Marching Windows: Scalable Mesh Generation for Volumetric Data with Multiple Materials

Wenhua Zhang, Yating Yue, Hao Pan, Zhonggui Chen, Chuan Wang, Hanspeter Pfister, *Senior Member, IEEE,* and Wenping Wang *Fellow, IEEE*

Abstract—Volumetric data abounds in medical imaging and other fields. With the improved imaging quality and the increased resolution, volumetric datasets are getting so large that the existing tools have become inadequate for processing and analyzing the data. Here we consider the problem of computing tetrahedral meshes to represent large volumetric datasets with labeled multiple materials, which are often encountered in medical imaging or microscopy optical slice tomography. Such tetrahedral meshes are a more compact and expressive geometric representation so are in demand for efficient visualization and simulation of the data, which are impossible if the original large volumetric data are used directly due to the large memory requirement. Existing methods for meshing volumetric data are not scalable for handling large datasets due to their sheer demand on excessively large run-time memory or failure to produce a tet-mesh that preserves the multi-material structure of the original volumetric data. In this paper we propose a novel approach, called *Marching Windows*, that uses a moving window and a disk-swap strategy to reduce the run-time memory footprint, devise a new scheme that guarantees to preserve the topological structure of the original dataset, and adopt an error-guided optimization technique to improve both geometric approximation error and mesh quality. Extensive experiments show that our method is capable of processing very large volumetric datasets beyond the capability of the existing methods and producing tetrahedral meshes of high quality.

Index Terms—large volumetric data, multiple material, marching windows, mesh simplification, topology guarantee.

1 Introduction

Precise 3D imaging, such as CT, MRI, or electron microscopy, is increasingly used in medicine, science, and engineering, producing extremely large 3D high-resolution volumetric data that contain multiple materials with rich delicate structural details. For example, Fig. 1 shows a 3D image of dimension $1024\times1024\times100$ with 400 distinct material labels, and this is just a small part of an EM (Electron Microscopy) scanned connectome data set. The sheer large size of such data in the voxel format precludes efficient visualization analysis and simulation of such 3D image data. In this paper, we address this challenge by developing a pipeline for converting large-scale volumetric data into a compact tetrahedral mesh that preserves the structural detail of the data, which would facilitate the visualization, analysis, and simulation of large-scale volumetric data.

Meshing a large volumetric data set is a challenging problem. For example, directly triangulating the volumetric data in Fig. 1 into a tetrahedral mesh by connecting neighboring vertices would require around 420 GB of memory, making it impossible to load the mesh into memory for visualization or simulation. Therefore, when meshing volumetric data, it is imperative to produce a compact mesh representation that is storage-efficient and to minimize memory footprints during computation so that the mesh can be computed on a common computing platform. In addition, large-scale volumetric data, especially those in medical imaging, contain numerous complex regions of different materials, such as axons and dendrites. These intricate and non-manifold features form a complex topology that is critical in imaging analysis. Hence, generating compact tetrahedral meshes from such imaging data, which can be viewed as a simplification procedure, must accurately represent the boundary surfaces of different regions and faithfully preserve the topological features of these regions.

To recap, the problem of computing a tetrahedral mesh to represent a large-scale volumetric imaging data set with labelled multiple materials needs to meet the following requirements:

1) *Mesh size*: the output tet-mesh should be compact so can be used readily for visualization, analysis, and simulation;
2) *Memory footprint*: given the large size of the volumetric data, the algorithm needs to be memory-friendly, using a sufficiently small memory foot-print that so it can run on a general computer;
3) *Structure preservation*: the output mesh needs to preserve the structure of the regions of multiple materials contained in the original volumetric data;
4) *Boundary approximation*: the mesh needs to accurately

- W. Zhang, Y. Yue, and C. Wang are with the Department of Computer Science, The University of Hong Kong, Hong Kong SAR, China. E-mail: winniezhangcoding@gmail.com, ytyue@cs.hku.hk, and cwang.hku@gmail.com
- H. Pan is with Microsoft Research Asia, Beijing, China. E-mail: haopan@microsoft.com
- Z. Chen is with the School of Informatics, Xiamen University, Xiamen, China. E-mail: chenzhonggui@xmu.edu.cn
- H. Pfister is with the John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. E-mail: pfister@g.harvard.edu
- W. Wang is with the Department of Visualization, Texas A&M University, Texas, USA. E-mail: wenping@tamu.edu

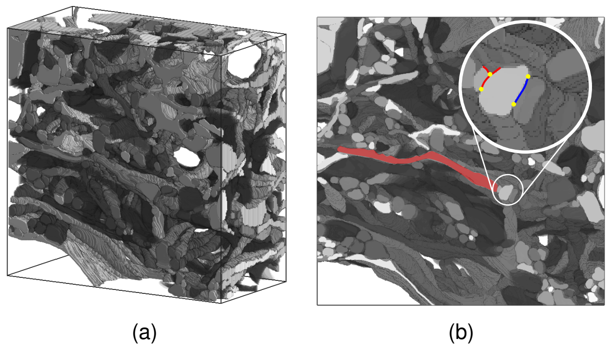


Fig. 1. (a) A part of 3D volumetric connectome data containing 400 regions of distinct material. The data has the resolution $1024 \times 1024 \times 400$. (b) One 2D slice of the imaging data in (a) of resolution 1024×1024, revealing numerous details and features. A slender feature is shown in red. Some feature lines and points are shown in close-up.

approximate the boundary surfaces of different regions;

5) *Mesh quality*: the quality of the tet-mesh elements needs to be as high as possible, as required for accurate and robust simulation.

We propose a memory-efficient and scalable method, called *Marching Windows*, for converting large-scale volumetric data with numerous regions of multiple materials into a compact, feature-preserving, and high-quality tetrahedral mesh. By *compact*, we mean that our generated meshes are much coarser than the original mesh, with usually less than 1% of vertices. Our result meshes are *tet-meshes* which are composed of tetrahedra that only intersect at boundary elements of lower dimensions (vertices, edges, triangle faces). They are commonly used by offering a good compromise between simplicity of mesh generation, generality, the ability to conform to complex geometries and numerics [1].

Marching Windows does not need to load the entire volumetric data into memory for processing. Instead, it uses a marching window for local mesh generation and a disk-swap scheme for I/O management. As a result, it is capable of processing arbitrarily large volumetric data while requiring only a moderate amount of runtime memory. For example, for the dataset shown in Fig. 1, *Marching Windows* converts it into a high-quality tet mesh using only 1.2 GB runtime memory, with a window of size $40 \times 40 \times 40$. *Marching Windows* is also topologically faithful and geometrically accurate. The connectivity within a region of the same material and the boundary surfaces between different regions are guaranteed to be preserved (see Figs. 16, 17, 18, and 19). Furthermore, *Marching Windows* produces high-quality mesh elements (see Sec. 6 for comparison).

The *Marching Windows* method is enabled by the following key design considerations. Firstly, we use a *marching window* for local mesh generation and a disk-swap scheme to accommodate large-scale volumetric data. As shown in Fig. 2, we sequentially process fixed-size data bounded in the marching window without compromising the quality of generated meshes. To this end, we designed a specific marching strategy that moves the window by half the window size and fixes the vertices on the boundary at each step. In addition, as the working window proceeds, we swap out the inactive data to the disk of external storage and swap in new active data to memory. In comparison, as will be reviewed in Sec. 2, existing direct mesh generation methods generally have to load the entire dataset at once, so would be limited by the available computational resources. In addition, the naive method of cutting the data into small blocks and processing one block at a time would lead to degraded mesh quality due to the arbitrary boundary cut.

Secondly, we reduce the size of the initial dense mesh induced by the voxel grid through iterative edge contraction and introduce effective criteria to ensure topology correctness. In comparison, the widely used Delaunay refinement methods [2], [3], [4] for quality mesh generation have difficulty in preserving intricate topology and geometric details (see Fig. 17, 18, and 19). Similar issues arise for particle-based mesh generation [5], which samples more densely at feature regions to reduce topological and geometric errors but without guarantees.

Finally, the *Marching Windows* algorithm generates high-quality tetrahedral meshes by selectively contracting edges to improve mesh quality. The simplification is also constrained by density variation (Sec. 5.4) to achieve smooth gradation of mesh elements. After simplification, we further apply the constrained Optimal Delaunay Triangulation (CODT) with constrained boundary faces for several iterations to improve mesh quality [6], [7], [8], [9]. As a result, the resulting mesh quality is comparable to more sophisticated meshing methods where they are applicable (Sec. 6.3).

In summary, *Marching Windows* is the first mesh-generation method that handles very large volumetric datasets with multiple labeled materials. It runs on modest and easily accessible computational resources, and produces high-quality meshes that preserve the connectivity topology and accurately approximate the boundary surfaces of the input volumetric data. We will release the code for *Marching Windows* soon.

2 Related Work

There are many works on generating 3D tetrahedral meshes from volumetric data; here we review those most relevant to our problem: meshing volumetric data with multiple materials. According to the general methodology taken, we broadly divide the related works into three categories, i.e., Delaunay-based methods, variational methods, lattice-based methods, and mesh simplification methods.

2.1 Delaunay-based Methods

Delaunay refinement [2], [3], [4] is able to generate high-quality meshes via repeatedly inserting points and building Delaunay triangulation after each insertion. For multi-material data, it demands extra effort to preserve the feature lines and surfaces between different materials. Conforming or constrained Delaunay refinement are generally used to handle the feature constraints [10], [11], [12], [13], [14], [15], which require the specification of feature lines as input. [16], [17] use protective balls around the features edges during the refinement process. However, small features like isolated voxels do not have feature lines and can be missed by refinement.

