A scalable geometrical model for musculotendon units

Francis Laclé[1], Nicolas Pronost[1,2]

[1] Department of Information and Computing Sciences, Utrecht University

Princetonplein 5, 3584 CC Utrecht, The Netherlands

[2] Université de Lyon, CNRS, Université Claude Bernard Lyon 1, LIRIS

8 boulevard Niels Bohr, 69622 Villeurbanne, France

Abstract

Physics-based simulation of systems such as virtual humans has benefited from recent advances in muscle actuation. However, to be manageable for motion controllers, muscles are usually solely represented by their action line, a polyline that does not include data on the tridimensional geometry of the muscle. This paper focuses on combining, by a controllable enhancement process, a functional and biomechanical model of musculotendon units with its high resolution geometrical counterpart. The method was developed in order to be invariant to spatial and polygonal configurations, and to be scalable in both longitudinal and latitudinal directions. Results with 48 musculotendon units for the lower body show a drop of 84% with respect to the number of vertices when compared to the high resolution model, while maintaining the functional information. A real-time simulation experiment resulted in a runtime of 135Hz.

Keywords: Multi-scale virtual human, Musculotendon unit modeling, Geometrical enhancement

1 Introduction

Physics-based animation aims at replicating real environments by modeling the physical laws and conditions that define it. This technique has freed animators from worrying about enforcing certain motion characteristics which come implicitly with the presence of physics, and has granted virtual characters with a freedom of motion, that is unrestricted but physically plausible. Character models can exhibit different levels of detail in terms of the skeleton, actuators and tissues. Due to the computational complexity, real-time systems have been limited to the use of simplified biomechanical models which usually represent musculotendon units (MTUs) as polylines without tridimensional geometry, called action lines [1, 2]. This simplification allows motion controllers to fully actuate a virtual character using activation signals that are transformed into muscle forces applied on the body parts. As the actual shape of the muscle is not used, the polyline routing of a MTU becomes of primary importance to accurately represent the direction of the force, and it has indeed been shown to play a crucial role in the control of such systems [3]. On the other hand, the active control of human motion from fiber contraction on surface or volumetric models is a very challenging and computationally intensive task [4] that is usually not compatible with real-time controls.

In this paper, we propose a modeling method that enhances an action line with a scalable geometrical counterpart. Our method makes a step towards coupling muscle-based motion control and muscle deformation, in order to improve motion realism as potentially representing much better the actual interactions in the human body, especially the change of direction of muscle force due to deformation. Indeed, the polyline could then be displaced according to deformations of the muscle meshes in order to optimize the action line routing, for instance when concerning muscle moment arms. That type of coupled approach is a newly research trend in computer animation where only little works have been proposed but showing promising results [2]. Our modeling method allows user-defined resolution of the muscle mesh, from the simple polyline to a high resolution surface mesh. Each level is constructed from points defined on the action line so that each level preserves at best the biomechanical properties of the action line. This would allow muscle-based motion controllers to use muscle deformations to configure the MTU routing online instead of using static polylines. We expect the use of our method to produce more realistic simulations of virtual characters, not limited to humans. In general, applications in real-time computer animation that use biomechanical models, such as wound or musculoskeletal injuries simulation, orthopedic training, and validation of musculoskeletal pathologies for animated characters can benefit from our method. The main contribution of this paper is a novel method to enhance the scalable geometry of anatomical meshes, that we demonstrate for the purpose of MTU modeling.

2 Related works

Polygonal muscle models for computer graphics and animation are visualized and controlled using different approaches to augment character movement and simulate musculoskeletal deformation [5]. One of the first approaches to incorporate muscles into this domain used a free-form deformation lattice to kinematically deform skin with underlying muscle [6]. Later, Thalmann et al. [7] used meta-balls to define an underlying musculature. At specific intermediate distances, along each bone segment, they create orthogonal cross sections to define the shape of the combined meta-balls. This is similar to the construction presented in our method, except here we create cross sections along the musculotendon action line obtained from a biomechanical model. The authors also used ray casting to extract contours in order to form the skin of the character, which is the basis of the technique used in our method to approximate features of high resolution meshes. The work from Wilhelms and van Gelder [8] was also an inspiration for the cylinder-based model presented here, as it brought with it practical real-time benefits such as the radii-based deformation.

Early works showed that the Finite Element and Finite Volume Method (FEM and FVM) could be combined with a classical biomechanical model for calculating muscle forces. This paved the way for a new generation of FEM and FVM-based approaches [9, 10]. While our approach does not rely on FEM, our match detection technique discussed in section 4.2.3 for sampling high resolution meshes is quite similar to the winding number concept proposed by Jacobson et al. [11]. While their method uses generalized winding numbers to segment inside space from the outside, our method samples a high resolution mesh from specific points along the biomechanical action line, rendering the use of ray casting as a more suitable technique for our purpose as it allows us to adaptively scale the geometry of the cylinder-based model. Berraren et al. [12] used volumetric meshes with a modified Hill's model for real-time deformation analysis where contractile muscle forces operate between adjacent nodes of the mesh. This is one of the few recent works that shows promising real-time performance, though the result included a single muscle. Another recent work is the very complete approach of Si et al. [2] that includes a muscle-based swimming controller, fluid and deformable body simulators in which MTU activation signals were used to drive muscle contraction. However, they had to operate modeling approximations to balance computational complexity, geometric resolution, biomechanical accuracy, and robustness of the simulation (e.g. tendons were not represented and the original action lines were used to calculate muscle lengths). By using our modeling method in complement to such an approach, one could use the scalability property to adapt to computational power and complexity of algorithms and use our deformed cylinder-based model to efficiently and accurately calculate muscle length.

Other works have created volumetric meshes by mapping a template hexahedral mesh to another target hexahedral mesh using projections and Delaunay triangulation [13]. As a target shape for each map, two dimensional outlines of cross sections, sampled with MR imaging were used and projected in three dimensions, which is slightly similar to the enhancement technique presented in section 4.2. While this approach could ultimately outperform classical Hill-type models, it is still considered early work that could be surpassed by more accurate real-time classical models, which were tested against recent established benchmarks [14].

More recently, Kohout and Kukačka proposed another template-based modeling method for fibrous muscles [15]. In this method, the 3D model is sliced using iso-surfaces on a scalar field and fibers are morphed within the contour of each slice. In our method we want to ensure that, at each scale of our muscle model, an inner point within a slice best fits the original action line in order to preserve its physiological properties (so that, to give an example, a unique motion controller can be used regardless of the scale). This is why a method that enhances the action line is necessary. In addition, as we use the via-points present in the biomechanical action line, our method allows for a better local resolution where the muscle has the more chance to deform, modeling at best the routing and therefore the change of direction of muscle force. This is also a reason why we preferred to use polylines, as given by an existing biomechanical model, rather than only origin and insertion areas that can be extracted from medical imaging. Lastly, our method provides a direct control of the resolution of the output mesh.

To improve realism of mass-spring systems applied to fiber structures, Sanchez et al. [16] stepped away from template-based approaches and proposed a new workflow for embedding subject-specific fiber fields in models of musculotendons. They show that incorporating this information into their models led to a 10%-20% difference in predicting net muscle forces of specific patients. As these last methods and others such as Tan et al. [17] imply, the next frontier is modeling on the scale of fibers, however it will probably take years before this can be considered a possibility in the real-time domain. Meanwhile we aim at providing an automatic modeling method for real-time applications that is scalable from a high resolution cylinder-based model to a polyline-based MTU model.

The remainder of this paper is structured as follows. In sections 3 and 4, we introduce the input data sets and our scalable modeling method. Section 5 discusses the results and we give concluding remarks with hints at future applications and improvements in section 6.

3 The data sets

Our method aims to develop a scalable model that approximates anatomical accuracy and is capable of real-time or interactive performance. To help reach this goal, the following two criteria are considered. First, the model should include at least one anatomical model and one biomechanical model to help approximate anatomical and physical correctness. Second, the geometrical model has to be flexible, meaning that its scalability has to be controllable with respect to a given level of detail (LOD).

3.1 The biomechanical model

Biomechanical models of the muscle system include data that are relevant to the physical aspects of an actuable system such as positions of attachment points of the muscles, lengths and arrangement of muscle fibers. We used a biomechanically validated data set consisting of two times (left and right) 24 MTUs from literature [18] that was developed for the *OpenSim* [19] simulation tool.

The biomechanical data set is integrated automatically to the virtual character for animation by aligning the respective joint hierarchies. The virtual character is only defined by the joint hierarchy on which a surface skinned mesh is attached. Scaling ratios were calculated per body part and rotational alignments were done using the shortest rotational arc between the body parts in the respective hierarchies. The MTU properties (attachments position, maximum isometric force etc.) were scaled to the proportions of the virtual character (see Figure 1a).

3.2 The anatomical model

The Ultimate Human Model (UHM) data set [20] is recognized as one the most complete and accurate set of 3D models of the human musculoskeletal system. Despite being artistically edited, its accuracy comes from anatomical texts, papers, and custom magnetic resonance imaging scans (see Figure 1b).

