Utrecht University

Master Thesis

Real-Time Collision-aware Musculoskeletal Model for Virtual Human Animation

Author:
Francis P. Laclé
ICA-3503534

Supervisor:
Dr. Nicolas G. Pronost

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in the

Virtual Human Technology Lab
Department of Information and Computing Sciences

March 2014

Universiteit Utrecht

Declaration of Authorship

I, Francis P. Laclé, declare that this thesis titled, 'Real-Time Collision-aware Musculoskeletal Model for Virtual Human Animation' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____________________________________

Date: ____________________________________

“As computers have become more powerful, computer graphics have advanced to the point where it’s possible to create photo-realistic images. The bottleneck wasn’t, ’How do we make pixels prettier?’ It was, ’How do we engage with them more?’”

Jefferson Han

UTRECHT UNIVERSITY

Abstract

Bètawetenschappen
Department of Information and Computing Sciences

Master of Science

Real-Time Collision-aware Musculoskeletal Model for Virtual Human Animation

by Francis P. Laclé

Keywords: *biomechanics, computational geometry, computer animation, computer graphics, the Ultimate Human Model.*

This MSc project focused on combining a biomechanical musculoskeletal model with a high poly musculotendon model for the lower human body. A low-poly cylinder-based musculotendon unit is constructed that segments muscle and tendon into separate compartments. The shape of high poly meshes is approximated through ray-triangle intersection tests. Point samples are then used to move action lines of the biomechanical model within the high poly model. Once inside, ray-triangle intersection tests are carried out again to get a more accurate approximation of the surface of the high poly model. The method was developed in order to be invariant to spatial and polygonal configurations. Results with 48 musculotendons for the lower body show a drop of $\approx 84\%$ with respect to the amount of vertices when compared to the high poly model.

The project also considered real-time as a criterion and introduced a scalar for the geometrical model allowing each cylinder to be scaled in both longitudinal and latitudinal directions. The amount of information could therefore be increased or decreased to retain real-time performance on faster or slower computer hardware. An experiment with 10 muscles resulted in a runtime of 164Hz on average having a total of 25 constraints. Finally, a custom collision constraint between musculotendons was introduced, coupled with an area preservation technique used during deformation, which takes advantage of the cylindrical construction of the real-time model.

Acknowledgements

First I would like to express my gratitude towards my supervisor. He provided me with all the necessary resources and support that was always accompanied by great enthusiasm. His remarks and suggestions helped me to focus in the right direction throughout the course of this MSc project.

Second, I would like to thank all my family and friends. My family for constant support, and friends for giving me the space and bearing the incremental social distance while I was working on this thesis.

A special thank you goes to Robin, Bernd, and Michal for proof reading this thesis, which was a great help in pointing out some obvious grammatical and notational errors.

I am also grateful to live in the 21^{st} century and in a developed country where one can shower with warm water, which for me served as an idea-incubator for most of the breakthroughs implemented within this MSc project.

The *most important* gratitude goes to Nora, whom I love very much. Being someone quite stubborn, she showed me how to change mindsets and disregard previous erroneous ideas, and also how to strike the right chord between perfection and efficiency against the always persistent arrow of time.

Finally, I would like to address our little one (I'm writing this while you are making funny noises in the background.) Henry, regardless of which path you choose in life, remember that you only need two things, a thirst for knowledge and perseverance.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

CoM	**C**entre **o**f **M**ass
CSS	**C**ross **S**ectional **S**calar
DOF	**D**egree **O**f **F**reedom
FE	**F**inite **E**lement
FEM	**F**inite **E**lement **M**ethod
FFD	**F**ree **F**orm **D**eformation
FPS	**F**rames **P**er **S**econd
FVM	**F**inite **V**olume **M**ethod
GPU	**G**raphics **P**rocessing **U**nit
HACD	**H**ierarchical **A**pproximate **C**onvex **D**ecomposition
IAR	**I**nstantaneous **A**xis of **R**otation
LOD	**L**evel **O**f **D**etail
LSS	**L**ongitudinal **S**egment **S**calar
OGRE	**O**pen-source 3d **GR**aphics **E**ngine
RAGE	**R**eal-time **A**nimation **G**ame **E**ngine
UID	**U**nique **ID**entifier
UHM	**U**ltimate **H**uman **M**odel
XML	e**X**tensible **M**arkup **L**anguage

Symbols

A	vector space defining action-line of a musculotendon	$a_k = (x, y, z)^T$ in $\mathbb{R}^3$
B^R	OpenSim body in relative coordinates	$(x, y, z)^T$ coordinates in $\mathbb{R}^3$
B^W	OpenSim body in world coordinates	$(x, y, z)^T$ coordinates in $\mathbb{R}^3$
C_k	vector space defining each cross section of a_k	$c_i^k = (x, y, z)^T$ in $\mathbb{R}^3$
D_k	vector space defining a deformed cross section of a_k	$d_i^k = (x, y, z)^T$ in $\mathbb{R}^3$
M	main vector space defining a musculotendon	$m_g = (x, y, z)^T$ in $\mathbb{R}^3$
O	transformed coordinates from OpenSim into RAGE	vertex positions in $\mathbb{R}^3$
U	high poly mesh from the UHM	vertex positions in $\mathbb{R}^3$
r_i^k	radius that determines location of c_i^k	$\mathbb{R}$
V	vector space defining via points of a musculotendon	equal to a particular a_k
α	pennation angle	rad
Δ	function for triangle area with given parameters	$\mathbb{R}$
κ	transversal line segment used for collision constraint	two tagged c_i^k's
μ_k	displacement of a_k due to collision	$(x, y, z)^T$ coordinates in $\mathbb{R}^3$
ϕ	alternate interior angle determined by κ	rad
ρ	penetration depth determined by κ	$\mathbb{R}$
σ	permutation on E_k	indices of matches $e^k \in \mathbb{N}$
ω	*free area* for deforming C_k	$\mathbb{R}$

Note: The above lists some of the more important symbols present within this thesis. In addition, arrows for denoting vectors are omitted in order to improve readability.

To my pioneering father. May he rest in peace.

Chapter 1

Introduction

1.1 Motivations and Contributions

In computer science there exists the underlying difficulty of balancing real-time performance versus computational accuracy, as illustrated in Figure 1.1. This tug of war is also prevalent in the fields relevant to this MSc project, namely computer graphics and animation, where visualisation and naturalness of character motion plays a large role.

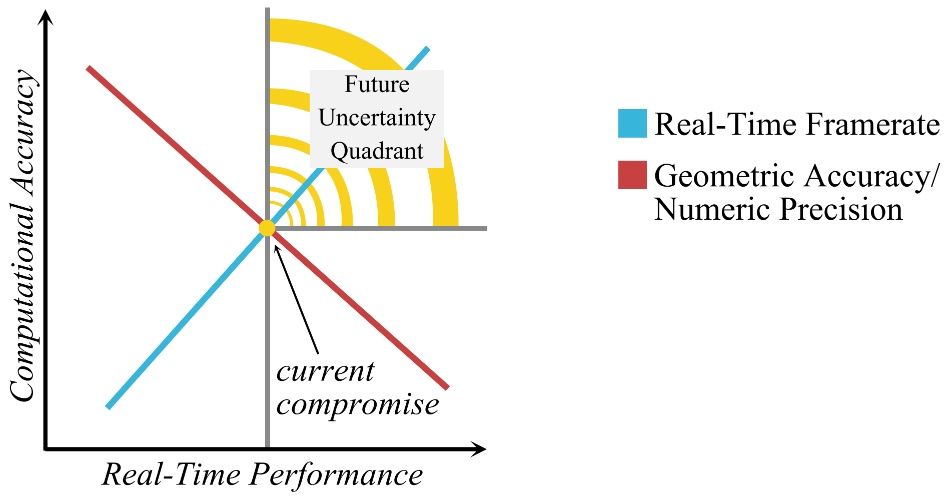


Figure 1.1: Schematic plot of real-time performance of commodity hardware versus computational accuracy. As hardware improves, the middle way (illustrated as the yellow dot) will shift to an undetermined point in the upper right quadrant with ever increasing accuracy. Therefore, this intersection point is dependent on the current state of computational accuracy and graphical performance.

For this MSc project, this meant creating a middle way between having an accurate anatomical musculoskeletal model suitable for FE simulations and real-time performance of the animation/simulation. Because the compromise will shift in the future depending on the hardware being used, it is useful to create a musculotendon model that is scalable with respect to accuracy. This scaling for real-time applications was incorporated into the project, next to a Hill-type biomechanical model that brings with it certain benefits. For instance, muscle paths are usually prearranged in models and thus prevents the tedious work of manually attaching each muscle with guidance of an anatomist. It also allows the validation of approaches that use individual as well as groups of muscles for static-optimisations on top of kinematic-based motions. An even more exciting possibility is using models to help solve dynamic-optimisation problems and achieve realistic physics-driven motion. Another advantage for students and researchers new to the field is that these rigorously tested biomechanical models can be freely obtained through open source simulators such as OpenSim (The National Center for Simulation in Rehabilitation Research, 2010), speeding up collaboration and progress. As Millard et al. (2013) described, biomechanical models usually come with a myriad of data such as coordinates for attachment points, maximum isometric forces, optimal fibre lengths, pennation angles, and so forth.

Current biomechanical models, although accurate and fast for real-time applications (Millard et al., 2013), are also heavily simplified and usually represent musculotendons as polylines without any polygonal geometry. This MSc project is meant to bridge that gap by allowing high poly meshes to be approximated with embedded biomechanical polylines. The use of an underlying biomechanical model removes the need to manually align anatomical musculotendons, while the approximation of high poly models remove the need for mesh preparation, as approximated versions remain simple with segmented area and volume for muscle and tendon compartments. Thus making them ready to be used in real-time applications where volumetrically discretised musculotendons are required.

The original aim of this MSc project was to investigate the possibility of simulating musculoskeletal injuries in real-time using accurate numerical techniques such as FEM(Laclé, 2013). However, as the project progressed it became clear that there are other problems obscured by the real-time simulation of injuries:

1. The inclusion of anatomical polygonal models for higher geometric accuracy.
2. The inclusion of biomechanical models for biomechanical accuracy.
3. Implementing a collision-free system coinciding with real-time performance.

Applications in real-time computer animation that would use biomechanical musculoskeletal models, such as musculoskeletal injury simulation, orthopaedic training, and validation of biomechanical effects for animated characters are thus directly dependent on the solution to these problems. Hence, this project focused mainly on addressing these three problems.

The main contribution of the MSc project and this thesis are a novel way to approximate high poly anatomical meshes of musculotendons in a way that can be discretised into tetrahedrons while retaining a low amount of vertices with as much features of its high poly counterpart as possible. Making it possibly useful for real-time FE simulations. Moreover, a collision detection and response strategy was designed for geometric musculotendon models in real-time with an accurate area and consequently volume preservation technique. Additionally a strict consideration for sustainable real-time deformation was given, using a cylinder-based musculotendon construction that is similar to the early work of Wilhelms and Van Gelder (1997), but created using triangle fans and strips for an optimal GPU footprint. In addition, for finer accuracy on more capable hardware, an adaptive LOD scheme is developed.

What is not included within the scope of this MSc project are a physics-based representation and a FE solver. Although a physics-based representation was developed in Bullet Physics (Coumans, 2013) with a spring-mass scaffold controlled by a couple of kinematic constraints. However, as discussed in Chapter 4, this approach turned out be unsuitable for musculoskeletal modelling and was left out of scope. Nonetheless, the model presented here was constructed in a way that makes it portable for real-time physics-engines.

1.2 Chapters overview

The remainder of this thesis is structured as follows. Chapter 2 briefly discusses previous musculoskeletal models used in the field of computer animation and computer graphics, followed by a brief review of volumetric models developed for FE simulations. Chapters 3 and 4 will thoroughly discuss the musculoskeletal model and the collision strategy that were developed for this project. The elaborate chapters are also meant for readers who are considered as newcomers to this particular area within the field of computer animation and computer graphics. Chapter 5 discusses the results, and finally Chapter 6 ends with concluding remarks with hints at future applications and improvements for the combined musculotendon model.

Chapter 2

Related Work

2.1 Musculoskeletal Models in Computer Graphics

Polygonal musculotendon models for computer graphics and animation are visualised and controlled using different approaches to augment character movement and simulate musculoskeletal deformation (Lee et al., 2010; Ramos and Larboulette, 2009; Aubel and Thalmann, 2001a,b; Albrecht et al., 2003; Mohr and Gleicher, 2003; Lee and Ashraf, 2007; Sueda et al., 2008). One of the first approaches to incorporate muscles into this domain was Chadwick et al. (1989) that used an FFD lattice to kinematically deform skin with underlying muscle. Later, Thalmann et al. (1996) used meta-balls to define the 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 musculotendon construction discussed in Chapter 3, except here cross sections are formed along musculotendon action lines obtained from a biomechanical model. Thalmann et al. (1996) also used ray-casting to extract contours in order to form the skin of the character, which is the basic principle used in this MSc project to approximate features of high poly musculotendon meshes. A year later, Scheepers et al. (1997) published another geometric-based approach that included a procedurally created skeleton and elliptical muscles with added volume preservation. In that same year, Wilhelms and Van Gelder (1997) publishes a different geometric approach that uses discretised cylinders for musculotendons attached unto skeleton meshes. It was one of the first works to use the concept of a pivot point, which is equivalent to the contemporary labelled via point. Their work was also the primary inspiration for the cylinder-based musculotendon model presented here, as it brought with it some practical real-time benefits such as radii-based deformation, as discussed in Chapter 4.

2.2 Musculotendon Models for Volumetric Deformation

Early work of Chen and Zeltzer (1992) showed that FEM could be combined with a classical biomechanical model for calculating muscle forces, which paved the way for a new generation of FEM and FVM-based approaches such as Zhu et al. (1998); Hirota et al. (2001); Lemos et al. (2001); Teran et al. (2003, 2005); Lee et al. (2009). Regarding FEM mesh generation, Jacobson et al. (2013) introduced in 2013 a novel method that is slightly similar to the winding-principle used in the match detection technique discussed in subsection 3.5.3.2 for the creation of volumetric meshes. While their method outputs FEM-ready meshes that are watertight, they still contain at least three times more vertices than the original input mesh, and thus not suitable for real-time applications. Berranen et al. (2012) used both FEM meshes and a modified Hill model for real-time deformation analysis, where contractile muscle forces operate between adjacent nodes of the FEM mesh. This is one of the few recent work found that combined a Hill-type biomechanical model with a volumetric FEM mesh with promising real-time performance, albeit the result included just a single musculotendon. Around half a decade earlier, Lee et al. (2009) published an elaborate upper body model, using the same dataset as this MSc project, namely the UHM (Snoswell, 2003). The UHM geometry was processed, as is the case here (refer to subsection 3.5.1), and volumetrically discretised for FEM. They also incorporated a Hill-type biomechanical model on the action line for the computation of muscle forces for forward dynamics.

In general, FEM is still considered to be too computationally expensive for real-time simulations. Even more so when not one but several musculotendon units that are attached to the skeleton have to be simulated in concert, with active collision detection and response strategies next to the FE solver. To start looking at these challenges, it is handy to set a lower bound for the least amount of vertices that a musculotendon model should contain. Such a lower bound is discussed in subsection 3.6.2 having the goal of allowing a very low amount of vertices to be considered but still with an acceptable limit of accuracy for real-time applications.

Work from Blemker and Delp (2005) also created volumetric meshes by mapping a template hexahedral mesh to another target hexahedral mesh using projections and Delauney triangulation. 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 similar to the UHM enhancement technique discussed in section 3.5. Fibre geometries were also created separately as rational Bezier spline curves with template schemes for parallel, pennate, and fan based muscle fibre architectures. Blemker and Delp (2005) concluded that this three dimensional approach to musculotendons such as the Gluteus group, Psoas, and Iliacus exhibited enough diverse behaviours that it provided an added value of accuracy for estimating moment arms of muscle fibres; at least for this tested group of musculotendons. While this approach could ultimately render the use of classical Hill-type models, which are polyline-based with via points obsolete, it is still considered early work particularly when real-time usage of classical models are increasing in accuracy as the latest benchmark result of Millard et al. (2013) shows.

Similarly, work of Kohout et al. (2012) took a combined approach labelled as surface-first and fibres-first. With surface-first, volume preservation is solved followed by decomposing the muscle compartment into fibres, where they are arranged using the template-based approach from Blemker and Delp (2005). Fibres-first starts from muscle-decomposition and represents fibres as a spring-mass system. They claim to solve collisions for both approaches, but unfortunately not much detail is given on how this is done. Furthermore, their result is not fast enough for real-time applications requiring above 60Hz performance on commodity hardware. To improve realism, recent work of Sánchez et al. (2014) stepped entirely away from template-based approaches and proposed a new workflow for embedding subject-specific fibre fields in FE 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 references and others such as Tan et al. (2012) imply, the next frontier is indeed modelling on the scale of fibres, however it could still take some years before this can be considered a possibility in the real-time domain.

Chapter 3

The Musculoskeletal Model

3.1 Anatomical Accuracy and Real-Time Performance

This project aims to develop a model that approximates anatomical accuracy and is capable of real-time or interactive performance. To help reach this objective, the following two criteria are considered:

- The model should include at least one anatomical model and one biomechanical model to help approximate anatomical and physical correctness.
- The geometrical model has to be flexible i.e. scalable with respect to its level of detail (LOD). On faster computer hardware this allows more geometrical information to be processed in real-time.

The process that culminated in the end result of each of the above mentioned criteria is described in the following sections.

3.2 The Biomechanical Musculoskeletal Model

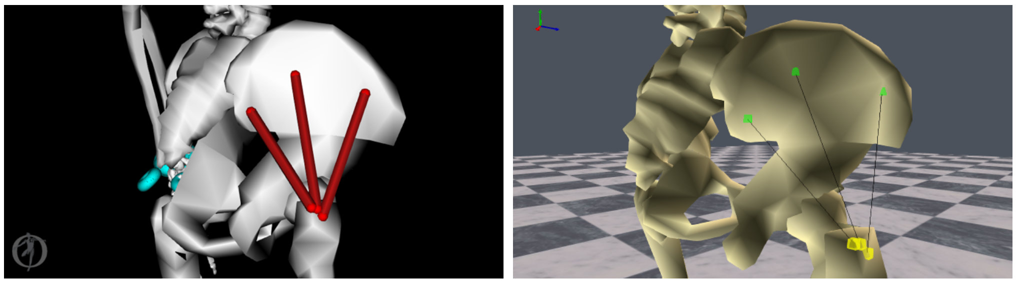
Biomechanical models include data that are relevant to the physical aspects of the musculoskeletal model such as positions of attachment points of muscles relative to their attached skeletal bone, length and arrangement of muscle fibres, isometric forces, etc. For this project it was decided to use the combined model from Menegolo (2011) that was developed for the open source software known as *OpenSim* (The National Center for Simulation in Rehabilitation Research, 2010; Delp et al., 2007; Seth et al., 2011)[1].

[1]OpenSim allows researchers, students, and health professionals to develop and test biomechanical models for dynamic simulations of movement.

It is based on research that was carried out for both the upper and lower halves of the human body, which became the main reason to include it in the project. The model of Menegolo (2011) is backed up by the following associated publications:

1. An interactive graphics-based model of the lower extremity to study orthopaedic surgical procedures (Delp et al., 1990).
2. A planar model of the knee joint to characterize the knee extensor mechanism (Yamaguchi and Zajac, 1989).
3. A dynamic optimization solution for vertical jumping in three dimensions (Anderson and Pandy, 1999).
4. Dynamic optimization of human walking (Anderson et al., 2001).
5. A model of the upper extremity for simulating musculoskeletal surgery and analyzing neuromuscular control (Holzbaur et al., 2005).

The model of Menegolo (2011) also includes polygonal meshes for the bones of the human skeleton. Using the same meshes for the skeleton came in handy while testing and verifying the attachment of muscles to make sure they are correct, especially when the skeleton meshes were scaled and oriented semi-automatically to fit the current animated character. Placement of points relative to transformed skeleton meshes could thus be visually compared with the result in OpenSim. Figure 3.1 shows one example where this is the case.



(a) Musculoskeletal Model in OpenSim. **(b)** Musculoskeletal Model in RAGE.

Figure 3.1: Screenshots from OpenSim (a) and custom implementation of a Musculoskeletal Model (b) within the Real-time Animation Game Engine (RAGE) from Utrecht University. Visible are three muscles from Menegolo (2011) rendered as action lines. Green cubes represent attachment points for each muscle's origin, while yellow cubes represent attachment points of each muscle's insertion. All three muscles collectively represent the Gluteus Medius muscle on the right side of the sagittal plane.

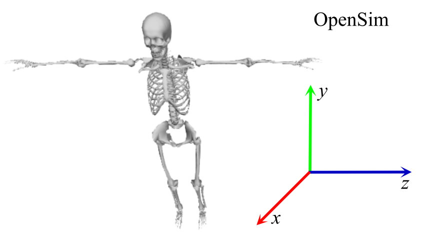
One drawback of Menegolo (2011) that became apparent was that not all parameters were included for the upper body, notably the parameters *mass* and *centre of mass* were missing from all skeleton meshes pertaining to the upper body. These would have been beneficial for use in a physics-based simulator. The model does include a complete set of upper and lower body skeleton meshes that were used within this project, however solely the biomechanical muscle data of the lower body was included as first tryout. This is because the lower body, which includes the pelvis, is the predominant actuator of active and passive forces during locomotion of the bipedal human body, and this ability to move from one place to another plays an important part in the synthesis of human movement. Moreover, data from upper body biomechanical models can always be included at a later stage. Appendix A gives a complete list of all the muscles used in this project and Table 3.1 lists all six biomechanical parameters that were used for the lower body.

Parameter name	(Muscle) Dataset	Object Type
maximum isometric force	Thelen 2003 & Schutte 1993	musculotendon
optimal fiber length	Thelen 2003 & Schutte 1993	musculotendon
tendon slack length	Thelen 2003 & Schutte 1993	musculotendon
pennation angle	Thelen 2003 & Schutte 1993	musculotendon
centre of mass	Menegolo (2011) (lower body)	OpenSim body
mass	Menegolo (2011) (lower body)	OpenSim body

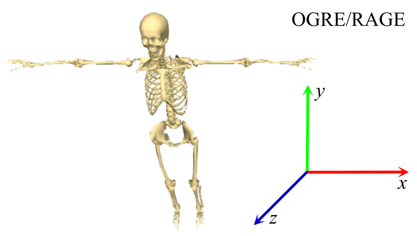
Table 3.1: Biomechanical parameters from Menegolo (2011) included in this project.

