the spatial extent of each surface must be defined by computing its hull boundary, the boundary polygon that traces the outer edge of the point cloud and delimits the area over which the mesh will be generated (Graham, 1972; Shewchuk, 1996). Defining an accurate hull boundary is a critical preprocessing step, as errors in the boundary propagate directly into the final volumetric mesh (Caumon et al., 2004, 2009; Si, 2015).

Standard convex hull algorithms, such as the Graham Scan (Graham, 1972), provide only limited success for non-convex shapes and complex surface manifolds (Shewchuk, 1996). To address these challenges, PyMeshIt employs a hybrid, topology-aware strategy that selects the hull generation method based on the geometric character of the input surface, as illustrated in Figure 2.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

For quasi-planar surfaces such as faults, planar feature extraction is performed by projecting 3D points onto their optimal 2D plane using Principal Component Analysis (PCA), which identifies the plane of minimum variance through the point set (Pearson, 1901; Hotelling, 1933). A 2D Delaunay triangulation is then computed on this projection, and the resulting 2D mesh is interpolated back onto the 3D space to create its boundary hull (Shewchuk, 1996), known in the interface as Delaunay 2D.

For geometrically complex surfaces such as wavy horizons or erosional features, PyMeshIt relies on an adaptive alpha shape algorithm as a generalization of the convex hull strategy (Edelsbrunner et al., 1983). The alpha parameter (α) is computed from the local point density, specifically the median edge length of the point set, so that the boundary adapts to the resolution of the input data. This allows the hull to trace concave bays and inlets that a convex hull would artificially bridge, recovering the true geological extent. The difference in behavior between the two approaches is illustrated in Figure 2.

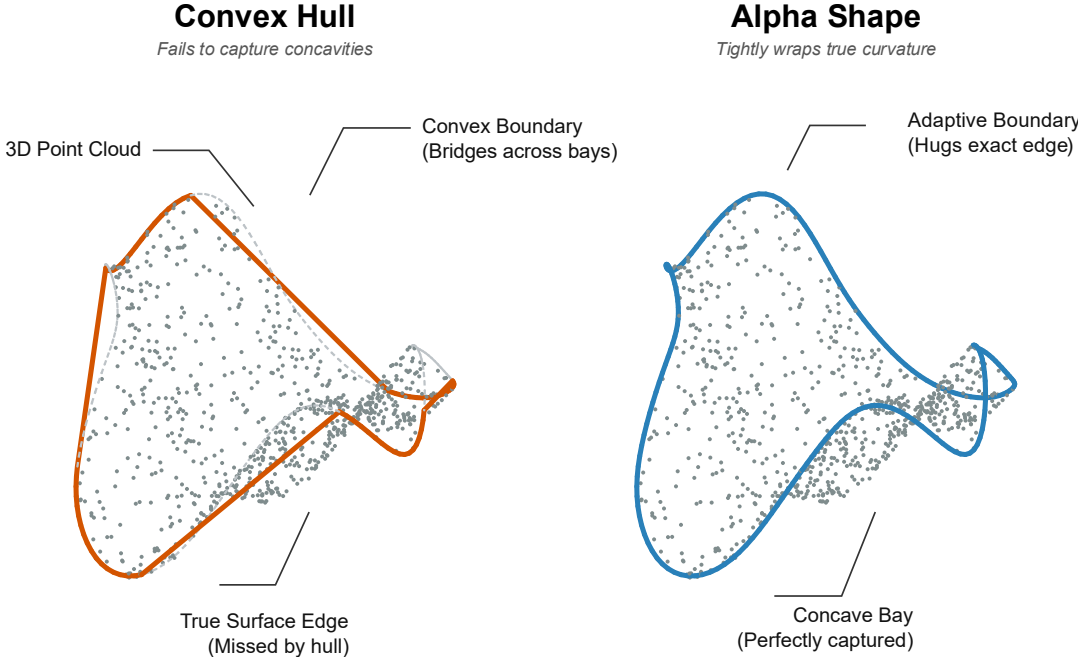


Figure 2. Shows the difference between the Convex Hull and the Alpha Shape. Convex Hull (Right) fails to capture the concavities of the true surface edge. Alpha Shape (Left) captures the curvature of the hulls and properly maps the surface edge

2.3 Preliminary surface generation and interpolation

Having computed the hull boundary for each geological surface of interest, the internal topology of each surface geometry is reconstructed in the "Preliminary Triangulation" stage, which serves as a basis for the subsequent intersection detection algorithm. The workflow generates triangulated surfaces in three steps: First, the 3D points from the hull generation are projected onto a PCA. Second, a 2D Delaunay triangulation is computed on this projection using the Triangle library (Shewchuk 1996). Third, the 2D mesh nodes are assigned elevation values by interpolating the out-of-plane coordinates from the original 3D point cloud onto the PCA-projected plane perpendicularly to the best-fit plane using either Inverse Distance Weighting (IDW), for legacy compatibility as in MeshIt, or Thin Plate Spline (TPS), before being projected back to the final 3D space (Figure 3) (Shepard, 1968; Duchon, 1977).

Both IDW and TPS serve the same geometric purpose: given a set of 2D projected points with known out-of-plane coordinates inherited from the original 3D point cloud, they estimate the out-of-plane coordinate at any new 2D location introduced by the triangulator, such as Steiner points added during constrained Delaunay refinement.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Inverse Distance Weighting (IDW) estimates the out-of-plane coordinate at each new point as a weighted average of the values at neighbouring data points, with the weight assigned to each neighbour decreasing with its distance to the target sampling point (Shepard, 1968). This makes IDW computationally efficient and robust for gently varying surfaces. Still, it has a fundamental limitation for geologically complex surfaces since the weighted-average mechanism smooths sharp gradients, attenuating displacement amplitudes near fault tips and fold hinges where the out-of-plane coordinate changes rapidly over short distances. Still, it has a limitation for geologically complex surfaces since the weighted-average mechanism attenuates sharp gradients more strongly than smoother alternatives, reducing displacement amplitudes near fault tips and fold hinges where the out-of-plane coordinate changes rapidly over short distances.

Thin Plate Spline (TPS) interpolation instead fits a globally smooth function that passes exactly through all data points while minimizing the integrated second derivative of the surface, a quantity analogous to the bending energy of a thin elastic plate (Duchon, 1977; Bookstein, 2002). In practice, this means TPS produces the smoothest possible surface that is consistent with all input data points, without introducing localized averaging artifacts of IDW. For surfaces with sharp fault offsets, fold hinges, or strongly varying relief, TPS preserves the full amplitude of structural features and ensures that newly introduced mesh nodes follow the true geometry of the surface rather than a smoothed approximation of it. For these reasons, TPS is the default interpolation kernel in PyMeshIt for all surfaces requiring constrained remeshing; IDW is retained as a computationally lighter alternative for very dense point clouds, such as those derived from 3D seismic interpretation, where solving the TPS system becomes computationally expensive. The quantitative impact of this choice on mesh geometric fidelity is evaluated in Section 4.3.

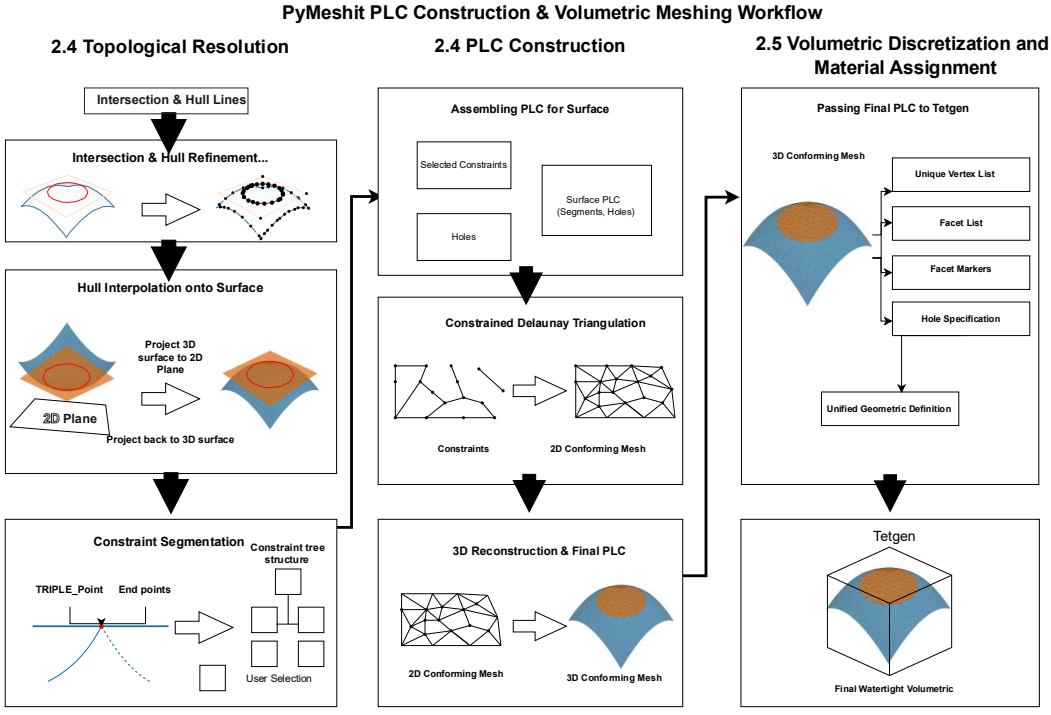


Figure 3. Schematic overview of the PyMeshIt PLC construction and volumetric meshing workflow. The workflow is organized into three consecutive stages. (Left) Topological resolution: detected surface intersections and convex hull lines are iteratively refined, interpolated onto the target surfaces via 2D projection, and segmented into a constraint tree structure with explicit triple points and endpoints. User then select the proper constraint for remeshing using the tree selection (Center) PLC construction for surface meshing: selected constraints and hole definitions are assembled into a surface PLC, which is triangulated using constrained Delaunay triangulation in 2D and subsequently reconstructed into a 3D conforming surface mesh forming the final PLC. (Right) Volumetric discretization and material assignment: the finalized PLC is passed to TetGen, where vertices, facets,

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

markers, and hole specifications are combined into a unified geometric definition to generate a watertight, constraint-conforming tetrahedral volume mesh.

2.4 Topological resolution: generating the PLC

The core of the meshing kernel is the construction of a watertight PLC (Si, 2015). This is achieved by determining the internal intersections between all input surfaces and remeshing them to preserve all computed intersections in a set of intersection polylines with a specified mesh resolution.

Using the "Preliminary Triangulations" as input, surface-surface and surface-polyline intersections are detected via an Octree-accelerated triangle-triangle test (Figure 4) (Möller, 1997). This produces a set of 3D intersection polylines, geologically representing fault and stratigraphic cutoff lines. To preserve the topological consistency, PyMeshIt identifies special geometric entities during preprocessing. These include: "Triple Points", which are points where three or more intersection lines meet each other; "Corner Points", which are hull vertices, whose local angle, computed from two adjacent hull segments, is smaller than 135°; and "End Points" that are the terminal vertices of open intersection polylines. For circular closed loops, no end point is defined, and the closure is marked separately (Cacace and Blöcher, 2015). These points are the basic information required to resample the final geometry of the input surface and are considered as fixed vertices to prevent numerical drift (Mallet, 2002, p.341)

All intersection polylines are then refined (resampled) through a user-defined refinement length to determine the mesh density. They are segmented into proper selections for constraint selection using Triple Points, Common Convex Hull points, and Corner Points as hard geometric constraints for the successive Delaunay Boundary conforming triangulation stage of all the surfaces, producing conforming meshes that respect all computed intersection polylines and create a watertight PLC by assemblage (Chew, 1987; Si, 2015).

Surface triangulation in PyMeshIt is performed using Triangle's constrained Delaunay refinement algorithm (Shewchuk, 1996). For each geological surface, the user specifies a target refinement length, L_{target}, which defines the characteristic discretization scale used during remeshing. In the current implementation, this length is converted into the maximum allowable triangle area according to Eq. (1):

$$A_{\max} = 1.5L_{\text{target}}^2 \quad (1)$$

where $A_{\max}$ is the maximum triangle area supplied to Triangle in Eq. 1. The coefficient 1.5 is an empirical scaling factor used by PyMeshIt to translate the user-defined refinement length into an area constraint. Consequently, L_{target} should be interpreted as a characteristic refinement parameter rather than as the exact edge length of every generated triangle.

The triangle quality option is additionally used to impose a user-defined minimum interior angle during surface remeshing. Together, the area and angle constraints generate a constrained surface triangulation that preserves intersection polylines, corner points, triple points, and convex-hull boundaries. This approach differs from the spatially varying, gradient-controlled refinement strategy used in the original C++ MeshIt implementation, where the prescribed element size increases progressively from fine discretization near constraint features to coarser discretization within surface interiors (Cacace and Blöcher, 2015). The resulting differences in mesh density and element count are discussed in Section 4.1.

During development, we also tested Triangle's optional *triunsuitable* callback to enforce locally adaptive surface refinement. This callback allows a user-defined function to evaluate candidate triangles during refinement and request further subdivision according to local criteria such as distance to an intersection constraint, surface curvature, or local point density (Cacace and Blöcher, 2015). In principle, this mechanism can reproduce behaviour closer to the MeshIt spatially adaptive refinement. However, applying the callback globally to all geological surfaces produced overly dense triangulations near constraint boundaries and introduced tolerance-sensitive inconsistencies during subsequent PLC assembly and TetGen tetrahedralization. In particular, small coordinate discrepancies between Triangle-generated vertices and the snapped intersection or hull constraint points degraded facet conformity in some test cases. PyMeshIt therefore uses the formula-based area constraint for two-dimensional geological surfaces and restricts locally adaptive refinement to simpler one-dimensional constraint features, such as well trajectories, where the geometry is less ambiguous, and the snapping problem does not arise. The quantitative effect of this design choice is evaluated in Section 4.1

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

A topologically distinct case arises when a fault surface terminates laterally within the model volume without reaching any bounding surface, with a finite fault geometry (Figure 5 (a)). In this configuration, the intersection-polylines between the fault and the host geological surfaces are open curves: their terminal vertices are free endpoints that do not connect to any hull boundary or triple point. PyMeshIt handles this case through an explicit classification and snapping procedure. After intersection refinement, the *refine_intersection_line_by_length* is called, which is based on the C++ RefineByLength function of MeshIt to test whether the first and last vertices of each refined polyline coincide within a geometric tolerance. If the polyline is open, its terminal vertices are assigned the point types START_POINT and END_POINT; if it forms a closed loop, the final vertex receives a LOOP_END tag instead (Figure 5 (b)). The END_POINT vertices, which correspond geometrically to the fault tip line, are then processed by the align_intersection_endpoints_to_hull function, which projects each free tip onto the nearest edge segment of the hull of the intersected surface and inserts it as a new COMMON_INTERSECTION_CONVEXHULL_POINT, using scale-adaptive snapping tolerances derived from the hull diagonal (Figure 5 (c)). This anchors the tip of the fault polyline to the bounding surface and ensures that the PLC contains no topological gaps or dangling edges at the fault termination. The open intersection polyline, with its tip now snapped to the hull, is passed to the constrained Delaunay triangulation stage as an interior constraint rather than a closed boundary loop. Triangle triangulates the surface on both sides of the fault conformally up to the tip point, and TetGen subsequently tetrahedralises the full volume with the fault embedded as a two-sided internal facet that simply ends at the tip line (Figure 5 (d)).

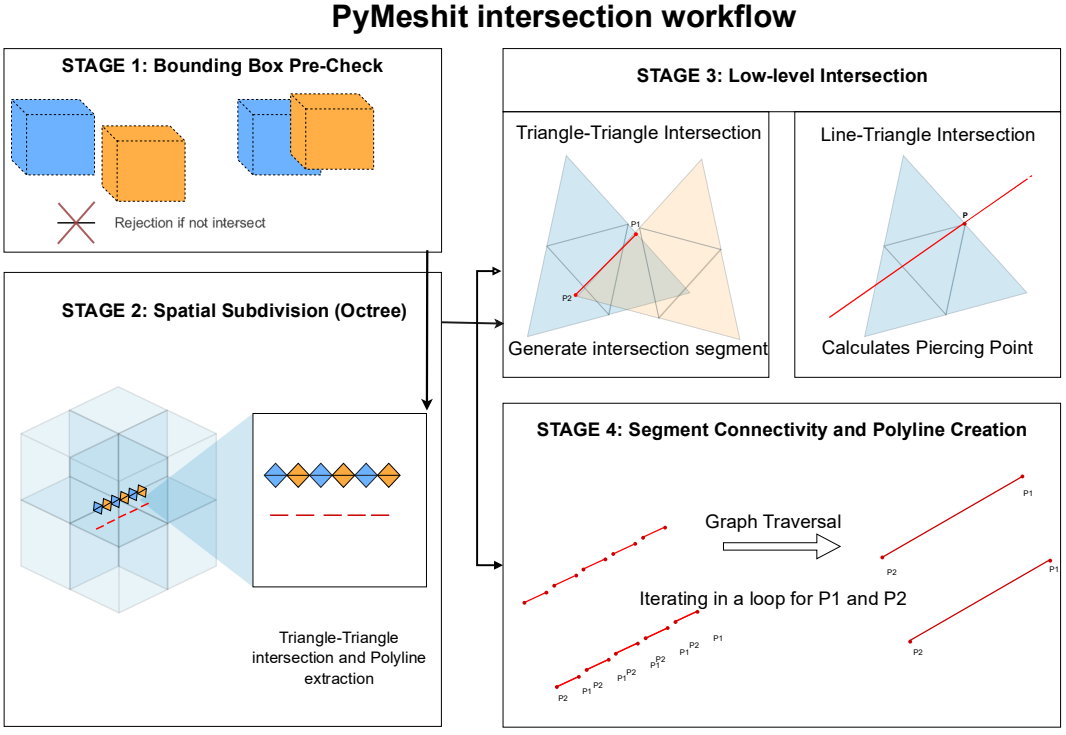


Figure 4. PyMeshIt inner intersection workflow: (Stage 1) Input a 3D Bounding box and pre-check whether two surfaces intersect. (Stage 2) Spatial subdivision through Octree for Triangle-Triangle intersection and line segment extraction. (Stage 3) Low-level intersection showing the Triangle-Triangle intersection (Surface-Surface), Line-Triangle Intersection (Well-Surface), and segment extraction. (Stage 4) The process of extracting segments through iteration and connecting segments to create a single polyline representing the intersection line.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

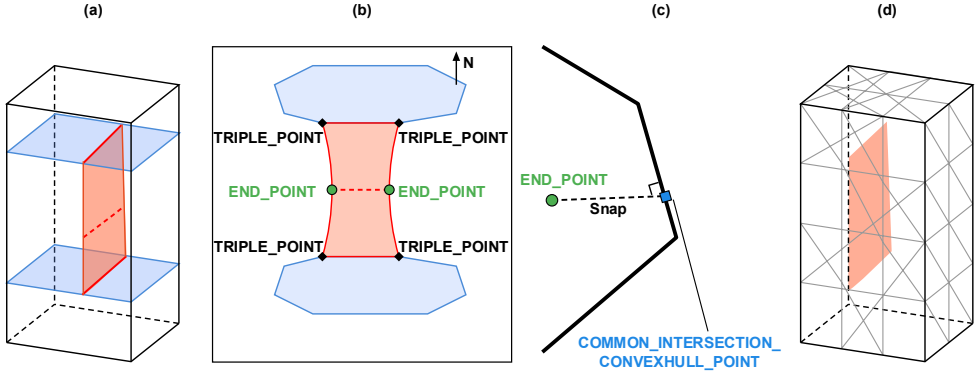


Figure 5. Schematic illustration of a fault handling workflow in PyMeshIt. (a) Input geometry showing a fault surface terminating literally within the model volume (b). Detected intersection polylines with classified END_POINT and TRIPLE_POINT vertices. (c) Orthogonal projection of the free fault tip onto the nearest hull edge. (d) Final Tetrahedral mesh showing the embedded fault surface and continuous manifold transition beyond the fault tip line.

2.5 Volumetric discretization and material assignment

The final watertight PLC is passed to TetGen for Constrained Delaunay Tetrahedralization (Chew, 1987; Si, 2015). TetGen is configured to enforce PLC conformity, control element quality through a maximum radius-edge ratio of 1.2 (Si, 2015), and suppress the insertion of additional vertices on the PLC boundary facets, preserving the geometric fidelity of the input surfaces throughout the tetrahedralization. A limitation of the standard Python wrapper for TetGen is that it does not provide a mechanism to pass surface identifier IDs (e.g., distinguishing a "Fault" face from a "Unit" face) into the PLC before tetrahedralization, nor to retrieve the boundary face connectivity from the output mesh (Si, 2015). To address this, PyMeshIt extends the wrapper to accept per-facet integer markers on the input stage, which are passed directly to TetGen's internal facet marker list. Each PLC facet is assigned a unique identifier before tetrahedralization; fault surfaces receive a dedicated marker distinct from those assigned to stratigraphic horizons and domain boundaries. At the output stage, the extended interface exposes the boundary triangle connectivity and the corresponding face markers from TetGen output, allowing PyMeshIt to select the exact set of triangulated faces associated with each geological surface from the final volumetric mesh (Figure 3). This extension was contributed to the open-source TetGen Python wrapper as Pull Request #98 (https://github.com/pyvista/TetGen/pull/98), merged into TetGen v0.7.0 on PyPi, making it available to the broader scientific community.

