Journal of Computational Science 78 (2024) 102278

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

HybridOctree_Hex: Hybrid octree-based adaptive all-hexahedral mesh generation with Jacobian control

Hua Tong [a], Eni Halilaj [a,b,c], Yongjie Jessica Zhang [a,b,*]

[a] Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA
[b] Department of Biomedical Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA
[c] Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA

ARTICLE INFO

Keywords:
All-hexahedral mesh generation
Hybrid octree
Dual mesh
Adaptive mesh
Jacobian
Quality improvement

ABSTRACT

We present a new software package, "HybridOctree_Hex," for adaptive all-hexahedral mesh generation based on hybrid octree and quality improvement with Jacobian control. The proposed HybridOctree_Hex begins by detecting curvatures and narrow regions of the input boundary to identify key surface features and initialize an octree structure. Subsequently, a strongly balanced octree is constructed using the balancing and pairing rules. Inspired by our earlier preliminary hybrid octree-based work, templates are designed to guarantee an all-hexahedral dual mesh generation directly from the strongly balanced octree. With these pre-defined templates, the sophisticated hybrid octree construction step is skipped to achieve an efficient implementation. After that, elements outside and around the boundary are removed to create a core mesh. The boundary points of the core mesh are connected to their corresponding closest points on the surface to fill the buffer zone and build the final mesh. Coupled with smart Laplacian smoothing, HybridOctree_Hex takes advantage of a delicate optimization-based quality improvement method considering geometric fitting, Jacobian and scaled Jacobian, to achieve a minimum scaled Jacobian that is higher than 0.5. We empirically verify the robustness and efficiency of our method by running the HybridOctree_Hex software on dozens of complex 3D models without any manual intervention or parameter adjustment. We provide the HybridOctree_Hex source code, along with comprehensive results encompassing the input and output files and statistical data in the following repository: https://github.com/CMU-CBML/HybridOctree_Hex.

1. Introduction

Hexahedral (hex) mesh generation is a widely adopted volumetric discretization method that is crucial for solving partial differential equations in diverse fields such as computer graphics, medical modeling, and engineering simulations [1]. Compared with tetrahedral meshes, all-hex elements are often preferred due to their superior performance in terms of increased accuracy, smaller element counts, and improved reliability [2,3]. Despite the acknowledged benefits of hex meshes, the automatic generation of high-quality, conforming meshes remains a challenging problem [4,5]. Due to the geometrical stiffness of hex elements, it is difficult to introduce local modifications or adapt mesh refinement strategies compared to quadrilateral (quad) or tetrahedral meshes [6]. Existing techniques for automation have significant trade-offs, and there is a need for further research to overcome technical hurdles and develop improved hex-meshing algorithms and software packages [7]. The complexities also involve decomposing geometry, handling assembly models with multiple components and materials, and propagating the mesh across shared surfaces. Addressing these challenges requires sophisticated algorithms and tools [8]. In addition, all-hex mesh generation poses significant challenges due to the complexity of satisfying geometric and topological constraints, the limitations of current algorithms in handling diverse and complex geometries, and the open questions regarding the untangling of elements, prediction of element quality, and extension of meshing algorithms to implicitly satisfy a broader range of constraints [9]. In addition, all-hex meshing also requires sharp feature preservation, robust quality improvement, and high-order element construction for intricate domains [5,10].

Recent research efforts have largely focused on developing polycube or cross-field methods. The volumetric polycube method facilitates the conversion of a volume to an all-hex mesh by initially embedding axis-aligned boxes within the 3D volume, followed by mapping the volume onto multiple interconnected boxes [11]. The modified centroidal Voronoi tessellation is taken into account in the space of normals or

* Corresponding author.
E-mail address: jessicaz@andrew.cmu.edu (Y.J. Zhang).

https://doi.org/10.1016/j.jocs.2024.102278
Received 14 January 2024; Received in revised form 23 February 2024; Accepted 16 March 2024
Available online 22 March 2024

eigenfunctions to segment the surface and construct polycubes [12,13]. The input model can also be parameterized into a polycube structure to integrate geometry design with isogeometric analysis [14]. CubeCover employs a valid 3D frame field to globally parametrize and generate all-hex meshes [15]. A sophisticated all-hex meshing framework is introduced, which is guided by a singularity-restricted field for volume parameterization [16]. The octahedral field method is improved with singularity graph correction and hex-meshable constrained octahedral field [17]. While these methods are capable of producing near-optimal meshes that follow sharp features, they often struggle to guarantee all-hex or adaptive meshes or to avoid inverted elements.

In recent years, the generation of all-hex meshes using octree data structures is the only solution with the capability to satisfy rigorous scalability and robustness criteria for arbitrary shapes based on adaptive Cartesian grids. Extending the dual contouring method to achieve crack-free interval volume 3D meshing with boundary feature-sensitive adaptation enables the rapid extraction of adaptive and high-quality 3D finite element meshes from volumetric imaging data [18,19]. Octree data structure is exploited to generate high-quality meshes for implicit solvation models of biomolecular structures [20]. Grid-based methods have proven effective in addressing multi-material scenarios [21,22], and they are capable of managing intricate non-manifold computer-aided design (CAD) assemblies as well [23]. An integrated tool, LBIE-Mesher, is designed for constructing 2D or 3D finite element meshes from improved images, supporting features such as multiscale modeling, automatic mesh generation for heterogeneous domains, sharp feature preservation in all-hex meshes, robust quality improvement for non-manifold meshes, and construction of high-order elements [1,5].

In the octree-based pipeline, we need to fit a 3D grid of hexes in the volume and then add hex elements at the boundaries to fill gaps. The interior hexes are filled with templates so that the mesh is adaptive (dense at the boundary and coarse inside). Conversely, the boundary elements need to align with the input geometry, albeit without any guarantee of quality. There are two challenges in octree-based methods. The first challenge is the occurrence of hanging nodes, which arise when neighboring cells possess differing resolution levels, leading to nodes positioned in the middle of adjacent faces or edges. An algorithm that resolves hanging nodes by connecting them with polyhedra has been previously introduced, enabling the extraction of all-hex meshes as duals of polyhedral cells [24]. The cutting process is improved to reduce the number of elements in the interior mesh [25]. The same dual mesh scheme is adopted in [26], but with a different approach, whereas the templates are not thoroughly covered in the paper. By weakly balancing the octree, the class of adaptive meshes can be expanded, thereby relaxing the strong topological constraints. This leads to an enhanced convergence towards valid solutions, optimization of the number and distribution of singular mesh edges, and a reduction in the element count [27]. The generalized pairing criterion is capable of notably decreasing both grid and mesh size while ensuring to produce an all-hex mesh [28].

The second challenge is to maintain the quality of the surface mesh. Laplacian smoothing is a straightforward and easy-to-implement technique that repositions a vertex to the average of its neighbors [29]. While this method is cost-effective and simple to implement, it carries the risk of inverting surrounding elements. To address this challenge, an optimization-based approach that assesses the quality of elements surrounding a node and strives to improve them is proposed [19,30]. A hybrid approach, integrating Laplacian smoothing with optimization, is advocated to balance efficiency and robustness [29,31]. A novel method for enhancing the quality of non-manifold hex meshes in microstructure materials, employing a vertex categorization approach and a comprehensive technique involving pillowing, geometric flow, and optimization, addresses challenges in previous works [32]. Iterative smoothing techniques are proposed to gradually relocate vertices towards the boundary, prohibiting local smoothing adjustments in case of hex flipping [24,33]. These methods are computationally efficient but may occasionally fail to preserve the geometry. A global deformation method exhibits robust alignment of the resulting mesh with the input surface with sharp features and ensures the alignment within an error bound [26]. However, the robustness comes at the cost of increased computational and memory resources, which might be prohibitive for standard hardware configurations.