3.3 The Polygonal Skeleton Meshes

The polygonal meshes from Menegolo (2011) representing the human skeleton had to be prepared and converted prior being able to use them in OGRE/RAGE. The conversion process started with transforming the skeleton meshes from the Cartesian coordinate system in OpenSim to the Cartesian coordinate system used in OGRE/RAGE. OpenSim utilises the *laboratory coordinate system* (Lund and Hicks, 2012) that is different than what is used in motion capture systems or popular 3D authoring software. Figure 3.2 illustrates orthonormal basis vectors of OpenSim versus ones used in OGRE/RAGE.



(a) Orthogonal basis vectors in OpenSim.



(b) Orthogonal basis vectors in OGRE/RAGE.

Figure 3.2: Different orthogonal bases in OpenSim compared to OGRE/RAGE. OpenSim uses the *laboratory coordinate system* where the z axis is switched with the x axis.

The orthonormal transformation O equals the multiplication of coordinates of every vertex of every skeleton mesh in OpenSim and is given by the 3x3 rotation matrix:

$$O = \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} \tag{3.1}$$

where $(x_o, y_o, z_o)^T$ are the coordinates of vertices of the skeleton meshes in OpenSim and $(x_r, y_r, z_r)^T$ are the new transformed coordinates of these vertices, ready to be used.

3.3.1 Attaching the Skeleton Meshes

Once the coordinates were transformed and exported came the task of attaching the polygonal meshes unto an animated character. Throughout this project, one of the characters of the Virtual Human Technology Lab, nicknamed *Ralph*, was used as the character that incorporated the musculoskeletal model. Ralph comes with its own joint (or bone) hierarchy for animation[2]. As reference, Ralph's complete bone hierarchy is given in Appendix B. In order to attach each skeleton geometry on a particular joint or bone of Ralph, the two hierarchies first had to be mapped. This mapping between the two separate hierarchies occurred offline as both OpenSim and OGRE have different ways of representing each.

The OpenSim framework stores attached skeletal geometry in so-called *bodies* provided by the *Simbody* toolkit. Bodies are linked through OpenSim's *joint* system. Each can have multiple geometries attached forming a hierarchical tree where each linked body represents one node and each node can consist of zero or more leafs that represent skeleton meshes[3].

All the relationships between parents and children were mapped using the Extensible Markup Language (XML) and added via OGRE using `Ogre::TagPoint` objects. The following snippet gives a snapshot of the end result:

```xml
<ogre_bone id="15">
    <opensim_mesh ogre_child="16" opensim_child="24">22</opensim_mesh>
    <opensim_mesh ogre_child="16" opensim_child="24">23</opensim_mesh>
</ogre_bone>
<ogre_bone id="16">
    <opensim_mesh ogre_child="-1" opensim_child="-1">24</opensim_mesh>
    <opensim_mesh ogre_child="-1" opensim_child="-1">25</opensim_mesh>
    <opensim_mesh ogre_child="-1" opensim_child="-1">26</opensim_mesh>
    <opensim_mesh ogre_child="-1" opensim_child="-1">27</opensim_mesh>
```

[2]To prevent confusion, the term *bone hierarchy* will be used throughout this thesis to represent the matrix palette skinning technique typically used for skinned animations in OGRE and RAGE.

[3]The OpenSim hierarchy of Menegolo (2011) is visualised in Appendix C for a complete overview

```
        <opensim_mesh ogre_child="-1" opensim_child="-1">28</opensim_mesh>
        <opensim_mesh ogre_child="-1" opensim_child="-1">29</opensim_mesh>
        <opensim_mesh ogre_child="-1" opensim_child="-1">30</opensim_mesh>
        <opensim_mesh ogre_child="-1" opensim_child="-1">31</opensim_mesh>
    </ogre_bone>
    <ogre_bone id="17">
        <opensim_mesh ogre_child="18" opensim_child="37">32</opensim_mesh>
    </ogre_bone>
```

Due to the difference in hierarchies, an XML schema proved to be versatile in the creation of a quick mapping scheme. Unique identifiers (UID's) of skeleton meshes could be added to specific bones satisfying the 1 : N cardinality requirement, where e.g. both tibia and fibula would fall under the right knee bone of Ralph. In addition, relationships between parent and child could be mapped by adding other UID's as children. -1's were added for user friendliness, indicating unused values.

3.3.2 OpenSim Bodies in Cartesian Coordinates

As mentioned in the previous subsection, OpenSim encapsulates skeleton meshes within body objects for their physics-based simulator. Each body is defined in internal coordinates that are generalised in order to help solve multi-body dynamics (Sherman and Eastman, 2012), shown in Figure 3.3. As each coordinate of a body is linked and parameterised to another body in a relative frame of reference (Hicks, 2012b), it was useful to know where OpenSim bodies were positioned relative to OpenSim's world origin. This unlinking of bodies came in handy during the skeleton alignment algorithm described in subsection 3.3.3 and also when verifying the transformation of custom OpenSim joints.

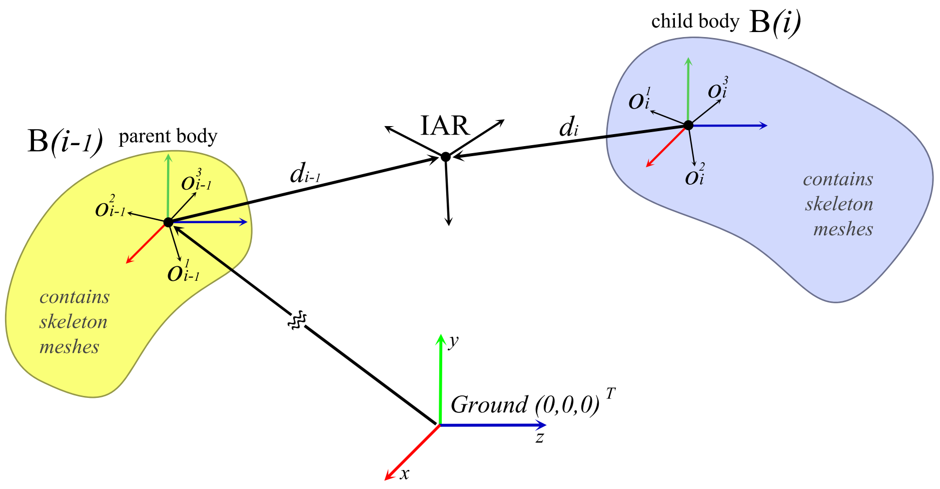


Figure 3.3: Kinematic relationship between two bodies in OpenSim defined in a relative frame of reference and connected by a joint, the instantaneous axis of rotation (IAR). Adapted from Christophy (2010); Hicks (2012b).

The Cartesian coordinates of each OpenSim body are given by:

$$\mathrm{B}^{\mathrm{W}}(i) = \begin{cases} \mathrm{B}^{\mathrm{R}}(0) & \text{if } i = 0 \\ \mathrm{B}^{\mathrm{R}}(i-1) + \mathrm{B}^{\mathrm{W}}(i-1) - \mathrm{B}^{\mathrm{R}}(i) & \text{if } i > 0 \end{cases} \tag{3.2}$$

where B^{W} gives world coordinates for position and rotation of a bodies, $i \in \mathbb{N}$ is an index in the body's hierarchy, and B^{R} represents the relative coordinates with respect to the IAR. Which define the displacement vectors d and co-rotational bases $\{o^1, o^2, o^3\}$ that both displace and rotate each body about the connecting point of the joint (the IAR) in Figure 3.3.

OpenSim also includes complex biomechanical joints next to the traditional variety of mechanical joints (such as welded, pin, slider, etc.) to model joints more accurately (Seth et al., 2011). One type of complex joint uses natural cubic splines to let the IAR interpolate along curved paths. One example for rotation of the knee along xy-plane with a fitted natural cubic spline is shown in Figure 3.4.

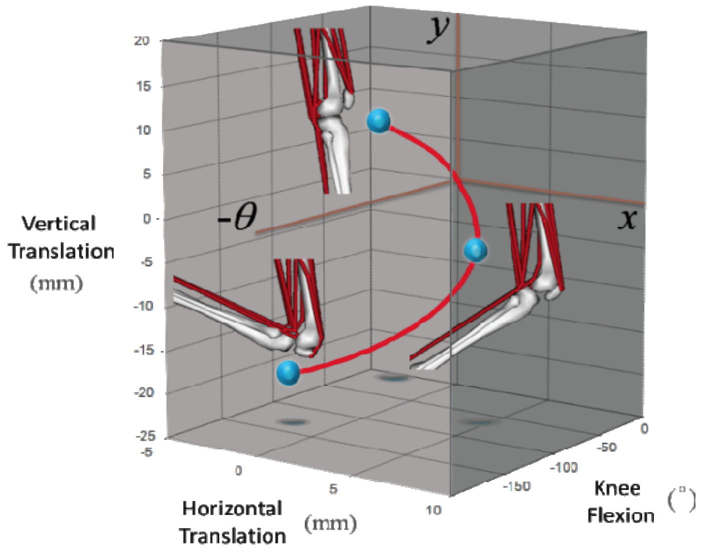
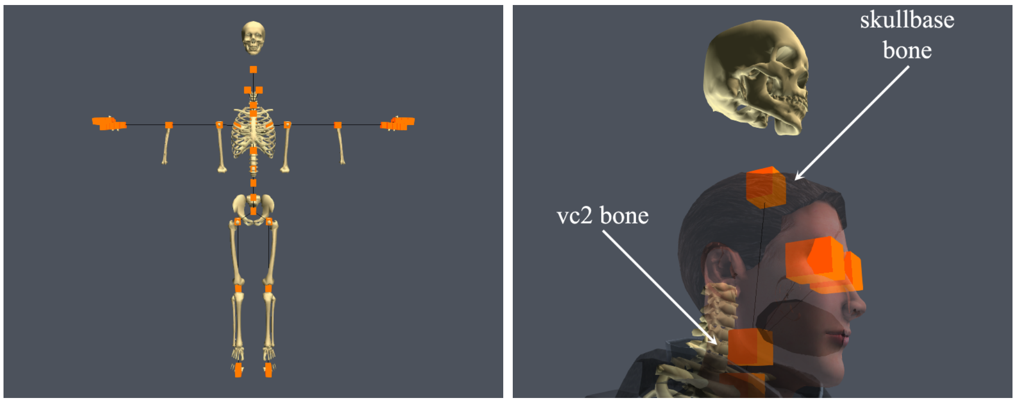


Figure 3.4: xy planar displacement of the fibula and tibia interpolated along a spline curve with respect to the knee joint angle θ. The position of the knee is illustrated at three flexion (negative) angles, namely 0°, 60°, and 120°. From Seth et al. (2011).

The natural cubic spline function used in OpenSim had to be reimplemented in this project in order to complete the unlinking of bodies to acquire the world positions and orientations of all subsequent children bodies. With all bodies unlinked it is possible to compute the length of each skeleton mesh for the semi-automatic skeleton alignment described in the next subsection.

3.3.3 Semi-Automatic Skeleton Mesh Alignment

The next step, once the skeleton was attached, was to align each mesh along Ralph's bone hierarchy. A couple of problems that needed to be addressed, shown in Figure 3.5, became apparent after the attachment routine. The first problem was dealing with different starting poses. The starting pose used in Menegolo (2011) is the *relaxed-pose* with both arms hanging loose, while in Ralph the starting pose is the more common *t-pose*. Next, scaling issues were also noticeable due to different anthropometric proportions in the hierarchical body compared to Menegolo (2011). Each skeleton mesh is also relatively positioned from its own world origin, which caused differences at certain areas.



(a) Front-view with complete skeleton attached. **(b)** Close-up of Ralph's head with floating skull.

Figure 3.5: The bone hierarchy is visualised as black line segments connected by orange cubes. (a) shows the difference in initial poses between OpenSim and Ralph, scaling issues such as gaps at the knee joint, and a more obvious problem highlighted in (b) with the skull base meshes due to relative transformations present in mesh files.

In addition, certain bones were not correctly placed anatomically. For instance, the `skullbase` bone was artistically placed at the top of the head to include all vertices during the skinning process, however this placement is anatomically incorrect because it lies too much on the top end of the skull. A more representative location would be in the centre of the skull, placing the `skullbase` bone equally distanced between the anatomical occipital and frontal bones. One solution for the skull base meshes is to simply reattach them on another bone, such as the last neck bone `vc2`. The hierarchy map written in XML is flexible enough to allow this, but unfortunately preliminary tests showed that the skull was still not entirely at the centre of Ralph's head. These unresolved complications resulted in the development of a semi-automatic approach where certain transformations would be initially carried out online and supplemented with manual adjustments.

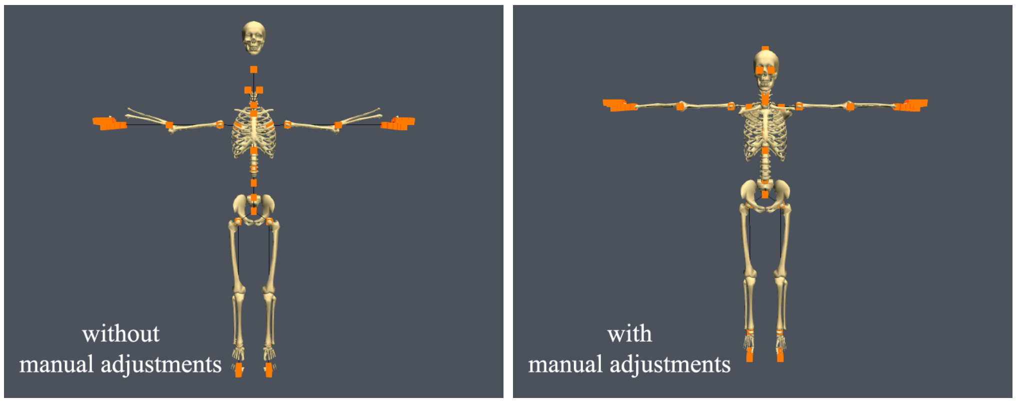
The automatic alignment of skeleton meshes depended on the two transformations *rotation* and *scale* as these required less granular adjustments. To address scale, the transformation from internal coordinates to Cartesian coordinates for OpenSim bodies opened up the possibility to calculate anthropometric distances in Menegolo (2011) and compare these with distances from the bones of Ralph. A scaling ratio was then calculated and applied in the transformation of each mesh. To address rotation, the shortest rotational arc between bones and bodies from the separate hierarchies were calculated using quaternion functions from OGRE[4].

The final result can be seen in Figure 3.6a. Scaling has improved for some areas such as the gap at the knee joints, but there are still visible problems with rotation (e.g. with the arm attachments) due to relative transformations present in the skeleton mesh files. Figure 3.6a makes clear that although the automatic scaling and rotation techniques improves initial errors, the following pair of problems still remains to be solved:

- Relative displacements present in each skeleton mesh file.
- Artistic differences in the bone hierarchy compared to the OpenSim hierarchy.

Furthermore, one requirement with automatic alignment is that each transformation requires a link with a child in both hierarchies in order to compare distances. Such cases are not always present, such as skeleton meshes of the forefoot that are attached to leaf nodes in the bone hierarchy of Ralph. Hence, to resolve the above mentioned issues, it was decided to add an extra pre-processing step that would adjust all three transformations, namely translation, rotation, and scale with hard coded values. For translations, manual adjustments were already planned as automating translations requires techniques to automatically determine boundaries of each skeleton mesh to determine how far away a mesh should reside from a bone. In addition, translations require fine control (on each separate axis) to achieve a good result. Manual adjustments also created the possibility to interactively alter each mesh, which speeded up the manual alignment process. The outcome of the semi-automatic alignment process is visible in Figures 3.6b and 3.7.

[4]As a reference, the function that computes the rotational difference in quaternions can be studied in Torus Knot Software Ltd (2013).



(a) Front-view showing automatic alignment. **(b)** Front-view showing semi-automatic alignment.

Figure 3.6: Automatic skeleton alignment with respect to scale and rotational transformation is not enough as shown in (a). The final result is shown in (b). Adjustments can be made for each of the three transformations with three DOF totalling nine DOF.

Ralph's skin-transparency was also set to be interactively adjusted, which demonstrated to be an effective way to visualise penetrations of the inner skeleton and later on the musculoskeleton.

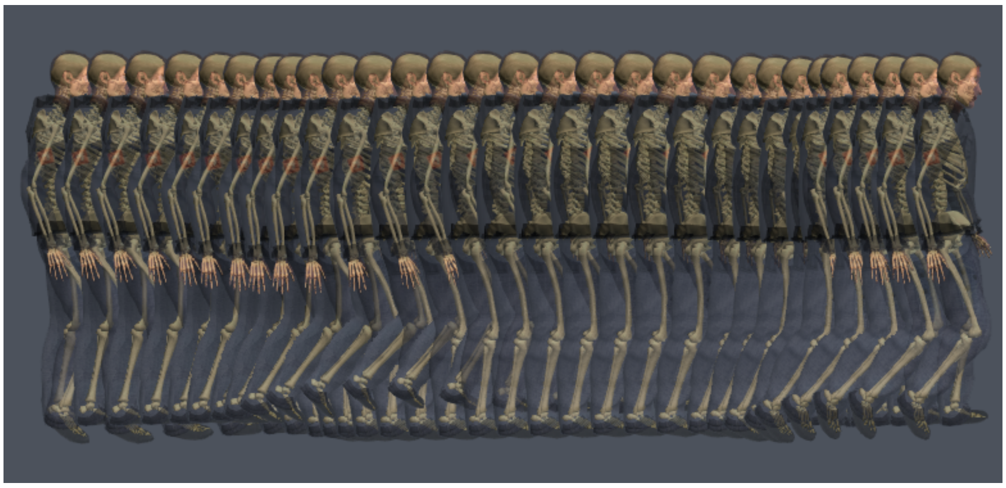


Figure 3.7: Once the semi-automatic skeleton alignment was completed on the initial pose, an animation for Ralph was loaded to further look for penetrations with the mesh of Ralph, i.e. if the skeleton was fully contained. As a visual aid, an extra widget was added to RAGE's GUI to interactively change the transparency of Ralph's mesh in OGRE. The above figure shows an animation history plot of Ralph loaded with a walk-cycle animation and Ralph's mesh transparency set to 50%.

3.4 A Simple Geometric Musculotendon Model

The geometrical musculotendon model combines muscle and tendons into a single geometrical object. Each musculotendon can be defined by the following vector space M

with dimension n as the index counter is positive and zero-based:

$$M = \left\{m_0, m_1, ..., m_{n-1} \mid m_g \in \mathbb{R}^3\right\} \tag{3.3}$$

where each element m_g is a position vector in $\mathbb{R}^3$ Cartesian coordinates. The initial shape of the musculotendon can be thought of as a closed cylinder with the starting point, the *origin point*, lying at the centre of the top cap. This point is attached relative to a specific bone in the bone hierarchy, which can also contain other siblings representing other musculotendons or other skeleton meshes. The end point, or *insertion point*, at the centre of the bottom cap is attached relative to either the same or another bone in the hierarchy. OpenSim refers to these two attachments as *fixed points* in their framework (Hicks, 2012b)[5]. All attachments are children of a bone where the attached skeleton meshes are part of the same OpenSim body as the musculotendons in the data of Menegolo (2011)[6].

3.4.1 Adding Action Lines and Via Points

Biomechanical models, such as those used in OpenSim (The National Center for Simulation in Rehabilitation Research, 2010), usually simplify a musculotendon as a line or a set of connected lines, referred as the *action line* between the origin and insertion coordinates (Delp et al., 1990) that defines each muscle's path. The action line, shown in Figure 3.1a, lies at the core of the models in Menegolo (2011), which for some parts was designed to match moment arms that where measured experimentally (Holzbaur et al., 2005). An action line is defined as a vector space of subset A in this thesis and is always contained within the boundary of M, hence $A \subsetneq M$, and given as:

$$A = \left\{a_0, a_1, ..., a_{n-1} \mid a_k \in \mathbb{R}^3\right\}, |A| \geq 2 \text{ and} \tag{3.4}$$

where each a_k is a position vector on the action line representing coordinates in $\mathbb{R}^3$. The first and last element of A represent the attachment coordinates linking musculotendons to their respective meshes of the skeleton. During animation, the origin and insertion coordinates are updated kinematically as each attached skeleton mesh also moves with the character's animated bones.

Besides origin and insertion attachments, the biomechanical model of Menegolo (2011) includes an additional set of constraints that are supported by The National Center for Simulation in Rehabilitation Research (2010), such as *wrapping surfaces*, *via points*, and

[5]Points are sometimes used as simplified constructs in real-time simulators. In actual biological systems musculotendons are usually attached at sites, known as *entheses*, and not at a single point (McGonagle and Benjamin, 2013).

[6]Refer to section 3.3 for more information on the bone hierarchy used for this project.

moving muscle points that allow the model to approximate empirical data more closely. A brief description of each is given in Table 3.2.

Constraint Type	Description
Wrapping Surface	Simple primitives (spheres, ellipsoids, cylinders, and torii) that allow the action line to wrap around the primitive's surface. Uses two extra points placed on tangents of the primitive that define the point of contact initiation and of contact breaking.
Via Point	Extra attachment points fixed to an OpenSim body and activated when a specific parameter (such as a joint's knee angle) lies in a given range.
Moving Muscle Point	The coordinates of this constraint are passed as functions such as natural cubic splines instead of constants. Useful for moving the action line when an articulate body is being flexed or extended and a wrapping surface is not sufficient to recreate the proper motion.

Table 3.2: Summarised descriptions of constraints within OpenSim (Hicks, 2012a).

