

# automesh: Automatic mesh generation in Rust

Chad B. Hovey  <sup>1\*</sup> and Michael R. Buche 

1 Sandia National Laboratories, United States \* These authors contributed equally.

DOI: [10.21105/joss.08768](https://doi.org/10.21105/joss.08768)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

---

Editor: [Henrik Finsberg](#)  

## Reviewers:

- [@jeremylt](#)
- [@vijaysm](#)

Submitted: 06 May 2025

Published: 04 November 2025

## License

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

## Summary

automesh is an open-source Rust software program that uses a **segmentation**, typically generated from a 3D image stack, to create a finite element **mesh**, composed either of hexahedral (volumetric) or triangular (isosurface) elements. automesh **converts** between segmentation formats (.npy, .spn) and mesh formats (.exo, .inp, .mesh, .stl, .vtk). automesh can **defeature** voxel domains, apply Laplacian and Taubin **smoothing**, and output mesh quality **metrics**. automesh uses an internal octree for fast performance.

## Introduction

Computed Tomography (CT), Magnetic Resonance Imaging (MRI), and other similar imaging modalities that stack a sequence of 2D digital images (composed of pixels) to create a 3D voxel (volumetric) representation of a subject have revolutionized the fields of engineering and medicine by providing non-invasive and non-destructive insights into the internal structures of complex systems.

- In the medical field, 3D voxel data sets are instrumental for visualizing internal organs and soft tissues, facilitating accurate diagnoses and enhancing surgical planning.
- In dentistry, they play a vital role in jaw surgery and the placement of dental implants.
- Biomedical engineering benefits include fit and integrity of surgical implants and prosthetic devices.
- In biomechanical engineering, the human digital twin can be used in injury risk prediction in environments not suited for human surrogate models; for example, automotive barrier crash testing and military environments that have blast and ballistic exposure.
- In materials engineering, 3D imaging is employed to analyze the internal structures of materials, enabling the detection of voids, inclusions, and defects.
- Aerospace engineering utilizes these techniques to inspect critical components, such as turbine blades, ensuring their structural integrity.
- Civil and geotechnical engineering applications include imaging voids, rebar, and anomalies in concrete structures, as well as characterizing subsurface geology through seismic reflection and refraction.
- Petroleum engineering applications include characterization of reservoir rocks and fluids, providing insights into porosity and permeability.

This compendium of applications underscores the versatility and significance of 3D voxel data sets derived from CT and MRI as a fundamental ingredient of model development for patient-specific, site-specific, or as-built geometry. This concept is referred to as a *digital twin*. The geometric aspects of a digital twin involve image processing,<sup>1</sup> and in particular,

<sup>1</sup>Image processing subtopics can include acquisition details (exposure time, magnification, resolution), pre-processing (calibration, distortion control, noise reduction), stitching (feature detection, feature matching, homography estimation, warping, blending), baseline correction (background subtraction, normalization) and post-processing (image enhancement, downscaling, and segmentation).

segmentation.

Segmentation is the process of classifying each voxel in a data set as belonging to a specific class within the geometric domain, typically as a unique material, air, or void. We use a segmented 3D voxel data set as our start point. We seek to transform the voxel set into a finite element mesh suitable for numerical analysis, our end point.

The most direct way to undertake this conversion is to create a one-to-one map from voxel to hexahedral element. This rudimentary approach, while attractive due to its effectiveness and simplicity, can result in billions to trillions of elements that can overwhelm current-day solvers, create massive input/output bottlenecks, and result in excessive consumption of digital storage resources.

To achieve a more tractable number of elements, on the order of millions, **downsampling** of the original image data can be used to decrease the effective resolution, resulting in fewer voxels. Additionally, **adaptivity** of the finite element mesh, allowing non-uniform element sizes to span several length scales, can be used.

## Statement of Need

Our endeavor to create a finite element mesh suitable for analysis from a segmentation has ten specific requirements. These requirements arise from the unique character of our mission space, collaborators, and sponsors. The first requirement comes from our start point. The next five requirements stem from our end point. The remaining four requirements follow from the workflow, converting a segmentation into a mesh.

