Contents

1 Generation of Meshes for the Multiscale Models

Multiscale models of skeletal muscles describe phenomena on different length scales and combine them into a single description. The phenomena are modeled by different sets of equations which need individual discretizations and solvers. For that, various geometrical meshes describing different physical domains are required.

The discretization considered in this work involves three-dimensional (3D) and one-dimensional (1D) meshes. The solids of muscle and tendon are treated as 3D domains. Muscle fascicles and myofibrils are represented by 1D fibers that are embedded in the 3D domain of the muscle.

Generation of these meshes should be based on biomedical imaging data in order to represent actual human anatomy. The generated meshes should be of good quality such that finding numerical solutions with low errors is possible. Good mesh quality usually involves mesh cells with similar lengths and angles. It should also be possible to easily partition the mesh into multiple, equally sized subdomains. This is required for efficient parallel computation. The two requirements of good mesh quality and easy partitioning lead to the decision to employ *hexahedral* elements and a *structured* mesh for the 3D domains.

1.1 Overview and Notation of Required Meshes

In the following, we summarize the present meshes and introduce their notation used in the following discussions.

The domain of the muscle belly is denoted by Ω_M. A layer of fat and skin tissue is located on top of the muscle belly. It is denoted as the body domain Ω_B. The muscle belly

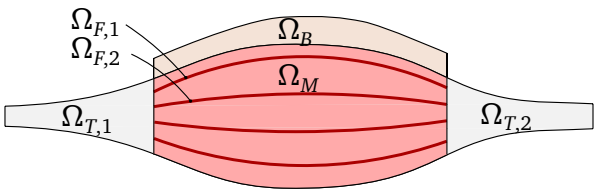


Figure 1.1: Visualization of the computational domains: tendons $\Omega_{T,1}, \Omega_{T,2}$, muscle belly Ω_M, body domain Ω_B and fiber domains $\Omega_{F,i}$.

is attached to tendons on both longitudinal ends. The tendon domains are denoted $\Omega_{T,1}$ and $\Omega_{T,2}$. These domains are all subspaces of the 3D Euclidean space, $\Omega_M, \Omega_B, \Omega_{T,i} \subset \mathbb{R}^3$.

Additionally, a number n_f of individual muscle fibers $\Omega_{F,i} \subset \mathbb{R}^3$ for $i \in \{0, \dots, n_f\}$ is introduced. Each fiber is a 1D manifold embedded in the 3D domain, i.e., $\Omega_{F,i} \subset \Omega_M$. Figure 1.1 summarizes the notation of the domains.

For the application of the Finite Element Method (FEM), we create meshes for each of these domains. Formally, a 3D mesh Ω_{3D} is given by a number of 3D elements $\{U_{3D,i}\}_{i=1,\dots,n}$ with $U_{3D,i} \subset \mathbb{R}^3$ such that their disjoint union approximates the domain, $\dot{\bigcup}_{i=1}^{n} U_{3D} \approx \Omega_{3D}$. Similar holds for 1D meshes.

The elements are non-overlapping and can be defined by nodes and edges. In the discretizations used here, no hanging nodes are allowed, i.e., at any node all adjacent elements share the node.

Furthermore, only structured, hexahedral meshes are considered in this section. A 3D structured mesh is isomorphic to a 3D cartesian grid with equidistant elements. This has advantages for programmatically indexing nodes and elements as well as for parallel partitioning of the domain.

The number n of 3D elements is the product of the numbers n_i, n_j and n_k of elements in the three coordinate directions x, y and z of the cartesian grid, i.e., $n = n_i\, n_j\, n_k$. Each element can be indexed by a triple (i, j, k) of indices with the ranges $i \in \{0, \dots, n_i - 1\}$, $j \in \{0, \dots, n_j - 1\}$ and $k \in \{0, \dots, n_k - 1\}$. In the program, typically, consecutive indices ι are used that iterate over all elements $\iota \in \{0, \dots, n-1\}$ and are obtained from the index triples by the mapping $(i, j, k) \mapsto \iota = k\, n_i\, n_j + j\, n_i + i$.

The elements of such a mesh can have different numbers of nodes depending on the desired spatial order of consistency of the Finite Element discretization. The number $n_{\text{el},d\text{D}}$ of nodes in a d-dimensional element is computed from the number of nodes $n_{\text{el,1D}}$ along one edge of the element as $n_{\text{el},d\text{D}} = n_{\text{el,1D}}^d$. Consequently, linear elements have two

nodes in 1D meshes and eight nodes in 3D meshes. Quadratic elements have three nodes in 1D and 27 nodes in 3D.

It is sufficient to develop a method for constructing structured 3D meshes with linear elements. Higher order elements can be composed geometrically by using the nodes of multiple adjacent linear elements. Therefore, at first we always generate meshes with odd numbers n_i, n_j and n_k of elements in the coordinate directions. Then, linear and quadratic meshes can be extracted from the set of nodes. Similarly, a linear 1D mesh with an odd n_i can be easily converted into a quadratic 1D mesh.

The next sections describe workflows and algorithms to construct 3D meshes for the domains $\Omega_M, \Omega_B, \Omega_{T,i}$ and 1D meshes for the fibers $\Omega_{F,i}$ based on anatomical information. Section 1.2 gives an overview over available meshing software tools and existing algorithms in the literature. Then, Sec. 1.3 presents a workflow to extract a smooth surface representation from anatomical imaging data. In Sec. 1.4, two serial algorithms are presented to generate 3D meshes and 1D fibers meshes. The next section, Sec. 1.5, extends these serial algorithms formulating a parallel algorithm and shows and discusses results. Finally, Sec. 1.6 gives a summary and concludes this chapter.

1.2 Related Works

Generating volumetric meshes for domains enclosed by a given surface is a task that is frequently needed in computational science. It is a preprocessing step whenever spatially discretized models should be solved numerically. In consequence, a vast amount of literature has addressed this algorithmic task and various approaches and methods have been proposed. Moreover, numerous software packages that solve this problem exist. Especially tools for Computer Aided Design and Engineering (CAD/CAE) as well as free and commercial preprocessing tools and Finite Element solver software include functionality to generate meshes from given surfaces.

An example from the biomechanical domain is [Unt13]. The study develops a Finite Element model of the lower limb of an occupant of a car with the aim to investigate injury scenarios during traffic crashes. The lower extremity geometry was obtained by computer tomography (CT) and magnetic resonance imaging (MRI) scan data of a 50th percentile male volunteer. Different meshes of bones and ligaments were created using the three tools IA-FEMesh, TrueGrid and Hyper-Mesh which will be outlined in the following.

IA-FEMesh (University of Iowa, Iowa City, USA) is an open source tool to generate hexahedral meshes [Gro09]. The user is presented with an interactive environment where existing geometries can be loaded. In a visualization window, bounding cuboids, called blocks, can be positioned around the geometry. A structured grid on the block is then projected onto the surface of the geometry. Multiple blocks can be placed to account for more complex geometry. The surface mesh is improved using Laplacian smoothing which equalizes the edge lengths of the elements. The interior nodes are generated using interpolation. The result will be a structured mesh if only one block is used or an unstructured mesh if multiple blocks are used. Further operations to manage mesh density, visually manipulate the meshes and add material properties, load and boundary conditions are available. The model can be exported in a file format for Finite Element analysis with ABAQUS.

The second tool is *TrueGrid* (XYZScientific Applications, Livermore, USA). It is a commercial toolkit to generate hexahedral meshes. The project was started in the early 1990s as the successor to the even older preprocessor software *INGRID*. The user interface still feels like being from those older times. Similar to *IA-FEMesh*, a projection method and multi-block technique is used. Some effort has been put into dealing with holes and sewing together dissimilar blocks.

The third tool is *Hyper-Mesh*, the commercial pre- and postprocessing toolkit of Hyperworks (Altair HyperWorks, Troy, USA). Altair sells infrastructure and solvers for a multitude of physics and is targeted at a wide range of industries. Being a commercial vendor, information about the internals of their preprocessing software are hardly provided.

More meshing software exists, such as CGALmesh [Jam15] for tetrahedral meshes. The package gives quality guarantees of their generated meshes and include four mesh optimization algorithms to further improve the mesh quality.

Another application-oriented work dedicated to the use of commercial tools is [Ram18]. A workflow for patient-specific modeling, simulation and analysis of the interaction between a residual lower limb stump and the socket of a prosthesis is presented. Imaging data was taken from magnetic resonance diffusion tensor imaging where also the preferred diffusion direction of water molecules along muscle fibers is captured. The open source tool MedInria (National Institute for Research in Digital Science and Technology (Inria), France) [Vic12] was used to extract muscle fibers. The residual limb data was processed using the commercial 3D image segmentation software Simpleware ScanIP

(Synopsys, Mountain View, USA). Auxiliary tasks were performed using MATLAB (MathWorks, Natick, USA) scripts. The commercial multiphysics solver LS-DYNA (LSTC/Ansys, Canonsburg, USA) was used for the simulations.

Commercial tools usually have the advantage that more development effort was put into them, than is possible for open source codes from the scientific community. This often leads to more robust and user-friendly software. An advantage of open source software is that the used algorithms are disclosed to everyone. They are often well documented or described in a publication. This allows to assess the expected quality of the generated meshes. Conversely, commercial vendors usually have no interest in revealing their internal algorithms.

For our simulation, structured, hexahedral meshes are needed. Several of the described tools are able to generate hexahedral meshes, however the meshes are typically unstructured. For our special need of 1D muscle fibers embedded in a 3D mesh, we develop our own method that is based on the ideas of existing algorithms. In the following, an overview over the algorithmic common knowledge of creating simplex meshes and hexahedral meshes is given.

Triangulating a 2D domain is the archetype of mesh creation. The triangulation named after B. Delaunay was formulated in 1934 [Del34]. For a given set of points, it maximizes the minimum angle of the triangles and, thus, avoids small angles. Therefore, a guarantee on the quality of the triangulation is given.

In 1995, J. Ruppert presented the Delaunay refinement algorithm [Rup95], which constructs a Delaunay triangulation conforming to prescribed connected points. This algorithm is still commonly used and also part of numerous derived meshing techniques.

In 1997, P. Chew developed an algorithm for meshing a 3D domain with tetrahedra [Che97] and proved that the aspect ratio of the tetrahedra is bounded, i.e., degenerate, “flat” tetrahedra, called slivers, are avoided.

The authors of [All05] propose a variational approach to meshing where a quadratic energy function is minimized. During minimization both vertex positions and connectivity are optimized. This leads to better quality meshes than by simple Delaunay triangulations.

Hexahedral meshes can be obtained from certain tetrahedral meshes by splitting up each tetrahedron into four hexahedra. This is discussed in [Epp99]. A remaining issue is that the generated meshes from this procedure are highly unstructured and some

hexahedra have poor quality. Instead, the goal would be to construct elements that are almost equilateral.

A different approach is to directly generate a hexahedral mesh for the given surface geometry. The survey in [Owe98] identifies four different strategies for generating unstructured hexahedral meshes.

The first one is a *grid-based* approach. It was introduced in [Sch96]; [Sch97]. The interior of a given solid is filled with a grid of as many hexahedral elements as fit into the space. Then the gaps at the surface are filled with additional elements. This method is robust but can lead to poor quality elements near the surface. The orientation of the interior grid highly depends on the orientation of the initial surfaces and may not be the natural orientation of the given volume.

The second approach for generation of hexahedral meshes are *medial surface methods* [Pri95]; [Pri97]. First, the volume is decomposed into subregions by medial surfaces such that the resulting domains are one of only 13 possible types. Predefined templates are used to fill the domains with hexahedral elements. Then, the continuity between the domains is ensured using linear programming. This approach gives good results for some geometry but has robustness issues when general geometry is considered.

The third approach is called *plastering*. It was first described by [Bla93] and continued by [Sta06]; [Sta10]. It is a moving-front method where hexahedral elements are placed in layers starting at the border and moving towards the interior. Intersection of faces has to be detected when the fronts meet in the interior and rules for connecting to existing faces have to be defined. During this process, complex shaped voids can occur in the interior. When it is no longer possible to fill the voids with hexahedrons already placed elements have to be removed. A new method, called unconstrained plastering, starts from an unmeshed volume boundary. The approach has general robustness issues and is not guaranteed to find a solution for arbitrary boundary.

The forth approach is *whisker weaving*, introduced by [Tau96] and extended by [Led08]; [Kaw08]. Here, the dual of the hexahedral mesh is considered. The dual consists of the three surfaces per hexahedron that lie in the planes of symmetry. The surfaces of all hexahedrons form topological loops. The principle is now to first construct the dual of the mesh, which can be easily determined from the given boundary surface. Then, the actual hexahedral mesh can be created from the dual, using the surfaces as guides where to place the elements. The dual forms topological loops inside the volume. One important criterion for generating good quality meshes is that self-intersections of these loops are resolved in a first step. The approach, used with subsequent smoothing, can

produce meshes of good quality. However, no guarantee is given. One problem is that the resulting mesh depends on the quality of the surface mesh and that the number of nodes can increase significantly during the method.

For the whisker weaving method and for some plastering methods, a quadrilateral mesh of the surface is required. Algorithms for creating high quality quadrangulations of closed surfaces exist [Don05]; [Kov11]; [Bes12]; [Men16].

Other approaches start with 3D volumetric medical imaging data instead of surfaces. In [Zha03]; [Zha05], adaptive tetrahedral and hexahedral meshes are created from volumetric data using octree subdivision. The method avoids hanging nodes and allows a feature sensitive adaptivity. While adaptive meshing methods can reduce the number of elements in the interior of the volume, a problem is that the worst quality elements are generated at the border, the location where the solution in a Finite Element study usually is the most interesting.

Multiple reasons make the previously outlined approaches unsuited regarding the needs for our parallel muscle simulation.

(i) The generated meshes are unstructured. When performing domain decomposition for parallel computing on unstructured meshes, graph-partitioning methods have to be used. Storing an unstructured mesh requires storage of element adjacency information. Partitioned meshes additionally require storage of the adjacent processes. In contracst, structured meshes can be trivially decomposed and stored efficiently. A decomposition can be represented in memory by a very low number of constant properties.

(ii) The presented methods are designed for hexahedral meshing of arbitrary volumes. Robustness and mesh quality at the same time remain issues that are not completely solved for most of the algorithms. Often, expensive smoothing steps are needed to increase mesh quality.

(iii) In general, either no assumption can be made about orientation or alignment of hexahedrons in the interior, or, for the grid-based approach, the elements at the surface have poor quality. Having a mesh that is consistently aligned with, e.g., the main diffusion direction or the preferential direction of the anisotropic material can reduce numerical errors in the Finite Element solution.

Consequently, a more scenario specific solution is needed that can avoid the mentioned issues. Such solutions can also be found in the literature. An example is [Ble05], where 3D Finite Element models for various complex muscle geometries around the hip are generated from magnetic resonance images. Segmentation and surface mesh generation

are performed using the old, unmaintained software *Nuages* (Inria, France). Then, a 3D hexahendral mesh is generated using TrueGrid. A structured template mesh on a unit square is mapped to the horizontal slices of the muscle geometry that resulted from the segmentation. After mesh smoothing, the slices are connected vertically to form a 3D mesh.

Fiber directions are described by Bezier curves in a reference volume and mapped to the muscle geometry using the same mapping. The fiber direction then are used in a transversely-isotropic material formulation. Simulations are performed using the Finite Element solver *Nike3D* (Lawrence Livermore National Lab, Livermore, USA) [Mak91].

We base our work on this study and use a similar mapping from a template mesh to the actual muscle volume. In comparison to [Ble05], we use an improved mapping based on harmonic maps, which potentially leads to better quality meshes on the slices of the muscle. Instead of the unit circle template mesh, we experiment with different reference meshes and evaluate their quality.

In the study of [Ble05], fiber directions are defined based on anatomically assumed directions. However, the definition is carried out on the cuboid reference geometry. This means that the authors mentally morph the muscle geometry into the reference geometry in order to define fiber directions, using their expertise. Then, the fiber directions together with the cuboid are transformed back to the actual geometry. This approach simplifies the definition of the fibers. However, defining the fiber direction directly on the muscle geometry can lead to better results. Thus, our approach is to automatically estimate fiber directions and define fibers directly in the muscle domain. At the same time, the 3D mesh and 1D fibers are aligned to allow for better numerical properties of the discretization.

The definition of fiber directions follows a method proposed in [Cho13]. The directions are assumed to follow a divergence free vector field. Such a field can be created by taking the gradient of the solution of the Laplace equation. Neumann boundary conditions are defined at the attachment points of the muscle tendon complex. The solution of the Laplace equation corresponds to the pressure values of a potential flow. Its gradient corresponds to the velocity and individual fascicles or fibers can be obtain by tracing streamlines through the velocity field. This approach is extended and validated by the studies in [Ino15] and [Han17]. We incorporate this method into our workflow.

1.3 Preprocessing of the Muscle Geometry

The first step towards creating a structured mesh is to obtain a representation of the surface of the muscle. Starting point is a human biomedical imaging data set. In this section, two possible workflows are presented how to extract the muscle and tendon surfaces from imaging data. The two workflows are visualized in Fig. 1.2. The workflow using the branch on the left side in Fig. 1.2 is automized but only works for the particular data set and extracting the biceps muscle. The right branch involves manual steps and is applicable for any muscle geometry.

1.3.1 Data Source

Anatomic images provide the basis for the extraction of muscle geometries. Our used data set originates from the Visible Human Project [Spi96] of the United States National Library of Medicine. The project has published anatomic images derived from a male cadaver, among other data sets. The data, known as “Visible Human Male”, were published in 1994. Colored images of transversal cross sections were obtain by cryosectioning. A total of 1871 images with dimensions of 2048 by 1216 pixels and 24 bit color depth visualize the whole human body. Parts of the upper arms are contained in approximately 500 of these images. The size of a pixel is 0.33 mm in transversal direction and 1 mm in axial direction. The size of the complete set of JPEG compressed images is 772 MB. Cropping and selecting the relevant portions of the upper arm extracts a dataset with the size of 35 MB.

An extract of an image of the upper arm is given in Fig. 1.3. The location of biceps and triceps brachii muscles can be identified in the dark red tissue. For the biceps, the two muscle heads are visible, separated by the bright diagonal line from bottom left to top right. For the triceps, at least two of the three heads can be identified. The blue background is colored frozen gelatin that was needed during cryosection to stabilize the arms.

1.3.2 Automatic Surface Extraction

This section outlines the automatic algorithm to obtain the muscle surface from the Visible Human Male data set. The scheme corresponds to the left branch in Fig. 1.2. The algorithm was implemented in a Python script as part of the Bachelor thesis of Kusterer

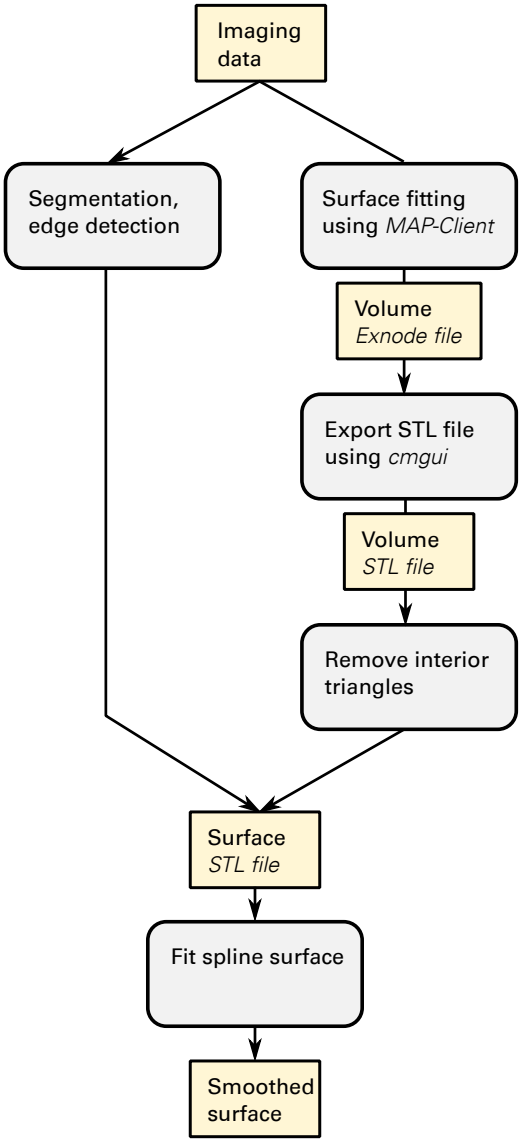


Figure 1.2: Workflow of generating a surface representation of the muscle and tendons from imaging data. Operations and intermediate results are shown as gray and yellow boxes, respectively. Two alternatives are given by the two branches. On the left, the imaging data is automatically processed to directly retrieve points on the surface of the muscle. The right branch achieves the same with three steps of which the first one involves manual adjustments. At the end, a spline surface smooths the collected data from both possibilities to yield the resulting surface representation.

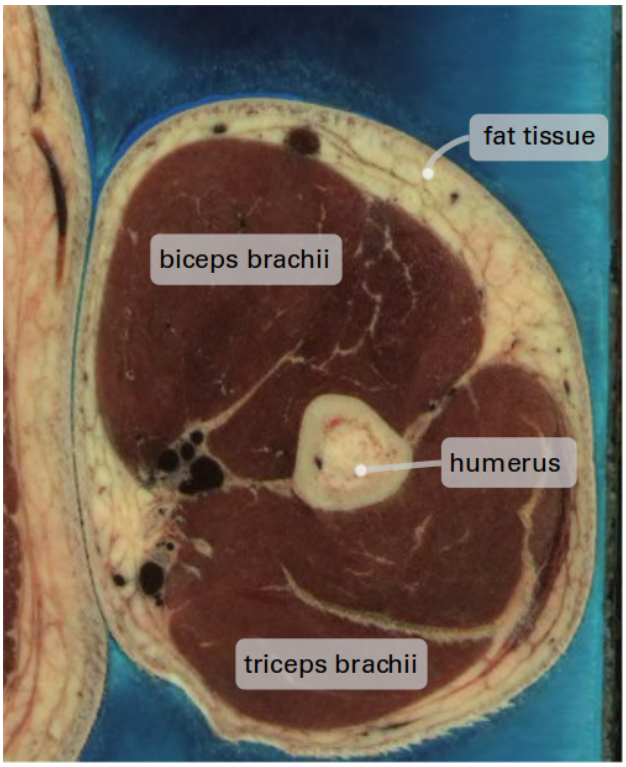


Figure 1.3: Exemplary extract of image number 483 from the Visible Human Male. A transversal slice of the left upper arm is shown as seen from the bottom. The biceps and triceps muscles as well as the humerus bone can be identified.

[Kus19] that was supervised by me. The algorithm is capable of extracting muscle and bone geometries from the mentioned imaging dataset.

At first, the color values in the images are used to segment the pixels into muscle tissue, surrounding tissue and skeletal structure. The algorithm traverses the selected and cropped relevant parts of the images. For example, to consider the biceps muscle, the image region of pixels coordinates (x, y) with $x \in [1300, 1720]$ and $y \in [1030, 1720]$ are considered in the images with numbers 284 to 778.

For every such part of an image, pixels that match a certain range in the RGB color space are marked and categorized. The categories are muscle tissue and, for demonstration, also bone tissue. The corresponding color ranges are given in Tab. 1.1.

The color based classification does not succeed everywhere as the white shade corresponds not only to bone material but also to fat and other tissue. Therefore, the algorithm removes artifacts located near the outer gelatine from the set of pixels that was categorized as bone.

	red	green	blue
muscle	$60-100$	$30-75$	$15-60$
bone	$145-255$	$135-205$	$60-160$

Table 1.1: Ranges in the RGB color space to identify pixels of muscle and bone segments. The numbers correspond to 24 bit colors with the range $[0, 255]$ for every color channel.

Exemplary results for image number 483 are given in the left column of Fig. 1.4. It can be seen that the marked regions for muscle and bone have gaps in the interior resulting from differently colored tissue inside muscles and bones. On some images, the set of pixels also includes small objects outside the actual muscle and bone regions.

To reduce the gaps and small objects, the morphological operations *closing* and *opening* are applied on the data. These operations consist of *dilation* and *erosion* steps. Both are pixel based operations that traverse the dataset and for every pixel consider a window of 3×3 pixels centered at the current position. Dilation picks the maximum value and erosion the minimum value from this window and assigns it as the pixel's value in a new image. In our case, values of zero and one correspond to non-categorized and categorized pixels, respectively.

Closing consists of dilation followed by erosion and closes small gaps or holes in the marked objects. Opening consists of erosion followed by dilation and removes small artifacts outside the actual bone and muscle areas. It was found effective to perform both dilation and erosion twice in sequence to yield good results containing almost no more holes nor unwanted small objects.

Next, the algorithm determines the contours of all regions with marked pixels. This leads to lines with a width of one pixel that enclose the muscle and bone areas. The right column of Fig. 1.4 shows the results after this step. It can be seen that numerous gaps have been closed by the morphological operations. In some images, as in the considered example, the muscle area gets split into multiple smaller enclosed regions, which is not desired. However, proper contours of the biceps are found in the majority of images.

In the next step, a single contour for each of muscle and bone is obtained in every image. If there are multiple contours per image, the one that is located the most in upper right location within the image is selected for the muscle. If all contours in an image are shorter than 20 pixels, this is an indication for bad segmentation quality and the whole image gets discarded.