Volumetric regions corresponding to geological units are identified through a seed-based material assignment strategy. A seed point is placed within each closed compartment of the PLC corresponding to the individual stratigraphic units or fault blocks, and TetGen propagates a unique material identifier to all tetrahedra within the enclosed volume.

Where provided, well trajectories are incorporated as one-dimensional edge constraints within the PLC, ensuring that well nodes coincide with mesh nodes in the final tetrahedral mesh and preserving fracture-well connectivity for coupled simulation (Adams and Bischof, 1994; Cacace and Blöcher, 2015).

2.6 Mesh quality assessment

After tetrahedralization, PyMeshIt evaluates the generated volumetric mesh using element-count and element-quality statistics. This assessment is necessary because a geometrically conforming mesh is not necessarily suitable for numerical simulation. Poorly shaped tetrahedra can reduce the accuracy, numerical stability, and convergence of finite-element and finite-volume solvers, particularly in geological models containing intersecting faults, narrow compartments, or strongly curved surfaces(Freitag and Ollivier-Gooch, 1997; Knupp, 2001). The principal quality measure reported by PyMeshIt is the tetrahedral radius-edge ratio (Shewchuk, 1996; Miller, 1998; Si, 2015), defined in the following Eq. (2) as:

$$qRE = \frac{R}{L_{min}} \tag{2}$$

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

where R is the radius of the circumsphere of the tetrahedron, and L_{min} is its shortest edge length, as in Eq. 3:

$$q_{\mathrm{RE,min}} = \frac{\sqrt{6}}{4} \approx 0.612 \quad (3)$$

which represents the theoretical minimum for this definition. Increasing values indicate progressively poorer tetrahedral shapes, including elongated, flattened, or sliver-like elements. In the current PyMeshIt implementation, TetGen is run with a radius–edge threshold of $q=1.2$, so tetrahedra exceeding this value are targeted for refinement. This threshold should be distinguished from the theoretical optimum of approximately 0.612. The radius–edge ratio therefore provides a compact, dimensionless measure of tetrahedral quality and permits comparison with TetGen-based meshing workflows and the criterion used in the original MeshIt implementation (Crippen and Havel, 1988; Shewchuk, 1996; Cacace and Blöcher, 2015; Si, 2015). For each mesh, PyMeshIt computes a standard set of quality statistics, element and vertex counts, and the radius-edge ratio directly from the TetGen output using NumPy, and displays them in the interface. These statistics form the basis of the quantitative comparison presented in Section 3.

Because PyMeshIt preserves geological constraints during tetrahedralization, locally poor elements may still occur near intersections, fault tips, fold hinges, or other strongly constrained geometric features. This behaviour is consistent with the original MeshIt workflow, where the Delaunay quality criterion is partially relaxed near imposed geological constraints to preserve the input geometry. Consequently, the reported radius-edge statistics should be interpreted together with geological conformity, watertightness, material-region preservation, and boundary/fault marker recovery, rather than as a standalone measure of mesh validity.

The reported average and maximum quality values should be interpreted in relation to the selected RefineByLength parameter (Table 1). Higher radius–edge values do not, by themselves, indicate that a mesh is unsuitable for numerical simulation; rather, they describe the element shapes generated at the specified refinement scale while preserving the imposed geological constraints. The same meshing parameters were used for the synthetic benchmark cases to evaluate workflow consistency rather than to optimize each mesh individually. Consequently, the reported values represent the outcome of those fixed test conditions and may change substantially when a smaller RefineByLength is applied, producing a finer mesh with a different element-quality distribution.

2.7 Meshing geometrically complex folded surfaces

The workflow described in Sections 2.2 to 2.5 assumes that each geological surface can be represented by projecting its three-dimensional point cloud onto a planar surface. This assumption holds well for gently dipping faults and sub-horizontal stratigraphic horizons, which are the dominant structural elements in relatively simple 3D models (Caumon et al., 2009; Wellmann and Caumon, 2018). It breaks down, however, for strongly folded surfaces, particularly recumbent and refolded folds (Ramsay, 1967). For these geometries, projecting the point cloud onto any plane maps points from different limbs into the same 2D region; the triangulator therefore cannot distinguish between which points belong to which limb, and may generate triangles that connect the two limbs across the hinge. This kind of projection-related ambiguity in folded geometries is a well-documented limitation of explicit surface modelling and planar parameterization methods (Lévy et al., 2002; Floater and Hormann, 2005; Sheffer et al., 2006). To handle these cases, PyMeshit applies two extensions in a sequence: (i) a three-dimensional boundary detection method for extracting the hull of a folded surface without projecting it onto a plane; then (ii) a geodesic unfolding method that allows the interior of the surface to be triangulated consistently.

2.7.1 Extracting the Boundary of a Folded Surface: The 3D Angular Gap Method

Computing the boundary polygon of a surface - the hull boundary- is the first step in the PyMeshIt workflow (Section 2.2). For a folded surface, PyMeshIt provides a hull detection method that operates entirely in three dimensions, without any projection step and solving related issues (Amenta et al., 1998; Gumhold et al., 2001; Ni et al., 2016).

The method is based on a simple geometric observation: a point that lies on the edge of a surface has neighbours on only one side of it, while a point in the interior of a surface has neighbours distributed all the way around it. To detect this difference, a local tangent frame is estimated by applying PCA to the coordinates of its k-nearest neighbours, yielding a local normal direction and two tangent axes spanning the local surface patch (Figure 6(b)) (Cover and Hart, 1967; Hoppe et al., 1992; Tenenbaum et al., 2000; Belkin and Niyogi, 2003). In simple words, for each point, it identifies its nearest neighbours on the local surface tangent plane, and measures the largest empty angular sector (the largest angular gap) in the circular distribution of neighbour directions. A point in the interior of a well-sampled

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

surface will have neighbours spread approximately evenly around it in all directions, leaving no large gap. A point on the boundary of the surface will have no neighbours on the outward side, leaving a gap of at least 90° in the angular distribution. Points whose largest angular gap exceeds this threshold are classified as boundary points. These boundary points are then connected into an ordered closed loop that defines the hull of the surface in 3D (Gumhold et al., 2001; Ni et al., 2016). Because this procedure never requires projecting the surface onto a plane, it remains geometrically valid, but in extreme cases where the points are too close to each other, it might fail.

2.7.2 Triangulating the interior of a folded surface: Isomap geodesic unfolding

Once the boundary of a folded surface has been established, its interior must be triangulated. Again, projecting the interior points onto a flat plane fails for the same reason as before: the fold limbs overlap. PyMeshIt handles this through a geodesic unfolding approach based on the Isomap algorithm (Tenenbaum et al., 2000).

The central idea is to measure distance between the points not by means of their Euclidean distances, but along the surface manifold itself (geodesic distance). Consider two points on opposite limbs of a tight fold: they may be physically close to each other in space, but to travel from one to the other while staying on the surface, one must traverse the entire fold hinge. Therefore the geodesic distance correctly captures the fact that the two points are geometrically far apart, even if they are spatially close (Roweis and Saul, 2000; Tenenbaum et al., 2000). Isomap estimates geodesic distances by first connecting each point to its nearest neighbours to form a network, and then computing the shortest path through this network between every pair of points using Dijkstra's algorithm (Dijkstra, 1959; Tenenbaum et al., 2000). These geodesic distances are then used to construct a flat two-dimensional map of the surface, a process called multidimensional scaling, in which points that are far apart along the surface are placed far apart in the map, regardless of how close they are in the three-dimensional space (Borg and Groenen, 2005). When this map is computed correctly, the two limbs of a fold that overlap in any flat projection are separated into distinct regions of the map, and a standard constrained Delaunay triangulation can be performed on this unfolded representation without generating cross-limb connections. This method has a limitation that, in very tightly sampled folds relative to the point density, the nearest-neighbour graph may include edges that cross hinges, locally corrupting the estimated geodesic distances (Section 4.7).

The practical workflow in PyMeshIt proceeds as follows. First, Isomap is applied to the full point cloud surface to obtain a two-dimensional embedding that approximately preserves geodesic distances along the folded surfaces. Second, the boundary points, intersection polyline nodes, triple points, and corner points are located in the unfolded 2D map using nearest-neighbour lookup (Figure 6(a)). Third, a constrained Delaunay triangulation is performed in this 2D map with all geometric constraints enforced (Shewchuk, 1996; Si, 2015). Fourth, additional mesh nodes introduced by the triangulator, i.e., the Steiner points, are placed back in the three-dimensional space. Original boundary and constraint nodes are returned to their exact original 3D coordinates. Steiner nodes are assigned 3D positions by TPS interpolation (Duchon, 1977; Bookstein, 2002); which fits the smoothest possible surface through the known 3D points (Figure 6(a)); this prevents the localized roughness artefacts that can occur with simpler interpolation methods (e.g., IDW) near fold hinges. Finally, triangles are reported for user inspection rather than being automatically modified, allowing the affected regions to be reviewed and, where necessary, remeshed after adjustment of the input geometry or meshing parameters.

To avoid applying Isomap unnecessarily to surfaces that are already planar enough for standard projection, PyMeshIt automatically assesses each surface before triangulation. The planarity of a surface is measured from the spread of its point cloud using PCA: a surface whose points are confined to a thin slab has a low planarity ratio and is routed to standard PCA projection, and if the points of the surface extend significantly in all three dimensions, then it is routed to Isomap. Extremely tight or thin isoclinal folds may represent an edge case because their point clouds can also satisfy the planarity criterion despite their folded geometry; this limitation is discussed in Section 4.7.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

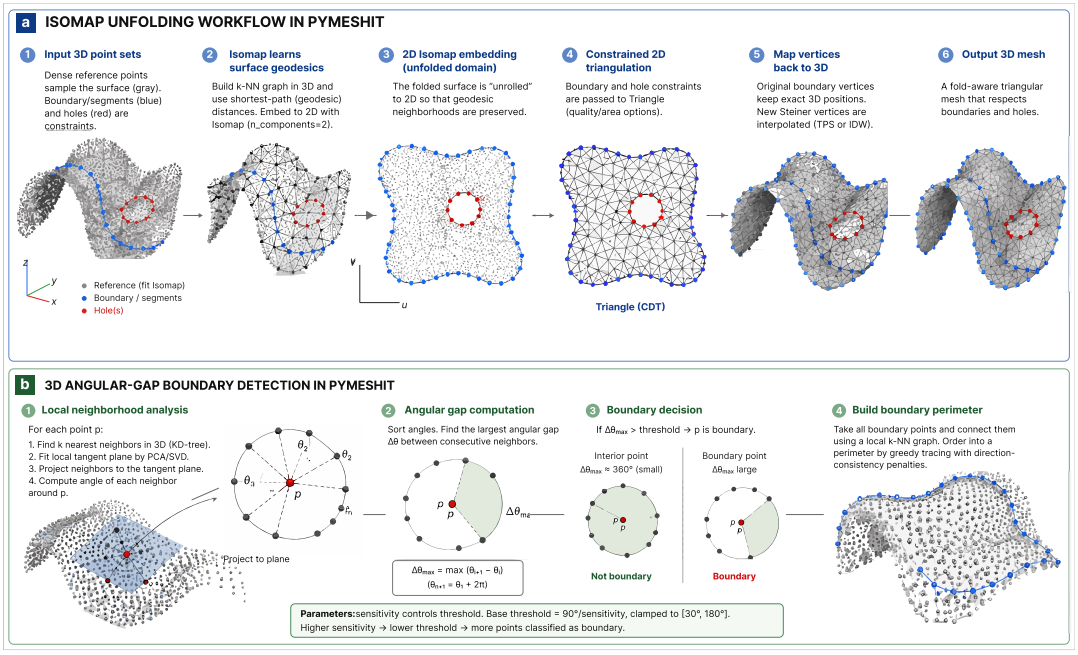


Figure 6. Overview of the PyMeshIt workflows for (a) Isomap-based surface unfolding and constrained surface meshing, and (b) angular-gap–based boundary detection for automatic extraction of surface perimeters from irregular 3D point clouds. Abbreviations: k-NN, k-nearest neighbours; PCA, principal component analysis; SVD, singular value decomposition; CDT, constrained Delaunay triangulation; TPS, Thin Plate Spline.Working with Pre-triangulated Surfaces from PZero

When a folded surface has already been triangulated in PZero (see also next paragraph), a different and more direct approach is available. Instead of computing the hull from the point cloud, PyMeshIt uses the PZero bridge boundary extraction method (get_clean_boundary) to extract the exact topological boundary of the existing surface directly from its mesh connectivity, as described in Section 2.7.1. This boundary is then passed to the Isomap triangulation pipeline, which remeshes the surface interior at a new resolution or with new intersection constraints while preserving the exact original boundary shape. This combination is the recommended workflow for surfaces in the PZero-PyMeshIt integration since PZero already provides a topologically reliable boundary from a geological surface, and Isomap provides a geometrically correct interior triangulation that respects the fold geometry.

2.8 PZero Integration

PyMeshIt works with its own standalone GUI or integrated, as an embedded dockable panel, within the GUI of the PZero geological modelling platform (Bistacchi et al., 2021), enabling an end-to-end workflow from geological interpretation to volumetric mesh generation without leaving the PZero environment. The integration is implemented through two dedicated components (Figure 7): a bridge module (pzero_bridge.py) and a launcher embedded in the PZero project window (project_window.py).

The bridge module exposes the Geological entities stored in the PZero project, including geological surfaces, faults, wells, digital outcrop models, and model boundaries, to the PyMeshIt GUI workflow. Each entity is represented by a typed record containing its name, topological type, geological role, scenario label, and point count, allowing the imported data to be sorted and filtered within the PyMeshIt interface (Figure 7). During import, the bridge maps PZero entities according to their geometric dimensionality and role: surface entities are transferred to the 2D surface-meshing workflow, whereas polyline entities, such as well trajectories, are retained as 1D constraints. This distinction is required because surfaces and 1D constraints undergo different subsequent meshing operations in PyMeshIt.

When a conventional box-shaped boundary defined in the PZero boundary collection is imported, the bridge exposes its six faces individually as bottom, top, front, back, left, and right boundary surfaces. These faces can then be selected

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

independently by the user and can be extended outward to ensure intersection for the intersection stage. This six-face decomposition is therefore an important convenience for the standard PZero boundary representation rather than a restriction of the PyMeshIt model domain. More general boundary configurations, including combinations of lateral boundary surfaces with independently defined top and bottom surfaces, can also be used.

A key advantage of the PZero bridge over the standard point-cloud import workflow of the standalone PyMeshIt GUI is its treatment of already triangulated geological surfaces. When such a surface is imported from PZero in its native form, the bridge retrieves both the existing triangulation and its topological boundary directly from the mesh connectivity. As described in Section 2.8, the resulting ordered boundary polyline is passed directly to the remeshing workflow as the prescribed surface boundary, thereby bypassing boundary reconstruction from the point cloud. This preserves the geometric outline of the original geological surface, including concave and irregular boundaries or erosional truncations that could otherwise be altered by hull-reconstruction algorithms. The original triangulation is retained as reference geometry, while the extracted boundary connectivity is subsequently represented as explicit constrained segments during constrained Delaunay triangulation. PyMeshIt can therefore generate a new interior triangulation at a different resolution and incorporate new intersection constraints while preserving the imported boundary geometry. If a valid boundary cannot be extracted, this may indicate a topological problem in the source surface; the current implementation retains a convex-hull fallback as a recovery mechanism, although the resulting boundary should be verified by the user. Alternatively, when the user selects the option to import the surface as points, the original triangulation and its topological boundary are not used, and its vertices are treated as an unstructured point cloud to which the complete boundary-generation workflow described in Section 2.2 is applied.

The launcher is invoked from the PZero main menu through a dedicated action and opens the PyMeshIt interface as a dockable panel within the PZero project window. The integrated interface supports bidirectional transfer of geological entities between PZero and PyMeshIt. The resulting triangulated and conforming geological surfaces, including horizons, faults, and model-boundary surfaces, together with the final volumetric tetrahedral mesh, can be exported back into PZero, closing the workflow between geological modelling and mesh generation for forward simulation.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

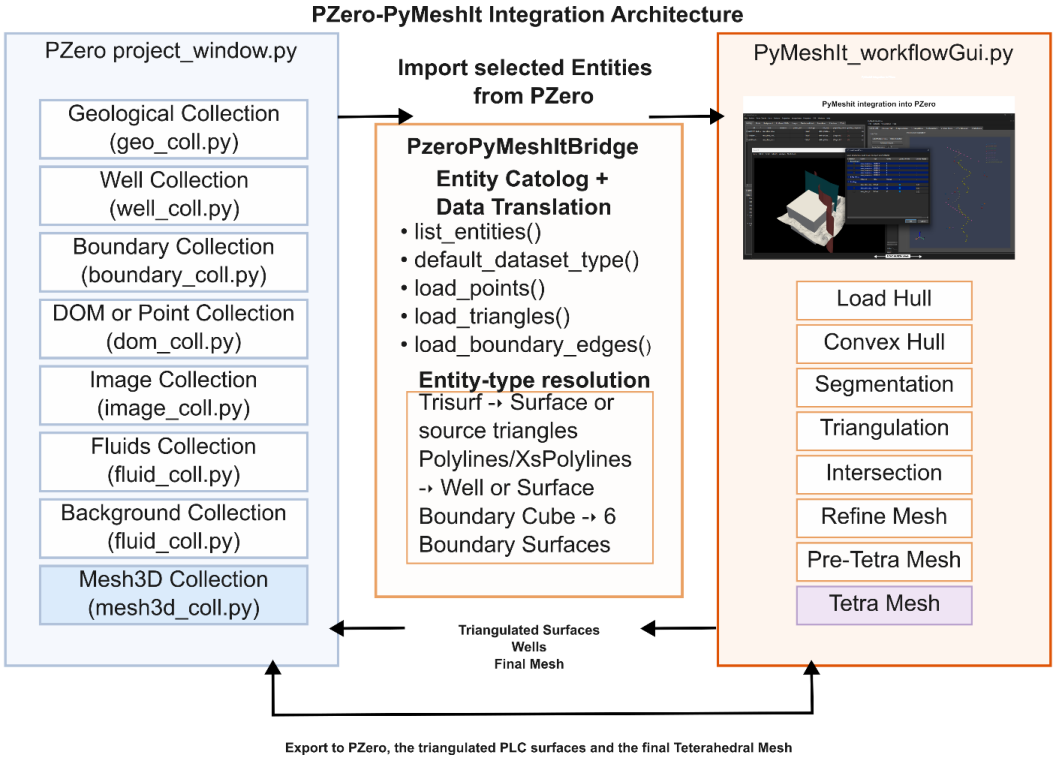


Figure 7. A schematic diagram of PZero-PyMeshIt integration software architecture. The left side shows PZero project_window.py and its collections, which are imported as entities to PyMeshIt_workflowGUI.py on the right through the PZeroPyMeshItBridge. The diagram shows the transfer of data, vice versa, for both software.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

3 Applications and benchmarks

In this chapter, we present several applications and benchmarks, starting from simple ones and progressing to more challenging geological structures. We will first present simple synthetic datasets - generated with PZero – that can be meshed with standard constrained Delaunay triangulation; then we will present benchmarks from the library used in MeshIt, and finally we will move to more complex cases, including real-world models and synthetic ones aimed at the problem of complex folded surfaces. Additionally, two benchmarks from the original C++ implementation were tested to ensure compatibility across both synthetic and real-world applications (Blöcher, 2020).

3.1 Standard CDT Benchmarks: PZero–PyMeshIt Integration

This subsection presents results of the PZero-PyMeshIt integration. Unlike the standalone version, these examples were generated within PZero using its native interpretation and modelling tools, and then remeshed through PyMeshIt. The goal is to demonstrate that (i) PyMeshIt works for simple but typical geological models and that (ii) it successfully integrates in the PZero modelling platform.

3.1.1 Synthetic model 1

The synthetic benchmark of the test PZero project includes four distinct stratigraphic units (Units 1, 2, 3, and 4) characterized by an anticlinal fold (Figure 8). A single high-angle fault plane crosscuts all surfaces, creating a non-manifold topological intersection that must be resolved in the final 3D mesh. To define the volumetric domain for the PLC generation, the model is enclosed within a bounding box that limits its extent. For this model, the hull choice was a Delaunay 2D-based extraction method as described in Section 2.2, and the projection method was TPS, which was used for both Triangulation and remeshing. These relatively simple algorithms are considered sufficient to deal with the topology and geometry of the problem.