We list each of these requirements below:

- (1) **Segmentation start point.** Many finite element mesh applications use geometric primitives (e.g., tessellations) or analytical geometry (e.g., splines, NURBS, isosurfaces) as a start point. In contrast, our start point is a segmentation composed of a 3D stack of voxels.
- (2) **All-hex mesh.** While tetrahedral finite elements offer excellent adaptivity characteristics, hexahedral finite elements offer superior performance characteristics. Moreover, higher-order tetrahedral elements, such as the 10-noded composite formulation, can result in a very large number of degrees of freedom, potentially overwhelming solver and serialization capabilities. In contrast, 8-noded hexahedral elements provide an economy of degrees of freedom and can be underintegrated to offer excellent performance with incompressible materials, so long as hourglass energy modes are monitored and controlled.
- (3) **Multi-material.** The mesh must be able to accommodate numerous materials throughout the domain. For example, mesh generation for microstructures that have prolific grain boundaries must be handled.
- (4) **Multi-scale.** The characteristic length scale of the features we seek to resolve can be significantly smaller than the length scales of the overall domain. Mesh adaptivity can be helpful to reconcile these differences of scale.
- (5) **Quality.** Elements must retain sufficient quality to ensure use with numerical computation. Common quality metrics include minimum scaled Jacobian, skew, edge ratio, and minimum angle. Poor element quality can lead to inaccurate results and convergence issues.
- (6) **Input/Output Format.** The segmentation input format must support both NumPy ([NumPy Developers, 2025](#)) and SPN ([Sandia National Laboratories, 2025](#)). The mesh output must support both ExodusII ([Schoof & Yarberry, 1994](#)) and ABAQUS ([Dassault Systèmes, 2025](#)).
- (7) **Automated.** Creating finite element meshes from image stacks can be a time-consuming task, often representing the biggest bottleneck in an analyst's workflow. The workflow

from segmentation to mesh should be completely automated, without the need for human intervention.

- (8) **Fast.** The program must be as fast or faster than other available mesh generation software.
- (9) **Open-source.** All the source code must be freely available as open-source software.
- (10) **Professional.** The application source code must reside on GitHub, Gitlab, or an equivalent platform, have a comprehensive test suite to assure quality and reproducibility, use continuous integration and continuous delivery principles (CI/CD), and be supported by the authors.

## Existing Art

While there are many applications suitable for creation of finite element meshes, we found none that could meet all of our requirements.

Commercial (closed-source) applications such as Altair HyperMesh ([Altair Engineering, Inc., 2025](#)), ANSYS Meshing ([ANSYS, Inc., 2025](#)), COMSOL Multiphysics ([COMSOL, Inc., 2025](#)), Cubit ([Sandia National Laboratories, 2025](#)), Gridgen ([Pointwise, Inc., 2025](#)), Hexotic ([INRIA Gamma Team, 2023; Maréchal, 2009, 2016](#)), Sculpt ([Owen et al., 2014; Owen & Shelton, 2014](#)), Siemens Simcenter ([Siemens AG, 2025](#)), and TrueGrid ([TrueGrid, 2025](#)) were not suitable due to our open-source requirement.

While existing open-source software packages provide valuable features for finite element mesh generation, they often lack one or more of our specific requirements, particularly in terms of segmentation input, all-hex meshing, and comprehensive input/output format support. We found specific limitations as follows:

- **cinolib** supports all-hex meshing, but it does not support NumPy, SPN, ExodusII, and ABAQUS file types ([Livesu, 2019](#)). Additionally, for meshing cases that rely on integer linear programming ([Pitzalis et al., 2021](#)), **cinolib** depends on Gurobi, which is closed-source, commercial software ([Gurobi Optimization, LLC, 2025](#)).
- **FEniCS** supports segmentation, but does not create an all-hex only mesh. It does not support SPN and ABAQUS file types ([FEniCS Project Developers, 2025](#)).
- **FreeFEM** supports segmentation, but primarily focuses on geometric primitives. It does not support all-hex meshing. It does not support NumPy or SPN input formats. It can output to ExodusII but not to ABAQUS ([FreeFEM Developers, 2025](#)).
- **GetFEM++** supports segmentation, but does not produce a mesh that is all-hex. It does not have NumPy or SPN support. It supports ExodusII but not ABAQUS outputs ([GetFEM Developers, 2025](#)).
- **GMSH** primarily uses geometric primitives, with limited support for voxel-based segmentation. It supports all-hex meshing, but not exclusively. It lacks direct NumPy and SPN input formats. It can output to ExodusII but not to ABAQUS ([GMSH Developers, 2025](#)).
- **MeshLab** supports segmentation, but not all-hex meshing. It has limited multi-material support and no mesh adaptivity. It does not support NumPy, SPN, ExodusII, and ABAQUS file types ([ISTI-CNR, 2025](#)).
- **NETGEN** is an automatic tetrahedral mesh generator. It supports neither segmentation input nor all-hex mesh generation. It does not support NumPy and SPN file types ([Netgen Developers, 2025](#)).

