Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 163 (2016) 251 – 261

Procedia Engineering

www.elsevier.com/locate/procedia

25th International Meshing Roundtable (IMR25)

All-Hex Meshing of Multiple-Region Domains without Cleanup

Muhammad A. Awad[a], Ahmad A. Rushdi[b,c], Misarah A. Abbas[d], Scott A. Mitchell[c], Ahmed H. Mahmoud[a], Chandrajit L. Bajaj[b], Mohamed S. Ebeida[c,*]

[a] *University of California, Davis, CA 95616, U.S.A.*
[b] *Institute for Computational Engineering and Sciences, University of Texas, Austin TX 78712, U.S.A*
[c] *Center for Computing Research, Sandia National Laboratories, Albuquerque NM 87185, U.S.A.*
[d] *Alexandria University, Alexandria, Egypt*

Abstract

In this paper, we present a new algorithm for all-hex meshing of domains with multiple regions without post-processing cleanup. Our method starts with a strongly balanced octree. In contrast to snapping the grid points onto the geometric boundaries, we move points a slight distance away from the common boundaries. Then we intersect the moved grid with the geometry. This allows us to avoid creating any flat angles, and we are able to handle two-sided regions and more complex topologies than prior methods. The algorithm is robust and cleanup-free; without the use of any pillowing, swapping, or smoothing. Thus, our simple algorithm is also more predictable than prior art.

Peer-review under responsibility of the organizing committee of IMR 25
Keywords: All-hexahedral Meshing; Mesh Generation; Guaranteed Quality

1. Introduction

We are given a 3d domain $\mathcal{D}$ of multiple regions, defined by a standard b-rep boundary representation. An all-hex mesh $\mathcal{M}$ has only hexahedral elements, and accurately represests the domain $\mathcal{D}$. The quality of the mesh plays a significant role in the accuracy and stability of numerical simulations or solution of PDEs, e.g., Finite Element Analysis (FEA) [1–3] and Computer-Aided Design (CAD) [4,5].

In 2d, unstructured all-quad meshing algorithms are usually categorized into two main categories: *indirect* and *direct*. A classical indirect approach starts with a triangular mesh, and then transforms the triangular elements into quadrilateral elements, via optimization [6,7], refinement and coarsening [8], or simplification [9]. A class of indirect methods start with a triangular mesh and applies the mid-point subdivision rule [10,11] to split a triangle into three quad elements. This can be achieved using local subdivision operations. For example, [12] implements a recursive subdivision algorithm based on a regular tiling composed of only diamonds and kites, but does not handle domain boundaries. Q-Morph [13] is a popular indirect approach that follows a sequence of systematic triangle transforma-

* Corresponding author. Tel.: +1-505-844-0456
E-mail address: msebeid@sandia.gov

Peer-review under responsibility of the organizing committee of IMR 25
doi:10.1016/j.proeng.2016.11.055

tions to create an all-quadrilateral mesh. However, Q-Morph requires topological cleanup and smoothing to guarantee the quality of the final all-quad mesh. Q-Tran [14] is another indirect algorithm that produces quadrilaterals with provably-good quality without a smoothing post-processing step, and manages to handle domain boundaries. Nevertheless, the class of indirect methods typically suffers from a large number of irregular nodes that are connected to more (or less) than four mesh elements, which is typically undesired in several numerical simulations. Direct approaches, on the other hand, construct quadrilaterals directly. The advancing front algorithms (e.g., the paving algorithm [15]) successfully generate all quad meshes with high quality, by placing mesh points on the boundaries of the input domain and form quad elements by recursively projecting edges on the front towards the interior of the domain until the whole domain is covered with quads [16,17]. However, they suffer from stability problems that require heuristic cleanup operations. Grid based methods construct a uniform Cartesian or quadtree background grid and aim at modifying that grid to conform to the domain boundaries [18,19]. These methods are easy to implement and can provide quality guarantees and angle bounds [20–22]. However, they often result in inverted elements.

In 3d, the all-hex meshing problem has not been completely solved yet. Many diverse approaches have been tried, including constructing meshes from volumetric data [23], from surface quad meshes [24], using the medial axis transform [25], using midpoint subdivision [26], singularity-restricted frame fields [27], and volumetric PolyCube deformation [28]. However, automating these types of methods [29–31] faces several challenges, especially when proper meshing constraints are taken into consideration [32]. Many methods can only achieve hex-dominant meshes [33,34] and some require parallel implementations [35]. Different applications require different mesh properties, which makes the choice of the proper meshing approach more difficult [36–38].

1.1. Grid-based Hex Meshing

Our approach belongs to the grid-based or octree family of methods [19,39–41]. The two key challenges for octrees are to capture the domain boundary with high-quality hexes [37], and to generate all-hex elements in the presence of hanging nodes arising from size transitions. In addition, these methods tend to produce a large number of hexes. To capture the boundary, a common approach is to introduce a boundary layer of hexes. *Pillowing* is one method. For each hex, one face is on the boundary and the opposite face mates with a face of the axis-aligned octree. High quality hexes are challenging with this abrupt orientation change, and smoothing is typically required. Interior to the domain, a variety of transition templates are used to resolve hanging nodes. Early work [19] had fairly restrictive templates, and these have been expanded to allow more rapid and flexible size transitions [40,41]. Hexotic [39] adds ideas from midpoint subdivision to its octree to reduce the number of hexes generated. Its dual transition templates ensure that every corner and hanging node of the octree has six edges, each of which has four faces, but the polyhedral cells are not necessarily hexes. This can be immediately dualized to form hexes, which is equivalent to performing midpoint subdivision on the hexes, then combining the eight hexes surrounding each original node.