For this project it was decided to include solely the via point fixed constraint due to the following reasons. An action line, as used in OpenSim is not enough to create a polygonal representation of a musculotendon. To approximate a more accurate shape of a musculotendon, an enhancement process was also integrated into the musculotendon model[7]. The exclusion of other constraints, such as moving muscle points, gave the geometrical model more freedom to realign each action line A within a high poly mesh. Furthermore, no physics-driven biomechanical simulation was required for this project and therefore no accurate force-length and force-velocity relationships were needed, i.e. the derivative of an action line's length was not dependent on neither forces driving each musculotendon unit nor static optimisations. For that reason, the inclusion of fixed via points as a way to approximate the curve of A before executing the enhancement routine was sufficient. Finally, Rankin and Neptune (2012) states that the algorithms used for wrapping surfaces are computationally expensive and thus computational overhead was also reduced by excluding this type of constraint.

As briefly described in Table 3.2, via points also exhibit a limited activation range. As a result, via points are not present all the time on A. Instead of switching via points on and off during gait, it was opted to discard the activation range and activate all via

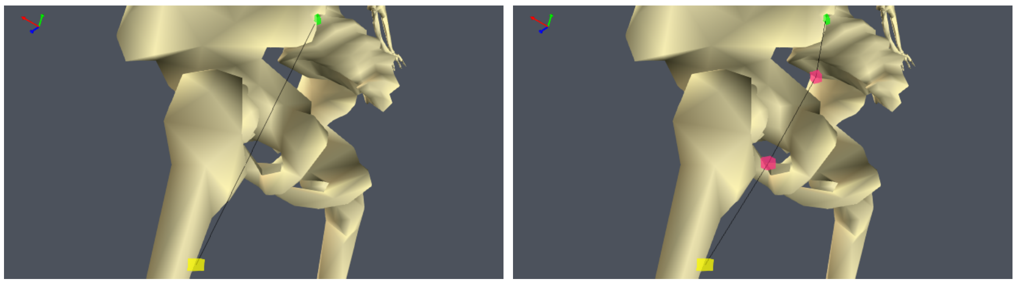
[7]Refer to section 3.5 for a thorough overview of enhancement with the UHM.

points by default. This would not only simplify the simulation's update loop but more importantly set the amount of points along the action line as a constant finite set. This brings two benefits, namely:

1. LOD techniques for the action line can be introduced now that the model uses a fixed amount of via points.
2. Musculotendons defined as a system of spring-masses have a fixed amount of kinematic nodes from the get-go, which is beneficial for real-time physics engines.

The drawback is that for some intervals, there is the risk that A would not wrap around a skeleton mesh and would instead cause penetration. However, seeing that the geometrical model is represented as a simple polygonal mesh, a collision management system would have to prevent penetration for all vertices and this would automatically solve all wrapping issues. In addition, the ability to include LOD techniques on a constant amount of points to adapt the surface and volume of the musculotendon brings more accuracy to the geometrical model than not including LOD but dynamically adding and removing points on and off of A.

With the addition of via point constraints, the model now provides finer kinematic control over the direction of each muscle along the skeleton. Via points also approximate the continuity of a muscle more closely than a mere line segment effectively turning each muscle's action line into a polyline as shown in Figure 3.8.



(a) Action line without via points. **(b)** Action line with via points.

Figure 3.8: Screenshots from RAGE showing the action line of one section of the Gluteus Maximus muscle from Menegolo (2011) without via points (a) and with via points (b). The via points, visualised as pink cubes, add extra kinematic control to the geometrical model and better approximation to the curve of the action line.

Not all musculotendons present in Menegolo (2011) contain via points, the Gluteus Medius, Rectus Femoris, and the Pectineus are some examples. Therefore, the least amount of elements that can be contained in A is two for the origin and insertion points. When via points are present in a muscle, each element v from the set V lies always at another point on the action line in the geometrical model presented in this thesis meaning $V \subseteq A$.

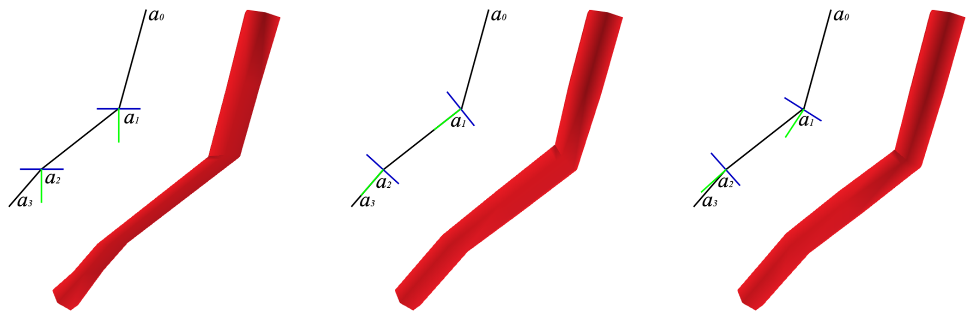
3.4.2 Adding Volume with Cross Sections

The inclusion of via point constraints in the model is not enough to give each muscle volume. At this moment, M is still a polyline defined by A. To actually add volume, extra vertices are needed for its surface. The initial geometric state of the model, a polygonal mesh constructed as a closed cylinder, would already include two cross sections for the top and bottom caps. To also account for muscles that contain via points, each closed cylinder can be split into smaller segments by introducing *bisecting* cross sections at each element of A. The subset 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:

$$C_k = \left\{c_0^k, c_1^k, ..., c_{n-1}^k \mid c_i^k \in \mathbb{R}^3\right\}, C_k \subsetneq M \text{ with} \tag{3.5}$$

$$C_k \cap A = \varnothing. \tag{3.6}$$

With C_k defined as being another proper subset of M, the relationship $A \subsetneq M$ is now also conformed. The cardinality of C_k is also proposed as $|C_k| \geq 6 \wedge |C_k| = 2\aleph$ for reasons discussed in subsection 3.5.3.2. Another area of C_k that is also worth defining is the new direction of the *up-vector* for a_k. This would allow the segmented cylinder to "follow" the curve of A more closely, as shown in Figure 3.9b and 3.9c.



(a) Parallel to y (default). **(b)** Parallel to $(a_{k+1} - a_i)^T$. **(c)** Parallel to $(a_{k+1} - a_{k-1})^T$.

Figure 3.9: 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 $0 < k < (|A| - 1)$.

The default up-vector is shown in Figure 3.9a, which in this case coincides with the $-y$ axis for this particular muscle, the Gluteus Maximus 3. This is because each muscle is built starting at the origin and ending at the insertion point, and for this particular musculotendon from Menegolo (2011), the origin point lies above the insertion point in the y direction. Hence the direction is downwards in the $-y$ direction given as $(0, -1, 0)^T$. Next is Figure 3.9b showing up-vectors pointing in the forward direction parallel to a segment of the action line between a_{k+1} and a_k. The last tried approach, Figure 3.9c, includes information from both sides of a_k with the vector $(a_{k+1} - a_{k-1})^T$ resulting in an approximation that is centralised and therefore more robust when dealing with acute angles between two other cross sections.

3.4.3 Including Tendons into the Model

Before the model can be recognised as a musculotendon model, a geometrical representation of the tendon must be included. Unlike graphical representations of musculotendons such as Sueda et al. (2008); Geijtenbeek et al. (2013); Teran et al. (2005); Scheepers et al. (1997), tendons are considered as a single piece lying on one side of the musculotendon unit in classical models such as Hill's and Zajac's (Zajac, 1988). However, like their graphical counterparts most real tendons are located on each side of a muscle (Encyclopedia Britannica, 2014b). The biomechanical models in Menegolo (2011) include parameters that would satisfy the equations in classical models, yet are not sufficient to provide an accurate geometric representation for each tendon. For instance, the tendon on the side of the origin might take 80% of the total tendon length with the remaining 20% by the tendon on the insertion side, or vice versa. Therefore, the length distribution of each individual tendon remains unknown. In addition, the parameter *tendon slack length* not only represents the total length of both tendons but also combines the length of free and aponeurotic tendons (Delp et al., 1990), and there is no distinction as to where the free tendon stops and the aponeurotic tendon begins. As a result, no dataset was found for this project that included lengths for each separate tendon and for each component of the tendon, dividing each slack length parameter into the two separate lengths. Figure 3.10 illustrates a schematic representation of the problem.

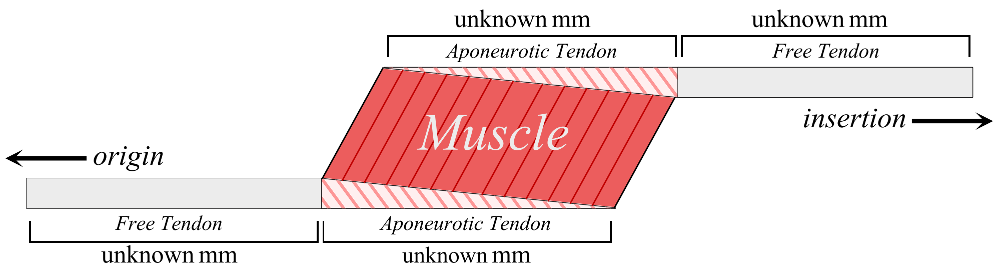


Figure 3.10: Classic musculotendon models such as Zajac's (Zajac, 1988) combine all four unknown lengths into a single term for the tendon. While accurate in biomechanics, these unknowns make it challenging to define an anatomical three dimensional model of a musculotendon with two individual tendons. Adapted from Hoy et al. (1990).

Without the necessary biomechanical data to calculate the correct anatomical length of each individual tendon, it became a challenge to define a way of incorporating tendons into the model. Although anatomically inaccurate, it was 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 in anatomical illustrations such as Taylor (2013). Hence, within this model, the length of the tendon is computed backwards, starting at the insertion point and traversing along the action line in the direction of the origin point for all musculotendons. To compute the tendon length, two calculations must be preceded beforehand, which are:

1. Calculate the rest length of the musculotendon.
2. Calculate the scaled rest length of the musculotendon.

This generates a total of four separate variables for lengths, namely the length of the action line and the length of the scaled action line, denoted by l^A and l^{AS} respectively, the rest length denoted by l^R, and finally the scaled rest length denoted by l^{RS}. Scaled versions of l^A and l^R are used in this model because the skeleton was transformed during the semi-automatic alignment process and so were all the attachment points in order to fit Ralph. It is also worth noting that due to the use of a simple kinematic system for animation, the model does not include any static nor dynamic optimisation pipelines. This fact combined with knowledge from Nordin and Frankel (2012); Benjamin et al. (2006) that the elastin proportion is approximately 2% and maximum strain around 6% led to the decision that tendons are assumed to be infinitely stiff in this model. The assumption that the tendon is rigid would allow 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 given in Delp et al. (1990) to derive the rest length l^R as follows:

$$l^R = l_s^T + \left(l_o^F \cos\alpha\right) \tag{3.7}$$

where l_o^F represents the optimal fiber length and α the pennation angle present in pennate musculotendons. All three parameters including the l_s^T were used from Menegolo (2011). To derive the scaled rest length l^{RS}, the ratio between the $|A|$ in Menegolo (2011) and the scaled $|A|$ in OGRE must be calculated first. The generic calculation for the $|A|$ is given by the sum of the Euclidian length between two a_k elements:

$$\sum_{k=0}^{n-2} \left| \, (a_{k+1} - a_k)^T \, \right| \tag{3.8}$$

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:

$$l^{RS} = \frac{l^R l^{AS}}{l^A}. \tag{3.9}$$

With l^{RS} assigned, the scaled tendon slack length l_s^{TS} can also be obtained with:

$$l_s^{TS} = \frac{l^{RS} l_s^T}{l^R}. \tag{3.10}$$

After the UHM enhancement, which will be discussed in section 3.5, it was found that for most musculotendons the tendon length is longer than a noticeable free tendon length in the high poly meshes. For the purpose of representation, it was then decided to use $\frac{1}{2} l_s^T$ which gave visually a more correct result but still remained anatomically inaccurate. Table 3.3 demonstrates a result with the Tibialis Anterior of the right leg.

l_o^F	α	l_s^T	l^A	l^{AS}	l^R	l^{RS}	l_s^{TS}	$\frac{1}{2} l_s^{TS}$
0.098	0.087	0.223	0.303	0.292	0.320	0.309	0.215	0.107

Table 3.3: For the Tibialis Anterior, its scaled tendon slack length l_s^{TS} was found to be ≈ 0.215 with half of that length being ≈ 0.107 using Equations 3.7 till 3.10. These were computed after the skeleton was attached and aligned to fit Ralph. All values are given in metres and rounded up to millimetres for this thesis.

Equation 3.10 is then used to determine which segment of A contains the tendon point. Let first the function $d(k)$ equal a length of a negative vector between two points on A:

$$d(k) = \left| (a_k - a_{k-1})^T \right| \tag{3.11}$$

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

$$d_{t_g} = \sum_{k=n-1}^{1} d(k) \left[d(k) \leq \frac{1}{2} l_s^{TS} \right] \tag{3.12}$$

adding only the lengths where the condition $d(k) \leq \frac{1}{2} l_s^{TS}$, 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:

$$j = k \text{ for } \left\{ d(k) \leq \frac{1}{2} l_s^{TS} \right\}. \tag{3.13}$$

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 3.12 returned a 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 further illustrated in Figure 3.11. This figure also illustrates schematically what was discussed previously with lack of biomechanical data prompting to implement a single tendon using $\frac{1}{2}l_s^{TS}$ as a way to visually represent the free tendon attached on the insertion side, but this remains anatomically inaccurate.

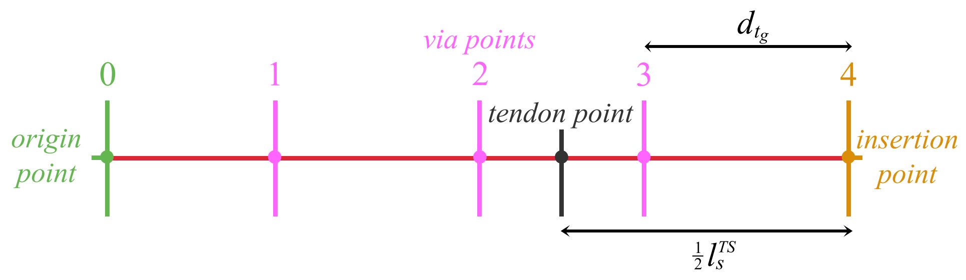


Figure 3.11: A schematic representation of the musculotendon model including a single tendon point lying somewhere between a_2 and a_3.

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

$$t_g = a_{j-1} + s\left(j\right) \text{ with} \tag{3.14}$$

$$s\left(j\right) = \frac{\left(d\left(j\right) + d_{t_g}\right) - \frac{1}{2}l_s^{TS}}{d\left(j\right)} \left(a_j - a_{j-1}\right)^T . \tag{3.15}$$

$s\left(j\right)$ computes the ratio between 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 is added to divide tendon and muscle. 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 UHM enhancement process are completed, but prior to loading any animations, i.e. when Ralph is still in his t-pose configuration.

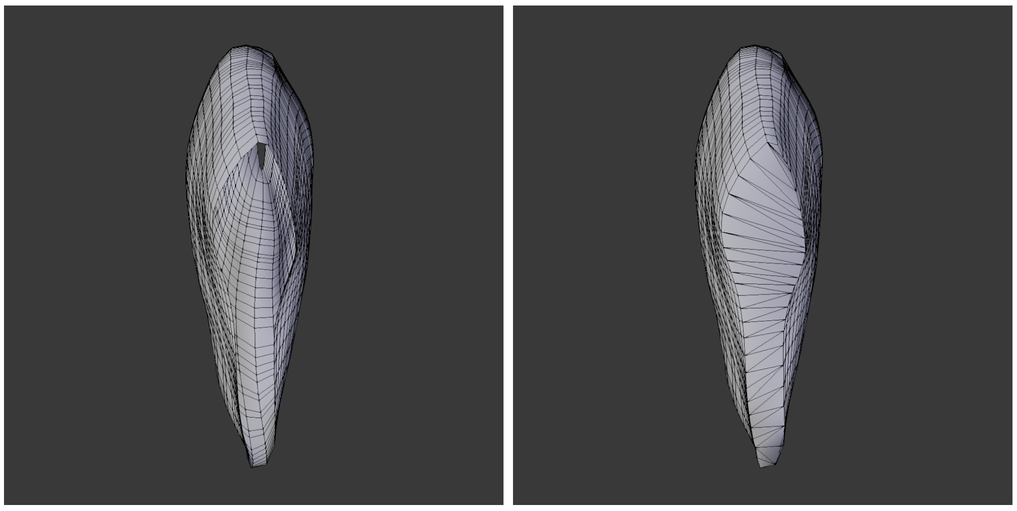
3.5 Geometrical Enhancement with the UHM

The Ultimate Human Model data set (UHM) is according to Snoswell (2003), *"the most complete and accurate set of 3D models of the human musculoskeletal system that has ever been created"*. However, it is also possible that the data set is artistically tweaked as no reference to any work is given at `www.cgcharacter.com` (Snoswell, 2003). Nonetheless, it does state that its accuracy comes from anatomical texts, papers, and custom MRI scans that went into its more than 12 man years of creation.

The goal of using the UHM in this project is to enhance the simple cylindrical shape of each musculotendon with detailed geometries, thereby leveraging the accuracy of the real-time musculotendon model. The enhancement starts by preparing each mesh for import into RAGE, where they will be attached to their equivalent musculotendon of OpenSim[8]. Once the UHM meshes are all attached and oriented in the right configuration, the enhancement process discussed in subsection 3.5.3, can commence. The first step, mesh preparation, is discussed in the following subsection.

3.5.1 Preparing the UHM Meshes

Because the musculotendon model should stay simple and volumetric for future volume discretisation techniques, it was required to adapt certain meshes of the UHM. Lee et al. (2009) confirms this adaptation by stating that the UHM meshes are not well conditioned for a physics-based simulator and report aspect ratios for elements as high as 60:1 and edge length ratios between 1000:1. Certain meshes also exhibit open areas, which could pose problems for volume discretisation techniques as no closed surface can be defined. Thus, similarly to Lee et al. (2009), holes present in meshes representing musculotendons needed to be closed. For this project, this was accomplished using the 3D authoring software Blender Foundation (2006). One example is shown in Figure 3.12.

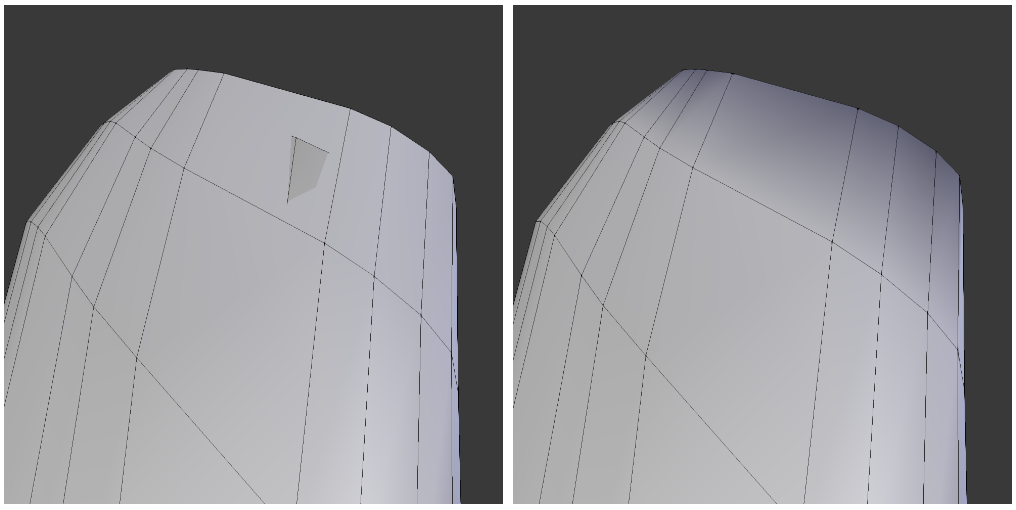


(a) Right Soleus represented as an open mesh. **(b)** Right Soleus represented as a closed mesh.

Figure 3.12: Comparison between an open and closed UHM mesh of the Soleus. The large open area on the front is due to Gastrocnemius and Calcaneal that were occupying that space. The final result, shown in (b), was manually processed using Blender (Blender Foundation, 2006).

[8]Appendix D shows the final result of the mapping process.

Furthermore, some meshes would exhibit self-intersections and these would give incorrect results after the enhancement process was completed for reasons discussed in subsection 3.5.3. To ameliorate potential risk areas, vertices that were positioned on opposite sides were repositioned so as to keep the polygonal UHM mesh penetration free. An example is given in Figure 3.13.



(a) Self-intersection of one vertex (top). **(b)** Corrected mesh without self-intersections.

Figure 3.13: Self-intersection issues with the UHM meshes such as the Iliotibial Band. The corrected result, shown in (b) was manually processed using Blender.

Finally, some UHM meshes were combined to fit a single musculotendon unit in OpenSim, while others where divided for cases where OpenSim represents single musculotendons with multiple action lines, such as the Gluteus muscles that are characterised with broad attachment sites (Blemker and Delp, 2005). Figure 3.14 shows one example for the divided Gluteus Maximus processed in Blender. A complete list, overviewing both divided and combined the UHM meshes, is given in Appendix D.

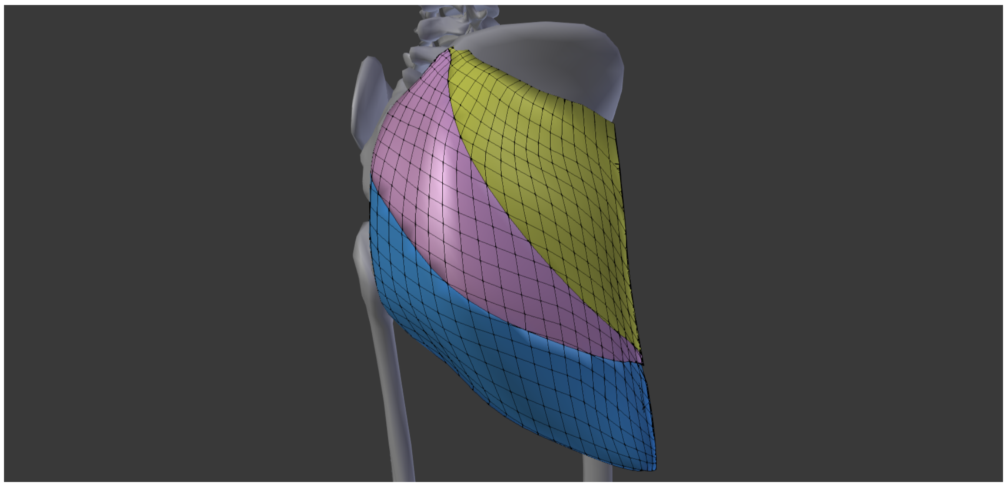


Figure 3.14: An example of the Gluteus Maximus divided (or cut) in Blender for cases where OpenSim uses multiple action lines due to large attachment sites.