The tetrahedral mesh comprises 7,090 elements and 1,542 vertices (Table 1). The mean radius-edge ratio of 0.855 suggests a predominantly isotropic and uniform mesh configuration. This is corroborated by the seamless reconstruction of surface manifolds, thereby confirming how TPS interpolation, along with length-based refinement procedures, is effectively implemented on the input point cloud to reduce surface curvature artifacts.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

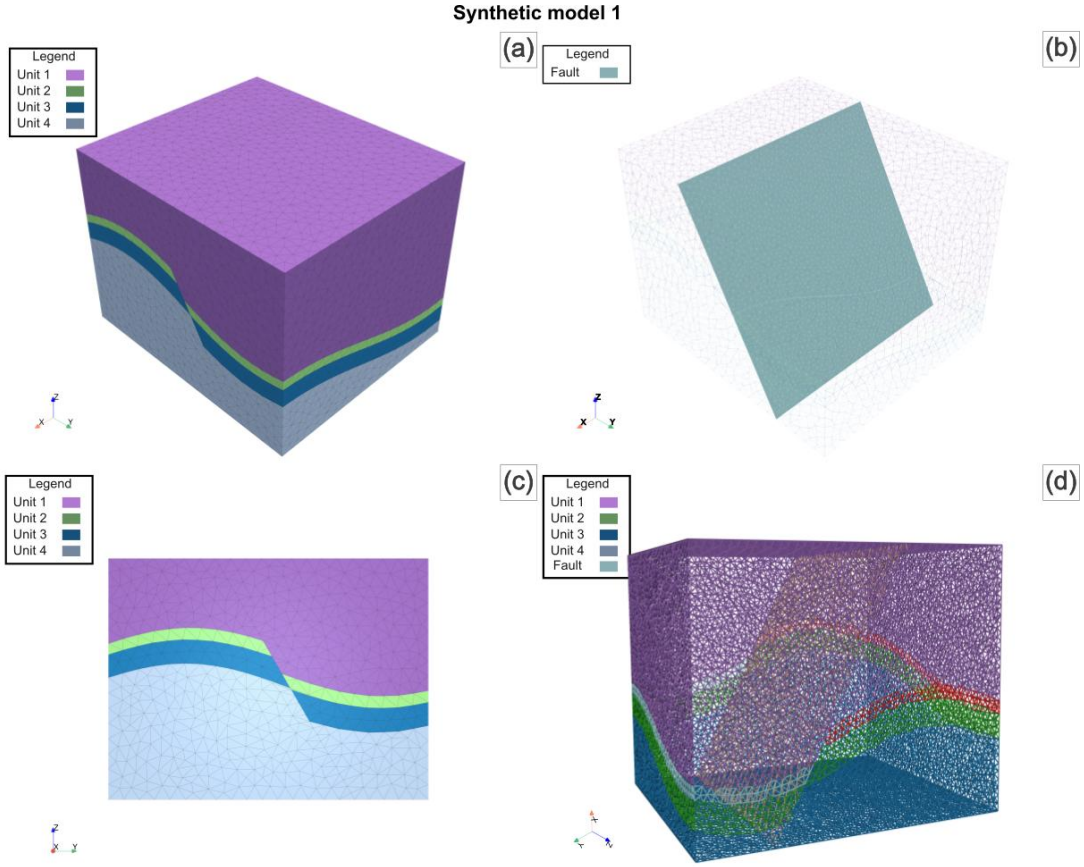


Figure 8. Volumetric discretization of Synthetic Model 1. (a) External surface mesh of the domain, illustrating the watertight boundary conformability. (b) Wireframe visualization of the internal volume, revealing the precise preservation of the internal fault plane. (c) Side view (cross-section) highlighting the geometric fidelity of the fault. (d) Wireframe view of all units and fault showing the crosscutting relationship between fault and surfaces.

3.1.2 Synthetic Model 2

The model comprises a system of non-planar, inclined faults that crosscut the domain, creating a complex internal topology characterized by sharp intersection polylines (Figure 9). This geometry is designed to stress-test the handling of multi-surface intersections. The geometry is enclosed within a rectilinear boundary frame to ensure a watertight volumetric domain for the generation of the PLC. For this model, the hull choice was Delaunay 2D, and TPS was used for both Triangulation and remeshing.

The discretization of this intersecting fault system was performed to validate the algorithm's scalability and efficiency in handling high-angle junctions. The analysis of the resulting tetrahedral mesh reveals 113,670 elements, significantly higher than in the first benchmark, reflecting the increased topological complexity (Table 1). The average radius-edge ratio of 1.34 indicates a generally consistent mesh quality across the domain. This suggests that the TPS reconstruction successfully maintained surface fidelity even with a higher element count.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

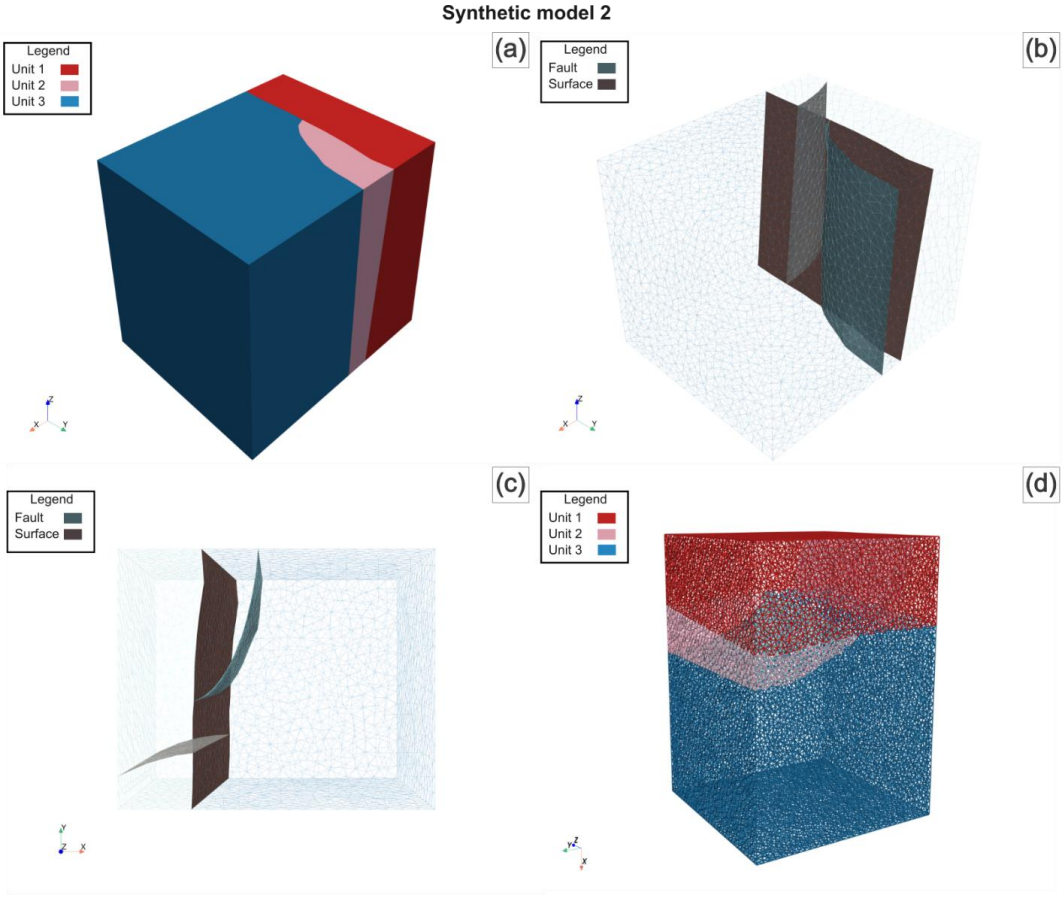


Figure 9. Meshing results for Synthetic Model 2. (a) External surface mesh showing the watertight closure of the domain boundaries. (b) Internal wireframe visualization, demonstrating Fault offset due to the surface. (c) Cross-sectional wireframe view, highlighting the preservation of layer thickness and the continuous mesh transition. (d) Wireframe view of mesh showing fault cross-cutting through the surface.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

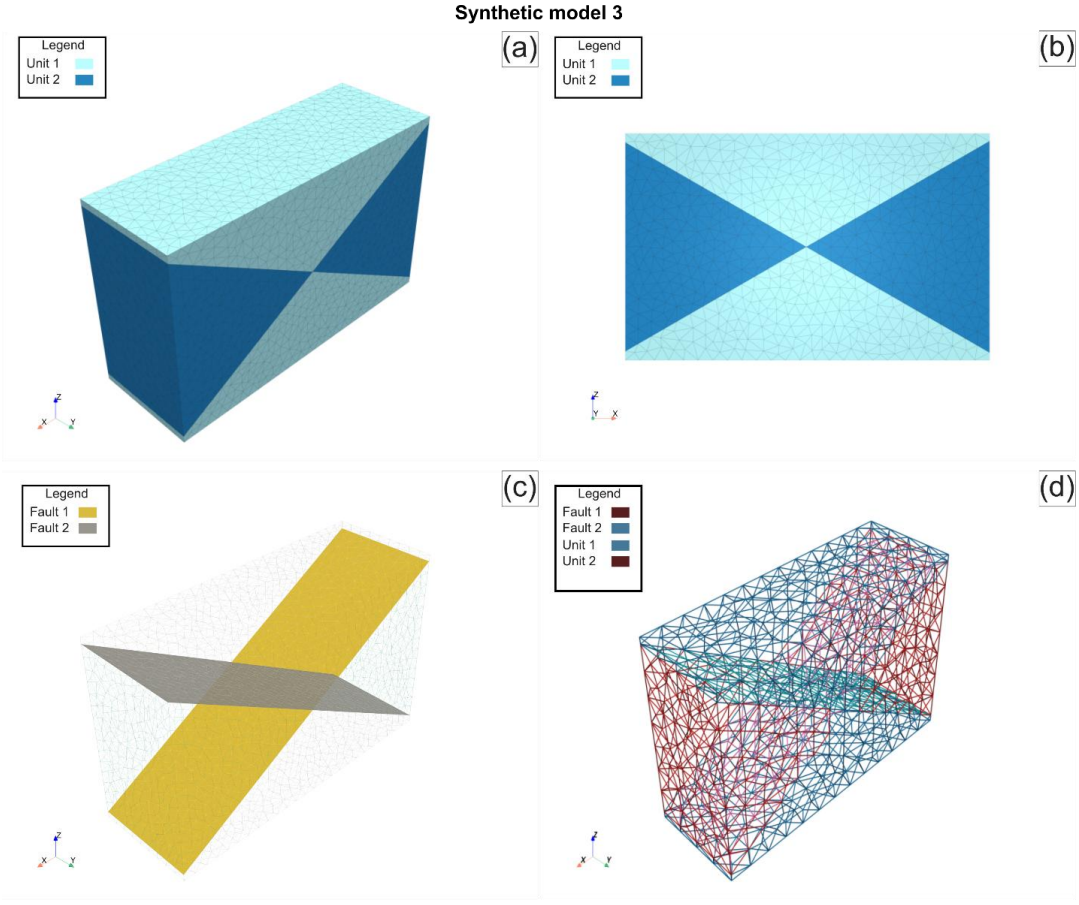


Figure 10. Meshing results for Synthetic Model 3. (a) External surface mesh showing the watertight closure of the domain boundaries. (b) Sideview visualization, demonstrating that faults cross-cut each other. (c) Wireframe view, highlighting the internal faults preserved in the final mesh. (d) Wireframe view of Synthetic Model 3 highlighting both faults and units.

3.1.3 Synthetic Model 3

This dataset encompasses two intersecting faults dividing the model geometry into four separated subdomains. The structural model represents a complex heterogeneous domain characterized by two material regions (Unit 1 and Unit 2) cross-cut by Fault 1 and Fault 2 (Figure 10). This multi-region geometry serves to stress-test the software's seed-based region growing algorithm, validating its ability to correctly identify and discretize multiple watertight volumes within a single PLC. For this model, the hull choice was Delaunay 2D, and IDW was used for both triangulation and remeshing.

The discretization of this multi-domain structure was performed to validate the scalability of the meshing kernel for larger, more complex geological models. The analysis of the resulting tetrahedral mesh consists of 120,744 elements (Table 1). The average radius ratio of 1.32 (where 1.0 is optimal) indicates a high degree of element regularity across the six domains.

3.2 Comparative Benchmarks: PyMeshIt vs. Standard MeshIt C++ Implementation

To validate the correctness and performance of the Python-based implementation, the results from synthetic benchmark 1 (Fault Discontinuities) and synthetic benchmark 2 (Dying Fault) are directly compared against the published results of the original C++ MeshIt implementation (Cacace and Blöcher, 2015; Blöcher, 2020).

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

To evaluate the scalability of the PyMeshIt kernel, synthetic Benchmark 1 (Fault Discontinuities and Offset) was discretized using a high-resolution setting to mirror the mesh density of the reference C++ implementation (Cacace and Blöcher, 2015). This benchmark involves a complex structural setting in which three geological units are cross-cut by two fault zones: a planar fault (Fault 1) and a volumetric fault (Fault 2), resulting in a significant stratigraphic offset. Both benchmarks were tested using the same refinement numbers, i.e., RefineByLength option (the option in the GUI that resamples hull boundaries and intersection polylines by inserting points equally at user-defined spacing).

3.2.1 Benchmark 1

This benchmark considers a non-planar fault surface that displaces a stack of geological layers, completely separating the hanging-wall and footwall blocks (Figure 11 (a) and (b)). The primary challenge is to generate watertight, conforming meshes for the hanging-wall and footwall blocks while preserving the stratigraphic offset across their shared fault surface.

The standard MeshIt implementation generated a mesh of 1,150,388 elements (192,132 vertices) to resolve the fault offset using adaptive refinement (Cacace and Blöcher, 2015). On the other hand, PyMeshIt achieved a valid watertight discretization using 847,538 elements (146,756 vertices) (Table 1). This difference does not reflect a direct like-for-like comparison: as discussed in Section 4.1, the two implementations translate the same refinement parameters into element sizes through different internal algorithms, with the C++ version applying node-level gradient control that produces finer surface meshes near constraint features. The reduction in element count is therefore a consequence of these algorithmic differences rather than a loss of geometric fidelity.

3.2.2 Benchmark 2

Benchmark 2 demonstrates PyMeshIt's ability to discretize a blind or dying fault, here called a finite fault, e.g., a fault surface terminating laterally with its tipline within the model volume, without reaching any bounding surface (Figure 12 (a) and (b)). The resulting mesh comprises 61,751 elements, representing a 44% reduction compared to the 112,309 elements of the C++ reference implementation (Cacace and Blöcher, 2015) with topological correctness of the fault tip line confirmed by visual inspection and full conformance of all intersection polylines in the final mesh (Table 1)

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

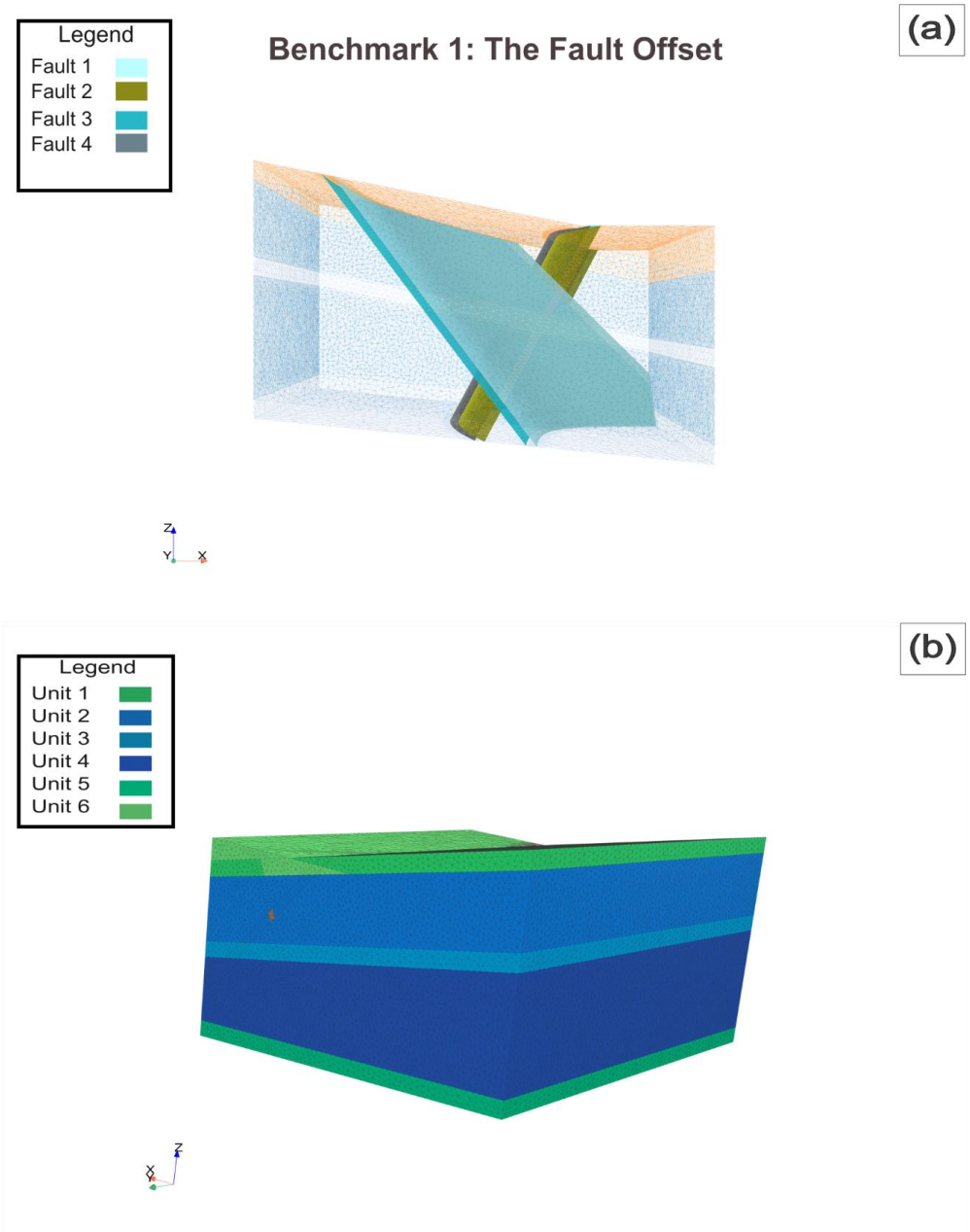


Figure 11. Meshing results for Benchmark 1: "The Fault offset". (a) Wireframe view showing Fault 1, 2, 3, and 4. (b) Surface-mesh view showing the proper material assignment, highlighting the different formations as Units 1- 6.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

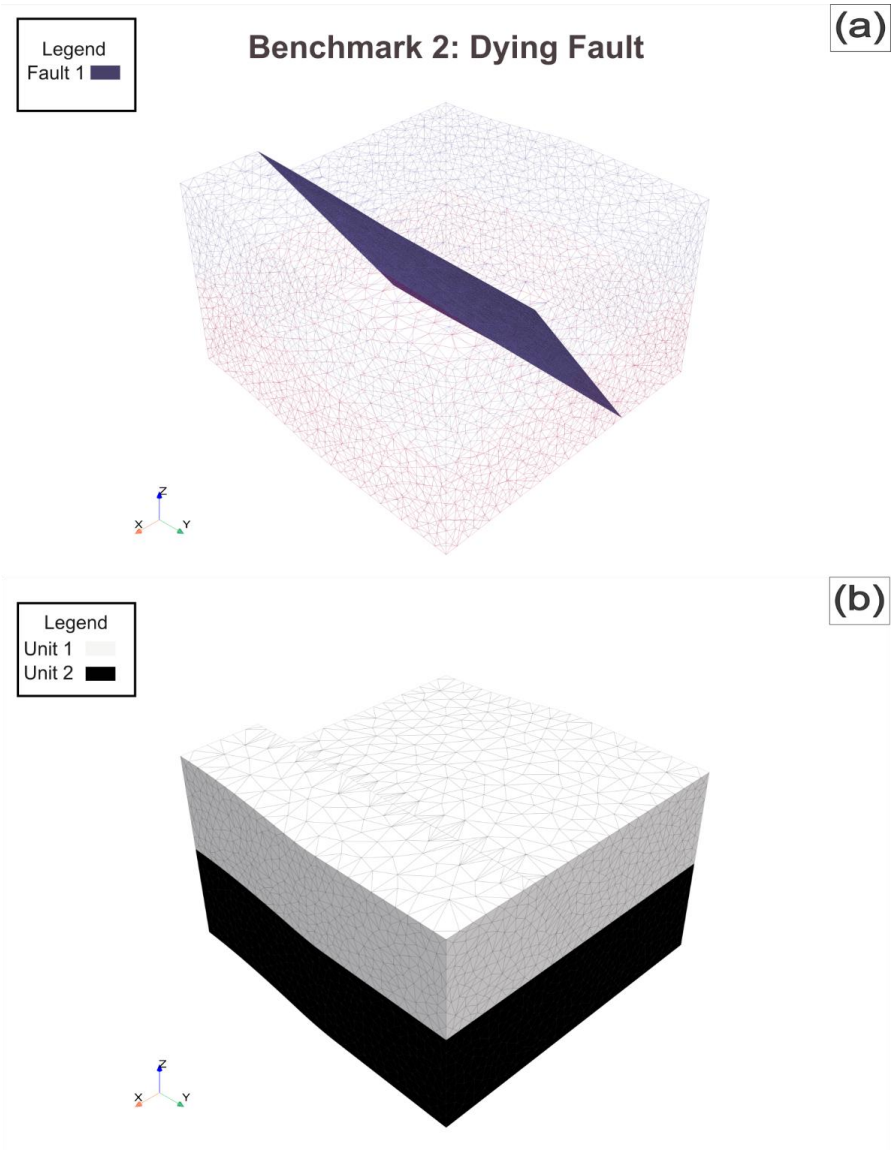


Figure 12. Meshing results for Benchmark 2 "The Dying Fault": (a) Wireframe view of proper fault extraction from the Dying Fault benchmark, (b) Proper volumetric mesh of the Dying Fault, highlighting Units 1 – 2.

