The Journal of Open Source Software

MyMesh: General purpose, implicit, and image-based meshing in Python

Timothy O. Josephson [1,2¶] **and Elise F. Morgan** [1,2,3]

1 Department of Biomedical Engineering, Boston University, United States **2** Center for Multiscale and Translational Mechanobiology, Boston University, United States **3** Department of Mechanical Engineering, Boston University, United States ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

A mesh is a discrete representation that subdivides a geometry or computational domain into a collection of points (nodes) connected by simple shapes (elements). Meshes are used for a variety of purposes, including simulations (e.g., finite element, finite volume, and finite difference methods), visualization and computer graphics, image analysis, and additive manufacturing. `mymesh` is a general purpose set of tools for generating, manipulating, and analyzing meshes. `mymesh` is particularly focused on implicit function and image-based meshing, with other functionality including:

- geometric and curvature analysis,
- intersection and inclusion tests (e.g., ray-surface intersection and point-in-surface tests),
- mesh boolean operations (intersection, union, difference),
- sweep construction methods (extrusions, revolutions),
- point set, mesh, and image registration,
- mesh quality evaluation and improvement, and
- mesh type conversion (e.g. volume to surface, hexahedral or mixed-element to tetrahedral, first-order elements to second-order elements).

State of the field

Mesh-based representations of geometries are essential in a wide variety of research applications, and as such, there is a need for robust, efficient, and easy-to-use software for creating, analyzing, and manipulating meshes. There are a variety of software packages for working with and generating meshes. Some are general purpose, like CGAL (The CGAL Project, 2025), VTK (Schroeder et al., 2006), and Gmsh (Geuzaine & Remacle, 2009), while others are more focused on specific tasks, such as triangular or tetrahedral mesh generation (e.g., Triangle (Shewchuk, 1996) and TetGen (Si, 2015), respectively). In Python (van Rossum, 1995), most meshing packages depend on (or are direct wrappers to) one or more of these libraries, such as PyVista (Sullivan & Kaszynski, 2019) (a Pythonic interface to VTK), pygalmesh (a Pythonic interface to CGAL) (Schlömer, 2021), nanomesh (an image-based meshing workflow tool that utilizes Triangle and TetGen) (Smeets et al., 2022), MeshPy (which interfaces to Triangle and TetGen) (Kloeckner et al., 2025), and PyMesh (which depends on CGAL, Triangle, TetGen, and others) (Zhou, 2019). While these interfaces are useful and provide access to powerful mesh generation tools, their reliance on external dependencies can make them less easy to use and limit code readability, making it more difficult to understand how the code works. TriMesh (Dawson-Haggerty, 2026) stands out as a capable, pure-Python library focused on triangular surface meshes, but it isn't intended for use with quadrilateral, mixed-element, or volumetric meshes. Given the intended focus and/or design philosophies of these existing softwares, it was determined that building `mymesh`, rather than making contributions to existing software,

DRAFT

was the best way to achieve a full-featured, accessible, and easy to use Python package for creating and working with meshes.

Statement of need

`mymesh` strives to meet the need for a library of meshing tools, written in Python, with clear documentation that makes it both easy to use and easy to understand. `mymesh` has a particular focus on implicit function and image-based meshes, but also supplies a wide variety of general purpose tools. Rather than wrapping other libraries, algorithms are implemented from scratch, often based on or inspired by published algorithms and research. By providing an easily usable interface to both high-level and low-level functionality, we hope to provide both complete solutions and a set of building blocks for the development of other mesh-related tools.

Research impact statement

`mymesh` was originally developed in support of research within the Skeletal Mechanobiology and Biomechanics Lab at Boston University. It was used extensively in the scaffold design optimization research by Josephson & Morgan (2024) and is currently being used within multiple labs and institutions (e.g., Lim, M., pers. comm., Mtchedlishvili, M., pers. comm.), for various ongoing projects including vertebral modeling, hip fracture modeling (Olowu et al., 2026), growth modeling of skeletal tissue, and analysis of objects and biological tissues imaged using micro-computed tomography (μCT). `mymesh` has proven useful in a variety of research applications, well beyond those that inspired its original development, and we expect it to remain a valuable tool in future research efforts.