In contrast, our simplification-based approach always preserves the features and surfaces between materials by

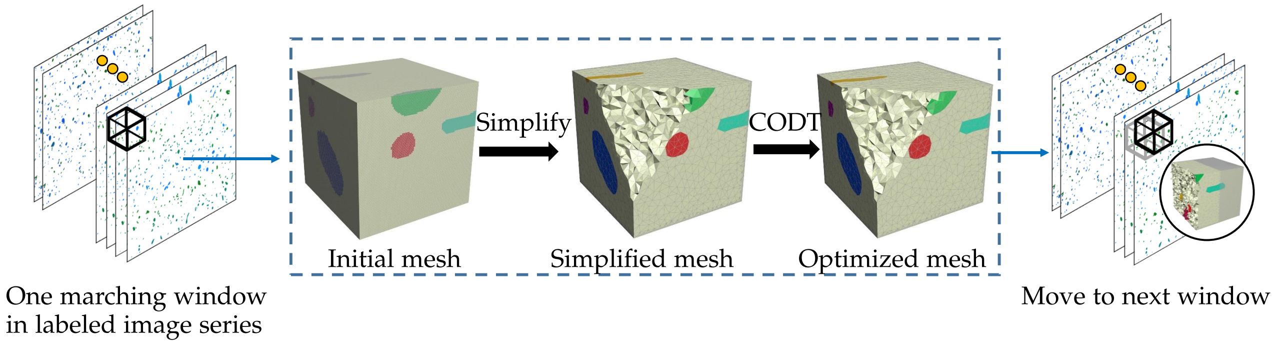


Fig. 2. Overview of the pipeline. Following the *marching windows* approach (Sec. 4), inside each window, (a) we first generate an initial mesh from the bounded input image stack (Sec. 5.1), (b) then simplify it to reduce data size and produce a quality mesh with geometric fidelity (Sec. 5.2, 5.3, and 5.4), and (c) apply CODT iterations to further improve the mesh quality (Sec. 5.5). Both simplification and CODT optimization are guaranteed to preserve material subdomain topology. We show cut-off views of the tetrahedral meshes to better illustrate the mesh improvements.

topology filtering, ensuring topology correctness and keeping geometric fidelity while promoting mesh quality during simplification.

2.2 Variational Methods

Starting from an initial triangulation, variational methods optimize the triangulation by moving vertices and updating the connectivity of vertices, guided by minimizing objective functions that measure the mesh quality and constrained by necessary boundary conditions. Many geometric measures have been proposed for objective functions [18]. Among them, the methods based on centroidal Voronoi tessellation [19], [20] and optimal Delaunay triangulation (ODT) [6], [7], [8], [9] have been proved effective for generating meshes of higher quality than other methods. The problem with these methods is that the collective vertex and connection update are non-local and therefore computationally expensive, which would be unacceptable for large-scale data. We also use the effective ODT method to improve mesh quality, but only after the mesh is simplified significantly. Only a few iterations of ODT applied to the coarse mesh of reasonably good quality are sufficient to give us a final mesh of high quality.

While most variational methods mesh a single material homogeneous volume, Meyer et al. [5] focus on multi-material meshing. They use a variational particle-based method to first generate a good meshing of the surfaces between materials and then construct a tetrahedral mesh within each material region using constrained Delaunay refinement. To properly sample the surfaces with an appropriate number of particles, they rely on a sizing field defined by local feature size [21] which involves the computation of the medial axis of the material regions. Computing the medial axis is a non-local process that not only takes much time but also requires loading a complete material region at once, which is computationally infeasible for very large-scale data sets. Also, [5] uses pre-processing to remove very thin regions of the data so that the particle-based optimization could work properly. However, this alters the geometric details and topology of the input data, which is critical for downstream tasks like medical analysis. In comparison, our method requires no such complex pre-processing, ensures correct topology, preserves geometric details, and can process datasets of a very large scale.

2.3 Lattice-based Methods

Bronson et al. [1] describe a meshing algorithm for generating tetrahedral meshes for multi-material domains. They rely on a regular background lattice that is subdivided (or cleaved) to conform to the material boundaries. The output meshes approximately conform to interfaces between materials and have high-quality tetrahedral elements with guaranteed fidelity to sufficiently large features. However, it could not deal with large-scale datasets like Mito, as it needs to compute on the whole mesh but loading the large datasets entirely into memory is hardly feasible.

2.4 Mesh Simplification Methods

Mesh simplification methods usually start with a dense mesh and gradually remove vertices from the mesh according to certain criteria. Therefore, it is suitable to use simplification in our solution, while we design novel criteria to preserve the complicated details and features in the final coarse mesh.

Many simplification algorithms [22], [23], [24], [25] have been developed for simplifying triangular surface meshes. The basic ideas of tetrahedral mesh simplification are similar [26], [27], [28], [29], [30]; these works can be divided into several categories: vertex clustering, vertex decimation, and edge contraction. The quadric error metric (QEM) method [24] is the most prevailing algorithm in this category due to its simplicity and superior efficiency. Over the years, numerous works have improved and extended QEM [31], [32], [33], [34], [35], [36]. Our approach is based on edge contraction with a customized error metric similar to QEM to achieve the goal of generating high-fidelity surfaces and high-quality mesh elements simultaneously; see Sec. 5 for details.

Moore et. al [37] present a simplification-based method to reconstruct surface models from multi-material volumetric data. To compare, our method generates tetrahedral meshes that contain surface models, and achieves high mesh quality and topology correctness in a simpler manner. For example, [37] adds quality control to QEM through a scale-invariant quality metric, to evaluate the triangle quality according to the area and edge length. However, the traditional QEM cannot simplify flat surfaces since costs there are always zero. Therefore, [37] perturbs planes specifically by adding small random noise. In contrast, our error metric combines QEM with an extended error matrix (Sec. 5.2) that measures element quality, so we can simplify flat surfaces and volume consistently and obtain quality results.

Faraj et al. [38] solve the remeshing of multi-material tetrahedral meshes by iterative applications of local operations, including edge split/collapse, face flipping, and vertex smoothing, that are adapted to preserve the material subdomains. While their overall paradigm shares similarity with our simplification-based approach, we focus on processing large-scale data that is challenging for their remesher [38] but handled with ease by our *Marching Windows* approach (Table 2). Moreover, we note that their topological filtering is more restricted than our criteria (Fig. 9).

Vivodtzev et. al [39] also aim to simplify the tetrahedral meshes while preserving the feature substructure topology. They extend link conditions [40] to multiple-material feature complexes to check if an edge can be collapsed without changing feature topology. The correctness of the extended link conditions for multi-material simplification is proved in [41]. However, similar to MAD [38], the conditions are sufficient for preserving the volume topology but not necessary, as they are designed to focus on keeping the topology of the embedded surfaces instead of the subdomains volumes. For example, as shown in Fig. 9, the edge v_0v_1 is not collapsible with their method[1], while it can be handled properly with our algorithm.

2.5 Distributed Mesh Generation

Distributed mesh generation methods [42], [43], [44], [45] aim to generate large-size meshes with distributed and parallel computational resources. They usually start with an initial coarse mesh and then decompose the geometry into multiple sub-geometries, the meshing of which are dispatched to multiple computational units. Thus, these works are often more focused on mesh partitioning, load balancing, and partition boundary mesh face matching. Our method avoids these challenges by processing the data sequentially with our memory-disk swapping scheme. Besides, it is difficult for these methods to generate coarse meshes first with very large-scale and complex data like Mito [46], while our method does not build an initial coarse mesh and can easily process large datasets sequentially.

3 Overview

Fig. 2 shows the pipeline of our fast mesh generation algorithm for large-scale multi-material volumetric data. Given a volumetric input data consisting of a series of 2D sections, our algorithm adopts *marching windows* to process the data blocks sequentially (Sec. 4). Within each window, the mesh generation consists of three steps:

1) generate an initial tetrahedral mesh by subdividing the input voxels (Sec. 5.1);
2) simplify the mesh while capturing both the labeled volumetric regions and the surfaces between them with topology guarantee, quality control, and density variation (Sec. 5.2, 5.3, and 5.4);
3) further improve the quality of the simplified tetrahedron mesh with constrained ODT iterations (Sec. 5.5).

1. Precisely, $\mathrm{Lk}_0 v_0 \cap \mathrm{Lk}_0 v_1 \neq \mathrm{Lk}_0 v_0 v_1$ due to the existence of the bottom right vertex v_3 in Fig. 9(a).

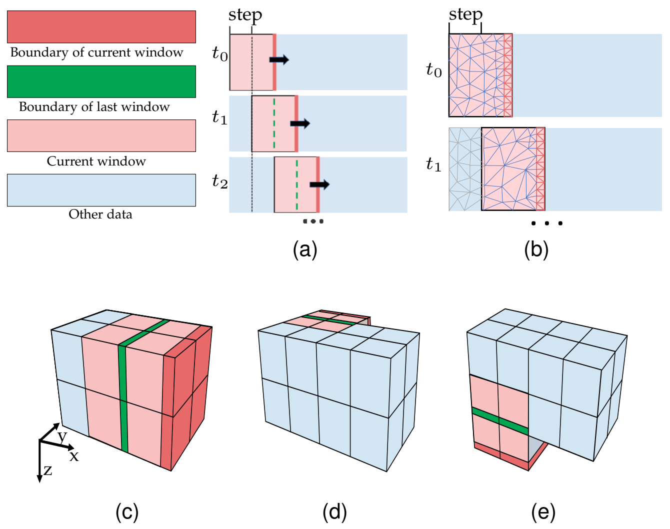


Fig. 3. Window marching strategy illustration. (a) 2D illustration of the window marching steps, with (b) mesh generation in each window. (c) When moving one step along the x axis in 3D, the marching boundary of the x axis will be included in the new window and simplified. (d) The window moves along y axis when it reaches the end of x axis. (e) The window moves along z axis when it reaches the end of y axis.

Section 4 focuses on the window marching strategy and the memory-storage swapping method, while Section 5 presents our mesh simplification and optimization algorithms within each window. We comprehensively evaluate the proposed method in Section 6 and conclude the paper in Section 7.

4 Scalable Processing by *Marching Windows*

Traditional algorithms usually require data to be entirely loaded into memory before processing, which becomes impractical when handling very large-scale data (see Fig. 1 and Table 1). Therefore, we adopt a *marching windows* strategy to solve this challenge: we locally generate a small part of the mesh at a time; as the window ranges over the entire data volume, a compact mesh is generated while the memory cost is upper-bounded by the size of the window. To ensure that the generated meshes within different windows are consistent, the marching process overlaps windows with special care; to swap in and out the data blocks for memory saving, we use on-demand data management. We present the details of the process next.

4.1 Window Marching Strategy

Assuming that the volumetric data has an axis-aligned box shape in 3D space, the marching windows move along the x, y, z axes sequentially, advancing for half of the window's side length for the corresponding axis at each step. For example, for a volume data of size $X \times Y \times Z$ and a window size $W \times H \times D$, where W, H, D are even numbers, there are $\lceil \frac{2X}{W} \rceil \times \lceil \frac{2Y}{H} \rceil \times \lceil \frac{2Z}{D} \rceil$ steps in the process.