- OpenFOAM primarily uses geometric definitions, has limited voxel support, and does not support exclusively all-hex meshing. It does not support NumPy, SPN, and ABAQUS file types. ([OpenFOAM Foundation, 2025](#)).
- TetGen primarily works with geometric input, such as points, edges, and faces, and creates tetrahedral elements. It supports neither segmentation input nor all-hex mesh generation. It does not support NumPy, SPN, and ExodusII file types ([Si, 2025](#)).

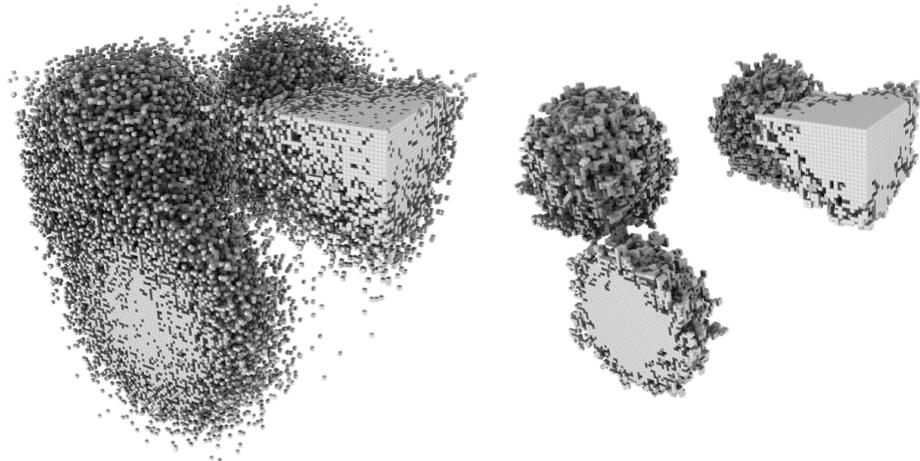
## Features and Implementation

automesh is available for download from [crates.io](#), the official package registry for the Rust programming language. A Python interface is also available from [PyPI](#). The [User Guide](#) introduces the segmentation-to-mesh workflow, covers software installation, and contains examples based on unit tests and larger analysis sets. Source code can be obtained from [GitHub](#) ([Hovey & Buche, 2025](#)).

One significant attribute of automesh is its implementation in Rust, a modern programming language known for its performance and memory safety. Rust's ownership model prevents common programming errors such as null pointer dereferencing and buffer overflows, ensuring that automesh operates reliably and securely. Furthermore, Rust's zero-cost abstractions allow automesh to deliver high performance without sacrificing code readability or maintainability. The language's strong type system and concurrency features also enhance the robustness, making it well-suited for handling complex computational tasks efficiently ([Klabnik & Nichols, 2019](#)).

automesh is used via its command line interface. automesh directly converts voxels classified by a non-negative integer in the segmentation input file into a hexahedral finite element mesh. Segmented classes can be arbitrarily eliminated from the resultant mesh. For example, air or void surrounding a subject, often segmented with integer 0, can be eliminated, which can significantly reduce the resultant mesh size and complexity.

**Defeaturing** is accomplished by specifying a voxel threshold. A cluster of voxels, defined as two or more voxels that share a face (edge and node sharing do not constitute a cluster) with count at or above the threshold will be preserved, whereas a cluster of voxels with a count below the threshold will be eliminated through resorption into the surrounding material. Figure 1 shows an example of a mesh before and after defeaturing.