Software design

The `mymesh` package is designed around meshes defined by two fundamental components, the coordinates of nodes (`NodeCoords` or `points`) and the connectivity of those nodes to form elements (`NodeConn` or `cells`). These components are stored in the `mesh` object, which contains a variety of convenience functions and cached properties (e.g., `Centroids`, `NodeNormals`) that can be calculated on-demand and stored for future use. `mymesh` was developed from the beginning to support various element types and mixed-element meshes, so the node connectivity can be defined as either a `numpy` (Harris et al., 2020) array or a non-rectangular list of lists, with the code designed to take advantage of the added efficiency of `numpy` arrays when possible without being reliant on them in a way that would prohibit mixed-element meshes.

In addition to overall ease of use, the framework of `mymesh` was designed to be easy to get into and out of, so that users can easily utilize the strengths and benefits of other code or software. The `mesh` object facilitates conversion to the data structures of two other popular meshing softwares, `meshio` (Schlömer, 2024) and `pyvista`, and, through `meshio`, facilitates the reading and writing of the mesh to and from many different file formats (`mesh.read(filename)`, `mesh.write(filename)`). Additionally, most low-level functions in `mymesh` operate on just the node coordinates and connectivity, making it easy for users of other software/packages to directly utilize individual functions, without needing to convert to `mymesh`'s `mesh` data structure.

Python was chosen as the programming language for `mymesh` because of its popularity in computational research and its focus on factors such as simplicity and readability. Many other languages, such as Matlab (The MathWorks Inc., 2026) and Julia (Bezanson et al., 2017), as well as finite element softwares, such as Abaqus (Dassault Systèmes Simulia Corp., 2026) and FEniCS (Baratta et al., 2023), interface with Python, extending the value of `mymesh` beyond Python users. While Python is often regarded as relatively inefficient compared to other languages, vectorization with `numpy` and just-in-time compilation with `numba` (Lam et al.,

DRAFT

2026) are used in performance-critical operations to achieve efficiency competitive with other languages.

Features and Examples

A key focus of `mymesh`, and part of the original motivation for its development, is meshing of implicit functions. Implicit functions take the form $f(x, y, z) = 0$, with 0 indicating the surface of an object and, by convention, negative values indicating the inside of an object. Geometries described by these functions, such as those representing triply periodic minimal surfaces, cannot always be generated in traditional, parametric, computer aided design (CAD) softwares. For example, the implicit function representation of the Fischer-Koch S surface (Figure 1.a, Fischer & Koch (1987), von Schnering & Nesper (1991)) is

$$f(x, y, z) = \cos(2x)\sin(y)\cos(z) + \cos(x)\cos(2y)\sin(z) + \sin(x)\cos(y)\cos(2z) = 0.$$

Triangular surface meshes and tetrahedral volume meshes can be generated from implicit functions by using contouring approaches like marching cubes (Lorensen & Cline, 1987) and marching tetrahedra (Bloomenthal, 1994). Implicit meshing approaches can also be used for boolean operations to merge or modify different shapes (Figure 1.b). Many of the same approaches used for implicit mesh generation can also be applied to image-based mesh generation, which is useful for visualizing, modeling, and analyzing objects captured by imaging techniques such as CT scans (Figure 2).