In this paper, we introduce a new software package, "HybridOctree_Hex," designed for the automatic, robust, and efficient generation of adaptive and quality all-hex meshes with Jacobian control. Our method ensures the absence of self-intersections and produces high scaled Jacobians (> 0.5), which well surpass the minimum scaled Jacobian of interior template elements. Moreover, this innovation is particularly adept at capturing intricate, detailed features with precision. Given a 3D closed manifold surface as the starting point, our method first initializes an octree structure and builds a strongly balanced octree. Inspired by our earlier preliminary hybrid octree mesh generation technique [25], we catalog all possible transition scenarios encountered during the all-hex dual mesh generation process into pre-defined templates. This comprehensive approach allows better mesh adaptation and exhibits faster template transformation. We also name the software after "HybridOctree" to emphasis the templates are derived from the hybrid octree technique which guarantees an all-hex mesh generation. Subsequently, we remove elements outside and around the boundary and then connect the boundary points of the interior core mesh with their corresponding closest points on the boundary surface to fill the buffer zone. During the final step of meshing the buffer zone, we incorporate geometric fitting, the scaled Jacobian, and the Jacobian into our energy function, leverage the smart Laplacian smoothing to expedite convergence, and address issues of surface points getting trapped in local minima. Our extensive experimental results, obtained by processing dozens of models, serve as a testament to the robustness and efficacy of our proposed HybridOctree_Hex package. In comparison to the current state-of-the-art technique outlined in [26], our method produces superior-quality meshes with a reduced number of elements. To facilitate further research and collaboration, we have made available our HybridOctree_Hex software source code and a comprehensive collection of meshes generated using our algorithm, along with corresponding input and output data, at https://github.com/CMU-CBML/HybridOctree_Hex.

The paper structure is outlined as follows: Section 2 delineates the comprehensive algorithm for initializing an octree, generating a strongly balanced octree, designing templates based on our earlier preliminary hybrid octree work, generating the interior core mesh, and meshing the buffer zone with geometric fitting Jacobian control. Section 3 presents meshing results and Section 4 concludes the paper and outlines future research.

2. Hybrid octree and all-hex mesh generation

Starting from a closed, manifold surface triangular mesh that defines the domain to mesh, our goal is to generate high-quality adaptive all-hex meshes based on a hybrid octree. The detailed algorithm can be explained in five steps. As shown in Fig. 1, we first initialize an octree structure refined at high-curvature and narrow regions to capture detailed features. The octree is further refined to be strongly balanced following the balancing and pairing rules. Inspired by our earlier preliminary hybrid octree work [25], we design templates for transition configurations to enable an all-hex dual mesh extraction from the strongly balanced octree. After that, the elements outside the boundary and in the buffer zone are cleared. The buffer zone is then meshed by connecting the outmost points of the core mesh with their closest points on the input surface. In the final step, the smart Laplacian smoothing and optimization are coupled to improve the mesh quality with geometric fitting and minimum scaled Jacobian control. Here are definitions of an octree and the level of octree.

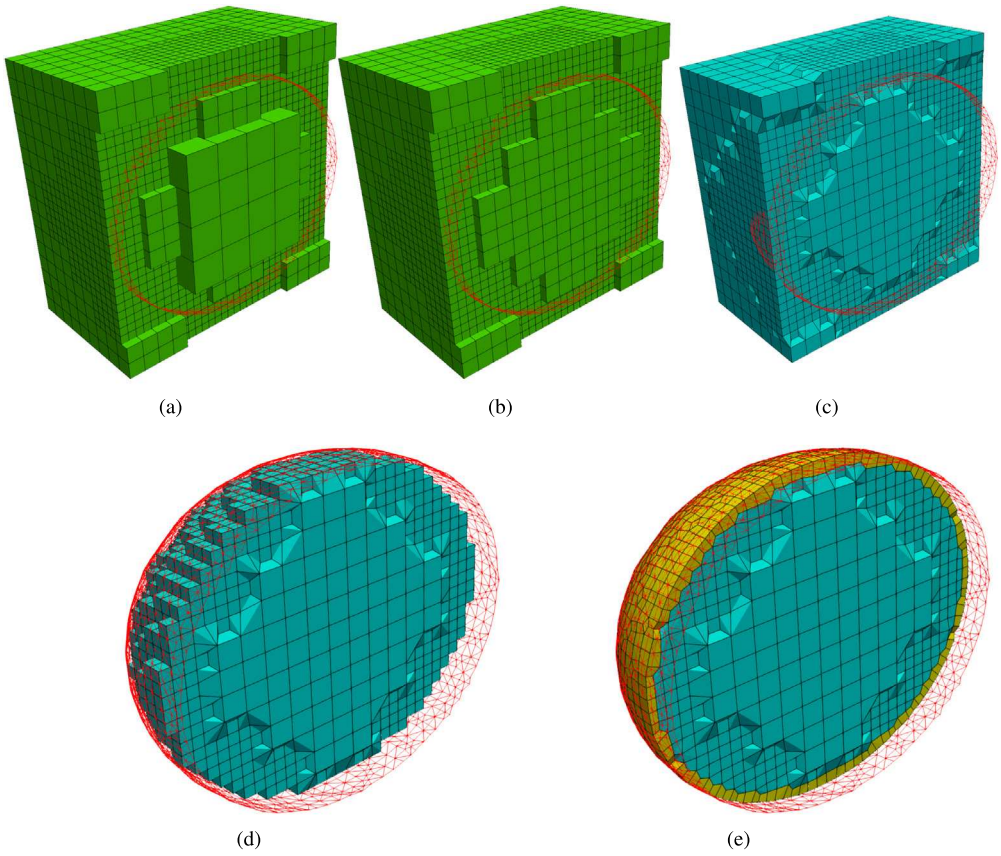

Fig. 1. HybridOctree_Hex overview. (a) Initializing the octree from the surface triangular mesh (red) with feature preservation; (b) Transforming the initialized octree into a strongly balanced octree, satisfying the balancing rule and the pairing rule; (c) Constructing an all-hex dual mesh using pre-defined templates; (d) Clearing elements outside and around the boundary; (e) Meshing the buffer zone (yellow) with geometric fitting and Jacobian control.

Octree: *An octree is a tree-based data structure used in the three-dimensional space. It recursively subdivides a cubic space into eight octants (or children) until a desired level of detail or termination criterion is reached. Each node in the octree represents a cubic volume, and the children of a node represent the eight subdivisions of that volume.*

Level of Octree: *The level of an octree refers to the depth or hierarchy of the tree structure. The root node, representing the entire space, is at level* 0*. Each subsequent level represents a finer subdivision of space. The level increases as traversing down the octree, indicating a higher level of detail or smaller cubic volumes being represented. The children of a node at level* n *are at level* $n+1$*. The maximum level of an octree is determined by the desired resolution.*

2.1. Initializing the octree and building a strongly balanced octree structure

The input surface triangular mesh describes the boundary surface with a set of triangular elements. The triangular mesh is embedded into a cube, which is the root of the octree, marked as level 0. Then, the octants are recursively subdivided based on the surface feature. Numerical simulations aim to use a minimal number of elements, while still preserving crucial features on the boundary surface, to ensure both efficiency and accuracy in calculations. To better capture high curvatures and narrow regions on the surface, we need to refine these regions to desired octree levels.

The Gaussian curvature G at point P_i is computed by $||\sum_{j\in\mathcal{N}(i)}(\cot\alpha_{ij}+\cot\beta_{ij})(P_j-P_i)||_2/(4A_i)$, where $j\in\mathcal{N}(i)$ is the index of vertices directly adjacent to P_i, α_{ij} and β_{ij} are two angles opposite to edge P_iP_j, and A_i is the Voronoi cell area around P_i [34]. Five curvature thresholds $G_{thres}=\{0.5,1,2,4,8\}$ are adopted. If an octree cell at level $l+4$ ($l=0,1,\ldots,4$) containing or intersecting with the triangular boundary [35] satisfies $G(l+4)>G_{thres}[l]$, we refine it to level $l+5$. We use curvature to preserve detailed surface features. In addition, the thickness T at point P_i is measured by shooting a ray from P_i along its normal direction to find the shortest intersection with other triangular elements on the boundary. We set five thickness thresholds $T_{thres}=\{16,8,4,2,1\}$, and refine all the octree cells with $T(l+4)<T_{thres}[l]$ to level $l+5$. An octree structure is initialized after these operations, where the resulting octree level of each octant ranges from level 5 to 9. The following two rules are applied to convert the initialized octree to a strongly balanced one [25].

Balancing rule: *The level difference between two neighboring octants is at most one.*

Pairing rule: *If an octant is subdivided to comply to the balancing rule, its siblings (the other seven octants belonging to the same parent) are subdivided along with it.*

After implementing the balancing and pairing rules, the resulting strongly balanced octree consists of cubes with hanging nodes scattered within the octree structure due to the size variation. In this paper, we design templates based on the cutting procedure algorithm in our earlier preliminary work [25], and the all-hex dual mesh is constructed using these templates. In the following, we provide key definitions pertinent to the subsequent algorithmic description.