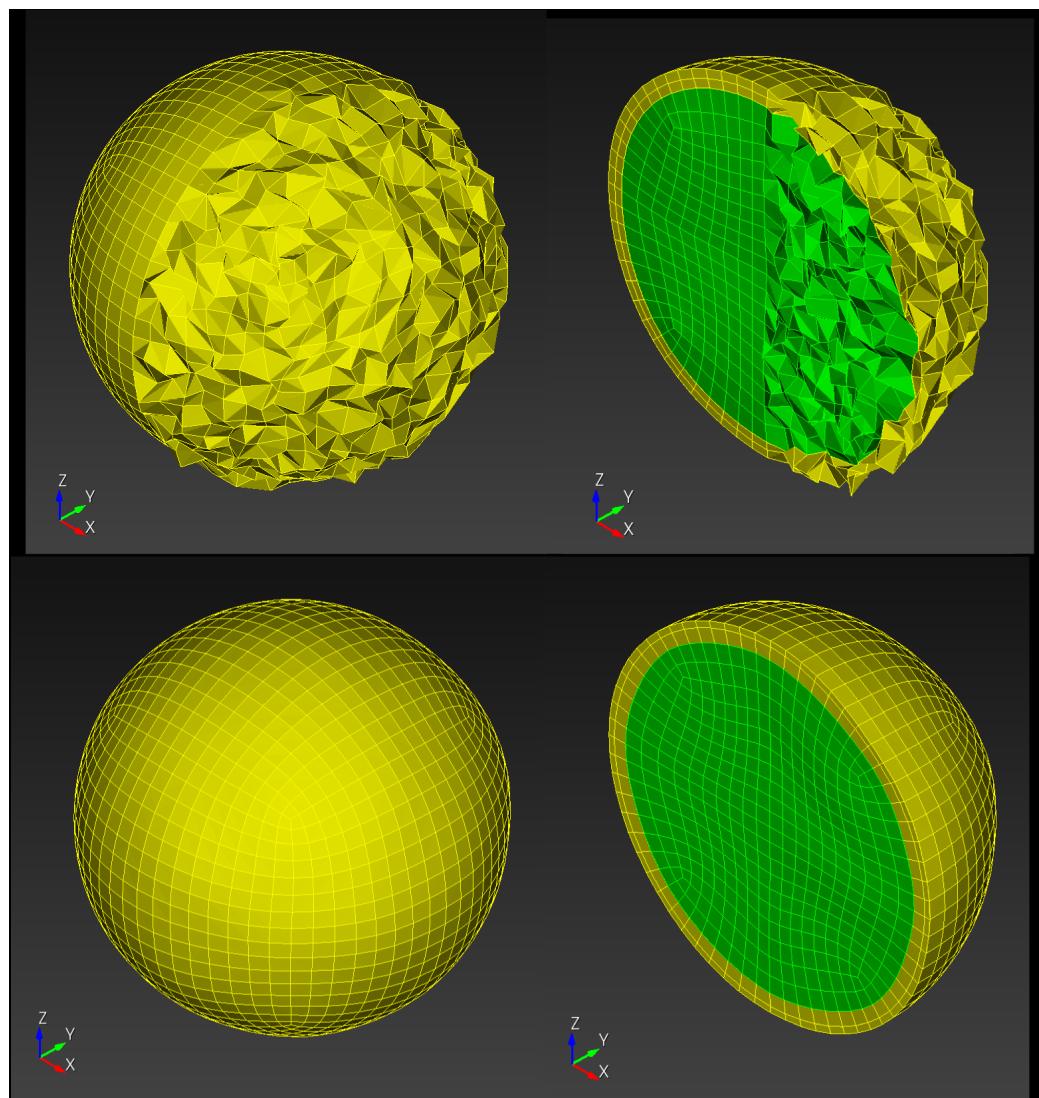


**Figure 1:** Example of defeaturing: (left) mesh prior to defeaturing, (right) mesh after defeaturing.

automesh provides both Laplace ([Sorkine, 2005](#)) and Taubin ([Taubin, 1995](#)) formulations for

mesh **smoothing**, with optional hierarchical control. Hierarchical control classifies all nodes as prescribed, boundary, or interior. With hierarchical control, the updated (smoothed) position of a boundary node is influenced only by prescribed nodes and other boundary nodes; whereas, the updated position of an interior node is influenced by all types of nodes (prescribed, boundary, and interior). Smoothing with hierarchical control can help maintain domain boundaries better than smoothing alone ([Chen & Ostoja-Starzewski, 2010](#)).

