Bohm, Sebastian; Runge, Erich

Fast algorithm for extracting domains and regions from three-dimensional triangular surface meshes

Original published in:	Computer aided design : CAD. - Amsterdam [u.a.] : Elsevier Science, 180 (2025), art. 103824, 8 pp.
Original published:	2024-11-19
ISSN:	1879-2685; 0010-4485
DOI:	10.1016/j.cad.2024.103824
[*Visited:*	2025-04-14]

Computer-Aided Design 180 (2025) 103824

Contents lists available at ScienceDirect

Computer-Aided Design

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

Research Paper

Fast algorithm for extracting domains and regions from three-dimensional triangular surface meshes

Sebastian Bohm *, Erich Runge

Technische Universität Ilmenau, Institute of Physics, Group 'Theoretical Physics I', Weimarer Straße 25, 98693 Ilmenau, Germany

ARTICLE INFO

Keywords:
Domain extraction
Mesh handling
Boundary element method
Surface triangulation

ABSTRACT

Correct mesh handling and in particular domain extraction is an important step of many simulation workflows. In the boundary element method, for example, only the surfaces of arbitrarily shaped domains have to be meshed. Here, an algorithm is presented that extracts all distinct spatial regions and domains from a given non-manifold three-dimensional surface triangulation without the need for a volume mesh. Only the coordinates of the mesh vertices and the connectivity matrix of the triangulation need to be given. The process is robust, easy to implement, and efficient. No geometric features such as edges or faces need to be detected or specified for the method to work. Thus, the method is suitable for mesh formats that do not contain information about the geometry partitioning. The algorithm is presented in detail, test cases are discussed, and an exemplary implementation is given.

1. Introduction

Surface triangulations are a powerful tool for approximately representing three-dimensional geometries. They are widely used in surface visualization and numerical simulation. Surface triangulations are particularly important for the Boundary Element Method because, unlike the Finite Element Method, only the surface of the simulation domains need to be meshed. However, new challenges can arise when handling surface triangulations instead of volume meshes. For example, the individual domains contained in the geometry cannot be extracted directly from the connectivity matrix representing the surface triangulation. Such domain partitioning, however, is important for many numerical applications and can also be of interest for modeling and design tasks. In numerical simulations it is often the case that different material properties have to be taken into account in different spatial areas of the geometry [1–3]. Likewise, several separate spatial domains occur if different equations have to be solved to model different physical processes on the same geometry. A typical example would be a fluid–structure interaction simulation, where fluid dynamics can be ignored in the solid domains [4–8]. Another typical example is the solution of multiphase flow problems [9,10]. In these problems, each phase is represented by a separate spatial area.

For the numerical treatment of all these problems, the in general non-manifold surface triangulation must be partitioned into the individual domains. However, not all mesh formats support this partitioning by default and the underlying CAD representation of the geometry is not always available. For example, the frequently used STL file format contains only the coordinates of the mesh vertices and the connectivity matrix of the triangulation. In such a case, the information about the underlying domains is not directly available and has to be extracted first. For 3D volume meshes, an accurate domain segmentation is only possible if additional information on domain assignments is available in addition to mesh connectivity and vertex positions. If this information is available, however, partitioning is an easy task because the individual mesh elements can be directly assigned to the domains and the surface of each domain is formed by all facets that are used by only one element of the respective domain. In the case of non-manifold surface meshes, the problem cannot be solved with such simplicity, given that multiple domains may share the same triangles. This paper presents a robust, efficient and easy to implement algorithm to extract all domains from a given surface triangulation. This method requires only that the coordinates of the mesh vertices and the connectivity matrix are given. This means that no faces or edge connectivities or any other geometric features need to be known. In addition, the algorithm automatically provides oriented connectivity matrices for each domain present in the geometry, meaning that the normal vectors on the surface of each domain are equally oriented. As part of the process of creating tetrahedral meshes from surface triangulations, it is necessary to subdivide the underlying geometry into the contained domains. The algorithm presented here can therefore be useful and employed as an initial step for the process of volume mesh creation using methods like the one described by Jacobson et al. [11]. Jiang

* Corresponding author.
E-mail address: sebastian.bohm@tu-ilmenau.de (S. Bohm).

https://doi.org/10.1016/j.cad.2024.103824
Received 17 May 2024; Received in revised form 1 November 2024; Accepted 7 November 2024
Available online 19 November 2024