3.3 Real Case Studies (Standard CDT)

3.3.1 Mauerstetten (Germany)

This case study focuses on a low-permeability carbonate reservoir located in the southwestern sector of the South German Molasse Basin (SGMB). The target reservoir - the Upper Jurassic Malm aquifer- lies at approximately 3,700 meters depth and is characterized by a prominent fault system striking East-West to Northeast-Southwest. The structural model, spanning an area of 8.7 × 9.8 km with a vertical extent of 1,080 m (Figure 13), incorporates three primary geological units, two faults showing vertical separation in the 150–200 m range and approximately 150-meter-

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

wide damage zones, and a hydraulically induced fracture ca. 300 x 200 m, situated near the damage zone (Cacace et al., 2013; Hofmann et al., 2014). The reservoir includes a well passing through the hydraulic fracture.

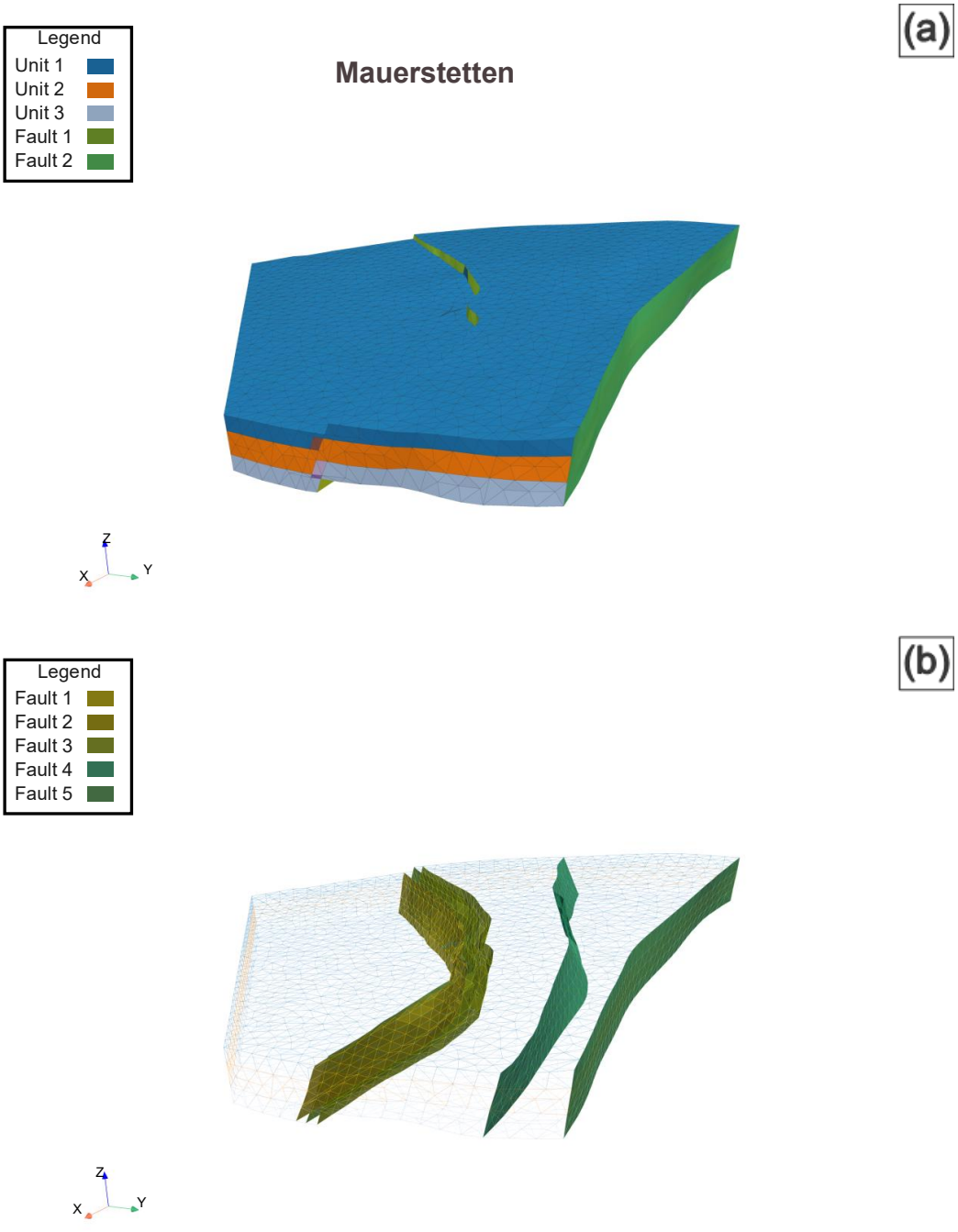
In this case, PyMeshIt generated a computationally efficient mesh consisting of 236,847 elements (tetrahedra) (Table 1). The average radius-edge ratio was 1.84, indicating a mesh quality suitable for reservoir simulation.

3.3.2 Groß Schönebeck (Germany)

The second case study models the deep geothermal reservoir of Groß Schönebeck, located north of Berlin in the Northeast German Basin. The reservoir, situated in Lower Permian sedimentary rocks at depths of 4,050–4,250 m, comprises a sequence of six stratigraphic units crosscut by two major NW-SE fault zones and two minor NE-SW faults with negligible separation (Figure. 14). The model covers a domain of 5.4 × 4.8 km x 594 m, and also includes four hydraulic fractures associated with production and injection wells (Zimmermann et al., 2009, 2010; Blöcher et al., 2010). The domain is bounded by two fault zones and two minor faults that act as no-flow barriers, requiring precise watertight sealing. This benchmark also includes a well that passes through multiple fractures.

This case study served as a stress test for the seed-based region growing algorithm due to the high number of internal compartments resulting from crosscutting faults and stratigraphic units. The PyMeshIt kernel generated a volumetric grid of 66.175 elements (Table 1). The solver successfully identified and assigned unique material IDs to the six geological units and the complex fault/fracture network (Units 1 - 6, Faults 1 - 4, Water and Multi Fractures). The global average radius-edge ratio was 7.5. While higher than the synthetic benchmarks, this value reflects the extreme geometric constraints imposed by meshing thin, high-aspect-ratio hydraulic fractures and the inclusion of a well (heights 95–145 m) within a km-scale domain.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026



598

599 **Figure 13.** Meshing results for Mauerstetten: (a) Surface-mesh view showing the proper material assignment, highlighting the
600 different formations as Units 1- 3, along with faults; (b) Wireframe view showing Fault 1 -6.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

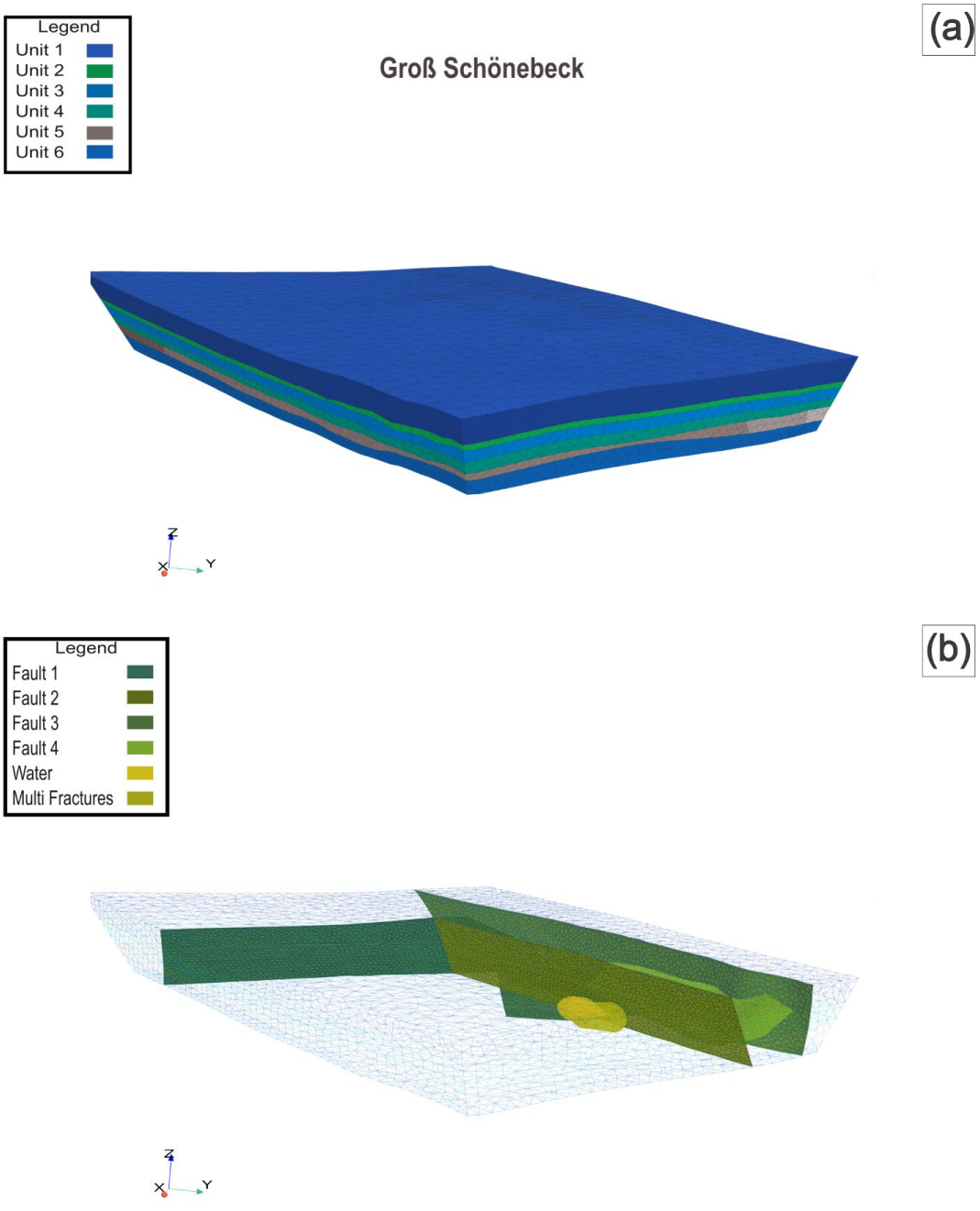


Figure 14. Meshing results for Groß Schönebeck: (a) Surface-mesh view showing the proper material assignment, highlighting the different formations as Units 1- 6; (b) Wireframe view showing Fault 1 – 4 along with Water and Multi Fractures

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

3.3.3 Utah Forge Phase 3 (US)

In this case study, PyMeshIt is used to remesh the Utah FORGE Phase 3 numerical model from the Frontier Observatory for Research in Geothermal Energy (FORGE) project near Milford, Utah, USA. The original numerical model covers a domain of approximately 4.0 km × 4.0 km × 4.2 km beneath the FORGE site and represents the reservoir and surrounding geological volume. The model (Figure 15) includes two formations, bounded by three surfaces, and two wells that run parallel with respect to each other, crossing multiple fractures (Podgorney, 2020).

This case study served as a test for internal intersection due to the wells passing multiple fractures. The PyMeshIt kernel generated a volumetric grid of 293,358 elements (Table 1). The solver successfully identified and assigned unique material IDs to the two geological units and multiple Fractures. The global average radius-edge ratio was 2.38. The value is a bit high due to the inclusion of fractures that are more refined around the well feature points, as two parallel wells pass through the multi-fracture settings. This model was also tested on a simulation by (Gruzdeva et al., 2026).

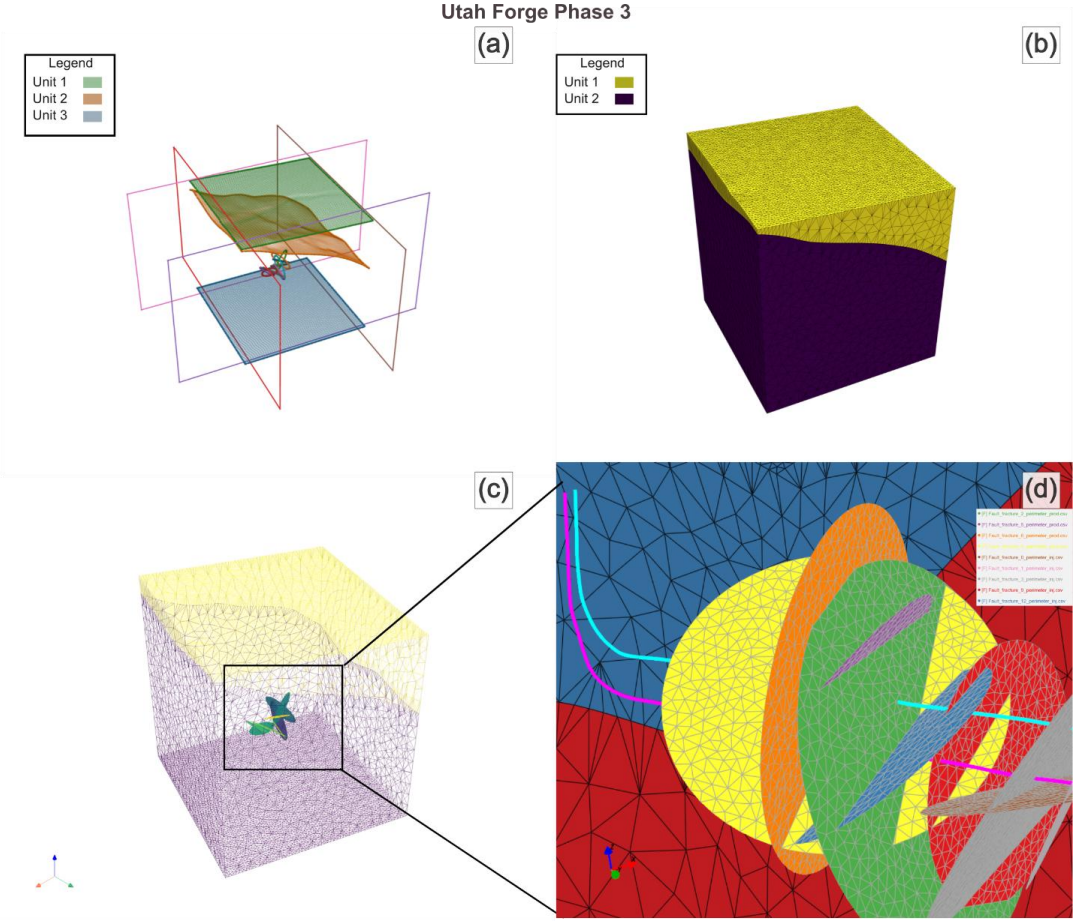


Figure 15. Remeshing results for Utah Forge Phase 3: (a) Computing hulls of Unit 1-3 along with multiple fractures; (b) Surface view showing Formation 1 and 2 as Unit 1 and Unit 2; (c) Wireframe view showing the internal fractures and surfaces; (d) Zoom-in view of Fractures and Wells within the final mesh cross-cutting multiple fractures

Table 1. Comprehensive PyMeshIt Benchmark Statistics: Quantitative analysis of mesh complexity and element quality across all synthetic and real-world scenarios. "PZero" datasets represent folded structures imported from PZero. Quality is measured via the Radius-Edge Ratio.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Benchmark Case	Domain Type	Total Elements	Avg. Quality	Min Quality	Max Quality
Synthetic 1 (PZero)	Wavy Fold	**126,704**	1.40	1.00	306.89
Synthetic 2 (PZero)	Fault Cross-Cut	**113,670**	1.34	1.00	37.17
Synthetic 3 (PZero)	Fault region cross-cut	**120,744**	1.32	1.00	29.16
Synthetic 4 (Benchmark 1)	Dying Fault	**61,751**	21.9245	1.00	8.54
Synthetic 5 (Benchmark 2)	Fault Offset	**847,538**	1.50	1.00	634.629
Mauerstetten	Faulted Layered Reservoir	**236,847**	1.84	1.00	23807.5
Groß Schönebeck	Faulted and Fractured Reservoir	**66,175**	7.5	1.00	141135
Utah Forge phase 3	Crystalline Geothermal Reservoir	**723,338**	2.38	1.00	441632

3.4 Extended Capability: Folded Surface Meshing

The three benchmarks in this section test PyMeshIt's extended capability, with respect to MeshIt, for complex folded surfaces. They cover two distinct input scenarios: (i) a synthetic point cloud processed with the 3D angular gap hull method, and (ii) pre-triangulated surfaces imported from PZero real-world projects at different scales.

3.4.1 Synthetic folds with 3D Angular Gap Hull

This benchmark tests PyMeshIt on a strongly folded, multi-hinged surface characterized by steep limbs and pronounced three-dimensional curvature (Figure 16). The surface was provided as a raw point cloud with no pre-existing triangulation.

The geometry is neither isoclinal nor recumbent; however, its pronounced non-planarity makes conventional two-dimensional boundary extraction and triangulation unsuitable. When the point cloud is projected onto a single plane, geometrically separate parts of the folded surface can overlap or become closely spaced in the projected domain. A Delaunay-based 2D boundary therefore tends to connect points that are close in projection but are not adjacent along the three-dimensional surface, producing artificial shortcuts across fold limbs or hinges. Similar limitations affect other boundary methods operating solely in the projected plane.

This case served as a test of the new algorithms of PyMeshIt, i.e., Isomap Triangulation and 3D angular Gap method. The PyMeshIt kernel generated a volumetric grid of 14,550 elements (Table 2). The solver successfully identified and assigned unique material IDs to a single isoclinal fold. The global average radius-edge ratio was 1.66.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

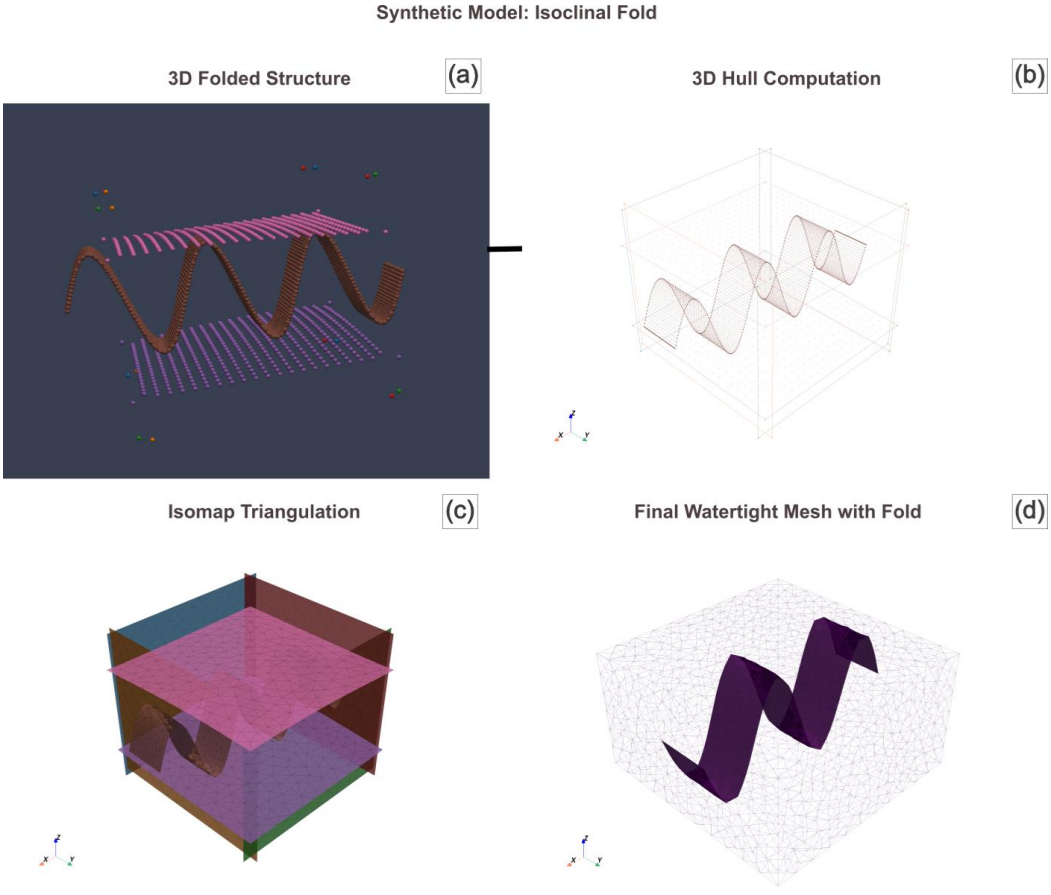


Figure 16. Synthetic benchmark demonstrating the performance of the new PyMeshIt algorithms on an isoclinal fold geometry. (a) Raw 3D point-cloud representation of the synthetic isoclinal fold, characterized by subparallel limbs and a steeply inclined axial plane, resulting in significant overlap in planar projection; (b) 3D hull computation using the newly implemented angular-gap–based reconstruction approach, overcoming the limitations of conventional convex-hull and alpha-shape methods that incorrectly bridge across the fold hinge; (c) Isomap-based triangulation showing successful manifold reconstruction and boundary preservation of the folded surface; (d) Final watertight volumetric mesh generated by the PyMeshIt kernel, with correct material assignment for the folded structure.