Figure 2 shows automesh results of Taubin smoothing on an all-hexahedral mesh in the shape of a sphere (green) with a concentric shell (yellow). Similar to the original demonstration of Taubin smoothing on a tessellation ([Taubin, 1995](#)), we apply Taubin smoothing to a hexahedral domain, with noise added to the  $x > 0$  hemisphere.

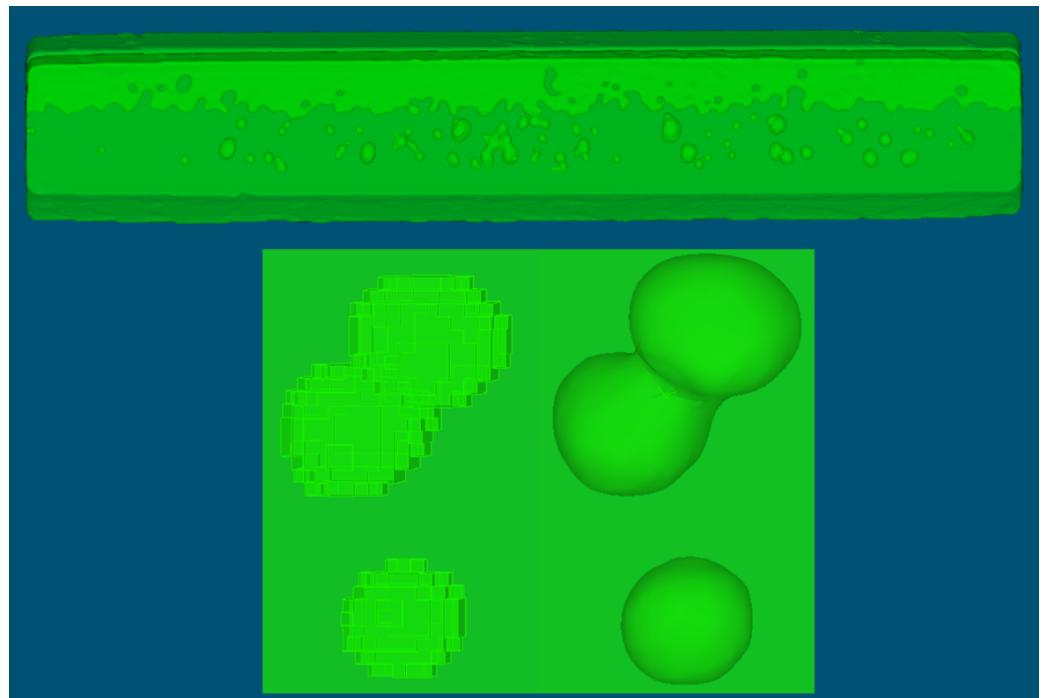


**Figure 2:** Example of Taubin smoothing applied to a volume mesh: (top left) isometric view of mesh with noised  $x > 0$  hemisphere, (top right) noised domain with cut plane, (bottom left) smoothed mesh after 200 iterations, (bottom right) smoothed mesh with cut plane.

Unlike Laplace smoothing, which drastically reduces volume, Taubin smoothing nearly preserves volumes. For the example in Figure 2, Laplace reduced the volume by 16% in 10 iterations; whereas, Taubin increased the volume by 1% in 200 iterations, while effectively smoothing. With Taubin, the noise in the  $x > 0$  hemisphere was removed, with a very small volumetric

change. The  $x < 0$  hemisphere did not degrade from its original configuration. Notice the smoothing capability of automesh is available for any .inp or .stl input format, regardless of whether the mesh originated from a segmentation.

Surface reconstruction of a 3D, all-triangular mesh **isosurface** can also be created from a segmentation. Figure 3 shows a two-material laser weld segmentation, composed of approximately 6.7 million voxels (Karlson et al., 2023; Polonsky et al., 2023). The workflow used defeathering and smoothing to create an .stl isosurface output composed of 762,396 facets.