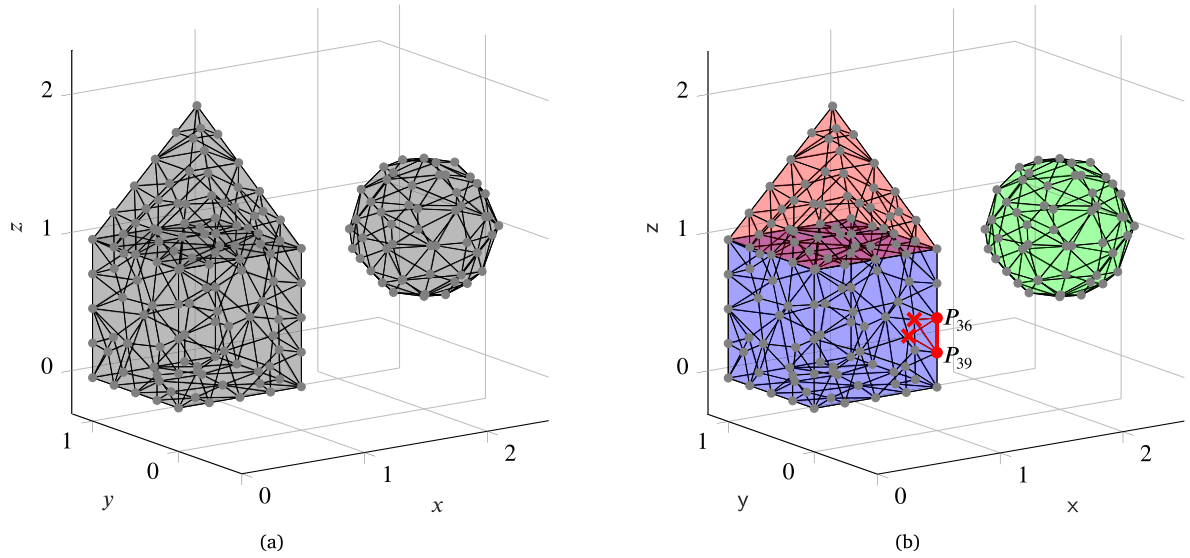

Fig. 1. (a) Example of a geometry consisting of two regions with three inner and one outer domain. The geometry is represented by the surface triangulation only and no information about the domain connectivities are given. (b) The individual inner domains were identified and are shown here in different colors. The green domain forms one region, the blue and the red domains together form the second region. The gray dots mark the mesh vertices. The thick red line marks an edge which is formed by two vertices P_i and P_j, e.g. $i = 36$, $j = 39$. The thin red lines mark the neighboring triangles of the specified edge and the red crosses mark the neighboring vertices, i.e. the neighbors of the edge. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and Bunke [12] introduced an algorithm which solves an equivalent problem for two-dimensional meshes represented as line segments. Their algorithm determines all the domains contained in such a two-dimensional line graph with optimal runtime $O(M \log(M))$, where M is the number of lines. It is a greatly improved version of the algorithm previously introduced by Fan et al. [13]. However, both algorithms are limited to the two-dimensional case, and we are not yet aware of any existing extension to three dimensions. Such a generalization to the three-dimensional case is presented subsequently. It allows the extraction of all domains contained in a surface triangulation with a time complexity of $O(M \log(M))$.

The paper is organized as follows. First, the geometric quantities required for the method are introduced and defined in Section 2. Then, the algorithms required to partition the geometry into domains are introduced in Section 3. To do this, the geometry is first partitioned into regions. This procedure is described in Section 3.1. In Section 3.2, the algorithm for identifying the individual domains is described in detail. Finally, a procedure for correcting nested domains and regions is presented in Section 3.3. Although the extraction of individual regions and the correction for nested geometry parts are important parts of the overall domain extraction procedure, the algorithm for extracting individual domains described in Section 3.2 is the main result of this paper. Afterwards, the method is validated for several test geometries in Section 4, including an analysis of the efficiency and runtime. Finally, the results are summarized and conclusions are drawn in Section 5.

2. Definitions

The surface triangulation representing the geometry is assumed to be given by an $M \times 3$ sized connectivity matrix K and an $N \times 3$ sized coordinate matrix P, where M is the total number of triangles and N is the total number of mesh vertices. The ith row of the connectivity matrix K contains the three indices of the mesh vertices that form the ith triangle. The ith row of the coordinate matrix P contains the x, y and z coordinates of the ith vertex of the triangulation. It is assumed that the triangulation is error-free, e.g., no triangles are duplicated or triangles have degenerated into lines. If this is not the case, the triangulation must first be corrected using appropriate methods [14,15].

In general, the surface triangulations can contain disconnected regions, and each region has potentially non-manifold edges and faces leading to multiple interior domains. An example of such a geometry is shown in Fig. 1(a). Each surface triangulation contains at least one region and each region contains at least one inner domain. A region is a connected component of the geometry that is given by a subset of triangles where none of the triangles share a common vertex with any of the remaining triangles. An inner domain is a three-dimensional spatial 'water-tight' oriented polyhedron with a closed surface and is given by a subset of triangles of the corresponding region. In addition to the inner domains, each geometry has a single outer domain, which is formed by all outer faces, i.e. those triangles that are only part of one inner domain. Each triangle consists of six oriented edges v. Each edge $v = [i, j]$ is formed by two vertices i, j and has at least two neighbors. A neighbor of an edge v is a vertex P that is opposite of the edge in any triangle that contains the edge v. This means, that this vertex P is not part of the edge v itself but is connected to the two vertices of the edge $[i, j]$ via two other edges v' and v''. The total number of edges in the triangulation is denoted as E. For the extraction of the individual domains, a representation of the triangulation via so-called 'wedges' is used. A wedge W is a four-element vector, where the elements represent indices of vertices of two adjacent oriented triangles of the mesh. The name wedge is chosen to emphasize the similarity to the method of Jiang and Bunke [12] for the region extraction in 2D line graphs and is not necessarily to be interpreted geometrically, since a wedge represents either a planar patch of the surface of the geometry or a tetrahedron. The determination and use of wedges is explained in more detail in Section 3. The geometry shown in Fig. 1(a) consists of two regions, three inner and one outer domain. The identical geometry with the individual extracted inner domains is shown in Fig. 1(b). The outer domain is formed by all the faces of the three domains except the face shared between the pyramidal and cubic, i.e., the red and blue, domains.

3. Algorithm

3.1. Determine the region connectivity matrices

In order to determine all domains of a given mesh, the connectivity matrices of the regions are determined first. There are several ways to do this. A naive approach would be to start with an arbitrary triangle and determine all neighboring triangles one by one. As soon as all

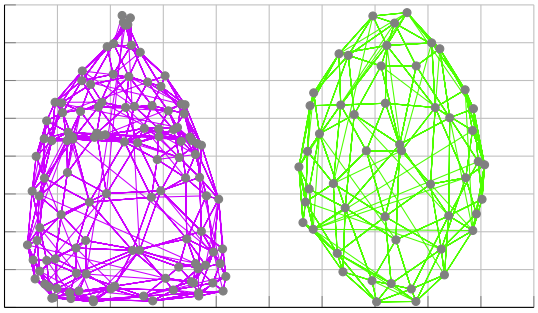

Fig. 2. Example of the graph associated with the geometry shown in Fig. 1. The separate regions are visible as unconnected components in the graph. The light green part of the graph corresponds to the green spherical shaped region and the magenta-colored part corresponds to the region composed of the cube- and pyramid-shaped red and blue domains of Fig. 1(b). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

neighbors have been run through once, the connectivity matrix of the region has been found. If there are still untested triangles, the procedure is repeated for these triangles until all triangles of the initial triangulation have been tested once. However, another much more efficient method is used here based on the graph representation of the mesh. Each triangulation can be represented as a graph, where the mesh vertices are the nodes in the graph and the edges in the graph are given by the edges in the mesh. There are very efficient algorithms to determine all connected components of a given graph. Here, we use the search algorithm introduced by Tarjan [16]. Its runtime depends linearly on the sum of the number of mesh vertices and mesh edges, i.e. $O(N+E)$. An example of the resulting graph for the geometry shown in Fig. 1(a) is presented in Fig. 2. For each vertex of the surface triangulation, the partitioning algorithm of Tarjan returns the corresponding index of the region. The connectivity matrix K_{R}^{i} of the ith region can subsequently be easily extracted and is formed by all triangles, i.e., rows of the initial connectivity matrix, that contain vertices with the corresponding index of the region. Thus, each connectivity matrix K_{R}^{i} is formed by a subset of the triangles of the initial connectivity matrix K. The index i runs from 1 to N_{R}, where N_{R} is the number of regions in the geometry.