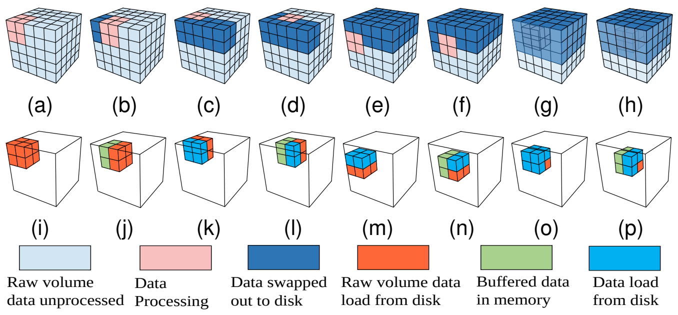


Fig. 4. Memory-disk data swapping. After processing in each window, the program will swap out some of the data to the disk and swap in new data to the memory accordingly. (a)-(h) The moving routes along x, y, z axes, respectively. (i)-(p) The data source for each window. Note that each window below corresponds to the processing window(in pink) above respectively.

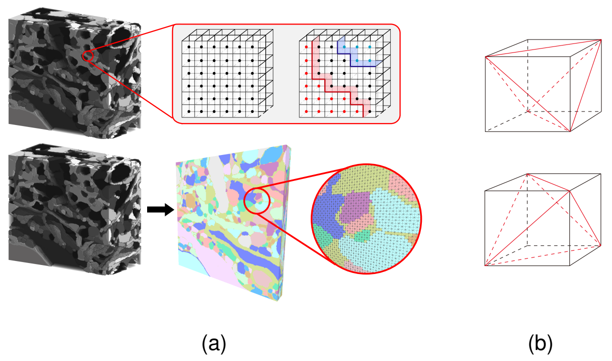


Fig. 5. Generating the initial mesh from the multiple labeled volumetric data. (a)The input data with the constructed cubes (the top one) and the input data with the constructed mesh (the bottom one). (b) Dividing a cube into tetrahedra. The cubes are divided into 5 tetrahedra with two different ways. Adjacent cubes use the methods in (b) respectively so that the interfaces of cubes are consistent.

The meshing inside a window must be carried out in such a manner that the elements are conforming across the windows. To achieve this target, first, the newly included voxels inside a window are subdivided into tetrahedra in two alternating ways depending on the parity of the global voxel index (see Fig. 5). We call the mesh inside the window at step t as active mesh and denote it as $M_t = (V_t, \Sigma_t)$, where $\Sigma_t = \{\sigma_j = (v_{j1}, v_{j2}, v_{j3}, v_{j4}) \subset V_t\}$ is the set of tetrahedra intersected by the current window, and V_t is the corresponding active vertex set. Then, we maintain a set of boundary elements that are constrained during the simplification and optimization of M_t, and only process them when they become interior elements in later steps. To be specific, the constrained elements are the tetrahedra in Σ_t which are touched or crossed by the window boundaries (excluding the data volume boundaries); the vertices of these constrained tetrahedra are skipped by simplification and optimization (Sec. 5), while the mesh is processed into $M'_t = (V'_t \subset V_t, \Sigma'_t)$ and saved as a part of the final result. Fig. 3 illustrates the moving routes of *Marching Windows* in 3D and the constrained boundary elements in 2D.

4.2 Memory-Disk Data Swapping

We limit memory usage during the process by a memory-disk swapping method based on the smallest data management unit, a data block of $\frac{1}{8}$ size of the marching window, as shown in Fig. 4. After the simplification procedure within each window, we divide that window into eight data blocks. The association of processed data blocks and their mesh elements is maintained by a table mapping the blocks to their contained tetrahedra. In particular, the data blocks are identified by their global positions inside the whole data volume. The mesh vertices are identified by their global indices as voxel corners. Each tetrahedron is identified by its four vertices stored into the entry of the data block that intersects it.

After processing in each window, the processed data blocks are either cached in memory if it appears in the next window, or swapped out to disk to save memory. It will be swapped into memory later if it becomes part of the active window again. We move to the next window by combining the cached data and new data from the disk, where the new data is either raw data or processed data, according to the position of the active window. See Fig. 4 for an illustration of the different memory-disk swapping scenarios.

With this swapping strategy, we are able to constrain the memory usage of the program. Table 2 compares the maximum memory usages of *Marching Windows* and of processing the entire data set at once, and shows that memory usage is drastically reduced by our method.

5 Mesh Generation within a Window

Within each window, we apply a fast mesh generation process that produces high quality elements in one pass. In particular, we adopt a mesh simplification plus optimization method, with designs that guarantee the topology and preserve the geometry of the multi-material subdomains.

5.1 Mesh Initialization

For each raw data block, we can construct a high-resolution initial mesh to capture all the details present in the data. To be specific, each voxel of the input volumetric data is subdivided into 5 tetrahedra, as shown in Fig. 5; two adjacent cubes are subdivided in ways that are symmetric by reflection so that the interface between them is consistent. The material of a tetrahedron is derived from its original voxel. A triangle face is a surface separating two subdomains if it is adjacent to two tetrahedra with different material labels. Note that this simple mesh construction method is favorable than traditional methods like Marching Cubes [47] and Marching Tetrahedra [48], as they are nontrivial to extend to multi-material data without possibly missing detailed features.

The simplification-based method allows us to start with a high-resolution initial tetrahedral mesh to obtain a more compact mesh, while emphasizing the mesh quality and preserving the topology and geometry of the material subdomains. To this end, we apply edge collapsing filters to ensure the topology correctness of simplified mesh, and utilize an extended error metric of the classic Quadratic Error Minimization (QEM) method [24], [33] to preserve the subdomain geometry. We also achieve smooth grading of tetrahedra by target edge variance control.

To adapt the simplification process for complicated multi-material data, we modulate the process to achieve the following targets:

1) the topology of material subdomains should be strictly preserved;
2) the shapes of subdomains described by their boundary surfaces should be well preserved;
3) the mesh elements should be of high quality.

5.2 QEM Driven Simplification

The key issue of mesh simplification is to determine the order of contracting the mesh edges. Following the QEM framework, every vertex v is associated with a quadratic form $\mathbf{x}^T \mathbf{Q}_v \mathbf{x}$, where the variable $\mathbf{x} = (x_0, x_1, x_2, 1)^T \in \mathbf{R}^4$ is a homogeneous coordinate vector, and $\mathbf{Q}_v$ is a symmetric 4×4 matrix. In each iteration of the simplification, a set of collapsible edges are filtered by structural validity constraints; the collapsible edges are then evaluated for geometric errors, with the error metric of an edge $v_i v_j$ derived from its two endpoints as $\min_{\mathbf{x}} \mathbf{x}^T \left(\mathbf{Q}_i + \mathbf{Q}_j\right) \mathbf{x}$, where the minimization has a closed form solution due to the positive definiteness of $\mathbf{Q}$; the minimum error edge $e^* = v_p v_q$ is then collapsed by merging for example v_q into v_p, replacing the references of v_q to v_p for the tetrahedra containing v_q, updating $\mathbf{v}_p$ as the optimal solution $\mathbf{x}^*$ of the error metric, and assigning $\mathbf{Q}_p \leftarrow \mathbf{Q}_p + \mathbf{Q}_q$ for later error evaluation.

The QEM framework has been proposed with a variety of quadratic errors, including the traditional error measuring the sum of squared distances of a vertex to the planes of its supporting triangles [24], and also a Euclidean distance error measuring the sum of squared distances of a vertex to the data points associated to the vertex [33]. In our algorithm, we combine these two types of quadratic errors to measure both the approximation to the interfaces and the quality of mesh elements. To be specific, for a surface vertex v we define the surface error quadratic form matrix as

$$\mathbf{Q}_v^s = \sum_{\Delta_j \sim v} \mathbf{g}_j \mathbf{g}_j^T, \tag{1}$$

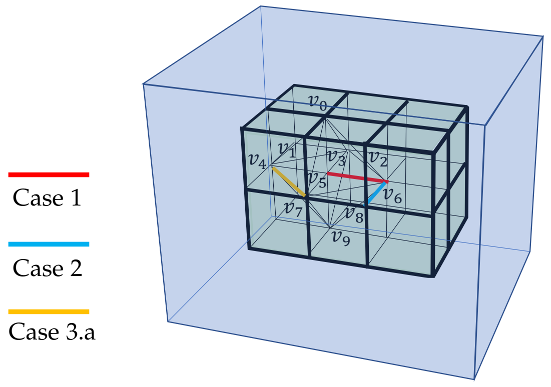
where Δ_j is the j-th interface triangle neighboring v, and $\mathbf{g}_j$ is a 4-dimensional vector representing the triangle's plane equation coefficients, i.e., $\mathbf{g}_j^T \mathbf{x} = 0$ is the plane equation containing the triangle and the first three components of $\mathbf{g}_j$ form the unit normal vector of the plane. The element quality error term is defined as $\mathbf{x}^T \mathbf{Q}_v^d \mathbf{x} = \|\mathbf{x} - \tilde{\mathbf{v}}\|^2$, with $\tilde{\mathbf{v}}$ the homogeneous coordinates of the vertex position; therefore,

$$\mathbf{Q}_v^d = \begin{pmatrix} \mathbf{I} & -\mathbf{v} \\ -\mathbf{v}^T & \|\mathbf{v}\|^2 \end{pmatrix}, \tag{2}$$

where $\mathbf{I}$ is the 3×3 identity matrix. The total quadratic error matrix for a surface vertex (order > 1) is $\mathbf{Q}_v^s + \mathbf{Q}_v^d$; an interior vertex instead only has the element quality error term $\mathbf{Q}_v^d$.

5.3 Multi-Material Topology Preservation

The topology of each material subdomain is a fundamental feature of the complex data sets, and should be preserved strictly by meshing. However, unfiltered edge collapses



(a)

Fig. 6. Illustration of simple cases filtered by our algorithm. Depicted is a background material (blue box) containing an inner block as an isolated region. We omit meshes of the background material for simplicity. (a) Case 1: both vertices are inner vertices, $e = v_5 v_6$; Case 2: one vertex is inner vertex and the other one is boundary vertex, $e = v_6 v_8$; Case 3.a: order of the edge is smaller than both endpoint vertices $e = v_4 v_7$.

in the dense mesh simplification process may introduce arbitrary topology changes. In this section, we present algorithms to restrict feasible edge collapses and preserve the subdomain topologies. In particular, similar to [38], we classify the boundary elements and interior elements, and filter out obvious violations of topology by type checking. In addition, for complex scenarios around the boundary regions that cannot be determined by type checking, we further use the dual simplicial complex [49], [50] to track and guarantee the topology preservation.