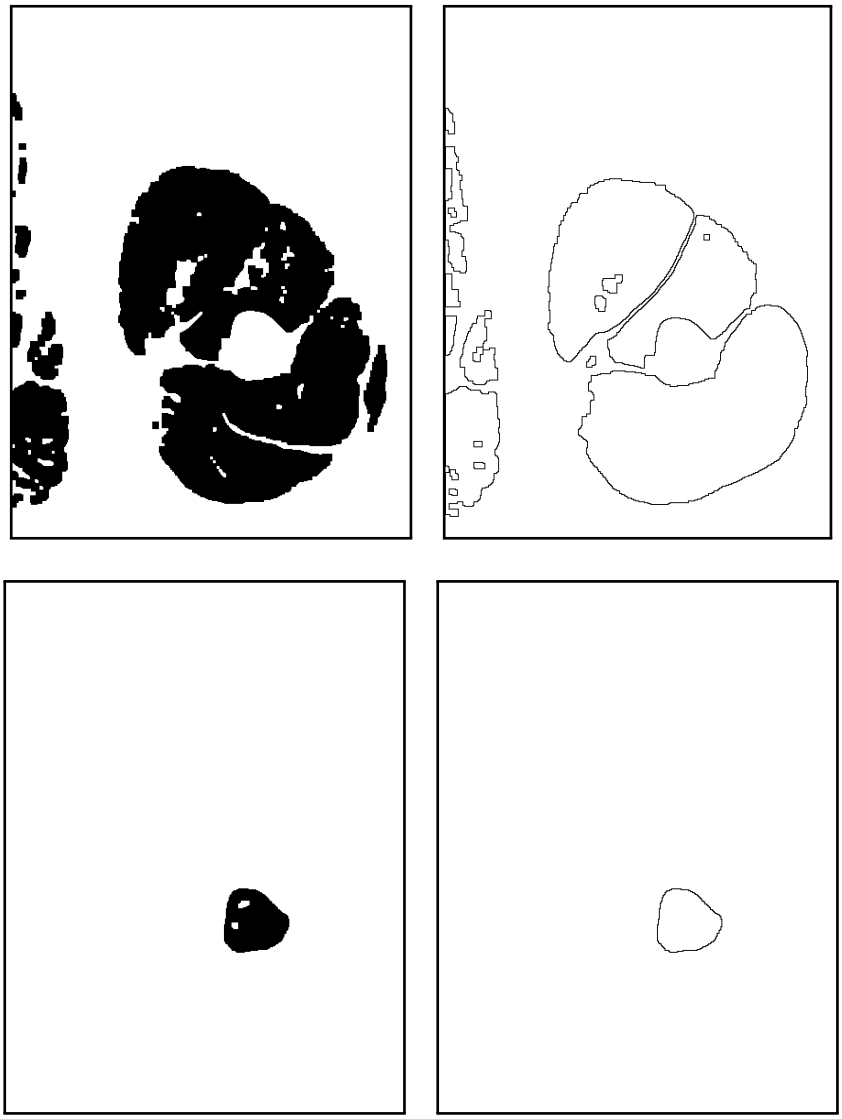
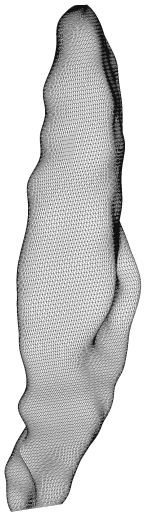


Figure 1.4: Intermediate steps of the algorithm to determine surface geometry of muscles and bones. The left columns shows pixels from the image in Fig. 1.3 that were categorized to be muscle tissue (top) and bone material (bottom). The right column shows a later step in the algorithm, where the surface of muscle (top) and bone (bottom) is estimated.



(a) Surface of the biceps brachii muscle. At the right side of the muscle, the groove of the humerus bone can be seen.

(b) Surface of the humerus

Figure 1.5: Resulting surfaces of biceps and humerus bone obtained by the automatic surface extraction algorithm.

The result is a set of contours for muscle and bone in the cross-sectional planes of the images. Combining these, we get a point cloud in 3D space that approximates the surface of the biceps muscle and the surfaces of the considered bones humerus, ulna and radius. Using these points, a spline surface can be fitted and subsequently triangulated. Resulting surfaces for the biceps and humerus bones are shown in Fig. 1.5.

The runtime for the algorithm is 121 min on a AMD Ryzen 5 1600 processor with 6 cores, 3.2 GHz and 16 GB RAM, of which a maximum of 2 GB was used at maximum. Because processing of the images can be done in parallel, the runtime can be reduced to approximately half (62 min) using 2 threads and to a quarter (30 min) using 6 threads.

The advantage of the presented algorithm is that the outcome solely depends on the imaging data and, thus, no modeling error by manual approximation of the geometry occurs. For example, the obtained surfaces of biceps and humerus geometrically fit perfectly into each other. Intermediate steps are stored as black and white images. By editing these between the steps of the algorithm, manual tweaking is possible and can be used to increase the quality of the results.

A disadvantage is that the algorithm relies on color information in the imaging data to differentiate between muscle and other tissue. Because the involved tissue has similar colors, these differences are often small. Furthermore, the color ranges need to be

determined experimentally. Therefore, the algorithm is not very robust with respect to image noise and needs adjustments when it should be used to extract other muscles. Expert knowledge about the location and shape of human muscles cannot be used easily to improve the results of the algorithm.

An alternative approach is to manually segment the imaging data and construct surfaces with the help of a tool. This approach is described in the following section.

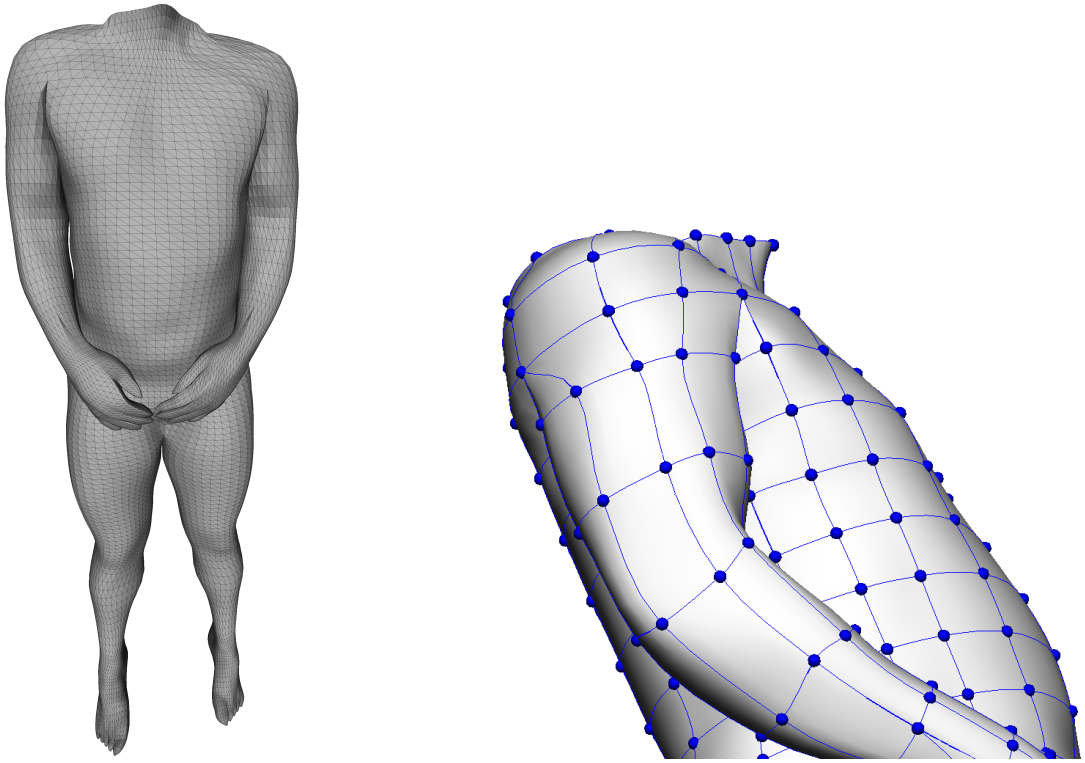
1.3.3 Manually Guided Surface Extraction

Manually guided segmentation can be done using the *MAP client* of the Musculoskeletal Altas Project (MAP) [Zha14]. This application allows to create and execute a workflow to achieve data processing and simulation tasks. In a graphical window, the user can place and connect various workflow steps. When executing the workflow, each step shows a dialog where the required configuration can be entered or the operations can be performed on a visual presentation of the data at this workflow stage.

Possible workflow steps include source and sink operations such as reading image data and writing meshes. Imaging data such as the 2D images from the Visual Human Male can be visualized in a 3D representation. The user can place points in the 3D space and try to match borders of the muscles and structures to extract from the data. Further workflow steps allow to create meshes of predefined geometrical shapes, such as cubes and cylinders and merge them into a common mesh. These meshes can be fitted to point clouds of user defined points. This is done by a least squares approach minimizing the distances between user created points and the mesh surface. Details can be found in [Fer18].

The MAP client has a plugin architecture and allows to create new workflow steps. It imports features from OpenCMISS, especially data processing formats and tools from OpenCMISS Zinc. Meshes can be created with 3D cubic Hermite elements that allow a high geometric modeling flexibility with a low number of nodes. Such meshes are stored in the OpenCMISS file format of `exnode` and `exelem` files.

As a result, meshes of individual muscles or the whole human organism can be created. Figure 1.6 shows meshes that were create from the cryosectioning data of the Visible Human Male. In Fig. 1.6a almost the whole body has been extracted. In Fig. 1.6b, the mesh consisting of cubic Hermite elements is visualized. A relatively coarse mesh width suffices to model a smooth surface of the body. When exported in the exfiles format from



(a) Mesh of the trunk and limbs, the surface has been triangulated for visualization.

(b) Detail view of part of the right upper arm and the trunk with blue nodes and edges of a cubic Hermite element mesh.

Figure 1.6: Mesh of the Visible Human Male from the Visible Human Project.

the MAP client, the data can be visualized, e.g., using *cmgui*, the visualization tool of OpenCMISS Zinc.

The mesh width of the meshes obtained using the MAP client was chosen such that the surface fitting yielded good results. The meshes are not necessarily ready for use in a simulation, especially if a high mesh resolution is desired. Apart from the mesh width also the type of elements can be different than what is needed for a Finite Element simulation. Our goal is to obtain meshes with linear or quadratic Lagrange elements with configurable mesh widths for the specified upper arm muscles, such as the biceps brachii.

Therefore, the next step of the workflow, as visualized by the right branch of Fig. 1.2, is to transform the volume mesh into a surface mesh which then can be used as start for further meshing. The further meshing steps are visualized in Fig. 1.7. The start is the Hermite mesh shown in the left-most image.

The Hermite elements can be triangulated and stored as an STL file using the tool `cmgui`. This process triangulates the non-planar faces of all Hermite elements. This leads to a

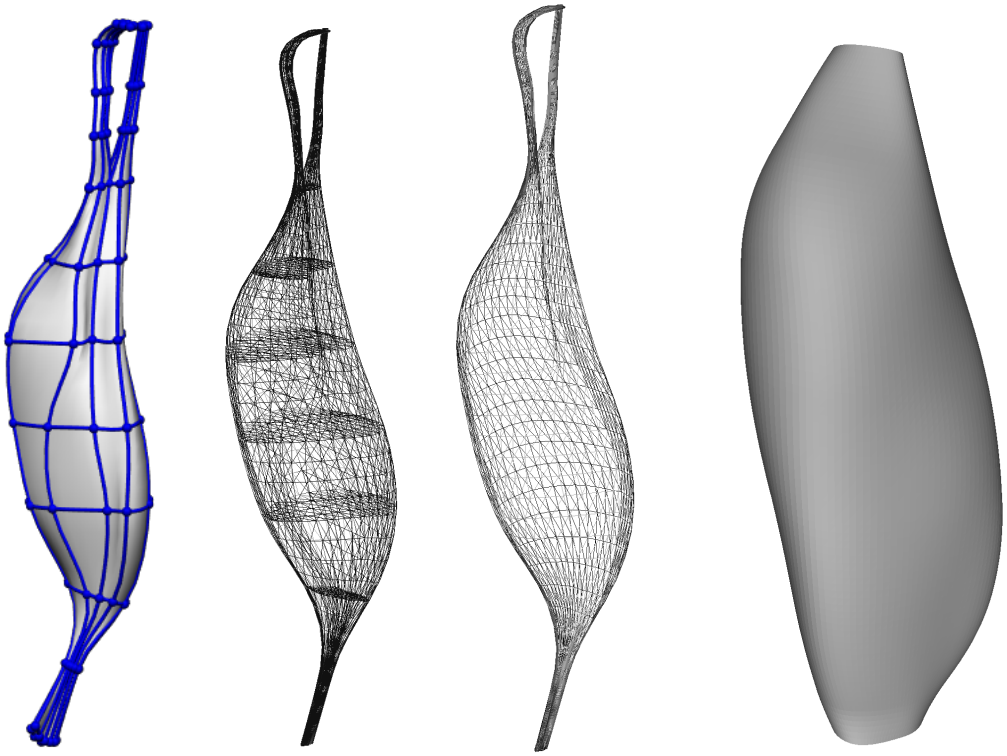


Figure 1.7: Processing the geometry of the biceps brachii muscle. From left to right: mesh with cubic Hermite elements, STL mesh with inside triangles, STL surface mesh where triangles lying inside have been removed, Spline surface of the muscle belly.

dataset with triangles both on the surface and in the inside of the volume, as can be seen in the second image of Fig. 1.7. At this stage, the use of the MAP and OpenCMISS related tools is finished and further processing steps are performed using tools from OpenDiHu that we developed on our own.

A Python script removes the triangles in the inside of the volume. The detection whether a triangle is inside the volume is done by casting four rays from the center of gravity of the triangle and determining if the rays intersect any other triangles. The rays have directions $(x, y, z) = (\pm 1, \pm 1, \frac{1}{3})$, where the z axis is oriented along the muscle and the x and y axes are oriented in radial direction. The ray-triangle intersection is done using the fast Möller-Trumbore algorithm [Möl97]. For every ray, all triangles are checked. Only if all four rays intersect at least one more triangle, the starting triangle is considered to be inside the volume and subsequently removed from the dataset.

This algorithm has a quadratic time complexity $\mathcal{O}(n^2)$ in the number of triangles n. It could be improved by organizing the triangles in a spatially adaptive data structure, such as an octree. However, since this preprocessing step has to be performed only once for a

given geometry the runtime is not a concern and there is no need for such optimization.

The result of this operation is a triangulated surface, shown in the third image of Fig. 1.2. The next step is to create a Spline surface of the muscle belly, as shown in the right-most image of Fig. 1.2. This is described in the next two sections.

1.3.4 Introduction of Spline Surfaces

The surface representation of the muscle could be obtain from either the left or the right branch of the preprocessing workflow in Fig. 1.2. The surface is given by a point cloud or a number of triangles. To remedy eventual outliers or unphysiological sharp edges from the segmentation, a Spline surface is fitted to the data. This leads to a smooth surface representation and later to a better conditioned Finite Element mesh in the simulation. However, this step is optional. It is also possible to use the surface triangulation from previous section 1.3.3 for the meshing algorithm in the next section 1.4.

The surfaces uses Nonuniform Rational B-splines (NURBS). A NURBS surface is a generalization of a B-spline surface. From a modeling point of view, B-spline surfaces have three advantages. First, the B-spline surface can be constructed with given smoothness properties. Second, the definition of a particular B-spline surface builds on geometric information, which simplifies their creation. More specifically a control polygon mesh in the 3D space is defined. Its convex hull is guaranteed to contain the surface. Third, the geometric parameters of a B-spline surface have only local impacts on the shape of the surface. This allows a B-spline surface of a fixed, low polynomical degree to approximate point clouds with any number of points without loosing approximation quality.

A limitation of B-spline surfaces is that circular and spherical shapes cannot be represented. This limitation is overcome by NURBS surfaces. NURBS surfaces are defined as the perspective projection into 3D space of a B-spline surface in 4D space.

The mathematical description is given in this section, following the notation of [Pie12]. The building blocks are the B-spline basis functions of polynomial degree p. Given a knot vector

$$\Xi = (\xi_1, \xi_2, \ldots, \xi_k), \quad \text{with } a = \xi_1 \leq \xi_2 \leq \cdots \leq \xi_k = b,$$

the ith B-spline basis function $N_{i,n}$ of degree n is defined recursively starting with the piecewise constant function $N_{i,0}$ for $n = 0$,

$$N_{i,0}(\xi) = \begin{cases} 1 & \xi_i \leq \xi < \xi_{i+1}, \\ 0 & \text{else}, \end{cases}$$

and using the following relation to define the functions of higher degree, $n > 0$,

$$N_{i,n}(\xi) = \frac{\xi - \xi_i}{\xi_{i+n} - \xi_i} N_{i,n-1}(\xi) + \frac{\xi_{i+n+1} - \xi}{\xi_{i+n+1} - \xi_{i+1}} N_{i+1,n-1}(\xi), \quad i > 0.$$

Because neighbouring entries in the knot vector can be equal, the fraction $0/0$ can occur. In this case, $0/0 := 0$ is defined.

A B-spline curve $\mathbf{C} \in \mathbb{R}^d$ of polynomial degree p is defined as

$$\mathbf{C}(u) = \sum_{i=1}^{\ell} N_{i,p}(u)\, \mathbf{P}_i, \quad u \in [a, b].$$

The coefficients $\mathbf{P}_i \in \mathbb{R}^d, i = 1, \ldots, \ell$ to the basis functions $N_{i,p}$ are called *control points* and define the control polygon. The number ℓ of basis functions and control points is determined from the number of knots k in an open knot vector and the polynomial degree p as $\ell = k - p - 1$.

The number of equal entries in series in the knot vector is the *multiplicity* of the respective knot value. Usually *open* knot vectors Ξ are used where the first and the last knot occur with a multiplicity of $p + 1$. This make the first and last points of the B-spline curve coincide with the control polygon, $\mathbf{C}(a) = \mathbf{P}_1$ and $\mathbf{C}(b) = \mathbf{P}_\ell$.

The multiplicities of the knots in the knot vector encode information about the smoothness of the B-spline curve. If the knot value $\hat{\xi}$ has a multiplicity of m, the B-spline curve will be $(p - m)$ times continuously differentiable at $\mathbf{C}(\hat{\xi})$.

An exemplary B-spline curve is shown in Fig. 1.8. It uses a *non-uniform* knot vector for polynomial degree $p = 3$, where the differences $\xi_{i+m} - \xi_i$ between neighbouring knot values vary. The effect of different multiplicities can be seen. The multiplicity $m = p = 3$ places the knot on the respective control point, as for $\xi = 49$ in the example. The multiplicity $m = p - 1 = 2$ places the knot on the control polygon, as in the example at $\xi = 10$. A lower multiplicity $m < p - 1$ does not yield a higher smoothness and in turn does not force the curve on the control polygon. It can also be seen that the B-spline

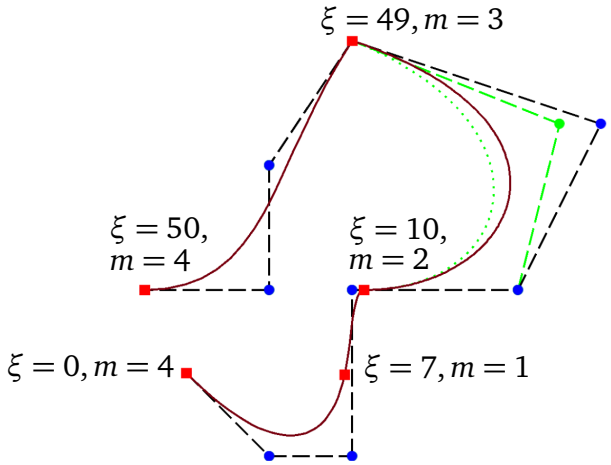


Figure 1.8: Exemplary B-spline curve (red) of degree $p = 3$ for the knot vector $\Xi = (0,0,0,0,7,10,10,49,49,49,50,50,50,50)$, control points (blue) and control polygon (black). Positions of the curve $\mathbf{C}(\xi_i)$ at the knots ξ_i are indicated by the red squares and the knot value ξ and its multiplicity m is given. The effect of moving one control point is shown in green.

curve stays inside the convex hull of the control polygon which is a property of B-spline curves [Pie12].

The effect of moving one of the 10 control points is visualized with green color in Fig. 1.8. The B-spline basis function $N_{i,p}$ has a local support of $S = (\xi_i, \xi_{i+p+1})$. Consequently, only the corresponding part of the curve, $\mathbf{C}(\xi)$ for $\xi \in S$ changes.

A B-spline surface $\mathbf{S}$ is given by the tensor product of two B-spline curves :

$$\mathbf{S}(u,v) = \sum_{i=1}^{\ell^{(1)}} \sum_{j=1}^{\ell^{(2)}} N^{(1)}_{i,p^{(1)}}(u) N^{(2)}_{j,p^{(2)}}(v) \mathbf{P}_{i,j}. \tag{1.1}$$

Here, we have two polynomial degrees $p^{(1)}$ and $p^{(2)}$, the ansatz functions $N^{(1)}$ and $N^{(2)}$ with number of ansatz functions $\ell^{(1)}$ and $\ell^{(2)}$ are constructed from corresponding knot vectors, each.

NURBS, B-spline curves and surfaces are formulated using *homogeneous coordinates*. Every point in Cartesian coordinates $(x,y,z) \in \mathbb{R}^3$ has a set of homogeneous coordinates $(\tilde{x},\tilde{y},\tilde{z},w) = (x\,w, y\,w, z\,w, w)$. Thus, the Cartesian coordinates can be obtain from the homogeneous coordinates by the *perspective division*, i.e., dividing all but the last coordinate by the weight w.

A NURBS surface is given by the same definition as the B-spline surface in Eq. (1.1) except that the control points $\mathbf{P}_{i,j} \in \mathbb{R}^3$ are enriched with scalar weights $w_{i,j}$ and, thus,

replaced by $(\mathbf{P}_{i,j}, w_{i,j}) \in \mathbb{R}^4$. The resulting surface $\mathbf{S}$ is given in homogeneous coordinates. Executing the perspective division yields the form:

$$\mathbf{T}(u,v) = \sum_{i=1}^{\ell^{(1)}} \sum_{j=1}^{\ell^{(2)}} R_{i,j}(u,v)\,\mathbf{P}_{i,j},$$

$$\text{with } R_{i,j}(u,v) = \frac{N_{i,p^{(1)}}(u)\,N_{j,p^{(2)}}(v)\,w_{i,j}}{\sum_{r=1}^{\ell^{(1)}} \sum_{s=1}^{\ell^{(2)}} N_{r,p^{(1)}}(u)\,N_{s,p^{(2)}}(v)\,w_{r,s}}.$$

The new rational basis functions $R_{i,j}$ and the possibly non-uniform knot vectors give rise to the name Non-Uniform Rational B-spline surface (NURBS).

1.3.5 Fitting a Spline Surface to the Muscle Geometry

In order to find a NURBS surface for the given triangulated surface of a muscle, at first, the part of the geometry corresponding to the tendons is removed such that the resulting triangles model only the surface of the muscle belly. The resulting belly has a length of 12.8 mm.

Then, twelve cross sections are extracted from the surface triangles. In result, we get twelve horizontal circumferential rings. On each ring, 9 equidistant points are determined. The first point is appended after the last point in every ring, such that in total we obtain a grid of 10×12 points.

Then, the least squares surface approximation algorithm by [Pie12] is used to fit a NURBS surface to the points. The implementation of the algorithm is given by the NURBS-Python (geomdl) library. Polynomial degrees of $p^{(1)} = 3$ and $p^{(2)} = 2$ are used where the first dimension corresponds to the cross-sectional direction of the muscle. The knot multiplicity is chosen as $m = 1$ for both coordinate directions. In result, we obtain a two times respective one times continuously differentiable surface in u and v direction. The resulting NURBS surface and the control polygon is visualized in Fig. 1.9. Note that the control polygon is different from the grid of points against which the surface is fitted.

Figure 1.10a shows the result of this approach in more detail. It can be seen that the surface is non-differentiable and has a kink at the seam line where the first and last points of each ring meet. The reason for this is that the surface fitting algorithm does not

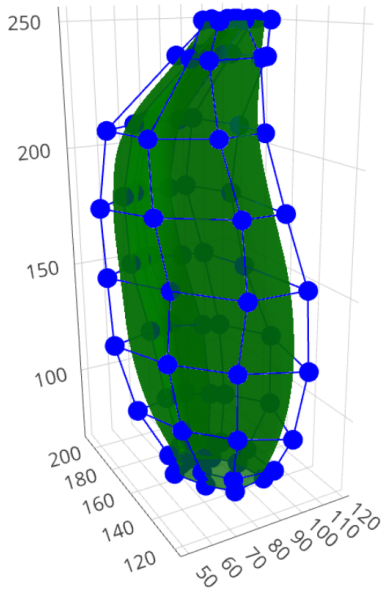


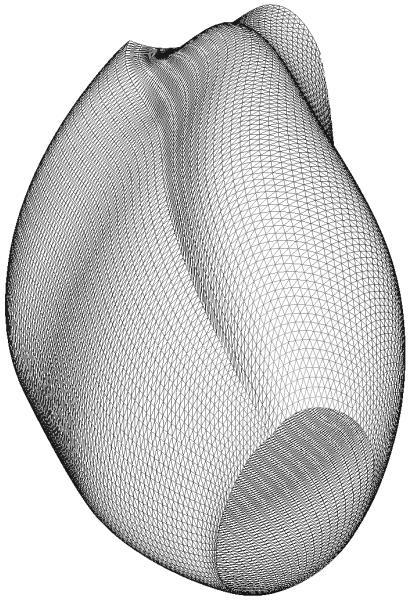
Figure 1.9: Fitted NURBS surface of the biceps muscle (green) and the control polygon (blue).

pose any conditions on the tangents at the edges of the fitted NURBS surface. Thus, the tangents mismatch.