1.1.1. Snapping

Hexotic and dual contouring, among others, *snap* (project) nodes to the geometry, then fix hex quality by boundary layers and pillowing. The fundamental shortcoming of snapping is that there is no known way to handle geometry whose local topology is more rich than the topology of the octree. That is, an octree edge has at most four faces, and what does one do if the geometry has an edge with five faces? Also nodes are limited to edge-degree six.

Snapping and our approach are fundamentally different. We do not attempt to match the grid topology to the geometry topology, but instead intersect octree cells with the geometry. This allows us to handle domains with arbitrary topology in principle.

1.2. Contribution

In this paper, we extend the recently developed cleanup-free all-quad meshing approach in [42] (extended for sharp corners in [43]), introducing a new direct grid-based algorithm that produces an all-hex mesh without post-processing cleanup. Conceptually, we follow similar steps to those in [42,43]. Starting with a Cartesian grid or an octree domain decomposition, we repel the grid points *away* from the input geometry, intersect each cube with the geometry faces, use midpoint subdivision to split the intersected elements into hexes, and finally apply 2-refinement templates to

eliminate any hanging nodes. Our method is stable, conforms to both the interior and exterior of the meshed domain boundary, and can easily adapt to changes in the input domain boundaries, via simple *local* operations.

1.3. Terminology

To easily distinguish between the initial grid, the domain geometry, and the final mesh, we use the following terminology. The initial Cartesian or octree grid is composed of *cubes*, each of which has six *sides* and eight *corners*. Corners of adjacent smaller cubes appear as *hanging nodes*. The side length of a cube is denoted s. The geometry is composed of *faces* (which are planar surfaces) and *vertices*. The intersection of the octree and geometry results in *polyhedrons* to be meshed with midpoint subdivision. The final all-hexahedral mesh is composed of mesh *elements* and *nodes*.

2. Algorithm Overview

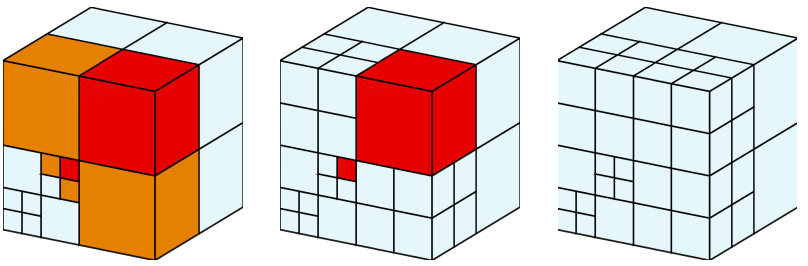
Our all-hex meshing algorithm can be described in a rigorous set of repelling, splitting, and refinement steps to achieve the desired mesh properties. We explain the details of these steps as follows:

1. Start with a strongly-balanced octree, with uniform-size cubes on the domain boundary.
2. Perturb the grid by repelling its corners *away* from the domain boundaries.
3. Split all grid cells intersected by the domain boundary into polyhedra conforming to the boundary.
4. Apply the mid-point subdivision rule to split polyhedra into hex elements. Some elements of this mesh will possess hanging nodes.
5. Employ the 2-refinement templates to get rid of hanging nodes.
6. The result is the final conforming all-hex mesh.

In the rest of this section, we visit each of these steps with more details and illustrations. In Section 3 we describe how to start with non-uniform cubes on the boundary, and apply the 2-refinement template before midpoint subdivision to ensure conforming boundary hexes.

2.1. Octree Initialization, Refinement, and Balancing.

Our algorithm can start with any all-hex mesh. We chose to start with a Cartesian grid and refine it to form an octree that captures the fine details of the geometric domain boundaries. Our algorithm requires that the octree is strongly balanced, where corner-adjacent cubes differ in size by at most one; see Figure 1 for an illustration. We also chose uniform sizing function along the boundaries.

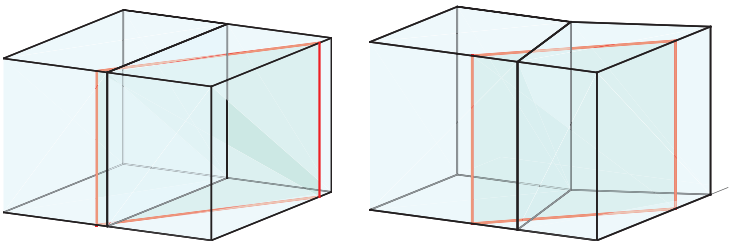


(a) An unblanced octree. (b) A balanced octree. (c) A strongly-balanced octree.

Fig. 1: (a) An unbalanced octree of 3 levels, some cubes of a difference of 2 in levels (orange/red) share sides, and some (red) share an edge. Balancing the octree (b) by refining the larger cube of a violation: no side is shared between two cubes with more than a difference of 1 in levels. In (c), a strongly-balanced octree is one where no edge belongs to two cubes of more than a difference of 1 in levels. We require a strongly-balanced octree to enable the refinement templates in the last step of the algorithm.