3.4.2 Pre-triangulated isoclinal recumbent folds

This case study is based on the San Bernardino Pass region within the Penninic domain of the Central Alps, which comprises continental and oceanic crust derived from the distal European margin and subsequently subducted during the Alpine orogeny (Figure 17). In this area, the Adula nappe is tectonically juxtaposed against the overlying Tambò Misox Zonenappe along a several-hundred-metre-thick (500 m) top-to-the-E extensional shear zone, the San Bernardino Shear Zone, forming part of the eastern flank of the Lepontine Dome together with the underlying Simano Nappe and the overlying Tambo and Suretta nappes (Montemagni et al., 2026). The study focuses on the simplified 3D reconstruction of the shear zone and the pre-existing fold system developed within the Zervreilla orthogneisses and paragneisses containing eclogitic boudins of the Adula-Misox contact. The folding system is composed of recumbent isoclinal folds subsequently refolded by open to gentle folds. Given the complexity and tight spacing of these structures, they were simplified in the model as a single large, refolded isoclinal fold representing the dominant interference geometry, partially obliterated by the San Bernardino Shear Zone (Monti et al., 2024).

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

This case served as a test of the integration of PyMeshIt within PZero, which allows inferring the topology of triangulated surfaces and their boundaries from PZero (i.e., pre-triangulation) and then applying Isomap Triangulation as described in Section 2.7. In this case, PyMeshIt generated a volumetric grid of 286,787 elements (Table 2). The solver successfully identified and assigned unique material IDs to a single isoclinal fold. The global average radius-edge ratio was 3.8.

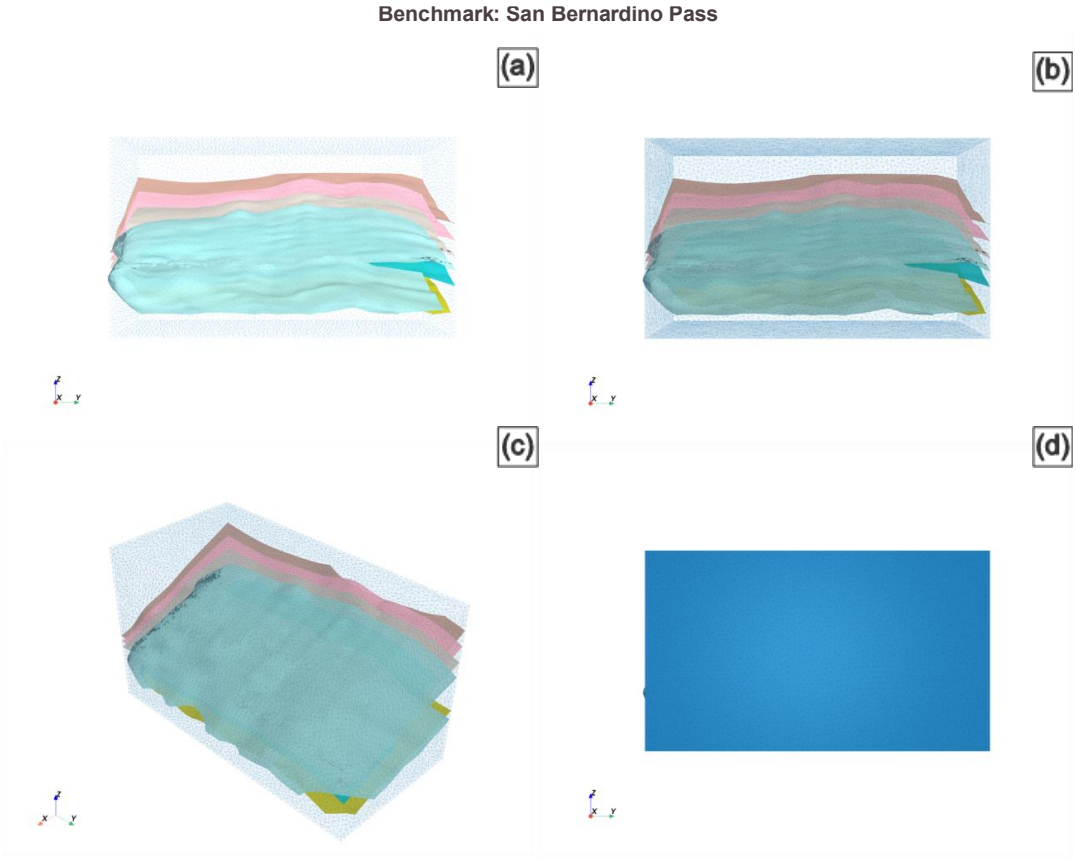


Figure 17. Benchmark results for the San Bernardino Pass case study demonstrating the integration of PZero with PyMeshIt. (a) Initial input point cloud representing the reconstructed geological horizons and structural framework of the San Bernardino Pass model; (b) Side-view visualization of the generated surface reconstruction after applying the Isomap Triangulation and PZero bridge boundary extraction method as described in Section 2.7; (c) Isometric view of the final watertight volumetric mesh preserving the folded stratigraphic geometry and shear-zone continuity; (d) Surface view of the generated mesh highlighting the external discretized boundary of the geological model

3.4.3 Regional Grand Saint Bernard (GSB) model with pre-triangulated tectonic units

This case study uses a subset of the regional-scale 3D structural model of the Northern Aosta Valley, corresponding to the Grand Saint Bernard nappe system sub-mode (Arienti et al., 2024). We refer to it hereafter as the Regional GSB model. The model discussed here, which includes 17 tectono-metamorphic units, bounded by tectonic boundaries and faults, has been updated with new structural data collected for sheet 069 Gran San Bernardo of the Italian National Mapping project CARG, and clipped to the extent of this sheet. Tectonic boundaries and fault surfaces have been interpolated with both implicit and explicit methods in an area of ca. 400 km^2 (methods in (Arienti et al., 2024)), and the PZero – PyMeshIt workflow is used here to build watertight PLC volumes for each tectonic unit by assembling the geological boundary surfaces into closed, watertight volumetric regions, and to generate tetrahedral meshes. In contrast with other case studies discussed above, which fill a hexahedral box, the N boundary runs along the Swiss-

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Italian border and is highly irregular; the top boundary of the model is represented by a reduced-resolution DEM, and the lower boundary surface is undulated and reaches variable depths according to the locally variable reliability of the model. In terms of modelling challenges, in addition to the large size, the model includes recumbent tight folds, finite faults, and wedge-shaped intersections at a very low angle.

The pre-triangulated surfaces from PZero were imported to PyMeshIt through the PZero bridge; as a result, their exact topological boundary was extracted from the mesh connectivity. Dual triangulation was used for different surfaces. Standard constrained Delaunay triangulation was used for the majority of the surfaces, whereas Isomap-based triangulation was applied to 13, 01, 05, and 16 (Figure 18 (a)). The bounding box was automatically decomposed into six faces and extended to ensure full cross-cutting of the internal surfaces. The final constrained Delaunay tetrahedralization produced a watertight volumetric mesh with material identifiers preserved for all geological regions.

PyMeshIt generated a volumetric mesh of 1,000,666 elements (Table 2). Distinct unique material IDs were assigned to multiple units (Figure 18). The global average radius-edge ratio was 1.4.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

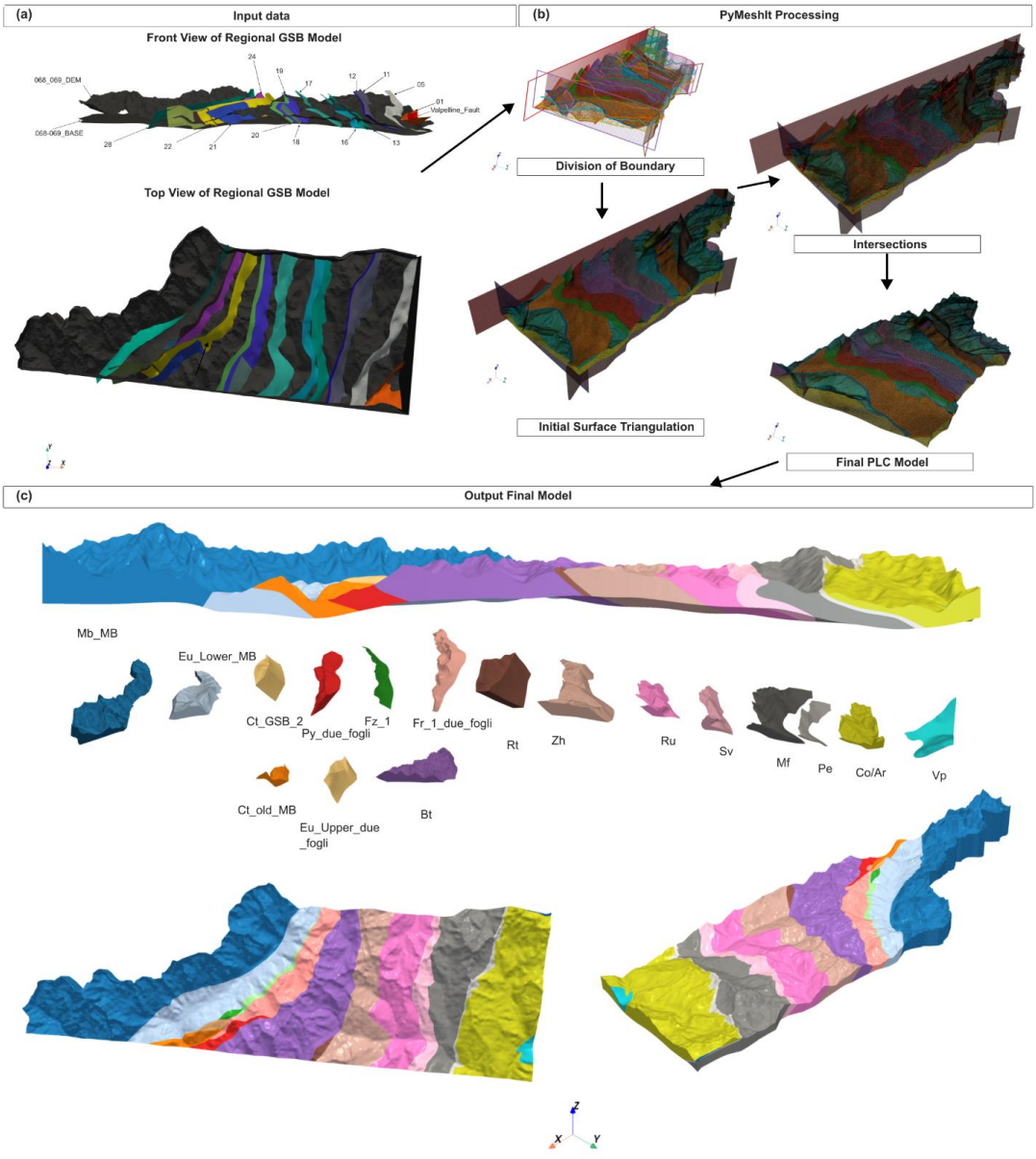


Figure 18. PZero–PyMeshIt workflow and resulting regional geological model. (a) Front and top views of the input triangulated geological surfaces imported from PZero. (b) PyMeshIt processing sequence, including boundary subdivision, initial surface triangulation, intersection computation, and construction of the final PLC. Standard constrained Delaunay and Isomap-based triangulation were applied according to surface geometry. (c) Final watertight tetrahedral model after material assignment, shown as the assembled model, individually extracted tectonic units, and oblique views of the reconstructed geological volume.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Table 2. Comprehensive PyMeshIt Benchmark Statistics: Quantitative analysis of mesh complexity and element quality across all folded models. "PZero" datasets represent folded structures imported from PZero. Quality is measured via the Radius

Benchmark Case	Domain Type	Total Elements	Avg. Quality	Min Quality	Max Quality
Synthetic Benchmark: Isoclinal Fold	Isoclinal Fold	**14,550**	1.66	1.00	523.949
Pre-Triangulated Benchmark: San Bernardino	Overturned Fold	**286,787**	3.8	1.00	7.2
Regional GSB Model (Remeshing)	Multiple folded surfaces with a complex boundary and a Fault	**1,000,666**	1.4	1.00	628

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

4 Discussion

This study presents PyMeshIt, an open-source Python software for generating watertight PLCs and quality tetrahedral meshes for geological modelling. Results from synthetic benchmarks and real-world case studies demonstrate that the software achieves a better mesh quality, with an element count comparable to or lower than that of the reference code MeshIt, implemented in C++.

The following sections examine five aspects of the implementation in detail: the algorithmic basis for the differences in element count between the two implementations (Section 4.1); the native handling of finite fault geometries in PyMeshIt as an explicit Python implementation of MeshIt-style topological preprocessing for open fault-intersection constraints (Section 4.2); the role of boundary extraction (Section 4.3) and interpolation kernel choice in geometric fidelity for folds(Section 4.4); the scientific and practical implications of the tight integration of PyMeshIt within PZero (Section 4.5).

4.1 Algorithmic Efficiency: Refinement and Element Count

As introduced in Section 3.2, the element count difference observed between PyMeshIt and MeshIt C++ in the benchmark comparisons (Section 3.2) arises from the difference in refinement strategy between the two implementations, i.e., gradient-controlled adaptive refinement in MeshIt versus formula-based uniform area constraints in PyMeshIt (Shewchuk, 1996; Cacace and Blöcher, 2015; Si, 2015)(Section 2.5).

A fundamental difference between PyMeshIt and the reference C++ MeshIt implementation (Cacace and Blöcher, 2015) concerns how the same user-specified refinement parameter, RefineByLength, translates into actual element sizes in the final mesh. In MeshIt, this parameter is applied at the node level with a gradient control parameter that governs rapid element size transitions spatially across each surface. In PyMeshIt, the same parameter is converted into a uniform area constraint applied globally across each surface via Triangle's built-in switches, as described in Section 2.5. This means that identical RefineByLength values in the two implementations do not produce identical meshes; they produce meshes with different spatial distribution of element size and element counts, which cannot be compared directly without accounting for this algorithmic difference.

To quantify the impact of this design choice and assess whether the approach impacts mesh quality, we experimentally enabled Triangle's Global callback (triunsuitable) across all surfaces for Benchmark 1 in PyMeshIt, copying the locally adaptive behaviour of MeshIt gradient control.

Table 3. Showing mesh statistics for Benchmark 1 with and without triunsuitable.

Mesh Metric for Benchmark 1	Global callback (triunsuitable)	Refinement (Current)
0D Entities (Points)	28,775	23,307
2D Fault Triangles	9,735	5,841
3D Elements (Tetrahedra)	155,355	124,949
Avg Quality (Radius-Edge)	1.428	1.437

The data show that enforcing MeshIt triangle global callback generates an approximately 24% increase in the total tetrahedral element count and drastically increases the 2D fault triangle count. Visual inspection of the generated meshes corroborates these statistics (Figure 19): the global callback extension forces an excessively dense surface triangulation near geometric boundaries, and this aggressive refinement propagates into the adjacent volumetric domain, increasing element count without any associated improvement in the mesh structure (Table 3). Importantly, this increased computational burden does not produce a geometrically superior mesh. The mean radius-edge ratio remains practically identical between the two approaches (1.428 with the callback versus 1.47 without), confirming that the formula-based approach achieves equivalent mesh quality with significantly fewer elements. The occasionally higher maximum radius-to-edge ratio observed with the simplified approach reflects localized anomalous cells near complex geometric junctions. These isolated elements do not affect the global suitability of the mesh for numerical simulation.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

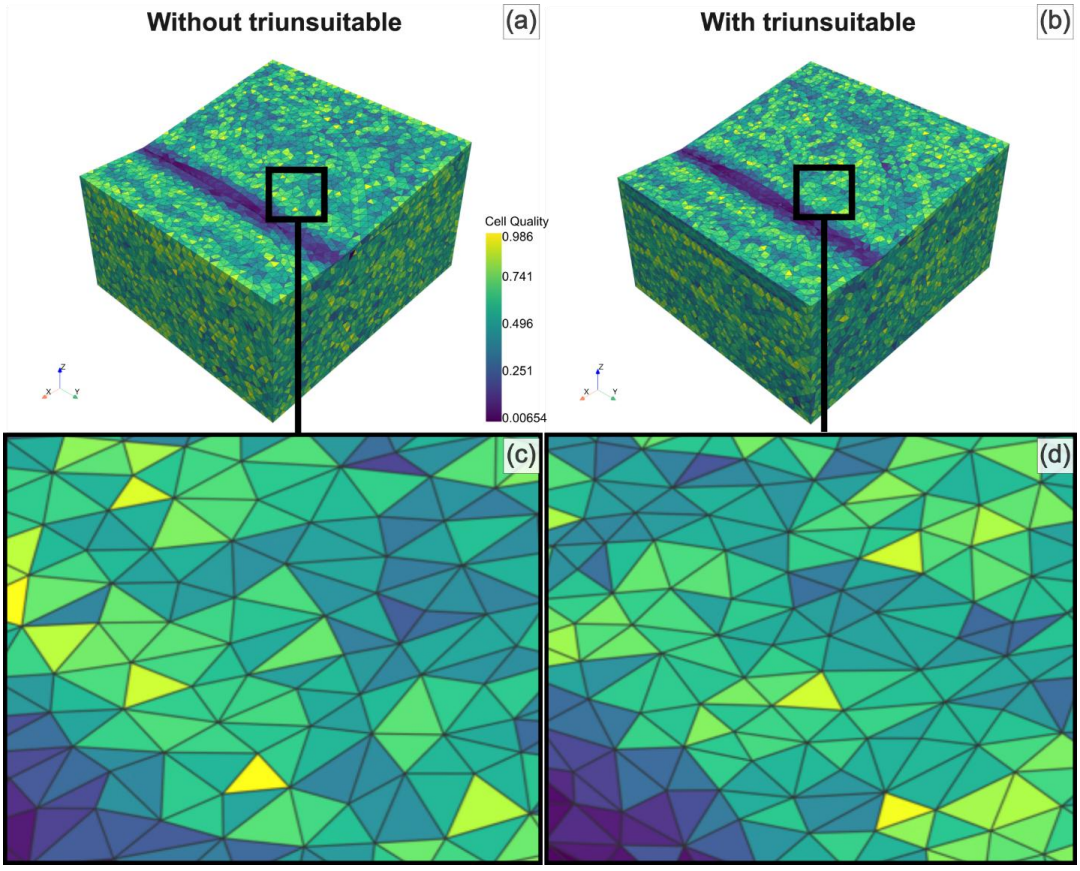


Figure 19. Comparative evaluation of mesh quality for Benchmark 4 with and without the triunsuitable refinement criterion. (a) Volumetric mesh generated without the triunsuitable constraint, showing localized low-quality triangular elements concentrated near the central structural discontinuity; (b) Mesh generated with the triunsuitable criterion enabled, resulting in improved element distribution and enhanced local mesh regularity. (c) Enlarged view of the mesh region highlighted in (a), illustrating irregular element sizing and reduced triangulation uniformity in the absence of adaptive refinement control; (d) Enlarged view of the corresponding region in (b), demonstrating improved triangle quality and more consistent element gradation after applying the triunsuitable criterion. Cell colors represent mesh quality metrics, with warmer colors indicating higher-quality elements and cooler colors indicating lower-quality elements.

Moreover, enforcing triunsuitable globally in a Python environment introduced stability issues, as described in Section 2.5, where denser boundary triangulations produced near constraint features conflicted with the hull snapping tolerances of the align-intersection-to-convex-hull procedure, causing TetGen to reject or remove constrained PLC facets in a subset of test cases (Si, 2015). Resolving these mismatches would require architectural changes to the snapping tolerance system across the full pipeline, at the risk of disrupting benchmark consistency (Si 2015). Additionally, at resolution settings equivalent to MeshIt gradient-controlled output, the Python overhead, particularly during dense triangle-triangle intersection computation and surface interpolation, made global callback execution computationally heavy.

Given these findings, PyMeshIt adopts a hybrid strategy: the Global callback is restricted exclusively to one-dimensional well trajectory features, where its locally adaptive behaviour is genuinely advantageous and the snapping problem does not arise, while the formula-based area constraint is retained for all two-dimensional geological surfaces. This targeted implementation performs reliably across all benchmarks, including the complex well configurations in the Mauerstetten, Groß Schönebeck, and Utah Forge models. This hybrid design represents a deliberate algorithmic

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

adaptation of the original MeshIt workflow to a Python-based implementation, prioritizing robustness, reproducibility, and simulation-ready geological attribution over exact replication of the C++ refinement strategy (Gruzdeva et al., 2026).

These results reveal an important operational trade-off in the PyMeshIt framework. The lower element counts observed in the benchmarks (26% fewer in Benchmark 1, 44% fewer in Benchmark 2) arise from the different refinement algorithms rather than from faster computation. PyMeshIt's uniform area constraint produces fewer, more evenly distributed elements than MeshIt's gradient-controlled output, matching MeshIt mesh density. PyMeshIt's practical advantages lie instead in its simplified single-parameter interface that removes the need to tune gradient control in its native integration with the Python scientific ecosystem and PZero (Section 4.5); and in the workflow reproducibility and automation it enables through scripted pipelines.