The data set is unfortunately not well conditioned for a physics-based simulator and ex-

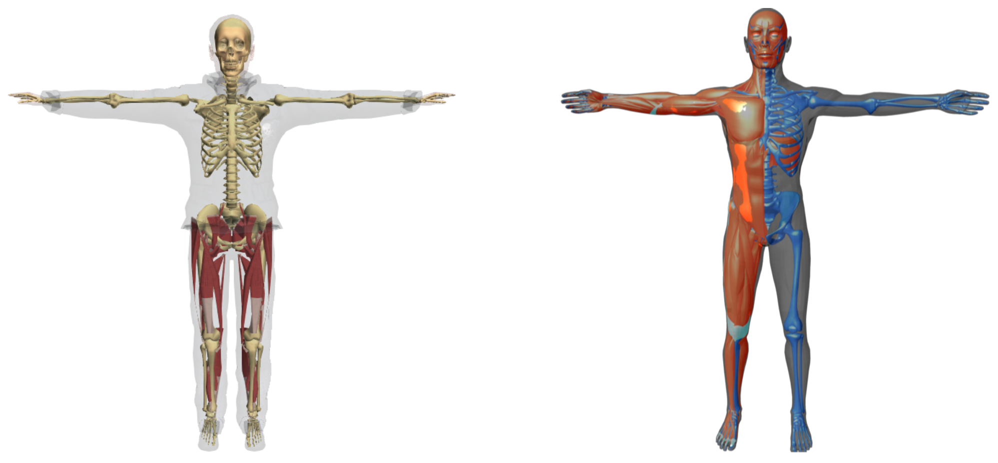


Figure 1: The virtual character model augmented with MTUs (red shapes) (a), and the artistic anatomical model of a human (b).

hibits open areas that can pose problems for volume discretization techniques. Therefore we manually processed the raw data using a 3D authoring software by closing the meshes and repositioning vertices from faces that exhibited self-intersections. Finally, some meshes were combined to fit a single MTU from the biomechanical model, while others were divided for cases where the biomechanical model represents single MTUs with multiple polylines.

To confirm our expectation about the necessity to create a scalable model, we have analyzed the shape of the meshes. We have calculated the isoperimetric quotient and performed convex-concave tests of sampled slices of each muscle mesh. The isoperimetric quotient, denoted by q, is used to calculate the ratio between the area a of an arbitrary slice U^{S_k} and the area of a circle with the same perimeter p as U^{S_k}. It is calculated as $q = \frac{4\pi a}{p^2}$. In order to compare the relation between the isoperimetric quotient and the perimeter, i.e. the elongation, we normalized the area of each slide to 1, giving us what we call a normalized perimeter. After sorting edges of each slice, the z-component of the cross product between pairs of consecutive edges (triplets of points) for each polygon is extracted. If all z-components are of the same sign then the slice is tagged as convex, else as concave. The slices were also visually inspected. We found that more convex slices appear when the normalized perimeter gets smaller and that 75% of the 72 slices tested are more inclined towards circular than elongated shapes (when $q > 0.5$). The result also indicates a concentration of shapes with an isoperimetric quotient of around 60%.

Although the results of the analysis show a circular tendency, highly concave slices are nevertheless present. While concavity is not an issue for real-time musculotendon models, more vertices are typically required to approximate concave shapes than convex ones and our goal is to minimize the number of vertices of the meshes since the processing and rendering time depends on the number of vertices. Taking the circular tendency and real-time performance into account, each vertex of the low dimensional model is repositioned at the surface of its corresponding high resolution mesh in order to approximate its shape. This is accomplished with ray-triangle intersection tests as described in section 4.2 with the goal of having the low resolution model remaining as a simple polygonal representation of its high resolution counterpart. Finally, because the method scales adaptively, a user can increase the resolution in case the final model requires a better approximation.

4 Scalable MTU modeling

4.1 The geometrical model

4.1.1 Action lines and via-points

The geometrical muscle model combines muscle and tendons into a single object. The initial shape of the MTU can be thought of as a closed cylinder with the starting point, the *origin point*, lying at the center of the top cap (see Figure 2a). This point is attached relative to a specific bone in the hierarchy, which can also contain other siblings representing other musculotendon objects. The end point, or *insertion point*, at the center of the bottom cap is attached relative to either the same or another bone in the hierarchy. This line connecting origin and insertion is referred to as the *action line* of the muscle. This is a polyline as most of the time, the origin is not directly connected to the insertion but indirectly through additional locations, called via-points. Not all MTUs present in the biomechanical data set contain via-points, such as the *gluteus medius*, *rectus femoris*, and *pectineus*. For such MTUs, we include additional via-points that will allow the model to approximate empirical muscle path more closely. In this paper, the following notation is used to define an improved action line A:

$$A = \left\{a_0, a_1, ..., a_{n-1} \mid a_k \in \mathbb{R}^3\right\}, |A| \geq 2 \tag{1}$$

where each a_k is a position vector representing coordinates in $\mathbb{R}^3$. The first and last element of A represent the origin and insertion points.

During animation, each a_k is updated kinematically with relation to the body part it is attached to, and consequently the shape of our geometrical model changes automatically. It is worth noting that current musculoskeletal simulators allow the addition and removal of points that are wrapping around solid primitive shapes such as cylinders and ellipsoids. These are not included in our method as our goal is to propose an action line enhanced with its geometrical counterpart. The deformation of the geometrical action line should then give us a more accurate estimation of the change of direction of muscle force than wrapping points.

4.1.2 Including tendons into the model

For our goal, we decided to include a geometrical representation of the tendon. Unlike graphical representations of musculotendons [21, 9], tendons are considered as a single piece lying on one side of the MTU in classical models such as Hill's and Zajac's [22]. However, like their graphical counterparts most real tendons are located on each side of a muscle. The biomechanical data sets include parameters that would satisfy the equations in classical models, yet they do not provide an accurate geometric representation for each tendon as the length distribution on separate sides of musculotendons remains unknown. In addition, the tendon slack length also combines the length of free and aponeurotic ten-

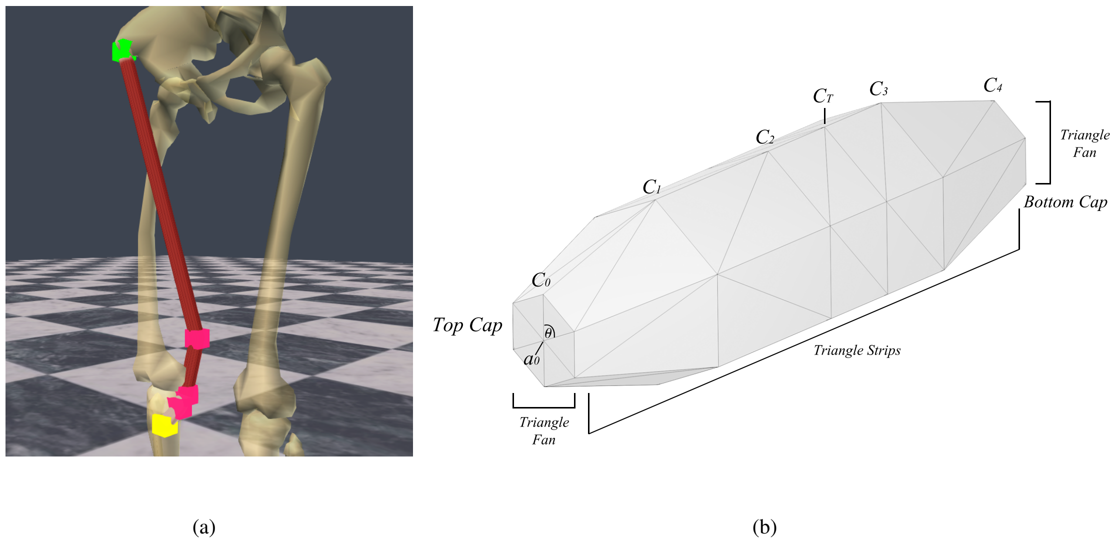


Figure 2: The MTU is initially represented as a closed cylinder centered on the action line before adding cross sections, depicted here for the right sartorius. (a) It runs from the origin point (green box) to the insertion point (yellow box) through the via-points (pink boxes). (b) For visualization, triangle fans and strips connect the vertices of the cross sections.

dons [18], and there is no distinction as to where the free tendon stops and the aponeurotic tendon begins. As a result, no data set was found that included lengths for each separate tendon and for each component of the tendon, dividing each slack length parameter into four separate lengths.

Although anatomically incorrect, we still decided to implement just one tendon under the assumption that, at least for the lower body, most free tendon lengths seemed to be distributed on the insertion side. Hence, in our method, the length of the tendon is divided by half and computed backwards, starting at the insertion and traversing along the action line in the direction of the origin. To compute the tendon length, we first calculate the rest length and the scaled rest length of the musculotendon. We use a total of four separate variables for lengths, namely the length of the action line l^A, the length of the scaled action line l^{AS}, the rest length l^R, and the scaled rest length l^{RS}. Scaled versions of l^A and l^R are used in this model because the skeleton was transformed during the alignment process that included all of the attachment points in order to fit to the virtual character. The positions of the joints in the musculoskeletal model and the high resolution virtual human were used to scale each body part individually and uniformly in three dimensions. We assumed the tendon in our model to be rigid [23], which allows its length to be set equal to its slack length, denoted by l_s^T. This opened up the possibility to adapt the implemented version of the classical Zajac model to derive the rest length $l^R = l_s^T + \left(l_o^F \cos\alpha\right)$, where l_o^F represents the optimal fiber length and α the pennation angle present in pennate muscles. The unscaled parameter values are defined in Delp et al. [18] that the reader is invited to consult for more details about their definitions and calculations. To derive the scaled rest length l^{RS}, the ratio between the $|A|$ and the scaled $|A|$ in the virtual character is calculated. The generic calculation for the $|A|$ is given by the sum of the Euclidian length between two a_k elements:

$$\sum_{k=0}^{n-2} \left| \, (a_{k+1} - a_k)^T \, \right| \tag{2}$$

where the outcome can be assigned to l^A or l^{AS} depending on which state of A the calculation was executed. Next, l^{RS} is given by $l^{RS} = \frac{l^R l^{AS}}{l^A}$. With l^{RS} assigned, the scaled tendon slack length l_s^{TS} can also be obtained with $l_s^{TS} = \frac{l^{RS} l_s^T}{l^R}$.