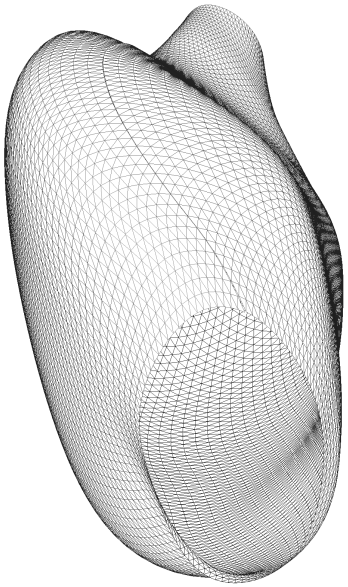
Since no implementation of a fitting algorithm specifically for a tubular NURBS surface with periodicity in tangential direction is available, we develop a different remedy. We modify the point grid that is used for the surface fitting. The series of 9 equidistant points on each ring is replicated twice and the first point is again added as the last point. This leads to a grid of $(3 \cdot 9 + 1) = 28 \times 12$ points which wrap around the muscle volume in cirumferential direction three times. The NURBS surface fitting algorithm is applied on this grid. The resulting NURBS surface also wraps around the muscle three times with the two ends being again not properly fitting to each other. From these three wraps, the middle one is extracted. In the biceps example, this corresponds to restricting the NURBS surface $\mathbf{T}(u, v)$ from $(u, v) \in [0, 1]^2$ to $(u, v) \in [0.4, 0.733] \times [0.1]$.

The result is depicted in Fig. 1.10b and it can be seen that the tangents now match very well between the two sides of the NURBS surface. Additionally, the comparison with the inital approach in Fig. 1.10a shows that an artifical bulge at the top of the muscle in the perspective of the visualization is removed. The overall shape of the muscle now looks smoother and more natural. Also, a comparison with the result of the automatic algorithm given in Fig. 1.5a shows that the results of our new approach are smoother.

The generated tubular surface has two holes at the top and bottom which prevent it



(a) First approach with 10×12 points. The kink at the seam line along the muscle is clearly visible.



(b) Second, improved approach with 28×12 points. It can be seen that the tangent at the seam line matches very well.

Figure 1.10: Fitted NURBS surface of the biceps muscle, triangulated for visualization purposes.

from being an enclosing surface to the muscle belly volume. The borders of these holes each lie in a plane and, thus, the missing surfaces are treated as being planar during the subsequent creation of the 3D meshes.

1.4 Serial Algorithm to Create Muscle and Fiber Meshes

Next, a 3D mesh for the muscle volume and 1D meshes for muscle fibers need to be generated from the surface representation described in the last sections. In this section, first an algorithm for the 3D mesh is described. Then, a second algorithm that reuses results from the first algorithm is presented which generates one dimensional meshes for muscle fibers. Both algorithms are executed in serial. A derived algorithm that can run in parallel and, thus, can handle larger datasets on a distributed memory hardware is given in the next section, Sec. 1.5.

The steps of a serial algorithm for the generation of a 3D mesh are given in Alg. 1. Input is the set of triangles at the tubular surface of the muscle. The tubular surface is

oriented along the z axis. In the following descriptions the muscle in considered to be oriented upright such that the z axis points in vertical direction towards the top. The borders at bottom and top have a constant z coordinate.

Algorithm 1 Serial algorithm for generation of 3D meshes

```
1 procedure Create_3D_mesh
  Input: Triangulated tubular surface
  Output: Structured 3D volume mesh

2   Slice geometry
3   Triangulate 2D slices
4   Compute harmonic maps u,v
5   Construct regular grid in parameter space and map to slices
6   Form 3D quadrilateral elements between the 2D slices' meshes
```

1.4.1 Slicing of the Geometry

The first step in line 1.2 slices the geometry. This means that horizontal *slices* of the cross-sectional area are extracted from the surface mesh. First, the muscle is divided into equidistant positions $z_i, i = 1,\dots,n$ along the z-axis where the slices are to be extracted. As can be seen in Fig. 1.11a, $n = 13$ z coordinates are selected. Next, for every position z_i, all surface triangles T_j that intersect the plane $Z_i = \{\mathbf{p} = (x,y,z) \mid z = z_i\}$ are considered and the intersection lines $P = T_j \cap Z_i$ are computed. The method of computing plane-triangle intersection is described in the following.

Given is a triangle T with points $\mathbf{p}^1, \mathbf{p}^2, \mathbf{p}^3 \in \mathbb{R}^3$ and a value $\hat{z}$, the wanted result is the set of points $P = T \cap \{\mathbf{p} = (\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z) \mid \mathbf{p}_z = \hat{z}\}$ which corresponds to a line segment $\overline{\mathbf{p}^a\mathbf{p}^b}$.

We describe the points in the triangle by two barycentric coordinates ξ_1 and ξ_2 as

$$\begin{aligned} &\mathbf{p}(\xi_1,\xi_2) = (1-\xi_1-\xi_2)\,\mathbf{p}^1 + \xi_1\,\mathbf{p}^2 + \xi_2\,\mathbf{p}^3, \\ &\text{with } \xi_1+\xi_2 \le 1, \quad 0 \le \xi_1, \xi_2 \le 1. \end{aligned} \tag{1.2}$$

By letting $\mathbf{p}_z(\xi_1,\xi_2) = \hat{z}$ we calculate the equation for the line segment $\overline{\mathbf{p}^a\mathbf{p}^b}$ in barycentric coordinates to be

$$\xi_1 = m \cdot \xi_2 + c, \quad m = -\frac{\mathbf{p}_z^3 - \mathbf{p}_z^1}{\mathbf{p}_z^2 - \mathbf{p}_z^1}, \quad c = \frac{\hat{z} - \mathbf{p}_z^1}{\mathbf{p}_z^2 - \mathbf{p}_z^1}, \quad \mathbf{p}_z^2 \neq \mathbf{p}_z^1.$$

For $\mathbf{p}_z^1 = \mathbf{p}_z^2 \neq \mathbf{p}_z^3$ we swap $\mathbf{p}_z^2$ and $\mathbf{p}_z^3$.

We consider the three sides $\overline{\mathbf{p}^1\mathbf{p}^2}, \overline{\mathbf{p}^2\mathbf{p}^3}$ and $\overline{\mathbf{p}^3\mathbf{p}^1}$ of the triangles and check which of them is intersected by the $z = \hat{z}$ plane.

1. On the triangle side $\overline{\mathbf{p}^1\mathbf{p}^2}$, the condition $\xi_2 = 0$ holds and the side intersects the plane at $\mathbf{p}(c, 0)$ iff $0 \leq c \leq 1$.
2. On the triangle side $\overline{\mathbf{p}^1\mathbf{p}^3}$, we have the condition $\xi_1 = 0$ and the side intersects the plane at $\mathbf{p}(0, -c/m)$ iff $m \neq 0 \wedge 0 \leq -c/m \leq 1$.
3. The third triangle side $\overline{\mathbf{p}^2\mathbf{p}^3}$ is intersected for $\xi_1 = (c+m)/(1+m)$ at $\mathbf{p}(\xi_1, 1-\xi_1)$ iff $m \neq -1 \wedge 0 \leq (c+m)/(1+m) \leq 1$.

If two of these three conditions for intersection of the triangle sides are met, there is an intersecting line segment $\overline{\mathbf{p}^a\mathbf{p}^b}$ with $\mathbf{p}^a \neq \mathbf{p}^b$ and the two intersection points $\mathbf{p}^a$ and $\mathbf{p}^b$ on the triangle sides need to be computed. The case $\mathbf{p}^a = \mathbf{p}^b$ and the case where two or more of the triangle points are lying on the $z = \hat{z}$ plane are handled separately.

After the presented computations are performed for all planes Z_i and all triangles T_j, we have a number of line segments that form a geometric "ring" for each z plane. The line segments are ordered according to their adjacency and a counter-clockwise orientation with respect to the z axis is ensured. The length of each ring is computed. A number $m = 16$ of equidistant points is selected on each ring.

Because the selected points on the rings are later used as border points of the volume mesh their position relative to each other should be in a tidy manner. When viewing the points on the surface in a $x-z$ or $y-z$ projection, they should approximately form a uniform regular grid like in the result in Fig. 1.11a. With given rings and number m of equidistant points per ring, only the position of one point per ring is not yet fixed. To close the definition, we formulate a first condition that relates the point positions of two neighbouring rings and a second condition for one point at the bottom-most ring.

The first condition should ensure that the point positions on neighbouring rings are as similar as possible. This is done by minimizing the distance between the one point on every ring that is fixed first. In the algorithm, the z planes are traversed from bottom to top. The first point $\tilde{\mathbf{p}}_{i,0}$ on a ring at $z = z_i$ is determined from the first point $\tilde{\mathbf{p}}_{i-1,0}$ of the previous ring at $z = z_{i-1}$ as the one with the minimal distance $|\tilde{\mathbf{p}}_{i,0} - \tilde{\mathbf{p}}_{i-1,0}|$. Thus, the searched point $\tilde{\mathbf{p}}_{i,0}$ has the property that the line between $\tilde{\mathbf{p}}_{i,0}$ and $\tilde{\mathbf{p}}_{i-1,0}$ and the tangent of the ring are perpendicular.

Given any point $\mathbf{p}$ on the ring at z_i and the tangent vector $\mathbf{u}$ at this point, we can project the connection vector $\mathbf{v}$ from $\mathbf{p}$ to the start point $\tilde{\mathbf{p}}_{i-1,0}$ of the previous ring, $\mathbf{v} = \tilde{\mathbf{p}}_{i-1,0} - \mathbf{p}$, onto the tangent $\mathbf{u}$. This leads to the plumb foot point $\mathbf{p}_0$ by the computation

$$\mathbf{p}_0 = \mathbf{p} + t\,\mathbf{u} \quad \text{with } t = \frac{\mathbf{v}\cdot\mathbf{u}}{|\mathbf{u}|^2}\,.$$

Performing this calculation for every line segment u on a ring allows to select the plumb foot $\mathbf{p}_0$ with the smallest distance to the start point $\tilde{\mathbf{p}}_{i-1,0}$ of the previous ring to be the start point $\tilde{\mathbf{p}}_{i,0}$ of the current ring. This is the point that fulfills the first condition.

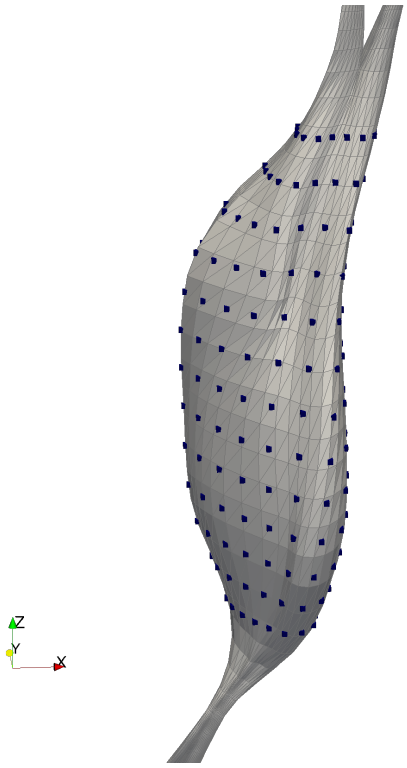
With this first condition all points are only fixed relative to each other. The definition of one point, the start point $\tilde{\mathbf{p}}_{0,0}$ of the bottom-most ring, is missing. The second condition fixes this point by a prescribed plane at $x = \hat{x}$ and selects $\tilde{\mathbf{p}}_{0,0}$ such that its x coordinate lies in this plane. From the (usually two) points that meet this condition, the one with lower y coordinate is selected. The actual value of $\hat{x}$ is determined experimentally such that the resulting point positions are visually uniform. Not every value leads to a good result because of the shape of the biceps muscle, especially the groove where the humerus bone is located.

The resulting grid of points on the biceps surface is visualized in Fig. 1.11a. It can be seen that all points of the same ring have the same z coordinate. By connecting neighbouring points horizontally and vertically, a regular grid can be formed. This overall grid in this $x-z$ perspective view looks relatively uniform, e.g., compared with the gray surface triangulation mesh of the biceps geometry. The spacing between the points is lower at the top and bottom of the muscle because of the smaller circumference at these locations.

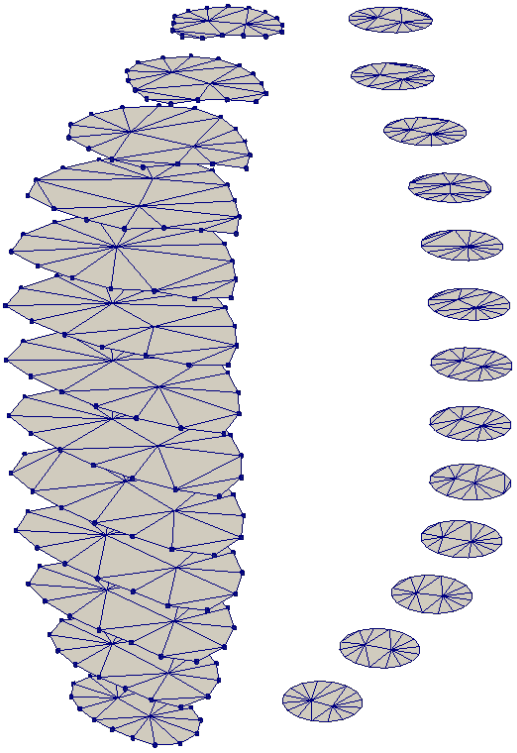
1.4.2 Triangulation of the Slices

The points of each ring enclose a planar, polygonal surface, a *slice* S_M of the muscle. The next step in the algorithm, line 1.3, is to triangulate the extracted slices, i.e. construct triangles that decompose the polygons. The result of this step is visualized at the left side in Fig. 1.11b.

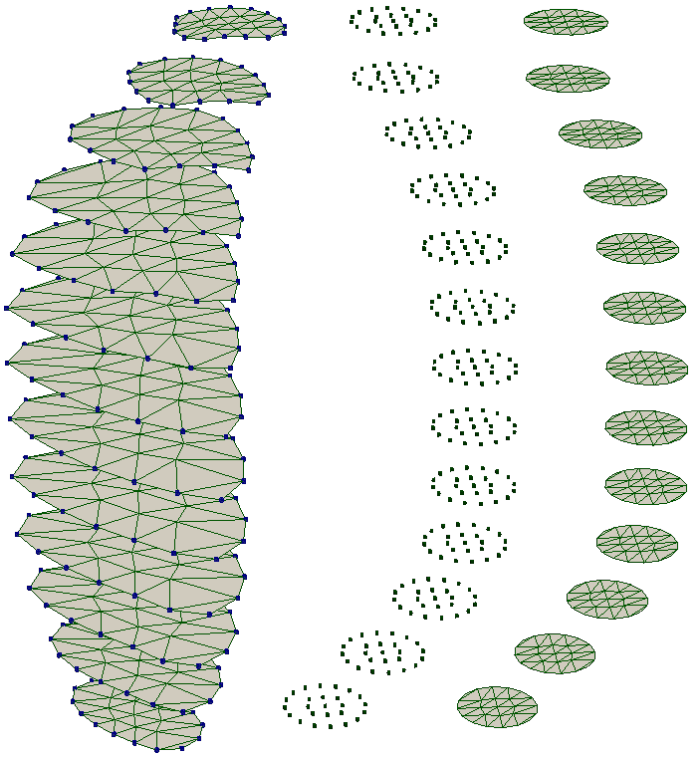
We select three different methods to construct this triangulation. The first and second methods are based on Delaunay triangulations. The third method creates a custom triangulation using a simple construction scheme with only one additional point. Figure 1.12 visualizes results of the three methods for one slice.



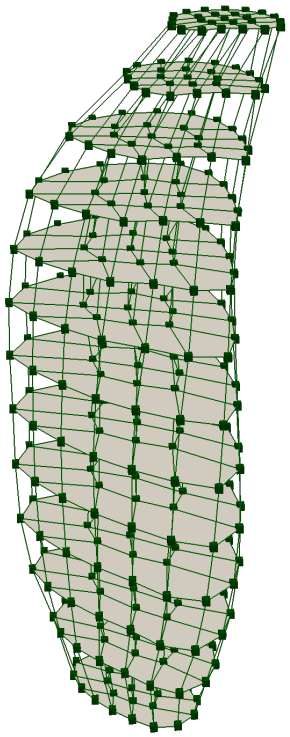
(a) Extracted border points (blue) on the biceps surface mesh (gray). This is the result of line 1.2.



(b) The generated triangulation of the slices (left) and the image $\mathbf{y}(\mathbf{x})$ of the triangulation under the harmonic map (right). This figure shows the result of lines 1.3 and 1.4.



(c) Grid in parameter space (right) and muscle domain (left), result after line 1.5.



(d) 3D elements formed by connecting the slices, result after line 1.6.

Figure 1.11: Steps of the serial algorithm, Alg. 1, executed directly on the surface mesh of the biceps muscle (not the B-spline surface).

The first method uses the tessellation algorithm from the spatial algorithms and data structures module of the Python package *SciPy*. The Quickhull algorithm [Bar96] is used which triangulates the convex hull of the points. In consequence, the triangulations of concave slices have triangles that lie outside the interior of the slice, which is a disadvantage. An advantage is that the triangulation uses all given points and no new points are added. However, this often results in meshes of lower quality than if adding additional points were allowed. The example in Fig. 1.12a shows such a concave slice. At the bottom of the domain, the triangles are outside the slice and almost degenerate.

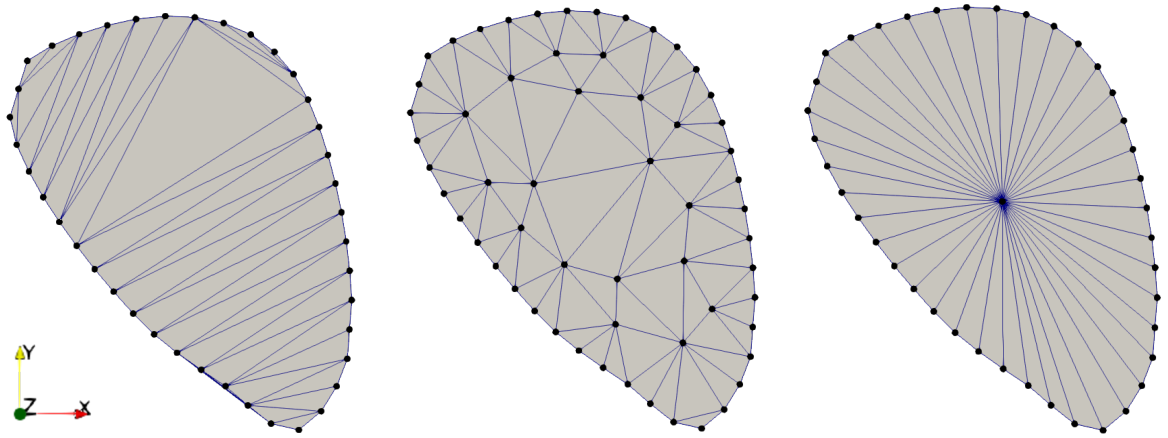
The second method uses a Delaunay refinement algorithm described in [She02] and implementated in the *Triangle* software [She96]. A conforming, constrained Delaunay triangulation is created. The triangulation correctly handles convex and concave domains. Conforming means that the triangulation uses the given points at the border. Additional points at the border as well as in the interior are added. The triangulation is constraint to generate triangles with minimum angles of 20 degrees and a maximum area A that is set to a value dependend on the area of the bounding box. In consequence, all generated triangulations have a guaranteed mesh quality in terms of angles and about the same size and number of triangles.

Comparing the result of the second method in Fig. 1.12b with the result of the first method in Fig. 1.12a shows the better triangulation quality as the triangles all have higher angles.

The third method places one additional point in the center of gravity of the given points. Triangles are constructed by connecting the center point with two adjacent points on the border, for all given points. The resulting triangulation resembles a pie chart. For some extreme concave slices, this method also creates triangles that partly lie outside the slice, but this occurs rarely with muscle cross-sections. The advantage of this approach is its simplicity. Figure 1.12c shows the result for an exemplary slice. Here, in contrast to the first method, the third method creates a valid triangulation despite the concave domain.

1.4.3 Harmonic Maps

Next, harmonic maps are created that allow to smoothly map a given 2D reference mesh onto an actual cross-section of the muscle. The initial application of harmonic maps to meshes used for biomedical simulations is given by [Mar10] and [Mar11]. The authors improve a given, oversampled surface mesh obtained from classical segmentation. This is done by partitioning the surface into multiple mesh partitions of zero genus (i.e.,



(a) First triangulation method (b) Second triangulation method (c) Third triangulation method

Figure 1.12: Result of different triangulation methods for a slice in the center of the biceps muscle.

containing no holes) and transforming them to a reference space using harmonic maps. There, controlled remeshing is carried out before the transformation is reversed.

In our algorithm, harmonic maps are also used for the purpose of generating high quality meshes. In contrast to the literature, the mapping is based on the muscle slices instead of the surface. Also, different parameter domains are investigated.

In the algorithm, the next step is line 1.4, to compute the harmonic maps u and v. For a given slice S_M, the functions u and v map from points $\mathbf{x} \in S_M$ to coordinates $u(\mathbf{x}), v(\mathbf{x}) \in \mathbb{R}$ of a parameter domain $\Omega_P \subset \mathbb{R}^2$. The parameter domain is either a unit circle or a unit square.

The vector $\mathbf{y}(\mathbf{x}) := (u(\mathbf{x}), v(\mathbf{x}))^\top$ for $\mathbf{x} \in S_M$ is interpreted as position in Ω_P. The maps are constructed such that the border ∂S_M of the slice S_M is evenly mapped to the border $\partial \Omega_P$ of the parameter domain Ω_P. The mapping $\mathbf{y} : S_M \to \Omega_P$ is bijective and harmonic, i.e. the Laplacians of u and v are zero.

More specifically, the definition is

$$\begin{aligned} &u : S_M \to \mathbb{R}, \quad v : S_M \to \mathbb{R}, \\ &\Delta u(\mathbf{y}) = 0, \quad \Delta v(\mathbf{y}) = 0 \quad \forall \mathbf{y} \in S_M. \end{aligned} \tag{1.3}$$

For a uniform parametrization $\mathbf{p} : [0,1] \to \partial S_M$ of the border ∂S_M of the slice, i.e.,

$$\frac{\partial \ell(t)}{\partial t} = c \in \mathbb{R} \quad \forall t \in [0,1], \quad \text{where } \ell(t) := \int_0^t |\mathbf{p}'(\tau)| \, \mathrm{d}\tau,$$

we require the image of the border parametrization in Ω_P to be also uniform, i.e.,

$$\frac{\partial \ell_P(t)}{\partial t} = c_P \in \mathbb{R} \quad \forall t \in [0,1], \quad \text{where } \ell_P(t) := \int_0^t |\mathbf{y}'\big(\mathbf{p}(\tau)\big)| \, \mathrm{d}\tau.$$

Corresponding border points $\mathbf{x}_{\mathrm{M,border}} \in S_M$ and $\mathbf{x}_{\mathrm{P,border}} = (u_{\mathrm{P,border}}, v_{\mathrm{P,border}})^\top$ can be defined. This leads to the following Dirichlet boundary conditions that close the definition in Eq. (1.3):

$$u(\mathbf{x}_{\mathrm{M,border}}) = u_{\mathrm{P,border}}, \quad v(\mathbf{x}_{\mathrm{M,border}}) = v_{\mathrm{P,border}}. \tag{1.4}$$

Equations (1.3) and (1.4) describe a boundary value problem of ordinary differential equations in u and v. We solve it using the Finite Element Method and the spatial discretization given by the triangulation of the slices. Depending on the method of triangulation, a different number of degrees of freedom is given. For the first method with the Quickhull algorithm, no degree of freedom is present and no system of equation needs to be solved. For the third method, only one degree of freedom for the center point needs to be computed. The second method has as many degrees of freedom as there are additional points inserted during the Delaunay refinement.

The first step is to compute the prescribed border points $\mathbf{x}_{\mathrm{P,border}}$ in parameter space. When using the first and third triangulation methods, the border points on the slices are equidistant and therefore the same number of points need to be sampled equidistantly on the border $\partial\Omega_P$ of the parameter space. If the second triangulation method which potentially adds additional points is used, the same number of points are created on the border $\partial\Omega_P$ of the slice as are given on the slice ∂S_M. The border points are created such that the relations of their distances is the same on $\partial\Omega_P$ as for the original points on ∂S_M.