3.2. Determine the domain connectivity matrices

Once the regions and their connectivity matrices are identified, we turn to the more challenging task of domain extraction. This step is done separately for each region. As a result of the algorithm presented below, a connectivity matrix $K_{\mathrm{RD}}^{i,j}$ is obtained for each inner and outer domain contained in the ith region. The index j runs from 1 to $N_{\mathrm{RD},i}$, where $N_{\mathrm{RD},i}$ is the number of domains in the ith region. In other words, the connectivity matrices $K_{\mathrm{RD}}^{i,j}$ of the contained domains are extracted from the connectivity matrix K_{R}^{i} of the respective region, meaning that each domain connectivity matrix $K_{\mathrm{RD}}^{i,j}$ is formed by a subset of the triangles of the region connectivity matrix K_{R}^{i}. If the geometry consists of nested or multiple regions, the connectivity matrices must subsequently be combined and corrected in a suitable manner. This procedure is described in Section 3.3. As a result, the final connectivity matrices K_{D}^{i} of the domains contained in the geometry are obtained, where i runs from 1 to N_{D} and N_{D} denotes the total number of domains in the geometry. The algorithm can be understood as follows: The triangles of the region connectivity matrix form the nodes and each wedge describes an edge, i.e. a connection, between two adjacent oriented triangles. The contained domains are extracted by performing a flood fill-like search on this graph without having to build this graph explicitly. In the graph, each triangle is contained twice but with reversed orientation, reflecting the fact that each triangle in the region connectivity matrix must be part of two different domains.

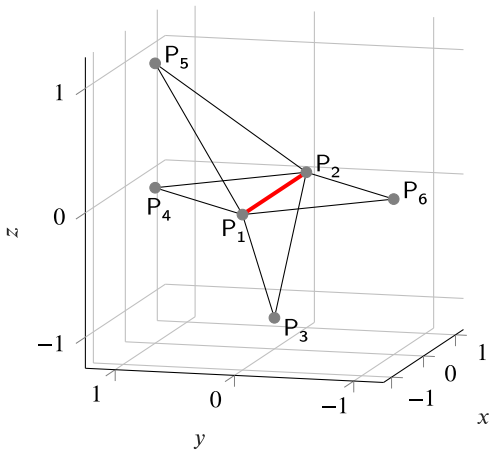

Fig. 3. The figure shows a part of a triangulation with four triangles that share a common edge [1, 2]. Several domains are therefore adjacent to this edge. The four neighboring vertices P_6, P_3, P_4, and P_5 of the edge [1, 2] happen to be already oriented clockwise.

3.2.1. Creation of the wedge list

As a first step, a list of wedges is created for each region by performing the following steps:

(i) A list with all non-oriented edges of the region connectivity K_{R}^{i} is created. The direction of the edges is arbitrary but no edge should appear twice. The list must therefore only contain the three non-oriented instead of all six oriented edges of each triangle in the respective region.

(ii) The adjacent triangles and the corresponding neighboring vertices are determined for each edge. If an edge is non-manifold, i.e. has more than two neighboring vertices, these vertices are first 'oriented', i.e. sorted into a list encircling the edge, e.g., clockwise. For this purpose, the rotation angles φ and ϑ corresponding to the spherical coordinates of the edge vector are determined, which are necessary to rotate the edge vector $\vec{x}(v_j) - \vec{x}(v_i)$ on the x-axis. The coordinates of all neighboring vertices of the respective edge vector are then transformed by the same rotation:

$$\begin{aligned}\vec{a}' &= R_y(\varphi)R_z(\vartheta)\vec{a} \\ &= \begin{pmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{pmatrix} \begin{pmatrix} \cos\vartheta & 0 & \sin\vartheta \\ -\sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{pmatrix} \vec{a}\end{aligned} \tag{1}$$

The neighboring vertices can then easily be oriented by their projection onto the y', z'-plane, e.g., by determining and sorting the angles relative to the y'-axis clockwise. The resulting sorted index arrays of the neighboring vertices are stored for each edge of the region. The resulting orientation of neighboring vertices is uniquely determined for an error-free surface triangulation.

(iii) Now, the list of wedges WR can be created. For this purpose, the wedges corresponding to the edge $v_p = [i, j]$ with k_p sorted, i.e. oriented, neighbors $[n_{p,1}, n_{p,2}, \dots, n_{p,k_p}]$ are defined via lists of four vertex indices each according to:

$$\begin{aligned} WR_{p,1} &= [n_{p,1}, i, j, n_{p,2}] \\ WR_{p,2} &= [n_{p,2}, i, j, n_{p,3}] \\ &\dots \\ WR_{p,k_p} &= [n_{p,k_p}, i, j, n_{p,1}] \end{aligned} \tag{2}$$

The first and last column of this $k_p \times 4$ matrix contain the neighboring vertices of each edge v_p. Finally, the resulting matrix is enlarged by appending the matrix once again with the columns arranged in reverse. I.e., in addition to, e.g., the

