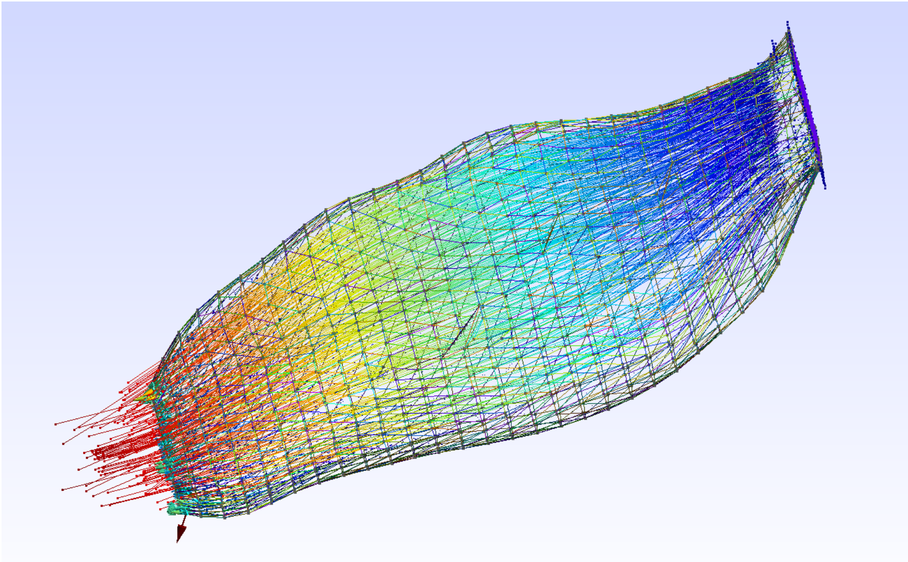
Muscular fascicle arrangement based on Laplacian vector fields

Jan Kusterer, Niven Ratnamaheson, Raimund Rolfs, and Tobias Walter



Abstract—

Index Terms—muscle, fascicle, mesh, Laplacce, electrostatic, streamline

1 Introduction

Skeletal muscles have a wide range of anatomical architectures. They often form a heterogenous curvature, while the tendon and bone attachments vary in their morphology [2]. Each muscle consists of multiple muscle fibers which can contract via Myosin motors [1] and thus create a force pulling on the bones. These fibers are not long enough to spread across the whole muscle, so they are grouped in paralleled bundles called fascicles. According to [3] the biceps of an athletic individuals has around 300.000 muscle fibers, which are single cells approximately 50m in diameter and several centimetres long [5]. Each fiber has a thick myosin filament and a thin actin filament, which surrounds the myosin. During the process of muscle contraction, the actin and myosin filaments slide past each other resulting in shortening of the fiber. Since the mechanical force is transmitted along the muscle fibers, it is important for biomechanical simulations to obtain an adequate representation of their trajectories. This way we can get much more realistic geometries e.g. of a contracted biceps. However, this is not a simple task because of the complexity and diversity of the various muscles. To simplify the problem by a small degree, we can simulate the fascicles instead of every single fiber. To simulate the contraction of a muscle we need an approximation of the fascicles, that stretch across the muscle. Although we can see them with our bare eyes, it is not possible to determine their pathway using an ordinary CT-scan. Therefore, we need a method to calculate the stream of these fascicles. Muscles are often modelled using a lumped-parameter approach that assumes a simplified arrangement of fascicles. However, as Choi and Blemker stated, these models are not able to cover three-dimensional deformations. One of the more promising approach was using a Laplacian vector field as presented in [2]. The goal of our work is the approximation of Muscle fascicles based on the 3D-model of a muscle, primarily the musculus biceps brachii, with an approach using a Laplacian vector field. Gmsh [4] is a tool for generating 3-D finite element meshes with built in pre- and post processing. We use it for multiple reasons as it is free software that features it's own scripting language, which uses code similar to C++ with loops, conditionals and user-defined macros. We use these features, because they match our needs for Meshing, Simulation and analysis of the Model. It can be compiled without the GUI, directly from the command line. For other manipulations and computations we use Python.

2 Methods

The first step of calculation of the streamlines based on a CT-Scan of the muscle, using gmsh is to repair the stl file. Small holes in the surface would make it impossible for gmsh to mesh the volume. Most of the mesh processing tools (eg. Meshlab, Blender, Microsoft 3D Builder) feature a repair function.

Since our pipeline is not suited to compute the streamlines for a biceps with a fork included, we need to cut the 3D-model just below the fork and some part from the opposite side. As a nice side-effect we get a better distribution of the streamlines in the center part of the muscle.