As stated previously, it was decided to use $\frac{1}{2} l_s^T$ which gave visually correct results but remains anatomically inaccurate. Table 1 shows the values obtained on the *tibialis anterior* muscle of the right leg.

action line l^A	rest l^R	tendon slack l_s^T	scaled action line l^{AS}	scaled rest l^{RS}	scaled tendon slack l_s^{TS}
0.303	0.320	0.223	0.292	0.309	0.215

Table 1: The lengths of the MTU parameters obtained for the *tibialis anterior* muscle. All values are given in meters.

The scaled tendon slack length is then used to determine which segment of A contains the point where the tendon ends and the muscle starts, called tendon point. Let first the function $d\left(k\right)$ equal a length of a negative vector between two points on A:

$$d\left(k\right) = \left| \left(a_k - a_{k-1}\right)^T \right| \tag{3}$$

where k is the index iterating over A in $\mathbb{N}$. Starting from the insertion point $n-1$, the traveled distance d_{t_g} along A is then computed with:

$$d_{t_g} = \sum_{k=n-1}^{1} d\left(k\right) \left[d\left(k\right) \leq \frac{1}{2} l_s^{TS}\right] \tag{4}$$

adding only the lengths where the condition $d\left(k\right) \leq \frac{1}{2} l_s^{TS}$, contained within Iverson brackets, is satisfied. To return the last index j that represents the last point before the tendon point, a final condition is required on j:

$$j = k \text{ for } \left\{d\left(k\right) \leq \frac{1}{2} l_s^{TS}\right\} \tag{5}$$

To give an example, let the number of points n for a particular musculotendon be equal to 5. During a particular computation, the condition in Equation 4 returned 1 up until $k = 3$, consequently leaving $j = 3$ as well. Because of the $\leq$ sign present in the condition, the tendon point in A would lie somewhere between a_2 and a_3. This example is illustrated in Figure 3.

The actual position of the tendon point t_g based on $\frac{1}{2} l_s^{TS}$ can be computed using linear interpolation between two points on A:

$$t_g = a_{j-1} + s\left(j\right) \text{ with} \tag{6}$$

$$s\left(j\right) = \frac{\left(d\left(j\right) + d_{t_g}\right) - \frac{1}{2} l_s^{TS}}{d\left(j\right)} \left(a_j - a_{j-1}\right)^T \tag{7}$$

$s(j)$ computes the ratio between the scaled tendon slack length and the distance until the point a_{j-1}, and then scales the magnitude of the vector between a_j and a_{j-1}. The position of a_{j-1} plus this outcome is used to calculate t_g.

At t_g, a cross section C_t will be added to divide tendon and muscle (see section 4.2.1). Each element of C_t is also linearly interpolated between the previous and next cross sections as each cross section can have different shapes. To retain the assumption that the tendon is infinitely stiff in this model, $\frac{1}{2}l_s^{TS}$ is computed only once to fix the length of the free tendon. This is done after the skeleton is aligned and the enhancement process presented hereafter are completed, but prior to loading any animations.

4.2 Geometrical enhancement of MTUs

The general idea for enhancing a MTU is illustrated in Figure 4 and additional illustrations can be found in the accompanying video.

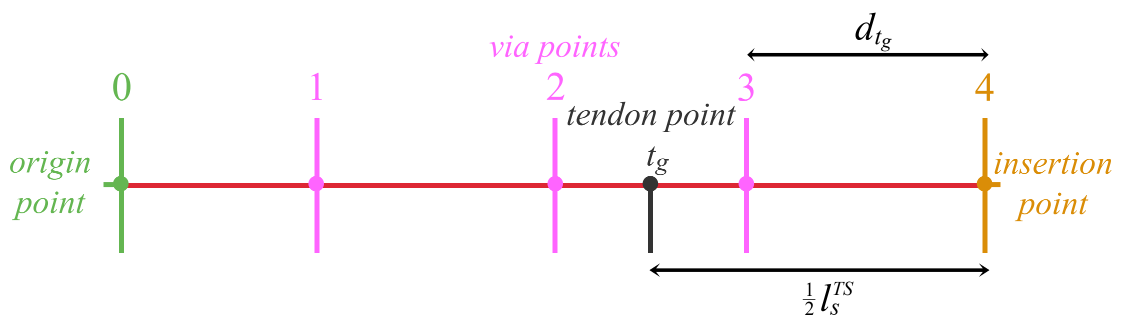
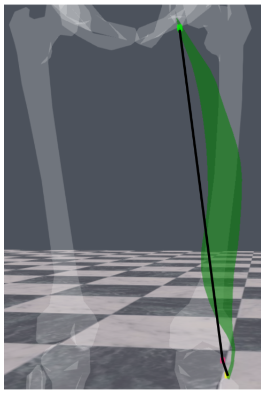
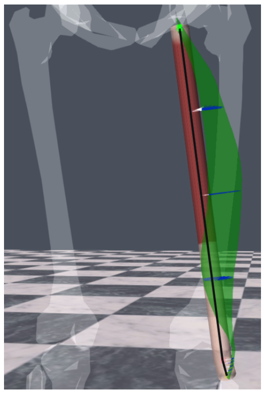
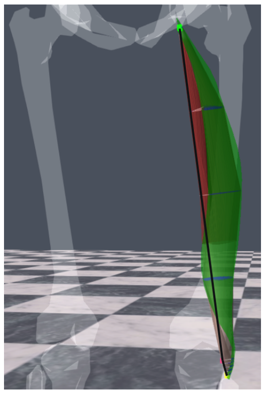
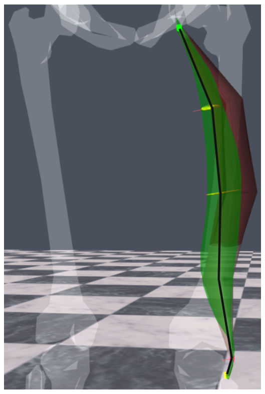


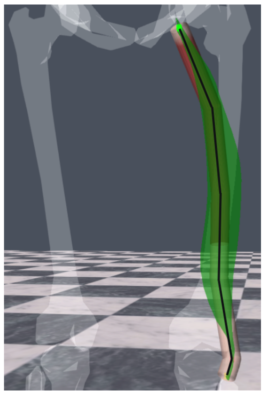
Figure 3: A schematic representation of the musculotendon model including a single tendon point lying between a_2 and a_3.

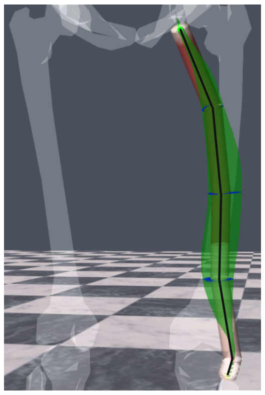

(a) Action line (black line) and high resolution mesh (green) are both mapped into the virtual character (section 3)

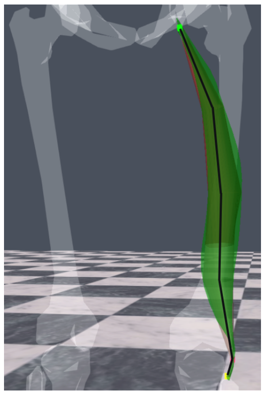

(b) The action line is represented as a closed cylinder (section 4.1) and rays are cast from the cross sections (blue lines)


(c) After direct translation to the intersections, not all vertices are repositioned on the high resolution mesh (section 4.2.2)


(d) Moving the action line towards the centroid of the new cross sections shows the incomplete process (section 4.2.3)


(e) We start again with the closed cylinder using the new action line routing


(f) This repositioned action line gives appropriate ray casting (section 4.2.4)


(g) All vertices are now located on the high resolution mesh

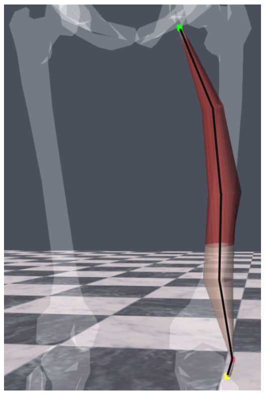

(h) We obtain a scalable MTU based on the enhancement of the action line

Figure 4: Overview of the enhancing process of the action line (a) into the scalable MTU (h) applied on the right biceps femoris.

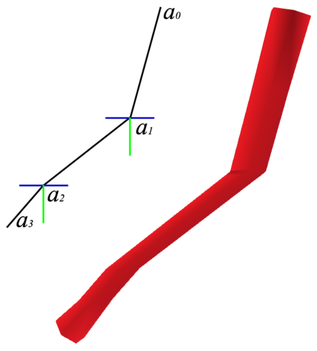
4.2.1 Adding volume with cross sections

We have opted to enhance the biomechanical representation towards an anatomical mesh instead of trying to reduce the resolution until reaching real-time performance for several reasons. Beside the former being more intuitive to control and computationally less demanding, our goal is to preserve biomechanical features at each level of enhancement. We took a particular interest in preserving at best the original relative positions of the via-points at each level and a uniform distribution of the resolution in-between these points. Indeed, via-points are present where the direction of the force changes in the muscle and the routing of the action line is associated with its physiological properties. If the muscle meshes were to be decimated from a high resolution model, we would have to ensure that the decimation process preserves the uniformity of the resolution along the action line and make certain that the decimated mesh is still representative of the position of the via-points, which would be complicated. By enhancing the model from the action line, we preserve these properties at best.