3.5.2 The UHM Mesh Shape Analysis

With the inclusion of the UHM meshes for the lower body, a more accurate representation could now be achieved for each musculotendons' custom shape. As musculotendons have different architectures, and each musculotendon has its own unique anatomical shape, a dilemma arose as to what modelled shape would cover the most space contained within each UHM mesh.

To address the above dilemma, it was decided to make a small analysis of each high poly mesh borrowed from the UHM. As an input source for the analysis, three slices from each processed mesh were sampled at approximately first quarter, half way, and third quarter lengths, each orthogonal to the longitudinal axis of each musculotendon. To get an idea for what kind of shapes the high poly meshes would consists of, its isoperimetric quotient is calculated and convex-concave tests are performed on each slice. 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:

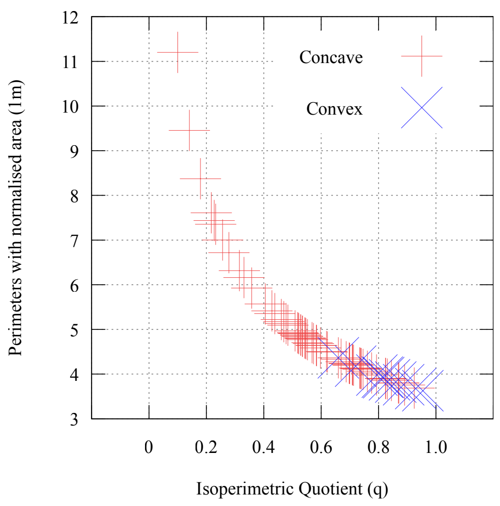
$$q = \frac{4\pi a \left(U^{S_k}\right)}{p \left(U^{S_k}\right)^2}. \tag{3.16}$$

Convex-concave tests were then calculated by first sorting the edge list of U^{S_k} using a method described in Appendix E. After the edges were sorted in a cyclic manner, the z component of the cross product was used to determine whether the simple polygons represented by U^{S_k} were convex or concave. The computation starts first by aligning all slices in the same XY plane. For each vertex u_i of U^{S_k} with coordinates u_i^x, u_i^y, and $u_i^z = 0$, the following calculation is performed:

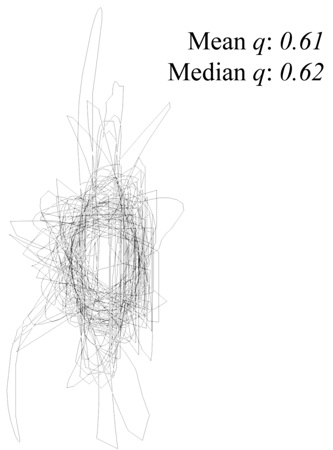
$$Z_i = \begin{cases} \left(u_{i+1}^x - u_i^x\right)\left(u_{i+2}^y - u_{i+1}^y\right) - \left(u_{i+1}^y - u_i^y\right)\left(u_{i+2}^x - u_{i+1}^x\right) & \text{if } 0 \leq i < (n-2) \\ \left(u_{i+1}^x - u_i^x\right)\left(u_0^y - u_{i+1}^y\right) - \left(u_{i+1}^y - u_i^y\right)\left(u_0^x - u_{i+1}^x\right) & \text{if } i = (n-2) \\ \left(u_0^x - u_i^x\right)\left(u_1^y - u_0^y\right) - \left(u_0^y - u_i^y\right)\left(u_1^x - u_0^x\right) & \text{if } i = (n-1) \end{cases} \tag{3.17}$$

with i denoting a zero-based index in $\mathbb{N}$,n equals the number of vertices in U^{S_k}, and Z representing the z component of the cross product. If all Z_i's have the same sign, i.e. they are all either positive or negative, then no angle has surpassed the $[0°, 180°]$ or $[-180°, 0°]$ thresholds. In this case the simple polygon is considered convex, otherwise it is considered concave. Figures 3.15a and 3.15b shows the results of the analysis[9].

[9]It is worth noting that the area of each slice was normalised to 1, making it possible to overlay each slice on top of each other to get the composited image render shown in Figure 3.15b.



(a) Plot of perimeters versus isoperimetric quotients.



(b) Top-view of 72 slices from 24 UHM meshes overlaid.

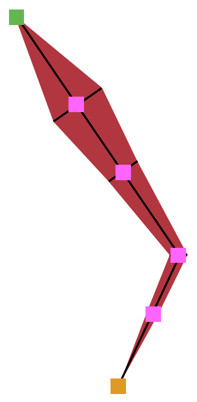
Figure 3.15: Two results from the analysis of 72 slices taken from 24 UHM meshes for the lower body. (a) shows a plot where for each slice its isoperimetric quotient was calculated from its perimeter with a normalised area. (b) shows overlaid renders, also from 72 slices with normalised areas. Each slice was centred around their centroid and positioned on the same plane. The composite image was blended with the difference mode in inverted RGB colours in order to highlight intersections between slices. All UHM meshes that were used in this analysis are given in Appendix D.

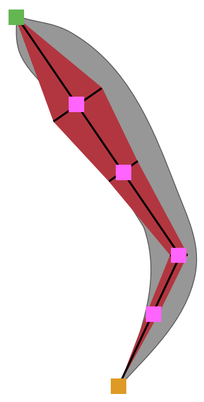
Apparent from Figure 3.15a is that there are more concave than convex slices, but as the normalised perimeter gets smaller, there appears to be more convex slices, with convexity increasing for $q > 0.6$. Furthermore, 75% (or 54) of the 72 slices are more inclined towards circular than elongated shapes, i.e. when $q > 0.5$. Finally, the result also indicates that there is a concentration of shapes with an isoperimetric quotient around 60%. This can be inferred from the difference of 1% between the median and arithmetic mean of q (Figure 3.15b) that presumes a symmetrical distribution for the 72 slices that were sampled from 24 UHM meshes for the lower body.

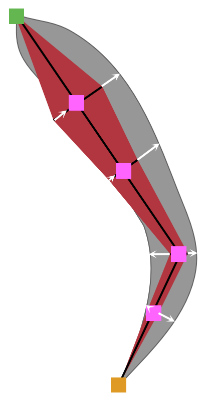
Although the results of the analysis show circular tendencies, there are also highly concave slices, as shown in Figure 3.15b (bottom left). This poses a problem for real-time musculotendon models, as more vertices are required to approximate concave than convex shapes. Nonetheless, by taking the circular tendencies and real-time performance into consideration, it was decided to approximate the high poly geometry by trying to reposition each element of C_k at the surface of its corresponding UHM mesh. This is accomplished with ray-triangle intersection tests described in the next subsection with the goal of having C_k remaining as a simple polygonal representation of its high poly counterpart. In addition, if the real-time model requires better approximation, it would be possible to increase the amount of elements in C_k before casting rays for the intersection tests with the UHM.

3.5.3 A Technique for Musculotendon Enhancement

The general idea for enhancing the musculotendon model is illustrated in Figure 3.16. It starts by scaling the UHM mesh so that the attachment points of the cylinder-based musculotendon are corresponding to similar areas within the high poly mesh. Each mesh is then oriented to face its right direction during t-pose. Next, for each surface vertex, or each element of an arbitrary cross section C_k, rays are shot in the normal direction to search for intersections with the high poly mesh. Once intersection points are found, each element of C_k is translated towards each intersection point along the direction of the ray. The intersection tests can be repeated a couple of times in order to determine whether elements of A should also be repositioned at a new *inner point* of a cross section (or slice) of an UHM mesh. This is further discussed in subsection 3.5.3.2.


(a) Cylinder-based musculotendon with via points.


(b) High poly mesh aligned between origin and insertion.


(c) Rays in normal direction intersecting high poly mesh.

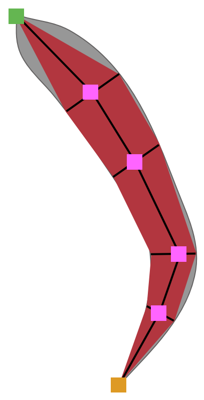

(d) Final enhanced musculotendon.

Figure 3.16: General idea for enhancing the musculotendon model with a high poly reference mesh with the help of ray-triangle intersection tests, in the order of (a) to (d).

It was chosen to use the fast ray-triangle intersection algorithm of Möller-Trumbore (Möller and Trumbore, 1997), instead of contour hull algorithms such as Gurram et al. (2007) and Duckham et al. (2008) as both the UHM meshes and cylindrical musculotendons are configured differently with respect to the amount of vertices and with respect to the three-dimensional position and orientation of their vertices.

3.5.3.1 Repositioning C's on the Surface of U

Generally speaking, 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 an UHM 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 normalised vector $\widehat{\left(c_i^k - a_k\right)}^T$ and the negative direction being the normalised vector $\widehat{\left(a_k - c_i^k\right)}^T$. In addition, each surface triangle of U that intersects this 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 is when:

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 of C_k in the plane containing C_k.

The technique incorporated two additional preferences to cope with the two sides of each triangle's face, i.e. when an intersected triangle was 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 3.17.

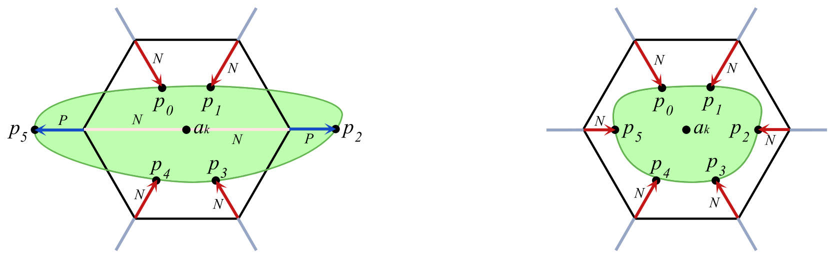


Figure 3.17: Showing ray-triangle intersection outcome number three (left) and outcome number two (right). U^{C_k} is represented by the green shapes and p_i's denote intersection points. p_i's can be found with a ray cast in the positive direction P or in the negative direction N. The left figure shows that here N of p_5 equals p_2 and N of p_2 equals p_5, illustrated with the light-red line segment. The figure on the right illustrates a case where no P-points were found, and instead resorted to finding N-points.

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, shown in Figure 3.18. The combination of these two preferences allows each c_i^k to find its corresponding p_i.

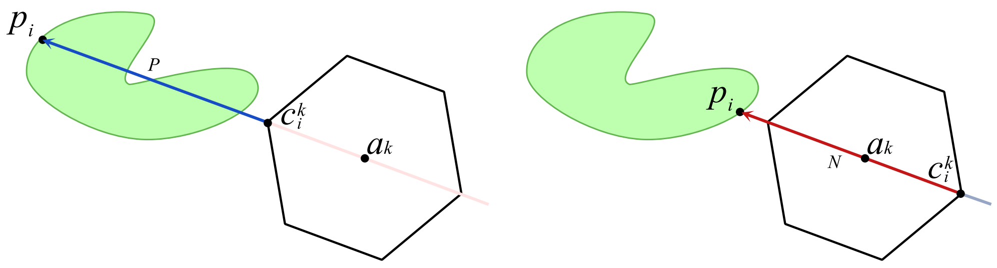


Figure 3.18: Schematic figures showing outcome one and highlighting the second preference: If more than one p_i is found for P, choose the furthest back-face triangle. Conversely, if more than one p_i is found for N, choose the closest front-face triangle.

3.5.3.2 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[10]. Each radius r_i^k is obtained with $\left| \left(c_i^k - a_k\right)^T \right|$ or $\left| \left(a_k - c_i^k\right)^T \right|$. 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 that will be used throughout the remaining subsections. The example uses the index $k = 1$ in the longitudinal direction where U^{C_1} turns out to be a closed concave shape positioned outside of C_1 and is schematically illustrated in Figure 3.19. The illustration shows the end result indicating why locating a point within U^{C_k} (an inner point) is necessary as the result equals a false representation with the deformed hexagon being complex instead of simple.

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 following condition:
One of the two opposing algebraic radii has a negative sign.

[10]For a more detailed description of this process, refer to section 3.7.

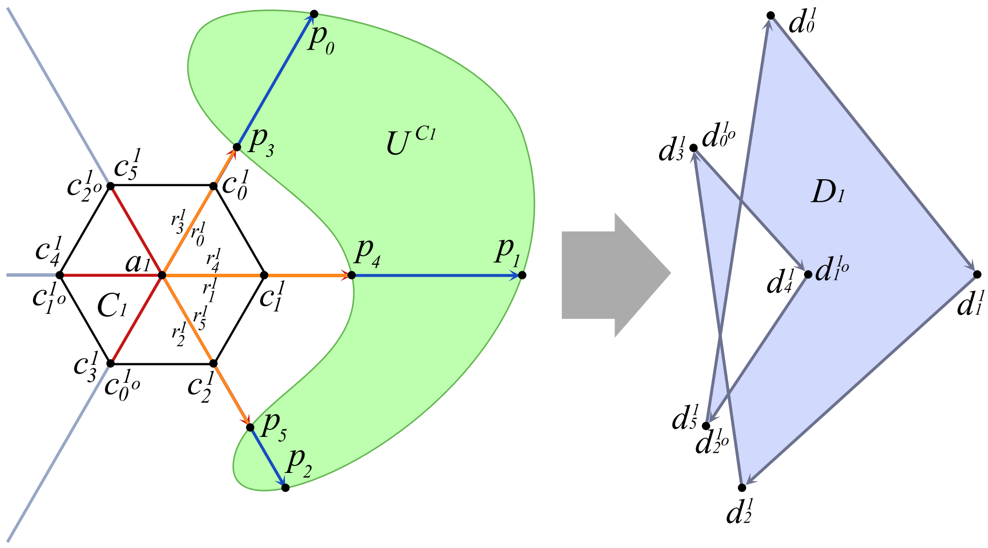


Figure 3.19: Showing a key example with a concave slice of an UHM mesh, denoted as U^{C_1}. Applying the preferences and conditions discussed previously results in the complex polygon D_1, which is a false representation due to a_1 being positioned outside of the targeted cross section U^{C_1}.

In order for a pair of cast rays to be considered equal, for instance c_1^1 and c_4^1 in Figure 3.19, it is important that both c^1's are symmetrical counterparts with respect to a_1, i.e. *opposites*. This is ensured by requiring that the cardinality for every C_k is always even and never odd, which is the case here for the current example that uses six vertices, i.e. a hexagon. From this example it is also visible that for every C_k with $|C_k| \in 2\aleph$ and $|C_k| = 6$, the maximum amount of matches would be always equal to $\frac{1}{2}|C_k|$ thereby reducing the amount of match-lookups to half. Hence, each opposing element of c_i^k can be deduced by taking the opposing index i^o with:

$$i^o = i + \frac{1}{2}|C_k| \text{ where } \left\{ i \in \mathbb{N} \;\middle|\; i < \frac{1}{2}|C_k| \right\}. \tag{3.18}$$

Each detected match helps to determine when a ray passed a centre point a_k as this suggests the possibility that U^{C_k} lies somewhere (at least partly) 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 following condition: $\left| \, (p_i - c_i^k)^T \, \right| > r_i^k$. In this case, a negative sign is added to update the radius with the following new result: $-\left| \, (p_i - a_k)^T \, \right|$. In all other cases, the radius remains positive. In the current example, radii r_3^1, r_4^1, and r_5^1 become negative, depicted with orange line segments in Figure 3.19 and overlapping 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 locate an inner point that is discussed in the next paragraph. Visually, 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 through the continuity of the shape, and reaching the final term $\left(n-1, e_n^k - 1\right)$ at the other end of U^{C_k}; given that the amount 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. The example given in Figure 3.20 illustrates that when starting the search at c_0^1, the elements of E_1 would contain the same set of matches. 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.

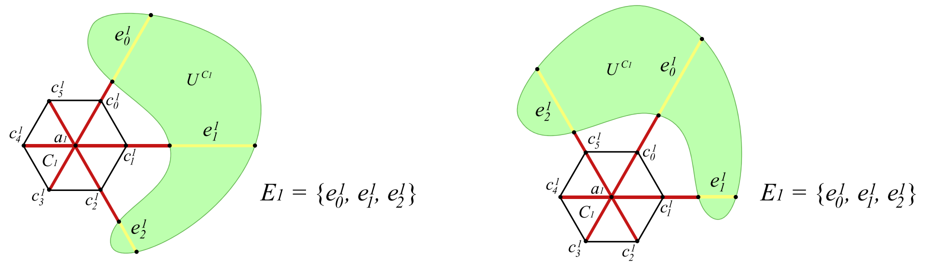


Figure 3.20: Schematic figure showing the same set E_1 with matches detected in the same order but for two different shape configurations, prompting the need to track if a c_i^1 was repositioned to d_i^1 with a positive or negative algebraic radius.

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 search 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. $i^+ = i^o$. This example is further elaborated in Table 3.4 that shows all possible permutations with i^+ using the same configuration as previous examples 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 (or $\frac{1}{2}|D_k|$) have to be searched to get all six permutations.

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

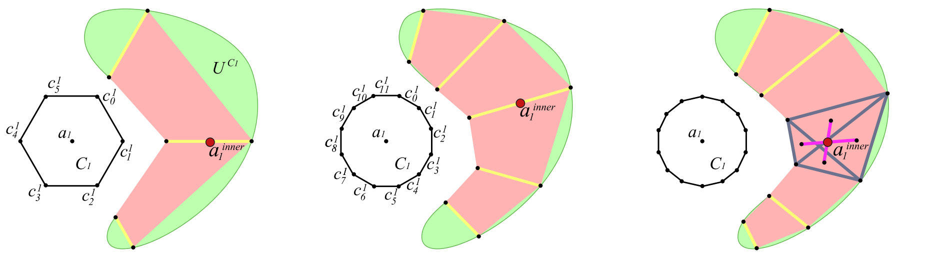
With the example given in Table 3.4, the actual permutation σ computed on indices of E_k (visible in the middle column) is given in pseudo-code with Algorithm 1.

```
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 σ 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}.

σ can also be used to calculate the area and consequently the centroid (or centre of gravity) with equations given in Appendix F. It is worth pointing out however that the centroid is not suitable for this application, as not always its derivation will result in a point located within U^{C_k}, e.g. in boomerang like shapes.

Determining an Interior "Inner" Point As discussed previously, when a_k lies inside of U^{C_k}, the amount of false positives is presumed to be less thereby reducing the initial problem illustrated in Figure 3.19. 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, dividing them into even and odd after being permuted by σ. When odd, the mid point of the middle match is taken as the inner point where a_k should be repositioned. When even, a single match cannot be determined, thus a more complicated approach is used. Here, the two middle matches are taken that collectively form a quadrilateral. Each vertex of this quadrilateral is then projected into two dimensional space, where its centroid can be calculated using the method of Patterson (2003). The method of Patterson (2003) requires an intersection test for two line segments between pairs of centroids of each of the four triangles present in every quadrilateral. This intersection point is computed using the method of Goldman (1990) and represents the centroid of the quadrilateral. Thus, this point is the inner point where a_k has to be translated to. Figure 3.21 shows three examples for different dimensions (or cardinality) of C_k with Figure 3.21c illustrating an example for the centroid approximation of a middle quadrilateral.[11].



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

Figure 3.21: An example illustrating two types of inner point determination. (a) shows the inner point determined by the midpoint of the middle match for an uneven 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.

3.5.4 Iterating the Enhancement Process

With a_k now repositioned to $a_k^{Centroid}$, the enhancement process can be repeated in order to get a simple and more accurate polygonal representation of U^{C_k}, given in Algorithm 2. It is also worth mentioning that Line 14 and Line 15 were included to give preference to the attachment positions of OpenSim instead of the UHM. So that even after the origin and insertion points have been repositioned, they will still go back to their initial OpenSim locations.

[11]Refer to section 3.6 for more information on scaling the dimension of C_k.

```
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 order of steps executed during the UHM enhancement process.

3.6 Adapting the LOD of the Musculotendon Model

Most musculotendons in Menegolo (2011) contain zero via points with the Sartorius having a maximum of three via points. This creates a problem for leveraging the accuracy during the enhancement process discussed in subsection 3.5.3 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, it was decided to adapt the musculotendon model to be scalable in both longitudinal and latitudinal dimensions, thereby introduction a LOD technique. Allowing the LOD to be adapted in both directions also satisfied the requirement to simulate a more accurate model in real-time on faster computer hardware.

3.6.1 Longitudinal and Latitudinal Scaling

As discussed in subsection 3.4.2, 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) it was decided to look at the amount 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. The 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, given as:

$$t_{\text{LON}} = 2^{\text{LSS}} \left(|V| + 1\right) + 1 \text{ with LSS} \in \mathbb{N} \tag{3.19}$$

The musculotendon model can also scale in the latitudinal dimension, denoted by t_{LAT}, with the amount of vertices at each cross section falling within the range of $|C_k| \geq 6 \wedge |C_k| = 2\aleph$. This is used to locate inner points as discussed previously. A copy of M, denoted as M_{copy}, is taken where the latitudinal dimension is scaled to a high enough value so that every a_k of M_{copy} gets repositioned inside the volume of an UHM mesh U. Once the enhanced 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 shapes. This is done by introducing a scalar, the cross sectional scalar, denoted as CSS to differentiate between latitudinal scales and is used as:

$$t_{\text{LAT}} = \text{CSS } |C_k| \text{ with CSS} \in \mathbb{N}. \tag{3.20}$$

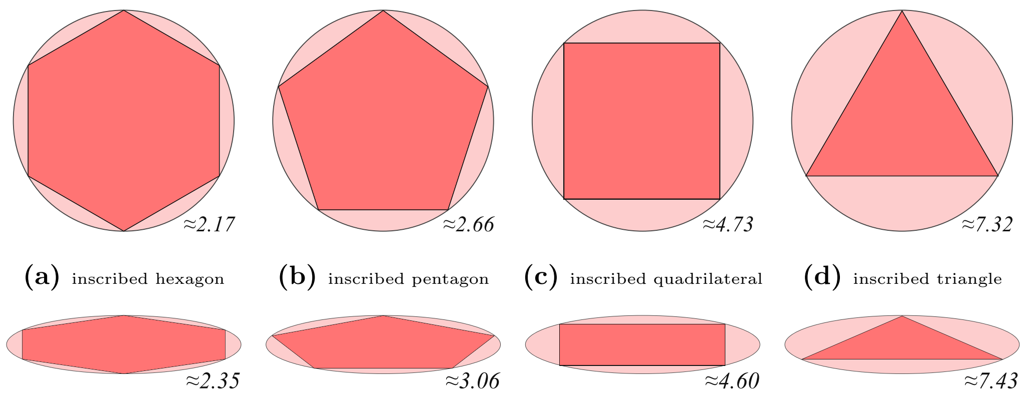
There is also a lower bound on C_k's cardinality for reasons discussed in the next subsection.