Using the standard procedure of the Finite Element method for $\Delta u(\mathbf{x}) = 0$ on S_M and $u = f(\mathbf{x})$ on ∂S_M, e.g. as outlined in [Rem10], leads to the weak form with ansatz and

test functions ϕ,

$$\int_{S_M} (\nabla u^\top \nabla \phi + \nabla f(\mathbf{x})^\top \nabla \phi)\, \mathrm{d}\mathbf{x} = 0 \quad \forall \phi \in \mathcal{H}_0^1. \tag{1.5}$$

Standard linear hat functions are used on the triangles, such that they provide the interpolation property $\phi_i(\mathbf{x}_j) = \delta_{ij}$. Using the barycentric parametrization of triangles with points $\mathbf{p}^1, \mathbf{p}^2$ and $\mathbf{p}^3$ introduced in Eq. (1.2), we define the ansatz functions and get their derivatives within the elements by:

$$\phi_1^{(e)} = (1-\xi_1)(1-\xi_2), \qquad \phi_2^{(e)} = \xi_1(1-\xi_2), \qquad \phi_3^{(e)} = (1-\xi_1)\xi_2,$$

$$\nabla\phi_1^{(e)} = (\xi_2 - 1, \xi_1 - 1)^\top, \qquad \nabla\phi_2^{(e)} = (1-\xi_2, -\xi_1)^\top, \qquad \nabla\phi_3^{(e)} = (-\xi_2, 1-\xi_1)^\top.$$

The superscript $\square^{(e)}$ refers to the definition of the functions within elements. The global assembly involves composing the global nodal functions $\phi_i(\mathbf{x})$ for nodes indexed by $i = 1, \ldots, n_{\text{nodes}}$ and using a mapping between the barycentric coordinates $\xi_1, \xi_2 \in [0,1]^2$ inside the elements to the global coordinates $\mathbf{x} \in S_M$. Inserting the discretization

$$u_h(\mathbf{x}) = \sum_{i=1}^{n_{\text{nodes}}} u_i\, \phi_i(\xi_1(\mathbf{x}), \xi_2(\mathbf{x}))$$

into Eq. (1.5) leads to the form

$$\sum_{i=1}^{n_{\text{nodes}}} u_i \int_{S_M} \nabla_{\mathbf{x}} \phi_i^\top \nabla_{\mathbf{x}} \phi_j \, \mathrm{d}\mathbf{x} + \sum_{i=1}^{n_{\text{nodes}}} f_i \int_{S_M} \nabla_{\mathbf{x}} \phi_i^\top \nabla_{\mathbf{x}} \phi_j \, \mathrm{d}\mathbf{x} = 0 \quad \forall j = 1, \ldots, n_{\text{nodes}}. \tag{1.6}$$

The integrations are executed element-wise and over the elemental coordinates ξ_1, ξ_2. The transformation to elemental coordinates involves the computation of the Jacobian $J = \mathrm{d}\mathbf{x}/\mathrm{d}\boldsymbol{\xi}$ of the mapping between element coordinates $\boldsymbol{\xi} = (\xi_1, \xi_2)$ and global coordinates $\mathbf{x}$. From the definition in Eq. (1.2) it follows that

$$J = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\boldsymbol{\xi}} = [\mathbf{p}^2 - \mathbf{p}^1, \mathbf{p}^3 - \mathbf{p}^1].$$

The metric tensor for this mapping is given by

$$\mathcal{M} := \left(\frac{\mathrm{d}x}{\mathrm{d}\xi}\right)^T \left(\frac{\mathrm{d}x}{\mathrm{d}\xi}\right).$$

The transformation of the integrals in Eq. (1.6) introduces an additional integration factor of $\sqrt{\det \mathcal{M}}$.

We get the following matrix equation:

$$M_u \mathbf{u} = -M_f \, \mathbf{f}, \tag{1.7}$$

with the vector $\mathbf{u}$ of nodal solution values, the vector $\mathbf{f}$ of nodal Dirichlet boundary condition values at the border and the global stiffness matrices M_u and M_f. The two global stiffness matrices are assembled from the element stiffness matrices $M^{(\mathrm{e})}$ for the degrees of freedom at all nodes respectively at the border nodes. The entries of the element stiffness matrices are given by

$$M^{(\mathrm{e})}_{i,j} = \int_0^1 \int_0^{1-\xi_1} \nabla \phi_i^{(e)}(\xi_1, \xi_2)^\top \, \mathcal{M}^{-1} \, \nabla \phi_j^{(e)}(\xi_1, \xi_2) \, \sqrt{\det \mathcal{M}} \, \mathrm{d}\xi_2 \mathrm{d}\xi_1.$$

By solving Eq. (1.7) for $\mathbf{u}$ we get the discretized harmonic map u. The Finite Element formulation and computation for v is analogous and uses the same global stiffness matrices.

Figure 1.13 visualizes the triangulation of S_M and the solutions $u(\mathbf{x})$ and $v(\mathbf{x})$ in the first two plots. The color range from bright yellow to dark violet corresponds to increasing values of u and v. It can be seen that the u values increase from left to right whereas the v values increase from bottom to top, corresponding to the horizontal and vertical coordinate axes y_1 and y_2 of Ω_P.

Applying the computed harmonic map $\mathbf{y}(\mathbf{x})$ to the triangulation of the slices results in a triangulation of the parameter domain Ω_P. This is shown in the third plot of Fig. 1.13 and in Fig. 1.11b. In both figures, the triangulation of the slices was generated using the second triangulation method with the constrained Delaunay triangulation. On the right of Fig. 1.11b, the image $\mathbf{y}(\mathbf{x})$ under the harmonic map of the triangulation in the slices is shown on the unit circle parameter domains.

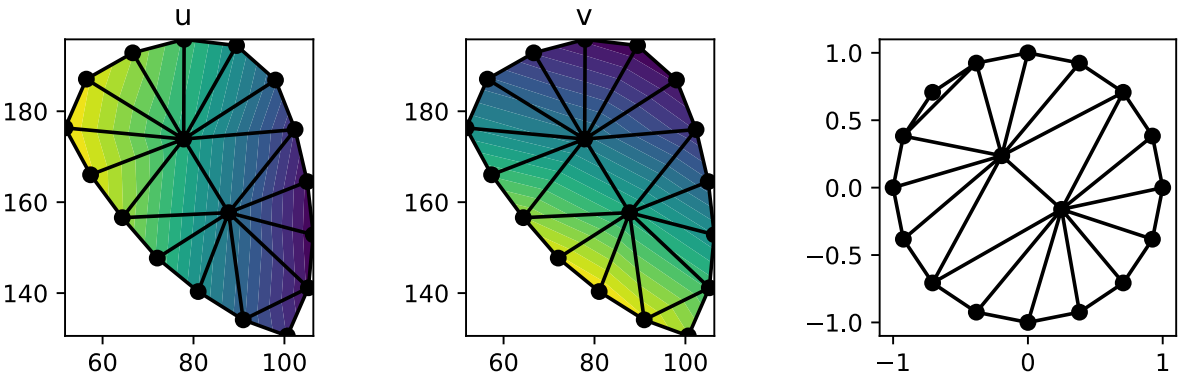


Figure 1.13: Triangulations and harmonic map for a slice S_M of the biceps muscle. The first two plots show the solutions of u and v on the slice S_M. The third plot shows the image in Ω_P of the triangulation in S_M under the harmonic map.

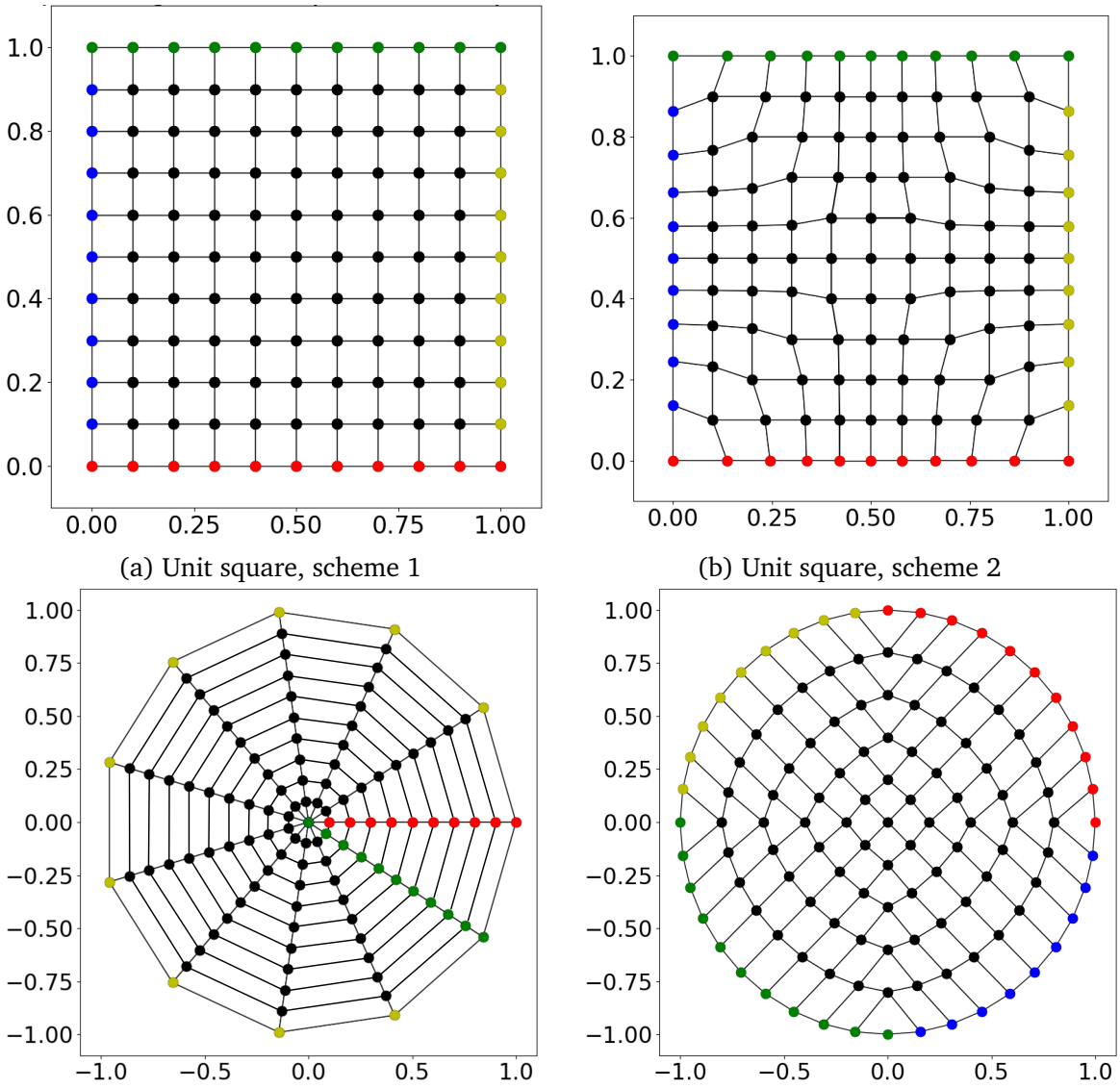
1.4.4 Construction of a Regular Grid in the Parameter Domain

The next step in the Alg. 1 is the construction of a 2D structured, regular grid in the parameter domain Ω_P, as stated in line 1.5. This grid will then be mapped to the slices S_M. Creating a structured grid in a given domain is also called *quadrangulation*.

The parameter domain Ω_P can be selected to be either a unit square or a unit circle. For both choices, two different schemes how to generate a grid with given number of cells can be selected. Figure 1.14 shows all four possiblities.

The first scheme, Fig. 1.14a, uses an equidistant regular grid in a unit square. This is the easiest possibility to generate a quadrangulated reference domain. A possible issue concerns the corners of the square. The grid will be mapped to a cross section of the muscle which has no sharp corners. Therefore, the cells of the grid will be distorted at the corners, usually shortening diagonals that point towards the corners and lengthening the other diagonals. This assumption motivates the second scheme in Fig. 1.14b. Here, the elements are already distorted in the described manner, with increasing distortion closer to the corners. The rationale is that the mapped cells in S_M will then be less distorted.

A different approach is to use a unit circle, which has no corners and therefore might more resemble a muscle cross section. The first scheme of the unit circle is given in Fig. 1.14c. It uses the radial and circumferential directions for the two dimensions of the grid. A disadvantage of this scheme is that the quadrilaterals at the center are degenerated to triangles. Additionally, the area of the cells varies significantly and the outer cells have unequal side lengths. To remedy this problem, the second scheme given in Fig. 1.14d is constructed. When traversing from the outer border towards the center point and considering the circumferential lines of grid points, the circle morphs into a square as



(a) Unit square, scheme 1

(b) Unit square, scheme 2

(c) Unit circle, scheme 1

(d) Unit circle, scheme 2

Figure 1.14: Different quadrangulation schemes of the parameter domain with 11×11 nodes. The borders of the grid are colored for better perceptiblity.

the number of grid points decreases. This approach has the disadvantage that some cells have an inner angle of nearly 180°, especially four elements at the border. Apart from that, all elements have similar sized sides and angles.

Each construction schemes allows to specify the (squared) number of nodes and in consequence the number of cells. The examples in Figures 1.14a, 1.14b, and 1.14d have 11×11 nodes and 10×10 cells. For Fig. 1.14c, the numbers are slightly different. There are $10 \times (11 + 1)$ nodes resulting in 10×11 cells.

Next, the grid in the parameter domain is transferred to the muscle domain by applying the harmonic map $\mathbf{y}(\mathbf{x}) \in S_M$ on every point of the quadrangulation $\mathbf{x} \in \Omega_P$. This is illustrated in Fig. 1.11c for a parameter domain consisting of the unit circle, with quadrangulation scheme 2 and 5×5 nodes. The cells of the grid in Ω_P are shown in the right-most stack of domains. The grid points are visualized left of the grids. The resulting image of the mesh in the slices S_M is shown on the left. For visualization reasons, each quadrilateral has been split into two triangles.

1.4.5 Formation of Three-Dimensional Elements

The result of the previous steps is a number of quadrangulated muscle slices. The grid on every slice has the same number of nodes and elements. The nodes on the border of neighbouring slices are positioned similarly.

The final step of Alg. 1 is line 1.6, the formation of 3D elements. Inserting vertical edges between all corresponding nodes on two neighbouring slices creates a set of 3D elements and, thus, an overall 3D quadrilateral mesh of the muscle volume. This step is visualized in Fig. 1.11d.

1.4.6 Generation of Fiber Meshes

1D fiber meshes are created following the approach of computing a divergence free vector field introduced in [Cho13]. The steps are given in Alg. 2.

The Laplace problem to be solved can be stated as

$$\Delta p(\mathbf{x}) = 0 \quad \text{for } \mathbf{x} \in \Omega_M. \tag{1.8}$$

As proposed by [Cho13], Neumann boundary conditions can be specified for the bottom and top surfaces of the muscle volume, $\partial\Omega_{M,\text{bottom}}$ and $\partial\Omega_{M,\text{top}}$:

$$\begin{aligned} \frac{\mathrm{d}p(\mathbf{x})}{\mathrm{d}\mathbf{x}} \cdot \mathbf{n} &= F_{\text{in}}, \qquad \text{for } \mathbf{x} \in \partial\Omega_{M,\text{bottom}}, \\ \frac{\mathrm{d}p(\mathbf{x})}{\mathrm{d}\mathbf{x}} \cdot \mathbf{n} &= F_{\text{out}}, \qquad \text{for } \mathbf{x} \in \partial\Omega_{M,\text{top}}. \end{aligned} \tag{1.9}$$

The in and outflow values $F_{\text{in}} < 0$ and $F_{\text{out}} > 0$ are balanced such that the total inflow $F_{\text{in}} \cdot \mu(\partial\Omega_{M,\text{bottom}})$ compensates the total outflow $F_{\text{in}} \cdot \mu(\partial\Omega_{M,\text{bottom}})$. Here, $\mu(\partial\Omega)$ is the surface area of the respective boundary.

Alternatively, Dirichlet boundary conditions can be specified:

$$\begin{aligned} p(\mathbf{x}) &= 0, \quad \text{for } \mathbf{x} \in \partial\Omega_{M,\text{bottom}}, \\ p(\mathbf{x}) &= 1, \quad \text{for } \mathbf{x} \in \partial\Omega_{M,\text{top}}. \end{aligned} \tag{1.10}$$

The specification of Dirichlet boundary conditions is easier. Like Neumann boundary conditions, they enforce in and outflows orthogonal to the border.

The boundary value problem given by Equations (1.8) and (1.9) or Equations (1.8) and (1.10) is discretized by the Finite Element method with linear or quadratic ansatz functions and solved by OpenDiHu using the 3D mesh generated from Alg. 1. The divergence free gradient field is visualized in Fig. 1.15a. The gradient values are directly given by the Finite Element discretization. The gradient is elementwise constant for linear ansatz functions and trilinear for quadratic ansatz functions.

The next step is line 2.3 in the algorithm, to trace streamline through the gradient field. Seed points are selected at the vertical center of the muscle domain and horizontally at regular subdivisions of the existing 3D elements. Because the 3D mesh was created using harmonic maps, the spacing between the seed points is very uniform.

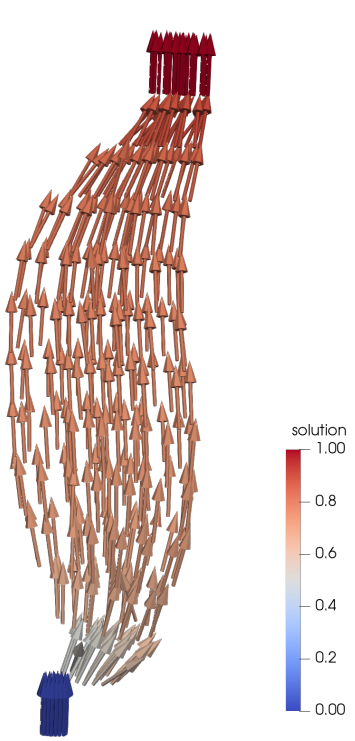
For the tracing of streamlines, the semi-analytical Pollock's method [Pol88] is often used, which was originally developed for fixed 2D finite difference grids. Extensions to irregular 3D grids and for given velocities at nodes instead of fluxes over faces have been formulated [Hæg07]. Other, more accurate algorithms exist [Cor92], including higher order formulations [Jua06].

Because modeling muscle fascicles is only a heuristic approach, the generated streamlines do not have to be exceptionally accurate. Therefore, we use a fully numeric method. The streamlines are generated by explicit Euler integration of the gradient vectors in top and bottom direction. A small spatial step width of $h = 1 \cdot 10^{-2}$ is used. Details of the algorithm are given in the next section, Sec. 1.4.7. In line 2.4, all generated streamlines are resampled to obtain the desired widths of the 1D elements. Figure 1.15b visualizes the resulting streamlines in the biceps muscle.

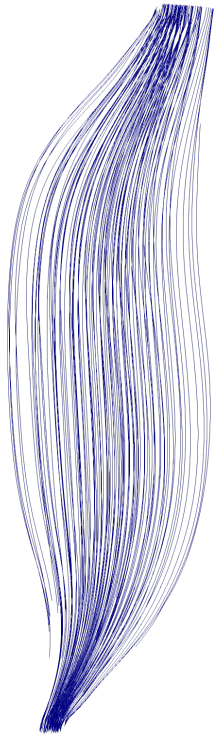
Algorithm 2 Serial algorithm

```
1 procedure Create_1D_meshes
  Input: Structured 3D volume mesh
  Output: 1D fiber meshes

2   Solve Laplacian flow problem
3   Trace streamlines in the gradient field
4   Resample 1D fiber meshes
```



(a) Solution (color coding) and direction vectors of the gradient field for the boundary value problem Eq. (1.8) with Dirichlet boundary conditions Eq. (1.10).



(b) Resulting streamlines that were traced through the gradient field.

Figure 1.15: Setup and solution of the Laplace problem for the biceps geometry that is used to estimate muscle fibers by streamline tracing.

1.4.7 Algorithm for Streamline Tracing

The algorithm for streamline tracing uses an efficient method to traverse the mesh, which makes use of its structuredness. At first, the element $E^{(0)}$ in the mesh that contains the first seed point $\mathbf{p}^0$ needs to be found. By construction of the mesh generation algorithms, this is always the element with the lowest index. However, if the seed points is not found there, the scheme is robust enough to search in all other elements.

Starting from the seed point $\mathbf{p}^0 = \mathbf{p}^{(i)}$ in element $E^{(i)}$, the next point $\mathbf{p}^{(i+1)}$ of a streamline is computed as

$$\mathbf{p}^{(i+1)} = \mathbf{p}^{(i)} + h\,\nabla p(\mathbf{p}^{(i)}).$$

After $\mathbf{p}^{(i+1)}$ has been computed, the mesh element $E^{(i+1)}$ where it is located needs to be identified. It is needed to evaluate the gradient value $\nabla p(\mathbf{p}^{(i+1)})$ at the new point.

At first, the element $E^{(i)}$ of $\mathbf{p}^{(i)}$ is checked whether it also contains $\mathbf{p}^{(i+1)}$. If not, the neighboring element in the direction of the streamline is considered. This neighboring element is chosen among all up to 26 possible neighbors such that the direction from the previous element $E^{(i-1)}$ to the current element $E^{(i)}$ continues.

If this element is also not the right one, all other neighbors of $E^{(i)}$ are subsequently checked, ordered by their plausibility according to the previous streamline direction. If none of the 27 considered elements contains the computed point, a search among all elements of the entire mesh is performed. This case happens only for unsuited choices of the integration width h, i.e., if the streamline tracing skips whole elements.

The end of a streamline is detected when the streamline reaches the final z plane, either at the bottom or top of the muscle volume. To make the algorithm more robust, also the case is considered where the streamline leaves the muscle domain to the side shortly before reaching the end of the muscle. This can happen due to discretization errors for streamlines that start close to the border of the muscle. In such a case, the missing rest of the streamline is interpolated from up to four existing neighboring parallel streamlines.

After the end of the streamline is found, tracing of the next streamline starts at the next seed point $\mathbf{p}^0_{\text{next}}$. For this purpose, the element where $\mathbf{p}^0_{\text{next}}$ is located is required next. At first, this element is searched among all elements with the same z coordinates as the previous seed point. The strategy for this search in a 2D grid of elements is analogous to the previously explained search in the 3D grid of elements for the streamlines: First, the element of the previous seed point is checked. Then, the most plausible neighbor, the

one opposite to the previous element, is checked. With this scheme, new seed points are predicted along the x axis of the 3D mesh.

After a row of elements with seed points in x direction is completed, the next row of elements begins parallel to the previous one. Accordingly, the algorithm knows to switch to the new row after the end in x direction was reached.

The presented scheme avoids traversing all elements of the mesh by predicting the next elements according to streamline direction and organization of seed points. This is facilitated by the structured mesh, which has well-defined element neighbor relations. At the same time, the scheme is robust enough to also efficiently handle streamlines in other use cases. It can also be reused, e.g., for non-vertex free vector fields or in muscle fiber tracing applications of more complex shaped muscle where the fibers change directions.

1.4.8 Results and Discussion

The presented Algorithms 1 and 2 generate a 3D mesh and 1D muscle fibers from a triangulated surface. Three different triangulation strategies for the slices and four different reference quadrangulations can be chosen. In the following, the different choices are evaluated.

The different triangulation methods for the slices that are discussed in Sec. 1.4.2 are visualized in the three columns of Fig. 1.16. The top rows show the triangulation of S_M and the harmonic map u as color coded value from violet to yellow. u is the horizontal coordinate on the reference domain. A point with violet color in S_M will be mapped to a left-most point in the parameter space Ω_P. Similarly, a yellow point will be mapped to a point far at the right in Ω_P.

The middle and bottom rows in Fig. 1.16 show the image of the triangulation in Ω_P under the harmonic map, for the unit square and the unit circle, respectively. The mapping between the colored border points stems from the Dirichlet boundary conditions in the formulation of the harmonic map. In consequence, the border points are by construction equally spaced both in the muscle domain S_M and in the parameter domain Ω_P.

It can be seen that the triangulation appears distorted in the parameter domains. The effect is highest for the square in the columns Fig. 1.16b and Fig. 1.16c. For the latter the center point of the muscle domain S_M gets mapped far off the center of the squared parameter domain. This effect does not occur for the circle.