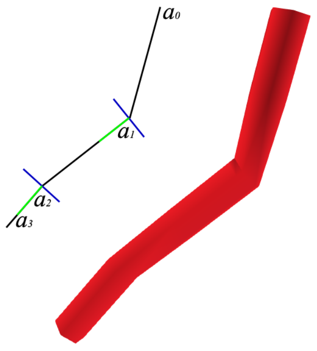
To add volume to the polyline, extra vertices are needed for its surface. The initial geometrical state of the model, a polygonal mesh constructed as a closed cylinder, already includes two cross sections for the top and bottom caps. To also account for muscles that contain via-points, the closed cylinder can be split into smaller segments by introducing bisecting cross sections at each element of A (see Figure 2). The set C_k containing each element of an arbitrary cross section that is responsible for volume, of which none are on the action line, is defined as:

$$C_k = \left\{c_0^k, c_1^k, ..., c_{n-1}^k \mid c_i^k \in \mathbb{R}^3\right\} \text{ with } C_k \cap A = \varnothing \tag{8}$$

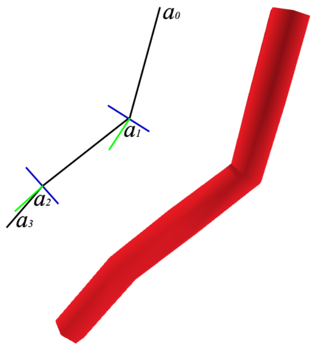
The cardinality of C_k is also proposed as $|C_k| \geq 6$ (i.e. hexagonal cross sections) and $|C_k| = \{2a \mid a \in \mathbb{Z}\}$ for reasons discussed in section 4.2.3. We then need to define the direction of the up-vector for an a_k. We decided to use the approach that includes information from both sides of a_k with the vector $(a_{k+1} - a_{k-1})^T$ resulting in an approximation that is centralized and therefore more robust when dealing with acute angles between two other cross sections (Figure 5).



(a) Parallel to y



(b) Parallel to $(a_{k+1} - a_i)^T$



(c) Parallel to $(a_{k+1} - a_{k-1})^T$

Figure 5: Three possibilities for choosing an up-vector for bisecting cross sections. The chosen computation is (c) which takes the up-vector $(a_{k+1} - a_{k-1})^T$ as this approximation includes information from the point before and the point after the current point. Bisecting cross sections are defined such that $a \in A$ and $k \in \mathbb{N}$ with $0 < k < (|A| - 1)$.

4.2.2 Repositioning C's on the surface of U

The next step is to move the cross section vertices to fit at best the high resolution model U. We proceed using a two-pass algorithm that involves ray casting and vertex translation. An element c_i^k for an arbitrary C_k is repositioned when an intersection point p_i is found between a surface triangle of the mesh U and a ray that has been cast from c_i^k. For each c^k, rays are cast in two normal directions on the plane of every C_k, namely positive and negative. With the positive direction being the normalized vector $\left(\widehat{c_i^k - a_k}\right)^T$ and the negative direction being the normalized vector $\left(\widehat{a_k - c_i^k}\right)^T$. It was chosen to use the fast ray-triangle intersection test algorithm of Möller-Trumbore [24] for computational speed as ray-casting is used more than once in our method.

Each surface triangle of U that intersects the plane creates a Jordan curve, denoted as U^{C_k}, in the same plane as C_k. The ray-triangle intersection tests are used to find three possible outcomes, which are:

1. U^{C_k} lies completely outside of C_k in the plane containing C_k.
2. U^{C_k} lies completely inside of C_k in the plane containing C_k.
3. U^{C_k} lies partly outside and partly inside of C_k in the plane containing C_k.

Our method incorporates two additional preferences to cope with the two sides of each face, i.e. when an intersected triangle is either back or front-faced. The first preference is for choosing intersections found through rays being cast in the positive direction P above the negative direction N, shown schematically in Figure 6.

The second preference is that for P, the furthest p_i of each back-face triangle is returned, while for N the closest p_i of each front-face triangle is returned, as illustrated in Figure 7. The combination of these two preferences allows each c_i^k to find its correct corresponding p_i.

4.2.3 Repositioning elements of A within the volume of U

C_k's are constructed with algebraic radii, denoted by the set R_k, between elements of C_k and corresponding elements a_k of A. Each radius r_i^k is obtained with $\left| \left(c_i^k - a_k\right)^T \right|$ or

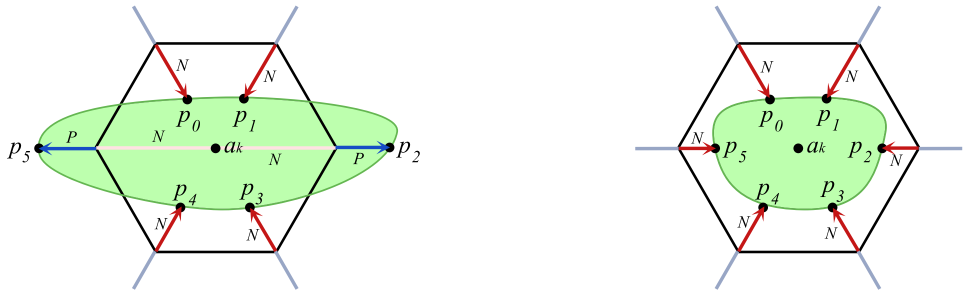


Figure 6: Showing ray-triangle intersection outcome number three (left) and outcome number two (right). U^{C_k} is represented by the green shapes and p_i's denote intersection points. p_i's can be found with a ray cast in the positive direction P or in the negative direction N. The left figure shows that here N of p_5 equals p_2 and N of p_2 equals p_5, illustrated with the red line segment. The figure on the right illustrates a case where no P-points were found, and instead resorted to finding N-points.

$\left| \left(a_k - c_i^k\right)^T \right|$. Using radii proved to be versatile as it allowed C_k to have an initial shape that was useful for the intersection tests and allowed each deformation to be kept in memory by storing distances as radii. This brought the benefit that a point within U^{C_k} could be determined by checking which radii have been updated and consequently iterate the process discussed previously to achieve a better approximation for each U^{C_k}.

To explain the approximation process visually, a key example is introduced in Figure 8. The example uses the index $k = 1$ in the longitudinal direction where U^{C_1} is a closed concave shape positioned outside of C_1. In the figure we show the intermediate result indicating why locating a point within U^{C_k} (an inner point) is necessary in as the result equals a false representation with the deformed hexagon being complex instead of simple. This is done as follows.

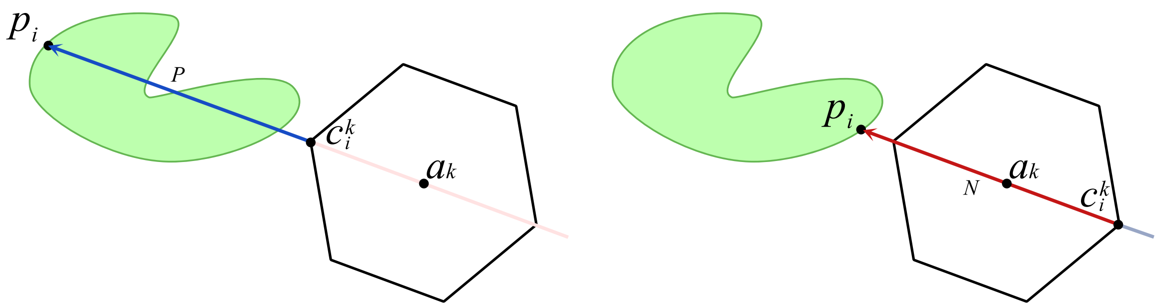


Figure 7: Schematic figures showing outcome one and highlighting the second preference: if more than one p_i is found for P, choose the furthest back-face triangle. Conversely, if more than one p_i is found for N, choose the closest front-face triangle.

Match Detection First comes the task of detecting a match between two algebraic radii that have been updated. A match is denoted by e^k and is defined as an ordered pair of indices (i^+, i^-) that correspond to an index i of p where its distance is established with either a positive or negative algebraic radius for two opposing elements of C_k. In this respect, the or-relationship is exclusive and imposes the condition that one of the two opposing algebraic radii has a negative sign.