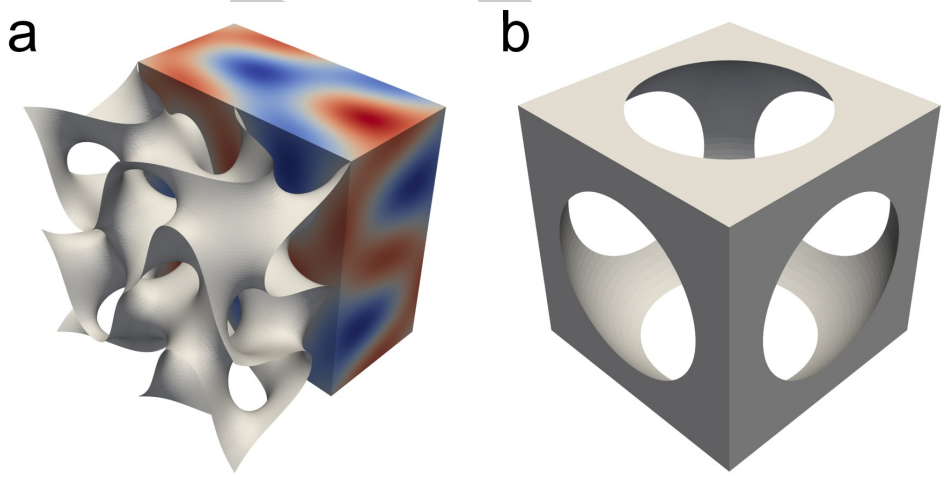

Figure 1: Examples of implicit mesh generation: (a) the Fischer-Koch S TPMS surface shown as both a function evaluated over a domain and the meshed surface at $f(x, y, z) = 0$ and (b) a geometry constructed by subtracting an implicit representation of a sphere from a cube.

DRAFT

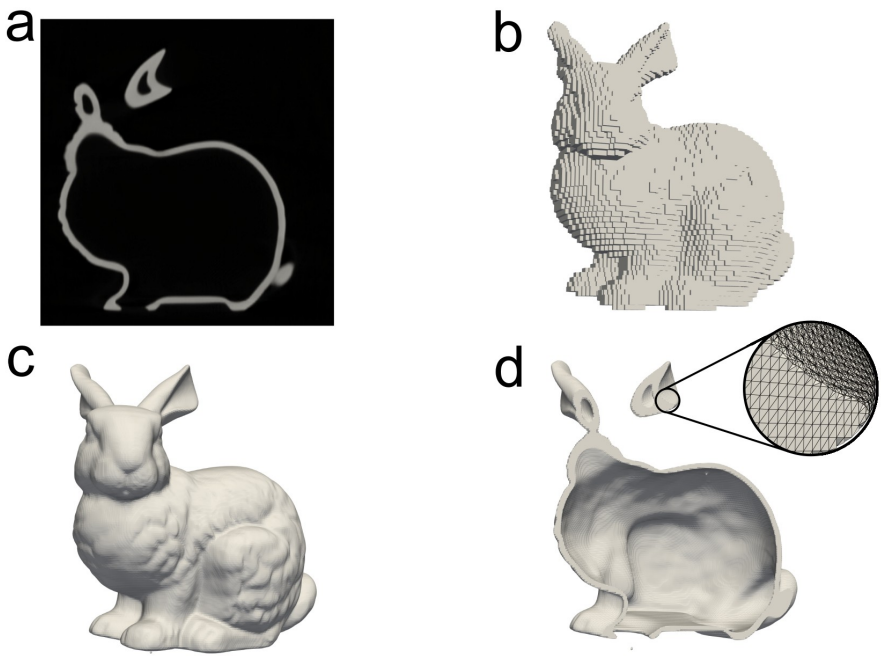

Figure 2: Image-based meshing of the CT-scanned Stanford Bunny (Stanford University Computer Graphics Laboratory, 2026; Turk & Levoy, 1994): (a) One mid-plane of the 3D image, (b) a coarsened voxel mesh, (c) a triangular surface mesh, and (d) a cross-sectional view of a tetrahedral volume mesh. A zoomed in region shows the mesh edges to illustrate the arrangement of the tetrahedral mesh.

While implicit and image-based meshing is a focus of `mymesh`, it is not the only functionality. `mymesh` has a variety of low-level capabilities, like determining node/element connectivity and adjacency information, calculating surface normal vectors, and conversion between meshes of different types, which can be useful building blocks for more complex meshing algorithms. `mymesh` also possesses capabilities for geometric analysis (such as surface curvature calculation, Figure 3.a), mesh refinement, coarsening, and/or quality improvement (Figure 3.b), registration or alignment of meshes and images, and contouring/thresholding (Figure 3.c). In addition to the capabilities of the software itself, the documentation features a theory guide intended as an educational resource to help those who are curious to understand the algorithms and approaches used by `mymesh`.