5.3.1 Element types

First, we define the *order* of a mesh element (vertex, edge, face) as the number of distinct materials of its adjacent tetrahedra. Based on the notion of order, we further define the following types.

- **Inner face, edge, and vertex** have order 1.
- **Boundary face**: We define two kinds of boundary faces here. Material boundary face is a face that is adjacent to two tetrahedra of different labels and has order 2, e.g. face $v_0 v_1 v_2$ in Fig. 6. Volume boundary face is a face that is on the boundary of the volume of the data, which has order 1. We use the term *boundary faces* to denote the union of the two kinds of faces in the following description.
- **Boundary edge** is an edge of a boundary face, either material boundary face or volume boundary face. A boundary edge of a material boundary face has order ≥ 2, e.g., edge $v_0 v_1$ in Fig. 6. While a boundary edge of a volume boundary face could have order 1.
- **Boundary vertex** is an endpoint of a boundary edge. If the boundary vertex is on a boundary edge of a material boundary face, it has order ≥ 2, e.g. vertex v_0 in Fig. 6. Otherwise, it has the order of 1.

5.3.2 Dual graph

Following [49], [50], each primal tetrahedron has a *dual vertex*, each primal face has a *dual edge*, each primal edge has a *dual face*, and each primal vertex has a *dual polyhedron*. To

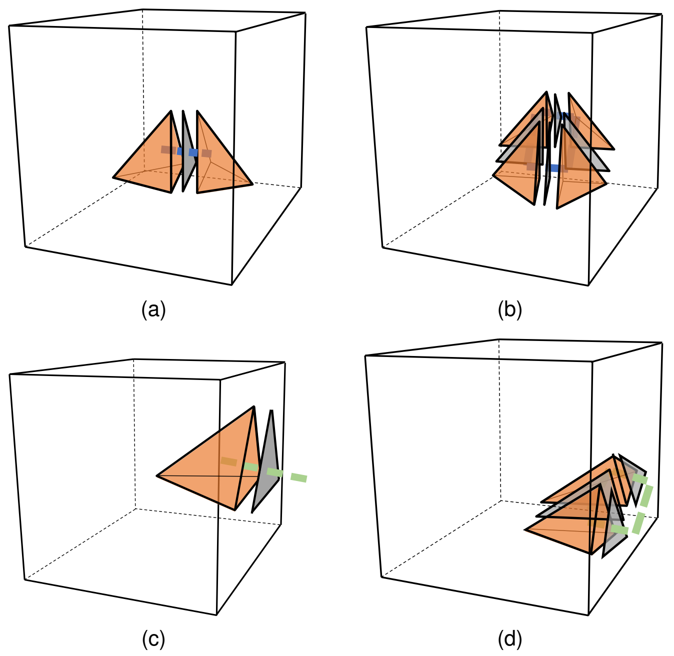


Fig. 7. Illustration of dual edges and faces. (a) The blue dotted line illustrates the material-dual edge of the face. (b) The face surrounded by the four blue dotted lines illustrates the material-dual face of the edge. Here each blue dotted line represents a material-dual edge of the corresponding face. (c) and (d) illustrate domain-dual edge and domain-dual face, respectively.

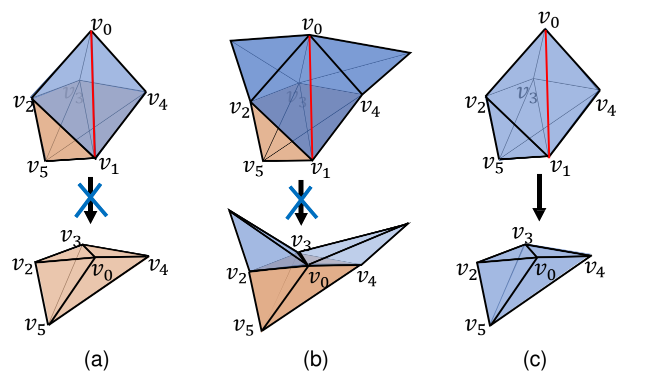


Fig. 8. Case 3.b examples, trying to collapse v_0v_1. Colors indicate different materials. (a) when neither of $v_0v_2v_3$, $v_1v_2v_3$, $v_0v_3v_4$, $v_1v_3v_4$ has dual edges, the blue component is in isolation and v_0v_1 is not collapsible. (b) when $v_0v_2v_3$, $v_0v_3v_4$ has dual edges, the blue component is not isolated, but since $v_0v_1v_3$ has the dual edge and neither does v_0v_3 nor v_1v_3 has dual face, v_0v_1 is not collapsible; otherwise collapsing it will cut off the connection through $v_0v_1v_3$. (c) when $v_1v_2v_3$ and $v_1v_3v_4$ have dual edges, it is not isolated; furthermore, since $v_0v_1v_3$ has dual edge and v_1v_3 has dual face, v_0v_1 is collapsible.

handle multiple materials, here we define the *material-dual* elements only for primal tetrahedra of the same material, i.e., the dual vertices forming the dual edges and faces must correspond to primal tetrahedra of the same label, as shown in Fig. 7. Intuitively, the dual edges and faces ensure that a primal tetrahedron is connected to other tetrahedra in the subdomain, thus providing a measurement of subdomain topology. Precisely for each subdomain, the nerve theorem states that the dual graph is homotopy equivalent to the primal tetrahedral mesh [49].

Additionally, to preserve domain boundary, we define *domain-dual* edges and faces. For a primal face on the boundary of the domain, it is assumed to have a domain-dual edge, that connects the dual vertex of its only adjacent tetrahedron with a virtual tetrahedron outside the domain (Fig. 7(c)). Similarly, if all the tetrahedra around an edge on the boundary are of the same material, we assume it has a domain-dual face (Fig. 7(d)).

Both material-dual and domain-dual edges and faces are used to restrict simplification from breaking the homotopy of input data. Unless necessary, in the following discussions we do not explicitly specify the two kinds of dual connections (material- and domain-).

5.3.3 Edge collapse filtering algorithm

Given an edge $e = v_pv_q$, we check whether collapsing it will change subdomain topology by matching it with the following cases:

- **Case 1.** If both vertices are inner vertices, the edge e is able to collapse, as edge v_5v_6 in Fig. 6. The contraction position is determined by QEM.
- **Case 2.** If one vertex is inner vertex and the other one is boundary vertex, the edge e is able to collapse, e.g. edge v_6v_8 in Fig. 6. The contraction position is the position of the boundary vertex.
- **Case 3.** If both vertices are boundary ones:
  - **Case 3.a** Check edge order: if it is smaller than both vertex orders, the edge e is not able to collapse, e.g. edge v_4v_7 in Fig. 6.
  - **Case 3.b** Visit each tetrahedron $\sigma{\sim}e$ adjacent to e, and check whether any dual connections (if exist) through the two remaining faces of σ will be cut by contraction; if cut off, the edge is not collapsible. If no dual connection exists for the remaining faces of every σ, the component is in isolation and the edge is not collapsible. Otherwise, the edge can collapse. See Fig. 8 for an illustration. The contraction position will follow the vertex with higher order; if they have equal orders, QEM determines the position.

We note that cases 1 and 2 are straightforward type checking that specifies how the inner vertices can be collapsed and snapped to boundary to preserve the boundary shape. Case 3.a intuitively allows for boundary vertex collapse only when the edge is also a boundary, which prevents snapping of two vertices on different boundary surfaces but allows for the simplification of feature curves met by multiple subdomains. Illustrations are given in Fig. 6.

The case 3.b is the most complex and uses the dual graph to check for feasibility. However, at its center is the preservation of the dual graph connection. While case 3.b gives a conceptual description, we further present the computational details that implement the rule in Alg. 1. Note that in the implementation we have checked that, if the subdomain is isolated around the edge and would disappear due to collapse, the edge would be skipped. In addition, to ensure that subdomains in contact with data cube boundaries do not lose their connection during simplification, we specially mark the boundary faces as always having domain-dual edges. This essentially guarantees that the connections on the data volume boundary are preserved.

Algorithm 1: Implementation of Case 3.b.

```
Input: edge $v_0v_1$
Output: if the edge $v_0v_1$ is collapsible
$isolated$ = True;
for each tet $v_0v_1v_iv_j \sim v_0v_1$ do
    // Material connection filtering
    if $v_0v_iv_j$ or $v_1v_iv_j$ has material-dual edges then
        $isolated$ = False;
        if $v_0v_1v_j$ has material-dual edge then
            if neither $v_1v_j$ nor $v_0v_j$ has material-dual faces then
                return False;
            end
        end
        if $v_0v_1v_i$ has material-dual edge then
            if neither $v_1v_i$ nor $v_0v_i$ has material-dual faces then
                return False;
            end
        end
    end
    // Volume domain connection filtering
    if $v_0v_1v_j$ has domain-dual edge then
        if neither $v_1v_j$ nor $v_0v_j$ has domain-dual faces then
            return False;
        end
    end
    if $v_0v_1v_i$ has domain-dual edge then
        if neither $v_1v_i$ nor $v_0v_i$ has domain-dual faces then
            return False;
        end
    end
end
return !$isolated$;
```

Theorem. *The edge collapse filtering algorithm preserves subdomain topology in terms of homotopy equivalence.*

Proof: Cases 1 and 2 do not change sub-domain topology because of the existence of inner vertex. Case 3.a only prevents edge collapse and does not change topology. Case 3.b preserves dual connections by construction. According to the nerve theorem [49], the dual connections are homotopy equivalent to the primal local tetrahedral mesh. We therefore have the homotopy equivalence between tetrahedral patches before and after the collapse, bridged by the dual connections.

Remark 1. While the type checking is similar to [38], by using the dual graph for topology measurement, we are capable of dealing with cases where [38] fails to proceed, e.g., the red edges in Fig. 9: as v_0, v_1 and v_2 are boundary vertices but the face $v_0v_1v_2$ is an inner face, the edge v_0v_1 is prohibited to collapse by [38], while in our algorithm, we find that the edge is not critical in sub-domain topology preservation, because each adjacent tetrahedron has dual connections that the collapse won't destroy, so we allow it. Such cases are indeed important for generating more regular surfaces by simplification from the initial cubical meshes, as illustrated in Fig. 9b, 9c.