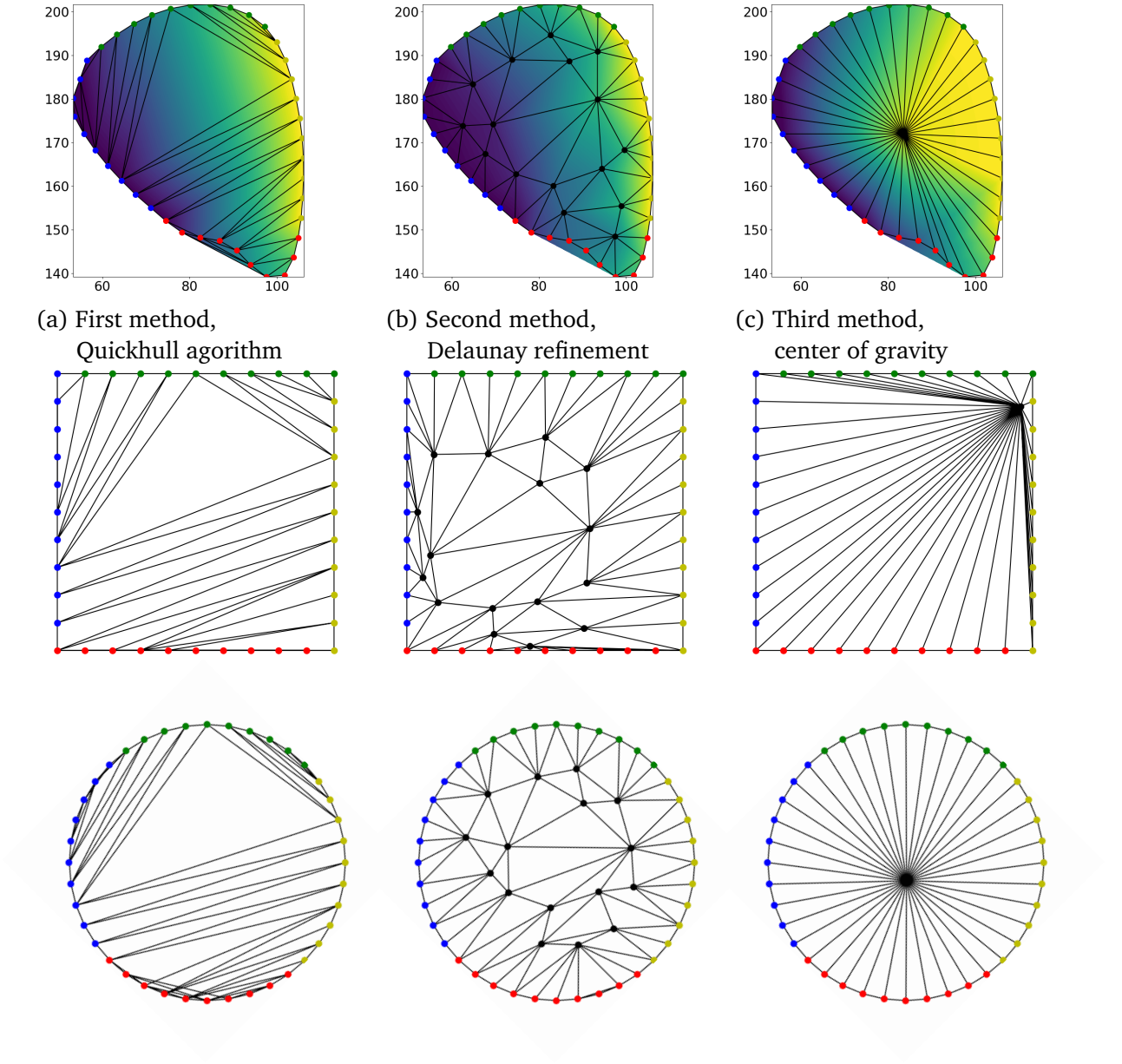
The reason for this lies in the triangulation of S_M together with the border shape. In the third method, only the value at the center point is a degree of freedom in the computation of u while the values at the border points are fixed. By the triangulation, the value of u varies linearly from the border towards the center point. By comparing the colored border points in the muscle slice in the top row with the square in the second row, it can be seen that the prescribed values for u are 0 at all blue points, 1 at all yellow points and linearly increasing from 0 to 1 at the green and red points, increasing from left to right. The yellow points of S_M with the prescribed constant value of $u = 1$ are approximately located on a vertical line. The first two derivatives of u in vertical coordinate direction are therefore almost zero, in consequence, the Laplace equation forces the derivatives in horizontal direction to also be approximately zero. Therefore, the solution value at the center point becomes close to 1. This leads to the mapped center point being close to the right border in the square parameter domain. The same happens for the vertical coordinate v of the harmonic map.

For the circle, neighboring border points are not located on horizontal or vertical lines and, thus, the Dirichlet bondary conditions for u and v vary along the border. Therefore, a better mapping is obtained. The shape of the muscular slice is more similar to the circle than to the square.

Another result that can be seen in Fig. 1.16 is the effect on the harmonic map of the failure of the first method to handle concave slices. As the top left image shows, the first triangulation method produces triangles outside the domain. The triangles are located around the red border points. In the square parameter domain, these triangles are degenerated and lie on the bottom border. In the circle parameter domain where the respective triangles can be seen at the bottom they even intersect other triangles. This yields an invalid triangulation.

The reasons for degenerate triangles in the square parameter domains are not solely the concave muscle slices. Also, the straight sides of the unit square lead to degenerate triangles whenever three border points of the same side form a triangle. In the example in Fig. 1.16 this occurs for the square in column (a). As can be seen in the triangulated slice in the top row, there are three triangles that are entirely made up of blue border points. These triangles get mapped onto the left side of the square parameter domain where they have a vanishing surface area.

The same effect also occurs with the second triangulation method in column (b) where a triangle at the bottom right is comprised of three red border points and therefore gets mapped to the bottom side of the square. Because of the guaranteed minimum



(a) First method, Quickhull agorithm

(b) Second method, Delaunay refinement

(c) Third method, center of gravity

Figure 1.16: Top row: Different triangulation methods for S_M, the color represents the solution u of the harmonic map. Middle and bottom row: triangulation mapped to the parameter domain Ω_P, for the unit square (middle) and the unit circle (bottom). Each column corresponds to one triangulation method.

angle in the second triangulation method this circumstance occurs less often and only for muscle cross-sections where the border makes sharp turns, such as the muscle slice in this example. The third triangulation method is guaranteed to avoid this problem as all triangles are connected with the center point.

In the next step, the algorithm creates a quadrilateral mesh in the parameter domain

and computes its image in the muscle domain using the inverse of the harmonic map. The results are shown in Fig. 1.17 for the three different triangulation methods (columns) and the four different schemes to create the quadrilateral mesh (rows).

In the images in column (c) and rows (d) and (e) it can be seen that the previously observed effect of a bad mapping for squares and the third triangulation method also leads to a mesh in S_M of poor quality. The result for the two square schemes in column (b) is better but still not satisfactory. Good results with the square reference domain are only observed for the first triangulation method in this example.

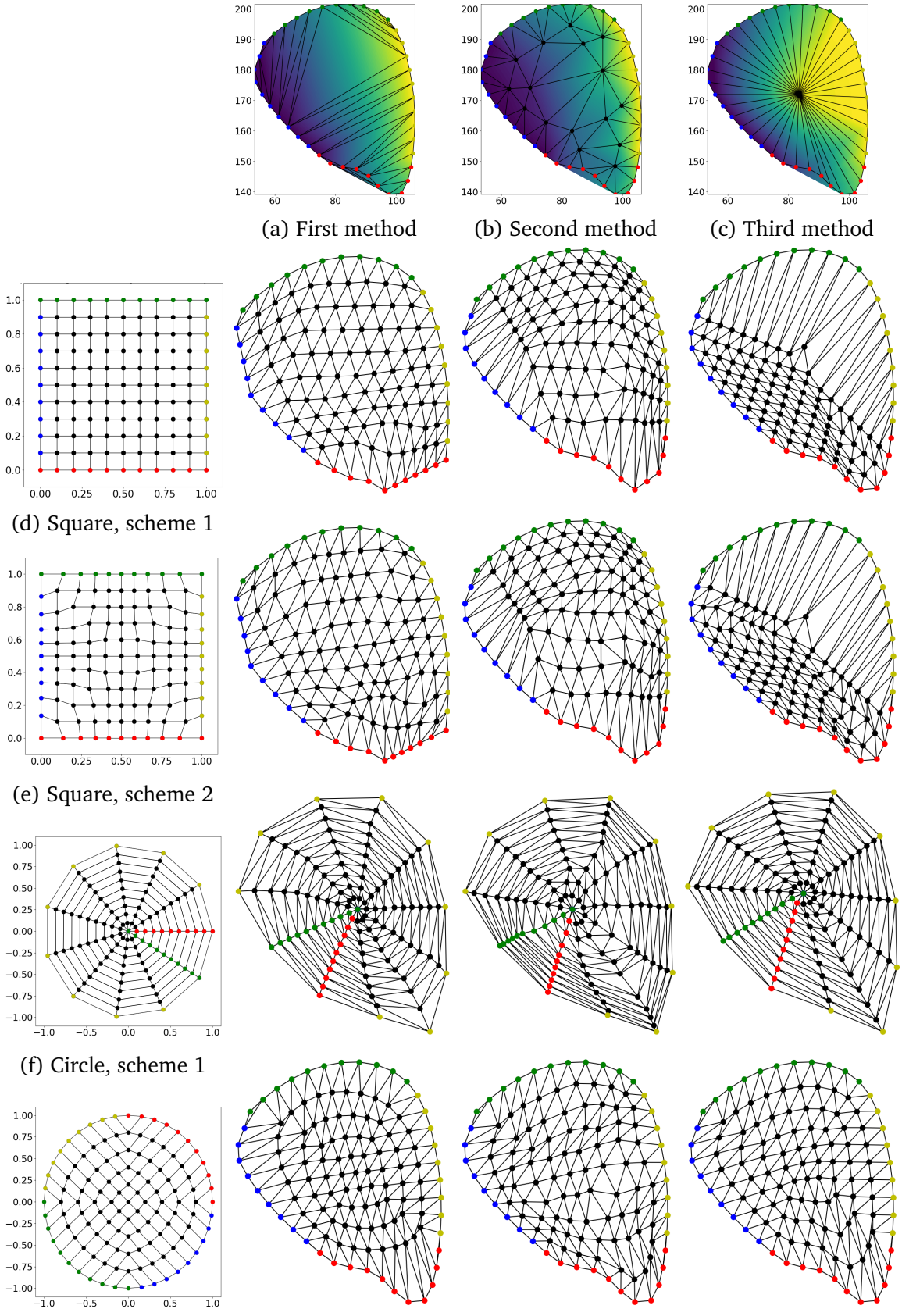
It can be seen that the approximation quality of the border of the domain varies. Most of all, the combination of the first triangulation method (column (a)) and the square parameter domain (rows (d) and (e)) reproduces the shape of the slice poorly. The mismatch occurs at the blue and red border points. Additionally, the second triangulation method (column (b)) for the squares fails to correctly represent the round border at the bottom of the domain. The degenerate triangles in the parameter domain are the cause for this effect. The harmonic map $\mathbf{y} : S_M \to \Omega_P$ is not injective and therefore its inverse does not exist. In our implementation, the points on the degenerate triangles in Ω_P are mapped anywhere inside the corresponding triangles in S_M. Thus, the mapping is correctly inverted at locations of valid triangles and only creates different border points in the invalid areas.

Furthermore, it can be seen that an inaccurate representation of the border also occurs with the parameter mesh in the unit circle generated by scheme 1. In this case, the reason is the low number of elements at the border in the parameter domain quadrangulation.

The two schemes for the circle parameter domain in rows (f) and (g) both generate reasonable meshes for all triangulation methods, despite the different structure of the generated meshes. The best results for both schemes occur with the third triangulation method.

Next, a quantitative comparison of the resulting mesh quality for different parameters of the presented algorithm is carried out. The mesh quality could easily be improved by a smoothing step, e.g. by applying Laplacian smoothing [Fie88]. This technique iteratively improves the local mesh quality. Because of the local scheme, the quality of the final result depends on the quality of the initial mesh. Therefore, the following study forgoes additional smoothing steps and evaluates the direct outcome of the meshing algorithm.

The algorithm was executed for all variants with 43 slices of the biceps muscle, resulting in 43 meshes for every combination of triangulation method and quadrangulation scheme.



(a) First method (b) Second method (c) Third method

(d) Square, scheme 1

(e) Square, scheme 2

(f) Circle, scheme 1

(g) Circle, scheme 2

Figure 1.17: Meshes in S_M for quadrangulations (rows) and triangulations (columns).

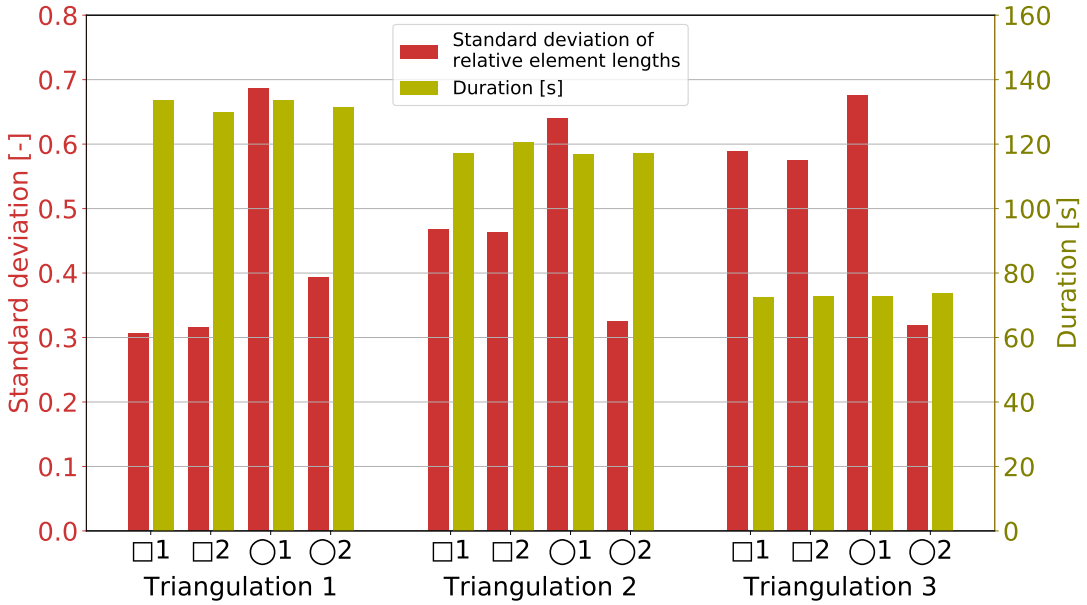


Figure 1.18: Mesh quality (red, lower is better) and duration (yellow) for the three different triangulation methods and the different parameter space quadrangulation schemes. A low value for the standard deviation of relative element lengths indicates good quality.

In each generated mesh the edge lengths of the elements were collected and normalized to have a mean of 1. The normalization was done to allow a comparison between meshes with different bounding box sizes. The standard deviation of the normalized lengths was determined in each mesh. The total mean of all the standard deviations was computed. This value is a measure for the quality of the mesh. A low value means that in every slice the generated mesh has similar edge length and, in consequence, the overall mesh has good quality.

Figure 1.18 visualizes the results. Three groups of bars are displayed for the three triangulation methods. For every type of mesh in the parameter space, i.e. unit square (□) or unit circle (○) and scheme 1 or 2, the standard deviation is given by the red bar and the duration of the overall algorithm is given by the yellow bar. The corresponding axis labels for standard deviation and duration are given on the left and right of the diagram.

The diagram shows the lowest standard deviation of edges and therefore the best mesh quality for the first triangulation method and the square (□1 and □2), with scheme 1 having a slightly better value than scheme 2. This shows that the modified placement of the nodes in scheme 2 has no positive effect compared to scheme 1. However, from the

observations in Fig. 1.16 it is known that the borders are not represented correctly. This behaviour does not influence the result because of the chosen metric of uniform relative edge lengths. Similar good results can be seen for scheme 2 in the circular parameter domain and the second and third triangulations.

Moreover, it can be seen that certain connections between parameter domain and suited triangulation scheme exist. The square parameter domain works best with the first triangulation method. The second scheme for the parameter mesh on the unit circle works best with the triangulation methods 2 and 3.

The first scheme for the parameter mesh on the unit circle (○1) shows bad results for all triangulation methods. This can be explained by looking at the generated meshes in row (f) of Fig. 1.17. By construction, the elements have a bad aspect ratio. This results in the high standard deviation values. However, the generated meshes still look uniform to a certain extent and can be useful in applications where such type of mesh is needed. The score could be improved by adding more nodes in circumferential direction.

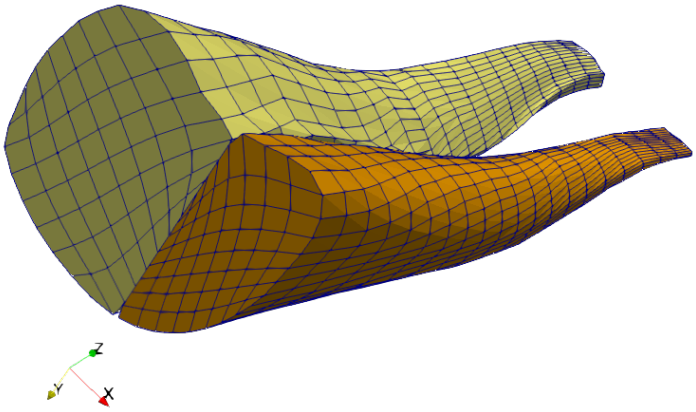
The runtime of the algorithm is approximately the same for the different parameter domain meshes. It mainly depends on the triangulation of the slices. The first triangulation using the *SciPy* package takes the most time, followed by the Delaunay refinement. The fastest triangulation is the custom one where only one additional point needs to be placed. In conclusion, when runtime is an issue, the third triangulation should be chosen. It achieves good quality meshes only with the second scheme of the circular parameter domain. This combination also does not suffer from the bad approximation quality of the border, as is the case for the unit circle with the first triangulation method.

Figure 1.19 shows three structured meshes $\Omega_{T,i}$ for the tendons of the biceps brachii muscle that were created using Alg. 1. The tendon at the bottom of the muscle are represented by a single mesh. At the top there are two tendons that extend the two muscle heads of the biceps. Because the meshes need to be structured two tendon meshes are created at the top. It can be seen that the algorithm creates meshes with similar sized elements despite the difficult, wound geometry of the surfaces.

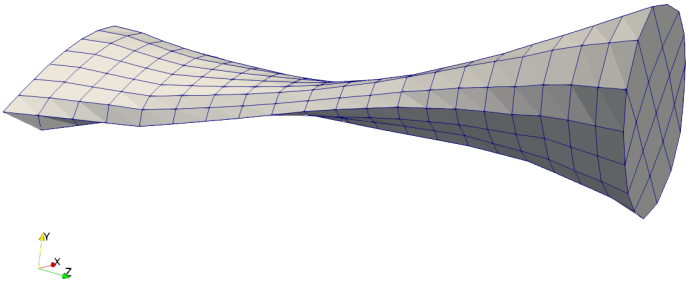
How To Reproduce

The described algorithms are part of the `fiber_tracing` examples. Execute the following commands to get the results in this chapter:

```
cd $OPENDIHU_HOME/examples/fiber_tracing/streamline_tracer/scripts
. run_evaluation.sh
```



(a) Two meshes for the top tendons with $9 \times 9 \times 21 = 1701$ nodes each.



(b) One mesh for the bottom tendon with $5 \times 5 \times 25 = 525$ nodes.

(c) The tendon meshes in the volume of the whole biceps muscle.

Figure 1.19: Tendon meshes that were created using the serial algorithm for mesh creation.

Then, the visualizations will be created under `../processed_meshes`. Create Fig. 1.18 with `plot_mesh_quality.py`. How to create the tendon meshes is explained at the end of Sec. 1.5.7.

1.5 Parallel Algorithm to Create Muscle and Fiber Meshes

The previously presented algorithm to create 3D and 1D meshes is not parallelized. This restricts the resources that can be employed during the execution of the algorithm to those accessible by one hardware core. Thus, the size of the handled meshes is limited by the available memory of the computer. An algorithm that can be used with shared memory

parallelization could, in contrast, benefit from more total memory that is accessible at different cores. Furthermore, the tracing of the streamlines could be performed in parallel which has the potential to reduce runtimes.

In the following, we present an extended algorithm based on the one presented in Sec. 1.4 that can be run in parallel on multiple cores. The extended algorithm employs a partitioning of the 3D volume. Every process only stores data corresponding to its own partition. This allows to run the algorithm on a shared memory system, where data transfer between the processes occurs by sending messages using the Message Passing Interface (MPI). It is possible to create meshes with larger sizes than could fit into a single processes' memory. This enables the algorithm for meshes with very high resolution that can be used for simulations in the field of High Performance Computing. These meshes are partitioned into subdomains for every compute core and can be read from and written to disk concurrently.

1.5.1 Overview of the Parallel Algorithm to Create Muscle and Fiber Meshes

The steps of the algorithm and its input and output are given in Alg. 3. Input and output are the same as for the Alg. 1 presented in Sec. 1.4. The input is a triangulated tubular surface of the muscle that can be obtained as described in Sec. 1.3. A second input, the variable called *border_points*, is used only during recursive calls and is not set at the beginning. The output consists of the 3D mesh of the muscle volume Ω_M and embedded 1D fiber meshes $\Omega_{F,i}$.

During execution of the algorithm, the 3D mesh of the muscle is recreated iteratively with increasing resolution and increasing number of subdomains. The algorithm is formulated recursively. At first, a single process executes all the steps listed in lines 3.2 to 3.11. This corresponds to recursion level $\ell = 0$. Then, in line 3.12 the procedure is called again and in the first recursion executed by eight processes with eight subdomains. On the ℓth recursion level, the number of involved processes and subdomains is 8^ℓ. After a specified maximum recursion depth $\ell_{\max}$ is reached, all involved processes execute the first branch of the **`if`** statement in line 3.8 and generate the final output meshes in 3.9.

In more detail, the goal during the recursive calls is to determine smooth borders for the subdomains. At the end of the algorithm, the constructed partitioning is used to create the final 3D mesh. On each recursion level, the subdomain borders are determined by tracing streamlines through the entire muscle volume, similar to the approach in Alg. 2.

For this purpose, a new 3D mesh is created on each level and a global Laplace problem is solved on the mesh. The number of elements in this mesh increases by the factor of eight on each level. All subdomain borders are determined anew on each recursion level. This means that the subdomain borders change slightly as the mesh refines.

As the subdomain borders in the interior of the volume refine, so do the outer borders given by the surface of the muscle. The given triangulation of the surface is sampled again on each recursion level yielding increasingly fine representations.

The steps in Alg. 3 are executed concurrently by the involved processes at the respective level. Some of the steps only operate on the locally stored data and, thus, are independent of other processes. Other steps involve communication between processes. Whether an instruction effects only the own domain of the process or involves global communication is denoted in parantheses at the beginning of the lines in Alg. 3.

Algorithm 3 Parallel algorithm to create muscle and fiber meshes

```
1  procedure Create_3D_meshes_parallel
   Input: Triangulated tubular surface
   Input: border_points:  4×4 points per slice
   Output: Structured 3D volume mesh
   Output: 1D fiber meshes

2    (own domain) Create_3D_mesh(border_points)
3    (own domain) Fix and smooth 2D meshes
4    (global)     Solve Laplace problem
5    (global)     Communicate ghost elements to neighboring subdomains
6    (own domain) Trace streamlines for new subdomain boundaries
7    (global)     new_border_points ← Construct new subdomains

8    if recursion ends then
9      (own domain) Trace streamlines for fiber meshes
10   else
11     (global)     communicate border points
12     (global)     Create_3D_meshes_parallel(new_border_points)
```

1.5.2 Generation of the 3D Mesh

Next, the steps of the algorithm are discussed in detail. Algorithm 3 starts its execution with one process and a triangulation of the tubular muscle surface being the only input. The following procedure is described in this section with a focus on the serial execution in this first call.

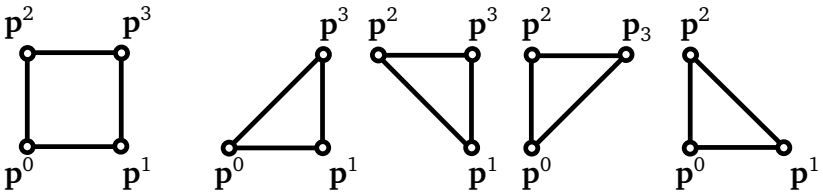


Figure 1.20: A quadrilateral (left) and the four triangles (right) that can be constructed from its four points. These triangles are needed for the check in Alg. 3 if the quadrilateral is valid.

The first step in line 3.2 is to execute Alg. 1 to generate a 3D mesh of the volume. The second triangulation method is used with a circular reference domain quandrangulated by the second scheme as explained in Sec. 1.4.

Next, line 3.3 improves the mesh quality of the 2D muscle slices S_M from which the 3D mesh is formed. This action consists of two steps. The first step is to ensure that no self-intersecting or degenerate quadrilaterals exist in the slice. The second step applies Laplacian smoothing to improve the mesh quality of the slice.

Theoretically, the first step should not be necessary, as the chosen quadrangulation algorithm always produces valid elements. However, in practice, small or irregularly shaped, concave domains occur and together with rounding and numerical errors in the Laplace problem computations lead to invalid meshes with self intersecting elements, especially for higher recursion depths in Alg. 3. Executing the first step therefore increases the robustness of the implementation.

The algorithm performs this step by repeatedly iterating over all interior mesh points in every slice S_M and fixing invalid elements. To find invalid elements, for every quadrilateral the four triangles that can be formed from the points of the quadrilateral are considered, as shown in Fig. 1.20. For every triangle with points $\mathbf{p}^0, \mathbf{p}^1$ and $\mathbf{p}^2$, the orientation of the triangle is determined. The orientation is counterclockwise if the oriented triangle area A_{012} is positive. The oriented triangle area is the determinant of the 3×3 matrix that contains the row vectors $(p_x^i, p_y^i, 1)$ for the triangle points $\mathbf{p}^i = (p_x^i, p_y^i)^\top$ and can be computed by the following formula [Sed11]:

$$A_{012} = (p_x^1 - p_x^0)(p_y^2 - p_y^0) - (p_x^2 - p_x^0)(p_y^1 - p_y^0).$$

If the orientation is counterclockwise, a score value of the triangle is set to one, if it is clockwise, the score is set to zero. The score values of the four triangles are added up to yield a score s for the quadrilateral. Only if this score is $s \geq 3$, the quadrilateral is considered valid. For $s = 3$ the quadrilateral is concave, for $s = 4$ it is convex.