DRAFT

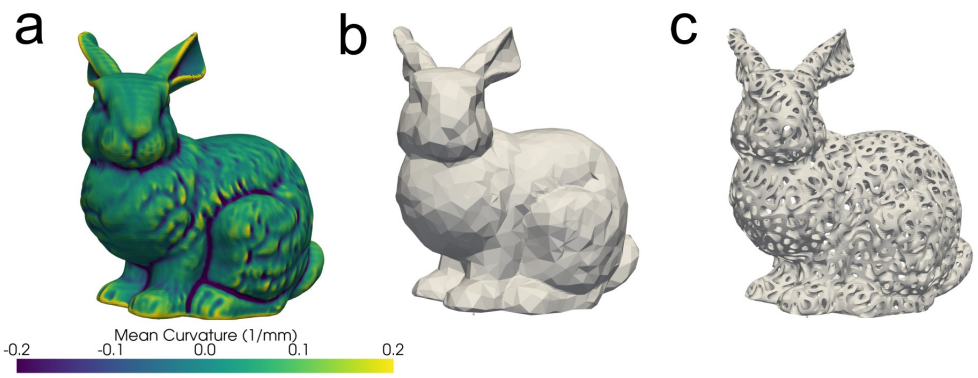

Figure 3: Examples of additional capabilities of `mymesh`: (a) Mean curvature calculated on the surface of the Stanford bunny, (b) the Stanford bunny coarsened from 504k triangles (Figure 2.c) to 15.5k triangles, (c) the Stanford bunny contoured by a thickened version of the Fischer-Koch S TPMS (Figure 1.a).

License & Availability

`mymesh` is distributed under the MIT license. It is available on PyPI and GitHub, and is archived on Zenodo. The documentation provides guides for getting started, examples, and detailed usage information for each function.

AI usage disclosure

Generative AI was not used to write this paper, the documentation, or the functionality of `mymesh`. Initial development of `mymesh` began in the summer of 2021, before the release of OpenAI's ChatGPT (Nov. 30, 2022) and the widespread proliferation of powerful generative AI chatbots. While generative AI was never used to generate code for `mymesh`, it was in some instances consulted alongside other resources (e.g., scientific literature, StackExchange). Generative AI has been used in the following ways throughout the development of `mymesh`:

- as a resource for some mesh-specific and general-purpose programming concepts, such as methods for improving efficiency of certain operations,
- assistance in setting up packaging infrastructure (e.g., pyproject.toml, GitHub workflows), and
- conceptualization of test cases for some unit tests.

CRediT Author Statement

Timothy O. Josephson: Conceptualization, Software, Methodology, Writing - Original Draft. **Elise F. Morgan:** Conceptualization, Supervision, Resources, Funding acquisition, Writing - Review & Editing

Acknowledgements

This work was developed with funding support from the National Institutes of Health (Grant #AG073671). We are additionally grateful to all of the users who have tested the code, reported bugs, requested features, and provided feedback which has been vital to the development of `mymesh`.

DRAFT

References

Baratta, I. A., Dean, J. P., Dokken, J. S., Habera, M., Hale, J. S., Richardson, C. N., Rognes, M. E., Scroggs, M. W., Sime, N., & Wells, G. N. (2023). *DOLFINx: The next generation FEniCS problem solving environment*. Zenodo. https://doi.org/10.5281/zenodo.10447665

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*, 65–98. https://doi.org/10.1137/141000671

Bloomenthal, J. (1994). An implicit surface polygonizer. *Graphics Gems*, 324–349. https://doi.org/10.1016/b978-0-12-336156-1.50040-9

Dassault Systèmes Simulia Corp. (2026). *Abaqus analysis user's manual, version 2026*.

