The Visual Computer manuscript No.
(will be inserted by the editor)

A practical framework for generating volumetric meshes of subject-specific soft tissue

Pieter Peeters · Nicolas Pronost

Abstract Studying human motion using musculoskeletal models is a common practice in the field of biomechanics. By using such models, recorded subject's motions can be analyzed in successive steps from kinematics and dynamics to muscle control. However simulating muscle deformation and interaction is not possible, but other methods such as a *finite element* (FE) simulation are very well suited to simulate deformation and interaction of objects. In this paper we present a practical framework for the automatic generation of FE ready meshes based on subject-specific segmented MRI data. The proposed method resolves several types of data inconsistencies: noise, an incomplete dataset and self-intersections. This paper shows the different steps of the method, such as solving overlaps in the segmented surfaces, generating the volume mesh and the connection to a musculoskeletal simulation.

Keywords Virtual human · Finite element simulation · Musculoskeletal simulation · Subject-specific modeling · Computational geometry

1 Introduction

The simulation of human motion using anatomically-based musculoskeletal models is of interest in many fields including computer graphics, biomechanics, motion analysis, medical research and virtual character animation. Much research has been done to understand the musculoskeletal system, and so there is a large amount of data [1] describing the mechanics of muscle, the geometric relationships between muscles and bones, and the motions of joints. In the medical field, the neuromuscular system has been studied to get a better understanding of movement disorders in patients with cerebral palsy, stroke, osteoarthritis and Parkinson's disease. Thousands of patients have been studied, recording their neuromuscular excitation patterns both before and after treatment. However, the detailed understanding of the function of each of the elements of the musculoskeletal system remains a major challenge.

Researching these diseases in real-life experiments has the following limitations. First, many important variables are hard to measure in an experiment, such as the force generated by a muscle. Second, it is difficult to deduce cause effect relationships in complex dynamic systems based on experimental data alone. These problems that arise when analyzing experimental data can be solved for a large part by combining the experimental data with a neuromuscular simulation framework. Neuromuscular simulation allows one to study the different facets of neuromuscular activity, specifically the cause-and-effect relationships between muscular excitation patterns, muscle forces and resulting motion of the body. It can integrate theoretical models describing the anatomy and physiology of the musculoskeletal system and the mechanics of multi-joint movement. Simulations also enrich experimental data by providing estimates of important variables such as muscle and joint forces, which are difficult to measure experimentally.

Another approach to simulating motion is the use of *finite element analysis* (FEA). Finite element analysis is a method to simulate a complex environment

P.W.A.M. Peeters
Utrecht University (The Netherlands), Games and Virtual Worlds

N. Pronost
Utrecht University (The Netherlands), Games and Virtual Worlds
Tel.: +31-302537578
Fax: +31-302534619
E-mail: nicolas.pronost@uu.nl

by representing it as a set of small elements which are interconnected by a means of (differential) equations, which define the properties of the environment. Typical data sources for creating FEA setups are volume scans of subjects, using techniques such as *magnetic resonance imaging* (MRI) or *X-ray computed tomography* (CT). Using such a volumetric dataset of a subject, a detailed FE simulation setup, consisting of a set of elements that describe bones, muscles and tendons, can be created. The main problem of these datasets is the inaccuracies which arise from and are inherent to the data acquisition process. Even the best acquisition methods such as done for the visible human dataset need some steps to refine the data [2]. Since FEA offers such an advantage in simulating interaction between muscles and bones over a more high-level type of simulation as musculoskeletal simulation, the question arises whether we could create a bridge between these types of simulation environments.

The aim of this study is to propose a practical framework capable of preparing the raw volumetric dataset of a subject to an FE simulation of the deformation of its soft tissue. The framework has also been designed in a way that connection to musculoskeletal simulations of the same subject be facilitated. We demonstrate our method on rigid and soft bodies of the lower limb with a particular attention to the knee area. We believe that our framework is a step towards the automatic simulation of muscle deformations in virtual humans in a predictive manner through the interaction of the muscle anatomy with applications in computer graphics. Similarly, the method can simulate a complex muscle structure so that muscle function can be investigated in biomedical engineering.

2 Related work

The biomechanics community has ongoing efforts to create detailed human musculoskeletal models. Although recent models [3,4] provide accurate muscle parameters for the whole body, they by definition do not provide subject-specific geometrical data needed to simulate detailed muscular models. These models use lumped parameter 1D muscle models that do not account for muscle geometry. In musculoskeletal representation physiological parameters such as muscle lengths and muscle forces have been of primary interest, and the realistic visualization has played a secondary role. Over the years, neuromuscular simulation has evolved from a fragmented community where each research group developed their own simulation software into a more collaborative environment. This has partly been the result of the efforts by the creators of the OpenSim platform [5,3], by providing an open-source platform that lets the user develop a wide variety of musculoskeletal models that can be shared due to the open nature of the software. We also observe a recent tendency to transfer these musculoskeletal models and techniques to the computer animation community [6].

Many attempts have been made at simulating muscles in high detail using the method of finite element simulation [7–9], for example, to investigate intramuscular pressures [10]. Teran et al. have designed a framework for extracting and simulating musculoskeletal geometry from the visible human dataset [2,11]. In their study they had to use a motion from a different subject, since the visual human dataset is obtained from a deceased subject and no motion capture has been previously performed. The same dataset was used in ref. [12] to create a 3D model of the human leg, specifically for visualization of deformations, and incorporated also the rendering of muscle fibers using textures.