2.2. Perturbing the Octree

The goal of this step is to prevent small mesh edges. We repel corners of cubes *away* from nearby geometry, in coordinate axis directions. A raw geometry-octree intersection might be arbitrarily close to another point, or at a very small angle. To avoid these problems, we follow a *grid-aligned repelling* strategy. In specific, we move all corners that are too close to the geometry away from it, in either x, y, or z direction until a distance δ between the intersection point and the new corner location is achieved. An example of this repelling strategy is shown in Figure 2. We experimentally chose δ to be $\frac{s}{4}$, where s is the size of the cube containing the intersection point.

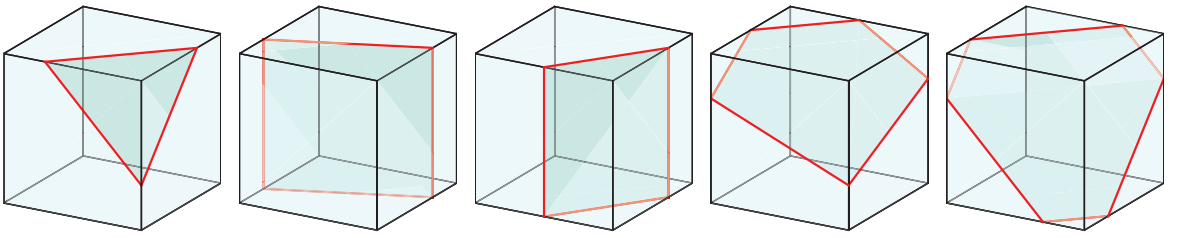


(a) Grid points close to boundary. (b) Grid-aligned repelling.

Fig. 2: To handle grid points that are too close to a boundary surface (a), our grid aligned repelling algorithm (b) moved those points on a grid line away from the boundary until the grid line distance between the new point location and the boundary-grid intersection point is exactly δ.

2.3. Polyhedron Formation

Polyhedra result from splitting cubes by the domain boundary faces. Cubes split by a single face could have 4 to 7 faces, 6 to 14 edges, and 4 to 9 vertices[1]. Multiple faces sharing a single vertex form locals cone around the vertex, of arbitrary degree. Each cone is truncated by the cube into a polyhedron. To ensure convex polyhedra, some cones are further subdivided. To recover all-hex status, we refine all polyhedra into smaller hexahedrons using the mid-point subdivision rule. Figure 3 shows an illustrative example of elements before and after refinement using basic refinement operations, when a hex element is intersected by one or more boundary segments. As shown, a polyhedron gets divided into a set of hexes that meet at a central vertex.



(a) $n = 3, k = 0$ (b) $n = 4, k = 4$ (c) $n = 4, k = 2$ (d) $n = 5, k = 2$ (e) $n = 6, k = 2$

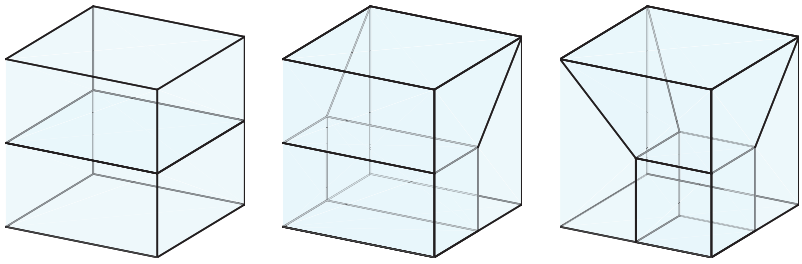
Fig. 3: Grid-boundary intersection scenarios. The intersection between a planar boundary and a regular grid cube (of 6 faces, 12 edges, and 8 vertices) results in an irregular polygon that splits the cube into irregular polyhedral elements. We identify each of the cases above using n: the number of geometry-grid intersection points, and k: the largest number of parallel edges intersected by the geometry. The resulting polyhedral elements have 4 to 7 faces, 6 to 14 edges, and 4 to 9 vertices.

[1] Euler's Formula: for any polyhedron with F faces, E edges, and V vertices, $F + V - E = 2$

2.4. Handling Hanging Nodes

Cubes that intersected the domain boundary formed polyhedra, which were subdivided into hexes with midpoint subdivision. We now consider the remaining empty cubes. An empty cube may have hanging nodes, either because its adjoining cubes are smaller, or because the adjacent cube contained some geometry and was midpoint subdivided. From the perspective of the empty cube, both cases are identical. Cubes with no hanging nodes are already perfect hexes, and need no modification. Cubes with hanging nodes are meshed using the refinement templates with corner marking [44]. Some arrangements of hanging nodes do not provide a cube that can be readily subdivided into hexes.

We use the corner marking to guide adding more hanging nodes, to reduce these arrangements to cases amenable to subdivision, while still ensuring that adjoining cubes have identical hanging nodes and subdivided faces on their shared boundary. In essence, the marking provides a local and globally consistent way to pair adjacent cubes with one or more hanging nodes together, restoring an all-hex mesh. The marking uses a checker board pattern: corners alternate between marked and unmarked. The refinement templates are shown in figure 4, and split an element with hanging nodes into two, three, or four hexes.