Dawson-Haggerty, M. (2026). *Trimesh*. https://trimesh.org/

Fischer, W., & Koch, E. (1987). On 3 periodic minimal surfaces. *Zeitschrift Fur Kristallographie - New Crystal Structures*, *52*, 31–52. https://doi.org/10.1524/zkri.1987.179.1-4.31

Geuzaine, C., & Remacle, J. F. (2009). Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, *79*, 1309–1331. https://doi.org/10.1002/nme.2579

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

Josephson, T. O., & Morgan, E. F. (2024). Mechanobiological optimization of scaffolds for bone tissue engineering. *Biomechanics and Modeling in Mechanobiology*, 1–18. https://doi.org/10.1007/S10237-024-01880-0

Kloeckner, A., Brun, L., Liu, B., Klemenc, S., Fkikl, A., Gohlke, C., Coon, E., Oxberry, G., Veselý, J., Wala, M., Smith, M., Potrowl, P., & Kurtz, A. (2025). *MeshPy*. Zenodo. https://doi.org/10.5281/zenodo.7296459

Lam, S. K., stuartarchibald, Pitrou, A., Florisson, M., Markall, G., Seibert, S., Self-Construct, E., Anderson, T. A., Leobas, G., rjenc29, Kaustubh, Patel, S., Collison, M., luk-f-a, Bourque, J., Meurer, A., Oliphant, T. E., Wang, M., densmirn, … MattyG. (2026). *Numba*. Zenodo. https://doi.org/10.5281/zenodo.4343230

Lorensen, W. E., & Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987*, *21*, 163–169. https://doi.org/10.1145/37401.37422

Olowu, B. D., Auger, J. D., & Morgan, E. F. (2026). Can proton density-weighted MRI-based finite element predict bone strength? *Bone (In Review)*.

Schlömer, N. (2021). *Pygalmesh: Python interface for CGAL's meshing tools*. Zenodo. https://doi.org/10.5281/zenodo.5564818

Schlömer, N. (2024). *Meshio: Tools for mesh files*. https://doi.org/10.5281/zenodo.1173115

Schroeder, W., Martin, K., & Lorensen, B. (2006). *The visualization toolkit (4th ed.)*. Kitware. ISBN: 978-1-930934-19-1

Shewchuk, J. R. (1996). Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. *Workshop on Applied Computational Geometry*, 203–222. https://doi.org/10.1007/bfb0014497

Si, H. (2015). TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions*

DRAFT

on Mathematical Software, *41*. https://doi.org/10.1145/2629697

Smeets, S., Renaud, N., & Willenswaard, L. J. C. van. (2022). Nanomesh: A Python workflow tool for generating meshes from image data. *Journal of Open Source Software*, *7*, 4654. https://doi.org/10.21105/joss.04654

Stanford University Computer Graphics Laboratory. (2026). *The Stanford 3D scanning repository: Stanford bunny*. https://graphics.stanford.edu/data/3Dscanrep/

Sullivan, C., & Kaszynski, A. (2019). PyVista: 3D plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK). *Journal of Open Source Software*, *4*, 1450. https://doi.org/10.21105/joss.01450

The CGAL Project. (2025). *CGAL user and reference manual* (6.1 ed.). CGAL Editorial Board. https://doc.cgal.org/6.1/Manual/packages.html

The MathWorks Inc. (2026). *MATLAB version: 26.1 (R2026a)*. The MathWorks Inc. https://www.mathworks.com

Turk, G., & Levoy, M. (1994). *Zippered polygon meshes from range images*. 311–318. https://doi.org/10.1145/192161.192241

van Rossum, G. (1995). *Python tutorial, technical report CS-R9526*. Centrum voor Wiskunde en Informatica (CWI).

von Schnering, H. G., & Nesper, R. (1991). Nodal surfaces of Fourier series: Fundamental invariants of structured matter. *Zeitschrift Für Physik B Condensed Matter*, *83*, 407–412. https://doi.org/10.1007/BF01313411

Zhou, Q. (2019). *Pymesh—geometry processing library for Python*. https://github.com/PyMesh/PyMesh

DRAFT