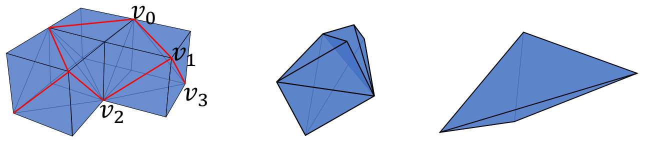


Fig. 9. An example which violates the topology filter of [38] and can be handled by our approach without restriction. The red edges in (a) are not collapsible by MAD [38], producing the irregular result in (b), while our method handles them properly, yielding the more compact result in (c). This shows that our method is more flexible in simplification.

Remark 2. The edge collapse problem is order-dependent: even if a sequence of edges can be collapsed, a shuffling of the sequence may not. Intuitively, for example, collapse may start from the leaf nodes of a tree-like structure and eventually remove all branches, but cannot start from an inner junction. The shrinking of branches of a tree structure is possible due to the homotopy-equivalence preserving property of our algorithm; however, significant shrinking is unlikely due to the large geometric errors caused.

5.3.4 *Boundary features*

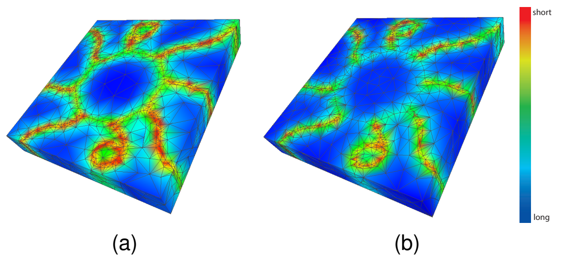
Besides preserving volume subdomain topology, our algorithm also preserves the critical lower dimensional boundary features during simplification. Following [17], [38], [39], we define the boundary feature elements as lower dimensional elements adjacent to multiple labels representing shape boundaries between subdomains: (i) 2-features are the faces with orders larger than 1; (ii) 1-features are the edges with orders larger than 2; (iii) 0-features are the vertices with orders larger than 3. Features of these three types form a cell complex, where each cell consists of consecutive feature elements of the same order [17], [38], [39]. As can be seen from Case 3.a, our algorithm preserves the feature complex by collapsing edges that strictly reside in a cell (1-feature or 2-feature) of the complex; on the other hand, Cases 1 and 2 involve edges residing off the feature complex and inside the volume regions, which has no impact on feature complex either.

5.4 *Improving Tetrahedra Regularity*

Our strategies on multiple material simplification can preserve features and topology. However, when the number of vertices is small enough, there may be very few vertices inside slender features, which will lead to the poor quality of the tetrahedra. To achieve high simplification rates as well as high mesh quality, adaptive grading of mesh elements according to local shape complexity is needed.

A typical approach for adaptive grading is to define a density field by local feature size [21] and distribute the vertices accordingly [7]. However, the computation of local feature size involves loading the entire dataset into memory and computing the medial axes globally, which is prohibitive for large-scale data.

We propose an online and local measurement, *normalized edge length variance*, to simulate the adaptive gradation induced by the local feature size. To be specific, the normalized edge length variance of a tetrahedron σ is defined



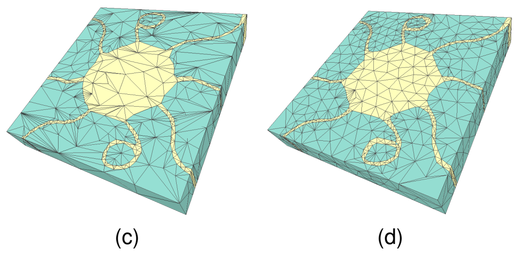


Fig. 10. Comparison of gradation by density field (left) or normalized edge length variance (right). (a) is the result of density field with 4.8K vertices. (b) is the result of normalized edge length variance with 4.8K vertices. The color coding denotes edge length: red for short edges, and blue for long edges. With the same number of vertices, (a) and (b) show similar gradation and distribution of edge lengths. However, when more aggressive simplification targets are used (1.5K), (d) the result of normalized edge length variance has 2.3K vertices and avoids oversimplification, while (c) the result of density field has 1.5K vertices and worse quality.

as the ratio of edge length variance to the squared average edge length:

$$EV(\sigma) = \frac{1}{6\bar{e}^2}\sum_{i=1}^{6}(|e_i| - \bar{e})^2, \tag{3}$$

where $\bar{e}$ is the average edge length of all edges e_i in σ, and $|e|$ measures the edge length of e.

During the simplification process, we use a threshold τ_e for normalized edge length variance to control the gradation of edge length: for an edge e, it is a valid edge for contraction only if the EV of all tetrahedra adjacent to e are smaller than τ_e after contraction. The threshold τ_e avoids dramatic changes of edge lengths and achieves smooth gradation in the simplified mesh.

Fig. 10 shows a comparison of the results computed by local feature size based density control and by the normalized edge variance control. For density field control, we compute the global density field as the inverse of local feature size, following [7]. The density value is then used to modulate the priority for simplification:

$$\mathbf{Q}' = d_v\mathbf{Q}, \tag{4}$$

where d_v is the density value of the vertex v, $\mathbf{Q}$ is the quadratic error matrix (Sec. 5.2), and $\mathbf{Q}'$ is the modulated metric. As expected, a larger density value causes a larger quadratic error and in turn a lower priority in simplification. As shown in Fig. 10, given the same number of target vertices, we can see that the results of density field control and normalized edge length variance control are similar.

TABLE 1
Basic information of datasets. Many of the datasets we used are very large-scale and complicated.

Dataset	Source	Volume Sizes	#Vertex	# Materials
Spine	MRI	$545 \times 171 \times 171$	15.23M	19
Teeth	MRI	$305 \times 237 \times 227$	15.84M	30
Chest	CT	$512 \times 512 \times 438$	109.96M	24
Kidney	CT	$512 \times 512 \times 611$	160.17M	3
Mito	EM	$2048 \times 2048 \times 1000$	4202.6M	24547

5.5 Mesh Optimization by Constrained ODT

After simplification within each window, we strive to further improve the mesh quality by the constrained ODT (CODT) iterations that optimize both vertex distribution and mesh connectivity simultaneously. Though CODT optimization is a time-consuming operation for large-scale meshes, applying it inside a marching window is fast, as the small number of vertices reduces the cost significantly. We compute constrained Delaunay triangulation of the vertices by TetGen [51], with the surface mesh between materials constrained. This step is to find the optimal connection of mesh vertices. Note that as discussed in Sec. 4, we also fix tetrahedra that are on or cross the window boundary to ensure consistency across windows.

Table 3 illustrates the statistics of the tetrahedral quality. It measures the mesh quality with two metrics, the *radius ratio* and the percentage of *slivers*. The *radius ratio*, defined as the ratio of radii of the inscribed sphere to the circumscribing sphere of a tetrahedron, is directly related to the shape of the tetrahedron [3]. A radius ratio of 1 means a regular tetrahedron, and a ratio of 0 means a tetrahedron that has collapsed down to a plane. Here we use the minimum radius ratio to measure the most poorly shaped tetrahedron and the average radius ratio to measure the overall mesh quality. We also use the percentage of *slivers* to measure the mesh quality. Slivers are flat tetrahedra with vertices that are nearly co-planar [52]; we set the radius ratio threshold of 0.1 for a tetrahedron to be considered silver. From Table 3 we can see that the constrained ODT optimization largely improves mesh quality.

6 Results and Discussion

We apply the *Marching Windows* algorithm on several scanned real-world datasets provided by [46], [53], [54], [55], [56]. Their basic information is listed in Table 1. Through experiments, we show that *Marching Windows* is capable of dealing with datasets of very large size within tractable time and limited memory, while preserving topological and geometrical features and achieving high mesh quality. We also compare with standard 3D meshing methods like Delaunay refinement [3], [57] and the multi-material adaptive volume remesher (MAD) [38] where they are applicable, to explore the strengths and limitations of different approaches. Specifically, we set the target edge length of MAD the same as the average edge length of our result mesh for each dataset for a fair comparison in each experiment. We also set the *number of iterations* as 10 and *remeshing boundaries* as true.

TABLE 2
Time and memory cost of different datasets, running on an Intel(R) Xeon(R) CPU with 128GB of memory. The iteration number of CODT optimization is 10, the window size is $40\times40\times40$, and the target edge variance is 10.

	Result #V			Memory Usage			Time Usage		
Dataset	Delaunay Refinement	MAD	Ours	Delaunay Refinement	MAD	Ours	Delaunay Refinement	MAD	Ours
Spine	7,813(0.05%)	61,884(0.39%)	69,546 (0.44%)	90 M	44.99 G	0.23 G	21s	1.87h	0.88h
Teeth	10,559(0.06%)	76,589(0.47%)	68,892 (0.42%)	99 M	46.39 G	0.42 G	27s	1.93h	0.93h
Chest	165,663(0.01%)	NA*	485,664 (0.42%)	952M	NA*	1.18 G	158s	NA*	6.7h
Kidney	3,120(0.002%)	NA*	646,151 (0.40%)	679M	NA*	1.20 G	33s	NA*	8.95h
Mito	NA*	NA*	2,291,769 (0.05%)	NA*	NA*	122.13 G	NA*	NA*	306.37h

* We set a memory limit 128G for all the datasets. MAD fails to process Chest, Kidney, and Mito datasets within the memory limits. *Marching Windows* processes the Mito dataset within around 1 GB; 122.13G is the space used only for merging the processed blocks into a single standard mesh data structure, which echoes the extreme complexity of the dataset.

TABLE 3
Tetrahedron quality statistics of datasets with target edge variance of 10. Results show that the CODT applied after simplification generally improve mesh quality.

	W/O CODT		W. CODT	
Dataset	Min/Avg RR	Sliver (%)	Min/Avg RR	Sliver (%)
Spine	3.24e-6/0.49	11.25(%)	2.04e-6/0.58	4.75(%)
Teeth	1.56e-6/0.49	11.28(%)	2.30e-6/0.58	4.83(%)
Chest	1.51e-6/0.48	11.55(%)	1.54e-6/0.57	5.18(%)
Kidney	1.46e-6/0.49	11.32(%)	2.28e-6/0.58	4.90(%)
Mito	NA	NA	2.32e-8/0.57	3.97(%)

TABLE 4
Mesh quality of output meshes with different target edge variances(TEV), tested on the teeth dataset. Larger target variance leads to higher simplification ratios and lower mesh quality.