In order to simulate interactions between rigid and soft bodies within a common framework, two-way coupling methods have been investigated. For example, Shinar et al. [13] proposed a time integration scheme for solving dynamic elastic deformations in soft bodies interacting with rigid bodies. Despite heavy computation times required for the finite element simulation, their framework can handle two-way coupled contact, collision, stacking, friction, articulation, and PD control. Real-time performances have been achieved by Kim and Pollard [14] by using reduced deformable models and linear-time algorithms for rigid body dynamics. The primary focus of their work was to develop a fast physically based simulation system for skeleton-driven deformable body characters. In [15] Stavness et al. focused on modeling and simulating subject-specific anatomical structures composed of both hard and soft tissue components. They demonstrated their framework on the dynamics of the jaw tongue hyoid complex where the deformation of the 3D FEM tongue model is driven by muscle activations. Their work on a solver insists on the need to integrate constraints directly into the velocity solution.

In [2] the segmentation of anatomical entities was performed by creating a level set representation of each entity relevant for the simulation. The signed distance function required for the level set procedures and to generate the triangulated surface was the fast marching method [16]. They use slice-by-slice contour sculpting to repair problem regions. First they manually exam-

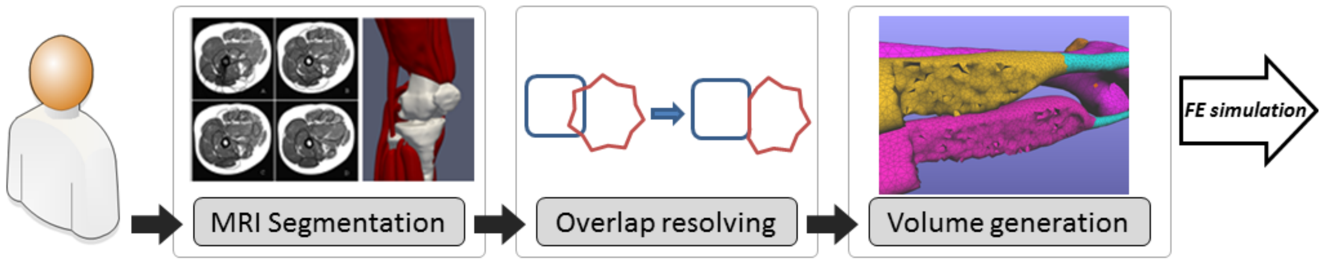

Fig. 1 Our framework prepares subject-specific segmented MRI datasets for the FE simulation of soft tissues.

ine each slice visually to check for and eliminate errors. Level-set smoothing techniques are applied afterwards, such as motion by mean curvature [17] to eliminate any further noise. In this work, we used MRI data of a much lower resolution than the visual human dataset. Also, the segmentation process we used is automatic and therefore needs a more rigorous repair process.

Blemker and Delp [18] used an MRI dataset to create simulations of several sets of muscles to study the variation in moment arms across fibers within a muscle and later to predict *rectus femoris* and *vastus intermedius* fiber excursions [19]. They used MRI of live and cadaver specimen, and manually segmented the areas of interest. They also created a fiber map for each muscle of interest, based on template fiber geometries morphed to each muscles target fiber geometry. They used a manual segmentation process instead of an automatic segmentation as is used in this work. While this method of segmentation provides a surface mesh of higher quality, the method is not automatic. The segmentation method used in this project comes from Schmid and Magnenat-Thalmann [20]. This method is based on an earlier work of Gilles et al. [21], who proposed a registration and segmentation method for clinical MRI datasets based on discrete deformable models.

The main contribution of this study to the field of musculoskeletal and FE simulations is the semiautomated pipeline, that resolves data inaccuracies, such as muscle overlaps and self-intersecting muscles. Further developments of the framework will eventually allow for the connection between musculoskeletal and FE simulations, and thereby building a step towards a unified platform for subject-specific motion analysis.

This paper is structured as follows. In Section 3 we detail the main features of our framework manipulating the data sources and preparing the final FE simulation. In Section 4 and section 5 we respectively discuss the limitations of the framework and summarize the findings of the study and suggesting possible research directions.

3 Practical framework

3.1 Overview

Our framework consists of several key steps (see Figure 1). First the lower limbs of a subject are scanned in an MRI machine. The acquisition protocol used in our experiments is presented in Appendix B. Then the lower limbs are segmented using the algorithm of Schmid and Magnenat-Thalmann [20]. The segmentation algorithm produces closed surfaces of muscles and bones and the attachment sites where the muscle connects to the bones (Section 3.2).

The closed surfaces from the segmentation result cannot be converted directly to volume meshes as the segmented data may contain segmentation artifacts and noise (Sections 3.3, 3.5 and 3.6) or be incomplete (Section 3.4). These inconsistencies are first solved before the surfaces can be given to the mesh generator (Section 3.7). To complete the FE ready meshes, we finally create attachment specifications that can be used by a FE solver to constraint the movement of the soft tissues (Section 3.8).

3.2 Segmenting MR images

To illustrate the features of our framework, we use a segmented dataset resulting from a template method that uses a minimal energy optimization to fit a template muscle or bone to the acquired MRI data [20]. Since the method is dependent on forces, balancing the weights of the non-penetration constraints and the other constraints can be a difficult if not impossible task. Therefore, the resulting surfaces may suffer from intersections between surfaces and self-intersections, which we resolve in this study. Since the density of the vertices of the shapes is uniformly spread, the resulting shapes have nearly the same property. The segmentation algorithm also includes tendon and attachment locations specified by vertex indices in the resulting muscle meshes.

The segmentation algorithm produces surfaces of bones and muscles. Surfaces are represented as a list of vertices $V = \{v_i : 1 \leq i \leq n_V\}$ and a list of faces $F = \{f_k : 1 \leq k \leq n_F\}$, each face $f_k = (i_1^k, \ldots, i_{n_{f_k}}^k)$ consisting of a sequence of indices in the vertex list. A surface $S = \{V, F\}$ is a pair of a vertex list combined with a face list. All surfaces are consisting of only triangles, so each face has always three vertices $f_k = (i_1^k, i_2^k, i_3^k)$.

An attachment site A is defined in the segmentation result as indices in a corresponding vertex list V in the order they are defined in the muscle files: $A = \{i_1, \ldots, i_{n_A}\}$.