4.2 Topological fidelity in boundary extraction

A methodological advancement in PyMeshIt with respect to MeshIt is the implementation of an adaptive Alpha Shape algorithm for boundary definition, addressing a fundamental limitation in standard convex hull approaches (Edelsbrunner et al., 1983). The MeshIt C++ relied on standard convex hull algorithms, especially Graham Scan (Graham, 1972; Cacace and Blöcher, 2015). While computationally efficient, Graham Scan runs in O (N log N) time, meaning computation grows only slightly faster than linearly with the number of input points, making it fast even for large datasets. These algorithms act as a "rubber band" around the dataset, always producing the smallest convex polygon that encloses points. The resulting boundary has a convexity of exactly 1.0, meaning it contains no inward-curving section whatsoever; every part of the boundary curves outward or is straight, regardless of the actual shape of the geological surface. In geological contexts, this artificially "fills in" concave features such as folded or wavy surfaces, or sinuous channels (Figure 20 (a) and (b)). This creates spurious mesh elements in void regions where no data are available, potentially introducing significant errors in the boundary conditions for the subsequent constrained triangulation.

To evaluate the impact of the Alpha Shape approach, Figure 20 illustrates the difference between the two methods, i.e., Delaunay2D and Alpha Shape, when reconstructing the hulls of fault surfaces of Benchmark 1, and particularly in Figure 20 (c) and (d), arrows mark the concave regions where the standard convex hull produces straight bridging segments, while Alpha Shape correctly traces the boundary (Figure 20 (a) and (b).

Table 4. Quantitative Comparison of Boundary Extraction Algorithms: Analysis of hull generation on synthetic fault surfaces. "Delaunay" refers to the standard Projected Delaunay (Convex) Hull, and "Alpha" refers to the Adaptive Alpha Shape algorithm.

Dataset	Method	Vertices	Perimeter (m)	Convexity	Time (ms)
Fault 1 and Fault 2	Delaunay	12	2,096.54	1	1.46
	Alpha	24	2,096.54	0.999	35.65
Fault 3 and Fault 4	Delaunay	9	2,511.58	1	0.53
	Alpha	26	2,512.00	0.989	35.61

The results for Fault 3 and 4 highlight the topological difference. The standard Delaunay hull reduced the boundary to just nine vertices, with a convexity of 1.0, effectively collapsing the irregular fault edge into a simplified polygon that erases all concave features and generates a boundary extending into void regions where no data exist (Figure 20). In contrast, the Alpha Shape identified 26 boundary vertices, nearly 3 times the geometric resolution, with a convexity of 0.989, which mathematically confirms that the algorithm successfully preserved wavy features (concavities) that the convex hull erased. For Fault 1 and 2, which are geometrically closer to convex, both algorithms produce comparable results: the alpha shape convexity of 0.999 reflects only minor concavities, confirming that the alpha shape degrades gracefully to near-convex behaviour when the surface geometry does not require boundary tracing. This adaptive behaviour is important; it means the algorithm does not over-complicate genuinely convex boundaries, avoiding vertex proliferation.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

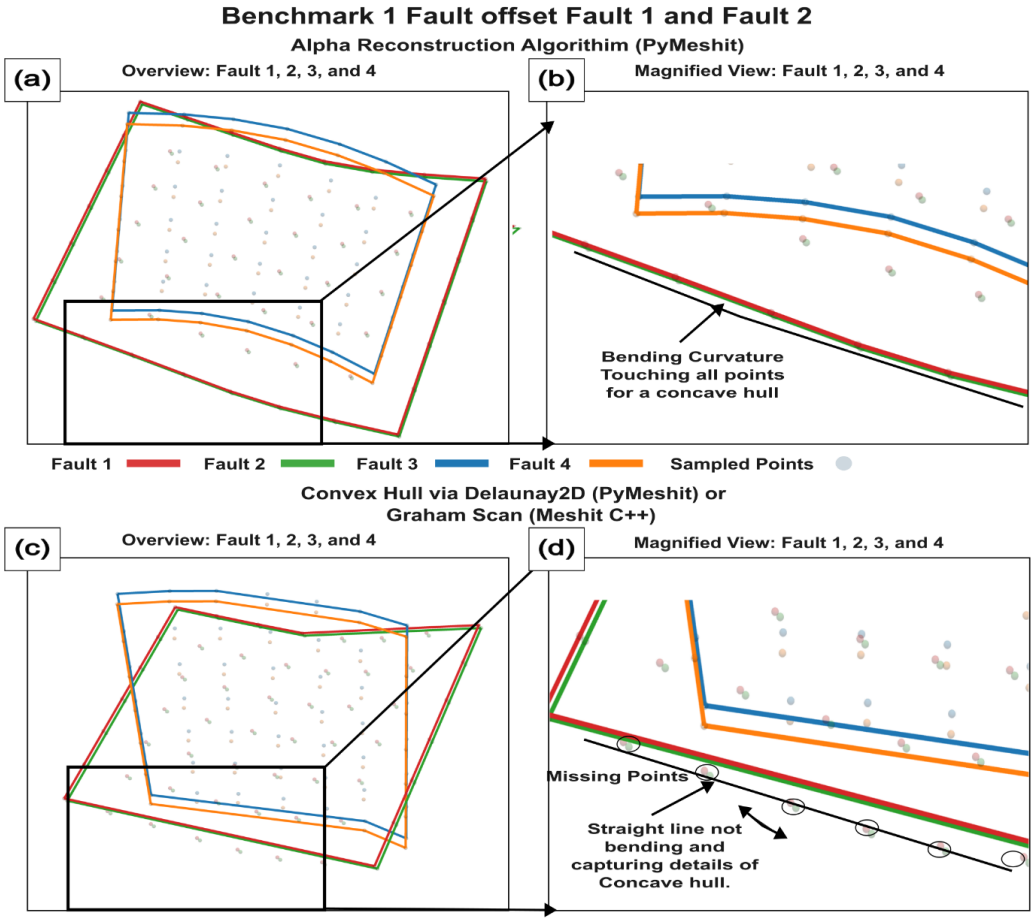


Figure 20. Comparison of concave boundary reconstruction for intersecting fault traces in Benchmark 4. (a) The alpha-shape–based method (PyMeshIt) effectively captures concave geometries (b). (c) Approaches based on convex hulls (Delaunay2D in PyMeshIt and Graham scan in MeshIt C++) enforce convexity globally, which leads to missing points, producing straight lines and failing to capture the concavity properly (d).

While the Alpha Shape calculation is computationally more expensive (≈35 ms vs. ≈0.5 ms for Delaunay), the absolute time cost remains negligible in workflows within the full meshing pipeline, where surface-surface intersection detection and constrained tetrahedralization dominate total runtime by several orders of magnitude. This result validates that the trade-off is favorable: PyMeshIt achieves superior geometric fidelity while preventing the generation of spurious elements in void space, without incurring a noticeable performance penalty for the user. This improvement is particularly consequential for real-world geological models where non-convex surface geometries are the norm, such as irregular fault terminations, wavy stratigraphic horizons, and erosional unconformities, as demonstrated by the real and synthetic benchmarks in Section 3.

Although the adaptive alpha shape approach improves boundary recovery for concave, projection-compatible surfaces, it does not remove the fundamental limitation of projection-based meshing. The method still requires a three-dimensional plane before the boundary is extracted. This assumption is valid only when the surface can be represented as a single-valued distance function with respect to the projection plane. For recumbent or overturned folds, or otherwise non-monotonic folded surfaces, opposite limbs may overlap in the projected domain, so points that are

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

geometrically distinct in three dimensions become indistinguishable in two dimensions (Figure 21 (b)). In such cases, changing the alpha parameter, projection plane, or sampling density cannot fully resolve the error, because the loss of topology is introduced by the projection itself (Wellmann and Caumon, 2018; Serazio et al., 2021). These geometries therefore require the separate folded-surface workflow implemented in PyMeshIt, based on three-dimensional boundary detection and geodesic unfolding, rather than a projection-based alpha-shape boundary (Mallet, 2002, p.3; Anquez et al., 2019).

The 3D Angular Gap method described in Section 2.7.1 was developed to overcome the limitations of boundary extraction based on a single global projection. Rather than projecting the complete surface onto one plane, the method operates on local neighbourhoods in three-dimensional space and evaluates the angular distribution of neighbouring points within a locally estimated tangent plane, following the general principles of neighbourhood-based feature and boundary detection in point clouds (Gumhold et al., 2001; Mineo et al., 2019). Interior points of a sufficiently sampled surface are generally surrounded by neighbours in all directions within the local tangent plane, whereas points close to a surface boundary exhibit an open angular sector toward the exterior. Consequently, the method does not require the entire geological surface to be mapped onto a single two-dimensional plane without overlapping different parts of the surface. This makes it better suited to strongly folded, overturned, or other non-monotonic geometries for which global projection can place spatially separate limbs on top of one another. In the benchmarks presented in Section 3.4.1 and Figure 21, the 3D Angular Gap method recovered the external boundaries of strongly folded surfaces for which projection-based convex-hull and alpha-shape approaches produced geometrically incorrect boundaries, providing suitable boundary constraints for subsequent Isomap-based triangulation.

It is worth noting that the 3D Angular Gap method and the alpha shape are complementary tools applied to different geometric solutions. For quasi-planar faults and gently dipping horizons, the alpha shape remains the appropriate choice; it is computationally lighter, better suited to irregularly sampled planar point clouds, and more robust to variations in local point density than the neighbourhood-based angular gap estimator. The automatic planarity check implemented in PyMeshIt routes surfaces below the SVD planarity threshold of 0.5 to alpha shape extraction and surfaces above it to the 3D angular Gap method. In addition to this automatic selection, the user always has the choice to select whatever method the user wants for their data.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

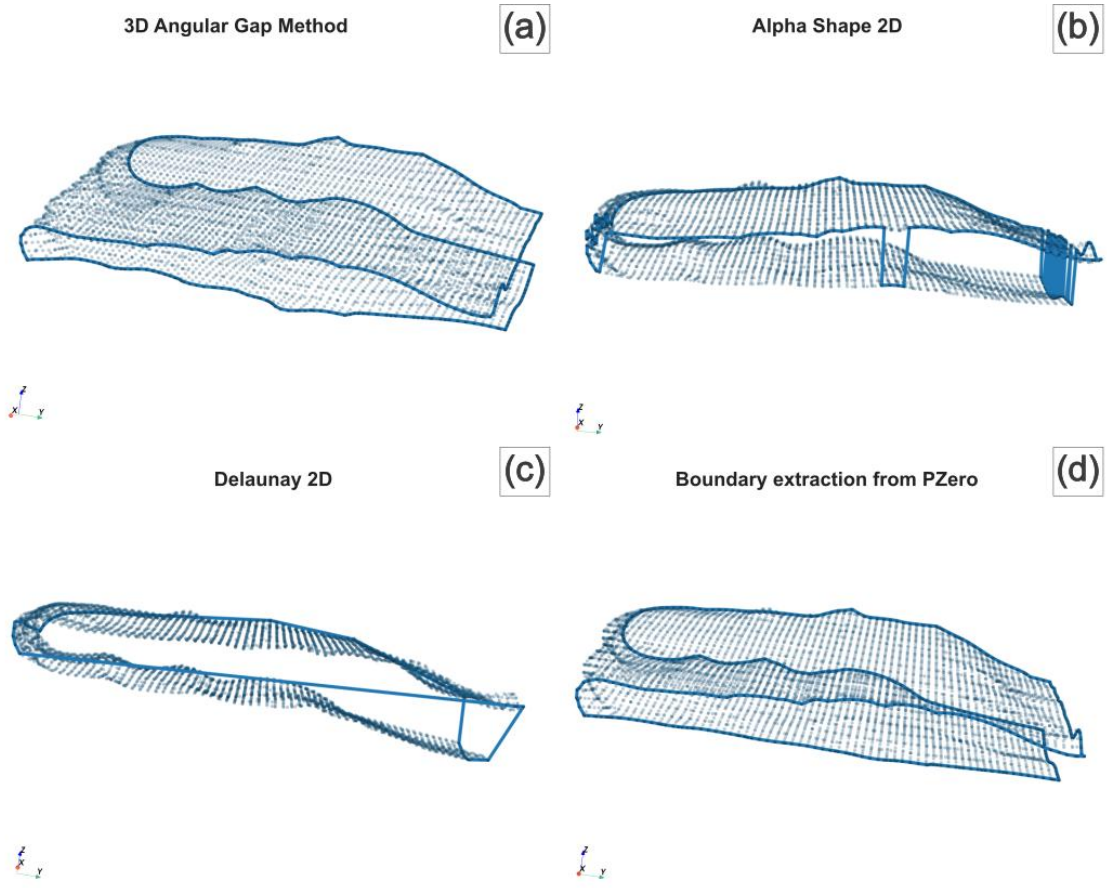


Figure 21. Comparison of boundary extraction methods applied to a folded geological surface with non-monotonic geometry. (a) 3D Angular Gap method correctly reconstructs the external boundary of the folded surface directly in three-dimensional space using local neighbourhood angular analysis in the tangent plane. (b) Adaptive Alpha Shape computed after 2D projection, showing erroneous boundary bridging across overlapping fold limbs caused by the non-injective projection of the recumbent geometry. (c) Delaunay 2D boundary extraction producing severe cross-limb connections and geometrically invalid triangulation due to projection-induced overlap. (d) Boundary extracted using the boundary extraction method from PZero, illustrating partial boundary recovery but with remaining geometric inconsistencies in highly folded regions.

4.3 Extending Automated Meshing to Folded Geological Surfaces

The results presented in Section 3.4 demonstrate that PyMeshIt can generate quality tetrahedral meshes for folded geological surfaces. This section discusses what makes folded surface meshing difficult, what the two workflows contribute, and what their current limitations are.

Folded surfaces have been a persistent challenge in geological mesh generation for a simple geometric reason: almost all existing automatic explicit meshing workflows rely at some point on projecting a three-dimensional surface onto a flat plane to apply two-dimensional triangulation algorithms (Mallet, 2002, p.109; Caumon et al., 2009; Cacace and Blöcher, 2015; Pellerin et al., 2017). This approach is straightforward for gently dipping surfaces, and can be applied to relatively planar surfaces with whatever dip angle, provided that a best-fit projection plane is used (Mallet, 2002), but fails irreversibly for surfaces that fold back on themselves, because points from different parts of the surface, that are geometrically and topologically distinct, get projected in close positions on the projection plane (Figure 6) (Section 2.7). The only established solution has been, so far, to divide the folded surface manually into separate patches, mesh each part individually, using appropriate projection planes, and then stitch them together, in a process that requires

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

expert geological judgment and is difficult to automate (Mallet, 2002, p.109). The Isomap unfolding approach eliminates the need for this manual decomposition by replacing the planar projection with a geodesic parameterization that respects the actual geometry of the fold (Tenenbaum et al., 2000). The triangulation is then performed in this geodesic representation rather than in a flat projection, so points on different limbs remain correctly separated throughout the meshing process.

The choice between the two pathways, 3D angular gap hull from a raw point cloud or PZero bridge extraction from a pre-triangulated PZero surface, depends on the geological context and on what input data is available (Muller and Preparata, 1978; Kettner, 1999; Gumhold et al., 2001; Ni et al., 2016; Bistacchi et al., 2021).

The real-case study Regional GSB model (Section 3.4.3) illustrates why the ability to combine different triangulation methods for different surfaces within a single model is a practical necessity for complex geological models. The model consists of geological formations with highly varying geometry: some are quasi-planar surfaces for which standard constrained Delaunay triangulations produce topologically valid meshes, while others, specifically tectonic boundaries 13, 01, 03, and 16, exhibit folding that caused CDT to produce minor self-intersecting triangulations even when the individual triangulation quality metrics are acceptable. Switching these surfaces to Isomap triangulation resolved the self-intersections and produced topologically valid meshes suitable for PLC assembly. The remaining surfaces were processed with standard CDT, which was both faster and geometrically sufficient for their planar character. This selective application of triangulation methods, guided by the automatic planarity check and confirmed by visual inspection in the PyMeshIt viewport, required no modification of the input data and no manual decomposition of surfaces into patches. The entire model, including hull computation, intersection detection, constraint selection, triangulation, and tetrahedralization, was completed under twenty minutes of user time. For comparison, the equivalent model in a commercial platform took approximately two working days (Arienti et al., 2025). This difference reflects not merely computational speed but the reduction in manual intervention: PyMeshIt's GUI guides the user through each stage sequentially, with real-time 3D visual feedback at every step, so that geometric issues such as missing intersections or inverted triangles are identified and corrected immediately rather than discovered only at the final tetrahedralization stage.

Two additional approaches were tested on the Regional GSB model (Section 3.4.3) for completeness. Loading all surfaces as raw point clouds and applying the 3D Angular Gap hull method produced geometrically valid individual boundaries, but did not achieve the same topological precision as the PZero boundary extraction pathway for surfaces that had already been carefully triangulated in PZero. The neighbourhood-based boundary detection introduced minor positional discrepancies at surface edges that propagated into the PLC as small gaps. Standard CDT applied uniformly to all surfaces, including the folded units, produced a mesh containing self-intersecting elements near the fold cores that rendered the final tetrahedral mesh topologically invalid. These experiments confirm that the hybrid approach, topological boundary extraction from the existing triangulation combined with selective Isomap triangulation for folded surfaces, is not merely one option among several but the appropriate strategy for this class of complex models.

The current implementation of the Isomap unfolding has one known geometric limitation. The algorithm estimates geodesic distances by finding shortest paths in a k-nearest-neighbour graph. When the two limbs of a fold are very close together relative to the typical distance between neighbouring points (i.e., when limbs are very close when compared to mesh resolution), the nearest-neighbour graph may connect points on opposite limbs directly, creating a shortcut through the fold core that does not follow the surface. When this happens, the geodesic distances are underestimated across the hinge, the unfolded 2D map is distorted near the fold core, and the resulting triangulation may contain inverted or poorly shaped elements near the hinge line. This problem becomes more severe as the fold tightness increases and the interlimb angle decreases. Potential mitigations under investigation are discussed in Section 4.7.

A further limitation encountered in the model concerns material assignment. In complex multi-formation models with many closed compartments, PyMeshIt's seed-based region identification occasionally fails to assign the correct material identifier to one or two sub-domains, such as Co (Figure 18), due to limitations in how geological legend information is currently encoded in the software. This does not affect the geometric validity of the tetrahedral mesh but requires post-processing correction of material identifiers before the mesh can be used for simulation. (Monti et al., 2026) are developing a solution based on the Structural Topology Model (STM), a workflow for constructing geological legends with explicit topological and ontological constraints defining the relationships between geological

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

units and their boundaries, currently being implemented within PZero (Monti et al., 2026). Integration of the STM with PyMeshIt's material assignment stage is expected to resolve this limitation in a future release.

4.4 Impact of interpolation kernels on geometric fidelity

In surface reconstruction, geometric fidelity refers to how accurately a reconstructed mesh reproduces the true geometry of the target surface, including sharp discontinuities, fault offsets, and high-curvature features. One assumption is that this accuracy scales primarily with sampling density: by increasing the number of input points (fine sampling), the interpolation problem becomes locally constrained enough that even simple, distance-based methods such as Inverse Distance Weighting (IDW) should be capable of resolving complex structural geometries. The benchmarks were designed in part to test this assumption. However, testing against the 'Fault Offset' (Benchmark 1) revealed fundamental limitations in this density-driven assumption (Figure 22 (a) and (b)). While IDW remains computationally efficient for quasi-planar domains, it consistently introduced artificial rugosity and "bullseye" artifacts in regions of sharp displacement (Figure 22 (a) and (b)) (Franke 1982). Even with dense refinement, the local weighted-average nature of IDW attenuated sharp gradients (Shepard 1968). It smoothed displacement amplitudes near fault tips, leading to systematic misalignments between the interpolated hull and intersection traces (Sambridge et al., 1995). Quantitatively, this geometric degradation was evidenced by RMSE values approximately 3-4 times higher than Thin Plate Splines (TPS), with maximum absolute errors up to 6 times larger and a marked reduction in correlation ($R^2 \approx 0.85$ for IDW versus ≈ 0.99 for TPS) (Table 4) (Duchon, 1977). The same results were also reported for Groß Schönebeck.

Table 5. Quantitative assessment of interpolation fidelity. Comparison of Root Mean Square Error (RMSE) and Coefficient of Determination (R^2) between Thin Plate Spline (TPS) and Inverse Distance Weighting (IDW) kernels across the segmented fault network. The results demonstrate that TPS consistently achieves superior geometric accuracy (R^2 > 0.98) with RMSE values approximately 3-4 times lower than IDW, verifying its efficacy in reconstructing high-curvature surfaces without introducing the artificial rugosity associated with distance-weighted averaging.