Hanging node: *A hanging node is a corner vertex hanging in the middle of neighboring faces or edges. In other words, it is a corner vertex for some elements but not for other adjacent elements containing it.*

Octree (Quadtree) block: *In a 3D octree, eight children cells belonging to the same parent cell form an octree block. Similarly in a 2D quadtree, four children cells belonging to the same parent cell form a quadtree block.*

Transition face: *A transition face is a face in an octree block that is shared by two octree cells with different octree levels.*

Transition edge: *A transition edge is an edge in an octree block that is shared by three octree cells with different octree levels.*
Hybrid octree: *A hybrid octree is derived from a strongly balanced octree to remove all hanging nodes and ensure each interior grid point is always shared by eight leaf cells [25]. The leaf cells of the hybrid octree can be polyhedra instead of all-hex.*

2.2. Designing templates and constructing all-hex dual mesh

The strongly balanced octree exhibits gradual size variations, although it still contains hanging nodes throughout. A cutting algorithm, designed in [25], aims to eliminate hanging nodes in both 2D quadtrees and 3D octrees. This algorithm cuts strongly balanced quadtree/octree blocks into polygonal/ polyhedral cells and extracts the dual mesh from the resulting hybrid quadtree/octree to obtain an all-quad/ hex dual mesh. In 2D, the process involves cutting along the transition edge, while in 3D, it is more intricate, addressing five transition cases on faces and edges. The key point of the cutting method is to ensure that within the hybrid quadtree/octree derived from the strongly balanced quadtree/octree via the cutting procedure, every grid point is shared by four polygonal cells in 2D and eight polyhedral cells in 3D. This results in an all-quad/ hex dual mesh generation. In this paper, we combine the two steps of creating a hybrid octree from a strongly balanced octree and generating a dual mesh of the hybrid octree together in our implementation by designing templates. We develop five templates in 3D to construct all-hex dual mesh directly by detecting transition faces and edges in the strongly balanced octree, which skips the sophisticated implementation of a hybrid octree and accelerates the entire meshing process. In 2D, only one template is needed.

2D template. In a strongly balanced quadtree, four quadrants belonging to the same parent form a quadtree block. A transition edge is formed when two neighboring quadtree blocks have different levels. Four possible configurations occur in 2D. In Fig. 2(a), when one of the two adjacent quadtree blocks (blue) is subdivided, there is a transition edge (golden) between the pink and blue blocks. There are two hanging nodes on the edge, marked by red dots. Note that each hanging node is shared by three cells while each regular grid point is shared by four cells. To obtain the dual mesh of the strongly balanced quadtree [19, 30], each cell center is selected as the associated mesh vertex. For each grid point, its dual element is created by connecting the mesh vertices from cells sharing it. Given that every grid point is shared by either three or four cells, the resulting dual mesh consists of triangles and quads.

The resulting triangles in the dual mesh need to be removed. We design a template in Fig. 2(e) to transform those two triangles and one quad in between into four quads without affecting the surrounding connectivity. Furthermore, when transitions occur in different directions to the same quadtree block, we transform them independently with the help of the pairing rule. Fig. 2(b-d) show configurations with both horizontal and vertical transition edges, and each direction is similar to Fig. 2(a). By checking all the possible configurations, we obtain one unique transition template as shown in Fig. 2(e) to achieve all-quad dual mesh extraction from the strongly balanced quadtree.

Lemma 1. *For dual mesh extraction from the strongly balanced quadtree, only one transition template is needed: the hybrid elements consisting of two triangles and one quad in between are converted to four quads, as shown in Fig. 2(e).*

Discussion 2.1. In the case of a strongly balanced quadtree, we directly detect the transition edges and generate all-quad dual meshes using the designed template. This eliminates the need to generate a complex and memory-intensive hybrid quadtree, leading to significant improvements in computational efficiency. By independently addressing each transition edge with the dedicated template, we ensure a seamless mesh transformation process.

Table 1
Number of polyhedra and hexes in each template.

Element type	Transition on Face	Transition on Edge			
	Fig. 3(a)	Fig. 3(b)	Fig. 3(c)	Fig. 3(d)	Fig. 3(e)
Polyhedra	9	3	3	3	3
Hexes	13	5	4	3	3

3D templates. The 3D transition configurations are similar to the 2D cases, albeit with significantly greater complexity [25]. Following the 2D cases, we check all the possible configurations and design a template to transform each configuration independently. Based on the pairing rule, an octree block is created by grouping eight child octants that originate from the same parent octant. This grouping occurs because these child octants have identical behavior — either all of them are further subdivided, or none of them is subdivided. A transitional scenario arises when the adjacent blocks have level differences. Degenerated elements appear on the transition faces/ edges, necessitating their conversion into all-hex elements. Similar to 2D, the face transition case is shown in Fig. 3(a). Octants of the same color (blue or pink) belong to the same block. Only half of each block adjacent to the transition face is depicted. There is a yellow transition face between these two blocks since the blue block is subdivided, the red dots represent hanging nodes on the transition face, and the dual mesh of the strongly balanced octree is shaded in light green, consisting of one hex, four pyramids, and four triangular prisms. The designed template can generate an all-hex dual mesh to deal with the face transition scenario. Unlike 2D transitions, 3D transitions maintain quad elements on the top and bottom faces of the frustum but introduce two new vertices on each side face. Therefore, we must enumerate all transition cases involving neighboring octree blocks and devise a transitional template for each to accommodate the two new vertices generated by the face transition template.

By checking all the possible configurations, the position relationship between two transition faces falls into four distinct cases, corresponding to four octree blocks sharing a transition edge at different levels. One, two, and three of the four octree blocks are subdivided to a finer level as shown in Fig. 3(b-e). Only one-fourth of each block adjacent to the transition edge is drawn, octants with the same color (blue, green, or purple) belong to the same block. Fig. 3(b, d) are the configurations between two vertically placed face transitions. Two new points are placed at the diagonal to match face transition templates. Fig. 3(c) depicts the configuration between two face transitions placed side by side. Two new points are placed on each side face to match the face transition. Fig. 3(e) is a special case of Fig. 3(b, d), where four face transitions are placed in a cross. Two new points are placed at the two diagonals to match the face transitions on both diagonals. For each transition case, the number of polyhedra and the resulting hexes after transformation is listed in Table 1. The number of polyhedra and hexes in Fig. 3(a) is greater than the other configurations because Fig. 3(a) is the face transition case, and the other configurations are edge transition cases.

It is worth noting that when an octree block has three transition faces in the XYZ directions, our templates leave a hex at the corners of the transition, which is not shown in Fig. 3. By placing hexes at those corners, we achieve an all-hex dual mesh. Additionally, we have a minimum scaled Jacobian of 0.258 in the designed templates which will be further improved in the resulting dual mesh through quality improvement.

Lemma 2. *For dual mesh extraction from the strongly balanced octree, five transformation templates are specifically designed to handle one face transition and, additionally, four edge transitions. These templates are based on the resulting dual mesh of the hybrid octree described in [25]. The cutting procedure ensures that each grid point in the obtained hybrid octree is always shared by eight polyhedra, resulting in an all-hex dual mesh.*

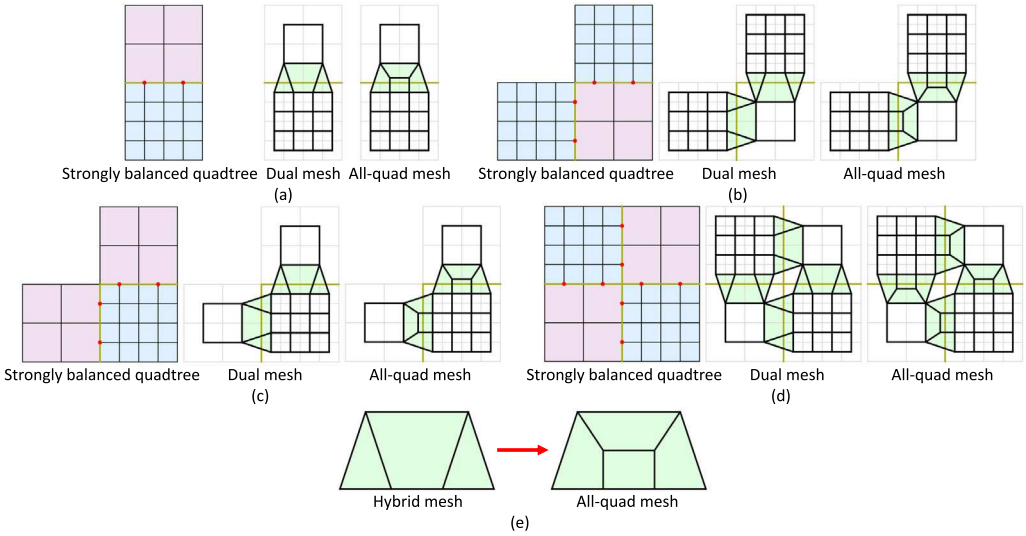

Fig. 2. Quadtree transition configurations (a-d) showing the strongly balanced quadtree, hybrid dual mesh, and the resulting all-quad mesh after applying the transformation template in (e). There are two red hanging nodes on each golden transition edge. (a) Two neighboring quadtree blocks with one being refined; (b, c) Three neighboring quadtree blocks in an *L* shape with the left and right blocks refined or the corner block refined, respectively; (d) Four neighboring quadtree blocks with a pair of diagonal blocks refined.