(a) two hexes (b) three hexes (c) four hexes

Fig. 4: Refinement templates.

3. Extensions

We discuss extensions to more general settings than we have demonstrated. The correctness requirements establish abstract conditions that are sufficient for the the algorithm to succeed. These turn out to be quite general, leading to the possibility of many diverse extensions that can also satisfy these conditions. These extensions are related in a way that the correctness requirements make clear.

3.1. Correctness Requirements

Abstractly, our method requires two conformal partitions of space (the domain). One is the octree or other background partition we create with unconstrained coordinates; the *space partition*. The other is the boundary representation of the geometric domain itself, or some subdivision of it, with constrained coordinates; the *geometry partition*. We push the nodal coordinates of the space partition away from the geometry partition, and intersect the partitions to create a single *intersection partition*.

For midpoint subdivision to produce topological hexes, the intersection partition must consist of *general polyhedra*, e.g. non-degenerate, having the topology of cells formed by the intersection of half-spaces in general position. A general polyhedron has nodes of edge-degree three, and edges of face-degree two. Recall an octree has the issue that adjacent boxes of different size do not meet conformally, and small adjacent boxes imprint extra edges and nodes. A vertex surrounded by four small hexes imprints a 4-valent hex onto its larger neighbor, and this must be resolved before performing midpoint subdivision. However, some other types of non-conformal imprints are not a problem. For example, a single imprinted edge spanning a face introduces its end points as vertices, each of which will have edge degree three.

To get positive Jacobian hexes, convex polyhedral cells are sufficient. To get good quality hexes, it is sufficient to have cells and facets with modest aspect ratios, and bounded minimum to maximum edge lengths, and bounded dihedral and edge-angles; the resulting quality is a complicated continuum based on these *well-shapedness* criteria.

For the intersection partition to achieve these requirements, it suffices for the space partition to consist of well-shaped general polyhedra, and the geometry partition to be *locally* general and well-shaped. This is sufficient because we can then move the nodes of a space cell some small fraction (related to the angle bounds) of its minimum edge length, and preserve some fraction of its well-shapedness. It is certainly sufficient for the geometry partition to be globally general and well-shaped, but this is not necessary in the following sense.

Since we only require that the intersection of each geometry cell has nodes of edge-degree three, and edges of face-degree two, we may have e.g. a space cell containing a geometry edge with four adjacent geometry facets, as long as these extend beyond the space cell, because the space cell will be cut into four separate polyhedra. Note that unlike snapping, there is no limit to the node-edge and edge-face degrees of the geometry. A geometric edge may have five or more adjacent faces, and a node may have more than six edges.

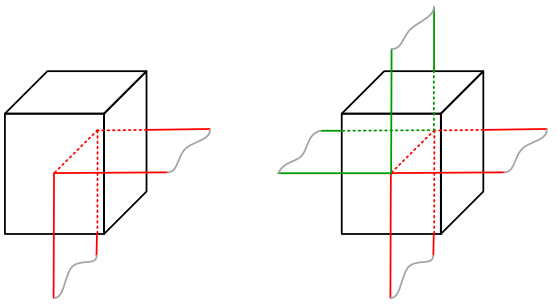
3.2. Non-Uniform Sizing

Sufficient conditions for non-uniform sizing follow immediately from the correctness requirements. If we use an octree with non-conformal boxes, that is with hanging nodes and edges, then it suffices to resolve the hanging nodes and edges through standard transition templates, *before* moving its nodes away from the geometry partition cells.

That approach creates an all-hex space partition, but this is not required. The hanging nodes could be resolved by generating other general polyhedra, such as a mix of tets, triangular prisms, and the like. In principle, one could also use some other spatial decomposition than an octree in the first place. The key difficulty for octrees is a face that has been subdivided into four quads by its neighbors, because the center of the four quads is a node with four edges.

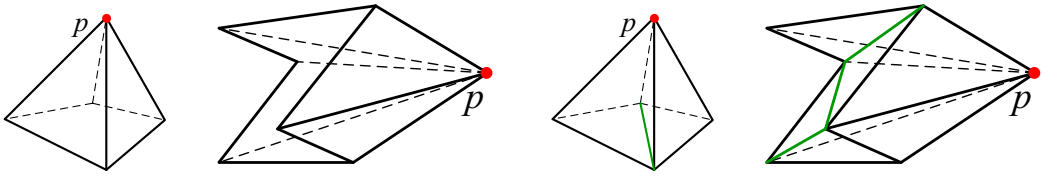
3.3. Reflex Geometry and Complex Geometric Topology

The key geometric difficulty is not sharp features, but reflex-angled ones, because these are not locally convex. One approach is to extend planar and cylindrical surfaces so large-angled features are partitioned into locally convex ones; see Figure 5 where these surfaces extend to the domain boundary (infinity). We call this the *extended virtual-geometry* solution. Figure 6 shows a couple of examples of how to apply this solution to input geometries. For example, the "lightning bolt cone" in Figure 6 is non-convex, and is star-shaped with only the cone vertex in the kernel. For this sort of very challenging geometry, a robust and general solution is to tetrahedralize the geometry, introducing tets into the geometric partition. Unlike dual hex sheets, these extra geometric decompositions (extended geometry and tets) are not required to extend throughout the entire domain. They may be terminated and kept local using the observations in the next subsection.