3.6.2 Hexagons as a Lower Bound for C_k

With the ability to scale the amount of elements in C_k, as explained in subsection 3.6.1, it would be also beneficial to establish a lower bound for the cardinality of C_k. This limit would define the lowest amount of geometrical information that is reasonably achievable for the real-time simulation. The lower bound was chosen to be six for every C_k instead of the remaining three polygons with lesser sides, illustrated in Figure 3.22. Using hexagons as the lower bound for $|C_k|$ brings forth two advantages:

1. As with all polygons with even Schlåfli numbers, hexagons have the property of being *reflectively symmetric*. This comes in handy for the pairwise ray-triangle intersection tests when determining whether a hexagonal-plane needs to be translated into the volume of another target musculotendon geometry with higher resolution[12]. Thereby excluding pentagons and triangles as possible candidates for a lower bound.

[12]An in-depth discussion on the pairwise ray-triangle intersection tests is given in subsection 3.5.3



(a) inscribed hexagon **(b)** inscribed pentagon **(c)** inscribed quadrilateral **(d)** inscribed triangle

(e) inscribed hexagon **(f)** inscribed pentagon **(g)** inscribed quadrilateral **(h)** inscribed triangle

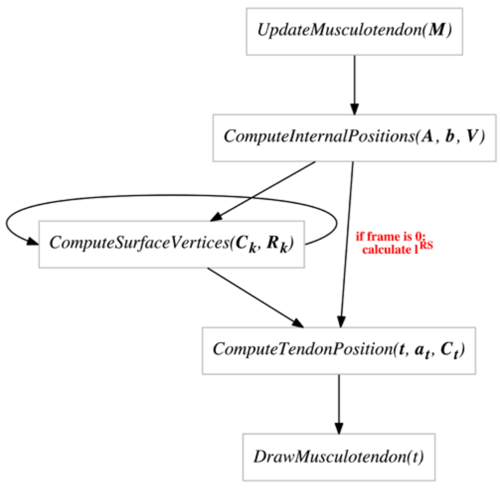
Figure 3.22: Illustration of areas covered by polygons with Schläfli numbers between {6} and {3} inscribed in conic sections. Both the circle and ellipse have the same area. Clearly, both regular and irregular hexagons have a better perimeter to area ratio. The calculation of each covered area is the subtraction of standard area equations for these polygons from the area of the circle or ellipse. The approximated solution to each problem is given in the lower right corner where a lower value indicates better coverage.

2. Second, even though musculotendons consist of different architectures, the analysis in subsection 3.5.2 indicates that anatomical shapes of musculotendons have circular tendencies[13]. Thus, a cross section of a musculotendon unit could be considered as a closed curve that deviates from an idealised conic section[14]. Because hexagons have a higher perimeter to area ratio than quadrilaterals, hexagons are thus more fitting to a model with circular tendencies compared to quadrilaterals.
3. Hexagons have a lower amount of vertices than octagons, and because the three polygons with lesser sides are already excluded, the remaining option is the hexagon.

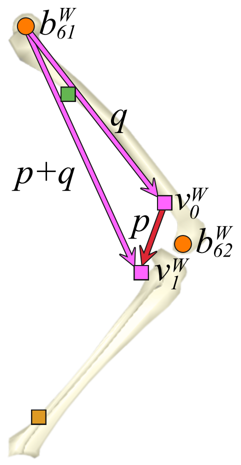
3.7 Drawing the Cylinder-based Musculotendon Unit

Drawing M in each time step comprises of a couple of steps outlined in Figure 3.23a. Each M is defined in its own internal coordinate system based on the world position of the bone parented to the *origin point*. The first routine computes local positions of A as illustrated in Figure 3.23b. Afterwards, C_k is computed with all radii from the set R_k. Similar to the internal coordinate system used for the relative positions of a_k, the directions of r_i^k are also computed in intervals relative to angle θ, given as $\frac{1}{|C_k|}360°$. Next, one tendon point is interpolated and stored separately as a_t and C_t. Finally, the muscle is drawn, shaded, and textured using OGRE.

[13]A circle is also one of the most common shapes found in nature (Whitaker, 2006).
[14]Except for aponeuroses, which are flat and broad tendons. These are not included in biomechanical models as discussed in subsection 3.4.3 and thus were considered beyond scope.


UpdateMusculotendon(M)
ComputeInternalPositions(A, b, V)
ComputeSurfaceVertices(C_k, R_k)
if frame is 0; calculate l^RS
ComputeTendonPosition(t, a_t, C_t)
DrawMusculotendon(t)

(a) Steps involved prior the drawing routine.



(b) Example showing the internal coordinate system.

Figure 3.23: (a) shows the order of steps before drawing each musculotendon in OGRE. In (b), an example is given with two via points where the second via point v_1^R (in internal coordinates) is derived with the help of p. For each constraint, a vector is taken between the world position of an attachment point and the parented bone of the origin point. In this example, $q = \left(v_0^W - b_{61}^W\right)^T$ and $p + q = \left(v_1^W - b_{61}^W\right)^T$ therefore $p = p + q - q$ and so the relative position v_1^R can be derived with $v_0^R + p$.

For the actual drawing routine, it was opted to use the `Ogre::ManualObject` class as it provides wrapper functions for the creation of custom objects. To prevent performance hits, each M is created once with the inclusion of extra vertices for the tendon point and is afterwards updated with new positions in subsequent time intervals, ensuring that GPU memory is allocated once, with pre-calculated vertex and index buffers.

3.7.1 Triangle Fans and Triangle Strips

Instead of using triangle lists, it was opted for a combination of fans and strips, illustrated in Figure 3.24. These render types give less vertex redundancy with each 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 the graphical musculotendon in real-time.

3.7.2 Applying Textures

The final addition to the musculotendon model was a visual aid to inspect the length of tendons derived with the linear interpolation function discussed in subsection 3.4.3. Adding textures proved to be a practical way to solve this issue and moreover it increased the visual realism of the model. Some examples are given in Figure 3.25.

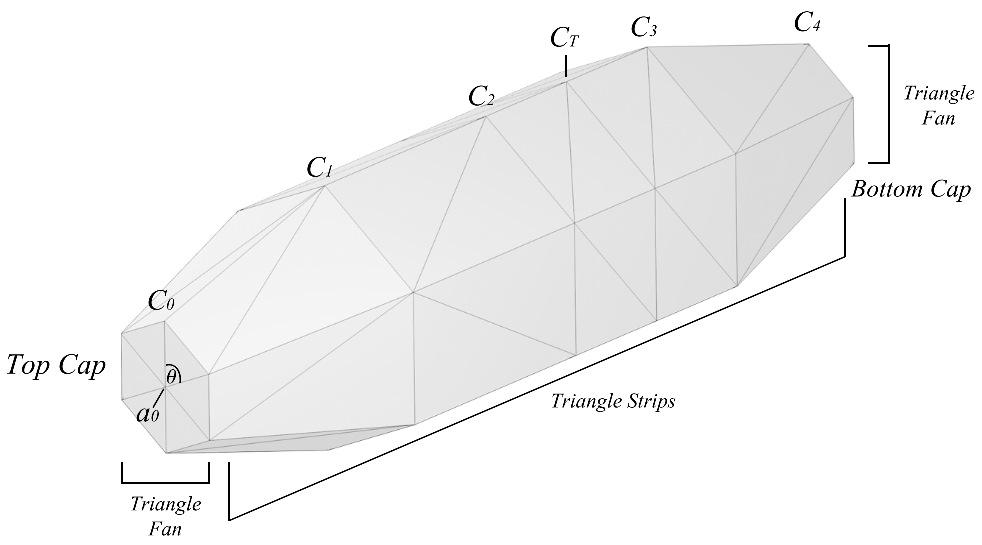


Figure 3.24: Showing the cylinder-based musculotendon with triangle fans and strips.

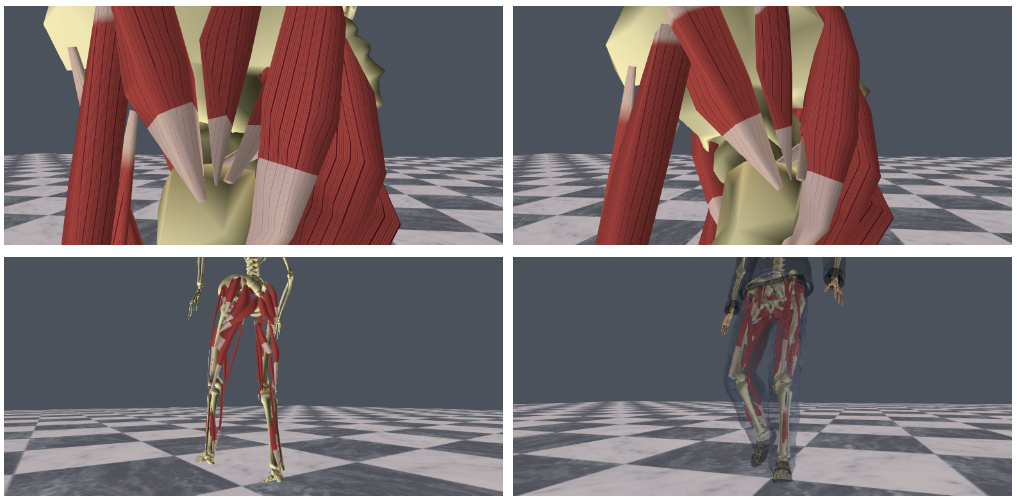


Figure 3.25: The top rows shows interpolated tendons up close, while the bottom row shows a back and front view of the lower body model that is embedded within Ralph.

Chapter 4

Real-Time Collision Detection and Response

The spatial arrangement of musculotendons is considered problematic as they are grouped and intertwined with the skeleton, with themselves, and in tight spaces. A recent survey from Lee et al. (2010) mentions that solving collisions would provide a significant advancement in the area of musculoskeletal simulation. Indeed, a collision-free system would for example allow volumetric models to be used with the finite element method (FEM). This project tried to look for a solution using different techniques, but in the end resorted to a custom strategy that tries to isolate the problem of collision detection and response.

To solve collisions, the relatively fast GJK-EPA algorithm (Cameron, 1997) that is taken for granted in most real-time contact problems could not directly be used in this case due to the concavity inherit within the musculoskeletal anatomy. An attempt was therefore made with Bullet Physics (Coumans, 2013) by representing each musculotendon as a soft body comprising of spring-masses. Unfortunately, this turned out to be impossible to model muscle behaviour. The addition of extra links to reduce bending and preserve volume did not solve the issue either and hinted at possibly reaching a limit for the stiff equations that are used by Bullet's integrator, or lack of knowledge and experience from the author. The second issue was a exceedingly low amount of frames per second (FPS) even when using collision groups and masks for all bodies, and hierarchical approximate convex decomposition (HACD) for the rigid skeleton bodies (Mamou, 2013). One other option was to represent each musculotendon as a chain of connected rigid cross sections. However, this would create other issues such as not being able to deform cross sections and missing collisions within the inner segments, due to the thinness of the plate like cross sections.

4.1 Custom Strategy for Discrete Collision Detection

The above mentioned problems hint at the complex arrangements within components of the musculoskeletal system that are not suitable for the standard methods and techniques within the field of real-time collision detection, such as sweep and prune and spatial partitioning. Besides the presumption that most anatomical objects are concave, there are interdependencies for contacts between skeletal muscle, skeletal bone, and between fat and adipose tissues. Even in a simplified model that uses just skeletal muscle and bone, collisions would occur between the muscles and the skeleton, within the same muscle, between different muscles, and also between different bones of the skeleton, all influenced by and depending on a certain pose of a particular animated character. With such a high degree of outcomes for colliding objects, it made sense to isolate the problem of collision detection for this specific problem-type and introduce a new pairwise collision constraint suitable for both low and high poly musculotendon models.

4.1.1 Collision Constraints for Muscle and Skeleton

Besides having a penetration free state, muscles should follow a path from origin to insertion, passing around other muscles and other skeletal bones. The musculotendon model discussed in this thesis uses via-points from OpenSim and approximated inner points of the Ultimate Human Model (UHM) to create a more accurate path[1]. However, these additions alone do not guarantee a convergence to a penetration free state, especially when coupled with an animation sequence that wraps, bends, twists, and stretches musculotendons.

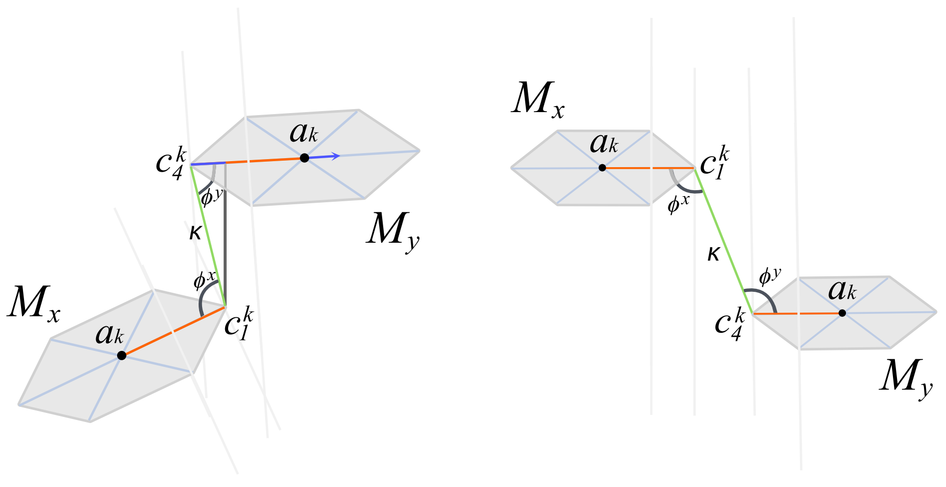
To try to solve this problem, a new constraint denoted by κ is introduced that forms a transversal line segment between two cross sections of two arbitrary musculotendons. Each constraint would connect a vertex of a tagged cross sectional plane $C_k \in M_x$ with another vertex of another tagged cross sectional plane $C_k \in M_y$. The aim is to try to solve two problems in real-time collision detection within this domain, namely:

1. Finding a correct location of M_x with respect to M_y.
2. Reducing the amount of discrete collision checks.

A constraint κ consists of two tagged c_i^k's, denoted as κ^x and κ^y, from two separate musculotendon meshes. Geometrically, each κ represents a transversal with two alternate interior angles ϕ^x and ϕ^y, shown in Figure 4.1 that vary with the orientation of each cross

[1]OpenSim uses similar constraints such as wrapping surfaces, discussed in 3.4.1, however these are aimed at approximating empirical data more closely for a polyline representation of the action line.

section. When either of the two angles are acute, the solver uses the scalar projection of $\left(\kappa_k^y - c_i^k\right)^T$ unto $\left(a_k - c_i^k\right)^T$ as the penetration depth ρ^x, or $\left(\kappa_k^x - c_i^k\right)^T$ unto $\left(a_k - c_i^k\right)^T$ as the penetration depth ρ^y. If both angles are acute, the sum of both depths is taken.



(a) A potential collision is detected. **(b)** No collision is detected.

Figure 4.1: A transversal constraint κ consisting of two tags, c_1^k of M_x and c_4^k of M_y. (a) illustrates one acute angle ϕ^y with the penetration depth ρ^y (marked in blue). The penetration depth is used to determine whether a_k of M_y should move in the opposite direction $\left(c_{i^o}^k - a_k\right)^T$ by distance ρ^y. Due to the symmetry of even dimensions for $|C_k|$, the opposite index denoted by i^o can be found. (b) illustrates a case where no acute angles are present on either side of κ, thus indicating a possible collision-free state.

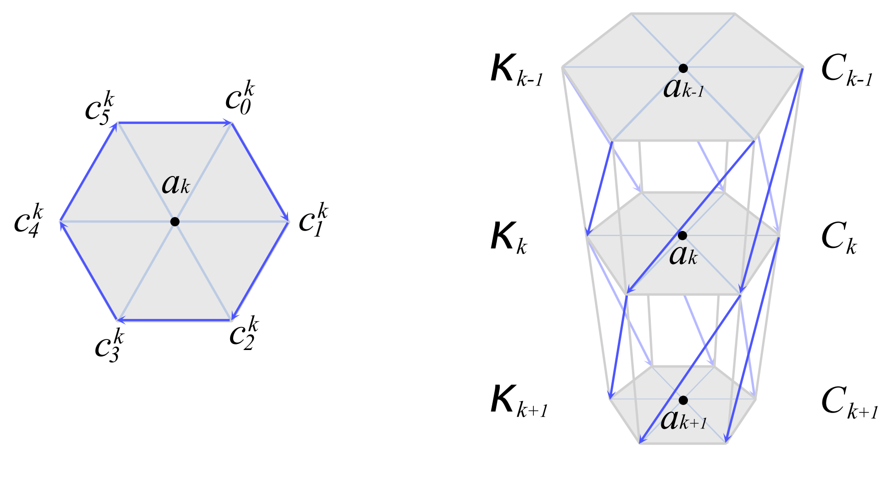
ρ is then used as the distance to determine whether either a_k should be moved in the opposite direction $\left(c_{i^o}^k - a_k\right)^T$, depending on which reference frame the solver is computing from[2]. Using the example of Figure 4.1a, if a user-defined tag places M_y as its frame of reference, then for this particular κ^y the solver is instructed to check if the constrained cross section centred at a_k should move in the opposite direction given by $\left(c_1^k - a_k\right)^T$ with a tag being placed on c_4^k, of M_y. In addition, tagged c_i^k's are used to instruct the solver to let all elements of C_k be checked for collisions against specific skeleton meshes, and not just other muscles.

4.1.2 Discrete Collision Detection with Ray Casting

A transversal constraint alone does not guarantee a penetration free state as it depends too much on where manual tags are placed, and when leading with low scale geometries this does not provide sufficient robustness. Hereto, they are only used to check if a cross section should be moved or not. To increase the accuracy of the discrete collision solver, ray-triangle intersection tests are carried out with the same Möller-Trumbore algorithm

[2]Opposing indices denoted by i^o are found with Equation 3.18.

(Möller and Trumbore, 1997) that was used for the UHM enhancement process[3]. Although all triangles of a tested object could be taken into account during narrow phase detection, the amount of rays being cast was limited in this implementation to the amount of constraints placed on the object. This results in less computations, except when checking for collisions between muscle and skeletal bones as each face of a skeleton mesh has to be tested. Each intersection test starts at a C_k of a tagged c_i^k where rays are cast from each c_i^k in one sweep consisting of two different directions, illustrated in Figure 4.2.



(a) Latitudinal Sweep. **(b)** Diagonal Sweep.

Figure 4.2: Direction of cast rays confined by transversal constraints.

At each c_i^k rays are cast in the latitudinal and diagonal normal directions with:

$$\mathcal{D}_{lat}(i,k) = \begin{cases} \left| c_{i+1}^k - c_i^k \right| & \text{if } i < |C_k| - 1 \\ \left| c_0^k - c_i^k \right| & \text{if } i = |C_k| - 1 \end{cases} \tag{4.1}$$

$$\mathcal{D}_{diag^P}(i,k) = \begin{cases} \left| c_{i+1}^{k+1} - c_i^k \right| & \text{if } i < |C_k| - 1 \wedge k < |A| - 1 \\ \left| c_0^{k+1} - c_i^k \right| & \text{if } i = |C_k| - 1 \wedge k < |A| - 1 \end{cases} \tag{4.2}$$

$$\mathcal{D}_{diag^N}(i,k) = \begin{cases} \left| c_{i+1}^{k-1} - c_i^k \right| & \text{if } i < |C_k| - 1 \wedge k = |A| - 1 \\ \left| c_0^{k-1} - c_i^k \right| & \text{if } i = |C_k| - 1 \wedge k = |A| - 1 \end{cases} \tag{4.3}$$

where $\mathcal{D}_{lat}$ is the normal direction of the cast ray around C_k in the latitudinal direction, and $\mathcal{D}_{diag^P}$ and $\mathcal{D}_{diag^N}$ represent the positive diagonal and negative diagonal directions respectively[4]. The presumption here is that extra rays tests would lower the chances

[3]Refer to subsection 3.5.3 for how these tests were used during the UHM enhancement process.
[4]Note that Equation 4.3 is only used for cases when $k = |A| - 1$.

of false negatives during narrow phase detection. The following section describes the discrete iterative resolution, i.e. converging to a penetration-free state after penetrations were already detected.

4.2 Collision Response

In this implementation, it was determined that it would be best to move a_k of M_x on three occasions, namely:

1. when $\rho^x + \rho^y > \left| \left(a_k - c_i^k\right)^T \right|$,
2. when radii are adjusted,
3. when penetration is detected with rays cast in a diagonal direction.

The first occasion will be described here while the second and third occasions will be described in the next subsection. As mentioned previously, κ's main purpose is to find a correct location of M_x with respect to M_y, and is particularly useful for cases when more than half of the volume of M_x lies in M_y or when M_x lies on the opposite side of M_y of its supposed location. In these cases, when penetration depths are above a certain criteria, $a_k \in A$ are displaced. The actual displacement of a_k uses translation vectors that were purposed for the UHM enhancement, but because at this stage the enhancement process is already complete, their values can be adjusted for a new purpose. Each new position is calculated by first subtracting the applied translation from the UHM. Let μ_k denote the translation vector given by $\left(a_k - a_k^{orig}\right)^T$ where a_k^{orig} denotes the original OpenSim position before the UHM. Subtracting a_k from μ_k gives then a_k^{orig}. With the obtained a_k^{orig}, the new μ_k is then calculated as:

$$\mu_k = c_i^k + \left(\rho^x + \rho^y \frac{\left(a_k - c_i^k\right)^T}{\left| \left(a_k - c_i^k\right)^T \right|} \right) - a_k^{orig} \tag{4.4}$$

There are also cases where a_k does not have to be translated, here radii of R_k are adjusted in order to preserve the same initial area prior deformation.