**Figure 3:** Surface reconstruction of a real weld: (top) isosurface mesh generated from a CT segmentation, (bottom) example of voxel domain (left) used to generate a smooth isosurface (right).

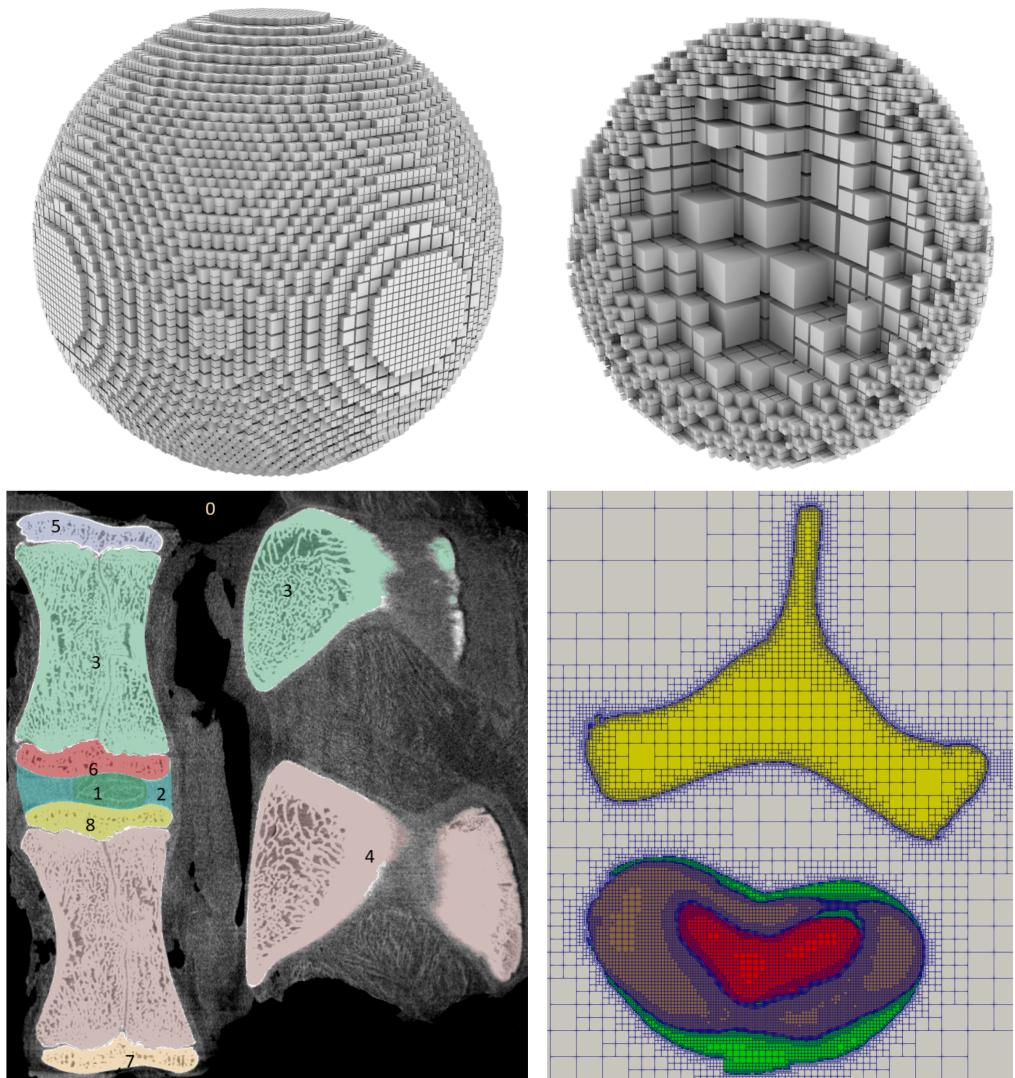
automesh uses an **octree** for efficient performance (Meagher, 1980). The octree serves as an adaptive segmentation, which accelerates defeathering. Our current implementation requires the octree to be strongly balanced, not weakly balanced.