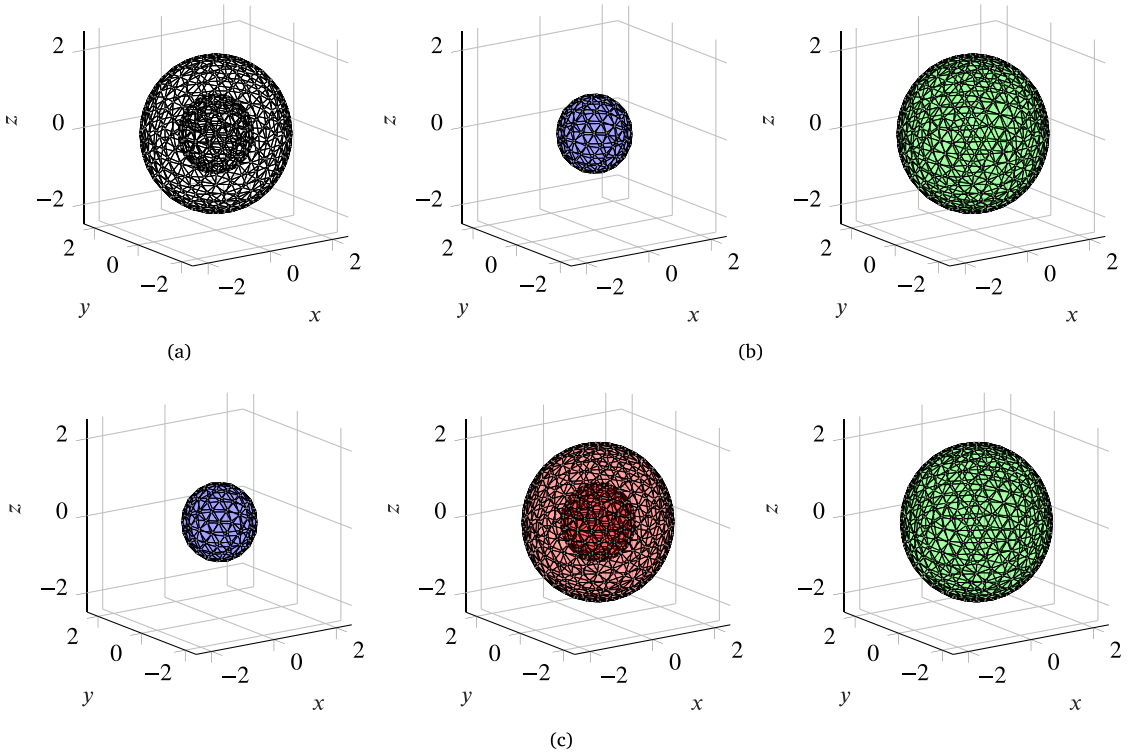

Fig. 4. (a) Input surface triangulation describing the geometry of two nested spheres: The geometry consists of two regions where each region consist of one inner and one outer domain. (b) Extracted domains using the algorithm introduced in Section 3: The two spheres that form the two regions are detected separately. For each sphere, there is one connectivity for the inner domain and one for the outer domain, which differ only in the orientation of the normal vectors. (c) The desired and correct identification of the individual domains: The geometry is partitioned into the inner sphere (blue), the spherical shell between the two spheres (red), and the outer domain (green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

wedge $[n_{p,1}, i, j, n_{p,2}]$, the entire wedge matrix contains the wedge $[n_{p,2}, j, i, n_{p,1}]$ as well. Therefore, in addition to the edge $[i, j]$, the edge $[j, i]$ is now also included in the wedge list, which means that all six oriented edges of each triangle in the region are considered. For the edge $[1, 2]$ with neighbors $[5, 4, 3, 6]$ as illustrated in Fig. 3, the following eight wedges result:

$$\begin{aligned} &[5,1,2,4] \qquad [4,2,1,5]\\ &[4,1,2,3] \qquad [3,2,1,4]\\ &[3,1,2,6] \qquad [6,2,1,3]\\ &[6,1,2,5] \qquad [5,2,1,6] \end{aligned} \tag{3}$$

This is repeated for all edges and the resulting lists are concatenated, until finally for a triangulation with M triangles, $6M$ such wedges have been formed. This is easiest to see for a closed triangulation of a single domain. In this case, each edge has two adjacent triangles and each triangle is formed by six oriented edges. Thus, the total number of edges is $3M$. Since each edge has two neighbors in this case, $6M$ wedges can be formed in total. The steps (i) and (iii) can be done in linear time while step (ii) is the most time consuming part with a time complexity of $O(M \log(M))$ since a wedge list with a total of $6M$ entries needs to be sorted.

3.2.2. Grouping of wedges

Once the wedge list has been created for the connectivity matrix of the ith region, the contained domains can be extracted by grouping the wedges. To do this grouping, the following steps are carried out:

(i) Set domain index $m = 1$

(ii) Set $k_{\text{last}} = 1$ and $n = 1$. Select any wedge, e.g. the first one, $W_n^m = [WR_{1,1}, WR_{1,2}, WR_{1,3}, WR_{1,4}]$ from the list of all remaining wedges WR. Use this wedge as starting wedge W_s^m, remove it from the wedge list WR and add it into the connectivity list $K_{\text{RD}}^{i,m}$ of the mth domain.

(iii) Find the next wedge of the form $[W_{n,3}^m, W_{n,2}^m, W_{n,4}^m, X]$ with any index X among the remaining wedges WR and add it as W_{n+1}^m to the connectivity list $K_{\text{RD}}^{i,m}$. Remove this wedge from WR and set $n = n + 1$. Repeat until the last element in the list, i.e. W_n^m matches the pattern $W_n^m = [X, W_{s,2}^m, W_{s,1}^m, W_{s,3}^m]$.

(iv) After finding a list of new wedges W_k^m with $k \in [k_{\text{last}}, n]$ in step (iii), it is now tested for each newly added and not yet checked wedge whether a remaining wedge with $W_{n+1}^m = [W_{k,2}^m, W_{k,3}^m, W_{k,1}^m, X]$ exists in WR.
If yes: Add this wedge to the connectivity list $K_{\text{RD}}^{i,m}$ and remove this wedge from WR. Set $W_s^m = W_{n+1}^m$ and $k_{\text{last}} = k$ and $n = n+1$. Go back to (iii).
If no: The complete connectivity matrix $K_{\text{RD}}^{i,m}$ of the mth domain was found. If all wedges have been checked, all connectivity matrices have been found and go to (v). If wedges still remain, set $m = m+1$ and go to (ii) for the extraction of the next domain connectivity.

(v) The result is a $3J_{i,m} \times 4$ matrix with $3J_{i,m}$ *continuous* wedges for each domain. This list contains the connectivity matrix of the mth domain in the first three columns. However, each of these matrices contains each connectivity of the domain three times, since all cyclic permutations occur. To obtain the final connectivity lists, the rows that occur more than once must be removed. To do this, the matrices are sorted row-wise and duplicate rows are removed, resulting in a $J_{i,m} \times 3$ sized connectivity matrix $K_{\text{RD}}^{i,m}$ for each domain, where $J_{i,m}$ is the number of triangles in the mth domain in the ith region.