At the current mesh point in the loop over all points that are not at the border of the mesh, the four adjacent quadrilaterals are considered. If any of them is invalid, the algorithm tries to improve the situation by deflecting the point by a random, small vector. A maximum of 200 random deflections from the original position with exponentially increasing deflection vector sizes are tried. After each modification of the point the scores of the four adjacent quadrilateral elements are evaluated. If the sum of the four element scores increases, the point is kept and the iteration over all interior mesh points starts anew.

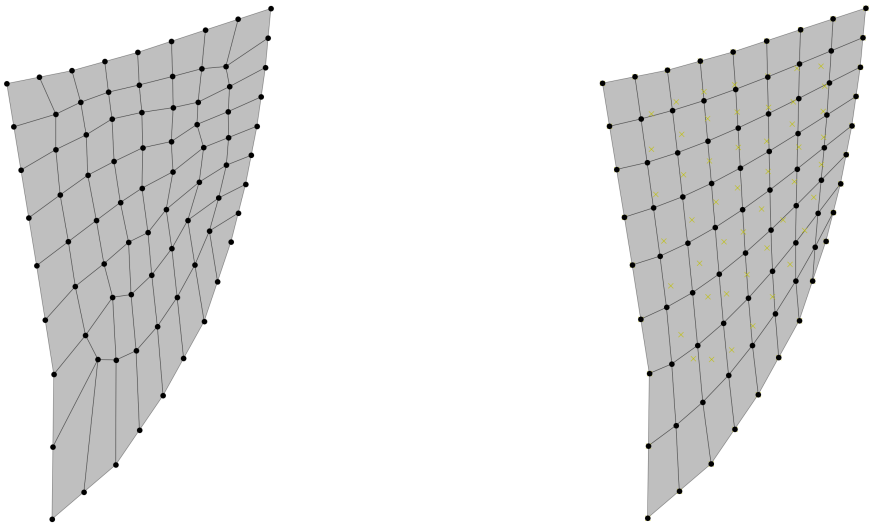
Note that this does not necessarily mean that the invalid element was fixed, only its score was improved. If it was not fixed, it will be considered again in the next iteration. For example, a convex element that initially is oriented clockwise instead of counterclockwise has a score of $s = 0$. In the first iteration, one point is deflected such that the quadrilateral intersects itself but has a higher score $s \geq 0$. At least one more iteration is needed until the quadrilateral is oriented correctly. When all elements in the slice S_M are valid, this step is complete.

The second step is the smoothing step that improves the mesh quality of the 2D slices. 20 iterations of Laplacian smoothing [Fie88] are executed. Laplacian smoothing in our case subsequently visits all points of the mesh and sets the location of a point to the center of gravity of its eight neighbours. Figure 1.21 shows the effect of Laplacian smoothing for a slice in a subdomain on the first recursion level. It can be seen how the smoothing equalizes the element lengths and angles.

1.5.3 Computation of Subdomains

After the 3D mesh has been created and smoothened, the next steps construct the eight subdomains for the recursive calls by subdividing the own domain. At the beginning of the procedure in Alg. 3 the own domain on each process is specified by their circumferential border. More specifically, for each slice 4×4 border points are given. Figure 1.22a visualizes the border points of a subdomain by the red boundary. The visualization uses a squared grid whereas in reality the domain has the shape given by (part of) the muscle cross sections. Note that a full grid with this border would contain $5 \times 5 = 25$ grid points whereas the number of considered border points is $4 \times 4 = 16$.

The task in the procedure is now to determine borders for eight subdomains. This is achieved by subdividing the given 2D domain into four subdomains and additionally



(a) Initial 2D mesh of a subdomain at the border of the biceps muscle.

(b) The mesh of (a) after 20 iterations of Laplacian smoothing.

Figure 1.21: Effect of Laplacian smoothing of a 2D grid which occurs in line 3.3 of Alg. 3.

(a) Grid of 4×4 border points, which occurs at the beginning of the procedure of Alg. 3.

(b) Grid with 4×8 border points, which occurs after a refinement step at beginning of the procedure of Alg. 3.

Figure 1.22: Logical grid of a subdomain (gray) specified by the border points (red) during execution of Alg. 3.

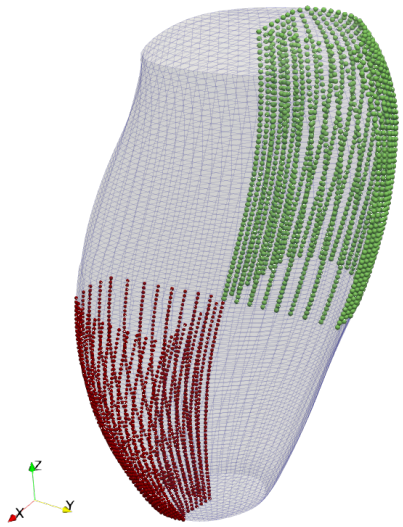
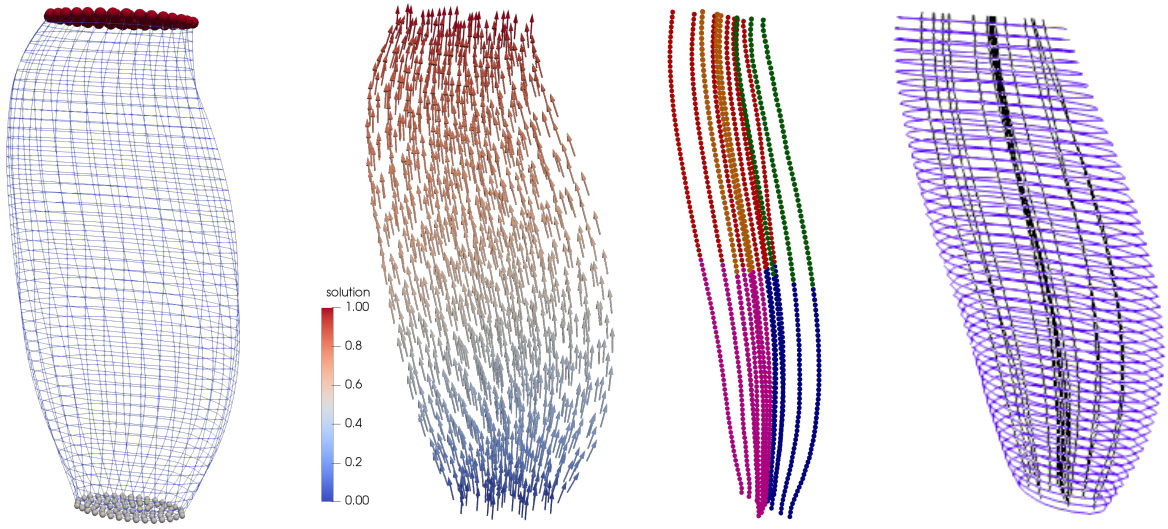


Figure 1.23: Partitioning of the muscle volume into eight subdomains during the first call to the procedure in Alg. 3. The first (red) and the eighth subdomain (green) are shown.

splitting the muscle volume at the center in vertical direction. A visualization of the resulting border points for the first and the eighth subdomain is given in Fig. 1.23.

For this reason, the given 4×4 border points are refined to twice the amount of border points by inserting new points at the centers between neighboring points. The resulting grid is shown in Fig. 1.22b. Now, it would be possible to subdivide the grid to obtain four instances of the needed grid in Fig. 1.22a. However, this would result in constant straight connection lines between the initial border points. In all further recursive calls, the additional points would all be placed on these lines and thereby not properly refine the subdomain borders. Instead, a different approach is desired where the subdomain's borders in the volume follow the directions of streamlines and fibers. Thus, the approach is to define the subdomain borders in the interior of the global domain by traced streamlines and sample the outer borders from the surface triangulation with the desired mesh width. The required steps of this approach are discussed next.

Algorithm 3 proceeds with the next step in line 3.4, which is solving the Laplace problem. The same step also occurs in Alg. 2 and is explained in Sec. 1.4.6. Dirichlet boundary conditions of $p(\mathbf{x}) = 0$ and $p(\mathbf{x}) = 1$ are prescribed at the bottom and top of the domain, as shown by the spheres in Fig. 1.24a. Alternatively, Neumann boundary conditions can be used. After the solution $p(\mathbf{x})$ is obtained, the gradient field $\nabla p(\mathbf{x})$ is computed. The solution and the gradient directions are visualized in Fig. 1.24b.



(a) Location of Dirichlet boundary condition nodes at the bottom and top.

(b) Solution p of the Laplace problem and direction of gradient ∇p.

(c) Traced streamlines that split the domain into eight subdomains.

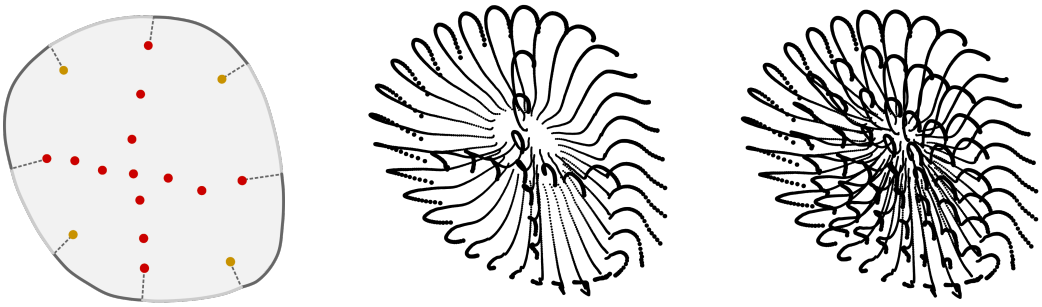
(d) Rings of the slices S_M and traced streamlines in the interior.

Figure 1.24: Process of subdividing the muscle volume into eight subdomains, which is an important step in the procedure of Alg. 3.

The next step, line 3.5, is to communicate ghost elements to neighboring subdomains. This step is only meaningful in later recursions and requires no operation in the first, serial call to the algorithm because the domain is not yet partitioned into subdomains.

Next, the algorithm traces streamlines through the gradient field in line 3.6. This step is again similar to the analogous step in Alg. 2. The same method of explicit Euler integration is used. The seed points are again chosen in the cross-sectional plane at the vertical center of the muscle. From there, streamlines are traced in both directions towards the ends of the muscle following the positive and negative gradient directions. In contrast to Alg. 2, the horizontal locations of the seed points of the streamlines are chosen differently, because the purpose of the streamlines at this stage is not to create fiber meshes as in Alg. 2. Instead, the streamlines in Alg. 3 are needed to determine the subdomain borders to partition the domain into eight subdomains. This happens in line 3.7.

The seed points are selected from the set of nodes in the structured mesh of the 2D slice. The points selected during the first call to the procedure where the domain is not yet partitioned are shown by the red and yellow points in Fig. 1.25a. As can be seen, the seed points consist of the nodes of the mesh at the horizontal and vertical centers,



(a) The seed points used to determine the subdomain borders.

(b) Streamlines and lines on the muscle surface that define the new subdomain borders, view from the top of the muscle.

(c) All streamlines and lines on the muscle surface that are created if the algorithm is run with one process.

Figure 1.25: Seed points and streamlines that occur during the first call to the procedure in Alg. 2.

which form a *plus sign* shape. They are visualized by red points in Fig. 1.25a. In addition, there are four points near the corners of the structured mesh, given by the yellow points. The border of the slice is also visualized together with a method of splitting it into eight sectors by choosing the splitting points such that they are the closest to the given outer seed points.

The streamlines for the seed points of the plus sign are depicted in Fig. 1.24c. As shown by the different colors, these streamlines constitute the interior border points for eight subdomains that partition the muscle volume. Figure 1.24d shows all streamlines of this first set in black and the circumferential rings of the muscle slices in blue that were extracted during the call to Alg. 1 in line 3.2.

For the new partitioning, the border points at the outer surface of the muscle are yet to be determined. For this reason, every circumferential ring needs to be split into four quarter parts for the four adjacent subdomains. For each of these new subdomains, this part corresponds to two neighboring sides of the subdomain border given in Fig. 1.22a. Therefore, a splitting point is needed that further splits every quarter part of the circumferential ring into the two sides for the new subdomain. In other words, the ring needs to split into eight parts that fit to the inner subdomain borders. The eight split points are determined by the eight outer streamline points. As seen in Fig. 1.25a, the four outer red points of the plus sign and the four yellow points are considered. For each, the nearest point on the circumferential ring is determined. The algorithm for calculating the coordinates of the point on a ring that has the shortest distance to a given, second point is described in Sec. 1.4.1.

After the two sections of the circumferential rings are determined for all new subdomains, the sections are equistantly sampled in circumferential direction to create the outer border points for the subdomains. Also in longitudinal direction of the muscle, i.e., the z axis, points are sampled on each streamlines to yield a predefined number of points. As an example, we set the number to 51. The resulting border points are shown in Fig. 1.25b. In result, one pass of the procedure in Alg. 3 has created eight subdomains, that each are defined by $4 \times 4 \times 51$ border points. In the algorithm, these points are stored in the variable *border_points* that is accessed in the next recursion.

If the current recursion level ℓ equals the predefined maximum level ℓ_{max}, the first branch of the **if** statement in 3.8 is taken. Then, the final 3D mesh and 1D fiber meshes are constructed. In this case, the prepared border points are not needed for a further subdivision of the domain. Instead, they are used to construct the final mesh. In line 3.9, additional streamlines are traced from the remaining grid points beginning at the slice at the vertical center of the muscle. Figure 1.25c shows the result of this action for $\ell_{\text{max}} = 1$.

Next, all streamlines are sampled at equidistant positions with a prescribed distance. The set of points forms the resulting 3D mesh of the domain Ω_M. At the same time, the values can be interpreted as 1D fiber meshes $\Omega_{F,i}$, one for each traced streamline.

In the case $\ell < \ell_{\text{max}}$ the second branch of the **if** statement in 3.8 is chosen and the next recursion level $\ell_{\text{new}} = \ell + 1$ begins. Execution continues with the eight times higher number of processes $8^{\ell_{\text{new}}}$. In line 3.11, each process communicates its computed border points for the subdomains to the seven appropriate processes. Only the first subdomain remains on the same process. Next, in line 3.12 all now involved processes call the procedure and continue the algorithm on their own subdomains.

1.5.4 Procedure on Higher Recursion Levels

In the following, the procedure of Alg. 3 is illustrated in more detail for higher recursion levels, where multiple processes concurrently work on their own subdomain. The input consists of the given $4 \times 4 \times 51$ border points for the local subdomain of the process, which, as mentioned, get initially refined to $8 \times 8 \times 51$ border points. Figure 1.23 shows the refined border points for the subdomain of the first process in recursion level $\ell = 1$ in red. This subdomain will be considered in the following examples.

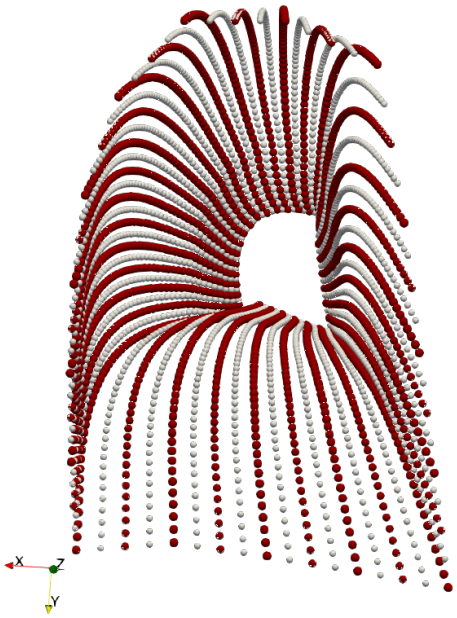
To obtain a sufficiently fine 3D mesh, the 2D mesh gets refined further by increasing the number of elements per coordinate direction by a factor of $r \in \mathbb{N}$. For example, $r = 2$ refines the grid to 16×16 border points on each slice. This is shown in Fig. 1.26a in a view in negative z direction. The red points are the border points of the 9×9 grid, the additional white points are added in between. At the lower left of the figure, it can be seen that always five neighboring points lie on a straight line since the initial border points were refined two times by taking center points between existing points. The border points on the curved outer boundary were instead sampled from the surface triangulation and do not show this behavior.

In line 3.2 of Alg. 3, the harmonic mesh algorithm to construct a 3D mesh, Alg. 1, is called. Its input consists of the points shown in Fig. 1.26a, which define the 2D slices of the volume. This means that the Alg. 1 does not need to construct the slice border rings from the surface triangulation, instead, the formulation of Alg. 1 can directly start with line 1.3 to triangulate the slices and then compute the harmonic map.

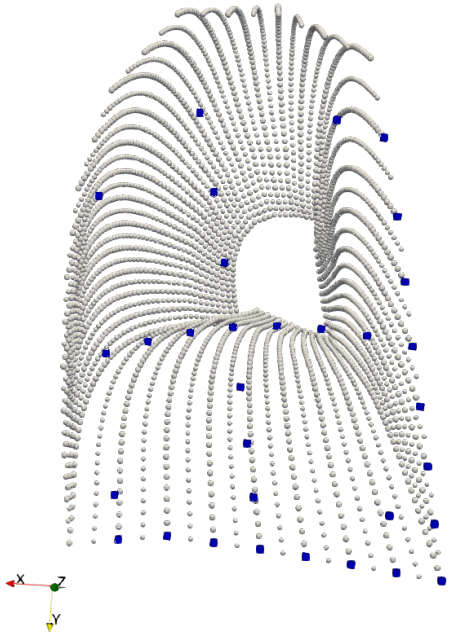
In line 3.4 of Alg. 3, the Laplace problem gets solved. The equation is formulated globally and the discretization uses the existing partitioning. The system matrix is indefinite if Dirichlet boundary conditions are used. A parallel GMRES solver obtains the solution in a small number of iterations, e.g., for the biceps muscle that yields a linear system with 4131 degrees of freedom, 26 iterations are needed to obtain a residual norm of below $1 \cdot 10^{-4}$.

Next, the seed points for the streamlines are determined. Figure 1.26b shows the seed points in blue. In addition to the plus sign shape and the four outer seed points, two lines of points in approximate x and y directions are selected at the lower and right edges of the figure. These are needed to determine the outer borders of the new subdomains that lie in the interior of the muscle. Thus, the existing subdomain borders inside the muscle are recreated using streamlines. The streamlines are traced in the gradient field of the solution of the Laplace problem that is solved on the finer mesh. In general, such lines of seed points are required on any border of the subdomain, whenever the border is in the interior of the muscle.

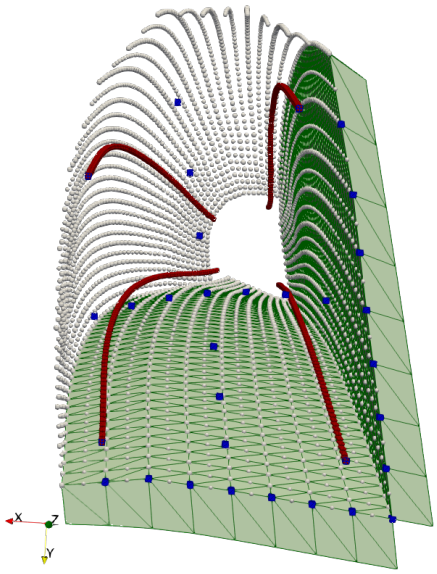
Tracing those interior border streamlines using only the locally stored data of the subdomain usually fails. These streamlines start exactly on the border of the subdomain and potentially leave the local subdomain by a small distance because to the solution on the finer mesh is slightly different than the solution on the previous, coarser mesh. To account for this effect, the solution data from the neighboring 3D mesh elements is needed at those locations. For this reason, the algorithm communicates one layer of *ghost elements*



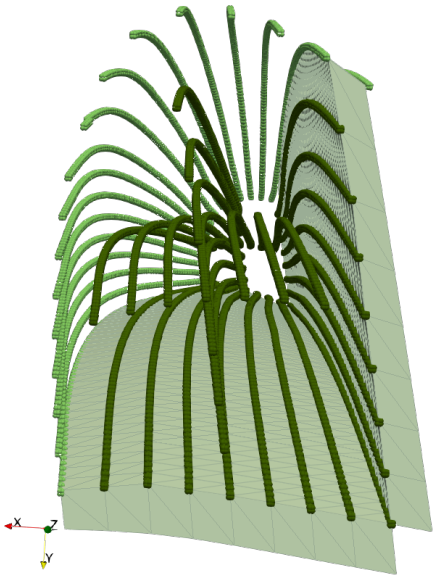
(a) Refined border points.



(b) Seed points for the streamlines.



(c) The four border streamlines (red) and the green layers of ghost elements at the bottom and right of the subdomain.



(d) New border points at the outer (light green) and interior border (dark green).

Figure 1.26: Streamlines in the first subdomain for recursion level $\ell = 1$ of the parallel algorithm for mesh generation.

between neighboring processes such that every subdomain knows the adjacent elements at the interior borders. This occurs in line 3.5 of the algorithm and involves communicating the node positions and the values of the gradient field between the respective processes. Figure 1.26c shows the received ghost elements on the considered subdomain in green.

The next step is to trace the streamlines in line 3.6 of the algorithm. Since the streamlines traverse the entire muscle from the center to bottom and top, this step involves multiple processes. All processes are numbered in z direction from bottom to top by an index $i_z \in \{0, 1, \ldots, n_z - 1\}$ where the number of processes in z direction is $n_z = 2^\ell$.

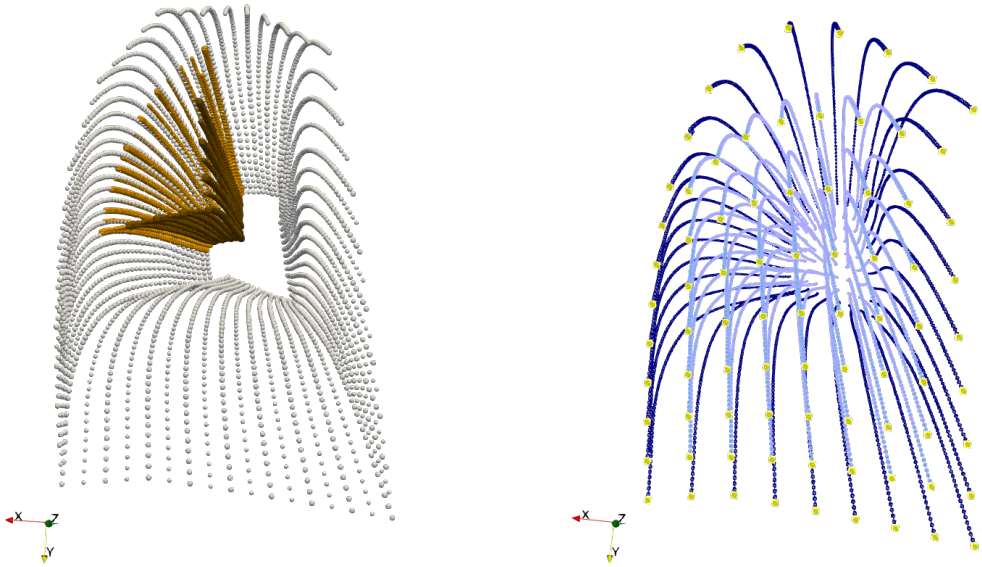
The initial seed points are determined on the processes with index $\lfloor n_z/2 \rfloor$. They are communicated to the processes below with index $\lfloor n_z/2 \rfloor - 1$. Then, these two groups of processes trace the streamlines starting from the same seed points through their subdomains, the upper processes in upward and the lower processes in downward direction. Then, the end points of the traced fibers are communicated to the next processes which continue the tracing of the streamlines. The procedure repeats with further processes until the streamlines reach the bottom and top ends of the overall muscle domain. The time complexity of this approach is $\mathcal{O}(n_z) = \mathcal{O}(\sqrt[3]{n_{\text{proc}}})$ with the number n_{proc} of processes.

The tracing algorithm in the subdomains uses the efficient scheme of predicting subsequently traversed elements described in Sec. 1.4.7. The implementation is adjusted in a way to also take into account the ghost elements.

The red streamlines in Fig. 1.26c are the ones to split the border sides at the outer boundary of the global domain. For simplicity, the algorithm always computes the four streamlines in every corner of the mesh. In the considered example, only the streamline in the upper left corner is needed to sample the border points. Figure 1.26d shows the sampled border points at the surface in light green color. The two sides of the own domain will be split into four sides for the new subdomains, therefore the surface gets sampled at $4 \times 4 = 16$ lines. A comparison with the white lines in Fig. 1.26c shows that the newly sampled points are different from the initially sampled points.

Figure 1.26d visualizes the traced streamlines in the interior with dark green color. They are used as border points for the eight new subdomains. The border of the first of the new subdomains on level $\ell = 2$ is shown in Fig. 1.27a. It consists of the outer border (dark yellow lines) and the interior border (brown streamlines) and is nearly geometrically similar to the subdomain on level $\ell = 1$.