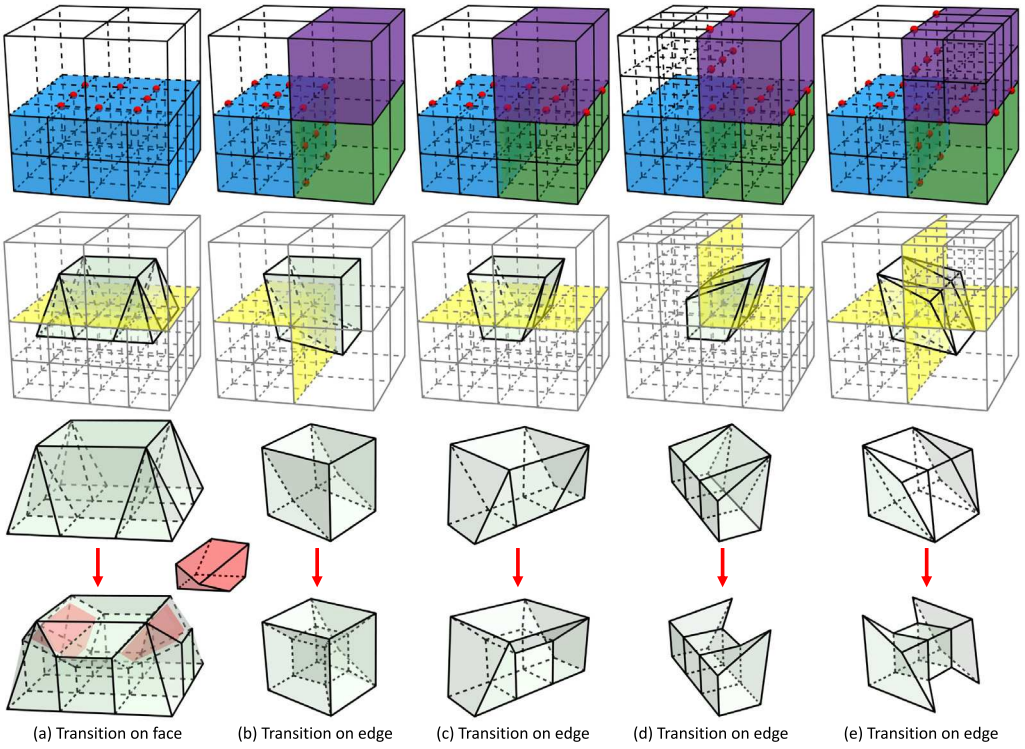

Fig. 3. Octree transition configurations (a-e) showing the strongly balanced octree (first row), hybrid dual mesh (second row), and the transformation template (third and fourth row). Hanging nodes are in red, transition faces are in yellow, and octants with the same color belong to the same block. (a) Two neighboring octree blocks with one being refined, resulting in the transition on the face. The hexes with the minimum scaled Jacobian 0.258 among all the templates are highlighted and extracted; (b-e) Four neighboring octree blocks at different levels sharing a transition edge.

Therefore after applying the transition templates, our resulting mesh is guaranteed to be all-hex.

Discussion 2.2. In contrast to the cutting method described in [25], our approach not only produces the same all-hex meshes but also introduces template-based transformations specifically tailored for independently handling transition edges and faces. The cutting method involves complex polygonal/ polyhedral cells in the hybrid quadtree/ octree, which are challenging to store and manipulate during software implementation. Instead, our octree transformation employs five pre-defined templates to independently address each transition face or edge, simplifying the mesh generation process and enhancing the ease of implementation. Specifically, we directly detect transition faces and edges in the strongly balanced octree and apply these pre-defined templates to generate an all-hex mesh. This approach obviates the need for a complex and memory-intensive hybrid octree, thereby significantly improving computational efficiency.

2.3. Clearing buffer zone

After generating the all-hex dual mesh, the interior core mesh is obtained by removing elements outside and in the vicinity of the boundary surface. The region near the boundary is referred to as the buffer zone. Similar to [36], if the minimum distance from a vertex to the boundary falls below a threshold $\epsilon_s = s_{\max}/2$, where $s_{\max}$ represents the maximum size among the elements sharing that vertex, all elements connected to it are deleted. During the implementation, we observed that this setting can be sensitive to large elements located in size transition regions, potentially leaving holes in the surface. To address this issue, we calculate the signed distance function for corner points associated with every hex element [37,38]. Each hex has eight signed distance functions $f(x^i)$, where $i = 0, 1, \cdots, 7$, attached to the eight corner points. We compute $f_{\min}$ and $f_{\max}$ and remove the hex element if the condition $f_{\min} + 0.1 \times f_{\max} < 0$ is met.

A buffer zone exists between the interior core mesh and the input surface. The extracted core mesh cannot be used directly for buffer layer meshing, and we need a more precise operation to remove elements on the boundary of the core mesh that may lead to poor-quality elements in the buffer zone. We adopt the scaled Jacobian as our metric for this purpose [19]. Within each hex, for every corner node x, three edge vectors are defined as $e_i = x_i - x$ $(i = 0, 1, 2)$. Then the Jacobian matrix at x is defined as $[e_0, e_1, e_2]$, and its *Jacobian* $J(x)$ is the determinant of the Jacobian matrix. We obtain the *Scaled Jacobian* $SJ(x)$ if e_0, e_1, e_2 are normalized. For each hex, we compute the (scaled) Jacobian at eight corners and the body center. For the body center, e_i $(i = 0, 1, 2)$ is computed using three pairs of opposite face centers.

Here we introduce a restriction on the buffer zone clearance. Assume x is a boundary point of the core mesh, and it is shared by m quad faces on the core mesh boundary. We consider the triangles formed by point x and its two adjacent points. Denoting the normal vectors of these triangles as n_i, where $i = 0, 1, \ldots, m-1$, we have a total of m triangles being considered. This also represents that m new hexes will be generated around x after meshing the buffer zone. These hexes are denoted as $h_i, i = 0, 1, \cdots, m-1$. The edge vector from x to its corresponding surface point is denoted as e. According to the scaled Jacobian definition, a valid e needs to satisfy $\min SJ(h_i) \le \min SJ(x) \le \min(n_i \cdot e)$, where $i = 0, 1, \cdots, m-1$. If $\min(n_i \cdot e) < 0$, then we have $\min SJ(h_i) < 0$. Therefore we define the restriction:

Restriction for buffer zone clearance: *Any three normals n_i, n_j and n_k of the m quad faces surrounding x need to satisfy $(n_i \times n_j) \cdot n_k > 0$. In the implementation, we remove one hex attached to x iteratively until $(n_i \times n_j) \cdot n_k > 0$ for all $i, j, k = 0, 1, \cdots, m-1$ and $i \neq j \neq k$.*

As shown in Fig. 4(a), our buffer zone clearance restriction is different from [36], which deletes the single elements sharing only a point, an edge, or a face with other elements, as well as elements with non-manifold connectivity. We notice that the non-manifold criterion is not sufficient to ensure good element quality in the buffer zone. Instead, the normals around a boundary point of the core mesh are the key factors, and we believe this is the main reason why we could improve the buffer zone element quality to surpass the interior template element quality (the minimum scaled Jacobian of the template elements is 0.258, which will be improved to higher than 0.5 after quality improvement; see Section 2.4).

Finally, it is worth mentioning that we iterate over all the surface points of the core mesh. For each surface point, we iteratively remove one hex in each iteration until all the surface points satisfy the restriction. In each iteration, we record the hexes that are attached to boundary points and that violate the restriction, along with the number of boundary faces they share. We prioritize deleting the hex with the most boundary faces to prevent leaving holes in the resulting core mesh.