The stl-file does not connect the surface triangles with its neighbours. Gmesh however needs one big surface to create a volume in which we can compute the streamlines. To connect all the small surfaces, we reclassify the mesh with a threshold of 0. This runs a edge-detection for all triangles with an angle to its neighbour greater than 0 (so basically every neighbour) and we can connect all the detected surfaces to one big surface.

The reclassified muscle can then be reassambled within a gmsh specific script file. Another step here is to detect the inflow and outflow surfaces for later steps. A python programm is used for this detection.

3D-Meshing the volume is done completely by gmsh.

As stated by Choi and Blemker [2], a Laplacian vector field

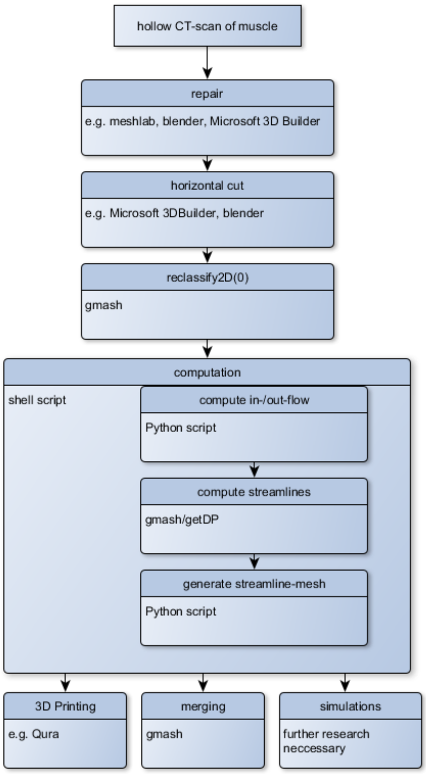


Figure 1. Flowchart of our Pipeline

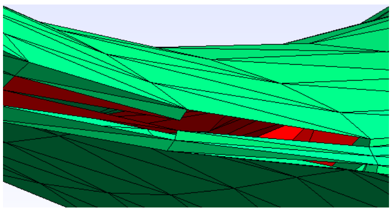


Figure 2. Holes in the forked area of the biceps, which need to be closed. The inside of the biceps is colored red for better contrast.

yields reasonable results when simulating fascicle arrangement. Thats why we use a simulation of an electronic field. GetDP the solver, used by gmsh, calculates the vectors inside the mesh of the biceps.

To create the streamlines, we apply the streamlines-plugin on the vector-view. We than have decent streamlines in the biceps brachii.

With help of another Python script we extract the streamlines from the postprocessing file and write them in a new geo file. Merging this file with the original biceps surface displays their arrangement.

3 Preparations

As we start from a CT-scan we first need to check whether the Volume is complete and closed. We discovered little holes in the Surface, as seen in Figure 2, when we tried to Mesh the Volume, which caused gmsh to shut down. Therefore we used a Repair feature which most of the common 3D modelling tools share. The Biceps, as the name indicates has two heads. For easier computation and better outcome of the streamlines, we cut the muscle horizontal just below the fork and below the main volume.

4 Reassembling

The stl-file does not connect the surface triangles with its neighbours. Gmesh however needs one big surface to create a volume in which we can compute the streamlines. To connect all the small surfaces, we use the reclassify option with a threshold of 0. This runs an edge-detection for all triangles with an angle to its neighbour greater than 0. This excludes the two cut surfaces, since these surfaces all have an angle of 0. The prepared muscle is then processed in our pipeline, see Figure 1. Finally we can recombine all the detected surfaces to one big surface. The recombination is done within gmsh by iterating over the surfaces and declaring the recombined surface as one. The basic operations done in the script are the following:

```
//declare ss as surface
ss[] = Surface {:};
//combine the surfaces
Compound Surface{ss[]};
//create a surface loop
Surface Loop(10000) = {ss[]};
//define the volume inside the loop
Volume(100) = {10000};
//physical entities are needed for simulation
Physical Surface (100) = {ss[]};
Physical Volume ("Body",10) = 100;
//meshing 3D when executing script
Mesh 3;
//disable Automatic Remeshing
Solver.AutoMesh = 0;
```

With this new reassembled 3D-structure we now run our python script to detect the two cut surfaces. Since we cut the biceps orthogonal to the Z-axis the process is fairly simple. We look for all vertices with the highest z-coordinate for the upper boundary and the lowest z-coordinate for the lower boundary, respectively. These two sets are grouped and form two new surfaces. With this surface we save the maximum and minimum of the y- and x- coordinates for later use of the streamlines. We do this to cover the whole surface with the starting points of the streamlines.