Algorithm 1 gives a summary of this grouping procedure in pseudocode. Step (i) and (ii) of the grouping process can be performed in

Algorithm 1: Grouping of wedges

Description of the algorithm for grouping the wedges in order to extract the domains of the i-th region as pseudocode. X is an arbitrary vertex index. If there is more than one possible wedge in step 12, a wedge will be selected at random.

Input: list WR with all wedges of the i-th region
1 $m \leftarrow 1$;
2 $k_{\text{last}} \leftarrow 1$;
3 $n \leftarrow 1$;
4 $W_s^m, W_n^m, K_n^m \leftarrow [WR_{1,1}, WR_{1,2}, WR_{1,3}, WR_{1,4}]$;
5 Remove W_s^m from WR;
6 **do**
7 $\quad W_{n+1}^m, K_{n+1}^m \leftarrow$ find $[W_{n,3}^m, W_{n,2}^m, W_{n,4}^m, X]$ in WR;
8 $\quad$ Remove W_{n+1}^m from WR;
9 $\quad n \leftarrow n+1$;
10 **while** $W_n^m \neq [X, W_{s,2}^m, W_{s,1}^m, W_{s,3}^m]$;
11 **for** $k \leftarrow k_{\text{last}}$ **to** n **do**
12 $\quad$ **if** $[W_{k,2}^m, W_{k,3}^m, W_{k,1}^m, X]$ *exist in* WR **then**
13 $\quad\quad k_{\text{last}} \leftarrow k$;
14 $\quad\quad n \leftarrow n+1$;
15 $\quad\quad W_s^m, W_n^m, K_n^m \leftarrow [W_{k,2}^m, W_{k,3}^m, W_{k,1}^m, X]$;
16 $\quad\quad$ Go to 5;
17 $\quad$ **else**
18 $\quad\quad$ **if** WR *is empty* **then**
19 $\quad\quad\quad$ **for** $k \leftarrow 1$ **to** m **do**
20 $\quad\quad\quad\quad K^k \leftarrow$ first three columns of K^k;
21 $\quad\quad\quad\quad$ Sort K^k row-wise;
22 $\quad\quad\quad\quad$ Remove duplicated rows in K^k;
23 $\quad\quad\quad\quad K_{\text{RD}}^{i,k} \leftarrow K^k$;
24 $\quad\quad\quad$ **end**
$\quad\quad\quad$ **Output:** Connectivity lists $K_{\text{RD}}^{i,1}, \ldots, K_{\text{RD}}^{i,m}$
25 $\quad\quad$ **else**
26 $\quad\quad\quad m \leftarrow m+1$;
27 $\quad\quad\quad$ Go to 2;
28 $\quad\quad$ **end**
29 $\quad$ **end**
30 **end**

constant and linear time respectively, while the steps (iii), (iv) and (v) have a time complexity of $O(M \log M)$. As a result the total time complexity of the generation and grouping of the wedges is $O(M \log M)$ while the memory complexity is $O(M)$. The time complexity is demonstrated for an exemplary geometry in Section 4 for meshes of different sizes. The most expensive part of the grouping process is the search for the next following wedge in step (iii). However, this step can be simplified if at the beginning a list of wedges is stored for each vertex with each vertex as the first entry. If the wedge $[W_{n,3}^m, W_{n,2}^m, W_{n,4}^m, X]$ has just been found in step (iii), it is thus sufficient to search for the following wedge in the remaining wedges that start with the index $W_{n,4}^m$. This means that only a small sub-list of wedges needs to be checked and updated.

Since the edge $[i, j]$ occurs in the next wedge in reverse order $[j, i]$, the normal vectors of the triangles of each domain are oriented in the same way, i.e. they all point inwards or outwards. The method therefore results directly in oriented connectivity matrices for each domain. The orientation, i.e. whether the normals are pointing inwards or outwards, is easily obtained by calculating the signed volume of the respective domain using Gauss's theorem. For this purpose, a vector field $\vec{F} = \vec{x}/3$ with divergence $\vec{\nabla} \cdot \vec{F} = 1$ is introduced. The application of Gauss's theorem yields for the volume:

$$V^{i,m} = \int_{\Omega^{i,m}} \vec{\nabla} \cdot \vec{F} \mathrm{d}V = \frac{1}{3} \int_{\partial\Omega^{i,m}} \vec{x} \cdot \vec{n} \mathrm{d}A = \frac{1}{3} \sum_{j=1}^{J_{i,m}} \vec{x}_j^{i,m} \cdot \vec{n}_j^{i,m} \tag{4}$$

where $\vec{x}_j^{i,m}$ are the vertex coordinates of an arbitrary vertex of the jth triangle of the mth domain of the ith region and $\vec{n}_j^{i,m}$ is the normal vector for the corresponding triangle. Normal vectors oriented outwards lead to a positive volume and normal vectors oriented inwards lead to a negative volume. To correct the orientation, it is sufficient to permute two arbitrary columns of the connectivity matrix $K_{\text{RD}}^{i,m}$. It should be noted that the connectivity matrix of the outer domain, i.e. the domain formed by all outer faces, is automatically found for each region. This outer domain corresponds to the outer domain of the entire model if there is only a single region in the geometry present. Once the calculation of the normal vectors is defined, the presented algorithm provides the same sign consistently for the volumes for all inner domains and just the opposite sign for all outer domains of each region independent of the input triangulation. This means that the sign of the volume alone can be used to decide whether the domain is an inner or outer domain of the investigated region.