2.4. Quality improvement with Jacobian control

In the last step, we mesh the buffer zone by connecting each boundary point x_i of the core mesh with its closest point on the input triangular surface x_i^s to form hex elements h_j, as shown in Fig. 4(b). The resulting elements in the buffer layer may exhibit poor quality or even possess a negative Jacobian. The objective is to preserve the positions of boundary points on the surface while relocating corners of worst-Jacobian elements to enhance the overall mesh quality. Here, we couple optimization with smart Laplacian smoothing. The optimization is performed in each iteration to identify the worst-Jacobian element in the entire mesh and adjust its corner points, whereas smart Laplacian smoothing is performed every 1000 iterations on the outmost two layers of vertices only to expedite the quality improvement. Specifically, within the smart Laplacian smoothing procedure, for surface points, we compute the average position of their neighboring surface points and determine the closest point on the input surface to this average position; for interior points, we calculate the average of their neighboring points. We only relocate a point if the minimum scaled Jacobian of the elements affected by the movement exceeds ϵ_{SJ} or if the distance from the point to the input surface is the greatest.

In the optimization, we propose a new energy function, consisting of the geometry fitting, scaled Jacobian and Jacobian terms. We have

$$\begin{aligned} E &= E_{GF} - E_{SJ} - E_J \\ &= \sum_{i=0}^{nvert-1} \|x_i - x_i^s\|_2^2 - \sum_{j=0}^{np-1} \min SJ(h_j) - \sum_{k=0}^{nn-1} \min J(h_k), \end{aligned} \tag{1}$$

where $nvert$ is the number of surface vertices, np is the number of positive-Jacobian hexes, and nn is the number of negative-Jacobian hexes. We adopt a gradient-based method to iteratively minimize the energy function. All the mesh vertices are optimized by

$$x_i \to x_i - \alpha \nabla E|_{x_i}, \quad i = 0, 1, \ldots, 2 \times nvert - 1, \tag{2}$$

where we choose the weight $\alpha = 0.8 \times 10^{-3}$ for all the tested models in this paper.

In finding the closest surface point x_i^s for each boundary point x_i of the core mesh efficiently, we iterate through all the triangles in a bounding box with the size of 10 times the local triangular edge length and compute the closest point on each triangle to x_i. For faster computation, the closest triangle number to each x_i is updated every 1000 iterations. If the maximum value of the minimum distance from all points to the surface is $< 10^{-6}$, we directly pull every x_i onto its corresponding x^s.

The scaled Jacobian and Jacobian terms in Eq. (1) are only applied to hexes with the scaled Jacobian below a pre-defined threshold ϵ_{SJ}, which is initialized as 0.01. In every 1000 iterations, we increase ϵ_{SJ} by 0.01 until all the hexes have $\min SJ(h_i) \ge \epsilon_{SJ}$ and all the boundary points are on the input surface, and finish the optimization if the mesh quality cannot be further improved. Note that when only using the scaled Jacobian in the optimization, one problem arises: the scaled Jacobian is in-differentiable at $|P_0P_1| = 1, 4$ as shown in Fig. 5(b) or has a valley at $|\nabla E| = 1.161$ as shown in Fig. 5(d). To address this issue, we use the combined scaled Jacobian and Jacobian approach. The scaled Jacobian term is selected for optimization when it is differentiable and positive. Otherwise, the Jacobian term is chosen, which is differentiable in the entire domain and does not have minima in the negative-Jacobian region. It worth noting that the Jacobian value cannot be used in the region of $\min J(h_i) > 0$, because the to-be-optimized point cannot converge to the optimal position $|P_0P_1|$ or $|\nabla E| = 0$ and will continue to $|P_0P_1|$ or $|\nabla E| \to -\infty$ as shown in Fig. 5(c, e).

Discussion 2.3. For quality improvement, previous methods first remove non-manifold points on the core mesh surface and then apply geometric flow to enhance the overall mesh quality [20,30]. Afterward, optimization-based smoothing is employed to improve the

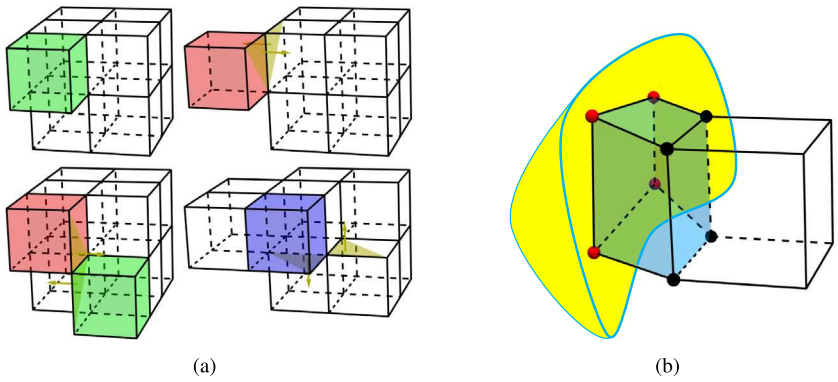

Fig. 4. (a) Four cases comparing with [36]. Hexes removed by both methods are in red, hexes removed only by [36] are in green, and the hex removed by our method only is in blue. Shaded yellow triangles represent their normals violating our restriction and one adjacent hex needs to be removed; (b) The buffer layer is formed by connecting core mesh boundary points (black dots) and their corresponding points (red dots) on the input surface.

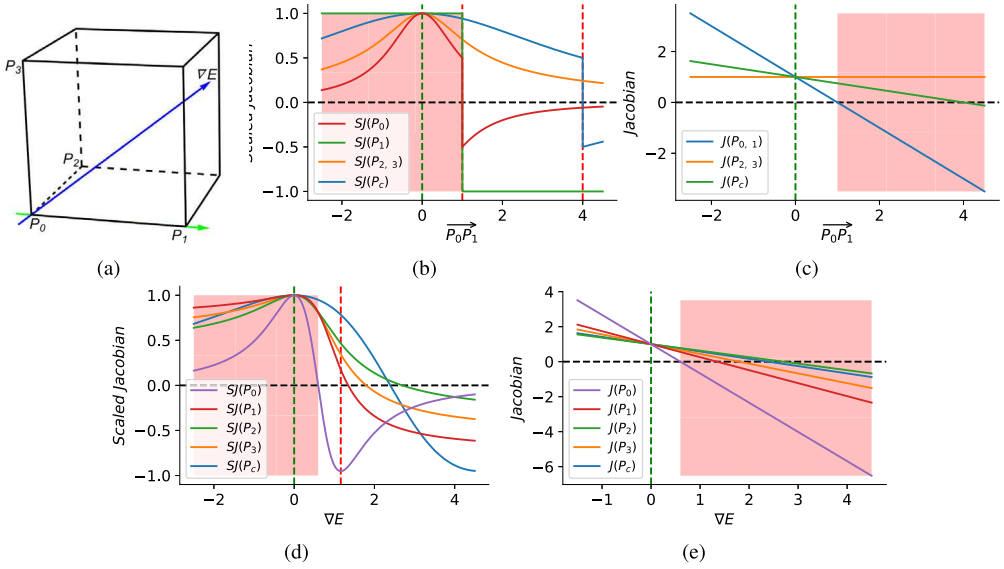

Fig. 5. (a) A hex with $|P_0P_1| = 1$ is to be optimized, where P_0 moves along the $\overrightarrow{P_0P_1}$ direction (green arrow) in (b, c) or an arbitrary direction ∇E (blue arrow) in (d, e). The optimum point is $|P_0P_1|$ or $|\nabla E| = 0$, marked as dashed green lines in (b-e). The intervals of adopting Jacobian or scaled Jacobian in the algorithm are shaded in red. The in-differentiable points and valleys are marked as dashed red lines in (b, d); (b, d) The scaled Jacobian at P_0, P_1, P_2, P_3, and the body center P_c; (c, e) The Jacobian curves.

lowest-quality elements of the mesh [31]. In contrast to this, our approach involves computing face normal directions around each surface point. Subsequently, we remove one or multiple hexes, ensuring that no strict angle limitations exist during the meshing of the buffer zone. Within the gradient-based optimization, we utilize a combined scaled Jacobian and Jacobian method to avert the optimizer from getting stuck in local minima. Numerical results presented in Section 3 further indicate that our novel buffer zone clearance and mesh quality enhancement techniques lead to a significantly higher minimum scaled Jacobian (> 0.5).

3. Numerical results and discussion

We run HybridOctree_Hex on a range of complex models without the need for parameter adjustments and compare our algorithm with previous methods [25,26]. Twelve representative meshing outcomes are displayed in Figs. 6 and 7. Our results were computed on a PC equipped with a 3.6 GHz Intel i7-12700 CPU and 32 GB of memory.