If the recursion ends at level $\ell = 1$, the algorithm creates 3D and 1D meshes instead of new subdomains. Then, streamlines are traced for all grid points of the current sub-



(a) Border points of the first subdomain on level $\ell = 2$ (dark yellow and brown) embedded in the border (white) of level $\ell = 1$, generated if $\ell_{\max} > 1$.

(b) Seed points (yellow), traced interior streamlines (light blue) and border points (dark blue), generated if $\ell_{\max} = 1$.

Figure 1.27: Results of the procedure in the parallel algorithm for mesh generation.

domain mesh. Figure 1.27b visualizes the seed points and the traced streamlines in this case. In the parallel setting, this action is organized analogous to the previous tracing of streamlines: The tracing starts at the vertical center of the muscle and subsequently propagates through all layers of subdomains in z direction.

At the end, the data is written collectively by all processes into a single file. This is done using the parallel file I/O functionality of MPI. It is possible because the absolute position in the file of every point can be calculated from the index of the point in the structured mesh.

1.5.5 Repair of Incomplete Streamlines

Practical tests have shown that for irregular muscle geometries some of the streamlines generated in lines 3.6 and 3.9 of the algorithm are incomplete. This means that it was not possible to obtain a streamline that runs through the entire subdomain or the entire muscle domain from top to bottom, instead points are missing for some ranges of z values. This happens when the streamlines run out of the subdomain.

To fix these cases, four different repair mechanisms are introduced that interpolate the missing data from valid streamlines. Figure 1.28 visualizes the cases by examples in a setting of four subdomains with grids of 5×5 fibers each. The repair mechanisms #1 to #3 only apply to border points. They are executed in line 3.6 of the algorithm after the local portions of the streamlines have been traced and before the end points of the streamlines are sent to the neighbor processes below and above that continue the streamline tracing. Mechanism #4 repairs invalid streamlines in the final result and is executed during line 3.9.

Mechanism #1 checks all streamlines at borders in the interior, which are shared between neighboring processes. If a streamline is incomplete on one process but complete on the neighbor process, the complete data is transferred such that both processes have the same valid points for this streamline. In the example in Fig. 1.28, the valid streamline data is sent from the top left to the top right subdomain.

Mechanism #2 checks streamlines at the outer corners of the subdomains. Incomplete streamlines at these locations are recreated from the given border points. Because the set of border points is twice as coarse as the required number of sampled points at these streamlines, every second point gets interpolated from the top and bottom neighbor points.

Mchanism #3 is concerned with streamlines at interior borders that could not be fixed by mechanism #1 because the streamlines are incomplete on both sharing processes. In this case, the streamlines are interpolated from the two complete neighboring streamlines that are located next along the border as shown in the example in Fig. 1.28. Instead of the factors $\frac{1}{3}$ and $2\frac{2}{3}$, the actual relation of distances between the seed points of the streamlines is used. The same interpolation is executed independently on both involved processes. Because the valid streamlines have the same data on both subdomains the resulting fixed streamlines will also be identical.

Mchanism #4 follows a similar approach. It is applied to the interior fibers of the final result and can repair any number of incomplete fibers that are located between complete fibers. In the example #4a in Fig. 1.28, the two invalid streamlines are interpolated from their left and right valid neighbors. In the example #4b, no valid right neighbor exists. Instead, the streamlines are interpolated by using valid positions from the upper and lower neighbors.

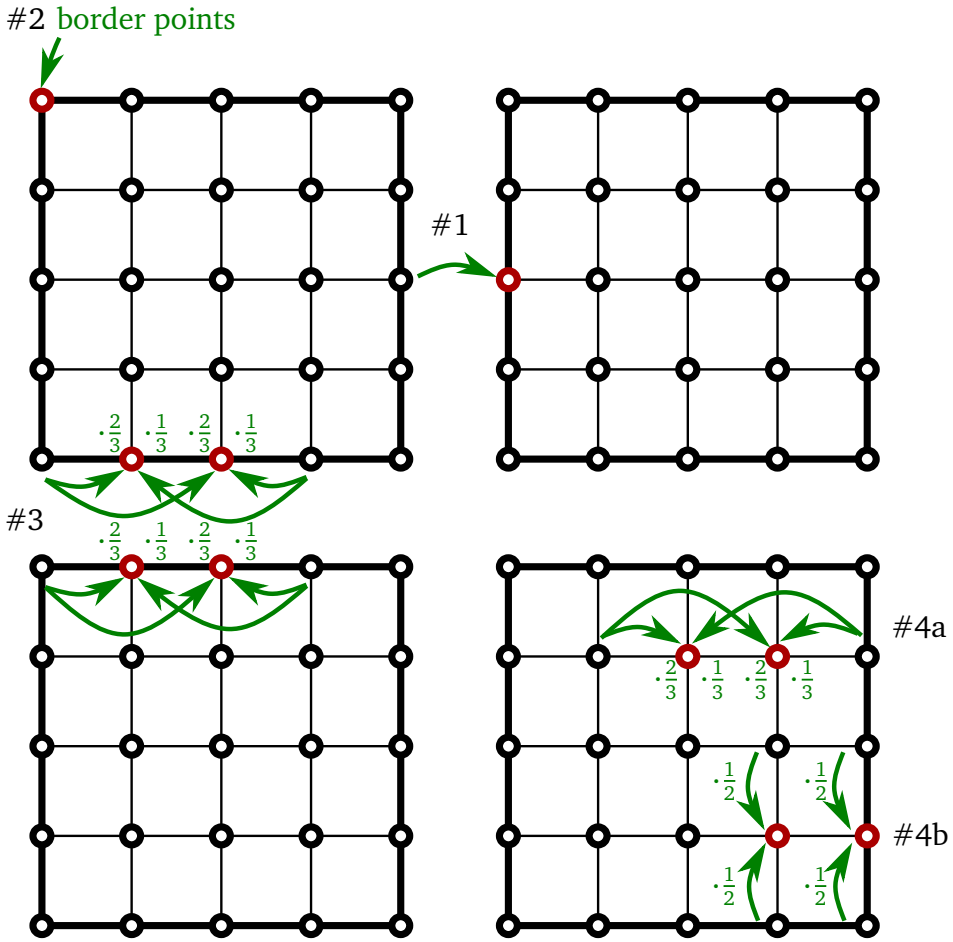


Figure 1.28: Examples of the four repair mechanisms for estimating incomplete streamlines during the parallel algorithm for mesh generation. Invalid streamlines are indicated by red circles, valid streamlines by black circles. The green arrows show the direction of data transfer.

1.5.6 Postprocessing of the Generated Streamlines

After repairing invalid streamlines, the final result of the algorithm is a grid with $(8\,n_x + 1) \times (8\,n_x + 1)$ fibers in the x-y plane and a configurable number of points in z direction, where $n_x = 2^{\ell_{\max}}$ is the number of subdomains per coordinate direction on the last recursion level.

If a higher number of fibers is desired than is naturally generated by the parallel algorithm, additional fibers can be created by interpolation in the existing grid of fibers. The implementation of the presented algorithm in OpenDiHu includes this postprocessing functionality as part of the mesh generation program. Alternatively, the step can be applied separately on any binary output file that contains a grid of fibers.

The action of increasing the number of fibers proceeds as follows. The initial grid contains the fibers that were created from the streamlines, called *key fibers*. A specified number m of additional fibers is placed evenly between the key fibers in both x and y coordinate directions. The additional fibers together with the key fibers form a grid of fibers in the muscle cross sections with an m times finer mesh width. In the grid of key fibers, every portion bounded by 2×2 key fibers contains $(2+m)^2-4$ additional fine fibers. The total number of fibers depending on n_x and m therefore is $N = (8\,n_x\,(1+m)+1)^2$. Due to construction of the algorithm, this number is always odd. This is a desired property because it yields an even number of elements per coordinate direction and this allows to construct a mesh with quadratic elements.

The new fibers are computed by barycentric interpolation. The location of every new point $\mathbf{p}$ is calculated from the nearest points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_3$ of key fibers in the x-y plane, numbered according to Fig. 1.20, by

$$\mathbf{p} = (1-\alpha_x)(1-\alpha_y)\,\mathbf{p}_0 + \alpha_x\,(1-\alpha_y)\,\mathbf{p}_1 + (1-\alpha_x)\,\alpha_y\,\mathbf{p}_2 + \alpha_x\,\alpha_y\,\mathbf{p}_3.$$

Here, the factors $\alpha_x, \alpha_y \in [0,1]$ are chosen in a way to create the fine grid of fibers:

$$\alpha_x = i/(m+1), \quad \alpha_y = j/(m+1) \quad \text{for } i,j = 0,\ldots,m, \quad (i,j) \neq (0,0).$$

In result, we can generate a 3D mesh where the number of points in x and y direction can be adjusted by the parameter m.

The points are stored in a binary file format. The contents of this output file can be either interpreted as grid points of a 3D mesh or as points of individual fibers. This is an advantage in a multi-scale simulation where both a 3D muscle mesh and multiple embedded 1D fiber meshes occur: First, all mesh information of both Ω_M and $\Omega_{F,i}$ can be given by a single file. And second, the 3D mesh is aligned with the 1D fibers and all 3D mesh points are also 1D mesh points.

The spacing in z direction between points on a fiber is chosen as $\Delta z = 0.01\,\text{cm}$. This value was found to ensures a low error in the model for propagation of electric stimuli along the muscle. The value leads to 1481 points per fiber on the belly of the biceps muscle. Every point coordinate is stored in the output file as double precision value with 8 B. The file contains a header of 72 B with descriptive information such as the number of fibers, some parameter values and a time stamp. The total file size therefore can be calculated by $72 + N \cdot 1481 \cdot 3 \cdot 8\,\text{B}$.

Often, the spatial resolution of the 3D mesh does not need to be as high as those of the fibers. In such a case, the 3D mesh can be constructed by only using a subset of the points given in the file. A stride in x, y and z direction is used to sample points for the structured 3D mesh.

A remaining issue concerns the mesh quality on the outer border. In general, the 3D mesh created by Alg. 3 has good quality because the interior points result from smooth streamlines that were traced through a divergence free vector field. The points at the border, however, are sampled from a triangulation of a tubular surface of the muscle. This surface is derived from imaging data, as described in Sec. 1.3. Therefore, the quality of the border points of the created mesh depends on the quality of the muscle surface and its triangulation. In a case where this quality is poor, only the outer layer of elements of the created 3D mesh is affected. Figure 1.29 shows an example for this effect in a grid of 9×9 fibers. It can be seen that only the fibers at the bottom of the image have an irregularity at their center. Such an irregularity potentially occurs at every z coordinates where a new subdomain begins. The cause is that the points on the rings are slightly shifted relative to each other.

A remedy in such a case is to discard the outer layer of fibers and construct the mesh only from points of the inner streamlines. Accordingly, the implementation of the presented algorithm Alg. 3 always creates two different output files. The first output file contains all fibers, the second contains all except the outer layer of fibers. The second file contains only $N = (8\,n_x\,(1+m)-1)^2$ instead of $N = (8\,n_x\,(1+m)+1)^2$ fibers.

1.5.7 Results and Discussion

In the following section, meshes that were generated by Alg. 3 are shown and the influence of various parameters is discussed. Figure 1.30 visualizes some results for the biceps and triceps muscles. If the recursion level is set to $\ell_{\max} = 0$, the algorithm generates meshes with the smallest possible number of fibers, which is a grid of 7×7 fibers. Figure 1.30a shows a grid of 7×7 fibers and the corresponding 3D mesh that was sampled from the fiber data using every 50th point in z direction of the fiber meshes. It can be seen that the generated fibers traverse all nodes of the generated 3D mesh and, thus, the 3D mesh is aligned with the fiber direction.

Figure 1.30b shows a similar result with 9×9 fibers. Here, the colors correspond to the solution of an electrophysiology simulation. Blue regions indicate that the fiber membranes have an electric potential equal to their resting potential, which indicates no

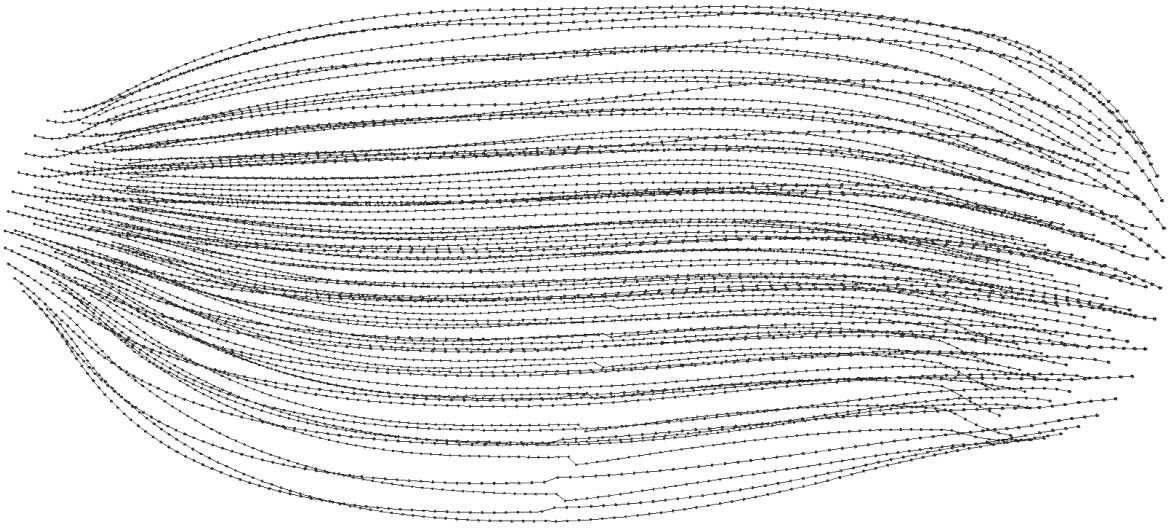


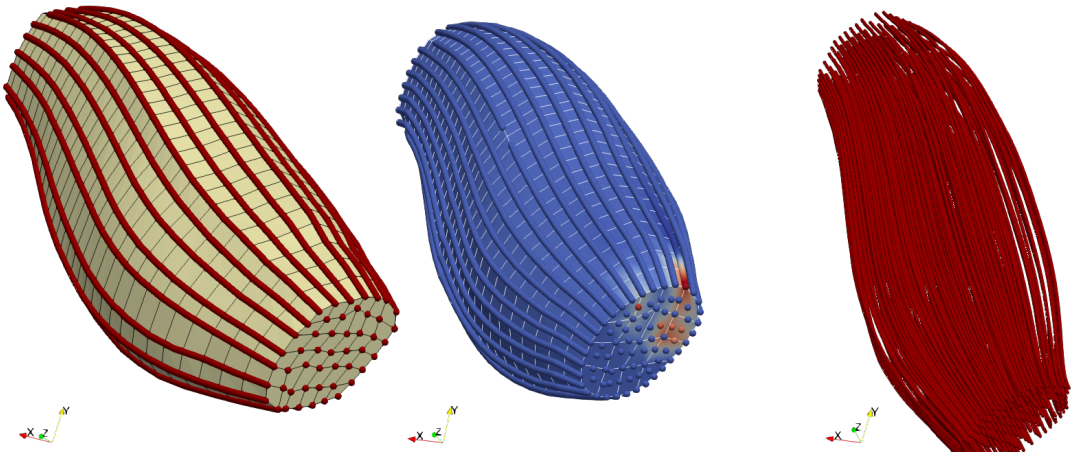
Figure 1.29: Resulting fibers and points on the fibers created with the parallel algorithm, 9×9 fibers with 1481 nodes each. Irregularities in the outer surface can be seen in the center at the bottom if the image.

activation. Orange and red colors correspond to activated regions. It can be seen that the activation is present at the same locations on both the fibers and the 3D mesh. In the simulation, this requires data mapping from the fiber meshes to the 3D mesh. Because all nodes of the 3D mesh are located on the fibers, this data transfer becomes trivial.

Figures 1.30c and 1.30d present grids with 13×13 and 67×67 fibers of the biceps muscle, respectively. Results with larger numbers of fibers are not shown here because in such visualizations the fibers become less distinguishable. Figures 1.30e and 1.30f show fibers for the triceps geometry.

The choices of the maximum recursion level ℓ_{max} and the fine grid parameter m determine the resulting number of fibers and the file size of the binary output file. Table 1.2 lists exemplary numbers of fibers and file sizes for different values of ℓ_{max} and m. The number n_{proc} of required processes to reach the maximum recursion level is also listed, it depends on ℓ_{max} by $n_{\text{proc}} = 2^{\ell_{\text{max}}}$. Two different numbers of fibers and corresponding file sizes are listed for every parameter combination. The two variants correspond to the two files that include respectively omit the fibers at the boundary.

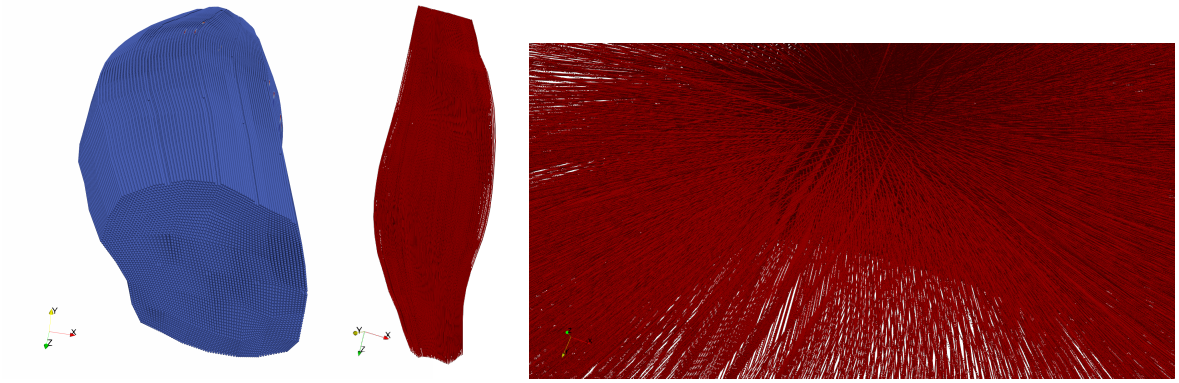
The table shows that meshes with different sizes can be constructed by appropriate choices of parameters. A realistic biceps muscle contains about 200 000 to 400 000 muscle fibers [Mac84]. The table shows that constructing a mesh in this range yields a file with a size of ca. 10 GiB. A mesh that contains 1 % of the realistic number of fibers can be stored in a file with size of ca. 100 MiB.



(a) Grid of 7×7 fibers (red) and the aligned 3D mesh with $7 \times 7 \times 30$ nodes (yellow).

(b) Grid of 9×9 fibers and 3D mesh with the solution of an electrophysiology simulation.

(c) Grid of 13×13 fibers.



(d) Grid 67×67 muscle fibers for the biceps geometry

(e) 25×25 fibers of triceps

(f) Grid of 67×67 fibers for the triceps geometry as seen from within the muscle. The total number of points is 8 982 489.

Figure 1.30: 1D fiber meshes and corresponding 3D meshes created by Alg. 3. The biceps geometry is used in (a)-(d), the triceps geometry in (e) and (f). The fibers have 1481 nodes each in the biceps muscle and 2001 nodes each in the triceps muscle.

The binary files to store the generated meshes are small compared to ASCII-based file formats as each point coordinate is represented by only 8 B. For comparison, the ASCII-based *exnode* format defined within the OpenCMISS framework uses 24 characters, i.e., 24 B to store one point coordinate. Additionally, a larger memory overhead for the description of the data is needed such that *exnode* files are more than three times larger than the binary files used in OpenDiHu.

The binary file format uses no compression that could further reduce the file size. The reason is that no extra effort should be needed when writing programs that parse these files. Thus, they can easily be handled by codes in different programming languages. For example, within OpenDiHu the file format is understood by various Python scripts and C++ programs.

Some numbers of fibers can be achieved with multiple, different parametrizations. Table 1.2 contains such alternatives for ℓ_{max} and m separated by slashes in the second and third row. For example, the combinations $(\ell_{\text{max}} = 0, m = 3)$ as well as $(\ell_{\text{max}} = 2, m = 0)$ both lead to a grid of 31×31 fibers (without boundary layer). However, the spatial location of the fibers in the muscle is not identical for these alternatives, as the fibers are determined by streamline tracing on differently resolved grids. In the first case with recursion depth $\ell_{\text{max}} = 0$ and fine grid interpolation parameter $m = 3$, many of the resulting fibers are interpolated from a coarse grid whereas in the second case with $\ell_{\text{max}} = 2$ and $m = 0$ all fibers are key fibers and are obtained by streamlines tracing in a fine grid.

Figure 1.31 shows parts of the resulting meshes at the longitudinal center of the muscle for the two considered cases. In Fig. 1.31a, the mesh obtained with $\ell_{\text{max}} = 0$ consists of a grid of traced key fibers and an interpolated finer grid of fibers. The key fiber grid is given by the corners of the gray checkerboard pattern in the image. It can be seen that the mesh consists of patches with 4×5 or 5×5 fibers that each have equal element lengths and angles. In comparison, the mesh in Fig. 1.31b that was obtained with $\ell_{\text{max}} = 2$ consists only of key fibers. Here, the change in shape going from one element to its neighbors occurs more smoothly than in Fig. 1.31a. This qualitatively implies a higher mesh quality.

To quantify this effect, all angles are considered that occur in an element in the x-y plane. The mean value of all angles is $\pi/2$. The variance can be used as a measure for mesh quality. If the variance is low, this indicates similar elements and, thus, good mesh quality. For the present example, the variance was computed for the mesh with 31×31 fibers and 1481 nodes per fiber and, thus, 1 332 000 3D elements in total. Figure 1.32

max. level ℓ_{max}	fine grid m	# proc. n_{proc}	# fibers	file size
0	0	1	$9 \times 9 = 81$	2.7 MiB
			$7 \times 7 = 49$	1.7 MiB
0/1	1/0	1/8	$17 \times 17 = 289$	9.8 MiB
			$15 \times 15 = 225$	7.6 MiB
0/1/2	3/1/0	1/8/64	$33 \times 33 = 1089$	36.9 MiB
			$31 \times 31 = 961$	32.6 MiB
0	7	1	$65 \times 65 = 4225$	143.2 MiB
			$63 \times 63 = 3969$	134.5 MiB
2	7	64	$257 \times 257 = 66\,049$	2.2 GiB
			$255 \times 255 = 65\,025$	2.2 GiB
2	15	64	$513 \times 513 = 263\,169$	8.7 GiB
			$511 \times 511 = 261\,121$	8.6 GiB

Table 1.2: Different parameter choices of ℓ_{max} and m and the resulting number n_{proc} of processes, number of fibers and file size. Some results can be achieved with different parameter combinations, e.g. both $\ell_{\text{max}} = 0, m = 1$ and $\ell_{\text{max}} = 1, m = 0$ result in 17×17 fibers. These combinations are separated by slashes.

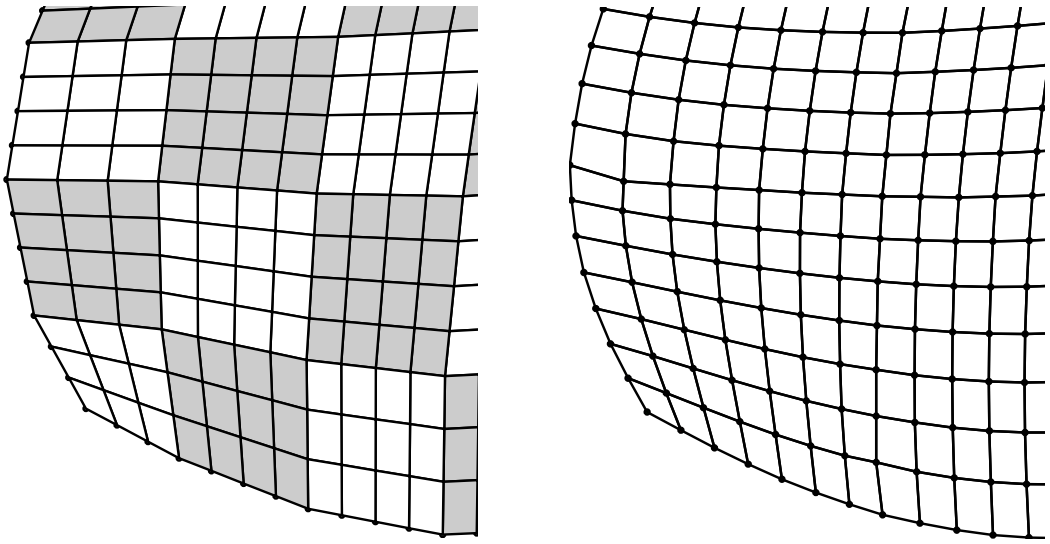
plots the variance for the three cases given in the third row of Tab. 1.2 with parameter combinations $(\ell_{\text{max}} = 0, m = 3)$, $(\ell_{\text{max}} = 1, m = 1)$ and $(\ell_{\text{max}} = 2, m = 0)$. Thus, the lowest bar includes the mesh shown in Fig. 1.31a and the upper-most bar includes Fig. 1.31b.

It can be seen that the quality of meshes on higher recursion levels with less interpolation increases as expected. This emphasizes the benefit of the parallel algorithm that uses finer meshes compared to the mesh used during serial execution of the algorithm.

In addition to ℓ_{max} and m, further options exist to tune the behaviour of Alg. 3. The number of border points in x or y direction as well as the number in z direction can be set to different values. The number in x and y direction is usually set to four, the number in z direction is set to 30.