Since some tasks in the pipeline change the amount and ordering of the vertices in the data files, we process the attachment sites to be index-invariant by defining the attachment areas by geometry. This is explained in Section 3.8. The tendons are defined in the same way as the attachments, and for those we also need to convert the indices to a geometric representation. This is the protocol that we used to specify attachments and tendons but any equivalent representation is possible and can be used with minimal adjustments to our algorithms.

3.3 Smoothing the surfaces

First, we have to apply a smoothing method to remove the high frequency artifacts from the segmentation algorithm. In this work we applied the method of Taubin [22], since it is fitted to our type of biological surfaces, which are closed surfaces with a nearly uniform vertex distribution. This method does not introduce shrinking and is also the most suitable option with regards to implementation complexity.

The algorithm is a low-pass filter, with three parameters: λ, μ and the iteration count N. The low-pass filter properties are the pass-band frequency k_{PB}, the pass-band ripple δ_{PB}, the stop-band frequency k_{SB}, and the stop-band ripple δ_{SB}. The parameters are related to the low-pass filter properties as follows.

$$\lambda < \frac{1}{k_{SB}} \tag{1}$$

$$\frac{1}{\lambda} + \frac{1}{\mu} = k_{PB} \tag{2}$$

$$\left[(\lambda - \mu)^2/(-4\lambda\mu)\right]^N < 1 + \delta_{PB} \tag{3}$$

$$\left[(1 - \lambda k_{SB})(1 - \mu k_{SB})\right]^N < \delta_{SB} \tag{4}$$

Figure 2 shows the result of the operation on the *gastrocnemius* muscle using the parameters given in Appendix B.

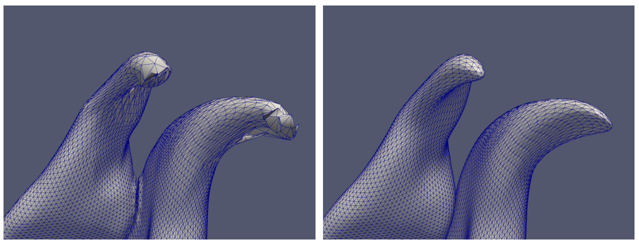
Fig. 2 Smoothing results. Left is before and right is after the smoothing operation.

3.4 Generating missing tendons

It is usual that after segmentation of the anatomical entities from MRI some tendons are not present or that tendons are present but their attachment to the muscle is not defined. Due to MRI acquisition protocols, location and shape of muscles, typical missing tendons on the lower body are observed for the *gastrocnemius* muscle and the *soleus* muscle, which are attached to the femur, tibia or fibula at the top of the muscle and are missing their connection to the foot. Since these muscles are valuable for a simulation of the lower body, we developed an algorithm that automatically edits the surface mesh to incorporate a tendon structure, including an attachment and tendon specification expected further along in the pipeline.

The tendon generator gets a list of muscles that do not have tendons. For these muscles, the 'bottom' part of the muscle is detected, and is extruded towards the foot. In our example muscles, the bottom direction is the negative direction along the superior-inferior axis, which in our dataset is the $-z$ direction. It should be adapted to other scanning configurations, for example by detecting the principal axis of the tibia bone.

As a first step of the algorithm, the z-value $v^z_{foot_{max}}$ of the top vertex of the foot S_{foot} is determined so that $v^z_{foot_{max}} > v^z_i, v_i \in V_{foot}$. For each muscle S_{muscle}, the faces on the bottom part are recursively traversed, starting with the faces connected to the vertex $v^z_{muscle_{min}}$ with the lowest z-value, creating a set F_{bottom} of faces belonging to the bottom part of the muscle. The set F_{bottom} is defined as follows.

$$\begin{aligned}
F_{bottom_0} &= \{f_k : v^z_{min} \in f_k\} \\
F_{bottom_{t+1}} &= F_{bottom_t} \cup \{f_k : f_k^{af} \cap F_{bottom_t} \\
&\quad \neq \emptyset, angle(f_k) < \theta_{muscle}\} \\
angle(f_k) &= \tfrac{\cos^{-1}(fn_k \cdot -\mathbf{z})}{180\pi}
\end{aligned}$$

Here f_k^{af} is the set of faces adjacent to f_k, $angle(f_k)$ the function providing the angle between a face and the xy-plane, fn_k is the face-normal of face f_k and iterations of F_{bottom} are denoted as $F_{bottom_\#}$. Summarized, we collect the set F_{bottom} of adjacent faces that have

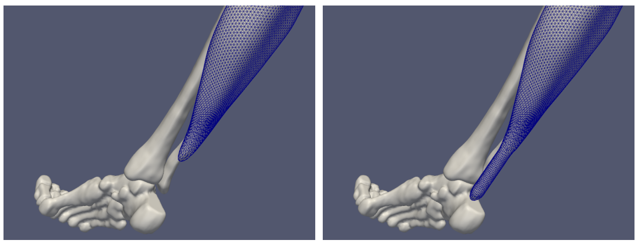
Fig. 3 Left: The *soleus* muscle without tendons connecting to the foot. Right: The *soleus* muscle with an attachment to the foot generated automatically.

an angle with the xy-plane lower than θ_{muscle}. For our missing tendons we used a value θ_{muscle} of 60 degrees.

After determining the faces that comprise the bottom of the muscle, we start the extrusion. For each face $f_k \in F_{bottom}$, we set the z-value of each of its vertices to $v_i'^z = v_i^z + (v^z_{foot_{max}} - v^z_{muscle_{min}})$. Then we determine the edges E_{border} on the border of F_{bottom}. The vertices in the border edge set E_{border} are duplicated and are set to their original position. The gap between the original border and the duplicated border is filled with regular spaced vertices and corresponding faces F_{gap}. All the faces from F_{bottom} and the newly added faces F_{gap} are combined into a new set $F_{tendon} = F_{bottom} \cup F_{gap}$, and are together locally smoothed according to the method described in Section 3.3. In Figure 3 it is shown how the newly generated tendon for the *soleus* muscle compares to the version without tendon.