(a) Reflex-angled geometry feature. (b) Virtually extended geometry.

Fig. 5: A geometry (red) with a reflex angle in a grid cell (black). Adding virtual geometries (green) eliminates the reflex angle.

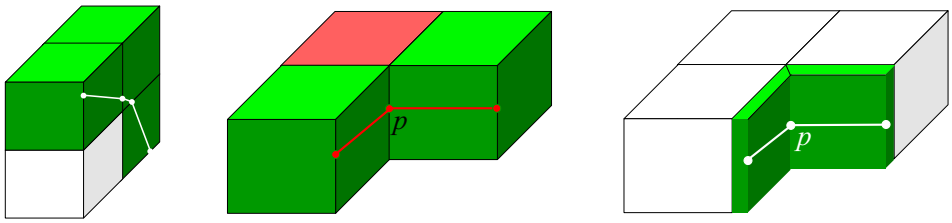


(a) Input models with singular points p. (b) Adding extended virtual geometries.

Fig. 6: A singular topological geometrical vertex p is processed via adding extended virtual geometries to tetrahedralize the input geometry.

3.4. Limiting Geometric Decompositions

Virtual geometry decompositions may be made local by terminating them at the boundary of some space partition cells, creating an imprinted edge on their face neighbors. Recall that a single imprinted edge across a general polyhedron is not a topological problem, and just produces a polyhedron with an extra face, two edges, and two vertices. But we must avoid crossing edges creating a four-valent node, and a single imprinted node splitting an edge into two, etc. See Figure 7. An easy way to avoid this is to enclose a set of hexahedral space cells (the *pillow set*) with a pillow layer. Each hex of the pillow layer shares four faces with other pillow layer hexes, and one *boundary face* with a pillow set hex, and one face with the complement of the pillow set. Thus, a decomposition may be terminated at the boundary of the pillow set, imprinting one edge along each boundary face. For tetrahedral decompositions and virtual geometry planes that intersect, the remaining challenge is to avoid imprinting a four-or-more-valent vertex on a boundary face. One potential way to accomplish this is to terminate only *some* of the decompositions at the boundary faces and let others extend through the pillow layer and beyond. Using a series of pillow layers, all of these decomposition surfaces may eventually be terminated.



(a) Valid imprints. (b) Invalid imprints. (c) Pillowing.

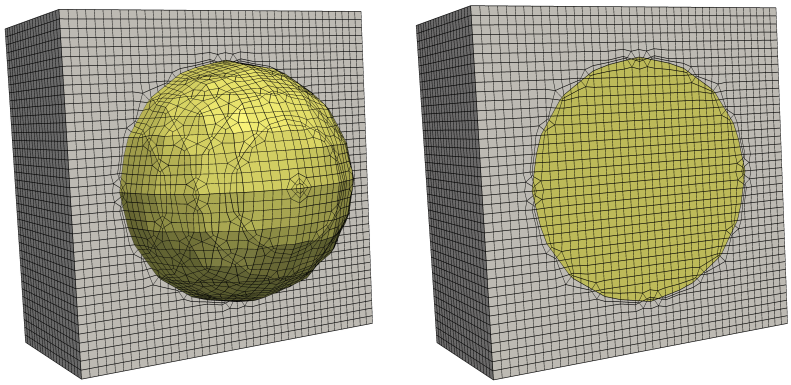
Fig. 7: Stopping the propagation of extended virtual geometries: (a) valid imprint, (b) invalid imprints, vertex p is 2-edge valent in the red grid cube, and (c) using pillowing to eliminate the invalid imprint at p.

4. Experimental Results

Figures 8, 9, 10, 11, and 12 show our all-hex meshes for sphere, 2spheres, donut, doll, and cylinder models. Quality metrics of the resulting meshes are summarized in Table 1, including the minimum and maximum dihedral angles $\theta_{min}/\theta_{max}$, the minimum and maximum edge length ratios $\alpha_{min}/\alpha_{max}$, and the minimum and maximum scaled Jacobian J_{min}/J_{max}. Our algorithm resulted in valid bounds for the dihedral angles and edge lengths, and positive Jacobians everywhere without any cleanup operations.

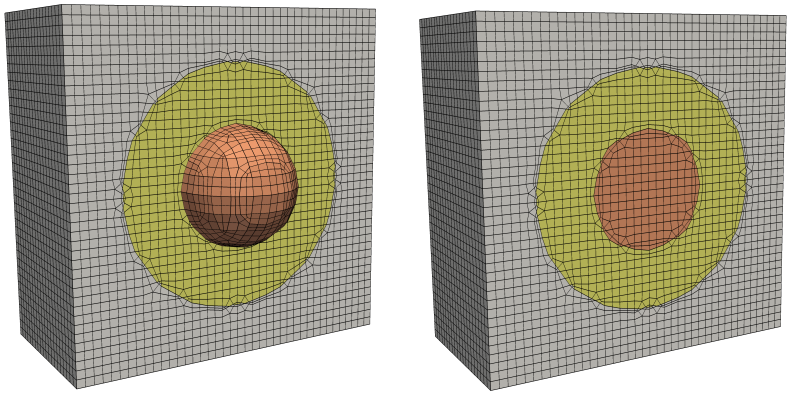
5. Conclusions

We have introduced a new algorithm for all hex meshing of non-convex domains, with connected regions, conforming to both the interior and exterior of the domain. This work is an extension of our prior 2d work, and we