TEV	# Verts	Min/ Avg Radius Ratio	Sliver (%)
3	160,204(0.98%)	1.62e-5/0.69	1.12(%)
5	101,441(0.62%)	6.91e-6/0.65	2.12(%)
10	68,892(0.42%)	2.30e-6/0.58	4.83(%)

6.1 Scalability

Marching Windows is capable of dealing with datasets of extremely large size, as it only uses bounded resources as determined by the window size. In comparison, previous methods based on mesh simplification [37], [58] or remeshing [38] cannot deal with extra-large datasets, because they have to load the whole dataset into memory. For example, we tested the state-of-the-art MAD [38] with its CGAL implementation [59] on a server computer with 128GB memory: it can only handle small datasets like Spine dataset and Teeth dataset, while the other datasets exceed this scale and cause the out-of-memory error (Table 2). For the Mito dataset, due to its extremely large size, we increase the memory limit to 300GB but still find that neither MAD nor Delaunay refinement can load the data to process it properly.

6.2 Efficiency

In terms of running time, we show that the efficiency of Marching Windows can be affected by window size in two ways. On one hand, with a smaller window size, more time would be spent on disk IO to swap more blocks; on the other hand, less time would be spent on simplifying the fewer elements inside a window. Trade-off details by varying window sizes can be found in Table 5. Our choice of window size 40 strikes a balance of IO and window processing such that the total time is the least on our experimental device.

When comparing with other method, *Marching Windows* is more efficient than MAD, but is slower than Delaunay refinement (Table 2). However, when taking result quality into consideration, we find that while both MAD and *Marching Windows* preserve the input topology and geometry features, the Delaunay refinement approach is prone to losing these important features; see Table 6 and Figs. 17, 18 and 19 for both quantitative and qualitative comparisons. More discussions are made in Sec. 6.5.

Our efficiency over MAD can be attributed to two factors. First, our simplification-based scheme is fast and involves only local operations and evaluations of mixed QEM metric, in contrast to the more costly implicit surface fitting of MAD for material boundaries. Second, the simplified vertices are well distributed so that only a few iterations of constrained ODT are sufficient to produce high-quality mesh inside each window. In comparison, MAD uses an ensemble of mesh improvement techniques, e.g. edge splitting/collapse, flipping, and vertex smoothing, iteratively to enhance mesh quality. However, we note that our approach complements the advanced remeshing of MAD, by deriving sparse tetrahedral meshes from excessively challenging big data that can be further improved by MAD.

6.3 Mesh Quality

Marching Windows improves the quality of the generated meshes by constrained optimal Delaunay triangulation. Table 3 shows the tetrahedron quality statistics of each dataset with CODT and without CODT, respectively. Results show that CODT can significantly improve the mesh quality of the generated meshes.

As we use target edge variance to control the mesh quality and avoid oversimplification, different target variances correspond to different simplification ratios and mesh quality, with a smaller target variance leading to denser simplified meshes and higher quality. For example, Table 4 shows the relationship between target edge variance and mesh quality and Fig. 15 visualizes them, all tested on the teeth dataset. We show in Fig. 13 that window size would also affect the coarseness of the result mesh. However, once

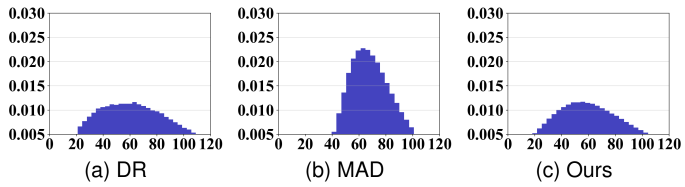


Fig. 11. Mesh quality of different methods, tested on the teeth dataset. We show the dihedral angle distribution to illustrate the mesh quality of the generated meshes. Our result is not as good as MAD but is comparable to Delaunay refinement.

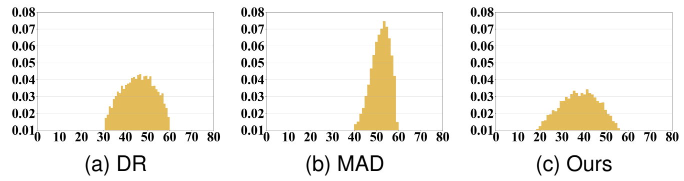


Fig. 12. Boundary mesh quality of different methods, tested on the teeth dataset. We show the minimum angle distribution of the boundary faces to illustrate the quality of generated meshes. Though not as good as the result mesh of MAD, our result has a comparable quality to that of Delaunay Refinement.

TABLE 5
Time usage for different window sizes, tested on the teeth dataset. With a smaller window size, more time would be spent on disk-IO and less time would be spent on simplification.

Window size	IO time	Process time	Total
100×100×100	72s	3730s	3792s
80×80×80	79s	3574s	3653s
60×60×60	103s	3437s	3540s
40×40×40	156s	3259s	3415s
20×20×20	2815s	3015s	5830s

the window size is large enough to accommodate the TEV threshold, the coarseness of the result mesh remains stable.

We also show the dihedral angle distribution of results generated by different methods in Fig. 11. We can see that although our result quality in terms of dihedral angle distribution is not as good as MAD which focuses primarily on mesh quality improvement, it is comparable to that of Delaunay Refinement. In addition, the surface mesh quality of region boundaries for different methods have similar patterns, as shown in Fig. 12. We note that our result mesh can be used as a good initialization and further improved by MAD if a very high mesh quality is needed.

6.4 Topology Preservation

Besides the proof of homotopy equivalence in Section 5.3, for validation, we also randomly picked a material from a complex dataset, Mito, as it comprises of long slender features and is fragile under simplification methods. Fig. 14 shows that *Marching Windows* exactly keeps the topology of the material subdomain. Fig. 16(d) shows another example of a challenging material domain preserved.

6.5 Feature Preservation

In this section, we compare in detail the result quality and fidelity in preserving input features both geometrically and

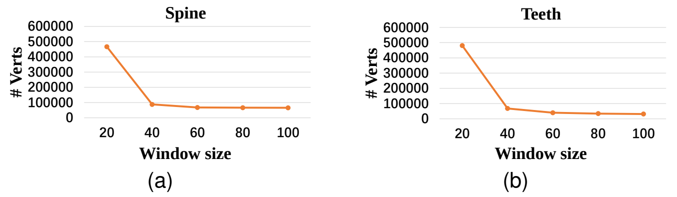


Fig. 13. Number of result vertices varies with window sizes. The plots show that once the window size is large enough with respect to the TEV threshold, the coarseness of the result mesh remains stable.

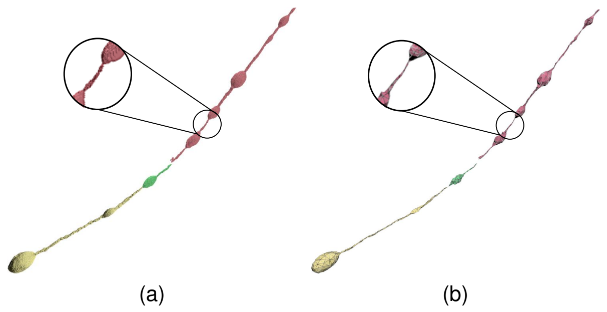


Fig. 14. Topology preservation in terms of homotopy equivalence. (a) Mesh of a random material before processing. (b) Mesh after processing. We use different colors for different connected components. Such challenging slender material demonstrates that *Marching Windows* keeps the topology exactly.

topologically for different methods. Quantitatively, Table 6 reports the largest Hausdorff distance among all the result material components from their corresponding materials of the input meshes, as well as the change of material components from input for Delaunay refinement.

As can be seen from Table 6, the most profound issue with Delaunay refinement is that it may fail in preserving the topology and detailed geometry features of input data, especially for the medical imaging data which have very complex components. As a result, numerous components can be lost or disconnected, and there can be a large deviation from the input geometry. Figs. 17 and 18 show how the input topology is changed by Delaunay refinement, and Fig. 19 highlights some components in the input that are entirely missed in the result.

When compared with MAD [38] in Hausdorff distance, our result is also slightly better, mostly because we use a mixed QEM geometric objective that is locally more robust than the implicit smooth surface fitting of MAD, which can be difficult on the slender features.

7 Conclusion

We have presented an efficient *Marching Windows* algorithm for generating high-quality compact mesh representations of large-scale volumetric datasets with multiple labeled materials. *Marching Windows* is capable of dealing with datasets of very large size with limited computational resources by localizing to one window at each step. It also preserves strictly the subdomain topologies and with high fidelity the geometric shapes, which is vital as the subtle features frequently define important characteristics of the data. The

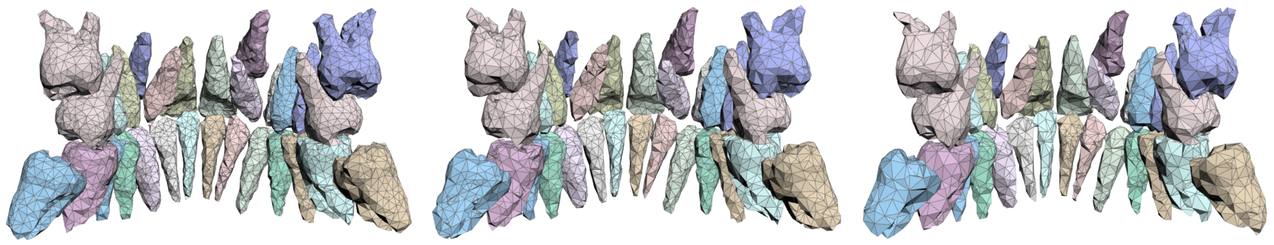


Fig. 15. Simplification results of the teeth dataset with different target edge variances. (a) With target edge variance of 3, the result keeps around 0.98% of original vertices. (b) With target edge variance of 5, the result keeps around 0.62% of original vertices. (c) With target edge variance of 10, the result keeps around 0.42% of original vertices. Smaller target edge variance leads to denser output mesh with higher mesh quality. Geometric features are also better preserved with smaller target edge variance.

TABLE 6
Hausdorff distance comparison among the result meshes of different methods. The numbers in parentheses for Delaunay Refinement represent the changes in the number of model components. Since the other two methods are designed to preserve topology, we omit the zero changes for them.

	Delaunay Refinement	MAD	Ours
Spine	13.29 (+1)	12.23	6.51
Teeth	12.62 (-10)	11.58	5.11
Chest	74.02 (-55)	NA	18.83
Kidney	26.68 (-19)	NA	5.55
Mito	NA	NA	36.15

key components enabling this quality is a mesh simplification approach with topological filtering and mixed error metric for edge collapse, local edge length variance control, as well as iterations of CODT optimization to quickly boost the mesh quality. We have demonstrated the practical effectiveness of our method on several challenging real-world datasets, for which we obtain much more compact but faithful geometric representations suitable for further applications.