In order for a pair of cast rays to be considered equal, for instance c_1^1 and c_4^1 in Figure 8,

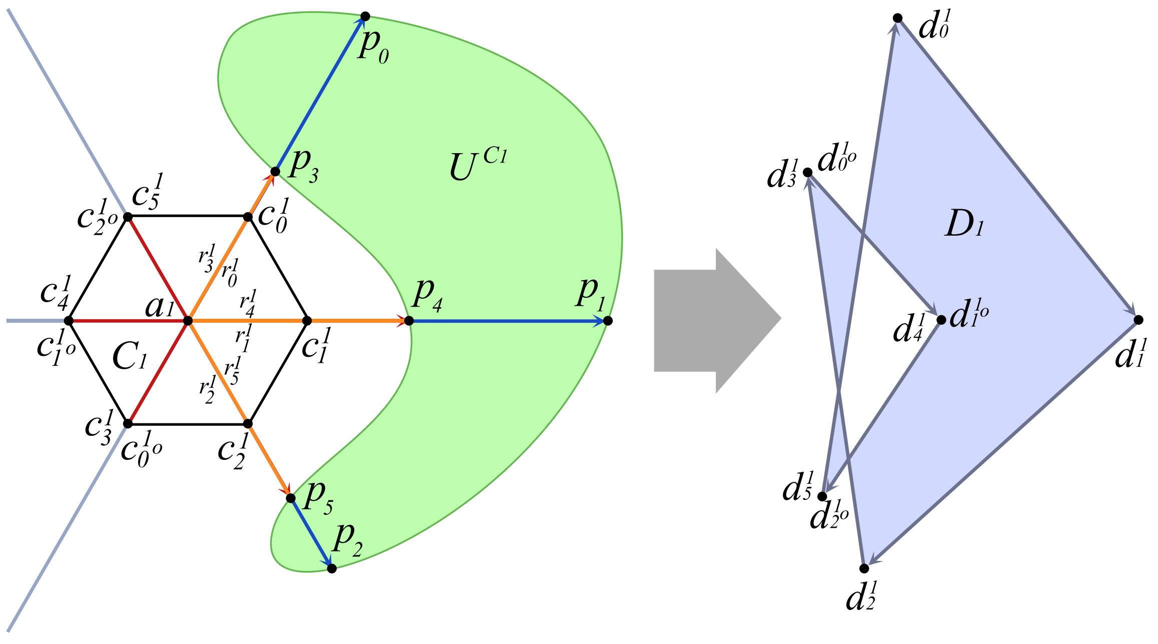


Figure 8: Showing a key example with a concave slice of a mesh, denoted as U^{C_1}. Applying the preferences and conditions discussed previously results in the complex polygon D_1, which is a false representation due to a_1 being positioned outside of the targeted cross section U^{C_1}.

it is important that both c^1's are symmetrical counterparts with respect to a_1. This is ensured by requiring that the cardinality for every C_k is always even, which is the case here for the default example that uses six vertices. It is also visible that for every C_k with even cardinality, the maximum number of matches would be always equal to $\frac{1}{2}|C_k|$ thereby reducing the number of lookups by half. Hence, each opposing element of c_i^k can be deduced by taking the opposing index i^o with:

$$i^o = i + \frac{1}{2}|C_k| \text{ where } \left\{ i \in \mathbb{N} \;\middle|\; i < \frac{1}{2}|C_k| \right\} \tag{9}$$

Each detected match helps to determine when a ray passed a center point a_k as this suggests the possibility that U^{C_k} lies (partly) somewhere outside of C_k. Because each surface vertex or c_i^k is drawn by computing its radius, a ray that surpassed a_k in the negative direction N needs a negative algebraic radius to correctly represent the intersection point's mirrored position. This is accomplished with the condition $\left| \left(p_i - c_i^k\right)^T \right| > r_i^k$. In this case, a negative sign is added to update the radius with the following new result: $-\left| \left(p_i - a_k\right)^T \right|$. In all other cases, the radius remains positive. In the example of Figure 8, radii r_3^1, r_4^1, and r_5^1 become negative (depicted with orange line segments) and overlap the positive radii r_0^1, r_1^1, and r_2^1 lying in the same direction.

Permuting Matches Once all matches are detected, they are stored in a single container, denoted by the set E_k with $|E_k| \leq \frac{1}{2}|D_k|$, and where D_k represents a deformed cross section at index k. Permuting the elements of E_k is necessary for the solution to determine an inner point and is discussed in the next paragraph. Intuitively, the result of the permutation on E_k

is a sequence of matches so that the first term $\left(0, e_0^k\right)$ represents a match at one end of U^{C_k}, winding along the boundary of the shape, until reaching the final term $\left(n-1, e_n^k - 1\right)$ at the other end of U^{C_k}; given that the number of elements is greater than one. Because relative spatial configurations of U^{C_k} with respect to C_k have many possibilities, the intersections tests will not necessarily return a sequence of matches in an ordered fashion when the tests always start at c_0^k. One example is when U^{C_k} is intersected by rays cast from elements c_5^k and c_0^k. Thus, without taking the sign of each algebraic radius into consideration, there is no easy way of determining whether the first match starts with the rays cast initially through c_0^1 or through c_5^1.

By filtering the type of intersection that produced either a positive or negative algebraic radius, the elements of E_k can be rearranged so that the sequence winds correctly, similar to the concept of chain codes. Searching with index $i = 0$ up until $i = \frac{1}{2}|D_k| - 1$, a sign of the computed radius for all d_i^k is chosen. If the new position d_i^k was established with a negative algebraic radius, then its opposing element $d_{i^o}^k$ is chosen such that this would contain the correct point, i.e. $i^+ = i^o$. In Table 2, we show all possible permutations with i^+ using a configuration with U^{C_k} lying outside and intersected by three pairs of rays cast from C_k with $|C_k| = 6$. Note that only half of the points of D_k have to be searched to get all six permutations. The algorithm to compute the permutation σ based on indices of E_k is given in pseudo-code in Algorithm 1.

```
input  : A set $E_k$ containing ordered pairs of matches $(i^+, i^-)$.
output: A sequence $\langle e_n^k \rangle$ with a range of $\left\{ e_n^k : 0 \leq n < |E_k| \right\}$.

1  if $|E_k| > 1$ then
2      $E_k \leftarrow$ SortAscending($E_k$); // sort numerically from
       low-high
3      $\langle e_n^{temp} \rangle \leftarrow E_k$;
4      $temp\_match \leftarrow \left\{ e_0^k \right\}$;
5      $mindex \leftarrow e_{0_1}^k$; // note that $e_{n_1}^k$ represents $i^+$ for every $\left\{ e_n^k \right\}$
6      $nshifts \leftarrow 0$; // index where to insert new term in $\langle e_n^{temp} \rangle$
7      for $m \leftarrow 1$ to $|E_k| - 1$ do
8          if $\left( e_{m_1}^k - mindex \right) \neq 1$ then
9              $temp\_match \leftarrow \left\{ e_m^k \right\}$;
10             $\langle e_n^{temp} \rangle \leftarrow \langle e_n^{temp} \rangle - (m, e_m^{temp})$;
11             $\langle e_n^{temp} \rangle \leftarrow \langle e_n^{temp} \rangle + (nshifts, temp\_match)$;
12             $nshifts \leftarrow nshifts + 1$;
13         end
14         $mindex \leftarrow temp\_match_1$;
15     end
16     $\langle e_n^k \rangle \leftarrow \langle e_n^{temp} \rangle$;
17 end
```

Algorithm 1: Permutation given in pseudo-code that returns E_K containing a sequence

of matches winding through from one end till the other end of U^{C_k}.

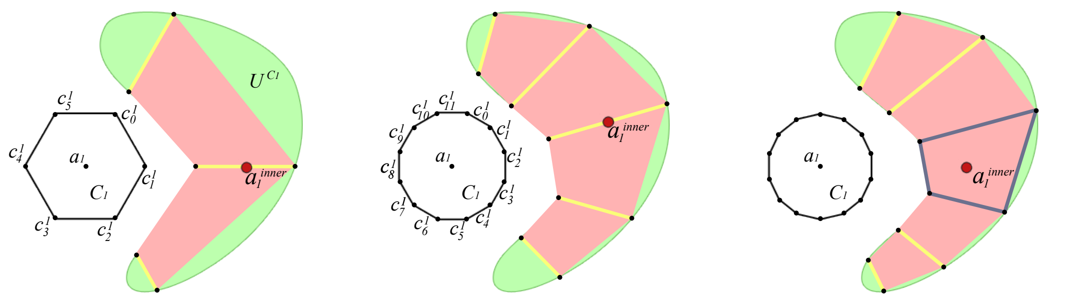
Sign of r for $\left(d_0^k, d_1^k, d_2^k\right)$	Permutation with i^+ of d_i^k	Rearranged matches
$f:(+\ +\ +) \to \left(d_0^k, d_1^k, d_2^k\right)$	$\sigma(0,1,2) = (0,1,2)$	$\langle e_0^k, e_1^k, e_2^k \rangle$
$f:(+\ +\ -) \to \left(d_0^k, d_1^k, d_{2^o}^k\right)$	$\sigma(0,1,5) = (5,0,1)$	$\langle e_2^k, e_0^k, e_1^k \rangle$
$f:(+\ -\ -) \to \left(d_0^k, d_{1^o}^k, d_{2^o}^k\right)$	$\sigma(0,4,5) = (4,5,0)$	$\langle e_1^k, e_2^k, e_0^k \rangle$
$f:(-\ -\ -) \to \left(d_{0^o}^k, d_{1^o}^k, d_{2^o}^k\right)$	$\sigma(3,4,5) = (3,4,5)$	$\langle e_0^k, e_1^k, e_2^k \rangle$
$f:(-\ -\ +) \to \left(d_{0^o}^k, d_{1^o}^k, d_2^k\right)$	$\sigma(3,4,2) = (2,3,4)$	$\langle e_2^k, e_0^k, e_1^k \rangle$
$f:(-\ +\ +) \to \left(d_{0^o}^k, d_1^k, d_2^k\right)$	$\sigma(3,1,2) = (1,2,3)$	$\langle e_1^k, e_2^k, e_0^k \rangle$

Table 2: Taking the sign of r into consideration allows E_k to be rearranged.

Determining an inner point When a_k lies inside of U^{C_k}, the number of false positives is presumed to be low thereby reducing the initial problem illustrated in Figure 8. Each detected match therefore suggests the possibility of D_k resulting into a complex shape, which should not be the case as polygonal meshes suited for deformation should remain simple. To determine an inner point, a heuristic is used on the number of matches that chooses the median match-index after being permuted by σ. When the number of matches is odd, the mid point of the middle match-index is taken as the inner point where a_k should be repositioned. When even, a single match-index cannot be chosen, thus a more complicated approach is used. Here, the two middle matches are taken that collectively form a quadrilateral. At this point, the quadrilateral is still in E_3 representing four vertices of a particular cross section. Each vertex of this quadrilateral gets projected into E_2 on the same plane as the current cross section in order to calculate the centroid by averaging the projected coordinates of the vertices of the quadrilateral. This centroid point is the inner point where a_k has to be translated to. Figure 9 shows three examples for different cardinalities of C_k.