4.2.1 Retaining Cross Sectional Areas through Inflation

To keep the same cross sectional area after being deformed by the UHM enhancement process, radii r_i^k that would deform again due to detected collisions, are adjusted. Here again the usefulness of using a cylindrical-based musculotendon construction comes forward as it prevents Euclidian lengths of vectors to be calculated. Since in order to deform the musculotendon all that is needed is the adjustment of a radius. To retain cross sectional area, an iterative approach is taken that incrementally expands r_i^k a tiny bit in each time step resulting in an inflation effect for the *free area* of C_k. Before inflation, the deformed area after the UHM enhancement is stored and used as a target area τ. In order to know which r_i^k can be adjusted and which are currently being deformed by another object, a distinction is made between radii, segmenting them into *fixed* and *free*. A radius is flagged as fixed when it has to expand inwards due to its surrounding surface edges detecting penetration of another object. Remaining radii are considered free and are thus able to expand outwards. Both fixed and free indices i of each r_i^k are stored in the set $\mathcal{R}_k^{fixed}$ and $\mathcal{R}_k^{free}$ respectively. As a rule of thumb, when confronted with a penetration, fixed radii will decrease in distances of 10% until a C_k with κ reaches a penetration free state[5]. When r_i^k approaches zero, given as numerical limit ϵ, its centroid a_k is translated in direction of $\frac{1}{10}\left(c_{i^o}^k - a_k\right)^T$, which is also 10% of the radius in the opposite direction obtained with Equation 3.18. This allows C_k to translate slowly out of the maximum penetrated area and not create a big enough gap between the two collided objects at constrained cross sections. This same technique for displacement is also applied on the third occasion when rays are cast in the diagonal in order to push a_k a bit outwards in each time step.

An important aspect of radii adjustment is to not lose the initial shape of τ, which is achieved by calculating a factor for all other free radii with respect to a particular radius that is fixed, and thus represents a radius that has finished deforming or is still undergoing deformation. A radius factor f_i^k can be calculated for each r_i^k as the function:

$$f_i^k(j) = \frac{r_j^k}{r_i^k}, \forall j : 0 \leq j < |C_k| \wedge i \neq j \tag{4.5}$$

To reach convergence, areas comprising of fixed and free radii, denoted as τ_{fixed} and τ_{free} are also taken into consideration. Due to the cylindrical construction, each C_k can be split into its constituent triangles, allowing areas τ_{fixed}^k and τ_{free}^k to be calculated separately using scalar projections[6]. τ_{fixed}^k and τ_{free}^k are calculated as:

$$\tau_{fixed}^k = \sum_{i=0}^{i<(|C_k|-1)} \Delta\left(a_k, c_i^k, c_{i+1}^k\right) \text{ if } i \in \mathcal{R}_k^{fixed}, \tag{4.6}$$

$$\tau_{free}^k = \sum_{i=0}^{i<(|C_k|-1)} \Delta\left(a_k, c_i^k, c_{i+1}^k\right) \text{ if } i \in \mathcal{R}_k^{free}, \text{with} \tag{4.7}$$

[5]To increase convergence accuracy, the 10% could be lowered and will then take more iterations to reach satisfaction.

[6]Uses just two length calculations, one in Equation 4.8 and 4.9, thus less costly than Heron's formula.

$$\Delta\left(a_k, c_i^k, c_{i+1}^k\right) = \frac{1}{2} \ \left| \left(c_{i+1}^k - c_i^k\right)^T \right| \ h\left(a_k, c_i^k, c_{i+1}^k\right), \text{ and} \tag{4.8}$$

$$h\left(a_k, c_i^k, c_{i+1}^k\right) = \left| \left(c_i^k + \left(\left(a_k - c_i^k\right)^T \cdot \widehat{(c_{i+1}^k - c_i^k)}^T \widehat{(c_{i+1}^k - c_i^k)}^T\right)\right) - a_k \right| \tag{4.9}$$

with Δ representing the standard area equation for arbitrary triangles, and h outputting the height of the projected vector along the base, minus a_k. Finally, each time step computes the maximum possible free area given as a function ω:

$$\underset{\tau_{free}^k}{\operatorname{argmax}}\, \omega\left(\tau_{free}^k\right) := \left\{\tau_{free}^k \middle| \left(\tau - \tau_{fixed}^k - \tau_{free}^k\right) > \epsilon\right\} \tag{4.10}$$

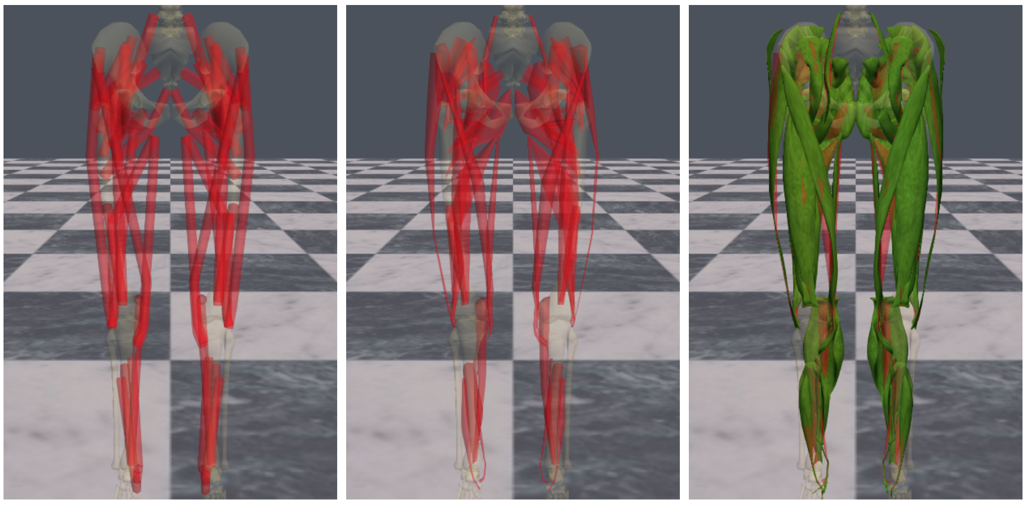
where ϵ denotes again a predefined error margin for numerical accuracy, e.g. 1×10^{-6}.

Chapter 5

Results and Discussion

5.1 The UHM 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 poly musculotendon geometry for the lower part of the human body. Figure 5.1 shows one result visually, while Table 5.1 lists the total amount of vertices for 12 different scales compared to the total amount of vertices used in meshes from the UHM.



(a) Before the UHM enhancement. (b) After the UHM enhancement. (c) Result covered by the UHM.

Figure 5.1: This result was obtained with 48 musculotendons having $t_{LAT} = 36$ with the cross sectional scalar CSS = 6, which was down sampled back to the lower scale of $|C_k| = 6$. LSS was set to 1 to add one extra segment between two cross sections. The complete model includes just 1536 vertices. Notice the approximations of C in the lower half of (b) with thin strips of tendon around the ankle and knee joints that show good approximation.

LSS / $\lvert C_k \rvert$	$Hexagon\{6\}$	$Octagon\{8\}$	$Decagon\{10\}$	$Dodecagon\{12\}$	UHM
0	960	1248	1536	1824	
1	1536	2016	2496	2976	54723
2	2688	3552	4416	5280	

Table 5.1: Number of vertices for a total of 48 musculotendons in 12 different scales compared to the same amount of 48 UHM meshes.

Table 5.1 shows a drastic reduction for the amount of vertices used with in the real-time model. For instance, at the scale of $|C_k| = 12$ and LSS = 2, the real-time model uses $\approx 90\%$ less vertices than the UHM. The 54723 vertices represent the total number of vertices of 48 triangulated UHM meshes. Triangulation is a common practice for polygonal meshes used in real-time game engines, however even without triangulation all 48 UHM meshes still consisted of 42528 vertices[1].

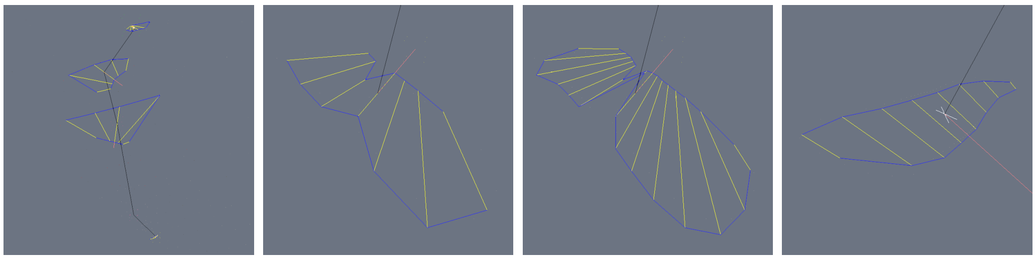
5.1.1 Non-linear Longitudinal Scaling

In subsection 3.6.1, an argument was given for the exponential term in Equation 3.19, related to the uniform splitting of one segment between two other cross sections. There is also a second argument for the non-linear increase that has to do with via points. On the one hand, if LSS would increase linearly, then the original relative position of a via point would be lost when each added cross section is placed exactly at the midpoint between two other cross sections in order to retain uniformity. On the other hand, if the relative position of a via point is to be remained fixed then there is no way of systematically determining on which side of the via point the extra cross section would be placed, either above or below the via point. Each possibility loses one benefit at the expense of the other, therefore the solution of adding one extra cross section at the midpoint between two other cross sections allows both benefits to be gained; scaling in the longitudinal direction and retaining the relative positions of via points intact.

5.1.2 Approximating the UHM at Different Scales

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 project. Figure 5.2 shows a couple of examples.

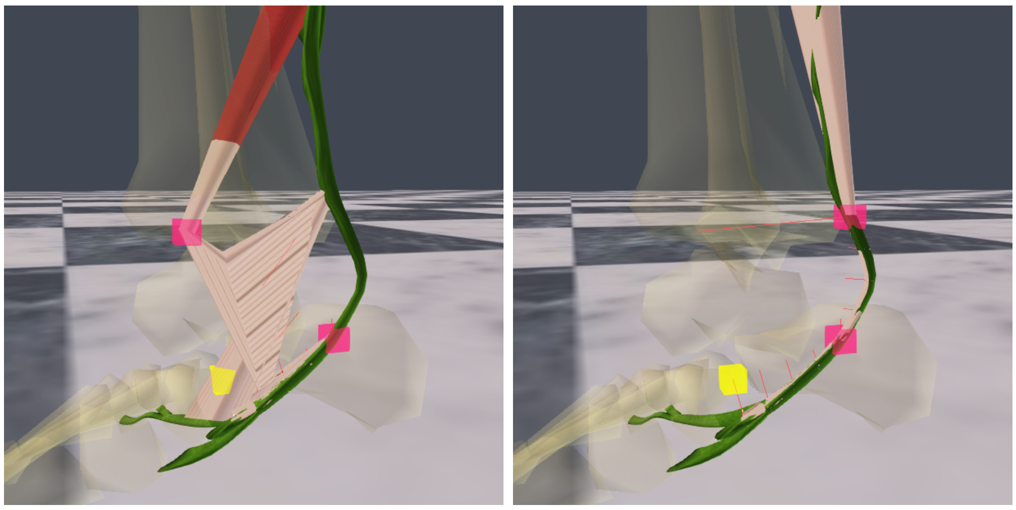
[1]Refer to Appendix D for a complete overview of all 48 UHM meshes exported from Blender.



(a) Several cross sections. **(b)** C_k with $|C_k| = 24$. **(c)** C_k with $|C_k| = 48$. **(d)** Centroid of quad.

Figure 5.2: Some examples for inner point determination. The line segments are coloured as follows. Yellow represents matches e, blue represents segments between vertices, black is the action line A, and finally red is the translation of a_k from its original position to the new inner point located within the approximated UHM mesh. (c) also shows the additional detail captured form a highly concave mesh. (d) shows a case where the centroid of a quadrilateral is taken. White represents the line segment that connects adjacent centroids of each of the four triangles contained within a quadrilateral.

Figure 5.3a and 5.3b shows one example where adapting the latitudinal scale proved to be useful for the Tibialis Posterior. 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 with LSS $= 2$, as shown in Figure 5.3b.



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

Figure 5.3: Misplaced action line due to partial convergence. (a) Shows an incorrect translation when $|C_k| = 8$ and $CSS = 3$, however doubling the amount of cross sections during approximation with $CSS = 6$ resulted into a complete converged tendon (c).

Figure 5.4 shows the approximation of a high poly UHM mesh of the Tibialis Anterior at different scales. This shows 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 case for the Tibialis Anterior, the fact that it contains via points also increases the convergence rate. Because once via points are repositioned to their correct location, it also prevents bisecting cross sections from being misplaced.

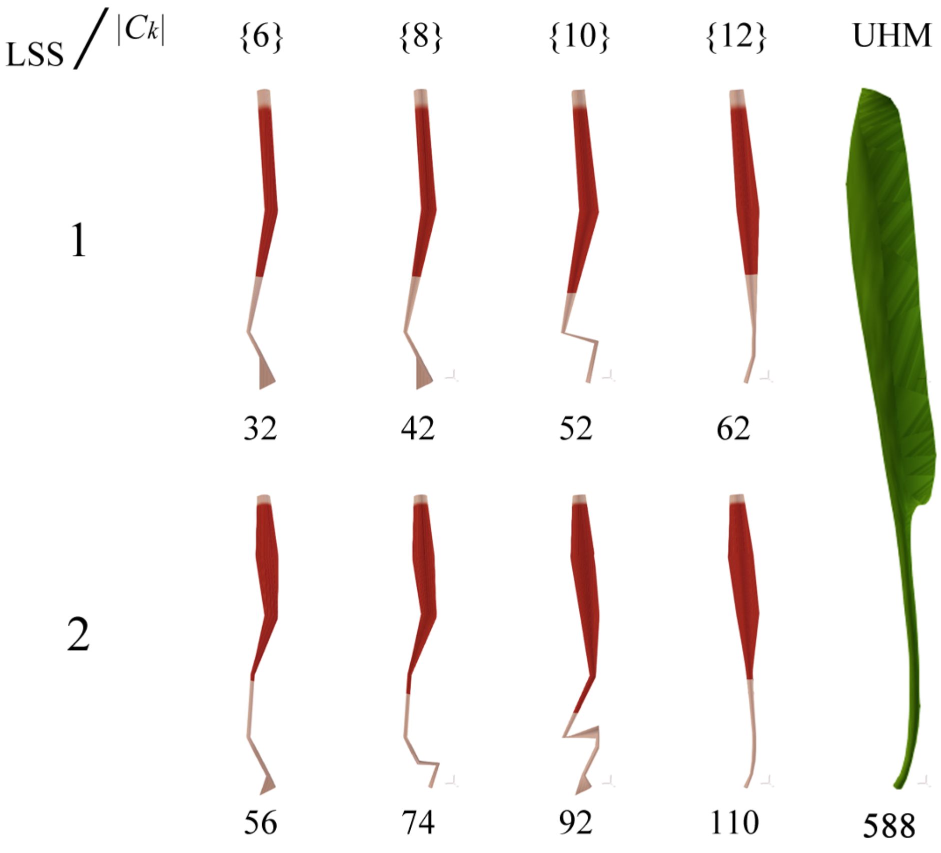


Figure 5.4: The UHM approximation of the Tibialis Anterior in eight different scales of C_k. The number below each mesh represents the total amount of vertices (without the interpolated tendon cross section C_t). For this particular mesh convergence is reached when $|C_k| = 12$, for both LSS = 1 and LSS = 2 while keeping $CSS = 1$. The convergence for $LSS = 2$ shows a better approximation for the belly of the muscle.

In order to quantify the overall result, ray-triangle tests are carried out after the UHM process is completed. These tests determine whether a_k is actually inside a cross section of an UHM 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 $c_{i^o}^k - a_k$. If both rays intersect a back-facing triangle, then it is assumed that a_k lies inside an UHM mesh. The results are listed in Table 5.2 and 5.3. The worse case for LSS = 1 represents just 23 (or 9.58%) of the total amount of 240 cross sections, while the worse case for LSS = 2 represents just 42 (or 9.72%) of the total amount of 432 cross sections. It is also worth noting that complete convergence is reached when the amount 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 (as shown in Table 5.2) or for instance when $|C_k| = 36$ and CSS $= 1$[2]. The latter, where $|C_k| = 36$ and CSS $= 1$, is considered the optimal choice of parameters as it contains the most information. In this configuration, the amount of vertices for all 48 musculotendons totals 8736, which is still $\approx 16\%$ of the UHM.

CSS / $\|C_k\|$	$Hexagon\{6\}$	$Octagon\{8\}$	$Decagon\{10\}$	$Dodecagon\{12\}$
1	23	19	7	8
2	8	7	7	3
3	7	3	1	0
4	3	2	0	0
5	1	0	0	0
6	0	0	0	0

Table 5.2: Amount of a_k outside an UHM mesh for LSS = 1.

CSS / $\|C_k\|$	$Hexagon\{6\}$	$Octagon\{8\}$	$Decagon\{10\}$	$Dodecagon\{12\}$
1	42	24	12	17
2	17	12	10	5
3	10	5	2	0
4	5	3	0	0
5	2	0	0	0
6	0	0	0	0

Table 5.3: Amount of a_k outside an UHM mesh for LSS = 2.

5.1.3 Match Detection

As discussed in Chapter 3, the classification of each match uses information from normals of each triangle's face. A previous version disregarded this information entirely due to the fact that the normals of the UHM meshes were not correctly set. This previous version took the furthest intersection point when casting rays in the positive direction, and the closest intersection point when casting rays in the negative direction, regardless of the normals. In the end, this resulted in slightly more errors when wrong intersection points were taken in highly curved segments, as shown in Figure 5.3a. This led to the inclusion of normal information by manually checking normals of modified UHM meshes that were incorrectly set in Blender Foundation (2006) to reduce the amount of errors and adapt the conditions outlined in subsection 3.5.3.2. These conditions also impose a

[2]In addition, for gathering the results, Line 14 and Line 15 of Algorithm 2 were disabled to isolate the tests from OpenSim.

limitation when the technique encounters two separate U^{C_k} in the same plane. In this case, depending on the spatial configuration of C_k with respect to either U^{C_k}, a new inner point will move within either one of the two closed curves, or when no matches are detected, would result with a deformed C_k that emcompasses the empty space in between and results in an incorrect areal representation. To prevent this from happening, areas such as the head of the Medial Gastrocnemius that consists of two attachment sites had to be closed in Blender Foundation (2006)[3].

5.2 Collision Detection and Response

The strategy discussed in Chapter 4 tried to approach the problem of collision detection and response from a different angle. The main takeaways are a new type of constraint that works with alternate interior angles of the transversal for muscle reconfiguration. This type of constraint prevents expensive length calculations to be performed such as is the case with unilateral constraints or signed distance functions. Next to transversal constraints, a narrow phase detection step that uses the same fast ray-casting algorithm (Möller and Trumbore, 1997) is also included for both muscle versus muscle and muscle versus skeleton penetration tests. The complete strategy was tested on a small set of 10 muscles, which comprises of a total of five muscle versus muscle constraints, or transversal constraints, and 20 muscle versus skeleton constraints. Three results are given in Figure 5.5.

Unfortunately, the system is not stable for the complete set of 48 musculotendons due to the exclusion of a physics engine, which forces deformations to be permanent. This creates instabilities when too many interdependencies are introduced; such as when muscle M_x affects M_y, and M_y is affected by skeleton mesh $\mathcal{S}_z$ that in turn affects back M_x. The instability is exaggerated with muscles in and around the pelvic area as these are heavily intersected with each other.

Another drawback of the solution is that transversal constraints have to be manually placed and are scale-dependent, i.e. the same set of transversal constraints for $|C_k| = 6$ would not work for $|C_k| = 8$. The placement of each transversal constraint is also not straightforward. Care has to be taken on which c_i^k's of two to be constrained M's have to be tagged, as wrong tags can also cause more displacement than necessary. Although this can be remedied with physics-based techniques such as springs where equilibrial displacement positions are converged to.

[3]This is further discussed in subsection 6.1.2.2.

Finally, it is worth mentioning that during the implementation of a_k displacements, Catmull-Rom splines were added, where each constrained a_k was used as a control point for the spline, while the remaining a_k's are interpolated. The assumption was that the curvedness of a spline would help pull the cross sections without a constraint a bit further out thereby needing less constraints to reach a penetration free state. However, during the implementation this was found to be untrue, and so the added spline computations were removed.

5.2.1 Radii Inflation

For the iterative inflation of radii, this turned out to be a good approach in preserving the same cross sectional area prior deformation in real-time, and provides a simple implementation that takes advantage of the model's cylindrical construction. Figure 5.6 shows one result for the three top cross sections of the Vastus Intermedius.

5.3 Real-Time Performance

Without the collision solver, the complete model with 48 loaded musculotendons having $|C_k| = 8$ and LSS $= 1$, and upper and lower body skeleton meshes, ran 181Hz on average with a loaded walk-cycle animation on the character Ralph. This was tested on an Intel i5-3210M x64 CPU running at 2.50GHz with a NVIDIA GeForce GT 645M mobile graphics card, and having OGRE compiled in release mode.

With an active collision solver, having a total of 25 collision constraints and 10 muscles due to stability limitations ($\approx \frac{1}{5}$ of the complete model set), the runtime was 164Hz on average using the same system configuration and walk-cycle animation that was mentioned previously. For this current solution, a fixed interval of one for every three frames was used for the collision solver to aid during debugging, thereby slowing down the rate of convergence to one iteration for every three cycles. However, this can be decreased to one to provide a maximum and faster resolution of one iteration for each cycle.

It is worth noting that both results are without any GPU or software-based parallelisation techniques. As reported in subsection 5.1.2, the optimal choice of parameters is with $|C_k| = 36$ and CSS $= 1$. Here the performance was 135Hz on average with the same system configuration as mentioned previously. Thereby satisfying the > 60Hz requirement of real-time engines. Finally, it is also worth noting that the UHM 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 each run.