The newly generated piece of muscle is also saved as being tendon material by immediately generating a tendon convex hull for the new vertices, as described in Section 3.8.

This correction is not an ideal solution but the main objective is to attach the muscle to the bone. Nonetheless, it is not advised to rely on simulations performed on an area of interest where tendons have been generated this way. In our experiments on the knee joint, only tendons near the foot needed to be generated, so we are confident that our simulations are relevant in regards to this issue.

3.5 Resolving self-intersections

The surface data provided by the segmentation algorithm can contain self-intersections. The intersected faces are inverted parts of the volume that the closed surface describes. As these parts occur from bad segmentation, our goal is not to untangle them as it would typically be done in cloth simulation [23]. Indeed, that would add information to the surface that should not be present in the dataset, and it would be generated without purpose. The only semantically correct way of resolving them is by removing them in a way that the final surface is still closed. We solved this issue by adapting the algorithm proposed by Jung et al. [24] originally designed to remove self-intersections from a raw offset triangular mesh. The algorithm uses a region growing approach, keeping a list of valid triangles. Starting with an initial seed triangle, the algorithm grows the valid region to neighboring triangles until it reaches triangles with self-intersection. Then the region growing process crosses over the self-intersection and moves to the adjacent valid triangle. Therefore the region growing traverses valid triangles and intersecting triangles adjacent to valid triangles only. The region growing process is complete when no more faces have to be added to the valid region. We will describe here the adaptations we made to the algorithm of Jung et al. to fit it to our purpose.

As in the original algorithm, we remove faces that have edges smaller that a predefined ε_e and reconfigure faces that have angles smaller than ε_α. All faces in each surface are checked for these two properties and solved accordingly. For faces where an edge $e_p = (v_i, v_j)$ is smaller than ε_e, that edge is collapsed by setting $v_i = (v_i + v_j)/2$ and v_j is removed. Then the faces containing v_j are updated to refer to v_i. But after applying this operation a number of times, the geometry might contain some new topological problems. The surface can contain two triangles that are opposite to each other. We have adapted the algorithm so that these faces are detected and removed, which is a valid operation, since it is a part of the surface that defines no volume. For a triangle f_k that has an angle smaller than ε_α, we do not remove any edges, but we change its configuration. First we determine the longest edge $e_u = (v_i, v_j)$, and find the triangle f_m sharing that edge. We then have $f_k = \{v_h, v_i, v_j\}$ and $f_m = \{v_g, v_j, v_i\}$. We now swap the edge e_u, such that $f_k = \{v_h, v_i, v_g\}$ and $f_m = \{v_g, v_j, v_h\}$.

Since the method originally was developed for raw offset triangular meshes, selecting a seed triangle was easily determined by calculating the convex hull $VC \subset V$, and picking any triangle $f \in F$ which has at least vertex $v_i \in VC$. In our case however, triangles taking part in the convex hull can also be invalid, as is the case in the *vastus intermedius* muscle. For this reason, we added a constraint to the seed triangle selection. First, we calculate the center of mass of the vertices of the surface, by a simple averaging of the vertex positions.

$$p_{com} = \frac{\sum_{v_i \in V} v_i}{|V|} \tag{5}$$

Since the surface meshes in this work are of approximately of uniform density (see Section 3.2), we have a rough estimate of the center p_{com} of the object. We use this value to determine if a vertex-normal vn_i of a vertex on the convex hull is face outward of the object or inward. The vertex normal is based on the average face-normals of the faces surrounding v_i.

$$v_i^{af} = \{k : i \in f_k\}$$
$$vn_i = \frac{\sum_{f_k \in v_i^{af}} fn_k}{|v_i^{af}|} \tag{6}$$

Here, v_i^{af} are the faces that contain v_i. We then compose the following equation that gives a measure of how much a vertex-normal is facing outwards.

$$\mathbf{u} = (v_i - p_{com})$$
$$d_i = vn_i \cdot \mathbf{u} \tag{7}$$

We calculate the dot product of the vector from the center of the object to the vertex v_i with the vertex normal vn_i. The value d_i scales from 1 to -1, where 1 means vn_i is pointing straightly outwardly and -1 means vn_i is pointing exactly to the center p_{com}. We calculate this value for all $v_i \in VC$, and choose the vertex v_{opt} with the highest d-value. From this vertex v_{opt}, we pick a random face $f_k \in v_{opt}^{af}$ to be the seed triangle, thus ensured to be a valid triangle.

3.6 Resolving surface overlaps

The surfaces resulting from segmentation algorithms may also contain surfaces that intersect each other. As illustrated in Figure 4, these intersections can be quite severe.

Exact subtractions of surfaces can be made using Boolean operators. We used a combination of an existing implementation of Boolean operators and an adaptation of the self-intersection removal method.

We used the Boolean operator implemented in the Carve constructive solid geometry library (CSG) [25], since it is efficient and the main freely available CSG library. It is able to perform Boolean operations such as union, difference and subtraction. We use the subtraction operator to resolve the surface overlaps. The library can take two polygon meshes, S_x and S_y, and produces a mesh S'_x of which the part intersecting with S_y is removed.

The self-intersection removal algorithm presented in Section 3.5 can also be employed as a subtraction algorithm. The method provides an intersection detection mechanism from which we can reuse nearly all components. The method is described as follows. We have surface S_x and surface S_y, and we want to calculate $S_x - S_y$. We merge the two surfaces into one data structure S_z but we invert the face normal directions of S_y.