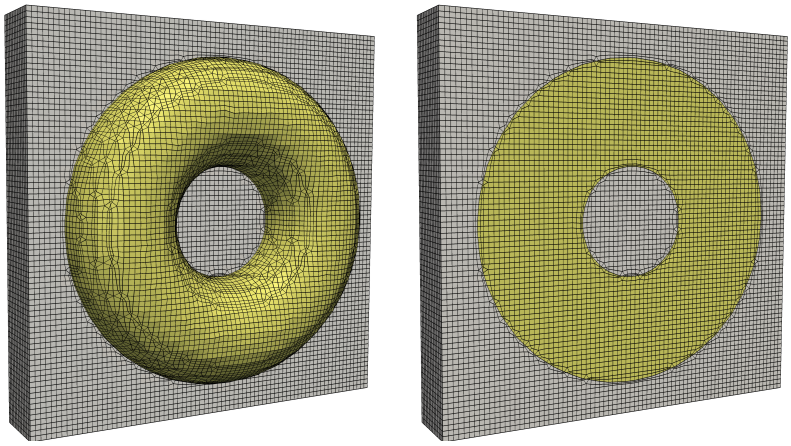
(a) Mesh showing model interface. (b) Mesh showing interior elements.

Fig. 8: All-hexahedral meshing of a sphere model.



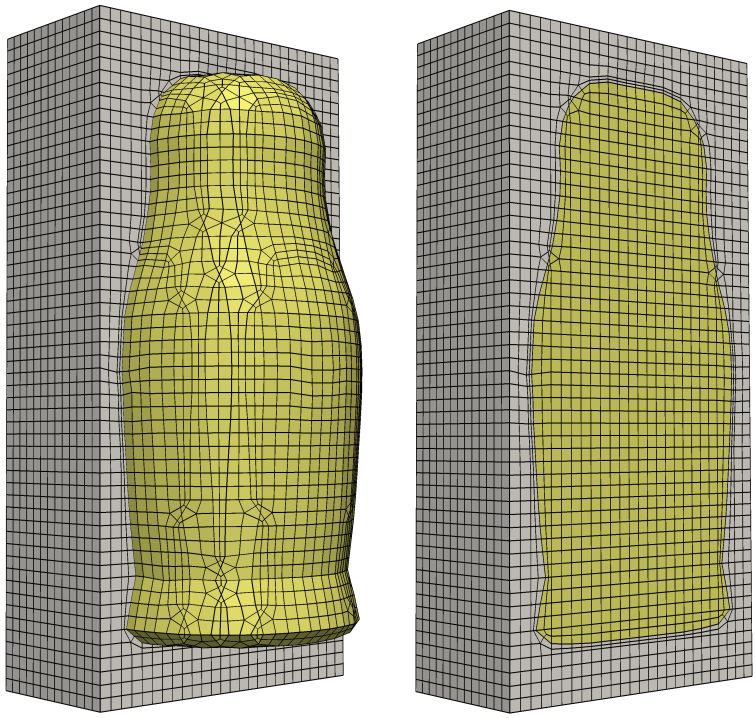
(a) Mesh showing model interface. (b) Mesh showing interior elements.

Fig. 9: All-hexahedral meshing of two-spheres model.



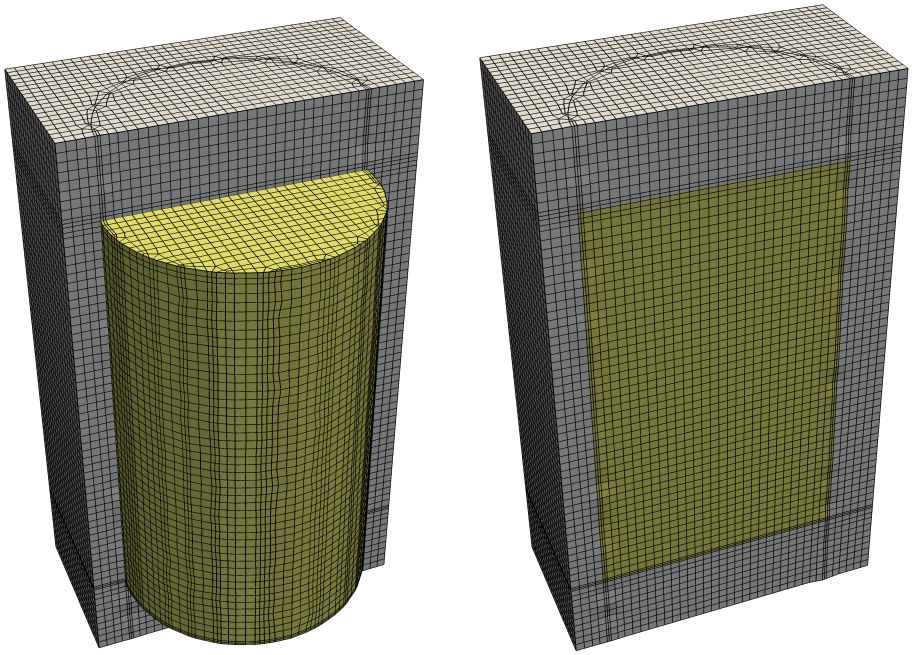
(a) Mesh showing model interface. (b) Mesh showing interior elements.