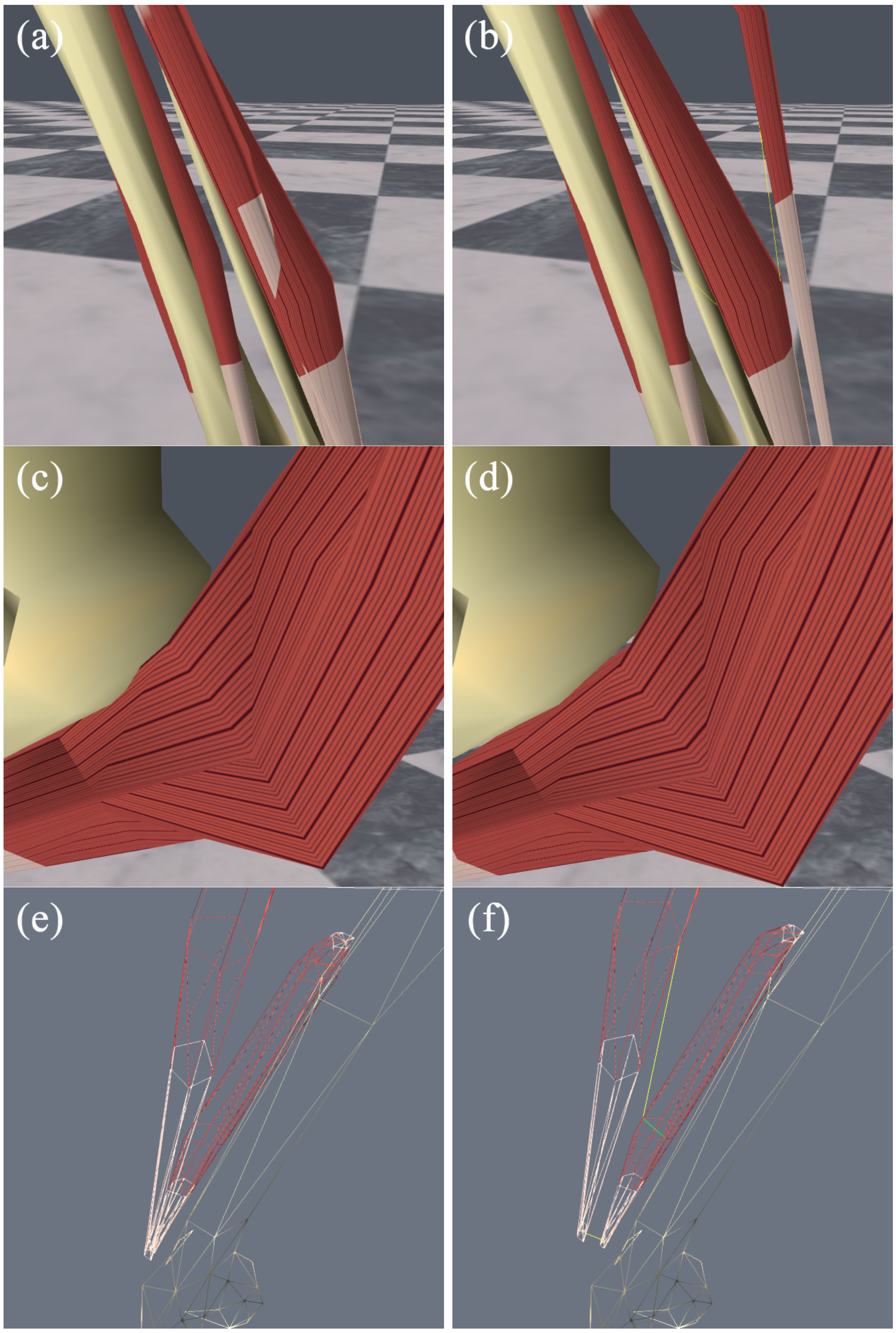
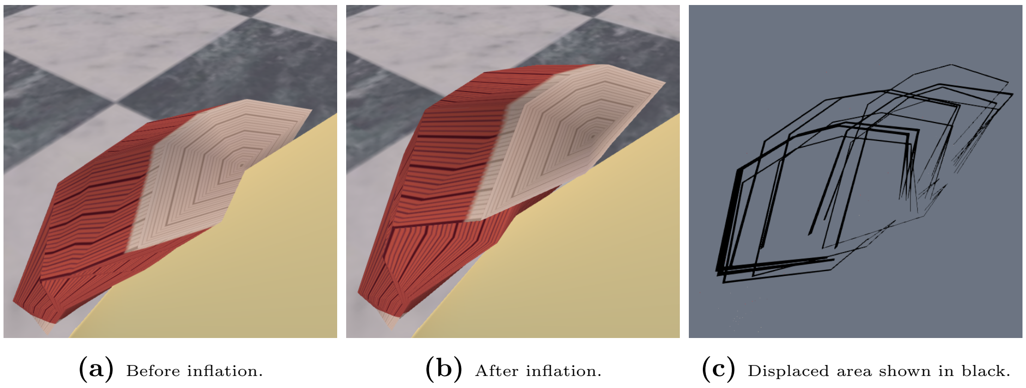


Figure 5.5: Three examples without and with constraints. (a) and (b) show how transversal constraint could help in reconfiguring a correct path between two muscles. (c) and (d) show how small a_k displacements push the thick Gluteus Maximus outward. (b) and (f) show the transversal constraint (in yellow) with (f) also showing the line segment between a_k and c_i^k (in green), and the displacement of the right muscle from the femur bone due to inflation. The example of (e) and (f) is also given in Figure 5.6.



(a) Before inflation. **(b)** After inflation. **(c)** Displaced area shown in black.

Figure 5.6: Each displacement of the area of a triangle on C_k for two consecutive c_i^k's is shown above as a black line segment of one pixel thick, offset from the base of every triangle. The thickness of culminated lines therefore gives an indication on the total displaced area after one or more radii have been pushed out of the femur bone. For this particular example, the three areas of the first three cross sections with values: 9.486×10^{-6}, 1.916×10^{-4}, and 2.587×10^{-4} where each area converged in parallel with 8, 4, and 6 iterations using Equation 4.10. Rest areas had a numerical threshold set by $\epsilon = 1 \times 10^{-6}$. Areas were calculated using Equation 4.8.

Chapter 6

Conclusion and Future Work

This MSc project explored the creation of a collision-free and simple musculoskeletal model that could be used in the future for applications such as musculoskeletal injury simulation. Due to computational limits imposed by a real-time criterion, models usually have to deal with the tug of war between accuracy and real-time performance. This project tried to keep the real-time performance in check by applying a lower bound on the cardinality of C_k, and using triangle fans and strips for the cylinder-based musculotendon unit. To leverage accuracy, low poly musculotendon models were systematically enhanced by adding more detail from high poly models. This process, labelled the UHM enhancement, introduced a permutation algorithm on pairs of intersection points. The permutation allows the possibility to locate a point inside a cross section of a high poly mesh by another separate cross section of a low poly mesh in a way that is invariant to the spatial and polygonal configuration between the two meshes. Thus providing a different way to represent high poly meshes from other data sets while keeping a simple low poly representation for future considerations discussed in this section.

Collision management for soft bio-based materials still remains a challenge in computer-based physics simulations, especially when real-time requirements are added on top. After an unsuccessful attempt with Bullet Physics, the mental effort shifted back to basics, classifying the type of information that is available and determining how the simple musculotendon model can be exploited to detect and respond to collisions. A transversal constraint was introduced with the purpose of correctly arranging one musculotendon with respect to another musculotendon spatially. This compensated unintended side effects attained from combining a polyline representation with action lines from Menegolo (2011) and polygonal meshes from the UHM (Snoswell, 2003)[1]. In addition, an iterative

[1]One example is given in Figure 5.5 where action lines of OpenSim, when fitted inside geometrical models, resulted in musculotendons intersecting each other.

method that inflates triangulated areas of each cross section was devised to preserve cross sectional area in real time.

6.1 Improvements and Future Considerations

6.1.1 Biomechanical Accuracy

This project tackled the problem of combining an action line based biomechanical model with a high poly model. As was previously discussed in subsection 3.4.3, the tendon slack length currently includes both the length of the free tendon and the length of the aponeurotic tendon (Delp et al., 1990). The latter represents the part of the tendon that is internal to a muscle's belly, i.e. it also extends through and within the muscle fibres and does not end at the point that represents the starting point of the physical muscle. In addition, classical Hill-type models such as Zajac (1988) combine the length of each separate free tendon into a single variable, termed as "in-series" in biomechanics, while in reality most musculotendon units consists of at least one tendon on each side. Unfortunately, during the course of this MSc project, no representation has been found that divides the tendon length into separate lengths for the origin and insertion, and also into its two constituents, free and aponeurotic. This is probably due to the complexity inherit in anatomical and biomechanical measurements and difficulties in generalising these measurements to be subject independent (Benjamin et al., 2006; Gerus et al., 2012).

While simplifications such as these work well for state-of-the-art polyline-based biomechanical simulators (Millard et al., 2013), a combined tendon representation is not correct for applications such as musculoskeletal injuries as geometrical or polygonal based representations of musculotendons are indispensable[2]. It is here where a crossroad between the two worlds meet as a three dimensional representation would allow volume to be segmented with differing material properties alike their real anatomical counterparts. Thus, until such progress is made, as argued by Epstein et al. (2006); Siebert et al. (2008), assumptions have to be taken such as on the length of free and aponeurotic tendons for musculoskeletal-dependent applications where musculotendon geometry plays an active role. Future endeavours in computer animation that would need to combine musculoskeletal geometry with biomechanical models should therefore investigate other ways to combine these two separate worlds.

[2]For example, injuries on bi-articular muscles such as hamstrings where musculotendon geometry plays an even larger role (Lenhart and Thelen, 2012).

6.1.2 Extending the Musculotendon Model

For this project, the lower body was used as a test case for reasons mentioned in section 3.2. The next logical step would be to include both lower and upper biomechanical models for the human body. However, to accomplish this, improvements suggested to the following class of similar problems should be considered first.

6.1.2.1 Linked Transformation Correction for A

Section 3.3 in Chapter 3 covered the great deal of work that went into the successful attachment of the skeleton onto the animated character nicknamed Ralph. There is however still one remaining improvement in the area of anthropometry that is described as follows. Each musculotendon unit is attached relative to the origin of an OpenSim body object. After applying the transformations required for the semi-automatic skeleton mesh alignment technique, some attachment points of muscles, which also inherited the same transformations, were slightly off. This has to do with the skeleton hierarchy affecting relative distances after being attached to the bone hierarchy of Ralph. The illustration in Figure 6.1 gives an example. Perhaps a future experimental project could establish which approach, such as comparing marker placement of motion-capture data, is able to best solve this anthropometric scaling problem.

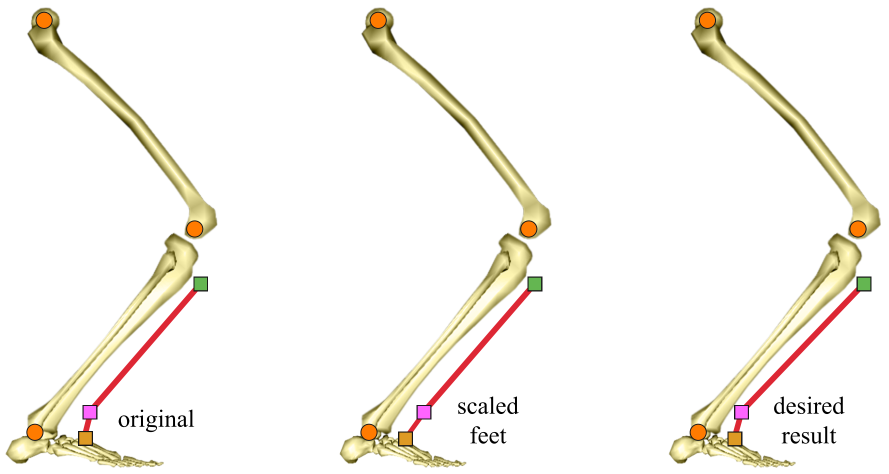


Figure 6.1: Linked transformation correction for a muscle path. The via point is attached to the parent of the foot and not influenced by the scaling of linked children. This results in a curve deviating from the original path. In this example, the correction would entail transforming the via points such that all internal angles are respected.

6.1.2.2 Multiple Action Lines for Wide Attachments

Within the human body there are some musculotendons that consist of more than one tendon at each side of attachment, such as the Extensor Digitorum Communis and the Abductor Pollicis Longus found on the forearm and palmar regions (Holzbaur et al., 2005). There are also broad and flat tendons known as *aponeuroses* that join muscle with muscle or muscle with skeleton along a broader area (Encyclopedia Britannica, 2014a). The representation of aponeurotic tendons is especially important if an upper body model has to include the ventral abdominal region or the dorsal lumbar region. Finally there is the problem area for the Gluteus group of musculotendons for the lower body, where some biomechanical models, like the one used for this project, divides each musculotendon into three separate units for their action line based representation. This resulted in high quality meshes being manually, and therefore inaccurately segmented as shown in Figure 3.14.

To conclude, the musculotendon model presented here should be extended to include a representation with multiple action lines and multiple attachment points to account for all types of musculotendons, but nevertheless be part of (or enclosed by) a single simple geometrical shape.

6.1.2.3 Distinct Heads

The Medial Gastrocnemius from the UHM had to be adapted in Blender (Blender Foundation, 2006) in order to prevent the match detection process from converging solely at either one of its two heads. Similar to the problems discussed in the previous subsection, the musculotendon model should be able to have a "splitting" option to account for multiple heads, as musculotendons in the upper body (e.g. the Biceps Brachii) also exhibit this feature. All in all, these improvements would make the geometric musculotendon model more robust as it does not have to depend on the limitations imposed by action line based biomechanical models.

6.1.2.4 A Geometric Representation for Ligaments

Ligaments connect skeletal bones with each other and can therefore be considered the real physical joints within the musculoskeletal system. They extend mechanical support to joints, improving its stability, help guide joint motion, and prevent excessive motion. The inclusion of ligaments would open up new applications for musculotendon modelling in computer animation. For instance, simulating musculoskeletal injuries that include damaged ligaments (e.g. spinal injuries).

6.1.3 Complete Collision Detection and Response

The addition of transversal constraints allow musculotendons to be configured spatially with awareness of each other, however each constraint is currently being placed manually. One improvement would be to develop a technique that allows automatic placements of constraints. Furthermore, the complete collision strategy is currently stable for just 10 musculotendons with 25 constraints. This instability can certainly be improved by including the system within a real-time physics engine as section 5.3 has demonstrated that the real-time performance is not hindered by the collision manager. In this scenario, spring-mass systems with added dampers could be used to reach optimal equilibrium.

6.1.4 The Musculotendon Model for FEM and Inverse Dynamics

Future applications for this musculotendon model, such as realistic musculoskeletal injuries, require that 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 the tetrahedralisation approach shown in Figure 6.2. The cylinder-based geometrical model presented in this thesis is therefore suitable for deformation techniques using FEM. This approach works as follows. Each segment of the geometrical musculotendon can be subdivided into a set of irregular triangular frustums, shown in Figure 6.2a and 6.2b[3]. In turn, each frustum can be further subdivided into three separate tetrahedrons, shown in Figure 6.2c. Using for instance a latitudinal dimension of six and for a particular musculotendon, the muscle compartment would encompass four segments, then the amount of tetrahedrons would be $3 \times 6 \times 4 = 72$, which is around 1% of the total amount of tetrahedrons used in the demonstration of Berranen et al. (2012) that had a performance result of 45Hz.

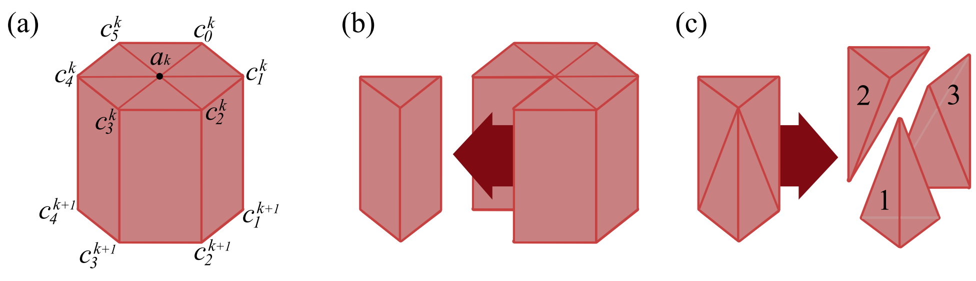


Figure 6.2: Schematic figures showing how volume of segments can be computed.

With the componential representation of the total volume for every segment that represents muscle, and because all of the vertices from the musculotendon model are known,

[3]Note that the graphical object does not actually render faces between segments as only the surface representation is sent to OGRE and drawn with triangle fans and strips, as described in subsection 3.7.

each tetrahedron's volume can be calculated geometrically as:

$$\nu = \frac{|\,(w-z)\cdot((x-z)\times(y-z))\,|}{6} \tag{6.1}$$

with w, x, y, z representing four known vertices from a musculotendon model M. Summing up the volume for every tetrahedron that occupies the segments that represents the muscle compartment (up until j from Equation 3.13) gives thus the overall volume for just the muscle part, denoted by ν^M. Using an estimated muscle tissue density of e.g. 1059.7kg/m^3 for mammalian muscle (Dorn, 2014), would allow the mass of each tetrahedron to be estimated as well. In addition, different material properties for every tetrahedron can be separately assigned to the muscle and tendon compartments.

Because the current musculotendon model includes an embedded biomechanical model, it also has the necessary information to solve inverse dynamics. However, given the inclusion of geometric information, this additionally allows the model to estimate individual muscle forces in a geometrical way, similar to Lee et al. (2009). This works as follows. Using the optimal fibre length l_o^F from Menegolo (2011), the alternative and faster computation of the physiological cross-sectional area, denoted as PCSA$_2$ can be calculated for pennate muscles, given as:

$$\text{PCSA}_2 = \frac{\nu^M \cos\alpha}{l_o^F}. \tag{6.2}$$

This in turn allows an individual muscle force F_M to be estimated geometrically as:

$$F_M = \text{PCSA}_2 \mathcal{S}_M \tag{6.3}$$

where F_M represents the estimated muscle force and $\mathcal{S}_M$ denotes the *specific tension* metric (which is a constant) for a particular muscle[4]. Tension information, e.g. within a range of $32 - 61$Ncm^{-2}, can be obtained from experimental measurements as well as biomechanical models in literature such as Dorn (2014); O'Brien et al. (2010); Maganaris et al. (2001); Arnold et al. (2010); Hansen et al. (2006); Vasavada et al. (1998); Lieber et al. (1992); An et al. (1981); Langenderfer et al. (2004); Jacobson et al. (1992).

[4]Note however that the estimated force can vary widely as it depends on the accuracy of musculotendon model and estimated measurements of specific tension (Fukunaga et al., 1996; Buchanan, 1995).

Bibliography

Albrecht, I., Haber, J., and Seidel, H. (2003). Construction and animation of anatomically based human hand models. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 98–109. Eurographics Association.

An, K., Hui, F., Morrey, B., Linscheid, R., and Chao, E. (1981). Muscles across the elbow joint: a biomechanical analysis. *Journal of biomechanics*, 14(10):659–669.

Anderson, F. and Pandy, M. (1999). A dynamic optimization solution for vertical jumping in three dimensions. *Computer Methods in Biomechanics and Biomedical Engineering*, 2(3):201–231.

Anderson, F., Pandy, M. G., et al. (2001). Dynamic optimization of human walking. *Transactions-American Society of Mechanical Engineers Journal of Biomechanical Engineering*, 123(5):381–390.

Arnold, E., Ward, S., Lieber, R., and Delp, S. (2010). A model of the lower limb for analysis of human movement. *Annals of biomedical engineering*, 38(2):269–279.

Aubel, A. and Thalmann, D. (2001a). Efficient muscle shape deformation. In *Deformable avatars*, pages 132–142. Springer.

Aubel, A. and Thalmann, D. (2001b). Interactive modeling of the human musculature. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*, pages 167–255. IEEE.

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

Berranen, Y., Hayashibe, M., Gilles, B., and Guiraud, D. (2012). 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.

Blemker, S. and Delp, S. (2005). Three-dimensional representation of complex muscle architectures and geometries. *Annals of biomedical engineering*, 33(5):661–673.

Blender Foundation (2006). Blender.org - home of the blender project - free and open 3d creation software. Available from: `http://www.blender.org/` [cited 14th of January 2014].

Bourke, P. (1988). Calculating the area and centroid of a polygon. Available from: `http://paulbourke.net/geometry/polygonmesh/centroid.pdf` [cited 28th of January 2014].

Buchanan, T. (1995). Evidence that maximum muscle stress is not a constant: differences in specific tension in elbow flexors and extensors. *Medical engineering & physics*, 17(7):529–536.

Cameron, S. (1997). Enhancing gjk: Computing minimum and penetration distances between convex polyhedra. In *ICRA*, volume 97, pages 20–25. Citeseer.

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

Chen, D. and Zeltzer, D. (1992). *Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method*, volume 26. ACM.

Christophy, M. (2010). *A detailed open-source musculoskeletal model of the human lumbar spine*. PhD thesis, University of California.

Coumans, E. (2013). Real-time physics simulation: Home of the open source bullet physics library and physics discussion forums. Available from: `http://bulletphysics.org/` [cited 9th of February 2014].

Delp, S., Anderson, F., Arnold, A., Loan, P., Habib, A., John, C., Guendelman, E., and Thelen, D. (2007). Opensim: open-source software to create and analyze dynamic simulations of movement. *Biomedical Engineering, IEEE Transactions on*, 54(11):1940–1950.

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

Dorn, T. (2014).
Opensim::musclemetabolicpowerprobeumberger2010_metabolicmuscleparameter.

Available from: `https://simtk.org/api_docs/opensim/api_docs31/classOpenSim_1_1MuscleMetabolicPowerProbeUmberger2010__MetabolicMuscleParameter.html` [cited 20th of February 2014].

Duckham, M., Kulik, L., Worboys, M., and Galton, A. (2008). Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition*, 41(10):3224–3236.