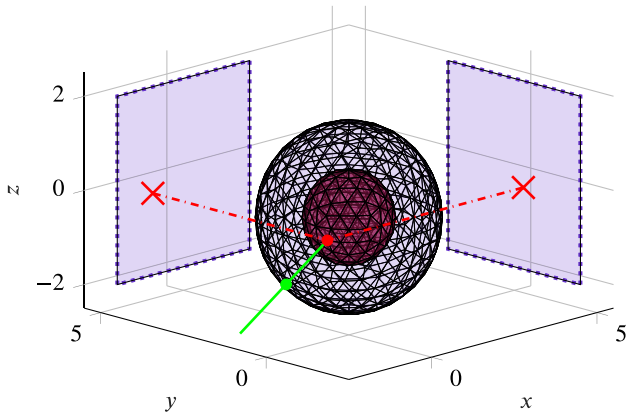

Fig. 5. To check efficiently whether a region K_{R} is nested in a region K'_{R}, an arbitrary vertex of the K_{R} is selected (red point). It is then tested whether the projections of this point (red crosses) lie within the projections of the bounding boxes of the region K'_{R} on two different coordinate surfaces (blue squares). Only if this is the case, nesting of K_{R} in K'_{R} is possible. Therefore, in the next step, a ray (green line) is shot from the selected point in an arbitrary direction and the parity of the number of intersections with the outer hull of the region K'_{R} is determined. In the depicted case, there is only a single intersection point (green dot) and since the number is odd the red region K_{R} is indeed nested in the region K'_{R}. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.3. Identification and correction of nested and multiple regions

The combination of the so far introduced algorithms for the identification of regions and the extraction of domains works for a geometry such as the one shown in Fig. 1(a). However, in the general case, regions may be nested inside of other regions or inside domains of other regions without a geometric connection, i.e. a shared vertex, of the corresponding connectivities. As an example, two nested spheres are considered as shown in Fig. 4(a). Applying the method discussed so far would identify the two spheres as separate regions with two domains each (an inner and outer domain). Correctly, however, this geometry is broken down into three domains: an inner sphere, the surrounding spherical shell, and the surrounding outer sphere as shown in Fig. 4(c).

In the following, it is shown how the results from Section 3.2 can be easily adapted to achieve the correct partitioning of the geometry. The first step is to test whether a region K_{R} lies inside another region K'_{R}. If this is the case, it is checked in which domain K'_{RD} of K'_{R} the

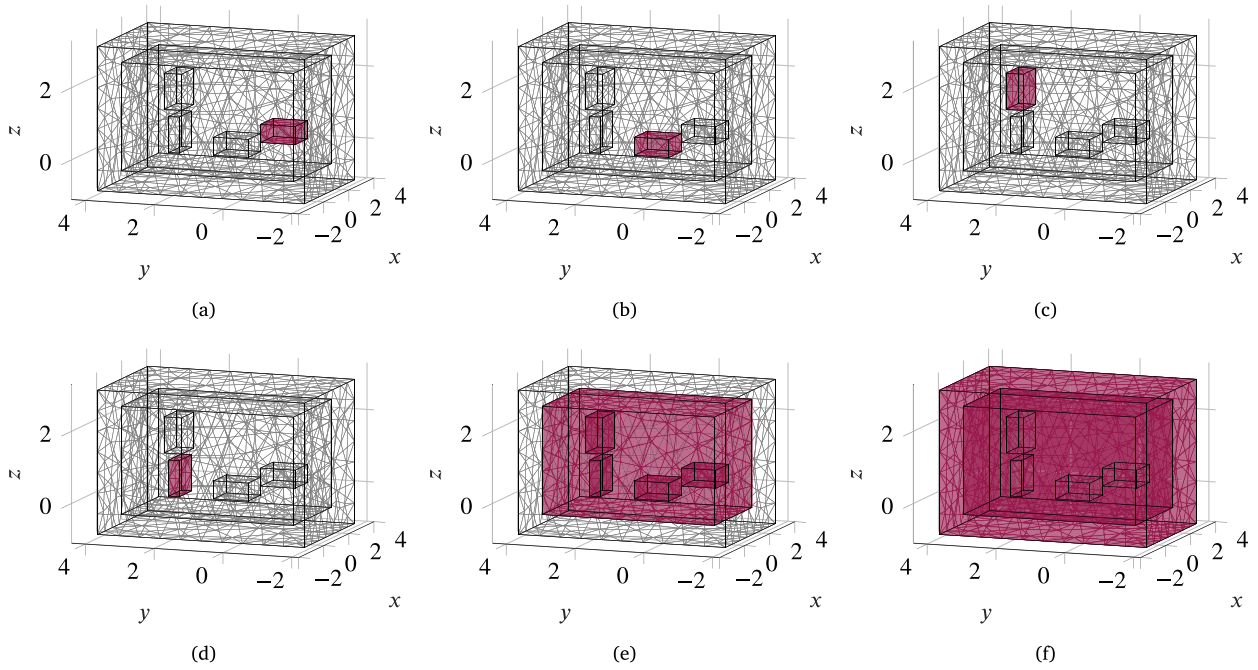

Fig. 6. Example of a double-nested geometry. The triangulation consists of 1034 elements. The individual extracted inner domains are highlighted in red. As can be seen, the individual domains are correctly identified. The identification of all regions as well as the complete extraction of all domains and the subsequent correction due to the nesting takes less than 50 ms with the provided program code on a conventional laptop with an i5-1135G7 processor @2.4 GHz and 8 GB RAM. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

region K_R is located. The connectivity lists of the surrounding domain K'_{RD} in K'_R and that of the outer domain K_{RD} of the nested region K_R are then concatenated. No columns need to be permuted to obtain the correct orientation of the normal vectors, since only the connectivity matrix of the outer domain of the nested region is appended, and their normal vectors are already oriented in the opposite direction to the inner domains of K_R. The two regions K_R and K'_R are henceforth considered to be one region. This procedure is repeated for all pairs of regions.

To test whether a region lies within another region or within a domain, the ray intersection method described by Möller [17] is used. Since an intersection of regions is not possible, we only need to check if a region is located fully inside or outside of another region. To do this, it is sufficient to test whether a single point of a region lies within another region. A ray is shot from the selected point in an arbitrary direction and the number of intersections with the outer domain of the region is determined. If the number of intersections is even, the point is located outside and if the number is odd, the point is inside. If the point lies inside, the test is repeated for each domain contained in the outer region. Since only individual points are tested and the number of regions is usually small, this step in geometry partitioning only plays a minor role for the total runtime. However, to further accelerate this test it is first checked whether the bounding boxes of the regions overlap and thus a nesting of the region is possible at all. For this purpose, it is sufficient to check if a single vertex lies inside of the projection of the bounding boxes onto two different coordinate planes. The process is illustrated in Fig. 5.

Once the correction of the nested geometry parts is completed, the final connectivity matrices K_D^i of the geometry were extracted.