The presented models demonstrate a range of intricate features such as high curvature and narrow regions, which are widely present in Fig. 6(b, d, e, f) through tips, as well as slim cylindrical structures in Fig. 7(b, e, f). High-genus topologies can also lead to narrow regions; see Figs. 6(a, e) and 7(e, f). If these narrow regions are not adequately detected and refined, the resulting mesh may exhibit holes, as exemplified in the zoom-in picture [26] of Fig. 6(c'). By incorporating the thickness measurement, our algorithm maintains the input topology faithfully. Note that the model's thickness bears a direct proportion to $cos\theta$, where θ is the angle between the thickness direction and the octree structure orientation. In the worst case following the long diagonal direction of the octree structure, we have $\theta = \arcsin(\sqrt{3}/3) \approx 35.3°$ and more elements are needed to preserve the narrow region. In the oil pump model (Genus-4) shown in Fig. 7(e), the octree orientation exhibits $\theta = 30°$ with the thin plate's thickness direction. Our algorithm demonstrates its ability to preserve the model's topology correctly without leaving holes on the thin plates, although it requires a higher mesh density. Figs. 6(c-f) and 7(a-d, f) showcase a diverse array of detailed features. Without proper detection and refinement, these features may appear blurred in the final mesh. Our findings reveal that the curvature function is remarkably proficient in pinpointing such crucial surface details, and the quality improvement algorithm faithfully reproduces them in the resulting meshes. This is evident in the representation of eyes in Figs. 6(c, f) and 7(a), as well as noses in Figs. 6(c, d, f) and 7(a, b).

Table 2 presents a thorough analysis of mesh statistics. When considering mesh size alongside refinement level, our method demonstrates superior mesh adaptation. This is achieved by employing local

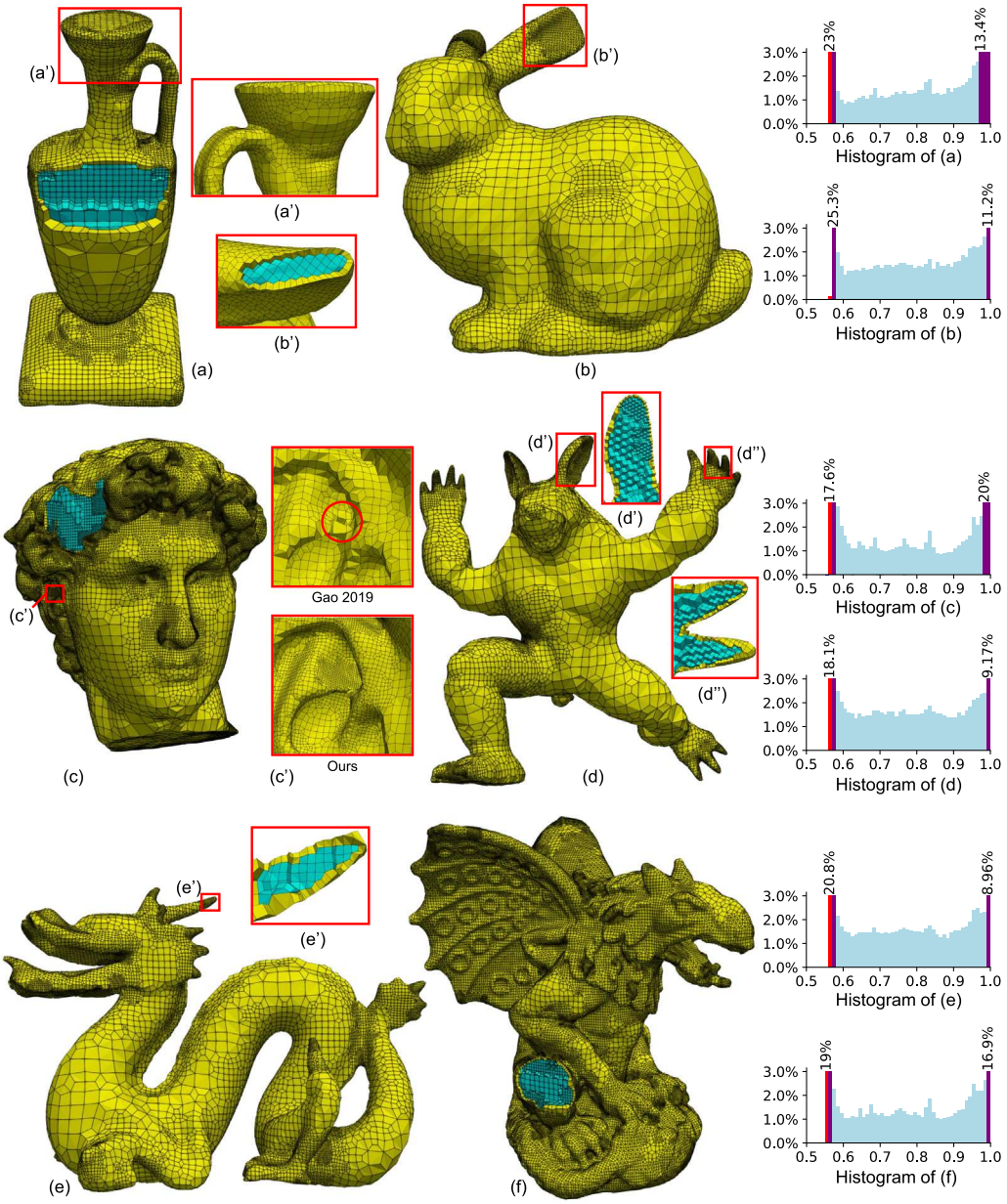

Fig. 6. (a) The bottle1; (b) The bunny; (c) The david; (d) The deformed armadillo; (e) The dragon stand; and (f) The gargoyle. (a', b', c', d', d'', e') show zoomed-in images; (c') shows a zoomed-in comparison between our mesh and [26] mesh, which generates a hole at the thin region; and the scaled Jacobian histograms are shown in the last column. The red bar represents the minimum scaled Jacobian and purple bars are truncated ones due to a higher frequency ($\geq 3\%$).

refinement exclusively in regions of high curvature and narrow regions, resulting in meshes with elevated refinement levels that minimize redundant elements. Through the utilization of face-normal-based core mesh surface enhancement, coupled with the Jacobian and scaled Jacobian optimization technique, we have attained notably improved minimum scaled Jacobians (> 0.5). Additionally, our method excels in terms of efficiency, generating meshes in the shortest time in most models compared to [25,26]. This is attributed to our strategic implementation of smart Laplacian smoothing, which is restricted to the outermost two layers of vertices and executed once every 1000 iterations. Previous methods often employ this technique for every mesh vertex on a global scale.

4. Conclusion and future work

In this paper, we have presented HybridOctree_Hex, a software package for adaptive all-hex mesh generation. Our approach leverages an octree structure to efficiently detect key surface features with curvature and thickness and construct a strongly balanced octree. From the strongly balanced octree, we directly generate an all-hex dual mesh using pre-defined templates, bypassing the implementation of sophisticated hybrid octree construction steps in [25]. Then we clear the buffer zone of the all-hex mesh with a face-normal-based surface improvement technique to obtain a core mesh with the $\min SJ$ of 0.258. After that, the buffer zone is meshed by connecting core mesh boundary points with their corresponding points on the input surface. In the quality improvement step, we fit the outmost points to the input surface by iteratively minimizing a comprehensive energy function composed of geometry fitting, scaled Jacobian, and Jacobian terms. Smart Laplacian smoothing is performed every 1000 iterations to smooth the outmost two layers of the vertices and drag surface points that are stuck in the local minima. Our empirical evaluation demonstrates the robustness and efficiency of HybridOctree_Hex, as it handles dozens of complex