Fig. 10: All-hexahedral meshing of a donut model.



(a) Mesh showing model interface. (b) Mesh showing interior elements.

Fig. 11: All-hexahedral meshing of a doll model.



(a) Mesh showing model interface. (b) Mesh showing interior elements.

Fig. 12: All-hexahedral meshing of a cylinder model.

Input Domain	$\theta_{\min}$	$\theta_{\max}$	$\alpha_{\min}$	$\alpha_{\max}$	J_{min}	J_{max}
Sphere	19.8°	157.6°	0.12	1	0.2	1
2 Spheres	23.9°	157.9°	0.12	1	0.24	1
Donut	20.8°	159.1°	0.12	1	0.17	1
Doll	20.7°	158.9°	0.11	1	0.18	1
Cylinder	22.5°	157.4°	0.14	1	0.36	1

Table 1: Quality measures: $\theta_{\min}$ and $\theta_{\max}$, the minimum and maximum dihedral angles; $\alpha_{\min}$ and $\alpha_{\max}$, the minimum and maximum edge length ratios of an element; and the min/max scaled Jacobians J_{min} and J_{max}.

have demonstrated it on a few 3d domains. We generated all-hex meshes with positive Jacobians everywhere without any cleanup operations. We showed results using uniform cells on the boundary. Mid-point subdivision converts all general polyhedra into hexes.

We outline principles to extend our approach to models with sharp corners and complex features, handle non-uniform cube size on the domain boundary, employ a balanced octree in the interior, and implement refinement 2-ref templates to eliminate all hanging nodes.

6. Acknowledgments

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] K. Ho-Le, Finite element mesh generation methods: a review and classification, Computer-Aided Design 20 (1988) 27–38.
[2] V. D. Liseikin, Grid generation methods, Springer Science & Business Media, 2009.
[3] W. Hackbusch, Multi-grid methods and applications, volume 4, Springer-Verlag Berlin, 1985.
[4] F. Lee, Principles of CAD/CAM/CAE systems, Addison-Wesley Longman Publishing Co., Inc., 1999.
[5] A. Thakur, A. G. Banerjee, S. K. Gupta, A survey of CAD model simplification techniques for physics-based simulation applications, Computer-Aided Design 41 (2009) 65–80.
[6] M. L. Brewer, L. F. Diachin, P. M. Knupp, T. Leurent, D. J. Melander, The mesquite mesh quality improvement toolkit., in: IMR, 2003.
[7] R. V. Garimella, M. J. Shashkov, P. M. Knupp, Triangular and quadrilateral surface mesh quality optimization using local parametrization, Computer Methods in Applied Mechanics and Engineering 193 (2004) 913–928.
[8] B. D. Anderson, S. E. Benzley, S. J. Owen, Automatic all quadrilateral mesh adaption through refinement and coarsening, in: Proceedings of the 18th International Meshing Roundtable, Springer, 2009, pp. 557–574.
[9] J. Daniels, C. T. Silva, J. Shepherd, E. Cohen, Quadrilateral mesh simplification, in: ACM Transactions on Graphics (TOG), volume 27, ACM, 2008, p. 148.
[10] J. Peters, U. Reif, The simplest subdivision scheme for smoothing polyhedra, ACM Transactions on Graphics (TOG) 16 (1997) 420–431.
[11] H. Prautzsch, Q. Chen, Analyzing midpoint subdivision, Computer Aided Geometric Design 28 (2011) 407–419.
[12] D. Eppstein, Diamond-kite adaptive quadrilateral meshing, Engineering with Computers 30 (2014) 223–235.
[13] S. J. Owen, M. L. Staten, S. A. Canann, S. Saigal, Q-morph: an indirect approach to advancing front quad meshing, International Journal for Numerical Methods in Engineering 44 (1999) 1317–1340.
[14] M. S. Ebeida, K. Karamete, E. Mestreau, S. Dey, Q-TRAN: a new approach to transform triangular meshes into quadrilateral meshes locally, in: International Meshing Roundtable, volume 19, Sandia National Laboratories, 2010, pp. 23–34.
[15] T. D. Blacker, M. B. Stephenson, Paving: A new approach to automated quadrilateral mesh generation, International Journal for Numerical Methods in Engineering 32 (1991) 811–847.
[16] J. Zhu, O. Zienkiewicz, E. Hinton, J. Wu, A new approach to the development of automatic quadrilateral mesh generation, International Journal for Numerical Methods in Engineering 32 (1991) 849–866.
[17] D. R. White, P. Kinney, Redesign of the paving algorithm: Robustness enhancements through element by element meshing, in: 6th International Meshing Roundtable, Citeseer, 1997, pp. 323–335.