4. Validation of the algorithm

The individual steps of the introduced algorithm were implemented in Matlab®. The corresponding program code can be obtained from Zenodo [19]. The library also includes several test geometries to validate the algorithm on realistic application examples. In the following,

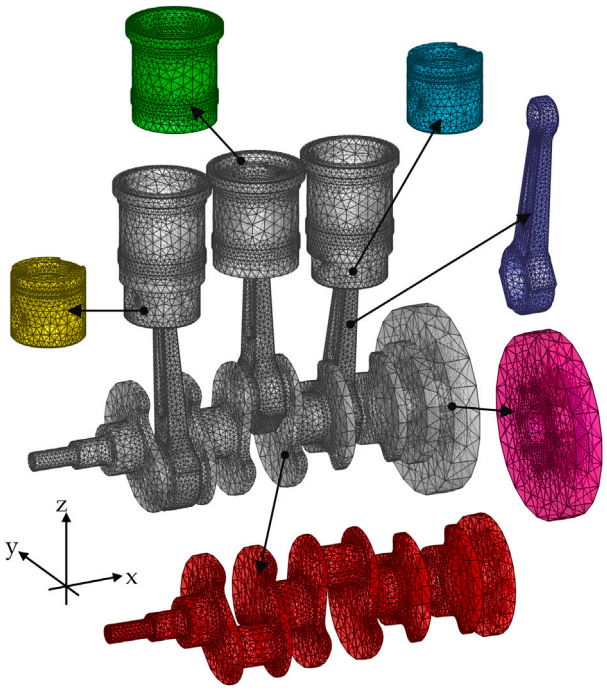

Fig. 7. Three-dimensional geometry that represents parts of a piston engine as an example of a realistic and complex geometry. The geometry was taken from the application library of COMSOL Multiphysics [18]. The geometry consists of a single region with 19 inner and one outer domain. Some of the extracted domains are shown in color. The corresponding STL-file representing the mesh is included in the supplied code library. The surface triangulation consists of 50 878 elements. The domain extraction takes about 2.5 s with the provided code on a standard laptop with an i5-1135G7 processor @2.4 GHz and 8 GB RAM.

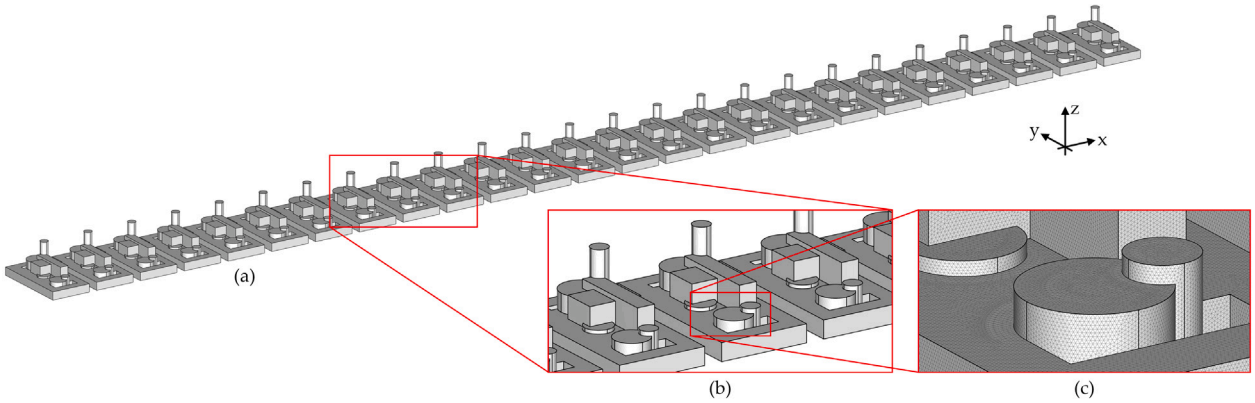

Fig. 8. (a) Complex geometry used to benchmark the runtime of the presented algorithm as a function of the number of mesh elements: The geometry consists of 25 regions. Each region consists of 40 domains. In total, the geometry therefore consists of 1000 inner domains and one outer domain. (b) Enlarged image with a detailed view of the geometry. (c) A further enlarged image showing the finest of the used surface meshes with over 5 million triangular elements.

the efficiency of the presented method is tested and evaluated on different geometry examples. Furthermore, the runtime of the algorithm is determined as a function of the number of mesh elements.

As a first example, the geometry shown in Fig. 6 is considered. The geometry contains regions that are nested multiple times. In addition, several inner domains are connected only at a single vertex or along a single edge. Even such a complicated geometry can be easily partitioned into its underlying domains. The complete partitioning of the geometry takes approximately 50 ms on conventional hardware, which illustrates the efficiency of the algorithm. As next example, the three-dimensional model of a piston engine is examined. This is an example of a realistic and complex three-dimensional geometry. It consists of only one region with multiple internal domains. The results of the partitioning process are shown in Fig. 7. This proves that the presented algorithm is suitable for the partition of complex and realistic geometries into their domains.

To investigate the dependence of the runtime on the number of mesh elements, the significantly more complex geometry shown in Fig. 8 is used. This geometry consists of 1000 inner domains and 50 regions, with each of the 25 outer regions containing one nested region. To test the performance of the introduced algorithm as a function of the number of elements, the geometry was triangulated with meshes of different resolutions. For each of the resulting meshes, all of the steps described in the previous sections are performed and the time required is determined. The resulting graph is shown in Fig. 9. Clearly, the runtime scales approximately like $O(M \log(M))$ demonstrating the high performance of the algorithm. This shows that even complex surface triangulations with millions of elements containing several hundred domains and regions can be efficiently partitioned.

5. Conclusion

An efficient algorithm for extracting all contained spatial domains in a given non-manifold three-dimensional surface triangulation was presented. The method is a generalization from two to three dimensions of the algorithm introduced by Jiang and Bunke and shares the high efficiency of their method, which is reflected in a runtime of approximately $O(M \log(M))$. A major advantage of the method is that no information about geometric features such as edge or face connectivity is necessary. The knowledge of the connectivity matrix and the vertex coordinates of the initial surface triangulation is sufficient. The method can also be used for handling large meshes with millions of triangular elements on simple hardware. It is therefore ideally suited for use in realistic modeling tasks or for use in efficient boundary element method software. Finally, it should be noted that algorithms based on the grouping of the face adjacency of the geometry represent potential alternatives, although the grouping of the faces into the individual domains represents a particular challenge.