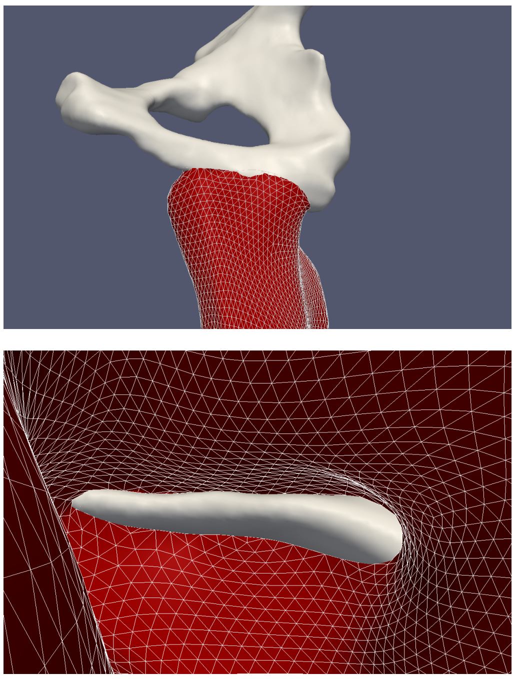
Fig. 4 Top: View of the *adductor magnus* intersecting with the pelvis bone surface. Bottom: Same intersection as seen from the 'inside' of the *adductor magnus* muscle geometry.

We then have one surface S_z, on which we can apply the self-intersection removal algorithm with a seed triangle from S_x. The algorithm will detect the intersections between the two surfaces, and will regard the part of S_y as a self-intersecting part of S_x.

The method gives the same quality results as the Carve library, but is in some cases less stable (when dealing with degenerate cases) and our implementation is less optimized than the Carve implementation. Therefore we primarily use the Carve library in the final algorithm. In the situations where Carve cannot find a correct solution, we automatically fall back to the subtraction by self-intersection removal method. The Carve library fails to create a subtraction result when one of the input surfaces has one or more holes.

Figure 5 shows an example of the subtraction of a bone (femur) from a muscle (*vastus medialis*).

The order in which the operations are executed determines the resulting mesh configuration. The object that is processed first will most likely lose the most of its geometry, and will have no influence on the other objects. The object that is processed last will not be influenced at all by other objects. We chose the bones to always be processed last, since their segmentation quality is the highest. The bones are relatively easy to distinguish on an MRI scan, therefore the segmentation result is the most reliable and less prone to segmentation errors. Since the bones are processed lastly, they

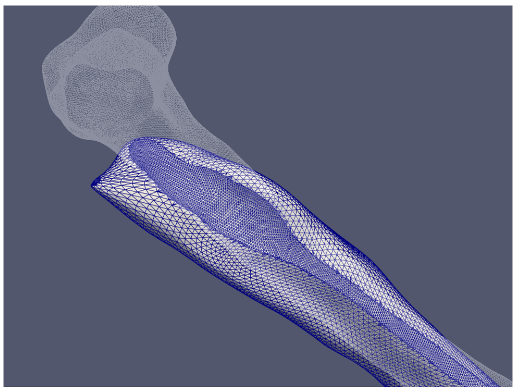
Fig. 5 The *vastus medialis* after the femur has been subtracted. The femur is represented as the transparent wireframe surface.

will never be influenced by muscle objects. The muscles and bones are among themselves ordered in descending order of volume. This way the surfaces that are smaller will be processed later, meaning they will be the last to be subtracted from. This is appropriate, since the impact of changes on small objects is usually higher. We use a simple bounding box method to predetermine if a subtraction should be executed to reduce the number of subtractions.

In our experiments we have tested the above algorithm with few muscles and bones. We believe that most of the interaction problems that will occur when attempting to model more anatomically realistic examples will be resolved. Indeed, the management of numerous interactions between the different anatomical entities during the FE simulation will be taken care of by the solver (but increased complexity by adding entities means significant increase in solver time). Nevertheless, we can imagine very specific cases where entities have been so badly segmented (or segmented on very noisy MRI) that our solving technique will not produce realistic results (e.g. small muscle completely removed or cut into several parts).

During this research, we tried several other methods of approaching the overlap problem which are shortly described in Appendix A along with their advantages and limitations regarding our goal.

3.7 Generating volume mesh

Since the FE simulation is based on a volumetric representation of elements, we need to convert the triangular surface representation of our objects to a tetrahedral volume representation. We used the algorithm provided by the CGAL library for this operation [26–28].

The volume mesh produced by the algorithm is a set of vertices $V = \{v_i : 1 \leq i \leq n_V\}$, and a set of tetrahedral cells $C = \{c_k : 1 \leq k \leq n_C\}$. Each cell $c_k = (i_1^k, i_2^k, i_3^k, i_4^k)$ consists of a sequence of exactly four indices in the vertex list, since the algorithm only produces tetrahedral cells. A volume mesh $M = \{V, C\}$ is a vertex list combined with a cell list.

We apply cleanup steps after the mesh generation to remove 'loose' tetrahedra. For example, tetrahedra that have only one connection to the rest of the mesh are removed as these tetrahedra can lag behind because of inertia and can produce oscillations as they do not move in sync with the rest of the tissue they belong to. These oscillations can make the simulation unstable and can prevent the solver of finding solutions. If c_i^{ac} represents the cell indices of cells surrounding the cell c_i, we can define the loose cells as $CL = \{c_i : |c_i^{ac}| \leq 1\}$. After selecting these elements, we remove them from the volume mesh.

3.8 Creating attachment and tendon convex hulls

As explained in Section 3.2, the attachments and tendons are specified as indices in vertex lists that are saved in a surface mesh file. In the previous stages in the pipeline, these files are changed topologically, and finally converted to volume meshes. To preserve the attachment data, we convert the index-based representation to a geometric representation.