Encyclopedia Britannica (2014a). Aponeurosis. Available from: `http://www.britannica.com/EBchecked/topic/30151/aponeurosis` [cited 19th of February 2014].

Encyclopedia Britannica (2014b). Tendon (anatomy) – encyclopedia britannica. Available from: `http://www.britannica.com/EBchecked/topic/587171/tendon` [cited 6th of January 2014].

Epstein, M., Wong, M., and Herzog, W. (2006). Should tendon and aponeurosis be considered in series? *Journal of biomechanics*, 39(11):2020–2025.

Fukunaga, T., Roy, R., Shellock, F., Hodgson, J., and Edgerton, V. (1996). Specific tension of human plantar flexors and dorsiflexors. *Journal of Applied Physiology*, 80(1):158–165.

Geijtenbeek, T., van de Panne, M., and van der Stappen, A. (2013). Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, 32(6):206.

Gerus, P., Rao, G., and Berton, E. (2012). Subject-specific tendon-aponeurosis definition in hill-type model predicts higher muscle forces in dynamic tasks. *PloS one*, 7(8):e44406.

Goldman, R. (1990). Intersection of two lines in three-space. In *Graphics Gems*, page 304. Academic Press Professional, Inc.

Gurram, P., Rhody, H., Kerekes, J., Lach, S., and Saber, E. (2007). 3d scene reconstruction through a fusion of passive video and lidar imagery. In *Applied Imagery Pattern Recognition Workshop, 2007. AIPR 2007. 36th IEEE*, pages 133–138. IEEE.

Hansen, L., De Zee, M., Rasmussen, J., Andersen, T., Wong, C., and Simonsen, E. (2006). Anatomy and biomechanics of the back muscles in the lumbar spine with reference to biomechanical modeling. *Spine*, 31(17):1888–1899.

Hicks, J. (2012a). Muscle editor - opensim documentation. Available from: `http://simtk-confluence.stanford.edu:8080/display/OpenSim/Muscle+Editor` [cited 23th of December 2013].

Hicks, J. (2012b). Opensim models: Documentation. Available from: `http://simtk-confluence.stanford.edu:8080/display/OpenSim/OpenSim+Models` [cited 29th of December 2013].

Hirota, G., Fisher, S., State, A., Lee, C., and Fuchs, H. (2001). An implicit finite element method for elastic solids in contact. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*, pages 136–254. IEEE.

Holzbaur, K., Murray, W., and Delp, S. (2005). A model of the upper extremity for simulating musculoskeletal surgery and analyzing neuromuscular control. *Annals of biomedical engineering*, 33(6):829–840.

Hoy, M., Zajac, F., and Gordon, M. (1990). A musculoskeletal model of the human lower extremity: the effect of muscle, tendon, and moment arm on the moment-angle relationship of musculotendon actuators at the hip, knee, and ankle. *Journal of Biomechanics*, 23(2):157–169.

Jacobson, A., Kavan, L., and Sorkine-Hornung, O. (2013). Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph*, 32:4.

Jacobson, M. D., Raab, R., Fazeli, B., Abrams, R., Botte, M., and Lieber, R. (1992). Architectural design of the human intrinsic hand muscles. *The Journal of hand surgery*, 17(5):804–809.

Kohout, J., Clapworthy, G., Tao, Y., Dong, F., Kelnhoffer, P., Cholt, D., and Zhao, Y. (2012). Technologies for modelling fibrous muscle in motion. In *VPH 2012 London*. Virtual Physiological Human.

Laclé, F. P. (2013). A literature review for the master project: Real-time musculoskeletal model for injury simulation on 3d human characters. Available from: `http://vhtlab.nl/sites/default/files/A%20Literature%20Review%20on%20Real-Time%20Musculoskeletal%20Model%20for%20Injury%20Simulation%20on%203D%20Human%20Characters.pdf` [cited 21th of February 2014].

Langenderfer, J., Jerabek, S., Thangamani, V., Kuhn, J., and Hughes, R. (2004). Musculoskeletal parameters of muscles crossing the shoulder and elbow and the effect of sarcomere length sample size on estimation of optimal muscle length. *Clinical Biomechanics*, 19(7):664–670.

Lee, D., Glueck, M., Khan, A., Fiume, E., and Jackson, K. (2010). A survey of modeling and simulation of skeletal muscle. *ACM Transactions on Graphics*, 28(4).

Lee, K. and Ashraf, G. (2007). Simplified muscle dynamics for appealing real-time skin deformation. In *CGVR*, page 160. Citeseer.

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

Lemos, R., Epstein, M., Herzog, W., and Wyvill, B. (2001). Realistic skeletal muscle deformation using finite element analysis. In *Computer Graphics and Image Processing, 2001 Proceedings of XIV Brazilian Symposium on*, pages 192–199. IEEE.

Lenhart, R. and Thelen, D. (2012). Influence of musculoskeletal geometry on model-based predictions of plantarflexor function during gait. In *GCMAS Annual Conference*.

Lieber, R., Jacobson, M., Fazeli, B., Abrams, R., and Botte, M. (1992). Architecture of selected muscles of the arm and forearm: anatomy and implications for tendon transfer. *The Journal of hand surgery*, 17(5):787–798.

Lund, K. and Hicks, J. (2012). Coordinates and utilities - opensim documentation. Available from: `http://simtk-confluence.stanford.edu:8080/display/OpenSim/Coordinates+and+Utilities` [cited 27th of December 2013].

Maganaris, C., Baltzopoulos, V., Ball, D., and Sargeant, A. (2001). In vivo specific tension of human skeletal muscle. *Journal of applied physiology*, 90(3):865–872.

Mamou, K. (2013). Hierarchical approximate convex decomposition of 3d meshes. Available from: `http://sourceforge.net/projects/hacd/` [cited 9th of February 2014].

McGonagle, D. and Benjamin, M. (2013). Muscle attachments to bone as an unrecognised cause of pain. Available from: `http://www.enthesis.info/anatomy/muscle_and_enthesis.html` [cited 4nd of January 2014].

Menegolo, A. (2011). Simtk.org: Upper and lower body model: Overview. Available from: `https://simtk.org/home/ulb_project` [cited 26th of August 2013].

Millard, M., Uchida, T., Seth, A., and Delp, S. (2013). Flexing computational muscle: Modeling and simulation of musculotendon dynamics. *Journal of Biomechanical*

Engineering, 135(2):021005–021005. Available from: `http://dx.doi.org/10.1115/1.4023390`.

Mohr, A. and Gleicher, M. (2003). Building efficient, accurate character skins from examples. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 562–568. ACM.

Möller, T. and Trumbore, B. (1997). Fast, minimum storage ray-triangle intersection. *Journal of graphics tools*, 2(1):21–28.

Nordin, M. and Frankel, V. (2012). *Basic Biomechanics of the Musculoskeletal System*. Wolters Kluwer Health, 4th edition.

O'Brien, T., Reeves, N., Baltzopoulos, V., Jones, D., and Maganaris, C. (2010). In vivo measurements of muscle specific tension in adults and children. *Experimental physiology*, 95(1):202–210.

Patterson, M. (2003). The centroid of the quadrilateral. Available from: `http://jwilson.coe.uga.edu/emt668/EMT668.Folders.F97/Patterson/EMT%20669/centroid%20of%20quad/Centroid.html` [cited 8th of March 2014].

Ramos, J. and Larboulette, C. (2009). Real-time anatomically based character animation. In *Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, Posters and Demos*.

Rankin, J. and Neptune, R. (2012). Musculotendon lengths and moment arms for a three-dimensional upper-extremity model. *Journal of biomechanics*, 45(9):1739–1744.

Sánchez, C., Lloyd, J., Fels, S., and Abolmaesumi, P. (2014). Embedding digitized fibre fields in finite element models of muscles. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, pages 1–14.

Scheepers, F., Parent, R., Carlson, W., and May, S. (1997). Anatomy-based modeling of the human musculature. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 163–172. ACM Press/Addison-Wesley Publishing Co.

Seth, A., Sherman, M., Reinbolt, J., and Delp, S. (2011). Opensim: a musculoskeletal modeling and simulation framework for in silico investigations and exchange. *Procedia IUTAM*, 2:212–232.

Sherman, M. and Eastman, P. (2012). Simbody: Multibody physics api: Overview. Available from: `https://simtk.org/home/simbody` [cited 27th of December 2013].

Siebert, T., Rode, C., Herzog, W., Till, O., and Blickhan, R. (2008). Nonlinearities make a difference: comparison of two common hill-type models with real muscle. *Biological cybernetics*, 98(2):133–143.

Snoswell, M. (2003). Cg character - the ultimate human. Available from: `http://www.cgcharacter.com/ultimatehuman.html` [cited 12th of January 2014].

Sueda, S., Kaufman, A., and Pai, D. (2008). Musculotendon simulation for hand animation. In *ACM Transactions on Graphics (TOG)*, volume 27, page 83. ACM.

Tan, J., Turk, G., and Liu, C. (2012). Soft body locomotion. *ACM Transactions on Graphics (TOG)*, 31(4):26.

Taylor, T. (2013). Muscles of the leg and foot – male view. Available from: `http://www.innerbody.com/anatomy/muscular/leg-foot-male` [cited 6th of January 2014].

Teran, J., Blemker, S., Hing, V., and Fedkiw, R. (2003). Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 68–74. Eurographics Association.

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

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

The National Center for Simulation in Rehabilitation Research (2010). Official opensim website. Available from: `http://opensim.stanford.edu/` [cited 24th of August 2013].

Torus Knot Software Ltd (2013). Ogre source: Ogrequaternion.cpp — bitbucket. Available from: `http://www.ogre3d.org/docs/api/1.9/OgreVector3_8h_source.html#l00655` [cited 2nd of January 2014].

Vasavada, A., Li, S., and Delp, S. (1998). Influence of muscle morphometry and moment arms on the moment-generating capacity of human neck muscles. *Spine*, 23(4):412–422.

Whitaker, A. (2006). Sacred-geometry: The first step. Available from: `http://www.ancient-wisdom.co.uk/sacredgeometry.htm` [cited 13th of January 2014].

Wilhelms, J. and Van Gelder, A. (1997). 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.

Yamaguchi, G. and Zajac, F. (1989). A planar model of the knee joint to characterize the knee extensor mechanism. *Journal of Biomechanics*, 22(1):1 – 10. Available from: `http://www.sciencedirect.com/science/article/pii/0021929089901796`.

Zajac, F. (1988). Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Critical reviews in biomedical engineering*, 17(4):359–411.

Zhu, Q., Chen, Y., and Kaufman, A. (1998). Real-time biomechanically-based muscle volume deformation using fem. In *Computer Graphics Forum*, volume 17, pages 275–284. Wiley Online Library.

Appendix A

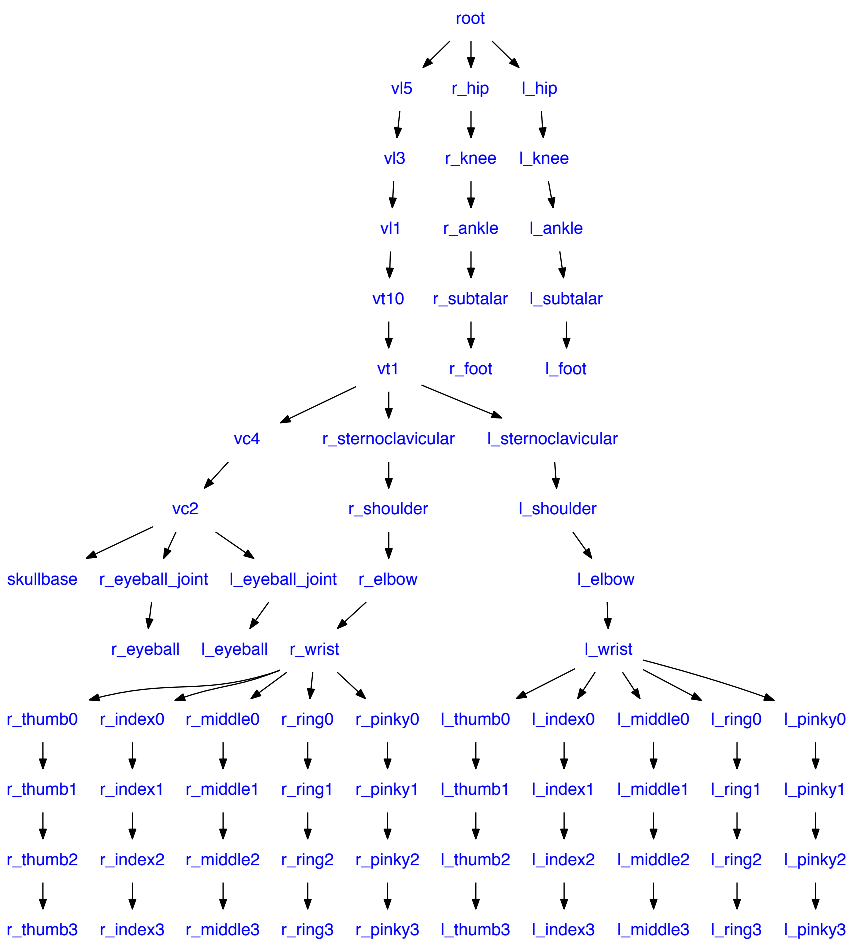
List of Included Musculotendons (OpenSim)

Musculotendons from Menegolo (2011) for the lower body, sorted alphabetically. Each musculotendon is included twice in the project to account for both sides (left and right) of the human body reaching a combined total of 48 biomechanical musculotendons during the simulation's runtime.

ID	**Anatomical Label**	**ID**	**Anatomical Label**
6	Adductor Magnus 2	20	Medial Gastrocnemius
3	Biceps Femoris-Long Head	8	Pectineus
4	Biceps Femoris-Short Head	17	Piriformis
10	Gluteus Maximus 1	14	Psoas Major
11	Gluteus Maximus 2	15	Quadratus Femoris
12	Gluteus Maximus 3	18	Rectus Femoris
0	Gluteus Medius 1	5	Sartorius
1	Gluteus Medius 2	21	Soleus
2	Gluteus Medius 3	7	Tensor Fasciae Latae
9	Gracilis	23	Tibialis Anterior
13	Iliacus	22	Tibialis Posterior
16	Inferior Gemellus	19	Vastus Intermedius

Appendix B

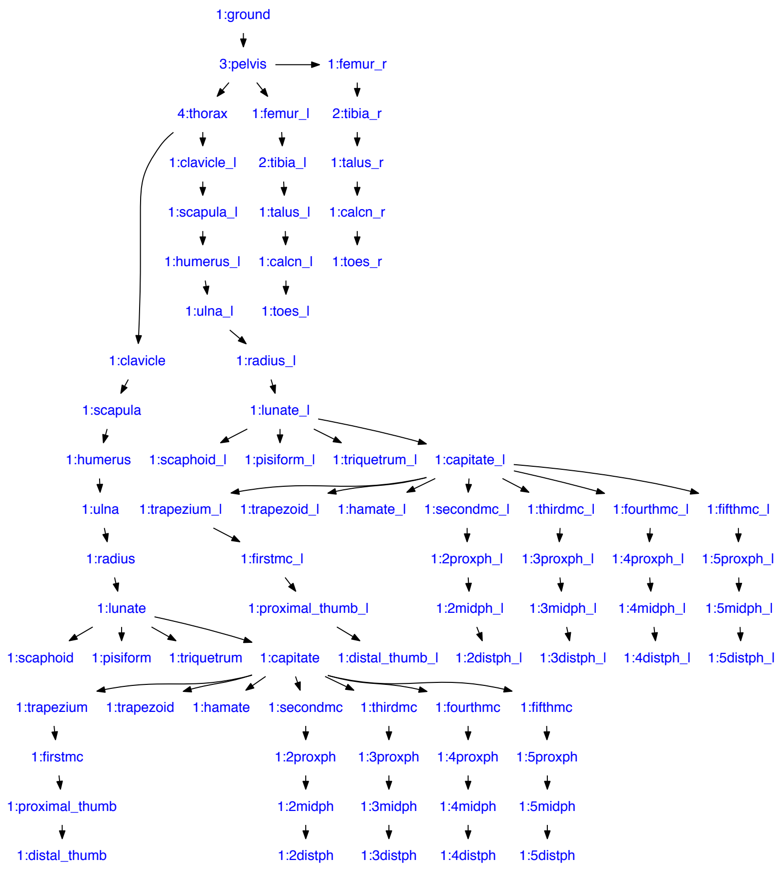
Ralph's Bone Hierarchy (RAGE)



Appendix C

Upper and Lower Body Skeleton Hierarchy (OpenSim)

A digit prefixing each node's label indicates the amount of OGRE meshes attached to each OpenSim body.



Appendix D

List of Included UHM Meshes

Musculotendon mesh files from the Ultimate Human Model data set Snoswell (2003) for the lower body sorted alphabetically. Note that some mesh files were combined to fit a single musculotendon unit in OpenSim, while others where divided for cases where OpenSim represents a single muscle with multiple action-lines, such as the Gluteus Maximus Blemker and Delp (2005). The ID field in the table below refers to mapped ID's of the OpenSim musculotendon where a particular UHM mesh was attached to, given in Appendix A. In addition, each processed mesh is included twice in the project to account for both sides (left and right) of the human body.

ID	**UHM Mesh**	**Combined with**	**Divided into**	**Anatomical Label**
0	r_mbu002		r_mbu002.1	Gluteus Medius
1			r_mbu002.2	
2			r_mbu002.3	
3	r_mul004_b			Biceps Femoris (b)
4	r_mul004_a			Biceps Femoris (a)
5	r_mul007			Sartorius
6	r_mul003_b			Adductor Magnus (b)
		r_mul003_c		Adductor Magnus (c)
7	r_mul010			Tensor Fasciae Latae
		r_lig027		Iliotibial Band
8	r_mhi006			Pectineus

ID	UHM Mesh	Combined with	Divided into	Anatomical Label
9	r_mul005			Gracilis
10	r_mbu001_c		r_mbu001_c.1	Gluteus Maximus (c)
11			r_mbu001_c.2	
12			r_mbu001_c.3	
13	r_mhi003_f			Iliospoas (f)
14	r_mhi003_d			Iliospoas (d)
15	r_mhi009			Quadratus Femoris
16	r_mhi001			Inferior Gemellus
17	r_mhi007			Piriformis
18	r_mul006_b			Rectus Femoris (b)
19	r_mul011_b			Vastus Intermedius (b)
		r_mul011_a		Vastus Intermedius (a)
20	r_lig044			Calcaneal Tendon
		r_mll005_a		Gastrocnemius (a)
			r_mll005_b	Gastrocnemius (b)
21	r_mll010_a			Soleus (a)
		r_mll010_b		Soleus (b)
22	r_mll012_g			Tibialis Posterior (g)
		r_mll012_a		Tibialis Posterior (a)
		r_mll012_b		Tibialis Posterior (b)
		r_mll012_c		Tibialis Posterior (c)
		r_mll012_d		Tibialis Posterior (d)
		r_mll012_e		Tibialis Posterior (e)
		r_mll012_f		Tibialis Posterior (f)
23	r_mll011_c			Tibialis Anterior (c)
		r_mll011_a		Tibialis Anterior (a)
		r_mll011_b		Tibialis Anterior (b)

Appendix E

Edge List Sort Algorithm

```
# Script to sort edge list
# of a simple polygon with
# 1 face.
# Something like this:
#      0_________3
#      /         \
#    4/           \2
#     \           /
#     5\_________/1
# becomes sorted as {[4,5],[5,1],...,[0,4]}.
# Tested with Blender v2.67.
# ##! Make sure to be in Edit Mode !##

import bpy
import os

#clear screen (command for windows)
os.system('cls')

scn = bpy.context.scene
obj = bpy.context.object

#get mesh data from object
mesh = obj.data

edge_count = mesh.total_edge_sel
print("number of elements in m:", edge_count)

if(edge_count > 0):

    e = []
    e.append(mesh.edges[0])
    flip = []

    #main loop for edge e
    for e_iter in mesh.edges:
        eb = e[-1].vertices[1] #end vertex of last edge of e
        found = False
```

```python

        #first search
        for m in mesh.edges:
            ma = m.vertices[0] #begin vertex of edge m
            mb = m.vertices[1] #end vertex of edge m
            if(eb == ma):
                flipFound = False
                for n in e:
                    if(n.vertices[0] == mb and n.vertices[1] == ma):
                        flipFound = True
                        break
                if(flipFound == True):
                    continue
                else:
                    e.append(m)
                    found = True
                    break

        #check for reverse direction in case first search failed
        if(found == False):
            for m in mesh.edges:
                ma = m.vertices[0] #begin vertex of edge e
                mb = m.vertices[1] #end vertex of edge e

                #...also exclude existing m's in e
                if(mb == eb and m not in e):
                    #create duplicate to reverse vertex indices
                    m_dup = mesh.copy()
                    f = m_dup.edges[m.index]
                    f.vertices[0] = mb
                    f.vertices[1] = ma
                    e.append(f)
                else:
                    continue

    #remove last element (was added twice)
    del e[-1]

    print("number of elements in e:",len(e))
    print()
    print("--- sorted edge list ---")
    for idx, val in enumerate(e):
        print("[",val.vertices[0],",",val.vertices[1],"]")
    print()
    print("--- original edge list ---")
    for i in mesh.edges:
        print("[",i.vertices[0],",",i.vertices[1],"]")
```

Appendix F

Area and Centroid Calculation

For cases where the centroid (also known as the centre of gravity) is needed, one can use the equations from Bourke (1988) and the permutation σ in Algorithm 1. This goes as follows. The projected two dimensional area of C_k is first calculated with:

$$area = \frac{1}{2}\sum_{i=0}^{n-1} x_i y_{i+1} - x_{i+1} y_i \tag{F.1}$$

where x and y represent the two coordinates lying in the two dimensional coordinate system[1]. Then, the centroid of each U^{C_k} can now be calculated as well with:

$$centroid_x = \frac{1}{6(area)}\sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \text{ and} \tag{F.2a}$$

$$centroid_y = \frac{1}{6(area)}\sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{F.2b}$$

However, as discussed in Chapter 3, the centroid does not always lie within a slice taken from a UHM mesh and therefore is not suitable for this purpose.

[1]Note that the centroid is a different calculation than the standard arithmetic mean seen in other contour extraction approaches and represents the centre of gravity of the C_k.