Octree balancing refers to how much neighboring cells can differ in their level of refinement and the type of adjacency (face, edge, vertex) that is considered (Livesu et al., 2021):

- A **strongly balanced** octree requires neighboring cells that share a face, edge, or vertex to differ by no more than one level of refinement.
- A **weakly balanced** octree requires neighboring cells that share a face to differ by no more than one level of refinement, while edge and vertex neighbors may differ by more than one level of refinement.

The top two items of Figure 4 show a visualization in HexaLab (Bracci et al., 2019) of an automesh octree composing a spherical domain. The cut plane on the right item exposes the octree adaptivity. Four levels of cell refinement are present.

The bottom two items of Figure 4 show application of automesh to a large problem, illustrating the importance of the octree implementation. The micro CT segmentation input is composed of one billion voxels, representable in an octree with only 10 million cells. Five million of those octree cells are removable void. With this  $200\times$  reduction, only 36 seconds is required to create the mesh.



**Figure 4:** Application of the automesh octree: (top left) octree mesh of a spherical domain, (top right) cut plane through the domain to expose the adaptivity of the octree, (bottom left), sagittal view of a micro CT of a spinal unit, courtesy of ([Nicolella et al., 2025](#)) (used with permission) with segmentation IDs 0 through 8, (bottom right) axial view of automesh octree, used to create a finite element mesh from the segmentation.

## Conclusion

automesh presents a robust solution for automatic mesh generation from segmentation data, bridging the gap between 3D imaging modalities and finite element analysis. By leveraging defeatureing, smoothing, and an efficient octree structure, automesh not only enhances the quality of the generated meshes but also significantly reduces computational overhead. Its ability to handle multiple input and output formats, including support for both hexahedral and triangular elements, makes it versatile for various applications across engineering and biomedical fields. The integration of automated workflows ensures that users can efficiently convert complex voxel data into high-quality meshes without manual intervention.

## Acknowledgements

This work was supported by the Office of Naval Research (Dr. Timothy Bentley) under grant N0001424IP00025.

## References

- Altair Engineering, Inc. (2025). *Altair HyperMesh*. <https://www.altair.com/hypermesh/>
- ANSYS, Inc. (2025). *ANSYS meshing*. <https://www.ansys.com/products/meshing/ansys-meshing>
- Bracci, M., Tarini, M., Pietroni, N., Livesu, M., & Cignoni, P. (2019). HexaLab.net: An online viewer for hexahedral meshes. *Computer-Aided Design*, 110, 24–36. <https://doi.org/10.1016/j.cad.2018.12.003>
- Chen, Y., & Ostoja-Starzewski, M. (2010). MRI-based finite element modeling of head trauma: Spherically focusing shear waves. *Acta Mechanica*, 213(1), 155–167. <https://doi.org/10.1007/s00707-009-0274-0>
- COMSOL, Inc. (2025). *COMSOL multiphysics*. <https://www.comsol.com/comsol-multiphysics>
- Dassault Systèmes. (2025). *Abaqus*. <https://www.3ds.com/products/simulia/abaqus>
- FEniCS Project Developers. (2025). *FEniCS project*. <https://fenicsproject.org/>
- FreeFEM Developers. (2025). *FreeFEM*. <https://freefem.org/>
- GetFEM Developers. (2025). *GetFEM++*. <http://getfem.org/>
- GMSH Developers. (2025). *GMSH*. <http://gmsh.info/>
- Gurobi Optimization, LLC. (2025). *Gurobi optimizer*. <https://www.gurobi.com/>
- Hovey, C. B., & Buche, M. R. (2025). *Automesh: A rust crate for automatic mesh generation*. <https://doi.org/10.5281/zenodo.15328507>
- INRIA Gamma Team. (2023). *Hexotic: A software for hexahedral mesh generation*. <https://team.inria.fr/gamma/gamma-software/hexotic/>
- ISTI-CNR. (2025). *MeshLab*. <https://www.meshlab.net/>
- Karlson, K. N., Skulborstad, A. J., Madison, J. D., Polonsky, A. T., Jin, H., Jones, A., Sanborn, B., Kramer, S. L., Antoun, B. R., & Lu, W.-Y. (2023). Toward accurate prediction of partial-penetration laser weld performance informed by three-dimensional characterization—part II:  $\mu$ CT based finite element simulations. *Tomography of Materials and Structures*, 2, 100007. <https://doi.org/10.1016/j.tmat.2023.100007>
- Klabnik, S., & Nichols, C. (2019). *The rust programming language*. <https://doc.rust-lang.org/book/>
- Livesu, M. (2019). Cinolib: A generic programming header only c++ library for processing polygonal and polyhedral meshes. *Transactions on Computational Science XXXIV*, 64–76. [https://doi.org/10.1007/978-3-662-59958-7\\_4](https://doi.org/10.1007/978-3-662-59958-7_4)
- Livesu, M., Pitzalis, L., & Cherchi, G. (2021). Optimal dual schemes for adaptive grid based hexmeshing. *ACM Transactions on Graphics*. <https://doi.org/10.1145/3494456>
- Maréchal, L. (2009). Advances in octree-based all-hexahedral mesh generation: Handling sharp features. *Proceedings of the 18th International Meshing Roundtable*, 65–84. [https://doi.org/10.1007/978-3-642-04319-2\\_5](https://doi.org/10.1007/978-3-642-04319-2_5)
- Maréchal, L. (2016). All hexahedral boundary layers generation. *Procedia Engineering*, 163,