The first step is to convert the index-based representation $A = \{i_1, \ldots, i_{n_A}\}$ to a position-based representation $AP = \{p_1, \ldots, p_{n_A}\}$. This is done simply by looking up the vertices in a compatible vertex list V. We can use any vertex list, as long as it has the same vertex ordering as the original vertex list from the segmentation algorithm. Therefore, we use the vertex list that results from the smoothing stage as described in Section 3.3. From AP, we can calculate a convex hull, to obtain the area where the attachment is active. The convex hull is a surface $AS = \{V, F\}$ on which we can apply standard surface operations. We use the convex hull algorithm from CGAL to implement this conversion [29]. To account for possible rounding errors, we grow the surface slightly using a raw offset operation: $v_i' = v_i + \lambda vn_i$, where v_i' is the new vertex position of v_i grown in the vertex normal vn_i direction, and λ is a growth factor.

Figure 6 shows resulting attachment and tendon regions for the *vastus lateralis* muscle. To calculate the tendon regions from the specification, we use the exact same process as for the attachments.

As a final step, we convert the attachment and tendon regions into indices in the volume meshes $M = \{V, C\}$. We check for each vertex $v_i \in V$ if it lies inside

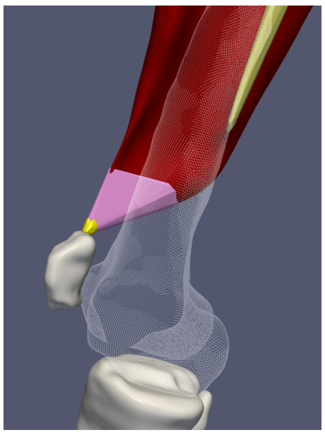

Fig. 6 *Vastus lateralis* muscle in red, the tendon convex hulls are purple, the attachment convex hulls are yellow and the femur bone is transparent.

the attachment surface AS. We do this check using the Carve library [25]. The new attachment specification can be described as follows.

$$AM = \{i : v_i \in V_M, inside(v_i, AS)\} \quad (8)$$

Here, AM is the set of indices that point to vertices in the volume mesh M. This new representation can be used directly in the next features of the framework.

4 Discussion

The framework described in Section 3 produces a number of data units that have to be combined to be ready to be used by a finite element solver. In Appendix B we indicate the most relevant parameters of the framework. It is also to remember that if one wants to use the motion from a musculoskeletal simulation platform such as OpenSim, one needs to convert the joint motion from the platform system to a transformation (rotation and translation) of the bones in the finite element simulation. Fortunately, usually muscle models and MRI data both contain a set of markers designed to be located on common anatomical landmarks.

As gathering subject-specific data from versatile sources is a tedious task and executing the pipeline (from segmentation to FE simulation) is time-consuming, we have presented results for only one subject, with a limited set of muscles. But we are confident that our framework is robust and transferable to a large variety of datasets of the same nature.

The objective of our framework is to allow for the direct connection of a subject-specific motion with the FE simulation of the soft tissue. Eventually, the framework will also take into account the muscle activations determined by musculoskeletal analysis of the same motion. Active muscle contraction could then be used to deform the muscle shape in a more realistic way. The bones could be either actively moved by constraints in the FE solver, or passively moved by the muscle deformations. In the latter case one of the goals of the simulation would be for the resulting motion to match the original recorded motion as closely as possible. Ultimately we can imagine a dual coupling approach, where the capabilities of the musculoskeletal platform are extended by taking into account subject-specific deformation of muscles.

Our dataset did not contain the subject-specific fiber information from the MRI. Indeed, we are dealing here with quite low-resolution images, producing the artifacts previously presented. Nevertheless, this information could eventually be approximated from muscle models commonly used in musculoskeletal simulation (*e.g.* by using pennation angles or template fiber maps). Anisotropic muscle and tendon materials could then be used to obtain more accurate deformations.

Materials which support fiber directions have been proposed in the literature and even implemented in FE solvers such as FEBio [30,31], but need a significant elaborate configuration. Together with muscle activation these materials would greatly contribute to the accuracy of the simulation. The disadvantage is of course the increase in parameters, and most likely the manual configuration of fiber directions for each muscle.

Alternative models could have been obtained on a dataset of higher quality. Higher resolution of MRI data with less noise can be obtained by using a scanner with a stronger magnetic field. The resulting segmentation would have less artifacts and the resulting FE ready meshes would be more accurate. In addition, if dynamic MR images of a leg in motion could be acquired, segmented bones and soft tissue could be compared to the results of FE simulations performing the same motion. This way, the deformation of the muscles in the FE simulation can be compared to a real world example, which would validate the deformation of the muscles, thereby confirming or invalidating the parameters of the simulation. This would serve as a fine tuning process of the parameters of the materials as well.

5 Conclusion

We have presented a semi-automated practical framework that resolves data inaccuracies dedicated to subject-specific soft tissue deformation. Segmentation artifacts from low MRI quality are smoothed out, self-intersections are removed and inter-object overlaps are resolved. The framework has the ability to semantically clean up the data, by removing objects from specific data files and generating new tendon geometry in case of missing tendons. The volume mesh generation is also performed automatically but should be configured to the user's need, depending on the user's preferred detail and performance of the simulation. The attachment sites and tendon areas are automatically added to the output of the pipeline: an FE model that has the volume data, material and attachment specifications.

We hope to propose a step towards a unified platform for neuromuscular and finite element simulation. Ultimately the framework will be able to extend the capabilities of musculoskeletal platforms by taking generated subject-specific motions to drive MRI-based finite element simulations. Future research could also create a reverse connection, by using and visualizing FE simulations in, for example, OpenSim such as initiated by Pronost et al. [32].

Acknowledgements This work was initiated by the European Marie Curie Program under the 3D ANATOMICAL HUMAN project (MRTN-CT-2006-035763). The authors would like to acknowledge Jerome Schmid, Anders Sandholm, Nadia Magnegnat-Thalmann, Daniel Thalmann and Arjan Egges for their assistance in this work.