Dataset	TPS_RMSE	IDW_RMSE	TPS_R2	IDW_R2
Fault1	2.692147	8.639292	0.985776	0.853523
Fault2	2.692147	8.639292	0.985776	0.853523
Fault3	1.412091	5.141743	0.997917	0.972385
Fault4	1.412091	5.141741	0.997917	0.972385

In contrast, the TPS algorithm demonstrated superior performance for fault-related and folded geometries by fitting the smoothest possible surface through all data points without locally averaging away sharp gradients (Figure 22 (c) and (d)) (Duchon, 1977)(Section 2.3). In the same benchmark experiments, TPS preserved a spatially coherent displacement field across the fault offset, ensuring the reconstructed hull remained topologically compatible with the intersection traces (Duchon, 1977). It should be noted, however, that TPS is not immune to numerical constraints; extreme refinement densities (over-segmentation) can render the global spline system ill-conditioned, increasing sensitivity to noise. Consequently, we reframe the kernel choice in terms of **geometric suitability** rather than solely in terms of sampling density. While IDW is adequate for gentle topographies, TPS is essential for correctly snapping to the intersection network in highly curved structures and fault tips, motivating its selection as the default kernel for complex structural modeling in PyMeshIt.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

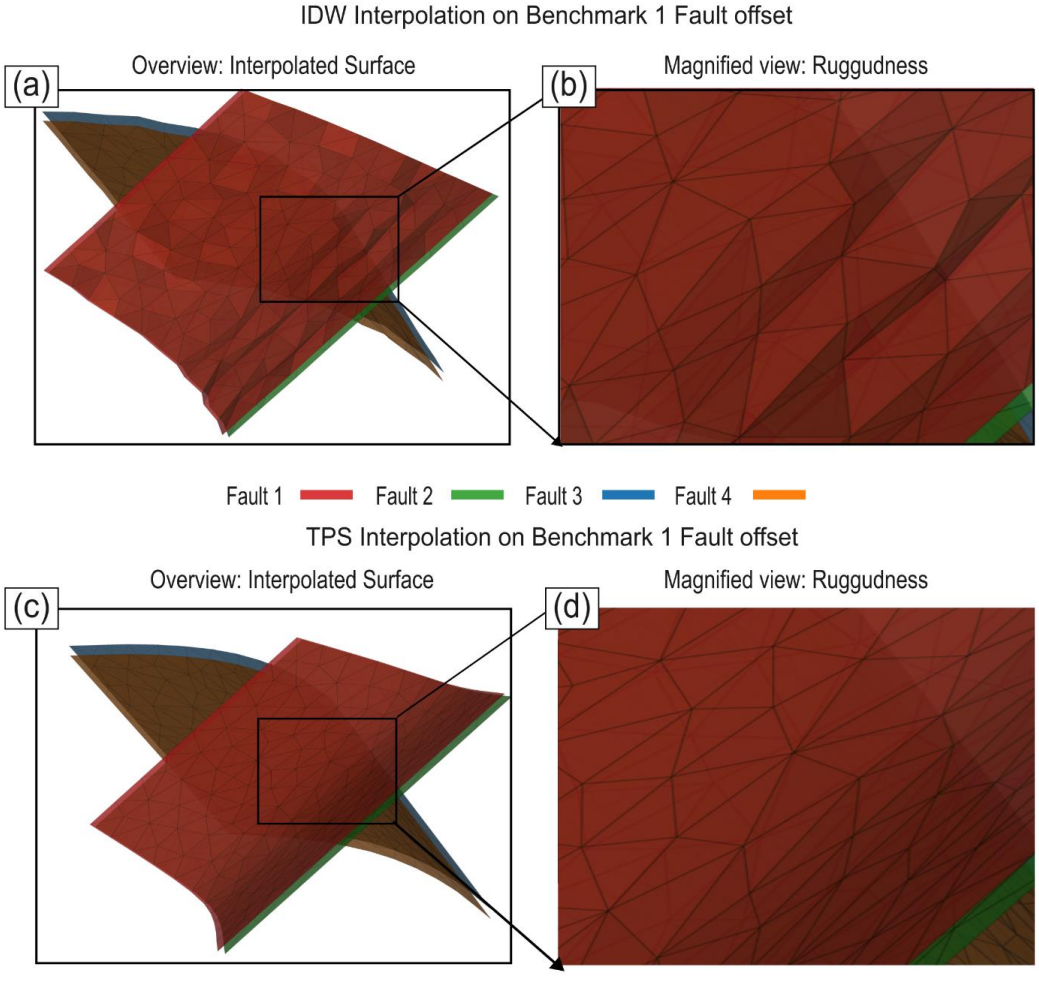


Figure 22. Comparison of surface interpolation methods for Benchmark 4 fault offset reconstruction. (a) Inverse Distance Weighting (IDW) interpolation produces a locally rough surface, with triangulation-scale artifacts and faceted gradients evident in the magnified view (b). (c) Thin Plate Spline (TPS) interpolation yields a smoother surface with continuous curvature and reduced high-frequency noise, better preserving large-scale fault geometry. Inserts highlight local surface behavior within the same region for direct visual comparison (d).

4.5 Integration with PZero: from data ingestion to simulation-ready meshes

A key improvement over the original standalone C++ MeshIt (Cacace and Blöcher, 2015) is the integration of PyMeshIt within PZero (Bistacchi et al., 2021). This integration removes a long-standing bottleneck in geological modeling: the fragile handoff between geomodelling codes (typically surface- and object-based) and numerical meshing (typically point- and PLC-based). By integrating PyMeshIt as a meshing engine inside the same environment that can create models, surfaces, horizons, and support many geological formats and property fields, PZero and PyMeshIt act as a unified modeling ecosystem rather than a loose toolchain.

More specifically, the advantages of this integration are:

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

- The direct exchange of VTK entities (polylines, surfaces, meshes) avoids lossy format conversions - a known source of geometric inconsistencies such as self-intersections, gaps, or sliver elements near faults (Table 1).
- The ability to use pre-optimized, structurally consistent folded surfaces and fault networks from PZero, as in Synthetic models 1-3 and the Regional GSB model, is essential to maintain the intended compartmentalisation pattern while still obtaining watertight volumetric meshes.
- Whereas PyMeshIt can work as a standalone GUI, embedding it into PZero enables a level of automation and reproducibility that was harder to achieve with a separate application.

- PZero includes a system of metadata aimed at transparently managing alternative modelling scenarios. With the PyMeshIt integration, this system is extended to meshing workflows that transparently refer to geological interpretation scenarios and meshing choices.
- The possibility to easily extend PZero functions with the Python scientific ecosystem (NumPy/SciPy for interpolation, filtering, geometric queries, Pandas/xarray for property handling, etc.) will greatly simplify adding preprocessing and postprocessing tools for PyMeshIt.

- Embedding PyMeshIt in PZero means that tetrahedral meshes are generated directly in the same environment where the objects representing geology, wells, fluid domains, and property grids are stored with their petrophysical properties, stratigraphic attributes, and structural data (Benedetti et al., 2023; Monti et al., 2024; Hussain et al., 2025).
- When PyMeshIt converts interpreted surfaces to a PLC and then to a volumetric mesh, it can preserve logical grouping of surfaces into structural units and boundary conditions (e.g., sealing vs. transmissive faults, inflow vs. outflow boundaries), and the mapping between mesh entities (cells, faces) and geological features, which is essential for assigning heterogeneous boundary conditions and material properties in simulators as exports to EXODUS format (Aziz and Settari, 1979; Mallet, 2002; Jackson et al., 2013; Regnier et al., 2022)
- The project-level coordinate reference system management saves users from inconsistencies when exporting/importing geometry between packages.

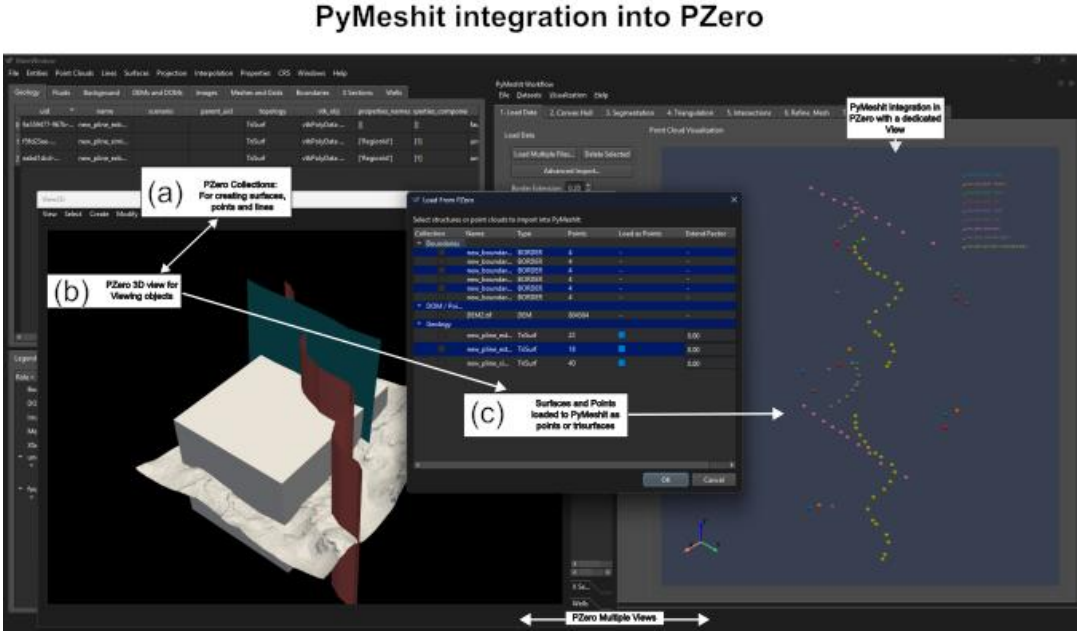


Figure 23. A screenshot of the PZero main window display showing the typical workflow of PyMeshIt integration into PZero. (a) PZero collections showing PZero entities like surfaces and points. (b) A dedicated view for viewing different types of objects supported by PZero; in this example, a Digitally Enhanced Model (DEM), two trisurfaces generated by LoopStructural, and a bounding box. (c) An advanced import table that connects PyMeshIt with PZero, which allows the extraction of PZero entities

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

into PyMeshIt, either as points or trisurfaces such as borders and surfaces, and even DEM. The rest of the figure shows PZero capability to run PyMeshIt as a dock widget and a View 3D running in parallel with respect to each other. The GUI shows the workflow for Synthetic Dataset 2 (Section 3.1.2 in Results).

4.6 Enabling Exact Fault Extraction: Extending the Python–TetGen Interface

A limitation encountered when porting the MeshIt workflow from its original C++ implementation to Python was the loss of geological surface identity during tetrahedralization. The native TetGen C++ interface allows an integer identifier to be assigned to each input facet of the piecewise-linear complex. After tetrahedralization, it also provides the triangular faces associated with the external model boundary and internal interfaces, the identifiers inherited from the corresponding input facets, and the tetrahedra adjacent to each face (Si 2015). These arrays allow downstream codes to reconstruct which boundary triangles correspond to which geological surfaces (e.g., a specific fault plane versus a stratigraphic horizon) and to treat them accordingly in material assignment and boundary-condition specification. However, the widely used TetGen Python wrapper (the TetGen package maintained under PyVista) (https://github.com/pyvista/TetGen/pull/98) only exposes output mesh vertices and connectivity of the tetrahedral cells, and does not provide either (i) complete access to the triangular interface faces and their geological identifiers, or (ii) allow identifiers associated with the input PLC facets to be passed to TetGen.

For geological applications, this deficiency is not merely an implementation detail but a structural limitation. In geological models, faults are represented as internal surfaces embedded within the three-dimensional mesh (Caumon et al., 2004). The ability to later identify all mesh faces that belong to a given fault surface is therefore essential for: (i) assigning distinct geological properties to fault faces, (ii) specifying boundary or internal conditions, and (iii) reconstructing fault surfaces for post-processing.

To address this limitation, during the development of PyMeshIt, we also developed an extension to the TetGen Python wrapper, described in Section 2.5. From the perspective of PyMeshIt and its integration with PZero, this extension closes the loop between interpreted surfaces and volumetric meshes. Since each PLC facet inherits a unique marker, the resulting tetrahedral mesh carries an imprint of the geological model and therefore PyMeshIt can (i) Remesh and extract any individual geological surface, fault, stratigraphic horizon, or domain boundary as an exact subset of the output mesh, with no approximation and (ii) rely on TetGen to relate these boundary faces to adjacent tetrahedra, which is critical for building transmissibility matrices and mixed-dimensional couplings in flow and geomechanical simulators.

Beyond its use in PyMeshIt, the functionality introduced by pull request #98 is also available directly via the official Python interface to TetGen. Users of other scientific and engineering applications therefore do not need to adopt PyMeshIt to benefit from this contribution. They can assign identifiers to input facets and retrieve the corresponding triangular interface faces, facet identifiers, and face-to-tetrahedron adjacency from the generated mesh. This capability is relevant to finite-element and finite-volume workflows in which material boundaries, membranes, electrodes, cracks, or other internal interfaces must remain identifiable after volumetric mesh generation (Zhang et al., 2005; Frey and George, 2008; Geuzaine and Remacle, 2009). By exposing these data through Python, the extension allows users to combine TetGen's Delaunay-based tetrahedralization capabilities (Si, 2015) with direct recovery of labelled boundary and interface faces. PyMeshIt represents one geological application of this general interface extension, using facet identifiers to recover faults, horizons, and external model-boundary surfaces from the tetrahedral mesh.

4.7 Current Limitations and Future Development

PyMeshIt is a robust and automated explicit meshing workflow for faulted and folded geological models, handling surface-surface and surface-polyline intersections, triple-point identification, constrained Delaunay triangulation, and tetrahedral mesh generation through an integrated pipeline. The extension to folded geological surfaces, through the 3D Angular Gap hull method and Isomap-based geodesic triangulation, substantially broadens the range of geological structures that can be meshed without minimum manual intervention. The real-case results presented in Section 3.4 confirm that this extended workflow is functional and produces geometrically valid meshes for genuinely complex geological inputs.

The close-limb limitation of the Isomap triangulation, where very tightly folded surfaces cause the k-nearest-neighbor graph to connect points across fold limbs rather than along the surface, is discussed in detail in Section 4.3. It represents the primary geometric boundary of the current implementation for folded surfaces, and its mitigation through normal-consistency graph pruning following (Budninskiy et al., 2019) is a possible priority for the next development cycle.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Beyond this, two further geometric challenges remain open and are not yet addressed by any component of the current implementation. The first concerns the interaction of folded surfaces with faults. A fault that cross-cuts a recumbent or overturned fold produces intersection polylines that traverse multiple sheets of the folded surface. In the current workflow, the intersection algorithm treats each surface as a single entity and assigns intersection segments to it without distinguishing which sheet they belong to. For a multiply-sheeted surface, this produces intersection polylines that jump between sheets, generating geometrically inconsistent constraints that the constrained Delaunay triangulation stage cannot correctly resolve. Handling this correctly requires explicitly tracking which sheet of a folded surface each intersection segment belongs to, computing triple points consistently across all intersecting sheets, and assembling the PSLG boundary constraints in a sheet-aware order, which would constitute a non-trivial extension of the current topological resolution pipeline (Section 2.4). This is the central algorithmic open problem for the next development cycle, and we regard it as the key capability required to make PyMeshIt fully applicable to some classes of really complex 3D geological models.

The second open challenge concerns surfaces where even Isomap fails due to extreme fold tightness, isoclinal folds where the interlimb angle is very small, and the two limbs are nearly parallel and in close contact throughout the surface. For these geometries, even normal-consistency pruning of the neighbour graph may not reliably prevent cross-limb connections, because the normal vectors on adjacent limbs are nearly antiparallel rather than strongly divergent. A more robust solution for this class of surfaces is a multipatch triangulation strategy, in which the surface is first decomposed into geometrically coherent sheets using local normal orientation and fold angle, each sheet is triangulated independently using standard constrained Delaunay triangulation, and the sheets are stitched back together in three dimensions along the hinge zone. This approach avoids the global geodesic estimation step entirely and is therefore immune to the cross-limb neighbour problem, at the cost of requiring a reliable sheet-decomposition algorithm that can robustly identify fold hinges in noisy geological point clouds (Mallet 2002; Grose et al. 2021).

In addition, one of the main outstanding issues with PyMeshIt is the correct assignment of materials. In the Regional GSB model, one material was not assigned correctly due to limitations in the assignment of material seeds in the final PLC. (Monti et al., 2025) are working on a possible solution via the Structural Topology Model (STm). This is a workflow designed for creating geological legends for 3D modelling, incorporating topological and ontological constraints and defining the relationships between geological units and their boundaries. An application of the STm is now being implemented in PZero, which is part of a wider programme of work to be completed in the near future.

Finally, a practical limitation that affects all benchmarks at high resolution is the computational cost of the Python runtime for dense triangle-triangle intersection computation and large-scale surface triangulation, discussed in Section 4.1. Accelerating the intersection kernel through Cython or C extension modules, replacing the current pure-Python loop over triangle pairs with a compiled implementation, is identified as a straightforward engineering improvement that would substantially reduce preprocessing time for large real-world reservoir models without requiring any changes to the algorithmic pipeline.

5 Conclusion

This study presented PyMeshIt, an open-source Python application for constructing watertight Piecewise Linear Complexes and tetrahedral meshes from raw point clouds and/or pre-triangulated surfaces in complex three-dimensional geological models. PyMeshIt transfers the core MeshIt concept of PLC-based constrained Delaunay tetrahedralization into a modular Python environment, and extends and simplifies the modelling-to-mesh pipeline thanks to the integration within PZero.

In the synthetic and real benchmark cases, PyMeshIt reproduced the principal capability of the original C++ MeshIt workflow: the generation of watertight, boundary-conforming tetrahedral meshes with preserved material regions and geological interfaces. The comparison with MeshIt shows that PyMeshIt produces meshes of comparable quality while using a simpler formula-based surface refinement strategy. This refinement choice generally leads to more spatially uniform triangulations and lower element counts in the tested cases.

Several extensions improve the applicability of the workflow to geological models that are difficult to mesh automatically. Adaptive alpha-shape boundary extraction improves the reconstruction of concave surface outlines compared with convex-hull-based boundaries. Thin Plate Spline interpolation improves coordinate recovery for

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

remeshed surfaces with strong curvature or sharp offsets. PyMeshIt also preserves per-facet identifiers through the Python-TetGen interface, allowing faults, stratigraphic contacts, and model boundaries to be recovered as exact subsets of the final tetrahedral mesh rather than by approximate geometric post-processing.

A major extension of PyMeshIt is its treatment of folded geological surfaces. Standard projection-based constrained Delaunay triangulation fails when fold limbs overlap in planar projection. PyMeshIt addresses this problem through two complementary pathways: a 3D angular-gap method for boundary extraction from raw folded point clouds, and topological boundary extraction through PZero boundary extraction for pre-triangulated PZero surfaces. In both cases, Isomap-based geodesic unfolding provides a two-dimensional parameter domain in which folded surfaces can be triangulated while reducing the risk of artificial cross-limb connections. The synthetic isoclinal fold, San Bernardino, and Regional GSB model-derived examples demonstrate that this strategy can generate watertight volumetric meshes for folded geological geometries while preserving geological attribution of volumetric meshes.

The integration with PZero reduces the handoff between geological interpretation and numerical meshing. Geological surfaces, polylines, wells, and boundary objects can be passed directly into PyMeshIt, remeshed, tetrahedralized, and returned as simulation-ready volumetric meshes with preserved material and boundary information. The successful remeshing of a complex-derived model further indicates that PyMeshIt can also serve as a downstream open-source meshing tool for models generated in external geological modelling platforms, provided they can be exported in standard surface formats.

Remaining limitations are associated mainly with very tight folds, where the k-nearest-neighbor graph used for Isomap may connect points across adjacent limbs instead of along the surface, and with more complex fold–fault interactions requiring explicit multi-sheet intersection handling. Future work will focus on normal-consistency pruning of the Isomap graph, multipatch triangulation for surfaces that cannot be unfolded reliably, and performance optimization of the Python intersection kernel. PyMeshIt is distributed as open-source Python software and a standalone graphical application, providing a reproducible and extensible framework for geological surface-to-volume meshing.

Acknowledgments

This work was supported by the Italian National Recovery and Resilience Plan (PNRR). Part of the research was conducted during a research-abroad period at the GFZ German Research Centre for Geosciences, whose support and scientific environment are gratefully acknowledged. The authors thank Yulia Gruzdeva for continuous testing of PyMeshIt and for her detailed feedback during software development. We also acknowledge the open-source scientific Python and geomodelling communities, in particular the developers of NumPy, SciPy, PyVista/VTK, TetGen, Triangle, and PZero, whose tools form part of the computational foundation of PyMeshIt.