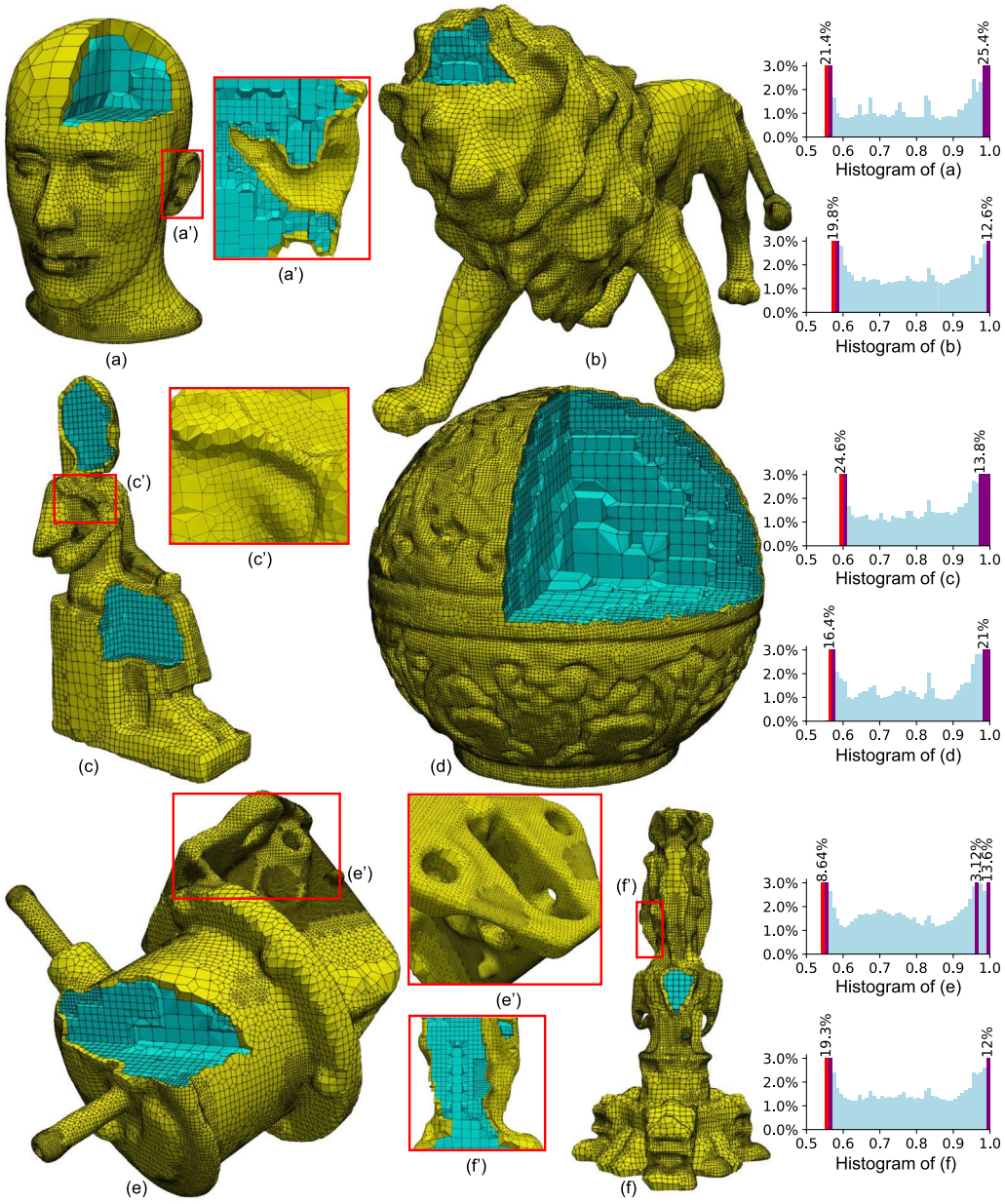

Fig. 7. (a) The head; (b) The lion recon; (c) The ramses; (d) The red circular box; (e) The oil pump; and (f) The thai statue. (a', c', e', f') demonstrate zoomed-in images; and the scaled Jacobian histograms are shown in the last column. The red bar represents the minimum scaled Jacobian, and the purple bars are truncated ones due to a higher frequency ($\geq 3\%$).

3D models without requiring manual intervention or parameter adjustment. Among all the models we tested, our algorithm generates meshes in the shortest time compared to [25,26], with high adaptation and the $\min SJ$ of > 0.5 after quality improvement. The source code and comprehensive statistical and meshing results are made available to facilitate further research and development in the field; see https://github.com/CMU-CBML/HybridOctree_Hex.

While HybridOctree_Hex has demonstrated promising results in terms of fast, robust, and high-quality all-hex mesh generation, there remain several avenues for future research. One such direction is the exploration of more advanced techniques for curvature and narrow region detection to further improve the identification of key surface features. In our current approach, we employ pre-assigned thresholds, set conservatively to ensure the correct topology of the resulting mesh. However, this can lead to unnecessarily dense elements. A possible solution lies in introducing machine learning, which has already shown potential in 2D quad mesh generation [39]. Neural networks can be leveraged to predict the octree level distribution on the surface, learning from both successful and failed adaptive octree configurations. Another area of focus is the energy function employed in the optimization process. In theory, the energy terms can be expanded to generate customized hex meshes. For example, we can substitute the Jacobian and scaled Jacobian terms with alternative mesh quality metrics, enabling control over mesh quality from diverse perspectives, such as anisotropic mesh generation. Furthermore, the geometry fitting term can be strengthened to preserve user-defined sharp features by fixing corner points and constraining edge points to move exclusively along feature edges. HybridOctree_Hex has already exhibited superior efficiency compared to previous methods [25,26]. Nevertheless, there is scope for further improvement in computational cost through the utilization of parallel computing, paving the way for real-time applications.

Table 2
Mesh statistics of the testing models.

Model	Method	Mesh Size (vertex ♯; element ♯)	Scaled Jacobian [worst; best]	Refinement Level	Time (s)
Bottle1	[26]	(39,326; 33,635)	[0.181; 0.999]	4	8,175
(Genus-1)	ours	**(36,091; 30,145)**	**[0.560; 1.0]**	4	**218**
Bunny	[25]	(46,476; 39,605)	[0.00100; 1.0]	3	462
	[26]	(35,330; 29,698)	[0.292; 0.999]	4	3,569
(Genus-0)	[36]	(119,799; 106,730)	$[3.85 \times 10^{-5}; 1.0]$	3	258
	ours	**(26,375; 21,695)**	**[0.570; 1.0]**	4	**358**
David	[26]	**(127,778; 112,314)**	[0.126; 0.999]	4	38,866
(Genus-0)	ours	(319,465; 282,957)	**[0.560; 1.0]**	6	**10,450**
Deformed Armadillo	[26]	(43,591; 35,611)	[0.0678; 0.999]	4	6,573
(Genus-0)	ours	**(43,216; 34,939)**	**[0.560; 1.0]**	5	**3,431**
Dragon Stand2	[26]	(74,618; 61,441)	[0.0290; 0.999]	5	23,062
(Genus-1)	ours	**(62,576; 50,853)**	**[0.560; 1.0]**	4	**2,052**
Gargoyle	[26]	**(157,008; 135,737)**	[0.0702; 0.999]	4	–
(Genus-0)	ours	(273,704; 236,689)	**[0.550; 1.0]**	4	8,769
Head	[36]	(64,258; 56,419)	[0.0130; 1.0]	3	**169**
(Genus-0)	ours	**(62,782; 55,038)**	**[0.550; 1.0]**	5	444
Lion Recon	[26]	(134,140; 115,245)	[0.178; 0.999]	4	18,755
(Genus-0)	ours	**(112,847; 95,251)**	**[0.570; 1.0]**	4	**2,046**
Oil Pump	[26]	**(75,033; 63,012)**	[0.117; 0.999]	4	14,301
(Genus-4)	ours	(233,702; 196,455)	**[0.540; 1.0]**	5	**9,742**
Ramses	[26]	(54,634; 46,923)	[0.0161; 0.999]	4	9,395
(Genus-0)	ours	**(44,790; 37,993)**	**[0.590; 1.0]**	4	**713**
Red Circular Box	[26]	(409,011; 367,583)	[0.215; 0.999]	4	71,626
(Genus-0)	ours	**(351,881; 313,866)**	**[0.560; 1.0]**	4	**7,547**
Thai Statue	[26]	(70,561; 59,470)	[0.180; 0.999]	4	26,075
(Genus-3)	ours	**(64,764; 53,831)**	**[0.550; 1.0]**	4	**1,845**

CRediT authorship contribution statement

Hua Tong: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Eni Halilaj:** Writing – review & editing, Project administration, Funding acquisition, Conceptualization. **Yongjie Jessica Zhang:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

H. Tong and Y. J. Zhang were supported in part by the US National Science Foundation (NSF), USA grant CMMI-1953323 and a Honda grant.