4.2.4 Iterating the enhancement process

With a_k now repositioned to $a_k^{Centroid}$, the enhancement process can be repeated in a second pass in order to get a simple and accurate polygonal representation of U^{C_k}, as given in Algorithm 2.



U^{C_l}
c_0^l c_1^l c_2^l c_3^l c_4^l c_5^l
a_l
C_l
a_l^{inner}
c_0^l c_1^l c_2^l c_3^l c_4^l c_5^l c_6^l c_7^l c_8^l c_9^l c_{10}^l c_{11}^l
a_l
C_l
a_l^{inner}
a_l
C_l
a_l^{inner}

(a) Midpoint of medial match (uneven) (b) Same as (a) for a Dodecagon (c) Even results take the centroid of a quad

Figure 9: An example illustrating two types of inner point determination. (a) shows the inner point determined by the midpoint of the middle match for an odd number of matches. (b) shows the same situation for a dodecagon. (c) shows the resulting centroid determined with an even number of matches for a tetradecagon.

```
foreach $M$ do
    $k \leftarrow 0$;
    for $k \leftarrow 0$ to $k = |A| - 1$ do
        $C_k \leftarrow$SetRadii($C_k$); // initial configuration
        ComputeRayTriangleTests($C_k$);
        $D_k \leftarrow$UpdateRadii($C_k$);
        $E_k \leftarrow$ DetectMatches($D_k$);
        $\langle e_n^k \rangle \leftarrow \sigma$;
        $a_k \leftarrow$ ComputeInnerPoint($\langle e_n^k \rangle$);
        $C_k \leftarrow$ResetRadii($D_k$);
        ComputeRayTriangleTests($C_k$);
        $D_k \leftarrow$UpdateRadii($C_k$);
    end
    $a_0 \leftarrow$ResetActionPoint($a_0$); // origin position
    $a_{|A|-1} \leftarrow$ResetActionPoint($a_{|A|-1}$); // insertion position
end
```

Algorithm 2: The enhancement algorithm.

4.3 Scalability of the musculotendon model

Some MTUs in the biomechanical model contain zero via-points. This creates a problem for leveraging the accuracy during the enhancement process discussed in section 4.2 as not much information is present in the latitudinal and longitudinal dimension, i.e. the amount of detail for each C_k's at every a_k of A. To solve this problem, we decided to adapt the musculotendon model to be scaled independently in the longitudinal and latitudinal dimensions. Allowing the model to be adapted in both directions satisfies the requirement to increase or decrease the resolution of the meshes depending on the available computational power and the targeted scenario.

As discussed in section 4.2.1, musculotendons where via-points are present can be split into smaller segments by introducing extra bisecting cross sections that lie orthogonal to the vector $(a_{k+1} - a_{k-1})^T$ for $a \in A \wedge 0 < k < (|A| - 1)$. For the longitudinal segment scalar (LSS) we use the number of extra segments instead of cross sections. A segment is defined as the volume between two cross sections. When LSS $= 0$ this would equal one segment, when LSS $= 1$ this would equal two segments, LSS $= 2$ equals four segments, and so forth. Our reasoning is that when a cross section is added it actually splits an existing segment into two, hence when this splitting pattern is repeated, the resulting expression would be in the form of 2^n. Furthermore, each newly created a_k is added at exactly the mid point between the two opposing cross sections on each side. This allows the dimension of A to scale in a uniform fashion. The same repeating pattern is applied as well to via-points, as each via-point also splits an existing segment into two. Adding everything together results in the following equation to calculate the target cardinality in the longitudinal dimension:

$$t_{\text{LON}} = 2^{\text{LSS}}\left(|V| + 1\right) + 1 \text{ with LSS} \in \mathbb{N} \tag{10}$$

The musculotendon model can also scale in the latitudinal dimension, denoted by t_{LAT}, with the number of vertices at each cross section respecting the constraint $|C_k| \geq 6 \wedge |C_k| = \{2a \mid a \in \mathbb{Z}\}$. This is used to locate inner points as discussed previously. A copy of the polyline A, denoted as A_{copy}, is taken where the latitudinal dimension is scaled to a high enough value so that every a_k of A_{copy} gets repositioned inside the volume of a mesh U. Once the enhancement process is finished, the repositioned a_k are copied back into the original container where ray-triangle intersections tests are executed again to get the final approximated shape. This is done by introducing a scalar, the cross sectional scalar, denoted as CSS to differentiate between latitudinal scales and is used as:

$$t_{\text{LAT}} = \text{CSS}\,|C_k| \text{ with CSS} \in \mathbb{N} \tag{11}$$

We have established a lower bound for the cardinality of C_k defining the lowest amount of geometrical information that is reasonably achievable for the simulation. In our experiments, the lower bound was chosen to be six for every C_k as this cardinality is (1) reflectively symmetric, (2) has a reasonable perimeter to area ratio and (3) has a low number of vertices. Illustrations of the longitudinal and latitudinal scales are given in Figure 10.

4.4 Drawing the MTUs

Drawing the model at each simulation time step comprises of few steps. First, C_k is computed with radii from the set R_k. Similar to the internal coordinate system used for the

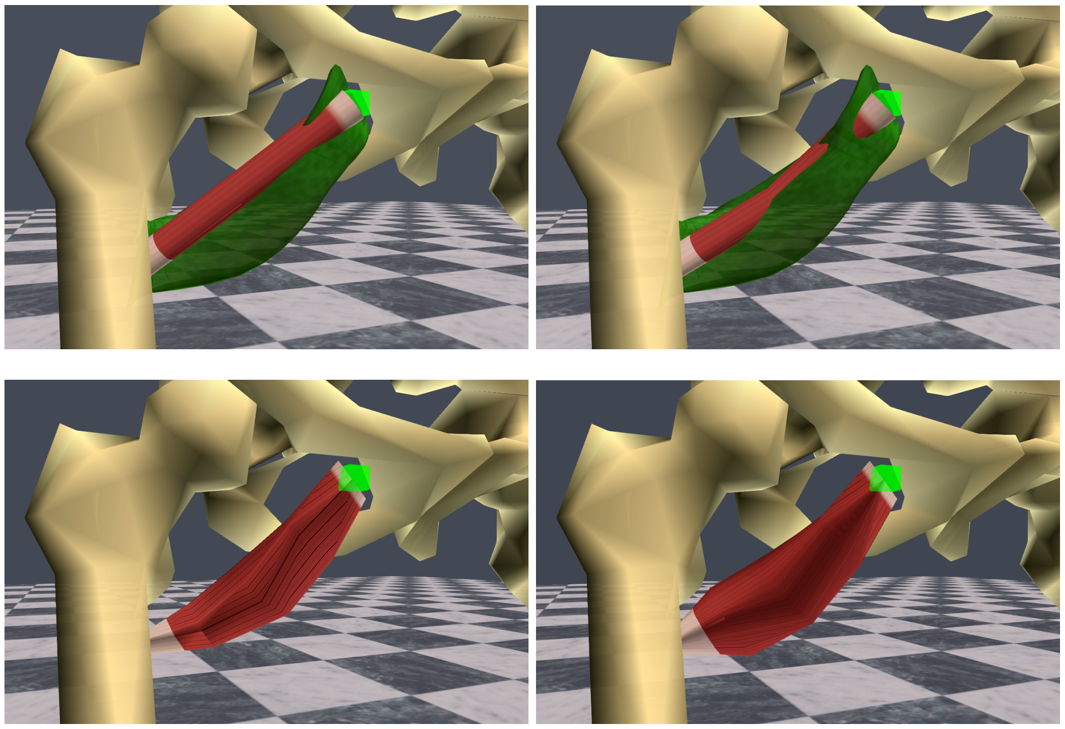


Figure 10: The MTU can be independently scaled in the longitudinal and latitudinal dimensions, here depicted on the pectineus muscle. The top row demonstrates the longitudinal scale with $LSS = 0$ (left) and $LSS = 1$ (right) where we note the better bending in the latter due to the additional intermediate cross sections. The bottom row demonstrates the latitudinal scale with $|C_k| = 6$ (left) and $|C_k| = 36$ (right), and both with $LSS = 2$ and $CSS = 1$. We note the better fit of the latter to the high resolution model (green mesh) due to the higher number of vertices per cross section.

relative positions of a_k, the directions of r_i^k are also computed in intervals relative to angle θ, given as $\frac{2\pi}{|C_k|}$. Next, one tendon point is interpolated and stored separately as a_t and C_t. Finally, the muscle is drawn, shaded, and textured.

Instead of using triangle lists, we opted for a combination of triangle fans and strips, as illustrated in Figure 2b. These render types give less vertex redundancy, needing just $n + 2$ vertices to draw n triangles. Coupling fans and strips with vertex and index buffers results in an efficient way to draw graphical musculotendons in real-time.

The final addition to the musculotendon model the inclusion of separate textures to inspect the length of tendons on the insertion side derived with the functions discussed in section 4.1.2. Adding textures proved to be a practical way and increased the visual realism of the model. Note that the white texture on the origin side of a muscle is solely texture added for visualization purposes only and does not represent a separate tendon compartment. Some examples are given in Figures 1a and 10.