5 Meshing

Meshing with Gmsh is fairly easy. On default gmsh selects between three 2D algorithms and two 3D algorithms. The automatic algorithm selection tries to select the most apropriate for the given structure. For 2D algorithms there are "MeshAdapt", "Delauny" and the "Frontal" algorithm. Every one of them has different uses. According to Gmsh The "MeshAdapt" works best for very complex, curved surfaces. "Frontal" is the best choice, when high element quality is important. And "Delauny" is fastest for large meshes of plane Surfaces. As stated in the manual for Gmsh the automatic selection chooses "Delauny" for plane surface and "MeshAdapt" for all other surfaces. For 3D algorithms there are "Delauny" and "Frontal". The Delaunay algorithm is the most robust and the fastest. However, this algorithm will sometimes modify the surface mesh, and is thus not suitable for producing hybrid structured/unstructured grids. In that case the Frontal algorithm should be preferred. As our mesh is only unstructured, the "Delauny" is our algorithm. The quality of the elements produced by both algorithms is comparable. For our 3D Mesh, first, the 2D surfaces, then the Volume is meshed.

6 Simulation

Our goal is to calculate streamlines of laplacian vector fields, which according to [2] are fairly close to the muscular fascicle arrangement. Gmsh's solver getDP features a plugin, which calculates the Streamlines based on the vector field. Therefore we need a simulation on fluid flow, Electrostatics or similar. We tried out both thermodynamic and electrostatic simulation. In figure 3 we see the unsatisfying results we earned using the thermodynamic approach. At the corner of the L-figure there are streamlines leaving the Model.

The result of the electrostatic simulation is spread evenly and has fewer whirls. It has a overall better coverage of the model.

The Laplace equation can be used in 3D just as in 2D. getDP's problem definition files (.pro) are used to descripe the models for simulation. In this model we consider the calculation of the electric field given a static distribution of electric potentiol. This matches to an "electrostatic" physical model. On the one End of the Muscle we have a conducting surface on top of a dielectric Volume, called "Body". A Dirichlet boundary sets the potential on the boundary of the conducting surface , called "Electrode", to 10 mV and to 0 V on the other end of the Muscle, called "Ground". A homogeneous Neumann boundary condition is definded on the surface of the muscle to truncate the domain.
We Based our Problem definition on the Tutorial for electrostatics. [4] The Structure of the File is as Follows:

Group start by giving meaningful names to physical regions defined in the mesh file. We only use the Regions Body, Electrode and Ground. After that we define abstract regions, that are used below.

Function Here we define Material laws.

Constraint The Dirichlet boundary condtition is defined piecewise. The constraint "Dirichlet_Ele" is invoked later in the FunctionSpace.

Group This is the domain definition of The FunctionSpace, which lists al regions on wich a field is defined. The Domain contains both the Volume and the Surface of the muscle.

FunctionSpace The functionSpace is used to pick the electric scalar potential. The solution is defined by:
-the domain defenition
-a type
-a set of basis functions
-a set of entities to wich the basis functions are associated (here all the nodes of the domain)
-a constraint (here the Dirichlet boundary conditions)

Jacobian Jacobians are used for specifying the mapping of elements in the mesh. "Vol" represents the classical 1-to-1 mapping for identical dimension whereas "sur" represents the mapping between 2D and 3D and "lin" is used to map line segments to segments in three dimensional space.

Integration The "Integration" segment specifies how many points are used for the Gauss quadrature rules.

Formulation A GetDP formulation encodes a weak formulation of the partial differential equation. i.e. $-div*(\varepsilon*grad(v))=0$ In our simulation we are looking for functions v such that $(\varepsilon*grad(v),grad(v'))\_Vol\_Ele=0$ holds for all v'. The "Integral" statement in the Formulation is a representation of this weak formulation. It features four seperated arguements:
-the density
-the domain of integration,
-the Jacobian of the transformation,
-the integration method to be used.

Resolution In the Resolution it is specified what to do with the weak formulation. We simply generate a linear system, solve it and save the solution.

Post Processing There are two parts of postprocessing available in getDP. First we can evaluate the outcome of the formulation. Here we the quantities are the scalar electric potetial, the electric field and the electric displacement. The second part are postprocessing operations. In our case the Streamline generation is done here. The Script, which detects the inflow and outflow surfaces writes the X/Y/Z values here so that we get a good coverage of starting points for our streamlines. The Operation is then run and the output saved to a postprocessing file (.pos).