References

1. D. Lee, M. Glueck, A. Khan, E. Fiume, and K. Jackson. Modeling and simulation of skeletal muscle for computer graphics: A survey. *Foundations and Trends in Computer Graphics and Vision*, 7(4):229–276, 2012.
2. J. Teran, E. Sifakis, S. Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw. Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):317–328, May 2005.
3. S.L. Delp, F.C. Anderson, A.S. Arnold, P. Loan, A. Habib, C.T. John, E. Guendelman, and D.G. Thelen. OpenSim: Open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, 54(11):1940–1950, 2007.
4. M. Damsgaard, J. Rasmussen, S.T. Christensen, E. Surma, and M. de Zee. Analysis of musculoskeletal systems in the Anybody modeling system. *Simul. Model. Pract. Theory*, 14:1100–1111, 2006.
5. S. L. Delp, J. P. Loan, M. G. Hoy, F. E. Zajac, E. L. Topp, and J. M. Rosen. An interactive graphics-based model of the lower extremity to study orthopaedic surgical procedures. *IEEE Trans. Biomed. Eng.*, 37(8):757–767, 1990.
6. J.M. Wang, S.R. Hamner, S.L. Delp, and V. Koltun. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Transactions on Graphics*, 31(4), 2012.
7. A.W.J. Gielen, C.W.J. Oomens, P.H.M. Bovendeerd, T. Arts, and J.D. Janssen. A finite element approach for skeletal muscle using a distributed moment model of contraction. *Computer Methods in Biomechanics and Biomedical Engineering*, 3(3):231–244, 2000.
8. T. Johansson, P. Meier, and R. Blickhan. A finite-element model for the mechanical analysis of skeletal muscles. *Journal of Theoretical Biology*, 206(1):131–149, 2000.
9. M. Kojic, S. Mijailovic, and N. Zdravkovic. Modelling of muscle behaviour by the finite element method using Hill's three-element model. *International Journal for Numerical Methods in Engineering*, 43(5):941–953, 1998.
10. T.R. Jenkyn, B. Koopman, P. Huijing, R.L. Lieber, and K.R. Kaufman. Finite element model of intramuscular pressure during isometric contraction of skeletal muscle. *Physics in Medicine and Biology*, 47:4043, 2002.
11. U.S. National Library of Medicine. The visible human project. http://www.nlm.nih.gov/research/visible/.
12. F. Dong, G.J. Clapworthy, M.A. Krokos, and J. Yao. An anatomy-based approach to human muscle modeling and deformation. *IEEE Trans. Vis. Comput. Graph.*, 8(2):154–170, 2002.
13. T. Shinar, C. Schroeder, and R. Fedkiw. Two-way coupling of rigid and deformable bodies. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, volume edited by D. James and M. Gross, pages 95–103, 2008.
14. J. Kim and N.S. Pollard. Fast simulation of skeleton-driven deformable body characters. *ACM Transactions on Graphics*, 30(5):1–19, October 2011.
15. I. Stavness, J.E. Lloyd, Y. Payan, and S. Fels. Coupled hardsoft tissue simulation with contact and constraints applied to jawtonguehyoid dynamics. *International Journal for Numerical Methods in Biomedical Engineering*, 27(3):367–390, 2011.
16. J.A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences of the United States of America*, 93(4):1591–1595, 1996.
17. S. Osher and R.P. Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153. Springer Verlag, 2003.
18. S.S. Blemker and S.L. Delp. Three-dimensional representation of complex muscle architectures and geometries. *Annals of Biomedical Engineering*, 33(5):661–673, 2005.
19. S.S. Blemker and S.L. Delp. Rectus femoris and vastus intermedius fiber excursions predicted by three-dimensional muscle models. *Journal of Biomechanics*, 39(8):1383–1391, 2006.
20. J. Schmid and N. Magnenat-Thalmann. MRI bone segmentation using deformable models and shape priors. In *MICCAI*, volume LNCS 5241, pages 119–126, 2008.
21. B. Gilles, L. Moccozet, and N. Magnenat-Thalmann. Anatomical modeling of the musculoskeletal system from MRI. *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 289–296, 2006.
22. G. Taubin. Curve and surface smoothing without shrinkage. In *Fifth International Conference on Computer Vision*, pages 852–857, 1995.
23. D. Baraff, A. Witkin, and M. Kass. Untangling cloth. *ACM Transaction on Graphics*, 22(3):862–870, July 2003.
24. W. Jung, H. Shin, and B.K. Choi. Self-intersection removal in triangular mesh offsetting. *Computer-Aided Design and Applications*, 1(1-4):477–484, 2004.
25. Carve, constructive solid geometry library. http://code.google.com/p/carve.
26. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.
27. L. Rineau, S. Tayeb, and M. Yvinec. 3D mesh generation. In *CGAL User and Reference Manual*. 2009.
28. S. Oudot, L. Rineau, and M. Yvinec. Meshing volumes bounded by smooth surfaces. In *Proceedings of the 14th International Meshing Roundtable*, pages 203–219, 2005.

29. S. Hert and S. Schirra. 3D convex hulls. In *CGAL User and Reference Manual*. 2009.
30. FEBio, Finite Elements for Biomechanics. http://mrl.sci.utah.edu/software.php.
31. S.A. Maas, B.J. Ellis, D.S. Rawlins, and J.A. Weiss. A comparison of FEBio, ABAQUS, and NIKE3D results for a suite of verification problems. 2009.
32. N. Pronost, A. Sandholm, and D. Thalmann. A visualization framework for the analysis of neuromuscular simulations. *The Visual Computer*, 27:109–119, 2011.
33. J.L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.
34. O.K.C. Au, C.L. Tai, H.K. Chu, D. Cohen-Or, and T.Y. Lee. Skeleton extraction by mesh contraction. *ACM Transaction on Graphics*, 27(3):1–10, 2008.