[18] P. L. Baehmann, S. L. Wittchen, M. S. Shephard, K. R. Grice, M. A. Yerry, Robust, geometrically based, automatic two-dimensional mesh generation, International Journal for Numerical Methods in Engineering 24 (1987) 1043–1078.
[19] R. Schneiders, R. Schindler, F. Weiler, Octree-based generation of hexahedral element meshes, in: Proceedings of the 5th International Meshing Roundtable, Citeseer, 1996.
[20] X. Liang, M. S. Ebeida, Y. Zhang, Guaranteed-quality all-quadrilateral mesh generation with feature preservation, Computer Methods in Applied Mechanics and Engineering 199 (2010) 2072–2083.
[21] X. Liang, Y. Zhang, Hexagon-based all-quadrilateral mesh generation with guaranteed angle bounds, Computer Methods in Applied Mechanics and Engineering 200 (2011) 2005–2020.
[22] X. Liang, Y. Zhang, Matching interior and exterior all-quadrilateral meshes with guaranteed angle bounds, Engineering with Computers 28 (2012) 375–389.
[23] Y. Zhang, C. Bajaj, Adaptive and quality quadrilateral/hexahedral meshing from volumetric data, Computer methods in applied mechanics and engineering 195 (2006) 942–960.
[24] M. Kremer, D. Bommes, I. Lim, L. Kobbelt, Advanced automatic hexahedral mesh generation from surface quad meshes, in: Proceedings of the 22nd International Meshing Roundtable, Springer, 2014, pp. 147–164.
[25] W. R. Quadros, Laytracks3d: A new approach for meshing general solids using medial axis transform, Computer-Aided Design 72 (2016) 102–117.
[26] T. Li, R. McKeag, C. Armstrong, Hexahedral meshing using midpoint subdivision and integer programming, Computer methods in applied mechanics and engineering 124 (1995) 171–193.
[27] Y. Li, Y. Liu, W. Xu, W. Wang, B. Guo, All-hex meshing using singularity-restricted field, ACM Transactions on Graphics (TOG) 31 (2012) 177.
[28] J. Gregson, A. Sheffer, E. Zhang, All-hex mesh generation via volumetric polycube deformation, in: Computer graphics forum, volume 30, Wiley Online Library, 2011, pp. 1407–1416.
[29] T. Blacker, Automated conformal hexahedral meshing constraints, challenges and opportunities, Engineering with Computers 17 (2001) 201–210.
[30] T. Blacker, Meeting the challenge for automated conformal hexahedral meshing, in: 9th international meshing roundtable, Citeseer, 2000, pp. 11–20.
[31] W. Yu, K. Zhang, X. Li, Recent algorithms on automatic hexahedral mesh generation, in: Computer Science & Education (ICCSE), 2015 10th International Conference on, IEEE, 2015, pp. 697–702.
[32] J. F. Shepherd, C. R. Johnson, Hexahedral mesh generation constraints, Engineering with Computers 24 (2008) 195–213.
[33] V. Vyas, K. Shimada, Tensor-guided hex-dominant mesh generation with targeted all-hex regions, in: proceedings of the 18th International Meshing Roundtable, Springer, 2009, pp. 377–396.
[34] N. Ray, L. Untereiner, B. Levy, N. Ray, D. Sokolov, L. Alonso, P. Chassaing, E. Reingold, R. Schott, F. Boudon, et al., Hexahedral-dominant meshing (2015).
[35] S. J. Owen, M. L. Staten, M. C. Sorensen, Parallel hexahedral meshing from volume fractions, Engineering with Computers 30 (2014) 301–313.
[36] W. R. Quadros, K. Shimada, Hex-layer: Layered all-hex mesh generation on thin section solids via chordal surface transformation., in: IMR, Citeseer, 2002, pp. 169–180.
[37] S. J. Owen, T. R. Shelton, Evaluation of grid-based hex meshes for solid mechanics, Engineering with Computers 31 (2015) 529–543.
[38] C. S. Verma, P. F. Fischer, S. E. Lee, F. Loth, An all-hex meshing strategy for bifurcation geometries in vascular flow simulation, in: Proceedings of the 14th international meshing roundtable, Springer, 2005, pp. 363–375.
[39] L. Maréchal, Advances in octree-based all-hexahedral mesh generation: handling sharp features, in: Proceedings of the 18th International Meshing Roundtable, Springer, 2009, pp. 65–84.
[40] D.-Y. Kwak, Y.-T. Im, Remeshing for metal forming simulationspart ii: Three-dimensional hexahedral mesh generation, International Journal for Numerical Methods in Engineering 53 (2002) 2501–2528.
[41] Y. Ito, A. M. Shih, B. K. Soni, Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates, International Journal for Numerical Methods in Engineering 77 (2009) 1809–1833.
[42] A. A. Rushdi, S. A. Mitchell, C. L. Bajaj, M. S. Ebeida, Robust all-quad meshing of domains with connected regions, Procedia Engineering 124 (2015) 96–108. Proceedings of the 24th International Meshing Roundtable.
[43] A. A. Rushdi, S. A. Mitchell, A. H. Mahmoud, C. L. Bajaj, M. S. Ebeida, All-quad meshing without cleanup, CAD (2016). To appear.
[44] M. S. Ebeida, A. Patney, J. D. Owens, E. Mestreau, Isotropic conforming refinement of quadrilateral and hexahedral meshes using two-refinement templates, International Journal for Numerical Methods in Engineering 88 (2011) 974–985.