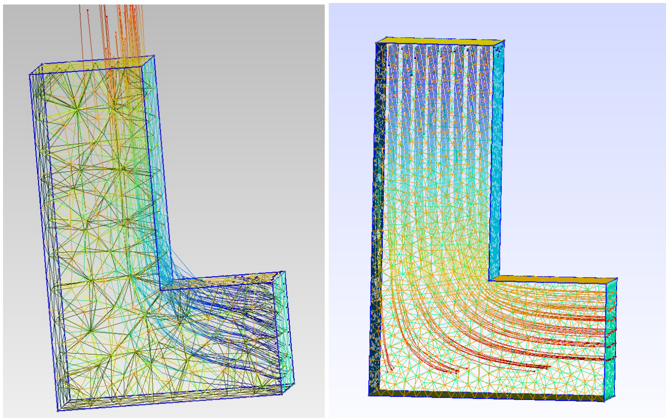


Figure 3. Comparison of thermodynamic (left) and electrostatic (right) calculation.

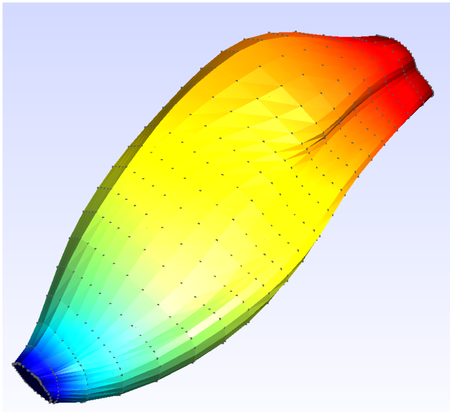


Figure 4. The cut Biceps after the simulation. The colors show the Distance from the inflow based on the vector field.

7 Streamlines

So now we use the earlier mentioned Plugin within the post processing described above, which calculates Streamlines. The starting points of the Streamlines are on the upper surface of the biceps, which was calculated in the Reassembling section. The number of Streamlines can be set when executing the Pipeline. The parameters are given as number of points on the X- and Y- axis. The extends of the surface, are the Minimal an Maximal X and Y values. Thats why not all of the Streamlines will start inside of the Muscle. Additionally we can set the number of steps we want to execute and the length of a step. The speed of the streamlines depends on the potential at the point of calculation at that timestep. Afterwards instead of extracting the complete streamlines, we can only extract the stepwise vector data of the streamlines into a postprocessing file in our case this is called MuscleStreamline.pos.

8 Representation of Data

Our next step is to add up the vectors of each streamline separately. This is done by our Python Script Streamline-converter, which gets the vector data from the BicepsStreamline.pos file. Furthermore, the script creates a new file streamlines.geo with the data of each streamline in it. All in all, we can now use GMSH to merge the streamlines.geo file with our biceps surface to visualize the result. However there are Multiple uses of how to represent the streamlines. Another visualization is 3D-printing the streamlines.

9 Printing

In preparation for Printing the Streamlines must get some volume. For this step a python script creates points along the streamlines at the wanted radius. The points are then connected and Meshed. As of shape we choose a simple hexagonal profile. After volume is created all of the streamlines are combined into one stl file. Streamlines that lay outside of the Biceps will get erased in this step, because they will not have any points in the file. The muscle fibres are substracted off of the full biceps using blender in order to have a hollowed out model with space for the fibres. The 3D-Printer we use is the Ultimaker 3 extended it has two extruders so we can print with two different materials or colours. The software used to pepare printing is Cura. We tried printing with a clear material but we didnt get a result, wich allowed the inner setup to be visible. We came up with another solution which is:-. In Cura the hollowed out muscle and the fibres are merged together and colours and materials are set.

10 Conclusion

11 Future work

-alignment of 3D fascicles with ”real thickness”

Acknowledgments

This work was mostly funded by cake and caffein.

References

[1] J. Chen, X. Zhang, and H. Wang. Research on biomechanical model of muscle fiber based on four-state operating mechanism of molecular motors. In *10th IEEE International Conference on Nano/Micro Engineered and Molecular Systems*, pages 529–532, April 2015.

[2] H. F. Choi and S. S. Blemker. Skeletal muscle fascicle arrangements can be reconstructed using a laplacian vector field simulation. *PLOS ONE*, 8:1–7, 10 2013.

[3] A. A. Etemadi and F. Hosseini. Frequency and size of muscle fibers in athletic body build. *The Anatomical Record*, 162(3):269–273.

[4] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331.

[5] C. GM. The cell: A molecular approach. 2nd edition.