Appendix A

A.1 Push-based method

During this research, we tried several methods of approaching the overlap problem. We first developed a method where two objects would push each other away. The reason for using a push-based method is that the topology of the object remains intact, and the attachment and tendon information are transformed together with the surface. The surfaces that have smaller volumes have a higher priority, meaning they will be the last to be deformed, because their changes will most likely have the most impact on the shape of the object. The bones will have the highest priority since they have the best segmentation quality. First we apply an iterative smoothing algorithm on the pushed surface to represent the general trend that the surface has. We then use this information to determine the direction where to each overlapping vertex should move when pushed away by the other surface. As we also want to take into account the geometry of the pushing surface, this direction is combined with the average vertex normal of the closest vertices in the pushing surface. This lookup is done using a kd-tree, for which the implementation from the CGAL library is used [26,33]. We take the 10 closest vertices, calculate the vertex normals of those and average them. This gives us a second pushing direction vector which is combined to the smoothed direction with an equal weight. A vertex is pushed away in that direction until it does not lie inside the other surface anymore. Unfortunately, this method fails if a surface lies too far inside another one, because pushing directions cannot be accurately determined anymore. Furthermore, the convergence of this kind of technique will be difficult to ensure when the dataset is made of many objects *e.g.* with all muscles, fat-tissue and skin layers.

A.2 Shrink-based method

We also looked at methods that can make two surfaces push each other away by both moving vertices to a shrunk copy of the surface. For this, we need an algorithm that can give for a surface S a shrunk version S'. Shrinking an object iteratively gives a sequence of shrunk surfaces $S^1 \dots S^n$. Similarly, each vertex $v_i \in V$, has a sequence of shrunk positions $v_i^1 \dots v_i^n$. By applying the shrinking algorithm iteratively on the overlapping surfaces, they will naturally move apart. We tried the following two shrinking algorithms.

Shrinking by smoothing. We used a shrinking algorithm based on standard Laplacian smoothing [22]. Each vertex is moved in the average direction of its neighbors.

$$v_i^{t+1} = v_i^t + \lambda \frac{\sum_{j \in v_i^{av}} v_j^t - v_i^t}{|v_i^{av}|} \tag{9}$$

Here λ is the scaling factor of the displacement done at each shrinking step. This method produces smooth shrinking, and can be done in arbitrarily small steps. The problem of this method is that globally, the surface will shrink, but locally, the surface can expand. Specifically, concave parts of the surface will expand, and convex parts will shrink.

Shrinking to skeleton. The second shrinking algorithm we developed moves the vertices of the surface to a pre-calculated 'skeleton', which represents the global minimal structure of an object. We determine this skeleton using the method of Au et al. [34]. The algorithm takes a surface $S = \{V, F\}$ and produces a curve-skeleton $K = (U, E)$ with skeleton nodes U and edges E, where $U = \{u_1, u_2, \dots, u_{n_U}\}$ are the node positions. The algorithm also produces a mapping $coll(v_i) = u_k$ from vertices to skeletal nodes, which gives for each vertex v_i the node u_k to which it was collapsed in the skeleton. The shrinking algorithm we developed moves the vertices towards an attraction point on the skeleton. The attraction points are dynamically determined by the mapping $coll(v_i)$, where the influence of the parent node of each vertex increases as the vertex moves closer to the skeleton. The smoothness of the shrinking could be improved by a different strategy of moving toward the skeleton. Unfortunately, the algorithm has a bigger defect: the algorithm produces for some muscles a skeleton that lies outside of the surface shape, which is not tolerable for our application, since the vertices can never go outside of the original shape.

Appendix B

Static MRI scans of the whole lower limb have been performed. The subject was lying in a resting pose. In close collaboration with radiologists, adequate protocols for the imaging of soft and bony tissues was defined. The following protocol was used: 1.5T Philips Medical Systems machine, axial 2D T1 Turbo Spin Echo, TR/TE = 578/18 ms, FOV/FA = 40 cm/90°, matrix/resolution = 512x512/0.78x0.78 mm, thickness: from 2 mm (near joints) to 10 mm (long bones).

Not all algorithms in the pipeline can be used without configuration. We describe hereafter the most significant parameters used in the pipeline. The smoothing algorithm (Section 3.3) has three parameters: λ factor, μ factor and number N. The surfaces in our dataset are smoothed with the values $\lambda = 0.33$, $\mu = -0.331$ and $N = 1000$ iterations.

In the tendon generation step (Section 3.4), we add regularly spaced vertices to fill the gap between the end of the tendon and the start of the muscle. The space between these rows of vertices is set to 2 mm.

The degenerate triangle method (Section 3.5), needs two ϵ values, ϵ_e and ϵ_α, that respectively indicate the minimal edge length and minimal angle. In the experiment, we used $\epsilon_e = 0.5$ mm and $\epsilon_\alpha = 0.01$ degrees.

The overlap solving algorithm (Section 3.6) creates a raw-offset of the surface that will be subtracted. The parameter λ indicating the distance of the offset is set to $\lambda = 1$ mm.

The volume mesh generation step has several parameters determining the final density and accuracy of the output mesh. Table 1 lists the parameters used on our dataset. For both muscle and bone objects, we choose the same values for angular bound and radius-edge bound. This is because these values do not influence the amount of elements created in the output mesh, but only influence the runtime of the mesh generation. Facet angular bound is set to 30 degrees because the CGAL algorithm will guarantee a solution for this bound. The radius-edge ratio is set to 1.25, because we want to avoid unbalanced tetrahedra. For the radius bounds, we choose for the muscle a lower value since it will deform during the simulation and therefore needs a higher resolution. The bones are not going to deform so the size of the tetrahedra inside of the bones is not important, hence the high value.

Parameter name	Muscle	Bone
facet angle (deg)	30.0	30.0
facet size (mm)	5.0	15.0
facet distance (mm)	0.5	0.75
cell radius-edge ratio	1.25	4.0
cell size (mm)	3	100

Table 1 CGAL mesh generation parameters