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

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 & 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,
- mesh type conversion (e.g. volume to surface, hexahedral or mixed-element to tetrahedral, first-order elements to second-order elements).

Statement of need

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, 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), MeshPy (which interfaces to Triangle and TetGen), and PyMesh (which depends on CGAL, Triangle, TetGen, and others). 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, 2019) 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, was the best way to achieve a full-featured, accessible, and easy to use Python package for creating and working with meshes.

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

Research impact statement

50 mymesh was originally developed in support of research within the Skeletal Mechanobiology
51 and Biomechanics Lab at Boston University. It was used extensively in the scaffold design
52 optimization research by Josephson & Morgan (2024) and is currently being used in various
53 ongoing projects within multiple labs and institutions, including vertebral modeling, hip fracture
54 modeling, growth modeling of skeletal tissue, and analysis of objects imaged using micro-
55 computed tomography (μCT). mymesh has proven useful in a variety of research applications,
56 well beyond those that inspired its original development, and we expect it to remain a valuable
57 tool in future research efforts.

Software design

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

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

76 Python was chosen as the programming language for mymesh because of its popularity in
77 computational research and focus on factors such as simplicity and readability. Many other
78 languages, such as Matlab and Julia, as well as finite element softwares, such as Abaqus and
79 FEniCS, interface with python, extending the value of mymesh beyond Python users. While
80 Python is often regarded as relatively inefficient compared to other languages, vectorization
81 with numpy and just-in-time compilation with numba are used in performance-critical operations
82 to achieve efficiency competitive with other languages.

Features and Examples

84 A key focus of mymesh, and part of the original motivation for its development, is meshing
85 of implicit functions. Implicit functions take the form $f(x, y, z) = 0$, with 0 indicating the
86 surface of an object and, by convention, negative values indicating the inside of an object.
87 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), 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).
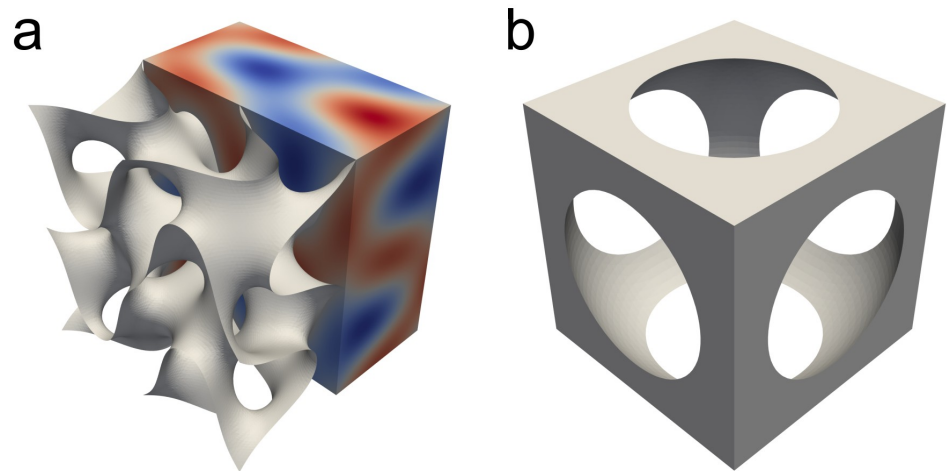