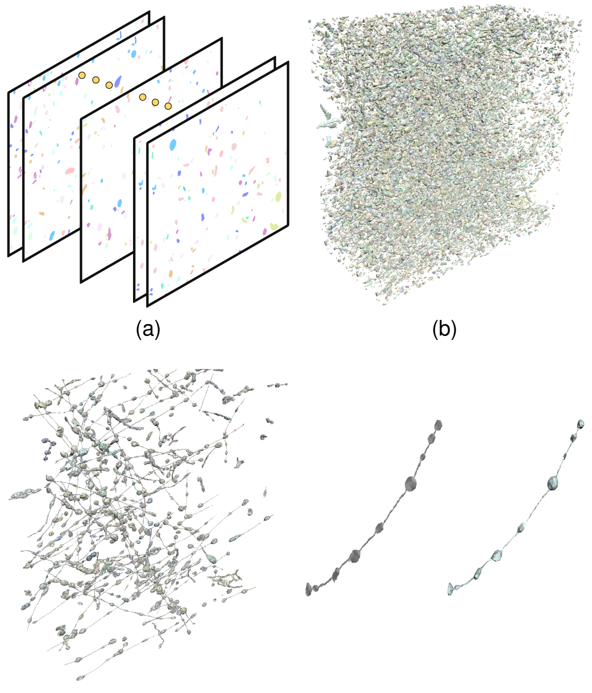


Fig. 16. Meshing results of the Mito dataset with 2M (0.13%) vertices. (a) Tiles of input images of the mitochondria dataset. (b) Generated mesh by *Marching Windows*. (c) Samples of relatively large mitochondria in the dataset. (d) One instance mitochondria of the dataset. Left is the instance model directly obtained from voxelization and right is the instance model after processing, showing that *Marching Windows* keeps such challenging slender features.

ACKNOWLEDGMENTS

The research of Z. Chen was supported by the National Natural Science Foundation of China (No. 61972327) and the Natural Science Foundation of Fujian Province (No. 2022J01001). The research of H. Pfister was partially supported by NSF grant NCS-FO-2124179 and NIH grant 5U54CA225088-03.

REFERENCES

[1] J. Bronson, J. A. Levine, and R. Whitaker, “Lattice cleaving: A multimaterial tetrahedral meshing algorithm with guarantees,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 20, no. 2, pp. 223–237, 2014.
[2] J. Ruppert, “A delaunay refinement algorithm for quality 2-dimensional mesh generation,” *Journal of algorithms*, vol. 18, no. 3, pp. 548–585, 1995.
[3] J. R. Shewchuk, “Tetrahedral mesh generation by delaunay refinement,” in *Proceedings of the fourteenth annual symposium on Computational geometry*. ACM, 1998, pp. 86–95.
[4] J. R. Shewchuk and H. Si, “Higher-quality tetrahedral mesh generation for domains with small angles by constrained delaunay refinement,” in *Proceedings of Symposium on Computational Geometry*. ACM, 2014, pp. 290–299.
[5] M. Meyer, R. Whitaker, R. M. Kirby, C. Ledergerber, and H. Pfister, “Particle-based sampling and meshing of surfaces in multimaterial volumes,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 6, pp. 1539–1546, 2008.
[6] J.-C. X. Chen Long, “Optimal delaunay triangulations,” *Computational Mathematics*, vol. 22, no. 2, pp. 299–308, 2004.
[7] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun, “Variational tetrahedral meshing,” *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 617–625, 2005.
[8] Z. Chen, W. Wang, B. Lévy, L. Liu, and F. Sun, “Revisiting optimal delaunay triangulation for 3d graded mesh generation,” *SIAM J. Scientific Computing*, vol. 36, no. 3, pp. A930–A954, 2014.
[9] X.-M. Fu, Y. Liu, J. Snyder, and B. Guo, “Anisotropic simplicial meshing using local convex functions,” *ACM TRANSACTIONS ON GRAPHICS*, vol. 33, no. 6, 2014.
[10] M. Murphy, D. M. Mount, and C. W. Gable, “A point-placement strategy for conforming delaunay tetrahedralization,” *International*

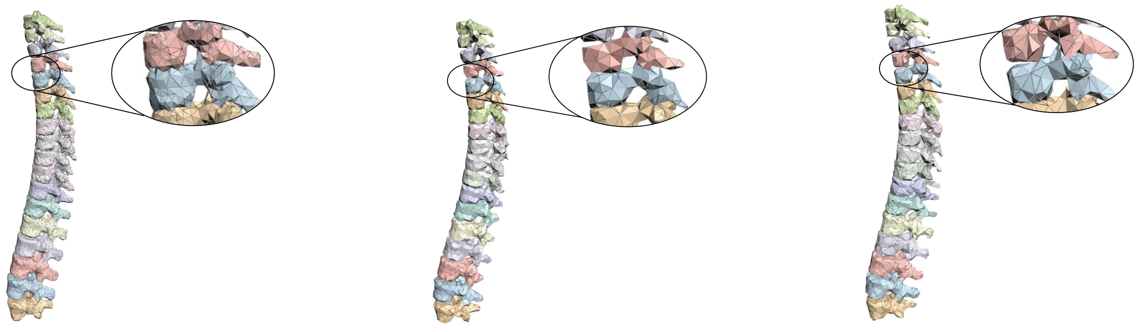


Fig. 17. Generated meshes of the spine dataset by different methods. The results above show that the mesh generated by Delaunay refinement does not keep topology, while the mesh generated by MAD and *Marching Windows* are accurate and compact.

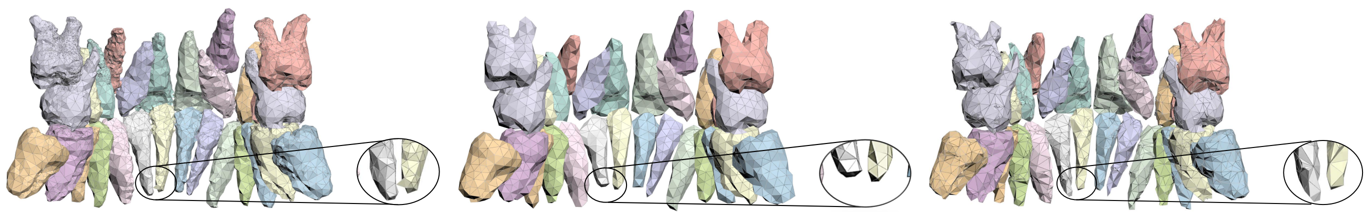


Fig. 18. Generated meshes of the teeth dataset by different methods. Same as Fig. 17, the mesh generated by MAD and *Marching Windows* are accurate and compact. The mesh processed by Delaunay refinement fuses different materials.

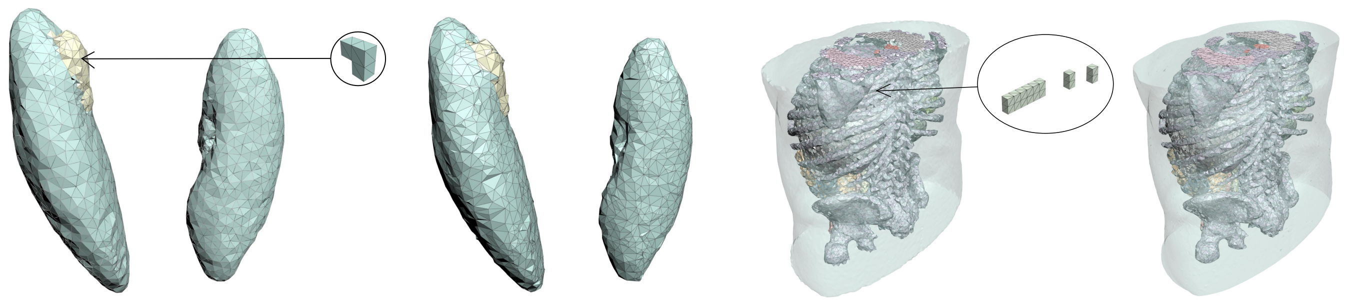


Fig. 19. Generated meshes of the kidney and chest datasets. (a) and (c) show that Delaunay refinement fails to preserve input topology. In the zoom-in circles, we highlight the lost parts by Delaunay Refinement; these substructures exist in the original data but disappear in the result meshes of Delaunay Refinement. In contrast to the loss of details by Delaunay refinement, *Marching Windows* handles them well with guaranteed multi-material topology preservation, as shown in (b) and (d). We do not show the results of MAD because it exceeds the memory and time limits when processing these two datasets.

Journal of Computational Geometry & Applications, vol. 11, no. 06, pp. 669–682, 2001.
[11] D. Cohen-Steiner, E. C. De Verdière, and M. Yvinec, "Conforming delaunay triangulations in 3d," in *Proceedings of Symposium on Computational geometry*. ACM, 2002, pp. 199–208.
[12] J. R. Shewchuk, "Constrained delaunay tetrahedralizations and provably good boundary recovery," in *IMR*, 2002, pp. 193–204.
[13] J.-P. Pons, F. Ségonne, J.-D. Boissonnat, L. Rineau, M. Yvinec, and R. Keriven, "High-quality consistent meshing of multi-label datasets," in *Information Processing in Medical Imaging*. Springer, 2007, pp. 198–210.
[14] D. Boltcheva, M. Yvinec, and J.-D. Boissonnat, "Mesh generation from 3d multi-material images," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2009*. Springer, 2009, pp. 283–290.
[15] H. Si, "Constrained delaunay tetrahedral mesh generation and refinement," *Finite elements in Analysis and Design*, vol. 46, no. 1, pp. 33–46, 2010.
[16] T. K. Dey and J. A. Levine, "Delaunay meshing of piecewise smooth complexes without expensive predicates," *Algorithms*, vol. 2, no. 4, pp. 1327–1349, 2009.
[17] S.-W. Cheng, T. K. Dey, and E. A. Ramos, "Delaunay refinement for piecewise smooth complexes," *Discrete & Computational Geometry*, vol. 43, no. 1, pp. 121–166, 2010.
[18] P. J. Frey and P.-L. George, *Mesh Generation: Application to Finite Elements*, 2nd ed. ISTE, May 2007.
[19] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi tessellations: applications and algorithms," *SIAM Review*, vol. 41, pp. 637–676, 1999.
[20] B. Lévy and Y. Liu, "Lp centroidal voronoi tessellation and its applications," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 119:1–119:11, Jul. 2010.
[21] N. Amenta and M. Bern, "Surface reconstruction by voronoi filtering," in *Proceedings of Symposium on Computational Geometry*. Association for Computing Machinery, 1998, p. 39–48.
[22] J. Rossignac and P. Borrel, *Multi-resolution 3D approximations for rendering complex scenes*. Springer, 1993.
[23] M. Soucy and D. Laurendeau, "Multiresolution surface modeling

based on hierarchical triangulation," *Computer vision and image understanding*, vol. 63, no. 1, pp. 1–14, 1996.