5 Results and discussion

5.1 Geometrical enhancement

Permuting matches to locate an inner point for each U^{C_k}, and adapting the longitudinal and latitudinal scales, proved to be a good solution to the approximation of high resolution muscle geometries for the lower part of the human body. Table 3 lists the total number of vertices for 12 different scales compared to the total number of vertices used in the original anatomical mesh. It shows a drastic reduction in the number of vertices used in our model. For instance, at the scale of $|C_k| = 12$ and LSS $= 2$, our model uses $\approx 90\%$ less vertices than the original anatomical mesh. The 54723 vertices represent the total number of vertices of the 48 original triangulated meshes.

	$\vert C_k\vert = 6$	$\vert C_k\vert = 8$	$\vert C_k\vert = 10$	$\vert C_k\vert = 12$	UHM
$LSS = 0$	960	1248	1536	1824	
$LSS = 1$	1536	2016	2496	2976	54723
$LSS = 2$	2688	3552	4416	5280	

Table 3: Number of vertices for a total of 48 musculotendons in 12 different scales compared to the 48 anatomical meshes (UHM).

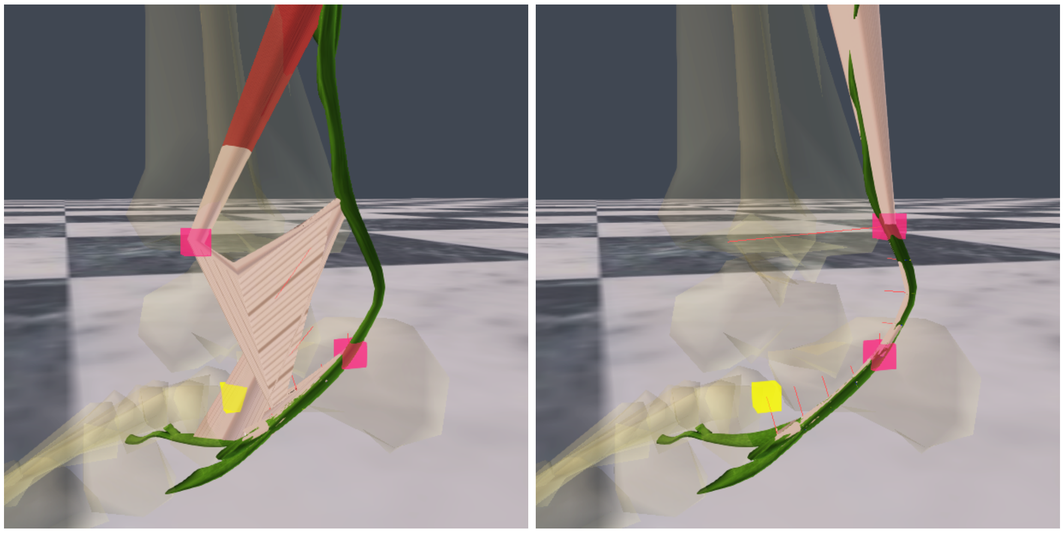
5.2 Latitudinal and longitudinal scalability

The permutation σ given in Algorithm 1 works for both irregular convex and irregular concave polygons as far as it was tested within the configurations and parameters used in this work. Figure 11 shows one example where adapting the latitudinal scale proved to be useful. The shortest intersected triangle found in the positive direction was unfortunately at other parts of the tendon resulting in misplaced c_i^k's. This was remedied by increasing CSS to a high enough value that resulted in a complete convergence for the via-points.

Figure 12 shows the approximation of the high resolution anatomical mesh of the *tib-*

ialis anterior at different scales. We see that different scale combinations produce different results before reaching convergence. The presumption here is that for thin segments, usually ligaments and tendons, one would need enough surface vertices to cast enough rays in order to detect enough matches. In this example, the fact that the muscle contains via-points also increases the convergence rate. Because once via-points are repositioned to their correct location, it prevents other bisecting cross sections from being misplaced.

In order to quantify the overall result, ray-triangle tests are carried out after the enhance-



(a) Incorrect result when $CSS = 3$ (b) Correct result when $CSS = 6$

Figure 11: Misplaced action line due to partial convergence in the *tibialis posterior*. The yellow box represents the position of the insertion of the muscle, the pink boxes are the via-points and the green object is the high resolution mesh. (a) an incorrect translation when $|C_k| = 8$ and $CSS = 3$, however doubling in (b) the number of cross sections during approximation with $CSS = 6$ resulted into a complete converged tendon.

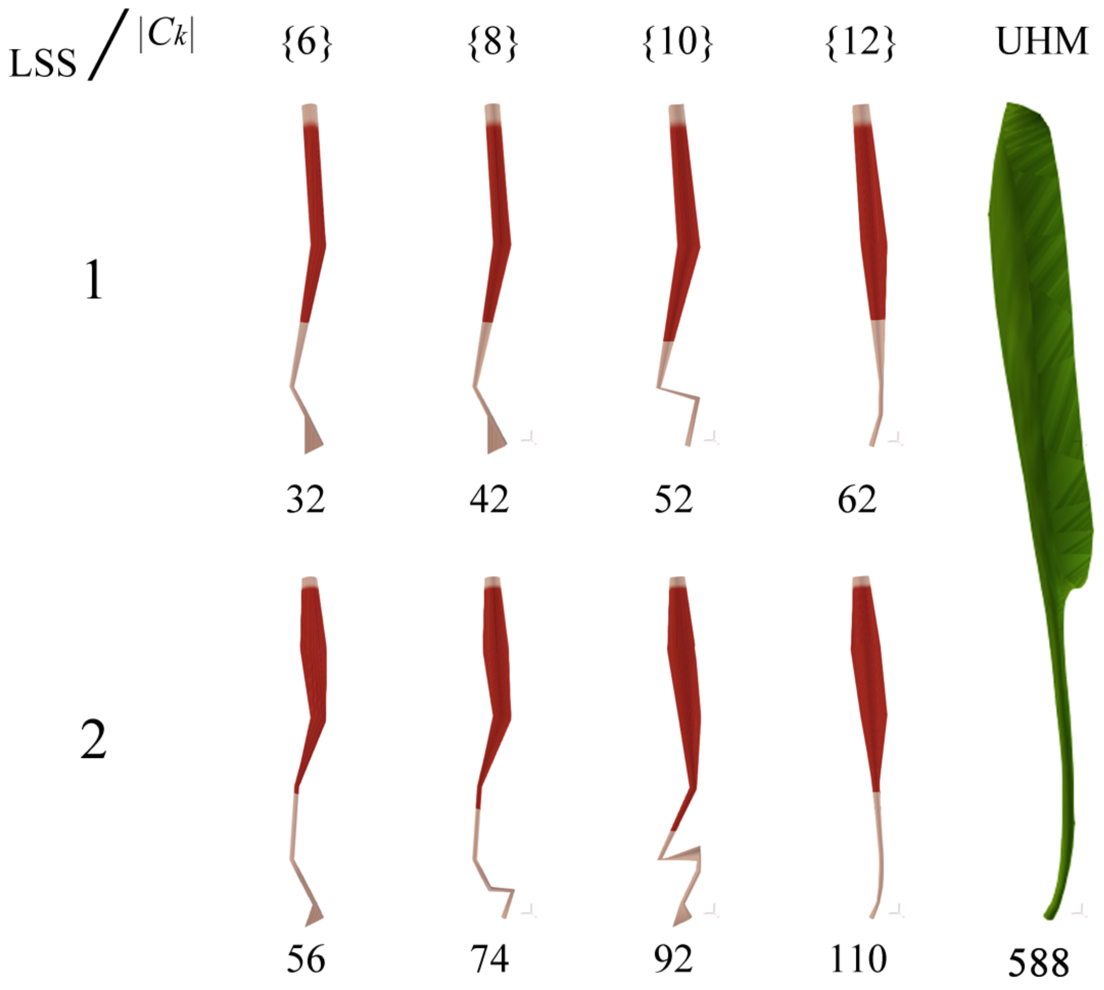


Figure 12: The approximation of the *tibialis anterior* (noted UHM) at eight different scales of C_k. The number below each mesh represents the total number of vertices. For this particular mesh, convergence is reached when $|C_k| = 12$, for both LSS $= 1$ and LSS $= 2$ while keeping $CSS = 1$. The convergence for $LSS = 2$ shows a better approximation for the belly of the muscle.

ment process is completed. These tests determine whether a_k is actually inside a cross section of the anatomical mesh. Specifically, this is achieved by casting rays in two directions, namely from a_k along the direction of $c_i^k - a_k$ and $c_{i^\circ}^k - a_k$. If all ray pairs intersect a back-facing triangle, then it is assumed that a_k lies inside the anatomical mesh. The results are listed in Table 4. The worse case for LSS $= 1$ represents just 23 (or 9.58%) of the total number of 240 cross sections, while the worse case for LSS $= 2$ represents just 42 (or 9.72%) of the total number of 432 cross sections. We also note that complete convergence is reached when the number of rays cast is > 35, for both tested scales of LSS $= 1$ and LSS $= 2$. This can be for instance when $|C_k| = 6$ and CSS $= 6$ or when $|C_k| = 36$ and CSS $= 1$. This latter is considered the optimal choice of parameters as it contains the most information. In this configuration, the number of vertices for all 48 musculotendons totals 8736, which is $\approx 16\%$ of the original set of high resolution meshes.