Figure 1: Examples of implicit mesh generation: (a) the Fischer-Koch S TPMS surface shown as both a function evaluated over a domain and the meshed surface at $f(x, y, z) = 0$ and (b) a geometry constructed by subtracting an implicit representation of a sphere from a cube.
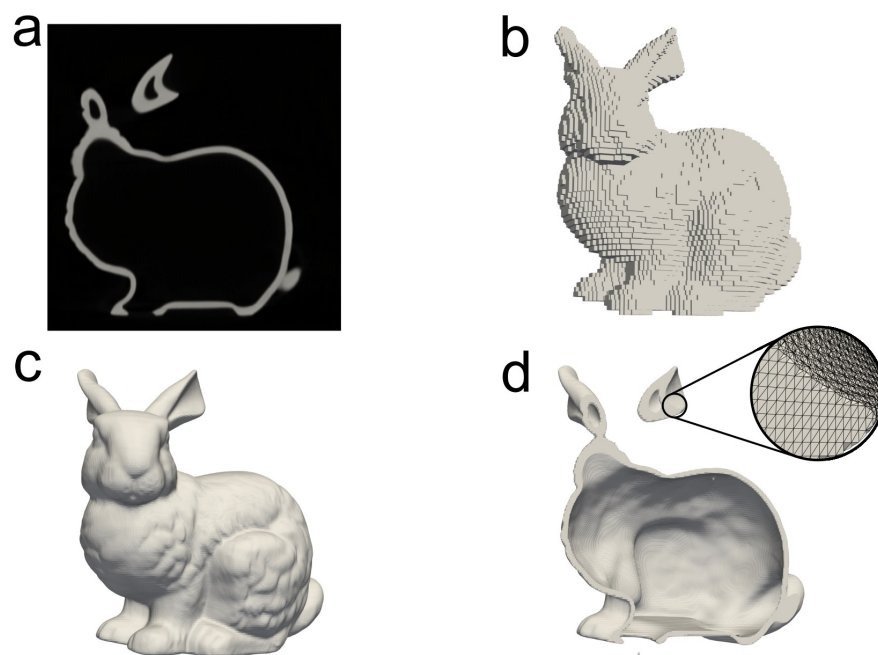
Figure 2: Image-based meshing of the CT-scanned Stanford Bunny (The Stanford volume data archive): (a) One mid-plane of the 3D image, (b) a coarsened voxel mesh, (c) a triangular surface mesh, and (d) a cross-sectional view of a tetrahedral volume mesh.

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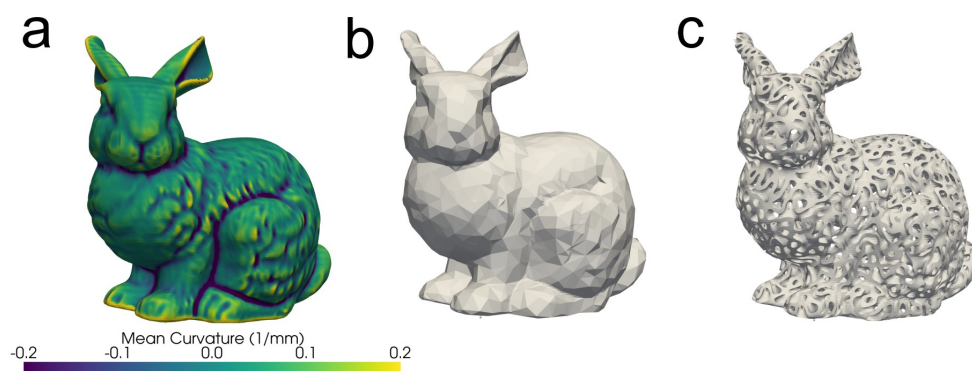107 used by mymesh.

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 in the writing of mymesh code or this paper. 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 the 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),
- assistance in the creation of test cases for some unit tests.

Acknowledgements

References

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

Dawson-Haggerty. (2019). *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*(11), 1309–1331. https://doi.org/10.1002/nme.2579

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

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*(4), 163–169. https://doi.org/10.1145/37401.37422

Schnering, H. G. von, & Nesper, R. (1991). Nodal surfaces of Fourier series: Fundamental invariants of structured matter. *Zeitschrift für Physik B Condensed Matter*, *83*(3), 407–412. https://doi.org/10.1007/BF01313411

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 on Mathematical Software*, *41*(2). https://doi.org/10.1145/2629697

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*(37), 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