References

[1] Y.J. Zhang, Geometric Modeling and Mesh Generation from Scanned Images, CRC Press, Taylor & Francis Group, 2016.
[2] S.E. Benzley, E. Perry, K. Merkley, B. Clark, G. Sjaardama, A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis, in: 4th International Meshing Roundtable, 1995, pp. 179–191.
[3] J.F. Shepherd, C.J. Tuttle, C. Silva, Y. Zhang, Quality Improvement and Feature Capture in Hexahedral Meshes, Technical Report UUSCI-2006-029, The University of Utah, 2006.
[4] S.J. Owen, A survey of unstructured mesh generation technology, Int. Meshing Roundtable 239 (267) (1998) 15.
[5] Y. Zhang, Challenges and advances in image-based geometric modeling and mesh generation, in: Image-Based Geometric Modeling and Mesh Generation, Springer, 2013, pp. 1–10.
[6] R. Schneiders, Algorithms for quadrilateral and hexahedral mesh generation, in: Proceedings of the VKI Lecture Series on Computational Fluid Dynamic, 2000.
[7] T. Blacker, Meeting the challenge for automated conformal hexahedral meshing, in: 9th International Meshing Roundtable, Citeseer, 2000, pp. 11–20.
[8] T.J. Tautges, The generation of hexahedral meshes for assembly geometry: Survey and progress, Internat. J. Numer. Methods Engrg. 50 (12) (2001) 2617–2642.
[9] J.F. Shepherd, C.R. Johnson, Hexahedral mesh generation constraints, Eng. Comput. 24 (3) (2008) 195–213.
[10] J.F. Shepherd, Y. Zhang, C. Tuttle, C. Silva, Quality improvement and boolean-like cutting operations in hexahedral meshes, in: The 10th ISGG Conference on Numerical Grid Generation, 2007.
[11] J. Gregson, A. Sheffer, E. Zhang, All-hex mesh generation via volumetric polycube deformation, Comput. Graph. Forum 30 (5) (2011) 1407–1416.
[12] K. Hu, Y.J. Zhang, Centroidal voronoi tessellation based polycube construction for adaptive all-hexahedral mesh generation, Comput. Methods Appl. Mech. Engrg. 305 (2016) 405–421.
[13] K. Hu, Y.J. Zhang, T. Liao, Surface segmentation for polycube construction based on generalized centroidal voronoi tessellation, Comput. Methods Appl. Mech. Engrg. 316 (2017) 280–296.
[14] Y. Yu, X. Wei, A. Li, J.G. Liu, J. He, Y.J. Zhang, HexGen and Hex2Spline: Polycube-based hexahedral mesh generation and spline modeling for isogeometric analysis applications in LS-DYNA, in: Geometric Challenges in Isogeometric Analysis, Springer, 2022, pp. 333–363.
[15] M. Nieser, U. Reitebuch, K. Polthier, Cubecover–parameterization of 3D volumes, Comput. Graph. Forum 30 (5) (2011) 1397–1406.
[16] Y. Li, Y. Liu, W. Xu, W. Wang, B. Guo, All-hex meshing using singularity-restricted field, ACM Trans. Graph. 31 (6) (2012) 1–11.
[17] H. Liu, P. Zhang, E. Chien, J. Solomon, D. Bommes, Singularity-constrained octahedral fields for hexahedral meshing, ACM Trans. Graph. 37 (4) (2018) 1–17.
[18] Y. Zhang, C. Bajaj, B.-S. Sohn, 3D finite element meshing from imaging data, Comput. Methods Appl. Mech. Engrg. 194 (48–49) (2005) 5083–5106.
[19] Y. Zhang, C. Bajaj, Adaptive and quality quadrilateral/hexahedral meshing from volumetric data, Comput. Methods Appl. Mech. Engrg. 195 (9–12) (2006) 942–960.
[20] Y. Zhang, G. Xu, C. Bajaj, Quality meshing of implicit solvation models of biomolecular structures, Comput. Aided Geom. Design 23 (6) (2006) 510–530.
[21] Y. Zhang, T.J. Hughes, C.L. Bajaj, Automatic 3D meshing for a domain with multiple materials, in: 16th International Meshing Roundtable, 2008, pp. 367–386.
[22] Y. Zhang, T.J. Hughes, C.L. Bajaj, An automatic 3D mesh generation method for domains with multiple materials, Comput. Methods Appl. Mech. Engrg. 199 (5–8) (2010) 405–415.
[23] J. Qian, Y. Zhang, Automatic unstructured all-hexahedral mesh generation from B-reps for non-manifold CAD assemblies, Eng. Comput. 28 (2012) 345–359.

[24] L. Maréchal, Advances in octree-based all-hexahedral mesh generation: handling sharp features, in: 18th International Meshing Roundtable, 2009, pp. 65–84.
[25] K. Hu, J. Qian, Y. Zhang, Adaptive all-hexahedral mesh generation based on a hybrid octree and bubble packing, in: 22nd International Meshing Roundtable, Orlando, FL. Oct, 2013.
[26] X. Gao, H. Shen, D. Panozzo, Feature preserving octree-based hexahedral meshing, Comput. Graph. Forum 38 (5) (2019) 135–149.
[27] M. Livesu, L. Pitzalis, G. Cherchi, Optimal dual schemes for adaptive grid based hexmeshing, ACM Trans. Graph. 41 (2) (2021) 1–14.
[28] L. Pitzalis, M. Livesu, G. Cherchi, E. Gobbetti, R. Scateni, Generalized adaptive refinement for grid-based hexahedral meshing, ACM Trans. Graph. 40 (6) (2021) 1–13.
[29] S.A. Canann, J.R. Tristano, M.L. Staten, et al., An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes, Int. Meshing Rountable 1 (1998) 479–494.
[30] Y. Zhang, C. Bajaj, G. Xu, Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow, Commun. Numer. Methods. Eng. 25 (1) (2009) 1–18.
[31] L.A. Freitag, On Combining Laplacian and Optimization-Based Mesh Smoothing Techniques, Technical Report, Argonne National Lab, 1997.
[32] J. Qian, Y. Zhang, W. Wang, A.C. Lewis, M.S. Qidwai, A.B. Geltmacher, Quality improvement of non-manifold hexahedral meshes for critical feature determination of microstructure materials, Internat. J. Numer. Methods Engrg. 82 (11) (2010) 1406–1423.
[33] H. Lin, S. Jin, H. Liao, Q. Jian, Quality guaranteed all-hex mesh generation by a constrained volume iterative fitting algorithm, Comput. Aided Des. 67 (2015) 107–117.
[34] M. Meyer, M. Desbrun, P. Schröder, A.H. Barr, Discrete differential-geometry operators for triangulated 2-manifolds, in: Visualization and Mathematics III, Springer, 2003, pp. 35–57.
[35] P. Guigue, O. Devillers, Fast and robust triangle-triangle overlap test using orientation predicates, J. Graph. Tools 8 (1) (2003) 25–32.
[36] Y. Zhang, X. Liang, G. Xu, A robust 2-refinement algorithm in octree or rhombic dodecahedral tree based all-hexahedral mesh generation, Comput. Methods Appl. Mech. Engrg. 256 (2013) 88–100.
[37] N. Paragios, M. Rousson, V. Ramesh, Matching distance functions: A shape-to-area variational approach for global-to-local registration, in: 7th European Conference on Computer Vision, 2002, pp. 775–789.
[38] M. Rousson, N. Paragios, Shape priors for level set representations, in: 7th European Conference on Computer Vision, 2002, pp. 78–92.
[39] H. Tong, K. Qian, E. Halilaj, Y.J. Zhang, SRL-assisted AFM: Generating planar unstructured quadrilateral meshes with supervised and reinforcement learning-assisted advancing front method, J. Comput. Sci. 72 (2023) 102109.

Hua Tong received his B.S. in Engineering Science from the University of Science and Technology of China in 2022. He is currently a Ph.D. student in the Department of Mechanical Engineering at Carnegie Mellon University. His research interests include mesh generation and machine learning.

Eni Halilaj received her B.A. in Engineering and Italian Studies in 2008 and Ph.D. in Biomedical Engineering in 2015 from Brown University. She became an assistant professor in Mechanical Engineering at Carnegie Mellon University with courtesy appointments in Biomedical Engineering and Robotics Institute in 2018. Her research interests include biomechanics, imaging, modeling, orthopedics, and osteoarthritis.

Yongjie Jessica Zhang holds B.Eng. in Automotive Engineering and M.Eng. in Engineering Mechanics from Tsinghua University; and M.Eng. in Aerospace Engineering and Engineering Mechanics and Ph.D. in Computational Engineering and Sciences from the Institute for Computational Engineering and Sciences, The University of Texas at Austin. She is the George Tallman Ladd and Florence Barrett Ladd Professor of Mechanical Engineering, Carnegie Mellon University with a courtesy appointment in Biomedical Engineering. Her research interests include computational geometry, isogeometric analysis, finite element method, data-driven modeling, image processing and mesh generation.