5.3 Performance

A complete model with 48 musculotendons having $|C_k| = 8$ and LSS $= 1$, and upper and lower body skeleton meshes, ran 181Hz on average with a walk-cycle animation on the virtual character. This was tested on an Intel i5-3210M x64 CPU (Intel Corporation, Santa Clara, California, United States) running at 2.50GHz with a NVIDIA GeForce GT 645M mobile graphics card. This result has been obtained without any GPU or software-based parallelization techniques. Let us recall that our goal is to propose a geometrical

	LSS = 1				LSS = 2			
$\|C_k\|$	6	8	10	12	6	8	10	12
$CSS = 1$	23	19	7	8	42	24	12	17
$CSS = 2$	8	7	7	3	17	12	10	5
$CSS = 3$	7	3	1	0	10	5	2	0
$CSS = 4$	3	2	0	0	5	3	0	0
$CSS = 5$	1	0	0	0	2	0	0	0
$CSS = 6$	0	0	0	0	0	0	0	0

Table 4: Number of a_k outside the mesh.

scalable model that can be adapted at will, e.g. to the computation power, and scalable in dimensions meaningful to the action line. The important information is given by the relative performance between the different scales (directly readable from the number of vertices in Table 3 or the quality of the mesh in Table 4). If the original data set had more vertices, we expect the results to scale up and the conclusions to be identical. Nevertheless, there exists a threshold under/above which visual quality evaluation would tell us that it is not useful to have a lower or higher resolution. Our method spans that range because we can obtain any visual quality from a polyline to a full resolution input model.

As reported in section 5.2, the optimal choice of parameters is with $|C_k| = 36$ and CSS $= 1$, for which the frame rate was 135Hz. Thereby satisfying the > 60Hz requirement of real-time simulation. Finally, the enhancement process itself takes around 10 seconds for

48 musculotendons with LSS $= 1$ and $|C_k| = 36$, and is executed once at the beginning of the simulation. It is important to keep in mind that the method is designed to be used in muscle-based simulations coupled with soft body deformation. These two components can take up a lot of computation time and a model that would be seen as visually realistic in computer graphics will not be suitable for such simulations.

6 Conclusion

In this paper we have proposed a method to create a scalable MTU model that can be used for musculoskeletal simulation, or any muscle-based motion controller. Due to the computational limits we imposed, our method keeps the performance in check by adapting the cardinality of the model and using triangle fans and strips for the rendering of the cylinder-based MTUs. The functional MTUs are systematically enhanced by its geometrical counterpart while trying to preserve at best the biomechanical properties of the action line. Our enhancement process allows the possibility to locate points inside an arbitrary cross section of a high resolution mesh by another separate arbitrary cross section of a low resolution mesh such that it is invariant to the spatial and polygonal configuration between the two meshes. This could be also useful for object reconstruction where a single polyline is needed that always lies within concave irregular meshes. We expect our approach to be used in real-time simulators when 3D meshes of the muscles are available, offering, for example, support for deformation simulation and better realism through a better estimation of muscle lengths, and therefore muscle moment arms and forces.

Improvements and future extensions are possible. For this project, the lower body was used as a test case. A logical step would be to include both lower and upper biomechanical models of the human body. In addition, the tendon slack length parameter currently includes both the length of the free tendon and the length of the aponeurotic tendon as classical Hill-type models combine the length of each separate free tendon into a single variable while in reality most MTUs consist of at least one tendon on each side. To our knowledge, no functional representation is available that divides the tendon length into separate lengths for the origin and insertion, and also into its two constituents, free and aponeurotic. While simplifications can work well for state-of-the-art polyline-based biomechanical simulators, a combined tendon representation is not correct for applications involving geometrical or polygonal based representations of musculotendons, as it is indispensable to allow volume to be segmented with different material properties.

In such a scenario, the model should be able to deform in a physically accurate way and thus physical properties such as the volume of each compartment of the model should be accounted for. Using the current construction, this can be achieved with a tetrahedralization approach of the volumes in-between cross sections. Using as an example a latitudinal dimension of six with four segments for the muscle compartment, the number of tetrahedrons would be 72, which is around 1% of the total number of tetrahedrons used in the demonstration of Berranen et al. [12] that had a performance result of 45Hz.

The current work has a limited support of multiple action lines and multiple heads.

For muscles with multiples lines (e.g. adductor magnus, gluteus group of muscles), we had to manually divide their meshes with a 3D modeling tool in order to approximate the amount of mass (in our case represented by the geometry) used by each action line. Then each action line was enhanced, as presented, using its respective part mesh. For muscles with multiple heads (only the biceps femoris in the musculoskeletal model we use), the UHM data set provided for two distinct meshes for each of the heads, so each MTU has been enhanced using its respective mesh. The seriousness of the problem cannot easily be determined without a proper evaluation of the simulation that will use the enhanced model later on, but we still expect this type of manual specification to produce acceptable results. Indeed, as the purpose would be to better estimate the paths of the action lines during animation, associating one line to a part of a mesh should behave correctly given that positional constraints are added between the vertices at the interface of parts of an individual muscle mesh. If a unique muscle mesh were necessary, the enhancement process could be adapted as well to be constructed from cross sections with multiple ’center-points’ (one for each line or each head). We could for example add links (line segments) between the ’center-points’ of the action lines that will be meant to support and group them during the creation of the geometry. An algorithm would have to be developed that makes sure all connected action lines have the same number of cross sections at similar distance among them. Then, ’center-points’ would be grouped allowing for a global contour to be created for each cross section using ray-triangle tests as in the current enhancement process. This process, although anatomically inaccurate, should produce models acceptable for realistic movements of characters, when a single mesh must be used to represent MTUs with multiple heads.

References

[1] Jack M. Wang, Samuel R. Hamner, Scott L. Delp, and Vladlen Koltun. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Trans. Graph.*, 31(4):25:1–25:11, 2012.

[2] Weiguang Si, Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. Realistic biomechanical simulation and control of human swimming. *ACM Trans. Graph.*, 34(1):10:1–10:15, 2014.

[3] Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Trans. Graph.*, 32(6):206:1–206:11, 2013.

[4] Robson R. Lemos, Jon Rokne, Gladimir V. G. Baranoski, Yasuo Kawakami, and Toshiyuki Kurihara. Modeling and simulating the deformation of human skeletal muscle based on anatomy and physiology. *Computer Animation and Virtual Worlds*, 16(3-4):319–330, 2005.

[5] Dongwoon Lee, Michael Glueck, Azam Khan, Eugene Fiume, and Ken Jackson. Modeling and simulation of skeletal muscle for computer graphics: A survey. *Found. Trends. Comput. Graph. Vis.*, 7(4):229–276, 2012.

[6] J.E. Chadwick, D.R. Haumann, and R.E. Parent. Layered construction for deformable animated characters. In *ACM Siggraph Computer Graphics*, volume 23, pages 243–252. ACM, 1989.

[7] D. Thalmann, J. Shen, and E. Chauvineau. Fast realistic human body deformations for animation and VR applications. In *Computer Graphics International, 1996. Proceedings*, pages 166–174. IEEE, 1996.

[8] J. Wilhelms and A. Van Gelder. Anatomically based modeling. In *Proceedings of the 24th annual conference on Computer Graphics and Interactive Techniques*, pages 173–180. ACM Press/Addison-Wesley Publishing Co., 1997.

[9] J. Teran, E. Sifakis, S.S. Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw. Creating and simulating skeletal muscle from the visible human data set. *Visualization and Computer Graphics, IEEE Transactions on*, 11(3):317–328, 2005.

[10] S.H. Lee, E. Sifakis, and D. Terzopoulos. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Transactions on Graphics (TOG)*, 28(4):99:1–99:17, 2009.

[11] A. Jacobson, L. Kavan, and O. Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics*, 32:33:1–33:12, 2013.

[12] Y. Berranen, M. Hayashibe, B. Gilles, and D. Guiraud. 3D volumetric muscle modeling for real-time deformation analysis with FEM. In *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*, pages 4863–4866. IEEE, 2012.

[13] 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.

[14] M. Millard, T. Uchida, A. Seth, and S.L. Delp. Flexing computational muscle: Modeling and simulation of musculotendon dynamics. *Journal of Biomechanical Engineering*, 135(2):021005–021005, 02 2013.

[15] J. Kohout and M. Kukacka. Real-time modelling of fibrous muscle. *Computer Graphics Forum*, 33(8):1–15, 2014.

[16] C.A. Sánchez, J.E. Lloyd, S. Fels, and P. Abolmaesumi. Embedding digitized fibre fields in finite element models of muscles. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 2(4):223–236, 2014.

[17] J. Tan, G. Turk, and C.K. Liu. Soft body locomotion. *ACM Transactions on Graphics (TOG)*, 31(4):26:1–26:11, 2012.

[18] S.L. Delp, J.P. Loan, M.G. Hoy, F.E. Zajac, E.L. Topp, and J.M.S Rosen. An interactive graphics-based model of the lower extremity to study orthopaedic surgical procedures. *Biomedical Engineering, IEEE Transactions on*, 37(8):757–767, 1990.

[19] 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. *Biomedical Engineering, IEEE Transactions on*, 54(11):1940–1950, 2007.

[20] Snoswell Design. The Ultimate Human Model Data set (UHM), http://www.turbosquid.com/search/artists/snoswell-design, accessed on July 2015.

[21] Shinjiro Sueda, Andrew Kaufman, and Dinesh K. Pai. Musculotendon simulation for hand animation. *ACM Trans. Graph.*, 27(3):83:1–83:8, 2008.

[22] F.E. Zajac. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Critical Reviews in Biomedical Engineering*, 17(4):359–411, 1988.

[23] M. Benjamin, H. Toumi, J.R. Ralphs, G. Bydder, T.M. Best, and S. Milz. Where tendons and ligaments meet bone: attachment sites ('entheses') in relation to exercise and/or mechanical load. *Journal of Anatomy*, 208(4):471–490, 2006.

[24] T. Möller and B. Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.