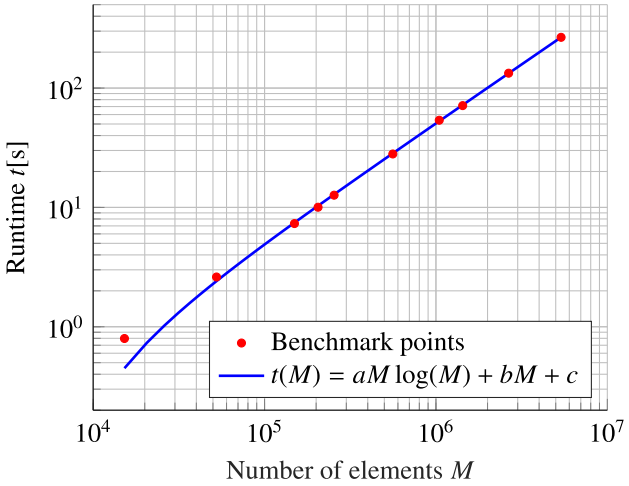

Fig. 9. Runtime as a function of the total number of mesh elements in the triangulation. The runtime refers to the total time required after the import of the mesh file for the division into regions, the extraction of all domains, and the correction due to nested domains averaged over 10 simulations. In all cases considered, more than 95% of the program's runtime was required for the extraction of the domains, which demonstrates that this step is decisive for the overall performance. The time spent on the domain extraction scales like $O(M \log(M))$, proving the high performance of the presented algorithm. All calculations were done on a conventional laptop with an i5-1135G7 processor @2.4 GHz with 8 GB RAM.

CRediT authorship contribution statement

Sebastian Bohm: Writing – original draft, Visualization, Software, Methodology. **Erich Runge:** Writing – review & editing, Validation, Project administration, Methodology, 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

We thank the German Federal Ministry for Economic Affairs and Climate Action for the funding of the research project within the framework of the Zentrales Innovationsprogramm Mittelstand (ZIM, funding number: ZF4457306PO9). In addition, we thank the staff of

the Zentrum für Mikro- und Nanotechnologien and of the HPC cluster of the Technische Universität for providing an excellent research environment.

Data availability

No data was used for the research described in the article.

References

[1] Sapountzakis EJ. Nonuniform torsion of multi-material composite bars by the boundary element method. Comput Struct 2001;79(32):2805–16. http://dx.doi.org/10.1016/s0045-7949(01)00147-x.

[2] Zhou A, Hui K, Lai Y. Simulating deformation of objects with multi-materials using boundary element method. Internat J Numer Methods Engrg 2008;74(7):1088–108. http://dx.doi.org/10.1002/nme.2202.

[3] Bordón JD, Álamo GM, Padrón LA, Aznárez JJ, Maeso O. MultiFEBE: A multi-domain finite element–boundary element solver for linear mixed-dimensional mechanical problems. SoftwareX 2022;20:101265. http://dx.doi.org/10.1016/j.softx.2022.101265.

[4] Hwang I-T, Ting K. Boundary element method for fluid-structure interaction problems in liquid storage tanks. J. Pressure Vessel Technol. 1989;111(4):435–40. http://dx.doi.org/10.1115/1.3265701.

[5] Everstine GC, Henderson FM. Coupled finite element/boundary element approach for fluid–structure interaction. J Acoust Soc Am 1990;87(5):1938–47. http://dx.doi.org/10.1121/1.399320.

[6] Opstal T, Brummelen E, van Zwieten G. A finite-element/boundary-element method for three-dimensional, large-displacement fluid–structure-interaction. Comput Methods Appl Mech Engrg 2015;284:637–63. http://dx.doi.org/10.1016/j.cma.2014.09.037.

[7] Bohm S, Runge E. Multiphysics simulation of fluid interface shapes in microfluidic systems driven by electrowetting on dielectrics. J Appl Phys 2022;132(22):224702. http://dx.doi.org/10.1063/5.0110149.

[8] Bohm S, Runge E. Efficient analytical evaluation of the singular BEM integrals for the three-dimensional Laplace and Stokes equations over polygonal elements. Eng Anal Bound Elem 2024;161:70–7. http://dx.doi.org/10.1016/j.enganabound.2024.01.013.

[9] Kumar A, Graham MD. Accelerated boundary integral method for multiphase flow in non-periodic geometries. J Comput Phys 2012;231(20):6682–713. http://dx.doi.org/10.1016/j.jcp.2012.05.035.

[10] Nagel M, Gallaire F. Boundary elements method for microfluidic two-phase flows in shallow channels. Comput & Fluids 2015;107:272–84. http://dx.doi.org/10.1016/j.compfluid.2014.10.016.

[11] Jacobson A, Kavan L, Sorkine-Hornung O. Robust inside-outside segmentation using generalized winding numbers. ACM Trans Graph 2013;32(4):1–12. http://dx.doi.org/10.1145/2461912.2461916.

[12] Jiang X, Bunke H. An optimal algorithm for extracting the regions of a plane graph. Pattern Recognit Lett 1993;14(7):553–8. http://dx.doi.org/10.1016/0167-8655(93)90104-l.

[13] Fan K-C, Chang C-Y. Surface extraction from line drawings of a polyhedron. Pattern Recognit Lett 1991;12(10):627–33. http://dx.doi.org/10.1016/0167-8655(91)90017-g.

[14] Hetroy F, Rey S, Andujar C, Brunet P, Vinacua A. Mesh repair with user-friendly topology control. Comput Aided Des 2011;43(1):101–13. http://dx.doi.org/10.1016/j.cad.2010.09.012.

[15] Yip M, Stahl A, Schellewald C. Robust hole-detection in triangular meshes irrespective of the presence of singular vertices. Comput Aided Des 2024;170:103696. http://dx.doi.org/10.1016/j.cad.2024.103696.

[16] Tarjan R. Depth-first search and linear graph algorithms. SIAM J Comput 1972;1(2):146–60. http://dx.doi.org/10.1137/0201010.

[17] Möller T, Trumbore B. Fast, minimum storage ray-triangle intersection. J Graph Tools 1997;2(1):21–8. http://dx.doi.org/10.1080/10867651.1997.10487468.

[18] COMSOL AB. COMSOL Multiphysics® v. 6.1, Stockholm, Sweden, (2024.05.17), https://www.comsol.com/.

[19] Bohm S. Software Domain_Extraction_Surface_Triangulation. 2024, http://dx.doi.org/10.5281/zenodo.13788175, URL https://zenodo.org/records/13788175.