- 5–19. <https://doi.org/10.1016/j.proeng.2016.11.007>
- Meagher, D. J. (1980). *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer* (IPL-TR-80-111). Rensselaer Polytechnic Institute.
- Netgen Developers. (2025). *Netgen: A 3D delaunay tetrahedral mesh generator*. <https://ngsolve.org/>
- Nicolella, D. P., Shaffer, S. K., Frazer, L. L., Pant, A. D., & Kote, V. B. (2025). *I-PREDICT*. MTEC.
- NumPy Developers. (2025). *Numpy.lib.format, NumPy v1.24 manual*. <https://numpy.org/doc/stable/reference/generated/numpy.lib.format.html>
- OpenFOAM Foundation. (2025). *OpenFOAM*. <https://www.openfoam.com/>
- Owen, S. J., & Shelton, T. R. (2014). Validation of grid-based hex meshes with computational solid mechanics. *Proceedings of the 22nd International Meshing Roundtable*, 39–56. [https://doi.org/10.1007/978-3-319-02335-9\\_3](https://doi.org/10.1007/978-3-319-02335-9_3)
- Owen, S. J., Staten, M. L., & Sorensen, M. C. (2014). Parallel hexahedral meshing from volume fractions. *Engineering with Computers*, 30, 301–313. <https://doi.org/10.1007/s00366-012-0292-8>
- Pitzalis, L., Livesu, M., Cherchi, G., Gobbetti, E., & Scateni, R. (2021). Generalized adaptive refinement for grid-based hexahedral meshing. *ACM Transactions on Graphics (TOG)*, 40(6), 1–13. <https://doi.org/10.1145/3478513.3480508>
- Pointwise, Inc. (2025). *Gridgen*. <https://www.pointwise.com/>
- Polonsky, A. T., Madison, J. D., Arnhart, M., Jin, H., Karlson, K. N., Skulborstad, A. J., Foulk, J. W., & Murawski, S. G. (2023). Toward accurate prediction of partial-penetration laser weld performance informed by three-dimensional characterization—part i: High fidelity interrogation. *Tomography of Materials and Structures*, 2, 100006. <https://doi.org/10.1016/j.tmater.2023.100006>
- Sandia National Laboratories. (2025). *CUBIT: A 3D mesh generation tool*. <https://cubit.sandia.gov>
- Schoof, L. A., & Yarberry, V. R. (1994). *EXODUS II: A finite element data model*. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States). <https://doi.org/10.2172/10102115>
- Si, H. (2025). *TetGen: A delaunay-based quality tetrahedral mesh generator*. <https://doi.org/10.1145/2629697>
- Siemens AG. (2025). *Siemens simcenter*. <https://www.plm.automation.siemens.com/global/en/products/simcenter/>
- Sorkine, O. (2005). Laplacian mesh processing. *Eurographics (State of the Art Reports)*, 4(4), 1.
- Taubin, G. (1995). A signal processing approach to fair surface design. *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, 351–358. <https://doi.org/10.1145/3596711.3596724>
- TrueGrid. (2025). *TrueGrid: Finite element mesh generation software*. <http://www.truegrid.com>