Three more options are presented in the following and their effects are evaluated afterwards. First, the type of boundary conditions to the Laplace problem in Eq. (1.8) can be selected among the Neumann boundary conditions given by Eq. (1.9) or the Dirichlet boundary conditions given by Eq. (1.10). Second, the mesh for discretizing the Laplace equation $\Delta p = 0$ can be refined prior to the solution by increasing the number of elements per coordinate direction by the refinement factor $r \in \mathbb{N}$. A value of $r = 1$ corresponds to no refinement, $r > 1$ increases the number of 3D elements by the factor r^3.

Third, two options exist for computing the gradient value $\nabla p(\mathbf{x})$ at a point $\mathbf{x}$ in the



(a) Result for parameters $\ell_{\max} = 0, m = 3$, i.e., with three interpolated fibers between every two traced fibers.

(b) Result for parameters $\ell_{\max} = 2, m = 0$, i.e., without interpolation.

Figure 1.31: Comparison of generated meshes of the biceps with different maximum recursion levels $\ell_{\max}$. A lower left portion of the full mesh with 31×31 fibers is shown.

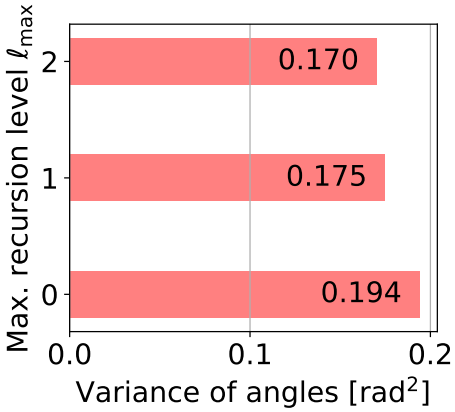


Figure 1.32: Variance of the element angles for meshes with the same number of 31×31 fibers, but created by different recursion levels $\ell_{\max}$. The parameters correspond to the third row of Tab. 1.2. A lower variance means better mesh quality.

domain, which is needed for the streamline tracing. Either, the gradient vector field is precomputed using finite differences and the nodal values of the solution field p. Or the gradient value is directly interpolated at $\mathbf{x}$ in the 3D element using a linear combination of the solution values and derivatives of the ansatz functions of the element.

In the following study, the first option is indicated by the characters "N" or "D" for Neumann or Dirichlet boundary conditions, the second option is specified by an integer value for r and the third option is indicated by "g" for the precomputed gradient field or "s" for using the solution values and derived ansatz functions.

For example, the scenario considered in Figures 1.31 and 1.32 is specified as "D2s", as it uses Dirichlet boundary conditions, a refinement factor of $r = 2$ and the solution values to compute the gradient.

A study of all possible combinations of the three options with $r \leq 4$ was conducted. The program was run with 8 processes. The parameters $\ell_{\max} = 1$ and $m = 1$ as well as the numbers of four and 30 border points in x and z direction were kept constant. Thus, meshes of the same size of 31×31 fibers were created by all considered variants of the algorithm.

As before, the variance of the element angles was used to rate each mesh quality. Additionally, also the variance of relative element lengths in the x-y planes was computed, using the calculation explained in Sec. 1.4.8. Most scenarios yielded a value of $2.2 \cdot 10^{-2}$. Scenarios with significantly different values were discarded, as their results contained incomplete streamlines.

The scenarios were ordered according to their mesh quality score, i.e., the variance of their element angles. Figure 1.33 plots the eight best results sorted by improving mesh quality from top to bottom. It can be seen that all values are close together, which indicates a similar good mesh quality for all shown parameter combinations. Nevertheless, the order shows some differences between the options. A better result was achieved if Neumann boundary conditions were used ("N") compared to Dirichlet boundary conditions ("D"). Also, a higher refinement factor r of the internal mesh was beneficial in this study. The variant without the precomputed gradient field ("s") performed better than the variant with gradient field computation ("g").

However, further studies with different recursion widths showed that the effect of these three options also depended on the scenario. For a higher recursion width, Dirichlet boundary conditions proved more robust in the sense that less incomplete streamlines occured. Larger refinement factors led to smaller mesh widths and in consequence to

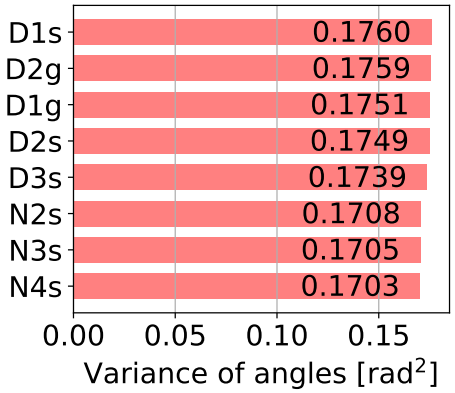


Figure 1.33: Comparison of the mesh quality that results from different options in the mesh generation algorithm. A lower variance means better mesh quality.

a smaller ghost layer for the streamline tracing, which always has a thickness of one element. Thus, more streamlines left their subdomain and became invalid. Therefore, e.g., a smaller value of r yielded better results for $\ell_{\max} = 2$.

In summary, for the maximum recursion level $\ell_{\max} = 1$ the best parameter combination among the tested combination is “N4s”. For $\ell_{\max} = 2$ a good combination is “D2s” and should be used instead.

How To Reproduce

The parallel algorithm is implemented in the example `parallel_fiber_estimation`. Numerous parameters can be set on the command line. After compilation, run the program as follows to get a description of all available options.

```
cd $OPENDIHU_HOME/examples/fiber_tracing/parallel_fiber_estimation/
  ↪ build_release
./generate ../settings_generate.py --help
```

Running the program without options and `--help` uses sensible default values. A given surface triangulation of the biceps muscle gets used by default. To compute the examples shown in this section, use and adjust the following script that runs the `generate` program and computes the mesh quality:

```
cd $OPENDIHU_HOME/examples/fiber_tracing/parallel_fiber_estimation/
  ↪ build_release
../run.sh
```

Computation of mesh and file sizes as shown in Tab. 1.2 can be done using the

compute_sizes.py script.

While the previously given commands are good for exploring the algorithms, generation of the meshes used for the simulation involves some more steps. Dedicated scripts exist that perform all steps and call the algorithms with the proper parameters. Starting from the STL file extracted from cmgui, as explained in Sec. 1.3.3, the next steps are:

(i) Scale the points from millimeters (used in the Visible Human dataset) to centimeters (used in the simulation),

(ii) remove the interior triangles,

(iii) translate the mesh such that the bounding box begins at $z = 0$, this is needed for the programs used in the next steps,

(iv) create the spline surface representation as explained in Sec. 1.3.4,

(v) compile and run the OpenDiHu programs to create the binary files of the 3D mesh and the 1D fibers meshes, the algorithm in Sec. 1.5 is used,

(vi) adjust the indexing and undo the translation in (iii),

(vii) refine the created meshes of key fibers by different numbers m of fine grid fibers, in total 10 different mesh sizes are created for differently refined simulations,

(viii) create meshes for the fat layer Ω_B ontop of the muscle surface, also in 10 different resolutions.

Two scripts are given for the biceps brachii and triceps brachii muscles. They perform all listed steps and also create intermediate output files that can be used to understand the process. Some steps are automatically skipped if the resulting output file already exists from a previous run. This is especially helpful for the removal of the interior triangles from the initial file which takes nearly a full day.

A third scripts creates three meshes $\Omega_{T,i}$ for the tendons of the biceps muscle, as visualized in Fig. 1.19. At the bottom, a single tendon mesh is created whereas at the top, two seperate tendons exist. The script involves numerous rotation and cropping operations of the initial surface, before the algorithm of Sec. 1.4 is executed. The three output files of the tendon meshes have the same file format

as the muscle meshes. The files have the extension *.bin* for "binary". The script `examine_bin_fibers.py` can be used to debug the created binary files.

The three scripts can be executed as follows:

```
cd $OPENDIHU_HOME/examples/electrophysiology/meshes
./process_meshes_biceps.sh
./process_meshes_triceps.sh
./process_meshes_tendons.sh
```

The output can be found in the subdirectory `processed_meshes`. For the total output about 46 GiB of drive space is required.

1.6 Conclusion and Future Work

This chapter presented algorithms for creating muscle meshes that are needed for multi-scale simulations of the musculoskeletal system. For the biceps muscle, 3D meshes for tendons on both ends and the muscle were created. Additionally, 1D fibers meshes were generated that are embedded in the mesh of the muscle. The 3D mesh and the 1D meshes are aligned with each other. This facilitates data mapping between the meshes and reduces numerical errors. All generated meshes are structured, which allows an efficient parallelization.

First, an overview of available meshing software and known algorithms in the literature was given. Very little software tools were capable of generating structured meshes and none fitted our special needs. Therefore, own algorithms were developed to generate meshes starting from medical imaging data.

A workflow was presented to generate a smooth surface triangulation from imaging data. Our base data was the male dataset from the Visible Human Project. Two alternatives within this workflow were presented, where the first alternative executed automatic image segmentation based on morphological operations and the second alternative used semi-automatic segmentation tools from the Physiome project. Then, smooth NURBS surfaces were fitted to the extracted boundaries of the muscle volumes.

Next, a novel algorithm to create structured meshes from a triangulated muscle surface was presented. The algorithm used harmonic maps on 2D slices to achieve good mesh quality. A method of computing streamlines in a divergence-free vector field to estimate muscle fibers, which is established in the literature was used. It allowed to embedded 1D

meshes for muscle fibers in the created 3D meshes of the muscle. Numeric experiments tested and evaluated different choices of triangulation and quadrangulation schemes for the 2D cross section and reference domains in our algorithm.

Next, a parallelized algorithm was introduced that was based on our first, serial algorithm. The algorithm used distributed memory parallelism and provided the same features as the serial algorithm, having the same formats for input and output. The difference was that it constructed a fine, partitioned mesh for streamline tracing that was distributed over all employed processes. Thus, it was possible to create finer meshes using more compute nodes. Differently resolved meshes of the biceps and triceps muscle volumes and muscle fibers were created using this algorithm. The superiority of the parallel algorithm using a higher number of processes compared to the serial execution was explained and demonstrated in a numerical experiment. Several options to fine-tune the algorithm were evaluated.

The presented algorithms and their implementation in OpenDiHu are the basis for further computations within this work. They are used to generate structured hexahedral meshes with good mesh quality. These meshes are required for efficient, parallel Finite Element simulations of various aspects of the neuromuscular system.

The presented algorithms are specialized for fusiform muscles and require the muscle geometry be oriented along one coordinate axis (the z axis) in order to generate a structured mesh that comprises planar slices that are normal to that direction. The algorithms can also be applied to any tubular surface geometry of more complex muscles and will construct the corresponding structured 3D mesh. The generated 1D fiber meshes, however, are only valid for muscles, where the approach of streamline tracing through the solution of the Laplacian potential flow problem with boundary conditions at the bottom and top ends of the muscle can be applied. In the literature, this approach has been successfully used for various muscles with more complex fiber architectures, such as the tibialis anterior, gluteus maximus and deltoid muscles [Cho13]. However, the locations where boundary conditions were prescribed was not always at the bottom and top ends of the muscle.

If in future work muscles with more complex layouts should be simulated, the approach could be as follows. Depending on the complexity of the outer geometry, first the presented algorithms (either Algorithms 1 and 2 or Alg. 3) can be used to create a structured 3D mesh. Then, a potential flow simulation can be manually setup in OpenDiHu using the 3D mesh and boundary conditions defined at proper locations. Seed points have to be defined and the streamline tracer of OpenDiHu can be used to create fiber meshes. In

consequence, the resulting 1D fibers will not be aligned with the 3D mesh. Algorithmicly, this poses no problem to the simulations in OpenDiHu as the data mapping functionality can handle arbitrarily positioned meshes. However, the parallel partitioning gets more involved as the combined domain of 1D and 3D meshes has to be partitioned equally for both mesh types.

Bibliography

[All05] **Alliez**, P. et al.: *Variational tetrahedral meshing*, ACM SIGGRAPH 2005 Papers, SIGGRAPH '05, Los Angeles, California: Association for Computing Machinery, 2005, pp. 617–625, isbn:9781450378253, doi:10.1145/1186822.1073238, https://doi.org/10.1145/1186822.1073238

[Bar96] **Barber**, C. B.; **Dobkin**, D. P.; **Huhdanpaa**, H.: *The quickhull algorithm for convex hulls*, ACM Trans. Math. Softw. 22.4, 1996, pp. 469–483, ISSN: 0098-3500, doi:10.1145/235815.235821, https://doi.org/10.1145/235815.235821

[Bes12] **Bessmeltsev**, M. et al.: *Design-driven quadrangulation of closed 3d curves*, ACM Trans. Graph. 31.6, 2012, ISSN: 0730-0301, doi:10.1145/2366145.2366197, https://doi.org/10.1145/2366145.2366197

[Bla93] **Blacker**, T. D.; **Meyers**, R. J.: *Seams and wedges in plastering: a 3-d hexahedral mesh generation algorithm*, Engineering with computers 9.2, 1993, pp. 83–93

[Ble05] **Blemker**, S. S.; **Delp**, S. L.: *Three-dimensional representation of complex muscle architectures and geometries*, Annals of biomedical engineering 33.5, 2005, pp. 661–673

[Che97] **Chew**, L. P.: *Guaranteed-quality delaunay meshing in 3d (short version)*, Proceedings of the thirteenth annual symposium on Computational geometry, 1997, pp. 391–393

[Cho13] **Choi**, H. F.; **Blemker**, S. S.: *Skeletal muscle fascicle arrangements can be reconstructed using a laplacian vector field simulation*, PLOS ONE 8.10, 2013, pp. 1–7, doi:10.1371/journal.pone.0077576

[Cor92] **Cordes**, C.; **Kinzelbach**, W.: *Continuous groundwater velocity fields and path lines in linear, bilinear, and trilinear finite elements*, Water resources research 28.11, 1992, pp. 2903–2911

[Del34] **Delaunay**, B. et al.: *Sur la sphere vide*, Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7.793-800, 1934, pp. 1–2

[Don05] **Dong**, S. et al.: *Quadrangulating a mesh using laplacian eigenvectors*, tech. rep., 2005

[Epp99] **Eppstein**, D.: *Linear complexity hexahedral mesh generation*, Computational Geometry 12.1-2, 1999, pp. 3–16

[Fer18] **Fernandez**, J. et al.: *Musculoskeletal modelling and the physiome project*, Multiscale Mechanobiology of Bone Remodeling and Adaptation, ed. by **Pivonka**, P., Cham: Springer International Publishing, 2018, pp. 123–174, isbn:978-3-319-58845-2, doi:10.1007/978-3-319-58845-2_3, https://doi.org/10.1007/978-3-319-58845-2_3

[Fie88] **Field**, D. A.: *Laplacian smoothing and delaunay triangulations*, Communications in applied numerical methods 4.6, 1988, pp. 709–712

[Gro09] **Grosland**, N. M. et al.: *Ia-femesh: an open-source, interactive, multiblock approach to anatomic finite element model development*, Computer methods and programs in biomedicine 94.1, 2009, pp. 96–107

[Hæg07] **Hægland**, H. et al.: *Improved streamlines and time-of-flight for streamline simulation on irregular grids*, Advances in Water Resources 30.4, 2007, pp. 1027–1045, ISSN: 0309-1708, doi:https://doi.org/10.1016/j.advwatres.2006.09.002, http://www.sciencedirect.com/science/article/pii/S0309170806001709

[Han17] **Handsfield**, G. G. et al.: *Determining skeletal muscle architecture with laplacian simulations: a comparison with diffusion tensor imaging*, Biomechanics and Modeling in Mechanobiology 16.6, 2017, pp. 1845–1855, ISSN: 1617-7940, doi:10.1007/s10237-017-0923-5, https://doi.org/10.1007/s10237-017-0923-5

[Ino15] **Inouye**, J.; **Handsfield**, G.; **Blemker**, S.: *Fiber tractography for finite-element modeling of transversely isotropic biological tissues of arbitrary shape using computational fluid dynamics*, 2015

[Jam15] **Jamin**, C. et al.: *Cgalmesh: a generic framework for delaunay mesh generation*, ACM Trans. Math. Softw. 41.4, 2015, ISSN: 0098-3500, doi:10.1145/2699463, https://doi.org/10.1145/2699463

[Jua06] **Juanes**, R.; **Matringe**, S. F.: *Unified formulation of velocity fields for streamline tracing on two-dimensional unstructured grids*, Comput. Methods. Appl. Mech. Engrg., submitted, 2006

[Kaw08] **Kawamura**, Y.; **Islam**, M. S.; **Sumi**, Y.: *A strategy of automatic hexahedral mesh generation by using an improved whisker-weaving method with a surface mesh modification procedure*, Engineering with Computers 24.3, 2008, pp. 215–229

[Kov11] **Kovacs**, D.; **Myles**, A.; **Zorin**, D.: *Anisotropic quadrangulation*, Computer Aided Geometric Design 28.8, 2011, Solid and Physical Modeling 2010, pp. 449–462, ISSN: 0167-8396, doi:https://doi.org/10.1016/j.cagd.2011.06.003, http://www.sciencedirect.com/science/article/pii/S0167839611000689

[Kus19] **Kusterer**, J.: *Extraktion anatomischer Strukturen und Darstellung durch NURBS*, Deutsch, Bachelorarbeit: Universität Stuttgart, Institut für Parallele und Verteilte Systeme, Simulation großer Systeme, Bachelorarbeit, 2019, http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=BCLR-2019-19&engl=0

[Led08] **Ledoux**, F.; **Weill**, J.-C.: *An extension of the reliable whisker weaving algorithm*, proceedings of the 16th International Meshing Roundtable, Springer, 2008, pp. 215–232

[Mac84] **MacDougall**, J. D. et al.: *Muscle fiber number in biceps brachii in bodybuilders and control subjects*, Journal of Applied Physiology 57.5, 1984, PMID: 6520032, pp. 1399–1403, doi:10.1152/jappl.1984.57.5.1399, eprint: https://doi.org/10.1152/jappl.1984.57.5.1399, https://doi.org/10.1152/jappl.1984.57.5.1399

[Mak91] **Maker**, B. N.: *Nike3d: a nonlinear, implicit, three-dimensional finite element code for solid and structural mechanics*, 1991

[Mar10] **Marchandise**, E. et al.: *Quality meshing based on stl triangulations for biomedical simulations*, International Journal for Numerical Methods in Biomedical Engineering 26.7, 2010, pp. 876–889

[Mar11] **Marchandise**, E. et al.: *High-quality surface remeshing using harmonic maps—part ii: surfaces with high genus and of large aspect ratio*, International Journal for Nu-

merical Methods in Engineering 86.11, 2011, pp. 1303–1321, doi:10.1002/nme.3099, eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.3099, https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.3099

[Men16] **Meng**, M.; **He**, Y.: *Consistent quadrangulation for shape collections via feature line co-extraction*, Computer-Aided Design 70, 2016, SPM 2015, pp. 78–88, ISSN: 0010-4485, doi:https://doi.org/10.1016/j.cad.2015.07.010, http://www.sciencedirect.com/science/article/pii/S0010448515001104

[Möl97] **Möller**, T.; **Trumbore**, B.: *Fast, minimum storage ray-triangle intersection*, Journal of Graphics Tools 2.1, 1997, pp. 21–28, doi:10.1080/10867651.1997.10487468, eprint: https://doi.org/10.1080/10867651.1997.10487468, https://doi.org/10.1080/10867651.1997.10487468

[Owe98] **Owen**, S. J.: *A survey of unstructured mesh generation technology.* IMR 239, 1998, p. 267

[Pie12] **Piegl**, L.; **Tiller**, W.: *The NURBS book*, Springer Science & Business Media, 2012

[Pol88] **Pollock**, D. W.: *Semianalytical computation of path lines for finite-difference models*, Groundwater 26.6, 1988, pp. 743–750, doi:10.1111/j.1745-6584.1988.tb00425.x, eprint: https://ngwa.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1745-6584.1988.tb00425.x, https://ngwa.onlinelibrary.wiley.com/doi/abs/10.1111/j.1745-6584.1988.tb00425.x

[Pri95] **Price**, M.; **Armstrong**, C. G.; **Sabin**, M.: *Hexahedral mesh generation by medial surface subdivision: part i. solids with convex edges*, International Journal for Numerical Methods in Engineering 38.19, 1995, pp. 3335–3359

[Pri97] **Price**, M. A.; **Armstrong**, C. G.: *Hexahedral mesh generation by medial surface subdivision: part ii. solids with flat and concave edges*, International Journal for Numerical Methods in Engineering 40.1, 1997, pp. 111–136

[Ram18] **Ramasamy**, E. et al.: *An efficient modelling-simulation-analysis workflow to investigate stump-socket interaction using patient-specific, three-dimensional, continuum-mechanical, finite element residual limb models*, Frontiers in Bioengineering and Biotechnology 6, 2018, p. 126, ISSN: 2296-4185, doi:10.3389/fbioe.2018.00126, https://www.frontiersin.org/article/10.3389/fbioe.2018.00126

[Rem10] **Remacle**, J.-F. et al.: *High-quality surface remeshing using harmonic maps*, International Journal for Numerical Methods in Engineering 83.4, 2010, pp. 403–425, doi:10.1002/nme.2824, eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2824, https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2824

[Rup95] **Ruppert**, J.: *A delaunay refinement algorithm for quality 2-dimensional mesh generation*, Journal of Algorithms 18.3, 1995, pp. 548–585, ISSN: 0196-6774, doi:https://doi.org/10.1006/jagm.1995.1021, http://www.sciencedirect.com/science/article/pii/S0196677485710218

[Sch96] **Schneiders**, R.: *A grid-based algorithm for the generation of hexahedral element meshes*, Engineering with computers 12.3-4, 1996, pp. 168–177

[Sch97] **Schneiders**, R.: *An algorithm for the generation of hexahedral element meshes based on an octree technique*, 6th International Meshing Roundtable, 1997, pp. 195–196

[Sed11] **Sedgewick**, R.; **Wayne**, K.: *Algorithms*, Addison-wesley professional, 2011

[She02] **Shewchuk**, J. R.: *Delaunay refinement algorithms for triangular mesh generation*, Computational Geometry 22.1, 2002, 16th ACM Symposium on Computational Ge-

ometry, pp. 21–74, ISSN: 0925-7721, doi:https://doi.org/10.1016/S0925-7721(01)00047-5, http://www.sciencedirect.com/science/article/pii/S0925772101000475

[She96] **Shewchuk**, J. R.: *Triangle: engineering a 2D quality mesh generator and delaunay triangulator*, Applied Computational Geometry: Towards Geometric Engineering, ed. by **Lin**, M. C.; **Manocha**, D., vol. 1148, Lecture Notes in Computer Science, From the First ACM Workshop on Applied Computational Geometry, Springer-Verlag, 1996, pp. 203–222

[Spi96] **Spitzer**, V. et al.: *The Visible Human Male: A Technical Report*, Journal of the American Medical Informatics Association 3.2, 1996, pp. 118–130, ISSN: 1067-5027, doi:10.1136/jamia.1996.96236280, eprint: https://academic.oup.com/jamia/article-pdf/3/2/118/2089636/3-2-118.pdf, https://doi.org/10.1136/jamia.1996.96236280

[Sta06] **Staten**, M. L. et al.: *Unconstrained paving and plastering: progress update*, proceedings of the 15th International Meshing Roundtable, Springer, 2006, pp. 469–486

[Sta10] **Staten**, M. L. et al.: *Unconstrained plastering—hexahedral mesh generation via advancing-front geometry decomposition*, International journal for numerical methods in engineering 81.2, 2010, pp. 135–171

[Tau96] **Tautges**, T. J.; **Blacker**, T.; **Mitchell**, S. A.: *The whisker weaving algorithm: a connectivity-based method for constructing all-hexahedral finite element meshes*, International Journal for Numerical Methods in Engineering 39.19, 1996, pp. 3327–3349

[Unt13] **Untaroiu**, C. D.; **Yue**, N.; **Shin**, J.: *A finite element model of the lower limb for simulating automotive impacts*, Annals of biomedical engineering 41.3, 2013, pp. 513–526

[Vic12] **Vichot**, F. et al.: *Cardiac interventional guidance using multimodal data processing and visualisation: medinria as an interoperability platform*, 2012

[Zha03] **Zhang**, Y.; **Bajaj**, C.; **Sohn**, B.-S.: *Adaptive and quality 3d meshing from imaging data*, Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications, SM '03, Seattle, Washington, USA: Association for Computing Machinery, 2003, pp. 286–291, isbn:1581137060, doi:10.1145/781606.781653, https://doi.org/10.1145/781606.781653

[Zha05] **Zhang**, Y.; **Bajaj**, C.; **Sohn**, B.-S.: *3d finite element meshing from imaging data*, Computer Methods in Applied Mechanics and Engineering 194.48, 2005, Unstructured Mesh Generation, pp. 5083–5106, ISSN: 0045-7825, doi:https://doi.org/10.1016/j.cma.2004.11.026, http://www.sciencedirect.com/science/article/pii/S0045782505000800

[Zha14] **Zhang**, J. et al.: *The map client: user-friendly musculoskeletal modelling workflows*, Biomedical Simulation, ed. by **Bello**, F.; **Cotin**, S., Cham: Springer International Publishing, 2014, pp. 182–192, isbn:978-3-319-12057-7