[24] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1997, pp. 209–216.

[25] P. S. Heckbert and M. Garland, "Optimal triangulation and quadric-based surface simplification," *Computational Geometry*, vol. 14, no. 1, pp. 49–65, 1999.

[26] K. J. Renze and J. H. Oliver, "Generalized unstructured decimation [computer graphics]," *Computer Graphics and Applications, IEEE*, vol. 16, no. 6, pp. 24–32, 1996.

[27] I. J. Trotts, B. Hamann, and K. I. Joy, "Simplification of tetrahedral meshes with error bounds," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 5, no. 3, pp. 224–237, 1999.

[28] P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno, "Simplification of tetrahedral meshes with accurate error evaluation," in *Visualization 2000. Proceedings*. IEEE, 2000, pp. 85–92.

[29] P. Chopra and J. Meyer, "Tetfusion: an algorithm for rapid tetrahedral mesh simplification," in *Visualization, 2002. VIS 2002. IEEE*. IEEE, 2002, pp. 133–140.

[30] S. Noguchi, M. Onosato, S. Kanai *et al.*, "Flexible control of multimaterial tetrahedral mesh properties by using multiresolution techniques," *Magnetics, IEEE Transactions on*, vol. 45, no. 3, pp. 1352–1355, 2009.

[31] Y. Wu, Y. He, and H. Cai, "Qem-based mesh simplification with global geometry features preserved," in *Proceedings of Computer graphics and interactive techniques in Australasia and South East Asia*. ACM, 2004, pp. 50–57.

[32] W. Ma, X. Ma, S.-K. Tso, and Z. Pan, "A direct approach for subdivision surface fitting from a dense triangle mesh," *Computer-Aided Design*, vol. 36, no. 6, pp. 525–536, 2004.

[33] M. Garland and Y. Zhou, "Quadric-based simplification in any dimension," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 2, pp. 209–239, 2005.

[34] O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee, "Skeleton extraction by mesh contraction," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 44:1–44:10, aug 2008.

[35] H. K. Choi, H. S. Kim, and K. H. Lee, "An improved mesh simplification method using additional attributes with optimal positioning," *The International Journal of Advanced Manufacturing Technology*, vol. 50, no. 1-4, pp. 235–252, 2010.

[36] G. Li, W. Wang, G. Ding, Y. Zou, and K. Wang, "The edge collapse algorithm based on the batched iteration in mesh simplification," in *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*. IEEE, 2012, pp. 356–360.

[37] R. Moore, G. Rohrer, and S. Saigal, "Reconstruction and simplification of high-quality multiple-region models from planar sections," *Engineering with Computers*, vol. 25, no. 3, pp. 221–235, 2009.

[38] N. Faraj, J.-M. Thiery, and T. Boubekeur, "Multi-material adaptive volume remesher," *Computers & Graphics*, vol. 58, pp. 150–160, 2016.

[39] F. Vivodtzev, G.-P. Bonneau, S. Hahmann, and H. Hagen, "Substructure topology preserving simplification of tetrahedral meshes," in *Topological methods in data analysis and visualization*. Springer, 2011, pp. 55–66.

[40] T. K. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev, "Topology preserving edge contraction," *Publ. Inst. Math. (Beograd) (N.S*, vol. 66, 1998.

[41] D. M. Thomas, V. Natarajan, and G.-P. Bonneau, "Link conditions for simplifying meshes with embedded structures," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 7, pp. 1007–1019, 2010.

[42] R. Said, N. Weatherill, K. Morgan, and N. Verhoeven, "Distributed parallel delaunay mesh generation," *Computer methods in applied mechanics and engineering*, vol. 177, no. 1-2, pp. 109–125, 1999.

[43] Y. Ito, A. M. Shih, A. K. Erukala, B. K. Soni, A. Chernikov, N. P. Chrisochoides, and K. Nakahashi, "Parallel unstructured mesh generation by an advancing front method," *Mathematics and Computers in Simulation*, vol. 75, no. 5-6, pp. 200–209, 2007.

[44] A. Loseille, V. Menier, and F. Alauzet, "Parallel generation of large-size adapted meshes," *Procedia Engineering*, vol. 124, pp. 57–69, 2015.

[45] D. Cabiddu and M. Attene, "Large mesh simplification for distributed environments," *Computers & Graphics*, vol. 51, pp. 81–89, 2015.

[46] D. Wei, Z. Lin, D. Franco-Barranco, N. Wendt, X. Liu, W. Yin, X. Huang, A. Gupta, W.-D. Jang, X. Wang *et al.*, "Mitoem dataset: Large-scale 3d mitochondria instance segmentation from em images," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2020, pp. 66–76.

[47] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, Aug. 1987.

[48] A. Doi and A. Koide, "An efficient method of triangulating equivalued surfaces by using tetrahedral cells," *IEICE TRANSACTIONS on Information and Systems*, vol. 74, no. 1, pp. 214–224, 1991.

[49] H. Edelsbrunner and J. Harer, *Computational topology: an introduction*. American Mathematical Soc., 2010.

[50] M. Desbrun, E. Kanso, and Y. Tong, "Discrete differential forms for computational modeling," in *Discrete differential geometry*. Springer, 2008, pp. 287–324.

[51] H. Si, "Tetgen, a delaunay-based quality tetrahedral mesh generator," *ACM Trans. Math. Softw.*, vol. 41, no. 2, pp. 11:1–11:36, Feb. 2015.

[52] D. Eppstein, "Global optimization of mesh quality," *Tutorial at the 10th International Meshing Roundtable*, vol. 10, p. 13, 2001.

[53] Z. Cui, C. Li, and W. Wang, "Toothnet: Automatic tooth instance segmentation and identification from cone beam ct images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6368–6377.

[54] N. Heller, F. Isensee, K. H. Maier-Hein, X. Hou, C. Xie, F. Li, Y. Nan, G. Mu, Z. Lin, M. Han *et al.*, "The state of the art in kidney and kidney tumor segmentation in contrast-enhanced ct imaging: Results of the kits19 challenge," *Medical Image Analysis*, p. 101821, 2020.

[55] J. Yao, J. E. Burns, D. Forsberg, A. Seitel, A. Rasoulian, P. Abolmaesumi, K. Hammernik, M. Urschler, B. Ibragimov, R. Korez *et al.*, "A multi-center milestone study of clinical vertebral ct segmentation," *Computerized medical imaging and graphics*, vol. 49, pp. 16–28, 2016.

[56] "IRCAD research data," https://www.ircad.fr/research/3d-ircadb-02/, accessed: 2020-06-25.

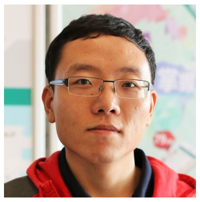
[57] P. Alliez, C. Jamin, L. Rineau, S. Tayeb, J. Tournois, and M. Yvinec, "3D mesh generation," in *CGAL User and Reference Manual*, 4th ed. CGAL Editorial Board, 2015.

[58] B. Cutler, J. Dorsey, and L. McMillan, "Simplification and improvement of tetrahedral models for simulation," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM, 2004, pp. 93–102.

[59] The CGAL Project, *CGAL User and Reference Manual*, 5.1.0 ed. CGAL Editorial Board, 2020. [Online]. Available: https://doc.cgal.org/5.1/Manual/packages.html

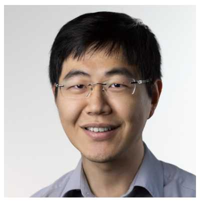
Wenhua Zhang received the BEng degree in computer science from Shandong University. She is currently working toward the PhD degree in computer science at The University of Hong Kong. Her research interests include medical image processing, computer vision, and computational geometry.

Yating Yue received the BEng degree in software engineering from Shandong University, and the PhD degree in computer science from The University of Hong Kong. Her research interests include computer graphics, HCI, and Mixed Reality.



Hao Pan received the BEng degree in software engineering from Shandong University, and the PhD degree in computer science from The University of Hong Kong. He is currently a researcher with Microsoft Research Asia. His research interests include computer graphics and computational geometry.

Zhonggui Chen received the BSc and PhD degrees in applied mathematics from Zhejiang University, in 2004 and 2009, respectively. He is working as a professor in the School of Informatics, Xiamen University, China. His research interests include computer graphics, computational geometry, and digital image processing. For more information, please visit http://graphics.xmu.edu.cn/~zgchen/.



Chuan Wang received the BEng degree in Electronic Information Engineering from University of Science and Technology of China, and the PhD degree in computer science from The University of Hong Kong. His research interests include computer vision, computer graphics, and machine learning.

Hanspeter Pfister received the PhD degree in computer science from the Stony Brook University and an MS degree in electrical engineering from ETH Zurich, Switzerland. He is the Academic Dean of Computational Sciences and Engineering and An Wang Professor of Computer Science in the Harvard John A. Paulson School of Engineering and Applied Sciences. He is also an affiliate faculty member of the Center for Brain Science at Harvard. His research in visual computing lies at the intersection of visualization, computer graphics, and computer vision and spans a wide range of topics, including biomedical visualization, image and video analysis, machine learning, and data science. For more information, please visit https://vcg.seas.harvard.edu/..

Wenping Wang received the PhD degree in computer science from the University of Alberta, Canada. He is a professor at The University of Hong Kong and Texas A&M University. His research covers computer graphics, computer vision, robotics, virtual reality, visualization, medical image analysis, and geometric modeling. He is journal associate editor of Computer Aided Geometric Design (CAGD), Computers and Graphics (CAG), IEEE Transactions on Visualization and Computer Graphics (TVCG, 2008-2012), Computer Graphics Forum (CGF), IEEE Computer Graphics and Applications, and IEEE Transactions on Computers. He is program chair of several international conferences, including Pacific Graphics 2003, ACM Symposium on Physical and Solid Modeling (SPM 2006), International Conference on Shape Modeling (SMI 2009), and conference chair of Pacific Graphics 2012, SIAM Conference on Geometric and Physical Modeling 2013 (GD/SPM'13), SIGGRAPH Asia 2013, and Geometry Summit 2019. For more information, please visit https://i.cs.hku.hk/~wenping/.