Code availability

PZero, the geological modelling environment used in this study, is open-source and available at https://github.com/gecos-lab/PZero under the GNU Affero General Public License v3.0. The version used in this study (v1.3.5) is archived on Zenodo at https://doi.org/10.5281/zenodo.18771413 (Bistacchi et al., 2026). PyMeshIt, the tetrahedral meshing engine used as PZero's explicit modelling engine, is open-source and available at https://github.com/waqashussain117/PyMeshit under the GNU General Public License v3.0 or later. The version used in this study (v0.9.0) is archived on Zenodo at https://doi.org/10.5281/zenodo.21792100 (Hussain, 2026)

Author contribution

W.H. developed PyMeshIt, implemented the core meshing workflow, performed the numerical experiments, analysed the benchmark results, prepared the figures, and wrote the original manuscript draft. A.B. conceptualized the project and proposed the development of PyMeshIt as a meshing and modelling engine integrated with PZero, supervised the research, contributed to the methodological design, and reviewed and edited the manuscript. M.C. contributed methodological guidance based on the original C++ MeshIt workflow, supported software testing and validation, provided feedback on the meshing strategy, and reviewed and edited the manuscript. R.M. contributed to software testing and workflow validation, provided feedback on the implementation, supported testing with PZero-derived datasets, and contributed geological case-study material, including the San Bernardino case study. All authors discussed the results and contributed to the final manuscript.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Competing interests. The contact author has declared that none of the authors has any competing interests.

References

Adams, R. and Bischof, L.: Seeded region growing, IEEE Trans. Pattern Anal. Mach. Intell., 16, 641–647, https://doi.org/10.1109/34.295913, 1994.

Amenta, N., Bern, M., and Kamvysselis, M.: A new Voronoi-based surface reconstruction algorithm, in: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, 415–421, https://doi.org/10.1145/280814.280947, 1998.

Anquez, P., Caumon, G., Pellerin, J., and Lévy, B.: Automatic sealing and simplification of 3D geological surface models using topology recovery, in: 79th EAGE Conference and Exhibition 2017, 1–5, https://doi.org/10.3997/2214-4609.201701143, 2017.

Anquez, P., Pellerin, J., Irakarama, M., Cupillard, P., Lévy, B., and Caumon, G.: Automatic correction and simplification of geological maps and cross-sections for numerical simulations, Comptes Rendus Géoscience, 351, 48–58, https://doi.org/10.1016/j.crte.2018.12.001, 2019.

Arienti, G., Bistacchi, A., Caumon, G., Dal Piaz, G., Monopoli, B., and Bertolo, D.: Regional-scale 3D modelling in metamorphic belts: An implicit model-driven workflow applied in the Pennine Alps, J. Struct. Geol., 180, 105045, https://doi.org/10.1016/j.jsg.2023.105045, 2024.

Arienti, G., Bistacchi, A., Caumon, G., Monopoli, B., and Dal Piaz, G.: 3D structural implicit modelling of folded metamorphic units at Lago di Cignana with uncertainty assessment, J. Struct. Geol., 191, 105329, https://doi.org/10.1016/j.jsg.2024.105329, 2025.

Attene, M., Campen, M., and Kobbelt, L.: Polygon mesh repairing: An application perspective, ACM Comput. Surv. CSUR, 45, 1–33, https://doi.org/10.1145/2431211.2431214, 2013.

Aziz, K. and Settari, A.: Petroleum Reservoir Simulation, Springer Netherlands, 1979.

Bangerth, W. and Heister, T.: What makes computational open source software libraries successful?, Comput. Sci. Discov., 6, 015010, https://doi.org/10.1088/1749-4699/6/1/015010, 2013.

Belkin, M. and Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation, Neural Comput., 15, 1373–1396, https://doi.org/10.1162/089976603321780317, 2003.

Benedetti, G., Casiraghi, S., Bistacchi, A., Arienti, G., and Bertolo, D.: Point cloud analysis and segmentation procedures in the PZero software, EGU General Assembly 2023, Vienna, Austria, https://doi.org/10.5194/egusphere-egu23-9549, 2023.

Bistacchi, A., Arienti, G., Pecorella, E., and Soldo, L.: PZero: a new open-source Python geomodelling platform, in: Proceedings of the 2021 RING Meeting, RING 2021, 2021.

Bistacchi, A., Benedetti, G., Hussain, W., Monti, R., mcbaguetti, Frigeri, A., GloriaArienti, and ivanobrunet: gecos-lab/PZero: v1.3.5, , https://doi.org/10.5281/zenodo.18771413, 2026.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

1203 Blöcher, G.: bloech/MeshIt: MeshIt 2020, , https://doi.org/10.5281/zenodo.4327281, 2020.

1204 Blöcher, M. G., Zimmermann, G., Moeck, I., Brandt, W., Hassanzadegan, A., and Magri, F.: 3D
1205 numerical modeling of hydrothermal processes during the lifetime of a deep geothermal reservoir,
1206 Geofluids, 10, 406–421, https://doi.org/10.1111/j.1468-8123.2010.00284.x, 2010.

1207 Bookstein, F. L.: Principal warps: Thin-plate splines and the decomposition of deformations, IEEE
1208 Trans. Pattern Anal. Mach. Intell., 11, 567–585, https://doi.org/10.1109/34.24792, 2002.

1209 Borg, I. and Groenen, P. J. F. (Eds.): The Four Purposes of Multidimensional Scaling, in: Modern
1210 Multidimensional Scaling: Theory and Applications, Springer New York, New York, NY, 3–18,
1211 https://doi.org/10.1007/0-387-28981-X_1, 2005.

1212 Budninskiy, M., Yin, G., Feng, L., Tong, Y., and Desbrun, M.: Parallel transport unfolding: a
1213 connection-based manifold learning approach, SIAM J. Appl. Algebra Geom., 3, 266–291,
1214 https://doi.org/https://doi.org/10.1137/18M1196133, 2019.

1215 Cacace, M. and Blöcher, G.: MeshIt—a software for three dimensional volumetric meshing of
1216 complex faulted reservoirs, Environ. Earth Sci., 74, 5191–5209, https://doi.org/10.1007/s12665-
1217 015-4537-x, 2015.

1218 Cacace, M., Blöcher, G., Watanabe, N., Moeck, I., Börsing, N., Scheck-Wenderoth, M., Kolditz,
1219 O., and Huenges, E.: Modelling of fractured carbonate reservoirs: outline of a novel technique via
1220 a case study from the Molasse Basin, southern Bavaria, Germany, Environ. Earth Sci., 70, 3585–
1221 3602, https://doi.org/10.1007/s12665-013-2402-3, 2013.

1222 Caumon, G., Lepage, F., Sword, C. H., and Mallet, J.-L.: Building and Editing a Sealed Geological
1223 Model, Math. Geol., 36, 405–424, https://doi.org/10.1023/B:MATG.0000029297.18098.8a, 2004.

1224 Caumon, G., Collon-Drouaillet, P., Le Carlier de Veslud, C., Viseur, S., and Sausse, J.: Surface-
1225 Based 3D Modeling of Geological Structures, Math. Geosci., 41, 927–945,
1226 https://doi.org/10.1007/s11004-009-9244-2, 2009.

1227 Chew, L. P.: Constrained Delaunay triangulations, in: Proceedings of the Third Annual
1228 Symposium on Computational Geometry, Waterloo, Ontario, Canada, 215–222,
1229 https://doi.org/10.1145/41958.41981, 1987.

1230 Cover, T. and Hart, P.: Nearest neighbor pattern classification, IEEE Trans. Inf. Theory, 13, 21–
1231 27, https://doi.org/10.1109/TIT.1967.1053964, 1967.

1232 Crippen, G. M. and Havel, T. F.: Distance Geometry and Molecular Conformation, Research
1233 Studies Press, 1988.

1234 Deutsch, C. V.: Geostatistical Reservoir Modeling, Oxford University Press,
1235 https://doi.org/10.1093/oso/9780195138061.001.0001, 2002.

1236 Dhont, D., Luxey, P., and Chorowicz, J.: 3-D modeling of geologic maps from surface data, Aapg
1237 Bull. - AAPG BULL, 89, 1465–1474, https://doi.org/10.1306/06270504108, 2005.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Dijkstra, E. W.: A note on two problems in connexion with graphs, Numer. Math., 1, 269–271, https://doi.org/10.1007/BF01386390, 1959.

Duchon, J.: Splines minimizing rotation-invariant semi-norms in Sobolev spaces, in: Constructive Theory of Functions of Several Variables, Lecture Notes in Mathematics, Berlin, Heidelberg, 85–100, https://doi.org/10.1007/BFb0086566, 1977.

Edelsbrunner, H. and Mücke, E. P.: Three-dimensional alpha shapes, ACM Trans. Graph. TOG, 13, 43–72, https://doi.org/10.1145/174462.156635, 1994.

Edelsbrunner, H., Kirkpatrick, D., and Seidel, R.: On the shape of a set of points in the plane, IEEE Trans. Inf. Theory, 29, 551–559, https://doi.org/10.1109/TIT.1983.1056714, 1983.

Farmer, C. L.: Geological Modelling and Reservoir Simulation, in: Mathematical Methods and Modelling in Hydrocarbon Exploration and Production, edited by: Iske, A. and Randen, T., Springer Berlin Heidelberg, Berlin, Heidelberg, 119–212, https://doi.org/10.1007/3-540-26493-0_6, 2005.

Fernández, O., Muñoz, J., Arbués, P., Falivene, O., and Marzo, M.: Three-dimensional reconstruction of geological surfaces: An example of growth strata and turbidite systems from the Ainsa Basin (Pyrenees, Spain), Aapg Bull. - AAPG BULL, 88, 1049–1068, https://doi.org/10.1306/02260403062, 2004.

Floater, M. S. and Hormann, K.: Surface Parameterization: a Tutorial and Survey, in: Advances in Multiresolution for Geometric Modelling, Mathematics and Visualization, Berlin, Heidelberg, 157–186, https://doi.org/10.1007/3-540-26808-1_9, 2005.

Freitag, L. A. and Ollivier-Gooch, C.: Tetrahedral mesh improvement using swapping and smoothing, Int. J. Numer. Methods Eng., 40, 3979–4002, https://doi.org/10.1002/(SICI)1097-0207(19971115)40:21%253C3979::AID-NME251%253E3.0.CO;2-9, 1997.

Frey, P. J. and George, P. L.: Basic Structures and Algorithms, in: Mesh Generation, 45–94, https://doi.org/10.1002/9780470611166.ch2, 2008.

Geuzaine, C. and Remacle, J.: Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, Int. J. Numer. Methods Eng., 79, 1309–1331, https://doi.org/10.1002/nme.2579, 2009.

Graham, R. L.: An efficient algorithm for determining the convex hull of a finite planar set, Info Proc Lett, 1, 132–133, https://doi.org/10.1016/0020-0190%252872%252990045-2, 1972.

Gruzdeva, Y., Degen, D., and Cacace, M.: A Hierarchical multi-fidelity approach for Bayesian inference for numerical process simulations, EGU General Assembly 2026, Vienna, Austria, EGU26-17948, https://doi.org/10.5194/egusphere-egu26-17948, 2026.

Gumhold, S., Wang, X., and Macleod, R.: Feature Extraction from Point Clouds, in: Proceedings of 10th international meshing roundtable, 2001.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E.: Array programming with NumPy, Nature, 585, 357–362, https://doi.org/10.1038/s41586-020-2649-2, 2020.

Hofmann, H., Babadagli, T., and Zimmermann, G.: Hot water generation for oil sands processing from enhanced geothermal systems: Process simulation for different hydraulic fracturing scenarios, Appl. Energy, 113, 524–547, https://doi.org/10.1016/j.apenergy.2013.07.060, 2014.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W.: Surface reconstruction from unorganized points, in: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques, 71–78, https://doi.org/10.1145/133994.134011, 1992.

Hotelling, H.: Analysis of a complex of statistical variables into principal components., J. Educ. Psychol., 24, 417, 1933.

Houlding, S. W.: Features of an Integrated 3D Computer Approach, in: 3D Geoscience Modeling: Computer Techniques for Geological Characterization, edited by: Houlding, S. W., Springer Berlin Heidelberg, Berlin, Heidelberg, 39–69, https://doi.org/10.1007/978-3-642-79012-6_4, 1994.

Hussain, W.: waqashussain117/PyMeshit: v0.9.0, , https://doi.org/10.5281/zenodo.21792100, 2026.

Hussain, W., Bistacchi, A., Benedetti, G., and Monti, R.: AI-Driven PZero Modeling: Enhanced Seismic Data Loading, Grid Section Management, and Automated Interpretation with Sobel, A* Pathfinding, and SAM2, EGU General Assembly 2025, Vienna, Austria, EGU25-16273, https://doi.org/10.5194/egusphere-egu25-16273, 2025.

Jackson, M. D., Gomes, J. L., Mostaghimi, P., Percival, J. R., Tollit, B. S., Pavlidis, D., Pain, C. C., El-Sheikh, A. H., Muggeridge, A. H., and Blunt, M. J.: Reservoir Modeling for Flow Simulation Using Surfaces, Adaptive Unstructured Meshes and Control-Volume-Finite-Element Methods, SPE Reservoir Simulation Symposium, Texas, SPE-163633-MS, https://doi.org/10.2118/163633-MS, 2013.

Journel, A. G. and Ch.J, H.: Mining geostatistics, Academic press, 600 pp., 1978.

Kettner, L.: Using generic programming for designing a data structure for polyhedral surfaces, Comput. Geom., 13, 65–90, https://doi.org/10.1016/S0925-7721(99)00007-3, 1999.

Knupp, P. M.: Algebraic mesh quality metrics, SIAM J. Sci. Comput., 23, 193–218, https://doi.org/10.1137/S1064827500371499, 2001.

Legentil, C., Pellerin, J., Cupillard, P., Froehly, A., and Caumon, G.: Testing scenarios on geological models: Local interface insertion in a 2D mesh and its impact on seismic wave simulation, Comput. Geosci., 159, 105013, https://doi.org/10.1016/j.cageo.2021.105013, 2022.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Legentil, C., Pellerin, J., Raguenel, M., and Caumon, G.: Towards a workflow to evaluate geological layering uncertainty on CO2 injection simulation, Appl. Comput. Geosci., 18, 100118, https://doi.org/10.1016/j.acags.2023.100118, 2023.

Lévy, B., Petitjean, S., Ray, N., and Maillot, J.: Least squares conformal maps for automatic texture atlas generation, ACM Trans Graph, 21, 362–371, https://doi.org/10.1145/566654.566590, 2002.

Mallet, J.-L.: Geomodeling, Oxford University Press, 624 pp., 2002.

Miller, G. L.: Control volume meshes using sphere packing, in: International Symposium on Solving Irregularly Structured Problems in Parallel, 128–131, 1998.

Mineo, C., Pierce, S. G., and Summan, R.: Novel algorithms for 3D surface point cloud boundary detection and edge reconstruction, J. Comput. Des. Eng., 6, 81–91, https://doi.org/10.1016/j.jcde.2018.02.001, 2019.

Möller, T.: A Fast Triangle-Triangle Intersection Test, J. Graph. Tools, 2, 25–30, https://doi.org/10.1080/10867651.1997.10487472, 1997.

Montemagni, C., Monti, R., Malaspina, N., Vannucchi, P., and Zanchetta, S.: Rapid transition from high-pressure metamorphism to exhumation driven by orogen-parallel extensional shearing (Adula unit, eastern Central Alps), Tectonics, 45, e2025TC009045, https://doi.org/10.1029/2025TC009045, 2026.

Monti, R., Bistacchi, A., Benedetti, G., and Hussain, W.: 3D Geomodelling of Alpine structures with PZero, 2024 RING Meeting, Nancy, France, 12, 2024.

Monti, R., Bistacchi, A., Hussain, W., Herwegh, M., and Musso Piantelli, F.: Building a geological legend for 3D geomodelling in metamorphic belts, in: EGU General Assembly Conference Abstracts, EGU General Assembly 2025, Vienna, Austria, EGU25-16567, https://doi.org/10.5194/egusphere-egu25-16567, 2025.

Monti, R., Bistacchi, A., Hussain, W., Favaro, S., Herwegh, M., Drvoderić, S., Furlan, M., Piantelli, F. M., Dal Piaz, G., and Monopoli, B.: The Structural Topology model: integrating topology and ontology in 3D geological modelling, Copernicus Meetings, 2026.

Muller, D. E. and Preparata, F. P.: Finding the intersection of two convex polyhedra, Theor. Comput. Sci., 7, 217–236, https://doi.org/10.1016/0304-3975(78)90051-8, 1978.

Ni, H., Lin, X., Ning, X., and Zhang, J.: Edge detection and feature line tracing in 3D-point clouds by analyzing geometric properties of neighborhoods, Remote Sens., 8, 710, https://doi.org/10.3390/rs8090710, 2016.

Pearson, K.: LIII. *On lines and planes of closest fit to systems of points in space*, Lond. Edinb. Dublin Philos. Mag. J. Sci., 2, 559–572, https://doi.org/10.1080/14786440109462720, 1901.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

Pellerin, J., Botella, A., Bonneau, F., Mazuyer, A., Chauvin, B., Lévy, B., and Caumon, G.: RINGMesh: A programming library for developing mesh-based geomodeling applications, Comput. Geosci., 104, 93–100, https://doi.org/10.1016/j.cageo.2017.03.005, 2017.

Podgorney, R.: Utah FORGE: Data for 3-D Model Development - Lithology, Temperature, Pressure, and Stress, Geothermal Data Repository [dataset], https://doi.org/10.15121/1605155, 2020.

Ramsay, J. G.: Folding and Fracturing of Rocks, McGraw-Hill, 568 pp., 1967.

Regnier, G., Salinas, P., Jacquemyn, C., and Jackson, M. D.: Numerical simulation of aquifer thermal energy storage using surface-based geologic modelling and dynamic mesh optimisation, Hydrogeol. J., 30, 1179–1198, https://doi.org/10.1007/s10040-022-02481-w, 2022.

Roweis, S. T. and Saul, L. K.: Nonlinear dimensionality reduction by locally linear embedding, science, 290, 2323–2326, https://doi.org/10.1126/science.290.5500.2323, 2000.

Sambridge, M., Braun, J., and McQueen, H.: Geophysical parametrization and interpolation of irregular data using natural neighbours, Geophys. J. Int., 122, 837–857, https://doi.org/10.1111/j.1365-246X.1995.tb06841.x, 1995.

Serazio, C., Tamburini, M., Verga, F., and Berrone, S.: Geological surface reconstruction from 3D point clouds, MethodsX, 8, 101398, https://doi.org/10.1016/j.mex.2021.101398, 2021.

Sheffer, A., Praun, E., and Rose, K.: Mesh Parameterization Methods and Their Applications, Found. Trends® Comput. Graph. Vis., 2, 105–171, https://doi.org/10.1561/0600000011, 2006.

Shepard, D.: A two-dimensional interpolation function for irregularly-spaced data, in: Proceedings of the 1968 23rd ACM National Conference, New York, NY, USA, 517–524, https://doi.org/10.1145/800186.810616, 1968.

Shewchuk, J. R.: Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator, in: Applied Computational Geometry Towards Geometric Engineering, vol. 1148, edited by: Lin, M. C. and Manocha, D., Springer Berlin Heidelberg, Berlin, Heidelberg, 203–222, https://doi.org/10.1007/BFb0014497, 1996.

Si, H.: TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator, ACM Trans Math Softw, 41, https://doi.org/10.1145/2629697, 2015.

Sullivan, C. and Kaszynski, A.: PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK), J. Open Source Softw., 4, 1450, https://doi.org/10.21105/joss.01450, 2019.

Tenenbaum, J. B., Silva, V. de, and Langford, J. C.: A global geometric framework for nonlinear dimensionality reduction, science, 290, 2319–2323, https://doi.org/10.1126/science.290.5500.2319, 2000.

https://doi.org/10.5194/egusphere-2026-4815
Preprint. Discussion started: 4 September 2026

1376 Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski,
1377 E., Peterson, P., Weckesser, W., and Bright, J.: SciPy 1.0: fundamental algorithms for scientific
1378 computing in Python, Nat. Methods, 17, 261–272, https://doi.org/10.1038/s41592-019-0686-2,
1379 2020.

1380 Wellmann, F. and Caumon, G.: Chapter One - 3-D Structural geological models: Concepts,
1381 methods, and uncertainties, in: Advances in Geophysics, vol. 59, edited by: Schmelzbach, C.,
1382 Elsevier, 1–121, https://doi.org/10.1016/bs.agph.2018.09.001, 2018.

1383 Zehner, B., Börner, J. H., Görz, I., and Spitzer, K.: Workflows for generating tetrahedral meshes
1384 for finite element simulations on complex geological structures, Comput. Geosci., 79, 105–117,
1385 https://doi.org/10.1016/j.cageo.2015.02.009, 2015.

1386 Zhang, Y., Bajaj, C., and Sohn, B.-S.: 3D finite element meshing from imaging data, Comput.
1387 Methods Appl. Mech. Eng., 194, 5083–5106, https://doi.org/10.1016/j.cma.2004.11.026, 2005.

1388 Zimmermann, G., Tischner, T., Legarth, B., and Huenges, E.: Pressure-dependent Production
1389 Efficiency of an Enhanced Geothermal System (EGS): Stimulation Results and Implications for
1390 Hydraulic Fracture Treatments, Pure Appl. Geophys., 166, 1089–1106,
1391 https://doi.org/10.1007/s00024-009-0482-5, 2009.

1392 Zimmermann, G., Moeck, I., and Blöcher, G.: Cyclic waterfrac stimulation to develop an
1393 Enhanced Geothermal System (EGS)—Conceptual design and experimental results, Geothermics,
1394 39, 59–69, https://doi.org/10.1016/j.geothermics.2009.10.003, 2010.

1395